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 3

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 3

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

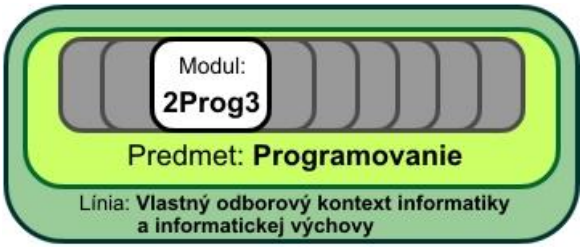
Autori:

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

Rukopis odovzdaný:

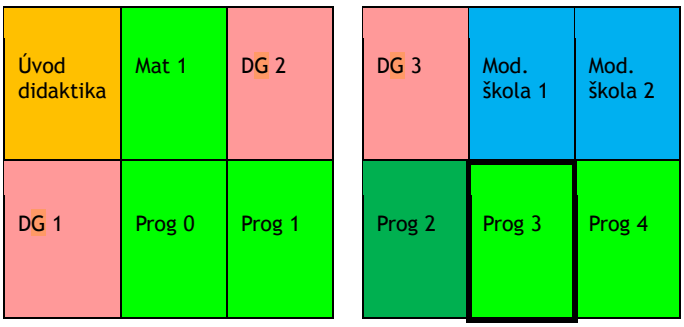
14. august 2009

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: 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 podmieneným príkazom v pascale: vetveniu programov na základe nejakej podmienky a cyklus (**while**) s podmienkou na začiatku. Zoznámi s logickými výrazmi a tiež s logickým typom.

Obsah

Úvod

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

1. vetvenie
2. cyklus s podmienkou

Cieľ modulu

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

1. podmienený príkaz (if)
2. viac príkazov vo vetve
3. len jedna vetva príkazu
4. vnorené podmienené príkazy (if)
5. zostavovanie zložených podmienok (logické operácie)
6. zložený podmienený príkaz (case)

Druhá tematická jednotka **Cyklus s podmienkou** pokrýva tieto témy:

1. cyklus s jednoduchými podmienkami
2. znovu cifry celého čísla
3. cyklus so zložitejšími podmienkami
4. zapamätanie si výsledku podmienky

Vstupné vedomosti

Požadované prerekvizity

Modul Programovanie 2 – premenné a for-cyklus.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie vysvetliť pojem premenná a popísať princíp fungovania for-cyklu,
- vie používať premenné a priraďovací príkaz,
- dokáže analyzovať zadanie a tiež možné chyby v riešení, chyby vie nájsť a opraviť,
- dokáže vytvoriť aplikácie, ktoré používajú jednoduché premenné a cykly s pevným počtom opakovaní,
- rozumie pojmu premenná, generátoru náhodných čísel a tiež princípu fungovania cyklu.

Preverenie vstupných vedomostí

Vytvorenie jednoduchej aplikácie so vstupným riadkom a s jedným tlačidlom, ktorá nakreslí rad opakujúcich sa kružníc, pričom ich počet je zadaný na vstupnom riadku.

1. tematická jednotka - Vetvenie

1. Podmienený príkaz (if)

Pri programovaní budeme často riešiť situácie, keď sa niektoré príkazy budú vykonávať len pri splnení istých podmienok. Napr. deliť čísla **A** a **B** budeme len vtedy, keď **B** ≠ 0, alebo v cykle budeme stále zväčšovať nejakú hodnotu, ale keď presiahne napr. 100, budeme to špeciálne riešiť. Už sme v minulom module mali situácie, keď sa farby štvorcov v rade vyfarbovali na striedačku. Všetky tieto situácie potrebujú nejaký spôsob otestovania podmienky.

Podmienkou budeme rozumieť taký výraz, ktorého výsledkom je buď pravda alebo nepravda. Podmienka je splnená, ak nadobúda hodnotu pravda, inak je podmienka nesplnená. Veľmi často to budú podmienky, v ktorých sa nejaké hodnoty navzájom porovnávajú, napr. na rovnosť, alebo ktorá hodnota je väčšia.

V Pascale na porovnávanie hodnôt máme k dispozícii tieto tzv. relačné operátory:

=	rovnaké hodnoty - rovnosť
<>	rôzne hodnoty - nerovnosť
<	prvá hodnota je menšia ako druhá
<=	prvá hodnota je menšia alebo rovná druhej
>	prvá hodnota je väčšia ako druhá
>=	prvá hodnota je väčšia alebo rovná druhej

Takto vyzerajú zápisy niekoľkých jednoduchých podmienok:

```
X > 100
I mod 5 = 0
I div 10 >= 1
A > B
Random(6) = 1
Abs(X) < 0
1 + 1 = 2
```

Výsledok podmienky väčšinou závisí od nejakých premenných (napr. **X** > 100), ale niektoré podmienky sú vždy nepravdivé (napr **Abs(X)** < 0), iné sú zas pravdivé vždy (napr. **X** < **X** + 1).

Teraz sa naučíme nový príkaz, tzv. **príkaz vetvenia** (hovoríme mu aj **podmienený príkaz**). Jeho zápis je nasledovný:

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

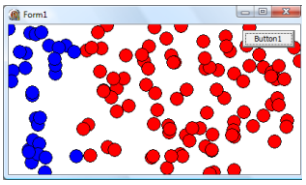
Podobne ako for-cyklus, aj tento príkaz v sebe obsahuje nejaké iné príkazy a v tomto prípade sú dva: ***príkaz1*** a ***príkaz2***. V*norený* príkaz ***príkaz1*** sa vykoná jedine vtedy, keď výsledok podmienky je pravdivý inak sa vykoná len v*norený* príkaz ***príkaz2***. Po slovensky by sme ho mohli povedať:

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

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

Prvý program, v ktorom použijeme podmienený príkaz, po zatlačení tlačidla **Button1** vygeneruje **X**, **Y** na náhodných pozíciách a ak bude **X** menšie ako 100 nakreslí modrý krúžok s polomerom 10 inak bude červený:

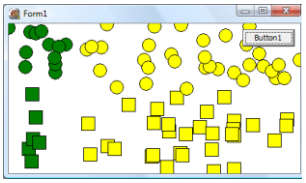
```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  if X < 100 then
    Image1.Canvas.Brush.Color := clBlue
  else
    Image1.Canvas.Brush.Color := clRed;
  Image1.Canvas.Ellipse(X - 10, Y - 10, X + 10, Y + 10);
end;
```



Po spustení a viacnásobnom zatlačení tlačidla môžeme vidieť, že všetky krúžky naľavo sú modré a všetky napravo sú červené:

Veľmi podobný bude druhý program, ktorý bude kresliť buď krúžky, alebo štvorčeky a to podľa toho, či náhodne vygenerované súradnice **X**, **Y** majú **Y < 100** alebo nie, zároveň sa podľa X-ovej súradnice nastavuje zelená, resp. žltá farba tak, že ak **X < 100** útvar (krúžok alebo štvorček) bude zelený, inak bude žltý:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  if X < 100 then
    Image1.Canvas.Brush.Color := clGreen
  else
    Image1.Canvas.Brush.Color := clYellow;
  if Y < 100 then
    Image1.Canvas.Ellipse(X - 10, Y - 10, X + 10, Y + 10)
  else
    Image1.Canvas.Rectangle(X - 10, Y - 10, X + 10, Y + 10);
end;
```



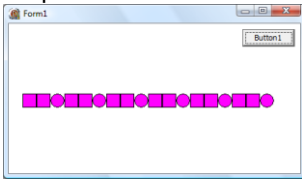
Vidíme, že oba príkazy vetvenia sme zaradili za seba: najprv sa nastavila farba výplne (**Brush.Color**) a druhým príkazom vetvenia sa kreslil buď krúžok alebo štvorček:

Treba si zapamätať to, že v príkaze vetvenia sa **nesmie** písať bodkočiarka za prvým vnoreným príkazom ***prikaz1*** ešte pred **else**, ale bodkočiarku píšeme až za druhým vnoreným príkazom ***príkaz2***. Tiež je veľmi dôležité formátovanie príkazu: zatiaľ vidíme že ***príkaz1***, **else** aj ***príkaz2*** sú každé na novom riadku, **else** je na tej istej úrovni ako **if** a oba ***príkaz1*** aj ***príkaz2*** sú odsunuté vľavo o 2 medzery. Neskôr uvidíme aj iné varianty tohto príkazu.

Ďalší program ukáže použitie vetvenia vo vnútri cyklu. Nakreslíme rad štvorčekov a krúžkov, pričom každý tretí bude krúžok, inak sú to samé štvorčeky:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  Image1.Canvas.Brush.Color := clFuchsia;
  for I := 1 to 18 do
    if I mod 3 = 0 then
      Image1.Canvas.Ellipse(I*20, 100, I*20 + 20, 120)
    else
      Image1.Canvas.Rectangle(I*20, 100, I*20 + 20, 120);
end;
```

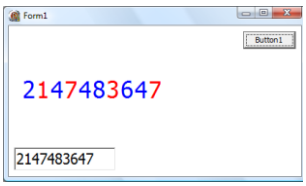
Po spustení:



Keďže celý príkaz vetvenia je vnorený vo for-cykle, tak celý je odsunutý o dve medzery vpravo. Na otestovanie každého tretieho útvaru sme použili podmienku **I mod 3 = 0**, ktorá zistí či hodnota počítadla je deliteľná číslom 3, t.j. pre hodnoty premennej **I**: 3, 6, 9, ... sa nakreslí krúžok, inak sú to štvorčeky.

V ďalšom programe budeme rozoberať číslo zo vstupného riadku na cifry a vypisovať ich buď modrou farbou, ak sú párne, alebo červenou, ak sú nepárne:

```
var
  I, Cislo, Cifra: Integer;
begin
  Image1.Canvas.Font.Height := 40;
  Cislo := StrToInt(Edit1.Text);
  for I := 10 downto 1 do
  begin
    Cifra := Cislo mod 10;
    Cislo := Cislo div 10;
    if Cifra mod 2 = 0 then
      Image1.Canvas.Font.Color := clBlue
    else
      Image1.Canvas.Font.Color := clRed;
    Image1.Canvas.TextOut(I*20, 70, IntToStr(Cifra));
  end;
end;
```

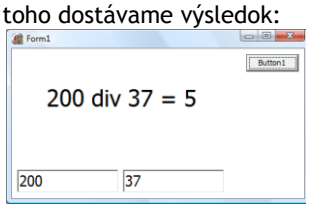


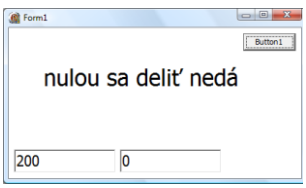
For-cyklus beží 10-krát, lebo na vstupe môže byť aj 10-ciferné číslo. Ak by bolo vstupné číslo menej ciferné, doplnilo by sa zľava nulami. Program otestujme najväčším možným celým číslom (2147483647 = 2^{31}-1).

V ďalšom programe budeme testovať celočíselné delenie dvoch čísel. Obe čísla budú zadané v dvoch vstupných riadkoch. Keďže vieme, že deliť nulou sa nedá, náš program to otestuje a až keď menovateľ je rôzny od 0, vykonáme delenie, inak vypíšeme nejakú správu:

```
var
  A, B: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  B := StrToInt(Edit2.Text);
  Image1.Canvas.Font.Height := 40;
  if B = 0 then
    Image1.Canvas.TextOut(50, 50, 'nulou sa deliť nedá')
  else
    Image1.Canvas.TextOut(50, 50, IntToStr(A) + ' div ' +
      IntToStr(B) + ' = ' + IntToStr(A div B));
end;
```

Podľa toho, či zadáme vhodné alebo nevhodné vstupné hodnoty, podľa toho dostávame výsledok:





Úlohy na precvičenie

Zadanie 1	pre číslo zo vstupného riadku vypíše či je párne alebo nepárne a tiež či je deliteľné 7 alebo nie je deliteľné 7
Zadanie 2	vypíše minimum a aj maximum z dvoch čísel zadaných v dvoch vstupných riadkoch
Zadanie 3	najprv vygeneruje dve náhodné čísla v intervale <50, 200> a potom nakreslí obdĺžnik, ktorého vodorovná strana má dĺžku väčšieho z týchto dvoch čísel a zvislá strana menšieho
Zadanie 4	po zatlačení tlačidla sa na náhodnú pozíciu vypíše buď slovo **Pascal** alebo slovo **Delphi**

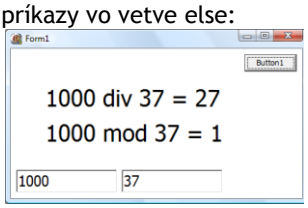
Zadanie 5	nakreslí čo najväčšiu kružnicu, ktorá sa celá zmestí do grafickej plochy a nepresiahne žiaden okraj - musíme vypočítať minimum zo šírky a výšky rozmerov plochy

2. Viac príkazov vo vetve

V prechádzajúcej časti sme počítali celočíselný podiel dvoch čísel. Doplňme program tak, aby okrem delenia (**div**) zistil aj zvyšok po delení (**mod**). Tu by sa nám zišlo, keby sme v druhej vetve (**else**) mohli zapísať dva príkazy. Použijeme zložený príkaz **begin** ... **end**, rovnako ako sme to urobili pri for-cykloch.

```
var
  A, B: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  B := StrToInt(Edit2.Text);
  Image1.Canvas.Font.Height := 40;
  if B = 0 then
    Image1.Canvas.TextOut(50, 50, 'nulou sa deliť nedá')
  else
  begin
    Image1.Canvas.TextOut(50, 50, IntToStr(A) + ' div ' +
      IntToStr(B) + ' = ' + IntToStr(A div B));
    Image1.Canvas.TextOut(50,100, IntToStr(A) + ' mod ' +
      IntToStr(B) + ' = ' + IntToStr(A mod B));
  end;
end;
```

Teraz sa vykonajú oba príkazy vo vetve else:

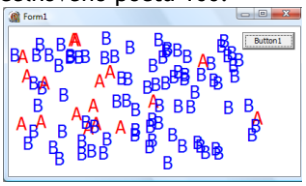


Všimnite si formátovanie podmieneného príkazu **if**: pravidlá sú rovnaké, s akými sme sa stretli pri formátovaní for-cyklu: ak je vnorený jeden príkaz, tak ten je odsunutý oproti **if** (resp. **else**) v predchádzajúcom riadku o 2 medzery vpravo, ak je vnorených viac príkazov (teda sme použili zložený príkaz), tak slová **begin** a **end** sú na rovnakej úrovni ako príslušný **if** a **else** a odsunuté sú až príkazy medzi **begin** a **end**.

Aj v ďalšom programe budeme vykonať viac príkazov za sebou - teraz už v oboch vetvách podmieneného príkazu. Úlohou bude na náhodné pozície 100-krát vypísať červené písmeno 'A' alebo modré písmeno 'B'. Ktoré z týchto písmen program vypíše, rozhodne náhodný generátor: hodí kockou s číslami 0 až 5 (t.j. zavolá funkciu **Random(6)**), a ak na kocke padne 5, tak to bude **A** inak **B**. Je jasné, že pravdepodobnosť **A** je 1/6 a pravdepodobnosť **B** je 5/6.

```
var
  I: Integer;
begin
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.Brush.Style := bsClear;
  for I := 1 to 100 do
    if Random(6) = 5 then
    begin
      Image1.Canvas.Font.Color := clRed;
      Image1.Canvas.TextOut(Random(350), Random(180), 'A');
    end
    else
    begin
      Image1.Canvas.Font.Color := clBlue;
      Image1.Canvas.TextOut(Random(350), Random(180), 'B');
    end;
end;
```

Po spustení môžeme vidieť, že písmen A je asi šestina z celkového počtu 100:



V tomto prípade, keď podmienka obsahuje náhodný generátor, nevieme dopredu povedať (nevieme to odtrasovať), aká bude hodnota podmienky.

Úlohy na precvičenie

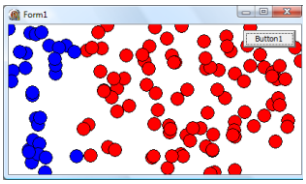
Zadanie 1	dva vstupné riadky obsahujú polomery dvoch sústredných kružníc, nakresliť ich treba tak, že najprv väčšiu a potom v nej menšiu

3. Len jedna vetva príkazu

Pripomeňme si príklad zo začiatku modulu: kreslili sme krúžky na náhodné pozície a podľa X-ovej súradnice sme ich farbili na modro alebo červeno:

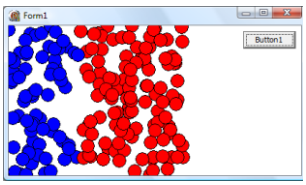
```
var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  if X < 100 then
    Image1.Canvas.Brush.Color := clBlue
  else
    Image1.Canvas.Brush.Color := clRed;
  Image1.Canvas.Ellipse(X - 10, Y - 10, X + 10, Y + 10);
end;
```

Viacnásobné zatlačenie tlačidla nakreslí asi takéto rozloženie:



Ak by sme ale chceli pridať ešte jednu podmienku, napr. že krúžky budú žlté, ale len keď je **X > 250** a inak nech sú modré alebo červené, budeme mať malý problém:

```
var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  if X < 100 then
    Image1.Canvas.Brush.Color := clBlue
  else
    Image1.Canvas.Brush.Color := clRed;
  if X > 250 then
    Image1.Canvas.Brush.Color := clYellow
  else
    // nemeň farbu - nechaj ju nastavenú, ako bola
  Image1.Canvas.Ellipse(X - 10, Y - 10, X + 10, Y + 10);
end;
```



Hoci sme naprogramovali presne, to čo sme si určili: pridali sme nový podmienený príkaz, ktorý testuje **X > 250** a podľa toho nastaví farbu na žltú, alebo ponechá tú čo tam bola nastavená predchádzajúcim podmieneným príkazom, nenakreslí sa ani jeden žltý krúžok (obrázok na okraji strany).

Problémom je použitie druhého if-príkazu. Keď sa na neho pozrieme pozornejšie tak, hovorí presne toto: ak je X > 250 nastav farbu výplne na žltú, inak vykonaj jeden príkaz za **else**. Lenže komentár sa nepovažuje za príkaz (to je len nejaká vsuvka pre nás, ale nie pre počítač) a príkazom za **else** je až nakreslenie kruhu (**Ellipse**). Preto sa žlté kruhy naozaj nekreslili, ale v podmienenom príkaze sa **buď** nastavila farba **alebo** kreslil kruh. Máme tu niekoľko možností, ako to správne vyriešiť. Napr. v else-vetve naozaj nič nevykonať:

```
  if X > 250 then
    Image1.Canvas.Brush.Color := clYellow
  else ;
    // nemeň farbu - nechaj ju nastavenú, ako bola;
```

všimnite si bodkočiarku za **else** - označuje koniec príkazu vo vetve, t.j. zapísali sme **prázdny príkaz**. Toto sa nezvykne zapisovať, programátori to považujú za veľmi nepekné riešenie! Trochu krajší by bol zápis

```
  if X > 250 then
    Image1.Canvas.Brush.Color := clYellow
  else
  begin
    // nemeň farbu - nechaj ju nastavenú, ako bola
  end;
```

ktorý je už správny, pričom do zloženého **begin ... end** príkazu nemusíme ani písať žiadny komentár.

V ďalšom možnom riešení tejto úlohy **znegujeme** podmienku a tým aj prenesieme príkaz z prvej then-vetvy do else-vetvy a prázdny príkaz z **else** za **then**:

```
  if X <= 250 then
  else
    Image1.Canvas.Brush.Color := clYellow;
```

Opäť je to správne ale dosť nezvyklé a väčšinou to ukazuje na to, že takýto programátor je začiatočník a má problémy so slušným zápisom algoritmov.

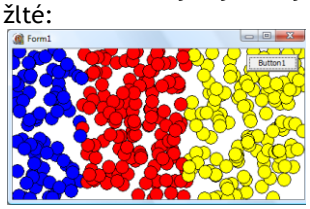
Najlepšie riešenie tohto problému je použitie neúplného podmieneného príkazu, t.j. variantu príkazu, ktorý nemá **else** vetvu. Jeho zápis je nasledovný:

```
if podmienka then
  príkaz;
```

Ak totiž za ***príkaz*** v prvej vetve zapíšeme bodkočiarku, označíme, že za ním nasleduje ďalší príkaz a teda **else**-vetva už nebude. Takže riešenie žltých krúžkov by mohlo vyzerať takto:

```
var
  X, Y: Integer;
begin
  X := Random(Image1.Width);
  Y := Random(Image1.Height);
  if X < 100 then
    Image1.Canvas.Brush.Color := clBlue
  else
    Image1.Canvas.Brush.Color := clRed;
  if X > 250 then
    Image1.Canvas.Brush.Color := clYellow;
  Image1.Canvas.Ellipse(X - 10, Y - 10, X + 10, Y + 10);
end;
```

A teraz sa naozaj objavia aj žlté:



Neskôr uvidíme aj vnorené cykly a táto úloha sa bude dať pekne riešiť aj pomocou nich.

Vráťme sa ešte k prázdnemu príkazu. Často totiž prázdny príkaz označuje chybu, ktorá sa veľmi zle odhaľuje. Napr. časť programu, v ktorej sme chceli vypočítať súčet čísel od 1 do 100:

```
Suma := 0;
for I := 1 to 100 do;
  Suma := Suma + I;
```

Tu sme bodkočiarku dali na veľmi zlé miesto: označuje, že medzi **do** a bodkočiarkou je ***prázdny príkaz*** a práve tento sa bude 100-krát opakovať namiesto priraďovacieho príkazu. Takže 100-krát sa spraví nič a potom jedno priradenie.

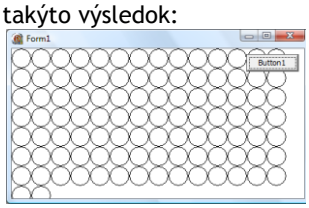
Okrem toho je tu aj iný problém: for-cyklus skončil a hneď za ním sa má vykonať priraďovací príkaz **Suma := Suma + I;** ktorý používa premennú **I**, t.j. počítadlo už skončeného cyklu. Samozrejme, že teraz to nevadí, lebo aj tak je v premennej

Suma zlý výsledok, ale použitie počítadla po skončení cyklu sa považuje za hrubú programátorskú chybu!

V ďalšom príklade potrebujeme nakresliť tesne vedľa seba napr. 300 kruhov. Pravdepodobne sa nevojdú do jedného riadku. Preto ich do prvého riadku nakreslíme najviac ako sa dá a v kreslení pokračujeme na ďalšom riadku. Použijeme dve premenné **X**, **Y**, ktoré označujú stred kresleného kruhu. Ak by sme pred nakreslením každého kruhu skontrolovali, či X-ová súradnica nie je už väčšia ako šírka plochy, tak by sme mohli **X** aj **Y** presunúť na začiatok nového radu, t.j. **X** nastavíme na začiatok a **Y** posunieme nadol o výšku radu, t.j. 2 polomery. Opäť tu bude podmienený príkaz s jednou vetvou:

```
const
  R = 14;
var
  I, X, Y: Integer;
begin
  X := R;
  Y := R;
  for I := 1 to 100 do
  begin
    if X + R > Image1.Width then
    begin
      X := R;
      Y := Y + 2 * R;
    end;
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
    X := X + 2 * R;
  end;
end;
```

Po spustení dostaneme takýto výsledok:

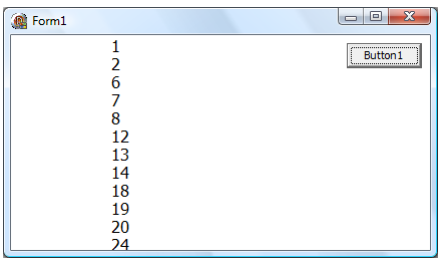


Riešenie tejto úlohy je zaujímavé aj tým, že ukazuje spôsob, ako môžeme dvojrozmerný priestor zapĺňať jedným cyklom. Neskôr toto uvidíme naprogramované pomocou vnorených cyklov.

Nasledujúci program bude vypisovať len také čísla, ktoré spĺňajú nejakú podmienku:

```
var
  I, Y: Integer;
begin
  Image1.Canvas.Font.Height := 20;
  Y := 0;
  for I := 1 to 100 do
    if I div 3 mod 2 = 0 then
    begin
      Image1.Canvas.TextOut(100, Y, IntToStr(I));
      Y := Y + 18;
    end;
```

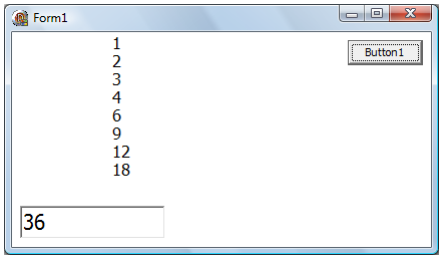
Podmienku pre vypísanie čísla sme nastavili tak, že keď ho celočíselne vydelíme tromi, tak dostávame párne číslo. V zápise **I div 3 mod 2** využívame tú vlastnosť aritmetických operácií, že nakoľko majú **div** aj **mod** rovnakú prioritu, vyhodnocujú sa zľava doprava, teda najprv sa vykoná delenie 3 a potom sa z výsledku zistí zvyšok po delení 2. Ako keby sme zapísali **(I div 3) mod 2**. Podľa veľkosti grafickej plochy bude vidieť len niekoľko výsledkov.



Podobnú ideu o vypisovaní čísel, ktoré spĺňajú nejakú podmienku, môžeme použiť na vypisovanie deliteľov nejakého zadaného čísla. Zo vstupného riadku prečítame číslo a postupne sa ho pokúsime vydeliť všetkými číslami, ktoré sú menšie ako samotné číslo. Ak ho niektoré naozaj delí, tak ho vypíšeme:

```
var
  Cislo, I, Y: Integer;
begin
  Image1.Canvas.Font.Height := 20;
  Cislo := StrToInt(Edit1.Text);
  Y := 0;
  for I := 1 to Cislo - 1 do
    if Cislo mod I = 0 then
    begin
      Image1.Canvas.TextOut(100, Y, IntToStr(I));
      Y := Y + 18;
    end;
end;
```

Otestujeme to pre nejaký vstup, ktorý vieme jednoducho posúdiť, či je správny:

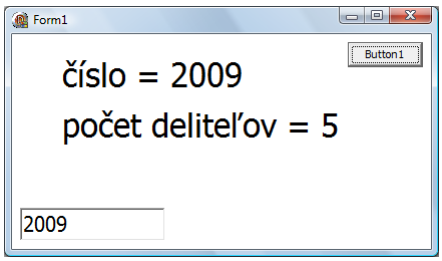


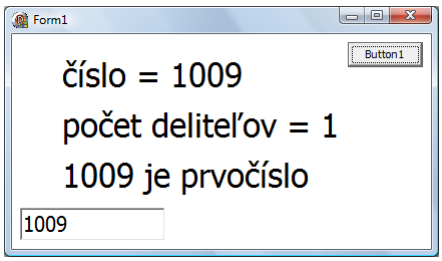
Odtiaľto je už len malý krôčik k programu, ktorý vie zistiť, či je zadané číslo prvočíslom. Zmeňme tento program tak, aby nevypisoval všetky delitele čísla, ale aby ich len počítal. Po skončení cyklu program tento počet vypíše a ak sa rovná 1, tak to bolo prvočíslo:

```
var
  Cislo, Pocet, I: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Font.Height := 40;
  Cislo := StrToInt(Edit1.Text);
  Pocet := 0;
  for I := 1 to Cislo - 1 do
    if Cislo mod I = 0 then
      Pocet := Pocet + 1;
  Image1.Canvas.TextOut(50, 20, 'číslo = ' + IntToStr(Cislo));
  Image1.Canvas.TextOut(50, 70, 'počet deliteľov = ' +
IntToStr(Pocet));
  if Pocet = 1 then
    Image1.Canvas.TextOut(50, 120, IntToStr(Cislo) + ' je prvočíslo');
end;
```

Poznámka:
Ak by sme potrebovali zistiť len to, či je číslo prvočíslom, netreba zisťovať počet všetkých deliteľov. Stačiť, či je aspoň jeden deliteľ z intervalu od 2 do druhej odmocniny z čísla.

Program spustíme pre niektoré hodnoty:

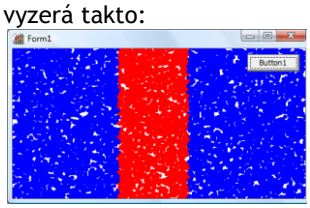




V ďalšom programe chceme otestovať podmienku **Abs(X-200) < 50**. Napíšeme program, ktorý vygeneruje väčšie množstvo náhodných bodov v rovine a tie, ktoré budú spĺňať túto podmienku zafarbí na červeno. Body budeme kresliť ako veľmi krátke ale zato hrubé úsečky:

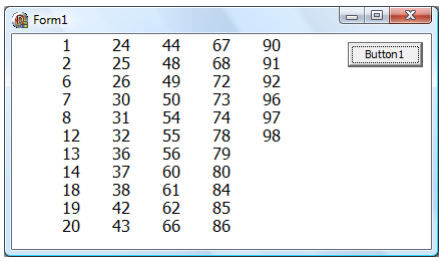
```
var
  I, X, Y: Integer;
begin
  Image1.Canvas.Pen.Width := 5;
  for I := 1 to 10000 do
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Pen.Color := clBlue;
    if Abs(X - 200) < 50 then
      Image1.Canvas.Pen.Color := clRed;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X + 1, Y);
  end;
end;
```

Po spustení programu sa nakreslí 10000 farebných bodiek a na obrazovke to vyzerá takto:



Program zafarbuje na červeno tie bodky, ktorých X-ová súradnica je v intervale <150, 250>.

Úlohy na precvičenie

Zadanie 1	robili sme program, ktorý vypisoval čísla spĺňajúce podmienku (**I div 3 mod 2 = 0**) do jedného stĺpca; dopíšte do tohto programu taký mechanizmus, ktorým sa tieto čísla budú vypisovať do viacerých stĺpcov vedľa seba, napr. takto 
Zadanie 2	program zistí súčet deliteľov zadaného čísla zo vstupného riadka (delitele od 1, ktoré sú menšie ako samotné číslo), vypíše ho a ak sa tento súčet rovná samotnému číslu, tak vypíše, že je to "***dokonalé číslo***" - môžete to potom otestovať napr. na týchto dokonalých číslach 6, 28, 496
Zadanie 3	program bude vypisovať 10000 bodiek (napr. veľkosti 5) na náhodných pozíciách, pričom s pravdepodobnosťou 1/200 bude meniť farbu bodiek na náhodnú
Zadanie 4	program vypíše vedľa seba 64 štvorcov, pričom pri každom ôsmom ich začne dávať do nového radu (vytvorí štvorcovú sieť 8x8), pričom ich zafarbí dvoma rôznymi farbami (napr. žltou a modrou) tak, ako na šachovnici

Zadanie 5	bez toho, aby ste používali počítač, zistite čo robí
	`N := 0;`
	`for I := 1 to 20 do`
	`  if I mod 3 = 0 then`
	`  N := N * 10 + I mod 10;`

Zadanie 6	v poslednom príklade v tejto časti, ktorý menil farbu bodiek podľa toho, či platila podmienka (**Abs(X-200) < 50**), otestujte tieto ďalšie podmienky a pouvažujte o využití takejto metódy na hodinách matematiky: • ak **Abs(Y-100) < 50**, tak zelené • ak **Abs(X-200) < 50**, tak červené a súčasne ak **Abs(Y-100) < 50**, tak zelené • **Abs(Abs(X-200) - Abs(Y-100)) < 20** • **Abs(Abs(X-200) + Abs(Y-100)) < 80** • **Sqr(X-200) + Sqr(Y-100) < 10000**, kde **Sqr** označuje druhú mocninu čísla

4. Vnorené podmienené príkazy (if)

Na to, aby sme vypísali správu o tom, či používateľ zadal (cez vstupný riadok) kladné číslo, záporné číslo alebo číslo rovné 0, môžeme použiť tri za sebou idúce príkazy vetvenia:

```
var
  N: Integer;
begin
  N := StrToInt(Edit1.Text);
  if N = 0 then
    Image1.Canvas.TextOut(50, 50, 'Číslo je rovné 0');
  if N < 0 then
    Image1.Canvas.TextOut(50, 50, 'Záporné číslo');
  if N > 0 then
    Image1.Canvas.TextOut(50, 50, 'Kladné číslo');
end;
```

Hoci je program správny, na ňom si môžeme uvedomiť to, že pri ľubovoľnej zadanej hodnote, program vždy testuje tri rôzne podmienky a pritom vôbec nevyužíva else-vetvu. Ak to teraz prepíšeme takto:

```
var
  N: Integer;
begin
  N := StrToInt(Edit1.Text);
  if N = 0 then
    Image1.Canvas.TextOut(50, 50, 'Číslo je rovné 0')
  else
    if N < 0 then                  // vnorený príkaz if
      Image1.Canvas.TextOut(50, 50, 'Záporné číslo')
    else
      Image1.Canvas.TextOut(50, 50, 'Kladné číslo');
end;
```

v prípade, že bola zadaná 0, sa už ďalej v programe nič netestuje. Ak zadáme nejaké kladné číslo, tak sa urobí ešte jeden test. Ak zadáme záporné číslo, tak sa tiež uskutočnia iba dva testy. Takémuto príkazu **if**, ktorý je vo vnútri iného if-príkazu, hovoríme ***vnorený podmienený príkaz***, a pri programovaní sa používa veľmi často. Všimnite si formátovanie príkazu: keďže vnorený príkaz **if** patrí do else-vetvy, je celý odsunutý o 2 medzery vpravo. Toto isté môžeme naformátovať aj takto:

```
var
  N: Integer;
begin
  N := StrToInt(Edit1.Text);
  if N = 0 then
    Image1.Canvas.TextOut(50, 50, 'Číslo je rovné 0')
  else if N < 0 then                  // vnorený príkaz if
    Image1.Canvas.TextOut(50, 50, 'Záporné číslo')
  else
    Image1.Canvas.TextOut(50, 50, 'Kladné číslo');
end;
```

to znamená, že ak **else**-vetva obsahuje iba vnorený príkaz **if**, pri zápise programu, môžeme **if** zapísať do toho istého riadku ako **else**. Niekedy potom môžu vznikať aj takéto konštrukcie:

```
if podmienka then
  príkaz
else if podmienka then
  príkaz
else if podmienka then
  príkaz
else if podmienka then
  príkaz
else
  príkaz
```

Tu sa postupne testujú štyri podmienky a len pre prvú z nich, ktorá bude pravdivá, sa vykonajú príslušné príkazy.

Pri vnorených podmienených príkazoch môžu vznikať aj problémy a to napríklad vtedy, keď vnorená **if** je bez vetvy **else**. Nasledujúci program by nás mal informovať o tom, či je nejaké celé číslo N deliteľné desiatimi:

```
if N > 0 then
  if N mod 10 = 0 then
    Image1.Canvas.TextOut(0, 0, 'je deliteľné 10')
else
  Image1.Canvas.TextOut(0, 0, 'zadaj prirodzené číslo');
```

Program by chcel pre kladné číslo zistiť deliteľnosť a pre zle zadané číslo, dať správu. Lenže takto pracuje správne iba pre niektoré čísla 10, 20, 50, ... - teda tie, ktoré sú deliteľné 10. Napríklad, pre N=33, už zobrazí nesprávny text "Zadaj prirodzené číslo". Vetva **else** sa viaže až k druhému (vnorenému) príkazu **if N mod 10 = 0 then**. Keďže 33 nie je deliteľné 10, vykoná sa príkaz z vetvy **else**.

Máme niekoľko možností, ako sa dá program opraviť:

- k druhému príkazu **if** pridáme vetvu **else** s prázdnym príkazom:

```
if N > 0 then
  if N mod 10 = 0 then
    Image1.Canvas.TextOut(0, 0, 'je deliteľné 10')
  else
else
  Image1.Canvas.TextOut(0, 0, 'zadaj prirodzené číslo');
```

- alebo použijeme príkazové zátvorky (zložený príkaz **befin** a **end**):

```
if N > 0 then
begin
  if N mod 10 = 0 then
    Image1.Canvas.TextOut(0, 0, 'je deliteľné 10');
end
else
  Image1.Canvas.TextOut(0, 0, 'zadaj prirodzené číslo');
```

- alebo zmeníme podmienky:

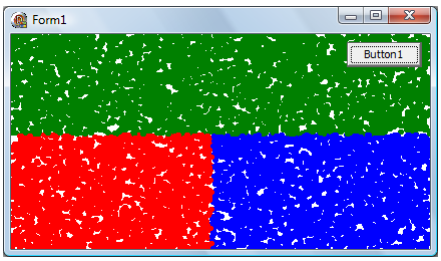
```
if N <= 0 then
  Image1.Canvas.TextOut(0, 0, 'zadaj prirodzené číslo')
else
  if N mod 10 = 0 then
    Image1.Canvas.TextOut(0, 0, 'je deliteľné 10');
```

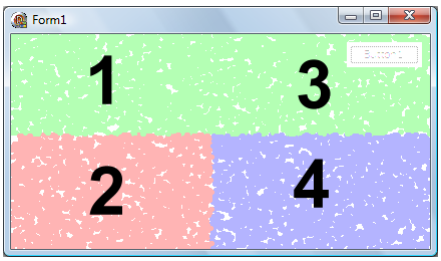
Ak sa dá, prednosť dáme poslednému riešeniu.

Poďme riešiť ďalší program. Už v niekoľkých predchádzajúcich programoch sme náhodne generovali body a zafarbovali sme ich podľa nejakých kritérií. Napr.

```
var
  I, X, Y: Integer;
begin
  Image1.Canvas.Pen.Width := 5;
  for I := 1 to 10000 do
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Pen.Color := clBlue;
    if X < 200 then
      Image1.Canvas.Pen.Color := clRed;
    if Y < 100 then
      Image1.Canvas.Pen.Color := clGreen;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X + 1, Y);
  end;
end;
```

Takto vybodkuje grafickú plochu:





Vidíme, že dvomi podmienkami **X < 200** a **Y < 100** sme rozdelili plochu na štyri kvadranty (čo sme znázornili na druhom obrázku), pričom dva podmienené príkazy určujú, ako sa vygenerovaná bodka zafarbí:

- najprv predpokladáme, že ani jedna podmienka nebude splnená, teda v cykle zvolíme modrú farbu (prvé priradenie modrej) – zatiaľ pre všetky 4 kvadranty
- potom testujeme, či **X < 200**, t.j. ak je v 1. a 2. kvadrante, bude červená
- potom testujeme, či **Y < 100**, t.j. ak je v 1. a 3. kvadrante, bude zelená

Teda bodka v 1. kvadrante, napr. tomu, že spĺňa obe podmienky, sa zafarbí na zeleno, lebo test **Y < 100** prešiel ako druhý a zmenil červenú farbu na zelenú.

Našou úlohou teraz bude tieto testy prepísať tak, aby bol každý kvadrant zafarbený inou farbou. Skúsme si takéto nové podmienky najprv povedať slovne:

- ak X < 200 (bude to 1. alebo 2. kvadrant), otestujeme ešte, či Y < 100 a tento test rozhodne, ktorý z týchto dvoch to bude naozaj
- ak neplatí X < 200, teda else-vetva (bude to 3. alebo 4. kvadrant), znovu otestujeme ešte, či Y < 100 a tento test rozhodne, ktorý z týchto dvoch to bude naozaj

Teraz to zapíšeme

a po spustení dostávame 4 rôzne zafarbené kvadranty:


```
var
  I, X, Y: Integer;
begin
  Image1.Canvas.Pen.Width := 5;
  for I := 1 to 10000 do
  begin
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Pen.Color := clBlue;
    if X < 200 then
      if Y < 100 then
        Image1.Canvas.Pen.Color := clGreen
      else
        Image1.Canvas.Pen.Color := clRed
    else
      if Y < 100 then
        Image1.Canvas.Pen.Color := clYellow
      else
        Image1.Canvas.Pen.Color := clBlue;
    Image1.Canvas.MoveTo(X, Y);
    Image1.Canvas.LineTo(X + 1, Y);
  end;
end;
```

Úlohy na precvičenie

Zadanie 1	pre zadané číslo zo vstupného riadka vypísať zvyšok po delení číslom 4 – tento výpis urobiť slovne, t.j. jedno z 'nula', 'jedna', 'dva' a 'tri'
Zadanie 2	nakresliť korále – teda čo najdlhší rad, v ktorom sa pravidelne striedajú farebné štvorčeky 10x10, krúžky s polomerom 5 a obdĺžniky 10x20; pritom sa budú pravidelne striedať 4 farby, napr. červená, zelená, žltá a modrá

5. Zostavovanie zložených podmienok (logické operácie)

Program by mal podľa zadaného času v hodinách pozdraviť jedným z pozdravov (čas zadávame len v celých číslach hodín):

Dobré ráno
Dobrý deň
Dobrý večer
Dobrú noc

Napr. pre 'Dobré ráno' máme predstavu časového intervalu od 5 do 8 hodín a pre 'Dobrý večer' od 17 do 21 hodín. Ak máme v premennej **Hodin** momentálny počet hodín, tak potrebujeme otestovať dve podmienky **Hodin ≥ 5** a súčasne **Hodin ≤ 8** a vtedy je určite ráno. Ak pri zadávaní momentálneho času do premennej **Hod** priradíme omylom hodnotu menšiu ako nula, alebo naopak väčšiu ako 24, tak by nás program nemusel zdraviť napr. 'Dobrú noc', ale radšej by nás mal upozorniť na chybne zadaný vstup. Takže by sa nám hodilo vedieť otestovať, či platí podmienka **Hodin < 0** alebo **Hodin >= 25**.

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

podm1 **and** *podm2*	- otestovanie, či súčasne platia obe podmienky ***podm1*** aj ***podm2***
podm1 **or** *podm2*	- otestovanie, či platí aspoň jedna z podmienok ***podm1*** a ***podm2***
not *podm1*	- otestovanie, či neplatí podmienka ***podm1***, to znamená znegovanie podmienky

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

Už vieme, že s prioritou operácií je to pri programovaní veľmi vážne a tieto tri operácie nám prinášajú nové komplikácie. Neskôr si ukážeme tabuľku priorít všetkých doteraz známych operácií, zatiaľ by nám mohlo postačiť to, že parametre logických operácií väčšinou budeme dávať do zátvoriek. Napríklad takto

```
(Hodin < 0) or (Hodin > 24)
(Hodin >= 5) and (Hodin <= 8)
(Hodin >= 9) and (Hodin <= 16)
(Hodin >= 17) and (Hodin <= 21)
(Hodin <= 4) or (Hodin >= 22)
```

A v podmienenom príkaze to môže vyzerať potom takto

```
if (Hodin < 0) or (Hodin > 24) then
  Image1.Canvas.TextOut(50, 50, 'Neblázni!')
else
```

čo znamená toľko, že ak zadaná hodnota hodín je menšia ako 0 alebo naopak väčšia ako 24, tak sa zadal zlý vstup a treba o tom vypísať nejakú správu. Ak táto podmienka splnená nie je (bol zadaný správny vstup), pokračuje sa na príkazoch za **else**. Pri takýchto zložených podmienkach je veľmi často vhodné si uvedomiť, kedy nastane prípad nesplnenej podmienky a teda sa vykoná else-vetva. Stačí si správne znegovať celú podmienku a dostávame:

```
not ((Hodin < 0) or (Hodin > 24)) =
  not (Hodin < 0) and not (Hodin > 24) =
    (Hodin >= 0) and (Hodin <= 24)
```

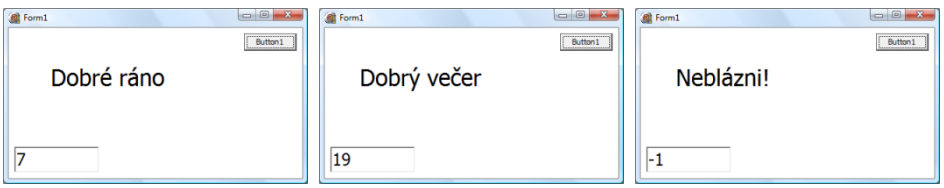
z matematiky poznáme tieto všeobecné vzťahy

- `not (P1 or P2) = (not P1) and (not P2)`
- `not (P1 and P2) = (not P1) or (not P2)`
- `not (not P1) = P1`

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

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

Podľa toho, akú hodinu zadáme, dostávame nejaký pozdrav:



Odkrokujte výpočet pre niektoré hodiny a zistite, či posledný príkaz vetvenia s podmienkou **(Hodin <= 4) or (Hodin >= 22)** je na tomto mieste nutný, či by tu nestačil len samotný **else.**

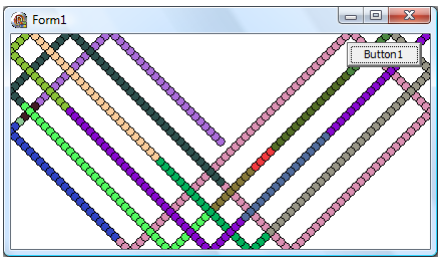
Z matematiky poznáme tzv. tabuľky pravdivostných hodnôt pre rôzne logické operácie. Napr. pre operáciu **and** vyzerá takto:

and	pravda	nepravda
pravda	**pravda**	**nepravda**
nepravda	**nepravda**	**nepravda**

Zostavte takúto tabuľku aj pre operácie **or** and **not**.

Ďalší zaujímavý program bude kresliť dráhu guľôčky, ktorá štartuje v strede grafickej plochy (súradnica **Image1.Width div 2**, **Image1.Height div 2**). Ďalej pokračuje niektorým zo štyroch šikmých smerov (rozhodne sa na začiatku náhodne) a pri vypadnutí z plochy, zmení smer tak aby sa vrátil do plochy. Využívame dve pomocné premenné DX a DY, ktoré určujú relatívnu zmenu guľôčky oproti momentálnej polohe. Tieto dve premenné môžu mať len hodnoty **±5** a preto ich na začiatku programu generujeme vzorcom **Random(2)*10-5**. Tento vzorec naozaj generuje len buď +5 alebo -5. Ak je momentálna poloha guľôčky napr. 100, 150 a DX=5, DY=-5, potom jej nasledovná poloha bude 105, 145, ďalšia 110, 140, atď. až kým napr. X-ová súradnica nenarazí na pravý okraj, napr. **X > 200**. Vtedy sa zmení DX na -DX, t.j. odteraz DX=-5 a teda sa X bude v ďalších krokoch zmenšovať.

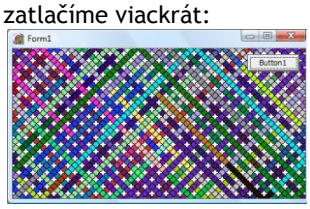
Aby bola výsledná cesta zaujímavejšia, raz začas náhodne zmeníme farbu guľôčky (napr. keď bude **Random(40)=0**, zmeň farbu). Cesta môže vyzerať asi takto:



Cesta závisí nielen náhodného generátora, ale aj od konkrétnych rozmerov grafickej plochy vo vašej aplikácii. Samotný program potom vyzerá takto:

```
var
  I, X, Y, DX, DY: Integer;
begin
  X := Image1.Width div 2;
  Y := Image1.Height div 2;
  DX := Random(2) * 10 - 5;
  DY := Random(2) * 10 - 5;
  Image1.Canvas.Brush.Color := Random(256 * 256 *256);
  for I := 1 to 500 do
  begin
    Image1.Canvas.Ellipse(X - 5, Y - 5, X + 5, Y + 5);
    X := X + DX;
    Y := Y + DY;
    if (X < 0)  or (X > Image1.Width) then
      DX := -DX;
    if (Y < 0)  or (Y > Image1.Height) then
      DY := -DY;
    if Random(40) = 0 then
      Image1.Canvas.Brush.Color := Random(256 * 256 *256);
  end;
end;
```

Zaujímavé obrázky vznikajú, aj keď tlačidlo štartujúce program, zatlačíme viackrát:



Vyskúšajte zmeniť aj hodnoty pre DX a DY, napr. **DX=±7** a **DY=±4**.

Úlohy na precvičenie

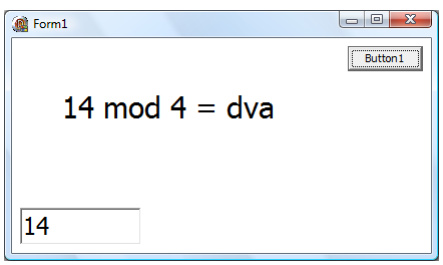
Zadanie 1	úlohu so 4 rôzne zafarbenými kvadrantmi z predchádzajúcej podkapitoly, riešte pomocou maximálne troch príkazov **if** - využite logické operácie **and**
Zadanie 2	najprv náhodne vygenerujte 2 body: obe súradnice prvého v intervale <30, 80>, obe súradnice druhého v intervale <120, 170>; potom náhodne vygenerujte 10000 bodov, ktoré ak sú vo vnútri obdĺžnika určeného dvomi vygenerovanými vrcholmi, tak tieto body budú červené, inak budú modré
Zadanie 3	nakresliť korále (vodorovný rad) z 50 farebných krúžkov s polomerom 8, v ktorých všetky tie, ktorých poradové číslo je deliteľné 6 alebo 7 budú červené, zo zvyšných tie, ktorých poradové číslo dáva zvyšok po delení číslom 5 buď 1 alebo 3 budú žlté, všetky zvyšné budú modré
Zadanie 4	program bude simulovať hádzanie troma kockami: najprv vygeneruje a vypíše 3 náhodné čísla z intervalu <1, 6>; ak sú všetky tri rovnaké, tak vypíše správu BINGO, ak sú rovnaké len nejaké dve čísla, tak vypíše správu SUPER, ak sú všetky tri čísla rôzne, vypíše správu ŠKODA
Zadanie 5	program prečíta tri celé čísla (z troch vstupných riadkov) a vypíše najmenšie a najväčšie z nich

6. Zložený podmienený príkaz (case)

V predchádzajúcich častiach sme sa mohli stretnúť s úlohou, v ktorej sa zisťoval zvyšok po delení zadaného čísla číslom 4. Výsledok mal byť uvedený slovne (jedným zo slov: nula, jeden, dva tri). Pomocou série podmienených príkazov ste to riešili asi takto:

```
var
  Cislo: Integer;
begin
  Image1.Canvas.Font.Height := 36;
  Cislo := StrToInt(Edit1.Text);
  if Cislo mod 4 = 0 then
    Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = nula')
  else if Cislo mod 4 = 1 then
    Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = jeden')
  else if Cislo mod 4 = 2 then
    Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = dva')
  else
    Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = tri');
end;
```

Po spustení



V tejto sérii za sebou vnorených príkazov **if** stále testujeme jednu a tú istú hodnotu: **Cislo mod 4** a porovnávame ju s 0, 1 a 2. Pascal poskytuje ešte jeden iný príkaz vetvenia, ktorý netestuje hodnotu nejakej podmienky na pravda/nepravda, ale podľa hodnoty nejakého číselného výrazu, vykoná jeden z možných výrazov. Príkaz zapisujeme takto:

```
case výraz of
  hodnota1: príkaz1;
  hodnota2: príkaz2;
  hodnota3: príkaz3;
  ...
else
  príkazE;
end;
```

Predchádzajúci program môžeme prepísať pomocou nášho nového príkazu **case** takto:

```
var
  Cislo: Integer;
begin
  Image1.Canvas.Font.Height := 36;
  Cislo := StrToInt(Edit1.Text);
  case Cislo mod 4 of
    0: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = nula');
    1: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = jeden');
    2: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = dva');
  else
    Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = tri');
  end;
end;
```

Príkaz **case** nemusí mať **else**-vetvu, vtedy, ak ani jedna z možností nie je medzi uvedenými hodnotami, tak príkaz nespraví nič, podobne, ako príkaz **if** bez **else**-vetvy.

Zapíšme tento program ešte raz, ale bez else-vetvy a aj bez pomocnej premennej **Cislo**:

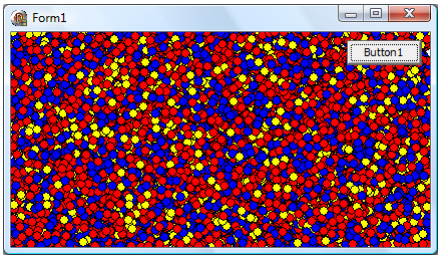
```
begin
  Image1.Canvas.Font.Height := 36;
  case StrToInt(Edit1.Text) mod 4 of
    0: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = nula');
    1: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = jeden');
    2: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = dva');
    3: Image1.Canvas.TextOut(50, 50, Edit1.Text + ' mod 4 = tri');
  end;
end;
```

Ešte pripomíname, že hodnoty pre samotné vetvy musia byť konštanty, ak chceme v nejakej vetve mať viac ako jeden príkaz, musíme ich uzavrieť do príkazových zátvoriek (**begin**, **end**), za každou vetvou musíme uviesť bodkočiarku.

V ďalšom príklade predvedieme, že jedna vetva môže byť určená aj pre viac konštánt. Budeme riešiť takýto problém: zobrazíme väčšie množstvo farebných guľôčok, ktoré sú na náhodných pozíciách a sú zafarbené podľa týchto pravidiel: hádžeme kockou, a ak na nej padne 1 alebo 2, guľôčka je modrá, ak padne 3, je žltá inak (pre zvyšné tri čísla) červená. Príkaz **case** najprv vygeneruje náhodné číslo (**Random(6)**) a presne podľa podmienok nastaví farbu výplne:

```
var
  I, X, Y: Integer;
begin
  for I := 1 to 10000 do
  begin
    case Random(6) of
      1, 2: Image1.Canvas.Brush.Color := clBlue;
      3: Image1.Canvas.Brush.Color := clYellow;
    else
      Image1.Canvas.Brush.Color := clRed;
    end;
    X := Random(Image1.Width);
    Y := Random(Image1.Height);
    Image1.Canvas.Ellipse(X - 5, Y - 5, X + 5, Y + 5);
  end;
end;
```

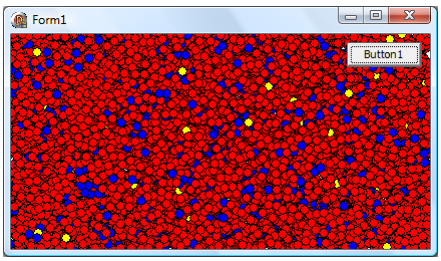
Po spustení:



Z riešenia úlohy by malo byť jasné, že pomer červených, modrých a žltých guľôčok je 3:2:1. Rovnakou metódou vieme dosiahnuť aj iné pomery, hoci napr. 89:10:1. Stačí malá úprava príkazu **case**:

```
    case Random(100) of
      0..9: Image1.Canvas.Brush.Color := clBlue;
      10: Image1.Canvas.Brush.Color := clYellow;
    else
      Image1.Canvas.Brush.Color := clRed;
    end;
```

Prvá vetva príkazu zahŕňa všetky čísla z intervalu <0, 9>, čo v Pascale zapisujeme ako **0..9**. Teraz pomer farieb vyzerá takto: 89:10:1 a po spustení vidíme:



Úlohy na precvičenie

Zadanie 1	nakreslí do radu **N** kružníc, pričom sa strieda nasledujúcich 5 farieb výplne kruhov: červená, modrá, žltá, zelená, biela
Zadanie 2	podľa zadaného čísla zo vstupného riadku (z intervalu od 1 do 6) nakreslí hraciu kocku s príslušným počtom bodiek (čierne krúžky); hracia kocka je štvorec veľkosti napr. 100x100, pre ktorý jednu bodku kreslíme v strede; priemer krúžku môže byť napr. 20

2. tematická jednotka – cyklus s podmienkou

1. Cyklus s jednoduchými podmienkami

Keď sme potrebovali opakovať nejaké príkazy, použili sme programovú konštrukciu for-cyklus. Táto konštrukcia je dostatočne jednoduchá a máme pritom k dispozícii počítadlo, ktoré môžeme často veľmi užitočne využiť v tele cyklu. Pri programovaní sa ale často stretávame so situáciou, že chceme nejaké príkazy opakovať, ale môže byť dosť náročné dopredu určiť ich počet. Aby mal cyklus zmysel, musí niekedy skončiť a teda musíme nejako počítaču oznámiť, kedy sme už spokojní s opakovaním príkazov a chceme pokračovať v nasledujúcich príkazoch za cyklom.

Na toto nám pascal poskytuje cyklus s podmienkou (budeme mu hovoriť aj while-cyklus). V tomto cykle príkazy, ktoré chceme opakovať, "zabalíme" do konštrukcie while:

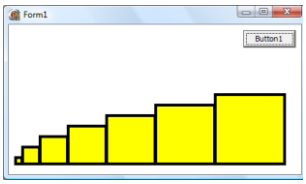
```
while podmienka do
  príkaz;
```

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

```
while podmienka do
begin
  príkaz;
  príkaz;
  ...
end;
```

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

Prvý príklad, v ktorom použijeme cyklus s podmienkou, bude kresliť rad žltých zväčšujúcich sa štvorcov. Boli by sme radi, keby to vyzeralo nejako takto:



Chceme vypísať maximálny počet štvorcov, ale tak, aby sa posledný ešte celý zmestil do grafickej plochy. Koľko štvorcov nakreslíme teda bude závisieť od veľkosti prvého štvorca, od toho, o koľko sa budú zväčšovať a aj od šírky grafickej plochy. Naprogramovať túto úlohu použitím for-cyklu by bolo zbytočné trápenie, pričom pomocou while-cyklu to bude takto jednoduché:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X, Velkost: Integer;
begin
  Image1.Canvas.Pen.Width := 4;
  Image1.Canvas.Brush.Color := clYellow;
  X := 10;
  Velkost := 10;
  while X + Velkost < Image1.Width do
  begin
    Image1.Canvas.Rectangle(X, 200, X + Velkost, 200 - Velkost);
    X := X + Velkost;
    Velkost := Velkost + 15;
  end;
end;
```

Prejdime spoločne toto riešenie:

- prvý štvorec by mal mať **Velkost** = 10 a začneme ho kresliť na x-ovej súradnici **X** = 10;
- skontrolujeme, či sa zmestí do grafickej plochy, to znamená, či je jeho pravá strana (**X + Velkost**) menšia ako pravý okraj grafickej plochy (**Image1.Width**);
- ak je podmienka splnená, môžeme postupne vykonať telo cyklu, t.j. príkazy medzi **begin** a **end**;
- všimnite si, že okrem nakreslenia štvorca v tele cyklu nastavíme novú hodnotu **X**

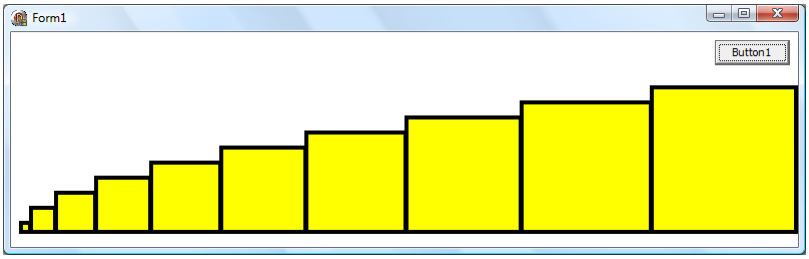
aj **Velkost** pre kreslenie ďalšieho štvorca – druhý štvorec by sa mal nakresliť na súradnici **X** = 20 a jeho veľkosť bude 25;

- keďže je to cyklus, tieto príkazy sa budú opakovať, kým je splnená podmienka cyklu, t.j. štvorce sa budú kresliť dovtedy, kým by sa mal nakresliť taký, ktorý by sa celý už nezmestil – vtedy cyklus skončí a pokračuje sa vo vykonávaní príkazov za cyklom (v našom príklade ďalej nasleduje už len koniec procedúry).

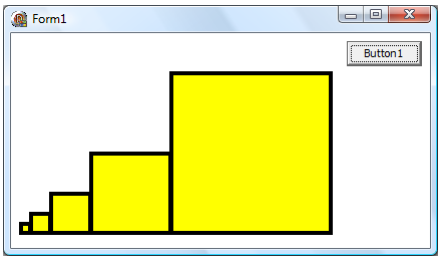
Zapíšme si postupne do tabuľky hodnoty premenných **X** a **Velkost**, aby sme lepšie videli priebeh while-cyklu (tu predpokladáme, že šírka plochy je 400):

	X	**Velkost**	**X + Velkost < Image1.Width**
1	10	10	*áno*
2	20	25	*áno*
3	45	40	*áno*
4	85	55	*áno*
5	140	70	*áno*
6	210	85	*áno*
7	295	100	*áno*
8	395	115	*nie*

Postupne sa nakreslí 7 zväčšujúcich sa štvorcov a keď by sa mal nakresliť ôsmy veľkosti 115 na súradnicu **X**=395, tak cyklus skončí a tento štvorec sa už nenakreslí. Toto riešenie má ešte aj tú výhodu, že bude správne fungovať, aj keď napr. zmeníme zväčšovanie štvorcov, buď o inú konštantu alebo iným vzorcom. Samozrejme, že program správne funguje aj pre inú šírku grafickej plochy, napr.



Pozmeňte tento program tak, aby sa štvorce zväčšovali vždy dvojnásobne od predchádzajúceho.



Vytvorte si trasovaciu tabuľku, aby ste preverili, že v tomto prípade sa naozaj nakreslí len 5 štvorcov (pre šírku plochy 400).

Aby sme si lepšie zvykli na spôsob zápisu algoritmov pomocou while-cyklu, vyriešime niekoľko menších úloh. Pozorne ich preštudujte.

1. Program vypočíta celú časť druhej odmocniny čísla **N**:

```
N := StrToInt(Edit1.Text);
Odmocnina := 0;
while Odmocnina * Odmocnina <= N do
  Odmocnina := Odmocnina + 1;
Image1.Canvas.TextOut(50, 50, '√ = ' + IntToStr(Odmocnina - 1));
```

Kým je druhá mocnina pomocnej premennej **Odmocnina** stále menšia alebo rovná ako skúmané číslo **N**, tak pomocnú premennú zvyšujeme o 1. Cyklus skončí, keď

podmienka už neplatí, t.j. neplatí **Odmocnina * Odmocnina <= N**, čo je to isté, ako **Odmocnina * Odmocnina > N**, preto **Odmocnina - 1** je hľadaný výsledok.

2. Úplne rovnakú ideu môžeme použiť celú časť tretej odmocniny čísla **N**:

```
N := StrToInt(Edit1.Text);
Odmocnina := 0;
while Odmocnina * Odmocnina * Odmocnina <= N do
  Odmocnina := Odmocnina + 1;
Image1.Canvas.TextOut(50, 50, '√ = ' + IntToStr(Odmocnina - 1));
```

3. Predpokladáme, že zadané celé číslo **N** je deliteľné 2 alebo jeho nejakou mocninou. Program zistí, akou najväčšou mocninou čísla 2 je **N** deliteľné:

```
N := StrToInt(Edit1.Text);
Delitel := 1;
while N mod 2 = 0 do
begin
  Delitel := 2 * Delitel;
  N := N div 10;
end;
Image1.Canvas.TextOut(50, 50, 'mocnina 2 = ' + IntToStr(Delitel));
```

4. Prepíšeme nasledujúci program pomocou while-cyklu:

```
for I := 1 to 20 do
begin
  R := 105 - 5 * I;
  Image1.Canvas.Ellipse(200 - R, 120 - R, 200 + R, 120 + R);
end;
```

Program kreslí 20 sústredných kruhov. A pomocou while-cyklu to prepíšeme takto

```
I := 1;
while I <= 20 do
begin
  R := 105 - 5 * I;
  Image1.Canvas.Ellipse(200 - R, 120 - R, 200 + R, 120 + R);
  I := I + 1;
end;
```

hoci while-cyklus nám to umožní prepísať aj bez počítadla cyklu (využívame to, že premenná **R** postupne nadobúda hodnoty 100, 95, 90, ..., 5):

```
R := 100;
while R > 0 do
begin
  Image1.Canvas.Ellipse(200 - R, 120 - R, 200 + R, 120 + R);
  R := R - 5;
end;
```

Ďalší program budeme riešiť podrobnejšie. Program bude hľadať také najväčšie celé číslo **N**, pre ktoré sa dá vypočítať faktoriál ale v celočíselnej aritmetike. Môžeme tu použiť preddefinovanú konštantu **MaxInt**, ktorá má hodnotu 2147483647, t.j. najväčšie možné celé číslo **Integer**. Najprv zapíšme dobre vyzerajúce ale zlé riešenie:

```
N := 1;
Faktorial := 1;
while Faktorial * N <= MaxInt do
begin
  Faktorial := Faktorial * N;
  N := N + 1;
end;
Image1.Canvas.TextOut(50, 50, 'max = ' + IntToStr(N));
```

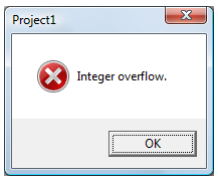
podmienka vo while-cykle chce testovať, či nejaké celé číslo, ktoré vznikne ako súčin dvoch čísel je menšie ako **MaxInt**. Lenže **MaxInt** je najväčšie možné celé číslo

a od neho väčšie neexistuje. Keď sa pokúsite tento program skompilovať, tak najprv dostanete varovanie (***Warning***) od *Delphi:

Comparision always evaluates to True

čo znamená, že táto podmienka je vždy pravdivá a teda cyklus nikdy neskončí. Ak napriek tomuto varovaniu zatlačíme tlačidlo a tým spustíme program, tak teraz môžu nastať dve situácie:

- ak máme zapnuté "**Overflow checking**", tak program spadne na chybovej správe:



- ak túto kontrolu máme vypnutú, tak sa program zacyklí a nereaguje na žiadne tlačidlá (musíme ho zastaviť z prostredia Delphi pomocou **Ctrl F2**).

Je teda jasné, že takto sa pýtať na už zlý súčin **Faktorial * N <= MaxInt** nedá. Preto si zapamätajme, že v takýchto prípadoch musíme upraviť tento výraz tak, aby obe jeho strany boli v poriadku. Dá sa to urobiť napr. takto:

```
  N := 1;
  Faktorial := 1;
  while Faktorial <= MaxInt div N do
  begin
    Faktorial := Faktorial * N;
    N := N + 1;
  end;
  Image1.Canvas.TextOut(50, 50, 'max = ' + IntToStr(N));
```

Podobný princíp použijeme aj v nasledujúcom príklade.

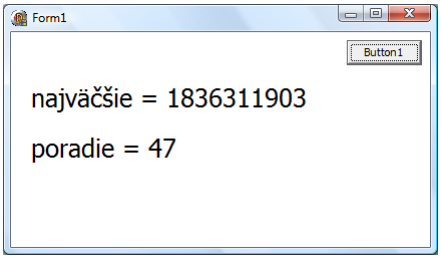
Vypočítajte a vypíšte pod seba prvých 15 Fibonacciho čísel. Fibonacciho postupnosť začína číslami 0 a 1. Každé ďalšie vypočítame ako súčet dvoch predchádzajúcich v postupnosti. Teda prvých niekoľko členov postupnosti vyzerá: 0, 1, 1, 2, 3, 5, 8, 13, ... Keďže Fibonacciho postupnosť rastie tiež veľmi rýchlo, bude zaujímavé vypočítať najväčšie také, ktoré ešte nie je väčšie ako **MaxInt**. Ak by sme to chceli riešiť takto

```
var
  Prve, Druhe, Nove, Pocit: Integer;
begin
  Prve := 0;
  Druhe := 1;
  Pocit := 2;
  Nove := 1;
  while Prve + Druhe <= MaxInt do
  begin
    Nove := Prve + Druhe;
    Prve := Druhe;
    Druhe := Nove;
    Pocit := Pocit + 1;
  end;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(20, 50, 'najväčšie = ' + IntToStr(Nove));
  Image1.Canvas.TextOut(20, 100, 'poradie = ' + IntToStr(Pocit));
end;
```

Urobili sme rovnaký problém ako pri faktoriáli: while-cyklus má stále pravdivú podmienku a ak máme zapnutú kontrolu pretečenia, tak spadne na chybovej správe. Musíme to teda prepísať:

```
var
  Prve, Druhe, Nove, Pocit: Integer;
begin
  Prve := 0;
  Druhe := 1;
  Pocit := 2;
  Nove := 1;
  while Prve <= MaxInt - Druhe do
  begin
    Nove := Prve + Druhe;
    Prve := Druhe;
    Druhe := Nove;
    Pocit := Pocit + 1;
  end;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(20, 50, 'najväčšie = ' + IntToStr(Nove));
  Image1.Canvas.TextOut(20, 100, 'poradie = ' + IntToStr(Pocit));
end;
```

a po spustení sa dozvieme, že najväčšie Fibonacciho číslo, ktoré sa dá vypočítať pomocou celočíselnej aritmetiky, je štyridsiate siedme:



Úlohy na precvičenie

Zadanie 1	program zistí, koľkokrát je zadané číslo deliteľné napr. číslom 3. T.j. chceme zistiť, akou najväčšou mocninou čísla 3 je zadané číslo deliteľné; napr. číslo 405 môžeme tromi deliť 4-krát
Zadanie 2	sústredné kružnice budú mať postupne polomer od najväčšej R=120; druhá má polomer o 5 menší, tretia o 10 menší od druhej, štvrtá o 15 menší od tretej, atď. kružnice treba prestať kresliť, ak by mali mať záporný polomer
Zadanie 3	na podobnom princípe, ako sme počítali najväčší faktoriál, zistite najväčšiu mocninu nejakého čísla **N**, ktorá nie je väčšia ako **MaxInt** (**N** nech je konštanta programu napr. 3)

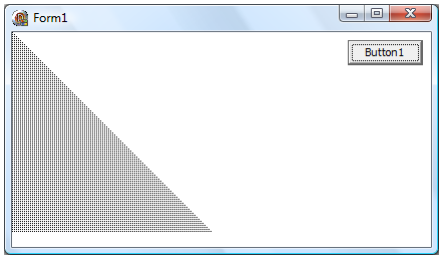
Zadanie 4	program by chcel vygenerovať dve navzájom rôzne náhodné čísla z intervalu <0, 9>, ale urobili sme v ňom chybu; opravte ju
	`A := Random(10);`
	`while Random(10) <> A do`
	`  B := Random(10);`
	`Image1.Canvas.TextOut(0, 0,`
	`  IntToStr(A)+' '+IntToStr(B));`

Zadanie 5

dvojica príkazov **MoveTo(X, Y); LineTo(X, Y+1);** nakreslí bodku na súradniciach (X, Y); zistite, koľko bodiek nakreslí tento program

```
X := 0;
Y := 0;
while Y < 100 do
begin
  Image1.Canvas.MoveTo(2 * X, 2 * Y);
  Image1.Canvas.LineTo(2 * X, 2 * Y + 1);
  X := X + 1;
  if X > Y then
  begin
    X := 0;
    Y := Y + 1;
  end;
end;
```

Možno vám pomôže, keď viete, že obrázok vyzerá takto:



2. Znovu cifry celého čísla

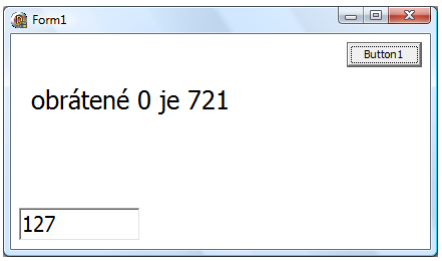
Keď sme sa zoznámili s for-cyklom, riešili sme úlohu, v ktorej sme obrátili poradie cifier celého čísla. V tejto úlohe bol problém, že pre for-cyklus sme museli zadať presný počet opakovaní cyklu a teda, ak bolo cifier menej ako prechodov cyklu, obrátené číslo sa sprava doplnilo nulami:

```
Nove := 0;
for I := 1 to 9 do
begin
  Nove := Nove * 10 + Cislo mod 10;
  Cislo := Cislo div 10;
end;
```

Tento cyklus pri obracaní čísla 12345 vyrobil číslo 543210000. Teraz tu už môžeme použiť **while**-cyklus, ktorý nepotrebuje vedieť počet opakovaní, ale podmienku, pokým sa majú nejaké príkazy opakovať:

```
var
  Cislo, Nove: Integer;
begin
  Cislo := StrToInt(Edit1.Text);
  Nove := 0;
  while Cislo <> 0 do
  begin
    Nove := Nove * 10 + Cislo mod 10;
    Cislo := Cislo div 10;
  end;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(20, 50,
      'obrátené ' + IntToStr(Cislo) + ' je ' + IntToStr(Nove));
end;
```

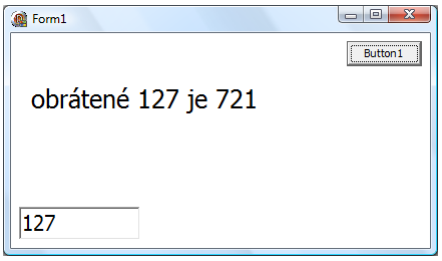
Program už teraz naozaj správne obráti poradie cifier aj bez zbytočných núl, len ten výpis nie je úplne v poriadku:



Program naozaj spracoval číslo 127: obrátil ho, ale vypísať sme chceli aj premennú **Cislo** aj premennú **Nove**. Lenže premennú **Cislo** sme v cykle menili: odhadzovali sme z nej po poslednej cifre, až kým nebola nulová. Preto sa aj táto 0 objavila vo výpise. Pri takýchto úlohách treba mať na pamäti, že niektoré premenné môžu v cykle stratiť svoju pôvodnú hodnotu, a preto si ju často musíme zapamätať v nejakej ***pomocnej premennej***, napr. takto

```
var
  Cislo, Nove, Pomoc: Integer;
begin
  Cislo := StrToInt(Edit1.Text);
  Nove := 0;
  Pomoc := Cislo;
  while Pomoc <> 0 do
  begin
    Nove := Nove * 10 + Pomoc mod 10;
    Pomoc := Pomoc div 10;
  end;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(20, 50,
      'obrátené ' + IntToStr(Cislo) + ' je ' + IntToStr(Nove));
end;
```

Teraz by mal byť výpis v poriadku:



V tomto konkrétnom prípade sme pomocnú premennú nemuseli použiť vôbec, lebo vo výpise stačilo vypisovať nie číslo, ale **Edit1.Text**:

```
Image1.Canvas.TextOut(20, 50,
    'obrátené ' + Edit1.Text + ' je ' + IntToStr(Nove));
```

Pozrime sa ešte na jednu konkrétnu vstupnú hodnotu a tou je číslo 0. Program dá správnu odpoveď (obrátené 0 je 0), aj keď v tomto prípade **while**-cyklus neprebehol ani raz. Zapamätajte si, že cyklus nemusí prejsť ani raz a to vtedy, keď je podmienka nesplnená už pred prvým prechodom.

Zistite, či program funguje správne aj pre zápornú hodnotu **Cislo**. Odtrasujte si to na nejakej malej hodnote, napr. -127.

V tomto prvom programe cyklus prebehol presne toľkokrát, koľko cifier malo zadané číslo. Pritom máme postupne od konca čísla k dispozícii všetky cifry vďaka zvyšku po delení (**Cislo mod 10**). Ukážme si niekoľko základných algoritmov, ktoré pracujú s ciframi celého čísla. Programy budeme uvádzať bez deklarácií.

Program zistí počet cifier teda, koľko je ciferné:

```
Cislo := StrToInt(Edit1.Text);
Pocet := 0;
while Cislo > 0 do
begin
  Pocet := Pocet + 1;
  Cislo := Cislo div 10;
end;
Image1.Canvas.TextOut(20, 50, 'počet cifier = ' + IntToStr(Pocet));
```

Ďalší program zistí počet výskytov cifry 7. Napr. v 1727374757 obsahuje 5 cifier 7.

```
Cislo := StrToInt(Edit1.Text);
Pocet := 0;
while Cislo > 0 do
begin
  if Cislo mod 10 = 7 then
    Pocet := Pocet + 1;
  Cislo := Cislo div 10;
end;
Image1.Canvas.TextOut(20, 50, 'počet 7 = ' + IntToStr(Pocet));
```

Teraz zistíme počet núl na konci nejakého celého čísla. Počítadlo tohto počtu budeme zvyšovať o jedna, len kým je na konci 0 a zakaždým túto nulu odhodíme (vydelíme 10):

```
Cislo := StrToInt(Edit1.Text);
Pocet := 0;
if Cislo <> 0 then
  while Cislo mod 10 > 0 do
  begin
    Pocet := Pocet + 1;
    Cislo := Cislo div 10;
  end;
Image1.Canvas.TextOut(20, 50, '0 na konci = ' + IntToStr(Pocet);
```

Niekedy potrebujeme zistiť nie poslednú (čo je jednoduché **Cislo mod 10**) ale prvú cifru nejakého čísla:

```
Cislo := StrToInt(Edit1.Text);
while Cislo div 10 > 0 do
  Cislo := Cislo div 10;
Image1.Canvas.TextOut(20, 50, 'prvá cifra = ' + IntToStr(Cislo));
```

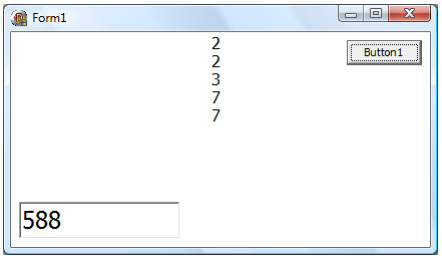
Nasledujúci algoritmus je dosť užitočný, lebo rozkladá ľubovoľné celé číslo na prvočinitele - to sú všetky prvočíselné delitele. Postupne sa pokúša deliť zadané číslo stále väčšími a väčšími deliteľmi a pričom skončí, keď číslo vydelí najväčším možným deliteľom:

```
var
  Cislo, Delitel, Y: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Y := 0;
  Cislo := StrToInt(Edit1.Text);
  Delitel := 2;
  Image1.Canvas.Font.Height := 20;
  while Cislo > 1 do
    if Cislo mod Delitel > 0 then
      Delitel := Delitel + 1
    else
    begin
      Image1.Canvas.TextOut(200, Y, IntToStr(Delitel));
      Cislo := Cislo div Delitel;
      Y := Y + 18;
    end;
end;
```

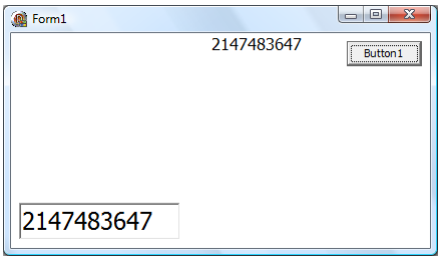
Odtrasujme tento algoritmus napríklad pre vstupné **Cislo** = 588:

Cislo	**Delitel**	**Cislo mod Delitel > 0**	**výpis**
588	2	pravda	2
284		pravda	2
147		nepravda	
	3	pravda	3
49		nepravda	
	4	nepravda	
	5	nepravda	
	6	nepravda	
	7	pravda	7
7		pravda	7
1			

Aj po spustení a zadaní 588 dostávame rovnaký výsledok:



Tento program nám umožní zistiť, že najväčšie celé číslo **MaxInt** je vlastne prvočíslo:



Úlohy na precvičenie

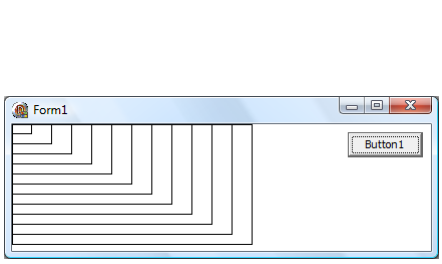
Zadanie 1	program vyhodí zo zadaného čísla všetky výskyty cifry 7, napr. z 127377457 vyrobí číslo 12345
Zadanie 2	program v zadanom čísle všetky výskyty cifry 0 nahradí cifrou 7, napr. z čísla 120300450 vyrobí číslo 127377457
Zadanie 3	program zistí, koľkokrát sa dá vydeliť zadané číslo nejakým iným zadaným číslom, napr. pre dvojicu 3072 a 2 vypíše 10, lebo $3072=2^{10}*3$
Zadanie 4	program zistí najväčšiu cifru v zadanom čísle, napr. pre 120123011 je to cifra 3
Zadanie 5	program vypočíta a vypíše ciferný súčet zadaného čísla, potom preverí pravidlo, podľa ktorého je číslo deliteľné číslom 3, ak 3 delí jeho ciferný súčet

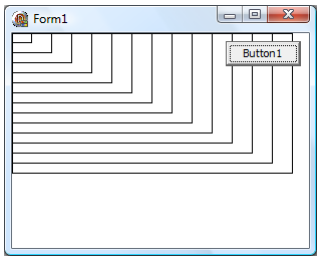
3. Cyklus so zložitejšími podmienkami

Potrebujeme nakresliť čo najviac zväčšujúcich sa obdĺžnikov, ktorých jeden vrchol je v 0, 0 a pritom protiľahlý nebude mimo grafickej plochy. Obdĺžniky majú postupne rozmery 20x10, 40x20, 60x30, ... t.j. výška sa zväčšuje o 10 a šírka je dvojnásobok výšky. Aby sa celý obdĺžnik zmestil do grafickej plochy, bude treba zabezpečiť, aby **Sirka ≤ Image1.Width** a zároveň **Vyska ≤ Image1.Height**. Aby platili súčasne oba tieto testy, budeme potrebovať operátor **and**. Zapíšme program:

```
var
  Sirka, Vyska: Integer;
begin
  Image1.Canvas.Brush.Style := bsClear;  // priesvitné obdĺžniky
  Sirka := 20;
  Vyska := 10;
  while (Sirka <= Image1.Width) and (Vyska <= Image1.Height) do
  begin
    Image1.Canvas.Rectangle(0, 0, Sirka, Vyska);
    Sirka := Sirka + 20;
    Vyska := Vyska + 10;
  end;
end;
```

a podľa rozmeru grafickej plochy sa nám nakreslí rôzny počet obdĺžnikov:



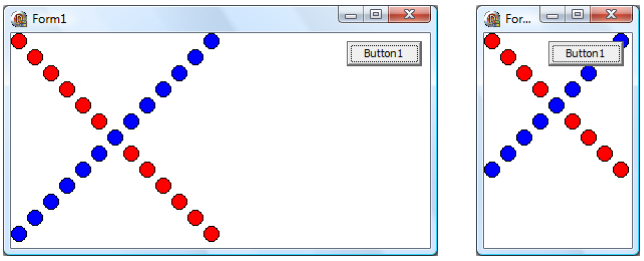


Cyklus **while** skončí, keď bude podmienka nesplnená, t.j. keď bude nepravdivá **(Sirka <= Image1.Width) and (Vyska <= Image1.Height)**. Už vieme, že to nastane vtedy, keď buď **Sirka < Image1.Width** alebo **Vyska < Image1.Height**, t.j. keď **(Sirka > Image1.Width) or (Vyska > Image1.Height)**.

Len pre porovnanie uvádzame ešte jednu verziu tohto istého programu, v ktorej používame len jednu premennú **Vyska**, nakoľko šírka sa z nej dá ľahko vypočítať ako **2*Vyska**:

```
var
  Vyska: Integer;
begin
  Image1.Canvas.Brush.Style := bsClear;  // priesvitné obdĺžniky
  Vyska := 10;
  while (2 * Vyska <= Image1.Width) and (Vyska <= Image1.Height) do
  begin
    Image1.Canvas.Rectangle(0, 0, 2 * Vyska, Vyska);
    Vyska := Vyska + 10;
  end;
end;
```

Podobný test bude aj v nasledovnom príklade. Nakreslíme rad červených kruhov z ľavého horného rohu šikmo smerom vpravo dolu, až kým tento rad nenarazí na spodný alebo pravý okraj grafickej plochy. Potom nakreslí druhý šikmý rad modrých kruhov, ktorý je zrkadlový k prvému. Podľa veľkosti grafickej plochy by mohli výstupy z programu vyzerať asi takto:



Pozrite uvedený program:

```
const
  R = 8;
var
  X, Y, Sirka, Vyska: Integer;
begin
  X := R;
  Y := R;
  Sirka := Image1.Width;
  Vyska := Image1.Height;
  Image1.Canvas.Brush.Color := clRed;
  while (X < Sirka - R) and (Y < Vyska - R) do
  begin
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
    X := X + 2 * R;
    Y := Y + 2 * R;
  end;
  X := X - 2 * R;
  Y := R;
  Image1.Canvas.Brush.Color := clBlue;
  while X > 0 do
  begin
    Image1.Canvas.Ellipse(X - R, Y - R, X + R, Y + R);
    X := X - 2 * R;
    Y := Y + 2 * R;
  end;
end;
```

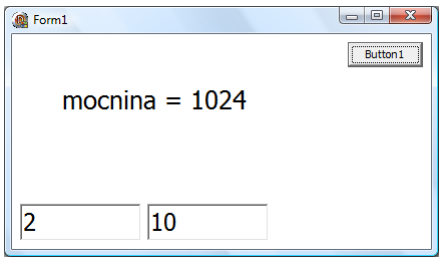
Všimnite si, že druhý while-cyklus má oveľa jednoduchšiu podmienku. Preskúmajte,

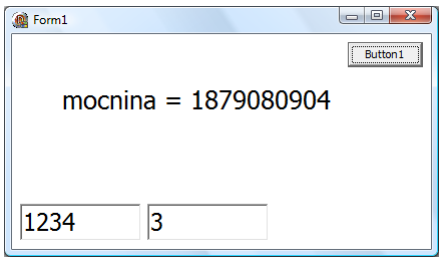
prečo.

Nasledovný program je opäť veľmi užitočný, lebo okrem toho, že počíta K-tu mocninu celého čísla N, zároveň kontroluje, či toto číslo nepresiahne maximálnu hodnotu **MaxInt**.

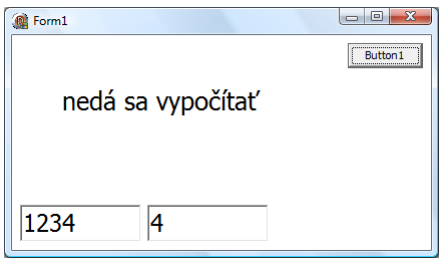
```
var
  N, K, Mocnina: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  N := StrToInt(Edit1.Text);
  K := StrToInt(Edit2.Text);
  Mocnina := 1;
  while (K > 0) and (Mocnina <= Maxint div N) do
  begin
    Mocnina := Mocnina * N;
    K := K - 1;
  end;
  Image1.Canvas.Font.Height := 30;
  if K <> 0 then
    Image1.Canvas.TextOut(50, 50, 'nedá sa vypočítať')
  else
    Image1.Canvas.TextOut(50, 50, 'mocnina = ' + IntToStr(Mocnina));
end;
```

Ak zadáme vhodné hodnoty, dostávame správny výsledok:





Program správne zareaguje správou, keď dostane na vstupe príliš veľké hodnoty, napr. 1234^4 = 2 318 785 835 536:

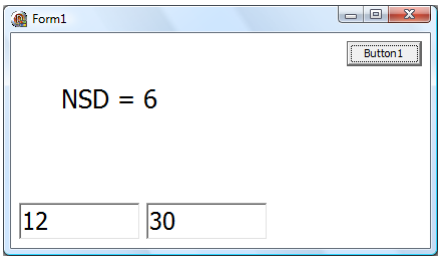


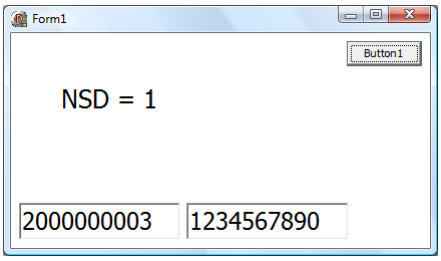
Iste všetci z matematiky poznáme pojem ***najväčší spoločný deliteľ***. **NSD** pre dve zadané hodnoty, je také najväčšie číslo, ktoré delí obe vstupné hodnoty. Napr. NSD(12, 30) = 6. Najprv prvé riešenie, v ktorom postupne kontroluje všetky čísla počnúc prvým zo vstupu až po 1. Podmienka while-cyklu je nastavená tak, aby cyklus skončil vtedy, keď nájdené číslo delí obe vstupné hodnoty. Teda pre **A** a **B** cyklus skončí, keď pre nájdené **NSD** platí **(A mod NSD = 0) and (B mod NSD = 0)**. V tele cyklu sa robí jediný príkaz, znižovanie premennej **NSD** o 1:

```
var
  A, B, NSD: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  B := StrToInt(Edit2.Text);
  NSD := A;
  while (A mod NSD <> 0) or (B mod NSD <> 0) do
    NSD := NSD - 1;

  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(50, 50, 'NSD = ' + IntToStr(NSD));
end;
```

Vidíme, že podmienka while-cyklu je naozaj negáciou podmienky, pri ktorej má cyklus skončiť. Preveríme ho na nejakých vstupných údajoch:





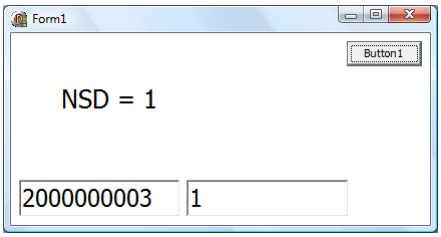
Takémuto algoritmu hovoríme, že je veľmi pomalý, lebo v prípade, že obe čísla sú nesúdeliteľné (ich **NSD** je 1), cyklus bude **A**-krát neúspešne kontrolovať podmienku cyklu. A pre veľké **A** to bude naozaj veľa.

Našťastie veľký matematik Euklides vymyslel oveľa rýchlejší algoritmus. Tento ***Euklidov algoritmus*** vychádza z toho, že ak A > B, tak NSD(A, B) = NSD(A-B, B). Ak budeme tento vzťah stále opakovať, tak sa hodnoty budú postupne znižovať až sa budú navzájom rovnať:

```
var
  A, B: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  B := StrToInt(Edit2.Text);
  while A <> B do
    if A > B then
      A := A - B
    else
      B := B - A;

  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(50, 50, 'NSD = ' + IntToStr(A));
end;
```

Zdá sa, že je tento algoritmus výrazne rýchlejší, ako ten predchádzajúci. Hoci stále má jeden nedostatok. Ak je napr. medzi **A** a **B** veľký rozdiel, napr.

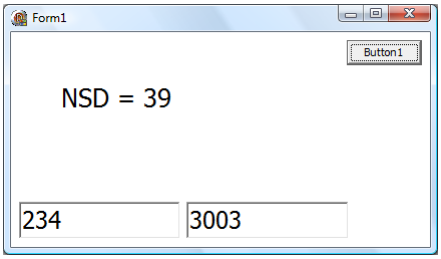


tak, v cykle sa 2000000002-krát odčituje od 2000000003 číslo 1, pritom z matematického hľadiska sa vlastne len počíta A mod B. Euklidov algoritmus preto vylepšíme takto:

```
var
  A, B, Pom: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  A := StrToInt(Edit1.Text);
  B := StrToInt(Edit2.Text);
  while B <> 0 do
  begin
    Pom := A mod B;
    A := B;
    B := Pom;
  end;

  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(50, 50, 'NSD = ' + IntToStr(A));
end;
```

Odporúčame ho odtrasovať pre rôzne dvojice hodnôt, napr. pre



trasovacia tabuľka môže vyzerať takto:

A	B
234	3003
3003	234
234	195
195	39
39	0

Vidíme, že algoritmus príde na výsledok veľmi rýchlo.

Úlohy na precvičenie

Zadanie 1	pre lotériu treba vygenerovať 3 náhodné ale navzájom rôzne čísla z intervalu <1, 10> - prvé z nich je úplne náhodné, druhé generujete dovtedy, kým sa nebude líšiť od prvého, na záver generujete tretie, ktoré musí byť rôzne od prvého aj od druhého; Vedeli by ste to naprogramovať aj pre 4 rôzne náhodné čísla?
Zadanie 2	vypočítajte NSN (najväčší spoločný násobok) dvoch čísel, t.j. také najmenšie číslo, ktoré je deliteľné aj jedným aj druhým
Zadanie 3	program zistí najväčšie **N**, pre ktoré platí, že **N!** nie je väčší ako 1000000

4. Zapamätanie si výsledku podmienky

Doteraz sme pracovali s podmienkami v príkazoch **while** alebo **if**, a hovorili sme, že sú to ***logické výrazy***, ktorých hodnota je pravda alebo nepravda. Hodnoty logických výrazov môžeme ukladať aj do premenných, len tieto premenné nemôžu byť celočíselného typu (**Integer**), ale musia byť ***logického typu***, tzv. typ **Boolean**. Tieto premenné potom môžeme použiť v logických výrazoch a v rôznych podmienkach. V pamäti takéto premenné zaberajú 1 bajt. Pripomíname, že typ Integer zaberá v

pamäti 4 bajty. Logický typ má preddefinované dve svoje konštanty pre pravdu **True** a pre nepravdu **False**.

Ukážme si, ako to deklarujeme a ako to môžeme používať:

```
var
  I: Integer;
  B, B1: Boolean;
begin
  B := True;
  I := 13;
  B := I > 10;
  B1 := (I mod 3 = 1) and B;
  while B1 do
    B1 := not B1;
```

Samotná časť programu nerobí nič rozumné, len predvádza logické výrazy v rôznych situáciách.

Zopakujme si ešte logické operácie:

`not a`	negácia	`True`, ak `a=False`
`a and b`	logický súčin	`True`, ak zároveň `a=True` aj `b=True`
`a or b`	logický súčet	`True`, ak aspoň pre jedno `a=True` alebo `b=True`

Už sme sa zoznámili s tým, že všetky relačné operácie (menší, rovný, väčší, ...) porovnávajú dve čísla a vrátia **True** alebo **False**, podľa výsledku porovnania. Porovnávať môžeme aj logické hodnoty. Hoci je to dosť neobvyklé, ale skúsení programátori to raz začas aj využijú. Treba si zapamätať, že

```
False < True
```

a potom nás neprekvapia ani výrazy napr.:

```
(I < 5) > (J > 10)
(I = 5) = (J = 10)
(I <= 5) <= (J <= 10)
```

Existujú dve užitočné funkcie na prácu s logickými hodnotami:

```
Ord(logická_hodnota)
```

vráti 0 alebo 1, podľa toho, či logická hodnota bola **False** alebo **True**

```
Boolean(číslo)
```

vráti **True** alebo **False** podľa toho či číslo je 0 alebo 1

Sú to konverzné funkcie. Prvá funkcia **Ord** umožňuje z logickej hodnoty získať jej vnútornú reprezentáciu: v počítači sú logické hodnoty uložené buď ako 0 (**False**) alebo 1 (**True**). Druhá funkcia **Boolean** konvertuje celé číslo na logickú hodnotu, t.j. je to opačná funkcia k **Ord**.

Pripomeňme si príklad, v ktorom sme počítali **K**-tu mocninu nejakého celého čísla **N**:

```
Mocnina := 1;
while (K > 0) and (Mocnina <= Maxint div N) do
begin
  Mocnina := Mocnina * N;
  K := K - 1;
end;
if K <> 0 then
  Image1.Canvas.TextOut(50, 50, 'nedá sa vypočítať')
```

Pomocou pomocnej logickej premennej to môžeme zapísať aj inak:

```
var
  N, K, Mocnina: Integer;
  OK: Boolean;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  N := StrToInt(Edit1.Text);
  K := StrToInt(Edit2.Text);
  Mocnina := 1;
  OK := True;
  while OK and (K > 0) do
  begin
    OK := Mocnina <= Maxint div N;
    if OK then
      Mocnina := Mocnina * N;
    K := K - 1;
  end;
  Image1.Canvas.Font.Height := 30;
  if OK then
    Image1.Canvas.TextOut(50, 50, 'mocnina = ' + IntToStr(Mocnina))
  else
    Image1.Canvas.TextOut(50, 50, 'nedá sa vypočítať');
end;
```

Toto riešenie nie je nijako lepšie ako predchádzajúce, len ilustruje použitie logickej premennej. Preštudujte toto riešenie.

Predtým sme tiež riešili úlohu, v ktorej sa zisťoval počet výskytov cifry 7. Ak nepotrebujeme vedieť počet výskytov, ale len to, či sa tam aspoň jedna 7 vyskytuje, môžeme to prepísať takto:

```
  Cislo := StrToInt(Edit1.Text);
  Vyskyt7 := False;
  while Cislo > 0 do
  begin
    if Cislo mod 10 = 7 then
      Vyskyt7 := True;
    Cislo := Cislo div 10;
  end;
  if Vyskyt7 then
    Image1.Canvas.TextOut(20, 50, 'v čísle je aspoň jedna 7')
  else
    Image1.Canvas.TextOut(20, 50, 'v čísle nie je žiadna 7');
```

Už skôr sme sa zoznámili s prioritami aritmetických operácií. Teraz ju doplníme aj o logické a relačné operácie:

```
not
*  /  div  mod  and
+  -  or
=  <  <=  >  >=  <>
```

Z tejto tabuľky vidíme, že porovnávanie hodnôt má najnižšiu prioritu, preto sa budú vyhodnocovať ako posledné. Pre výraz **X + 50 < 400** je to výhoda: najprv sa vypočíta súčet **X + 50** a až potom sa tento súčet porovná so 400. Keďže ale logické operátory **and** a **or** majú vyššiu prioritu ako relačné, tak výrazy bez zátvoriek narobia veľké problémy. Pozrime si výraz:

```
X < 400 and Y < 250
```

Keďže je bez zátvoriek, tak **and** má vyššiu prioritu ako relácia < a pascal chce najprv vypočítať **400 and Y** a tento výsledok porovnať, či je väčšie ako X. My sme to ale takto nemysleli, preto tento výraz zapíšeme aj so zátvorkami:

```
(X < 400) and (Y < 250)
```

Mali by sme si zapamätať, že vždy keď budeme zapisovať zložitejší výraz, ktorý obsahuje relačné a logické operátory, dáme si veľký pozor na priority operácií a nezabudneme správne použiť zátvorky.

Úlohy na precvičenie

Zadanie 1	program pri rozoberaní celého zadaného čísla, zistí či sú jeho cifry v neklesajúcej postupnosti, napr. číslo 1222566 spĺňa túto podmienku
Zadanie 2	program nakreslí na náhodné pozície 200 obdĺžnikov s náhodnými rozmermi z intervalu <20, 40>; na záver vypíše, či bol medzi nimi aspoň jeden štvorec

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- príkazy vetvenia **if** a **case**, podmienky,
- cyklus s podmienkou
- zložité podmienky, logické operácie, priorita operátorov

Okrem toho boli uvedené tieto dôležité koncepty:

- pravidlá formátovania programu
- rozoberanie čísla na cifry
- prvočísla, prvočinitele, Euklidov algoritmus

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Pre zadané celé číslo (pomocou vstupného riadku) do grafickej plochy vypíše tieto informácie:

- počet vlastných deliteľov (okrem 1 a samého seba)
- najväčší vlastný deliteľ
- ciferný súčet a počet cifier v 10-ovej sústave
- či je to prvočíslo

Literatúra a použité zdroje

Základné materiály:

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

Internetové zdroje pre prostredie Delphi

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

Internetové zdroje pre prostredie Lazarus

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

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

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

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

Podnázov Programovanie 3

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti doc. RNDr. Gabriela Andrejková, CSc.
RNDr. Ladislav Huraj, PhD.

Počet strán 40

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 978-80-89225-58-3