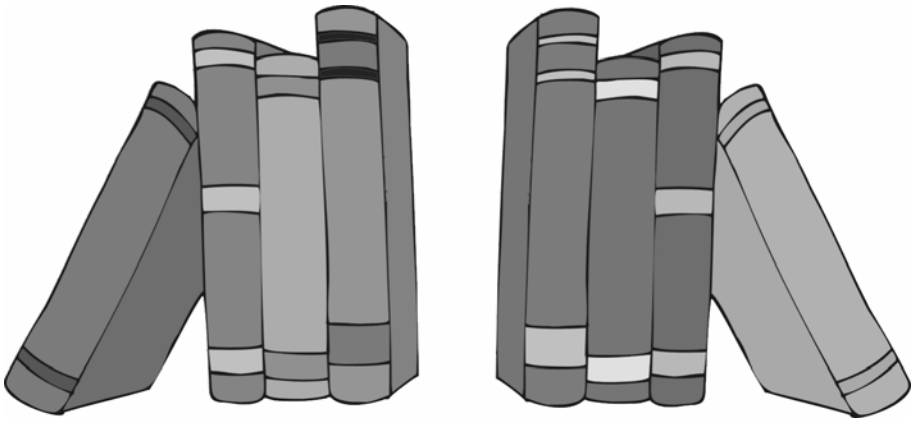
METODICKO-PEDAGOGICKÉ CENTRUM V PREŠOVE

Jana Machová

Začíname s DELPHI



- 2004 -

METODICKO-PEDAGOGICKÉ CENTRUM V PREŠOVE

Jana Machová

Začíname s DELPHI

- 2004 -

OBSAH

1 ZAČÍNAME S DELPHI

1.1 Prečo ideme programovať v Delphi?

- Delphi je vizuálne programové prostredie, ktoré slúži na vývoj aplikácií pre OS Windows (podobne ako Visual Basic, C++Builder a pod.) - takéto prostredia zjednodušujú programovanie Windows aplikácií - vzhľad aplikácie pod Windows môžeme zostavovať z ponuky komponentov, pričom samotné vývojové prostredie pomáha pri začleňovaní zodpovedajúceho kódu.
- Takéto moderné programovacie jazyky umožňujú začať programovať objektovým prístupom - je to metodológia, bez ktorej sa v súčasnosti nezaobídeme pri práci v tíme, pri práci s veľkými projektmi, pri tvorbe Windows aplikácií.
- V Delphi sa programuje moderným programovacím jazykom objektovým pascalom - tento vychádza z Turbo Pascalu, ale pritom si treba uvedomiť, že Pascal sa v informatickej komunite považuje za „esperanto“ programovacích jazykov.
- Pascal je ideálny jazyk ako úvodný pri vyučovaní programovania - môžeme sa v ňom naučiť veľmi efektívne programovať modernými metódami, kto ovláda Pascal, ľahko neskôr prejde na iné procedurálne programovacie jazyky.

1.2 Typy súborov v Delphi

Delphi programy obsahujú 3 hlavné súbory s príponami **.dpr .pas .dfm**

- **Project1.dpr** - delphi projekt (textový súbor = hlavný „pascalovský“ program) - zatiaľ je pre nás nezaujímavý.
- **Project1.res** - obsahuje len ikonu výsledného projektu.
- **Project1.dof**; - nastavenia kompilátora (textový súbor = možno si ho pozrieť v nejakom editore).
- **Project1.dsk** - rozloženie okien a súborov počas práce Delphi (textový súbor) - ak celý projekt niekam prenášate, tento súbor radšej zrušte.
- **Project1.exe** - spustiteľný program.
- **Unit1.pas** - programový popis formuláru, čo a ako má formulár robiť = podobá sa na klasický pascalovský program.
- **Unit1.dfm** - vnútorná definícia delphi formuláru = aké a kde sú komponenty vo formulári (napr. Image1 a Button1).
- **Unit1.dcu** - preložený Unit1.pas (v starom pascale.tpu).

Poznámky:

- každý Form (.**dfm**) **musí** mať **Unit** (.pas),
- môže byť **Units** bez **Form** (len kód),
- súbor **.dfm** má špeciálny pseudokódovaný tvar. Ak ho chcete vidieť, použite skratku **Alt F12**,
- vo všeobecnosti platí: **NEEDITUJTE súbor .dfm** (ako text), iba vkladajte objekty do formuláru a Delphi vytvára súbor **.dfm** za vás. Ak preeditujete súbor **.dfm** nesprávne, môžete si vytvoriť zbytočné problémy.

1.3 Vytváranie a ukladanie projektov

Zapamätajte si: Premenujte všetko hneď na začiatku práce s novým projektom!
Použite samostatný adresár (Zložku) pre každý projekt!
Vždy keď začínate vyvíjať novú aplikáciu - **File – New – Application,** hneď si ju radšej zapíšte v okne do novovytvoreného adresára.
Premenujte si všetky komponenty, keď ich prvýkrát použijete napr.

Button1 - StartBtn, alebo BStart, alebo jednoducho Tlacidlo1
ListBox1 - CustList, alebo LBCust, alebo jednoducho Zoznam1.

1.4 Programátorské zvyky

Prvé písmená z mien premenných, procedúr a iných názvov píšte veľkými písmenami, napr. PocetSlov : integer;
Pri programovaní zarovnávajte *begin* a *end*, časť medzi týmito príkazmi odsadzujte o 2 medzery:

```
for x:=1 to 22 do
  begin
    if y>32 then ....
  end;

if ... then
  begin
       ...
  end
          else if ... then begin ... end;
```

1.5 Dôležité klávesy

F1 - Help,
F12 - Toggle Form / Unit,

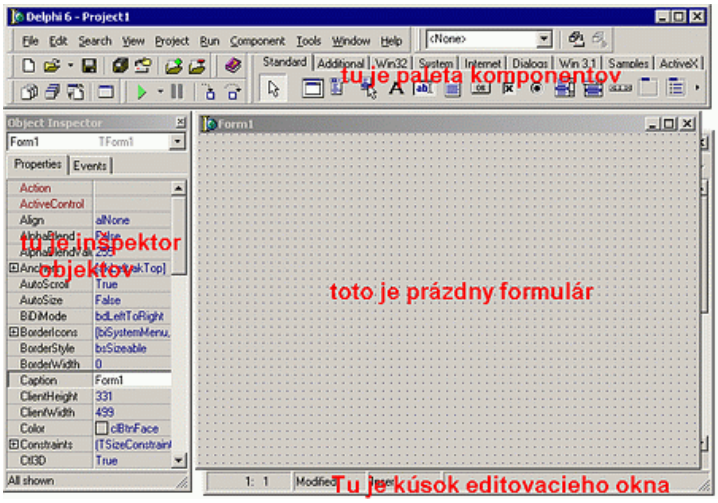
Alt F12 - Toggle Form/.dfm,
F9 - spustiť program,
Shift + F2 - reset (zastaviť) programu - aj taký, ktorý sa nám zacyklil, neskončil ...
Debugovanie programu - trasovanie pri hľadaní chýb (ako v TP),
F7 - jednoduché krokovanie (kroky aj do vnútra procedúr),
F8 - krokovanie (kroky vynechaním tela procedúry).

1.6 Prostredie Delphi

Delphi si pri štarte pamätá, čo sa robilo naposledy a preto, ak je pri štarte otvorený nejaký program, tak zvolíme v menu **File - New - Application** a otvoríme novú aplikáciu. (Ak v menu zvolíme **File - New - Other - Console Application**, môžeme v prostredí Delphi vytvárať a spúšťať programy v Turbo Pascale).

Pozrime sa na jednotlivé časti prostredia a zapamätajme si ich názvy, ďalej ich budeme používať.

Ak nevidíte „**Objektový inšpektor**“ stlačte **F11.**
Dve okná: **editovacie okno** unitu a **okno formuláru** sa často prekrývajú (kláves **F12** ich navzájom vymieňa).

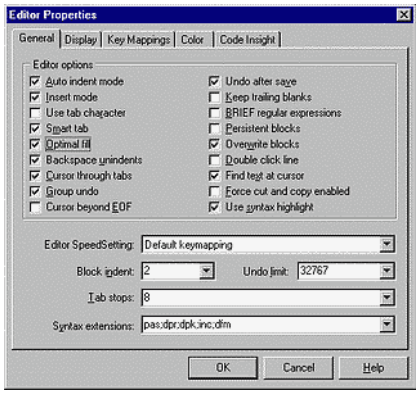


Delphi nám ponúka „novú aplikáciu“, ale v skutočnosti je to už predpripravený program, ktorý môžeme spustiť. Ak ju spustite (napr. klávesom **F9**, alebo zeleným trojuholníkom v ľavej časti hlavného okna), objaví sa len prázdne okno - tento program zatiaľ nič iné nevie - môžeme ho len zastaviť (napr. Alt-F4).

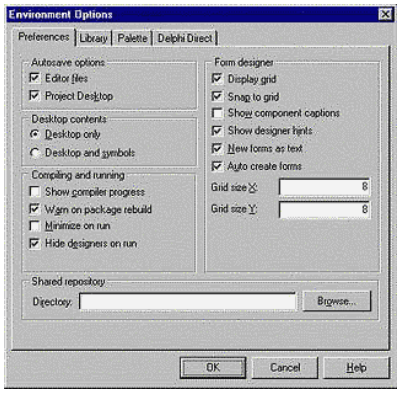
Pred samotným spustením musíme program uložiť na disk, najlepšie z menu **File - Save Project As ...** Nezabudnite, že je dobre každý projekt uložiť na disk do nového podadresára (napr. **D:\Projekty\Prog1**).

1.7 Nastavte si prostredie Delphi

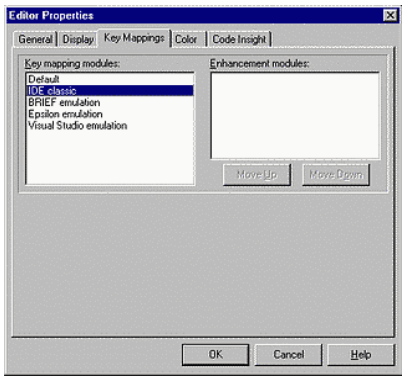
Správne nastavenie prostredia je nevyhnutné pre ďalšiu prácu, samotné Delphi poskytujú kontrolovanie správnosti nášho programu, takže to čo najviac využime.



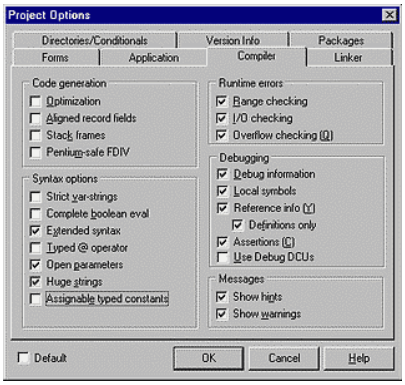
Tools –> Environment Options... –> Preferences



Tools –> Editor Options... –> General



Tools –> Editor Properties –> Key Mappings



Project –> Options... –> Compiler

2 PRVÝ PROGRAM

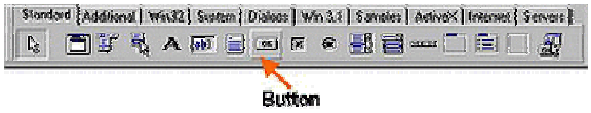
2.1 Úloha

Nakreslite obdĺžnik do grafickej plochy.

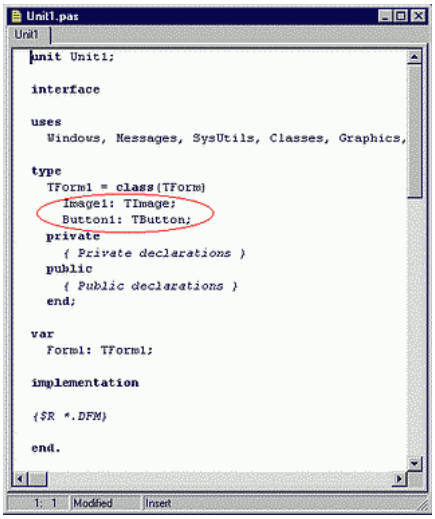
Riešenie:

Do prázdneho formuláru položíme „grafickú plochu“ - komponent **Image** z palety komponentov **Additional**:

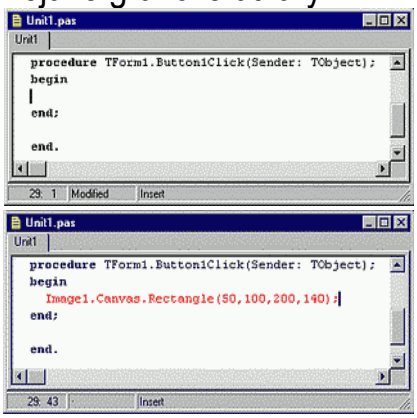
a tlačidlo **Button** z palety **Standard**:



Poznámka:
Ak do formuláru vložíte komponent, ktorý tam mať nechcete, kliknite naň a zrušte ho klávesom Del.



Ako teraz vyzerá editovacie okno?
Do programu pribudli dva nové riadky, ktoré definujú dva nové prvky v našom okne: **Image1** (grafická plocha) a **Button1** (tlačidlo).
Naprogramujeme, ako sa bude správať program, keď klikneme na tlačidlo. Chceli by sme, aby sa do grafickej plochy nakreslili nejaké grafické útvary.



Dvojklikneme vo formulári na tlačidlo **Button1** - v editovacom okne sa pripraví úsek programu, do ktorého môžeme písať postupnosť príkazov. Táto postupnosť sa bude vykonávať počas behu programu vždy po zatlačení tlačidla (kliknutí na tlačidlo):

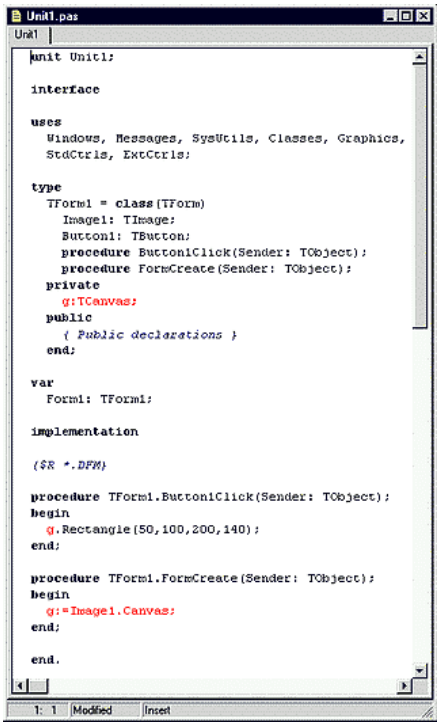
Pridajme jeden riadok:

```
Image1.Canvas.Rectangle(50,100,200,140);
```

Program spustíme klávesom **F9** alebo malou zelenou ikonou v ľavej časti. Ak máte dobre nastavené prostredie, Delphi vám ponúknu dialóg na zapísanie projektu. Projekt si zapíšte do nového adresára. Potom sa program spustí a po kliknutí na tlačidlo Button1 sa v bielej grafickej ploche objaví obdĺžnik.
Klávesmi Alt-F4 program ukončíme ...
Výsledok bude podobný, ako na obrázku:

2.2 Zjednodušenie na záver

Pomenujme si tú časť grafickej plochy, do ktorej sa kreslí (teda jej **Canvas** - plátno) písmenom **g**. Potom namiesto **Image1.Canvas** budeme písať jednoducho **g** a za bodkou meno príkazu, ktorý chceme vykonať (napr. už vieme, že **Rectangle** znamená obdĺžnik).



Riešenie:
Dvojkliknime do formulára mimo **Image1** aj mimo **Button1**, vtedy sa v editovacom okne objaví nové miesto na písanie programu – procedúry **FormCreate** (tu budeme definovať počiatočné inicializácie a nastavenia). Do tela dopíšme
g:=Image1.Canvas;
Vráťme sa o niekoľko riadkov vyššie a dopíšme jeden riadok aj do definície formuláru:
g:TCanvas; .
Nezabudnime ešte prepísať časť programu v procedúre **TForm1.Button1Click**. Celý program vyzerá tak, ako na obrázku. Po spustení program pracuje úplne rovnako, t. j. nakreslí obdĺžnik.

3 TROCHU TEÓRIE

3.1 Čo znamená objektovo - orientovaný jazyk?

V doterajších príkladoch v Pascale, či Delphi sme zadefinovali a používali buď jednoduché premenné (integer, real, char, ...), štruktúrované premenné (array, set, record, ...). V predchádzajúcom príklade sme však použili premenné typu grafická plocha **TImage**, typu tlačidlo **TButton** atď. Tieto premenné sa líšia od „obyčajných“ premenných tým, že:

- pamätajú si svoj momentálny stav v svojich tzv. stavových premenných (napr. pozícia, farba, tvar, ...),
- majú svoje súkromné príkazy, pomocou ktorých ich nejako riadime, resp. meníme ich stavové premenné - takýmto príkazom (sú to procedúry) hovoríme **metódy**,
- musia byť vytvorené (nie deklarované) špeciálnym spôsobom - **konštruktorom** (Create(...);) a kým sa takto nevytvoria, nesmú sa vôbec používať,
- takýmto premenným hovoríme **OBJEKT** - object a typom, z ktorých vytvárame objekty (napr. TCanvas) hovoríme TRIEDA - class, niekedy sa objektu hovorí aj inštancia triedy,
- stretli sme sa už s objektmi ako sú napr. Form1, Image1, Button1 aj premenná g, ktoré sú **inštanciou** triedy TCanvas,
- zatiaľ si o objektoch treba zapamätať, že sú to premenné, ktoré môžu v sebe obsahovať veľa stavových premenných a tiež „v sebe“ obsahujú nejaké svoje procedúry (metódy) => tomuto hovoríme **zapuzdrenie** - **enkapsulácia**, lebo v jednom „puzdre“ sú aj údaje (stavové premenné) aj algoritmy (metódy), ktoré vedia s týmito údajmi pracovať,
- okrem zapuzdrenia medzi ďalšie vlastnosti objektov patrí **dedičnosť** - **inheritance** - od definovaného objektu môžeme odvodiť celú hierarchiu objektov t. j. potomkov, ktoré dedia prístup k dátovým aj programovým zložkám a **polymorfizmus** - **zdieľanie** akcií v hierarchii objektov.

3.2 Zopakujme si, čo vieme z „Pascalu“!

Logické hodnoty, logické výrazy

- Každý výraz sa vyhodnocuje zľava doprava,
- vždy je buď pravdivý alebo nepravdivý, má jednu z dvoch logických hodnôt true alebo false,
- logickú hodnotu vrátia napr. relácie: <, <=, =, <>, >=, >,
- logické operátory: not, or, and, xor,
- priorita operátorov:

  not
  *, /, div, mod, and
  +, –, or, xor
  =, <>, <, >, <=, >=, in
  unárne
  multiplikatívne
  aditívne
  relačné

FOR - cyklus = cyklus a určitým počtom opakovaní

for *riadiaca_premenná:=výraz1* **to** *výraz2* **do** *príkaz*

Pre riadiacu premennú (premennú cyklu for) platí:

- priradí sa jej hodnota akú nadobúda výraz1,
- výraz 2 sa vyhodnotí na začiatku cyklu,
- ak je jeho hodnota menšia ako hodnota výrazu 1, cyklus sa nevykoná,
- ak je výraz 1 = výraz 2, cyklus prebehne práve raz,
- inak riadiaca_premenná prechádza rozsahom hodnôt medzi výraz 1 a výraz 2,
- musí byť ordinálneho typu (zatiaľ poznáme len celočíselný ordinálny typ - integer),
- jej hodnotu programátor v priebehu cyklu nesmie meniť, t. j. nesmie byť na ľavej strane žiadneho priraďovacieho príkazu počas prechodu for-cyklu,
- po skončení cyklu má nedefinovanú hodnotu.

Poznámka: Po skončení cyklu riadiaca premenná nemá hodnotu, ktorú nadobúda výraz 2 + 1 ani žiadnu inú, jej podobnú.

While - cyklus = cyklus s podmienkou na začiatku

Táto programová konštrukcia slúži na opakovanie príkazov, kým je splnená nejaká podmienka.
Všeobecný tvar príkazu je:

while *podmienka* **do** *príkaz*

- otestuje sa podmienka a ak je pravdivá (true), tak sa vykoná príkaz (t. j. telo cyklu) a celé sa to opakuje,
- ak bude podmienka nepravdivá (false), tak sa telo cyklu už nevykoná a pokračuje sa na príkaze za cyklom,
- ak podmienka bude mať stále hodnotu true, tak vznikne nekonečný cyklus - ak nekonečný cyklus (alebo veľmi dlho trvajúci cyklus) neobsahuje špeciálnu vsuvku pre Windows, tak počas cyklu sa systém nedostane ku slovu, a tým sa napr. normálne nedá zhodiť bežiaci program,
- jedným z riešení je raz začas použiť volanie **Application. ProcessMessages;**, ktoré pustí Windows na chvíľu ku slovu a tým dovolí používateľovi ukončiť aplikáciu alebo zatlačiť tlačidlo a pod. - toto volanie ale trochu spomalí priebeh výpočtu.

Repeat - cyklus = cyklus s podmienkou na konci
Táto programová konštrukcia slúži na opakovanie príkazov, až kým nie je splnená nejaká podmienka.

Všeobecný tvar príkazu je:

repeat
príkaz
príkaz 2
...
príkaz k
until *podmienka*

- tento cyklus opakovane vykonáva telo cyklu (príkazy medzi slovami repeat a until) a po každom takomto vykonaní týchto príkazov, otestuje podmienku cyklu (logický výraz za slovom until),
- keď je podmienka splnená (hodnota je true), cyklus končí a pokračuje sa príkazmi za cyklom,

- keď je podmienka nepravdivá (hodnota je false), cyklus pokračuje - hovoríme tomu podmienka ukončenia cyklu,
- tento cyklus sa správa „opačne“ ako cyklus while,
- telo cyklu sa vykoná vždy aspoň raz, aj keď je podmienka hneď na začiatku splnená.

If a Case

if *podmienka* **then** *príkaz 1* **else** *príkaz 2*

- ak je podmienka splnená, vykoná sa vnorený príkaz 1, ak je hodnota podmienky nepravda, tak sa vykoná vnorený príkaz 2. Neúplný príkaz nemá časť „else“.

```
case výraz of
   konštanta1: príkaz1;
   konštanta2: príkaz2;
   ...
   else príkazy
end;
```

pričom vetva else (podobne ako pri if) môže chýbať, výraz aj konštanty môžu byť len celočíselného typu.

Procedúry – podprogramy a funkcie

```
procedure MENO
(formálne parametre);
   ... // lokálne deklarácie
pre procedúru
   begin
   ... // telo procedúry
   end;
```

```
function MENO
(formálne parametre)
typ;
   ... // lokálne deklarácie
pre funkciu
   begin
   ... // telo funkcie
   end;
```

- pri volaní podprogramu mu môžeme poslať nejaké hodnoty, aby sme nemuseli rôzne špeciality nastavovať v globálnych premenných, v podprograme zadefinujeme špeciálne lokálne premenné, tzv. formálne parametre, do ktorých sa pred samotným volaním priradia (inicializujú) hodnoty skutočných parametrov.

Štandardné jednoduché typy INTEGER, REAL, BOOLEAN, CHAR

Integer
- hodnoty sú z intervalu -2147483648 .. 2147483647,
- zaberá 4 bajty = 32 bitov, 1bit znamienko,
- definované sú aritmetické operácie +, -, *, div, mod, obidva operandy sú celé čísla,
- relačné operácie <, <=, ..., funkcie pred a succ, príkazy inc a dec,
- štandardné funkcie, napr. sqr, abs,
- šestnástkové konštanty - $, napr. $1e=30.

Real
- čísla vyjadrené desatinným zápisom aj s exponentom ,
- obsahuje des. bodku, exponent približne ±300,
- real ± (5.0 x 10^–324 .. 1.7 x 10^308) 15-16 číslic 8 bajtov,
- ukážky reálnych čísel 1.23, -3.23989E+02, 9.89E-01,
- pre každé číslo nevieme určiť nasledovné reálne číslo, t. j. medzi 0 a 0.0001 možno existuje číslo, nemá zmysel nasledovník, t. j. nefunguje pred ani succ, inc, dec,
- operácie - +, -, *, /,
- automatická konverzia - keď sa pri reálnej operácii vyskytne celočíslený operand konvertuje sa na reálne číslo, napr. 1+1.5, 1/3 = 1.0/3.0,
- reálne operácie veľmi často majú chyby spôsobené realizáciou v počítači,
- štandardné funkcie, napr. sqrt, sqr, sin, cos, abs.

Boolean - konštanty - predefinované identifikátory konštanty true, false (ako predtým definované naše konštanty), v pamäti zaberá 1 bajt,
- operácie and, or, not, xor,
- FALSE,
- štandardné funkcie: ord, boolean(0 alebo 1), napr. boolean(0)=FALSE, pred, succ.

Char
- ordinálny typ, ktorý obsahuje sadu 256 znakov (tzv. ASCII), tieto sú usporiadané vnútorne sú reprezentované číselným kódom 0..255, t. j. zaberajú 1 bajt (8 bitov),
- znakové konštanty v apostrofoch alebo #kód,
- treba si pamätať: ' '<...<'0'<'1'<...<'9'<...<'A'<'B'<...<'Z'<...<'a'<'b'<...<'z',
- znakové funkcie: pred, succ, ord, chr (char),

- treba si pamätať: ord(' ')=32; ord('0')=48; ord('1')=49; ... alebo #32=' '; #48='0'; #49='1'; ... ord('A')=65; ord('a')=ord('A')+32=97; ... t.j. #65='A'; #97='a'; ...,
- ak c obsahuje znak cifry ('0'..'9'), tak ord(c)-ord('0')=cifra (podobne pre písmená),
- príkazy inc, dec fungujú aj pre znaky (napr. c:='A'; inc(c,32);).

3.3 Práca s farbami

Už sme sa zoznámili s niekoľkými situáciami, keď sme zadávali nejakú farbu (farba pera, farba výplne, farba pera) - doteraz sme používali preddefinované konštanty, napr. clBlack, clRed, clBlue a pod.

Vo Windows sa farba určuje trojicou čísel, ktoré vyjadrujú v akom množstve treba "namiešať" tri základné farby - červenú, zelenú a modrú, aby vznikla žiadaná farba - množstvá týchto zložiek sa zadávajú celým číslom od 0 do 255 , napr. ak namiešame 0 červenej, 0 zelenej a 255 modrej, vznikne jasná modrá farba; 0 červenej, 0 zelenej a 0 modrej je čierna; všetky zložky po 255 znamená bielu; 255 červenej, 255 zelenej a 0 modrej vyrobí žltú a pod.

Vo Windows sa tieto zložky nazývajú **Red Green Blue** a v Delphi môžeme farby definovať aj pomocou **funkcie RGB**, ktorá dostáva 3 čísla v desiatkovej sústave od 0 do 255, popisujúce zastúpenie jednotlivých zložiek základných farieb.

Mená niektorých preddefinovaných farieb:

clBlack	RGB(0,0,0)
clMaroon	RGB(128,0,0)
clGreen	RGB(0,128,0)
clOlive	RGB(128,128,0)
clNavy	RGB(0,0,128)
clPurple	RGB(128,0,128)
clTeal	RGB(0,128,128)
clGray	RGB(128,128,128)
clSilver	RGB(192,192,192)
clRed	RGB(255,0,0)
clLime	RGB(0,255,0)
clYellow	RGB(255,255,0)
clBlue	RGB(0,0,255)
clFuchsia	RGB(255,0,255)
clAqua	RGB(0,255,255)
clLtGray	RGB(192,192,192)
clDkGray	RGB(128,128,128)
clWhite	RGB(255,255,255)

4 NIEKOĽKO GRAFICKÝCH PRÍKAZOV

4.1 Niektoré grafické procedúry

g.MoveTo(x,y); - presun pera na súradnice [x y],
g.LineTo(x,y); - čiara z doterajších súradníc na súradnice [x y],
g.Rectangle(x1,y1,x2,y2); - pravouholník s ľavým horným a pravým dolným rohom,
g.Ellipse(x1,y1,x2,y2); - elipsa, vpísaná do pravouholníka s ľavým horným a pravým dolným rohom,
g.Polyline(pole-bodov); - vykreslí mnohouholník,
g.Polygon([Point(x,y),Point(x,y),Point(x,y),...]); - pospája body a vyplní útvar (mnohouholník) zadanou farbou.

Použitie príkazu demonštruje nasledujúci príkaz:

```
begin
  g.Brush.Color := clRed;
  g.Polygon([Point(50,50),Point(150,50),
       Point(100,150),Point(250,150)]);
end;
```

g.FloodFill(x,y,g.Pixel[x,y],fsSurface); - vyplní oblasť výplňou "fsSurface",
g.FillRect(Rect(X1,Y1,X2,Y2)) - nakreslí vyplnený obdĺžnik nastavenou farbou výplne.

4.2 Nastavenie farby a štýlu pera

g.Pen.Color:=farba; - nastavenie farby pera, namiesto slova farba doplňte jednu z konštánt (**clBlack, clWhite, clRed, clBlue, clYellow...** - ostatné nájdete v helpe),
g.Pen.Width:=hrúbka; - nastavenie hrúbky pera, namiesto slova hrúbka použijeme ľubovoľnú celočíselnú konštantu,
g.Pen.Style:=štýl; (psSolid, psDash, psDot, psDashDot, psDashDotDot, ...) - nastavenie štýlu pera, namiesto slova štýl doplňte jednu z konštánt,
g.Brush.Color:=farba; - nastavenie farby výplne,
g.Brush.Style:=štýl; (bsSolid, bsClear, bsHorizontal, bsVertical, ...) - nastavenie štýlu výplne (obdĺžnika, elipsy, polygonu).

RGB (červená, zelená, modrá) - funkcia na „namiešanie“ farby z troch základných farieb. Každá zložka je z intervalu 0..255. Na vygenerovanie náhodnej farby môžeme využiť generátor náhodných čísel. (viď nižšie)

4.3 Nastavenie farby, veľkosti a typu písma

g.TextOut(x,y,'nejaký-text'); - výpis textu na súradnice [x y],
g.Font.Color:=farba; - farba písma,
g.Font.Height:=veľkosť; - výška písmen,
g.Font.Name:=meno-fontu; - 'Arial','Times New Roman', ... - typ (font) písma,
g.Font.Style:=štýl; - podmnožina [fsBold, fsItalic, fsUnderline, fsStrikeOut] - rez písma,
g.Pixel[x,y] - vráti farbu ľubovoľnej bodky v ploche,
g.Pixel[x,y]:=farba; - zafarbí ľubovoľnú bodku na ploche.

4.4 Ďalšie pomocné funkcie

Sin(x), Cos(x), ... - hodnota goniometrických funkcií,
Round(x), Trunc(x), Abs(x),... - zaokrúhľovanie, odrezanie des. časti, absolútna hodnota čísla x,
Random(n) - náhodné celé číslo od 0 do n-1,
Randomize – nastavenie počiatku generátora náhodných čísel na náhodné miesto,
IntToStr(i) resp. FloatToStr(r) - vyrobí textový reťazec z celého, resp. reálneho čísla,
StrToInt(i) resp. FloatToStr(i) - vyrobí celé, resp. reálne číslo z textového reťazca,
Val (S, var V; var Code:integer); - prevedie reťazec **s** typu string na celé číslo. Ak hodnota premennej **Code** je 0, reťazec pozostával iba z číslic, v inom prípade nastala chyba.

Image1.Width - vráti šírku plochy,
Image1.Height – vráti výšku plochy,
Image1.ClientRect - vráti veľkosť obdĺžnika popisujúci grafickú plochu,

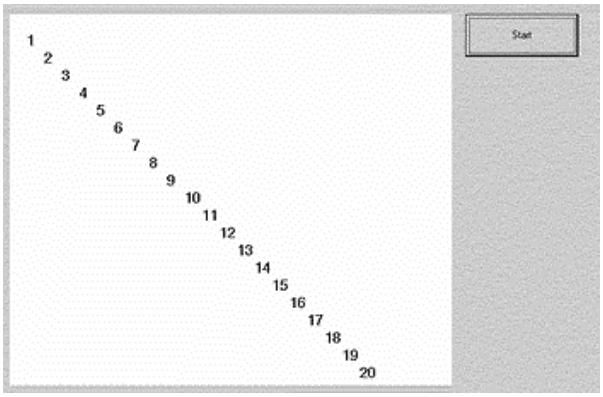
4.5 Zmazanie plochy

PRÍKLAD: Použitie predchádzajúcich príkazov na zmazanie celej plochy:

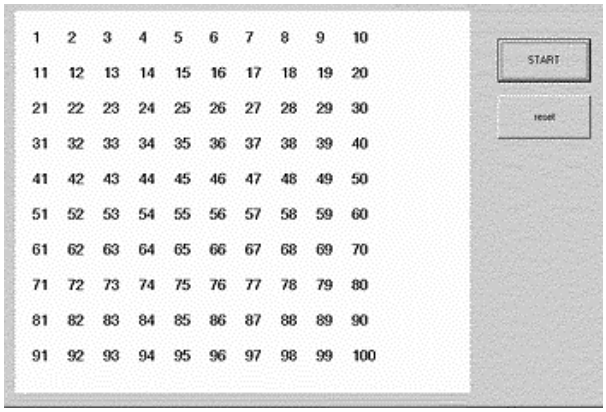
```
g.Brush.Color:=farba;
g.Brush.Style:=bsSolid;
g.FillRect(Image1.ClientRect);
```

5 JEDNODUCHÉ ÚLOHY

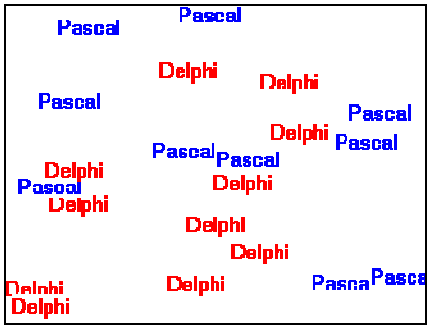
1. Vypíše čísla 1..20 na uhlopriečku zadefinovanej grafickej plochy:



2. Vypíše čísla 1..100 do riadkov a stĺpcov dvoma spôsobmi (výpis čísel vo forme matice po riadkoch znázorňuje obrázok):

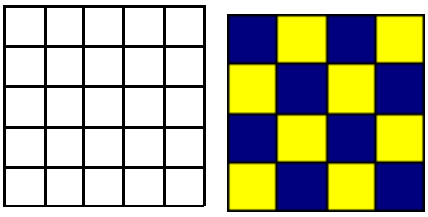


3. Na každé stlačenie tlačidla sa napíše na náhodnej pozícii modré slovo Pascal a červené Delphi. Náhodné číslo získate pomocou funkcie Random (HornáHranica). Na začiatku programu, napr. pri vytváraní formulára, použite procedúru Randomize, lebo inak sa program správa stále rovnako, t. j. slová sa vždy po spustení programu vypisujú na rovnakých pozíciách.

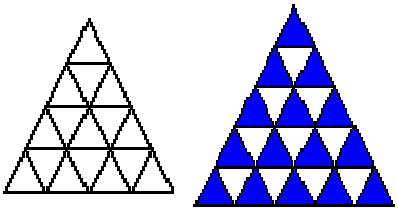


4. Na stlačenie tlačidla vypíše modré slová 'I love Delphi' so sivým tieňom. Vytvorte konštantu **const veta = 'I love Delphi'** (nezabudnite urobiť priesvitné pozadie písmen).

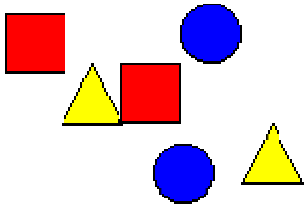
5. Na stlačenie tlačidla vytvorí štvorcovú sieť **NxN** štvorčekov s veľkosťou štvorčeka **a**. Obe tieto čísla vytvorte ako konštanty (obrázok vznikol pre deklaráciu **const a = 20; N = 5**). Obrázok vytvorte najprv pomocou príkazov **MoveTo** a **LineTo**, potom dvoma príkazmi **Rectangle** v cykle a nakoniec jedným príkazom v dvoch v sebe vnorených cykloch. (Pri poslednom spôsobe môžeme zároveň nastaviť farbu všetkých štvorčekov.) a) Upravte predchádzajúci príklad tak, že na sieti vytvoríte šachovnicový efekt.



6. Na stlačenie tlačidla nakreslí z trojuholníkov nasledujúcu pyramídu. Pyramída vznikla pre konštanty **const a = 20; N = 4** resp. pyramída s vyplnenými trojuholníkmi pre **const a = 20; N = 5**.

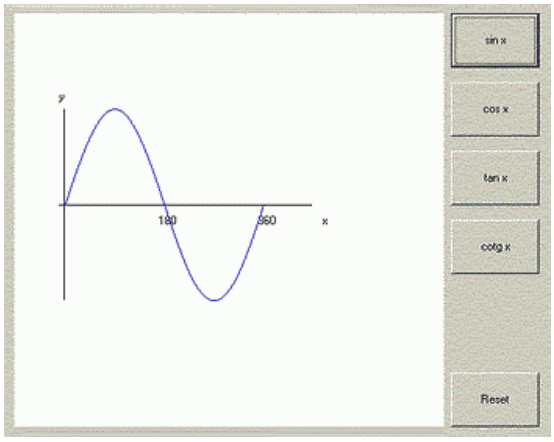


7. Po stlačení tlačidla na náhodných pozíciách, ale v štvorcovej sieti určenej veľkosťou konštanty **a** – strany štvorca, trojuholníka a polomeru kružnice, vykresľujte červený štvorec, žltý trojuholník a modrú kružnicu.

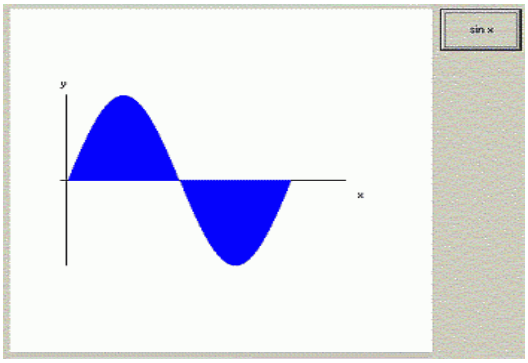


8. Nakreslite sústredné kružnice. Uvedomte si, čo znamená, že kružnice sú vyplnené. Vymyslite taký spôsob kreslenia, aby to nevadilo a nakreslite ich tak, aby bola každá z nich rôznej (náhodnej) farby.

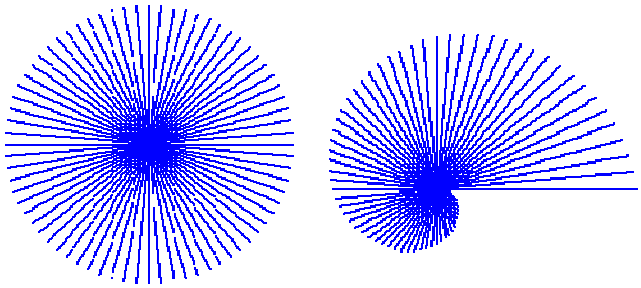
9. a) Nakreslite priebeh funkcie **sin(x)**.
   b) Vytvorte tlačidlá pre nakreslenie priebehov zvyšných goniometrických funkcií a tlačidlo pre zmazanie grafickej plochy podľa predlohy:



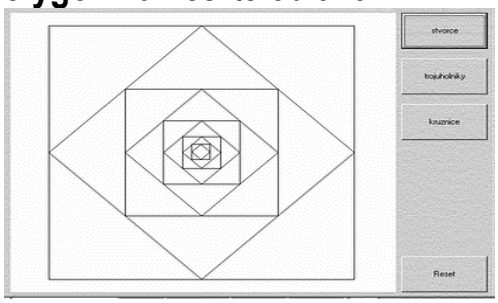
10. Nakreslite priebeh funkcie **sin(x)** s vyplneným efektom - získame ho pomocou čiar ťahaných vždy od x-ovej osi k vypočítanej y-ovej súradnici:



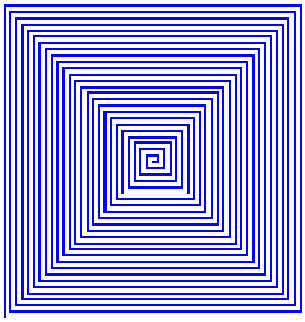
11. Pomocou funkcií **sin(x)** a **cos(x)** najprv nakreslite kružnicu, potom využitím tých istých funkcií nakreslite nasledujúcich „ježkov“



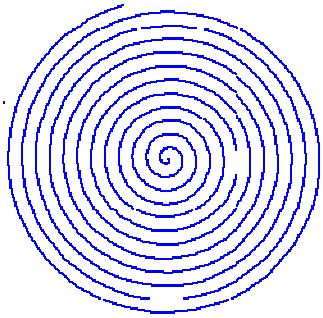
12. Pomocou príkazu **Polygon** nakreslite obrázok:



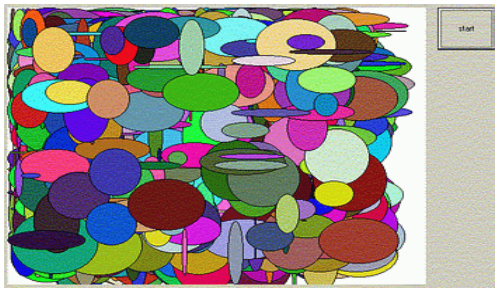
13. Po stlačení tlačidla sa nakreslí na grafickú plochu špirála z **n** čiar:



14. Po stlačení tlačidla sa nakreslí na grafickú plochu špirála podľa obrázka:

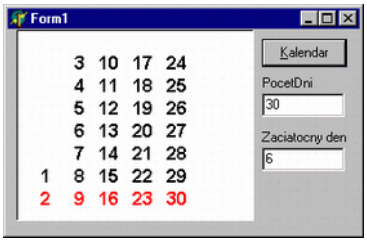


15. Po stlačení tlačidla sa vykreslí na obrazovku 100 elíps náhodnej farby, náhodnej veľkosti na náhodné miesto:



6 EDIT A LABEL

6.1 Príklad - Kalendár



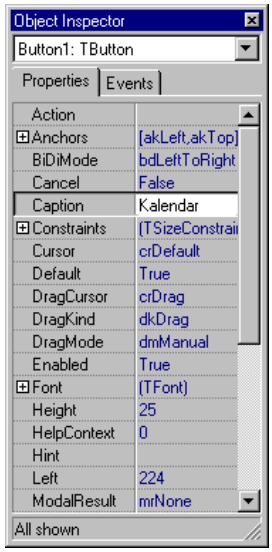
Navrhnime program na vytváranie kalendára na zvolený mesiac. Dni nech sa vypíšu po týždňoch do stĺpcov, nedele červeným.

Používateľ musí zadať počet dní v mesiaci a číslo dňa, ktorým daný mesiac začína - 1 pre pondelok, 2 pre utorok, 3 pre stredu...

Riešenie:

Do prázdneho formuláru tak ako predtým položíme grafickú plochu - komponent **Image** z palety komponentov **Additional** a tlačidlo **Button** z palety **Standard**. Tomuto tlačidlu zmeňme hneď meno.

V Objekovom inšpektore (ak ho nevidíte, stlačte F11) sa nastavme na časť **Caption** a prepíšme **Button1** na **Kalendár**. Zároveň sa pozrime, čo pribudlo v Unite1!




```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,

type
  TForm1 = class(TForm)
    Image1: TImage;
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

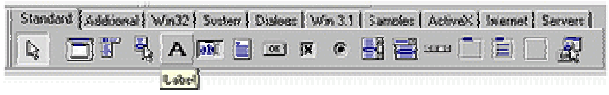
V **Objektovom inšpektore** si všimnime ešte položku **Name**, ktorú Delphi uplatnia pri **názvoch premenných**. Prepíšte ju napr. na Kalendár a opäť sa pozrite do deklarácií **Unitu1**.

```
type
  TForm1 = class(TForm)
    Image1: TImage;
    Kalendar: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

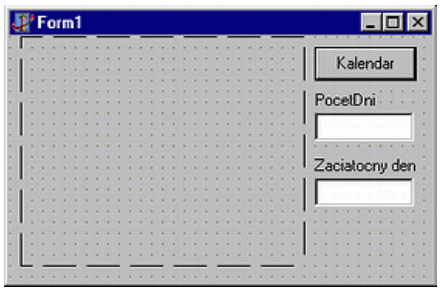
Teraz potrebujeme, aby sme realizovali vstupy od užívateľa - počet dní mesiaca a začiatočný deň. Na to použijeme **Edit** (editovacie okienko) z palety komponentov **Standard** a vložíme ho do formulára. Takéto editovacie okienka musíme vložiť dve. Používame ich vtedy, keď potrebujeme od užívateľa niečo načítať, niečo mu oznámiť - vypísať krátku správu.


```
type
  TForm1 = class(TForm)
    Image1: TImage;
    Kalendar: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

A opäť sa pozrite do deklarácií **Unitu1**.
Ak chceme, aby mali dané editovacie okienka nadpisy, použime komponent **Label** opäť z palety **Standard**. Nad každé editovacie okienko vložte takýto **Label** a v **Objektovom inšpektore** mu zmeňte položku **Caption** na príslušný nadpis, ten môže pozostávať z jedného aj viac slov, ak sa slová nezmestia, zväčšite si vo formulári príslušný **Label** ťahaním za niektorý roh.



Formulár vyzerá ako na obrázku:
Opäť sa pozrite do definícií **Unitu1**, ktorý za vás vytvára prostredie Delphi. Projekt si uložte.



Teraz máme pripravené prostredie, môžeme začať programovať. Najprv dvojkliknime na **Form1** (pozor, aby sme neklikli napr. na grafickú plochu, alebo na niečo iné). Takto nás Delphi prepnú do **Unitu1**, kde nám pripravili definíciu procedúry **TForm1.FormCreate**. Tak ako predtým doplňme do nej zjednodušenie odkazu na našu grafickú plochu - priradenie **g:=Image1.Canvas;** a deklaráciu premennej **g**.
Kalendár sa má nakresliť po kliknutí na tlačidlo - túto **udalosť** spracujeme, keď dvojklikneme vo formulári na toto tlačidlo - opäť nás Delphi prepnú do **Unitu1**, kde je už pripravená hlavička procedúry:

```
procedure TForm1.KalendarClick(Sender: TObject);
 begin
      //Tu napíšte riešenie - výpis samotného kalendára!
 end;
```

Pomôcky:
Ako získať zadané hodnoty od užívateľa z editovacích okien?

```
    var PocetDni, ZacDen:integer;
```

```
...
PocetDni:=StrToInt(Edit1.Text);
ZacDen:= StrToInt(Edit2.Text);
```

Ako nič nerobiť, keď nie sú zadané hodnoty v editovacích okienkach?

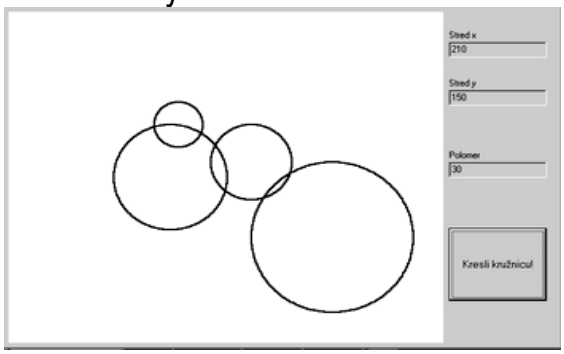
```
if Edit1.Text='' then exit;
if Edit2.Text='' then exit;
```

Ako otestovať, či hodnoty v okienkach sú čísla?

```
var PocetDni : integer;
    Code : integer;
    ...
    Val(Edit1.Text, PocetDni, Code);
    if Code<>0 then exit;
```

6.2 Úlohy na riešenie

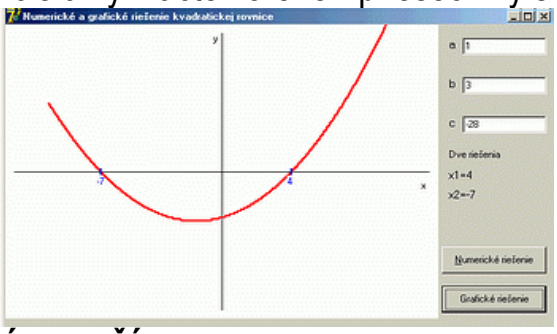
1. Vytvorte projekt, ktorý na grafickej ploche vykreslí **kružnicu s daným stredom a polomerom**. Súradnice stredu a polomer kružnice zadá užívateľ v editovacích oknách, po každom stlačení tlačidla sa vykreslí kružnica podľa zadaných parametrov. Hodnoty v editovacích oknách otestujte.



2. Vytvorte projekt, ktorý na grafickej ploche vypíše **zadané meno v 3D štýle**. Meno zadá užívateľ v editovacom okne. Súčasťou projektu je aj tlačidlo na vymazanie grafickej plochy a editovacieho okna.

3. Vytvorte projekt **RIEŠENIE KVADRATICKEJ ROVNICE**.
   a) Numerické riešenie a počet riešení vypíše prostredníctvom „Label“, pričom koeficienty sa načítajú z editovacích okienok.

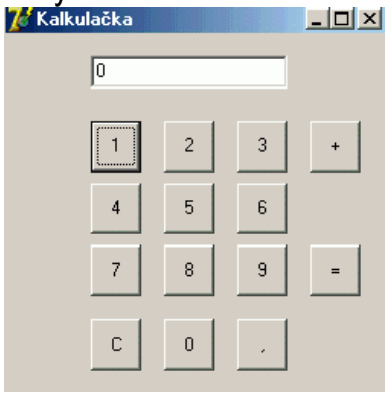
b) Do projektu doplňte grafické riešenie - zobrazenie odpovedajúcej kvadratickej funkcie a vyznačte korene - priesečníky s x-ovou osou.



4. Vytvorte projekt **HÁDAJ ČÍSLO**, ktorý vygeneruje náhodné hádané číslo z istého intervalu celých čísel a porovná ho s údajom od užívateľa. Vypíše hlásenie, či bolo číslo uhádnuté. Ak nie, ponúkne pomoc. Ak je číslo správne, vypíše počet pokusov.

5. Vytvorte projekt **PALINDROM**, ktorý otestuje vstupný reťazec, či tvorí palindrom. Ďalej bude generovať rôzne päťpísmenové slová zo spoluhlások a samohlások, ktoré opäť môžeme otestovať, či tvorí palindrom.

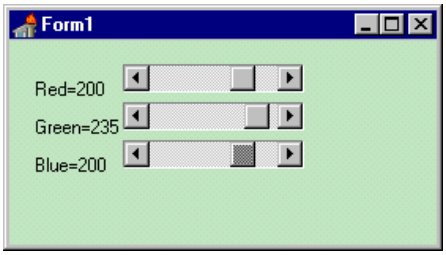
6. Vytvorte projekt **JEDNODUCHÁ KALKULAČKA**, ktorý na grafickej ploche bude obsahovať tlačidlá cifier a matematickej operácie sčítania. Vybrané cifry a mat. operácia sa bude zobrazovať v editovacom okne. Po stlačení tlačidla „=“ sa v editovacom okne zobrazí výsledok.



7 SCROLLBAR, TRACKBAR

7.1 Komponent scroolbar - Miešanie farieb

Navrhnime projekt, ktorý bude miešať farby a farbiť tak napr. plochu formuláru.



V riešení použijeme možnosti nového prvku - posuvníka - **scrollbar** - ktorý umožňuje spojite meniť hodnoty vo zvolenom intervale. Na ne sa môžeme odvolať parametrom **Position** - ktorý vráti aktuálnu pozíciu posuvníka.
Výsledok projektu predstavuje obrázok:

RIEŠENIE:

Do prázdneho formuláru položíme „grafickú plochu“ - komponent **Image** z palety komponentov **Additional**. Ďalej do formuláru položíme 3 komponenty **ScrollBar** (dajú sa aj kopírovať) z palety **Standard** a pridajme im návestia **Label**:
Zadefinujte novú metódu formuláru, napr. **ZmenVsetko**, ktorá nastaví aktuálnu farbu formulára a aktualizuje čísla v návestiach pre jednotlivé zložky farieb z pozícií posuvníkov, ako ich nastavil užívateľ:

```
procedure TForm1.ZmenVsetko;
begin
  Form1.Color:=RGB(ScrollBar1.Position,
                   ScrollBar2.Position,
                   ScrollBar3.Position);
  Label1.Caption:='Red='
                       +IntToStr(ScrollBar1.Position);
  Label2.Caption:='Green='
                       +IntToStr(ScrollBar2.Position);
  Label3.Caption:='Blue='
                       +IntToStr(ScrollBar3.Position);
end;
```

V Objektovom inšpektore zadefinujme každému **ScrollBaru** udalosť **OnChange**. Zmeny každého **ScrollBaru** vyvolajú metódu **ZmenVsetko**:

```
procedure TForm1.ScrollBar1Change(Sender: Tobject);
begin
  ZmenVsetko;
end;
```

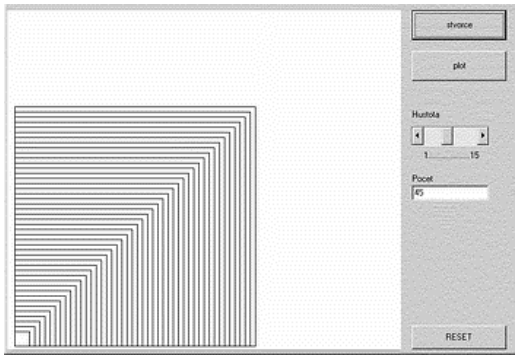
7.2 Posuvná lišta - TrackBar

Analogickú funkciu, akú má scrollbar, má aj komponent Trackbar. Je vhodnejší pri presných nastaveniach číselných hodnôt z daného intervalu.
Vyberieme komponent **posuvná lišta - TrackBar** z palety komponentov **WIN32**, položme ho do formulára a prispôsobme jeho veľkosť. V **Objektovom inšpektore** si nastavme hodnoty **Min** a **Max**.
Pri jeho posúvaní vznikne udalosť **OnChange** - aktuálnu hodnotu na lište môžeme zistiť stavovou premennou **Position**. Naprogramujeme udalosť **OnChange** pre zmenu hrúbky pera:

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  g.Pen.Width:=TrackBar1.Position;
end;
```

7.3 Úlohy na riešenie

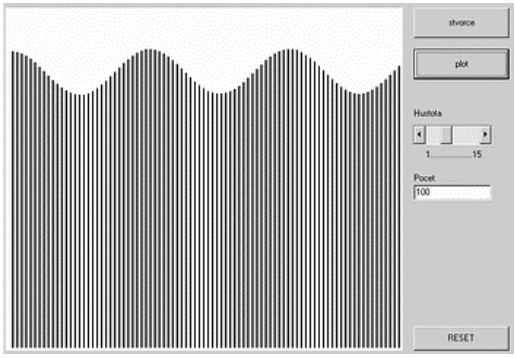
1. Vytvorte projekt, ktorý na grafickej ploche vykreslí N štvorcov so zväčšujúcou sa stranou. Prírastok strany štvorca je určený hustotou nastaviteľnou užívateľom pomocou posuvníka a počet štvorcov určuje užívateľ v editovacom okne. Súčasťou projektu je tlačidlo pre zmazanie grafickej plochy.



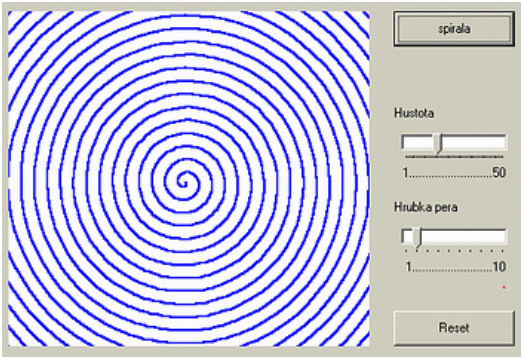
2. Zmeňte úvodný projekt tak, aby sa menila farba prázdnej malej grafickej plochy (nie formulára) podobne, ako to býva zvykom v grafických editoroch, kde sa vybraná - aktuálna farba zobrazí v malom okienku. (Grafická plocha môže mať malé rozmery.)

3. Vytvorte projekt, ktorý na grafickej ploche vykreslí plot z N "latiek" so zväčšujúcou sa výškou podľa fukcie sinus. Hustota latiek je nastaviteľná užívateľom pomocou posuvníka a počet latiek určuje užívateľ v editovacom

okne. Projekt môžete doplniť o posuvník pre hrúbku latiek. Súčasťou projektu je tlačidlo na zmazanie grafickej plochy.

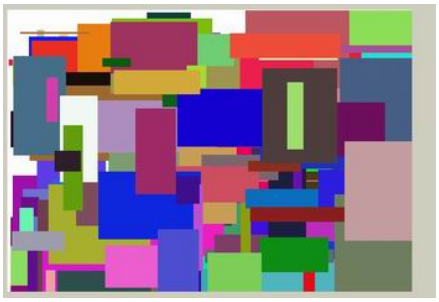


4. Vytvorte projekt, ktorý na grafickej ploche vykreslí kruhovú špirálu s nastaviteľnou hustotou a hrúbkou čiary. Hustota aj hrúbka čiary je nastaviteľná užívateľom pomocou posuvníkov v reálnych intervaloch hodnôt. Súčasťou projektu je tlačidlo na zmazanie grafickej plochy.



8 TIMER

8.1 Príklad 1. - Vykresľovanie útvarov



Navrhnime jednoduchý program, v ktorom sa bude plocha zapĺňať farebnými obdĺžnikmi náhodnej veľkosti - napr. každú desatinu sekundy v nej pribudne nový obdĺžnik.

Výsledok projektu predstavuje obrázok:

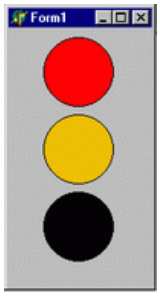
Ako zabezpečiť, aby sa niečo dialo pravidelne bez zásahu používateľa?

Používa sa na to komponent **Časovač - Timer** z palety **System**, je to neviditeľný komponent, ktorý sa pri kompilácii nezobrazí, ale môžeme sa naň v programe odvolávať - udalosťou **OnTimer**. Časovač si môžeme predstaviť ako hodiny, ktoré s nejakou frekvenciou „tikajú“ - pri každom tiknutí môžu vykonať nejakú akciu. Frekvenciu "tikania" nastavíme v stavovej premennej **Interval**, pozastavenie, resp. naštartovanie časovača môžeme urobiť logickou stavovou premennou **Enabled** - **true** znamená, že hodiny bežia, **false** znamená, že sme ich pozastavili.

Riešenie:
Začnime novú aplikáciu, štandardne do nej položme Image, prípadne si zadefinujte skratku na jeho Canvas, projekt uložme.
Položme do formulára komponent Timer, pozrime si jeho Property v Objektovom inšpektore a nastavte Interval na 10.

Dvojkliknime vo formulári na spomínaný komponent Timer a v editovacom okne Unitu zadefinujte príslušné správanie - vykresľovanie vyplnených obdĺžnikov (príkaz FillRect) na náhodné miesto náhodnej veľkosti náhodnou farbou (funkcie RGB a Random).

8.2 Príklad 2. - Semafor



Navrhnime program, ktorý bude zobrazovať fungovanie semaforu.
Pri riešení budeme potrebovať kresliť kružnice? V minulých aktivitách sme videli vykresľovanie kružníc do Image. Ak však chceme jednoduché tvary ako napr. štvorec, kružnicu..., môžeme použiť aj komponent **Shape** z palety **Additional**.

Riešenie:

Začnime novú aplikáciu, položme do nej jeden **Shape**, v **Objektovom inšpektore** mu zmeňme **property Shape** na **Circle**. Podobne dajme do formulára ďalšie dva komponenty Shape - komponenty sa dajú aj kopírovať! Ďalej vložte do formuláru Timer a navrhnite udalosť OnTimer.

Pri vytváraní formulára všetky 3 kruhy vyfarbite na čierno. Potom nech sa zobrazuje fungovanie semaforu - prepínanie farieb - zo zelenej cez oranžovú na červenú a naopak. Nezabudnite ďalšie farby prefarbovať na čierno.
Na vyfarbenie kruhov použite jeho „property“ Brush.Color, ktorú môžete nastaviť v Objektovom inšpektore, ale aj priamo v programe: Shape1.Brush.Color:=clRed;
Ďalej navrhnite tlačidlo pre chodcov - SpeedButton z palety Additional, pozrite si jeho property Down. Keď ho niekto zatlačí, dobehne cyklus semaforu na červenú a potom program počká, napr. jednu sekundu a semafor sa znovu rozbehne. Na túto udalosť môžete použiť druhý Timer, ktorý bude spolupracovať s prvým, ktorý je určený pre semafor. Využite property Enabled oboch časovačov.

8.3 Úlohy na riešenie

1. Vytvorte projekt, ktorý spustí prostredníctvom timer-a vykresľovanie náhodne hrubých čiar na náhodné miesto v odtieňoch červenej, čím získate efektné obrázky. Vykresľovanie ukončite vloženým tlačidlom.

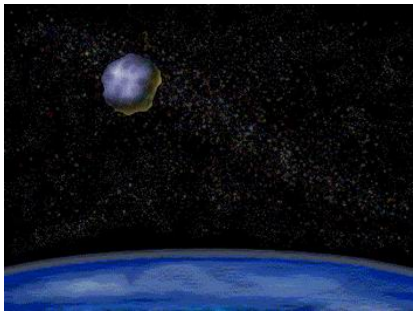
2. Vytvorte projekt, ktorý bude simulovať pohyby meteoru na povrch Zeme. Pri riešení využite telá metód - konštruktora, deštruktora a udalosti OnTimer časovača..

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  g:=Image1.Canvas;
  X:=Image1.Width div 2;
  Y:=Image1.Height div 2;
  Pozadie:=TBitmap.Create;
```

```
  Pozadie.LoadFromFile('obrazky\Zem.bmp');
  Meteor:=TBitmap.Create;
Meteor.LoadFromFile('obrazky\Meteor.bmp');
  Meteor.TransparentColor:=clWhite;
  Meteor.Transparent:=True;
end;

procedure TForm1.FormDestroy(Sender:TObject);
begin
  Meteor.Free;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  g.Draw(0,0,Pozadie);
  g.Draw(X-Meteor.Width div 2,Y-Meteor.Height div 2,Meteor);
  X:=X+4;
  Y:=Y+2;
  if X<0 then X:=X+Image1.Width;
  if X>=Image1.Width then X:=X-Image1.Width;
  if Y<0 then Y:=Y+Image1.Height;
  if Y>=Image1.Height then Y:=Y-Image1.Height;
end;
```



3. Vytvorte projekt, ktorý bude zobrazovať na grafickej ploche pohyb guličky, ktorá sa bude vo vhodných časových intervaloch pohybovať po „sinusoide“. (Guličku nahraďte obrázkom chrobáka, prípadne dokreslite pozadie a načítajte bitmapový obrázok do grafickej plochy.)

9 PRÁCA S MYŠOU

9.1 Udalosti MouseMove a MouseDown

Vytvorme jednoduchý textový editor, ktorý umožňuje kresliť myšou vybranou farbou nastavenou hrúbkou pera. Potrebujeme ale vedieť pracovať s udalosťami, ktoré sú spojené s pohybom myši a jej tlačidlami. Medzi základné udalosti patria **MouseMove**, ktorá nastane pri pohybe a **MouseDown**, ktorá nastane pri súčasnom stlačení tlačidla myši.

Začneme nový projekt, do formulára položme grafickú plochu (Image1) a urobíme skratku na jej Canvas. Na záložke **Events** grafickej plochy Image1 kliknime na udalosť **MouseMove** a dopíšme telo procedúry:

```
procedure TForm1.Image1MouseMove(Sender: TObject;
                    Shift: TShiftState; X, Y: Integer);
begin
  g.Pixels[x,y]:=clBlack;
end;
```

Ako kresliť iba vtedy, ak je stlačené ľavé tlačidlo myši? Využijeme na to parameter **Shift**, ktorý je typu **TShiftState**.

Platí:
TShiftState = set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble);

Najprv sa v procedúre spýtame, či je stlačené ľavé tlačidlo a až potom urobíme čiaru. Upravte telo procedúry:

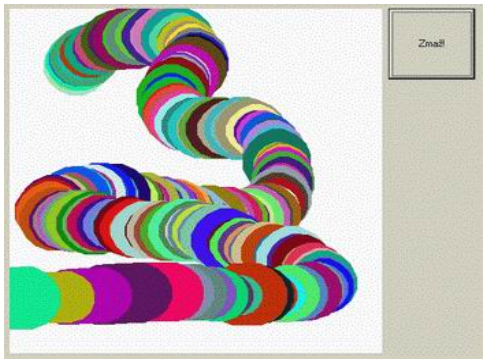
```
procedure TForm1.Image1MouseMove(Sender:TObject;
                    Shift:TShiftState; X,Y:Integer);
begin
  if Shift=[ssLeft] then g.LineTo(x,y);
end;
```

Na grafickej ploche sa však nakreslí čiara aj z miesta, kde bolo pustené tlačidlo myši - to nechceme. Pri stlačení myši musíme najprv presunúť grafické pero (MoveTo) na nové miesto a až odtiaľ kresliť - t.j. použijeme udalosť **OnMouseDown** vo formulári vyznačme **Image1**, v **Objektovom inšpektore** záložku **Events**, riadok **OnMouseDown** a dvojklikom na ňu môžeme definovať procedúru:

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

```
begin
  g.MoveTo(X,Y);
end;
```

Dodefinujme projekt tak, aby sa pri ťahaní myši so stlačeným pravým tlačidlom vykresľovali čiary náhodnej hrúbky a farby, čím vzniknú pekné efekty: Do formulára vložte tlačidlo na zmazanie grafickej plochy.

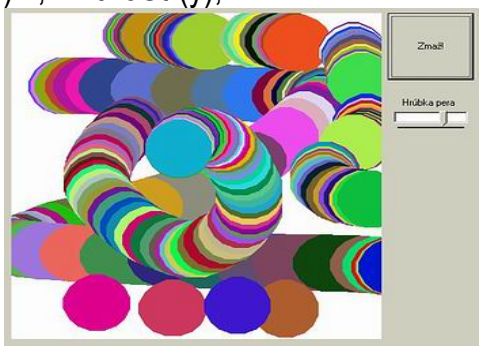


9.2 Výpis súradníc kurzora

Vypisovanie súradníc myši:

Ak chceme vidieť aktuálne súradnice myši pri pohybe nad grafickou plochou, položme na formulár prázdny komponent **Label**. Nasledujúci výpis pridáme do Image1MouseMove:

```
Label2.Caption:=IntToStr(x)+','+IntToStr(y);
```



9.3 Jednoduchý grafický editor

Už nám chýba len krok k vytvoreniu jednoduchého grafického editora, musíme vytvoriť ešte paletu farieb a aktuálnu vybranú farbu. Môžu to byť ďalšie grafické plôšky.

Položme na formulár ďalšiu graf. plochu (**Image2**), programom v tele procedúry FormCreate nakreslime do nej napr. 8 farebných obdĺžnikov. Vybraná farba sa bude zobrazovať v maličkom image (**Image3**) - položte ho na formulár

podľa obrázka. Doplňme do programu nakreslenie farebnej palety. Najskôr vložme skratky na dve nové graf. plochy (akt_farba a paleta).

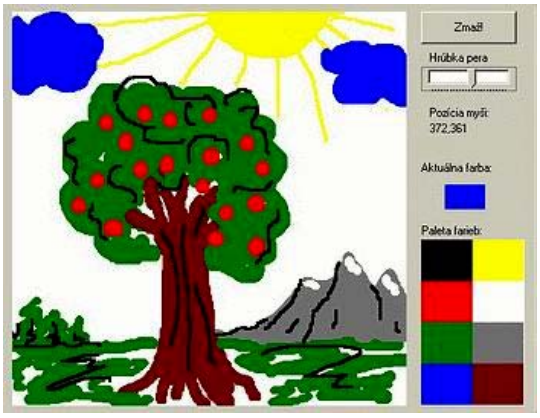
```
Private
  g:TCanvas;
  akt_farba:TCanvas;
  paleta:TCanvas;
            .....

procedure TForm1.FormCreate(Sender: TObject);
const p: array[0..7] of TColor = (clBlack, clRed,
             clGreen, clBlue, clYellow, clWhite, clGray,
             clMaroon);
var   w,h,i,j:integer;
begin
  g:=Image1.Canvas;
  akt_farba:=Image2.Canvas;
  paleta:=Image3.Canvas;
  w:=Image3.Width div 2;
  h:=Image3.Height div 4;
  paleta.Brush.Style:=bsSolid;
  for i:=0 to 1 do
    for j:=0 to 3 do
      begin
        paleta.Brush.Color:=p[i*4+j];
        paleta.FillRect(Rect(i*w,j*h,(i+1)*w,(j+1)*h));
      end;
  akt_farba.Brush.Style:=bsSolid;
  akt_farba.Brush.Color:=g.Pen.Color;
  akt_farba.FillRect(Image2.ClientRect);
end;
```

Aby to fungovalo, budeme meniť farbu pera grafickej plochy podľa kliknutého miesta v palete - **Event OnMouseDown**:

```
procedure TForm1.Image3MouseDown(Sender: TObject;
            Button: TMouseButton; Shift: TShiftState;
            X, Y: Integer);
begin
  g.Pen.Color:=paleta.Pixels[x,y];
  akt_farba.Brush.Color:=g.Pen.Color;
  akt_farba.FillRect(Image2.ClientRect)
end;
```

Výsledok projektu znázorňuje obrázok:



9.4 Úlohy na riešenie

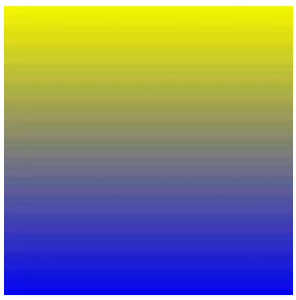
1. Navrhnite jednoduchý projekt, ktorý dovolí hrať hru „Chytaj“. Na formulári sa objaví zelený kruh. Úlohou hráča je do 1 sekundy na tento kruh kliknúť - vtedy sa zvýši počítadlo chytených, inak sa zvýši počítadlo nechytených. Po kliknutí alebo po uplynutí sekundy sa kruh objaví na inom mieste.

10 EFEKTY S OBRÁZKAMI

10.1 Hry s farbami

Tento projekt demonštruje niektoré známe efekty grafických editorov.
Na stránke nájdete vždy telá procedúr a ich účinok. Do formulára sme vložili dve grafické plochy, vytvorili skratky na ich Canvas - g1 a g2. Farebné efekty sa vykresľujú na ploche g2. Metóda „Repaint“ spomaľuje vykresľovanie.

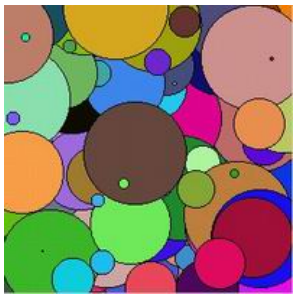
Farby



```
for I:=0 to 255 do begin
  g2.Pen.Color:=RGB(255-I,255-I,I);
  g2.MoveTo(0,I);
  g2.LineTo(256,I);
end;
```

Zameňte poradie zložiek, čím získate iné farebné prechody.

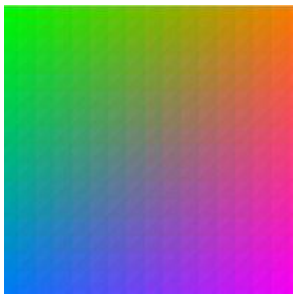
Pero a výplň



```
g.Pen.Color:=clBlack;
for I:=0 to 50 do
  begin
  g2.Brush.Color:=RGB(Random(256),
                      Random(256),Random(256));
  X:=Random(Image2.Width);
  Y:=Random(Image2.Height);
  R:=Random(50);
  g2.Ellipse(X-R,Y-R,X+R,Y+R);
end;
```

Nastavte aj obrysovú farbu útvarov (farbu pera) na vygenerovanú farbu, prípadne kreslite rôzne útvary.

Pixle



```
for Y:=0 to 255 do begin
  for X:=0 to 255 do
    g2.Pixels[X,Y]:=RGB(X,255-(X+Y) div 2,Y);
  Image2.Repaint;
end;
```

Zmeňte poradie zložiek farieb RGB pri vykresľovaní pixelov.

Kópia

Na grafickej ploche g1 načítame bitmapový obrázok (v property komponentu Image zvolíme – Picture...). Na grafickej ploche g2 sa kopírujú pixely obrázka z g1, pričom sa prepočtom vytvorí istý grafický efekt. Kópiu obrázka na g2 urobíme nasledujúcim kódom.

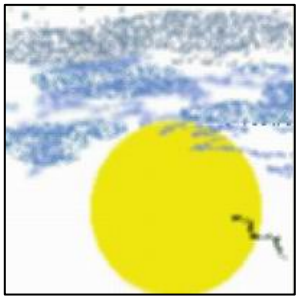

```
if not Image1.Visible then begin
  Button4.Caption:='kopia';
  Image1.Show;
  Exit;
end;
for Y:=0 to 255 do begin
  for X:=0 to 255 do
    g2.Pixels[X,Y]:=g1.Pixels[X,Y];
  Image2.Repaint;
end;
```

Zrkadlenie - symetria podľa osi x

```
for Y:=0 to 255 do begin
  for X:=0 to 255 do
g2.Pixels[X,Y]:=Image1.Canvas.Pixels[X,255-Y];
    Image2.Repaint;
end;
```

Ako by ste urobili zrkadlenie podľa osi y, resp. otočenie o 90^{o}?

Dvojnásobné zväčšenie



```
for Y:=0 to 255 do begin
  for X:=0 to 255 do
    g2.Pixels[X,Y]:=Image1.Canvas.Pixels
              [X div 2,Y div 2];
  Image2.Repaint;
end;
```

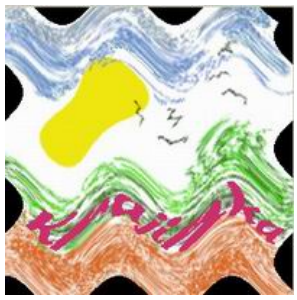
Vedeli by ste analogicky vytvoriť viacnásobné zväčšenie, či zmenšenie?

Náhodné farby

```
for Y:=0 to 255 do begin
  for X:=0 to 255 do
    g2.Pixels[X,Y]:=Image1.Canvas.
              Pixels[X+random(11)-5,Y+
              random(11)-5];
  Image2.Repaint;
end;
```

Pokúste sa zmeniť rozsah náhodných čísel. Ako to ovplyvní výsledný obrázok?

Efekty s vlnením



```
for Y:=0 to 255 do begin
  for X:=0 to 255 do
     g2.Pixels[X,Y]:=Image1.Canvas.
                     Pixels[X+Round(16* Sin(Y/16)),
                     Y+Round(16*Cos(X/16))];
  Image2.Repaint;
end;
```

Porozmýšľajte, ako by ste urobili známy efekt „Pixelize“ („utajení svedkovia“).

Inverzné farby


```
for Y:=0 to 255 do begin
  for X:=0 to 255 do begin
    F:=Image1.Canvas.Pixels[X,Y];
    r:=F mod 256;
    g:=F div 256 mod 256;
    b:=F div (256*256) mod 256;
    g2.Pixels[X,Y]:=RGB(255-r,255-g,
                                  255-b);
  end;
  Image2.Repaint;
end;
```

Čo sa zmení, ak vymeníme farby r, g, b? Nájdite analogický efekt v nejakom grafickom editore.

Odtiene šedej

```
for Y:=0 to 255 do begin
  for X:=0 to 255 do begin
    F:=Image1.Canvas.Pixels[X,Y];
    r:=F mod 256;
    g:=F div 256 mod 256;
```

```
    b:=F div (256*256) mod 256;
    i:=(r+g+b) div 3;
    g2.Pixels[X,Y]:=RGB(i,i,i);
  end;
  Image2.Repaint;
end;
```

Uvedomte si, kde sa nachádzajú odtiene šedej farby v modeli RGB.

10.2 Úlohy na riešenie

1. Vytvorte vlastný bitmapový obrázok o veľkosti 255x255 pixelov. Načítajte ho do grafickej plochy g1. Do projektu doplňte tlačidlá (pozri 4. obrázok), ktoré budú realizovať ďalšie spomínané efekty (zrkadlenie podľa osi y, zmenšenie, „utajení svedkovia, zosvetlenie, stmavenie obrázka a pod.).
2. Vytvorte projekt **FOTOALBUM**, ktorý umožní „listovať“ tam a späť albumom fotografií. Doplňte aj tlačidlo skokov na prvú a poslednú fotografiu.

11 TEXTOVÉ SÚBORY

11.1 Zopakujme si potrebné príkazy

V jazyku Turbo Pascal sme čítali dáta z textového súboru, uloženom na disku, alebo sme výsledky programu zapísali do súboru. Všetky nasledujúce typy a príkazy už poznáte (s malou zmenou):

Typy premenných

- **CHAR** - znak,
- **STRING** - reťazec,
- **TEXTFILE** - textový súbor

Procedúry a funkcie na prácu so súbormi:

- **ASSIGNFILE(f,'cesta\subor.txt')** - premenná f typu textfile bude označovať textový súbor „subor.txt“, pričom „cesta“ je relatívne, resp. absolútne umiestnenie súboru na disku.
- **EOF(f)** - funkcia vracia hodnotu true, ak je koniec súboru f.
- **EOLN(f)** - funkcia vracia hodnotu true, ak je koniec riadku v súbore f.
- **RESET(f)** - umožní čítať zo súboru, pričom nastaví čítaciu hlavu v súbore f (ak existuje) na začiatok, ak neexistuje, program vyhlási chybu.
- **REWRITE(f)** - umožní zapisovať do súboru f a nastaví zapisovaciu hlavu v súbore f na začiatok. V prípade existujúceho súboru súbor prepíše - zmaže sa jeho obsah.
- **READ(f,x)** - prečíta zo súboru f do premennej x a čítaciu hlavu posunie za posledný znak, ktorý prečíta.
- **READLN(f,x)** - prečíta zo súboru f do premennej x a čítaciu hlavu posunie na začiatok ďalšieho riadku v súbore f.
- **READLN(f)** - posunie čítaciu hlavu na začiatok ďalšieho riadku v súbore f, ale nič neprečíta.
- **WRITE(f,x)** - zapíše do súboru f hodnotu premennej x a zapisovaciu hlavu posunie za posledný znak, ktorý zapíše.
- **WRITELN(f,x)** - zapíše do súboru f hodnotu premennej x a zapisovaciu hlavu posunie na začiatok ďalšieho riadku v súbore f.
- **WRITE(f)** - = zapíše do súboru f prázdny riadok a zapisovaciu hlavu posunie na začiatok ďalšieho riadku v súbore f.
- **CLOSEFILE** - uzatvorenie textového súboru f.

11.2 Komponent „memo“, čítanie a zápis

Pre výpis výsledkov programu sme doteraz využívali komponent „label“, ale vhodný je aj komponent **Memo** z palety **Standard**. Do okna komponentu môžeme zapisovať pomocou vlastnosti **Lines**, napr.:

```
Memo1.Lines.Text:='Ahoj!';
```

Do okna môžeme vypisovať len reťazce znakov, pre premenné iného typu musíme použiť konverzné funkcie (IntToStr, FloatToStr, ...). Vymazanie obsahu memo okienka umožní príkaz **Clear**:

```
Memo1.Clear;
```

Do okna komponentu Memo môžeme **načítať text z textového súboru** príkazom:

```
Memo1.Lines.LoadFromFile('{adresa textového súboru}');
```

Pre **uloženie textu do textového súboru** z komponentu Memo zadáme príkaz:

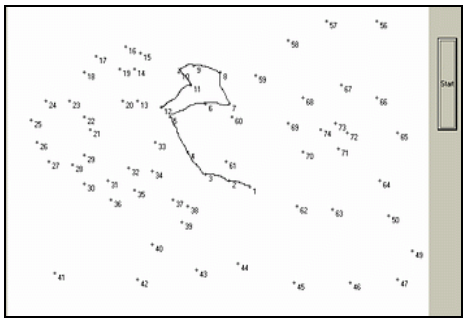
```
Memo1.Lines.SaveToFile('{adresa textového súboru}');
```

11.3 Príklad 1 - Memo a dátum

Vytvorte projekt, ktorý do memo okna po stlačení tlačidla vypíše aktuálny dátum a čas, po istom čase sa obsah okna zmaže. Nájdite význam funkcií v Delphi „helpe“ - *Now, Date, DateTimeToStr, DateToStr, Time, TimeToStr, FormatDateTime*, použite niektoré z nich v úlohe. Pri mazaní obsahu okna Memo použite **Timer** a jeho vlastnosť **Enabled**.

11.4 Príklad 2 - Čítanie zo súboru

Vytvoríme projekt na spájanie bodov podľa predlohy, aký poznáte z detských časopisov Vytvorte jednoduchý textový súbor **kresli.txt,** ktorý obsahuje riadky s troma číslami - poradové číslo, x-ová a y-ová súradnica bodu na grafickej ploche (oddelené sú jednou medzerou). Vytvorte projekt s veľkou grafickou plochou, vložte tlačidlo, ktorým súbor otvoríte a na miestach so súradnicami z textového súboru vykreslite malý krúžok a vypíšte poradové číslo bodu. Do projektu doplňte udalosť **OnMouseMove** pre voľné kreslenie myšou do grafickej plochy.



11.5 Príklad 3 - Štatistika súboru

a) Vytvorte projekt, ktorý otvorí ľubovoľný textový súbor a zistí v ňom počet medzier, ten vypíše v memo okne.
b) Projekt doplňte tak, aby ste urobili úplnú znakovú štatistiku súboru - zistite počty jednotlivých znakov v súbore. Využite ASCII kódy znakov, nastavte vlastnosť **ScrollBars** v **Memo** okne.

11.6 Príklad 4 - Zápis do súboru

Urobte program, ktorý vám umožní na stlačenie tlačidla zapisovať do zvoleného textového súboru text, ktorý sa nachádza v komponente **Edit**. Využite metódu na pripojenie textového reťazca z editovacieho okna do ďalšieho riadku Memo okna:

```
Memo1.Lines.Add(Edit1.Text);
```

11.7 Úlohy na riešenie

1. Vytvorte svoj **vlastný textový súbor s obrázkom**, pozmeňte kód príkladu 2 a presvedčite sa o jeho správnom tvare. Projekt doplňte o farebné vykresľovanie myšou. Upravte program tak, aby stačilo klikať len na vrcholy bodov a nebolo treba čiary kresliť myšou (vznikne obrázok tvorený úsečkami).
2. Vytvorte projekt, ktorý otvorí **zdrojový kód programu v Pascale** a otestujte jeho správne „ozátvorkovanie“ t. j. - či počet ľavých zátvoriek je rovnaký ako počet pravých.
3. Vytvorte projekt, ktorý otvorí zdrojový kód programu v Pascale a vykoná **štatistiku vybraných príkazov** - kľúčových slov (begin, end, if, then, else, while, do, repeat, until).
4. Vytvorte projekt, ktorý prostredníctvom editovacích okien umožní **vkladať údaje o žiakoch** (meno, priezvisko, vek, výšku), údaje zapíše do súboru (využite typ record). Potom nastaví čítaciu hlavu na začiatok súboru a zistí počet žiakov, priemerný vek a výšku žiakov, nájde meno žiaka s maximálnou a minimálnou výškou. Výsledky programu sa zapíšu do memo okna.

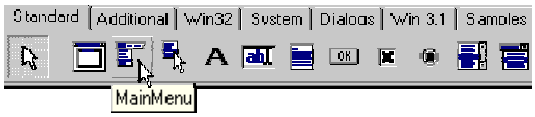
12 VLASTNÉ MENU

12.1 MainMenu

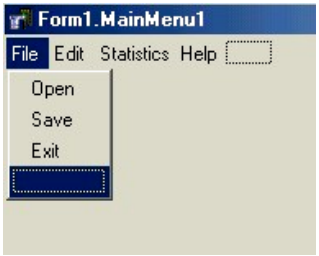
Aj v tejto lekcii budeme pracovať so súbormi, ale trochu inak. V okne aplikácie si vytvoríme vysúvacie menu - **MainManu**, akú poznáme z Windows aplikácií. Naše menu bude mať len niekoľko položiek, napríklad takýchto:

- File - Open, Save, Exit
- Edit - Clear
- Statistic - Count Lines, Count spaces, Count words,...

Kedykoľvek môžeme zmeniť názvy položiek, doplniť či zmazať položku. Ako teda vlastné menu vytvoríme?



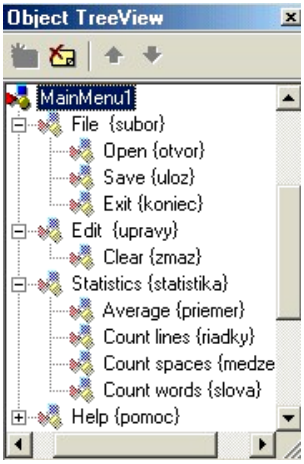
Do formulára vložíme komponent **MainMenu** z palety **Standard**. Dvojklik naň umožňuje vkladať názvy jednotlivých položiek menu aj názvy položiek v podmenu – vysunutá roletka s ponukou. Pri tvorbe menu sa kurzorovými klávesmi posúvame doprava, resp. nadol.



V strome objektov **Object TreeView** sa ako podstrom v objekte MainMenu1 zobrazia názvy položiek menu (Caption) napr. „Open“ a ich mená v zátvorke (Name) napr. „otvor“, prostredníctvom ktorých sa odvolávame v názvoch procedúr (napr. TForm1.otvor.Click).

Strom objektov prehľadne znázorňuje jednotlivé položky menu.

Kliknutím na ľubovoľnú položku v podmenu sa v editovacom okne vloží hlavička procedúry pre ošetrenie položky. Napr. ak klikneme na položku „Exit“, chceme, aby sa aplikácia ukončila. Do tela procedúry vpíšeme príkazy pre ukončenie programu **Alication.Terminate;**




```
procedure TForm1.koniecClick(Sender:TObject);
begin
  Alication.Terminate;
end;
```

12.2 OpenDialog, SaveDialog

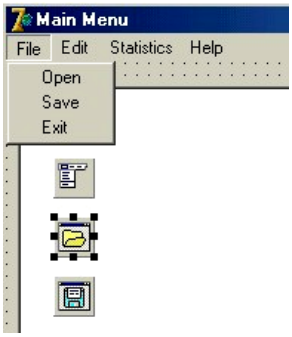
Nato, aby sme mohli realizovať „klasické“ otvorenie, resp. uloženie súboru pomocou dialógového okna, musíme vložiť neviditeľné komponenty **OpenDialog** a **SaveDialog** z palety **Dialogs**. Pre oba komponenty

nastavíme v Properties v položke **Options** položku **ofFileMustExists** na **True**. Tieto komponenty sa budú aktivizovať len pri volaní dialógového okna.

Nadefinujeme procedúry na otváranie a ukladanie súboru (napr. textového). Na formulári v komponente **MainMenu** klikneme na položku **Open** a do tela procedúry **TForm1.otvorClick** dopíšeme potrebné riadky:

Otvorenie súboru

```
procedure TForm1.otvorClick(Sender:
TObject);
var s:string;
      f:textfile;
begin
  GetDir(0,s);
  OpenDialog1.InitialDir := s;
  if not OpenDialog1.Execute then exit;
  AssignFile(f,OpenDialog1.Filename);
  reset(f);
     ...
  closefile(f);
end;
```



Metóda **GetDir** vloží do premennej **s** cestu aktuálneho adresára potrebného pre otvorenie dialógového okna. Ak užívateľ potvrdí v dialógovom okne vybraný súbor, jeho meno sa priradí do premennej **f**. Obsah súboru môžeme zobraziť napr. v **Memo** okne, prípadne v súbore vykonáme potrebnú štatistiku...

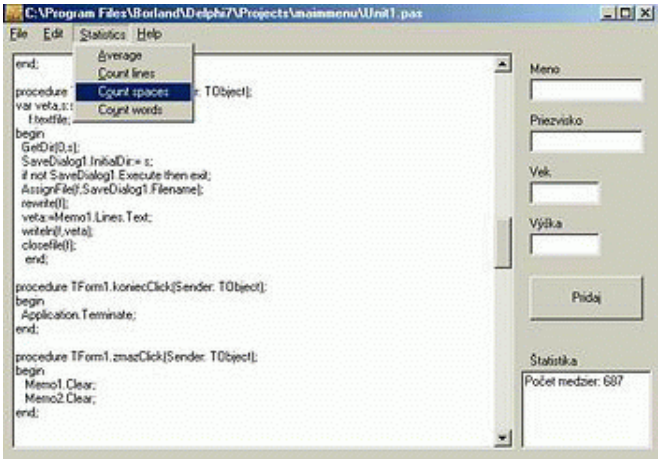
Rovnako postupujeme pri položke **Save** v **MainMenu** ak chceme nejaký (napr. textový) súbor uložiť:

Uloženie súboru

```
procedure TForm1.ulozClick(Sender: TObject);
var s:string;
    f:textfile;
begin
  GetDir(0,s);
  SaveDialog1.InitialDir:= s;
  if not SaveDialog1.Execute then exit;
  AssignFile(f,SaveDialog1.Filename);
  rewrite(f);
             ...
  closefile(f);
end;
```

Príklad
Doplňte procedúry pre ostatné položky menu, na zmazanie Memo okien, vložte štatistiky textového súboru, aké sú v úlohách v predchádzajúcej lekcie. Do projektu môžeme vložiť aj editovacie okná na realizovanie vstupov, tlačidlo na pridávanie údajov z editovacích okien do okna Memo a pod.

Výsledok môže mať napríklad aj nasledujúci formát:



12.3 Úlohy na riešenie

1. Doplňte do príkladu v položke „Edit“ možnosť **nahrádzania textu** iným textom v súbore.
2. Vytvorte **menu na farebné efekty s načítanými obrázkami** z odpovedajúcej lekcie, pričom do objektu Image sa otvorí ľubovoľný bitmapový obrázok, zmenený obrázok istým vybraným algoritmom z menu budeme môcť uložiť.

13 TRIEDA, INŠTANCIA

13.1 Trieda, objekt, inštancia

Doposiaľ sme definovali premenné - objekty štandardného typu TButton, TColor a podobne. Všetky spomínané objekty obsahoval formulár **TForm1 - objekt=inštancia triedy TForm**, ktorý predstavuje hlavné okno aplikácie.

Teraz si konečne vytvoríme vlastnú triedu - istý typ objektu spolu s vlastnosťami, ktoré budeme potrebovať. Vytvoríme triedu **TKruh** a jej objekty - kruhy, resp. kružnice. Všetky kružnice majú niečo spoločné - samotnú definíciu - súradnice stredu a polomer, farbu. Ale konkrétne kružnice sa líšia práve v týchto vlastnostiach, môžu sa pohybovať iným smerom a pod.

Príklad deklarácie formulára:

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Image1: TImage;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Button1lick(Sender: TObject);
    procedure timer1timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

13.2 Deklarácia triedy

Na vytvorenie vlastného typu zadefinujeme premenné objektu, metódu konštruktora (Create) a metódy (procedúry) pre zmenu vlastností objektu.

Napr. na deklaráciu triedy (class) TKruh sme použili premenné pre súradnice stredu (x,y), polomer (r), farbu kružnice (f) a viditeľnosť kružnice na grafickej ploche (pozri). Procedúra „Ukaz“ vykreslí kružnicu, „Skry“ ju prekreslí farbou grafickej plochy, procedúra „Zmenfarbu“ nastaví farbu kružnice a procedúra „Posun“ ju posúva v istom smere napr. pomocou Timera.

```
        TKruh= class
    x, y, r: Integer;
    f: TColor;
    vid: boolean;
    constructor Create(xx, yy, rr: integer);
    procedure Ukaz;
    procedure Skry;
```

```
  procedure ZmenFarbu(ff: TColor);
  procedure Posun(dx, dy: integer);
   end;
```

13.3 Konštruktor, inštancia

Ak chceme definovať objekt nového typu TKruh, musíme najprv vytvoriť tzv. **konštruktor objektu**. Ďalej vlastnosti objektu prostredníctvom deklarovaných metód - procedúr.

Dopíšte telá procedúr na vykreslenie, či „schovanie“ kružnice, nastavenie farby f a posun kružnice o dx a dy.

Deklarujte tri globálne **premenné typu TKruh - objekty**, ktoré sa tiež nazývajú **inštanciami triedy TKruh**:

```
var k1, k2, k3: TKruh;

constructor Tkruh.Create(xx, yy, rr: integer);
  begin
    x:=xx; y:=yy; r:=rr;
    vid:=false; f:=clBlack;
  end;
procedure Tkruh.Ukaz;
                    ...
procedure Tkruh.Skry;
                    ...
procedure Tkruh.ZmenFarbu(ff: TColor);
                    ...
procedure Tkruh.Posun(dx, dy: integer);
                ...
```

13.4 Vytvorenie objektu

Rezervované slovo **nil** u dynamických premenných v Turbo Pascale znamená, že dynamická premenná nikde neukazuje (chýbajúca referencia).

Priradenie **k1:=nil;** podobne znamená „žiadny objekt“ (ešte nie je vytvorený). Toto priradenie vložíme medzi inicializácie do tela metódy **FormCreate** (už sme tam dávali skratku na Canvas plochy Image), ktorá ich nastaví pri vytváraní formulára hneď po spustení.

Objekt sa vytvorí až metódou **Create**, pričom zmenu vlastností objektu uskutočníme príslušnou metódou triedy známou **bodkovou notáciou** napr.

- pre zmenu farby kružnice: **k1.ZmenFarbu();**
- pre zobrazenie kružnice: **k1.Ukaz;**

- pre posun kružnice: **k1.Posun();** a pod.

  Pri zmene viacerých vlastností objektu môžeme pre skrátený zápis použiť príkaz **with**, ako to vidieť z nasledujúceho kódu procedúry:

```
procedure TForm1.Button1click(Sender: TObject);
begin
  k1:=TKruh.Create(100,100,50);
  with k1 do
    begin ZmenFarbu(clGreen); Ukaz;
    end;
end;
```

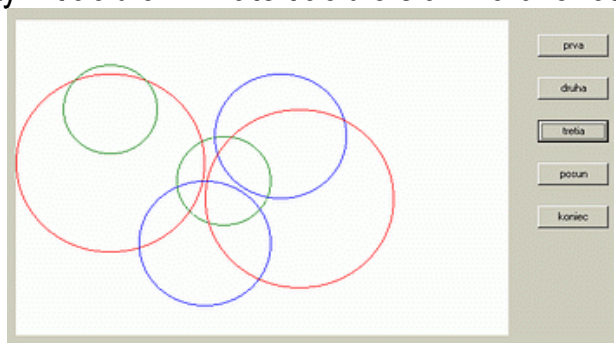
13.5 Uvoľnenie objektu

Na uvoľnenie prostriedkov, ktoré objekt v pamäti zaberal, sa používa metóda **Free**. Pri ukončovaní aplikácie je preto nutné túto metódu použiť. (Pozn. Metódy **Free** a **Destroy** nie je nutné použiť napr. pri objekte TForm, kde sa deštrukcia objektu robí automaticky.)

```
...
if k1<>nil then
  with k1 do
    begin
      if vid then Skry; Free; k1:=nil;
    end;
...
```

13.6 Výsledok projektu

Do okna formulára sme vložili tri tlačidlá pre vytvorenie troch rôznych transparentných kruhov, z ktorých má každý svoju veľkosť, farbu aj smer pohybu. Ten aktivujeme štvrtým tlačidlom. Piate tlačidlo slúži na ukončenie projektu.



13.7 Úlohy na riešenie

1. Vytvorte projekt na vytváranie a pohyb rozličných kruhov rôznych farieb a veľkostí podľa vyššie popísaného postupu. Využite uvedené zdrojové kódy.

14 ĎALŠIE ZADANIA ÚLOH

1. Vytvorte projekt, ktorý bude zobrazovať na grafickej ploche pohyb guličky, ktorá sa bude vo vhodných časových intervaloch pohybovať po „sinusoide“. (Guličku nahraďte obrázkom chrobáka, prípadne dokreslite pozadie a načítajte bitmapový obrázok do grafickej plochy.)
2. Vytvorte analógiu známej hry 15 - na poli 4x4 políčok po stlačení tlačidla sa náhodne rozmiestni 15 obrázkov (s číslicami od 1 do 15, resp. húsky obrázka jedného celku), jedno políčko ostáva voľné. Pomocou neho užívateľ posúva obrázky tak, aby ich usporiadal v správnom tvare.
3. Vytvorte jednoduchý textový editor, ktorý bude oproti vzorovému programu zo stránky obsahovať aj nástroj guma – bude možné meniť tvar gumy (kruh, štvorec) ako aj jej veľkosť.
4. V textovom súbore je na prvom riadku informácia o počte bodov a na ďalších riadkoch sú dané súradnice (x, y) jednotlivých bodov. Vyznačte tieto body na grafickej ploche a nájdite (vykreslite ju)najmenšiu kružnicu, ktorá obsahuje všetky body vo svojom vnútri, alebo na obvode.
5. Vytvorte projekt, ktorý vykreslí jednoduché hodiny spolu s hodinovou, minútovou a sekundovou ručičkou, ktoré ukazujú aktuálny čas. (Sekundová ručička sa posúva každú sekundu). Využite možnosti niektorých funkcií (pozri „help“ Delphi) - *Now, Date, DateTimeToStr, DateToStr, Time, TimeToStr, FormatDateTime*, použite niektoré z nich v úlohe.
6. Vytvorte projekt, ktorý bude náhodne vyberať slovenské slová v základnom tvare zo slovníka (textový súbor s aspoň 100 položkami) a vhodným spôsobom vyžiada jeho preklad do anglického jazyka. V projekte bude realizovaná kontrola správnosti slov ako a ak štatistické vyhodnotenie počtu správnych a nesprávnych odpovedí. Po ukončení vypíše percentuálnu úspešnosť užívateľa.
7. Vytvorte projekt, ktorý na základe šifry z textového súboru zašifruje vstupný text. Použite memo komponenty. (Šifra - tabuľka s jasným priradením znakov všetkým znakom).
8. Vytvorte projekt, ktorý vykreslí polynomickú funkciu 3. stupňa na základe vstupných koeficientov A0, A1, A2, A3 a pre vstupné celé hodnoty a, b vypočíta veľkosť plochy ohraničenej intervalom <a,b>, grafom funkcie a x-ovou osou.
9. Vytvorte obdobu známej hry LOGIK, ktorej cieľom je uhádnuť správne zoradenie 4 farieb zo 6. Program vygeneruje náhodnú štvoricu farieb (farby sa neopakujú) v istom poradí. Hráč háda farby a ich umiestnenie. Program vypíše počet uhádnutých farieb a počet správnych pozícií. Po uhádnutí program vyhodnotí počet pokusov. Počet hádaní môžete obmedziť, pričom pri neúspešných hrách vypíšete správne poradie farieb.

Použité zdroje

http://www.gphmi.sk/machova/delphi
http://www.input.delphi.sk
http://www.edi.fmph.uniba.sk/tomcsanyiova/4Delphi/
http://www.kurz_delphi.szm.sk

Názov	:	Začíname s DELPHI
Autor	:	RNDr. Jana Machová
Recenzenti	:	Ing. Kamil Hnath Ing. Drahoš Knapík
Jazyková úprava	:	PhDr. Zora Mihoková
Vydavateľ	:	Metodicko-pedagogické centrum v Prešove
Za vydanie zodpovedá	:	PaedDr. Ivan Pavlov, PhD. riaditeľ MPC
Obal a väzba	:	Rokus s. r. o.
Náklad	:	100 ks
Rok vydania	:	2004

1. vydania

ISBN 80-8045-353-5