Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Programovanie 5

Predmet: Programovanie

Línia: Vlastný odborový kontext informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

Moderné vzdelávanie pre vedomostnú spoločnosť/Projekt je spolufinancovaný zo zdrojov ES

Programovanie 5

Identifikácia modulu

Aktivita projektu:	1.2 Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Vlastný odborový kontext informatiky a informatickej výchovy
Predmet:	Programovanie

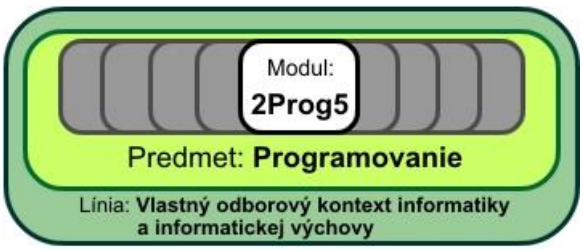
Garant predmetu:

RNDr. Andrej Blaho
KAI FMFI UK, Bratislava
andrej.blaho@gmail.com

Autori:

RNDr. Andrej Blaho, FMFI UK v Bratislave
RNDr. Ľubomír Salanci, PhD., FMFI UK v Bratislave

Zaradenie modulu



Moduly Programovanie 1 až Programovanie 9 nadväzujú na modul Programovanie 0. Všetky spolu vytvárajú ucelený kurz programovania, ktorý pokrýva stredoškolský obsah programovania aj s množstvom metodicky vhodných úloh, problémov a projektov. Všetkých 9 modulov je v prvých dvoch semestroch štúdia, nakoľko na nich nadväzujú ďalšie predmety z informatickej línie ale aj z didaktiky programovania.

Druhý semester vzdelávania:

Mat 2	DG 4	DG 5	Did. Inf. 1	Did. Inf. 2	Mod. škola 3
Prog 5	Prog 6	Prog 7	Prog 8	Prog 9	OS 1

Každý programátorský modul obsahuje 2 témy, ktoré môžu ale aj nemusia spolu súvisieť. Vyplýva to z predpokladanej organizácie štúdia, keď predpokladáme 2 štvorhodinové bloky, pričom každý bude obsahovať nejakú časť prednášky, cvičení a laboratórnej práce pri počítači.

Abstrakt modulu

Modul sa venuje pojmom procedúra, volanie procedúry a parametrom procedúr. Vysvetlí mechanizmus volania procedúr. Zároveň s tým sa ukáže aj princíp lokálnych a globálnych premenných.

Jedna z častí sa venuje aj komponentu časovač a spôsobu riešenia úloh pomocou časovača.

Obsah

Úvod

Modul sa skladá z dvoch tematických jednotiek každá približne rozsahu 3 až 5 vyučovacích hodín:

1. procedúry bez parametrov
2. procedúry s parametrami

Cieľ modulu

Prvá tematická jednotka **Procedúry bez parametrov** pokrýva tieto témy:

- procedúra bez parametrov
- mechanizmus volania procedúry
- lokálne a globálne premenné, viditeľnosť premenných
- časovač (komponent Timer)

Druhá tematická jednotka **procedúry s parametrami** pokrýva tieto témy:

- parametre prenášané hodnotou
- mechanizmus volania procedúry s parametrami
- lokálne (vnorené) a globálne procedúry
- hádanie čísla, ktoré si myslí počítač

Vstupné vedomosti

Požadované prerekvizity

Moduly Programovania 1-4

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie popísať princíp fungovania cyklov a podmienených príkazov,
- vie používať podmienené príkazy a cykly,
- dokáže analyzovať zadanie a tiež možné chyby v riešení, chyby vie nájsť a opraviť,
- dokáže vytvoriť aplikácie, ktoré používajú aj vnorené cykly s podmienenými príkazmi,
- rozumie pojmu podmienka, logická hodnota a logický výraz, a tiež princípu fungovania while-cyklu,
- rozumie spôsobu riešenia problémov pomocou reálnej aritmetiky.

Preverenie vstupných vedomostí

Vytvorenie jednoduchej aplikácie, v ktorej každé zatlačenie tlačidla na náhodnú pozíciu nakreslí nejaký obrázok, ktorý je zložený z viacerých rôznofarebných častí, napr. nejaký dopravný prostriedok, hlava, dom s oknami a pod.

1. tematická jednotka - procedúry

1. 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 sa majú 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. Okrem grafických príkazov sme používali už hotové podprogramy aj pre textovú plochu (**Clear**, **Append**) ale aj rôzne iné podprogramy, napr. **Sleep**, **Repaint**, **Close** a **Click**.

Podprogramom nazývame pomenovanie nejakého algoritmu, nejakej časti programu s tým, že túto časť môžeme vyvolať pomocou tohto mena.

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,
- **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 lokálnych premenných, konštánt a podprogramov
   //     pre danú procedúru
begin
   // telo procedúry
end;
```

Vidíme, že táto definícia sa zhoduje s procedúrou na definovanie akcie po stlačení nejakého tlačidla (napr. **Button1Click**). Uvidíme ale, že rozdiely tu predsa len nejaké sú: keď chceme definovať svoju vlastnú procedúru, musíme sa rozhodnúť na aké miesto ju vložíme a tiež meno procedúry musíme vymyslieť my – nik to za nás automaticky nespraví.

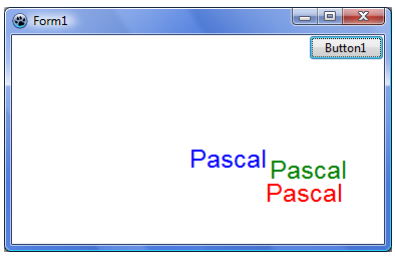
Ako prvú zadefinujeme jednoduchú procedúru s menom **VypisPascal**, ktorá na náhodnú pozíciu v grafickej ploche napíše slovo **Pascal**:

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

  procedure VypisPascal;
  begin
    Image1.Canvas.TextOut(Random(300), Random(200), 'Pascal');
  end;

begin
  Image1.Canvas.Font.Name := 'Arial';
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.Font.Color := clBlue;
  VypisPascal;
  Image1.Canvas.Font.Color := clRed;
  VypisPascal;
  Image1.Canvas.Font.Color := clGreen;
  VypisPascal;
end;
```

Všimnime si, že

- Procedúru sme definovali v tých miestach, kde sme doteraz deklarovali premenné, t.j. pred prvým riadkom **begin**.
- Definovali sme ju raz a použili (volali) sme ju trikrát – pri každom volaní, vždy keď vykonala, čo bolo treba, sa výpočet vrátil za toto volanie.
- Keď teraz spustíme program a klikneme na tlačidlo **Button1**, tak sa postupne začnú vykonávať príkazy medzi **begin** a **end** a pritom samotná procedúra **VypisPascal** sa bude vykonávať, až keď bude zavolaná.

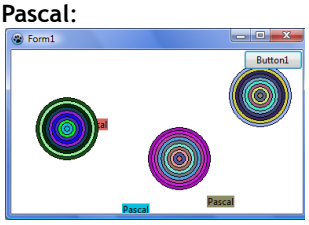
Náš program najprv nastaví atribúty písma (**Name**, **Height** a **Color**) a potom prvýkrát **zavolá** procedúru **VypisPascal**. Potom zmení farbu písma a opäť **zavolá** procedúru **VypisPascal**. Na záver zmení farbu písma na zelenú a tretíkrát **zavolá** procedúru **VypisPascal**. Teda na tri rôzne miesta tromi rôznymi farbami vypíše slovo **Pascal**.



V nasledujúcom programe pridáme novú procedúru **Terc**, ktorá na náhodnú pozíciu nakreslí 10 sústredných kružníc:

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

  procedure VypisPascal;
  begin
    Image1.Canvas.TextOut(Random(300), Random(200), 'Pascal');
  end;

  procedure Terc;
  var
    X, Y, I: Integer;
  begin
    X := Random(300) + 50;
    Y := Random(150) + 50;
    for I := 10 downto 1 do
    begin
      Image1.Canvas.Brush.Color := Random(256*256*256);
      Image1.Canvas.Ellipse(X-4*I, Y-4*I, X+4*I, Y+4*I);
    end;
  end;

begin
  Terc;
  VypisPascal;
  Terc;
  VypisPascal;
  Terc;
  VypisPascal;
end;
```

Po spustení dostaneme tri terče a tri výpisy slova **Pascal**:



Rôznych procedúr môžeme zadefinovať aj viac. Každú potom voláme jej menom.

V tomto príklade si všimnite, že procedúra **Terc** má svoje vlastné premenné **X**, **Y** a **I**. Hovoríme, že sú to jej ***lokálne premenné*** a nikto iný (žiadna iná procedúra) ich nemá právo používať.

V ďalšom príklade ukazujeme, že aj samotná procedúra **Button1Click** môže mať svoje lokálne premenné. Aby mohla procedúru **Terc** zavolať 10 krát, potrebujeme zadeklarovať premennú cyklu:

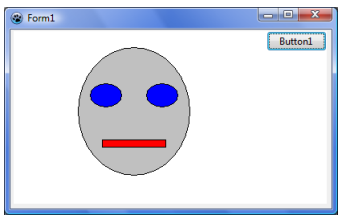
```
procedure TForm1.Button1Click(Sender: TObject);

  procedure  Terc ;
  var
    X, Y, I: Integer;
  begin
    X := Random(300) + 50;
    Y := Random(150) + 50;
    for I := 10 downto 1 do
    begin
      Image1.Canvas.Brush.Color := Random(256*256*256);
      Image1.Canvas.Ellipse(X-4*I, Y-4*I, X+4*I, Y+4*I);
    end;
  end;

var
  I: Integer;
begin
  for I := 1 to 10 do
    Terc;
end;
```

Všimnite si, že aj procedúra **Terc** má svoje lokálne premenné a aj procedúra **Button1Click** má svoju lokálnu premennú. Dokonca, obe majú premennú s rovnakým menom **I**. V Pascale to v poriadku. Každá procedúra má svoju vlastnú verziu lokálnej premennej s menom **I**.

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**.

Všimnime si, že všetky časti takto kreslenej tváre sa kreslia na absolútne pozície. Napr. hlava je elipsa sa stredom (150, 100) a s poloosami 70 a 80. Oči sú elipsy so stredmi (115, 80) a (185, 80) a ústa sú obdĺžnik so stredom (150, 140). Keby sme takúto tvár potrebovali nakresliť na iné súradnice, napr. stred hlavy na (X, Y), prepíšeme všetky procedúry vzhľadom na toto X a Y:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;

  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(X-70, Y-80, X+70, Y+80);
  end;

  procedure Oci;
  begin
    Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Ellipse(X-35-20, Y-20-15, X-35+20, Y-20+15);
    Image1.Canvas.Ellipse(X+35-20, Y-20-15, X+35+20, Y-20+15);
  end;

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

begin
  Zmaz;
  X := 100;
  Y := 100;
  Hlava;
  Oci;
  Usta;
  X := 190;
  Y := 200;
  Hlava;
  Oci;
  Usta;
  X := 260;
  Y := 90;
  Hlava;
  Oci;
  Usta;
end;
```

Po spustení dostávame:

Všimnite si, ako sme zadeklarovali dve lokálne premenné **X** a **Y**: ich deklarácia je ešte pred procedúrami **Hlava**, **Oci** a **Usta**. Vďaka tomuto, všetky tieto procedúry **vidia** obe tieto premenné. Zapamätajte si: premenné, ktoré sú deklarované v nejakej procedúre, sú **viditeľné** aj vo všetkých procedúrach vo vnútri tejto procedúry. Podmienkou je ale to, aby premenné boli definované skôr, ako procedúry a zároveň by tieto procedúry nemali mať definované svoje lokálne premenné s rovnakým menom.

Zhrňme teraz výhody podprogramov ako sme mohli vidieť v predchádzajúcich príkladoch:

- 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.).

V tomto kurze budeme programovať aj malé jednoúčelové podprogramy a to hlavne preto, aby sme sa ich naučili vytvárať a aj používať. V praxi ale programátori nevytvárajú procedúry iba na jedno použitie, ktoré iba predlžujú kód.

Úlohy na precvičenie

Zadanie 1	Program nakreslí hlavu robota: hranatá škatuľa, na ktorej sú štvorcové oči a nos a obdĺžnikové ústa. Časti tváre nech kreslia procedúry: **Hlava**, **Oci**, **Nos** a **Usta**.
Zadanie 2	Zatlačenie tlačidla zmaže plochu na modro a nakreslí nočnú oblohu s 10 malými žltými hviezdičkami na náhodných pozíciách. Hviezdičky kreslite pomocou procedúry **Hviezda**.

2. Mechanizmus volania procedúry

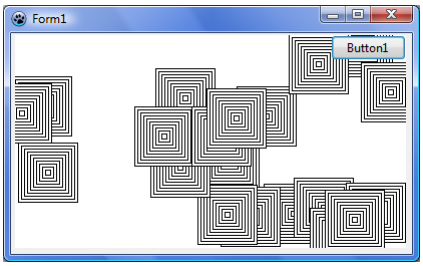
Aby sme lepšie rozumeli, ako prebieha volanie procedúr, podrobne si vysvetlíme mechanizmus, ktorý prebehne pri volaní každého podprogramu. Vieme, že **volanie procedúry** znamená, že sa pri vykonávaní programu príde na riadok s menom nejakého podprogramu. Vtedy sa postupuje nasledovne:

1. **zapamätá sa návratové miesto**, t.j. riadok, kam sa bude treba vrátiť po skončení procedúry (vo vnútri počítača sa toto realizuje tak, že sa zapamätá adresa príkazu v pamäti, kde sa bude pokračovať vo vykonávaní po návrate z procedúry);
2. **vytvoria sa lokálne premenné** procedúry (so zatiaľ nedefinovanou hodnotou) - napr. v procedúre **Terc** sú to premenné **X**, **Y**, **I**;
3. prenesie sa **riadenie** programu **do tela podprogramu** (za príslušný **begin**);
4. vykonajú sa všetky príkazy podprogramu (až po koncový **end**);
5. **zrušia sa lokálne premenné** - "zabudnú" sa lokálne premenné **X**, **Y**, **I**;
6. **riadenie sa vráti** za miesto v programe, odkiaľ bol podprogram volaný - t.j. **na návratové miesto**.

Tieto pravidlá volania procedúr vysvetľujú, ako je to s lokálnymi premennými, ktoré sú nielen v hlavnej procedúre, napr. **Button1Click**, ale aj v nej definovaných procedúrach. Pozrime sa na ukážku, v ktorej máme deklarované dve premenné s rovnakým menom:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;

  procedure Stvorce;
  var
    I, X, Y: Integer;
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    for I := 10 downto 1 do
      Image1.Canvas.Rectangle(X-3*I, Y-3*I, X+3*I, Y+3*I);
  end;

begin
  for I := 1 to 20 do
    Stvorce;
end;
```

Procedúra **Stvorec** vidí nielen svoje tri lokálne premenné **I**, **X**, **Y**, ale aj premennú **I**, ktorá je v procedúre **Button1Click**. Hoci ju vidí, ale pracovať s ňou v tomto prípade nemôže, lebo jeho vlastné **I** prekryje toto skôr definované **I**. Program nakreslí takýto obrázok:



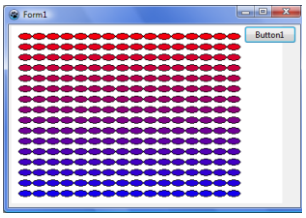
Aj ďalší program ukazuje použitie lokálnych premenných:

```
procedure TForm1.Button1Click(Sender: TObject);
const
  N = 16;
var
  Y: Integer;

  procedure Retaz;
  var
    I: Integer;
  begin
    for I := 1 to N do
      Image1.Canvas.Ellipse(20*I-10, Y-5, 20*I+10, Y+5);
  end;

var
  J: Integer;
begin
  for J := 1 to N do
  begin
    Image1.Canvas.Brush.Color := RGBToColor(260 - 15*J, 0, 15*J);
    Y := 15*J;
    Retaz;
  end;
end;
```

V tomto programe si všimnite, že sme použili konštrukciu **RGBToColor**, ktorá z troch čísel poskladá farbu v modeli **RGB**. V prostredí Delphi sme to robili pomocou funkcie RGB, v prostredí Lazarus je namiesto toho **RGBToColor**.

Zadeklarovali sme tu konštantu N a premennú **Y** ešte pred definíciou procedúry **Retaz**. To bude znamenať, že oba tieto identifikátory budú viditeľné aj vo vnútri procedúry **Retaz** a tá s nimi môže pracovať, ako keby to boli jej premenné. Premennú **J** sme deklarovali až za touto procedúrou a teda nebude v procedúre **Retaz** viditeľná. Procedúra **Retaz** si zase definuje premennú **I**, ktorú ale samotná **Button1Click** nevidí, lebo je jej lokálna a teda existuje len počas volania **Retaz**. Procedúra **Retaz** nakreslí rad **N** malých elíps veľkosti 20x10, pričom stredy týchto elíps majú v premennej **Y** svoju y-ovú súradnicu.

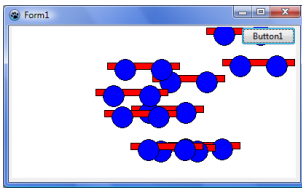
Samotný program nakreslí **N** takýchto reťazí (každá bude mať po **N** elíps). Každý rad bude zafarbený inou farbou:



V nasledovnom programe ukážeme niekoľko špecifík. Procedúra **Vozik** nakreslí na súradnice **X**, **Y** vozíček, ktorý je zložený z dvoch kolies (kruhov) a jednej dosky (obdĺžnika). Všimnite si, že procedúra **Vozik** volá ďalšie procedúry **Doska** a **Koleso**:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;

  procedure Doska;
  begin
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.Rectangle(X, Y, X + 100, Y - 10);
  end;
  procedure Koleso;
  begin
    Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Ellipse(X - 15, Y - 15, X + 15, Y + 15);
  end;
  procedure Vozik;
  begin
    Doska;
    X := X + 25;
    Koleso;
    X := X + 50;
    Koleso;
  end;

begin
  X := Random(300);
  Y := Random(200);
  Vozik;
end;
```

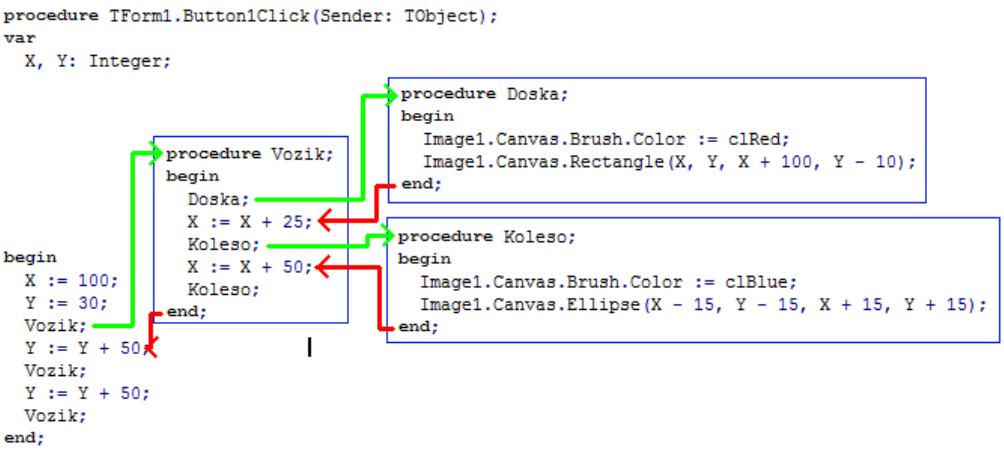
Po spustení programu môžeme tlačidlo zatlačiť viackrát:



Procedúry **Doska** a **Koleso** sú len pomocné na kreslenie vozíka v procedúre **Vozik**.

Všimnite si, že obe tieto procedúry **Doska** aj **Koleso** musia byť zadefinované ešte pred procedúrou **Vozik**, lebo táto ich chce volať a teda tieto dva identifikátory musia byť pre ňu viditeľné.

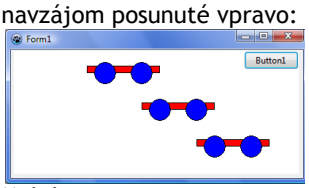
Nasledovný obrázok ukazuje, ako sa riadenie programu prenáša do volanej procedúry (zelené šípky) a ako sa riadenie vracia na zapamätané návratové miesto:



Teraz program pozmeňme tak, že najprv sa nakreslí vozík na súradnice (100, 30), pod neho sa nakreslí ďalší vozík o 50 nižšie a ešte jeden znovu o 30 nižšie:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;

  ...

  procedure Vozik;
  begin
    Doska;
    X := X + 25;
    Koleso;
    X := X + 50;
    Koleso;
  end;

begin
  X := 100;
  Y := 30;
  Vozik;
  Y := Y + 50;
  Vozik;
  Y := Y + 50;
  Vozik;
end;
```

Všimnite si, že kvôli vykresľovaniu kolies v procedúre **Vozik**, meníme obsah premennej **X**. Preto nie sú tri vozíky nakreslené presne pod sebou, ale sú navzájom posunuté vpravo:



Mali by sme si to zapamätať, lebo kreslenie vozíka budeme potrebovať aj neskôr v ďalších častiach.

Úlohy na precvičenie

Zadanie 1	Program po zatlačení tlačidla nakreslí na náhodné pozície 30 farebných štvorčekov: 10 z nich bude červených, 10 modrých a 10 žltých. Zadefinujte a použite takéto 4 procedúry: **Stvorcek**, **Cerveny**, **Modry** a **Zeleny**.
Zadanie 2	Program do jedného riadku grafickej plochy nakreslí náhrdelník farebných štvorčekov, krúžkov a elíps. Ich farby budú náhodné a tiež sa podľa náhodného generátora rozhoduje, v akom poradí za sebou idú. Veľkosť štvorčekov a krúžkov je 10, veľkosť elipsy je 15x10. Zadefinujte a využite procedúry **Stvorcek**, **Kruzok** a **Elipsa**.

3. Lokálne a globálne premenné, viditeľnosť premenných

Chceme riešiť takúto úlohu: keď klikneme na tlačidlo prvýkrát, tak sa nakreslí modrý štvorček na súradniciach (10, 100), keď klikneme druhýkrát, plocha sa zmaže a štvorček sa nakreslí na súradniciach (12, 100). Pri každom ďalšom kliknutí sa nakreslí posunutý o 2 vpravo. Napíšme prvú verziu, ktorá ešte nebude fungovať správne:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;

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

  procedure Stvorec;
  begin
    Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Rectangle(X, Y, X+40, Y+40);
  end;

begin
  Zmaz;
  X := 10;
  Y := 100;
  Stvorec;
  X := X + 2;
end;
```

Žiaľ, toto riešenie nefunguje: štvorec je stále na tých istých súradniciach. Čo sa deje s našimi premennými **X** a **Y**? Vždy, keď zatlačíme tlačidlo **Button1**, automaticky sa zavolá procedúra **Button1Click**. O volaní procedúry vieme, že na začiatku sa im vytvoria ich lokálne premenné (najprv s nedefinovanou hodnotou). Potom sa postupne vykonajú všetky príkazy a teda aj priradenia **X:=10** a **Y:=100**. Na konci procedúry (pri **end**) sa zrušia všetky lokálne premenné – "naveky" sa stratí ich hodnota – teda aj informácia o tom, že sme **X** zväčšili o 2.

Takže toto bolo zlé riešenie. Tu by nám pomohlo to, keby sme lepšie poznali, ako je to s viditeľnosťou premenných. Teraz uvedieme pravidlá, ktoré v Pascale hovoria o identifikátoroch premenných a identifikátoroch podprogramov:

- Hovoríme, že identifikátor (premennej alebo podprogramu) je definovaný v nejakej úrovni:
  - všetky štandardné identifikátory (napr. preddefinované grafické príkazy, matematické funkcie a pod.) sú definované v **0. úrovni**;
  - procedúry, ktoré spracovávajú akcie od udalostí (napr. kliknutie tlačidlá Button1Click, Button2Click a pod.), sú na **1. úrovni**;
  - všetky premenné a procedúry definované v procedúrach na 1. úrovni (tzv. vnorené procedúry) sú na **2. úrovni** (napr. naše procedúry **Terc**, **Vozík**, **Zmaz** a pod.);
  - lokálne premenné v týchto našich procedúrach sú už na **3. úrovni**.
- Každý identifikátor sa môže používať, až potom, keď bol zadeklarovaný, resp. zadefinovaný.
- Identifikátor je viditeľný v tej procedúre (na tej úrovni), v ktorej bol definovaný, ale aj vo všetkých v nej vnorených procedúrach.

O lokálnych premenných procedúry už vieme, že vznikajú pri volaní procedúry, existujú počas behu procedúry (napr. počas behu **Button1Click**) a zanikajú pri skončení procedúry. Takže lokálne premenné nám v prípade posúvaného modrého štvorca nepomôžu. Tieto sú na 2. úrovni.

Pomôžu nám premenné, ktoré zadefinujeme na 1. úrovni. Takéto premenné nazývame **globálne premenné**. Majú takéto špeciálne vlastnosti:

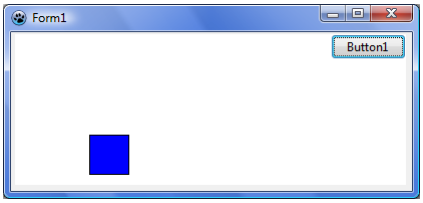
- vznikajú pri štarte programu,
- existujú so svojimi hodnotami počas celého behu programu,
- môžeme im nastaviť ich počiatočné hodnoty už pri deklarácii – ak to nespravíme, majú nulové hodnoty.

Tieto globálne premenné sú viditeľné vo všetkých procedúrach, ktoré sú v programe definované neskôr ako tieto premenné. Opravíme program so štvorcami tak, aby pracoval správne:

```
var
  X: Integer = 10;
  Y: Integer = 100;

procedure TForm1.Button1Click(Sender: TObject);

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

  procedure Stvorec;
  begin
    Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Rectangle(X, Y, X+40, Y+40);
  end;

begin
  Zmaz;
  Stvorec;
  X := X + 2;
end;
```

Takže, premenné **X** a **Y** sú **globálne** a preto sa vytvorili už pri štarte programu s hodnotami 10 a 100. Všimnite si, že tieto inicializačné hodnoty sme neprirad'ovali priraďovacími príkazmi, ale pomocou znaku rovná sa hneď za menom typu **Integer**.

Pri prvom zatlačení tlačidla **Button1** sa zmaže plocha a štvorec sa nakreslí na súradnice 10, 100. Potom sa premenná **X** zvýši o 2 a procedúra **Button1Click** skončí. Vďaka tomu, že **X** a **Y** sú globálne, ich hodnoty sa uchovajú aj pre ďalšie volania **Button1Click**. Preto sa štvorec pri každom zatlačení tlačidla bude prekresľovať na stále na nových a nových súradniciach – bude sa pomaly posúvať smerom vpravo:

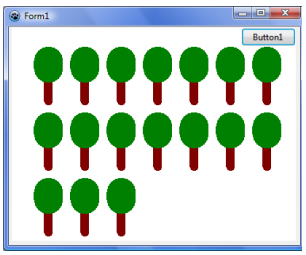


V ďalšom príklade ukážeme okrem globálnych premenných a procedúry aj podmienený príkaz. Vytvoríme procedúru **Strom**, ktorá pomocou dvoch hrubých čiar nakreslí strom. Strom sa bude kresliť tak, že najprv sa od pozície (X, Y) nakreslí hrubý hnedý kmeň a potom ešte hrubším perom krátka zelená úsečka.

Pri prvom zatlačení tlačidla nakreslíme strom na pozície (50, 100). Každé ďalšie zatlačenie tlačidla nakreslí strom posunutý o 50 vpravo (zväčší sa **X** o 50). Ak by sa ale nasledovný strom do tohto radu nezmestil, lebo sme už na pravom okraji plochy, nastavíme sa na začiatok ďalšieho radu (**Y** zväčšíme o 90 a **X** na 50). Program vyzerá takto:

```
var
  X: Integer = 50;
  Y: Integer = 100;

procedure TForm1.Button1Click(Sender: TObject);

  procedure Strom;
  begin
    Image1.Canvas.Pen.Color := clMaroon;
    Image1.Canvas.Pen.Width := 12;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X, Y-30);
    Image1.Canvas.Pen.Color := clGreen;
    Image1.Canvas.Pen.Width := 40;
    Image1.Canvas.MoveTo(X, Y-45);
    Image1.Canvas.LineTo(X, Y-55);
  end;

begin
  Strom;
  X := X + 50;
  if X > Image1.Width-30 then
  begin
    Y := Y + 90;
    X := 50;
  end;
end;
```

Po niekoľkých zatlačeniach tlačidla:



Úlohy na precvičenie

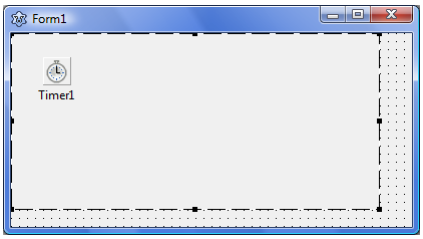
Zadanie 1	Program pri prvom zatlačení tlačidla nakreslí v strede plochy malý farebný kruh veľkosti 10. Každé ďalšie zatlačenie ho nakreslí o 10 väčší.
Zadanie 2	Program už pri štarte (v procedúre **FormCreate**) vypíše v strede slovo **Pascal**. Vo formulári sú ešte 4 tlačidlá s textami: **Vľavo**, **Vpravo**, **Hore** a **Dole**. Každé ich zatlačenie posunie tento text v príslušnom smere o 5. Napr. tlačidlo **Hore** zmenší **Y** o 5, zmaže obrazovku a slovo **Pascal** vypíše na tieto nové súradnice.
Zadanie 3	Program pri každom zatlačení tlačidla nakreslí ďalší stále menší a menší sústredný kruh.
Zadanie 4	Okrem globálnych premenných **X**, **Y** pre nakreslenie vozíka si v ďalších štyroch globálnych premenných **X1**, **Y1**, **X2**, **Y2** budeme uchovávať informácie o dvoch vozíkoch - tieto sa budú pohybovať paralelne. Napíšte program, v ktorom budú počiatočné súradnice vozíkov (10, 100) a (10, 200). Po zatlačení tlačidla sa vozíky posunú o náhodnú hodnotu z intervalu <0, 5> (napr. **X1 := X1 + Random(6);**) Takto môžeme zorganizovať preteky autíčok.

4. Časovač (komponent Timer)

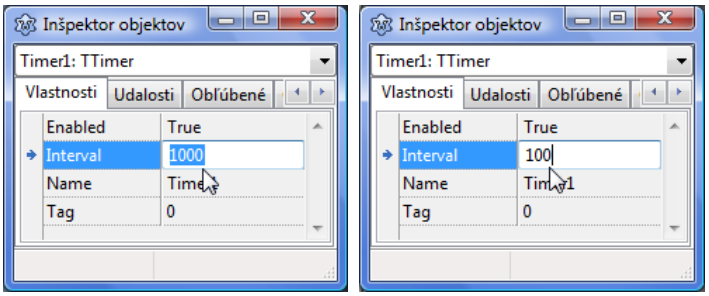
Už sme predtým viackrát riešili úlohy, v ktorých sme predpokladali, že po spustení používateľ bude musieť tlačiť tlačidlo veľakrát za sebou a takto sa postupne bude v grafickej ploche niečo diať. Napríklad sme náhodne kreslili nejaké objekty, hýbali sme vozík alebo štvorček, kreslili sme stále sa zväčšujúce alebo zmenšujúce kružnice alebo štvorce.

Pascal poskytuje mechanizmus, pomocou ktorého môžeme v našom programe nejakú akciu automaticky nechať stále vykonávať v nejakých časových intervaloch. Môžeme si to predstaviť tak, že v počítači sú špeciálne hodinky, ktorým môžeme nastaviť rýchlosť tikania (napr. každú sekundu, alebo každú stotinu sekundy a pod.) a zároveň môžeme týmto hodinkám povedať, čo majú robiť pri každom tiknutí. Sem teraz môžeme prekopírovať akciu, ktorú sme pred tým mali pre zatlačenie tlačidla a odteraz tieto akcie budú fungovať bez nášho zásahu.

Tento mechanizmus sa nazýva **časovač**. Aby mohol časovač fungovať v našom projekte, musíme do formulára vložiť špeciálny komponent **Timer.** Nájdeme ho v záložke komponentov **[System]** a je to komponent s obrázkom hodín  a s vnútorným menom **Timer**. Položíme ho na ľubovoľné miesto do formulára (aj tak ho po spustení programu vidieť nebude), napr. takto



Teraz nastavíme rýchlosť tikania: prepneme sa do Inšpektora objektov a zmeníme nastavenie Interval – toto nastavenie hovorí, koľko milisekúnd bude medzi jednotlivými tikaniami:



Číslo 1000, ktoré je tam na začiatku, označuje 1000 milisekúnd, teda 1 sekundu medzi tikaniami. Nám sa častejšie bude hodiť menšia hodnota. Nastavme sem číslo 100 – označuje 10 tiknutí za sekundu.

Teraz dvojklikneme do nášho nového komponentu vo formulári a pripraví sa nám procedúra na **spracovanie udalosti od časovača** – táto procedúra bude potom automaticky spúšťaná pri každom tiknutí:

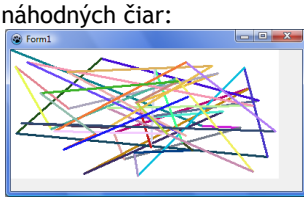
```
procedure TForm1.Timer1Timer(Sender: TObject);
begin

end;
```

Všimnime si meno tejto procedúry: Timer1Timer označuje, že je to udalosť pri tiknutí pre časovač Timer1.

Ako náš prvý pokus tam zadajme:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Canvas.Pen.Color := Random(256*256*256);
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.LineTo(Random(Image1.Width), Random(Image1.Height));
end;
```

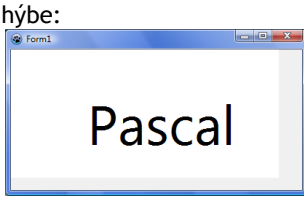
A hneď po spustení sa naštartuje kreslenie náhodných čiar:



V našom druhom programe rozhýbeme na obrazovke slovo **Pascal**. Využijeme jednu pomocnú globálnu premennú **X**, v ktorej si značíme momentálnu X-ovú súradnicu vypisovaného textu. Samotná procedúra najprv zmaže grafickú plochu, vypíše do nej nejaký text a posunie **X** o nejakú hodnotu vpravo:

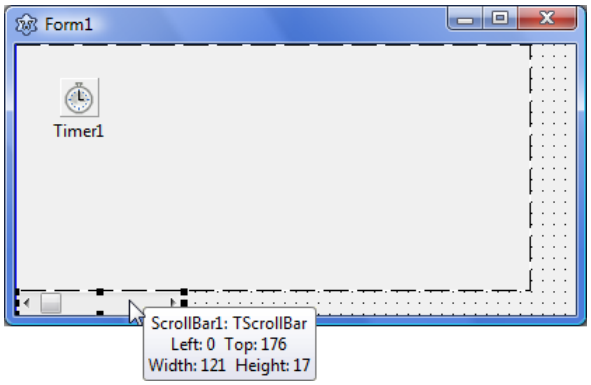
```
var
  X: Integer;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 100;
  Image1.Canvas.TextOut(X, 50, 'Pascal');
  X := X + 5;
  if X > Image1.Width then
    X := 0;
end;
```

Text, ktorý sa v ploche hýbe:



Časovač nastavme na **Interval** = 100 a po spustení programu sa automaticky rozbehne pohyb slova **Pascal**.

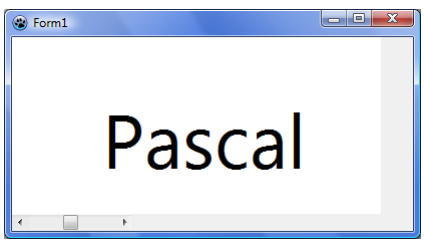
V predchádzajúcom module sme sa zoznámili s komponentom posúvač (**ScrollBar**) a pomocou neho sme menili parametre zobrazovaných grafov. Vždy keď používateľ posunie bežca na posúvači, zavolá sa procedúra na spracovávanie udalosti **ScrollBar1Change**. Momentálnu pozíciu bežca na posúvači zistíme pomocou **ScrollBar1.Position**. Teraz si ukážeme, ako využiť posúvač na zmenu rýchlosti časovača. Do formulára okrem časovača vložíme aj komponent posúvač, napr. takto:



Teraz dvojklikneme na tento komponent a dopíšeme procedúru **ScrollBar1Change**:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
 Timer1.Interval := ScrollBar1.Position * 10 + 10;
end;
```

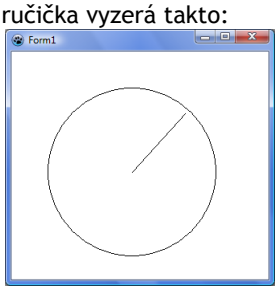
Po naštartovaní sa opäť pohybuje slovo **Pascal** zľava doprava:



Hýbaním bežca spomaľujeme, resp. zrýchľujeme pohyb textu po ploche.

Z minulého modulu vieme kresliť kružnicu pomocou matematických funkcií **Sin** a **Cos**. Teraz to využijeme na zobrazenie pohybu sekundovej ručičky. Časovaču nastavíme **Interval** = 1000, t.j. presne na jednu sekundu. V globálnej premennej **Uhol** si budeme pamätať uhol, pod ktorým sa kreslí ručička:

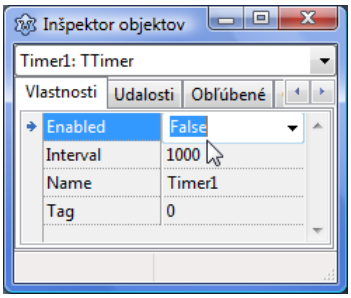
rozbehnutá sekundová ručička vyzerá takto:



Funkcie **Sin** a **Cos** pracujú v radiánoch, preto ich argument prepočítavame pomocou vzorca **Uhol/180*Pi**, aby sme mohli pracovať so stupňami. Keďže chceme, aby ručička obehla celý kruh za 60 tikov časovača, uhol meníme po 6 stupňoch.

```
var
  Uhol: Integer;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Ellipse(45, 45, 255, 255);
  Image1.Canvas.MoveTo(150, 150);
  Image1.Canvas.LineTo(150+Round(100*Sin(Uhol/180*Pi)),
                       150-Round(100*Cos(Uhol/180*Pi)));
  Uhol := Uhol + 6;
end;
```

V ďalšom programe ukážeme, ako skonštruovať vlastnú časomieru. Najprv spravíme jednoduchšiu verziu, v ktorej počítame len celé sekundy. V inšpektore objektov časovaču nastavíme **Interval = 1000** (frekvencia každú sekundy) a zároveň nastavíme **Enabled = False**.



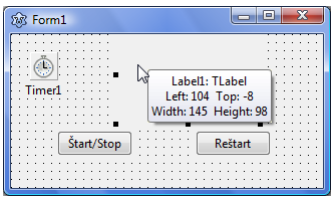
Nastavenie **Enabled** pre časovač na **False** označuje, že časovač nebeží, ale čaká na naštartovanie. Časovač teraz musíme naštartovať v programe, napr. pri zatlačení tlačidla môžeme zapísať:

Timer1.Enabled := True;

a tým časovač spustiť. Keď budeme zase potrebovať časovač zastaviť, napíšeme:

Timer1.Enabled := False;

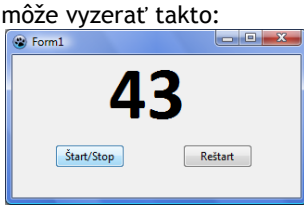
V programe použijeme jednu globálnu premennú **Sekundy**, ktorú budeme v procedúre **Timer1Timer** (pri každom "tiknutí" časovača) zvyšovať o 1 a vypisovať pomocou komponentu ***jednoduchý text*** (**TLabel**). Tento komponent položíme niekde do stredu formulára a a v inšpektore objektov mu nastavíme nejaký zväčšený font (napr. veľkosti 60). Okrem toho potrebujeme jedno tlačidlo na spúšťanie a zastavovanie časomiery (tlačidlo **Button1** bude mať popis **Caption = Štart/Stop**). Druhé tlačidlo bude nulovať počítadlo, **Button2** bude mať popis **Caption = Reštart**. Formulár pripravíme takto:



a program:

```
var
  Sekundy: Integer;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Sekundy := Sekundy + 1;
  Label1.Caption := IntToStr(Sekundy);
end;

procedure TForm1.Button1Click(Sender: TObject);    // Štart/Stop
begin
  Timer1.Enabled := not Timer1.Enabled;
end;

procedure TForm1.Button2Click(Sender: TObject);    // Reštart
begin
  Sekundy := 0;
  Timer1.Enabled := True;
end;
```

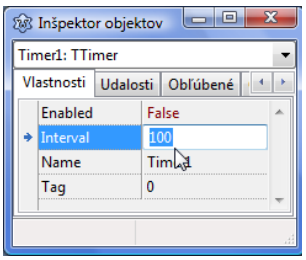
Po spustení a stlačení niektorého z tlačidiel to môže vyzerať takto:



Všimnite si zápis, ktorým sa štartuje a zastavuje časovač:

Timer1.Enabled := not Timer1.Enabled;

V tomto priradení, ak malo **Timer1.Enabled** hodnotu **False** (časovač zatiaľ stojí), tak sa mu priradí znegovaná hodnota (**not Timer1.Enabled**) teda **True** a tým sa časovač spustí. Ak má **Timer1.Enabled** hodnotu **True** (časovač beží), tak sa priradí znegovaná hodnota, teda **False** a tým časovač zastaví.

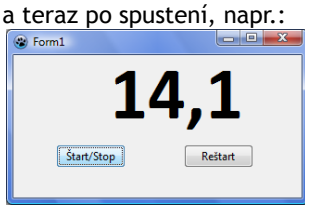


Teraz môžeme tento program vylepšiť aj o desatiny sekúnd. Pridáme ešte jednu globálnu premennú **Desatiny**, v ktorú budeme každú desatinu sekundy zväčšovať o 1 a keď dosiahne 10, tak ju vynulujeme a premennú Sekundy zvýšime o jedna. Takže nezabudneme zmeniť **Interval = 100** (každú desatinu sekundy).

Program bude teraz vyzerať takto:

```
var
  Sekundy, Desatiny: Integer;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Desatiny := Desatiny + 1;
  if Desatiny > 9 then
  begin
    Sekundy := Sekundy + 1;
    Desatiny := 0;
  end;
  Label1.Caption := IntToStr(Sekundy) + ',' + IntToStr(Desatiny);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Enabled := not Timer1.Enabled;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Sekundy := 0;
  Desatiny := 0;
  Timer1.Enabled := True;
end;
```

a teraz po spustení, napr.:



V skutočnosti by nám mala stačiť jediná globálna premenná **Desatiny**, v ktorej budeme počítať sekundy (**Desatiny div 10**) aj desatiny sekúnd (**Desatiny mod 10**). Opravte tento program.

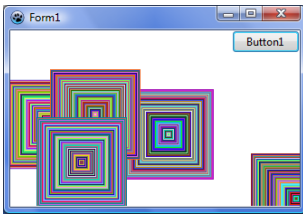
Teraz ukážeme riešenie jedného jednoduchého príkladu najprv bez časovača a

potom aj pomocou časovača. Obe riešenia potom porovnáme.

Program bude po kliknutí na tlačidlo kresliť 50 zmenšujúcich sa štvorcov s rovnakým stredom, ktorý sa vygeneruje náhodne. Farba každého štvorca bude tiež náhodná. Nechceme ale, aby sa vykreslili naraz všetky, ale aby sa vykresľovali postupne od najväčšieho po najmenší a medzi tým by sa počkalo napr. 100 milisekúnd.

Najprv to budeme programovať bez časovača. Po nakreslení každého štvorca použije nám už známu dvojicu príkazov: **Sleep(100); Image1.Repaint;** a tým na malú chvíľu pozdržíme výpočet:

Po niekoľkých zatlačeniach tlačidla dostávame:



Hoci sa táto úloha dala riešiť for-cyklom: **for A:=50 downto 1 do**, program sme zapísali while-cyklom, aby sa to ľahšie prepisovalo pre časovač. Uvedomte si, že v premennej **A** si uchovávame informáciu o tom, koľko štvorcov ešte zostáva nakresliť.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, A: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  A := 50;
  while A > 0 do
  begin
    Image1.Canvas.Pen.Color := Random(256*256*256);
    Image1.Canvas.Rectangle(X-A, Y-A, X+A, Y+A);
    Sleep(100);
    Image1.Repaint;
    A := A - 1;
  end;
end;
```

Všimnite si, ako sa správa aplikácia počas kreslenia 50 štvorcov: ak chcete okno posunúť, ak chcete okno zavrieť, ak chcete znovu zatlačiť tlačidlo, vždy treba najprv čakať (niekedy až 5 sekúnd), kým sa nedokreslí celá séria štvorcov.

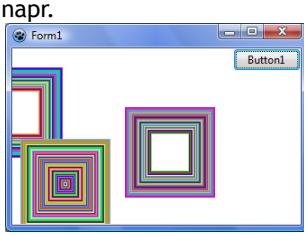
Teraz sa pozrime, ako to bude vyzerať pomocou časovača. Časovaču nastavíme **Interval = 100**, aby sa procedúra spúšťala 10 krát za sekundu a tiež **Enabled = False**, aby časovač pri štarte programu ešte čakal na stlačenie tlačidla. Premenné **X**, **Y** a **A** musia byť globálne a počiatočné hodnoty sa im nastavia pri zatlačení tlačidla:

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

var
  X, Y, A: Integer;

procedure TForm1.Button1Click(Sender: TObject);
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  A := 50;
  Timer1.Enabled := True;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if A > 0 then
  begin
    Image1.Canvas.Pen.Color := Random(256*256*256);
    Image1.Canvas.Rectangle(X-A, Y-A, X+A, Y+A);
    A := A - 1;
  end
  else
    Timer1.Enabled := False;
end;
```

Všimnite si zápis **Timer1.Enabled := False;**, ktorým zastavíme časovač. Tu sme to použili v situácii, keď je už **A = 0**, t.j. keď už sú všetky štvorce nakreslené.

Zdá sa, že po zatlačení tlačidla, sa program správa úplne rovnako: začne postupne kresliť 50 rôznofarebných stále sa zmenšujúcich sústredných štvorcov. Lenže toto riešenie je teraz už korektné: aj počas spustenia kreslenia môžeme celé okno posúvať, aplikáciu môžeme hocikedy ukončiť.

Dokonca funguje znovu zatlačenie tlačidla aj počas kreslenia. Vtedy sa preruší momentálne kreslená séria a začne sa kresliť nová, napr.



kde môžeme vidieť dve nedokončené série štvorcov.

Ak by sme vyžadovali, že nová séria sa môže začať kresliť až po dokončení rozbehnutej, resp. nedá sa naštartovať nová, kým nedobehla stará, môžeme to zabezpečiť takto jednoducho:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if not Timer1.Enabled then
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    A := 50;
    Timer1.Enabled := True;
  end;
end;
```

V takomto riešení sa naštartuje nová séria štvorcov (vygeneruje sa nové X, Y, A), len ak momentálne nebeží časovač.

Keď teraz porovnáme obe riešenia tejto úlohy: bez časovača pomocou **Sleep** a **Repaint** a s časovačom, vidíme, že s časovačom sa musí programovať trochu inak (možno zložitejšie) ako obyčajný spôsob spomaľovania pomocou **Sleep**. Ale tieto programy sa už správajú korektne a program zbytočne v niektorých situáciách nezamŕza. Zapamätajte si, že použitie **Sleep** nie je pri programovaní najvhodnejšie a vždy musíme zvážiť, či je jeho použitie na tomto naozaj nevyhnutné.

Túto úlohu môžeme riešiť aj trochu inak: časovač nebudeme nikdy zastavovať - bude bežať stále. Len niekedy bude kresliť štvorce a niekedy nie. Môžeme na to využiť premennú **A**: ak je v časovači hodnota premennej **A = 0**, štvorec sa kresliť nemusí, ak je **A > 0**, nakreslí sa štvorec a **A** sa zmenší o 1.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Timer1.Enabled := True;
end;

var
  X, Y, A, Pocitadlo: Integer;

procedure TForm1.Button1Click(Sender: TObject);
begin
  if A = 0 then
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    A := 50;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Pocitadlo := Pocitadlo + 1;
  Image1.Canvas.TextOut(0, 0, IntToStr(Pocitadlo));
  if A > 0 then
  begin
    Image1.Canvas.Pen.Color := Random(256*256*256);
    Image1.Canvas.Rectangle(X-A, Y-A, X+A, Y+A);
    A := A - 1;
  end
end;
```

Toto riešenie má ešte jednu výhodu oproti predchádzajúcemu: časovač okrem kreslenia štvorcov môže robiť aj iné veci, napr. zobrazuje bežiacu časomieru.

Všimnite si, že časovač by mal byť naštartovaný už pri štarte programu. Môžeme to

buď nastaviť v inšpektore objektov (riadku **Enabled** nastavíme **True**), alebo toto priradenie urobíme v štartovacej procedúre **FormCreate**.

Úlohy na precvičenie

Zadanie 1	Program bude nejakých v časových intervaloch, napr. 100 milisekúnd, prekresľovať nejaký text dvomi rôznymi farbami, napr. červenou a modrou. Mal by vzniknúť efekt blikania.
Zadanie 2	Program postupne kreslí zväčšujúcu sa štvorcovú špirálu: pri každom tiknutí časovača nakreslí jednu úsečku špirály.
Zadanie 3	Program každých 100 milisekúnd na náhodné pozície nakreslí malý farebný krúžok. Farba krúžkov sa zmení na náhodnú vždy po nakreslení 30 krúžkov.
Zadanie 4	Program bude simulovať fungovanie semaforu: najprv sa nakreslí červený kruh, ten sa napr. o 5 sekúnd prefarbí na žlto a o ďalšie 2 sekundy na zeleno, zelená farba vydrží napr. 8 sekúnd, opäť sa na 1 sekundu objaví žltá a celé sa to opakuje.
Zadanie 5	Program ako šetrič obrazovky: každých päť sekúnd zmaže obrazovku na niektorú tmavú farbu a veľkými svetlými písmenami napíše jedno zo známych slovenských porekadiel. Porekadlo vyberie náhodne z niekoľkých predpripravených (aspoň troch).
Zadanie 6	Program, v ktorom sa bude pohybovať po ploche pohybovať biliardová guľa: okrem polohy gule (**X**, **Y**) si budeme v globálnych premenných **DX** a **DY** pamätať momentálny smer pohybu. Pri každom tiknutí časovača, okrem vykreslenia gule, jej zmeníme polohu tak, že **DX** pripočítame k **X** a **DY** pripočítame k **Y**. Ak je x-ová súradnica mimo plochu, tak otočíme **DX** (**DX := -DX;**). Podobne otestujeme, či y-ová súradnica nie je mimo plochu, ak áno, otočíme **DY**. Premenné **DX** a **DY** môžu mať na začiatku hodnoty napr. 5 a 4.

2. tematická jednotka - Procedúry s parametrami

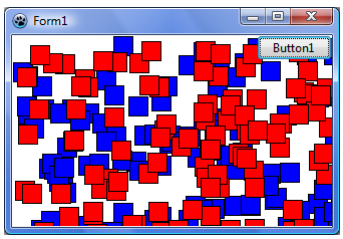
1. Parametre prenášané hodnotou

V predchádzajúcich častiach sme definovali rôzne typy procedúr, pričom často sme im museli niektoré nastavenia odovzdávať v lokálnych premenných nadradenej procedúry. Nasledujúci program nakreslí na náhodnú pozíciu štvorček veľkosti 20x20. Keďže chceme takéto štvorčeky kresliť rôznymi zadanými farbami, procedúra **Stvorcek** bude používať premennú **Farba**, ktorú zadefinujeme tak, aby ju **Stvorcek** videl:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Farba: TColor;

  procedure Stvorcek;
  var
    X, Y: Integer;
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Brush.Color := Farba;
    Image1.Canvas.Rectangle(X, Y, X+20, Y+20);
  end;

var
  I: Integer;
begin
  Farba := clBlue;
  for I := 1 to 100 do
    Stvorcek;
  Farba := clRed;
  for I := 1 to 100 do
    Stvorcek;
end;
```

TColor je typ, ktorý na rozdiel od **Integer**, neslúži na uloženie čísla, ale na uloženie nejakej farby. Typ **TColor** je kompatibilný s celočíselným typom **Integer**.

Program najprv nakreslí 100 modrých a potom 100 červených štvorcov:



Procedúry poskytujú elegantnejší spôsob tzv. posielanie parametrov. Procedúru môžeme zadefinovať tak, že pri jej volaní zadáme parametre, ktoré zapíšeme do okrúhlych zátvoriek. Aj všetky grafické príkazy sú definované s parametrami, napr. **Rectangle** má štyri celočíselné parametre. Keby sme túto procedúru chceli zavolať s iným počtom parametrov ako 4, alebo jej namiesto celého čísla poslali text, Pascal by nás upozornil na chybu v programe.

Keď budeme definovať nejakú svoju procedúru, počet parametrov, ich mená a aj typy závisia len od nás - programátorov. Začneme procedúrou **Stvorcek**, ktorá bude mať 1 parameter **Farba**.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Farba: TColor;

  procedure Stvorcek(Farba: TColor);
  var
    X, Y: Integer;
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Brush.Color := Farba;
    Image1.Canvas.Rectangle(X, Y, X+20, Y+20);
  end;

var
  I: Integer;
begin
  Farba := clBlue;
  for I := 1 to 100 do
    Stvorcek(clBlue);
  Farba := clRed;
  for I := 1 to 100 do
    Stvorcek(clRed);
end;
```

Zadefinovali sme procedúru **Stvorcek** ale tak, že ju môžeme používať iba, ak jej pošleme 1 parameter s farbou. Napr. volanie **Stvorcek(clBlue);** označuje, že zavoláme procedúru **Stvorcek** tak, že parameter (premenná **Farba**) bude mať hodnotu **clBlue**. Preto sa na náhodných súradniciach nakreslí modrý štvorček so stranou 20. Hodnotám parametrov ktoré zadávame pri volaní procedúry hovoríme **hodnoty vstupných parametrov**. Tieto môžu byť zadané konštantou (napr. clBlue), hodnotou premennej alebo ako výsledok aritmetického výrazu.

Pozorne pozrime, ako sme z nejakej premennej vytvorili parameter: Premenná **Farba** bola definovaná pred samotnou procedúrou **Stvorcek**. Túto deklaráciu prenesieme do hlavičky procedúry tak, že ju uzavrieme do zátvoriek. Všimnite si, ako sa táto deklarácia parametra líši od lokálnych premenných procedúry. Pri volaní tejto procedúry musíme oznámiť, aká bude hodnota tohto parametra. Opäť sa to robí tak, že sa táto hodnota (napr. **clBlue**) zapíše do okrúhlych zátvoriek.

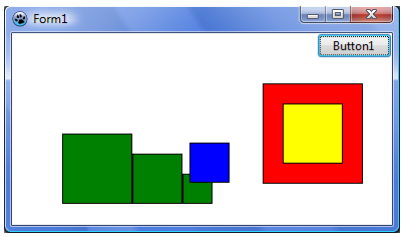
Teraz sa pozrime na ďalšiu procedúru **Stvorec**, ktorá bude mať 3 parametre: súradnice **X** a **Y** a parameter **Strana** (veľkosť strany štvorca):

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

  procedure Stvorec(X, Y, Strana: Integer);
  begin
    Image1.Canvas.Rectangle(X, Y, X+Strana, Y+Strana);
  end;

begin
  Image1.Canvas.Brush.Color := clGreen;
  Stvorec(50, 100, 70);
  Stvorec(120, 120, 50);
  Stvorec(170, 140, 30);

  Image1.Canvas.Brush.Color := clRed;
  Stvorec(250, 50, 100);
  Image1.Canvas.Brush.Color := clYellow;
  Stvorec(270, 70, 60);

  Image1.Canvas.Brush.Color := clBlue;
  Stvorec(Random(300), Random(200), 40);
end;;
```

Zadefinovanú procedúru **Stvorec** musíme preto volať s tromi celočíselnými parametrami. Napr. volanie **Stvorec(50, 100, 70);** označuje, že zavoláme procedúru

Stvorec tak, že prvý parameter (premenná **X**) bude mať hodnotu 50, druhý parameter (premenná **Y**) bude mať hodnotu 100 a tretí parameter (premenná **Strana**) bude mať hodnotu 70. Takže sa na súradniciach (50, 100) nakreslí štvorec so stranou 70. Vstupnými hodnotami parametrov hovoríme skutočné parametre, teda v našom prípade skutočnými parametrami sú hodnoty 50, 100, 70.

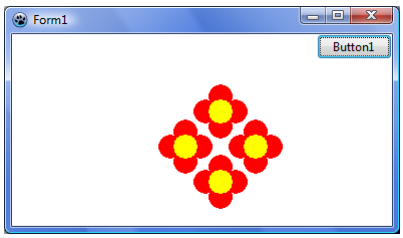
Program po spustení najprv nakreslí 3 zelené štvorce, potom jeden červený, vo vnútri ktorého je žltý a na záver na náhodnú pozíciu nakreslí modrý štvorec. Môže to vyzerať takto:



Pozrime si nasledujúci príklad. Je v ňom zadefinovaná procedúra **Kvet** s dvoma parametrami – súradnice, kde sa nakreslí kvietok so štyrmi červenými lupienkami a žltým stredom. Samotná procedúra **Button1Click** nakreslí 4 kvietky takto:

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

  procedure Kvet(X, Y: Integer);
  begin
    Image1.Canvas.Brush.Color := clRed;
    Image1.Canvas.Pen.Color := clRed;
    Image1.Canvas.Ellipse(X-12, Y-27, X+12, Y-3);
    Image1.Canvas.Ellipse(X-12, Y+3, X+12, Y+27);
    Image1.Canvas.Ellipse(X-27, Y-12, X-3, Y+12);
    Image1.Canvas.Ellipse(X+3, Y-12, X+27, Y+12);
    Image1.Canvas.Brush.Color := clYellow;
    Image1.Canvas.Ellipse(X-13, Y-13, X+13, Y+13);
  end;

var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  Kvet(X-35, Y);
  Kvet(X, Y-35);
  Kvet(X+35, Y);
  Kvet(X, Y+35);
end;
```

Po spustení sa programu sa na náhodnej pozícii vypíše štvorica kvetov:



V tomto príklade si všimnite, aké vstupné hodnoty sme poslali pre parametre **X** a **Y**. V predchádzajúcich príkladoch to boli konštanty (buď farby **clRed**, alebo celé čísla), tu sú to premenné alebo aritmetické výrazy: X, Y, X+35, Y-35, ... Zapamätajte si, že vstupnými hodnotami pre parametre procedúr môžu byť nielen konštanty, ale aj premenné a aritmetické výrazy.

Úlohy na precvičenie

Zadanie 1	Zadefinujte procedúru **Usecka(X1, Y1, X2, Y2)** a pomocou nej vyšrafujte štvorec so stranou 100 a s ľavým horným vrcholom X, Y.
Zadanie 2	Zadefinujte procedúru **Kruh** s takýmito parametrami: **X**, **Y** súradnice stredu, **R** polomer, **Vypln** je logická hodnota, ktorá keď je **True**, tak sa kružnica vyfarbí náhodnou farbou, inak bude kruh nevyplnený, teda bude to kružnica. Pomocou tejto procedúry nakreslite náhrdelník (rad kruhov), ktorých polomer bude náhodné číslo od 10 do 30 a každý druhý kruh nebude vyplnený farbou.
Zadanie 3	Program nakreslí N poschodovú pyramídu, pričom využije pomocnú procedúru kváder, ktorého prvé dva parametre sú **X**, **Y** stredu dolnej hrany kvádra a ďalšie dva parametre sú šírka a výška kvádra. N je konštanta programu, napr. 10. Každé vyššie poschodie je o 20 kratšie ako od neho nižšie poschodie. Výška kvádrov je 20. Kvádre môžu byť vyplnené náhodnou farbou.
Zadanie 4	Procedúra **Srafuj** vyšrafuje obdĺžnik so súradnicami X1, Y1, X2, Y2, piatym parametrom je šírka vodorovných čiar (hrúbka čiar). Šrafuje na striedačku dvomi náhodne zvolenými farbami. Pomocou tejto procedúry vyšrafujte celú grafickú plochu.
Zadanie 5	Zadefinujte pomocné procedúry pre kreslenie modrého štvorca, červeného trojuholníka, domčeka (nakresleného pomocou procedúr pre štvorec a trojuholník). Procedúry navrhnite tak, aby sa dalo určiť, kde sa domčeky nakreslia a ako budú veľké (veľkosti strán štvorcov a trojuholníkov).
Zadanie 6	Zadefinujte procedúru **Hviezdicka**, ktorá nakreslí hviezdu so stredom **X**, **Y**, počtom, dĺžkou a hrúbkou ramien **Pocet**, **Dlzka**, **Hrubka**. Ramená hviezdy sú úsečky, ktorých jeden koniec je v strede a druhý je na obvode mysleného kružnice s polomerom **Dlzka**. Ramená sú na kružnici rozložené rovnomerne. Nakreslite nočnú oblohu: na tmavomodrej ploche malé žlté hviezdičky s náhodným počtom ramien a dĺžok.

2. Mechanizmus volania procedúry s parametrami

V predchádzajúcich častiach sme popísali mechanizmus volania procedúry bez parametrov. Teraz, keď procedúra môže obsahovať aj parametre, tento mechanizmus treba doplniť. Teda pri **volaní procedúry** sa postupne:

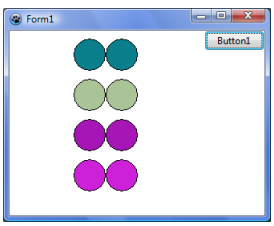
1. **zapamätá sa návratové miesto**, t.j. riadok, kam sa bude treba vrátiť po skončení procedúry;
2. **vytvoria sa všetky zadeklarované lokálne premenné** procedúry – tieto majú nedefinovanou hodnotou;
    - aj parametre sú lokálne premenné, preto sa na tomto mieste **vytvoria aj všetky parametre** (premenné) a do nich sa priradia príslušné vstupné hodnoty
3. prenesie sa **riadenie** programu **do tela podprogramu** (za príslušný **begin**);
4. vykonajú sa všetky príkazy podprogramu (až po koncový **end**);
5. **zrušia sa lokálne premenné** – a teda sa zrušia aj parametre;
6. **riadenie sa vráti** za miesto v programe, odkiaľ bol podprogram volaný – t.j. **na návratové miesto**.

Uvedomte si, že procedúra nemôže mať lokálnu premennú s rovnakým názvom ako má niektorý z parametrov.

Na niekoľkých programoch ukážeme tieto nové časti mechanizmu volania. Pozrime tento program:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;

  procedure Kruhy(X, Y: Integer);
  begin
    Image1.Canvas.Brush.Color := Random(256*256*256);
    Image1.Canvas.Ellipse(X-20, Y-20, X+20, Y+20);
    X := X + 40;
    Image1.Canvas.Ellipse(X-20, Y-20, X+20, Y+20);
  end;

begin
  X := 100;
  Y := 30;
  Kruhy(X, Y);
  Kruhy(X, Y+50);
  Kruhy(X, Y+100);
  Kruhy(X, Y+150);
end;
```

Všimnime si dve rôzne premenné **X**. Prvá z nich je lokálnou premennou v **Button1Click** a priradili sme do nej hodnotu 100. Druhá premenná **X** zatiaľ ešte neexistuje a vytvorí sa až po prvom zavolaní procedúry **Kruhy**. Vtedy sa do nej priradí ako vstupná hodnota obsah prvej premennej **X**, t.j. číslo 100. Procedúra **Kruhy** nakreslí najprv prvý kruh na súradnice (100, 30), zväčší svoju lokálnu premennú **X** o 40, aby nakreslila druhý kruh na súradnice (140, 30). Po nakreslení tejto druhej kružnice sa automaticky zrušia všetky lokálne premenné, t.j. parametre **X** a **Y** a riadenie sa vráti na ďalší riadok **Button1Click**, teda na riadok **Kruhy(X, Y+50);** Takto to postupne prejde všetky volania procedúry **Kruhy**. V grafickej ploche sa nakreslí:



Z tohto príkladu by sme si mali zapamätať, že parametre procedúr sú rovnaké ako obyčajné lokálne premenné, len už majú priradenú počiatočnú hodnotu, ktorá je

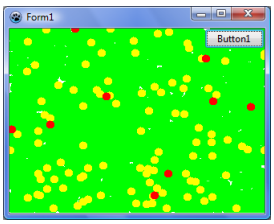
rovná vstupnej hodnote parametra. Okrem toho zmena hodnoty premennej parametra vo vnútri procedúry nemá žiaden vplyv na iné premenné (tak ako sme menili parameter **X** a tým sa nezmení obsah lokálnej premennej X).

V ďalšom príklade procedúra **Bodky** nakreslí **N** farebných bodiek.

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

  procedure Bodky(N: Integer; Farba: TColor);
  var
    X, Y: Integer;
  begin
    Image1.Canvas.Pen.Color := Farba;
    Image1.Canvas.Pen.Width := 10;
    while N > 0 do
    begin
      X := Random(Image1.Width);
      Y := Random(Image1.Height);
      Image1.Canvas.MoveTo(X, Y);
      Image1.Canvas.LineTo(X+1, Y);
      N := N - 1;
    end;
  end;

begin
  Bodky(5000, clLime);
  Bodky(100, clYellow);
  Bodky(10, clRed);
end;
```

Táto procedúra **Bodky** má dva parametre rôznych typov. Samozrejme, že pri volaní procedúry musíme dodržať poradie vstupných hodnôt pre parametre: prvý vstupnú parameter označuje počet bodiek a druhý ich farbu.

Všimnite si použitie parametra **N**. Keďže **N** je lokálna premenná, môže sa použiť aj ako počítadlo vo while-cykle. Program najprv nakreslí 5000 náhodných bledozelených bodiek, potom 100 žltých a na koniec 10 červených:



Úlohy na precvičenie

Zadanie 1	Procedúra **Panel** nakreslí šedý štvorec 40x40, v ktorom je bledomodrý menší štvorec (okno). Procedúra **Panelak** má dva parametre: počet panelov na jednom poschodí (šírka) a počet poschodí (výška). Nakreslite vedľa seba tri rôzne vysoké paneláky.
Zadanie 2	Zadefinujte procedúru **RadStvorcov**(**X**, **Y**, **N**), ktorá nakreslí vedľa seba N štvorcov veľkosti 20, pričom prvý z nich začína na súradnici (**X**, **Y**). Pomocou tejto procedúry zadefinujte procedúru **Pyramida**(N), ktorá nakreslí pyramídu výšky N poschodí.
Zadanie 3	Procedúra **Stvorec**(**X**, **Y**, **Farba**) nakreslí farebný štvorec veľkosti 30. Napíšte procedúru **Sachovnica**(N), ktorá nakreslí šachovnicu veľkosti NxN štvorcov (pomocou procedúry **Stvorec**), v ktorej sa striedajú dve farby štvorcov.
Zadanie 4	Naprogramujte preteky dvoch vozíkov: pomocná procedúra **Vozik**(X, Y, Farba), ktorú budeme volať dvakrát – pre každý vozík raz. V Časovači sa posunie o náhodnú hodnotu X-ová súradnica oboch vozíkov, plocha sa zmaže a nakreslia sa oba vozíky na nových súradniciach.
Zadanie 5	Program vypíše do grafickej plochy rozvrh hodín školských predmetov. Využije pomocné procedúry: **Tabulka** vykreslí prázdnu tabuľku pre 5 dní a **N** stĺpcov; **Predmet** vloží do políčka **Riadok**, **Stlpec** náhodne jeden z predmetov (text napr. 'Mat', 'Slov. jaz.', 'Agl. jaz.', 'Inf', ...). Náhodné generovanie predmetov môžete zapísať napr. konštrukciou: **case Random(8) of 0: ...** vypíš **'Mat'; 1: ...** vypíš **'Slov. jaz.'; ...**

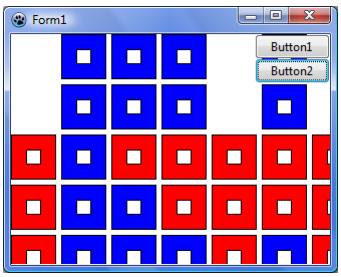
3. Lokálne (vnorené) a globálne procedúry

Všetky procedúry, ktoré sme doteraz napísali, sme mohli použiť len na tom mieste, kde sú vnorené. Niekedy sa môže pritrafiť situácia, keď by sme chceli tú istú procedúru využívať vo viacerých procedúrach pre stláčanie tlačidiel.

V nasledujúcom príklade tlačidlo **Button1** nakreslí rad červených štvorcov a **Button2** stĺpec modrých štvorcov. Obe tlačidlá definujú svoju pomocnú procedúru **Stvorec**, ktorá je v oboch prípadoch úplne rovnaká:

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

  procedure Stvorec(X, Y: Integer; Farba: TColor);
  begin
    Image1.Canvas.Brush.Color := Farba;
    Image1.Canvas.Rectangle(X, Y, X+45, Y+45);
    Image1.Canvas.Brush.Color := clWhite;
    Image1.Canvas.Rectangle(X+15, Y+15, X+30, Y+30);
  end;

var
  I, J: Integer;
begin
  I := 0;
  J := 50 * Random(6);
  while I < Image1.Width do
  begin
    Stvorec(I, J, clRed);
    I := I + 50;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);

  procedure Stvorec(X, Y: Integer; Farba: TColor);
  begin
    Image1.Canvas.Brush.Color := Farba;
    Image1.Canvas.Rectangle(X, Y, X+45, Y+45);
    Image1.Canvas.Brush.Color := clWhite;
    Image1.Canvas.Rectangle(X+15, Y+15, X+30, Y+30);
  end;

var
  I, J: Integer;
begin
  I := 0;
  J := 50 * Random(6);
  while I < Image1.Height do
  begin
    Stvorec(J, I, clBlue);
    I := I + 50;
  end;
end;
```

Po niekoľkých zatlačeniach tlačidiel **Button1** a **Button2** môžeme dostať:



Jedným z dôvodov, prečo sme začali používať procedúry, bolo to, že sme opakujúci

sa kód napísali raz a používali mnohokrát. Zatiaľ sme ale všetky procedúry definovali vo vnútri iných procedúr na spracovávanie udalostí od tlačidiel (**Button1Click**, **Button2Click**, ...). O premenných, ktoré sú definované vo vnútri procedúr, hovoríme, že sú **lokálne** a tie, čo sú mimo nich, sú **globálne**. Podobne môžeme hovoriť aj o procedúrach: všetky nami definované boli doteraz **lokálne** a ak by sme ich zadefinovali mimo procedúr, boli by **globálne**. Vyskúšajme to urobiť aj v našom príklade s procedúrou **Stvorec**. Procedúru vytiahneme pred **Button1Click** aj **Button2Click**:

```
procedure Stvorec(X, Y: Integer; Farba: TColor);
begin
  Image1.Canvas.Brush.Color := Farba;
  Image1.Canvas.Rectangle(X, Y, X+45, Y+45);
  Image1.Canvas.Brush.Color := clWhite;
  Image1.Canvas.Rectangle(X+15, Y+15, X+30, Y+30);
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  I, J: Integer;
begin
  I := 0;
  J := 50 * Random(6);
  while I < Image1.Width do
  begin
    Stvorec(I, J, clRed);
    I := I + 50;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I, J: Integer;
begin
  I := 0;
  J := 50 * Random(6);
  while I < Image1.Height do
  begin
    Stvorec(J, I, clBlue);
    I := I + 50;
  end;
end;
```

Ak sa pokúsime skompilovať a spustiť takto pozmenený program, Pascal už na prvom riadku tela procedúry **Stvorec** vyhlási chybu typu "neznámy identifikátor **Image1**". Prečo to predtým fungovalo a teraz to kompilátoru vadí? Odpoveď je skrytá v tom, že obe procedúry **Button1Click** aj **Button2Click** majú pred svojim menom **TForm1.** Tento zápis pre Pascal označuje, že vo vnútri týchto procedúr pracujeme s prvkami formuláru (napr. **Image1**, **Button1**). Naša nová globálna procedúra **Stvorec** ale nemá označenie, že patrí formuláru a preto Pascal nevie rozpoznať prvky, ktoré sú vo formulári.

V skutočnosti to znamená, že **Button1Click** je metódou triedy **TForm1** a táto má definované svoje atribúty **Image1**, **Button1**, ... Keď definujeme globálnu procedúru **Stvorec**, každý identifikátor (premennej alebo procedúry) musí byť presne špecifikovaný. Keďže **Image1** patrí triede **TForm1** a konkrétne inštancii **Form1**, musíme toto **Form1** uvádzať aj pred každým atribútom, na ktorý sa chceme z tejto triedy odvolávať. Mohli by sme procedúru **Stvorec** definovať ako metódu triedy **TForm1** a potom by sme samozrejme **Form1** pred každým atribútom nepísali. Toto je ale nad rámec tohto kurzu programovania.

Zapamätajme si, ak potrebujeme vytvoriť globálnu procedúru, ktorá pracuje s prvkami (komponentmi) formuláru, tak pred tieto prvky musíme písať slovo **Form1.** Teda, správne sme mali zapísať:

```
procedure Stvorec(X, Y: Integer; Farba: TColor);
begin
  Form1.Image1.Canvas.Brush.Color := Farba;
  Form1.Image1.Canvas.Rectangle(X, Y, X+45, Y+45);
  Form1.Image1.Canvas.Brush.Color := clWhite;
  Form1.Image1.Canvas.Rectangle(X+15, Y+15, X+30, Y+30);
end;
```

Teraz to už všetko funguje v poriadku. Je to istá komplikácia, ktorú musíme zvážiť, keď sa budeme rozhodovať o vytváraní globálnych procedúr. Globálne procedúry ale majú naproti tomu tie výhody, že ich môžeme využívať na viacerých miestach programu (nielen v udalostiach od kliknutia ako **Button1Click**), ale napríklad aj pri inicializácii programu (**FormCreate**), v časovači (**Timer1Timer**), udalosti od posúvača (**ScrollBar1Change**).

Už máme skúsenosti s viditeľnosťou identifikátorov v pascalovských programoch. Vieme, že ich treba definovať ešte pred ich prvým použití a preto zvyknú byť sústredené niekde na začiatku programu hneď za slovom **implementation**.

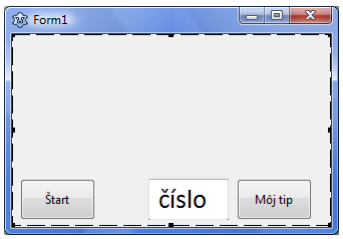
Úlohy na precvičenie

Zadanie 1	Zadefinujte globálne procedúry na kreslenie bielych kruhov so zadaným stredom a polomerom. Ďalej procedúru snehuliak, ktorá z troch rôzne veľkých kruhov postaví snehuliaka. Táto dostáva ako parameter pozíciu a veľkosť najväčšieho kruhu – z neho si vypočíta aj veľkosti menších kruhov a najmenšiemu z nich dokreslí oči (dva malé čierne kruhy). Program na šedé pozadie náhodne kladie rôznymi tlačidlami rôzne veľké snehuliaky.
Zadanie 2	Zadefinujte globálne procedúry **Stvorec**, **Obdlznik**, **Kruh** a **Elipsa** s parametrami o ich umiestnení, veľkosti. Vo formulári zadefinujte tlačidlá, ktoré budú kresliť časti tváre (hlava, oči, nos, ústa, ...) a tiež tlačidlá, ktoré budú nastavovať niektoré farby štetca (napr. červená, modrá, žltá, ...). Na kreslenie útvarov využite zadefinované globálne procedúry.
Zadanie 3	Napíšte globálne procedúry na kreslenie domu a potom aj ulice ako radu domov. Vo formulári bude viac tlačidiel: každé nakreslí tento rad domov s nejakými inými nastaveniami, napr. zmena farieb, zmena hrúbky pera, zmena počtu domov, zmena medzier medzi domami a pod. Na zmenu počtu, veľkosti a vzdialenosti môžete použiť posúvač.
Zadanie 4	Zadefinujte globálne procedúry, ktoré kreslia rôzne typy áut (napr. osobné, nákladné, traktor, ...)a majú parametre o pozícii a veľkosti. Rôzne tlačidlá v ploche využívajú tieto procedúry a kreslia rôzne rady za sebou stojacich áut: napr. zväčšujúca sa postupnosť, postupnosť zmenšujúcich sa aut, rad rovnako veľkých aut a pod.

4. Hádanie čísla, ktoré si myslí počítač

V tejto časti ukážeme takúto aplikáciu: Je to známa hra pre dvoch hráčov, v ktorej si jeden myslí nejaké číslo a druhý sa ho snaží uhádnuť.

Položme do formulára grafickú plochu, dve tlačidlá s textami "Štart" a "Môj tip" a jeden vstupný riadok (**TEdit**) s textom "číslo":



Napíšeme procedúry FormCreate, Button1Click a Button2Click:

```
var
  Cislo: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 36;
  Button2.Enabled := False;
end;

procedure TForm1.Button1Click(Sender: TObject);      // Štart
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.TextOut(20, 10, 'Myslím si číslo do 100.');
  Image1.Canvas.TextOut(20, 50, 'Skús ho uhádnuť.');
  Edit1.Text := 'číslo';
  Button2.Enabled := True;
  Cislo := Random(100)+1;
end;

procedure TForm1.Button2Click(Sender: TObject);     // Môj tip
var
  Hadas: Integer;
begin
  Hadas := StrToInt(Edit1.Text);
  if Hadas = Cislo then
  begin
    Image1.Canvas.FillRect(Image1.ClientRect);
    Image1.Canvas.TextOut(20, 50, 'HURÁ - uhádol si              ');
    Button2.Enabled := False;
  end
  else if Hadas < Cislo then
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je málo    ')
  else
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je veľa    ');
end;
```

Program je pravdepodobne jednoduchý a nemalo by vám robiť problém ho prečítať a pochopiť. Všimnite si, že na niekoľkých miestach sme použili zápis buď

Button2.Enabled := False;

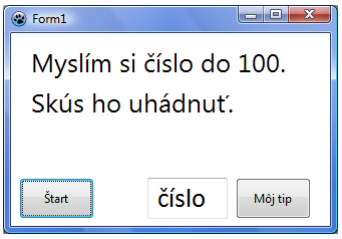
alebo

Button2.Enabled := True;

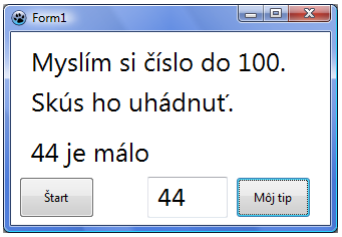
Podobá sa to na zastavovanie, resp. púšťanie časovača (Timer1.Enabled := False;) V tomto prípade takto môžeme zablokovať niektoré tlačidlo a preto sa nebude dať na

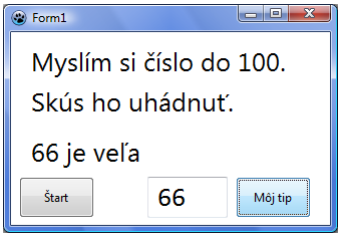
neho klikať. V našom programe v niektorých situáciách blokujeme tlačidlo "Môj tip": kým si počítač ešte nevymyslel číslo na uhádnutie, resp. keď sa nám ho už podarí uhádnuť. Tlačidlo odblokujeme (**Button2.Enabled := True;**), keď štartujeme hru a počítač si práve vymyslel nejaké číslo na hádanie.

Po spustení programu a naštartovaní hry program vyzerá takto:

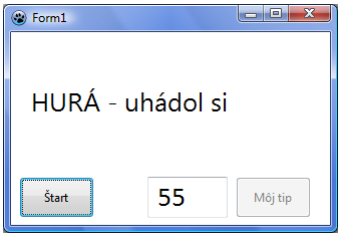


Teraz do vstupného riadku môžeme zadávať čísla a zisťovať, či sme sa trafili, napr.:





Až kým neuhádneme, napr.:



Program je funkčný, ale žiada si nejaké vylepšenia:

- keď namiesto čísla, zadáme zlý reťazec, program veľmi nepeknou hláškou oznámi, že nastala chyba pri zadávaní celého čísla - vylepšíme to našom spracovaním chyby;
- program by mohol počítať počet pokusov a napr. pri 10. pokuse by mohol oznámiť neúspech v hádaní;
- môžeme zabudovať stopky, ktoré napr. po minúte neúspešného hádania, zastaví hru aj s nejakou správou.

Najprv spracujeme zle zadané vstupné číslo. Použijeme iný variant štandardnej konverznej funkcie **StrToInt** -funkcia **StrToIntDef** funguje skoro rovnako, len v situácii, keď vstupný text neobsahuje "slušné" číslo, vráti hodnotu, ktorú zadáme ako druhý parameter funkcie. Napr. **StrToIntDef('123', 0)** vráti číslo 123, ale **StrToIntDef('x123', 0)** vráti hodnotu 0, lebo vstupná hodnota prvého parametra nie je dobre zadané číslo. Opravíme procedúru **Button2Click**:

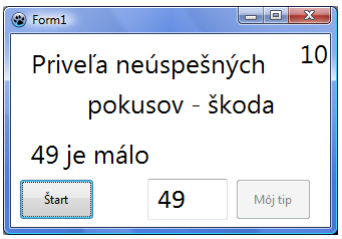
```
procedure TForm1.Button2Click(Sender: TObject);
var
  Hadas: Integer;
begin
  Image1.Canvas.FillRect(Rect(20, 100, 400, 150));
  Hadas := StrToIntDef(Edit1.Text, 0);
  if (Hadas < 1) or (Hadas > 100) then
  begin
    Image1.Canvas.TextOut(20, 100, 'Zadaj číslo od 1 do 100');
    Edit1.Text := '';
  end
  else if Hadas = Cislo then
  begin
    Image1.Canvas.FillRect(Image1.ClientRect);
    Image1.Canvas.TextOut(20, 50, 'HURÁ - uhádol si');
    Button2.Enabled := False;
  end
  else if Hadas < Cislo then
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je málo')
  else
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je veľa');
end;
```

Okrem použitia **StrToIntDef** si pozrite aj trochu iné volanie **Image1.Canvas.FillRect**. Ako parameter sme tam neposielali **Image1.ClientRect**, ktoré označuje, že chceme zmazať celú grafickú plochu, ale **Rect(20, 100, 400, 150)**, čo označuje, že chceme mazať len časť plochy a to presne tú, kam sa vypisujú správy, napr. správa **Zadaj číslo od 1 do 100**.

Teraz pridáme počítadlo pokusov. Potrebujeme ďalšiu globálnu premennú **Pocet**. pripíšeme ju k už predtým deklarovanej premennej **Cislo** a pri štarte hry (**Button1Click**) túto hodnotu vynulujeme. Pozrite, čo sme pridali do procedúry **Button2Click**:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Hadas: Integer;
begin
  Image1.Canvas.FillRect(Rect(20, 100, 400, 150));
  Hadas := StrToIntDef(Edit1.Text, 0);
  if (Hadas < 1) or (Hadas > 100) then
  begin
    Image1.Canvas.TextOut(20, 100, 'Zadaj číslo od 1 do 100');
    Edit1.Text := '';
  end
  else if Hadas = Cislo then
  begin
    Image1.Canvas.FillRect(Image1.ClientRect);
    Image1.Canvas.TextOut(20, 50, 'HURÁ - uhádol si');
    Button2.Enabled := False;
  end
  else if Hadas < Cislo then
  begin
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je málo');
    Pocet := Pocet + 1;
  end
  else
  begin
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je veľa');
    Pocet := Pocet + 1;
  end;

  Image1.Canvas.TextOut(Image1.Width-30, 0, IntToStr(Pocet));
  if Pocet = 10 then
  begin
    Image1.Canvas.TextOut(20, 10, 'Priveľa neúspešných    ');
    Image1.Canvas.TextOut(20, 50, '        pokusov - škoda');
    Button2.Enabled := False;
  end;
end;
```

V prvom rade, je to zvyšovanie premennej **Pocet** pri vypisovaní správ o neúspešných pokusoch. Potom sa ešte do pravého horného rohu plochy (Image1.Width-30, 0) vypíše počet pokusov a na záver sa spracuje situácia, keď nastalo 10 neúspešných pokusov. Aj v tomto prípade sa zablokuje druhé tlačidlo, aby bolo jasné, že hra skončila a môže sa to naštartovať znovu od začiatku:



Na záver pridáme časomieru, ktorá zastaví hru, ak sa nestihne uhádnuť do určitého času. Vložíme komponent časovač (**TTimer**) a nastavíme mu **Interval = 100** a **Enabled = False**, t.j. tikanie po desatinách sekundy a pri štarte ešte nebeží. Budeme potrebovať ešte jednu globálnu celočíselnú premennú, napr. **Cas**, v ktorej budeme počítať zostávajúci čas pre bežiacu hru. Časovač naštartujeme (**Timer1.Enabled := True**) a časomieru v premennej **Cas** nastavíme napr. 300 (čo je 30 sekúnd) v procedúre **Button1Click**.

Zároveň pripravíme spracovanie časovača v **Timer1Timer**: najprv skontrolujeme, či je druhé tlačidlo aktívne, ak nie je (**Button2.Enabled** je **False**) znamená to, že hra už nebeží, napr. sme uhádli, alebo sme si už minuli 10 pokusov, a preto zastavíme časovač (**Timer1.Enabled := False**). Inak znamená, že hra ešte beží. Znížime časomieru a vypíšeme ju zmenšeným písmom do ľavého horného rohu. Ak je jej hodnota už 0, hra končí, lebo sme si vyčerpali všetok čas. Uvádzame kompletný kód hry:

```
var
  Cislo, Pocet, Cas: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 36;
  Button2.Enabled := False;
end;
```

Časovač **Timer1Timer** znižuje zostávajúci čas (premennú **Cas**), vypisuje ho a kontroluje vypršaný čas:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if not Button2.Enabled then
    Timer1.Enabled := False
  else
  begin
    Cas := Cas - 1;
    Image1.Canvas.Font.Height := 18;
    Image1.Canvas.TextOut(0, 0, IntToStr(Cas) + '  ');
    Image1.Canvas.Font.Height := 36;
    if Cas = 0 then
    begin
      Image1.Canvas.TextOut(20, 10, 'Si príliš pomalý       ');
      Image1.Canvas.TextOut(20, 50, '                - škoda');
      Button2.Enabled := False;
    end;
  end;
end;
```

Všimli ste si, ako časovač pri vypršaní času (v premennej **Cas** je 0) zastavil aj sám seba? Tým, že pri skončení hry nastavil **Button2.Enabled** na **False**, pri ďalšom tiknutí časovača (o 0,1 sekundy), zastaví aj seba, lebo priradí **Timer1.Enabled := False**.

Procedúra **Button1Click** naštartuje hru:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.TextOut(20, 10, 'Myslím si číslo do 100.');
  Image1.Canvas.TextOut(20, 50, 'Skús ho uhádnuť.');
  Edit1.Text := 'číslo';
  Button2.Enabled := True;
  Cislo := Random(100)+1;
  Pocet := 0;
  Cas := 300;
  Timer1.Enabled := True;
end;
```

Procedúra **Button2Click** spracuje náš tip na číslo:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Hadas: Integer;
begin
  Image1.Canvas.FillRect(Rect(20, 100, 400, 150));
  Hadas := StrToIntDef(Edit1.Text, 0);
  if (Hadas < 1) or (Hadas > 100) then
  begin
    Image1.Canvas.TextOut(20, 100, 'Zadaj číslo od 1 do 100');
    Edit1.Text := '';
  end
  else if Hadas = Cislo then
  begin
    Image1.Canvas.FillRect(Image1.ClientRect);
    Image1.Canvas.TextOut(20, 50, 'HURÁ - uhádol si');
    Button2.Enabled := False;
  end
  else if Hadas < Cislo then
  begin
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je málo');
    Pocet := Pocet + 1;
  end
  else
  begin
    Image1.Canvas.TextOut(20, 100, Edit1.Text + ' je veľa');
    Pocet := Pocet + 1;
  end;

  Image1.Canvas.TextOut(Image1.Width-30, 0, IntToStr(Pocet));
  if Pocet = 10 then
  begin
    Image1.Canvas.TextOut(20, 10, 'Priveľa neúspešných    ');
    Image1.Canvas.TextOut(20, 50, '        pokusov - škoda');
    Button2.Enabled := False;
  end;
end;
```

Úlohy na precvičenie

Zadanie 1	Tento projekt sa dá prepísať tak, aby namiesto grafickej plochy (**TImage**) použil niekoľko jednoduchých textov (**TLabel**). Prepíšte ho, pričom niekoľkým jednoduchým textom nastavíte rôzne veľké fonty, približne tak, ako veľké písmo sa na tom mieste použilo (napr. čas bol vypisovaný malým písmom).
Zadanie 2	Pozmeňte program na hádanie mysleného čísla tak, aby boli vidieť všetky naše tipy aj s výsledným komentárom (napr. že naše číslo je menšie ako hľadané).
Zadanie 3	Zamyslite sa, či by sa podobným spôsobom nedali hádať cifry štvorciferného čísla, ako je to v hre MasterMind: pre každý náš tip počítač oznámi, koľko cifier sme uhádli na presne svojom mieste a koľko cifier sme uhádli, ale nie sú na svojom mieste.
Zadanie 4	Existuje aj farebná verzia hry Master Mind: počítač si myslí 3 alebo 4 farby z nejakej množiny farieb. My sa to snažíme uhádnuť zadávaním nejakej trojice (štvorice) farieb, napr. pomocou stláčania tlačidiel.

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- procedúry bez parametrov aj s parametrami
- lokálne a globálne premenné
- lokálne a globálne procedúry
- komponent časovač a

Okrem toho boli uvedené tieto dôležité koncepty:

- mechanizmus volania procedúry

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Program bude generovať sériu matematických úloh (napr. na malú násobilku) v tvare **7 * 5 = ?** a používateľ by mal do vstupného riadku napísať správnu odpoveď. Po minúte testovanie zastane a program vypíše, koľko úloh sa stihlo za tento čas vyriešiť a koľko z nich bolo vyriešených správne.

Literatúra a použité zdroje

Základné materiály:

[1] Blaho A., "Informatika pre stredné školy. Programovanie v Delphi", SPN Bratislava, 2006
[2] Cieľové požiadavky na vedomosti a zručnosti maturantov z informatiky, Štátny pedagogický ústav, http://www.statpedu.sk/

Internetové zdroje pre prostredie Delphi

[3] Delphi - vysokoškolské prednášky na FMFI UK, http://www.delphi.input.sk/
[4] prijímacie pohovory z informatiky na FMFI UK, http://www.prijimacky.input.sk/
[5] Delphi Programming, http://delphi.about.com/
[6] Marco Cantu, http://www.marcocantu.com/
[7] Torry's Delphi Pages, http://www.torry.net/

Internetové zdroje pre prostredie Lazarus

[8] Lazarus wiki, http://wiki.lazarus.freepascal.org/Main_Page/sk
[9] Lazarus Documentation/sk, http://wiki.lazarus.freepascal.org/Lazarus_Documentation/sk
[10] Free Pascal, http://www.freepascal.org/

Tento študijný materiál vznikol ako súčasť národného projektu Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika v rámci Aktivity „Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ“.

Autori © RNDr. Andrej Blaho
RNDr. Ľubomír Salanci, PhD.

Názov Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Podnázov Programovanie 5

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti RNDr. Peter Gurský, PhD.
RNDr. František Galčík, PhD.

Počet strán 36

Náklad 300 ks

Prvé vydanie, Bratislava 2009

Vydal Štátny pedagogický ústav, Pluhová 8, 830 00 Bratislava, v súčinnosti s Univerzitou Pavla Jozefa Šafárika v Košiciach, Univerzitou Komenského v Bratislave, Univerzitou Konštantína Filozofa v Nitre, Univerzitou Mateja Bela v Banskej Bystrici a Žilinskou univerzitou v Žiline

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-89225-98-9