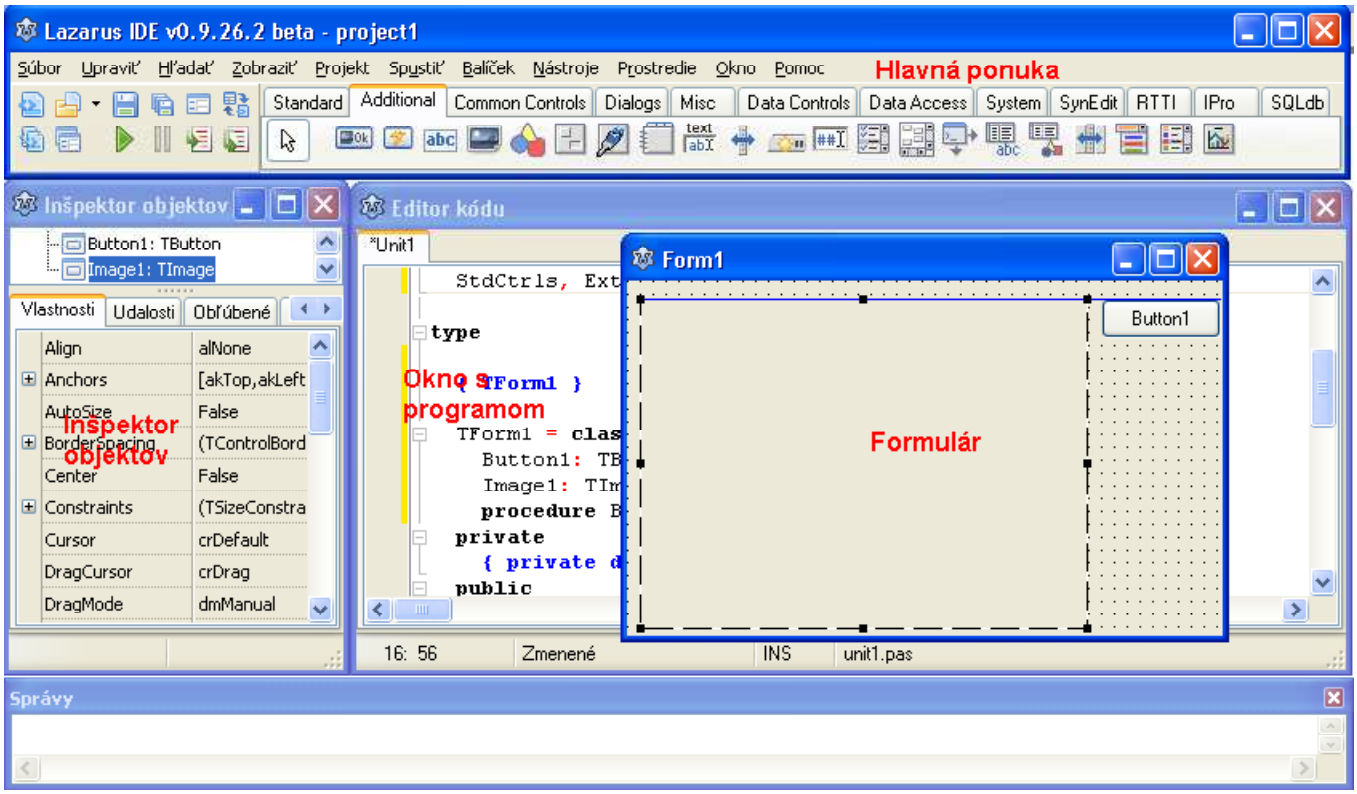
Úvod do programovania s Pascalom v prostredí Lazarus

TurboDelphi je pomenovanie programátorského prostredia, v ktorom sa programuje v jazyku **Objektový Pascal**. Existuje aj veľmi podobné prostredie s názvom **Lazarus**, ktoré poskytuje skoro identický **Objektový Pascal**. Toto prostredie je vyvíjané ako open source (otvorený kód). Na rozdiel od TurboDelphi je Lazarus voľne stiahnuteľný (http://sourceforge.net/projects/lazarus/files).
Samotný jazyk pascal je komunitou informatikov na celom svete považovaný za jazyk, ktorý je určený nielen na učenie programovania, ale aj na rozvoj algoritmického myslenia.
Pascal, podobne ako jazyk C, vznikol začiatkom 70-tych rokov. Oba tieto jazyky boli postavené na základe štruktúrovaných konštrukcií jazyka Algol a obidva sa inšpirovali vtedajšími jazykmi ako Fortran, Cobol a PL/1. Pascal sa začal šíriť vďaka univerzitám na celom svete. Na rozdiel od neho jazyk C vznikol pre systémových programátorov, teda pre už veľmi skúsených programátorov. Postupom času (v polovici 80-tych rokov) sa z Pascalu vyvinul moderný Objektový Pascal a C sa zmodernizovalo na objektovo orientované C++.

Programátorské prostredie (IDE - integrated development environment) je softvérový balík umožňujúci programátorovi:

- programy navrhovať, písať, upravovať, vyvíjať
- kompilovať (preložiť do binárneho kódu, ktorému už rozumie procesor)
- testovať, ladiť



Editovacie okno s programom slúži na zápis algoritmov – programovať budeme v programovacom jazyku objektový pascal. Programovanie vlastne bude znamenať to, že budeme popisovať, ako sa bude program správať v rôznych situáciách.
Formulár (zobrazuje sa stlačením klávesy F12) slúži na vizualizáciu nášho budúceho programu. Programy vo Windows majú väčšinou tvar obdĺžnikového okna a v ňom obsahujú rôzne väčšinou štandardné prvky ako texty, tlačidlá, posúvače, obrázky a pod. My budeme počas prípravy nášho budúceho projektu práve do tohto formuláru (budúceho okna aplikácie) rozmiestňovať a aj upravovať najrôznejšie súčiastky (komponenty).
Paleta komponentov z Hlavnej ponuky obsahuje predpripravené "súčiastky", z ktorých sa bude skladať okno našej vytváranej aplikácie. Táto paleta sa skladá z viacerých záložiek, ale my sa postupne zoznámime len s ich veľmi malou množinou.
Inšpektor objektov nám pomôže nastavovať rôzne parametre pre súčiastky (komponenty), ktoré sme už umiestnili v našom formulári.

Súčasné moderné programátorské prostredia, ktoré umožňujú vyvíjať nielen aplikácie pre grafické rozhranie (napr. Windows), ale aj aplikácie pre web – sú založené na vizuálnom princípe: všetko, čo bude mať v bežiacej aplikácii vizuálne znázornenie, sa už počas návrhu bude dať vizuálne poskladať z nejakých predpripravených častí. Programátor potom veľmi často "iba" doprogramováva správanie týchto komponentov v rôznych situáciách a grafická nadstavba mu pritom zabezpečí celkové správne fungovanie aplikácie.

Vytvorenie a spustenie novej aplikácie

1. Spustíme prostredie Lazarus.
2. Vytvoríme nový program (aplikáciu) príkazom z ponuky: *Súbor* ► *Nový* ► *Projekt-Aplikácia*
3. Na obrazovke uvidíme prázdne okno s nápisom **Form1**. Na lište bežiacich programov vo Windows uvidíme, že pribudol program s názvom *project1*.
4. Doporučuje sa takýto zatiaľ "prázdny" program uložiť – v ponuke **Súbor** zvolíme **Uložiť všetko.** Teraz si vo svojom adresári vytvoríme nový adresár, do ktorého budeme projekt ukladať – odporúča sa ukladať každý projekt do nového adresára. Každý projekt, kým ho ešte programujeme, sa na disku skladá z viacerých súborov a tieto by mali byť spolu v jednom priečinku a nemali by sa radšej prekrývať s inými projektmi. Potom je vhodné potvrdiť ponuku „**Premenovať na malé písmená**“
5. Túto aplikáciu spustíme povelom tlačidlom alebo z ponuky: *Spustiť* ► *Spustiť*.
6. Hoci sme ešte nič neprogramovali, prostredie nám s novou aplikáciou všetko pripravia tak, aby sme už mali funkčné jedno Windows okno. V okne s programom je už napísaný kúsok programu, ktorý vygenerovalo vývojové prostredie automaticky.
7. Spustený program ukončíme tak, ako iné programy – kliknutím na

Práca vo vývojovom prostredí

Vo vývojovom prostredí budeme vytvárať aplikácie tak, že:

- navrhneme **vzhľad aplikácie**, teda, ako má vyzerať okno nášho programu,
- budeme **programovať** v jazyku Pascal (presnejšie Object Pascal),
- budeme spúšťať a **ladiť**, čiže testovať a opravovať, naše programy.

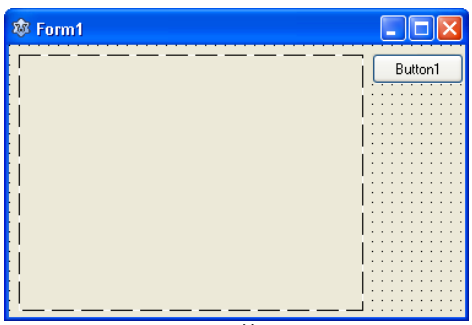
Vzhľad aplikácie vytvoríme tým, že z **palety komponentov** postupne umiestnime **komponenty** do **formulára** (pripomína to skladanie stavby z kociek Lega):
1. Komponent vyberieme z palety kliknutím myšou na ikonu komponentu.
2. Vybraný komponent potom umiestnime do formulára, kliknutím myši alebo ťahaním myši vo formulári.

Formulár predstavuje budúce okno našej aplikácie. Tak, ako teraz komponenty umiestnime do formulára, tak bude vyzerať naša aplikácia po spustení.
Komponenty sú stavebné prvky, z ktorých vytvárame aplikácie. Slúžia na interakciu používateľa s programom, na zadávanie vstupných a zobrazovanie výstupných hodnôt alebo na riadenie behu aplikácie.
Najčastejšie budeme používať komponenty tlačidlo (Button), obrázok (Image), editovacie políčko (Edit), textová plocha (Memo), zaškrtávacie políčko (CheckBox).

Komponent Button (tlačidlo) zo záložky Standard.

Komponent Image (obrázok) zo záložky Additional

Teraz vložíme do formulára komponenty *Button* (**tlačidlo**) a *Image* (**obrázok**). Tlačidlom budeme spúšťať príkazy programu. Obrázok bude slúžiť ako výstup programu.

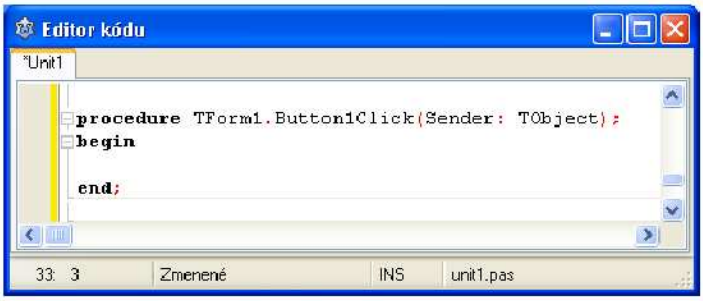


Formulár s tlačidlom a obrázkom

Prvý program

V okne s programom je už napísaný kúsok programu, ktorý vygenerovalo vývojové prostredie automaticky. Postupne sa dozvieme, čo jednotlivé slová znamenajú. Teraz však zrealizujme tieto pokyny:

1. Vo formulári dvojklikneme na tlačidlo (pozor, nie v bežiacom programe).
2. Zobrazí sa **okno s programom**, pričom sa doň automaticky dopísal ďalší text.



Zmena v okne s programom

3. Do riadka, kde bliká kurzor, medzi slová `begin` `end`, napíšeme **príkaz**:

```
Image1.Canvas.TextOut(100, 50, 'Ahoj');
```

↑*bodka* ↑*apostrof* ↑*bodkočiarka*

4. Program spustíme a **v okne bežiaceho programu** stlačíme tlačidlo.

Príkaz, ktorý sme dopisovali do programu, je súčasťou takzvanej **procedúry**:

```
procedure TForm1.Button1Click(Sender: TObject);
```

hlavička

```
begin
  Image1.Canvas.TextOut(100, 50, 'Ahoj');
end;
```

telo

Procedúru môžeme zatiaľ chápať ako zoskupenie príkazov. V zápise procedúry budeme rozlišovať dve časti - **hlavičku** a **telo**:

- Hlavičke procedúry zatiaľ nebudeme úplne rozumieť. Vidíme však, že hlavička začína slovom `procedure`. Ďalej v nej rozpoznáme slovo `Button1Clik` - to je názov procedúry.
- Slová `begin` a `end` vymedzujú telo procedúry (*begin = začiatok, end = koniec*). Do riadkov medzi `begin` a `end` budeme vpisovať aj ďalšie príkazy.

Prázdnu procedúru `Button1Click` vygenerovalo vývojové prostredie automaticky. Stalo sa tak vtedy, keď sme dvojklikli na tlačidlo vo formulári (viď. 1. bod postupu). Týmto úkonom sme zároveň zabezpečili, že príkazy z procedúry `Button1Click` sa vykonajú vtedy, keď v spustenom programe stlačíme tlačidlo.

Príkaz `Image1.Canvas.TextOut(100, 50, 'Ahoj')` zobrazí text, preto ho budeme nazývať **príkaz pre výpis textu**:

- Príkaz čítame takto *„obrázok 1, do svojej grafickej plochy napíš na súradnice 100, 50 text 'Ahoj'“*.
- Zátvorky vymedzujú **parametre** príkazu - *x*-ovú, *y*-ovú pozíciu a *text*.
- Text píšeme medzi apostrofy tak, ako píšeme priamu rľeč do úvodzoviek.

Komponenty, ktoré vkladáme do formulára dostávajú svoje jednoznačné mená. Napríklad obrázok dostal meno Image1, tlačidlo Button1.

Poznámka: pri práci v prostredí Lazarus sa v našom spustenom programe zafarbí plocha na čierno. Ak chceme, aby mal obrázok biele pozadie, musíme ho zmazať:

1. Dvojklineme do formulára (keď náš program nie je spustený),
2. V okne s programom vznikne ďalšia procedúra, do ktorej vpíšeme príkaz:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
end;
```

Grafickú plochu budeme takto mazať aj v budúcnosti, keď budeme v prostredí Lazarus vytvárať grafické projekty.

Počítačová súradnicová sústava je iná, ako sme boli zvyknutí v matematike:


```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(0, 0, 'Ahoj');
end;
```

Ak takýto program znovu spustíme, uvidíme, že bod so súradnicami 0, 0 leží v ľavom hornom rohu obrázka. Os *x*-ová rastie smerom vpravo, *y*-ová smerom nadol.

Do grafickej plochy môžeme vypisovať na rôzne miesta aj viac textov:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(80, 50, 'Ahoj');
  Image1.Canvas.TextOut(50, 70, 'Ako sa máš?');
end;
```

V jazyku Pascal oddeľujeme dva príkazy ; (bodkočiarkou). Hoci za posledným príkazom je bodkočiarka zbytočná, budeme ju tam písať pre prípad, že by sme chceli na koniec tela procedúry dopisovať ešte ďalšie príkazy.

Príklady ako sa správne čítajú niektoré príkazy:

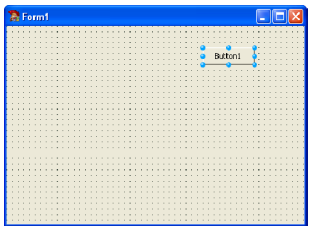
Príkaz pre výpis textu: ***Image1.Canvas.TextOut (80,50, 'Ahoj');*** – budeme pracovať s grafickou plochou **Image1**, na jej platno (**Canvas**) vypíšeme **text** „***Ahoj***“ na súradnice (80,50)

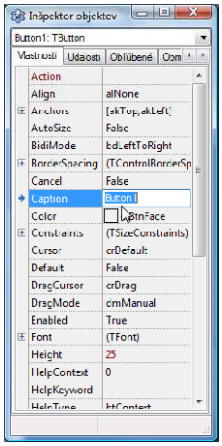
Priradzovací príkaz pre farbu pera: ***Image1.Canvas.Pen.Color := clRed;*** grafickej ploche **Image1** jej plátnu **Canvas** jeho peru **Pen** nastavíme farbu **Color - červená**

Príkaz pre nakreslenie obdĺžnika: ***Image1.Canvas.Rectangle (0,0,200,100);*** – budeme pracovať s grafickou plochou **Image1**, na jej platno (**Canvas**) budeme kresliť obdĺžnik (**Rectangle**) s prvým vrcholom so súradnicami (**0,0**) a s protiľahlým vrcholom (**200,100**)

Základné komponenty a ich vlastnosti

Komponenty (súčiastky) sú stavebné prvky, z ktorých vytvárame aplikácie. Slúžia na interakciu používateľa s programom, na zadávanie vstupných a zobrazovanie výstupných hodnôt alebo na riadenie behu aplikácie. Najčastejšie budeme používať komponenty tlačidlo (Button), obrázok (Image), editovacie políčko (Edit), textová plocha (Memo), zaškrtávacie políčko (CheckBox).
Aby sme mohli ukladať komponenty do formulára, tento musí byť vpredu – ešte pred editovacím oknom. Ak ešte nie je, stlačíme kláves **F12** a objaví sa formulárové okienko. Z palety komponentov zo štandardnej záložky vložíme napr. súčiastku tlačidlo (**TButton**).





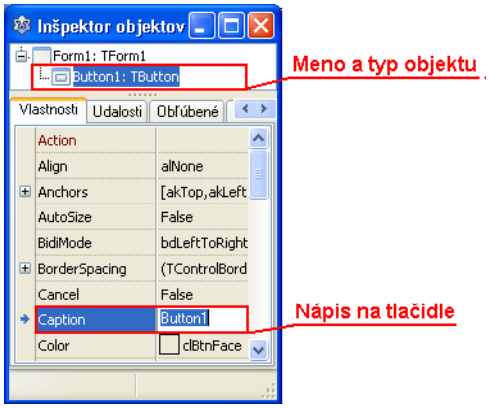
Toto tlačidlo môžete presunúť na ľubovoľnú pozíciu, prípadne mu zmeniť veľkosť ťahaním za modré guľôčky (resp. čierne štvorčeky v Lazarus). Popis na tlačidle zmeníme veľmi jednoducho: všimnite si, že keď je naše nové tlačidlo označené (má okolo seba modré alebo čierne značky), v **Inšpektore objektov** (Object Inspector) sa zobrazujú nejaké informácie, ktoré sa týkajú práve tohto tlačidla:
Inšpektor sa skladá z dvoch stĺpcov: ľavý obsahuje meno nastavenia (napr. **Caption** označuje popis na tlačidle) a v príslušnom pravom je momentálna hodnota (teraz je tam **Button1**). Informácie v pravom stĺpci môžeme meniť a tým sa bude meniť vzhľad tlačidla a niekedy aj jeho správanie. Zmeňte v Inšpektore nastavenie **Caption** na text, napr. **Tlačidlo**. Všimnite si, že zároveň sa zmenil popis na tlačidle.

Ak teraz spustíme projekt (kláves **F9**) na formulári sa objavilo tlačidlo s popisom **Tlačidlo** – na tlačidlo môžeme klikať, ale zatiaľ toto nemá žiadnu funkčnosť.

V Inšpektore objektov môžeme experimentovať ešte s týmito nastaveniami:

- **Caption** nadpis **(**text) na tlačidle
- **Font** nastavenie písma textu na tlačidle
- **Height** výška tlačidla
- **Width** šírka tlačidla
- **Left** x-ová vodorovná súradnica polohy tlačidla (vzdialenosť od ľavého okraja)
- **Top** y-ová zvislá súradnica polohy tlačidla (vzdialenosť od horného okraja)

Ak do formulára vložíme tlačidlo, pomocou myši nastavíme jeho polohu alebo veľkosť. Poloha aj veľkosť sú **vlastnosti** tlačidla.



Vlastnosť s menom Caption má hodnotu Button1. Túto hodnotu môžeme v *Inšpektore objektov* prepísať, napríklad na *„Klikni na mňa“*. Všimnime si aj, že pozícia tlačidla sa dá nastaviť dvomi spôsobmi: pomocou myši, ale aj pomocou *Inšpektora objektov*. Ak zmeníme tlačidlu vlastnosť Left na hodnotu 0, tlačidlo sa presunie k ľavému okraju formulára.

Vlastnosti komponentov sa dajú meniť aj počas behu programu, pomocou príkazu priradenia:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Caption:='Kuk';
end;
```

K vlastnostiam komponentov pristupujeme pomocou . (bodky) takto:

```
komponent.vlastnosť
```

Vlastnosť môžeme nielen nastavovať, ale aj zisťovať jej hodnotu:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Button3.Left:=Button3.Left+50;
end;
```

Na pravej strane od := sa teraz nachádza zložitejší výraz, ktorý hovorí: *„k súčasnej x-ovej pozícii tlačidla pripočítaj 50“*. Výsledok bude číslo o 50 väčšie, ako je súčasná x-ová pozícia tlačidla. Toto číslo sa použije pre nastavenie vlastnosti Left. Preto tlačidlo poskočí a vzdiali sa od ľavého okraja formulára o 50 bodov. Znamená to, že na pravej strane príkazu priradenia môže byť aj pomerne komplikovaný výraz. Pri vykonaní príkazu priradenia program postupuje nasledovne:

1. Najskôr sa vyhodnotí výraz na *pravej strane* od :=.
2. Potom sa výsledná hodnota priradí *ľavej strane* od :=.

Na začiatku sme spomínali, že čísla a texty sú rôzne typy údajov. Musíme na to pamätať aj pri práci s vlastnosťami:

- **číselné hodnoty:** Button1.Left := 100;
- **textové hodnoty:** Button1.Caption := 'Nový nadpis';
- **farby:** Image1.Canvas.Font.Color := clRed;

Typy údajov

V príkaze: *Image1.Canvas.TextOut (80,50, ‘Ahoj‘)* prvé dva parametre v príkaze výpisu musia byť **celé čísla**. Tretí, posledný parameter musí byť text, presnejšie, textový **reťazec**. V zápisoch v programe odlíšime textové reťazce od čísel alebo príkazov tým, že znaky textového reťazca napíšeme medzi apostrofy.

Čísla a texty sú pre počítač rozdielne **typy údajov**. S každým vykonávame iné úkony, inak s nimi manipulujeme. Preto napríklad v príkaze výpisu nemôžeme na mieste textového parametra napísať číslo:

- **nesprávne:** Image1.Canvas.TextOut(0, 0, 123) ... 123 je číslo
- **správne:** Image1.Canvas.TextOut(0, 0, '123') ...'123' je reťazec

Rôzne typy údajov môžeme chápať ako rôzne množiny. Tie majú v jazyku Pascal (Lazarus) svoje pomenovanie:

- celé číslo je typu (je z množiny) Integer
- desatinné číslo: Real
- textový reťazec: string
- logický typ: Boolean - môže nadobúdať iba hodnotu **true** (pravda) alebo **false** (nepravda)
- farba: TColor
- obrázok: TImage ... aj obrázky a tlačidlá sú pre počítač údaje!
- tlačidlo: TButton

Vždy, keď začíname vyvíjať nový projekt (najlepšie typu Aplikácia), je dobré si uložiť celý projekt do nového priečinku

Pri ukladaní programu sa vytvárajú viaceré súbory s rôznymi príponami:

- project1.ico - ikona projektu
- **project1.lpi** – (lazarus project information file) - informácie o projekte (uložený v XML; obsahuje nastavania špecifické pre konkrétny projekt)
- **project1.lpr** (lazarus program file); obsahuje pascalovský zdrojový kód hlavného programu
- **project1.lrs** -(lazarus resource file) - vygenerovaný resource súbor (nie je to súbor Windows resource)
- project1.manifest - súbor manifestu pre zapnutie tém Windows (jeho generovanie môžeme vypnúť vo Voľbách projektu)
- project1.rc - popis Windows resource
- **unit1.lfm** (lazarus form file) - popis vzhľadu formulára – obsahuje vlastnosti všetkých objektov na formulári (uložený v Lazarovskom špeciálnom textovom formáte; akcie sú uložené v Pascale v príslušnom `.pas` súbore)
- unit1.lrs – (lazarus resource file) - toto je generovaný súbor, nemýľte si ho s resource súbormi Windows
- **unit1.pas** - programová jednotka - obsahujúce kód formulára (popis formulára, čo a ako má formulár robiť)
- project1.exe – spustiteľný súbor

Pri archivovaní celého projektu, alebo pri jeho prenose na iný počítač, musíme preniesť v danom adresári minimálne tieto súbory: **project1.lpi**, **project1.lpr**, **project1.lrs**, **unit1.lfm** a **unit1.pas**.

Kreslenie geometrických útvarov

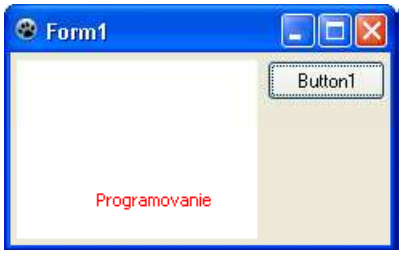
Naučíme sa príkazy, ktoré nám umožnia kresliť rôzne geometrické útvary, akými sú kruhy, štvorce alebo úsečky a naučíme sa pracovať s farbami.

Vymazanie obrazovky (2. spôsob) **: Image1.Canvas.Rectangle (-1,-1,1000,1000); - príkaz je potrebné použiť hneď na začiatku programu**

Písma

Texty, ktoré píšeme do grafickej plochy, môžeme písať farebne a rôznymi písmami. Napríklad, takto vypíšeme červený text:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Image1.Canvas.Font.Color:=clRed;
   Image1.Canvas.TextOut(50, 80, 'Programovanie');
end;
```



V programe sme použili pre nastavenie farby písma nový **príkaz priradenia**. Tento príkaz obsahuje symboly := (dvojbodka rovná sa). Na pravej strane od := je názov červenej farby clRed (cl = skratka zo slova **col**or, Red = červená). Príkaz pre nastavenia farby môžeme čítať nasledovne: *„obrázok 1, tvoja grafická plocha, tvoje písmo, jeho farba nastav na červenú“*. Základné farby sú pomenované:

clBlack, clRed, clGreen, clBlue, clWhite, clYellow, ... clAqua, clCream, clDark, clDkGray, clFuchsia (svetlofialová), clGray, clLime (svetlozelená - lipa), clLtGray, clMaroon (gaštanovohnedá), clMedGray, clNavy (námornícka modrá), clOlive, clPurple, clSilver, clSkyBlue

Ďalšími príkazmi môžeme nastaviť typ alebo veľkosť písma:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Image1.Canvas.Font.Color:=clRed;
   Image1.Canvas.Font.Name:='Times New Roman';
   Image1.Canvas.Font.Height:=24;
   Image1.Canvas.TextOut(30, 80, 'Programovanie');
end;
```

Vlastnosti písma, a neskôr uvidíme, že aj iné vlastnosti, nastavujeme pomocou príkazu priradenia takto:

meno vlastnosti := *nová hodnota*

Poradie príkazov

Poradie príkazov v programe býva veľmi dôležité. Skúsme zmeniť poradie príkazov:

```
Image1.Canvas.TextOut(200, 100, 'Programovanie');
Image1.Canvas.Font.Color:=clRed;
```

Akou farbou sa vypíše text po prvom a po druhom stlačení tlačidla? Po prvom sa vypíše text čiernou farbou, pretože farbu textu nastavujeme až potom, ako sa vykonal príkaz pre výpis textu. Nastavenie farby sa medzi stlačeniami tlačidla pamätá. Preto sa až po druhom stlačení vypíše text červenou farbou.

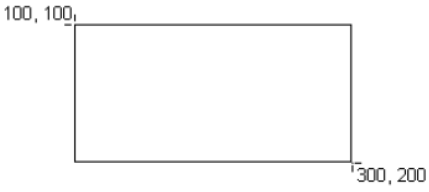
Používame viac tlačidiel

Do formulára môžeme vložiť ďalšie tlačidlo. Všimnime si, že prvé tlačidlo má nápis Button1 a druhé Button2. Dvojkliknite na druhé tlačidlo. V texte programu pribudne procedúra Button2Click. Do nej budeme dopisovať príkazy, ktoré sa vykonajú, ak počas behu programu stlačíme druhé tlačidlo. Podobným spôsobom môžeme pridávať a reagovať na stlačenie ďalších tlačidiel.

Kreslenie obdĺžnikov a štvorcov

Do grafickej plochy sa dajú kresliť rôzne jednoduché geometrické útvary. Napríklad **obdĺžnik** nakreslíme takto:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   Image1.Canvas.Rectangle(100, 100, 300, 200);
end;
```



Príkaz má 4 parametre. Sú to súradnice dvoch bodov, medzi ktoré sa nakreslí obdĺžnik. Prvé dve čísla (100, 100) sú súradnice prvého vrcholu, tretie a štvrté číslo (300, 200) sú súradnice protiľahlého vrcholu obdĺžnika. Obdĺžnik sa nakreslí tak, že bude mať strany rovnobežne so stranami grafickej plochy.

Štvorce tiež kreslíme pomocou príkazu na kreslenie obdĺžnikov. **Štvorec** je špeciálny prípad obdĺžnika, ktorého strany majú zhodnú dĺžku. Napríklad štvorec, so stranou dĺžky 100 nakreslíme takto:

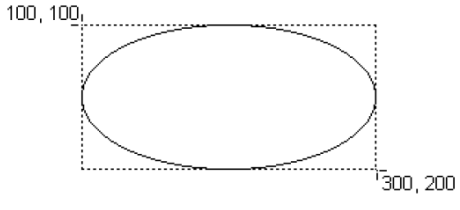
```
procedure TForm1.Button2Click(Sender: TObject);
begin
   Image1.Canvas.Rectangle(50, 20, 50+100, 20+100);
end;
```

Parametre môžu byť aj výrazy, ktoré sa najskôr vyhodnotia.

Kreslenie elíps a kruhov

Príkaz na kreslenie **elipsy** sa podobá príkazu na kreslenia obdĺžnika:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   Image1.Canvas.Ellipse(100, 100, 300, 200);
end;
```



Elipsa sa vpíše do pomyselného obdĺžnika, ktorého súradnice sme uviedli ako parametre príkazu.

Kruh je opäť špeciálny prípad elipsy. Ako nakreslíme kruh so stredom v bode 50, 50 a polomerom 10? Pre príkaz Ellipse musíme vypočítať súradnice dvoch protiľahlých vrcholov štvorca, do ktorého sa elipsa – vlastne už kruh, vpíše.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   Image1.Canvas.Ellipse(50-10, 50-10, 50+10, 50+10);
end;
```

Kreslenie úsečiek

Úsečku nakreslíme pomocou dvojice príkazov:

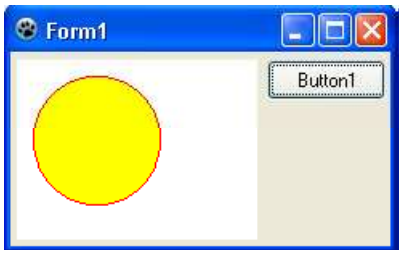
```
procedure TForm1.Button4Click(Sender: TObject);
begin
   Image1.Canvas.MoveTo(100, 50);
   Image1.Canvas.LineTo(200, 80);
end;
```

Príkaz Image1.Canvas.MoveTo presunie neviditeľné *grafické pero* do zadaného bodu. Príkaz Image1.Canvas.LineTo nakreslí s grafickým perom čiaru.

Farba obrysu a výplne

Pri kreslení obdĺžnikov a kruhov môžeme nastaviť farbu **výplne** (vnútra) a farbu **obrysu** (obvodu). Kruh so žltou výplňou a červeným obrysom nakreslíme takto:

```
procedure TForm1.Button5Click(Sender: TObject);
begin
   Image1.Canvas.Brush.Color:=clYellow;
   Image1.Canvas.Pen.Color:=clRed;
   Image1.Canvas.Ellipse(10, 10, 90, 90);
end:
```



Vysvetlenie:

Pen.Color znamená *farba pera* a určuje farbu čiaru, obrys útvaru Brush.Color znamená *farba štetca*, ktorou sa vymaľuje, vyplní plocha

Farba pera má vplyv aj pri kreslení úsečiek.

Náhodné čísla

Naše programy sa zatiaľ správali veľmi presne – vždy nakreslili rovnaký útvar, pretože

súradnice geometrického útvaru sa nezmenili. Pozrime sa ale na tento príklad:

```
procedure TForm1.Button6Click(Sender: TObject);
begin
   Image1.Canvas.LineTo(Random(400), Random(300));
end;
```

Vidíme, že po stlačení tlačidla sa kreslí vždy iná úsečka. *Random* znamená *náhodný*:

Random(400) zvolí niektoré číslo spomedzi čísel od 0 po 399.

Random(300) zvolí niektoré číslo spomedzi čísel od 0 po 299.

Obidve náhodne zvolené čísla sa použijú ako súradnice bodu, do ktorého sa nakreslí úsečka (úsečka začína v bode (0,0)). Ak stlačíme tlačidlo druhý krát, vygenerujú sa iné náhodné čísla, takže sa nakreslí úsečka do iného koncového bodu.

Parameter, ktorý uvádzame pre **vygenerovanie náhodného čísla** udáva rozsah náhodných čísel: Random(n) náhodne zvolí číslo spomedzi čísel od 0 po n-1.

Aké čísla môže vypísať program?

```
procedure TForm1.Button7Click(Sender: TObject);
begin
   Image1.Canvas.TextOut(0, 0, IntToStr(1+Random(6)));
end;
```

Random(6) generuje čísla od 0 po 5. Ak k vygenerovanému číslu pripočítame 1, hodnota výrazu bude v rozsahu od 1 po 6.

Zhrnutie grafických príkazov

Zhrnieme všetky grafické príkazy, s ktorými sme sa už zoznámili, resp. ich doplníme o niekoľko ďalších základných. Vo všetkých príkazoch predpokladáme, že začínajú napr. **Image1.Canvas**. Ak chcete o niektorých nastaveniach alebo príkazoch vedieť viac, pozrite sa do Helpu.

Zmazanie plochy: (vhodné zaradiť hneď na začiatku programu)

- **FillRect** (Image1.ClientRect); // dvojkliknutím priamo do formulára
- **Rectangle** (-1,-1,1000,1000); // implicitne je nastavená biela výplň

Nastavenie pera a štetca:

- **Pen.Color** := clRed; // ďalšie farby sú, napr. clBlack, clGreen, clBlue, clYellow, ...
- **Pen.Width** := 3; // hrúbka pera od 1 vyššie
- **Pen.Style** := psSolid; // typ čiary - rôzne bodkované a čiarkované čiary, napr. psDash, psDot, ...
- **Brush.Color** := clWhite; // farba výplne
- **Brush.Style** := bsClear // vypni výplň – napr. pri nakreslení 2 sústredných kružníc, ak by sme zadali najprv menšiu a potom väčšiu kružnicu, (ináč by sa nakreslili s bielou výplňou a menšiu by nebolo vidno)
- **Brush.Style** := bsSolid; // zapni výplň, ďalšie sú napr. bsVertical, bsHorizontal, ...

Kreslenie čiar a útvarov:

- **MoveTo**(x, y); // presunie pero bez kreslenia čiar
- **LineTo**(x, y); // od momentálnej pozície pera kreslí úsečku do bodu (x, y)
- **Rectangle**(x1, y1, x2, y2); // obdĺžnik - teda aj štvorec
- **Ellipse**(x1, y1, x2, y2); // elipsa a teda aj kruh - parametre sú ako pri obdĺžniku a znamenajú elipsu vpísanú do obdĺžnika

Písanie textov do grafickej plochy:

- **TextOut**(x, y, 'text'); // text v apostrofoch vypíše od súradnice (x, y)
- **Font.Height** := 20; // veľkosť písma
- **Font.Name** := 'Arial'; // meno fontu
- **Font.Color** := clRed; // farba písma
- **Font.Style** := [fsBold]; // do hranatých zátvoriek sa vymenujú atribúty písma,napr. fsItalic, fsUnderline

Vlastnosti obrázka:

Image1.Width; // šírka grafickej plochy (obrázka)
Image1.Height; // výška grafickej plochy (obrázka)

Premenná, výraz, priradenie, vstup údajov

Predstavme si, že máme riešiť takéto úlohy: „Do stredu grafickej plochy chceme nakresliť domček.“ (Nakreslíme ho na tabuľu. Naznačíme súradnice). „Vidíme, že viackrát bude potrebné počítať súradnice stredu: Image1.Width **div** 2 aj Image1.Height **div** 2. Potom najprv štvorec v strede plochy môžeme nakresliť príkazom: Image1.Canvas.Rectangle(Image1.Width div 2 - 10, Image1.Height div 2 – 10, Image1.Width div 2 + 10, Image1.Height div 2 + 10); Pomohlo by nám, keby sme jednotlivé hodnoty počítali iba raz a nejako si ich zapamätali, napr.: **x** := Image1.Width div 2; a **y** := Image1.Height div 2; “. Potom príkazy na nakreslenie štvorca, ale neskôr aj ďalších častí domčeka (napr. strecha), sa zjednoduchšia a sprehľadnia:

Image1.Canvas.Rectangle(**x** - 10, **y** – 10, **x** + 10, **y** + 10);

Alebo v ďalšej úlohe chceme vytvoriť program, ktorý počíta stlačenia tlačidla. Pomohlo by nám jednotlivé stlačenia si postupne pripočítavať do nejakej premennej, ktorú by sme si mohli nazvať napr. **pocet**. Tá by na začiatku nášho programu bola samozrejme najprv vynulovaná, t.z. pocet := 0; a neskôr po každom zatlačení tlačidla by sa obsah premennej zvyšoval o jednu príkazom: pocet := pocet + 1;

Premenné v matematike majú trochu iný význam, ako v počítači: Matematická premenná označuje neznámu,

hľadanú hodnotu alebo parameter.

Programátorská ***premenná*** je miesto v operačnej pamäti počítača na zapamätanie hodnoty. Každá premenná má svoje meno – ***identifikátor***. Aby sme mohli v programe používať premenné, musíme požiadať systém o pridelenie operačnej pamäte pre ne a pomenovať ich. To všetko sa stane deklaráciou (zadefinovaním) premennej. V programe potom už len používame mená premenných a systém vie, s ktorou časťou operačnej pamäte má pracovať. Takže, ***deklarácia premennej*** je fyzické pridelenie operačnej pamäte pre premennú a dočasné pomenovanie pridelenej pamäte identifikátorom. Aby systém vedel, koľko operačnej pamäte má pre premennú zadovážiť, musíme v deklarácii uviesť jej typ (celé čísla – *Integer*, znaky – *char*, apod.). Typ premennej zároveň určuje, čo do nej smieme uložiť. Pascal používa jednoduché a zložené typy premenných. Jednoduché typy sú pre celé čísla, racionálne čísla, znaky a logické hodnoty. Zložené typy sú pre reťazec znakov, pole, súbor, záznam, množinu a packet.

Deklarácia začína slovíčkom **var** (**var**iable – premenná), za ním nasledujú identifikátory premenných oddelené od seba čiarkami, potom nasleduje dvojbodka a typ premenných, napr. **var a, b: integer;** (obe premenné **a** i **b** môžu obsahovať **iba** nejaké **celé čísla**)

Do premennej môžeme **uložiť (zapísať) jej hodnotu** – pri tejto operácii sa starý obsah premennej prepíše (navždy sa stratí) a do premennej sa uloží nová hodnota. Na to treba vždy pamätať. Do premennej uložíme hodnotu operáciou priradenia (**:=** je operátorom priradenia), napr. a := 100; b := a +20;

Priradenie je jednoduchý príkaz, ktorým sa výsledná hodnota výrazu na pravej strane od operátora priradenia zapíše do premennej, ktorej identifikátor je uvedený na ľavej strane výrazu.

Napr. **o := 2*(a+b);** Tento príkaz priradenia sa vykoná takto: najprv procesor prečíta hodnotu premennej a, potom prečíta hodnotu premennej b. Prečítané hodnoty spočíta a vynásobí dvomi. Výsledok potom zapíše do premennej o.

Na ľavej strane príkazu priradenia je vždy iba jedna premenná – cieľová. Na pravej strane je zvyčajne matematický ***výraz***, ktorý pozostáva z **operandov**, medzi ktorými stoja **matematické operátory**, napr. +, -, *, /, ... Operandom môže byť konštanta, premenná alebo funkcia.

Pomerne často sa využívajú dve celočíselné operácie:
a) celočíselné delenie (operátor **div**) - vráti vždy celú časť podielu, pričom zvyšok po delení sa zahadzuje, napr.
14 **div** 5 = 2; 736 **div** 10 = 73

b) zvyšok po delení (operátor **mod)** - vráti len zvyšok po delení, napr.
14 **mod** 5 = 4; 14 **mod** 2 = 0; 736 **mod** 10 = 6; 736 **mod** 100 = 36

Typy hodnôt

V príkaze: *Image1.Canvas.TextOut (80,50, 'Ahoj')* prvé dva parametre v príkaze výpisu musia byť **celé čísla**. Tretí, posledný parameter musí byť text, presnejšie, textový **reťazec**. V zápisoch v programe odlíšime textové reťazce od čísel alebo príkazov tým, že znaky textového reťazca napíšeme medzi apostrofy.

Čísla a texty sú pre počítač rozdielne **typy údajov**. S každým vykonávame iné úkony, inak s nimi manipulujeme. Preto napríklad v príkaze výpisu nemôžeme na mieste textového parametra napísať číslo:

- **nesprávne:** Image1.Canvas.TextOut(0, 0, 123) ... 123 je číslo
- **správne:** Image1.Canvas.TextOut(0, 0, '123') ...'123' je reťazec

Niekedy však budeme chcieť zobraziť výsledky výpočtov:

- **toto je nesprávne:** Image1.Canvas.TextOut(0, 0, 5*4+1);
- **zobrazí 5*4+1:** Image1.Canvas.TextOut(0, 0, '5*4+1');
- **vypíše 21**: Image1.Canvas.TextOut(0, 0, IntToStr(5*4+1));

IntToStr je skratka zo slov *Integer To String* = zmeň *Celé číslo Na Reťazec*. Zmena prebehne v pamäti počítača, kde sa najprv vyhodnotí aritmetický výraz a potom sa z jeho výsledku (čiže z čísla 21) vytvorí postupnosť cifier, teda reťazec '21'. Hodnota niektorých výrazov však môže byť aj **desatinné** číslo:

- **nesprávne:** Image1.Canvas.TextOut(0, 0, IntToStr(1/2));
- **správne:** Image1.Canvas.TextOut(0, 0, FloatToStr(1/2));

Pomocou **FloatToStr** zmeníme desatinné číslo na text – z čísla 0.5 vznikne '0.5'. FloatToStr je skratka zo slov *Floating point number To String* = zmeň *Číslo s desatinnou bodkou Na Reťazec*.

Aj desatinné a celé čísla sú dva rozdielne typy. Hodnota výrazu, v ktorom sa delí pomocou /, bude vždy desatinné číslo. A to aj v vtedy, keby sme delili 10/2:

- **nesprávne:** Image1.Canvas.TextOut(0, 0, IntToStr(10/2));
- **správne:** Image1.Canvas.TextOut(0, 0, FloatToStr(10/2));

V jazyku Pascal sa rozlišuje desatinné a celočíselné delenie. Celočíselné delenie sa realizuje pomocou operátora **div**:

- **vypíše 5:** Image1.Canvas.TextOut(0, 0, IntToStr(10 **div** 2));
- **vypíše 3:** Image1.Canvas.TextOut(0, 0, IntToStr(10 **div** 3));

S celočíselným delením súvisí aj zvyšok, ktorý vznikne po delení dvoch celých čísel.

Ten vieme zistiť pomocou operátora **mod**:

- **vypíše 1:** Image1.Canvas.TextOut(0, 0, IntToStr(3 **mod** 2));
- **vypíše 4:** Image1.Canvas.TextOut(0, 0, IntToStr(14 **mod** 5));

V programoch tiež môžeme používať niektoré matematické funkcie. Funkcia **sqr** počíta **druhú mocninu** čísla, napr. sqr (4) = 16. Funkcia **sqrt** počíta **odmocninu** z čísla. Výsledok bude desatinné číslo:

- Image1.Canvas.TextOut(0, 0, FloatToStr(sqrt(64)));
- Image1.Canvas.TextOut(0, 0, FloatToStr(sqrt(5*5-4*4)));

Zadávanie vstupu

Ak sme doteraz mali program, ktorý pracoval s nejakou celočíselnou premennou, tak buď sme na začiatku programu do premennej priradili nejakú konštantu, alebo sme priradili nejakú náhodnú hodnotu z určeného intervalu. Ale potom ako sa program spustil, sme už do neho nemohli zasahovať, aby pracoval s nejakou inou konkrétnou hodnotou.

Editovacie políčko na palete komponentov

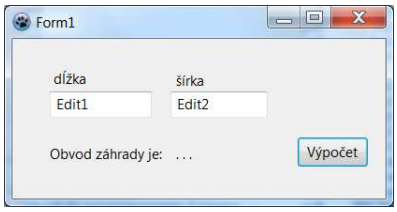
Ak v programe budeme potrebovať nastavovať hodnotu premennej aj počas behu programu môžeme použiť ďalší komponent ***vstupný riadok*** (resp. ***editovací riadok***), ktorý funguje ako jednoduchý textový editor a v programe je typu **TEdit**. Vstupný riadok funguje takto:

1) do formuláru našej aplikácie umiestnime niekde editovacie políčko (komponent vstupný riadok **TEdit**), do ktorého sa dá písať počas behu programu ľubovoľný text
2) po zatlačení tlačidla (napr. Button1) má náš program možnosť zistiť tento zapísaný text z vlastnosti *Text*. Tá nadobúda hodnotu typu string (textový reťazec) a ďalej ju spracovať, napr. tak, že môže z nej spraviť aj číslo napr. príkazom *StrToInt(Edit1.Text)* a priradiť do nejakej premennej
3) vďaka tomuto môžeme pred každým zatlačením tlačidla zadať nejaký nový text a program bude zakaždým pracovať s nejakou inou hodnotou

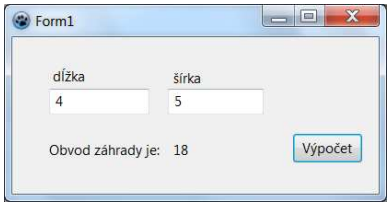
Príklad: Zostavte obsah procedúry tlačidla Button1, ktorá načíta rozmery obdĺžnikovej záhrady v metroch (dĺžka je v objekte Edit1, šírka v objekte Edit2) a vypíše do objektu Label3 koľko m pletiva treba kúpiť na oplotenie záhrady.

Riešenie: a)

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,o: integer;
begin
   a := StrToInt(Edit1.Text);
   b := StrToInt(Edit2.Text);
   o := 2*(a+b);
   Label3.Caption:=IntToStr(o);
end;
```



Po spustení aplikácie:

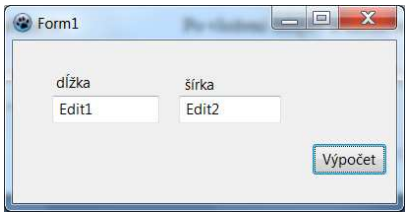


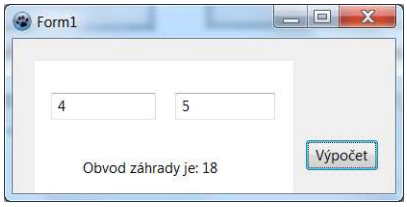
Po vložení údajov a zatlačení tlačidla:

- použité boli 4 komponenty Label (jednoduchý text): „dĺžka“, „šírka“, „ . . .“ a „Obvod záhrady je: “ , pričom Label3 sa po výpočte zmenil z „ . . .“ → „ 18 “

Riešenie: b)

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,o: integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);      // vymazanie plochy
   a := StrToInt(Edit1.Text);
   b := StrToInt(Edit2.Text);
   o := 2*(a+b);
   Image1.Canvas.TextOut(50,100,'Obvod záhrady je: ' + IntToStr(o));
end;
```





- pomocou operácie zreťazenia bol výsledok vypísaný naraz

IntToStr(*číslo*) → *text* .... vyrob z čísla text, napr. IntToStr (-37) → „-37“ – používa sa napr. pri výpise výsledku
StrToInt(*text*) → *číslo* .... vyrob z textu číslo, napr. StrToInt('-37') → -37 – pri prečítaní vstupnej hodnoty (Edit)

Farebný model RGB

Všetky farby v počítači sú namiešané z troch základných farieb: červenej, zelenej a modrej (tzv. model **RGB**, teda **R**ed, **G**reen, **B**lue). Konkrétna farba závisí od toho, koľko je v nej zastúpená každá z týchto troch farieb.

Zastúpenie jednotlivej farby vyjadrujeme číslom od 0 do 255 (zmestí sa do jedného bajtu 2^8=256), napr. žltá farba vznikne, ak namiešame 255 červenej, 255 zelenej a 0 modrej.

Na skladanie farieb máme k dispozícii funkciu **RGB**, ktorej zadáme tri čísla od 0 do 255 a ona vytvorí príslušnú farbu, napr. známe preddefinované farby majú takéto vyjadrenie:

clBlack = RGBtoColor(0, 0, 0); clRed = RGBtoColor (128, 0, 0); clGreen = RGBtoColor (0, 128, 0); clBlue = RGBtoColor (0, 0, 255); clGray = RGBtoColor (128, 128, 128); clYellow = RGBtoColor (255, 255, 0); clWhite = RGBtoColor (255, 255, 255)

Ak predpokladáme, že rôznych farieb v počítači je 256*256*256 = 16777216, tak by sme náhodnú farbu mohli generovať aj takýmto zápisom: Image1.Canvas.Brush.Color := **Random**(256*256*256);

Príklad: Zostavte program pre nakreslenie 3 farebných kruhov s rôznymi odtieňmi červenej, zelenej a modrej.

Riešenie:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Randomize;
  Image1.Canvas.Brush.Color := RGBtoColor (100+Random(156), 0, 0);
  Image1.Canvas.Ellipse (100-50, 150 – 50, 100 +50, 150 + 50);
  Image1.Canvas.Brush.Color := RGBtoColor (0, 100+Random(156), 0);
  Image1.Canvas.Ellipse (200-50, 150 – 50, 200 +50, 150 + 50);
  Image1.Canvas.Brush.Color := RGBtoColor (0, 0, 100+Random(156));
  Image1.Canvas.Ellipse (300-50, 150 – 50, 300 +50, 150 + 50);
end;
```

Jedna z farebných zložiek je náhodné číslo z intervalu ‹100,255) a zvyšné dve sú 0. Vďaka tomu sa nikdy nevygeneruje čierna farba RGBtoColor (0,0,0). Kvôli prehľadnosti sa nechali v príkaze Ellipse všetky štyri parametre v tvare výrazu, keďže **kružnica** so stredom [**x,y**] a s polomerom **r** sa kreslí pomocou **Ellipse(x-r,y-r,x+r,y+r)**.

Príkaz cyklu FOR

Jednu úsečku s náhodne umiestneným koncom sme kreslili takto:

Image1.Canvas.LineTo(Random(400), Random(300));

Ak chceme nakresliť viac, napríklad 3 úsečky, môžeme príkaz skopírovať:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Image1.Canvas.LineTo(Random(400), Random(300));
   Image1.Canvas.LineTo(Random(400), Random(300));
   Image1.Canvas.LineTo(Random(400), Random(300));
end;
```

Keby sme chceli kresliť 100 úsečiek, museli by sme veľa kopírovať. Pritom by sme sa nesmeli pomýliť v počte skopírovaných príkazov. Takýto spôsob vytvárania programov nie je ani elegantný ani efektívny.

V jazyku Pascal by sme predchádzajúci algoritmus zapísali takto:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  I: Integer;
begin
   for I := 1 to 100 do
   Image1.Canvas.LineTo(Random(400), Random(300));
end:
```

hlavička cyklu
telo cyklu
} Cyklus FOR

Príkaz čítame: *„nastavuj premennú I postupne na hodnoty od 1 po 100 a vykonávaj kreslenie náhodnej úsečky“*.

V programe najskôr deklarujeme celočíselnú riadiacu premennú cyklu I. Tá bude slúžiť ako počítadlo, koľko úsečiek sa nakreslilo a postupne sa bude zväčšovať o 1 a keď bude jej hodnota I = 100, telo cyklu sa vykoná poslednýkrát a cyklus skončí.

Cyklus je súbor príkazov, ktoré sa v programe viackrát opakujú.
Poznáme : a) cykly s pevným počtom opakovaní (cyklus FOR)
b) cykly ukončené nejakou podmienkou (cykly IF a WHILE)

Pri cykloch s pevným počtom opakovaní vieme presne povedať, koľkokrát sa má daná skupina príkazov opakovať. A to je aj náš prípad. Ide o tzv. **FOR cyklus**, ktorý vo všeobecnosti vyzerá takto:

for *premenná* := *dolná_hranica* **to** *horná_hranica* **do** *príkaz*;

ale aj takto: **for** *premenná* := *horná_hranica* ***downto*** *dolná_hranica* **do** *príkaz*;

napr. for I := -10 **to** 10 do ... // (cyklus sa vykoná 21-krát),
for I := 8 **downto** 1 do ...

Ak by dolná hranica bola väčšia ako horná (pri cykle s **to**), príkaz z tela cyklu by sa ani raz nevykonal.

Príklad: Zostavte program pre vypísanie do grafickej plochy čísla od 1 po 10 a vpravo od nich budú ich druhé mocniny.
Riešenie:

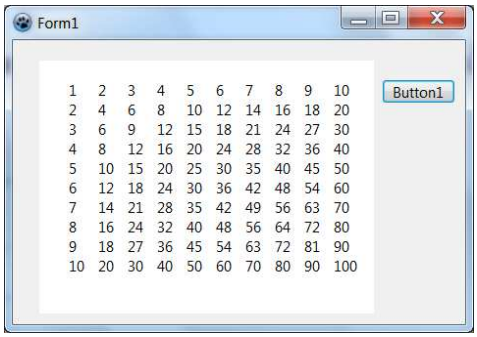
```
procedure TForm1.Button4Click(Sender: TObject);
var
  I: Integer;
begin
  for I:=1 to 10 do begin
    Image1.Canvas.TextOut(0, I*20, IntToStr(I));
    Image1.Canvas.TextOut(100, I*20, IntToStr(I*I));
  end;
end;
```

Slová **begin** a **end** fungujú ako jeden **zložený príkaz** alebo programové zátvorky. Označujú skupinu príkazov, ktoré sa majú spoločne opakovať v tele cyklu.

Telo cyklu môže v sebe obsahovať aj iný príkaz cyklu – tzv. **vnorený cyklus (cyklus v cykle)**

Príklad: Zostavte program pre vypísanie malej násobilky.
Riešenie:

```
procedure TForm1.Button5Click(Sender: TObject);
var
  I, J: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);  // zmazanie plochy
  for I:=1 to 10 do
    for J:=1 to 10 do
      Image1.Canvas.TextOut(J*30, I*20, IntToStr(I*J));
end;
```



Podmienený príkaz (príkaz vetvenia) - if

Ak sme potrebovali niekoľkokrát opakovať nejaké príkazy, pričom počet opakovaní sme vopred vedeli, použili sme programovú konštrukciu **for-cyklus**. Niekedy však vopred presne nebudeme vedieť koľkokrá máme cyklus zopakovať. Aby však mal cyklus zmysel, musí sa niekedy skončiť (vlastnosť: algoritmus musí byť konečný). Je potrebné teda počítaču nejakým spôsobom oznámiť, kedy je už počet opakovaní dostatočný.
Veľmi často sa to oznamuje pomocou podmieneného výrazu **if**. Pomocou tohto príkazu môžeme rozhodnúť, či sda nejaký príkaz, resp. postupnosť príkazov má vykonať alebo nie.

Jeho zápis je nasledovný:

```
if podmienka then
príkaz1
else
príkaz2;
```

V*norený* príkaz ***príkaz1*** sa vykoná jedine vtedy, keď výsledok podmienky je pravdivý inak sa vykoná len v*norený* príkaz ***príkaz2***. Po slovensky by sme ho mohli povedať:

ak je splnená *podmienka*, **potom** vykonaj *príkaz1*, **inak** vykonaj *príkaz2*;

Treba tomu rozumieť tak, že vždy sa vykoná len jeden z týchto dvoch príkazov a to buď iba ***príkaz1***, ak bola podmienka splnená alebo iba ***príkaz2***, ak bola podmienka nesplnená.

Napr.:

1. Program vypíše ***True***, ak je číslo 22 deliteľné číslom 7, inak vypíše ***False***

```
if  22 mod 7 = 0 then
   Image1.Canvas.TextOut (10, 50, 'True')
else
   Image1.Canvas.TextOut (10, 50, 'False');
```

2. Program do premennej max priradí to z čísel prve a druhé, ktoré je väčšie

```
if  prve > druhe then
   max : = prve
else
   max : = druhe ;
```

3. Program podľa veku dieťaťa rozhodne, či môže ísť do kina na film, ktorý je prístupný od 12 rokov

```
if  vek < 12 then
   Image1.Canvas.TextOut (10, 50, 'nemôže ísť do kina')
else
  Image1.Canvas.TextOut (10, 50, 'môže ísť do kina ');
```

4. Program sa s pravdepodobnosťou 1 : 2 rozhodne nakresliť buď štvorec alebo kruh

```
if  Random(2) = 0 then
   Image1.Canvas.Rectangle (x-r, y-r, x+r, y+r)
else
  Image1.Canvas.Ellipse (x-r, y-r, x+r, y+r);
```

Podmienený príkaz môže byť aj neúplný:

if podmienka **then**
príkaz

ak je splnená *podmienka*, **potom sa** vykoná *príkaz*, inak sa vynechá a pokračuje ďalšími príkazmi v programe

V programovacom jazyku máme k dispozícii tieto 3 základné logické operácie:

podm1 **and** *podm2* – otestovanie, či súčasne platia obe podmienky ***podm1***aj ***podm2***

podm1 **or** *podm2* – otestovanie, či platí aspoň jedna z podmienok ***podm1***a ***podm2***

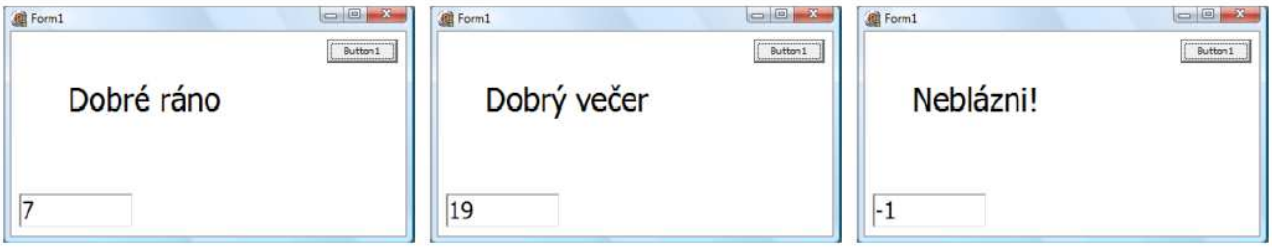
not *podm1* – otestovanie, či neplatí podmienka ***podm1***, to znamená znegovanie podmienky

Pomocou týchto operácií môžeme skladať rôzne komplikované podmienky. Vtedy si treba uvedomiť, že operandy týchto logických operácií by mali byť výrazy (najčastejšie relačné) , ktorých hodnota je **pravda** alebo **nepravda**.

Teraz napíšme program, ktorý zo zadanej hodiny zistí, aká je teraz časť dňa a podľa toho vypíše pozdrav:

```
var
  Hodin: Integer;
begin
  Hodin := StrToInt(Edit1.Text);
  Image1.Canvas.Font.Height := 40;
  if (Hodin < 0) or (Hodin > 24) then
    Image1.Canvas.TextOut(50, 50, 'Neblázni!')
  else if (Hodin >= 5) and (Hodin <= 8) then
    Image1.Canvas.TextOut(50, 50, ' Dobré ráno')
  else if (Hodin >= 9) and (Hodin <= 16) then
    Image1.Canvas.TextOut(50, 50, ' Dobrý deň')
  else if (Hodin >= 17) and (Hodin <= 21) then
    Image1.Canvas.TextOut(50, 50, 'Dobrý večer')
  else if (Hodin <= 4) or (Hodin >= 22) then
    Image1.Canvas.TextOut(50, 50, 'Dobrú noc')
end;
```

Podľa toho, akú hodinu zadáme, dostávame nejaký pozdrav:



Zložený podmienený príkaz (case)

Pascal poskytuje ešte jeden iný príkaz vetvenia, ktorý netestuje hodnotu nejakej podmienky na pravda/nepravda, ale podľa hodnoty nejakého číselného výrazu, vykoná jeden z možných výrazov. Príkaz zapisujeme takto:

case *výraz(hodnota)* **of**
hodnota1: príkaz1;
hodnota2: príkaz2;
hodnota3: príkaz3;
...
else
príkazE;
end;

(case –angl. stav). Ak je hodnota výrazu rovná napr. *hodnote2* → potom sa vykoná *príkaz 2* a pokračuje sa ďalšími príkazmi v programe, ktoré sú až za príkazom end. Ak sa nerovná hodnota výrazu ani jednej z hodnôt, potom sa vykoná *príkazE*. *Poznámka:* Taktiež môže byť ako v príkaze IF vetva else vynechaná.

Cyklus spodmienkou (while)

Príkaz zapisujeme takto:

while *podmienka* **do**
príkaz;

alebo, ak chceme opakovať postupnosť príkazov, tak podobne ako pre for-cyklus:

while *podmienka* **do**
begin
príkaz;

príkaz;
...
end;

Kde ***podmienka*** je výraz, ktorého hodnota je buď pravda alebo nepravda. Často je to porovnávanie nejakých hodnôt, napr. **X < 100** alebo **I <= N** a pod.

Procedúry (podprogramy)

Procedúra bez parametrov

S pojmom procedúra sa stretávame od začiatku programovania v Pascale: týmto mechanizmom sme začali pracovať hneď vtedy, keď sme definovali akcie, ktoré samajú vykonať pri zatlačení nejakého tlačidla. Dvojklikom na tlačidlo vo formulári sa na to automaticky pripravila konštrukcia **procedúry** (v tomto prípade tomu hovoríme procedúra na spracovanie nejakej udalosti - kliknutie na tlačidlo). Túto konštrukciu sme mohli ďalej rozpracovávať: pridávali sme deklarácie premenných a do tela procedúry sme zapisovali postupnosť nejakých príkazov, napr.:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.TextOut(Random(300), Random(200), 'Pascal');
end;
```

Okrem vytvárania takýchto nových procedúr sme pracovali aj s niektorými hotovými procedúrami: všetky grafické príkazy (napr. **Rectangle**, **MoveTo**, **TextOut**, ...) sú tiež procedúry, ktoré pre nás pripravili už skôr nejakí programátori. Vďaka tomuto nás nemusí zaujímať, ako sú naprogramované, len je pre nás dôležité vedieť, ako sa používajú. Pre väčšinu týchto príkazov sme ešte museli zadávať aj súradnice bodov, prípadne nejaký text – hovoríme im **parametre** procedúry. Pre nás je pri používaní procedúry dôležitý počet parametrov a tiež to, či musí byť príslušný parameter číslo alebo text.

Podprogramy sa zvyknú vytvárať aj z iných dôvodov, ako sme uviedli vyššie. Často pri tvorbe najmä väčších programov si celú našu úlohu najprv rozdelíme na menšie podúlohy. Potom riešime každú takúto podúlohu zvlášť a pritom dbáme na to, aby to dokopy dávalo riešenie našej pôvodnej úlohy. Často sa pri tom každá takáto podúloha dáva do samostatného podprogramu – **zadefinujeme podprogram**. Podprogram pritom dostáva **meno**, aby sa samotné riešenie úlohy mohlo na tieto **podprogramy** odvolávať práve týmto menom – budeme tomuto hovoriť **volanie podprogramu**. Znamená to, že ak na nejakom mieste programu uvedieme meno podprogramu, na tomto mieste sa použije (zavolá) samotný podprogram.

V Pascale môžu byť podprogramy dvoch typov:

- **procedúry** – postupnosť príkazov, ktoré riešia nejakú podúlohu, môžu byť bez parametrov alebo s parametrami (napr. X,Y,R a pod.)
- **funkcie** - postupnosť príkazov, ktoré majú za úlohu vypočítať nejaký výsledok - nejakú jednu hodnotu.

Keď už raz zadefinujeme nejaký podprogram, tak potom ho môžeme volať aj na viacerých miestach. Definovali sme ho raz, ale používame ho veľakrát. Ak pri ladení programu v ňom nájdeme nejakú chybu, tak ju opravíme len raz. Programátori často, keď napíšu podprogram, ho najprv odladia samostatne a až potom ho začnú používať (volať) v svojom hlavnom programe. Rozdelenie úlohy na podprogramy má ešte aj tú výhodu, že na týchto rozdelených podúlohách môže naraz pracovať viac programátorov, t.j. niektorí programátori môžu vytvárať podprogramy a iní ich vo svojich programoch (alebo aj podprogramoch) používajú.

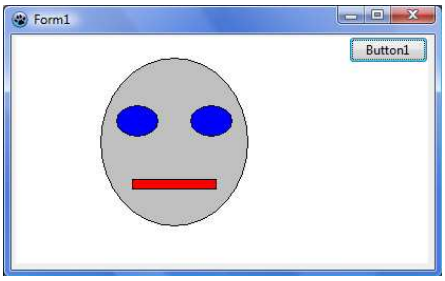
Najprv sa naučíme pracovať iba s procedúrami, t.j. podprogramami, ktoré pomenúvajú nejakú postupnosť príkazov. Pozrime sa, ako budeme definovať svoju vlastnú procedúru:

```
procedure meno;
    // deklarácie premenných pre danú procedúru
begin
    // telo procedúry
end;
```

Pozrite teraz takýto program. Pri zatlačení tlačidla **Button1**, sa niečo nakreslí. Vďaka tomu, že úlohu rozdelíme na procedúry, program sa nám bude lepšie čítať a aj rozumieť:

```
procedure TForm1.Button1Click(Sender: TObject);

procedure Zmaz;
begin
  Image1.Canvas.Brush.Color := clWhite;
  Image1.Canvas.FillRect(Image1.ClientRect);
end;

procedure Hlava;
begin
  Image1.Canvas.Brush.Color := clLtGray;
  Image1.Canvas.Ellipse(150-70, 100-80, 150+70, 100+80);
end;

procedure Oci;
begin
  Image1.Canvas.Brush.Color := clBlue;
  Image1.Canvas.Ellipse(115-20, 80-15, 115+20, 80+15);
  Image1.Canvas.Ellipse(185-20, 80-15, 185+20, 80+15);
end;

procedure Usta;
begin
  Image1.Canvas.Brush.Color := clRed;
  Image1.Canvas.Rectangle(150-40, 140-5, 150+40, 140+5);
end;

begin
  Zmaz;
  Hlava;
  Oci;
  Usta;
end;
```

Program je rozdelený na štyri procedúry: **Zmaz**, **Hlava**, **Oci** a **Usta**. Skôr, ako ho spustíme, môžeme sa dovtípiť, že nakreslí hlavu s nejakými očami a ústami. Tiež vidíme, že hlava bude šedá, oči modré a ústa červené. Teraz to spustíme:



Okrem toho, že sa tento program lepšie číta, v prípade chyby v ňom ľahšie nájdeme miesto, ktoré chceme opraviť. Napr., ak chceme na obrázku zmeniť ústa, budeme meniť len procedúru **Usta**.

Procedúra (podprogram) je **skupina príkazov** , ktorá je pomenovaná vhodným názvom a ktorá sa neskôr v hlavnom programe pomocou tohto názvu aj vyvolá a vykoná. Po ukončení procedúry sa pokračuje vo vykonávaní príkazov v hlavnom programe. Používajú sa pre uľahčenie práce pri programovaní.

Výhody používania podprogramov:

- Programy, ktoré využívajú procedúry, môžu byť zrozumiteľnejšie a aj čitateľnejšie. Namiesto dlhej postupnosti príkazov, môžeme vidieť ako programátor rozdelil veľkú úlohu na menšie podúlohy.
- Často bude program kratší, lebo sa nemusia opakovať rovnaké časti programu.
- Používanie procedúr môže pomôcť pri tvorbe náročnejších projektov:
- riešenie rozdelím na podprogramy a zvlášť potom programujeme jednotlivé časti.

- Ak sa dobre naučíme pracovať s procedúrami, potom sa nám budú takéto projekty lepšie ladiť. Ak nájdeme a opravíme chyby v podprograme, tak budú správne fungovať aj všetky jeho volania. Keď neskôr upravíme
- podprogram, zmena sa prejaví pri všetkých jeho volaniach.
- Vďaka idei podprogramov, môžeme jednoduchšie zdieľať časti programov medzi viacerými programátormi. Keď niekto naprogramuje a odladí nejakú procedúru, ostatní ju môžu používať. Takto napríklad fungujú aj knižnice príkazov (napr. grafické príkazy, matematické knižnice a pod.).

Procedúra s parametrami

Máme za úlohu nakresliť snehuliaka pomocou 3 kruhov nad sebou na presnom mieste v grafickej ploche.

Procedúru, ktorá nakrelí kruhy zadefinujeme nasledovne:

```
procedure Kruh (X, Y, R: Integer);
begin
    Image1.Canvas.Ellipse(X-R, Y-R, X+R, Y+R);
end;
```

X, Y a R v tomto zápise vystupujú ako **parametre**. Vieme si ale predstaviť, že keby sme za jednotlivé parametre dosadili konkrétne hodnoty, napríklad: X=200, Y=100 a R=30, kreslila by sa hlava snehuliaka. Pre X=200, Y=170 a R=40 by sa nakreslil stred snehuliaka. Znamená to, že náš predpis pre kreslenie ľubovoľného kruhu je všeobecný a pracuje s neznámymi hodnotami X, Y, R.

V hlavnom programe trikrát vyvoláme procedúru kruh aj s parametrami:

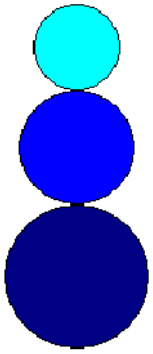
```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Kruh (200, 100, 30);
    Kruh (200, 170, 40);
    Kruh (200, 260, 50);
end;
```

Hovoríme, že premenné X, Y a R sú ***lokálne premenné*** procedúry Kruh a žiadna iná procedúra ich nemá právo používať. Po každom vyvolaní procedúry Kruh a po jej vykonaní, premenné X, Y a R potom zaniknú.

Premenné môžu byť dvojakého druhu – lokálne a globálne. ***Globálne premenné*** sú deklarované pred všetkými podprogramami, teda v prvej časti. Na rozdiel od lokálnych premenných má k ním prístup nielen hlavný program, ale aj všetky podprogramy. Globálne premenné nemusíme posielať podprogramom vo forme parametrov. ***Lokálne premenné*** deklarujeme v podprogramoch. K lokálnym premenným má prístup iba príslušný podprogram. Po ukončení vykonávania podprogramu sa priestor priradený príslušným lokálnym premenným uvoľní pre ďalšie použitie.

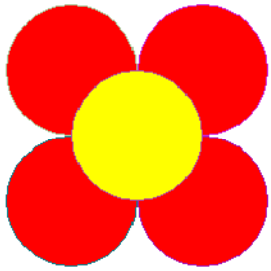
Teraz ešte vylepšíme procedúru Kruh tak, aby dokázala kresliť rôznofarebné kruhy. V programe bude potrebné upraviť aj príkazy, ktoré ju volajú. Okrem rozmerov pri volaní určíme aj farbu, akou sa kruh nakreslí:

```
procedure Kruh (X, Y, R: Integer; Farba: TColor);
begin
    Image1.Canvas.Brush.Color := Farba;
    Image1.Canvas.Ellipse (X-R, Y-R, X+R, Y+R);
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
    Kruh (200, 100, 30, clAqua);
    Kruh (200, 170, 40, clBlue);
    Kruh (200, 260, 50, clNavy);
```



Pomocou procedúry Kruh sa dá nakresliť aj takýto kvet:

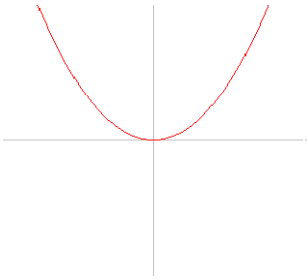
```
procedure TForm1.Kvet (x,y:integer);
var u:real; i:integer;
const n=4;                          // pocet lupenov
begin
   u:=2*pi/n;                       // uhol
   for i:=1 to  do Kruh(x+ round(20*cos(i*u)),y+round(20*sin(i*u)),20,clRed) ;
   Kruh (x,y,30,clYellow);          // žltý piestik (stred) kvetu
end;
```



Funkcie

Funkcie poznáme z matematiky. Napríklad kvadratická funkcia: $f(x)=x^2+2x+4$ má jeden **parameter x**, pre ktorý sa podľa uvedeného vzorca vypočíta **výsledná hodnota**. Programovanie vlastnej funkcie pripomína programovanie procedúry:

```
function F(X: Real): Real;
begin
Result:=X*X+2*X+4;
end;
```



Hlavička funkcie:

- začína slovom **function**, za ktorým nasleduje meno funkcie **f**,
- v zátvorkách za menom funkcie sme uviedli **meno** a typ **parametra**,
- za : sme napísali typ výslednej hodnoty (výsledkom bude desatinné číslo).

Telo funkcie obsahuje príkaz priradenia do Result. Result funguje ako lokálna premenná a číslo, ktoré do nej uložíme, bude výsledkom funkcie (*Result=výsledok*).

Našu funkciu f môžeme používať od miesta v programe, kde sme ju zadefinovali. Napríklad, skúsime vypísať funkčné hodnoty v bode 0, 1, alebo 100:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Image1.Canvas.TextOut(0, 0, FloatToStr(F(0)));
Image1.Canvas.TextOut(0, 20, FloatToStr(F(1)));
Image1.Canvas.TextOut(0, 40, FloatToStr(F(100)));
end;
```

Funkcie môžu obsahovať viacej príkazov, ktoré počítajú výsledok. Vo funkcii môžeme použiť aj známe konštrukcie – cyklus alebo podmienené príkazy.
V nasledujúcej funkcii budeme počítať súčet čísel od 1 po N. Číslo **N** bude **parametrom funkcie**. Súčet čísel od 1 po **N** v jazyku Pascal nemôžeme zapísať takto: Result:=1+2+3+4+5+ ... +**N**. Súčet ale môžeme zostaviť postupne:

```
Result:=0;
Result:=Result+1;
Result:=Result+2;
...
Result:=Result+N;
```

Znamená to, že k výsledku budeme postupne pripočítavať čísla od 1 po **N**. To dosiahneme použitím cyklu **for**.

Výsledkom takejto **funkcie** bude celé číslo:

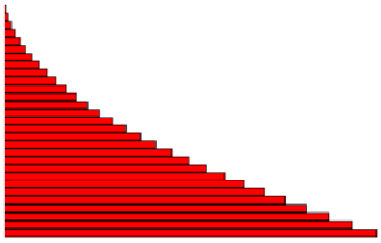
```
function Sucet(N: Integer): Integer;
var
I: Integer;
begin
Result:=0;
for I:=1 to N do Result:=Result+I;
end;
```

Súčet čísel od 1 po N vieme vypočítať aj bez použitia cyklu.

Ing. Peter Stenčík

Stĺpcovým grafom potom postupne znázorníme súčty až 30-tich čísel:

```
procedure TForm1.Button2Click(Sender: TObject);
var
N: Integer;
begin
Image1.Canvas.Brush.Color:=clRed;
for N:=1 to 30 do
Image1.Canvas.Rectangle(0, N*10, Sucet(N), N*10+10);
end;
```

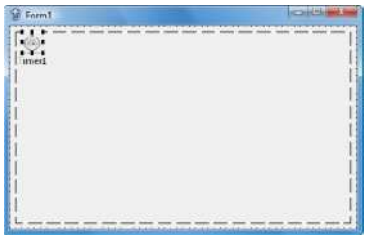


Časovač

Časovač (Timer) je komponent, ktorý v pravidelných intervaloch tiká, a tak generuje svoju udalosť OnTimer. To je výhodné, ak v programe plánujeme vykonávať periodické deje, napríklad počítať čas alebo zobrazovať animáciu.

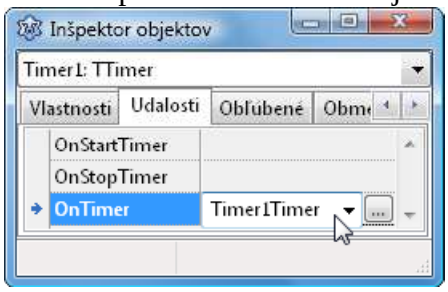
1. Do formulára vložíme obrázok a časovač. Časovač dostane meno *Timer1*.

Komponent časovač nájdeme na záložke *System*:



Časovač je nevizuálny komponent a počas behu programu ho neuvidíme. Bude sa však prejavovať svojou činnosťou – bude v pravidelných intervaloch volať procedúru.

2. Zobrazíme udalosti časovača a do prázdneho políčka *OnTimer* dvojklikneme:



3. Do programu a procedúry Timer1Timer, ktorá v programe vznikla, dopíšeme:

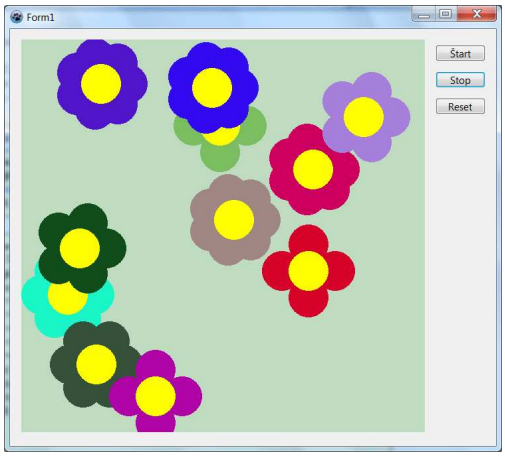
```
var
TM:  Integer = 0;
procedure TForm1.Timer1Timer (Sender: TObject);
begin
TM := TM + 1;
Image1.Canvas.TextOut (0, 0, IntToStr (TM));
end;
```

Po spustení programu uvidíme, ako sa každú sekundu mení obsah premennej **TM**.

Rýchlosť, akou časovač tiká, nastavíme v jeho vlastnosti Interval. Pôvodná hodnota 1000 znamená, že časovač tne raz za 1000ms (milisekúnd), teda raz za sekundu. Ak túto hodnotu zmeníme na 20, časovač tikne 50 krát za sekundu. Interval 20 ms budeme používať pri zobrazovaní animácií.

Nasledujucou procedúrou sa s využitím predchádzajúcej procedúry Kruh, sa budú v pravidelných časových intervaloch kresliť na náhodných súradniciach v grafickej ploche kresliť kvety s náhodným počom lupeňov s náhodnými farbami.

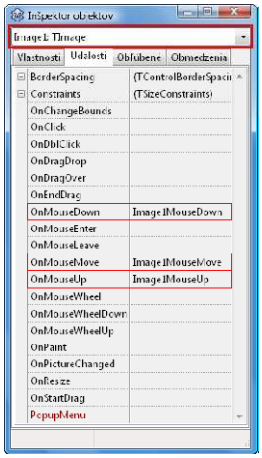
```
procedure TForm1.Timer1Timer(Sender: TObject);
var X,Y,N,I:Integer; U:real; F:TColor;
begin
  Timer1.Enabled:= true;
  X:= random(Image1.Width-100)+50;
  Y:= random(Image1.Height-100)+50;
  N:=(random(4)+4);                          // počet lupeňov (4 – 7)
  U:=2*PI/N;                                 // uhol
  F:=random(255*255*255);                    // náhodná farba lupeňov
  for I:= 1 to N do
  Kruh(x+(round(40*cos(I*U))),(Y+round(40*sin(I*U))),30,F);
  Kruh(X,Y,30,clYellow);                     // žltý piestik (stred) kvetu
end;
```



Práca s myšou

Myš je v súčasných programoch dôležité vstupné zariadenie. Pomocou myši ovládame textové a grafické editory, navigačné a riadiace systémy alebo hry. Preto sa aj my naučíme používať v našich programoch myš, naučíme sa zisťovať jej polohu, reagovať na jej pohyb alebo stlačenie či pustenie tlačidiel myši.

Pohyb myši alebo stlačenie tlačidla myši je tiež **udalosť**. 1. Na záložke *Udalosti* v okne *Inšpektora objektov* si zobrazíme zoznam udalosti, na ktoré nám umožňuje reagovať komponent obrázok Image1:



Udalosti sú pomenované – v ľavom stĺpci vidíme **meno udalosti**:

- *OnMouseDown* ... pri stlačení tlačidla myši
- *OnMouseMove* ... pri pohybe myši
- *OnMouseUp* ... pri pustení tlačidla myši

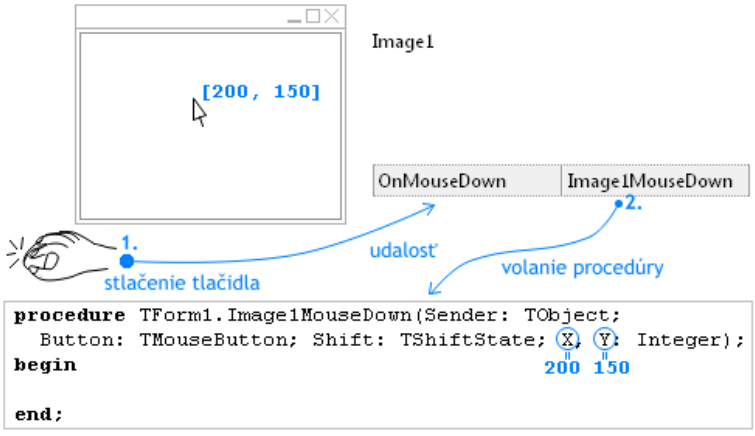
(Pozor, aj formulár má svoje udalosti:OnMouseDown,OnMouseMove,OnMouseUp).

Keď udalosť nastane, môžeme na ňu zareagovať napríklad tak, že nakreslíme obrázok. Preto sa v pravom políčku vedľa názvu udalosti zobrazuje **meno procedúry**, ktorá sa vykoná, keď udalosť nastane. Tieto políčka sú zatiaľ prázdne.

2. Dvojkliknime do prázdneho políčka vedľa udalosti OnMouseDown:

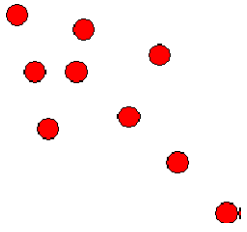
- do políčka vedľa OnMouseDown sa vygeneroval názov Image1MouseDown,
- v *Okne s programom* uvidíme, že pribudla nová prázdna procedúra.

Táto procedúra sa vykoná vždy, keď nad obrázkom stlačíme tlačidlo myši:



3. Hlavička procedúry má niekoľko parametrov. Z nich sa dozvieme **pozíciu myši** alebo aj informáciu o tom, ktoré tlačidlo používateľ stlačil. Pri zavolaní procedúry budú v parametroch X a Y uložené súradnice myši. My ich môžeme prečítať a použiť na nakreslenie červeného krúžku:

```
procedure TForm1.Image1MouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
Image1.Canvas.Brush.Color:=clRed;
Image1.Canvas.Ellipse(X-10, Y-10, X+10, Y+10);
end;
```



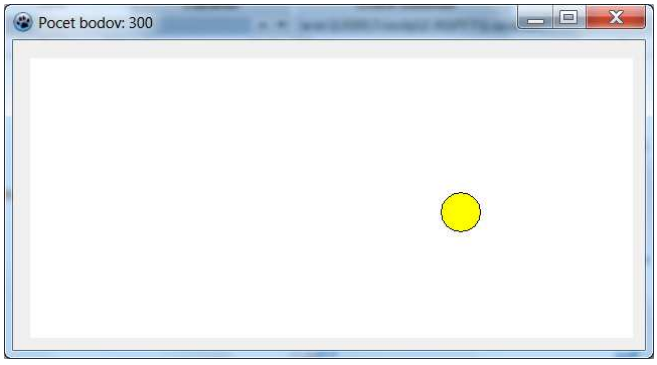
Jednoduchý kresliaci program

Vytvoríme jednoduchý kresliaci program. V programe budeme reagovať aj na ostatné spomínané udalosti myši. Pri stlačení a pri pustení tlačidla myši si poznačíme, či kreslíme. Ak áno, potom pri pohybe myši budeme kresliť čiaru.

```
var
Kreslime: Boolean=False;

procedure TForm1.Image1MouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
Kreslime:=True;
Image1.Canvas.MoveTo(X, Y);
end;

procedure TForm1.Image1MouseMove(Sender: TObject;
Shift: TShiftState; X, Y: Integer);
begin
if Kreslime then Image1.Canvas.LineTo(X, Y);
end;

procedure TForm1.Image1MouseUp(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
Kreslime:=False;
end;
```

Hra

Vytvoríme hru, v ktorej má hráč chytať zlaté dukáty. Dukáty padajú na obrazovke zhora nadol. Hráč získa body, ak stihne kliknúť na dukát skôr, ako zmizne z obrazovky:




```
var
  PozX: Integer=200; // pozicia dukatu
  PozY: Integer=-20;
  Skore: Integer=0; // skore hraca
procedure TForm1.Timer1Timer(Sender: TObject);
var  X, Y: Integer;
begin
  Image1.Canvas.Brush.Color:=clWhite; // zmazanie plochy:
  Image1.Canvas.FillRect(Image1.ClientRect);
  PozY:=PozY+1; // zmena polohy:
  if PozY>Image1.Height then begin
  Timer1.Enabled:=False; // zastavenie casovaca
  ShowMessage('Smola - koniec hry'); // zobrazenie spravy
  Close; // skoncenie programu
   end;
  Image1.Canvas.Brush.Color:=clYellow; // kreslenie:
  Image1.Canvas.Ellipse(PozX-20, PozY-20, PozX+20, PozY+20);
end;

procedure TForm1.Image1MouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Sqr(PozX-X)+Sqr(PozY-Y)<=Sqr(20) then begin
  Skore:=Skore+100; // klikli sme na dukat
  Caption:= ='Pocet bodov: ' + IntToStr(Skore);
  PozX:=random(Form1.Image1.Width);
  PozY:=-20;
  end;
end;
```