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 6

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 6

Identifikácia modulu

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

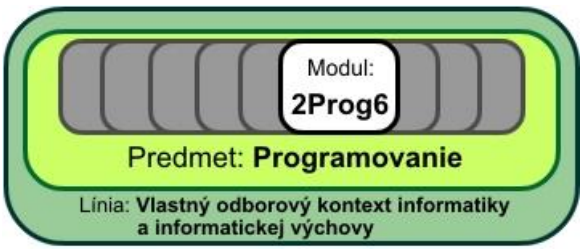
Garant predmetu:

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

Autori:

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

Zaradenie modulu



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

Druhý semester vzdelávania:

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

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

Abstrakt modulu

Modul sa venuje základným typom Char a znakový reťazec. Obsahuje množstvo riešených úloh na spracovávanie znakových reťazcov. Druhá polovica modulu sa venuje algoritmom na spracovanie udalostí myši v grafickej ploche.

Obsah

vod

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

1. Znaky a znakové reťazce
2. Algoritmy s myšou

Cieľ modulu

Prvá tematická jednotka **Znaky a znakové reťazce** pokrýva tieto témy:

- Znakový typ Char
- Znakové reťazce
- Výber znaku, podreťazec
- Pozície a podreťazce

Druhá tematická jednotka **Algoritmy s myšou** pokrýva tieto témy:

- Klikanie
- Pohyb myši
- Ťahanie
- Časovač a myš

Vstupné vedomosti

Požadované prerekvizity

Všetky moduly Programovanie 1 až 5.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie popísať princíp volania procedúry aj s parametrami,
- vie posúdiť umiestnenie procedúry a potrebu definovania parametrov,
- dokáže analyzovať zadanie a tiež možné chyby v riešení, chyby vie nájsť a opraviť,
- dokáže vytvoriť aplikácie, ktoré používajú skôr definované procedúry,
- rozumie rozdielom medzi lokálnymi a globálnymi premennými, vie posúdiť ich viditeľnosť v kóde.

1. tematická jednotka - Znaky a znakové reťazce

1. Znakový typ Char

V tejto časti sa zoznámime s ďalším zo základných typov – typ znak. Premenná tohto typu si bude môcť zapamätať práve jeden znak, napr. písmeno, číslicu ale aj iné znaky:

A B C … a b c … 0 1 2 … á ä č ď … @ # $ % … + - / * = < > ( ) [ ] { } …

Znakové konštanty zapisujeme medzi dva apostrofy, napr. toto sú znakové konštanty

'a', 'B', '3', '#', ' ', '.', 'ž', 'ô'.

Na rozdiel od celočíselného aj logického typu s týmto typom nemôžeme robiť žiadne operácie. Môžeme akurát nejakú znakovú hodnotu priradiť do premennej, môžeme ju porovnať s inou hodnotou a znakovú hodnotu môžeme ešte vypísať napr. do grafickej plochy.

V pamäti počítača je pre každú takúto premennú vyhradený jeden bajt (8 bitov) a preto sa do tejto premennej "zmestí" 256 rôznych hodnôt. Tieto hodnoty sú navzájom usporiadané, preto ich môžeme navzájom porovnávať nielen na rovnosť, ale aj reláciami menší a väčší. Vnútorne v počítači sú znaky kódované tzv. ASCII kódovaním: každému znaku prislúcha kód, čo je číslo od 0 do 255. Pascal potom pri porovnávaní dvoch znakov pozrie na ich ASCII kódy a ten znak, ktorý má menší tento kód je považovaný za menší.

Nemá zmysel si pamätať celú ASCII tabuľku, ale oplatí sa vedieť niekoľko kódov. Napr.

- kód znaku '**A**' je 65
- kód znaku '**a**' je 97
- kód znaku '**0**' je 48
- kód znaku medzera ' ' je 32

Ak viem, že kódy cifier idú za sebou a kód pre '0' je 48, pre '1' je 49, pre '2' je 50, tak by sme vedeli vypočítať, že kód pre '9' musí byť 57. Podobne je to s písmenami. Všetkých 26 písmen anglickej abecedy ide tesne za sebou. Ak prvé písmeno 'A' má kód 65, potom 26. písmeno 'Z' musí mať kód 90.

Typ znak sa v Pascale označuje slovom **Char**. Nasledujúci program ilustruje priradenie znaku do premennej a potom výpis do plochy pomocou **TextOut**:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Z: Char;
  I: Integer;
begin
  Image1.Canvas.Font.Height := 30;
  Z := 'G';
  Image1.Canvas.TextOut(50, 70, Z);
  Z := '@';
  for I := 1 to 10 do
    Image1.Canvas.TextOut(25*I, 100, Z);
  Z := 'A';
  if Z < '4' then
    Image1.Canvas.TextOut(0, 130, 'znak A je menší ako znak 4')
  else
    Image1.Canvas.TextOut(0, 130, 'znak A nie je menší ako znak 4');
end;
```

Program najprv vypíše do plochy písmeno G, potom pomocou for-cyklu desaťkrát znak @ a na záver navzájom porovná znaky 'A' a '4' a vypíše o tom správu:

Pascal má ešte niekoľko funkcií, ktoré vedia pracovať so znakmi. Dve najdôležitejšie sú:

- funkcia **Ord**(znak) vráti vnútorný kód znaku (ASCII kód), t.j. celé číslo z intervalu <0, 255>
- funkcia **Char**(číslo) je opačná k **Ord**: ak je celé číslo z intervalu <0, 255>, tak vráti prislúchajúci znak (pre túto funkciu sa môžete stretnúť aj s názvom **Chr**)

Napr. platia tieto zápisy:

```
Ord('a') = 97
Char(65) = 'A'
Char(Ord('1')+1) = '2'
Ord(' ') = 32
```

V programovacom jazyku Pascal budeme o niektorých typoch hovoriť, že sú ordinálne – to znamená, že každá konštanta tohto typu má svojho predchodcu aj nasledovníka. Samozrejme, že okrem minimálnej a maximálnej konštanty. Napr. celočíselný typ **Integer**, aj logický typ **Boolean** a aj znakový typ **Char** sú ordinálne typy. Reálny typ **Real** nie je ordinálnym typom. Ordinálne typy majú v Pascale isté špeciálne postavenie:

- počítadlo (riadiaca premenná) for-cyklu musí byť ordinálneho typu,
- hodnota, na základe ktorej sa príkaz **case** rozhodne, ktorú vetvu vykoná, musí byť ordinálneho typu,
- existujú štandardné príkazy **Inc** a **Dec**, pomocou ktorých vieme zvýšiť alebo znížiť obsah ordinálnej premennej,
- štandardné funkcie **Low** a **High** vrátia konštanty pre minimálnu a maximálnu povolenú hodnotu príslušného ordinálneho typu, napr.
  - **Low**(**Integer**) = -2147483648
  - **High**(**Integer**) = 2147483647
  - **Low**(**Char**) = **Char**(0)
  - **High**(**Char**) = **Char**(255)
  - **Low**(**Boolean**) = **False**
  - **High**(**Boolean**) = **True**
- neskôr uvidíme, že indexom polí musí byť ordinálny typ,
- pokročilí programátori vedia, že bázový typ množín musí byť ordinálnym typom.

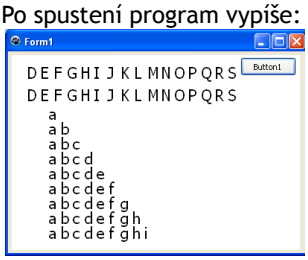
Štandardné procedúry **Inc** a **Dec** zvyšujú, resp. znižujú obsah premennej. Napr.

- ak má celočíselná premenná **I** hodnotu 17, tak **Inc(I);** ju zvýši o 1 (v premennej **I** je teraz číslo 18), **Inc(I, 5);** ju zvýši o 5 (v premennej **I** je teraz číslo 23)
- ak má celočíselná premenná **J** hodnotu 22, tak **Dec(J);** ju zníži o 1 (v premennej **J** je teraz číslo 21), **Dec(J, 3);** ju zníži o 3 (v premennej **J** je teraz číslo 18)
- ak znaková premenná **Z** má hodnotu 'A', tak **Inc(Z);** ju zvýši o 1, t.j. v **Z** je teraz 'B', a ďalšie **Inc(Z, 4);** ju zvýši o ďalšie 4 – teraz je v **Z** znak 'F'
- podobne to funguje aj pre **Dec** so znakovou premennou, napr. ak v **Z** je písmeno 'e' (s kódom 101), tak **Dec(Z, 32);** priradí do **Z** písmeno 'E' (z kódom 69)

Ukážeme použitie znakového typu ako riadiacej premennej vo for-cykle. Uvedomte si, že v takomto cykle počítadlo bude postupne nadobúdať všetky hodnoty od dolnej hranice až po hornú.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  V, Z: Char;
  I, X, Y: Integer;
begin
  Image1.Canvas.Font.Name := 'Tahoma';
  Image1.Canvas.Font.Height := 26;
  Image1.Canvas.Brush.Style := bsClear; // výpis s priesvitným pozadím

  X := 20;
  for Z := 'D' to 'S' do
  begin
    Image1.Canvas.TextOut(X, 10, Z);
    Inc(X, 18);                 // znamená X := X + 18;
  end;

  Z := 'D';
  for I := 1 to 16 do
  begin
    Image1.Canvas.TextOut(18*I+2, 40, Z);
    Inc(Z);
  end;

  Y := 65;
  for V := 'a' to 'i' do
  begin
    X := 50;
    for Z := 'a' to V do
    begin
      Image1.Canvas.TextOut(X, Y, Z);
      Inc(X, 16);
    end;
    Inc(Y, 20);
  end;
end;
```

Po spustení program vypíše:



Prvý cyklus vypíše do jedného riadka všetky znaky od 'D' až po 'S'. Druhý cyklus robí to isté, ale riadiacou premennou cyklu je celočíselná premenná **I** a vypisujeme tu premennú **Z**, ktorú v tele cyklu zvyšujeme na nasledovníka. Na záver sú dva vnorené for-cykly: oba majú riadiacu premennú znakového typu. Všimnite si, že vnútorný cyklus má hornú hranicu závislú od riadiacej premennej vonkajšieho cyklu.

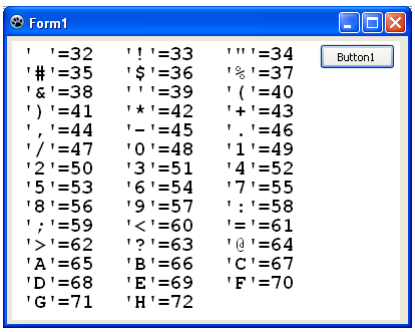
Teraz, keď už vieme, že for-cyklus funguje nielen pre celé čísla, ale aj pre iné ordinálne typy, precvičte si túto úlohu. Zistite, koľkokrát prejde telo týchto cyklov (pre **I, X: Integer; B: Boolean; Z: Char**):

- **for I := 1 to 100 do …**
- **for I := -10 to 10 do …**
- **for I := 20 to 20 do …**
- **for B := False to True do …**
- **for B := True to False do …**
- **for Z := 'A' to 'Z' do …**
- **for b := x=1 to x=2 do** … ak celočíselná premenná **x** má hodnotu buď 0, alebo 1 alebo 2

Ďalší program nám ukáže ASCII kódy niekoľkých znakov:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Z: Char;
  X, Y: Integer;
begin
  Image1.Canvas.Font.Name := 'Courier New';
  Image1.Canvas.Font.Height := 23;
  Image1.Canvas.Font.Style := [fsBold];

  X := 10;
  Y := 40;
  for Z := ' ' to 'H' do
  begin
    Image1.Canvas.TextOut(X, Y, '''' + Z + '''=' + IntToStr(Ord(Z)));
    Inc(X, 100);
    if X + 80 > Image1.Width then
    begin
      X := 10;
      Inc(Y, 19);
    end;
  end;
end;
```

Po spustení dostávame:



V programe si treba všimnúť tieto dve veci: pre výpis textu pomocou **TextOut** sme nastavili "tučné písmo" (Bold) - pre písmo (**Font**) použili sme nové nastavenie **Style**, ktorému priradíme buď

- prázdne hranaté zátvorky [] - vtedy zrušíme nastavenie rezu písma
- do hranatých zátvoriek jedno alebo viac slov oddelených čiarkou: **fsBold**, **fsItalic**, **fsUnderline**, ktoré znamenajú tučné písmo, kurzívu a podčiarknutie.

Druhou dôležitou vecou v tomto programe je spôsob, ako sme vypísali znak a jeho ASCII kód:

- najprv je štvorica apostrofov `''''` - toto je spôsob ako vytvoríme konštantu znak apostrof, totiž do apostrofov musíme uzavrieť dva apostrofy a vtedy to bude znamenať jeden apostrof,
- ďalej nasleduje znak **Z**,
- za tým reťazec `'''='`, t.j. jeden zdvojený apostrof a znak rovná sa,
- konverzia **IntToStr** s parametrom **Ord**(Z).

Takže sa vypíše jeden apostrof, za tým samotný znak, potom apostrof so znakom rovná sa a na záver číslo, ktoré zodpovedá ASCII kódu.

Na záver ešte niekoľko užitočných faktov o znakovom type:

- znakovú konštantu môžeme zapísať aj pomocou *#číslo* – kde *číslo* je nejaký ASCII kód, napr. #65 označuje znak 'A', #32 je medzera;
- funkcia **UpCase** z malého písmena abecedy spraví zodpovedajúce veľké (ostatné znaky sa pritom nemenia), napr.

```
UpCase('e') = 'E'
UpCase('M') = 'M'
UpCase('%') = '%'
UpCase(#97) = 'A'
```

Úlohy na precvičenie

Zadanie 1	Napíšte program, ktorý prevedie zadané číslo (zo vstupného riadka) do 16-ovej sústavy. Využite znakový typ, pomocou ktorého viete jednoducho napr. cifru 10 previesť na znak 'A'.
Zadanie 2	Napíšte program, v ktorom bude na tlačidle **Button1** jednoznakový text 'A'. Po každom zatlačení tlačidla sa na tlačidle objaví nasledujúci znak abecedy. Po stlačení tlačidla so znakom 'Z' program skončí (**Close**).
Zadanie 3	Napíšte program, ktorý po zatlačení tlačidla vypíše na náhodnú pozíciu náhodné písmeno veľkej abecedy.

2. Znakové reťazce

Typ znakový reťazec slúži na uchovávanie textov a na prácu s týmito textami. Preto je to oveľa komplexnejší typ, ako doterajšie základné typy. Premenná typu znakový reťazec môže obsahovať ľubovoľnú postupnosť znakov. My už vieme, že znaky, t.j. hodnoty typu **Char**, v počítači zaberajú jeden bajt a kódované sú tzv. ASCII kódovaním. Premenné typu znakové reťazce ale nemajú dopredu určenú svoju dĺžku. Táto závisí podľa toho, akú hodnotu sme tej ktorej premennej priradili.

Znakové reťazce, rovnako ako typ znak (**Char**), majú svoje konštanty, ktoré zapisujeme do apostrofov. Už sme sa s týmito konštantami stretli pri príkaze na výpis do textovej plochy **TextOut** ale aj pri nastavení textu na tlačidle (**Button**) alebo pri jednoduchom texte (**Label**):

```
Image1.Canvas.TextOut(50, 100, 'Lazarus');
Button1.Caption := 'Štart';
Label1.Caption := 'Vyhral si';
```

Teraz uvidíme ako môžeme takéto texty uchovávať v premenných, ako ich môžeme meniť a ako s nimi budeme pracovať. V prvom rade musíme mať zadeklarované premenné typu znakový reťazec. Typ sa nazýva **string** a do premennej tohto typu sa zmestí text, ktorý je obmedzený prakticky len operačnou pamäťou počítača (presnejšie do 2 miliárd znakov).

Nasledujúci príklad ukazuje, ako zadeklarujeme premennú **Slovo**, ako do nej priradíme nejaký reťazec a ako tento reťazec vypíšeme do plochy:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Slovo: string;
  X, Y: Integer;
begin
  Randomize;
  Image1.Canvas.Font.Name := 'Arial';
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.Brush.Color := clLtGray;

  X := Random(300);
  Y := Random(200);
  if X < 150 then
    Slovo := 'Lazarus'
  else
    Slovo := 'Pascal';
  if Y < 100 then
    Image1.Canvas.Font.Color := clRed
  else
    Image1.Canvas.Font.Color := clBlue;
  Image1.Canvas.TextOut(X, Y, Slovo);
end;
```

Ak program spustíme a zatlačíme niekoľkokrát tlačidlo, dostaneme približne takýto obrázok:

Texty sa tu vypisujú náhodne, ale v hornej polovici budú červené a v dolnej modré. V ľavej polovici sa vypisuje slovo **Lazarus** a v pravej **Pascal**.

Ešte jeden komponent, ktorý už poznáme, pracuje so znakovým reťazcom - je to vstupný riadok. Tento komponent slúži na zadávanie nejakého vstupu do programu a my ho využívame najmä na zadávanie čísel. Vtedy musíme túto hodnotu prekonvertovať na celé číslo pomocou konverznej funkcie **StrToInt**:

Cislo := StrToInt(Edit1.Text);

Takže hodnotou **Edit1.Text** je znakový reťazec, ktorý sme zadali v komponente vstupný riadok. Tento môžeme nielen priradiť do premennej typu znakový reťazec, ale vidíme, že sa znakový reťazec dá prekonvertovať na celé číslo pomocou **StrToInt**. Podobne, keď sme chceli vypisovať čísla do grafickej plochy, tak sme príkazu **TextOut** museli toto číslo prekonvertovať:

Image1.Canvas.TextOut(50, 100, IntToStr(Cislo));

Okrem tejto konverzie sme sa naučili, že ak chceme pomocou **TextOut** naraz vypísať viac textov, musíme ich spojiť pomocou operácie +. Takto môžeme vidieť, že existuje operácia, ktorá vie spájať reťazce do jedného reťazca, napr. v jednom z našich programov sme napísali:

```
Image1.Canvas.TextOut(30, 100,
        'súčet od 1 do ' + IntToStr(N) + ' je ' + IntToStr(Sucet));
```

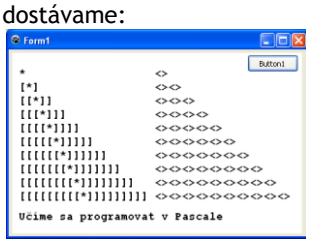
a v ďalšom

```
Image1.Canvas.TextOut(X, Y, '"' + Z + '"=' + IntToStr(Ord(Z)));
```

Táto operácia + sa nazýva zreťazenie a pomocou nej vieme kratšie reťazce pospájať do dlhších. Ďalší príklad ukazuje spôsob zreťazovania reťazcov:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A, B, C, Retazec: string;
  I: Integer;
begin
  Image1.Canvas.Font.Name := 'Courier New';
  Image1.Canvas.Font.Height := 20;
  Image1.Canvas.Font.Style := [fsBold];

  A := 'vat v Pascale';
  B := 'a programo';
  C := 'Učíme s';
  Image1.Canvas.TextOut(10, 230, C + B + A);

  Retazec := '*';
  for I := 1 to 10 do
  begin
    Image1.Canvas.TextOut(10, 20*I, Retazec);
    Retazec := '[' + Retazec + ']';
  end;

  Retazec := '';
  for I := 1 to 10 do
  begin
    Retazec := Retazec + '<>';
    Image1.Canvas.TextOut(210, 20*I, Retazec);
  end;
end;
```

Po spustení programu dostávame:



Program najprv vypíše zreťazenie troch reťazcov. Potom v prvom prechode cyklu vypíše hviezdičku, k tejto hviezdičke "*priraďuje*" hranaté zátvorky zľava aj sprava, teda vyrobí reťazec '[*]' a znovu to vypíše. Opäť k tomuto z oboch strán priraďuje hranaté zátvorky, dostáva reťazec '[[*]]' a tento vypíše. Toto opakuje viackrát. Posledný for-cyklus robí niečo podobné, ale začína s prázdnym reťazcom:

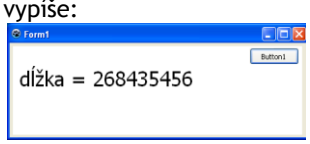
```
Retazec := '';
```

To znamená, že v premennej **Retazec** je znakový reťazec dĺžky 0. K tomuto sa priraťazí reťazec '<>' a vypíše sa. Opäť sa priraťazí '<>', dostávame '<><>' a tiež sa to vypíše. Toto sa opakuje viackrát.

Pomocou for-cyklu vieme vyrobiť aj dosť dlhé reťazce, napr.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Retazec: String;
  I, Dlzka: Integer;
begin
  Image1.Canvas.Font.Height := 38;

  Retazec := '#';
  for I := 1 to 28 do
    Retazec := Retazec + Retazec;

  Dlzka := Length(Retazec);
  Image1.Canvas.TextOut(10, 30, 'dĺžka = ' + IntToStr(Dlzka));
end;
```

A skutočne, po spustení vypíše:



Použili sme tu ďalšiu reťazcovú funkciu **Length**, ktorá nám vráti momentálnu dĺžku reťazca. Pred samotným for-cyklom ma reťazec dĺžku 1. Po prvom prechode cyklom, vyrobíme dvojznakový reťazec '##', ďalší prechod ho opäť zdvojnásobí, teda reťazec už obsahuje '####'. Keďže cyklus beží 28-krát, po skončení by mal mať dĺžku 2^{28}.

To znamená, že tento program na chvíľu zabral vyše štvrť gigabajtu pamäti počítača.

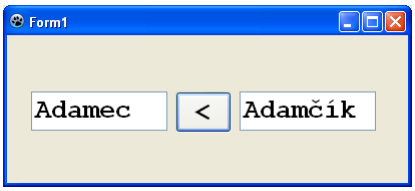
Reťazce môžeme navzájom porovnávať. Môžeme nielen zistiť či sa dva rovnajú, ale aj to, či je jeden menší, ako druhý. Hovoríme, že znakový reťazec je **lexikograficky** väčší ako iný reťazec, ak je v abecednom usporiadaní neskôr (napr. telefónny zoznam je usporiadaný podľa abecedy). V pascalovských reťazcoch sa ale rozlišujú malé a veľké písmená a zrejme veľké písmená sú v abecede skôr ako tie isté, ale malé.

Teraz napíšeme program, ktorý bude porovnávať dva zadané reťazce a oznámi nám, či je prvý menší ako druhý, alebo druhý je menší ako prvý, alebo sa oba rovnajú:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Edit1.Text < Edit2.Text then
    Button1.Caption := '<'
  else if Edit1.Text > Edit2.Text then
    Button1.Caption := '>'
  else
    Button1.Caption := '=';
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
  Button1.Caption := '?';
end;

procedure TForm1.Edit2Change(Sender: TObject);
begin
  Button1.Caption := '?';
end;
```

Vo formulári máme dva komponenty vstupný riadok (**Edit1** a **Edit2**) a jedno tlačidlo (**Button1**). Na tlačidlo sme umiestnili text "?". Využívame tu dve nové udalosti **onChange**, ktoré patria komponentom **Edit1** a **Edit2** (keď sa dvojklikne vo formulári na tieto komponenty, vytvorí sa pre ne prázdna procedúra **Edit1Change** a **Edit2Change**). Táto udalosť vzniká vtedy, keď sme niečo vo vstupnom riadku zmenili (napr. sme pripísali písmeno).

Program očakáva, že do oboch vstupných riadkov vpíšeme nejaký reťazec a zatlačíme tlačidlo **Button1**. Vtedy by sa malo vyhodnotiť, ktorý reťazec je menší a na tlačidlo sa vypíše zodpovedajúce relačné znamienko, napr.



Úlohy na precvičenie

Zadanie 1	Program zaplní grafickú, resp. textovú plochu postupne číslami od 1 do 1000 a to tak, že každý vypisovaný riadok obsahuje maximálne 40 znakov. Ak by mal byť riadok dlhší, tak ho nepredlží, ale ho vypíše a začne skladať ďalší riadok (znakový reťazec). Nastavte font na 'Courier New'.
Zadanie 2	Program postupne všetky čísla od 1 do 100 rozloží na prvočinitele a vypíše ich do riadkov textovej plochy v takomto tvare: 20=2*2*5 21=3*7 22=2*11 23=23
Zadanie 3	Program vypíše do textovej plochy z hviezdičiek vytvorený štvorec veľkosti NxN aj s oboma jeho uhlopriečkami. N je zadané vo vstupnom riadku. Nastavte font na 'Courier New'.

3. Výber znaku, podreťazec

So znakovými reťazcami môžeme pracovať aj po jednotlivých znakoch. Ak máme premennú znakový reťazec, môžeme pristupovať ku konkrétnemu znaku tak, že index, t.j. poradové číslo znaku, uzavrieme do hranatých zátvoriek. Znaky v znakovom reťazci sú očíslované od 1 pre prvý znak až po **Length** pre posledný znak. Napr. máme reťazcovú premennú **R**

R := 'reťazec';

Tento reťazec má dĺžku 7 (funkcia **Length(R)** by vrátila hodnotu 7) a preto indexom môže byť číslo od 1 do 7, napr.

R[4] má hodnotu 'a'
R[Length(R)] je posledný znak v reťazci, teda 'c'
R[I] je i-ty znak, závisí od hodnoty premennej **I** -hodnota musí byť z <1, 7>

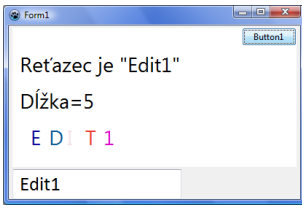
Index môžeme zadať nielen celočíselnou konštantou, ale aj premennou alebo aj aritmetickým výrazom. Dôležité je, aby to malo hodnotu z intervalu <1, **Length(R)**>. Takto indexovaný znak môžeme použiť nielen pri výpise, ale môžeme ho aj zmeniť pomocou obyčajného priradenia. Napr. pre premennú **R**, v ktorej je slovo 'reťazec', séria priradení:

```
R[3] := R[1];                 // v R[3] je teraz 'r', R = 'rerazec'
R[2] := Char(Ord(R[4])+1);    // v R[2] je teraz 'b', R = 'rbrazec"
R[1] := 'O';                  // v R[1] je teraz 'O', R = 'Obrazec'
```

zmení tento reťazec na slovo 'Obrazec'.

Teraz ukážeme niekoľko príkladov, na ktorých môžeme vidieť spôsob práce s reťazcovými premennými. Vo väčšine týchto príkladov, budeme nejako pracovať s jednotlivými znakmi reťazca a najčastejšie pritom využijeme for-cyklus. V programoch predpokladáme, že máme komponent **Edit1** - vstupný riadok:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Retazec: string;
  I: Integer;
begin
  Image1.Canvas.FillRect(Image1.ClientRect);   // zmaže plochu
  Image1.Canvas.Font.Height := 38;
  Image1.Canvas.Font.Color := clBlack;

  Retazec := Edit1.Text;
  Image1.Canvas.TextOut(10, 30, 'Reťazec je "' + Retazec + '"');
  Image1.Canvas.TextOut(10, 80, 'Dĺžka=' + IntToStr(Length(Retazec)));
  Randomize;
  for I := 1 to Length(Retazec) do
  begin
    Image1.Canvas.Font.Color := Random(256*256*256);
    Image1.Canvas.TextOut(I*25, 130, UpCase(Retazec[I]));
  end;
end;
```



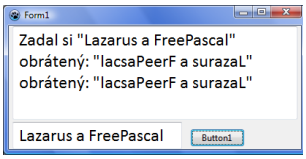
Postupne berie z reťazca znak za znakom, zakaždým nastaví náhodnú farbu písma a vypíše príslušný znak. Pritom malé písmená abecedy prerobí na veľké.

V ďalších programoch nahradíme grafickú plochu textovou (**Memo1**).

Pripomeňme si, že obsah textovej plochy vyčistíme príkazom **Memo1.Clear**.
Jeden riadok do plochy vypíšeme pomocou príkazu **Memo1.Lines.Append**.

Program zoberie zadaný text a vyrobí z neho reťazec, v ktorom budú znaky v opačnom poradí. Urobili sme to dvomi rôznymi spôsobmi:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Retazec, R1, R2: string;
  I, Dlzka: Integer;
begin
  Memo1.Clear;

  Retazec := Edit1.Text;
  Memo1.Lines.Append('Zadal si "' + Retazec + '"');

  Dlzka := Length(Retazec);
  R1 := '';
  for I := 1 to Dlzka do
    R1 := Retazec[I] + R1;
  Memo1.Lines.Append('obrátený: "' + R1 + '"');

  SetLength(R2, Dlzka);
  for I := 1 to Dlzka do
    R2[I] := Retazec[Dlzka-I+1];
  Memo1.Lines.Append('obrátený: "' + R2 + '"');
end;
```



Po spustení a zadaní nejakého textu môžeme vidieť, že oba spôsoby dávajú rovnaké výsledky.

Prvý cyklus postupne pridáva z pôvodného reťazca do nového **R1** po jednom znaku, ale všimnite si, že tento nový znak sa pridá pred už existujúce znaky v **R1** a preto sa tu poskladajú v opačnom poradí.

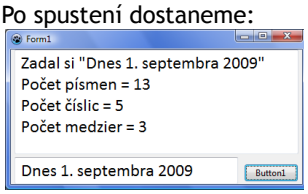
Druhý cyklus pracuje inak. Najprv sa pripraví nový reťazec **R2**, ktorému nastavíme počiatočnú dĺžku pomocou štandardného príkazu **SetLength** - tento príkaz zmení dĺžku reťazca na zadanú, pričom ak sa reťazec pritom zväčší, tak pridá nejaké dopredu nedefinované znaky. Vtedy sa predpokladá, že tieto znaky prepíšeme priraďovacími príkazmi - naozaj v cykle postupne do týchto znakov priraďujeme znaky z pôvodného reťazca. Všimnite si, že pri priraďovaní znakov indexujeme pôvodný reťazec komplikovaným vzorcom: **Dlzka-I+1**. Tento vzorec práve zabezpečí, že znaky sa budú priraďovať v opačnom poradí. Priebeh tohto cyklu môžeme skontrolovať aj takouto tabuľkou (predpokladáme, že pôvodný reťazec obsahuje text 'abcd' a preto **Dlzka** je 4):

	Retazec	I	Dlzka-I+1	R2
SetLength(r2, dlzka)	'abcd'	?	?	'????'
1. prechod for-cyklu		1	4	'd???'
2. prechod for-cyklu		2	3	'de??'
3. prechod for-cyklu		3	2	'dcb?'
4. prechod for-cyklu		4	1	'dcba'

Nasledujúci program počíta počet písmen, číslic a medzier v nejakom texte:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R: string;
  I, PocetPismen, PocetCislic, PocetMedzier: Integer;
begin
  Memo1.Clear;

  R := Edit1.Text;
  Memo1.Lines.Append('Zadal si "' + R + '"');

  PocetPismen := 0;
  PocetCislic := 0;
  PocetMedzier := 0;
  for I := 1 to Length(R) do
  begin
    if (R[I] >= 'A') and (R[I] <= 'Z') or
       (R[I] >= 'a') and (R[I] <= 'z') then
      Inc(PocetPismen);
    if (R[I] >= '0') and (R[I] <= '9') then
      Inc(PocetCislic);
    if R[I] = ' ' then
      Inc(PocetMedzier);
  end;

  Memo1.Lines.Append('Počet písmen = ' + IntToStr(PocetPismen));
  Memo1.Lines.Append('Počet číslic = ' + IntToStr(PocetCislic));
  Memo1.Lines.Append('Počet medzier = ' + IntToStr(PocetMedzier));
end;
```

Po spustení dostaneme:

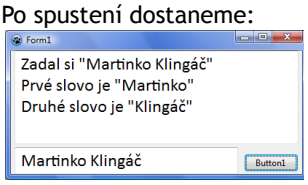


Vo for-cykle postupne prechádzame všetky znaky a pre každý zvlášť (**R[I]**) zisťujeme, či je to písmeno, číslica alebo medzera.

Ďalší program predpokladá, že sme zadali text s dvoma slovami, ktoré sú oddelené jednou medzerou. Nájde v texte túto medzeru a rozdelí ho na dva reťazce: prvé a druhé slovo.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R, Slovo1, Slovo2: string;
  I: Integer;
  BolaMedzera: Boolean;
begin
  Memo1.Clear;

  R := Edit1.Text;
  Memo1.Lines.Append('Zadal si "' + R + '"');

  Slovo1 := '';
  Slovo2 := '';
  BolaMedzera := False;
  for I := 1 to Length(R) do
    if BolaMedzera then
      Slovo2 := Slovo2 + R[I]
    else if R[I] = ' ' then
      BolaMedzera := True
    else
      Slovo1 := Slovo1 + R[I];

  Memo1.Lines.Append('Prvé slovo je "' + Slovo1 + '"');
  Memo1.Lines.Append('Druhé slovo je "' + Slovo2 + '"');
end;
```

Po spustení dostaneme:

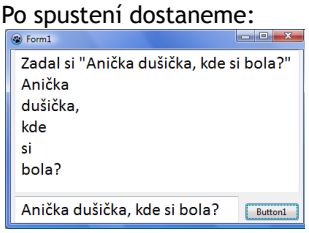


Všimnite si v tomto programe použitie logickej premennej **BolaMedzera**. Pomocou nej v cykle vieme, či príslušný znak dávame do prvého alebo druhého slova.

Veľmi podobný bude nasledujúci program: zadáme nejakú vetu, v ktorej môže byť aj viac slov. Program potom túto vetu rozseká na jednotlivé slová a každé vypíše do samostatného riadka:

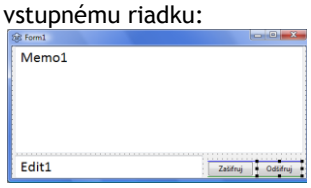
```
procedure TForm1.Button1Click(Sender: TObject);
var
  Veta, Slovo: string;
  I: Integer;
begin
  Memo1.Clear;

  Veta := Edit1.Text;
  Memo1.Lines.Append('Zadal si "' + Veta + '"');

  Slovo := '';
  for I := 1 to Length(Veta) do
    if Veta[I] <> ' ' then
      Slovo := Slovo + Veta[I]
    else if Slovo <> '' then
    begin
      Memo1.Lines.Append(Slovo);
      Slovo := '';
    end;

  if Slovo <> '' then
    Memo1.Lines.Append(Slovo);
end;
```

Po spustení dostaneme:



V tomto for-cykle sa každý nemedzerový znak prekopíruje do reťazca **Slovo**. Ak vo vstupnej vete nájdeme medzeru, tak to môže znamenať, že sme práve ukončili jedno slovo a treba ho vypísať. Lenže, mohlo sa stať, že vo vstupnej vete bolo medzi slovami viac medzier za sebou. Ak by sme tam nedali test, či je **Slovo** neprázdne (**Slovo** je rôzne od prázdneho reťazca), tak by sa vypisovali aj takéto prázdne slová - vyskúšajte, ako bude vyzerať výpis, ak tento test odstránime a vo vete dáme medzi slová viac medzier. Všimnite si, že sme museli dať ešte jeden výpis premennej **Slovo** aj za cyklus, keďže za posledným slovom nemusela byť vo vete medzera a slová sa v cykle vypisujú len, keď sa objaví medzera.

Do formulára pridáme ešte jedno tlačidlo a zmeníme texty aj tlačidlám aj vstupnému riadku:



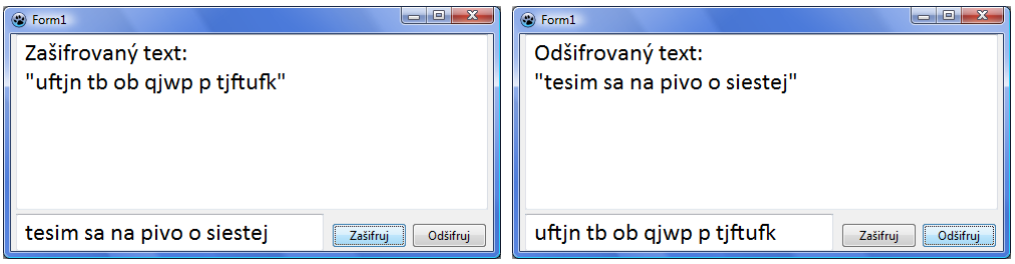
Prácu s reťazcami môžeme využiť aj na jednoduché zašifrovanie textu: Zadaný text zašifrujeme tak, že každé písmeno posunieme o jednu pozíciu v abecede dopredu, t.j. namiesto 'a' vypíšeme 'b', namiesto 'b' vypíšeme 'c', atď. až namiesto 'z' vypíšeme 'a'. V programe budeme pracovať len s malými písmenami a samozrejme, že bez diakritiky.

Program pre prvé tlačidlo zadaný text zašifruje, druhé tlačidlo zase zadaný text odšifruje. Obe procedúry sú veľmi podobné:

```
procedure TForm1.Button1Click(Sender: TObject);     // zašifruj
var
  Veta: string;
  I: Integer;
begin
  Memo1.Clear;

  Veta := Edit1.Text;
  for I := 1 to Length(Veta) do
    if Veta[I] = 'z' then
      Veta[I] := 'a'
    else if (Veta[I] >= 'a') and (Veta[I] < 'z') then
      Veta[I] := Char(Ord(Veta[I])+1);

  Memo1.Lines.Append('Zašifrovaný text:');
  Memo1.Lines.Append('"' + Veta + '"');
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);     // odšifruj
var
  Veta: string;
  I: Integer;
begin
  Memo1.Clear;

  Veta := Edit1.Text;
  for I := 1 to Length(Veta) do
    if Veta[I] = 'a' then
      Veta[I] := 'z'
    else if (Veta[I] >= 'a') and (Veta[I] < 'z') then
      Veta[I] := Char(Ord(Veta[I])-1);

  Memo1.Lines.Append('Odšifrovaný text:');
  Memo1.Lines.Append('"' + Veta + '"');
end;
```

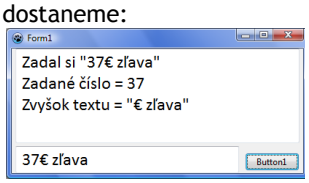
Po spustení najprv zadáme nejaký text a stlačíme tlačidlo **Zašifruj**: vypíše sa náš text zašifrovaný a môžeme ho poslať svojmu kolegovi. Ten zase tento zašifrovaný text zadá do tohto istého programu a stlačí tlačidlo **Odšifruj**. Vypíše sa mu naša pôvodná správa:



Ďalší program zo zadaného reťazca prečíta celé číslo, pričom vypíše zvyšok tohto reťazca. Konverzná funkcia **StrToInt** funguje len vtedy, keď reťazec obsahuje len číslo a žiadne iné znaky:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  R, Zvysok: string;
  I, J, Cislo, Cifra: Integer;
begin
  Memo1.Clear;

  R := Edit1.Text;
  Memo1.Lines.Append('Zadal si "' + R + '"');

  Cislo := 0;
  I := 1;
  while (I <= Length(R)) and (R[I] >= '0') and (R[I] <= '9') do
  begin
    Cifra := Ord(R[I]) - Ord('0');
    Cislo := Cislo*10 + Cifra;
    Inc(I);
  end;

  Zvysok := '';
  for J := i to Length(R) do
    Zvysok := Zvysok + R[J];

  Memo1.Lines.Append('Zadané číslo = ' + IntToStr(Cislo));
  Memo1.Lines.Append('Zvyšok textu = "' + Zvysok + '"');
end;
```

Po spustení zadáme nejaký text, ktorý začína číslom a dostaneme:



Použili sme while-cyklus, ktorý testuje, či je daný znak cifra. Ak áno, zoberie sa tento znak, prerobí sa na číslo a pridá sa ku vytváranej premennej **Cislo**. Stačí si uvedomiť, že pre znak '0' s kódom **Ord('0')**, potrebujeme získať cifru 0, pre znak '1' s kódom **Ord('1')**, čo je o jedna viac ako **Ord('0')** potrebujeme získať cifru 1, atď.

Okrem tohto, while-cyklus zisťuje aj to, či index **I** nie je už väčší ako dĺžka reťazca. Tento prípad by mohol nastať vtedy, keby sme do editovacieho riadka zadali samotné číslo bez znakov navyše. Lenže tu by mohol nastať problém. Predpokladajme, že sme zadali len dvojznakový reťazec '37' a ideme skontrolovať, ako sa prevedie na číslo:

1. pred while-cyklom sa priradí 0 do **Cislo** a 1 do premennej **I**;
2. testuje sa podmienka cyklu while:
   - **(I <= Length(R))** je pravdivé
   - **(R[I] >= '0')** je pravdivé
   - **(R[I] <= '9')** je pravdivé
3. vykoná sa telo cyklu a preto premenné: **Cislo** = 3, **I** = 2
4. opäť sa testuje podmienka cyklu while:
   - **(I <= Length(R))** je pravdivé
   - **(R[I] >= '0')** je pravdivé
   - **(R[I] <= '9')** je pravdivé
5. vykoná sa telo cyklu a preto premenné: cislo = 37, i = 3
6. opäť sa testuje podmienka cyklu while:
   - **(I <= Length(R))** nie je pravdivé
   - **(R[I] >= '0')** sa nedá otestovať, lebo **I** je viac ako dĺžka reťazca
   - **(R[I] <= '9')** sa nedá otestovať, lebo **I** je viac ako dĺžka reťazca

A toto je ten problém. V jednej podmienke chceme naraz testovať to, či je index v dobrom intervale a zároveň tento index používať na prístup ku znaku.

Našťastie FreePascal pri vyhodnocovaní podmienok, v ktorých sú logické operácie **and**, resp. **or**, nebude vyhodnocovať zvyšok logického výrazu, ak je už jasný výsledok. Napr. v našom príklade, počítač nebude vyhodnocovať zvyšok výrazu, ak je **(I <= Length(R))** nepravdivé.

V prostredí Delphi toto tiež funguje, ale to treba nastaviť vo voľbách kompilátora.

Úlohy na precvičenie

Zadanie 1	Program zo vstupného riadka prečíta tri slová oddelené medzerou a vypíše všetky permutácie týchto troch slov.
Zadanie 2	Program číta zo vstupného riadka dvojice čísel oddelené medzerou. Tieto čísla reprezentujú súradnice bodov v grafickej ploche. Program takto definované body spája úsečkami.
Zadanie 3	Program v zadanej vete všetky samohlásky nahradí nejakou konkrétnou samohláskou. Napr. pre samohlásku `'i'` by z vety `'Sedi mucha na stene'` spravil `'Sidi michi ni stini'`.
Zadanie 4	Ak máme v premennej (alebo v konštante) **R1** reťazec 'aeiou', tak potom **R1[Random(Length(R1))+1]** náhodne vyberie jeden znak. Ak budeme mať v inom reťazci **R2**, napr. nejaké spoluhlásky, tiež z neho môžeme náhodne nejaké vyberať. Napíšte program, ktorý si bude vymýšľať nové slová, napr. zložené z náhodných znakov: spoluhláska + samohláska + spoluhláska a tieto bude vypisovať do textovej plochy. Navrhnite aj iné pravidlá na "vymýšľanie" nových slov.

4. Pozície a podreťazce

V predchádzajúcich častiach sme sa zoznámili s týmito procedúrami a funkciami, ktoré nejako spracovávajú znakové reťazce:

- **TextOut**(x, y, text); // vypíše text do plochy
- **IntToStr**(číslo) // konvertuje číslo na reťazec
- **StrToInt**(text) // konvertuje reťazec na číslo
- **Length**(text) // vráti dĺžku reťazca
- **SetLength**(premenná, číslo); // nastaví novú dĺžku reťazcovej premennej

Pascal má definovaných oveľa viac veľmi užitočných funkcií a procedúr. Ak s nimi budeme vedieť pracovať, veľmi nám to pomôže pri programovaní. Neskôr sa naučíme vytvárať aj svoje vlastné reťazcové funkcie, ale zatiaľ sa naučme aspoň niekoľko najdôležitejších, ktoré sú už naprogramované profesionálmi.

Najdôležitejšou reťazcovou funkciou je **Copy**, ktorá vie z reťazca vybrať nejaký "***podreťazec***". Znamená to, že z daného reťazca **R** nemusíme vyberať len jeden znak **R[I]** a ten ďalej spracovávať, ale môžeme z neho vybrať aj nejakú súvislú časť. Funkcia **Copy** má 3 parametre: prvým je samotný reťazec, z ktorého chceme vyberať nejakú časť. Druhým parametrom je index, na ktorom začína tento podreťazec a tretím parametrom je dĺžka tohto podreťazca. Najlepšie bude, keď to predvedieme na príkladoch:

```
Copy('Jánošík', 3, 2) = 'no'
Copy('Jánošík', 2, 10) = 'ánošík'    // menej ako 10 znakov
Copy('Jánošík', 8, 2) = ''     // prázdny reťazec
Copy('Jánošík', 5, 1) = 'š'    // jednoznakový reťazec - nie Char
```

Vidíme, že funkcia funguje dobre aj vtedy, keď požadujeme väčší podreťazec, ako ich ešte ostalo v reťazci - vtedy nám vráti všetky znaky od počiatočného indexu až dokonca. Veľmi často sa stretáme so zápisom:

```
Copy(Retazec, I, MaxInt)
```

ktorý označuje, že z daného reťazca Retazec berieme od I-teho znaku všetky až dokonca. Ak napríklad chceme vyhodiť znak na I-tej pozícii, môžeme zapísať

```
Copy(Retazec, 1, I-1) + Copy(Retazec, I+1, MaxInt)
```

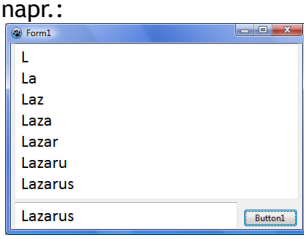
To znamená, že výsledný reťazec dostaneme ako spojenie dvoch reťazcov: znaky od začiatku až po znak pred I-tym a reťazec od nasledovného za I-tym až dokonca. Podobne zapíšeme, ak chceme I-ty (teda jeden znak) nahradiť viacerými, napr. reťazcom '...':

```
Copy(Retazec,1,I-1) + '...' + Copy(Retazec,I+1 MaxInt)
```

Prvý jednoduchý príklad vypíše zadaný text viackrát pod seba: najprv iba prvý znak, potom prvé dva, prvé tri a až celý reťazec:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Veta: string;
  I: Integer;
begin
  Memo1.Clear;
  Veta := Edit1.Text;
  for I := 1 to Length(Veta) do
    Memo1.Lines.Append(Copy(Veta, 1, I));
end;
```

Po spustení dostávame, napr.:

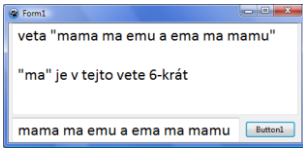


Všimnime si, že v cykle nevypisujeme celý reťazec, ale len jeho prvých i znakov.

Ukážeme, ako môžeme použiť funkciu **Copy** na zisťovanie počtu výskytov nejakého podreťazca. Napr. chceme vo vete 'mama ma emu a ema ma mamu' zistiť, koľkokrát sa v nej objavila dvojica znakov 'ma':

```
procedure TForm1.Button1Click(Sender: TObject);
const
  slabika = 'ma';
var
  Veta: string;
  I, Pocet: Integer;
begin
  Memo1.Clear;
  Veta := Edit1.Text;
  Memo1.Lines.Append('veta "' + Veta + '"');
  Pocet := 0;
  for I := 1 to Length(Veta) do
    if Copy(Veta, I, Length(Slabika)) = Slabika then
      Inc(Pocet);

  Memo1.Lines.Append('');
  Memo1.Lines.Append('"' + Slabika + '" je v tejto vete ' +
    IntToStr(Pocet) + '-krát');
end;
```

Ak po spustení programu zadáme našu vetu, dostaneme:



Podobnú činnosť, ako náš predchádzajúci príklad, robí ďalšia štandardná reťazcová funkcia **Pos**. Táto funkcia má za úlohu zistiť, kde sa vo vete nachádza nejaké konkrétne slovo. Ak je tam takýchto výskytov viac, tak funkcia oznámi prvý z nich. Funkcia hľadá index prvého (najľavejšieho) výskytu nejakého podreťazca v zadanom reťazci. Ak sa tam podreťazec nevyskytuje, funkcia vráti 0. Opäť niekoľko príkladov:

```
Pos('em', 'mama ma emu a ema ma mamu') = 9
Pos('emu', 'mama ma emu a ema ma mamu') = 15
Pos('Ema', 'mama ma emu a ema ma mamu') = 0
Pos(' ', 'mama ma emu a ema ma mamu') = 5
Pos('x', 'mama ma emu a ema ma mamu') = 0
Pos('raka', 'straka') = 3
```

Ukážme možné riešenie predchádzajúceho príkladu použitím funkcie **Pos**:

```
procedure TForm1.Button1Click(Sender: TObject);
const
  slabika = 'ma';
var
  Veta: string;
  Index, Pocet: Integer;
begin
  Memo1.Clear;
  Veta := Edit1.Text;
  Memo1.Lines.Append('veta "' + Veta + '"');
  Pocet := 0;
  Index := 1;
  while Index <> 0 do
  begin
    Index := Pos(Slabika, Veta);
    if Index <> 0 then
    begin
      Inc(Pocet);
      Veta := Copy(Veta, Index+1, MaxInt);
    end;
  end;

  Memo1.Lines.Append('');
  Memo1.Lines.Append('"' + Slabika + '" je v tejto vete ' +
    IntToStr(Pocet) + '-krát');
end;
```

V programe postupne hľadáme všetky výskyt hľadanej slabiky. Vždy, keď nájdeme (vtedy platí **Index <> 0**), tak skrátime pôvodnú vetu tak, aby sme z nej odstrihli

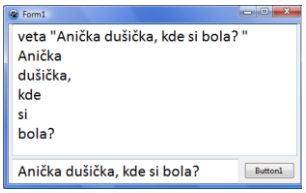
všetko pred danou slabikou a aj prvé písmeno nájdenej slabiky. Tak sa nám pôvodná veta 'mama ma emu a ema ma mamu' bude postupne po každom prechode while-cyklom skracovať:

```
'ama ma emu a ema ma mamu'
'a ma emu a ema ma mamu'
'a emu a ema ma mamu'
'a ma mamu'
'a mamu'
'amu'
```

Po poslednom skrátení, sa v tejto vete slabika 'ma' už nevyskytuje a preto bude **Index=0** a teda while-cyklus skončí.

Ďalší program ukáže, ako môžeme rozdeliť vetu na slová:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Veta, Slovo: String;
  Medzera: Integer;
begin
  Memo1.Clear;
  Veta := Edit1.Text + ' ';
  Memo1.Lines.Append('veta "' + Veta + '"');

  while Veta <> '' do
  begin
    Medzera := Pos(' ', Veta);
    if Medzera <> 0 then
    begin
      Slovo := Copy(Veta, 1, Medzera-1);
      Veta := Copy(Veta, Medzera+1, MaxInt);
      if Slovo <> '' then
        Memo1.Lines.Append(Slovo);
    end;
  end;
end;
```



Tu si všimnite, že už pri priradení textu z **Edit1** sme ku premennej **Veta** pridali jednu medzeru navyše. Táto medzera nebude mať žiaden efekt na výsledok, ale zjednoduší nám to program. Slová vo vete hľadáme tak, že hľadáme medzery, ktoré ich oddeľujú. Aby nebol problém ani s posledným slovom vo vete, aj to bude ukončené medzerou. Program spustíme a zadáme nejakú testovaciu vetu:

Ďalšou dvojicou štandardných reťazcových podprogramov sú procedúry **Delete** a **Insert**:

- Procedúra **Delete** má rovnaké parametre ako funkcia **Copy**: reťazec, index nejakého znaku a počet znakov - procedúra z daného reťazca vyhodí podreťazec od príslušného indexu danej dĺžky.
- Procedúra **Insert** slúži na vkladanie nejakého podreťazca do reťazcovej premennej a to od nejakého konkrétneho indexu - teda má tri parametre: podreťazec, premennú a index.

Tieto dve procedúry ukážeme na príkladoch:

```
s := 'abrakadabra'; Delete(s, 4, 6);      // s = 'abrra'
s := 'abrakadabra'; Delete(s, 6, 20);     // s = 'abrak'
s := 'abrakadabra'; Delete(s, 1, 1);      // s = 'brakadabra'
s := 'abrakadabra'; Insert('*', s, 2);    // s = 'a*brakadabra'
s := 'abrabra'; Insert('kada', s, 5);     // s = 'abrakadabra'
s := 'abrabra'; Insert('kada', s, 10);    // s = 'abrabrakada'
s := 'abxy'; Insert(s, s, 3);             // s = 'ababxyxy'
```

Chceme napísať program, ktorý v zadanom reťazci za každý znak pridá hviezdičku '*'. Použijeme príkaz **Insert**:

```
Veta := Edit1.Text;
for I := 1 to Length(Veta) do
  Insert('*', Veta, I + 1);
Memo1.Lines.Append(Veta);
```

Hoci táto časť programu vyzerá, že by mohla fungovať správne, žiaľ, vytvára chybný výsledný reťazec. Odtrasujte to. Uvedieme dve rôzne správne riešenia. Otestujte ich.

```
Veta := Edit1.Text;
for I := 1 to Length(Veta) do
  Insert('*', Veta, 2 * I);
Memo1.Lines.Append(Veta);
```

Alebo inak

```
Veta := Edit1.Text;
for I := Length(Veta)+1 downto 2 do
  Insert('*', Veta, I);
Memo1.Lines.Append(Veta);
```

Pascal poskytuje ešte veľké množstvo ďalších veľmi užitočných funkcií. Uvedieme aspoň niekoľko z nich.

- **UpperCase**(text) // vráti reťazec, v ktorom budú všetky písmená veľké
- **LowerCase**(text) // vráti reťazec, v ktorom budú všetky písmená malé
- **Trim**(text) // vráti reťazec bez medzier na začiatku a na konci
- **IntToHex**(číslo, počet_cifier) // vráti číslo v 16-ovej sústave
- **StrToIntDef**(text, číslo) // ako **StrToInt** – pri chybe vráti druhý parameter

Úlohy na precvičenie

Zadanie 1	Program z prečítaného reťazca vyhodí všetky zbytočné medzery – nechá len po jednej medzi slovami.
Zadanie 2	Program zo zadanej vety vyhodí všetky samohlásky.
Zadanie 3	Program o zadanom texte zistí, či je to ***palindrom***. Ak sa text rovnako číta zľava ako sprava (ak si nevšímame medzery), tak sa volá palindrom. Napríklad, veľmi známym palindromom je 'jelenovi pivo nelej'. Ak zadaný text vyhovuje podmienke palindromu, tak program vypíše správu 'je to palindrom'.
Zadanie 4	Program v zadanej vete každý výskyt medzery nahradí dvoma medzerami.

2. tematická jednotka - Algoritmy s myšou

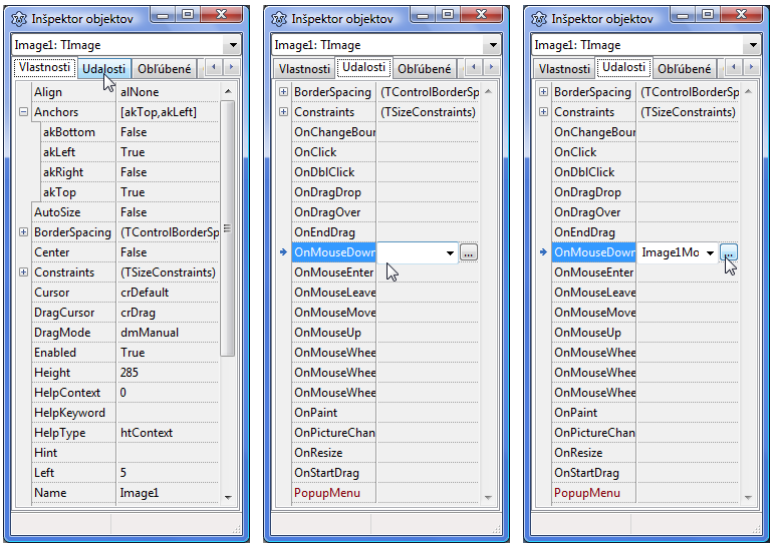
1. Klikanie

V tejto kapitole sa naučíme, ako môžeme naprogramovať prácu s myšou. Keď je pod systémom Windows spustená ľubovoľná aplikácia, systém môže tejto aplikácii oznamovať každý pohyb myši nad jej oknom a tiež aj rôzne typy klikania a ťahania myši. Aplikácie, ktoré sú napísané v Lazarus alebo Delphi, dostávajú tieto informácie formou udalostí. Ak pre nejakú z týchto udalostí napíšeme zodpovedajúcu procedúru na jej obsluhu, táto sa zavolá pri každom výskyte tejto udalosti. Zatiaľ sme sa zoznámili s týmito udalosťami:

- Button1Click – kliknutie myšou na tlačidlo
- Timer1Timer – tikanie časovača
- ScrollBar1Change – ťahalo sa s posúvačom

Aby sme mohli pridávať nové udalosti, musíme prejsť do Inšpektora objektov, pričom musíme mať nastavený príslušný komponent, pre ktorý chceme definovať nejakú udalosť, teda **Image1**. V Inšpektore objektov prejdeme do záložky **Udalosti** (Events) a nájdeme riadok s požadovanou udalosťou. Teraz, keď dvojklikneme do voľného políčka pri tejto udalosti, v editovacom okne sa predpripraví prázdna procedúra, napr. **Image1MouseDown**.

Začneme s udalosťou **OnMouseDown**, ktorá označuje, že myšou sme klikli niekde do grafickej plochy. Takže v **Inšpektore objektov** v záložke **Udalosti** nájdeme riadok **OnMouseDown** a dvojklikneme sem (v Lazarus môžeme kliknúť do tlačidielka [...]).

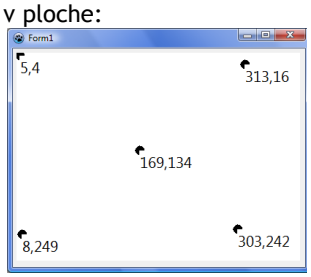


V editovacom okne sa pripravila prázdna procedúra:

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin

end;
```

Otestujeme to programom, v ktorom kliknutie do grafickej plochy na tomto mieste nakreslí malý kruh a vypíše súradnice kliknutého bodu:

Po spustení môžeme klikaním prekontrolovať súradnice niekoľkých bodov v ploche:

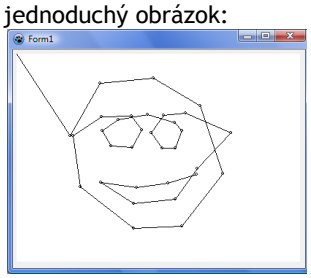



```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 6;
  Image1.Canvas.Font.Height := 30;
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.Ellipse(X-4, Y-4, X+4, Y+4);
  Image1.Canvas.TextOut(X, Y, IntToStr(X) + ',' + IntToStr(Y));
end;
```

Všimnite si, že procedúra **Image1MouseDown** má viac rôznych parametrov. Pre nás sú najdôležitejšie dva parametre **X** a **Y**, v ktorých dostávame súradnice kliknutého miesta v grafickej ploche.

Nasledujúci program ukáže, že kliknuté body môžeme veľmi jednoducho spájať čiarami (**LineTo**). V každom kliknutom bode nakreslíme aj malú kružnicu, aby bolo lepšie vidieť kliknuté miesto:

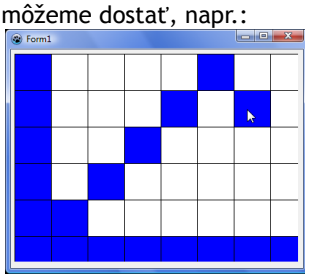
Po spustení môžeme klikaním vytvoriť aj jednoduchý obrázok:




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

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.LineTo(X, Y);
  Image1.Canvas.Ellipse(X-2, Y-2, X+2, Y+2);
end;
```

Ďalší program ukazuje spôsob, ako pracujeme s klikaním do štvorčekov štvorcovej siete. Procedúra **FormCreate** nakreslí prázdnu štvorcovú sieť. Potom každé kliknutie do plochy zistí, do ktorého štvorčeka sme klikli a ten potom vyfarbí modrou farbou. V programe sme definovali pomocnú globálnu procedúru **Stvorcek**, ktorá nakreslí štvorček v **R**-tom riadku a **S**-tom stĺpci štvorcovej siete. Globálna konštanta **D** má hodnotu veľkosti štvorčeka siete.

Po niekoľkých kliknutiach môžeme dostať, napr.:



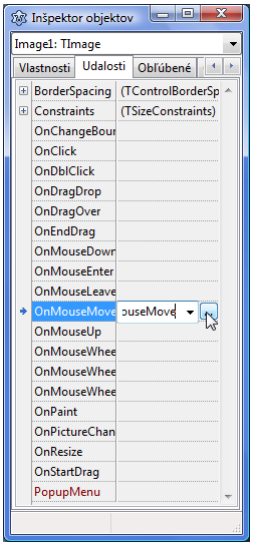

```
const
  D = 50;

procedure Stvorcek(R, S: Integer);
begin
  Form1.Image1.Canvas.Rectangle(S*D, R*D, S*D+D+1, R*D+D+1);
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  I, J: Integer;
begin
  for I := 0 to Image1.Height div D do
    for J := 0 to Image1.Width div D do
      Stvorcek(I, J);
  Image1.Canvas.Brush.Color := clBlue;
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Stvorcek(Y div D, X div D);
end;
```

Pri kliknutí do plochy vypočítame riadok (**Y div D**) a stĺpec (**X div D**) štvorčeka siete a ten potom nakreslíme s modrou výplňou.

Úlohy na precvičenie

Zadanie 1	Kliknutie do plochy na danom mieste nakreslí strom - hnedý kmeň a zelená koruna stromu (kružnica).
Zadanie 2	Prvé kliknutie nakreslí bodku, druhé kliknutie obdĺžnik medzi prvý a druhým kliknutým bodom. Ďalej sa robí to isté pre 3. a 4. kliknutie, 5. a 6., atď.
Zadanie 3	Do programu, v ktorom sme klikali do štvorcovej siete a kliknuté políčka sme zafarbovali na modro, doplňte tlačidlá, ktoré budú meniť farbu štetca, napr. modrá, červená, zelená, biela.

2. Pohyb myši



Ďalšou veľmi dôležitou udalosťou pri práci s myšou v grafickej ploche je pohyb myši. Táto udalosť vzniká vždy vtedy, keď sa kurzor myši pohne nad grafickou plochou bez ohľadu na to, či je niektoré tlačidlo myši zatlačené alebo nie.

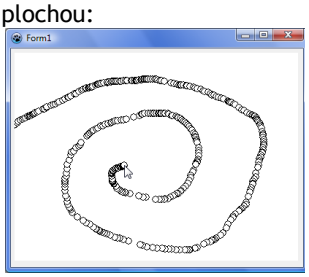
V **Inšpektore objektov** v záložke **Udalosti** nájdeme riadok **OnMouseMove** a dvojklikneme sem (v Lazarus môžeme kliknúť do tlačidielka [...])

Procedúra Image1MouseMove je podobná Image1MouseDown a tiež má dva parametre **X** a **Y**, ktoré sú momentálne súradnice myši nad plochou:

```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin

end;
```

Hýbanie myšou predvedieme na tomto malom projekte:

A po spustení stačí prejsť myšou nad grafickou plochou:

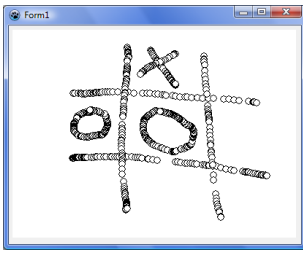



```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  Image1.Canvas.Ellipse(X-5, Y-5, X+5, Y+5);
end;
```

Procedúra **Image1MouseMove** má ešte jeden užitočný parameter: **Shift**. V ňom nám systém odovzdáva informáciu o stlačených tlačidlách myši. Práca s touto informáciou je trochu komplikovanejšia, nakoľko využíva dátovú štruktúru množina, s ktorou sme sa ešte neučili pracovať. Tento parameter môže nadobúdať rôzne hodnoty, ale pre nás sú asi najzaujímavejšie tieto:

- **Shift = []** // nezatlačili sme žiadne tlačidlo
- **Shift = [ssLeft]** // zatlačili sme len ľavé tlačidlo
- **Shift = [ssRight]** // zatlačili sme len pravé tlačidlo
- **Shift = [ssLeft, ssRight]** // zatlačili sme naraz obe tlačidlá

Ak chceme opraviť náš program, aby kreslil iba pri zatlačenom ľavom tlačidle, zapíšeme podmienku takto:



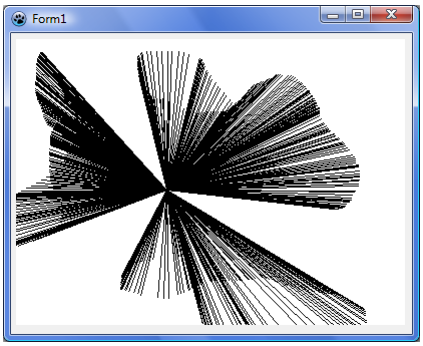

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if Shift = [ssLeft] then
    Image1.Canvas.Ellipse(X-5, Y-5, X+5, Y+5);
end;
```

Po spustení vidíme, že krúžky sa kreslia len pri zatlačenom ľavom tlačidle myši.

Ďalší program spája body na pozícii myši s nejakým dopredu zvoleným bodom:

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

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if Shift = [ssLeft] then
  begin
    Image1.Canvas.MoveTo(150, 150);
    Image1.Canvas.LineTo(X, Y);
  end;
end;
```

Ak teraz po spustení programu niekde do plochy klikneme a ťaháme, tieto body myši sa spájajú s bodom (150, 150). Vznikajú pri tom takéto efektné obrazce:



V mnohých aplikáciách, kde je dôležité poznať súradnice polohy myši, sa do **Image1MouseMove** pridáva výpis súradníc X a Y. Môžeme niekde do formulára vložiť komponent **jednoduchý text** (**TLabel**) a sem sa bude potom pri pohybe myši vypisovať X a Y, napr. takto:

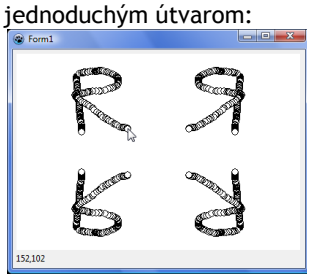
```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  Label1.Caption := IntToStr(X) + ',' + IntToStr(Y);

  ...
end;
```

Súradnice myši sa teraz budú vypisovať vždy, bez ohľadu na to, čo robí program.

Pri kreslení útvarov môžeme využiť nielen súradnice polohy myši, ale veľmi ľahko vieme vypočítať aj symetrické body: pre bod (X, Y) sú to aj (Sirka-X, Y), (X, Vyska-Y) a (Sirka-X, Vyska-Y). Vo **FormCreate** si do dvoch globálnych premenných **Sirka** a **Vyska** priradíme šírku a výšku grafickej plochy. Program pri zatlačenom tlačidle kreslí krúžky nielen na pozícii myši ale aj na ďalších troch symetrických miestach:

```
var
  Sirka, Vyska: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Sirka := Image1.Width;
  Vyska := Image1.Height;
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  Label1.Caption := IntToStr(X) + ',' + IntToStr(Y);

  if Shift = [ssLeft] then
  begin
    Image1.Canvas.Ellipse(X-5, Y-5, X+5, Y+5);
    Image1.Canvas.Ellipse(Sirka-X-5, Y-5, Sirka-X+5, Y+5);
    Image1.Canvas.Ellipse(X-5, Vyska-Y-5, X+5, Vyska-Y+5);
    Image1.Canvas.Ellipse(Sirka-X-5, Vyska-Y-5, Sirka-X+5, Vyska-Y+5);
  end;
  if Shift = [ssRight] then
    Image1.Canvas.FillRect(Image1.ClientRect);
end;
```

Symetrické kreslenie otestujeme nejakým jednoduchým útvarom:



Všimnite si, že pri ťahaní myši so zatlačeným pravým tlačidlom myši, sa zmaže obrazovka.

Úlohy na precvičenie

Zadanie 1	Pri zatlačenom tlačidle myši sa kreslia úsečky náhodnej farby: úsečka je z bodu (X, Y) do bodu (X, Y-30).
Zadanie 2	Pri hýbaní myšou so zatlačeným ľavým tlačidlom sa kreslia malé farebné krúžky, pri zatlačenom pravom tlačidle sa kreslia malé farebné štvorčeky.
Zadanie 3	Efekt kreslenia sprejom: nakreslí sa 50 náhodných bodiek v okolí pozície myši (okolie znamená že X-ová aj Y-ová súradnica nie je vo väčšej vzdialenosti ako napr. 15).
Zadanie 4	Symetrické kreslenie malými krúžkami doplňte tak, aby každá symetrická časť kreslila krúžky inej farby.

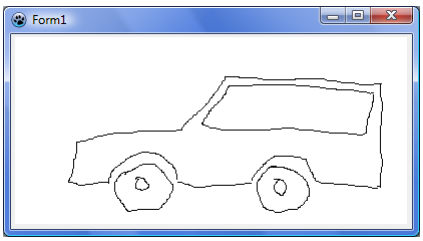
3. Ťahanie

Teraz si ukážeme, aké sú možnosti, keď využijeme obe udalosti myši, s ktorými sme sa už zoznámili. Môžeme totiž rozlíšiť kliknutie a ťahanie, pričom pri kliknutí (udalosť **onMouseDown**) sa niečo nastaví a pri ťahaní (udalosť **onMouseMove**) sa to využije.

Prvý príklad ukáže kreslenie krivky: kliknutie sem prenesie pozíciu pera (MoveTo) a ťahanie už kreslí čiary (LineTo):

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

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

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if Shift = [ssLeft] then
    Image1.Canvas.LineTo(X, Y);
end;
```

Toto je najjednoduchšie kreslenie kriviek, napr. môžeme nakresliť



Pri každom kliknutí môžeme nastaviť aj hrúbku alebo farbu kreslených čiar, napr. takto sa môžu nastaviť náhodné farby a hrúbky:

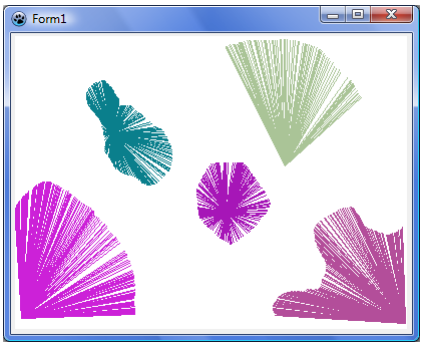
```
procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.MoveTo(X, Y);
  Image1.Canvas.Pen.Color := Random(256*256*256);
  Image1.Canvas.Pen.Width := Random(20) + 1;
end;
```

Prekontrolujeme to, napr. takto:

Pri kliknutí (**onMouseDown**) si môžeme v globálnych premenných **X1**, **Y1** zapamätať súradnice tohto kliknutého miesta a pri ťahaní myši (**onMouseMove**) môžeme túto súradnicu využiť. Napríklad nakreslíme rôznofarebné vejáre:

```
var
  X1, Y1: Integer;

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

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  X1 := X;
  Y1 := Y;
  Image1.Canvas.Pen.Color := Random(256*256*256);
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if Shift = [ssLeft] then
  begin
    Image1.Canvas.MoveTo(X1, Y1);
    Image1.Canvas.LineTo(X, Y);
  end;
end;
```

A otestujeme:



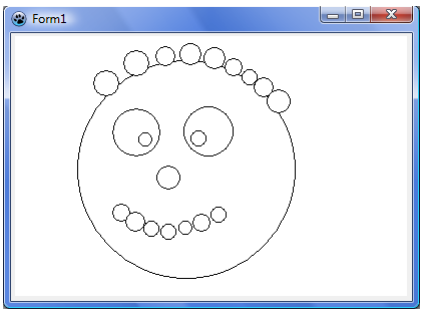
V ďalšom programe budeme kresliť kruhy tak, že kliknutie určí stred kruhu a ťahanie určuje bod na kružnici. Aby sme mohli takúto kružnicu nakresliť, musíme vypočítať jej polomer: to je vzdialenosť bodu na kružnici od jej stredu:

```
var
  X1, Y1: Integer;

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

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  X1 := X;
  Y1 := Y;
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
var
  R: Integer;
begin
  if Shift = [ssLeft] then
  begin
    R := Round(Sqrt((X-X1)*(X-X1) + (Y-Y1)*(Y-Y1)));
    Image1.Canvas.Ellipse(X1-R, Y1-R, X1+R, Y1+R);
  end;
end;
```

Je dobre si pamätať vzorec na výpočet vzdialenosti dvoch bodov, lebo ho budeme často potrebovať. Keďže Pascal má aj funkciu **Sqr**, ktorá vypočíta druhú mocninu parametra, vzdialenosť môžeme zapísať aj takto:

R := Round(Sqrt(Sqr(X-X1) + Sqr(Y-Y1)));

Otestujeme program, napr.

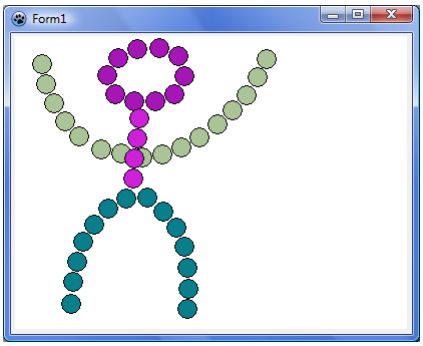


Zistite, čo sa bude diať, keď počas ťahania vyrobím veľkú kružnicu a potom ťahám smerom ku jej stredu.

Nasledujúci príklad ilustruje iné využitie výpočtu vzdialenosti dvoch bodov. Počas ťahania chceme kresliť náhrdelníky zložené z kruhov s polomerom napr. 20. Ak by sme pri ťahaní (**onMouseMove**) kreslili takéto kruhy, mohli by byť príliš nahusto a prekrývali by sa. Preto pre každý bod zistíme, či už je dostatočne vzdialený od predchádzajúceho (vzdialenosť aspoň 40) a ak áno, tak ho nakreslíme:

```
var
  X1, Y1: Integer;

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

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  X1 := X;
  Y1 := Y;
  Image1.Canvas.Brush.Color := Random(256*256*256);
  Image1.Canvas.Ellipse(X-10, Y-10, X+10, Y+10);
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if (Shift = [ssLeft]) and (Sqrt(Sqr(X-X1) + Sqr(Y-Y1)) >= 20) then
  begin
    Image1.Canvas.Ellipse(X-10, Y-10, X+10, Y+10);
    X1 := X;
    Y1 := Y;
  end;
end;
```

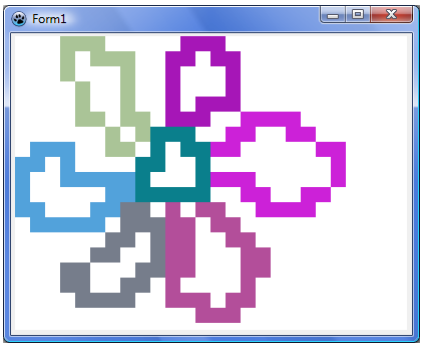
Všimnite si, kde sme vložili test, ktorý zistí vzdialenosť bodu myši od posledného nakresleného kruhu. Tiež po nakreslení kruhu nesmieme zabudnúť si zapamätať tieto súradnice ako nové (X1, Y1). Otestujeme:



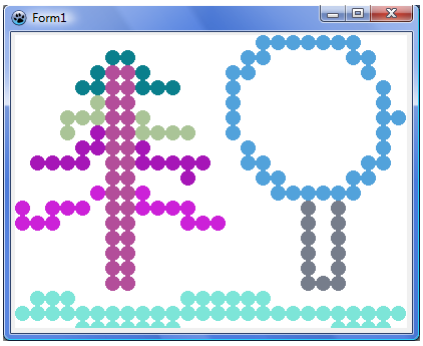
V predchádzajúcej časti sme klikali do políčok štvorcovej siete a zafarbovali sme ich. Podobnú ideu použijeme aj v nasledujúcom programe. Štvorčeky sa budú zafarbovať pri ťahaní myšou a kliknutie nastaví farbu:

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

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.Brush.Color := Random(256*256*256);
  Image1.Canvas.Pen.Color := Image1.Canvas.Brush.Color;
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
const
  D = 15;

  procedure Stvorcek(R, S: Integer);  // riadok a stĺpec siete
  begin
    Image1.Canvas.Rectangle(D*S, D*R, D*S+D, D*R+D);
  end;

begin
  if Shift = [ssLeft] then
    Stvorcek(Y div D, X div D);
end;
```

Konštanta **D** označuje veľkosť štvorčeka siete (môžete s ňou poexperimentovať). Po spustení programu otestujeme kreslenie v štvorcovej sieti:



Veľmi zaujímavý efekt môžeme dosiahnuť jednoduchou úpravou: namiesto štvorčekov (príkaz **Rectangle**) budeme kresliť kruhy (príkaz **Ellipse**). Obrázky potom vyzerajú takto:



Ešte existuje jedna užitočná udalosť v súvislosti s myšou v grafickej ploche – to je udalosť pustenie tlačidla myši **onMouseUp**. Bežne tieto tri udalosti nasledujú v takomto poradí: najprv príde jedna udalosť **onMouseDown**, potom za sebou príde

niekoľko (niekedy aj niekoľko tisíc) udalostí **onMouseMove** a na záver opäť jedna udalosť **onMouseUp**. Túto tretiu udalosť uvidíme v niektorých ďalších častiach.

Úlohy na precvičenie

Zadanie 1	Naprogramujte jednoduchý grafický editor: farba pera sa nastavuje kliknutím na jedno z tlačidiel (napr. červená, modrá, zelená, biele, ...), hrúbka pera sa môže meniť posúvačom (nastavte mu Min=1 a Max=20).
Zadanie 2	Napíšte program, ktorý kreslí domčeky rôznej veľkosti: kliknutie definuje prvý spodný vrchol domčeka (štvorec, na ktorom je trojuholník), ťahaním určujeme druhý spodný vrchol (jeho X-ovú súradnicu, Y-ová je rovnaká, ako má prvý vrchol). Pri kreslení domčeka sa najprv zmaže grafická plocha.
Zadanie 3	Prvé kliknutie určí stred (X1, Y1) kreslenej elipsy, ťahaním určujeme jeden z vrcholov opísaného obdĺžnika tejto elipsy – pritom sa táto elipsa stále kreslí. Uvedomte si, že keď poznáme jeden vrchol opísaného obdĺžnika (X, Y) a stred (X1, Y1), tak druhý vrchol vieme vypočítať ako (2*X1-X, 2*Y1-Y).
Zadanie 4	Ťahaním myši sa bude kresliť náhrdelník z veľkých písmen abecedy (nastavte veľkosť fontu na 20). Pri kliknutí sa nastaví náhodná farba fontu a vypíše sa písmeno 'A'. Pri nasledovnom ťahaní sa pripisujú ďalšie písmená abecedy (aby neboli písmená príliš nahusto, ďalšie písmeno vypíšte až v nejakej minimálnej vzdialenosti od predchádzajúceho).

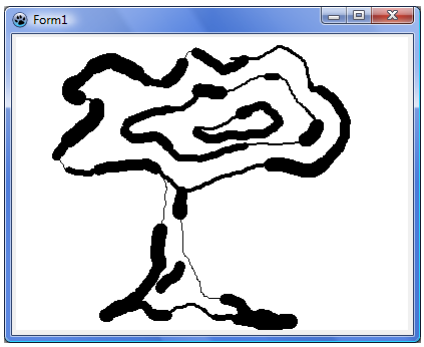
4. Časovač a myš

Ukážeme, že časovač môžeme veľmi pekne využiť aj v kombinácii s klikaním a ťahaním myši v grafickej ploche.

Do obyčajného kreslenia kriviek pridáme v časovači náhodnú zmenu hrúbky pera:

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

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

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

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if Random(5) = 0 then
    Image1.Canvas.Pen.Width := Random(15) + 1;
end;
```

Časovaču nastavíme **Interval = 50** a vďaka náhodnému generátoru kreslíme takéto zaujímavé kresby:



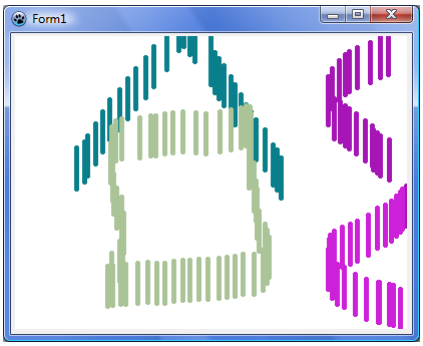
Pozrite, ako sa zmení správanie v tejto verzii časovača:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if Random(5) = 0 then
    if (Random(2) = 0) and (Image1.Canvas.Pen.Width > 3) then
      Image1.Canvas.Pen.Width := Image1.Canvas.Pen.Width - 3
    else if Image1.Canvas.Pen.Width < 15 then
      Image1.Canvas.Pen.Width := Image1.Canvas.Pen.Width + 3;
end;
```

V ďalšom príklade ukážeme, ako môžeme naštartovať nejakú krátku sekvenciu, napr. pomalé predlžovanie nejakej úsečky. V programe budeme sadiť farebné paličky. Začnú rásť s dĺžkou 5, postupne sa o 5 budú predlžujú a rast skončíia pri dĺžke 40. Časovaču nastavíme **Interval = 30**. Kliknutie do grafickej plochy (**onMouseDown**) nastaví farbu pera. Pri ťahaní (**onMouseMove**) skontroluje, či už niečo nerastie, a ak momentálne nie, naštartuje nový rast: zapamätá si pozíciu myši a počiatočnú dĺžku paličky nastaví na **D=5**. V časovači, ak má niečo rásť, tak vykreslí paličku, predĺži ju a ak je už dostatočne dlhá, tak **D** vynuluje a tým tento rast zastaví:

```
var
  X1, Y1, D: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 5;
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Image1.Canvas.Pen.Color := Random(256*256*256);
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if (Shift = [ssLeft]) and (D = 0) then    // ak ešte nič nerastie
  begin
    X1 := X;
    Y1 := Y;
    D := 5;
  end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if D > 0 then       // ak má niečo rásť
  begin
    Image1.Canvas.MoveTo(X1, Y1);
    Image1.Canvas.LineTo(X1, Y1-D);
    D := D + 5;
    if D > 40 then    // ak je už dostatočne dlhá
      D := 0;         // ukonči rast
  end;
end;
```

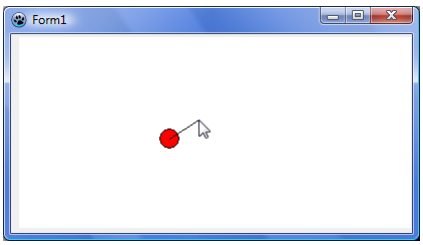
Po spustení programu treba myš ťahať veľmi pomaly, aby paličky stihli narásť. Môžeme dostať napr.:



V ďalšom programe predvedieme, ako môžeme naštartovať smer a rýchlosť lopty, ktorá sa odráža od okrajov grafickej plochy. Polohu lopty budeme uchovávať v globálnych premenných X1, Y1, posunutie lopty (vektor posunutia) v premenných DX, DY. Keď klikneme do plochy, budeme definovať nový vektor, preto starý vynulujeme. Pri ťahaní myšou zobrazujeme momentálne nastavovaný smer a rýchlosť (teda vektor) úsečkou od stredu lopty až po pozíciu myši. Pri pustení myši (udalosť **onMouseUp**) sa do DX a DY priradia nové hodnoty vektora a tým sa naštartuje pohyb lopty. Loptu pohybuje časovač (**Interval = 100**), pričom, ak by mala z plochy vypadnúť, tak príslušne otočí vektor posunu:

```
var
  X1, Y1, DX, DY: Integer;

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

procedure Kresli;
begin
  Form1.Image1.Canvas.Brush.Color := clWhite;
  Form1.Image1.Canvas.FillRect(Form1.Image1.ClientRect);
  Form1.Image1.Canvas.Brush.Color := clRed;
  Form1.Image1.Canvas.Ellipse(X1-10, Y1-10, X1+10, Y1+10);
end;

procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  X1 := X;
  Y1 := Y;
  DX := 0;
  DY := 0;
  Kresli;
end;

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
begin
  if Shift = [ssLeft] then
  begin
    Kresli;
    Image1.Canvas.MoveTo(X1, Y1);
    Image1.Canvas.LineTo(X, Y);
  end;
end;

procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  DX := X - X1;
  DY := Y - Y1;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  X1 := X1 + DX;
  if (X1+DX <= 0) or (X1+DX >= Image1.Width) then
    DX := -DX;
  Y1 := Y1 + DY;
  if (Y1+DY <= 0) or (Y1+DY >= Image1.Height) then
    DY := -DY;
  Kresli;
end;;
```

Vektor posunutia nastavujeme asi takto:



Úlohy na precvičenie

Zadanie 1

Naprogramujte takúto hru:

- na náhodnej pozícii sa nakreslí bodka, ak do 2 sekúnd na ňu používateľ klikne, získa bod (body sa niekde vypisujú) a generuje sa nová bodka;
- ak používateľ do 2 sekúnd nestihne na ňu kliknúť, generuje sa nová pozícia a stráca sa bod.

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- ďalší základný ordinálny typ Char a jeho vlastnosti
- zložený typ znakový reťazec a jeho štandardné funkcie
- udalosti myši v grafickej ploche: kliknutie, ťahanie a pustenie tlačidla myši

Okrem toho boli uvedené tieto dôležité koncepty:

- súčasné spracovanie udalosti časovača a udalostí myši

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Počas ťahania myši nad grafickou plochou (so zatlačeným tlačidlom) a kreslením krivky sa každú celú sekundu vypíše na pozíciu myši ďalšie a ďalšie písmeno abecedy.

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 6

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti RNDr. František Galčík, PhD.
Doc. RNDr. Gabriela Andrejková, CSc.

Počet strán 36

Náklad 300 ks

Prvé vydanie, Bratislava 2009

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

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-89225-99-6