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 4

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 4

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

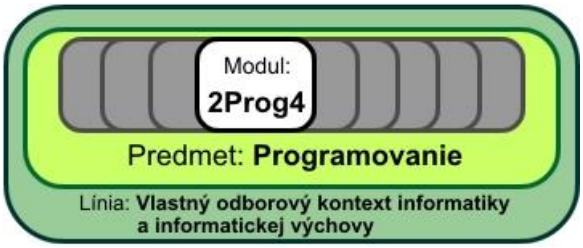
Autori:

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

Rukopis odovzdaný:

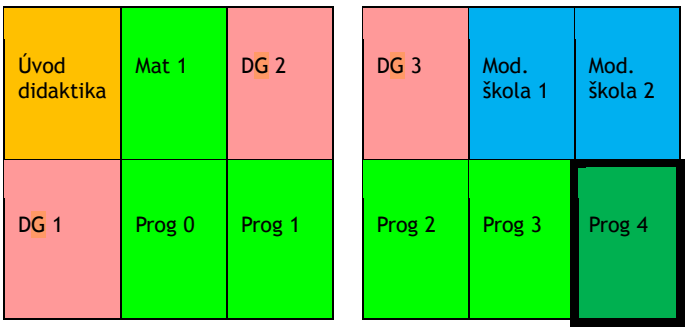
14. august 2009

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.

Prvý semester vzdelávania:



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 niektorým zaujímavým algoritmom s celočíselnou aj reálnou aritmetikou. Objasňuje problémy, ktoré súvisia s reálnou aritmetikou. Tiež sa zaoberá úlohami, v ktorých sa využívajú vnorené cykly.

Obsah

Programovanie 4 1
Identifikácia modulu 1
Zaradenie modulu 1
Abstrakt modulu 1
Obsah 2
Úvod 3
Cieľ modulu 3
Vstupné vedomosti 3
Požadované prerekvizity 3
Predpokladané vstupné vedomosti, skúsenosti a zručnosti 3
Preverenie vstupných vedomostí 3
1. tematická jednotka - úlohy s číslami 4
1. Číselné sústavy, prevody 4
2. Reálna aritmetika 8
3. Grafy funkcií 12
4. Kružnica, časť kružnice (výsek, oblúk) 19
2. tematická jednotka - vnorené cykly 25
1. Vnorené cykly 25
2. Nekonečný cyklus 31
Čo sme sa naučili v tomto module 33
Preverenie výstupných vedomostí 33
Literatúra a použité zdroje 33

Úvod

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

1. úlohy s číslami
2. vnorené cykly

Cieľ modulu

Prvá tematická jednotka **Úlohy s číslami** pokrýva tieto témy:

- číselné sústavy
- reálna aritmetika
- grafy funkcií
- kružnica

Druhá tematická jednotka **Vnorené cykly** pokrýva tieto témy:

- vnorené cykly
- nekonečný cyklus

Vstupné vedomosti

Požadované prerekvizity

Modul Programovanie 3 – vetvenie a while-cyklus.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie popísať princíp fungovania while-cyklu,
- 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 cykly. ktoré obsahujú podmienené príkazy,
- rozumie pojmu podmienka, logická hodnota a logický výraz, a tiež princípu fungovania while-cyklu.

Preverenie vstupných vedomostí

Vytvorenie jednoduchej aplikácie, ktorá zadané číslo (zo vstupného riadka) rozloží na prvočinitele a vypíše, či je to prvočíslo.

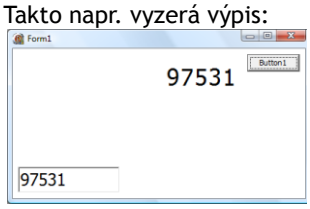
1. tematická jednotka - úlohy s číslami

1. Číselné sústavy, prevody

Pripomeňme si program z predchádzajúceho modulu, ktorým sme vypisovali cifry zadaného čísla:

```
var
  X, Cislo: Integer;
begin
  Image1.Canvas.Font.Height := 40;
  X := 300;
  Cislo := StrToInt(Edit1.Text);
  while Cislo > 0 do
  begin
    Image1.Canvas.TextOut(X, 20, IntToStr(Cislo mod 10));
    Cislo := Cislo div 10;
    X := X - 20;
  end;
end;
```

Takto napr. vyzerá výpis:

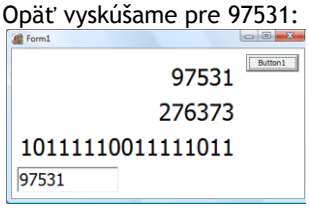


Tento program je poučný najmä v tom, že na ňom vidíme, ako postupne získavame cifry čísla v desiatkovej sústave: zoberieme poslednú cifru (**Cislo mod 10**) a číslo o poslednú cifru skrátime (**Cislo := Cislo div 10**).

Rovnaký princíp funguje pre ľubovoľnú číselnú sústavu: v tejto inej sústave poslednú cifru získame ako **Cislo mod Sustava** a skrátenie čísla o túto poslednú cifru ako **Cislo := Cislo div Sustava**. Dopíšme do programu výpis v osmičkovej a pod neho v dvojkovej sústave:

```
var
  X, Cislo: Integer;
begin
  Image1.Canvas.Font.Height := 40;
  // desiatková sústava
  X := 300;
  Cislo := StrToInt(Edit1.Text);
  while Cislo > 0 do
  begin
    Image1.Canvas.TextOut(X, 20, IntToStr(Cislo mod 10));
    Cislo := Cislo div 10;
    X := X - 18;
  end;
  // osmičková sústava
  X := 300;
  Cislo := StrToInt(Edit1.Text);
  while Cislo > 0 do
  begin
    Image1.Canvas.TextOut(X, 70, IntToStr(Cislo mod 8));
    Cislo := Cislo div 8;
    X := X - 18;
  end;
  // dvojková sústava
  X := 300;
  Cislo := StrToInt(Edit1.Text);
  while Cislo > 0 do
  begin
    Image1.Canvas.TextOut(X, 120, IntToStr(Cislo mod 2));
    Cislo := Cislo div 2;
    X := X - 18;
  end;
end;
```

Opäť vyskúšame pre 97531:



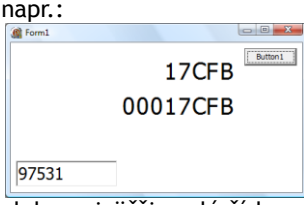
Teraz by ste vedeli vypísať celé číslo v ľubovoľnej číselnej sústave, len jej základ nemôže byť väčší ako 10. Pre väčšie číselné sústavy by sme potrebovali spoznať spôsob zobrazovania väčších cifier ako 9.

Z historických dôvodov je veľmi dôležitá šestnástková sústava. V tejto sústave máme 16 rôznych cifier: 0 až 9 a potom A, B, C, D, E a F, ktoré označujú 10 až 15. Táto sústava je stále medzi systémovými informatikmi populárna najmä preto, že obsah jedného bajtu vieme popísať maximálne dvojciferným šestnástkovým číslom od **0** do **FF**. Podobne 4 bajtové hodnoty (napr. Integer) reprezentujeme 8 cifernými číslami, kde najnižšia dvojica cifier popisuje najnižší bajt, ďalšia dvojica popisuje ďalší bajt atď.

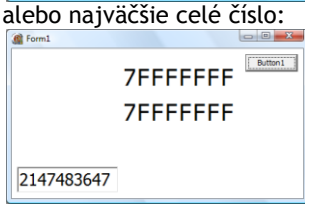
My zatiaľ z Pascalu nepoznáme reťazce ani polia, aby sa dal urobiť prevod do 16-ovej sústavy čo najelegantnejšie. Použijeme príkaz vetvenia **case**, ale neskôr to zapíšeme oveľa krajšie.

```
var
  X, Cislo, I: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 40;
  Cislo := StrToInt(Edit1.Text);
  X := 300;
  while Cislo > 0 do
  begin
    case Cislo mod 16 of
      0..9: Image1.Canvas.TextOut(X, 20, IntToStr(Cislo mod 16));
      10:   Image1.Canvas.TextOut(X, 20, 'A');
      11:   Image1.Canvas.TextOut(X, 20, 'B');
      12:   Image1.Canvas.TextOut(X, 20, 'C');
      13:   Image1.Canvas.TextOut(X, 20, 'D');
      14:   Image1.Canvas.TextOut(X, 20, 'E');
      15:   Image1.Canvas.TextOut(X, 20, 'F');
    end;
    Cislo := Cislo div 16;
    X := X - 20;
  end;
  Cislo := StrToInt(Edit1.Text);
  X := 300;
  for I := 1 to 8 do
  begin
    case Cislo mod 16 of
      0..9: Image1.Canvas.TextOut(X, 70, IntToStr(Cislo mod 16));
      10:   Image1.Canvas.TextOut(X, 70, 'A');
      11:   Image1.Canvas.TextOut(X, 70, 'B');
      12:   Image1.Canvas.TextOut(X, 70, 'C');
      13:   Image1.Canvas.TextOut(X, 70, 'D');
      14:   Image1.Canvas.TextOut(X, 70, 'E');
      15:   Image1.Canvas.TextOut(X, 70, 'F');
    end;
    Cislo := Cislo div 16;
    X := X - 20;
  end;
end;
```

Program môžeme otestovať s rôznymi hodnotami, napr.:



alebo najväčšie celé číslo:



Prevod do šestnástkovej sústavy sme urobili dvoma veľmi podobnými algoritmami: prvý rovnako a predchádzajúce prevody, vypisoval len cifry, kým bolo číslo nenulové, druhý algoritmus používa for-cyklus a teda vypisuje vždy 8 cifier.

Nájdite súvis medzi ciframi 16-ovej a dvojkovej sústavy. Porozmýšľajte aj o cifrách osmičkovej a dvojkovej sústavy.

Poznámka:

Existuje podobná funkcia ako **IntToStr**, ktorá prekonvertuje celé číslo do textu. **IntToStr** konvertuje do desiatkovej reprezentácie, naproti tomu nová funkcia **IntToHex** konvertuje dané číslo do 16-ovej reprezentácie. Táto funkcia má ešte jeden parameter, ktorý určuje počet cifier výsledku, napr.

```
IntToHex(97531, 6) = '017CFB'
IntToHex(97531, 4) = '7CFB'
IntToHex(-1, 8) = 'FFFFFFFF'
```

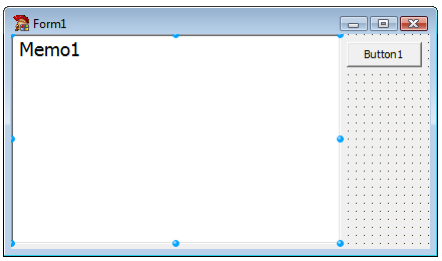
Poznámka:
Záporné čísla sú v pamäti uložené v tzv. doplnkovom kóde, a preto je hodnota -1 reprezentovaná tak, že v binárnom zápise obsahuje samé 1.

Doteraz sme všetky výpisy našich výpočtov robili **do grafickej plochy**. Kým toho

nebolo veľa, zvládli sme rôzne hodnoty vypisovať do riadku alebo stĺpca a používali sme na to pomocné premenné X, resp. Y: pri každom výpise sme zväčšili X, resp. Y o nejakú konštantu (v závislosti od veľkosti nastaveného písma), aby sa ďalší text písal za týmto.

Namiesto grafickej plochy môžeme v programe pracovať **s textovou plochou**: nebudeme používať príkaz na výpis textu **TextOut**, ktorý ho vypisuje na konkrétne súradnice, ale príkaz **Append**, ktorý vypíše text do nového riadka za naposledy vypísaný. Takýchto textov do textovej plochy môžeme vypísať aj viac, ako sa zmestí do plochy – vtedy sa text automaticky roluje a text sa pridáva vždy na koniec.

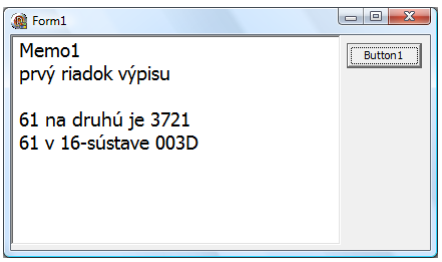
Vytvorme novú aplikáciu, ktorá namiesto grafickej plochy použije textovú plochu. Vložme do formuláru okrem tlačidla textovú plochu – komponent **TMemo**:



Zároveň môžeme tomuto komponentu zväčšiť písmo (dvojklikneme na nastavenie **Font** a zväčšíme jeho veľkosť). Otestujeme túto textovú plochu týmto programom:

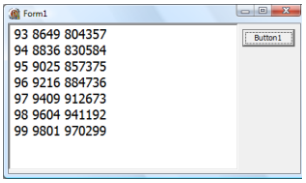
```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  Memo1.Lines.Append('prvý riadok výpisu');
  Memo1.Lines.Append('');
  I := 61;
  Memo1.Lines.Append(IntToStr(I) + ' na druhú je ' + IntToStr(I * I));
  Memo1.Lines.Append(IntToStr(I) + ' v 16-sústave ' + IntToHex(I, 4));
end;
```

Všimnime si, že výpis do textovej plochy robíme príkazom **Memo1.Lines.Append**, ktorý je analogický príkazu pre grafickú plochu **Image1.Canvas.TextOut**. Príkaz **Append** má jediný parameter, ktorým je vypisovaný text. Prvý príkaz **Append** vypíše vetu ***prvý riadok výpisu***. Za ním je výpis, ktorý obsahuje tzv. prázdny text, t.j. len dva apostrofy za sebou. Takýto výpis vypíše prázdny riadok. Ďalšie dva výpisy vypíšu výpočet druhej mocniny čísla 61 a tiež prevod do 16-ovej sústavy pomocou **IntToHex**:



Textová plocha sa najlepšie využije pri väčšom počte výpisov. Napr. vypíšeme tabuľku druhých a tretích mocnín všetkých dvojciferných čísel od 10 do 99:

```
var
  I: Integer;
begin
  Memo1.Clear;
  for I := 10 to 99 do
    Memo1.Lines.Append(IntToStr(I) + ' ' + IntToStr(I * I) + ' ' +
      IntToStr(I * I * I));
end;
```



Po spustení tohto programu vidíme len niekoľko posledných riadkov výpisu, tak ako to vidíme na okraji strany.

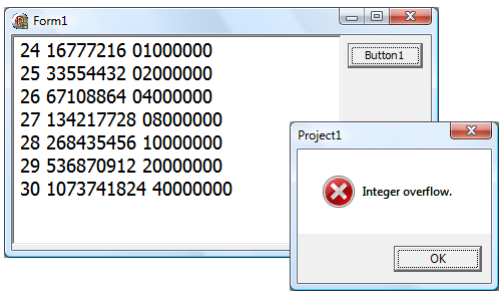
Ostatné riadky sa automaticky vyrolovali hore. Keď ale do textovej plochy klikneme myšou, môžeme sa v nej pohybovať pomocou šípok hore a dole, resp. klávesov <**Page Up**> a <**Page Down**> a pozrieť si ľubovoľné z vypísaných riadkov.

Všimnite si ešte prvý príkaz programu **Memo1.Clear**, ktorý zmaže celý obsah grafickej plochy a teda aj pôvodný prvý riadok s textom **Memo1**.

Ďalší program vypíše všetky mocniny 2 od 2^0 až do 2^{30}. Výpis urobíme v desiatkovej aj v šestnástkovej sústave:

```
var
  I, M: Integer;
begin
  M := 1;
  for I := 0 to 30 do
  begin
    Memo1.Lines.Append(IntToStr(I) + ' ' + IntToStr(M) + ' ' +
      IntToHex(M, 8));
    M := M * 2;
  end;
end;
```

Premenná **M** má pred for-cyklom hodnotu 1, čo je 2^0. V cykle sa táto hodnota vypíše aj v šestnástkovej sústave a vynásobí sa dvoma, teda má hodnotu 2^1 - toto je hodnota pre ďalší prechod cyklu, keď v **I** bude 1 a teda sa počíta 2^1. Každý prechod vynásobí **M** dvomi a keďže to robí 31-krát pri poslednom prechode by v **M** malo byť 2^{31}. Táto hodnota je ale väčšia ako **Maxint** a preto dostávame chybovú správu o pretečení:

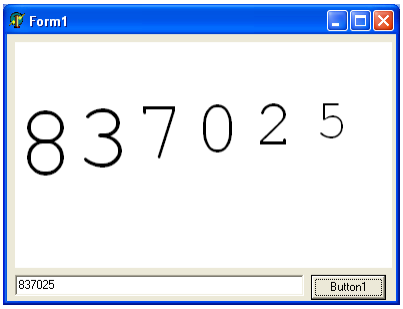


Navrhnite riešenie tejto úlohy tak aby program nespadol ani pri poslednom prechode for-cyklu.

Úlohy na precvičenie

Zadanie 1

číslo 1101 v rôznych číselných sústavách reprezentuje rôzne hodnoty, napr. v dvojkovej je to hodnota 13 a v trojkovej 37 ($3^3+3^2+3^0$); vypíšte toto číslo vo všetkých sústavách od 2 do 16 (teda robíme prevod zo sústavy N do 10-ovej sústavy)

Zadanie 2	program prečíta dve čísla **N** a **K** a zistí ciferný súčet čísla **N** v **K**-sústave, napr. pre **N**=16 a **K**=2 je ciferný súčet 1, lebo **N** v dvojkovej sústave je 10000; výsledok vypíše v 10-ovej sústave (napr. **IntToStr**)
Zadanie 3	program vypíše do grafickej plochy cifry zadaného čísla ale tak, že sa jednotlivé cifry budú zmenšovať, napr. 
Zadanie 4	program vypíše v desiatkovej aj v šestnástkovej sústave všetky čísla, ktoré v 16-ovej sústave obsahujú len cifry A, t.j. A, AA, AAA, ... (zrejme najväčšie z nich bude mať 8 cifier)
Zadanie 5	zadané číslo zobrazte v grafickej ploche tak, že každá jeho cifra sa zobrazí rôzne veľkými obdĺžnikmi a to podľa veľkosti cifry; tieto obdĺžniky naukladajte tesne vedľa seba, napr. pre cifru 0 bude obdĺžnik výšky 10, pre cifru 1 výšky 20, pre cifru 2 výšky 30, atď.

2. Reálna aritmetika

Pri riešení úloh pomocou počítača sa veľmi často stretávame s úlohami, v ktorých potrebujeme pracovať buď s desatinnými číslami, alebo s celými číslami, ktoré presahujú povolený rozsah pre **Integer**. Na toto nám pascal ponúka ďalší typ - nazýva sa typ reálne číslo (desatinné číslo) a má meno **Real**. Tento typ má veľmi podobné vlastnosti ako celočíselný typ **Integer**, ale má aj svoje špecifiká. V prvom rade by sme si mali uvedomiť, že niekedy nám môžu vzniknúť nie úplne presné výsledky - hovoríme, že reálna aritmetika má chyby (ukážeme to na príklade). Preto túto aritmetiku používame obozretne, len ak naozaj musíme a s jej chybami pritom počítame.

Pozrime sa na vlastnosti reálnych čísel (desatinných čísel) a porovnajte ich s celými číslami:

- konštanty obsahujú desatinnú bodku a/alebo exponenciálnu časť, napr.

```
3.14, -0.5, 3000000000.0, 3E9, 4.7E-3
```

- exponent v môže byť približne v rozsahu <-300, 300>,
- presnosť výpočtov je približne na 15 desatinných miest, ale niektoré operácie nie sú až tak presné, ako by sme si mysleli z matematiky zo základnej školy,
- premenné typu **Real** zaberajú v počítači 8 bajtov (dvojnásobne viac ako **Integer**),
- základné operácie sú +, -, *, /, pričom, ak jeden z operandov je celé číslo, automaticky sa konvertuje na reálne, napr. ak zapíšeme

```
1 + 1.5      // bude sa počítať 1.0 + 1.5
1 / 3        // bude sa počítať 1.0 / 3.0
```

- automatická konverzia sa robí aj pri priradení: ak do reálnej premennej chceme priradiť celé číslo, toto sa automaticky konvertuje na reálne, napr.

```
RCislo := 2*5;     // sa prekonvertuje na RCislo := 10.0;
```

- opačné priradenie (do celočíselnej premennej reálnu hodnotu) nie je povolené! Ak také niečo potrebujeme musíme použiť niektorú zaokrúhľovaciu funkciu.
- fungujú porovnávania <, <=, >, >=, <>, =
- existuje množstvo štandardných funkcií, napr.
  - **Sqrt** - odmocnina
  - **Sqr** - druhá mocnina
  - **Sin**, **Cos**, **Tan** - trigonometrické funkcie
  - **Abs** - absolútna hodnota
  - **Round** - zaokrúhľovanie - vráti celé číslo
  - **Trunc** - odtrhne desatinnú časť - vráti celé číslo
  - **FloatToStr** - vráti reťazec
  - **StrToFloat** - z reťazca spraví celé číslo
- reálny typ nemôžeme použiť
  - ako riadiacu premennú for-cyklu (môžeme to zapísať while-cyklom),
  - ako hodnotu v podmienenom príkaze case.

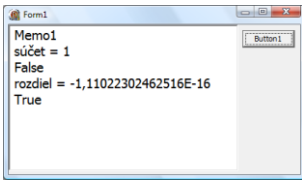
Prvý príklad, ktorý predstaví reálnu aritmetiku bude jednoduchá ukážka nepresnosti výpočtov.

```
var
  RCislo: Real;
  I: Integer;
begin
  RCislo := 0;
  for I := 1 to 10 do
    RCislo := RCislo + 0.1;
  Memo1.Lines.Append('súčet = ' + FloatToStr(RCislo));
  if RCislo = 1 then
    Memo1.Lines.Append('True')
  else
    Memo1.Lines.Append('False');

  Memo1.Lines.Append('rozdiel = ' + FloatToStr(RCislo-1));

  if Abs(RCislo - 1) < 0.001 then
    Memo1.Lines.Append('True')
  else
    Memo1.Lines.Append('False');
end;
```

Po spustení programu vidíme:



Tento program najprv 10-krát pripočíta k reálnemu číslu **RCislo** hodnotu 0.1. Potom toto číslo vypíše pomocou **FloatToStr** do textovej plochy: môžeme vidieť správny výsledok 1. Lenže tento výsledok nie je naozaj jedna: ak ho v nasledovnom podmienenom príkaze porovnáme s 1, dozvieme sa, že tento test (**RCislo = 1**) má hodnotu **False**. Ak ale vypíšeme, aký je rozdiel medzi **RCislo** a 1, dostávame zaujímavé číslo 1.11022302462516E-16, ktoré je veľmi malé (asi 0.0000000000000001), ale stačí na to, aby sa **RCislo** nerovnalo 1. Keďže programátori vedia o tomto probléme, častejšie ako rovnosť testujú, či je rozdiel dostatočne malý, napr. namiesto testu **A = B** použijeme **Abs(A - B) <** malá hodnota.

Poznámka: niektoré aj pekne vyzerajúce reálne konštanty, napr. 0.1, sú vnútorne v počítači reprezentované jej približnou hodnotou. Preto spočítavanie takýchto približných hodnôt niekedy nedá očakávaný výsledok.

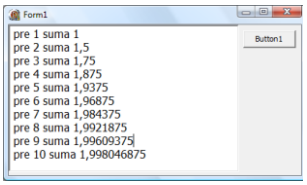
Reálna aritmetika sa používa najčastejšie na rôzne výpočty. Nasledujúci príklad počíta súčet radu

$$\sum 1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots$$

Na začiatok sa pozrime, ako sa postupne mení súčet po každom pričítaní člena:

```
var
  Suma, Clen: Real;
  I: Integer;
begin
  Suma := 0;
  Clen := 1;
  for I := 1 to 100 do
  begin
    Suma := Suma + Clen;
    Clen := Clen / 2;
    Memo1.Lines.Append('pre ' + IntToStr(I) +
      ' suma ' + FloatToStr(Suma));
  end;
end;
```

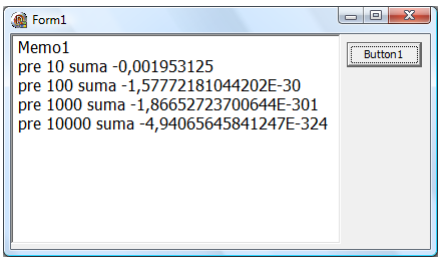
Môžeme vidieť, že sa výsledok veľmi rýchlo blíži k 2:



a od nejakej hodnoty (po 54 súčte) sa výsledok nemení a je stále 2. Ak by sme chceli zistiť, ako sa výsledok blíži ku 2 pre rôzne počty členov, môžeme tento program trochu pozmeniť. Sumu nepočítame od 0, ale od -2 a preto sa bude vo výpise objavovať rozdiel vypočítanej hodnoty od 2. Reálna aritmetika funguje s presnosťou približne 15 platných miest, a preto odčítanie úvodnej 2 v tomto príklade umožňuje počítať ďalšie cifry sumy.

```
var
  Suma, Clen: Real;
  I, N: Integer;
begin
  Suma := -2;
  Clen := 1;
  N := 10;
  for I := 1 to 10000 do
  begin
    Suma := Suma + Clen;
    Clen := Clen / 2;
    if I = N then
    begin
      Memo1.Lines.Append('pre ' + IntToStr(N) +
        ' suma ' + FloatToStr(Suma));
      N := N * 10;
    end;
  end;
end;
```

Keďže sme súčet začali s -2, dostávame presnejšie výsledky ako v predchádzajúcom príklade.



Výsledok pre 10000 už určite nie je presný, lebo premenná **Clen** nadobudla hodnotu 0 už oveľa skôr. Zistite, pri ktorom prechode cyklom sa **Clen** bude rovnať 0.

V predchádzajúcom module sme pri podmienenom príkaze **if** náhodne kreslili do grafickej plochy bodky a zafarbovali ich podľa rôznych pravidiel. V tom čase sme ešte nemali reálnu aritmetiku a až teraz môžeme ukázať, ako sa počíta vzdialenosť 2 bodov so súradnicami (x_1, y_1), (x_2, y_2). Z matematiky vieme, že je to tento

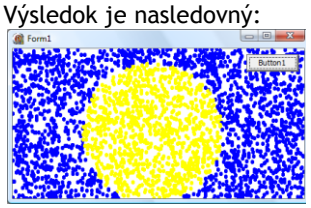
jednoduchý vzorec:

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Napíšme bodkovací program, ktorý bodky vzdialené od (200, 120) o viac ako 100 zafarbí na modro a bodky bližšie k (200, 120) zafarbí na žlto:

```
var
  X, Y, I: Integer;
begin
  Image1.Canvas.Pen.Width := 5;
  for I := 1 to 3000 do
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    if Sqrt(Sqr(X - 200) + Sqr(Y - 120)) > 100 then
      Image1.Canvas.Pen.Color := clBlue
    else
      Image1.Canvas.Pen.Color := clYellow;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X, Y + 1);
  end;
end;
```

Výsledok je nasledovný:



Dostávame vybodkovaný kruh s polomerom 100 a so stredom v (200, 120).

Ďalší príklad využíva tento istý mechanizmus, ale robí rozdiel dvoch kruhov: zisťujeme pozíciu, ktorá je v prvej kružnici, ale nie je v druhej, ktorá sa s ňou čiastočne prekrýva:

```
var
  X, Y, I: Integer;
begin
  Image1.Canvas.Pen.Width := 5;
  for I := 1 to 10000 do
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    if (Sqrt(Sqr(X - 150) + Sqr(Y - 120)) > 80) or
       (Sqrt(Sqr(X - 230) + Sqr(Y - 120)) < 100) then
      Image1.Canvas.Pen.Color := clBlue
    else
      Image1.Canvas.Pen.Color := clYellow;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X, Y + 1);
  end;end;
```

Po spustení dostaneme:

Úlohy na precvičenie

Zadanie 1	podobne ako posledný príklad pomocou bodkovania nakreslil mesiac ako rozdiel dvoch kruhov, znázornite oblasť ktorá je prienikom, resp. zjednotením dvoch kruhov
Zadanie 2	program, ktorý metódou farebných náhodných bodiek, vykreslí 5 olympijských farebných kruhov

Zadanie 3	Ludolfovo číslo môžme počítať rôznymi spôsobmi; program pre jeho výpočet, napr. pomocou týchto vzorcov: • $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$ • $\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdots$
Zadanie 4	vypočítať **e** (Eulerovo číslo - základ prirodzených logaritmov, jeho približná hodnota je 2. 7182818) podľa tohto vzorca: • $\frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots$
Zadanie 5	zistite, či sa dá pomocou reálnej aritmetiky vypočítať, ale presne, faktoriál väčšieho čísla, ako by sa dal iba pomocou celočíselnej aritmetiky
Zadanie 6	pre zadané koeficienty kvadratickej rovnice • $ax^2 + bx + c = 0$ vypočítať obidva korene

3. Grafy funkcií

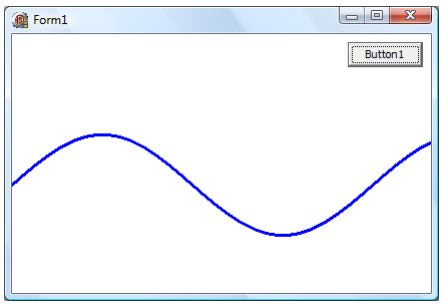
Bez reálnej aritmetiky sa nezaobídeme ani pri kreslení grafov funkcií. Ďalší program ukazuje, ako môžeme do grafickej plochy nakresliť grafy rôznych funkcií.

Graf funkcie kreslíme tak, že postupne pre **X** z nejakého intervalu počítame príslušné **Y**. Lenže grafická plocha má y-ové súradnice orientované opačne, ako sme zvyknutí z matematiky a 0 je v hornom riadku plochy. Preto pri výpočte **Y** jeho hodnotu otáčame a posúvame, napr. takto **150-50*Sin(X)**. Tiež si všimnite, že goniometrické funkcie sú definované pre radiány a preto sme ich prepočítavali zo stupňov na radiány. Použili sme pri tom preddefinovanú hodnotu **Pi**. Keďže **Y** je reálne číslo, nemôžeme ho poslať ako hodnotu do príkazu **MoveTo**. Preto sme ho museli prekonvertovať na celé číslo, napr. funkciou **Round**. Najprv graf funkcie **Y = Sin(X)**:

```
var
  I: Integer;
  X, Y: Real;
begin
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180;         // I je v stupňoch, X v radiánoch
    Y := 150 - 50 * Sin(X);
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
end;
```

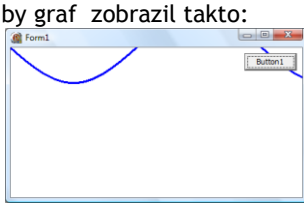
Všimnite si, že vo for-cykle sa rozhodneme, či urobíme **MoveTo** alebo **LineTo**, a to podľa toho, či sme v prvom prechode cyklu (**I = 0**) alebo už v ďalších prechodoch. Príkaz **MoveTo** by sme mohli robiť ešte pred cyklom, len výpočet **Y** by sme museli zopakovať aj pred cyklom.

Po spustení:



Pri výpočte hodnoty funkcie **Y = Sin(X)** sme túto hodnotu ešte násobili 50 a odčítali sme ju od 150: **Y := 150-50*Sin(X)**. O funkcii Sin vieme, že jej hodnotami sú desatinné čísla (reálne čísla) v intervale <-1, 1>. To, že sme tieto hodnoty funkcie násobili konštantou 50, znamená, že teraz **50*Sin(X)** bude dávať reálne hodnoty z intervalu <-50, 50>.

Program, v ktorom by sme do premennej **Y** počítali funkciu ako:
Y := 50 * Sin(X);
by graf zobrazil takto:



Pripočítaná hodnota 150 v tomto prípade posunie hodnoty funkcie, tak aby sa graf objavil v strede grafickej plochy (predpokladáme, že grafická plocha má výšku približne 300 bodov). Prečo sme sem ešte dali aj záporné znamienko:

Y := 150 - 50 * Sin(X);

Súradná sústava v grafickej ploche má Y-ovú súradnicu orientovanú opačne, ako je bežné v matematike: Y-súradnica 0 je na hornej hrane plochy a smerom nadol sa jej hodnota zvyšuje. Keďže sme chceli funkciu **Sin** znázorniť čo najvernejšie, ešte sme ju otočili. Môžete vyskúšať, ako to bude vyzerať bez odčítania, ale s pričítaním.

Všimnite si, že podobný postup budeme používať aj pri kreslení grafov iných funkcií.

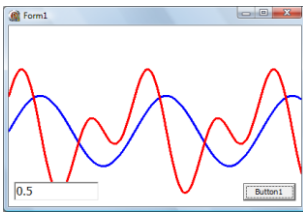
Teraz do pôvodného programu dopíšeme kreslenie grafu druhej funkcie **Cos(X)+Sin(2X)** a nakreslíme ho inou farbou:

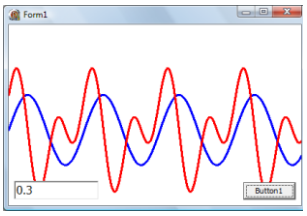
```
var
  I: Integer;
  X, Y: Real;
begin
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180;
    Y := 150 - 50 * Sin(X);
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
  Image1.Canvas.Pen.Color := clRed;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180;
    Y := 150 - 50 * (Cos(X) + Sin(2 * X));
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
end;
```

Po spustení dostávame:

Keďže šírka grafickej plochy v tomto príklade je okolo 400 bodov, z grafu funkcie **Sin** (modrá krivka) vidíme interval približne <0, 400> stupňov. Samozrejme, že môžeme prerábať naše vzorce v programe tak, aby sme videli funkcie nakreslené v iných mierkach. Pridáme vstupný riadok (**Edit1**), v ktorom môžeme zadať ***mierku*** na vykresľovanie grafu. Program upravíme takto:

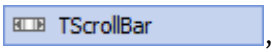
Podľa zadanej mierky:



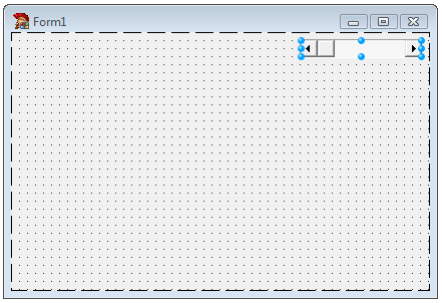


```
var
  I: Integer;
  X, Y, Mierka: Real;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Mierka := StrToFloat(Edit1.Text);
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180 / Mierka;
    Y := 150 - 50 * Sin(X);
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
  Image1.Canvas.Pen.Color := clRed;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180 / Mierka;
    Y := 150 - 50 * (Cos(X) + Sin(2 * X));
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
end;
```

Čo sa tu zmenilo: pridali sme zmazanie grafickej plochy na začiatku programu, do premennej **Mierka** sme priradili hodnotu zo vstupného riadku (použili sme konverznú funkciu **StrToFloat**) a X-ovú súradnicu sme touto mierkou vydelili. Ak bude Mierka = 1, tak sa nakreslí to isté, ako predtým. Ak bude mať mierka hodnotu 2, graf sme natiahli na dvojnásobok - teraz je ešte viac roztiahnutý. Hodnota mierky menšia ako 1 tento graf naopak "zahusťuje".

Ukážeme elegantnejšie riešenie zmeny mierky pri kreslení funkcií: využijeme grafický komponent posúvač (**ScrollBar**). Do formulára namiesto tlačidla **Button1** a vstupného riadku **Edit1** položíme komponent posúvač 

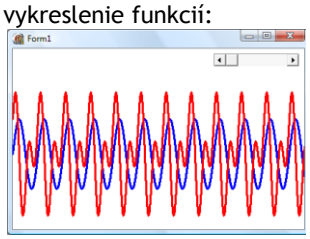
, ktorý teraz dostáva meno **ScrollBar1**. Tento komponent bude pri každej zmene, t.j. pri posúvaní bežca, vyvolávať procedúru **SCrollBar1Change**. Po položení komponentu do formulára to môže vyzerať takto:

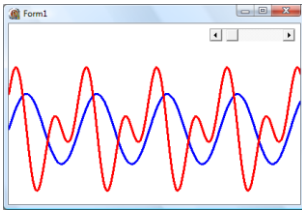


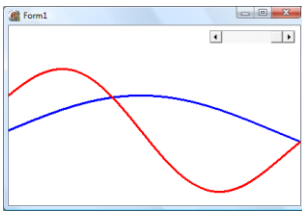
Teraz dvojklikneme na tento nový komponent a do procedúry **ScrollBar1Change**. prekopírujeme všetko, čo sme predtým mali v **Button1Change**:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
var
  I: Integer;
  X, Y, Mierka: Real;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Mierka := ScrollBar1.Position / 50 + 0.1;
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180 / Mierka;
    Y := 150 - 50 * Sin(X);
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
  Image1.Canvas.Pen.Color := clRed;
  for I := 0 to Image1.Width do
  begin
    X := I * Pi / 180 / Mierka;
    Y := 150 - 50 * (Cos(X) + Sin(2 * X));
    if I = 0 then
      Image1.Canvas.MoveTo(I, Round(Y))
    else
      Image1.Canvas.LineTo(I, Round(Y));
  end;
end;
```

Po spustení programu môžeme posúvať posúvač a pozorovať, ako sa mení vykreslenie funkcií:





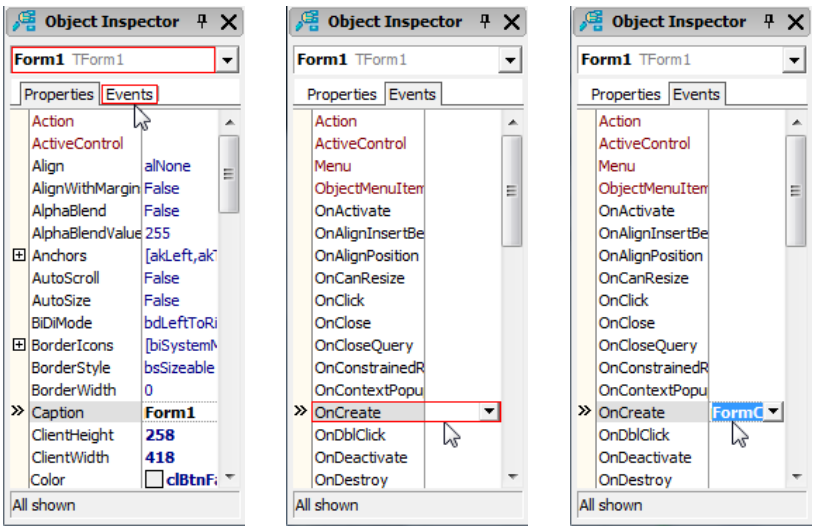


Ako vlastne funguje posúvač: pri každej zmene bežca na posúvači (napr. ťahaním myšou, alebo klikaním na smerové tlačidlá) sa vyvolá udalosť **onChange**, t.j. procedúra **ScrollBar1Change**. Je to podobný mechanizmus ako pri klikaní na obyčajné tlačidlá (napr. **Button1Click**). V našej procedúre **ScrollBar1Change** vieme zistiť, kde sa momentálne nachádza tento bežec. Zapisujeme to ako **ScrollBar1.Position**. Hodnotou tejto pozície je niektoré z čísel z intervalu hodnôt pre minimálnu a maximálnu hodnotu, t.j. <**Min**, **Max**>. Zrejme **ScrollBar1.Min** bude hodnota, keď bežec je úplne vľavo a **ScrollBar1.Max**, keď bude úplne vpravo. Keď bude bežec niekde v strede posúvača, tak hodnota **ScrollBar1.Position** bude tiež v strede intervalu. **Min**, **Max** aj **Position** môžeme nastaviť už pri návrhu formuláru – ešte pred spustením projektu.

Pri štarte tejto aplikácie sa automaticky nespúšťa procedúra ScrollBar1Change, lebo táto sa zavolá len pri zmene posúvača. Preto pri štarte sa ešte nič nevykreslí a celá plocha je zatiaľ prázdna. Až keď pohneme posúvačom, vidíme prvé zobrazenie grafu. Tu by sa nám zišlo, keby bol graf zobrazený už pri štarte projektu.

Využijeme na to jednu udalosť, ktorá automaticky vzniká pri štarte aplikácie. Už vieme, že pri každom zatlačení tlačidla, ale aj pri každom posunutí posúvača, vzniká v systéme udalosť a v našom programe píšeme procedúry, ktoré sa vykonávajú pri týchto udalostiach, napr. **Button1Click**, **ScrollBar1Change**.

Udalosť, ktorú môžeme používať na inicializáciu našich programov vzniká vo formulári počas jeho vytvárania (tzv. **Form Create**). Aby sme ju mohli používať, musíme prejsť do ***Inšpektora objektov***, pričom musíme mať nastavený objekt **Form1**. V Inšpektore objektov potom prejdeme do záložky **Events** (Udalosti) a nájdeme riadok s udalosťou **OnCreate**. Teraz, keď dvojklikneme do voľného políčka pri tejto udalosti, v editovacom okne sa predpripraví procedúra **FormCreate**. Táto bude slúžiť na inicializáciu našich programov.



Prázdna procedúra **FormCreate** vyzerá takto:

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

end;
```

Do tejto procedúry môžeme zapísať všetko to, čo by sa nám zišlo automaticky spustiť už pri štarte aplikácie. Často sú to priradenia pre rôzne nastavenia komponentov (napr. písmo pre grafickú plochu), ale môžeme tu aj zatlačiť tlačidlo, napr. **Button1.Click**.

My do tejto inicializácie v procedúre **Form1Create** zmeníme posúvač a tým sa vykreslí počiatočný tvar funkcií:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ScrollBar1.Position := 50;
end;
```

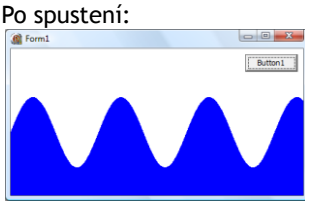
Poznámka:
*Veľmi podobným komponentom ako posúvač **ScrollBar** je aj posúvač **TrackBar** v palete komponentov **Win32**. Líši sa len malými detailmi a dizajnom. Môžete ho pokusne vyskúšať namiesto **ScrollBar**.*

Posúvačov môžeme mať v projekte ľubovoľné množstvo a každý z nich môže nastavovať iné parametre, napr. mierku pre y-ovú súradnicu (v našom programe je to konštanta 50, ktorou násobíme výsledok funkcie), posun x-ovej súradnice pre niektorú z funkcií, konštantu, ktorou násobíme parameter **X** v druhej funkcii a pod.

Vráťme sa ku kresleniu grafov funkcií. Nakreslíme graf funkcie **Sin(X)**, ale nie pomocou **for** ale pomocou **while**-cyklu:

```
var
  X, Y: Real;
begin
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  X := 0;
  while X < Image1.Width / 20 do
  begin
    Y := 120 - 50 * Sin(X);
    Image1.Canvas.MoveTo(Round(X * 20), Round(Y));
    Image1.Canvas.LineTo(Round(X * 20), Image1.Height);
    X := X + 0.05;
  end;
end;
```

Po spustení:

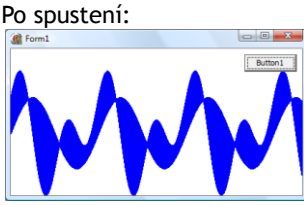


Všimnime si niekoľko zmien: premenná **X** má priamo hodnotu v radiánoch a mení sa s krokom 0.05. Aby sme takúto krivku vedeli vykresliť, X-ovú súradnicu násobíme 20 a Y-ovú 50. Každý nakreslený bod na krivke spojíme úsečkou s dolnou hranou grafickej plochy (súradnica **X, Height**).

Do tohto obrázku môžeme zabudovať aj druhú funkciu, napr. **Cos(X)+Sin(2X)** tak, že úsečkou budeme spájať body na jednej aj druhej krivke s rovnakou súradnicou **X**:

```
var
  X, Y, Y2: Real;
begin
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  X := 0;
  while X < Image1.Width / 20 do
  begin
    Y := 120 - 50 * Sin(X);
    Y2 := 120 - 50 * (Cos(X) + Sin(2 * X));
    Image1.Canvas.MoveTo(Round(X * 20), Round(Y));
    Image1.Canvas.LineTo(Round(X * 20), Round(Y2));
    X := X + 0.05;
  end;
end;
```

Po spustení:

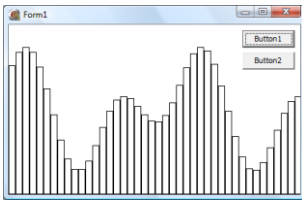


Niekedy sa stretávame s úlohou nakresliť stĺpcový diagram z nejakých zadaných hodnôt. My tu využijeme niektorú jednoduchú funkciu, napr. **Cos(X)+Sin(2X)**. Diagram budeme kresliť tak, že na zadanej krivke budeme počítať len body s nejakým väčším krokom pre **X**, napr. s krokom 10. Potom pre každé takéto **X**, **Y** nakreslíme obdĺžnik, ktorého spodná hrana je na spodnom okraji grafickej plochy.

Aby sme lepšie zviditeľnili graf funkcie, z ktorej sme vlastne vytvorili stĺpcový diagram, do aplikácie vložíme druhé tlačidlo **Button2**, ktoré do nakresleného diagramu dokreslí graf.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;
begin
  X := 5;
  while X < Image1.Width do
  begin
    Y := Round(120 - 50 * (Cos(X / 40) + Sin(2 * X / 40)));
    Image1.Canvas.Rectangle(X - 5, Y, X + 5, Image1.Height);
    X := X + 10;
  end;
end;
```

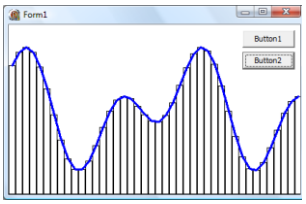
Tlačidlo **Button1** nakreslí:



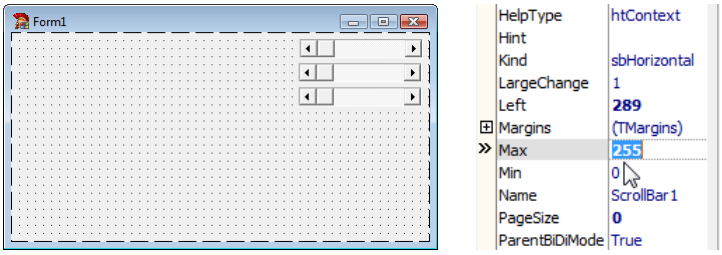
a pre tlačidlo **Button2** je to veľmi podobné:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  X, Y: Integer;
begin
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.Pen.Color := clBlue;
  X := 5;
  while X < Image1.Width do
  begin
    Y := Round(120 - 50 * (Cos(X / 40) + Sin(2 * X / 40)));
    if X = 5 then
      Image1.Canvas.MoveTo(X, Y)
    else
      Image1.Canvas.LineTo(X, Y);
    X := X + 10;
  end;
end;
```

Tlačidlo **Button2** dokreslí:



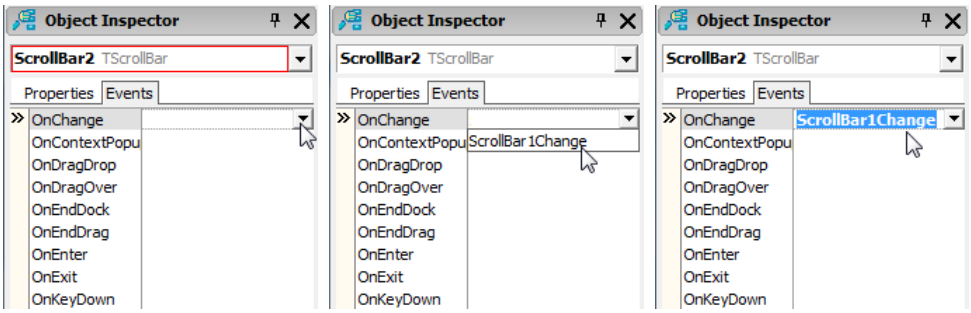
Na záver tejto časti ukážeme použitie troch posúvačov na vizualizovanie RGB farby. Do formulára vložíme 3 posúvače, každý s minimom 0 a maximom 255. Pozície bežca týchto troch posúvačov budú nastavovať zložky RGB a pri každej zmene niektorého z nich program zmaže grafickú plochu (**FillRect**) takto nastavenou farbou štetca. Najprv teda do formuláru vložíme tri komponenty **TScrollBar** a všetkým trom zmeníme ich nastavenie **Max**:



Najprv pripravíme správanie programu, keď sa pohne prvý posúvač:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
var
  R, G, B: Integer;
begin
  R  := ScrollBar1.Position;
  G  := ScrollBar2.Position;
  B  := ScrollBar3.Position;
  Image1.Canvas.Brush.Color := RGB(R, G, B);
  Image1.Canvas.FillRect(Image1.ClientRect);
  Caption := 'RGB(' + IntToStr(R) +', ' + IntToStr(G) +', ' +
    IntToStr(B) +')';
end;
```

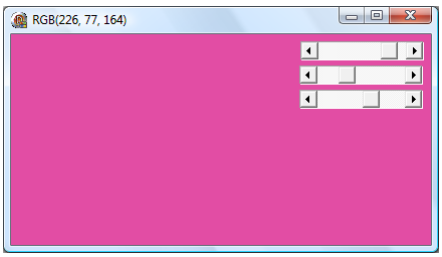
Program zatiaľ nefunguje správne, lebo správne reaguje len prvý posúvač. Rovnaký text programu by sme mohli prekopírovať aj do **ScrollBar2Change** aj do **ScrollBar3Change**, ale my sa tu naučíme niečo nové: nebudeme dvojklikať na tieto posúvače, ale v ***Inšpektore objektov*** oznámime, že oba tieto ďalšie posúvače sú obsluhované rovnakou procedúrou:

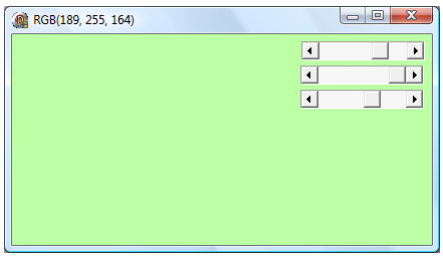


V ***Inšpektore objektov*** sa nastavíme na komponent **ScrollBar2** a v záložke **Events** nájdeme udalosť **OnChange**. Nebudeme sem dvojklikať, ale zatlačíme rozbaľovacie tlačidlo. Tu sa nám objaví procedúra **ScrollBar1Change**, ktorú zvolíme aj pre **ScrollBar2**. Rovnaký postup urobíme aj pre tretí posúvač.

Ešte si všimnite posledný riadok programu, v ktorom sa do **Caption** priradil nejaký text. V tomto prípade **Caption** označuje titulný riadok celej aplikácie (v modrom pruhu) a preto sa pri každej zmene niektorého z posúvačov v titulnom riadku vypíše momentálne nastavené RGB.

Pri posunutí ľubovoľného z týchto posúvačov môžeme "namiešať" ľubovoľnú farbu





Všimnite si teraz titulné riadky aplikácie: na prvej RGB(226, 77, 164) a na druhej RGB(189, 255,164).

Úlohy na precvičenie

Zadanie 1	podobne ako sa kreslil priebeh funkcie **Sin(X)**, sa nakreslí funkcia tangens **Tan(X)**; treba dať pozor na to, že v niektorých bodoch je funkcia nedefinovaná, resp. jej hodnota môže padnúť mimo rozsah typu **Real**
Zadanie 2	pozmeňte program, ktorý pomocou posúvača menil mierku funkcií **Sin(X)** a **COS(X)+SIN(2*X)** takto: namiesto mierky bude posúvač meniť koeficient **A** druhej funkcie **COS(X)+SIN(A*X); A** nech sa prepočíta z posúvača ako **Position/20**
Zadanie 3	zo zadaných parametrov **A**, **B** a **C** (tri vstupné riadky) vykreslite priebeh kvadratickej funkcie $A\,x^2 + B\,x + C$
Zadanie 4	pomocou štyroch rôznych posúvačov zviditeľnite priebeh nejakej funkcie so 4 parametrami, napr. `f(x) = a + b * sin(c x + d)` kde parametre **a**, **b**, **c**, **d** sa budú meniť pri posúvaní príslušných posúvačov

4. Kružnica, časť kružnice (výsek, oblúk)

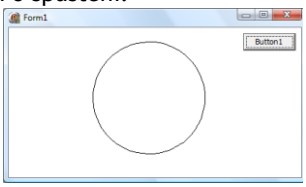
V predchádzajúcej časti sme vizualizovali funkcie, ktoré boli zadané explicitne v tvare ***y = f(x)***. Ďalším spôsobom je zápis v parametrickom tvare sústavou rovníc ***x = f1(t)***, ***y = f2(t)***, kde **t** je parameter funkcie. Najznámejšou a pre nás teraz najužitočnejšou parametrickou funkciou je rovnica kružnice:

```
x = x0 + r * cos t
y = y0 + r * sin t
```

kde **r** je polomer kružnice, (**x0**, **y0**) je jej stred a **t** je parameter funkcie z intervalu <0, 2π>. Zapíšme nakreslenie grafu tejto parametrickej rovnice do programu. V cykle využívame premennú **Uhol**, ktorá postupne nadobúda hodnoty od 0 do 6.3, čo je približne **2π**. Keďže ju zvyšujeme o 0.1, cyklus prejde 64-krát, t.j. kružnica sa nakreslí zo 63 krátkych úsečiek, ktoré spájajú body na kružnici. V tomto programe kreslíme kružnicu so stredom v (200, 100) a s polomerom 80 bodov:

```
var
  X, Y, Uhol: Real;
begin
  Uhol := 0;
  while Uhol <= 6.3 do
  begin
    X := 200 + 80 * Cos(Uhol);
    Y := 100 + 80 * Sin(Uhol);
    if Uhol = 0 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 0.1;
  end;
end;
```

Po spustení:



Už asi poznáme podmienený príkaz **if**, ktorý pri prvom prechode cyklu presúva pero do prvého bodu krivky. Všetky ďalšie prechody už kreslia čiary príkazom **LineTo**.

Táto najjednoduchšia verzia kreslenia tejto krivky je prakticky rovnaká ako nakreslenie kružnice (bez výplne) pomocou príkazu **Ellipse(120, 20, 280, 180)**. Lenže tento náš program má ďalšie veľmi zaujímavé výhody. Postupne ukážeme rôzne možnosti využitia.

Ako prvé si ukážeme ***spomalené kreslenie***. Po každom prechode cyklu program na veľmi krátky časový úsek pozastavíme (napr. 100 milisekúnd), zároveň počítaču nariadime, aby naozaj zobrazil obsah grafickej plochy. Inak sa obsah grafickej plochy aktualizuje až po ukončení programu, presnejšie povedané, až po ukončení procedúry, ktorá sa spustila zatlačením tlačidla, t.j. **Button1Click**. Preto vždy pri pozastavení príkazom **Sleep** musíme grafickú plochu prekresliť príkazom **Image1.Repaint**.

```
var
  X, Y, Uhol: Real;
begin
  Uhol := 0;
  while Uhol <= 6.3 do
  begin
    X := 200 + 80 * Cos(Uhol);
    Y := 100 + 80 * Sin(Uhol);
    if Uhol = 0 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 0.1;
    Sleep(100);
    Image1.Repaint;
  end;
end;
```

Príkaz **Sleep** pozastaví vykonávanie programu na zadaný počet milisekúnd. Napr. **Sleep(2000)** zastaví vykonávanie programu na 2 sekundy (2000 milisekúnd). Túto dvojicu príkazov môžeme vložiť aj do našich starších programov, aby sme mohli lepšie sledovať priebeh kreslenia.

Poznámka:
Bežiaci projekt môžeme ukončiť z prostredia Delphi a to tak, že stlačíme ***Ctrl/F2****.*

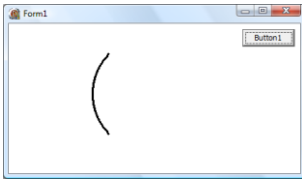
Pri používaní spomaľovania ale môže byť jeden malý problém. Napr. v našom programe cyklus beží 64-krát a teda pozdržanie 100 milisekúnd bude dokopy trvať 6.4 sekundy. Počas tohto času je ale program *nezastaviteľný*, a teda aplikácia sa počas tohto nedá jednoducho zavrieť. Ak dobre neodhadneme celkovú dĺžku trvania takéhoto spomaľovania, napr. do cyklu, ktorý má bežať milión krát, vložíme **Sleep(10)**, tento program bude bežať skoro 3 hodiny.

Prvá verzia programu na kreslenie kružnice pracovala s radiánmi. Teraz prácu s goniometrickými funkciami **Sin** a **Cos** prepíšeme na stupne, tak ako sme to robili už predtým.

```
procedure TForm1.Button1Click(Sender: TObject);
const
  U1 = 0;
  U2 = 360;
var
  X, Y, Uhol: Real;
begin
  Uhol := U1;
  while Uhol <= U2 do
  begin
    X := 200 + 80 * Cos(Uhol * Pi / 180);
    Y := 100 + 80 * Sin(Uhol * Pi / 180);
    if Uhol = U1 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 1;
  end;
end;
```

Polkružnica pre U1=0 a U2=180:

Štvrťkružnica pre U1=135 a U2=225:



Časť kružnice pre U1=120 a U2=380:

Čo si treba všimnúť na tomto programe: premenná **Uhol** je teraz zadaná v stupňoch a postupne nadobúda hodnoty od 0 to 360. Program nakreslí presne rovnakú kružnicu, ako predchádzajúca verzia s radiánmi, len predtým sa skladala zo 63 úsečiek a teraz až z 360-tich úsečiek. Konštanty **U1** a **U2** určujú interval stupňov, pre ktorý program nakreslí časť kružnice, napr.:

- pre U1=0 a U2=180, je to polkružnica
- pre U1=45 a U2=135, je to štvrťkružnica

Vyskúšajte zmeniť krok, s ktorým sa na konci cyklu mení premenná **Uhol**, napr.

```
Uhol := Uhol + 5;
```

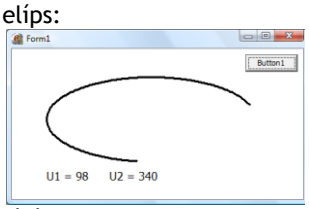
Podobným spôsobom vieme nakresliť aj elipsu, ktorej vzorec je:

$$x = x0 + A * \cos t$$
$$y = y0 + B * \sin t$$

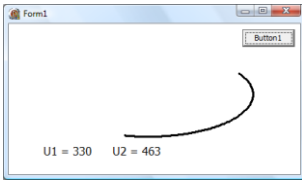
kde **A a B** sú polosi. Nasledujúci program kreslí časť elipsy, pričom uhly **U1** a **U2** sa generujú náhodne:

```
var
  X, Y: Real;
  Uhol, U1, U2: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 3;
  U1 := Random(360);
  U2 := U1 + Random(360) + 1;
  for Uhol := U1 to U2 do
  begin
    X := 200 + 150 * Cos(Uhol * Pi / 180);
    Y := 100 + 60 * Sin(Uhol * Pi / 180);
    if Uhol = U1 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
  end;
  Image1.Canvas.Font.Height := 20;
  Image1.Canvas.TextOut(50, 170, 'U1 = ' + IntToStr(U1) +
    '      U2 = ' + IntToStr(U2));
end;
```

Po niekoľkých zatlačeniach dostávame rôzne časti elípsy:



alebo



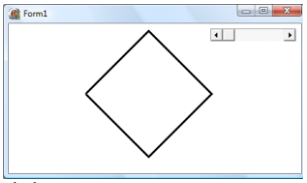
Všimnite si, že premennú **Uhol** mení for-cyklus.

Kružnicu sme zatiaľ kreslili ako napr. 360 krátkych úsečiek, lebo uhol sme menili po 1 stupni, Ak by sme ho menili po 5 stupňoch, stačilo by nám 72 úsečiek. V skutočnosti kružnicu kreslíme ako pravidelný 360-uholník, resp. 72-uholník.

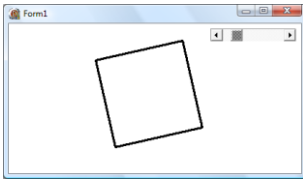
Ak budeme premennú **Uhol** zväčšovať o 90 stupňov (resp. **π/2** radiánov), nakreslí sa pravidelný štvoruholník, t.j. štvorec. Takto nakreslený štvorec ale nemusí mať strany rovnobežné s osami, ale s ľubovoľným uhlom. Nasledujúci program má namiesto tlačidla **Button1** posúvač **ScrollBar1** (s Min=0 a Max=100):

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
var
  X, Y, Uhol: Real;
  I: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 3;
  Uhol := ScrollBar1.Position * Pi / 90;
  for I := 1 to 5 do
  begin
    X := 200 + 90 * Cos(Uhol);
    Y := 100 + 90 * Sin(Uhol);
    if I = 1 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + Pi / 2;
  end;
end;
```

Pri rôznych posunutiach posúvača dostávame rôzne natočené štvorce:



alebo

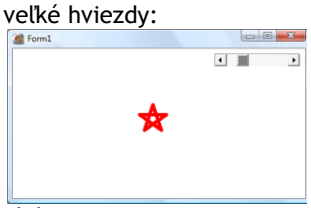


Všimnite si, že for-cyklus má pre štvorec 5 prechodov. Viete, prečo nestačia iba 4 prechody?

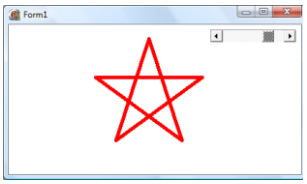
Podobne by sme vedeli nakresliť napr. aj pravidelný 5-uholník, vtedy by krok pre **Uhol** musel byť 72 stupňov. Zaujímavý výsledok dostávame pre krok 144 stupňov: nakreslí sa päťcípa hviezda. Nasledujúci program tiež využíva posúvač, ale na zmenu veľkosti hviezdy:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
var
  X, Y, Uhol: Real;
  I, Velkost: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 5;
  Image1.Canvas.Pen.Color := clRed;
  Velkost := ScrollBar1.Position;
  Uhol :=  -90;
  for I := 1 to 6 do
  begin
    X := 200 + Velkost * Cos(Uhol * Pi / 180);
    Y := 100 + Velkost * Sin(Uhol * Pi / 180);
    if I = 1 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 144;
  end;
end;
```

Pri rôznych posunutiach posúvača dostávame rôzne veľké hviezdy:



alebo



Uhly pre vrcholy sme teraz počítali nie v radiánoch, ale v stupňoch.

V matematike je známych veľa rôznych, tzv. cyklických kriviek. Tieto vychádzajú z rovnakej idey, ako sme kreslili kružnice a N-uholníky, len ich parametrická funkcia je trochu zložitejšia. Známe sú rôzne epicykloidy, hypocykloidy, astroidy, ruže a pod. Všetky, okrem svojej parametrickej funkcie, majú aj niekoľko parametrov, ktorých rôzne hodnoty niekedy úplne zmenia tvar krivky.

Vyskúšajme *epicykloidu*, ktorá je definovaná parametrickou funkciou takto:

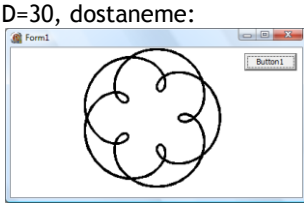
$$x = (a+b)\cos(t) - d\cos(t(a+b)/b)$$
$$y = (a+b)\sin(t) - d\sin(t(a+b)/b)$$

Parametrami sú A, B a D, premenná T je uhol. Prepíšme to do programu:

```
const
  A = 60;
  B = 20;
  D = 20;
var
  X, Y, Uhol: Real;
  I: Integer;
begin
  Image1.Canvas.Pen.Width := 3;
  Uhol := 0;
  for I := 0 to 500 do
  begin
    X := 200 + (A + B) * Cos(Uhol) - D * Cos(Uhol * (A + B) / B);
    Y := 100 + (A + B) * Sin(Uhol) - D * Sin(Uhol * (A + B) / B);
    if I = 0 then
      Image1.Canvas.MoveTo(Round(X), Round(Y))
    else
      Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 0.1;
  end;
end;
```

Po spustení:

Pre parametre A=50, B=20, D=30, dostaneme:



Na záver nakreslíme špirálu. Existuje veľa rôznych parametrických funkcií, my tu použijeme takýto najjednoduchší zápis:

$$x = a\ t\ \cos\ t$$
$$y = a\ t\ \sin\ t$$

Program využíva posúvač (s Min=0 a Max=1000), ktorý určuje počet prechodov cyklu:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
const
  A = 2;
var
  X, Y, Uhol: Real;
  I: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 3;
  Uhol := 0;
  Uhol := 0;
  Image1.Canvas.MoveTo(200, 100);
  for I := 1 to ScrollBar1.Position do
  begin
    X := 200 + A * Uhol * Cos(Uhol);
    Y := 100 + A * Uhol * Sin(Uhol);
    Image1.Canvas.LineTo(Round(X), Round(Y));
    Uhol := Uhol + 0.1;
  end;
end;
```

Podľa posunutia posúvača dostávame časť špirály:

alebo

Pri každom pohybe posúvača sa zmaže grafická plocha a špirála sa nakreslí znovu od začiatku. Všimnite si, že prvé **MoveTo** sme urobili ešte pred cyklom.

Úlohy na precvičenie

Zadanie 1	podobne, ako sa kreslil štvorec, ktorý sa otáčal pomocou posúvača, nakreslite pravidelný trojuholník (s krokom pre **Uhol** 120)
Zadanie 2	každé zatlačenie tlačidla nakreslí na náhodnej pozícii N-uholník s náhodnou veľkosťou a farbou, kde N je náhodné číslo od 3 do 8
Zadanie 3	nakresliť **N** rovnakých malých útvarov (napr. farebné štvorčeky alebo krúžky), ktoré sú na obvode kružnice s polomerom **R** - samotnú kružnicu nekreslite

Zadanie 4	hypocykloida je definovaná veľmi podobne ako epicykloida takto:
	`x = (a-b)cos(t) + d cos(t(a-b)/b)`
	`y = (a-b)sin(t) - d sin(t(a-b)/b)`
	`otestujte pre rôzne hodnoty parametrov A, B a D`

Zadanie 5	nakreslite astroidu, ktorá je definovaná takto: $x = a \cos^3(t)$ $y = a \sin^3(t)$

Zadanie 6
(náročnejšie)
	v matematike môže byť nejaká funkcia **F(t)** zadaná aj v polárnych súradniciach, potom ju môžeme nakresliť prepísaním na parametrickú funkciu takto
	`x = F(t) cos t`
	`y = F(t) sin t`
	krivky s menom ruža majú v polárnych súradniciach vyjadrenie:
	`F(t) = A cos(t * N/K)`
	nakreslite pre rôzne parametre A, N, K (napr. N=6, K=1)

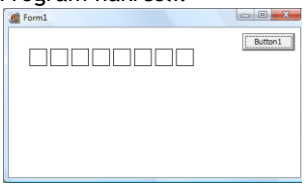
2. tematická jednotka - vnorené cykly

1. Vnorené cykly

Nasledujúci program kreslí rad štvorčekov:

```
var
  I: Integer;
begin
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30, 30*I+25, 55);
end;
```

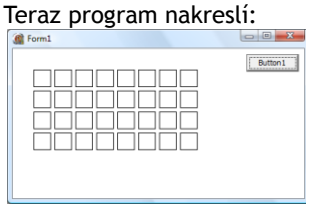
Program nakreslí:



Ak by sme takýchto radov potrebovali nakresliť viac, napr. 4 pod seba, skopírujeme cyklus ešte 2-krát a opravíme Y súradnice:

```
var
  I: Integer;
begin
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30, 30*I+25, 55);
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 60, 30*I+25, 85);
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 90, 30*I+25, 115);
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 120, 30*I+25, 145);
end;
```

Teraz program nakreslí:



Vidíme, že kreslenie každého ďalšieho riadku sa líši len v **Y** súradniciach: v **J**-tom riadku je druhý parameter volania **Rectangle** vlastne **30*J** a štvrtý parameter **30*J+25**. Môžeme to zapísať, napr. takto:

```
var
  I, J: Integer;
begin
  J := 1;
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30 * J, 30*I+25, 30 * J + 25);
  J := 2;
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30 * J, 30*I+25, 30 * J + 25);
  J := 3;
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30 * J, 30*I+25, 30 * J + 25);
  J := 4;
  for I := 1 to 8 do
    Image1.Canvas.Rectangle(30*I, 30 * J, 30*I+25, 30 * J + 25);
end;
```

Úmyselne sme to zapísali takto, aby bolo lepšie vidieť, že sa štyrikrát opakuje volanie for-cyklu, ale zakaždým s inou hodnotou **J**. My už vieme, že keď potrebujeme niečo opakovať viackrát, výhodne na to použijeme for-cyklus, v ktorom do jeho tela zapíšeme opakujúce sa príkazy:

```
var
  I, J: Integer;
begin
  for J := 1 to 4 do
    for I := 1 to 8 do
      Image1.Canvas.Rectangle(30*I, 30 * J, 30*I+25, 30 * J + 25);
end;
```

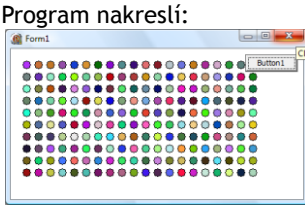
Z toho, ako sme tento program konštruovali, by nám mohli byť jasný mechanizmus, ako sa takýto ***vnorený cyklus*** bude vykonávať:

- **J** sa nastaví na hodnotu 1 a vykoná sa telo cyklu, t.j. vnútorný for-cyklus s počítadlom **I**: prebehne 8-krát,
- **J** sa zvýši o 1 a opäť sa vykoná vnútorný cyklus, teda nakreslí sa 8 štvorcov,
- toto sa opakuje dovtedy, kým nie je **J** väčšie ako 4.

Je dôležité si zapamätať, že pre dva vnorené for-cykly musí mať každý z nich svoje vlastné počítadlo. Nesmieme použiť tú istú premennú pre dva navzájom vnorené cykly.

Pozmeníme tento program tak, že nakreslí 10 radov po 20 malých kruhov, pričom každý bude zafarbený nejakou náhodnou farbou:

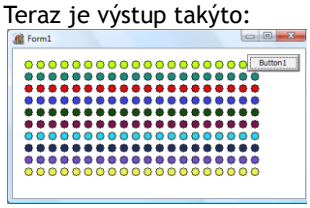
Program nakreslí:



```
var
  I, J: Integer;
begin
  for J := 1 to 10 do
    for I := 1 to 20 do
    begin
      Image1.Canvas.Brush.Color := Random(256 * 256 *256);
      Image1.Canvas.Ellipse(17 * I, 17 * J, 17 * I + 12, 17 * J + 12);
    end;
end;
```

Ak by mal byť ale každý celý rad zafarbený rovnakou náhodnou farbou, musíme príkazy organizovať inak:

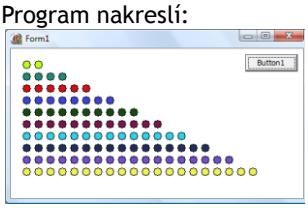
Teraz je výstup takýto:



```
var
  I, J: Integer;
begin
  for J := 1 to 10 do
  begin
    Image1.Canvas.Brush.Color := Random(256 * 256 *256);
    for I := 1 to 20 do
      Image1.Canvas.Ellipse(17 * I, 17 * J, 17 * I + 12, 17 * J + 12);
  end;
end;
```

Ukážeme ďalšie možnosti vnorených cyklov. Vnorený for-cyklus nebude stále bežať 20-krát, ale tento počet bude závisieť od čísla riadku, t.j. od premennej **J**. Nech má prvý riadok 2 kruhy, druhý riadok 4 kruhy, atď. **J**-ty riadok bude mať **2*J** kruhov:

Program nakreslí:



```
var
  I, J: Integer;
begin
  for J := 1 to 10 do
  begin
    Image1.Canvas.Brush.Color := Random(256 * 256 *256);
    for I := 1 to 2 * J do
      Image1.Canvas.Ellipse(17 * I, 17 * J, 17 * I + 12, 17 * J + 12);
  end;
end;
```

Vedeli by ste čo najrýchlejšie vypočítať, koľko malých kruhov sa nakreslilo v týchto desiatich radoch? Vedeli by ste vypočítať, koľko by bolo týchto kruhov, keby sme ich kreslili do **N** radov (prvý rad 2, druhý 4, ... posledný **N**-tý rad **2N** kruhov)?

Tieto dva for-cykly teraz prepíšeme na while-cykly. Chceme zaplniť celú grafickú plochu farebnými krúžkami, ale do každého riadku dáme maximálne toľko, koľko sa ich tam zmestí. Podobne chceme čo najviac riadkov. Samozrejme, že by sa to dalo urobiť aj for-cyklami, ale tu si trénujeme vnorené while-cykly:

```
var
  X, Y: Integer;
begin
  Y := 5;
  while Y + 12 < Image1.Height do
  begin
    X := 5;
    while X + 12 < Image1.Width do
    begin
      Image1.Canvas.Brush.Color :=
        RGB(X * 256 div Image1.Width, 0, Y * 256 div Image1.Height);
      Image1.Canvas.Ellipse(X, Y, X + 12, Y +12);
      X := X + 17;
    end;
    Y := Y + 17;
  end;
end;
```

Po spustení dostávame:

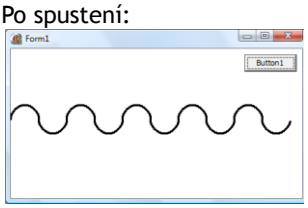
Všimnite si, ako sme prepočítavali zafarbenie každého kruhu podľa jeho pozície.

V ďalšom príklade využijeme kreslenie polkružnice z predchádzajúcej časti. Polkružnicu sme kreslili tak, že sme pri kreslení kružnice začali s uhlom na nejakom **U1** a skončili na uhle **U1+180** stupňov.

Pomocou takýchto polkružníc nakreslíme vlnky, v ktorých budeme tieto polkružnice prikladať vedľa seba - tam, kde jedna skočí, tam druhá nadväzuje. Budeme ich klásť do jedného radu, kým sa ešte zmestia do grafickej plochy. Ak si uvedomíme, že kreslíme dva typy polkružníc: jedna je pre uhly od 180 do 360 a druhá je pre uhly od 180 do 0, ale s opačným krokom, môžeme zapísať program:

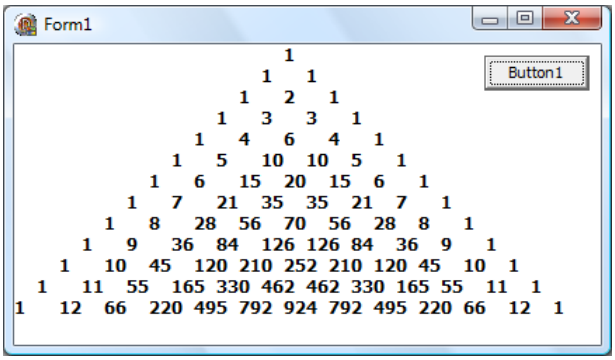
```
const
  R = 20;
var
  X, Y: Real;
  I, X0, Y0, Uhol, Krok: Integer;
begin
  X0 := R;              // X súradnica stredu polkruhu
  Y0 := 100;            // Y súradnica stredu polkruhu
  Uhol := 180;
  Krok := 10;           // či sa bude uhol zväčšovať/zmenšovať
  Image1.Canvas.Pen.Width := 3;
  Image1.Canvas.MoveTo(X0 - R, Y0);
  while X0 + R < Image1.Width do
  begin
    for I := 1 to 18 do
    begin
      X := X0 + R * Cos(Uhol * Pi / 180);
      Y := Y0 + R * Sin(Uhol * Pi / 180);
      Image1.Canvas.LineTo(Round(X), Round(Y));
      Uhol := Uhol + Krok;
      Sleep(30);
      Image1.Repaint;
    end;
    X0 := X0 + 2 * R; // posuň X súradnicu stredu kruhu
    Uhol := Uhol + 180;
    Krok := -Krok;    // otoč Krok pre Uhol
  end;
end;
```

Po spustení:



Na tomto príklade vidíme, ako zariadiť, aby sa pri jednom prechode cyklom hodnota premennej **Uhol** stále zväčšovala o **Krok** a pri ďalšom prechode zase stále zmenšovala (premenná **Krok** má raz hodnotu **10** a druhýkrát hodnotu **-10**). Do programu sme pridali aj spomaľovanie vykresľovania, aby sme videli jeho priebeh (dvojica príkazov **Sleep** a **Image1.Repaint**).

Matematici dobre poznajú tzv. ***Pascalov trojuholník*** (geometrické usporiadanie binomických koeficientov do tvaru trojuholníka). Ukážeme program, ktorý vykreslí niekoľko jeho riadkov:



Riadky budeme postupne kresliť od najvrchnejšieho (dáme mu poradové číslo 0) až po najspodnejší. Každý riadok začína číslom 1 a každé ďalšie v riadku vzniká tak, že ho niečím vynásobíme a niečím vydelíme. Matematici vedia ľahko odvodiť, že **K**-te číslo v **N**-tom riadku dostaneme z predchádzajúcej hodnoty v premennej **Cislo** ako

```
Cislo := Cislo * (N + 1 - K) div K
```

kde **K** ide postupne od 1 do **N** (v **N**-tom riadku je okrem úvodnej 0 ešte ďalších **N** čísel). Aby sa trojuholník pekne vycentroval, v každom riadku začíname vypisovať čísla na **X**-ovej súradnici, ktorá závisí od čísla riadku. Zapíšme program:

```
const
  M = 12;           // počet riadkov trojuholníka
var
  Cislo, N, K, X: Integer;
begin
  Image1.Canvas.Font.Height := 15;
  Image1.Canvas.Font.Style := [fsBold];
  for N := 0 to M do
  begin
    Cislo := 1;
    X := 16 * (M - N);
    Image1.Canvas.TextOut(X, N * 15, IntToStr(Cislo));
    X := X + 32;
    for K := 1 to N do
    begin
      Cislo := Cislo * (N + 1 - K) div K;
      Image1.Canvas.TextOut(X, N * 15, IntToStr(Cislo));
      X := X + 32;
    end;
  end;
end;
```

Poznámka:
Štýl písma môžeme zmeniť priradením do nastavenia **Image1.Canvas.Font.Style** napr. jednu z hodnôt:
[fsBold] pre tučné písmo
[fsItalic] pre kurzívu
[] normálne písmo
Možností je viac a môžete si ich pozrieť v Helpe.

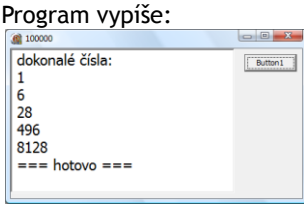
Okrem nastavenia tučného písma (**Font.Style := [fsBold]**) by sme v tomto programe mali poznať už všetko: Pred vnútorným cyklom sa najprv nastaví X-ová súradnica výpisu a vypíše sa úvodná 1. Vnútorný cyklus postupne prepočítava nové vypisované hodnoty a zakaždým posúva X-ovú súradnicu vpravo. Vonkajší cyklus zabezpečuje nastavenie N-tého riadka, pričom Y-ová súradnica sa tu počíta ako **N * 15**.

Pripomeňme si, čo sú to dokonalé čísla: sú to také čísla, ktoré sa rovnajú súčtu všetkých svojich deliteľov, samozrejme, že okrem samého seba. Napr. číslo 6 má troch deliteľov: 1, 2, 3 a ich súčet je samotné číslo 6.

Ďalší program nájde a vypíše všetky dokonalé čísla do 100000. Tentoraz nevypisujeme do grafickej ale do textovej plochy.

```
var
  Cislo, Sucet, Delitel: Integer;
begin
  Memo1.Clear;
  Memo1.Lines.Append('dokonalé čísla:');
  for Cislo := 1 to 100000 do
  begin
    Sucet := 1;
    for Delitel := 2 to Cislo div 2 do
      if Cislo mod Delitel = 0 then
        Sucet := Sucet + Delitel;
    if Cislo = Sucet then
      Memo1.Lines.Append(IntToStr(Cislo));
    Caption := IntToStr(Cislo);
  end;
  Memo1.Lines.Append('=== hotovo ===');
end;
```

Program vypíše:

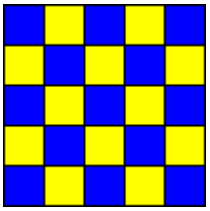


Ak tento program spustíme, veľmi rýchlo sa objavia prvé štyri dokonalé čísla: 6, 28, 496, 8128 a potom sa výpočet "zasekne" – veľmi dlho nedostávame žiadne informácie o tom, čo program robí. Preto sme na záver vonkajšieho cyklu pridali kontrolný výpis do titulného riadka aplikácie: **Caption := IntToStr(Cislo)**, v ktorom sa dozvedáme, ktoré číslo sa práve skontrolovalo na dokonalosť. Vďaka tomu vidíme, že program nezaspal, ale sa blíži k hornej hranici cyklu 100000.

V literatúre sa dá nájsť informácia, že ďalšie najbližšie dokonalé číslo je väčšie ako 33 miliónov. Tomuto nášmu programu by trvalo asi príliš dlho nájsť toto ďalšie číslo.

Úlohy na precvičenie

Zadanie 1	nakreslite pyramídu z obdĺžnikov veľkosti AxB, ktorá je vysoká N radov: prvý rad obsahuje jeden obdĺžnik, druhý dva, ... posledný N-tý obsahuje N obdĺžnikov; šírka obdĺžnika je dvakrát jeho výška; použite konštanty napr.
	`const`
	`  B = 20;`
	`  A = 2 * B;`
	`  N = 10;`
	pyramídu vycentrujte

Zadanie 2	pomocou stále sa zväčšujúcich sa polkružníc nakreslite špirálu – použite podobný princíp, ako sa kreslili vlnky
Zadanie 3	vypíšte do grafickej plochy malú násobilku, t.j. tabuľku 10x10 čísel, v ktorej v **I**-tom riadku a v **J**-tom stĺpci je súčin čísel **I*J**
Zadanie 4	nakreslite šachovnicu **N**x**N** štvorčekov striedajúcich sa dvoch farieb a to tak, že najprv nakreslíte veľký podkladový štvorec z prvej farby, potom sa kreslia iba štvorčeky zafarbené druhou farbou, napr. 
Zadanie 5	vypíšte do textovej plochy všetky trojice celých čísel do 100, ktoré sú dĺžkami strán pravouhlého trojuholníka ($a^2+b^2=c^2$)
Zadanie 6	pre zadaný interval <A, B> vypíšte všetky prvočísla z tohto intervalu; vypisujte do grafickej plochy po viac čísel do jedného riadka
Zadanie 7	vypíšte všetky prvočíselné dvojičky, ktoré nie sú väčšie ako 1000; dvojičkami sú také dve prvočísla, ktorých rozdiel je 2

2. Nekonečný cyklus

Skoro každý programátor sa niekedy stretol s nekonečným cyklom, ktorý mu dosť znepríjemnil život. Nekonečné cykly niekedy vznikajú malým, ľahko prehliadnuteľným preklepom:

```
var
  I: Integer;
begin
  I := 100;
  while I > 0 do;
    I := I - 7;
  Memo1.LinesAppend(IntToStr(I));
end;
```

V tomto programe bodkočiarka ; za slovom **do** určuje, že telo cyklu je **prázdny príkaz** a tento sa bude opakovať, kým bude **I** väčšie ako 0. Lenže znižovanie **I** o 7 je až za cyklom a nevykoná sa ani raz.

Podobný efekt vzniká pri takomto programe:

```
var
  I: Integer;
begin
  I := 100;
  while I > 0 do
    Memo1.LinesAppend(IntToStr(I));
    I := I - 7;
  Memo1.LinesAppend('hotovo');
end;
```

Program zrejme chcel vypisovať klesajúcu postupnosť kladných čísel 100, 93, 86, 79, ... lenže v cykle sa vykonáva jediný príkaz, ktorý vypisuje hodnotu **I** a samotné znižovanie premennej **I** o 7 sa robí až za cyklom. Keďže sa hodnota premennej **I** v cykle nemení, cyklus nebude nikdy končiť, teda opäť nekonečný cyklus. Zrejme tu chcel programátor dať oba príkazy do cyklu a teda ***iba*** zabudol ich uzavrieť do **begin** - **end** príkazových zátvoriek. Teraz pozrite túto modifikáciu:

```
var
  I: Integer;
begin
  I := 100;
  while I <> 0 do
  begin
    Memo1.LinesAppend(IntToStr(I));
    I := I - 7;
  end;
  Memo1.LinesAppend('hotovo');
end;
```

Teraz už chyba nie je až taká zrejmá, lebo tu sme telo cyklu uzavreli do zloženého príkazu. Ale programátor tu mierne vylepšil podmienku cyklu, lebo teraz bude cyklus bežať, kým hodnota premennej **I** bude rôzna od 0. Keďže sa **I** zmenšuje, je tu nádej, že niekedy to k 0 príde a cyklus skončí. Lenže v tomto konkrétnom prípade premenná **I** preskočí hodnotu 0 a ďalej pokračuje v znižovaní svojej hodnoty v záporných číslach.

Podobných problémových podmienok by sme asi vedeli zostaviť veľké množstvo, prípadne naši žiaci nás vyskúšajú so svojimi neočakávanými chybami.

Ďalšiu kategóriu tvoria zaokrúhľovacie chyby reálnej aritmetiky. Už sme predtým videli príklad, ktorý sa podobal na:

```
var
  R: Real;
begin
  R := 0;
  while R <> 1 do
  begin
    R := R + 0.1;
    Memo1.Lines.Append(FloatToStr(R));
  end;
  Memo1.Lines.Append('koniec');
end;
```

Tu sa dá naozaj oprávnene očakávať, že cyklus prejde 10-krát a potom určite skončí. My ale už vieme svoje o nepresnosti reálnej aritmetiky a o tom, že je veľmi odvážne testovať na rovnosť dve reálne čísla, ktoré boli vypočítané pomocou aritmetických výrazov. Tým skôr takýto test použiť v podmienke while-cyklu.

Čo môžeme teda robiť v situáciách, keď sa program zacyklí. Takýto program sa nedá ukončiť bežným uzavretím aplikácie (tlačidlo **Close** v pravom hornom rohu okna). Najjednoduchšie bude sa prepnúť do prostredia Delphi a tu buď zatlačiť tlačidlo <***Program Reset***>, alebo stlačiť klávesovú skratku **Ctrl+F2**.

Nájsť takéto chyby najmä vo väčších programoch si vyžaduje dosť programátorských skúseností, ale môžu nám pomôcť režimy trasovania programov (pozrite si voľby v menu **Run**), prípadne vloženie kontrolných výpisov do podozrivých častí kódu.

Predchádzajúci program, ktorý vypisoval reálne čísla do textovej plochy Memo1, asi nebude potrebovať kontrolné výpisy, lebo priamo textová plocha nám ukazuje, čo sa deje. Horšie je to s grafickou plochou, ktorej obsah vidíme, až keď algoritmus skončí. Napr. ho pozmeníme takto:

```
var
  R: Real;
begin
  R := 0;
  while R <> 1 do
  begin
    R := R + 0.1;
    Image1.Canvas.TextOut(Random(300), Random(200), FloatToStr(R));
    Image1.Repaint;
  end;
end;
```

Ak by tam nebol príkaz **Image1.Repaint**, nevidíme žiadnu aktivitu a preto sa ťažšie odhaľujú dôvody zamrznutia programu. Ak tam ale pridáme **Image1.Repaint**, prípadne aj s nejakým **Sleep(100)**, veľmi rýchlo chybu odhalíme.

Inokedy nám môže pomôcť nejaký výpis aj do titulného riadku aplikácie, napr.

```
    Caption := FloatToStr(RCislo);
```

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- reálna aritmetika, typ **Real**
- spôsob vykresľovania priebehu grafov funkcií
- kreslenie kružnice a jej častí pomocou parametrického zápisu funkcie

Okrem toho boli uvedené tieto dôležité koncepty:

- prevody do číselných sústav
- riadenie behu programu pomocou komponentu posúvač (**TScrollBar**)
- vypisovanie do textovej plochy (komponent **TMemo**)
- spomaľovanie behu programu

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Pre zadané N program nájde všetky prvočísla tvaru 2^K-1, ktoré sú menšie ako N.

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/

Poznámky

Poznámky

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 2

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti doc. RNDr. Gabriela Andrejková, CSc.
Mgr. Ján Guniš

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 978-80-89225-57-6