

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Programovanie 2

Predmet: Programovanie

Línia: Vlastný odborový kontext informatiky a informatickej výchovy



Programovanie 2

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

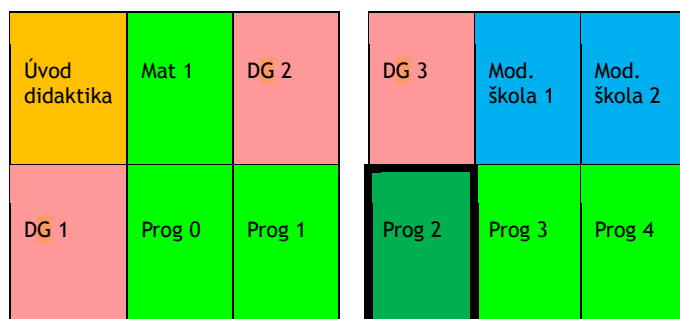


Rukopis odovzdaný:

14. august 2009

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 ne 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

Použitie premenných v programovacom jazyku, priradovací príkaz a jednoduchý cyklus s pevným počtom opakovaní.

Obsah

Programovanie 2.....	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 - Premenné.....	4
1. Pomenované konštanty.....	4
2. Jednoduchá celočíselná premenná a priradovací príkaz.....	6
3. Aritmetický výraz.....	9
4. Generátor náhodných čísel	14
5. Zadávanie vstupu.....	16
6. Farebný model RGB	20
2. tematická jednotka - Cyklus s pevným počtom opakovaní	23
1. Cyklus s pevným počtom opakovaní.....	23
2. Viac príkazov v tele cyklu.....	25
3. Využitie počítadla v tele cyklu.....	27
4. Pomocná premenná pre cyklus.....	30
5. Prevrátený cyklus.....	33
6. Cifry celého čísla	34
Zoznam rezervovaných slov	38
Čo sme sa naučili v tomto module.....	39
Preverenie výstupných vedomostí	39
Literatúra a použité zdroje	39

Úvod

Modul sa skladá z dvoch tematických jednotiek každá približne rozsahu 4 vyučovacích hodín:

1. Premenné
2. Cyklus s pevným počtom opakovaní

Cieľ modulu

Prvá tematická jednotka **Premenné** pokrýva tieto témy:

1. pomenované konštanty
2. jednoduchá celočíselná premenná a priradovací príkaz
3. aritmetický výraz
4. generátor náhodných čísel
5. zadávanie vstupu
6. farebný model RGB

Druhá tematická jednotka **Cyklus s pevným počtom opakovaní** pokrýva tieto témy:

1. cyklus s pevným počtom opakovaní
2. viac príkazov v tele cyklu
3. využitie počítadla v tele cyklu
4. pomocná premenná pre cyklus
5. prevrátený cyklus
6. cifry celého čísla

Vstupné vedomosti

Požadované prerekvizity

Modul Programovanie 1 - úvod do programátorského prostredia a grafické príkazy.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- ovláda základné grafické príkazy,
- dokáže opraviť program, ak je chyba v poradí príkazov,
- vie zadávať postupnosť grafických príkazov na nakreslenie zadaného obrázku,
- vie čítať jednoduché postupnosti príkazov a analyzovať ich,
- dokáže vytvoriť jednoduchý projekt využívajúc tlačidlá, udalosti a grafické príkazy,
- rozumie pojmom príkaz, postupnosť príkazov, procedúra, parameter, udalosť.

Preverenie vstupných vedomostí

Vytvorenie jednoduchej aplikácie s aspoň dvoma tlačidlami, ktorá dokáže nakresliť rôzne obrázky.

1. tematická jednotka - Premenné

1. Pomenované konštanty

Jazyk Pascal veľmi prísne rozlišuje typy hodnôt, s ktorými sa pracuje a na každý omyl nás upozorní chybovou správou, napr.

Čo už doteraz vieme:

Kresliť príkazmi do grafickej plochy:

- Image1.Canvas.Rectangle
- Image1.Canvas.Ellipse
- Image1.Canvas.MoveTo
- Image1.Canvas.LineTo
- Image1.Canvas.TextOut

Nastavovať rôzne vlastnosti:

- Image1.Canvas.Pen
- Image1.Canvas.Brush
- Image1.Canvas.Font
- Button1.Caption
- Button1.Top
- Button1.Left
- Label1.Caption

Príkazy majú rôzne počty aj typy parametrov:

- Image1.Canvas.Ellipse(100,150,200,200)
- Image1.Canvas.MoveTo(10, 10)
- Image1.Canvas.TextOut(100, 50, 'Delphi')

Vlastnosti meníme pomocou priradenia.

Priradované hodnoty musia byť presne zadaného typu

- číselné hodnoty (hrúbka, súradnica, veľkosť):

- Image1.Canvas.Pen.Width := 5
- Button1.Top := 0
- Image1.Canvas.Font.Height := 36

- textové hodnoty:

- Button1.Caption := 'Tlačidlo'
- Image1.Canvas.Font.Name := 'Arial'

- mená farieb:

- Image1.Canvas.Brush.Color := clRed
- Image1.Canvas.Font.Color := clBlue

Incompatible types: 'Integer' and 'string'

Vyskúšajte

```
Image1.Canvas.Pen.Color := 'biela';  
Image1.Canvas.TextOut(100, 100, 1968);
```

v oboch prípadoch Delphi vyhlásia chybu o nekompatibilitate dvoch typov.

Zatiaľ sme v našich programoch používali len konštanty rôznych typov:

- celočíselné: 0, 1, 100, 1234, -54, -99, ...
- texty: 'Delphi', 'programujeme', 'ako sa máš?', ...
- farby: clRed, clBlue, clBlack, ...

Pre čitateľnejší zápis programov môžeme niektorým konštantám dať vlastné mená. Zadefinujeme **identifikátor** konštanty a určíme mu jeho hodnotu. Zapisujeme to takto

```
const  
  Hrube = 10;  
  Tenke = 1;  
  Modra = clBlue;  
  Velkost = 120;
```

Tieto definície začínajú slovom **const**, za ktorým nasledujú dvojice: meno konštanty a jej hodnota. Medzi meno a hodnotu zapisujeme znak = a za definíciou píšeme znak bodkočiarka ; rovnako ako sme to robili za príkazmi. Tento príklad ukazuje definíciu pomenovaných 4 konštant a ďalej ich môžeme používať v našich programoch, napr. takto

```
Image1.Canvas.Pen.Width := Hrube;  
Image1.Canvas.Brush.Color := Modra;  
Image1.Canvas.Rectangle(0, 0, Velkost, Velkost);
```

Musíme sa ešte naučiť, že konštanty pomenovávame tzv. **identifikátormi** a tieto definície konštant musia byť v programe umiestnené na špeciálnom mieste. Teda, najprv sa dozvieme niečo o identifikátoroch.

Identifikátor v jazyku Pascal slúži na pomenovanie rôznych prvkov v programoch a má tieto pravidlá:

- skladá sa z písmen (len 26 písmen anglickej abecedy bez diakritiky), číslíc a znaku _ podtržník
- nesmie začínať číslicom
- identifikátor sa musí líšiť od rezervovaných slov v pascale takých ako **begin**, **end**, **procedure** (všetky sú vymenované na konci materiálu)

Je dôležité, aby pomenovanie konštanty čo najlepšie vyjadrovalo to, načo ju chceme používať. Pascal nerozlišuje malé a veľké písmená, preto všetky tieto mená popisujú ten istý identifikátor:

```
velkost, velKost, VELKOST, VeLKoST
```

V tomto materiáli sa budeme snažiť všetky identifikátory zapisovať s prvým veľkým písmenom, prípadne, ak je zložený z viacerých slov, tak každé slovo opäť s veľkým

písmenom. Uvedomte si, že identifikátor nesmie obsahovať medzery. Tieto pravidlá sa vzťahujú nielen na identifikátory konštant, ale neskôr uvidíme aj pomenovávanie premenných, procedúr a aj našich vlastných typov.

Deklarovanie konštant zapisujeme pred telo procedúry medzi riadky **procedure** a **begin**. Ukážeme to na príklade, keď na pozície X a Y vypíšeme nejaký text aj s jeho šedým tieňom:

```
procedure TForm1.Button1Click(Sender: TObject);
const
  Velkost = 30;
  NejakyText = 'Konštanty v programe';
  FarbaTextu = clBlue;
  X = 30;
  Y = 100;
begin
  Image1.Canvas.Font.Height := Velkost;
  Image1.Canvas.Font.Color := clLtGray;
  Image1.Canvas.TextOut(X+3, Y+6, NejakyText);
  Image1.Canvas.Font.Color := FarbaTextu;
  Image1.Canvas.TextOut(X, Y, NejakyText);
end;
```

Všimnite si, že pri určovaní súradníc tieňa, sme ku konštantám X a Y pripočítali malé posunutie vpravo 3 a dole 6. Každý číselný parameter nemusí byť len samotná konštantka, ale aj aritmetický výraz. Podrobné pravidlá aritmetických výrazov uvedieme neskôr.

Ak máme dve **ButtonXClick** procedúry a každá má svoje konštanty, tak tieto ich definície sú úplne nezávislé: ich platnosť začína deklaráciou a končí na príslušnom riadku **end**. Napr. dve procedúry kreslia štvorec: jedna pomocou príkazu **Rectangle** a druhá pomocou **MoveTo** a **LineTo** (kde X, Y je stred štvorca):

```
procedure TForm1.Button1Click(Sender: TObject);
const
  X = 150;
  Y = 100;
  A = 40;
begin
  Image1.Canvas.Rectangle(X - A, Y - A, X + A, Y + A);
end;

procedure TForm1.Button2Click(Sender: TObject);
const
  X = 200;
  Y = 200;
  A = 40;
begin
  Image1.Canvas.MoveTo(X - A, Y - A);
  Image1.Canvas.LineTo(X + A, Y - A);
  Image1.Canvas.LineTo(X + A, Y + A);
  Image1.Canvas.LineTo(X - A, Y + A);
  Image1.Canvas.LineTo(X - A, Y - A);
end;
```

Pravdepodobne, takto definované programy budú čitateľnejšie, ale hlavne ľahšie meniteľné: ak chceme program opraviť tak, aby štvorce nakreslil na iných pozíciách, alebo s inou veľkosťou, stačí zmeniť riadok v deklaráciách konštant.

Úlohy na precvičenie

Zadanie 1

Nakreslite štvorec, ktorého pravý dolný roh má súradnice X, Y a veľkosť strany je A.

Správne zapísané identifikátory:

XovaSuradnica,
Maximum,
Text1,
DlhyText,
VelkostTextu,
FarbaPodkladu

Nesprávne zapísané identifikátory:

Prvy Text,
2Text,
Polomer.Kruhu,
Velkost

Zadanie 2	Nakreslite kružnicu so stredom v X, Y a s polomerom R .
Zadanie 3	Nakreslite dve dotýkajúce sa kružnice s polomerami R_1 a R_2 , pričom stred prvej je v X, Y a druhá má rovnakú Y súradnicu.
Zadanie 4	Nakreslite pravouhlý trojuholník, ktorého vrchol s pravým uhlom je v X, Y a odvesny majú dĺžky D_1 a D_2 .
Zadanie 5	Nakreslite rovnoramenný trojuholník, ak je známa dĺžka spodnej strany $Dlžka$, výška $Vyska$ a súradnice jedného vrcholu X, Y (pozor, bude treba deliť dvoma, výpočet sa bude líšiť od voľby vrcholu).
Zadanie 6	Nakreslite štvorec, u ktorého poznáme súradnice X_1, Y_1 a X_2, Y_2 spodných dvoch vrcholov - predpokladajte, že $Y_1 = Y_2$.

2. Jednoduchá celočíselná premenná a priradovací príkaz

Hoci sú konštanty v programoch veľmi užitočná vec, ich nevýhodou je to, že počas fungovania programu nemôžu zmeniť svoju hodnotu. Veľmi často sa nám bude hodiť to, ak jeden identifikátor bude v rôznych častiach programu reprezentovať inú hodnotu. Teda, že túto hodnotu budeme môcť počas behu programu meniť. Na to ale už využijeme nie pomenované konštanty, ale premenné.

Premennou nazývame pomenované pamäťové miesto (pamäťové miesto označené nejakým identifikátorom), v ktorom sa uchováva nejaká informácia. Každé takéto pamäťové miesto môže mať uloženú len jednu hodnotu a to len nejakého konkrétneho (vopred určeného) typu. Na pomenovávanie premenných platia tie isté pravidlá ako pri konštantách: musia začínať písmenom (zväčša veľkým), za ktorým môžu nasledovať ďalšie písmená, číslice, resp. podtržník.

Povedzme, ako premenné v programoch fungujú:

1. keď sa rozhodneme v nejakej procedúre používať premenné, tak tieto **zadeklarujeme** ešte pred prvým **begin** procedúry - oblasti medzi **procedure** a **begin** hovoríme deklaračná oblasť a sem zapisujeme definície konštant ale aj deklarácie premenných;
2. pri použití identifikátorov konštant aj premenných musíme dávať pozor, aby sa nezhodovali s nejakým rezervovaným slovom jazyka pascal, ale tiež aby tieto identifikátory boli rôzne navzájom;
3. pri deklarácii premenných ku každému identifikátoru definujeme **typ** premennej, t.j. typ hodnôt, ktoré táto premenná môže nadobúdať - pripomeňme si, že sme zatiaľ videli typy *celé čísla*, *farby* a *texty* - pre jednoduchosť sa v tomto module budeme zaoberať iba celočíselným typom a až v nasledujúcich moduloch budeme postupne priberáť ďalšie typy (logický, znakový, reťazcový, desatinné čísla, polia a súbory); celočíselný typ sa označuje slovom **Integer**;
4. pamäťové miesta pre zadeklarované premenné sa vyhradia až pri vykonávaní danej procedúry, a teda si to môžeme predstaviť tak, že premenné vzniknú až pri slovíčku **begin** - tieto pamäťové miesta (premenné) majú pri svojom vzniku nedefinovanú hodnotu (dopredu neurčenú)
5. premenné v programe používame tak, že im hocikedy môžeme **priradiť** novú hodnotu, pričom ich pôvodná sa tým automaticky zabúda - v jednej premennej môže byť uložená vždy len jedna hodnota; okrem toho môžeme používať hodnoty premenných rovnako, ako sme pracovali s pomenovanými konštantami: všade, kde sa vyskytne meno premennej, pracuje sa s jej hodnotou;

6. keď beh procedúry končí (výpočet prišiel na zodpovedajúci **end**), všetky vyhradené pamäťové miesta sa zrušia - obsahy premenných sa zabudnú.

Postupne sa na tieto pravidlá pozrime.

Ako deklaruje premenné: za slovo **var** zapisujeme dvojicu identifikátor a typ - medzi ne dávame dvojbodku (na rozdiel od znaku rovná sa pri konštantách). Napr.:

```
var
  Velkost: Integer;
  Hrubka: Integer;
  X: Integer;
  Y: Integer;
```

pričom to môžeme skrátiť aj takto:

```
var
  Velkost, Hrubka, X, Y: Integer;
```

Ako premennej priradíme hodnotu: slúži na to priradovací príkaz, ktorý je úplne rovnaký, ako keď sme menili hodnotu rôznym nastaveniam grafickej plochy a tlačidlám. Priradovací príkaz má na ľavej strane premennú, do ktorej chceme niečo priradiť, potom nasleduje znak priradenia (dvojznak :=) a na pravej strane je hodnota, ktorú chceme priradiť.

Všimnite si, že na pravej strane priradovacieho príkazu môžeme použiť aj nejaký jednoduchý výpočet (aritmetický výraz). Tento príkaz funguje totiž tak, že **najprv vyhodnotí pravú stranu** (vypočíta hodnotu aritmetického výrazu) a túto hodnotu **potom priradí** do určenej premennej.

Skôr ako budeme pracovať s premennými v nejakých skutočných programoch, naučme sa ručne odtrasovať priebeh výpočtu, ktorý sa skladá z postupnosti priradovacích príkazov. Predpokladajme, že máme dve premenné **Vaha** a **Cena** a dve konštanty **JednotkovaCena** a **Priplatok**. Zostavíme program, ktorý vypočíta cenu balíka, ktorý váži zadaný počet kíl (v premennej **Vaha**), keď poznáme jednotkovú cenu za kilogram a tiež jednorazový príplatok za expresné doručenie:

```
const
  JednotkovaCena = 2;
  Priplatok = 5;
var
  Vaha, Cena: Integer;
begin
  Vaha := 7;
  Cena := Vaha * JednotkovaCena;
  Cena := Cena + Priplatok;
end;
```

Trasovanie výpočtu znamená, že si vytvoríme tabuľku, do ktorej v každom stĺpci budeme ukladať momentálnu hodnotu príslušnej premennej:

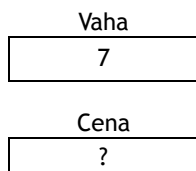
	Vaha	Cena
begin	?	?
Vaha := 7;	7	
Cena := Vaha * JednotkovaCena;		14
Cena := Cena + Priplatok;		19
end;	neex.	neex.

Otáznikom značíme zatiaľ nedefinovaný obsah premennej. V poslednom riadku vidíme, že **end** spôsobí zrušenie všetkých deklarovaných premenných a hoci procedúra aj niečo spočítala (cenu balíka), ak sme tento výsledok nikde neuchovali alebo nevy písali, už sa ho nedozvieme - je zrušený.

Príklady priradovacích príkazov:

```
Velkost := 150;
Hrubka := 2 * 3;
X := 10;
Y := X + 10;
```


Inokedy nepoužijeme tabuľku, ale namiesto nej niektoré premenné znázorníme schránkami (škatuľkami, rámikmi) a hodnoty zapisujeme dovnútra, prípadne prepisujeme staré novými. Napr.:



Môžeme vidieť, že v premennej **Vaha** je momentálne hodnota 7 a premenná **Cena** má zatiaľ nedefinovanú hodnotu.

Príklad operácií **div**:

```
14 div 5 = 2
14 div 2 = 7
736 div 10 = 73
736 div 100 = 7
```

a **mod**:

```
14 mod 5 = 4
14 mod 2 = 0
736 mod 10 = 6
736 mod 100 = 36
```

Trasovacia tabuľka nám môže pomôcť pri pochopení fungovania nejakého programu (algoritmu) hlavne neskôr pri náročnejších témach. V tabuľke je často prvý stĺpec s riadkami programu nahradený iba číslami riadkov.

V ďalšom príklade "rozoberieme" nejaké trojčiferné číslo (zadané v konštante **Cislo**) na cifry (do premenných **Cifra1**, **Cifra2**, **Cifra3**) a na záver spočítame súčet týchto cifier:

```
const
  Cislo = 736;
var
  Cifra1, Cifra2, Cifra3, Sucet: Integer;
begin
  Cifra1 := Cislo div 100;
  Cifra2 := (Cislo div 10) mod 10;
  Cifra3 := Cislo mod 10;
  Sucet := Cifra1 + Cifra2 + Cifra3;
end;
```

Vytvoríme trasovaciu tabuľku:

	Cifra1	Cifra2	Cifra3	Sucet
begin	?	?	?	?
Cifra1 := Cislo div 100;	7			
Cifra2 := (Cislo div 10) mod 10;		3		
Cifra3 := Cislo mod 10;			6	
Sucet := Cifra1 + Cifra2 + Cifra3;				16
end;	neex.	neex.	neex.	neex.

Použili sme tu dve celočíselné operácie: delenie (operátor **div**) a zvyšok po delení (operátor **mod**). Operácia **div** vráti vždy celú časť podielu, pričom zvyšok po delení sa zahadzuje. Operácia **mod** vráti zas len tento zvyšok po delení.

Na prácu s ciframi v desiatkovom zápise čísla nám najviac pomôžu práve delenie 10, resp. 100 a tiež zvyšok po delení 10, resp. 100.

Tiež si zapamätajme, že ak je zvyšok po delení nejakého čísla **A** číslom **B** nula, tak **B** je deliteľné **A**, napr. $14 \bmod 7 = 0$ znamená, že 7 delí 14. Zrejme platí, že číslo je párne, ak je deliteľné 2.

Náš program počítajúci ciferný súčet má ten "malý" nedostatok, že hoci ho počítač správne vypočítal, výsledok nám neoznámil. Už poznáme príkaz na vypisovanie textov do grafickej plochy (**TextOut**), len tento nevie vypísať čísla, ale len texty. Ak by sme zavolali

```
Image1.Canvas.TextOut(100, 100, Sucet);
```

už vieme, že dostaneme chybovú správu, v ktorej nám počítač oznamuje, že tretím parametrom musí byť text a nie číslo. Našťastie existuje konverzná funkcia **IntToStr**, ktorá dokáže prerobiť celé číslo na text a ten už dokážeme vypísať. Takže správne by sme mali zapísať

```
Image1.Canvas.TextOut(100, 100, IntToStr(Sucet));
```

Parametrom konverznej funkcie **IntToStr** nemusí byť len premenná, ale aj nejaký výpočet. Napr.

```
Image1.Canvas.TextOut(100, 130, IntToStr(1+2+3+4+5));
```

vypíše súčet čísel 1 až 5, t.j. číslo 15.

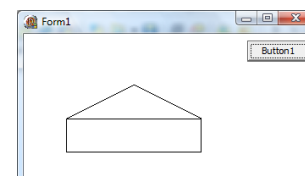
Nasledujúci program nakreslí domček, zložený z obdĺžnika a rovnoramenného trojuholníka, ktorého výška je rovnaká ako výška obdĺžnika. Pritom výška obdĺžnika je štvrtina šírky. Program používa konštanty **X**, **Y** na súradnice ľavého horného vrcholu obdĺžnika a **Sirka** na šírku obdĺžnika.

```

procedure TForm1.Button1Click(Sender: TObject);
const
  X = 50;
  Y = 100;
  Sirka = 160;
var
  Vyska, Xt, Yt: Integer;
begin
  Vyska := Sirka div 4;
  Image1.Canvas.Rectangle(X, Y, X + Sirka, Y + Vyska);
  Xt := X + Sirka div 2;
  Yt := Y - Vyska;
  Image1.Canvas.MoveTo(X, Y);
  Image1.Canvas.LineTo(Xt, Yt);
  Image1.Canvas.LineTo(X + Sirka, Y);
end;

```

Program nakreslí:



V programe sme použili okrem premennej **Vyska**, do ktorej sme vypočítali výšku obdĺžnika (štvrtina zo šírky), aj dve premenné **Xt** a **Yt**, do ktorých sme priradili súradnice najvyššieho vrcholu trojuholníka.

Úlohy na precvičenie

Zadanie 1	pomocou Image1.Width a Image1.Height môžeme zistiť presné rozmery grafickej plochy; nakreslite obdĺžnik, ktorý je od všetkých okrajov vzdialený presne o 10 bodov;
Zadanie 2	pre konštanty X1, Y1, X2, Y2 nakreslite úsečku z X1,Y1 do X2,Y2 a potom vypočítajte jej presný stred do X3,Y3 , tu nakreslite bodku veľkosti 3 a napíšte sem 'stred'
Zadanie 3	nakreslite štvorec otočený o 45 stupňov (kosoštvorec), ktorého strana je v konštante Strana a najvyšší vrchol má súradnice X, Y ; výpočty robte len v celočíselnej aritmetike: napr. (Strana * 141 div 100)
Zadanie 4	sumu v eurách (v konštante Euro) vypíšte a potom vypíšte aj jej prepočet na celé koruny (sumu vynásobte 30126 a potom vydajte 1000)
Zadanie 5	pre zadané konštanty X, Y a A nakreslite štvorec, ktorého ľavý horný roh je v X,Y a veľkosť strany je A ; potom v jeho vnútri presne v strede nakreslite štvorec s tretinovou veľkosťou strany - vypočítajte pritom premenné X3,Y3 (ľavý horný roh nového štvorca) a A3 veľkosť strany nového štvorca
Zadanie 6	zadané dvojciferné číslo (v konštante Cislo) rozoberte na cifry (premenne Cifra1 a Cifra2), a potom z nich zložte prevrátené číslo (najprv druhá cifra potom prvá); toto prevrátené číslo vypíšte

3. Aritmetický výraz

Teraz sa podrobnejšie pozrime na aritmetické výrazy, t.j. výrazy, ktoré obsahujú konštanty, premenné a aritmetické operácie. Keďže zatiaľ pracujeme len s celočíselnými hodnotami a premennými, budeme sa zaoberať len celočíselnými aritmetickými výrazmi. Z doterajších ukážok sme mohli vidieť, že máme k dispozícii týchto 5 operácií: **+**, **-**, *****, **div**, **mod**. Je veľmi dôležité rozumieť, ako počítač (teda jazyk pascal) vyhodnocuje výrazy, v ktorých je viac operácií za sebou. Napr.

$$5 + 6 \text{ div } 4 - 2$$

by mohol dávať rôzne výsledky v závislosti od poradia vyhodnocovania: ak by sa najprv urobil súčet potom delenie a odčítanie, tak výsledkom by bola 0; ak by sa najprv delilo a potom sčítalo a odčítalo, dostali by sme 4. Ešte iné výsledky by sme asi dostali, ak by sa najprv odčítalo.

Pascal má našťastie presné pravidlá, ako vyhodnocuje výrazy:

- najprv sa vyhodnocujú časti, ktoré sú v okrúhlych zátvorkách
- potom sa vyhodnotia operácie s vyššou prioritou: *, div, mod
- nakoniec operácie s nižšou prioritou +, -
- ak sú vedľa seba operácie s rovnakou prioritou, tak sa vyhodnocujú zľava doprava

Výnimku ešte predstavuje *unárne mínus*, ktoré sa, samozrejme, uplatní bezprostredne k svojmu operandu.

Pozrite si vyhodnocovanie niektorých konkrétnych výrazov:

$$\begin{aligned}5 + 6 \text{ div } 4 - 2 &= 5 + (6 \text{ div } 4) - 2 = 5 + 1 - 2 = 4 \\- 5 + 6 * 7 \text{ mod } 4 &= (-5) + ((6*7) \text{ mod } 4) = -5 + 2 = -3 \\(2 + 3) * (9 - 3) &= 5 * 6 = 30 \\(736 \text{ div } 10) \text{ mod } 10 &= 73 \text{ mod } 10 = 3\end{aligned}$$

Zátvorky sa často dávajú do aritmetických výrazov, aby sprehľadnili čitateľnosť programu, aj keď pre počítač sú niekedy zbytočné.

Pascal poskytuje aj niekoľko štandardných funkcií, z ktorých sú pre celočíselný typ zaujímavé **Abs** a **Sqr**. Funkcia **Abs** počíta absolútnu hodnotu čísla, napr. $\text{Abs}(7-5) = \text{Abs}(5-7) = 2$. Funkcia **Sqr** počíta druhú mocninu čísla, napr. $\text{Sqr}(3+4) = (3+4)^2 = (3+4)*(3+4) = 49$.

Prepíšte do pascalovského zápisu tieto výrazy:

- $(x-100)^2+(y-150)^2$
- $\frac{(a+b)+|a-b|}{2}$
- $2 \cdot 1000 + 7 \cdot 100 + 3 \cdot 10 + 6$
- $a x^2 + b x + c$
- $((x + 1)^2 + 1)^2 + 1)^2$

Pomocou prirad'ovacieho príkazu môžeme veľmi jednoducho meniť hodnotu samotnej premennej. Pozrime napr. takýto zápis:

```
Prem := Prem + 1;
```

označuje presne toto: zober obsah premennej **Prem**, pričítaj k nej číslo 1 a výslednú hodnotu prirad' do premennej **Prem**. Týmto zápisom sa zvýši obsah premennej o 1.

Pomocou trasovacej tabuľky zistíme obsahy premenných **Cislo** a **Sucet** po vykonaní postupnosti prirad'ovacích príkazov:

```
begin
  Cislo := 736;
  Sucet := 0;
  Sucet := Sucet + Cislo mod 10;
  Cislo := Cislo div 10;
  Sucet := Sucet + Cislo mod 10;
  Cislo := Cislo div 10;
  Sucet := Sucet + Cislo mod 10;
end;
```

Podobne môžeme zapísať

```
Strana := 2 * Strana;
Y := Y + Vyska;
Cislo := Cislo div 10;
Hodnota := Hodnota *
  Hodnota;
```

Prvé priradenie zdvojnásobí obsah premennej **Strana**. Druhé pripočíta k **Y** obsah premennej (alebo konštanty) **Vyska**. Tretie priradenie vydolí premennú **Cislo** desiatimi, t.j. odhodí z neho poslednú cifru. Posledné priradenie do premennej **Hodnota** priradí druhú mocninu samého seba, napr. ak bolo v **Hodnota** číslo 7, teraz tam už bude 49.

	Cislo	Sucet
begin	?	?
Cislo := 736;	736	
Sucet := 0;		0
Sucet := Sucet + Cislo mod 10;		6
Cislo := Cislo div 10;	73	
Sucet := Sucet + Cislo mod 10;		9
Cislo := Cislo div 10;	7	
Sucet := Sucet + Cislo mod 10;		16
end;	neex.	neex.

Tento program postupne rozoberá nejaké trojčiferné číslo na cifry (od konca) a pripočítava ich do premennej **Sucet**. Uvedomte si, že ak by sme na začiatku do premennej **Sucet** nepriradili nulu, tak by v nej ostala nedefinovaná hodnota a pripočítavať hocičo k takejto hodnote by nemalo žiaden zmysel - stále by tam bola nejaká nezmyselná hodnota.

Teraz ukážeme, ako môžeme vypísať obsahy premenných spolu s nejakými textami:

```
begin
  Cislo := 736;
  Image1.Canvas.TextOut(50, 50, 'číslo = ' + IntToStr(Cislo));
  Sucet := 0;
  Sucet := Sucet + Cislo mod 10;
  Cislo := Cislo div 10;
  Sucet := Sucet + Cislo mod 10;
  Cislo := Cislo div 10;
  Sucet := Sucet + Cislo mod 10;
  Image1.Canvas.TextOut(50, 80, 'CS = ' + IntToStr(Sucet));
end;
```

Použili sme tu operáciu zlepovania textov - podrobnejšie sa s týmto zoznámime až v module o znakových reťazcoch. Táto operácia funguje tak, že zoberie dva texty (nemusia to byť len textové konštanty) a zlepením z nich vytvorí jeden. Napr. text 'číslo = ' a `IntToStr(Cislo)` t.j. '736' zlepiť na text 'číslo = 736'.

Podobne by sme mohli zlepovať aj dve čísla, ale vtedy treba medzi ne vložiť nejaké znaky (napr. medzeru), inak by sme vo výpise videli len jedno veľké zlepené číslo. Pre $X = 123$ a $Y = 45$ by príkaz

```
Image1.Canvas.TextOut(50, 50, IntToStr(X) + IntToStr(Y));
```

vypísal text '12345'. Správne malo byť

```
Image1.Canvas.TextOut(50, 50, IntToStr(X) + ',' + IntToStr(Y));
```

ktoré vypíše '123,45'.

Ďalší príklad ilustruje spôsob, ako môžeme navzájom vymeniť obsahy dvoch premenných. Najprv tradičným spôsobom, pomocou ďalšej pomocnej premennej, do ktorej si odložíme obsah prvej premennej, potom do prvej priradíme obsah druhej a na koniec do druhej odložený obsah prvej:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P1, P2, Pom: Integer;
begin
  P1 := 1989;
  P2 := 314;
  Image1.Canvas.TextOut(50, 50, IntToStr(P1) + ':' + IntToStr(P2));
  Pom := P1;
  P1 := P2;
  P2 := Pom;
  Image1.Canvas.TextOut(50, 80, IntToStr(P1) + ':' + IntToStr(P2));
end;
```

Program vypíše tieto dva riadky:

```
1989:314
314:1989
```

Teda obsah premenných P1 a P2 pred výmenou a po výmene. Programátori niekedy vymýšľajú aj netypické riešenia rôznych problémov. Napr. aj výmena dvoch premenných sa dá urobiť bez pomocnej premennej:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  P1, P2: Integer;
begin
  P1 := 1989;
  P2 := 314;
  Image1.Canvas.TextOut(50, 50, IntToStr(P1) + ':' + IntToStr(P2));
  P1 := P1 + P2;
  P2 := P1 - P2;
  P1 := P1 - P2;
  Image1.Canvas.TextOut(50, 80, IntToStr(P1) + ':' + IntToStr(P2));
end;
```

Odporúčame odtrasovať oba spôsoby.

Pri celočíselnej aritmetike môžu niekedy nastať aj chyby. Jednou z nich je delenie nulou, ktoré spôsobí spadnutie programu. Niekedy takéto delenie nie je vidieť na prvý pohľad, napr. obe tieto priradenia by spôsobili spadnutie programu

```
Hodnota1 := 17 div (16 mod 4);
Hodnota2 := 17 mod (3 div 4);
```

Ďalším problémom je tzv. **pretečenie**, ktoré vyplýva z obmedzení celočíselnej aritmetiky. Celé čísla sa v Delphi ukladajú do 4 bajtov, teda do 32 bitov a keďže sa tu reprezentujú aj záporné čísla, interval pre všetky celé čísla je len <- 2 147 483 648, 2 147 483 647>, čo si môžeme približne zapamätať ako ±2 miliardy. (Číslo 2 147 483 647 je $2^{31}-1$ a keby sme ho zapísali v dvojkovej sústave, tak by sa skladalo z 31 cifier 1).

Tento problém ukážeme na výpočte 12! a 13! - najprv do premennej **Sucin** vypočítame 12! ako súčin čísel od 1 do 12 a túto hodnotu vypíšeme do grafickej plochy. Potom túto premennú vynásobíme 13 a tým do nej dostaneme 13!

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Sucin: Integer;
begin
  Sucin := 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 * 11 * 12;
  Image1.Canvas.Font.Height := 32;
  Image1.Canvas.TextOut(50, 50, IntToStr(Sucin));
  Sucin := Sucin * 13;
  Image1.Canvas.TextOut(50, 100, IntToStr(Sucin));
end;
```

Po spustení tohto programu sa pod seba vypíšu tieto dve hodnoty:

```
479001600
1932053504
```

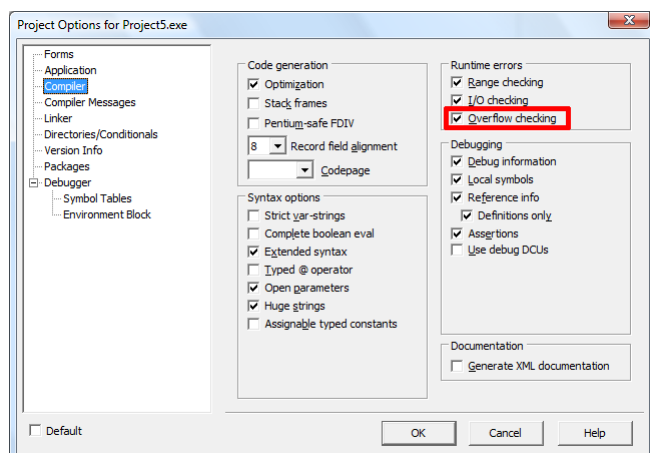
Ak by sme tieto výsledky prekontrolovali na kalkulačke, tak 12! je v poriadku, ale pre 13! by sme mali dostať

```
6227020800
```

My už ale vieme, že celé čísla fungujú len pre hodnoty niečo do dvoch miliárd a pritom 13! je už viac ako 6 miliárd. Z tohto príkladu si treba zobrať takéto poučenie: ak máme v programe nejakú chybu (napr. nekontrolujeme, či pri aritmetických operáciách nevzniklo pretečenie), môžeme dostať zlé výsledky a

pritom sa program "tvári", ako keby bolo všetko v poriadku.

Prostredie Delphi umožňuje nastaviť automatickú kontrolu pretečenia aritmetických operácií. V nastaveniach kompilátora zvolíme **Overflow checking**:



Ak teraz znovu spustíme náš program s výpočtom $12!$ a $13!$, tak program spadne na takejto chybovej správe: **Integer overflow**.

V nasledujúcom príklade napíšeme do grafickej plochy nejaký text (konštanta **Text**) veľkosti podľa konštanty **Velkost** tak, aby bol na riadky plochy vycentrovaný (voľný priestor nad aj pod textom má byť približne rovnaký). Na zistenie výšky grafickej plochy využijeme **Image1.Height**.

```
procedure TForm1.Button1Click(Sender: TObject);
const
  Text = 'Ján Amos Komenský';
  Velkost = 30;
var
  Y: Integer;
begin
  Y := (Image1.Height - Velkost) div 2;
  Image1.Canvas.Font.Height := Velkost;
  Image1.Canvas.TextOut(10, Y, Text);
end;
```

Namiesto premennej **Y** sme mohli celý aritmetický výraz zapísať ako parameter príkazu **TextOut**,

```
Image1.Canvas.TextOut(10, (Image1.Height-Velkost) div 2, Text);
```

ale vďaka pomocnej premennej **Y** bude program pravdepodobne čitateľnejší.

Úlohy na precvičenie

Zadanie 1	prekontrolujte 5^{13} a 5^{14} (mocninu počítajte násobením $5*5*5*5*$...)
Zadanie 2	pomocou kalkulačky zistite, akú najväčšiu mocninu 11 vieme vypočítať v celočíselnej aritmetike; potom to preverte programom, ktorý vypíše túto mocninu a aj jej 11-násobok (ktorý by mal spôsobiť pretečenie)
Zadanie 3	zistite, ako fungujú operácie div a mod , ak je deliteľom záporné číslo
Zadanie 4	program nakreslí štvorec so stranou A , a potom vypíše jeho obsah a obvod

4. Generátor náhodných čísel

Random je špeciálna funkcia, ktorá vytvára náhodné hodnoty. Príkladom použitia by mohla byť tombola alebo lotéria, z ktorej vie počítač náhodne vybrať výhercu. Táto funkcia umožňuje, aby počítač zakaždým, keď ju zavoláme, zvolil nejaké náhodné číslo. Funkcia má jeden parameter - celé číslo a počítaču ním oznámime, z akej veľkej množiny sa bude náhodne vybrať. Každé zavolanie funkcie **Random(N)** vyberie číslo od 0 po N-1, t.j. jedno z N rôznych hodnôt. Volanie **Random(6)** vyberie jednu z hodnôt zo 6 prvkovej množiny {0, 1, 2, 3, 4, 5} - môžeme si to predstaviť tak, že počítač si zoberie hraciu kocku, na ktorej sú čísla od 0 do 5 a vždy, keď od neho chceme nejakú náhodnú hodnotu **Random(6)**, tak si hodí touto kockou a vráti nám to číslo, ktoré padlo na tejto kocke. Zrejme, keď túto funkciu zavoláme viackrát, tak počítač môže dávať rôzne výsledky.

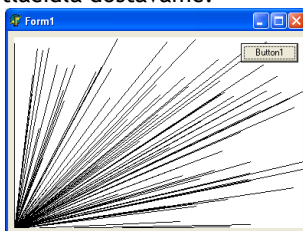
Funkciu **Random** otestujeme takýmto programom:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Cislo: Integer;
begin
  Cislo := Random(6);
  Image1.Canvas.Font.Height := 40;
  Image1.Canvas.TextOut(50, 50, IntToStr(Cislo));
end;
```

Poznámka:
keď program spustíme znovu, zakaždým bude generovať tie isté náhodné čísla.

Vždy, keď zatlačíme tlačidlo **Button1**, zavolá sa funkcia **Random(6)** a táto vygenerovaná hodnota sa vypíše. Ak budete tlačidlo tlačiť viackrát, vždy sa objaví nejaké nové číslo (hoci niekedy sa vygeneruje rovnaké číslo za sebou).

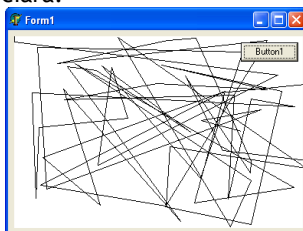
Po niekoľkých zatlačeniach tlačidla dostávame:



V ďalšom príklade budeme generovať úsečky, ktorých jeden vrchol je v bode (0, 250) a druhý je vygenerovaný dvojicou náhodných čísel: pre X od 0 do 399 a pre Y od 0 do 249:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.MoveTo(0, 250);
  Image1.Canvas.LineTo(Random(400), Random(250));
end;
```

Po spustení programu sa kreslí náhodná lomená čiara:



Úplne iný obrázok dostaneme, keď vyhodíme riadok s príkazom **MoveTo**:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  // Image1.Canvas.MoveTo(0, 250);
  Image1.Canvas.LineTo(Random(400), Random(250));
end;
```

Pripomíname, že ak riadok programu obsahuje dve lomky za sebou, tak zvyšok riadku sa považuje za komentár. Komentáre v programoch slúžia väčšinou ako vysvetľujúce informácie, alebo ako je to tento prípad, nejakú časť programu dočasne pred kompilátorom ukryjeme.

Pomocou funkcie **Random** môžeme generovať nielen čísla z intervalu $\langle 0, N-1 \rangle$, ale ak k výsledku niečo pripočítame, alebo ho vynásobíme, môžeme dostať aj iné náhodné množiny. Napr.

Random(2)	{0, 1}
Random(3) - 1	{-1, 0, 1}
Random(6) + 1	{1, 2, 3, 4, 5, 6}
2 * Random(2) - 1	{-1, 1}
2 * Random(4) + 2	{2, 4, 6, 8}

Random(21)-10	<-10, 10>
Random(181)-90	<-90, 90>
Random(B-A+1)+A	interval <A, B>
Random(2) * (B-A)+A	dvojprvková množina {A, B}

Ďalší príklad nakreslí obdĺžnik s náhodnými stranami od 50 do 100. Jeho ľavý horný roh nech je 50, 50:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.Rectangle(50, 50, 100+Random(50), 100+Random(50));
end;
```

Ak by sme takto chceli nakresliť štvorec s náhodnými stranami, tak už potrebujeme pomocnú premennú, do ktorej priradíme náhodnú dĺžku (napr. do 50 do 200) a až potom nakreslíme štvorec:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A: Integer;
begin
  A := Random(150) + 50;
  Image1.Canvas.Rectangle(50, 50, 50 + A, 50 + A);
end;
```

Niekedy potrebujeme viac pomocných premenných. Nasledovný príklad nakreslí na náhodnej pozícii kruh s náhodným polomerom od 10 do 30. Náhodnú pozíciu treba generovať tak, aby stred kruhu bol vo vnútri grafickej plochy.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y, R: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  R := Random(20) + 10;
  Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
end;
```

Opäť sme využili na zisťovanie šírky a výšky grafickej plochy **Image1.Width** a **Image1.Height**. Keď budeme tlačidlo stláčať viackrát, grafická plocha sa zaplní malými bielymi krúžkami. Neskôr sa naučíme ich zafarbovať náhodnými farbami.

Ďalší príklad ilustruje, ako generovať náhodné pozície tak, aby sa celý obdĺžnik zmestil do grafickej plochy. Program vykresľuje na náhodné pozície obdĺžniky, do ktorých vpisuje nejaký zväčšený text.

```
procedure TForm1.Button6Click(Sender: TObject);
const
  Text = 'Bratislava';
  Vyska = 40;
  Sirka = 160;
var
  X, Y: Integer;
begin
  X := Random(Image1.Width - Sirka);
  Y := Random(Image1.Height - Vyska);
  Image1.Canvas.Rectangle(X, Y, X + Sirka, Y + Vyska);
  Image1.Canvas.Font.Height := Vyska - 4;
  Image1.Canvas.TextOut(X + 2, Y + 1, Text);
end;
```


Úlohy na precvičenie

Zadanie 1	na náhodnej pozícii sa nakreslí štvorec veľkosti 100x100, ktorý má náhodnú hrúbku strán od 1 do 10
Zadanie 2	na náhodnej pozícii grafickej plochy sa nakreslí náhodne veľký kruh od 10 do 30 ale tak, aby sa celý zmestil do grafickej plochy
Zadanie 3	nakreslia sa 2 kruhy tesne vedľa seba na náhodných pozíciách, pričom prvý má náhodný polomer od 10 do 30 a druhý od 20 do 50
Zadanie 4	nakreslí sa náhodný trojuholník, ktorého vrcholy sú od okrajov plochy vzdialené aspoň na 20
Zadanie 5	na náhodnej pozícii sa nakreslia tri vpísané kruhy tak, že najväčší má náhodný polomer R1 od 70 do 100, druhý kruh má náhodný polomer R2 od 40 do R1-5 a najmenší má náhodný polomer od 10 do R2-5
Zadanie 6	program, ktorý vymyslí štvormiestny PIN pre náš mobilný telefón: program by mal náhodne vygenerovať 4 čísla z intervalu <1, 9> a vypísať ich vedľa seba - medzi ciframi nechceme mať nulu

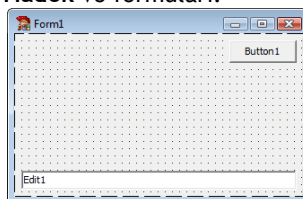
5. Zadávanie vstupu

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

Naučíme sa používať nový komponent, pomocou ktorého budeme vedieť nastavovať hodnotu premennej aj počas behu programu. Komponent sa nazýva **vstupný riadok** (niekedy **editovací riadok**) a v programe je typu **TEdit**. Vstupný riadok funguje takto:

- do formuláru našej aplikácie niekde umiestnime orámované políčko (komponent vstupný riadok **TEdit**), do ktorého sa dá písať počas behu programu ľubovoľný text
- po zatlačení tlačidla (napr. **Button1**) má náš program možnosť zistiť tento zapísaný text a ďalej ho spracovať, napr. tak, že z neho spraví číslo (prekonvertuje) a priradí do nejakej premennej
- vďaka tomuto môžeme pred každým zatlačením tlačidla zadať nejaký nový text a program bude zakaždým pracovať s nejakou inou hodnotou

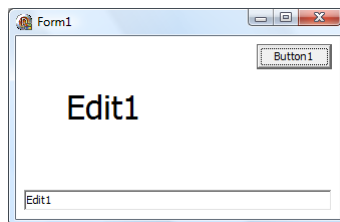
Takto vyzerá vstupný riadok vo formulári:



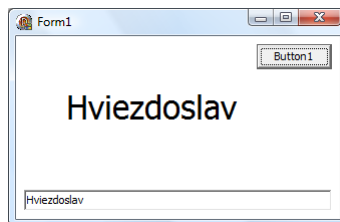
Ukážeme to na takomto jednoduchom príklade. Do formuláru okrem grafickej plochy (**TImage**), tlačidla (**TButton**) niekde vložíme aj komponent **vstupný riadok** (**TEdit**). Po spustení programu zatlačenie tlačidla "prečíta" text zo vstupného riadka a vypíše ho do grafickej plochy zväčšeným fontom:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 40;
  Image1.Canvas.TextOut(50, 50, Edit1.Text);
end;
```

Ak do vstupného riadku nezaadáme žiaden text a ostane tam pôvodný obsah 'Edit1', tak tento sa v programe prečíta (pomocou `Edit1.Text`) a vypíše:



Ak sem teraz zadáme ľubovoľný text a znovu zatlačíme tlačidlo `Button1`, tak sa program spustí so zmeneným vstupom. Napr.



Veľmi často ale budeme potrebovať text zo vstupného riadku priradiť do celočíselnej premennej ako číslo. V žiadnom prípade by nefungovalo takéto niečo:

```
var
  Cislo: Integer;
begin
  Cislo := Edit1.Text; // toto je chyba
  ...
```

My už vieme, že do celočíselnej premennej môžeme priradiť len celočíselný výraz, ale `Edit1.Text` je textová hodnota. Aj keby sme do vstupného riadku zapísali nejaké číslo, tak aj tak je to iba text. Problém je podobný ako pri vypisovaní celočíselnej premennej do grafickej plochy pomocou `TextOut`. Tam sme najprv museli číslo prekonvertovať na text pomocou `IntToStr` a až potom takýto text sme vypisovali.

Existuje podobná konverzná funkcia `StrToInt`, ktorá sa z textu pokúsi vyrobiť číslo. Pre porovnanie môžeme vidieť, že funkcia

<code>IntToStr(číslo) → text</code>	vyrob z čísla text
<code>StrToInt(text) → číslo</code>	vyrob z textu číslo

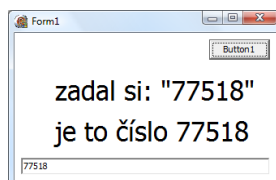
Pri tejto novej funkcii si ale treba dať pozor na to, že ak je nesprávny formát čísla, tak program vyhlási chybu a vykonávanie programu sa potom preruší.

Na otestovanie sme napísali takýto program:

```
var
  Cislo: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 40;
  Image1.Canvas.TextOut(50, 50, 'zadal si: ' + Edit1.Text + '');
  Cislo := StrToInt(Edit1.Text);
  Image1.Canvas.TextOut(50, 100, 'je to číslo ' + IntToStr(Cislo));
end;
```

Všimnite si, ako sme pomocou operácie zret'azenia pridali pred začiatok a na koniec vypisovaného textu zo vstupného riadku úvodzovky.

Keď zadáme text, ktorý obsahuje číslo v správnom formáte, dostávame



Príklady správnych textov pre `StrToInt`:

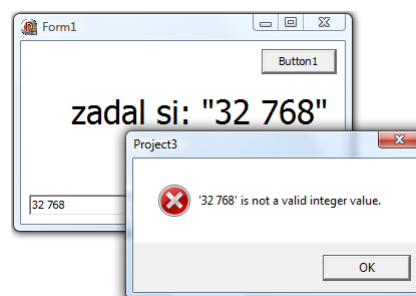
```
StrToInt('37') → 37
StrToInt('0037') → 37
StrToInt(' 37 ') → 37
StrToInt('-37') → -37
StrToInt(' +37 ') → 37
```

Príklady pre `StrToInt` s chybami - Delphi oznámia chybu:

toto nie je správne celé číslo

```
StrToInt('37 ')
StrToInt('x37')
StrToInt(' 37 ')
StrToInt('- 37')
StrToInt(' 37;')
```

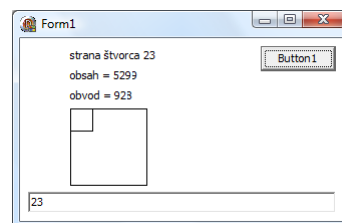
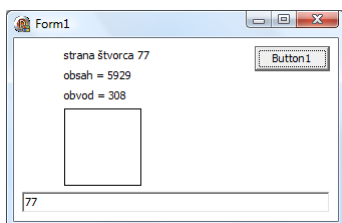
inak sa pri chybnom vstupe objaví správa o tom, že zadaný text nie je vhodný pre celé čísla (**Integer**), napr.



V ďalšom príklade nakreslíme štvorec s veľkosťou strany zadanej cez vstupný riadok a vypíšeme jeho obsah a obvod:

```
var
  A: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  Image1.Canvas.Rectangle(50, 70, 50 + A, 70 + A);
  Image1.Canvas.TextOut(50, 10, 'strana štvorca ' + IntToStr(A));
  Image1.Canvas.TextOut(50, 30, 'obsah = ' + IntToStr(Sqr(A)));
  Image1.Canvas.TextOut(50, 50, 'obvod = ' + IntToStr(4 * A));
end;
```

Všimnite si, že pri každom zatlačení tlačidla sme najprv zmazali grafickú plochu (príkaz **FillRect**) a až potom sme do nej písali a kreslili. Toto robíme vtedy, keď počítame s tým, že sa príkaz bude vykonávať viackrát za sebou (možno zakaždým s iným vstupným číslom) a nechceme, aby ostávali nejaké zvyšky z predchádzajúceho kreslenia. Vyskúšajte spustiť tento program aj bez úvodného mazania plochy (prvý je správny výpis, druhý výpis obsahuje zvyšky z predchádzajúceho zadania hodnoty 77):



Ďalší príklad ilustruje situáciu, keď sa nám hodí nemazať plochu pri každom zatlačení tlačidla. Program nakreslí kružnicu so zadaným polomerom a so stredom v (100, 70).

```
var
  R: Integer;
begin
  R := StrToInt(Edit1.Text);
  Image1.Canvas.Ellipse(100 - R, 70 - R, 100 + R, 70 + R);
end;
```

Radi by sme pomocou tohto programu dosiahli to, že postupne nakreslíme viac sústredných kružníc, t.j. postupne do vstupného riadku zadáme napr. 50, 30, 10, 70 a zakaždým stlačíme tlačidlo **Button1**. Kým zadávame stále menšie polomery, všetko je v poriadku - nové kružnice sa pridávajú k doteraz nakresleným. Ak ale teraz pridáme najväčšiu kružnicu s polomerom 70, tak pôvodne nakreslené kružnice sa zmažú. Kružnice sa totiž kreslia s vyplneným vnútrom bielou farbou. Ale aj tomuto sa dá pomôcť, lebo príkaz **Ellipse** (ale aj **Rectangle** a **TextOut**) vieme nastaviť tak, aby sa nevyplňal farbou štetca, ale mal priesvitné vnútro (**TextOut** používa farbu štetca ako podklad pod vypisovaný text). Použijeme nové nastavenie pre štetec **Style**, pomocou ktorého zrušíme alebo nastavíme vyplňanie útvarov:

```

Image1.Canvas.Brush.Style := bsClear; // vypni vyplňanie
Image1.Canvas.Brush.Style := bsSolid; // zapni vyplňanie

```

a teda náš program by mohol teraz vyzerat' takto:

```

var
  R: Integer;
begin
  R := StrToInt(Edit1.Text);
  Image1.Canvas.Brush.Style := bsClear; // vypni vyplňanie
  Image1.Canvas.Ellipse(100 - R, 70 - R, 100 + R, 70 + R);
end;

```

Pozor, bežiaci program si toto nastavenie pamätá a ak budeme chcieť niečo kresliť s vyplňaním, alebo mazat' plochu pomocou **FillRect**, tak sa nič nevyplní, kým opäť nezapneme **bsSolid**.

Na tomto malom príklade si ukážeme ešte jednu novú možnosť, ako využiť vstupný riadok: tento textový riadok môžeme z programu nielen čítať, ale ho aj meniť. Napr. vždy keď spracujeme jeho hodnotu, tak tento riadok vyprázdňujeme pomocou

```
Edit1.Text := '';
```

alebo jeho hodnotu zvýšime napr. o 10:

```

var
  R: Integer;
begin
  R := StrToInt(Edit1.Text);
  Image1.Canvas.Brush.Style := bsClear;
  Image1.Canvas.Ellipse(100 - R, 70 - R, 100 + R, 70 + R);
  Edit1.Text := IntToStr(R + 10);
end;

```

Ak napr. do vstupného riadku zapíšeme hodnotu 5 a stlačíme tlačidlo **Button1**, nakreslí sa kružnica s polomerom 5 a vo vstupnom riadku sa objaví hodnota 15. Opätovné zatlačenie tlačidla nakreslí kružnicu s polomerom 15 a zvýši hodnotu vstupného riadka na 25. Kolkokrát zatlačíme tlačidlo, tolko stále zväčšujúcich kružníc sa nakreslí. Do vstupného riadka už ďalej nič písať nemusíme, nové hodnoty sa tam objavujú samé.

Vstupných riadkov môžeme mať v programe aj viac. Nasledovný program ilustruje použitie dvoch vstupných riadkov, kde v prvom zadáme veľkosť textu a v druhom vypisovaný text. Program na náhodnú pozíciu vypíše zadaný text zadanej veľkosti. Už vieme, že pri definovaní komponentov vo formulári im môžeme meniť niektoré nastavenia v Inšpektore objektov. Takto môžeme obom vstupným riadkom (**Edit1** a **Edit2**) zmeniť ich nastavenie **Text**.

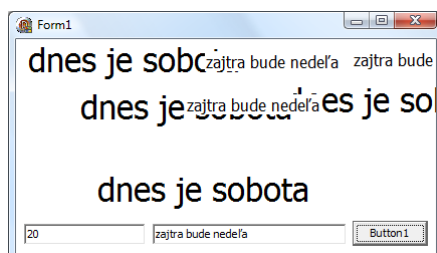
Samotný program potom pracuje s oboma vstupnými riadkami:

```

begin
  Image1.Canvas.Font.Height := StrToInt(Edit1.Text);
  Image1.Canvas.TextOut(Random(400), Random(150), Edit2.Text);
end;

```

Po niekoľkých zatlačeniach tlačidla dostávame niečo takéto:



Pre **Edit1** zapíšeme 40 a pre **Edit2** nejaký text, napr. :



Úlohy na precvičenie

Zadanie 1	nakresliť štvorec so zadanou veľkosťou strany a potom nakresliť kružnicu opísanú tomuto štvorcovi (polomer počítajte celočíselne približne podľa vzorca $a/2/2 \approx 70a/100$)
Zadanie 2	pre zadanú dĺžku strany nakresliť rovnostranný trojuholník (výšku trojuholníka vypočítajte približne podľa vzorca $a/3/2 \approx 87a/100$)
Zadanie 3	z jedného vstupného riadku prečíta nejaký text, z druhého číslo, ktoré vyjadruje posunutie tieňa; potom vypíše tento text aj s bledošedým tieňom, ktorý je posunutý aj v x-ovom aj v y-ovom smere o zadané číslo
Zadanie 4	z dvoch vstupných riadkov nakresliť obdĺžnik
Zadanie 5	zadané sú veľkosť a hrúbka štvorca
Zadanie 6	z dvoch vstupných riadkov prečíta dve hodnoty A a B a v grafickej ploche nakreslí dva štvorce so stranami A a B , ktoré sa dotýkajú jednou stranou a ich stredy ležia na priamke rovnobežnej s x-ovou osou

6. Farebný model RGB

S farbami sme sa stretli pri kreslení čiar (**Pen.Color**), pri vyplňaní plôch (**Brush.Color**) a pri písaní textu (**Font.Color**). Doteraz sme museli vystačiť s jednou z niekoľkých preddefinovaných (**clRed**, **clBlue** a podobne).

Teraz sa zoznámime s farbami v počítači a aj pri programovaní podrobnejšie. Môžeme predpokladať, že všetky farby v počítači sú namiešané z troch základných farieb: červenej, zelenej a modrej (tzv. model **RGB**, teda **Red**, **Green**, **Blue**). Konkrétna farba závisí od toho, koľko je v nej zastúpená každá z týchto troch farieb. Zastúpenie jednotlivých farieb vyjadrujeme číslom od 0 do 255 (zmestí sa do jedného bajtu), napr. žltá farba vznikne, ak namiešame 255 červenej, 255 zelenej a 0 modrej. Ak budeme zastúpenie každej farby trochu meniť, napr. 250 červenej, 240 zelenej a hoci 100 modrej, stále to bude žltá, ale iného odtieňa. Na skladanie farieb máme k dispozícii funkciu **RGB**, ktorej zadáme tri čísla od 0 do 255 a ona vytvorí príslušnú farbu, napr. známe preddefinované farby majú takéto vyjadrenie:

```
clBlack      =   RGB(0, 0, 0)
clRed        =   RGB(128, 0, 0)
clGreen      =   RGB(0, 128, 0)
clBlue       =   RGB(0, 0, 255)
clGray       =   RGB(128, 128, 128)
clYellow     =   RGB(255, 255, 0)
clWhite      =   RGB(255, 255, 255)
```

V skutočnosti sú farby v počítači zakódované v štvorbajtovej hodnote - rovnako ako celé čísla **Integer**, hoci jeden bajt tu má počítač "rezervovaný pre seba". Dokonca nám Delphi dovoľia pracovať s týmito farbami, ako keby to boli celé čísla. Ak predpokladáme, že rôznych farieb v počítači je $256*256*256 = 16777216$, tak by sme náhodnú farbu mohli generovať aj takýmto zápisom:

```
Image1.Canvas.Brush.Color := Random(256*256*256);
```

Toto môžeme zapísať aj pomocou funkcie **RGB**. Prípadne môžeme "namiešať" naozaj ľubovoľnú farbu podľa našich predstáv, t.j. môžeme generovať náhodné farby s rôznymi vlastnosťami, napr.:

`RGB(Random(256), Random(256), Random(256))` → úplne náhodná farba
`RGB(0, 0, Random(256))` → odtiene modrej (od čiernej po modrú)
`RGB(Random(256), 0, Random(256))` → náhodná fialová (červená až modrá)

Prvý program, ktorým otestujeme generovanie farieb, bude po zatlačení tlačidla kresliť kruh na náhodnú pozíciu, náhodnej veľkosti (od 10 do 40) a náhodnej farby:

```

var
  X, Y, R: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  R := Random(30) + 10;
  Image1.Canvas.Brush.Color := RGB(Random(256), Random(256),
  Random(256));
  Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
end;
  
```

Každé zatlačenie tlačidla nakreslí jeden náhodne zafarbený kruh. Veľa zatlačení zaplní grafickú plochu veľa farebnými kruhmi (keď na klávesnici podržíme kláves Enter, tak počítač bude generovať sériu zatlačení tlačidla).

Všimnite si, že kruhy sú orámované čiernou farbou - tento rámik môžeme zrušiť tak, že ho zafarbíme farbou štetca:

```

var
  X, Y, R: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  R := Random(30) + 10;
  Image1.Canvas.Brush.Color := RGB(0, 0, Random(256));
  Image1.Canvas.Pen.Color := Image1.Canvas.Brush.Color;
  Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
end;
  
```

Ďalší program bude kresliť hrubú farebnú úsečku od horného okraja po dolný. X-ová súradnica bude generovaná náhodne od 0 do 255 a farba bude úplne náhodná. Grafickú plochu môžeme nastaviť na veľkosť 256x256 (v Inšpektore objektov nastavíme vlastnosti **Width** a **Height** na 256).

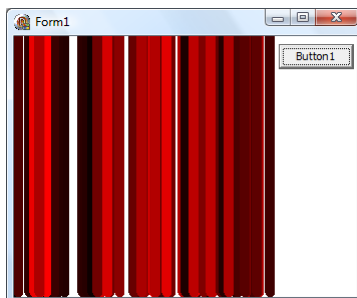
```

var
  X: Integer;
begin
  X := Random(256);
  Image1.Canvas.Pen.Color := Random(256 * 256 * 256);
  Image1.Canvas.Pen.Width := 10;
  Image1.Canvas.MoveTo(X, 0);
  Image1.Canvas.LineTo(X, 255);
end;
  
```

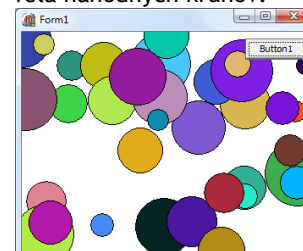
Zmenou riadku, ktorý nastavuje farbu pera (**Pen.Color**) môžeme dosiahnuť rôzne efekty, napr.

`Image1.Canvas.Pen.Color := RGB(Random(256), 0, 0);`

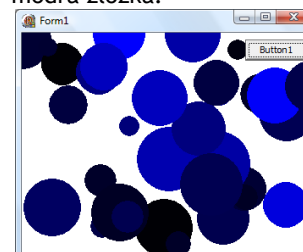
bude náhodne generovať len odtiene červenej (v tomto príkaze môžete vyskúšať aj odtiene iných farieb):



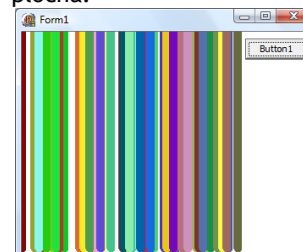
Veľa náhodných kruhov:



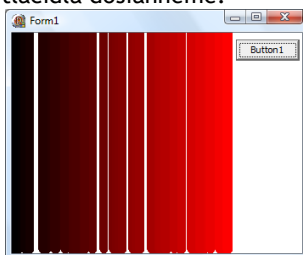
Zároveň sme pozmenili generovanie náhodnej farby: červenú a zelenú zložku farby sme vynulovali a náhodne sa generuje len modrá zložka:



Po zatlačení tlačidla sa nakreslí len jedna úsečka, ale po mnohých zatlačeniach sa vyfarbí celá plocha:



Veľanásobným stlačením tlačidla dosiahneme:



Ďalším zaujímavým vylepšením je zapísanie závislosti farebného odtieňa od X-ovej súradnice, teda od umiestnenia úsečky - čím viac napravo, tým svetlejší odtieň:

```
var
  X: Integer;
begin
  X := Random(256);
  Image1.Canvas.Pen.Color := RGB(X, 0, 0);
  Image1.Canvas.Pen.Width := 10;
  Image1.Canvas.MoveTo(X, 0);
  Image1.Canvas.LineTo(X, 255);
end;
```

odporúčame experimentovať so zápsmi typu

```
Image1.Canvas.Pen.Color := RGB(X, X, 0);
Image1.Canvas.Pen.Color := RGB(X, 255 - X, 0);
Image1.Canvas.Pen.Color := RGB(X, 250, 150);
Image1.Canvas.Pen.Color := RGB(X, X, X);
```

Zaujímavý je posledný zápis, v ktorom môžeme vidieť, ako sa vytvárajú odtiene šedej: od čiernej až po bielu - všetky tri farebné zložky musia mať rovnakú hodnotu.

Úlohy na precvičenie

Zadanie 1	náhodne zafarbíte písmená nejakého slova (napr. DELPHI) - každé zatlačenie tlačidla vypíše toto slovo s inak zafarbenými písmenami
Zadanie 2	navrhnete vlastnú vlajku, ktorá sa skladá z troch pruhov rôznej farby, každý pruh môže byť generovaný náhodne z nejakých odtieňov farby
Zadanie 3	nakreslí 4 sústredné kružnice (napr. s polomeri 20, 40, 60, 80), ktoré majú náhodne generované farby výplne - farby nemusia byť úplne náhodné, môžu byť náhodne generované podľa nejakých pravidiel (napr. 0 červenej, X zelenej a 256-X zelenej); pouvažujte, čo sa stane pri rôznych poradiach vykresľovania týchto kružníc

2. tematická jednotka - Cyklus s pevným počtom opakovaní

1. Cyklus s pevným počtom opakovaní

V predchádzajúcej časti sme vytvorili niekoľko programov, ktoré pri zatlačení tlačidla nakreslia jednu úsečku, alebo jeden kruh, alebo jeden obdĺžnik a pod. Pri týchto úlohách sme často predpokladali, že tlačidlo zatlačíme viackrát a vtedy sa na obrazovke objaví nejaký zaujímavý obraz. Tu by sme potrebovali nejaký mechanizmus, ktorý by jeden príkaz (alebo možno aj postupnosť príkazov) vedel zopakovať veľa krát, bez toho, aby sme to museli veľa krát zapisovať.

Takýmto mechanizmom je **cyklus**. Hovoríme tomu programová konštrukcia. Je to taký príkaz, ktorého súčasťou sú nejaké iné príkazy. V prípade konštrukcie cyklu bude tieto iné príkazy opakovať. Tento konkrétny cyklus má tzv. pevný počet opakovaní, t.j. dopredu musíme zadať, koľkokrát sa bude niečo opakovať. Princíp práce vysvetlíme na príklade, kde program kreslí náhodnú lomenú čiaru:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.LineTo(Random(400), Random(250));
end;
```

Teraz použijeme cyklus, ktorý nakreslí naraz 50 podobných čiar, ako v predchádzajúcom príklade:

```
var
  I: Integer;
begin
  for I := 1 to 50 do
    Image1.Canvas.LineTo(Random(400), Random(250));
  end;
```

Na to, aby mohol takýto cyklus fungovať, potrebujeme tzv. **počítadlo cyklu** (niekedy mu hovoríme aj **riadiaca premenná**), preto sme zadeklarovali premennú I, ktorá bude pre cyklus takýmto počítadlom. Počítač si v tejto premennej počíta (preto počítadlo) **prechody cyklu** - v našom prípade sú to hodnoty 1 až 50.

Konštrukciu môžeme zapísať takto:

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

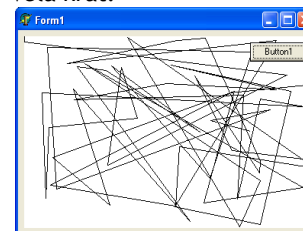
kde **premenná** je počítadlo cyklu a nesmieme ju zabudnúť zadeklarovať. **Dolná a horná hranica** cyklu sú dve celočíselné hodnoty definujúce interval hodnôt, pre ktorý sa bude postupne (pre každú hodnotu z tohto intervalu) vykonávať zadaný **príkaz**. To znamená, že najprv sa do počítadla priradí dolná hranica cyklu a vykoná sa príkaz (**telo cyklu**). Potom sa počítadlo zvýši o jednotku a opäť sa vykoná príkaz. Toto sa opakuje, až kým počítadlo nepresiahne hornú hranicu. Vtedy sa cyklus zastaví a pokračuje sa v príkazoch nasledujúcich za cyklom.

V ďalšom príklade sa 30-krát vypíše text na náhodné pozície:

```
var
  I: Integer;
begin
  Image1.Canvas.Font.Height := 32;
  Image1.Canvas.Brush.Color := clYellow;
  for I := 1 to 30 do
    Image1.Canvas.TextOut(Random(300), Random(200), 'Abc');
  end;
```

Môžeme si všimnúť, že veľkosť písma a pozadie pod textom sme nastavili iba raz ešte pred cyklom a potom sme 30-krát už iba písali text.

Po spustení programu sa kreslí náhodná lomená čiara - ale len, keď tlačidlo **Button1** zatlačíme naozaj veľa krát:



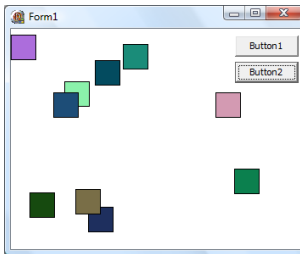
Dolná hranica cyklu (hodnota, od ktorej začína cyklus) nemusí byť číslo 1. Nasledujúci príklad ukazuje cyklus, ktorého dolná hranica je 10 a horná 20:

```
var
  I: Integer;
begin
  for I := 10 to 20 do
    Image1.Canvas.TextOut(Random(300), Random(200), '*');
  end;
```

Program bude postupne pre všetky hodnoty počítadla od 10 do 20 vypisovať jeden znak hviezdička - týchto hviezdičiek teda bude 11.

Pomocou cyklu môžeme vykonať nielen jeden grafický príkaz, ale napr. aj zatlačiť nejaké iné tlačidlo. Vytvoríme aplikáciu, v ktorej budú 2 tlačidlá: prvé napr. nakreslí jeden farebný štvorec, druhé desaťkrát zatlačí prvé tlačidlo, t.j. nakreslí sa 10 štvorcov:

Stačí jedno kliknutie na tlačidlo **Button2** a nakreslí sa 10 náhodných štvorcov. Pritom toto tlačidlo 10-krát vyvolalo zatlačenie tlačidla **Button1**:



```
procedure TForm1.Button1Click(Sender: TObject);
const
  A = 30;
var
  X, Y: Integer;
begin
  X := Random(Image1.Width - A);
  Y := Random(Image1.Height - A);
  Image1.Canvas.Brush.Color := Random(256*256*256);
  Image1.Canvas.Rectangle(X, Y, X + A, Y + A);
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 1 to 10 do
    Button1.Click;
  end;
```

Na záver tejto časti pozorne pozrime tento program:

```
var
  I: Integer;
begin
  for I := 1 to 100 do
    Image1.Canvas.Pen.Width := Random(10) + 1;
    Image1.Canvas.LineTo(Random(350), Random(250));
  end;
```

Program najprv 100-krát nastaví hrúbku pera pre kreslenie na náhodné číslo od 1 do 10 a potom nakreslí len jednu úsečku. V tomto prípade bolo použitie cyklu úplne zbytočné: pri kreslení záverečnej úsečky sa použije len posledne nastavená hrúbka a predchádzajúcich 99 nastavení hrúbky sa vôbec nevyužilo. Porozmýšľajte nad podobnými príkladmi, kde by sa zbytočne opakoval nejaký príkaz (napr. zmena farby, MoveTo, kreslenie úplne toho istého útvaru a pod.). Je dobre si toto uvedomovať, nakoľko je to častá chyba začiatočníka.

Úlohy na precvičenie

Zadanie 1 prekontrolujte, koľkokrát sa vykoná príkaz v cykle:

- ak dolná hranica > horná hranica
- ak dolná hranica = horná hranica
- ak dolná hranica = -10 a horná hranica = 1, (for I := -10 to 1 do)
- ak dolná hranica = -1 a horná hranica = -10, (for I := -1 to -10 do)

Zadanie 2

program najprv nakreslí modrú krivku z 10 náhodných úsečiek a potom červenú z 10 náhodných úsečiek (pre oba cykly môžete použiť jednu a tú istú premennú)

Zadanie 3

keď do grafickej plochy zapíšeme medzeru pričom nastavíme pozadie (Brush.Color) napr. na modrú, tak sa vykreslí malý farebný obdĺžnik; využijete túto vlastnosť a nakreslite 100 malých modrých obdĺžnikov, 100 malých červených obdĺžnikov a 100 malých žltých obdĺžnikov

2. Viac príkazov v tele cyklu

Pozrime posledný "nepodarený" program z predchádzajúcej časti, ktorý najprv zbytočne 100-krát menil hrúbku pera a potom nakreslil len jednu čiaru:

```
var
  I: Integer;
begin
  for I := 1 to 100 do
    Image1.Canvas.Pen.Width := Random(10) + 1;
    Image1.Canvas.LineTo(Random(350), Random(250));
  end;
```

Je pravdepodobné, že autor tohto programu zamýšľal 100-krát opakovať nielen príkaz na zmenu hrúbky pera, ale naraz oba príkazy: zmena hrúbky aj náhodná úsečka.

V programovacom jazyku pascal musíme špeciálne označiť **blok príkazov**, ktorý chceme, aby sa celý opakoval v príkaze for-cyklu. Začiatok bloku príkazov zapisujeme rezervovaným slovom **begin** a koniec bloku rezervovaným slovom **end**. Ukážme, ako by mal teraz vyzerat' opravený program so 100 rôzne hrubými čiarami:

```
var
  I: Integer;
begin
  for I := 1 to 100 do
    begin
      Image1.Canvas.Pen.Width := Random(10) + 1;
      Image1.Canvas.LineTo(Random(350), Random(250));
    end;
  end;
```

For-cyklus teraz funguje takto: 100-krát vykoná jeden príkaz, ktorý je za rezervovaným slovom **do**, ale keďže je to **blok príkazov**, tak sa postupne vykonajú všetky príkazy bloku. Ďalší program ukazuje len malú zmenu predchádzajúceho:

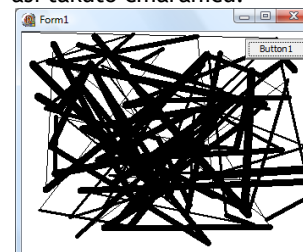
```
var
  I: Integer;
begin
  Image1.Canvas.Pen.Width := Random(10) + 1;
  for I := 1 to 100 do
    begin
      Image1.Canvas.Pen.Color := RGB(0, 0, Random(128) + 128);
      Image1.Canvas.LineTo(Random(350), Random(250));
    end;
  end;
```

Program najprv náhodne zvolí hrúbku a potom nakreslí 100 takto hrubých náhodných čiar, pričom všetky budú nejakého odtieňa modrej farby.

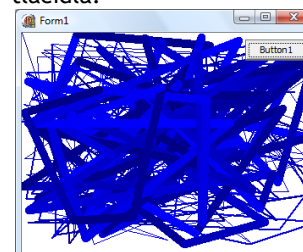
Teraz sa pozrieme na to, ako sme naše programy formátovali:

- do jedného riadka zapisujeme len jeden príkaz
- aj slová **begin** a **end** zapisujeme do samostatných riadkov

Keby bol program bez chyby, mohli by sme dostať asi takúto čmáranicu:



Po niekoľkých zatlačeniach tlačidla:



- všetky príkazy medzi **begin** a **end** sú o 2 medzery odsunuté oproti **begin** (príslušné **end** začína v tom istom stĺpci ako jeho **end**)
- vnorený príkaz (nie blok príkazov) vo for-cykle je tiež odsunutý o dve medzery
- na zvýšenie čitateľnosti používame medzery vo vnútri príkazov: medzi operácie, za čiarky, za dvojbodku, okolo dvojznaku priradenia :=
- všetky rezervované slová (**begin**, **end**, **for**, **to**, **do**, **var**) píšeme malými písmenami
- všetky identifikátory začínajú veľkým písmenom a ďalej spravidla nasledujú malé písmená, výnimkou sú veľké vo vnútri identifikátora, ak sa ním začína nejaké slovo, napr. **MoveTo**, **LineTo**, **RGB** (skratky troch slov)
- niektoré preddefinované konštanty začínajú dvoma malými písmenami, za ktorým už ide veľké (**clRed**, **bsClear**, **fsBold**)

Kompilátor pascalu takéto formátovanie nepotrebuje, on by zvládol aj takto naformátovaný posledný program:

```
procedure TForm1.button1click(sender: TObject);
var i: Integer; begin image1.canvas.pen.width:=
rANdOm(10)+1; FOR i:=1 to 100 do begin image1.
canvas.pen.coLOR:=rgb(0,0,rANdOm(128)+128); image1.
canvas.lINeto(rANdOm(350),rANdOm(250)); end; end;
```

A toto by sa nám naozaj zle čítalo. Zapamätajte si: programy neformátujeme pre počítač, ale pre človeka - pre seba a pre ďalších programátorov, ktorí budú tento program ešte čítať. Niekedy sa najmä začiatočník dost' natrápi, aby bol program správne naformátovaný. Ale program väčšinou píšeme len raz (len raz sa natrápime s formátovaním), a čítame veľa krát.

Dôležité je odsúvanie začiatkov riadkov. Všimnite si, že prvé **begin** a posledné **end** jediné začínajú v prvom stĺpci (okrem deklarácií). Všetky príkazy, ktoré sú "**vnorené**" medzi **begin** a **end** sú odsunuté o dve medzery (začínajú v treťom stĺpci). Kým sme nemali for-cykly, všetko sme naozaj písali od 3. stĺpca. Vnorené príkazy vo for-cykle sú opäť odsunuté o 2 medzery. Neskôr uvidíme vnorené príkazy aj v príkazoch vo vnútri for-cyklu - aj tieto budú odsunuté o 2 medzery.

Pozrite a zhodnot'te formátovanie časti programov:

for I := 1 to 10 do begin X := 5; Y := 7; end;	for I := 1 to 10 do begin X := 5; Y := 7; end;	for I := 1 to 10 do begin X := 5; Y := 7; end;
for I := 1 to 10 do begin X := 5; Y := 7; end;	for I := 1 to 10 do begin X := 5; Y := 7; end;	for I := 1 to 10 do X := 5; Y := 7;

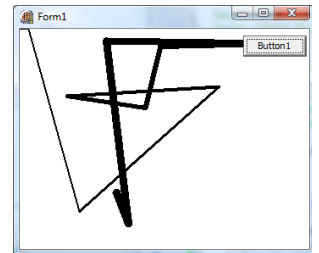
Úlohy na precvičenie

Zadanie 1	program 25-krát striedavo na náhodných pozíciách vypíše modrý text "Pascal" a červený text "Delphi"
Zadanie 2	program na náhodné pozície vypíše 100 textov "Ahoj" pričom sa budú striedať 4 farby podkladu textu, napr. červená, zelená, žltá a modrá
Zadanie 3	program nakreslí 1000 náhodných tenkých čiar, pričom im bude náhodne nastavovať aj RGB farbu, kde červená zložka bude 0, zelená náhodne od 128 do 255 a modrá tiež náhodne od 128 do 255

3. Využitie počítadla v tele cyklu

Zamerajme sa teraz na počítadlo cyklu, t.j. na premennú, ktorá počas behu cyklu postupne nadobúda hodnoty od dolnej hranice až po hornú. Túto premennú môžeme v cykle veľmi užitočne používať. Napr. nasledujúci cyklus nakreslí 10 náhodných čiar, pričom prvá má hrúbku pera 1, druhá 2, tretia 3, atď. až posledná 10. čiara má hrúbku pera 10:

```
var
  I: Integer;
begin
  for I := 1 to 10 do
  begin
    Image1.Canvas.Pen.Width := I;
    Image1.Canvas.LineTo(Random(350), Random(250));
  end;
end;
```

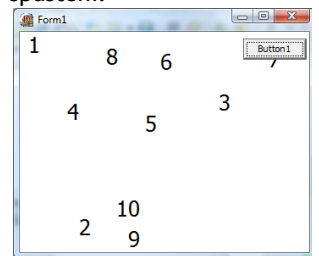


Vo vnútri cyklu môžeme používať hodnotu počítadla (môžeme ju priradiť, môže byť v nejakom výraze, môže byť parametrom nejakého príkazu), ale **nesmieme** do tejto premennej v tele cyklu nič priradovať - o to sa stará počítač.

Ďalší program vypíše čísla 1 až 10 na náhodné pozície:

```
var
  I: Integer;
begin
  Image1.Canvas.Font.Height := 30;
  for I := 1 to 10 do
    Image1.Canvas.TextOut(Random(350), Random(250), IntToStr(I));
  end;
```

A takto to vyzerá po spustení:



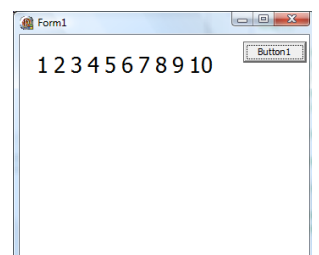
Program vylepšíme: čísla nebudeme vypisovať na náhodné pozície, ale vedľa seba do jedného riadka:

```
var
  I: Integer;
begin
  Image1.Canvas.Font.Height := 30;
  for I := 1 to 10 do
    Image1.Canvas.TextOut(I*20, 20, IntToStr(I));
  end;
```

X-ovú súradnicu vypisovaného čísla musíme zväčšovať napr. o 20, aby sa výpis neprekrýval. Preto bude počítač vypisovať čísla takto:

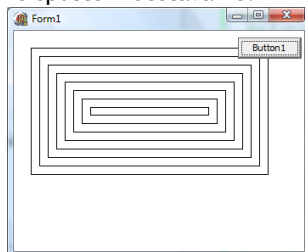
```
Image1.Canvas.TextOut(20, 20, '1');
Image1.Canvas.TextOut(40, 20, '2');
Image1.Canvas.TextOut(60, 20, '3');
Image1.Canvas.TextOut(80, 20, '4');
Image1.Canvas.TextOut(100, 20, '5');
Image1.Canvas.TextOut(120, 20, '6');
Image1.Canvas.TextOut(140, 20, '7');
Image1.Canvas.TextOut(160, 20, '8');
Image1.Canvas.TextOut(180, 20, '9');
Image1.Canvas.TextOut(200, 20, '10');
```

A ako to vyzerá po spustení vidíme na okraji stránky.



Pamätáte, kým sme nepoznali cyklus, museli sme to robiť presne takto? Podobnú metódu môžeme použiť aj vtedy, keď chceme zostaviť cyklus, v ktorom využijeme počítadlo a nevieme zostaviť aritmetické výrazy, ktoré bude treba zapísať. Napr. úlohou je nakresliť 8 sústredných obdĺžnikov: najväčší má ľavý horný roh v 20, 20 a jeho veľkosť je 280x150; každý ďalší o 20x20 menší a je posunutý o 10 vpravo a 10 dole. Bez cyklu by sme to zapísali takto:

Po spustení dostávame:



```
Image1.Canvas.Rectangle(20, 20, 300, 170);  
Image1.Canvas.Rectangle(30, 30, 290, 160);  
Image1.Canvas.Rectangle(40, 40, 280, 150);  
Image1.Canvas.Rectangle(50, 50, 270, 140);  
Image1.Canvas.Rectangle(60, 60, 260, 130);  
Image1.Canvas.Rectangle(70, 70, 250, 120);  
Image1.Canvas.Rectangle(80, 80, 240, 110);  
Image1.Canvas.Rectangle(90, 90, 230, 100);
```

My to samozrejme chceme zapísať pomocou for-cyklu:

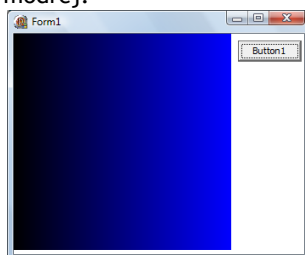
```
var  
  I: Integer;  
begin  
  for I := 1 to 8 do  
    Image1.Canvas.Rectangle(10+10*I, 10+10*I, 310-10*I, 180-10*I);  
  end;
```

V ďalšom príklade poexperimentujeme s farbami. Najprv nakreslíme husto vedľa seba 256 modrých úsečiek:

```
var  
  I: Integer;  
begin  
  Image1.Canvas.Pen.Color := RGB(0, 0, 255);  
  for I := 0 to 255 do  
    begin  
      Image1.Canvas.MoveTo(I, 0);  
      Image1.Canvas.LineTo(I, 255);  
    end;  
  end;
```

Modrú farbu sme tu zapísali ako RGB(0, 0, 255). Ak by sme menili poslednú zložku RGB, dostávali by sme rôzne odtiene modrej. Presunieme nastavovanie farby do cyklu a každý prechod

Program teraz nakreslí 256 úsečiek, pričom každá bude zafarbená iným odtieňom modrej:

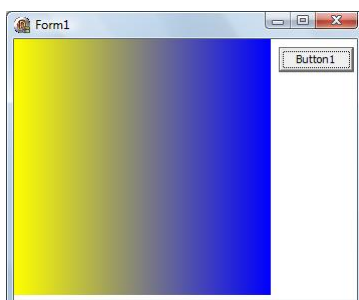


```
var  
  I: Integer;  
begin  
  for I := 0 to 255 do  
    begin  
      Image1.Canvas.Pen.Color := RGB(0, 0, I);  
      Image1.Canvas.MoveTo(I, 0);  
      Image1.Canvas.LineTo(I, 255);  
    end;  
  end;
```

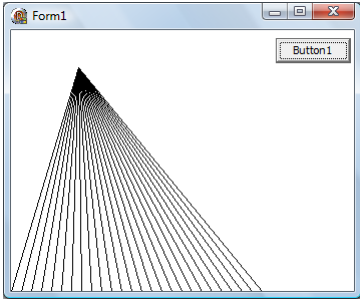
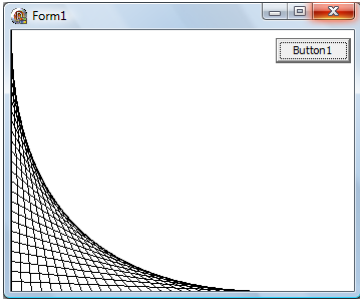
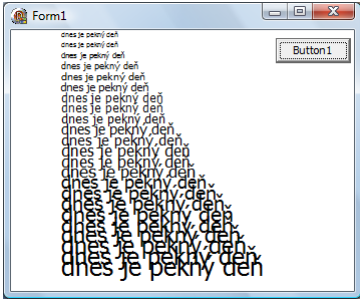
Poexperimentujte s týmto programom, kde môžete meniť parametre pre RGB, napr. takto

```
Image1.Canvas.Pen.Color := RGB(255-I, 0, I);  
Image1.Canvas.Pen.Color := RGB(0, 255-I, I);  
Image1.Canvas.Pen.Color := RGB(160, 255-I, I);  
Image1.Canvas.Pen.Color := RGB(255-I, 255-I, I);
```

V týchto a podobných príkladoch farby prechádzajú postupne od jednej do druhej:



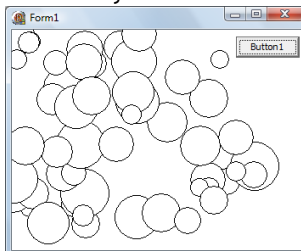
Úlohy na precvičenie

Zadanie 1	vypíšte postupne pod seba čísla od 11 do 30
Zadanie 2	vypíšte postupne pod seba čísla od 11 do 30 a ku každému pripíšte jeho druhú mocninu
Zadanie 3	<p>najprv náhodne vygeneruje bod X, Y a potom ho postupne bude spájať s bodmi na spodnej hrane plochy, napr. (0, 300), (10, 300), (20, 300), ...</p> 
Zadanie 4	vyšrafuje štvorec vodorovnými úsečkami, medzi ktorými je vzdialenosť 10
Zadanie 5	<p>program kreslí úsečky, v ktorých sa prvý bod pohybuje po zvislej priamke (sú to body (0, 0), (0, 10), (0, 20), ...) a druhý bod po vodorovnej na spodnom okraji plochy (napr. body (0, 300), (10, 300), (20, 300), ...)</p> 
Zadanie 6	<p>postupne sa zväčšujúci text pre veľkosti od 8 do 30, zároveň sa mení aj Y-ová súradnica výpisu 0, 10, 20, 30, ... (môžete vypnúť vyplňanie plochy pod textom pomocou <code>Brush.Style := bsClear</code>)</p> 

4. Pomocná premenná pre cyklus

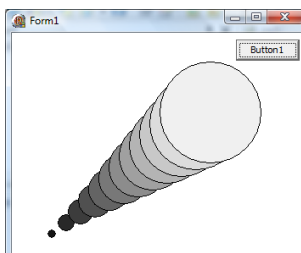
V tele cyklu okrem premennej počítadla môžeme používať aj ďalšie premenné. Napr.

nakreslí 50 náhodne veľkých náhodne umiestnených kruhov:



```
var
  I, X, Y, R: Integer;
begin
  for I := 1 to 50 do
  begin
    X := Random(300);
    Y := Random(250);
    R := Random(20) + 10;
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
  end;
end;
```

Tu si všimnime, že pomocné premenné **X**, **Y** a **R** sa vždy vypočítajú nanovo a nezávisia od počítadla cyklu. Často sa ale do premenných v cykle počítajú nové hodnoty na základe počítadla, napr. zmeňme predchádzajúci program:



```
var
  I, X, Y, R: Integer;
begin
  for I := 1 to 12 do
  begin
    X := 30 + 17 * I;
    Y := 250 - 13 * I;
    R := 5 * I;
    Image1.Canvas.Brush.Color := RGB(20*I, 20*I, 20*I);
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
  end;
end;
```

Každá pomocná premenná a aj farba štetca sa prepočítajú znovu a znovu na základe hodnoty počítadla. Dostávame 12 postupne sa zväčšujúcich kruhov stále svetlejších odtieňov šedej.

V cykloch ale môžeme premennú nielen stále počítat' odznovu, ale môžeme využiť jej predchádzajúcu hodnotu (z predchádzajúceho prechodu cyklu) a túto len opraviť. V prechádzajúcom príklade sa X-ová súradnica zväčšuje o 17, Y-ová zmenšuje o 13 a polomer zväčšuje o 5. V takomto prípade ale nesmieme zabudnúť nastaviť počiatočné hodnoty týmto premenným (inicializovať) ešte pred cyklom. Môžeme to prepísať takto:

```
var
  I, X, Y, R: Integer;
begin
  X := 47;
  Y := 237;
  R := 5;
  for I := 1 to 12 do
  begin
    Image1.Canvas.Brush.Color := RGB(20*I, 20*I, 20*I);
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
    X := X + 17;
    Y := Y - 13;
    R := R + 5;
  end;
end;
```

Tento program teraz robí presne to isté, ako predchádzajúci. Všimnite si, že vo vnútri cyklu sme najprv hodnoty **X**, **Y** a **R** použili v procedúre **Ellipse** a na konci cyklu sme im nastavili nové hodnoty pre ďalší prechod cyklom. Trochu inak by to muselo vyzerat', keby sme chceli vo vnútri cyklu najprv urobiť 3 priradenia do **X**, **Y** a **R** a až potom zavolať **Ellipse**: museli by sa zmeniť priradenia pred cyklom. Premyslite to. Prekontrolovať takéto riešenia môžete pomocou trasovacích tabuliek priebehu

výpočtu.

V ďalšej úlohe budeme vedľa seba do jedného radu kresliť 7 štvorcov: všetky sú položené na tej istej Y-ovej súradnici, napr. 200; veľkosti strán štvorcov sú postupne 10, 20, 30, 40, ... pričom X-ová súradnica ľavej strany sa posúva tak, aby boli na seba nalepené. Použijeme 3 pomocné premenné X, Y a Strana, kde X a Y sú súradnice ľavého dolného vrcholu a Strana je veľkosť strany. Môžeme zapísať program:

```
var
  I, X, Y, Strana: Integer;
begin
  X := 5;
  Y := 200;
  for I := 1 to 7 do
  begin
    Strana := 10 * I;
    Image1.Canvas.Rectangle(X, Y, X + Strana, Y - Strana);
    X := X + Strana;
  end;
end;
```

Tento istý program by sa dal zapísať aj bez použitia premenných Y a Strana, ale teraz môže byť tento program pre niekoho menej čitateľný:

```
var
  I, X: Integer;
begin
  X := 5;
  for I := 1 to 7 do
  begin
    Image1.Canvas.Rectangle(X, 200, X + 10 * I, 200 - 10 * I);
    X := X + 10 * I;
  end;
end;
```

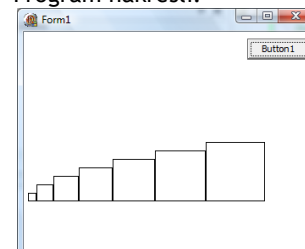
Tento mechanizmus, keď v cykle k nejakej premennej pripočítavame nejaké hodnoty, môžeme využiť aj na úlohy, v ktorých sa nekreslí, ale počíta. Potrebujeme spočítať rad čísel od 1 do 100. Matematici na to poznajú aj vzorec (súčet radu od 1 do N je $N(N+1)/2$) a my ho použijeme na prekontrolovanie správnosti výsledku:

```
const
  N = 100;
var
  I, Sucet: Integer;
begin
  Sucet := 0;
  for I := 1 to N do
    Sucet := Sucet + I;
  Image1.Canvas.Font.Height := 40;
  Image1.Canvas.TextOut(50, 50, IntToStr(Sucet));
  Image1.Canvas.TextOut(50, 150, IntToStr(N*(N+1) div 2));
end;
```

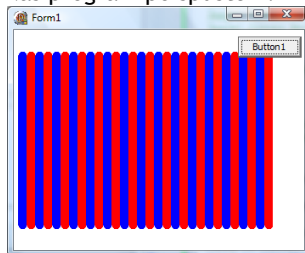
Po spustení dostávame ten istý výsledok **5050** v oboch prípadoch.

Pomocnú premennú využijeme aj vtedy, keď potrebujeme v cykle striedať nejaké dve farby, napr. modrú a červenú. Nakreslíme vedľa seba 30 hrubších farebných úsečiek a pritom použijeme pomocnú premennú Farba, v ktorej sa bude striedať modrá a červená. Najprv do nej teda priradíme modrú farbu (clBlue) a na konci cyklu ju zmeníme na tú druhú

Program nakreslí:



Náš program po spustení:



```
var
  I, X, Farba: Integer;
begin
  Image1.Canvas.Pen.Width := 10;
  Farba := clBlue;
  for I := 1 to 30 do
  begin
    X := 10 * I;
    Image1.Canvas.Pen.Color := Farba;
    Image1.Canvas.MoveTo(X, 30);
    Image1.Canvas.LineTo(X, 230);
    Farba := clBlue + clRed - Farba;
  end;
end;
```

Využili sme tú vlastnosť, že farby sú v počítači reprezentované ako celé čísla (fungovalo napr. priradenie `Random(256*256*256)`), a preto s nimi môžeme robiť aj celočíselnú aritmetiku.

Podobne, ak by sme napr. potrebovali striedať 2 hrúbky pera, napr. 3 a 7, do premennej `Hrubka` by sme priradili 3 a potom v cykle `Hrubka := 3 + 7 - Hrubka`.

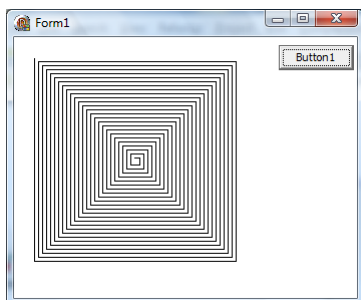
Tieto úlohy by sa dali riešiť aj tak, že by cyklus nebežal 30-krát pre 30 úsečiek, ale iba 15-krát a v cykle by sa nakreslila jedna dvojica úsečiek: modrá a červená. Premyslite to. Ako to treba spraviť, keď potrebujeme nakresliť nepárny počet úsečiek?

Úlohy na precvičenie

Zadanie 1	nakresliť rad 10 elíps s poloosami 15 a 7, pričom všetky majú rovnakú náhodnú farbu
Zadanie 2	nakresliť 10 kruhov, ktoré majú polomery 10, 20, 30, ..., všetky prechádzajú bodom (250, 250) a ich stred leží na zvislej priamke prechádzajúcej týmto bodom nad týmto bodom - použite nastavenie <code>Brush.Style := sbClear</code>
Zadanie 3	pomocou cyklu vypočítajte faktoriál nejakého konkrétneho čísla ≤ 12
Zadanie 4	pomocou cyklu vypočítajte K -tu mocninu čísla N (kde N a K sú konštanty), napr. 3^{10}
Zadanie 5	pomocou cyklu vypočítajte kombinačné číslo N nad K - môžeme ho počítať ako súčin čísel od $K+1$ do N , ktorý vydělíme číslami od 2 do K - vypočítajte pomocou dvoch cyklov napr. 10 nad 6
Zadanie 6	každé zatlačenie tlačidla na náhodnej pozícii nakreslí rad 10 náhodne zafarbených kruhov s polomerom 10 (ako treba zmeniť program, aby rad nebol vodorovný ale zvislý?)

Zadanie 7

nakresliť štvorcovú špirálu, ktorá začína v strede grafickej plochy: v cykle sa bude opakovať kreslenie štyroch na seba kolmých úsečiek: každá bude od predchádzajúcej o kúsok dlhšia, otočená o 90 stupňov napr. vpravo - okrem X , Y pre ľavý horný roh štvorice si treba v cykle pamätať (a zvyšovať) premennú **Dĺzka**:



5. Prevrátený cyklus

Už sme sa viackrát stretli s problémom kreslenia nepriehľadných sústredných kruhov: ak ich kreslíme od najmenšieho po najväčší, tak každý väčší zakryje všetky doteraz nakreslené. Napr.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y, R: Integer;
begin
  X := Random(250) + 50;
  Y := Random(150) + 50;
  for I := 1 to 10 do
  begin
    R := 5 * I;
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
  end;
end;
```

Hoci program kreslí 10 kruhov, na záver bude vidieť len ten najväčší z nich. Už vieme urobiť kruhy priesvitné pomocou **Brush.Style := bsClear**, ale farebné kruhy zrejme treba robiť inak: začneme od najväčšieho a polomer postupne zmeňujeme. Hoci by sme to vedeli urobiť pomocou zmeneného vzorca pre výpočet polomeru R , ukážeme na tomto príklade inú verziu for-cyklu. Tento druhý variant sa líši od prvého tým, že namiesto rezervovaného slova **to** má slovo **downto** a počítadlo sa nebude zakaždým zvyšovať o 1, ale znižovať. Vo všeobecnosti

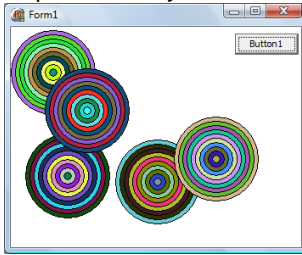
```
for premenná := horná_hranica downto dolná_hranica do
  príkaz;
```

Pre tento variant for-cyklu platia rovnaké, resp. analogické pravidlá ako pre bežný for-cyklus:

- hodnota premennej (počítadla) sa nesmie v tele cyklu meniť
- ak je horná hranica menšia ako dolná hranica, telo cyklu sa nevykoná ani raz

Zapišme predchádzajúci príklad pomocou prevráteného for-cyklu:

Po niekoľkých zatlačeniach tlačidla dostávame niekoľko skupín sústredných kružníc:

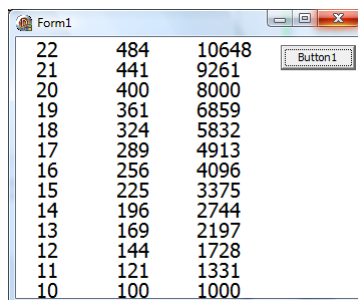


```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, X, Y, R: Integer;
begin
  X := Random(250) + 50;
  Y := Random(150) + 50;
  for I := 10 downto 1 do
  begin
    R := 5 * I;
    Image1.Canvas.Brush.Color := Random(256 * 256 * 256);
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
  end;
end;
```

Ďalší program vypíše pod seba tabuľku druhých a tretích mocnín čísel od 22 do 10:

```
var
  I, Y: Integer;
begin
  Y := 0;
  Image1.Canvas.Font.Height := 24;
  for I := 22 downto 10 do
  begin
    Image1.Canvas.TextOut(20, Y, IntToStr(I));
    Image1.Canvas.TextOut(100, Y, IntToStr(I * I));
    Image1.Canvas.TextOut(180, Y, IntToStr(I * I * I));
    Y := Y + 20;
  end;
end;
```

Po zatlačení tlačidla dostávame:



V skutočnosti sa tento variant for-cyklu využíva veľmi zriedka, lebo sa dá nahradiť bežným for-cyklom, len pritom niekedy treba prepísať nejaké aritmetické výrazy. Viac využijeme tento variant for-cyklu, až keď budeme pracovať s dátovou štruktúrou pole.

Úlohy na precvičenie

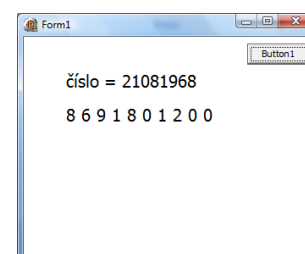
Zadanie 1	oba programy v tejto časti prepíšete s použitím obyčajného cyklu
Zadanie 2	nakreslite rad štvorcov so stranami 100, 90, 80, ..., 10, ktorých stredy ležia na vodorovnej priamke, štvorce sú pritom tesne vedľa seba
Zadanie 3	nakreslite 20 sústredných kruhov, pre ktoré sa strieda farba ich výplne, napr. žltá a zelená

6. Cifry celého čísla

Už sme naučili rozoberať celé číslo na cifry: pomocou **Cislo mod 10** získavame poslednú cifru nejakého čísla a pomocou **Cislo div 10** takýto desiatkový zápis skrátíme o poslednú cifru. Teraz môžeme v tomto procese pokračovať: z čísla, z ktorého sme odhodili poslednú cifru, zoberieme opäť poslednú (teda predposlednú z pôvodného čísla) a číslo skrátíme o túto poslednú cifru. Opäť v tomto pokračujeme až kým neprejdeme všetky cifry. Keďže najväčšie celé číslo môže byť okolo 2 miliárd, toto rozoberanie stačí robiť maximálne 10 krát. Zapišme

```
var
  I, X, Cislo, Cifra: Integer;
begin
  X := 50;
  Cislo := 21081968;
  Image1.Canvas.Font.Height := 24;
  Image1.Canvas.TextOut(50, 40, 'číslo = ' + IntToStr(Cislo));
  for I := 1 to 10 do
  begin
    Cifra := Cislo mod 10;
    Cislo := Cislo div 10;
    Image1.Canvas.TextOut(X, 80, IntToStr(Cifra));
    X := X + 18;
  end;
end;
```

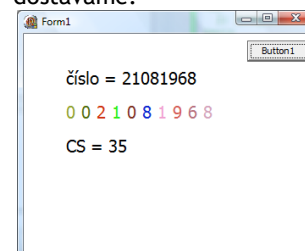
Keďže číslo bolo 8-ciferné a cyklus bežal až 10-krát, pri posledných dvoch prechodoch cyklu bola premenná **Cislo** už nulová a teda 2-krát na konci dostávame nulové cifry (obrázok na okraji stránky).



Program postupne vypísal všetky cifry do jedného radu od poslednej až po prvú, pričom ich doplnil nulami do 10 cifier. Zatiaľ nevieme programom zistiť, koľko cifier má zadané číslo a teda koľkokrát by mal cyklus na rozoberanie na cifry naozaj bežať (neskôr sa naučíme na toto vhodnejší *while*-cyklus). Tiež vidíme, že v táto vypísaná postupnosť cifier je v opačnom poradí. My to teraz vypíšeme od konca, teda dostaneme normálne poradie cifier, pričom ich náhodne zafarbíme. Okrem toho do premennej **Sucet** postupne pripočítame všetky tieto cifry a dostaneme tzv. **ciferný súčet**:

```
var
  I, X, Cislo, Cifra, Sucet: Integer;
begin
  X := 212;
  Cislo := 21081968;
  Image1.Canvas.Font.Height := 24;
  Image1.Canvas.TextOut(50, 40, 'číslo = ' + IntToStr(Cislo));
  Sucet := 0;
  for I := 1 to 10 do
  begin
    Cifra := Cislo mod 10;
    Cislo := Cislo div 10;
    Sucet := Sucet + Cifra;
    Image1.Canvas.Font.Color := Random(256 * 256 * 256);
    Image1.Canvas.TextOut(X, 80, IntToStr(Cifra));
    X := X - 18;
  end;
  Image1.Canvas.Font.Color := clBlack;
  Image1.Canvas.TextOut(50, 120, 'CS = ' + IntToStr(Sucet));
end;
```

Po spustení programu dostávame:



Keďže do premennej **Sucet** v cykle niečo pripočítavame, nesmieme ju zabudnúť ešte pred cyklom vynulovať.

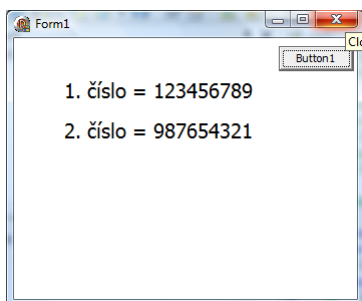
Podobným spôsobom ako vieme číslo rozoberať na cifry môžeme číslo z cifier skladat: ak číslo vynásobíme 10, vlastne mu na koniec prilepíme cifru 0, ak mu teraz pripočítame nejakú **cifru**, dostávame pôvodné číslo, ktorému sme na koniec prilepili novú cifru: **Cislo * 10 + Cifra**. Nasledujúci program poskladá číslo najprv z cifier 1 až 9 a potom z cifier 9 až 1:

```

var
  I, X, Cislo, Cifra, Sucet: Integer;
begin
  Image1.Canvas.Font.Height := 24;
  Cislo := 0;
  for I := 1 to 9 do
    Cislo := Cislo * 10 + I;
  Image1.Canvas.TextOut(50, 40, '1. číslo = ' + IntToStr(Cislo));
  Cislo := 0;
  for I := 9 downto 1 do
    Cislo := Cislo * 10 + I;
  Image1.Canvas.TextOut(50, 80, '2. číslo = ' + IntToStr(Cislo));
end;

```

a dostávame:



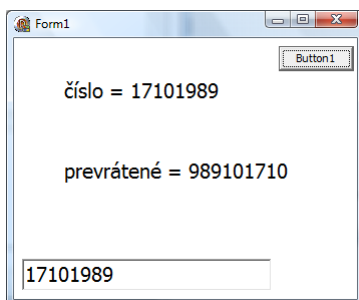
V poslednom príklade využijeme vstupný riadok **TEdit**. Do formulára pridáme tento komponent a zatlačenie tlačidla najprv zoberie hodnotu z tohto vstupu (**Edit1.Text**), prevedie tento text na celé číslo (pomocou funkcie **StrToInt**) a potom s týmto číslom spraví nejaký úkon - konkrétne vytvorí nové číslo, ktoré bude mať prevrátené poradie cifier:

```

procedure TForm1.Button1Click(Sender: TObject);
var
  I, Cislo, Prevratene: Integer;
begin
  Cislo := StrToInt(Edit1.Text);
  Image1.Canvas.Font.Height := 24;
  Image1.Canvas.TextOut(50, 40, 'číslo = ' + IntToStr(Cislo));
  Prevratene := 0;
  for I := 1 to 9 do
    begin
      Prevratene := Prevratene * 10 + Cislo mod 10;
      Cislo := Cislo div 10;
    end;
  Image1.Canvas.TextOut(50, 120, 'prevrátené = '+IntToStr(Prevratene));
end;

```

Po spustení programu najprv do vstupného riadka zadáme nejaké číslo a potom stlačíme tlačidlo:



Už by sme mali rozumieť, prečo sa v našom prípade na konci prevráteného čísla objavila 0. Mohol by vzniknúť nejaký problém, ak by for-cykľus nebol len do 9, ale až do 10?

Úlohy na precvičenie

Zadanie 1	pre losovanie lotérie potrebujeme vygenerovať náhodné 6-ciferné číslo, ktoré nebude obsahovať 0 (generujte po cifrách)
Zadanie 2	pre zadané číslo N (zo vstupného riadka) vygenerujete N-ciferné číslo, ktoré je zložené napr. zo samých 7
Zadanie 3	pre zadané číslo zo vstupného riadka vypočítajte jeho ciferný súčet
Zadanie 4	program prečíta z dvoch vstupných riadkov 2 štvorciferné čísla, vypíše ich a vytvorí nové číslo, ktoré vznikne zlepením prvého a prevráteného druhého (napr. 1234 a 5678 vytvorí 12348765)
Zadanie 5	zadané 10-ciferné číslo zo vstupného riadka vypíše ciframi pod seba, pričom sa pri vypisovaní čísel budú striedať modré a červené číslice

Zoznam rezervovaných slov

Tieto pre Pascal rezervované slová sa nesmú vyskytnúť ako identifikátory konštánt, premenných a pod.

and	array	as	asm
begin	case	class	const
constructor	destructor	dispinterface	div
do	downto	else	end
except	exports	file	final
finalization	finally	for	function
goto	if	implementation	in
inherited	initialization	inline	interface
is	label	library	mod
nil	not	object	of
or	out	packed	procedure
program	property	raise	record
repeat	resourcestring	sealed	set
shl	shr	static	string
then	threadvar	to	try
type	unit	unsafe	until
uses	var	while	with
xor			

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil čitateľa s týmito základnými stavebnými kameňmi programovacieho jazyka:

- premenná, typ premennej, celočíselný typ, deklarácie
- priradovací príkaz, aritmetický výraz, priorita operácií, vstupné hodnoty
- cyklus s pevným počtom opakovaní

Okrem toho boli uvedené tieto dôležité koncepty:

- generátor náhodných čísel, rozoberanie čísla na cifry
- farebný model RGB a využitie v programovaní

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Pre zadané číslo **N** vo vstupnom riadku (**Edit1**) zatlačenie tlačidla nakreslí rad **N** štvorcov veľkosti **AxA**, pričom v grafickej ploche bude tento rad vycentrován. V programe použite konštantu **A** - veľkosť štvorca.

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. Lubomír 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 Skalka, PhD.

Počet strán 40

Náklad 300 ks

Prvé vydanie, Bratislava 2009

Všetky práva vyhradené.

Toto dielo ani žiadnu jeho časť nemožno reprodukovat' bez súhlasu majiteľa práv.

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