



Moderné vzdelávanie pre vedomostnú spoločnosť / Projekt je spolufinancovaný zo zdrojov EÚ

ZBIERKA

RIEŠENÝCH A NERIEŠENÝCH

ÚLOH V DELPHI

I. DIEL

Materiál vznikol

na **Gymnáziu, Párovská 1, Nitra**

v rámci projektu **Implementácia kľúčových kompetencií v školskom vzdelávacom programe a modernizácia vyučovania prírodovedných predmetov**

aktivity **2.2 Inovácia vzdelávacieho programu voliteľných hodín predmetu informatika**

s cieľom aktivity **Zmodernizovať vyučovanie informatiky, pripraviť a zrealizovať školský vzdelávací program gymnázia pre vyučovanie voliteľných hodín predmetu informatika**





v období **od augusta 2009 do júla 2011**

autor **Jozef Piroško**

Prvý diel zbierky je zameraný na precvičenie algoritmizácie a programovania v Pascale v prostredí Delphi v rámci základného kurzu (povinný predmet informatika na stredných školách). Zbierka nemá v úmysle nahradiť výklad vyučujúceho, veď ani nie je učebnicou. Jej cieľom tiež nie je podať úplný prehľad o preberanej problematike. Vynechali sme napríklad viacnásobné vetvenie, príkaz case. Neuviedli sme tiež všetky možnosti príkazu for, medzery sú aj v možnostiach prebraných údajových typov. Napriek tomu si myslíme, že je prínosom, najmä čo sa týka množstva riešených úloh na jednotlivé algoritmické konštrukcie a im zodpovedajúce príkazy. Algoritmizácia a najmä programovanie sa nedá naučiť bez vytvorenia, odladenia a testovania množstva programov. V zbierke máte príklady rôznej náročnosti, návody aj hotové riešenia. Snažili sme sa o systematický výklad nevyhnutnej teórie.

Druhý diel, ktorý je určený pre študentov voliteľného predmetu informatika systematizuje poznatky z prvého dielu a precvičuje nové poznatky potrebné k maturitnej skúške z informatiky z časti programovanie.

Použitá symbolika:

-  študijný text, dobré by bolo si ho osvojiť
-  práca s počítačom, vyskúšať na počítači
-  problém na riešenie, stačí hlava, pero a papier
-  do pozornosti, upozornenie, zaujímavosť

Úvodné príklady

Vytvorte program, ktorý vás po spustení pozdraví slovom Ahoj!

Do formulára vložíme komponent Label1 dvojklikom na **A** v palete Standard. Ak hneď začneme písať slovo Ahoj!, automaticky prepíšeme vo vlastnosti Caption text Label1 na Ahoj! Po stlačení klávesu F9 sa spustí program a v strede formulára sa zobrazí text Ahoj! Beh programu ukončíme zavretím okna formulára - kliknutím na „krížik“ ZAVRIEŤ.

Text je nevýrazný a preto ho zväčšíme a zmeníme aj jeho farbu. Najprv musíme Delphi povedať, že chceme pracovať s komponentom Label1 a preto do neho klikneme (1x) - komponent je vybraný, ak vidieť jeho úchyty („čierne body“ v rohoch a stredoch strán). Zväčšenie plochy komponentu Label1 však neznamená zväčšenie písma v Label1! Zväčšiť písmo znamená v ľavom stĺpci, v stĺpci Object Inspector v záložke Properties kliknúť na vlastnosť Font a následne na tlačidlo (tri bodky), čím vyvoláme panel Písmo a môžeme meniť písmo, jeho rez, veľkosť aj farbu. Po kliknutí na OK sme dosiahli žiadaného výsledku. Po spustení programu - stlačení F9, zodpovedá pozdrav Ahoj! našim predstavám. Ak nie, nič nám nebráni, po ukončení behu programu, t.j. po kliknutí na „krížik“ ZAVRIEŤ v pravom hornom rohu formulára, cez vlastnosť Font komponentu Label1 spraviť ďalšie úpravy.



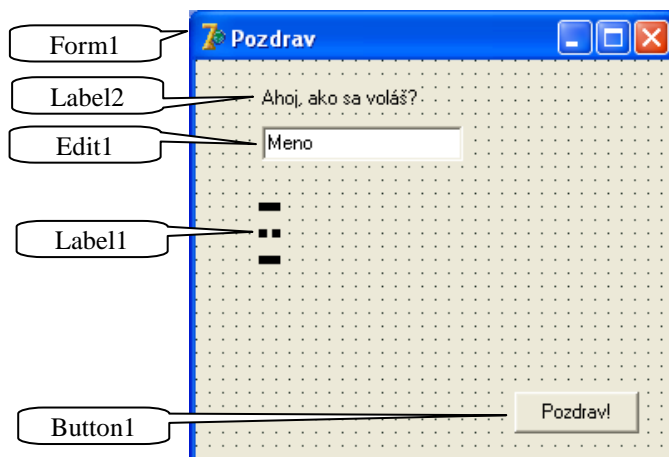
Titulok formulára sme zmenili z Form1 na Pozdrav. Pokúste sa aj vo vašom formulári o takúto zmenu.

Pred spustením programu je potrebné ho uložiť! Najjednoduchšie kliknutím na nástroj Save All (Shift+Ctrl+S) alebo File - Save All. Každý projekt (program) ukladajte do samostatného priečinka! Nový priečinok vytvoríte v okne Save Unit1 As kliknutím na nástroj Vytvorí nový priečinok. Následne kliknite na Otvoriť.

Nemeňte názov Unit1, len potvrdíte Uložiť! Uloženie projektu bude úspešné, ak vám program do novovytvoreného priečinka ponúkne uložiť aj Project1. Znova nemeňte názov, len potvrdíte Uložiť.

Program doplňte tak, aby sa opýtal na vaše meno a oslovil vás napríklad: Ahoj Peter!

Formulár treba doplniť o komponent, ktorý umožní doviest' do programu „zvonka“ naše meno. Na vstup údajov do programu slúži komponent Edit. V jeho vlastnosti Text sme prepísali Text na Meno. Program nás nemôže pozdraviť hneď po spustení, pretože sme ešte nestihli vpísať naše meno. Signálom, že nás môže program pozdraviť, bude kliknutie na tlačidlo Button1 s textom Pozdrav! v jeho vlastnosti Caption. Čiže napr. Ahoj Peter sa má vypísať až po udalosti - po kliknutí užívateľa na tlačidlo Button1. Príkazy, ktoré sa majú vykonať po nastaní nejakej udalosti (napr. po kliknutí na Pozdrav!), musí programátor napísať do procedúry (tehlička programu) prislúchajúcej k vzniknutej udalosti. S komponentom môže byť zviazaných viacej udalostí. Udalosti prislúchajúce ku komponentu si môžeme pozrieť v Object Inspectore v záložke



Events (udalosti). Po dvojkliku do poľa s názvom OnClik, pre komponent Button1, sa vloží na príslušné miesto:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
end;
```

čo možno voľne interpretovať: medzi begin (začiatok) a end (koniec) napíš príkazy, ktoré sa majú vykonať po kliknutí na tlačidlo Button1 vo formulári Form1. Nám tam stačí dopísať jediný príkaz:

```
Label1.Caption:= 'Ahoj ' + Edit1.Text;
```

ktorý sa vykoná sprava doľava (príkaz priradenia!) t.j. spoj text Ahoj s textom, ktorý nájdesh v Edit1.Text a výsledok zobraz pomocou komponentu Label1 a jeho vlastnosti Caption.

Vytvorte program, ktorý vypočíta priemernú teplotu dňa.

Vieme, že priemerná teplota dňa sa vypočíta ako súčet teplôt nameraných o 6-tej ráno, o 12-tej na obed, o 18-tej večer, a, keďže sa nechce meteorológom stávať o polnoci, večernú teplotu zarátajú dvakrát, a súčet vydedia štyrmi.

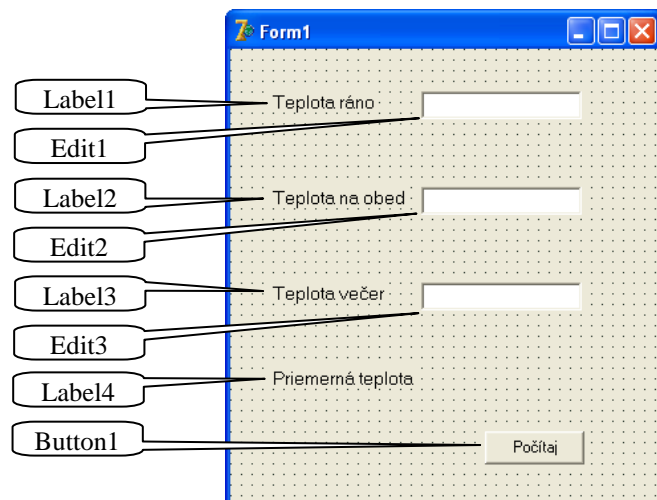
Analýza

$$t_{\text{priemerná}} = (t_1 + t_2 + 2t_3)/4$$

Na vstupe musia byť tri reálne čísla - namerané teploty t_1 , t_2 a t_3 .

Na výstupe sa má zobraziť, podľa vzorca vypočítaná, priemerná teplota.

Do formulára Form1 vložíme komponenty podľa obrázka (klikneme na komponent v palete a následne klikneme do formulára). Vlastnosti Caption v Label1 až Label4 prepíšeme textami Teplota ráno, Teplota na obed, Teplota večer a Priemerná teplota. Ak označíme ťahaním kurzora myši pri stlačení ľavom tlačidle myši do pravouhlej oblasti komponenty Edit1 až Edit3, môžeme naraz v ich vlastnosti Text vymazať text Edit1. Po kliknutí (1x) na tlačidlo Button1 môžeme v jeho vlastnosti Caption prepísať Button1 na Počítaj (nie vo vlastnosti Name!). Po vytvorení vizuálneho návrhu môžeme „začať“ programovať.



Chceme, aby po kliknutí na tlačidlo Počítaj, program „zobral“ z Edit1, Edit2 a Edit3 čísla - namerané teploty, podľa vzorca vypočítal priemernú teplotu a výsledok zobrazil cez komponent Label4. Po dvojkliku na tlačidlo Button1 (Počítaj) sa do unitu (tam, kde máme možnosť písať príkazy) vloží procedure TForm1.Button1Click(...), čo nám umožňuje medzi jej begin a end napísať príkazy, ktoré chceme, aby sa vykonali, keď klikneme na Button1 - Počítaj pri spustení programu. Na výpočet potrebujeme štyri premenné, ktoré označíme t_1 , t_2 , t_3 a t . Počítaču musíme povedať, že sú to premenné a čo do nich chceme uložiť. Preto medzi procedure TForm1.Button1Click(Sender: TObject); a begin „vyrobíme“ prázdny riadok (stlačíme kláves ENTER s kurzorom umiestneným napríklad pred begin, ak sa prázdny riadok „nevyrobil“, máte zapnuté prepisovanie, prepnete na vkladač mód stlačením klávesu INSERT a postup zopakujte). Vpíšeme var $t_1, t_2, t_3, t : \text{real}$; čím sme počítaču povedali, že sú to premenné (variables) a budú v nich uložené reálne čísla.

Teraz medzi begin a end môžeme napísať potrebné príkazy. Čo znamená napríklad := alebo StrToFloat sa dozvieme v ďalších kapitolách.

```
procedure TForm1.Button1Click(Sender: TObject);
var t1, t2, t3, t : real;           //poznámka: deklarácia - zavedenie premenných a ich typov
begin
t1 := StrToFloat(Edit1.Text);     //premena textu z Edit1 na reálne číslo a jeho uloženie do t1
t2 := StrToFloat(Edit2.Text);     //premena textu z Edit2 na reálne číslo a jeho uloženie do t2
t3 := StrToFloat(Edit3.Text);     //premena textu z Edit3 na reálne číslo a jeho uloženie do t3
t := (t1 + t2 + 2*t3) / 4;        //výpočet podľa vzorca a uloženie výsledku do premennej t
Label4.Caption := 'Výsledná teplota = ' + FloatToStr(t);           //zobrazenie výsledku
end;                               //koniec procedúry

end.                               // pod koncom procedúry musí byť ešte end s bodkou (koniec unitu), nezmažte ho!
```

☛ Program spustíme stlačením F9. Ak sa niektorý z riadkov v unite podfarbí, v danom riadku je chyba, skúste porovnať váš riadok s naším.

Poznámky:

SEKVENCIA

Príklad 1.1

Máme dve premenné označené napríklad X a Y. Napíšte také príkazy priradenia, ktoré vymenia hodnoty uložené v týchto dvoch premenných.

Analýza

Nech v premennej X je napríklad hodnota 5 a v premennej Y hodnota 7. Po vykonaní hľadaných príkazov priradenia má byť v X hodnota 7 a v Y hodnota 5.

Príkaz priradenia slúži na priradenie hodnoty premennej. Má tvar *premenná := výraz*. Vykonalie: vyhodnotí sa výraz na pravej strane príkazu priradenia a získaná hodnota sa priradí ako nová hodnota premennej na ľavej strane príkazu priradenia.

Napr.: $X := 7$ sa číta „X priradiť 7“ a znamená, že do premennej X bude uložená hodnota 7.

Predchádzajúca hodnota premennej X bude nenávratne prepísaná!

Pravdepodobne nám napadne napísať nasledujúcu sekvenciu (postupnosť) príkazov priradenia:

$X := Y;$

$Y := X;$

Príkazy sa od seba oddeľujú bodkočiarkou!

Na overenie správnosti uvedenej sekvencie si môžeme odsimulovať, čo sa bude diať v pamäti počítača počas vykonávania príkazov priradenia.

Po zavedení (deklarovaní) každej premennej sa v pamäti počítača vyhradí pamäťové miesto (priestor) ktoré znázorníme *Názov premennej*

Keďže v premennej X je na začiatku hodnota 5 a v premennej Y hodnota 7, môžeme nakresliť:

X Y

Počas vykonávania príkazu $X := Y$ sa zoberie hodnota premennej Y, t.j. 7 a uloží sa ako nová hodnota do X, preto po vykonaní prvého príkazu sú v premenných X a Y hodnoty

X Y

Počas vykonávania príkazu $Y := X$ sa zoberie hodnota premennej X, t.j. 7 (!) a uloží sa ako nová hodnota do Y, preto po vykonaní druhého príkazu sú v premenných X a Y hodnoty

X Y

Na prvý pohľad je zrejmé, že nedošlo k výmene hodnôt premenných X a Y.

Nájsť správne riešenie znamená uvedomiť si, kde nastala chyba. Už prvým príkazom, $X := Y$, strácame pôvodnú hodnotu X (prepíše ju hodnota Y), ktorú však ešte neskôr potrebujeme uložiť do Y. Preto hodnotu X musíme najskôr odložiť „bokom“, napríklad do pomocnej premennej POM. Hľadanou sekvenciou je

POM := X;
X := Y; (1)
Y := POM;

Odsimulovaním procesov „v pamäti počítača“ sa presvedčíme o správnosti uvedenej sekvencie.



Príklad 1.1 chceme zrealizovať na počítači. Program nám musí umožniť vložiť hodnoty do dvoch premenných a na príkaz Vymeň vymeniť hodnoty v týchto premenných.

Formulár môže vyzeráť napríklad takto - pozri obrázok. Texty Prvá hodnota a Druhá hodnota sú vložené cez komponenty Label vlastnosť Caption; polia, do ktorých sa vložia hodnoty, sú komponenty Edit, v ktorých vo vlastnosti Text sme zmazali text Edit1 resp. Edit2; tlačidlo Vymeň je komponent Button.

☹ Vlastnosť Name ani u jedného komponentu nemeňte!

Po dvojkliku na komponent Button1 (Vymeň) sa do programu (presnejšie unitu Unit1) vloží procedúra (podprogram) umožňujúca naprogramovať, čo sa má vykonať po kliknutí na tlačidlo Vymeň. Skôr, než medzi slová begin a end napíšeme príkazy (1), musíme počítaču povedať, koľko a aké veľké pamäťové miesta má vyhradiť v pamäti počítača. Robí sa to deklaráciou premenných. Nad begin dopíšeme var POM, X, Y: string; (nezabudnite na bodkočiarku!). Tým sme počítaču povedali, že obsahom premenných POM, X a Y budú reťazce znakov (aj čísla sú reťazce znakov, presnejšie číslic). Posledným krokom je zabezpečenie prepojenia medzi komponentom a príslušnou premennou, čo prakticky znamená, aby hodnota zapísaná do poľa Edit1 (prvá hodnota) sa dostala do premennej X a hodnota zapísaná do Edit2 (druhá hodnota) sa dostala do premennej Y. Opäť môžeme použiť príkaz priradenia, t.j. hodnotu z Edit1.Text priradiť do X a hodnotu z Edit2.Text priradiť do Y. Po výmene hodnôt musíme dať zobrazit' nové hodnoty v X a Y napríklad cez polia Edit1 a Edit2. Úplná časť programu (presnejšie unitu) aj s poznámkami:

```
procedure TForm1.Button1Click(Sender: TObject);
var POM,X,Y: string;
begin
X := Edit1.Text;      //získanie hodnoty X z Edit1
Y := Edit2.Text;      //získanie hodnoty Y z Edit2
POM := X;             //výmena hodnôt medzi X a Y (tri riadky)
X := Y;
Y := POM;
Edit1.Text := X;      //zobrazenie novej hodnoty X v poli Edit1
Edit2.Text := Y;      //zobrazenie novej hodnoty Y v poli Edit2
end;
end. // koniec unitu Unit1
```

📖 Pred spustením programu je potrebné ho uložiť! Najjednoduchšie kliknutím na nástroj  Save All (Shift+Ctrl+S) alebo File - Save All. Každý projekt (program) ukladajte do samostatného priečinka! Nový priečinok vytvoríte v okne Save Unit1 As kliknutím na nástroj  Vytvorí nový priečinok. Následne kliknite na Otvoriť.

☹ Nemeňte názov Unit1, len potvrdte Uložiť! Uloženie projektu bude úspešné, ak vám program do novovytvoreného priečinka ponúkne uložiť aj Project1. Znova nemeňte názov, len potvrdte Uložiť.

📖 Program spustíme stlačením klávesu F9. Prepínať medzi poliami a tlačidlom Vymeň môžeme stláčaním tabulátora. Po každom kliknutí na tlačidlo Vymeň dôjde k vykonaniu príkazov procedúry Button1Click, t.j. k výmene hodnôt.

☹ Ak budeme vymieňať len číselné hodnoty, nepotrebujeme pomocnú premennú POM!

✍ Napíšte sekvenciu príkazov priradenia, ktorá vymení hodnoty číselných premenných bez použitia tretej - pomocnej premennej.

Algoritmus


Tu je jedna z možných sekvencií:

```
X := X - Y;
Y := Y + X;
X := Y - X;
```

Overte ju a vymyslíte ďalšiu.

📖 Realizácia vyššie uvedenej sekvencie na počítači si však vyžaduje väčšie úpravy v programe. Je to spôsobené tým, že komponenty vstupu a výstupu v Delphi sú schopné prijať a zobrazit' len texty

(znakové reťazce, string). Preto, ak chceme, aby program spracoval číselný reťazec z komponentu Edit ako číslo, musíme použiť konverznú funkciu. Pritom nám a aj počítaču musí byť zrejmé, či bude spracované celé alebo reálne číslo.

 Preto existujú údajové typy:

reťazec - **string**

obsahuje postupnosť znakov; hodnoty typu string sa uvádzajú v apostrofoch (ľavé Alt+39); prázdny reťazec sa zapisuje " (dva apostrofy vedľa seba bez medzery).

Nech platí deklarácia var Meno: string; potom môžeme spraviť napríklad priradenie Meno := 'Juro'; Znamienkom + možno spájať reťazce, napr. Meno := Meno + ' Jánošík'; Čo bude novým obsahom premennej Meno?

celé číslo - **integer**

premenná typu integer môže obsahovať celé číslo z intervalu $\langle -\text{MaxInt} - 1, \text{MaxInt} \rangle$.


Konštanta MaxInt je v Delphi $2^{31} - 1$, čo je 2 147 483 647.

Keď chceme do premennej typu integer uložiť reťazec napríklad z komponentu Edit, musíme použiť konverznú funkciu StrToInt (String na Integer). Nech platí deklarácia var X: integer; potom môžeme X priradiť hodnotu z Edit1 zápisom X := StrToInt(Edit1.Text); Platí aj opačne, ak chceme zobrazit' cez komponent Label alebo Edit celé číslo, musíme použiť „opačnú“ konverznú funkciu IntToStr (Integer na String). Napríklad Label1.Caption := 'Nová hodnota X: ' + IntToStr(X);

reálne číslo - **real**

premenná typu real môže obsahovať reálne číslo.

Na jeho konverziu sa používajú funkcie StrToFloat a FloatToStr. Nech premenná Strana je typu real (var Strana: real;), potom môžeme použiť zápisy Strana := StrToFloat(Edit1.Text); - zoberie reťazec z poľa Edit1, zmení ho na reálne číslo a uloží do premennej Strana, a zápis Label1.Caption := 'Obsah štvorca = ' + FloatToStr(Strana*Strana); - vypočíta súčin Strana*Strana, zmení ho na reťazec, pripojí za reťazec 'Obsah štvorca = ' a zobrazí cez komponent Label. Všimnite si, že všetko sa deje sprava doľava, keďže hovoríme o vykonaní príkazov priradenia.

 Teraz nás už neprekvapí nižšie uvedený tvar „programu“.

```
procedure TForm1.Button1Click(Sender: TObject);
var X,Y: integer;           //Deklarácia dvoch premenných X a Y typu integer
begin
X := StrToInt(Edit1.Text); //Vstup reťazca z Edit1, jeho konverzia na celé číslo a uloženie do X
Y := StrToInt(Edit2.Text); //Vstup reťazca z Edit2, jeho konverzia na celé číslo a uloženie do Y
X := X - Y;                //Výpočet
Y := Y + X;
X := Y - X;
Edit1.Text := IntToStr(X); //Konverzia hodnoty X na reťazec a jeho uloženie do poľa Edit1
Edit2.Text := IntToStr(Y); //Konverzia hodnoty Y na reťazec a jeho uloženie do poľa Edit2
end;
end.                        // Toto end s bodkou už prestaneme uvádzať, snád' naň nezabudnete ☺
```

☛ Pri písaní v Delphi využívajte ponuku „dokončovača písania“, keď po napísaní prvých písmen názvu komponentu alebo funkcie stačí stlačiť Ctrl+Medzerník a Delphi nám ponúkne zoznam slov, ktoré prichádzajú do úvahy. Ak je vyššie v programe chyba, ponuka sa nezobrazí, stlačte F9 a odstráňte chybu!

Príklad 1.2

Vytvorte program, ktorý „uhádne“ myslené celé číslo z intervalu $\langle 0,100 \rangle$ po zadaní zvyškov, ktoré vzniknú po delení mysleného čísla 3, 5 a 7. Na výpočet mysleného čísla použijete vzorec $\text{MysleneCislo} = (70 * \text{Zvysok3} + 21 * \text{Zvysok5} + 15 * \text{Zvysok7}) \bmod 105$. Napríklad, ak si myslíme číslo 15, do programu zadáme: Zvyšok po delení 3: 0, Zvyšok po delení 5: 0 a Zvyšok po delení 7: 1. Program musí vypísať: Myslel si si číslo 15.

Algoritmus


Každý „poriadny“ algoritmus (program) obsahuje vstup, výpočet a výstup. Úlohou vstupu je dostať do počítača údaje (vstupné údaje), s ktorými sa má uskutočniť výpočet. V časti výpočet sa „počíta“, rieši zadaný problém s vloženými vstupnými hodnotami. Slovo počíta sme dali do úvodzoviek, pretože nemusí ísť o výpočet, ale aj o vyhľadanie v skupine údajov, utriedenie a pod. Prvé dve činnosti (vstup a výpočet) robíme s cieľom získať novú informáciu a tú sa dozvieme cez výstup, t.j. zobrazenie získaných (výstupných) hodnôt.


Vstupom v úlohe 1.2 sú: zvyšok po delení mysleného čísla tromi, zvyšok po delení mysleného čísla piatimi a zvyšok po delení mysleného čísla siedmimi.

Výpočet je v tomto prípade veľmi jednoduchý, realizovaný pomocou jediného vzorca.

Výstupom je zobrazenie mysleného - výpočtom získaného, čísla.

Keď nám je zrejmý algoritmus - všeobecný postup pri riešení nášho problému, môžeme pristúpiť k realizácii algoritmu v konkrétnom programovacom jazyku.

 **Algoritmus** - postup, ako mechanicky vyriešiť zadaný problém. Algoritmus treba vymyslieť a vhodne zapísať alebo sa naučiť už niekým vymyslený postup. **Program** - algoritmus prepísaný pomocou štruktúr (príkazov) a komponentov konkrétného programovacieho jazyka do tvaru umožňujúceho realizáciu algoritmu na počítači. V našom prípade prepis do programovacieho jazyka Object Pascal v programovacom prostredí Delphi.

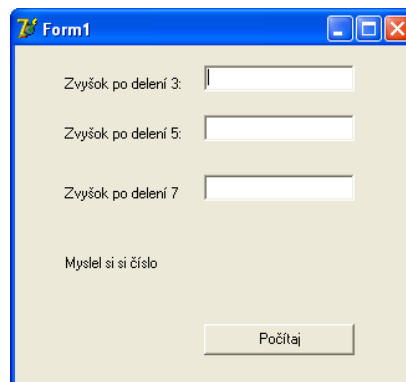
 Najprv sa sústreďme na to, ako sa dostanú vstupné údaje do počítača (až programu) a ako nám počítač oznámi výsledky. Na vstup a výstup údajov sa v moderných programovacích jazykoch používajú najmä komponenty Edit a Label (pozri prílohu 1). Vložíme ich do formulára „v zmysluplnom vizuálnom usporiadaní“. Po vložení komponentu Button - tlačidla, ktorým sa bude spúšťať výpočet, máme zrealizovanú vizuálnu stránku programu. Možný vzhľad formulára je na obrázku vpravo.

Po vpísaní premenných do deklaračnej časti - začína slovom var (variables - premenné) a uvedením ich typu (integer), dvojklikom na tlačidlo Button1 - Počítaj môžeme medzi slovami begin a end napísať príkazy, ktoré sa majú vykonať, keď dôjde ku kliknutiu na tlačidlo s nápisom (Caption) Počítaj.

Podstatná časť programu:

```

procedure TForm1.Button1Click(Sender: TObject); //vznikne po dvojkliku na Button1 vo formulári
var Zvysok3, Zvysok5, Zvysok7, MysleneCislo: integer; //deklarácia použitých premenných
begin
Zvysok3 := StrToInt (Edit1.Text); //vstup hodnôt potrebných k výpočtu
Zvysok5 := StrToInt (Edit2.Text);
Zvysok7 := StrToInt (Edit3.Text);
MysleneCislo := (70*Zvysok3 + 21*Zvysok5 + 15*Zvysok7) mod 105; //výpočet podľa vzorca
Label4.Caption := 'Myslel si si číslo ' + IntToStr (MysleneCislo); //vypísanie výsledku
end;
```



Príklad 1.3


Vytvorte program na výpočet obsahu a obvodu obdĺžnika.

Analýza

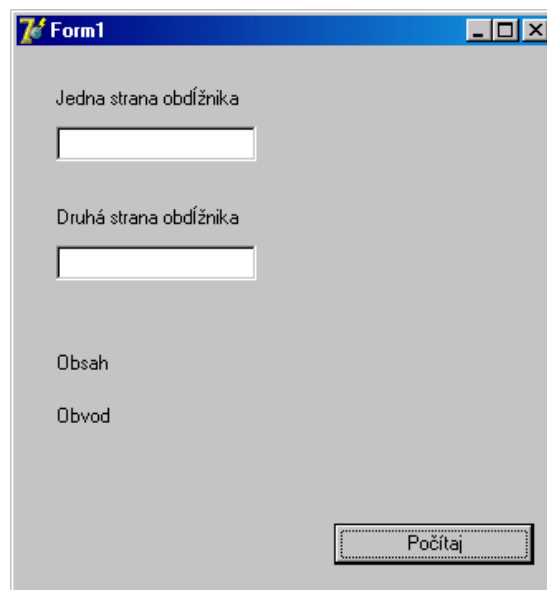
Z matematiky treba poznať vzorce na výpočet obsahu a obvodu obdĺžnika, Obsah = $a \cdot b$ a Obvod = $2 \cdot (a + b)$, kde a , b sú strany obdĺžnika.


Algoritmus


Z algoritmického hľadiska treba programu zadať rozmery obdĺžnika, program pomocou príkazov priradenia vypočíta obsah a obvod obdĺžnika a oznámi výsledok výpočtu.

 Predpokladáme, že strany obdĺžnika sú reálne čísla. Možné rozvrhnutie komponentov vo formulári je na obrázku (4 x Label, 2 x Edit, 1 x Button). Po zadaní strán obdĺžnika a kliknutí na tlačidlo Počítaj počítač uskutoční výpočet a oznámi výsledok.


```
procedure TForm1.Button1Click(Sender: TObject);
var StranaA, StranaB, Obsah, Obvod: real;
begin
StranaA := StrToFloat(Edit1.Text);
StranaB := StrToFloat(Edit2.Text);
Obsah := StranaA * StranaB;
Obvod := 2 * (StranaA + StranaB);
Label3.Caption := 'Obsah = ' + FloatToStr(Obsah);
//spojenie dvoch reťazcov, prvý je konštantou,
//druhý vznikne konverziou hodnoty premennej
Label4.Caption := 'Obvod = ' + FloatToStr(Obvod);
end;
```





 Ako môžeme pomenovať premennú? Označenie premennej musí začínať písmenom, ďalej sú dovolené len písmená alebo číslice (nie napríklad medzery, bodky). Dovoľené sú len písmená anglickej abecedy (nie slovenská diakritika) a medzi písmená je dodefinovaný podčiaričnik `_`. Odporúča sa pre meno premennej zvoliť výstižné označenie, ako napríklad Obsah, Obvod, StranaA, StranaB a nie S, O, a, b! Delphi nerozlišuje v označení premenných veľké a malé písmená, t.j. STRANAa, stranaA, StranaA, StRaNaA je vždy tá istá premenná.


 Vytvorte programy na výpočet obsahu a obvodu rôznych rovinných útvarov (štvorca, kruhu, trojuholníka,...).


 Delphi pozná konštantu π , na mieste, kde ju chceme použiť, stačí napísať PI alebo pi.

 Vytvorte programy na výpočet objemu a povrchu rôznych priestorových útvarov (kocky, kvádra, gule,...).

 Vytvorte program na výpočet hĺbky studne, ak poznáme čas v sekundách, za ktorý dopadne kameň do vody v studni. Návod: dráha voľného pádu - hĺbka studne h [m] = $\frac{1}{2}gt^2$, $g = 9.81\text{ms}^{-2}$.

 Keď budete zadávať reálne čísla cez vstup - komponent Edit, použite desatinnú čiarku. V programe však používajte desatinnú bodku, teda $g := 9.81$; alebo namiesto g rovno píšete jeho hodnotu.

 Vytvorte program na výpočet dráhy a rýchlosti rovnomerne zrýchleného pohybu po zadaní zrýchlenia a času pohybu. Počiatočná dráha a rýchlosť sú nulové. Návod: $s = \frac{1}{2} \cdot a \cdot t^2$, $v = a \cdot t$.

 Vytvorte program na výpočet výsledného odporu dvoch rezistorov po zadaní ich odporov R_1 a R_2 . Úlohu riešte pre sériové aj paralelné zapojenie rezistorov. Návod: $R_s = R_1 + R_2$, $R_p = R_1 \cdot R_2 / (R_1 + R_2)$.

Príklad 1.4

Vytvorte program simulujúci palubný počítač auta. Po zadaní prejdenej vzdialenosti v km a doby jazdy v hodinách oznámi priemernú rýchlosť. Predpokladajte, že bol pohyb rovnomerný.

Algoritmus

Na výpočet môžeme použiť vzorec $v = s/t$. Pred výpočtom musia byť zadané dráha a čas - vstup hodnôt cez komponenty Edit a po výpočte vypísanie výsledku cez komponent Label.

 Možný tvar formulára je na obrázku. Podstatná časť programu:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var Draha, Cas, PriemRych: real;
```

```
begin
```

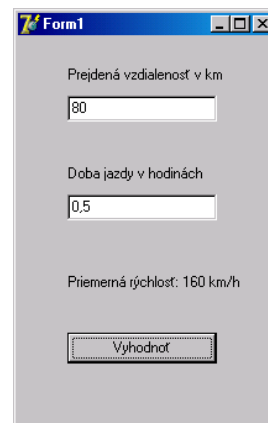
```
  Draha := StrToFloat(Edit1.Text);
```


```
  Cas := StrToFloat(Edit2.Text);
```


```
  PriemRych := Draha / Cas;
```

```
  Label3.Caption := 'Priemerná rýchlosť: ' + FloatToStr(PriemRych) + ' km/h';
```

```
end;
```



 Program palubný počítač doplňte o výpočet spotrebovaného benzínu po zadaní priemernej spotreby na 100 km. Do komponentu Edit pre priemernú spotrebu vložte počiatočnú hodnotu napr. 6,5 (litrov/100 km).

 Program palubný počítač doplňte o výpočet aktuálneho dosahu (vzdialenosť v km) a dĺžky jazdy v hodinách, po zadaní množstva benzínu v nádrži. Napr. pri 40 l benzínu v nádrži a priemernej spotrebe 6,5 l/100 km aktuálny dosah auta je 615 km ($40/6,5 \cdot 100$) a pri priemernej rýchlosti napr. 75 km/h to je 8,2 hod. ($615/75$).

Príklad 1.5

Vytvorte program na vypísanie doby splácania bezúročnej pôžičky po zadaní výšky pôžičky a výšky mesačnej splátky. Napríklad, ak si požičiame 1 000 € a mesačne budeme splácať po 50 €, bude nám splácanie trvať $1000/50 = 20$ mesiacov, čo je 1 rok a 8 mesiacov.

Algoritmus

Výška pôžičky aj mesačné splátky sú vždy celé čísla. Preto použijeme údajový typ integer. Súčasťou definície každého údajového typu je aj množina dovolených operácií. U typu integer je dovolené sčítanie (+), odčítanie (-), násobenie (*), ale nie delenie, pretože podielom dvoch celých čísel nemusí byť celé číslo. Dovoľené je však celočíselné delenie (div) a operácia zvyšok po celočíselnom delení (mod).

 Čo to ten div a mod je?

Začneme príkladmi: $15 \text{ div } 2 = 7$ a $15 \text{ mod } 2 = 1$, pretože $15 : 2 = 7$ a zvyšok 1, resp. $15 = 2 \cdot 7 + 1$; $7 \text{ div } 8 = 0$ a $7 \text{ mod } 8 = 7$, pretože $7 : 8 = 0$ a zvyšok 7, resp. $7 = 0 \cdot 8 + 7$; $5 \text{ div } 1 = 5$ a $5 \text{ mod } 1 = 0$, pretože $5 : 1 = 5$ a zvyšok 0, resp. $5 = 5 \cdot 1 + 0$;

div znamená celočíselný podiel a mod znamená zvyšok po celočíselnom delení, pričom pre všetky A, B kladné celé čísla platí: $A = (A \text{ div } B) \cdot B + (A \text{ mod } B)$, pričom $0 \leq (A \text{ mod } B) < B$ alebo ak $A \text{ div } B = D$ a $A \text{ mod } B = M$, potom $A = D \cdot B + M$ a $0 \leq M < B$.

☛ Funkcie div a mod sú „čarovné“. Pre náš príklad $1000 \text{ div } 50$ dostávame 20 (mesiacov) a pre $20 \text{ div } 12$ (mesiacov v roku) dostávame 1 (rok), pre $20 \text{ mod } 12$ dostávame 8 (mesiacov)!

Takže:

výška_pôžičky div mesačná_splátka = počet_mesiacov_splácania

počet_mesiacov_splácania div 12 = počet_rokov_splácania

počet_mesiacov_splácania mod 12 = počet_zvyšných_mesiacov

a ešte bonus:

výška_pôžičky mod mesačná_splátka = doplatok

Ak by sme totiž splácali mesačne napríklad po 30 €, predchádzajúce vzorce nám dajú výsledky 33 mesiacov, t.j. 2 roky a 9 mesiacov, teda by bolo splatených len 990 € (33×30) a ešte treba doplatiť 10 €. Doplatok sa dá vypočítať „cez bonus“ $1000 \text{ mod } 30 = 10$!

Netvrdíme, že to je na prvý pohľad ľahké, ale skúste div a mod venovať „dva pohľady“ ☺.

☞ Aký je doplatok pri mesačnej splátke 50 €?


☞ Možný tvar formulára vidíte na obrázku. Vedľa je podstatná časť programu.


```
procedure TForm1.Button1Click(Sender: TObject);
var VyskaPozicky, MesacnaSplatka, SplacatRokov, SplacatMesiacov,
    Doplatok: integer;
    PocetMesiacov: integer; //pomocná premenná
begin
    //priradenie vstupných hodnôt z komponentov Edit do premenných
    VyskaPozicky := StrToInt(Edit1.Text);
    MesacnaSplatka := StrToInt(Edit2.Text);
    //výpočet
    PocetMesiacov := VyskaPozicky div MesacnaSplatka;
    SplacatRokov := PocetMesiacov div 12;
    SplacatMesiacov := PocetMesiacov mod 12;
    Doplatok := VyskaPozicky mod MesacnaSplatka;
    //zobrazenie výsledkov pomocou komponentov Label
    Label4.Caption := '- počet rokov: ' + IntToStr(SplacatRokov);
    Label5.Caption := '- počet mesiacov: ' + IntToStr(SplacatMesiacov);
    Label6.Caption := '- doplatok: ' + IntToStr(Doplatok) + ' €';
end;
```

☛ Označenie premenných sa vám možno zdá nepraktické, robí však program čitateľnejším a po napísaní prvých písmen premennej môžeme opäť použiť klávesovú skratku CTRL+MEDZERNÍK na „dopísanie“ premennej.

☞ Výpočet vo vyššie uvedenom programe by sa dal zrealizovať aj bez pomocnej premennej, viete ako?

☞ Vytvorte program, ktorý čas v sekundách premení na hodiny, minúty a sekundy. Napríklad $3\,725 \text{ sek} = 1 \text{ hod}, 2 \text{ min a } 5 \text{ sek}$. Využite funkcie div a mod.

✍️  Vytvorte program, ktorý po zadaní sumy v korunách a hodnoty mince alebo bankovky určí, koľko mincí alebo bankoviek treba na vyplatenie zadanej sumy + doplatok. Napríklad suma: 127 €, hodnota mince: 5 €, na vyplatenie treba 25 mincí + doplatok 2 €. Využite funkcie div a mod.

✍️  Vytvorte program kalkulačka, ktorý po stlačení niektorého z tlačidiel +, -, * alebo / vypočíta a zobrazí súčet, rozdiel, súčin alebo podiel zadaných dvoch reálnych čísel.

SEKVENCIA V ŽIVOTE

Sekvenciu, ako postupnosť akcií, používame dennodenne, napríklad počas školského vyučovania nasledovne:

- akcia 1 ráno vstanem
- akcia 2 navštívim školu
- akcia 3 popoludňajšie aktivity (hudobná, výtvarná, jazykovka a pod.)
- akcia 4 príprava na vyučovanie (aspoň vymeniť zošity)
- akcia 5 večerná akcia

rozpis akcie 1

- akcia 1.1 skotúl'am sa z postele
- akcia 1.2 osprchujem sa
- akcia 1.3 naraňajkujem sa
- akcia 1.4 umyjem si zuby
- akcia 1.5 utekám na autobus do školy

rozpis akcie 1.3

- akcia 1.3.1 uvarím si čaj/kávu
- akcia 1.3.2 nájdem niečo pod zub
- akcia 1.3.3 zjem
- akcia 1.3.4 upracem po sebe (alebo uprosím niekoho iného)

Akákoli'vek naša činnosť sa v hrubých rysoch skladá z postupnosti akcií. Pri zjemňovaní, podrobnejšom opise, však len so sekvenciou nevystačíme, pretože napríklad kvalita večernej akcie môže byť podmienená finančnou sumou, ktorú máme k dispozícii a ak je nízka, musíme sa uskromniť alebo zohnať ďalšie financie. Takže, **ak** nemám dosť financií, **tak**... čo je už rozhodovanie, ako postupovať ďalej.

VETVENIE

Podmienený príkaz if

Ak v hociktorom z predchádzajúcich príkladov klikneme na tlačidlo spúšťajúce výpočet skôr, ako zadáme všetky vstupné hodnoty, program vypíše chybové hlásenie - približne: „Prázdny reťazec nie je platnou hodnotou“. Je to pochopiteľné, lebo program očakáva čísla - strany štvorca, prejdenu vzdialenosť, výšku pôžičky a pod. Ošetriť program na výskyt takejto chyby znamená uvedomiť si, že výpočet má prebehnúť, len ak sme zadali všetky vstupné hodnoty (ešte nezisťujeme, či z dovoleného intervalu!). Povedané algoritmickým jazykom:



ak VstupnaHodnota1 \neq " a zároveň VstupnaHodnota2 \neq " tak počítať. (*)


Pripomíname, že zápis " znamená prázdny reťazec. Prvým problémom môže byť symbol \neq , ktorý nenájdeme na klávesnici. Pri prepise do programu sa používa zápis $\langle \rangle$ (nie \gg). Konjunkcia „a zároveň“ sa zapisuje anglickým slovom and (negácia „nie je pravda, že...“ slovom not, disjunkcia „alebo“ slovom or). Otázkou zostáva ozátvorkovanie výrazu VstupnaHodnota1 $\langle \rangle$ " and VstupnaHodnota2 $\langle \rangle$ ", o čom rozhoduje priorita (poradie) operácií. Z matematiky vieme, že napríklad násobenie a delenie majú vyššiu prioritu ako sčítanie alebo odčítanie, t.j. $3+4*5 = 3+(4*5)$ a nie $(3+4)*5$. V programovaní:

 Priorita operácií:

1. negácia not, zmena znamienka
2. násobenie, delenie, div, mod, and
3. sčítanie, odčítanie, or
4. relačné operátory $<$, \leq , $=$, $\langle \rangle$, \geq , $>$

pri rovnosti operátorov sa výraz vyhodnocuje zľava doprava, zmeniť poradie vykonania operácií možno zátvorkami, pri použití viacerých zátvoriek sa výraz vyhodnocuje od vnútorných zátvoriek k vonkajším.

Keďže chceme, aby sa skôr vykonali porovnania $\langle \rangle$ ako and, v zápise musíme použiť zátvorky: (VstupnaHodnota1 $\langle \rangle$) and (VstupnaHodnota2 $\langle \rangle$), čím najprv získame dva výsledky typu boolean a na tie sa uplatní operátor and. Údajový typ boolean ešte nepoznáme, preto:

 logický typ - **boolean**

obsahuje hodnoty false (nepravda) a true (pravda).

Vyhodnotením výrazu typu boolean sa získa jedna z logických hodnôt false alebo true.

Dovolené operátory sú not - negácia, and - a zároveň, or - alebo.

Napríklad pre $X = 7$ výraz $X \bmod 2 = 0$ vracia false, keďže 7 je nepárne číslo a zvyšok po delení dvoma je zrejme 1.

Vráťme sa k nášmu problému. V riadku označenom (*) sme podčiarkli dve dôležité slová: ak ... tak ... Ako v bežnom živote, aj v programovaní slúžia na vyjadrenie **podmienenosti vykonania nejakej akcie**. „Ak nebude pršať, tak pôjdeme cez víkend na chatu.“

 **Vetvenie (neúplné, binárne)**

- použijeme, ak príkaz alebo skupina príkazov sa má vykonať, len ak je splnená určitá podmienka
- má tvar: ak *podmienka* tak *príkaz*;
- vykonanie: ak je podmienka splnená, vykoná sa príkaz

- príklady: ak $X \bmod 2 = 0$ tak piš (' Číslo je párne. '); ak (VstupnaHodnota1 = "") or (VstupnaHodnota2 = "") tak piš (' Chýba vstupná hodnota! '); ak Cislo ≥ 0 tak Odmocnina := sqrt(Cislo); (Poznámka: funkcia sqrt slúži na výpočet druhej odmocniny).


Podmieneny príkaz if (neúplný)

V programovaní vetveniu (rozhodovaníu) zodpovedá podmienený príkaz if

- použijeme, ak nejaký príkaz sa má vykonať, len ak je splnená podmienka
- má tvar: if *podmienka* then *príkaz*;
- vykonanie: ak vyhodnotením podmienky je hodnota true, vykoná sa príkaz za then, ak hodnota false, príkaz if je bez účinku.
- príklady:
 if Cislo > 0 then Label1.Caption := 'Číslo je kladné';
 if Slovo1 = Slovo2 then RovnakeSlova := true;
 if false then Label1.Caption := 'Toto sa nikdy nevypíše!!!';
 if Cislo < 0 then Cislo := - Cislo;


Príklad 2.1

Program 1.5 doplňte tak, aby výpočet prebehol, len ak vo vstupných poliach Edit1.Text a Edit2.Text sú neprázdne reťazce.

 Príslušná časť programu:

```

procedure TForm1.Button1Click(Sender: TObject);
var VyskaPozicky, MesacnaSplatka, SplacatRokov, SplacatMesiakov, Doplatok: integer;
    PocetMesiakov: integer;
begin
//získovanie, či sú zadané všetky vstupné hodnoty
if (Edit1.Text<>") and (Edit2.Text<>")
then begin
    //priradenie vstupných hodnôt z komponentov Edit do premenných
    VyskaPozicky := StrToInt(Edit1.Text);
    MesacnaSplatka := StrToInt(Edit2.Text);
    //výpočet
    PocetMesiakov := VyskaPozicky div MesacnaSplatka;
    SplacatRokov := PocetMesiakov div 12;
    SplacatMesiakov := PocetMesiakov mod 12;
    Doplatok := VyskaPozicky mod MesacnaSplatka;
    //zobrazenie výsledkov pomocou komponentov Label
    Label4.Caption := '- počet rokov: ' + IntToStr(SplacatRokov);
    Label5.Caption := '- počet mesiacov: ' + IntToStr(SplacatMesiakov);
    Label6.Caption := '- doplatok: ' + IntToStr(Doplatok) + ' €';
end
end;
```

 Všimnite si, že ak je podmienka v príkaze if splnená, má sa vykonať nie jeden, ale viacej príkazov za then. V takom prípade ich musíme uzatvoriť medzi slová begin a end, aby procesor vedel, ktoré príkazy patria k then (vytvorili sme tzv. **zložený príkaz**).

Program spĺňa požiadavky zadania, ak chýba čo i len jedna vstupná hodnota, výpočet neprebehne. Určite by bolo vhodnejšie, keby program aj vypísal, prečo neuskutočnil výpočet, t.j. že neboli

zadané všetky vstupné hodnoty. Prakticky to znamená doplniť vetvenie *ak ... tak ...* o vetvu *inak ...*, ktorej príkazy sa vykonávajú, ak podmienka nie je splnená.

Vetvenie (úplné, binárne)

- použijeme, ak príkaz alebo skupina príkazov sa má vykonať, len ak je splnená určitá podmienka a iná skupina príkazov sa má vykonať, ak podmienka nie je splnená
- má tvar: **ak podmienka tak príkaz1 inak príkaz2**;
- vykonanie: ak je podmienka splnená, vykoná sa príkaz1, inak sa vykoná príkaz2
- príklady: ak $X \bmod 2 = 0$ tak píš (' Číslo je párne. ') inak píš (' Číslo je nepárne ');
ak (VstupnaHodnota1 = "") or (VstupnaHodnota2 = "") tak píš (' Chýba vstupná hodnota! ') inak počítaj;

Podmienený príkaz if (úplný)

- použijeme, ak nejaký príkaz sa má vykonať, len ak je splnená podmienka a iný príkaz, ak podmienka nie je splnená
- má tvar: **if podmienka then príkaz1 else príkaz2**;
- vykonanie: ak vyhodnotením podmienky je true, vykoná sa príkaz1, ak false, vykoná sa príkaz2
- príklady:
if Cislo > 0 then Label1.Caption := 'Číslo je kladné' else Label1.Caption := 'Číslo nie je kladné';
if false then Label1.Caption := 'Toto sa nikdy nevypíše!!!' else Label1.Caption := 'Toto sa vždy vypíše!!!';
if Slovo1 = Slovo2 then RovnakeSlova := true else RovnakeSlova := false;

☛ Posledný príkaz if v príklade možno nahradiť aj príkazom priradenia: `RovnakeSlova := Slovo1 = Slovo2`;

☛ Pri vkladaní príkazu if do programu využívajte „dokončovač príkazov“. Napíšete if a stlačíte CTRL+J (za if nekladajte medzeru!). V ponuke (obrázok vpravo) máte neúplný príkaz if s begin a end (ifb), úplný príkaz if bez begin a end (ife), úplný príkaz if s begin a end (ifeb) a neúplný príkaz if bez begin a end (ifs). Ak použijete skratku, napr. ifeb, a stlačíte CTRL+J, nemusíte vyberať z ponuky.

if statement	ifb
if then (no begin/end) else (no begin/end)	ife
if then else	ifeb
if (no begin/end)	ifs

Vráťme sa k nášmu problému. Chceme, aby program vypísal: Neboli zadané všetky vstupné hodnoty!, ak niektoré zo vstupných polí je prázdne. Na výpis takýchto hlásení sa môže použiť príkaz ShowMessage.

Príkaz **ShowMessage** zobrazí v strede obrazovky okno so správou, s menom aplikácie (meno projektu) ako titulok a s tlačidlom OK.

Tvar: ShowMessage(reťazec)

Napr. ShowMessage('Chyba vstupu!');

Pozri aj ShowMessagePos, MessageDlg, MessageDlgPos a MessageBeep.



☛ Pozri aj... znamená použiť pomocníka: Help - Delphi Help - Index - napísať prvé písmená hľadaného pojmu v poli 1 až po jeho výber v poli 2. Rýchlejšie je rovno v programe (unite) napísať prvé písmená hľadaného pojmu a stlačiť CTRL+F1.

```

Program k príkladu 2.1 môžeme ešte vylepšiť správou, pretože nedošlo k výpočtu, teda doplniť:
end                                     //end na konci vetvy then, príkaz if nekončí, nesmie byť bodkočiarka
else ShowMessage('Neboli zadané všetky vstupné hodnoty!');                       //dopísaná vetva else
end;

```

☞ ☞ Ostatný program doplňte tak, aby riadok „- doplatok: ... €“ vypísal, len ak je doplatok nenulový.

☞ ☞ Aj v ostatných programoch z časti sekvencia môžete ošetriť chyby vstupu, napríklad aj zadanie záporného čísla ako strany štvorca, obdĺžnika, kvádra, polomeru kružnice, gule a pod.

1. Naprogramujte:

ak *neboli zadané všetky vstupné hodnoty alebo niektorá vstupná hodnota je záporná*

tak piš ('Chyba vstupu!')

inak počítaj

2. Naprogramujte:

ak *neboli zadané všetky vstupné hodnoty*

tak piš ('Neboli zadané všetky vstupné hodnoty!')

inak

ak *niektorá vstupná hodnota je záporná*

tak piš ('Zle zadané vstupné hodnoty!')

inak počítaj

☞ Svoje programy nezapodnajte otestovať na všetky možnosti; je ich viac, ako si možno myslíte!

Príklad 2.2

Vráťme sa k príkladu na výpočet výsledného odporu dvoch rezistorov po zadaní ich odporov R_1 a R_2 (ostatná úloha nad príkladom 1.3). Úlohu ste mali riešiť pre sériové aj paralelné zapojenie rezistorov. Úlohu doplníme o zadanie spôsobu zapojenia rezistorov z klávesnice a podľa vstupu (zapojené sériovo alebo paralelne) nech prebehne len jeden z výpočtov $R = R_1 + R_2$ alebo $R = R_1 \cdot R_2 / (R_1 + R_2)$.

Algoritmus

Jedno z možných riešení:

ak Zapojenie = 'séριοvo'

tak $R := R_1 + R_2$

inak $R := R_1 \cdot R_2 / (R_1 + R_2)$

kde Zapojenie je premenná typu reťazec.

☞ Všimnite si a uvedomte, pretože je slovo séριοvo v apostrofoch!!!


☞ Veríme, že uvedený algoritmus dokážete bez problémov odladiť v programe.

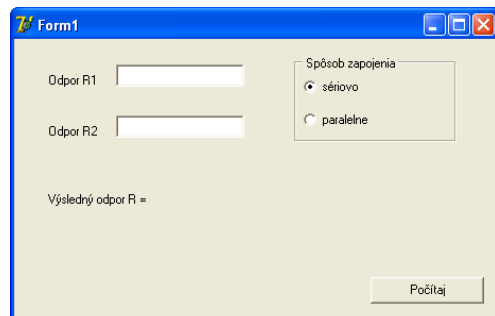
☞ ☞ Ako sa zachová program z príkladu 2.2, ak namiesto slova séριοvo napíšete iné slovo, napr. seriovo? Zovšeobecnite.

☞ ☞ Program z príkladu 2.2 upravte tak, aby počítal len na slová séριοvo alebo paralelne, inak nech vypíše Zapojenie nepoznám (napíš séριοvo alebo paralelne)!

☞ ☞ Program z príkladu 2.2 ošetríte aj na „prázdny vstup“.

☺ Pre tých, čo chcú vedieť viac z Delphi, popíšeme

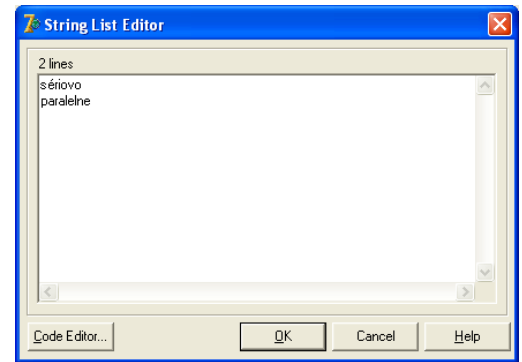
použitie komponentu RadioGroup , ktorého použitie je vhodné pri povinnom výbere jednej z viacerých možností. „Vyrobíme“ formulár s komponentmi, ako to je na obrázku vpravo. Komponent s prepínačmi séριοvo/paralelne je už spomenutý RadioGroup vložený zo záložky Standard.



Do Caption sme napísali text: Spôsob zapojenia. V Object Inspector sme klikli vpravo od vlastnosti Items, a následne na „trojbodkové“ tlačidlo, čím sme vyvolali String List Editor (obrázok), ktorý nám umožňuje vložiť položky RadioGroup. Prvá položka má index 0, druhá 1 atď. Ak teda chceme, aby po spustení programu bola označená prvá položka, do poľa ItemIndex napíšeme nulu.

Tu je podstatná časť programu:

```
procedure TForm1.Button1Click(Sender: TObject);
var R1, R2, R: real;
begin
if (Edit1.Text<>'' and (Edit2.Text<>''))
then begin
    R1 := StrToFloat(Edit1.Text);
    R2 := StrToFloat(Edit2.Text);
    if RadioGroup1.ItemIndex = 0
    then R := R1 + R2
    else R := R1*R2/(R1 + R2);
    Label3.Caption := 'Výsledný odpor R = ' + FloatToStr(R) + 'ohm';
end
else ShowMessage('Chyba vstupu! Nepočítam!');
end;
```



Príklad 2.3

Vytvorte program, ktorý po zadaní pH oznámi, či je prostredie kyslé, neutrálne alebo zásadité.

Algoritmus

Z chémie musíme vedieť, že pH je reálne číslo z intervalu $\langle 0, 14 \rangle$, pričom pri $\text{pH} < 7$ je prostredie kyslé, pri $\text{pH} = 7$ je prostredie neutrálne a pri $\text{pH} > 7$ je zásadité. V stĺpcoch sú tri riešenia:

ak $\text{pH} < 7$

ak $\text{pH} < 7$ **tak** pís('kyslé');

ak $\text{pH} < 7$ **tak** pís('kyslé');

tak pís('kyslé')

ak $\text{pH} = 7$

ak $\text{pH} = 7$ **tak**

pís('neutrálne');

tak pís('neutrálne')

ak $\text{pH} > 7$ **tak**

inak **ak** $\text{pH} = 7$

pís('zásadité');

inak pís('zásadité');

tak pís('neutrálne')

inak pís('zásadité');

Prezradíme vám, že jedno z riešení je zlé. Ktoré a prečo?


Úplný program aj s ošetrením zadaného pH z intervalu $\langle 0, 14 \rangle$.

```
procedure TForm1.Button1Click(Sender: TObject);
var pH: real;
begin
if Edit1.Text='' then ShowMessage('Nebolo zadané pH!')
else begin
    pH:= StrToFloat(Edit1.Text);
    if (pH < 0) or (pH > 14) then ShowMessage('Hodnota pH mimo dovolený interval!')
    else if pH < 7 then Label2.Caption:= 'kyslé.'
    else if pH = 7 then Label2.Caption := 'neutrálne.'
```

```
else Label2.Caption := 'zásadité.';
```

```
end;
```

```
end;
```

 Vytvorte program na riešenie kvadratickej rovnice, ak ste už na hodinách matematiky o kvadratickej rovnici „počuli“.

Algoritmus

Pripomenieme:

- kvadratický koeficient „a“ musí byť rôzny od nuly (ináč by rovnica nebola kvadratická).
- o riešení rozhoduje diskriminant $= b^2 - 4ac$
- ak je diskriminant < 0 , tak rovnica nemá v \mathbb{R} riešenie
- ak je diskriminant $= 0$, koreňom je číslo $\frac{-b}{2a}$
- ak je diskriminant > 0 , rovnica má dva rôzne reálne korene $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, kde D je diskriminant.

Jeden z možných tvarov programu:

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
var a, b, c, Diskriminant, x1, x2: real;
```

```
begin
```

```
if (Edit1.Text = '') or (Edit2.Text = '') or (Edit3.Text = '') then Label4.Caption := 'Nezadané všetky čísla!'
```

```
else if Edit1.Text = '0' then Label4.Caption := 'Rovnica nie je kvadratická!'
```

```
else begin
```

```
  a := StrToFloat(Edit1.Text);
```

```
  b := StrToFloat(Edit2.Text);
```

```
  c := StrToFloat(Edit3.Text);
```

```
  Diskriminant := b*b - 4*a*c;
```

```
  if Diskriminant < 0 then Label4.Caption := 'Rovnica nemá v R riešenie.';
```

```
  if Diskriminant = 0 then Label4.Caption := 'Rovnica má jeden dvojnásobný koreň x = ' +  

    FloatToStr(-b/(2*a));
```

```
  if Diskriminant > 0
```

```
  then begin
```

```
    x1 := (-b + sqrt(Diskriminant))/(2*a);
```


```
    x2 := (-b - sqrt(Diskriminant))/2/a; //všimnite si nepoužitie zátvoriek!
```

```
    Label4.Caption := 'x1 = ' + FloatToStr(x1) + ' x2 = ' + FloatToStr(x2);
```

```
  end;
```

```
end;
```

```
end;
```

 Vytvorte program, ktorý nájde najväčšie z troch (štyroch) celých čísel.

Algoritmus

Úloha má viacej správnych riešení, aby sme sa však nezamotali do porovnávaní, navrhujeme použiť tento najvšeobecnejší postup využívajúci neúplné binárne vetvenie:


```
Maximum := Cislo1;
```

```
ak Cislo2 > Maximum tak Maximum := Cislo2;
```

```
ak Cislo3 > Maximum tak Maximum := Cislo3;
```

```
píš (Maximum);
```

Veríme, že zrealizovať daný algoritmus vo forma programu je už pre vás hračkou, ako aj upraviť na program, ktorý je schopný nájsť maximum zo štyroch celých čísel.

 Vytvorte program, ktorý po zadaní vašej výšky a výšky vášho kamaráta v cm vypíše: Si vyšší o ...cm alebo Kamarát je vyšší o ...cm alebo Ste rovnako vysokí.

Algoritmus

Naznačíme algoritmus bez výpisu rozdielu výšok:

začiatok


čítaj(v1,v2);

ak $v1 > v2$ tak piš ('Si vyšší.')

inak ak $v1 = v2$ tak piš('Sme rovnako vysokí.')

inak piš('Kamarát je vyšší.')

koniec

 Nosnosť výťahu je 150kg. Vytvorte program, ktorý po zadaní hmotností dvoch čakateľov na odvezenie vypíše potrebný počet jász. Žiaden z čakateľov nemá viac ako 150kg.

Algoritmus


začiatok

čítaj(h1,h2);

ak $h1 + h2 \leq 150$ tak piš ('1 jazda')

inak piš('2 jazdy')

koniec

 Nosnosť výťahu je 150 kg. Vytvorte program, ktorý po zadaní hmotností troch čakateľov na odvezenie vypíše potrebný počet jász. Žiaden z čakateľov nemá väčšiu hmotnosť ako je nosnosť výťahu.

Algoritmus

začiatok

Nosnost:=150;


čítaj(h1,h2,h3);

ak $h1 + h2 + h3 \leq \text{Nosnost}$ tak piš ('1 jazda')

inak ak $(h1 + h2 \leq \text{Nosnost})$ alebo $(h1 + h3 \leq \text{Nosnost})$ alebo $(h2 + h3 \leq \text{Nosnost})$ tak piš('2 jazdy')

inak piš('3 jazdy')

koniec

 Nech sa nadváha počíta z hmotností v kg a výšky v cm vzorcom: $\text{nadváha} = \text{hmotnosť} - (\text{výška} - 100)$. Vytvorte program, ktorý vypočíta vašu nadváhu a vypíše: Máš nadváhu ... kg (ak je nadváha kladné číslo) alebo Si v norme, ak vaša nadváha je z intervalu -10 až 0, inak vypíše Si podvýživený o ... kg.

```
procedure TForm1.btUrciClick(Sender: TObject);
```

```
var Hmotnost, Vyska, Nadvaha: integer;
```

```
begin
```

```
Hmotnost:= StrToInt(Edit1.Text);
```

```
Vyska:= StrToInt(Edit2.Text);
```

```
Nadvaha:= Hmotnost - Vyska + 100;
```

```
if Nadvaha>0
```

```
then Label3.Caption:= 'Máš nadváhu '+IntToStr(Nadvaha)+' kg'
```


```
else if Nadvaha< -10
```


```
then Label3.Caption:= 'Si podvýživený o '+IntToStr(abs(Nadvaha+10))+' kg'
```

```
else Label3.Caption:='Si v norme';
```

```
end;
```

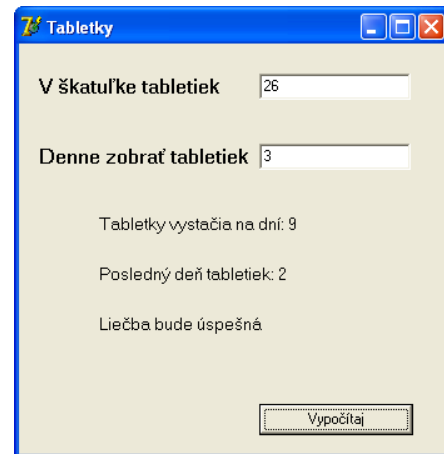



✎  Vytvorte program, ktorý po zadaní celého kladného čísla zistí a vypíše, či je číslo deliteľné 2, 3 alebo 5, prípadne, že nie je deliteľné žiadnym zo zadaných čísel.


✎  Vytvorte program, ktorý vypíše, na koľko dní máme lieky, po zadaní počtu tabletiiek v škatuľke a počtu predpísaných tabletiiek na deň. Program vypíše: Tabletky vystačia na ...dní. Na posledný deň zostalo tabletiiek.... Liečba je úspešná, ak trvá aspoň 7 dní. Nech program vypíše: Liečba bude úspešná alebo Liečba nebude trvať dostatočné dlho.


```


procedure TForm1.btVypocitajClick(Sender: TObject);
var Pocet, Denne, Dni, Zvysok: integer;
begin
Pocet:= StrToInt(Edit1.Text);
Denne:= StrToInt(Edit2.Text);
Dni:= Pocet div Denne;
Zvysok:= Pocet mod Denne;
if Zvysok>0 then Dni:= Dni+1;
Label3.Caption:= 'Tabletky vystačia na dni: '+ IntToStr(Dni);
Label4.Caption:= 'Posledný deň tabletiiek: '+ IntToStr(Zvysok);
if Dni>=7 then Label5.Caption:= 'Liečba bude úspešná'
else Label5.Caption:= 'Liečba nebude trvať dostatočne dlho'
end;
  
```





✎  Prvé auto prešlo dráhu s1 za čas t1. Druhé auto dráhu s2 za čas t2. Vytvorte program, ktorý vypočíta ich rýchlosti ($v=s/t$) a vypíše: Prvé auto išlo rýchlejšie o ...km/h alebo Druhé auto išlo rýchlejšie o ...km/h alebo Autá išli rovnako rýchlo.

✎  Vytvorte program, ktorý po zadaní troch kladných reálnych čísel oznámi, či môžu tvoriť strany trojuholníka, ak áno, či je rovnoramenný alebo rovnostranný, či je pravouhlý.

✎  a, b, c sú strany pravouhlého trojuholníka (c prepona). Neznámu stranu trojuholníka nezadáme (prázdny Edit). Napíšte program, ktorý vypočíta túto neznámu stranu. Napríklad sme zadali a: 3, b: nič, c: 5 (treba počítať stranu b). V pravouhlom trojuholníku platí $a^2 + b^2 = c^2$. Druhá odmocnina v Delphi je funkcia sqrt, t.j. $b := \text{sqrt}(c*c - a*a)$. Výstup: do prázdneho Edit-u dáme zapísať výsledok - číslo 4.

✎  Program kalkulačka z kapitoly sekvencia ošetríte na delenie nulou a doplníte o tlačidlo na výpočet druhej odmocniny z nezáporného reálneho čísla.

✎  Vytvorte program, ktorý utriedi vzostupne tri celé čísla. Návod: Porovnávajúte dvojice čísel, a ak treba, vymeňte čísla v premenných (napr. ak $a > b$ tak vymeň(a,b); výmena hodnôt dvoch premenných je popísaná v príklade 1.1). Po skončení programu by malo platiť $a \leq b \leq c$, kde do a, b, c sme uložili zadané čísla.

✎  Vytvorte program, ktorý abecedne zoradí tri zadané slová.

Poznámka:

Ak vyriešite predchádzajúcu úlohu, stačí vám v programe len zmeniť údajový typ integer na string a odstrániť konverziu StrToInt a opačne, program bude správne pracovať. Počítač vie, že napríklad 'ab' < 'ac'.

Vytvorte program, ktorý nájde riešenie rovnice $ax + b = 0$ pre $a, b \in \mathbb{R}$.

Algoritmus

Ak $a \neq 0$ tak riešením lineárnej rovnice $ax + b = 0$ je číslo $x = -b/a$,

ak $a = 0$, potom, ak aj $b = 0$, dostávame rovnicu $0.x + 0 = 0$, čo je zrejme splnené pre každé reálne číslo x (obor pravdivosti je celá množina \mathbb{R} , riešením je každé reálne číslo),

ak $b \neq 0$, dostávame rovnicu $0.x + \text{nenulové } b = 0$, čo zrejme nie je splnené pre žiadne x (obor pravdivosti je prázdna množina, rovnica nemá riešenie).

*Vytvorte program na riešenie rovnice $ax^2 + bx + c = 0$, $a, b, c, x \in \mathbb{R}$.

Poznámka:

Program je spojením riešenia kvadratickej rovnice a rovnice „nekvadratickej“, t.j. keď je $a = 0$ a treba riešiť rovnicu $bx + c = 0$ (predchádzajúci príklad).

Vytvorte program, ktorý po zadaní roku vypíše, či je priestupný alebo nepriestupný.

Poznámka: Rok je priestupný, ak je deliteľný 4 a nie je deliteľný 100, okrem rokov deliteľných 400. Napríklad rok 1600 je priestupný ale rok 1700 nie. Na vzorové riešenie stačí použiť jeden príkaz `if` a operátory `and` a `or`.

VETVENIE V ŽIVOTE

Každý deň robíme množstvo rozhodnutí, niektoré sú jednoduché iné zložitejšie.

Ak mám dnes vyučovanie, tak musím ráno vstať z postele, inak pohoda.

Ak doma nič nemáme (rozumej, čo by mi chutilo), tak nebudem raňajkovať.

Ak mám dosť peňazí, tak si to môžem kúpiť, inak, ak to veľmi chcem, tak si ich musím zadovážiť.

Ak sa mi prihovori, tak budem rád/rada, inak nech ju/ho...

Ak ma vyvolá, musím zahrať divadlo, inak dostanem „gul'u“.

Ak poviem áno, bude to rozprávka (mám ho/ju „na krku“ celý život) inak to bude „trapas“.

Ak prekročím dovolenú rýchlosť a niekoho zrazím, tak musím niesť následky.

Ak zoženiem lístky, pôjdeme na koncert, inak sedím doma.

...

Funkcia `InputBox` a komponent `Memo`

Na záver tejto kapitoly si ozrejmime ďalšie možnosti prostredia Delphi. Na vstup údajov do programu nemusíme použiť len komponent `Edit`, ale môžeme použiť aj funkciu `InputBox`. Jej výhodou je, že umožňuje zobrazíť vstupný panel a zadať vstupné hodnoty kedykoľvek počas behu programu. Vždy sa vyskytuje na pravej strane príkazu priradenia!

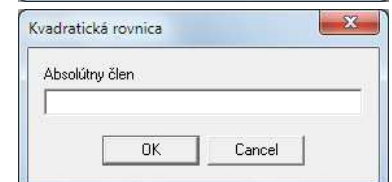
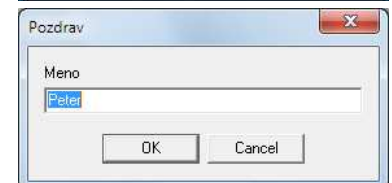
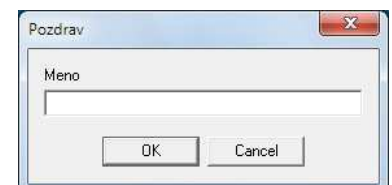
Použitie napríklad

```
Meno := InputBox ('Pozdrav', 'Meno', '');
```

```
Meno := InputBox ('Pozdrav', 'Meno', 'Peter');
```

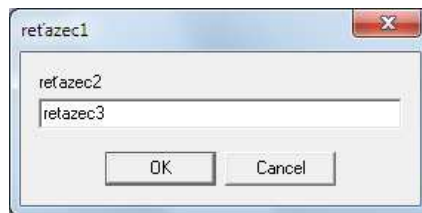
Peter je tzv. default hodnota, t.j. predvolená a môže ju užívateľ buď potvrdiť kliknutím na OK alebo zmeniť.

```
c := StrToFloat ( InputBox ('Kvadratická rovnica', 'Absolútny člen', ''));
```



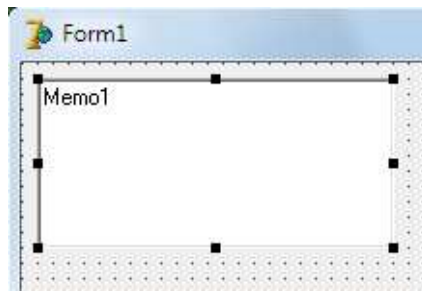
Všeobecne:

Výsledkom funkcie `InputBox ('reťazec1', 'reťazec2', 'reťazec3')` je vpísaný alebo potvrdený (default) `reťazec3`. Ten sa priradí premennej na ľavej strane príkazu priradenia. Ak má byť „pochopený“ ako číslo, musí sa použiť konverzná funkcia!



Komponent Memo

umožňuje viacriadkový výstup. Pred spustením programu možno text v Memo upraviť cez vlastnosť `Lines` (dvojklikom na `TStrings` sa vyvolá `String List Editor`). Počas behu programu možno reťazce po riadkoch pridávať príkazom `Memo1.Lines.Add (reťazec)`.



Príkaz `Memo1.Clear` zmaže Memo.

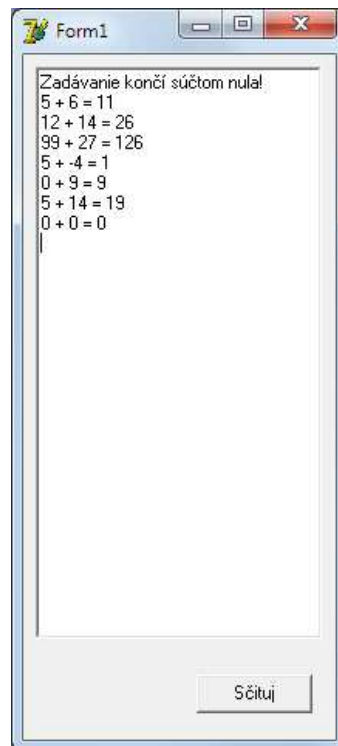
Vytvorte program, ktorý si vypýta dva sčítance - celé čísla a vypíše ich súčet. Program nech počíta, kým nezadáme súčet rovný nule!

Analýza:

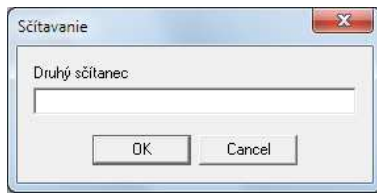
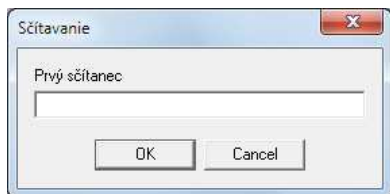
V programe je najjednoduchšie použiť funkcie `InputBox` a komponent `Memo`.

Dovolili sme si jeden prehrešok, použiť príkaz `repeat – until`, aj keď ešte nepoznáte. Sme presvedčení, že napriek tomu pochopíte, čo robí – zabezpečuje opakovanie príkazov napísaných medzi `repeat` a `until`, kým nebude súčet rovný nule.

```
procedure TForm1.ScitujClick(Sender: TObject);
var scitanec1, scitanec2, sucet: integer;
begin
Memo1.Lines.Add ('Zadávanie končí súčtom nula!');
repeat
// opakuj
scitanec1 := StrToInt (InputBox ('Sčítavanie', 'Prvý sčítanec', ''));
scitanec2 := StrToInt (InputBox ('Sčítavanie', 'Druhý sčítanec', ''));
sucet := scitanec1 + scitanec2;
Memo1.Lines.Add (IntToStr (scitanec1) + ' + ' + IntToStr(scitanec2)
+ ' = ' + IntToStr (sucet));
until sucet = 0;
// pokiaľ nebude sucet = 0
end;
```



Po kliknutí na tlačidlo `Sčítaj` sa najprv postupne zobrazia dva panely




umožňujúce zadať dve celé čísla.

☞ Pokúste sa „rozlúsknuť“ zápis v riadku `Memo1.Lines.Add (...)`;

☞ Predchádzajúci program modifikujte (zmeňte) tak, aby počítal súčin (podiel) zadaných čísel.

CYKLUS

Ako vypočítame 2^5 ak nemáme kalkulačku? Zrejme si povieme: dva krát dva je štyri, krát dva je osem (to je už dva na tretiu), krát dva je šesťnásť, krát dva je tridsaťdva. Opakujeme násobenie dvoma. Potreba opakovane vykonávať nejakú činnosť sa v praxi vyskytuje často a „neobišla“ ani programovanie. Ak pri riešení problému potrebujeme zabezpečiť, aby sa určitá skupina príkazov opakovane vykonávala, použijeme cyklus. Príkazov cyklu, ktoré zabezpečia opakovanie určitej skupiny príkazov potrebný početkrát, je viacej. Začneme s príkazom cyklu s pevným počtom opakovaní, s príkazom `for`.

 Pri cykloch sa nám môže najskôr beh programu „vymkne z rúk“. Zastaviť ho môžeme položkou **Run – Program Reset** (ale len keď nereaguje na „krížik“ Zavrieť)!


Cyklus s pevným počtom opakovaní

Príkaz `for`

- použijeme, ak
 1. potrebujeme, aby sa opakovane vykonávala skupina príkazov a zároveň
 2. poznáme potrebný počet opakovaní týchto príkazov
- cyklus s pevným počtom opakovaní je programová konštrukcia, do ktorej vložíme príkazy, ktoré sa majú opakovane vykonávať a zadáme, koľkokrát sa majú opakovať
- má tvar:

```
for RiadiacaPremennáCyklu := ZačiatočnáHodnota to KoncováHodnota do  
begin  
    Príkazy, ktoré sa majú opakovane vykonávať  
end;
```
- napr.:

```
for Pocitadlo := 1 to 100 do  
begin  
    Príkazy // Príkazy sa vykonajú 100-krát  
end;  
for i := 1 to N do ... // Pre  $N \geq 0$  sa príkazy vykonajú N-krát  
for i := 1 to 1 do ... // Príkazy sa vykonajú raz  
for i := 1 to 0 do ... // Príkazy sa nevykonajú ani raz, pretože  
// koncová hodnota je menšia ako začiatočná
```
- vykonanie `for`-cyklu:
 1. riadiacej premennej cyklu sa priradí začiatočná hodnota
 2. pokiaľ je hodnota riadiacej premennej cyklu menšia alebo sa rovná koncovkej hodnote, opakovane sa vykonávajú príkazy v cykle (medzi `begin` a `end`) a hodnota riadiacej premennej cyklu sa zvyšuje na nasledovníka riadiacej premennej cyklu.

 Ak sa má v príkaze `for` opakovať len jeden príkaz, stačí ho napísať za vyhradené slovo `do` bez `begin` a `end`, ako napríklad v nasledujúcich príkladoch!

Príklad 3.1.1

Vytvorte program, ktorý vypíše pod seba všetky prirodzené čísla od 1 po zadané prirodzené číslo.

Algoritmus:

Riešenie problému pozostáva z dvoch krokov:

1. opýtať sa a zadať, po ktoré prirodzené číslo sa majú vypisovať prirodzené čísla od 1 a
2. vypísať 1, potom 2, potom 3,... až zadané číslo.

Z toho vyplýva použitie cyklu (opakuje sa výpis vhodného čísla) a keďže vieme, koľkokrát sa má výpis zrealizovať, môžeme použiť cyklus s pevným počtom opakovaní, teda príkaz for.

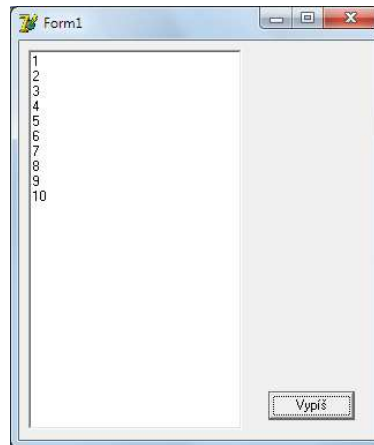
Napríklad pri použití premenných Cisko a Zadane: for Cisko := 1 to Zadane do „vypíš Cisko“;

Keďže potrebujeme viacriadkový výstup, musíme použiť komponent Memo.

Riešením je procedúra:

```
procedure TForm1.VypisClick(Sender: TObject);
var Cisko, Zadane: integer;
begin
Zadane:= StrToInt ( InputBox ( 'Výpis' , 'Vypísať od 1 po' , '10' ) );
for Cisko := 1 to Zadane do Memo1.Lines.Add ( IntToStr ( Cisko ) );
end;
```

☛ Ak vložíme veľkú hodnotu pre Zadane, začiatok výpisu nám „utečie“ („skroluje“) a preto je potrebné, pri ukončenom behu programu, po kliknutí do komponentu Memo1 vyhľadať v Object Inspector v záložke Properties jeho vlastnosť ScrollBars a zmeniť jej hodnotu z ssNone na ssVertical.



☛ Príkaz for nemožno nahradiť príkazom

```
for Cisko := 1 to Zadane do Label1.Caption := IntToStr ( Cisko);
```

Ak by sme odladili takýto program, vo formulári by sme uvideli len hodnotu čísla Zadane (Prečo?).

📖 Predchádzajúci program zmeňte tak, aby vypísal všetky prirodzené čísla od zadaného čísla nazvaného Zaciatok po zadané číslo nazvané Koniec. Napríklad pre Zaciatok = 3 a Koniec = 7 vypíše pod seba 3 4 5 6 7.

Príklad 3.1.2

Vieme, že pri kódovaní znakov sa v počítači používajú najčastejšie ASCII alebo Unicode. Zároveň by sme mali vedieť, že prvých 32 znakov (0-tý až 31. znak) sú riadiace znaky a „nevieme“ ich zobrazit'. Kódy znakov bežne využívame, veď mnohí vieme, že napríklad po vložení ľavé Alt+64 (stlačený ľavý kláves Alt a pripísané z numerickej klávesnice 64) sa vloží znak @ alebo pri Alt+39 sa vloží apostrof. Vyskúšajte vložiť aj iné kódy-znaky.

Vytvorte program, ktorý vypíše 32. až 127. znak kódovacej tabuľky (prvých 128 znakov sa v ASCII a Unicode zhoduje).

Analýza

Potrebujeme poznať funkciu chr (prirodzené číslo), ktorá vráti znak (char) zodpovedajúci zadanému prirodzenému číslu. Napríklad chr(64) vráti @, chr(32) zdanlivo nič nevráti, pretože to je medzera ☺. Pre prácu so znakmi bol zavedený aj samostatný údajový typ char.

📖 char - znak

údajový typ, ktorého hodnoty sú znaky použitej kódovacej tabuľky.

Možno použiť funkcie:

char (prirodzené číslo)	vráti znak zodpovedajúci zadanému prirodzenému číslu v použitej kódovacej tabuľke, prvých 32 znakov (od 0 po 31) sú riadiace Např. chr(65) = A, chr(90) = Z, chr(48) = 0, chr(97) = a
ord (znak)	vráti poradové (ordinálne) číslo zadaného znaku Například ord('A') = 65, ord('z') = 122, ord('9') = 57 Zrejme platí chr(ord(znak)) = znak a ord(chr(číslo)) = číslo
succ(znak)	vráti nasledovníka (successora) zadaného znaku (ak existuje) Například succ('A') = B, succ('9') = : , succ(' ') = !

pred(znak) vráti predchodcu (predecessora) zadaného znaku (ak existuje)
Napríklad pred('A') = @, pred(':') = 9, pred('!') = medzera,...
UpCase(znak) zmení malé písmeno anglickej abecedy na veľké, ostatné znaky nemení
Napríklad Odpoved := UpCase(Odpoved); if Odpoved = 'A' then...

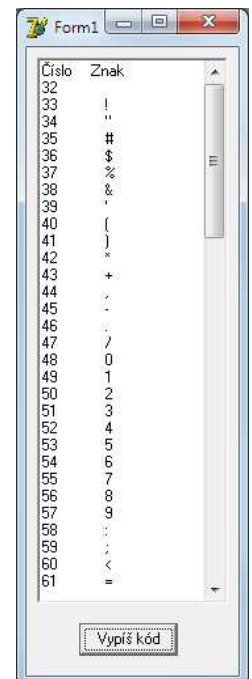
Vráťme sa k nášmu problému. Potrebujeme vypísať 32. potom 33. atď. až 127. znak. Opakuje sa výpis znaku po zadaní jeho poradového čísla v kódovacej tabuľke preto cyklus, vieme počet opakovaní, od 32 po 127 preto cyklus s pevným počtom opakovaní, for-cyklus. Viacriadkový výstup sa realizuje pomocou komponentu Memo.

Algoritmus

začiatok
pre Cislo := 32 až po 127 opakuj pís (chr(Cislo));
koniec;

V programe samozrejme chceme, aby sa zobrazilo aj poradové číslo znaku, nie len samotný znak, teda chceme dva stĺpce

Poradové číslo	Znak
32	
33	!
34	"
...	...
127	␣



Výsledok zobrazený vpravo dostaneme po napísaní:

```
procedure TForm1.VypisKodClick(Sender: TObject);  
var Cislo: integer;  
begin  
for Cislo:= 32 to 127 do  
Memo1.Lines.Add ( IntToStr (Cislo) + ' ' + chr (Cislo) );  
end;
```

V Memo1 sme prvý riadok zmenili na Číslo a Znak a doplnili ho vertikálnym posuvníkom (vlastnosť ScrollBars).

Príklad 3.1.3

Vytvorte program, ktorý zistí, či zadané prirodzené číslo je prvočíslo.

Analýza

V tejto fáze našich vedomostí z programovania sa uspokojíme s akýmkoľvek správnym (neznamená efektívnym) riešením.

Vieme, že prirodzené číslo je prvočíslo, ak má práve dvoch rôznych deliteľov, jednotku a seba samé. Z tvrdenia je zrejmé, že do úvahy prichádzajú len čísla väčšie ako jedna. Najjednoduchšie zistiť, či je zadané číslo prvočíslo, je pravdepodobne hľadať jeho delitele až po samotné číslo. Čiže skúsiť, či je číslo deliteľné 2, potom 3,..., číslo-1 preto cyklus (opakuje sa delenie) s pevným počtom opakovaní, keďže cyklus sa má vykonať pre deliteľ dvojku, trojku,... až číslo-1. Môžete namietnuť, že hneď, ako sa nájde deliteľ zadaného čísla, môžeme cyklus ukončiť. Máte síce pravdu, ale for-cyklus to neumožňuje. Na efektívnejšie riešenie si musíte počkať.

Algoritmus

Najprv vylúčime z riešenia všetky čísla menšie ako 2. Pre ostatné celé čísla budeme na začiatku výpočtu predpokladať, že zadané číslo je prvočíslo a toto tvrdenie zmeníme, len ak nájdeme deliteľa. Využijeme logický typ, ktorý pracuje s dvoma hodnotami nepravda (false) a pravda (true).

Teda do premennej JePrvocislo na začiatku hľadania deliteľa priradíme hodnotu pravda. Ak nájdeme deliteľa, zmeníme hodnotu v premennej JePrvocislo na nepravda! Po skončení cyklu nám už len stačí zistiť, aká hodnota je v premennej JePrvocislo.

Zistiť, či deliteľ delí zadané číslo, vieme pomocou funkcie mod. Ak $Cislo \bmod Delitel = 0$, deliteľ delí zadané číslo (bez zvyšku) a číslo nemôže byť prvočíslom.

Podstatná časť programu:

```

procedure TForm1.JePrvocisloClick(Sender: TObject);
var Cislo, Delitel: integer;
    JePrvocislo: boolean;
begin
Cislo:= StrToInt(Edit1.Text);           // zadanie celého čísla, o ktorom chceme zistiť, či je prvočíslo
if Cislo < 2
then Label2.Caption:= 'nie je prvočíslo'
else begin                               // zisťovanie prebehne len pre čísla väčšie ako 1
    JePrvocislo:= true;                  // predpokladáme, že zadané číslo je prvočíslo
    for Delitel:= 2 to Cislo-1 do if Cislo mod Delitel = 0 then JePrvocislo:= false;
    if JePrvocislo then Label2.Caption:= 'je prvočíslo'
    else Label2.Caption:= 'nie je prvočíslo';
end;
end;

```

☞ Všimnite si, že namiesto podmienky $JePrvocislo = true$ v poslednom príkaze if sme použili len $if JePrvocislo then...$ čo je postačujúce, vzhľadom k tomu, že za if sa očakáva logická hodnota true alebo false, a jednu z týchto hodnôt nadobudne aj premenná JePrvocislo.

📖 Vedeli by ste program upraviť tak, že ak je zadané číslo párne a väčšie ako 2, nech nezisťuje jeho delitele, keďže určite nie je prvočíslom? Stačí doplniť podmienku $if Cislo < 2...$

📖 Program doplňte tak, aby počas zisťovania hneď aj vypisoval delitele zadaného čísla (výpis môžete doplniť aj 1 a zadaným číslom).

📖 Funkcia **random** (celé_číslo) vráti náhodné celé číslo z intervalu $\langle 0, \text{celé_číslo} - 1 \rangle$. Funkcia random bez parametra vráti náhodne vybrané reálne číslo z intervalu $\langle 0, 1 \rangle$. Pri každom spustení programu sa vygeneruje (vyberie) rovnaké číslo. Znáhodnenie výberu aj prvého čísla sa dosiahne uvedením príkazu **randomize** pred prvým použitím funkcie random.

Napríklad $random(100)$ vráti náhodne vybrané celé číslo od 0 po 99. Výraz $random(6)+1$ vráti celé číslo od 1 po 6, pretože $random(6)$ vráti celé číslo od 0 po 5 a plus 1 ho zmení na 1 až 6. Výraz $2*Random(51) + 100$ generuje párne čísla od 100 po 200.

Funkcia random sa „vo veľkom“ využíva v počítačových hrách.

📖 Vytvorte program, ktorý do Memo vypíše sto náhodne vybraných reálnych čísel.

Príklad 3.1.4

Na precvičenie funkcií random a randomize vytvorte program simulujúci hod hracou kockou.

Analýza

Po kliknutí na tlačidlo Hod sa má náhodne vygenerovať a zobraziť celé číslo od 1 po 6. Program je veľmi jednoduchý, preto sme ho doplnili o nastavenie písma Arial s veľkosťou 200 a červenou farbou (tieto parametre komponentu Label sa dajú nastaviť aj „manuálne“ pred spustením programu vo vlastnosti Font).

```
procedure TForm1.HodClick(Sender: TObject);
begin
randomize;
Label1.Font.Name:= 'Arial';
Label1.Font.Size:= 200;
Label1.Font.Color:= clRed;

Label1.Caption:= IntToStr (random(6)+1);    // podstata programu ☺
end;
```



Príklad 3.1.5

Limonádový Joe si denne hádže mincou. Ak padne rub, ide do baru, ak líce, zostane doma. Vytvorte program simulujúci Joeov hod mincou. Program nech spočíta, koľkokrát za rok skončil doma a koľkokrát v bare.

Analýza

Máme dve možnosti. Môžeme využiť funkciu `random` s parametrom 2, t.j. `random(2)`, ktorá vracia dve hodnoty, nulu - rub a jednotku – líce s rovnakou pravdepodobnosťou, alebo funkciu `random` bez parametra. Potom interval $<0,1)$ možno rozdeliť na dve rovnaké časti $<0, 0.5)$ a $<0.5, 1)$. Číslo z prvého intervalu môže znamenať rub a číslo z druhého intervalu líce mince; opäť majú rovnakú pravdepodobnosť výskytu. Tým sme vyriešili simuláciu hodu.

K druhej časti programu: nech má rok 365 dní. Každý deň opakuje hod mincou preto cyklus s pevným počtom opakovaní, 365-krát. Vždy, keď padne rub, treba zvýšiť hodnotu v premennej `Doma` o jedna. Stačí spočítavať, koľkokrát skončil doma, t.j. koľkokrát padol napríklad rub. 365 – `Doma` dáva počet dní strávených v bare za rok.

Algoritmus

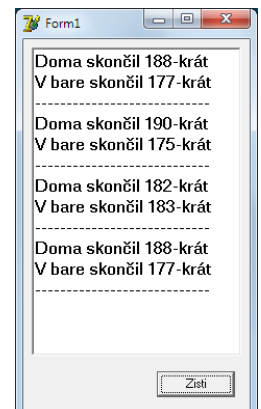
```
začiatok                // algoritmus je zaujímavý aj tým, že nemá žiaden vstup
Doma ← 0;
pre Den od 1 až po 365 opakuj ak „padol rub“ tak Doma ← Doma + 1;
píš (Doma);              // koľkokrát za rok skončil doma
píš (365-Doma);          // koľkokrát za rok skončil v bare
koniec;
```

Ak použijeme v programe komponent `Memo`, môžeme porovnať výsledky pre viac rokov – stačí opakovane stláčať tlačidlo `Zisti`, ktoré spúšťa simuláciu hodov za jeden rok.

Podstatná časť programu

```
procedure TForm1.ZistiClick(Sender: TObject);
const RokDni = 365;
var  Doma, Den: integer;
begin
randomize;
Doma:= 0;
for Den:= 1 to RokDni do if random(2) = 0 then Doma:= Doma + 1;

Memo1.Lines.Add ( 'Doma skončil ' + IntToStr (Doma) + '-krát' );
Memo1.Lines.Add ( 'V bare skončil ' + IntToStr (RokDni - Doma) + '-krát' );
Memo1.Lines.Add ( '-----' );
end;
```



V programe použite funkciu `random` bez parametra.

Príklad 3.1.6


Pokúste sa zistiť, čo robí nasledujúca časť programu.

```

procedure TForm1.Button1Click(Sender: TObject);
var PocetPismen, i: integer;
    Cosi: string;
begin
  randomize;
  PocetPismen:= StrToInt(Edit1.Text);

  Cosi:="";      // dva apostrofy vedľa seba, tzv. prázdny reťazec
  for i:= 1 to PocetPismen do Cosi:= Cosi + chr(random(26)+65);

  Memo1.Lines.Add(Cosi);
end;
  
```

 Čo sa zmení, ak vo výraze `random(26)+65` nahradíme číslo 65 číslom 97?

Posuňme sa v našich vedomostiach trochu ďalej, nebude to však bezbolestne 😊.

Príklad 3.1.7

Vytvorte program na výpočet súčtu všetkých prirodzených čísel od 1 po zadané N ($N \geq 0$).

Poznámka: Ak poznáte vzorec na výpočet takéhoto súčtu, ten máte povolené použiť len na kontrolu nášho výpočtu 😊.

Analýza:

- Akú algoritmickú konštrukciu treba použiť?
 Napríklad pre $N = 5$ treba sčítať čísla $1 + 2 + 3 + 4 + 5$. Vidíme, že sa opakuje sčítanie. Keďže opakovane treba vykonávať nejaký príkaz, musíme použiť **cyklus**. Vieme určiť aj počet opakovaní príkazov v cykle, preto môžeme použiť príkaz `for`.
- Akou hodnotou začína sčítanie?
 Zamyslime sa nad počiatočnou hodnotou súčtu. Skôr, než začneme sčítovať, sme ešte nič nesčítali a preto je logické, že súčet má hodnotu 0!
- Koľkokrát sa má opakovať príkaz v cykle?
 Teda v skutočnosti treba sčítať čísla
 $0 + 1 + 2 + 3 + 4 + 5$, všeobecnejšie $0 + 1 + 2 + 3 + \dots + N$
 t.j. treba vykonať N sčítaní a k priebežnému súčtu, ktorý začína nulou, pričítovať postupne čísla 1, potom 2, 3, 4 až N, čo vedie k príkazu `for Cislo := 1 to N do...`
 Premenná `Cislo` bude postupne nadobúdať hodnoty 1, 2, 3 až N.
- Aký príkaz sa má v cykle opakovať?
 Čo sa „tam“ deje - simulácia výpočtu:

	nový Sucet		predchádzajúci Sucet		Cislo
1. opakovanie	1	←	0	+	1
2. opakovanie	3	←	1	+	2
3. opakovanie	6	←	3	+	3
4. opakovanie	10	←	6	+	4
5. opakovanie	15	←	10	+	5

Vyzerá to tak, že vždy k predchádzajúcemu súčtu, na začiatku k nule (0), potom k 1 (0+1), k 3 (1+2), k 6 (3+3), k 10 (6+4),... treba pripočítať vhodné číslo, čím dostaneme nový súčet.

Vhodné číslo je pri prvom prechode cyklom 1, pri druhom 2, pri treťom 3 atď., až pri n-tom N.

Práve tieto hodnoty nadobúda riadiaca premenná cyklu `for` – premenná `Cislo`.

Preto: nový čiastočný Sucet = predchádzajúci čiastočný Sucet + Cislo


Ak dobre poznáme vykonanie príkazu priradenia, uvedomíme si, že ten presne pracuje s „novým a predchádzajúcim“, ak na jeho ľavej aj pravej strane je tá istá premenná (to isté pamäťové miesto)!

Preto v cykle treba opakovať príkaz: `Sucet := Sucet + Cislo;`

V príkaze takéhoto typu musíte neustále vidieť: nový `Sucet` = predchádzajúci `Sucet` + „čosi“ alebo všeobecne: nová HODNOTA = stará HODNOTA operátor a „čosi“.

```
procedure TForm1.btSucetClick(Sender: TObject);
var Cislo, N, Sucet: integer;
begin
N:= StrToInt (Edit1.Text);           // zadanie hodnoty N
Sucet:= 0;
for Cislo:= 1 to N do Sucet:= Sucet + Cislo;
Label1.Caption:= 'Súčet prirodzených čísel je ' + IntToStr(Sucet);
end;
```

Poznámka: Na výpočet súčtu prvých n prirodzených čísel možno použiť aj vzorec $SUCET = \frac{N}{2} \cdot (1+N)$, preto sme v zadaní úlohy uviedli, že chceme výpočet pomocou cyklu. Zároveň si uvedomte, že výrazne sa efektivitou výpočtu líšiacich algoritmov môže byť (a väčšinou aj býva) viacej!

 Program upravte tak, aby počítal súčet všetkých prirodzených čísel od zadaného A po zadané B vrátane.

Príklad 3.1.8

Vytvorte program na výpočet mocniny x^n , x reálne, rôzne od nuly, n prirodzené číslo vrátane nuly.


Analýza:


Začneme „sedliackym“ rozumom. Ak máme vypočítať napríklad 5^3 , násobíme $5 \cdot 5 \cdot 5 = 125$. Ďalej by sme mohli vedieť, že 0^0 nie je definované (asi preto, že 0 na „čokoľvek“ je 0 a „čokoľvek“ na 0-tú je 1, preto 0^0 by malo byť 0 a zároveň 1 a to nejde).


1. Opakovane sa vykonáva násobenie, preto treba použiť cyklus;
2. Násobenie treba vykonať známy početkrát, preto cyklus s pevným počtom opakovaní, teda príkaz `for`.
3. Vždy nový súčin sa rovná predchádzajúci súčin krát X , preto sa v cykle bude opakovať príkaz `Sucin := Sucin * X`.
4. Nesmieme zabudnúť na počiatočnú hodnotu premennej `Sucin`, ktorá musí byť 1. Keby sme zvolili nulu?


Podstatná časť programu:

```
procedure TForm1.btMocninaClick(Sender: TObject);
var i, N: integer;
    X, Sucin: real;
begin
X:= StrToFloat (Edit1.Text);           // zadanie základu mocniny x
N:= StrToInt (Edit2.Text);             // zadanie exponentu n
Sucin:= 1;
for i:= 1 to N do Sucin:= Sucin * X;
Label3.Caption:= FloatToStr(X) + ' na ' + IntToStr(N) + ' = ' + FloatToStr(Sucin);
end;
```

 Počíta náš program správne aj pre $N=0$ a ľubovoľné dovolené X , t.j. $X \neq 0$? Prečo je to tak?

 Počíta náš program správne aj pre $X=0$ a ľubovoľné dovolené N , t.j. $N > 0$? Prečo je to tak?

 Keďže už ovládáte príkaz `if`, doplňte procedúru tak, aby sa po zadaní `X=0` a `N=0` vypísalo Nedefinované!

 Použite komponent Memo na vypísanie medzivýsledkov umocňovania.

Príklad 3.1.9

Vytvorte program na výpočet aritmetického priemeru známeho počtu reálnych čísel.



Analýza:

Snád' všetci vieme, že aritmetický priemer sa vypočíta ako súčet daných čísel vydelený počtom čísel. Program si musí vypýtať:

1. počet čísel, z ktorých má vypočítať aritmetický priemer a
2. hodnoty jednotlivých čísel, pričom ich musí (prečo?) hneď sčítovať; pre Počet čísel si musí hodnotu na pričítanie vypýtať Počet-krát, takže cyklus s pevným počtom opakovaní – príkaz `for`.

```
procedure TForm1.btPriemerClick(Sender: TObject);
var Pocet, i: integer;
    Cislo, Sucet, Priemer: real;
begin
  Pocet:= StrToInt ( InputBox ( 'Vstup' , 'Počet čísel' , '10' ) );
  Sucet:= 0;
  for i:= 1 to Pocet do
  begin
    Cislo:= StrToFloat ( InputBox ( 'Vstup' , IntToStr(i) + '. číslo' , '' ) );
    Sucet:= Sucet + Cislo;
  end;
  Priemer:= Sucet / Pocet;           //príkaz možno vykonať, len ak Pocet ≠ 0
  Label1.Caption:= 'Priemer ' + IntToStr (Pocet) + ' čísel = ' + FloatToStr (Priemer);
end;
```

 V procedúre ošetríte možné delenie nulou, ak Počet bude nula, nevykoná sa výpočet priemeru!

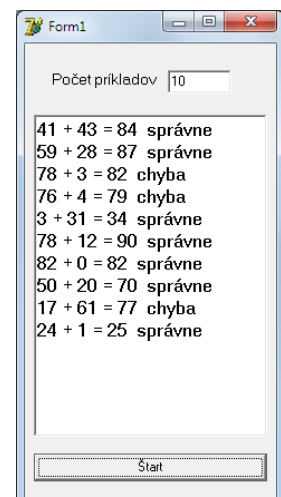
  Vytvorte program, ktorý, po zadaní počtu reálnych čísel, ich vygeneruje a vypíše do Mema a zistí ich súčet.

Príklad 3.1.10

Vytvorte program, ktorý po zadaní počtu príkladov vygeneruje dva sčítance, ktorých súčet nepresiahne 100 a opýta sa na výsledok. Po zadaní výsledku oznámi správne alebo chyba a vygeneruje ďalší príklad (obrázok vpravo).


Riešenie s komentárom

```
procedure TForm1.StartClick(Sender: TObject);
const MAXSUCET = 100;
var  Pocet, Cislo1, Cislo2, Vysledok, i: integer;
    Priklad: string;           // pomocná premenná – vytvára sa v nej riadok
begin
  randomize;
  Memo1.Clear;                // zmazanie Mema (začína nová sada príklad.)
  Pocet:= StrToInt (Edit1.Text); // zadaj počet príkladov
```




```
for i:= 1 to Pocet do          // vytvor a vypíš 1., 2., 3.,... príklad
begin
    Cislo1:= random (MAXSUCET);           // vygeneruj 1. sčítanca
    Cislo2:= random (MAXSUCET - Cislo1 + 1); // vygeneruj 2. sčítanca tak,
                                           // aby súčet ≤ MAXSUCET
    Priklad:= IntToStr (Cislo1) + ' + ' + IntToStr (Cislo2) + ' = ';
    Memo1.Lines.Add (Priklad);           // zobraz príklad
    Vysledok:= StrToInt (InputBox ( 'Cvičenie' , 'Výsledok' , " )); // výsledok?
    Priklad:= Priklad + IntToStr (Vysledok); // pridaj výsledok k príkladu
    Memo1.Lines.Delete(i-1);           // zmaž aktuálny riadok
    if Vysledok = Cislo1 + Cislo2       // ak je výsledok správny
    then Memo1.Lines.Add (Priklad + ' správne') // vypíš príklad s výsledkom + správne
    else Memo1.Lines.Add (Priklad + ' chyba'); // vypíš príklad s výsledkom + chyba
end;
end;
```

 Program upravte tak, aby sa opýtal na MAXSUCET, t.j. číslo, ktoré nemá súčet presiahnuť.

 Program prerobte na skúšanie malej násobilky (odčítania, delenia).

Vyhľadávanie v skupine údajov

Častou činnosťou v skupine údajov – zadaných hodnôt, je vyhľadávanie, t.j. činnosť s cieľom zistiť, či sa hodnota s požadovanou vlastnosťou nachádza v skupine dát, koľkokrát, na ktorých miestach,..., prípadne zistiť najväčšiu (najmenšiu) hodnotu v skupine, dve najmenšie (najväčšie) hodnoty a pod.

Aby sme nemuseli neustále zadávať z klávesnice údaje, na ktorých si overujeme naše vyhľadávacie algoritmy, prenecháme túto prácu počítaču.

Príklad 3.1.11

Vytvorte procedúru, ktorá po zadaní počtu celých čísel vygeneruje tieto, napríklad najviac dvojciferné, celé čísla a umožní ich spracovať.

Analýza

Program musí obsahovať zadanie počtu čísel užívateľom a cyklus, v ktorom budú postupne generované počítačom najviac dvojciferné celé čísla. Keďže zadáme počet čísel v skupine, môžeme pri generovaní čísel použiť for-cyklus. Pre lepší komfort použijeme premennú Riadok, do ktorej budeme postupne pridávať generované čísla oddelené medzerami, aby sme ich mohli, ako vytvorený reťazec, podľa potreby vypísať.

```
procedure TForm1.btSpracujClick(Sender: TObject);
var i, PocetHdnt, X: integer;
    Riadok: string;
begin
    PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt' , 'Počet čísel' , '10' ) );
    Riadok:= "";
    for i:= 1 to PocetHdnt do
    begin
        X:= random(100);           //do X uloží celé číslo z intervalu < 0 , 100 )
        Riadok:= Riadok + ' ' + IntToStr(X); //hodnotu X pridá, po medzere, do Riadok
        //spracovanie X
    end;
    Memo1.Lines.Add( Riadok ); // vypíše reťazec, v kt. sú uložené vygenerované čísla
```

```
//zobrazenie výsledku spracovania
end;
```

```
initialization // vykonanie príkazu randomize po spustení programu
randomize;
end.
```

Príklad 3.1.12

Vytvorte program, ktorý zistí, či sa hľadaná hodnota nachádza v práve generovanej postupnosti celých čísel.

Analýza

Procedúru z predchádzajúceho príkladu doplníme o zadanie hľadanej hodnoty a zistenie, či sa v generovanej postupnosti celých čísel nachádza. Použijeme dve nové premenné, Hladat, v ktorej je uložená hľadaná hodnota a Nasiel, ktorá je typu boolean a teda môže nadobúdať len dve hodnoty – False alebo True. Na začiatku predpokladáme, že sa hodnota uložená v Hladat, v postupnosti nenachádza (Nasiel:= False;). Ak počas generovania čísel nastane situácia, že vygenerovaná hodnota sa rovná hľadanej, zmeníme Nasiel z hodnoty False na True.

```
procedure TForm1.btVyskytClick(Sender: TObject);
var i, PocetHdnt, X, Hladat: integer;
    Riadok: string;
    Nasiel: boolean;
begin
PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt', 'Počet čísel', '10' ) );
Hladat:= StrToInt( InputBox( 'Skupina hodnôt', 'Hľadať číslo', '0' ) );
Nasiel:= False;
Riadok:= '';
for i:= 1 to PocetHdnt do
begin
    X:= random(100);
    Riadok:= Riadok + ' ' + IntToStr(X);
    if X = Hladat then Nasiel:= True;
end;
Memo1.Lines.Add(Riadok);
if Nasiel //Nasiel má hodnotu False alebo True, zápis je analogický s if Nasiel = True
then Memo1.Lines.Add( IntToStr( Hladat ) + ' sa v skupine vyskytuje')
else Memo1.Lines.Add( IntToStr( Hladat ) + ' sa v skupine nevyskytuje')
end;
```

Príklad 3.1.13

Vytvorte program, ktorý zistí, či sa hľadané meno nachádza v zadanej skupine mien.

Analýza

Príklad je analógiou predchádzajúceho príkladu, zmenili sme len údajový typ hodnôt z integer na string. Keďže zmysluplné mená by sa ťažko počítaču generovali, musí ich zadať užívateľ. Všimnite si, že algoritmus sa prakticky nezmenil.

```
procedure TForm1.btVyskytMenaClick(Sender: TObject);
var i, PocetHdnt: integer;
    Meno, Hladat, Riadok: string;
    Nasiel: boolean;
```

```
begin
PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt', 'Počet mien', '10' ) );
Hladat:= InputBox( 'Skupina hodnôt', 'Hľadať meno', '' );
Nasiel:= False;
Riadok:= '';
for i:= 1 to PocetHdnt do
begin
Meno:= InputBox( 'Skupina hodnôt', 'Zadaj slovo', '' );
Riadok:= Riadok + ' ' + Meno;
if Meno = Hladat then Nasiel:= True;
end;
Memo1.Lines.Add( Riadok );
if Nasiel
then Memo1.Lines.Add( Hladat + ' sa v skupine vyskytuje' )
else Memo1.Lines.Add( Hladat + ' sa v skupine nevyskytuje' )
end;
```

Príklad 3.1.14

Vytvorte program, ktorý zistí, koľkokrát sa hľadaná hodnota nachádza v generovanej postupnosti celých čísel.

Riešenie

Riešenie uvádzame bez podrobnejšieho opisu. Premenná PocetVy eviduje počet výskytov hľadanej hodnoty v generovanej postupnosti čísel; zrejme pred generovaním čísel má hodnotu nula.

```
procedure TForm1.btPocetClick(Sender: TObject);
var i, PocetHdnt, X, Hladat, PocetVy: integer;
    Riadok: string;
begin
PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt', 'Počet čísel', '10' ) );
Hladat:= StrToInt( InputBox( 'Skupina hodnôt', 'Hľadať číslo', '0' ) );
PocetVy:= 0;
Riadok:= '';
for i:= 1 to PocetHdnt do
begin
X:= random(100);
Riadok:= Riadok + ' ' + IntToStr(X);
if X = Hladat then PocetVy:= PocetVy + 1;
end;
Memo1.Lines.Add(Riadok);
if PocetVy > 0
then Memo1.Lines.Add( IntToStr( Hladat ) + ' sa v skupine vyskytuje ' + IntToStr( PocetVy ) + '-krát' )
else Memo1.Lines.Add( IntToStr( Hladat ) + ' sa v skupine nevyskytuje' )
end;
```

Príklad 3.1.15

Vytvorte program, ktorý vypíše miesta (poradové čísla) výskytov hľadanej hodnoty v generovanej postupnosti celých čísel.

Analýza

Napríklad v postupnosti: 5 2 0 4 7 0 2 sa hodnota 0 nachádza na 3. a 6. mieste v postupnosti, teda očakávame výstupný reťazec: 3 6.

Ak sa hľadaná hodnota nevyskytuje v postupnosti, nech sa vypíše: Hľadaná hodnota sa nevyskytuje v zadanej postupnosti.

V poslednom príkaze procedúry nižšie sme použili riadiace znaky #13 a #10, čo je kód klávesu ENTER, ako sa môžete presvedčiť z výpisu po spustení programu.

```

procedure TForm1.btMiestaClick(Sender: TObject);
var i, PocetHdnt, X, Hladat: integer;
    Riadok, Miesta: string;
begin
PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt' , 'Počet čísel' , '10' ) );
Hladat:= StrToInt( InputBox( 'Skupina hodnôt' , 'Hľadať číslo' , '0' ) );
Riadok:= '';
Miesta:= '';
for i:= 1 to PocetHdnt do
begin
    X:= random(100);
    Riadok:= Riadok + ' ' + IntToStr(X);
    if X = Hladat then Miesta:= Miesta + ' ' + IntToStr(i);
end;
if Miesta <> ''
then Memo1.Lines.Add( 'Miesta výskytov hľadanej hodnoty: ' + Miesta )
else Memo1.Lines.Add( 'Hľadaná hodnota sa nevyskytuje v zadanej postupnosti.' );
Memo1.Lines.Add( 'Zadaná postupnosť čísel:' + #13#10 + Riadok );
end;

```

Príklad 3.1.16

Vytvorte program, ktorý nájde najväčšiu hodnotu v generovanej postupnosti celých čísel.

Analýza

Nájsť najväčšiu hodnotu v skupine hodnôt znamená na začiatku vyhľadávania si „povedať“, že najväčšou hodnotou je hodnota prvého čísla. Potom sa „pozrieť“ na druhé číslo, ak je väčšie, zapamätať si ho ako priebežnú najväčšiu hodnotu, ak nie je väčšie, jeho hodnota nás nezaujíma. Potom sa „pozrieť“ na tretie číslo, ak je väčšie, zapamätať si ho ako novú priebežnú najväčšiu hodnotu, ak nie je väčšie, jeho hodnota nás opäť nezaujíma atď. až po posledné číslo v postupnosti. Ak premennú, do ktorej ukladáme aktuálne najväčšiu hodnotu, nazveme Max, ešte pred cyklom musíme do nej uložiť hodnotu prvého čísla, ako počiatočnú hodnotu premennej Max. V cykle for preto skúmame už až druhé, tretie až posledné číslo.

```

procedure TForm1.btMaximumClick(Sender: TObject);
var i, PocetHdnt, X, Max: integer;
    Postupnost: string;
begin
PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt' , 'Počet čísel' , '10' ) );
Max:= random(100);
Postupnost:= IntToStr(Max);
for i:= 2 to PocetHdnt do
begin
    X:= random(100);
    Postupnost:= Postupnost + ' ' + IntToStr(X);
    if X > Max then Max:= X;
end;
Memo1.Lines.Add( Postupnost );

```

```
Memo1.Lines.Add( 'Najväčšia hodnota je ' + IntToStr( Max ) );  
end;
```

Najčastejšou chybou pri hľadaní maxima alebo minima je určenie počiatocnej hodnoty. Chybou by bolo napríklad ako počiatocnú hodnotu určiť nulu, veď postupnosť by mohla obsahovať len záporné čísla, a už by bol výsledok zlý. Prakticky by sme mohli uvažovať o počiatocnej hodnote $-\text{MaxInt}$ pri hľadaní maxima a MaxInt pri hľadaní minima. Tá je však špecifická len pre typ integer. Najvšeobecnejšie je priradenie prvej hodnoty ako počiatocnej hodnoty.

Príklad 3.1.17

Vytvorte program, ktorý nájde dve najmenšie hodnoty (nemusia byť rôzne) zo skupiny minimálne dvoch celých čísel.

Algoritmus


Dohovor: Nech platí, že v Min1 je najmenšia hodnota a v Min2 druhá najmenšia hodnota, ak sú v skupine dve rovnaké najmenšie hodnoty, bude v Min1 aj Min2 rovnaká hodnota. Priradenie počiatocných hodnôt znamená, že napríklad do Min1 vložíme prvú hodnotu a do Min2 druhú hodnotu. Ak je hodnota v Min1 väčšia ako v Min2 , vymeníme ich hodnoty, aby platil dohovor. Pri skúmaní ďalších čísel nás zaujímajú len tie čísla, ktoré sú menšie ako aspoň Min2 . Ak sú menšie len ako Min2 , našli sme novú hodnotu Min2 . Ale ak je číslo menšie aj ako Min1 , našli sme novú hodnotu Min1 a zároveň posledná hodnota Min1 sa stáva novou hodnotou Min2 .

Algoritmicky:


```
ak X < Min2  
tak začiatok  
    ak X < Min1  
    tak začiatok  
        Min2:= Min1; Min1:= X;  
    koniec  
inak Min2:= X;  
koniec
```

```
procedure TForm1.btDveMinimaClick(Sender: TObject);  
var i, PocetHdnt, X, Min1, Min2, Pom: integer;  
    Postupnost: string;  
begin  
    PocetHdnt:= StrToInt( InputBox( 'Skupina hodnôt', 'Počet čísel (minimálne dve!)', '10' ) );  
    Min1:= random(100); Min2:= random(100);  
    if Min1 > Min2  
    then begin  
        Pom:= Min1; Min1:= Min2; Min2:= Pom;  
    end;  
    Postupnost:= Format( '%d %d', [ Min1, Min2 ] );  
    for i:= 3 to PocetHdnt do  
    begin  
        X:= random(100);  
        Postupnost:= Postupnost + ' ' + IntToStr(X);  
        if (X < Min2) and (X < Min1) //analogický zápis ako v algoritme  
        then begin  
            Min2:= Min1; Min1:= X;  
        end  
        else if (X < Min2) then Min2:= X;  
    end;  
    Memo1.Lines.Add( Postupnost );
```


```
Memo1.Lines.Add( 'Dve najmenšie hodnoty sú ' + Format( '%d a %d' , [ Min1, Min2 ] ) );
end;
```

☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet žiakov v triede a následne si vypýta príslušný počet študijných priemerov žiakov triedy (reálne čísla). Po zadaní uvedených údajov vypíše:


Spracovaných žiakov: // číslo udávajúce počet žiakov v triede
 Priemer triedy: // reálne číslo, aritmetický priemer zo zadaných priemerov
 Poznámka: Ak bol počet žiakov zadaný nula, nech sa priemer triedy nepočíta (delenie nulou)!

☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet položiek nákupu a následne si vypýta platby (celé čísla) za jednotlivé položky nákupu (napr. 0,60 € za časopis, 1 € za reďkvičku atď.).


Po zadaní platieb za všetky položky nákupu vypíše:
 Počet spracovaných položiek: // celé číslo
 Celková cena za nákup: // celé číslo, súčet platieb za jednotlivé položky nákupu

☞  Vytvor program, ktorý spraví sumár tvojich príjmov a výdavkov po zadaní počtu položiek. Napr. doma som dostal 10 € (+), kúpil som si žen'fu za 0,80 € (-), kúpil som si Colu za 1 € (-), kamarát mi vrátil požičaných 5 € (+) a kúpil som si lístok do kina za 3 €(-). Program po zadaní všetkých položiek vypíše:


Príjmy spolu: ... // súčet peňazí, ktoré som dostal (prijal)
 Výdaje spolu: ... // súčet peňazí, ktoré som zaplatil (vydal)
 V našom prípade: Príjmy: 15,00
 Výdaje: 4,80

☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet žiakov v triede a následne si vypýta výšku 1.žiaka, 2.žiaka,... (celé čísla, výška v cm). Po zadaní uvedených údajov vypíše:


Spracovaných žiakov: // číslo udávajúce počet žiakov v triede
 Priemerná výška: // reálne číslo, aritmetický priemer zo zadaných výšok
 Poznámka: Ak bol počet žiakov nula, nech sa priemerná výška nepočíta (delenie nulou)!

☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet žiakov v triede a následne si vypýta výšku 1.žiaka, 2.žiaka,... (celé čísla, výška v cm). Po zadaní uvedených údajov vypíše:


Výška najvyššieho žiaka: // maximum zo zadaných výšok
 Výška najnižšieho žiaka: // minimum zo zadaných výšok

☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet žiakov v triede a hraničnú hmotnosť. Následne si vypýta hmotnosť 1.žiaka, 2.žiaka,... (celé čísla, hmotnosť v kg). Po zadaní uvedených údajov vypíše:

Hraničnú a väčšiu hmotnosť má žiakov: // číslo udávajúce počet žiakov, ktorých
 // hmotnosť je rovná alebo vyššia ako zadaná hraničná hmotnosť


☞  Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet žiakov v triede a následne si vypýta výšku a hmotnosť 1.žiaka, 2.žiaka,... (celé čísla, výška v cm, hmotnosť v kg). Po zadaní výšky a hmotnosti žiaka, ak má žiak nadváhu, program okamžite vypíše:

Nadváha ... kg!
 Poznámka: žiak má nadváhu, ak jeho hmotnosť je väčšia ako jeho výška - 100.
 Napr. po zadaní dvojice čísel 175 a 85 program musí vypísať: Nadváha 10 kg!

 Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet tried v škole a následne si vyžiada počet žiakov v 1., 2., 3.,... triede. Po zadaní počtu žiakov v poslednej triede program vypíše:


Škola má žiakov: ... //celé číslo, súčet všetkých žiakov školy

V triede je priemerne žiakov: //celé číslo(!), aritmetický priemer počtu žiakov

 Vytvorte program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet tried v škole a následne si vyžiada počet chlapov a dievčat v 1., 2., 3.,... triede. Po zadaní počtu žiakov v poslednej triede program vypíše:

Škola má žiakov: ... //celé číslo, súčet všetkých žiakov školy


Z toho je dievčat: //celé číslo, súčet všetkých dievčat

 V škôlke sa rozhodli postaviť si máj. Každé dieťa donieslo farebnú stužku (modrú, bielu alebo červenú). Vytvorte program, ktorý vypočíta súčet dĺžok prinesených stužiek jednotlivých farieb. Program sa môže opýtať na počet stužiek, farbu stužky a jej dĺžku. Na záver program vypíše:

Biele stužky: ... cm

Modré stužky: ... cm


Červené stužky: ... cm

 V škôlke sa rozhodli postaviť si máj. Každé dieťa donieslo farebnú stužku (modrú, bielu alebo červenú). Vytvorte program, ktorý vypíše maximálne dĺžky stužiek jednotlivých farieb. Program sa môže opýtať na počet stužiek, farbu stužky a jej dĺžku. Na záver program vypíše:

Najdlhšia modrá stužka mala ... cm

Najdlhšia biela stužka mala ... cm

Najdlhšia červená stužka mala ... cm

 Turistickú trasu som si rozdelil na úseky. Vytvor program, ktorý sa, po kliknutí na tlačidlo Spracuj, opýta na počet úsekov a následne na dĺžku a čas potrebný na prejsenie 1., 2., 3. až posledného úseku (dĺžka - celé číslo v km, čas - celé číslo v min.). Na záver nech program vypíše:

Celkove prejdeš km:

Trasa bude trvať min.: ...

Priemerná rýchlosť v km/hod:

Upozornenie

Neuviedli sme všetky tvary a možnosti príkazu for. Jeho všeobecný tvar sa dozviete v treťom ročníku vo voliteľnom predmete informatika, alebo si ho môžete pozrieť v úvode druhého dielu tejto zbierky, v helpe vývojového prostredia Delphi, prípadne na internete.

S príkazom for by sme nevystačili pri riešení problémov, pri ktorých nepoznáme dopredu počet opakovaní príkazov v cykle alebo keď riadiaca premenná cyklu nenadobúda „pekne“, v našom prípade to znamená celočíselné, hodnoty. Vtedy musíme použiť cyklus s podmienkou, t.j. jeden z príkazov while alebo repeat.

Na precvičenie for-cyklu možno použiť aj príklady z časti **Pohrajme sa s reťazcami**, konkrétne príklady 3.4.2 až 3.4.6, 3.4.8, 3.4.9, 3.4.12 a 3.4.14.

Cyklus s podmienkou ukončenia

Pri neznámom počte opakovaní príkazov v cykle je nevyhnutné použitie cyklu s podmienkou ukončenia. Môžeme využiť while-cyklus alebo repeat-cyklus.

Príkaz while

- použijeme, ak môže nastať situácia, že sa príkazy v cykle nemajú vykonať ani raz
- má tvar: **while Podmienka do**
begin
Príkazy, ktoré sa majú opakovane vykonávať
end;
- tak ako v prípade for-cyklu, ak za vyhradeným slovom do sa má opakovať len jeden príkaz, nemusí byť umiestnený medzi begin a end
- napríklad:
Cislo:= 10; while Cislo >= 0 do Cislo:= Cislo - 2; príkaz za „do“ sa vykoná 6-krát pre Cislo s hodnotami 10, 8, 6, 4, 2 a 0
Cislo:= 100; while Cislo > 0 do Cislo := Cislo div 10; príkaz v cykle sa vykoná 3-krát pre Cislo s hodnotami 100, 10 a 1
Cislo:= 0; while Cislo > 0 do Cislo:= Cislo -1; príkaz v cykle sa nevykoná ani raz
Cislo:= 9; while Cislo <> 0 do Cislo:= Cislo -2; nekonečný cyklus (zacyklenie), pretože Cislo nikdy nenadobudne hodnotu nula
A:= 5; B:= 5; while A<>B do ... príkaz za „do“ sa nevykoná ani raz
- vykonanie while-cyklu:
Pokiaľ je podmienka splnená, opakovane sa vykonáva príkaz za „do“ (príkaz v cykle). Ak podmienka nie je splnená, while-cyklus sa ukončí.

Príklad 3.2.1

Vytvorte program na výpočet súčtu všetkých prirodzených čísel od 1 po zadané N ($N \geq 0$).

Analýza

Tento problém sme riešili v príklade 3.1.2 pomocou for-cyklu. Riešenie si ukážeme aj pomocou while-cyklu, aj keď vieme určiť počet opakovaní príkazov v cykle (N). Príkaz while je univerzálnejší ako for, všade tam, kde môžeme použiť for-cyklus, vieme použiť aj while-cyklus. Takže riešenie „bez dlhých rečí“:

```

procedure TForm1.btSucetClick(Sender: TObject);
var Cislo, N, Sucet: integer;
begin
N:= StrToInt (Edit1.Text);                      // zadanie hodnoty N
Sucet:= 0;
Cislo:= 1;
while Cislo <= N do
begin
    Sucet:= Sucet + Cislo;
    Cislo:= Cislo + 1;
end;
Label1.Caption:= 'Súčet prirodzených čísel je ' + IntToStr(Sucet);
end;
  
```


Uvedomte si, že príkaz for skrakuje zápis konštrukcie:

rpc := <i>ZačiatočnáHodnota</i> ;	Cislo:= 1;
while rpc <= <i>KoncováHodnota</i> do	while Cislo <= N do
begin	begin
príkaz;	Sucet:= Sucet + Cislo;
rpc:= rpc + 1;	Cislo:= Cislo + 1;
end;	end;
na zápis	for rpc:= <i>ZačiatočnáHodnota</i> to <i>KoncováHodnota</i> do príkaz;
resp.	for Cislo:= 1 to N do príkaz;

Príklady 3.1.3 a 3.1.4 riešte pomocou while-cyklu.

Príklad 3.2.2

Vytvorte program na výpočet ciferného súčtu zadaného prirodzeného čísla.

Analýza

Skôr, než začnete študovať analýzu, zopakujte si použitie funkcií div a mod (úvod zbierky, strana 13, príklad 1.5)!

Ciferný súčet napríklad čísla 123 je 6 (1+2+3). Rovnaký výsledok zrejme dostaneme, aj keď cifry čísla budeme sčítavať v opačnom poradí, teda 3+2+1. Opačné poradie nám vyhovuje preto, že sa „jednoducho“ vieme dostať k poslednej cifre (k jednotkám) čísla pomocou funkcie mod (123 mod 10 = 3). Keď sme už cifru spracovali, z čísla 123 nás zaujíma už len číslo 12, ku ktorému sa vieme dostať pomocou funkcie div (123 div 10 = 12). A opäť môžeme celý proces zopakovať pre číslo 12. Po spracovaní poslednej (v poradí pôvodného čísla prvej) cifry zostane vždy z čísla 0 (1 div 10 = 0), čo môžeme využiť na ukončenie while-cyklu. For-cyklus nemôžeme použiť, pretože nevieme jednoducho určiť počet cifier v zadanom čísle.

Algoritmus

```
začiatok
píš („Zadaj číslo, ktorého ciferný súčet chceš vypočítať“); čítaj (Cislo);
Sucet ← 0;
pokiaľ Cislo > 0 opakuj
zaciatok
    Cifra ← Cislo mod 10;
    Sucet ← Sucet + Cifra;
    Cislo ← Cislo div 10;
koniec;
píš (Sucet);
koniec;
```

Podstatná časť programu

```
procedure TForm1.CifSucetClick(Sender: TObject);
var Cislo, Cifra, Sucet: integer;
begin
Cislo:= StrToInt(Edit1.Text); // zadanie čísla, ktorého ciferný súčet sa má vypočítať
Sucet:= 0;
while Cislo > 0 do
begin
    Cifra:= Cislo mod 10;
    Sucet:= Sucet + Cifra;
    Cislo:= Cislo div 10;
end;
```

```
Label2.Caption:= IntToStr(Sucet);
end;
```

☞ Počíta program správne aj po zadaní nuly?

Príklad 3.2.3

Vytvorte program, ktorý zo zadaného prirodzeného čísla spraví opačné – vymení cifry v čísle, napríklad z čísla 123 spraví číslo 321.

Algoritmus

Zrejme bude podobný algoritmu z príkladu 3.2.2, keďže problémy sú veľmi príbuzné. Opäť sa potrebujeme dostať k cifrám čísla, tentoraz ich však nesčítujeme, ale „skladáme“ v opačnom poradí. Počet cifier vopred nevieme určiť preto cyklus s podmienkou ukončenia; ak zadáme nulu, príkazy v cykle sa nemajú vykonať ani raz preto while-cyklus.

Po získaní cifry 3 z čísla 123, potrebujeme ju posunúť v opačnom čísle na miesto desiatok, aby sme mohli pripočítať ďalšiu cifru – dvojku. To ľahko docielime vynásobením 10 ($3 \cdot 10 = 30$); pripočítaním 2 dostávame 32 (čiastočne opačné číslo). Opätovným vynásobením 10 a pripočítaním poslednej cifry dostávame $32 \cdot 10 + 1 = 321$ opačné číslo.

Schematicky

	Cislo	Cifra	Opacne	nové Cislo
počiatočné hodnoty	123	-	?	-
1. prechod cyklom	123	3	$? \cdot 10 + 3 = ? + 3 = 3$	12
2. prechod	12	2	$3 \cdot 10 + 2 = 30 + 2 = 32$	1
3. prechod	1	1	$32 \cdot 10 + 1 = 320 + 1 = 321$	0
výsledok	0	-	321	-

Otázkou je počiatočná hodnota opačného čísla – premenná Opacne. Zrejme to bude nula, keďže pre ňu dostávame $0 \cdot 10 + 3 = 0 + 3 = 3$ správnu hodnotu. Cyklus končí, keď spracovávané číslo má veľkosť nula.

Podstatná časť programu:

```
procedure TForm1.OpacneClick(Sender: TObject);
var Cislo, Opacne, Cifra: integer;
begin
Cislo:= StrToInt(Edit1.Text);
Opacne:= 0;
while Cislo > 0 do
begin
    Cifra:= Cislo mod 10;
    Opacne:= Opacne*10 + Cifra;
    Cislo:= Cislo div 10;
end;
Label2.Caption:= IntToStr(Opacne);
end;
```

Príklad 3.2.4

Vytvorte program, ktorý po zadaní dvoch prirodzených čísel zistí najväčšie celé číslo, ktoré delí obe zadané čísla.

Analýza

Ide o matematický problém, nájdenie najväčšieho spoločného deliteľa dvoch prirodzených čísel napríklad a a b, označovaného aj NSD(a,b).

Dá sa dokázať, že NSD dvoch čísel možno určiť opakovaným odčítaním väčšieho čísla od menšieho, až kým nedostaneme rovnaké čísla, t.j. napríklad pre 21 a 15 platí: $NSD(21,15) = NSD(6,15) = NSD(6,9) = NSD(6,3) = NSD(3,3) = 3$. Opakuje sa odčítanie preto použitie cyklu;

keďže nevieme, kedy a a b nadobudnú rovnaké hodnoty preto cyklus s podmienkou ukončenia; ak zadáme dve rovnaké čísla, nemá „prebehnúť“ ani jedno odčítanie (nemá sa ani raz vykonať príkaz v cykle!) preto while-cyklus.

Algoritmus I.

```
začiatok
čítaj(a,b);
pokiaľ a<>b opakuj
    ak a>b tak a ← a-b
    inak b ← b-a;
píš (a);
koniec;
```

Algoritmov na výpočet NSD je viac. Najznámejší je Euklidov algoritmus, ktorý Euklides popísal vo svojej knihe Elementy okolo roku 300 pred našim letopočtom. Nie je však jeho autorom a predpokladá sa, že je ešte o 200 rokov starší. Euklidov algoritmus je najstarším netriviálnym algoritmom, ktorý sa stále v súčasnosti používa. Pravdepodobne najznámejšia jeho verzia využíva funkciu mod.

Algoritmus II.

```
začiatok
čítaj(a,b);
pokiaľ (a≠0) a (b≠0) opakuj
    ak a>b tak a ← a mod b
    inak b ← b mod a;
ak a>0 tak píš (a)
inak píš (b);
koniec;
```

Pre $a = 21$ a $b = 15$ by výpočet podľa algoritmu II. nadobúdal hodnoty $\text{NSD}(21,15) = \text{NSD}(6,15) = \text{NSD}(6,3) = \text{NSD}(0,3) = 3$.

☛ Vetvenie ak $a > 0$ tak...inak...; možno nahradiť jedným príkazom priradenia. Prídete na to, akým?

📖 Oba algoritmy použite v programoch.

🔗 Čo sa „deje“ v nasledujúcej procedúre?

```
procedure TForm1.Button1Click(Sender: TObject);
var a, b, pom: integer;
begin
a:= StrToInt (Edit1.Text);
b:= StrToInt (Edit2.Text);
while (a<>0) do
begin
    if a < b
    then begin
        pom:= a; a:= b; b:= pom;
    end;
    a:= a mod b;
end;
Label3.Caption:= 'NSD = ' + IntToStr(b);
end;
```

Príkaz repeat

- použijeme, ak sa príkazy v cykle majú vykonať aspoň raz
- má tvar: **repeat**

Príkazy, ktoré sa majú opakovane vykonávať

until *Podmienka;*

- keďže príkazy, ktoré sa majú opakovať, sú uzavreté medzi repeat a until, nepoužívajú sa vyhradené slová begin a end

- napríklad:

```
Cislo:= 0;
```

```
repeat
```

```
    Cislo:= Cislo + 1
```

```
    // príkaz v cykle sa vykoná 10-krát
```

```
until Cislo = 10;
```

```
repeat
```

```
    Príkazy;
```

```
    // príkazy sa vykonávajú, kým nepotvrdíme N
```

```
    Odpoved:= InputBox ('Cyklus' , 'Opakovať (Nie -> N) ?' , 'N');
```

```
until Odpoved = 'N';
```

```
    // premenná Odpoved je typu string!
```

```
Cislo:=1;
```

```
repeat
```

```
    Cislo:= Cislo + 1
```

```
    // príkaz sa vykoná práve raz
```

```
until Cislo > 1;
```

- vykonanie repeat-cyklu:

Vykonajú sa príkazy v cykle a opakovane sa budú vykonávať, pokiaľ podmienka nenadobudne hodnotu pravda (true).

Nesplnenie podmienky znamená opakovanie príkazov v repeat-cykle.

Príklad 3.3.1

Vráťme sa k príkladu 3.1.4 – simulácia hodu hracou kockou. Pravidlá hovoria, že ak padne šestka, hráč hod opakuje. Doplňte program 3.1.4 o toto pravidlo.

Analýza

Zrejme platí: ak padla 6, opakuj hod. Toto však nie je celkom pravda, lebo z algoritmickeho hľadiska slovo „ak“ znamená vetvenie a teda len jedenkrát vykonanie nového hodu! Čo však, ak za 6 padne opäť 6. Teda do programu treba „dorobiť“ cyklus, a to s neznámym počtom opakovaní preto cyklus s podmienkou ukončenia, keďže nevieme, kedy skončí „padanie šestiek“. Uskutočniť sa má aspoň jeden hod, preto je vhodný repeat-cyklus.

Pri vygenerovaní nového čísla po 6 sa pôvodné v komponente Label prepíše, preto je nevyhnutné použiť komponent Memo a do neho vypisovať jednotlivé hody, čím sa žiaden hod nestratí presnejšie neprepíše.

Nová verzia programu si vyžaduje použitie premennej Padla na riadenie cyklu.

Finálna verzia:

```
procedure TForm1.HodClick(Sender: TObject);
```

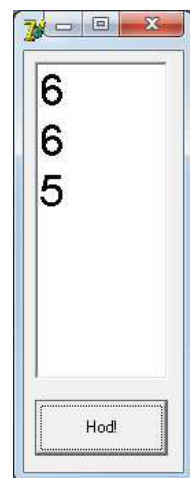
```
var Padla: integer;
```

```
begin
```


```
    randomize;
```

```
    Memo1.Clear;
```

```
    // zmaže Memo1 pred hodom nového hráča!
```



```
repeat
  Padla:= random(6)+1;
  Memo1.Lines.Add ( IntToStr (Padla) );
until Padla <> 6;
end;
```

 Manuálne zväčšíte písmo v komponente Memo cez jeho vlastnosť Font podobne, ako to je vidieť v ukážke.

Príklad 3.3.2

Naprogramujte hru Hádaj. Počítač nech vygeneruje náhodné celé číslo napríklad od 0 po 100. Hráč ho má uhádnuť. Zadáva tipy, počítač mu oznamuje Veľa, uber!, Málo, pridaj! alebo Uhádol si!


Analýza


Jednoznačne ide o cyklus s neznámym počtom opakovaní príkazov v cykle, s podmienkou na konci, teda repeat-cyklus. Zdôvodnenie je jednoduché. Hráč opakovane vkladá svoje tipy (cyklus), pričom nevieme, kedy uhádne číslo (s neznámym počtom opakovaní, cyklus s podmienkou ukončenia) a musí zadať aspoň jeden tip (s podmienkou na konci, repeat-cyklus). Výstup je asi vhodnejší do Memo, aby hráč mohol rekapitulovať svoje tipy.




Podstatná časť programu


```
procedure TForm1.HadajClick(Sender: TObject);
const MAXCISLO = 100; // definovanie konštanty MAXCISLO
var Uhadnut, Tip: integer;
begin
  randomize;
  Uhadnut:= random ( MAXCISLO + 1 ); // vygenerovanie čísla, ktoré treba uhádnuť
  Memo1.Lines.Add ( 'Vybral som celé číslo od 0 po ' + IntToStr ( MAXCISLO ) );
  Memo1.Lines.Add ( ' Hádaj!' );
  repeat
    Tip:= StrToInt ( InputBox ( 'Hra Hádaj' , 'Tip' , '' ) );
    if Tip <> Uhadnut
    then if Tip < Uhadnut
      then Memo1.Lines.Add ( 'Málo, pridaj!' )
      else Memo1.Lines.Add ( 'Veľa, uber!' )
    until Tip = Uhadnut;
  Memo1.Lines.Add ( 'U h á d o l s i ! ! !' );
end;
```

 Program doplňte o výpis, na koľký pokus sa podarilo číslo uhádnuť.

 Program doplňte o hodnotenie úspešnosti, napríklad pri uhádnutí do 6. pokusu nech vypíše Si špecialista atď.

 Program doplňte o možnosť zadania hľadaného čísla spoluhráčom cez komponent Edit. Aby sa v Edite nezobrazovali vkladané čísla, nastavte farbu písma na bielu.

   Ešte elegantnejším riešením predchádzajúceho problému je využitie vlastnosti PasswordChar komponentu Edit. Po vložení napríklad * (hviezdička), sa táto bude zobrazovať v Edite namiesto vkladaných znakov.

 Napríklad po vyriešení príkladu 3.1.5 (Limonádový Joe) sa pravdepodobne nik z nás neuspokojí s jedným výsledkom, ale bude v podstate simulovať cyklus opätovným stláčaním tlačidla Zisti 😊.

Opakovanie nejakého „programu“, podľa rozhodnutia užívateľa, možno ľahko zrealizovať repeat-cyklom, čo vlastne znamená príkazovú časť programu uzavrieť medzi slovami repeat a until. Keďže rozhodnutie o opakovaní má byť na užívateľovi, posledný príkaz v cykle by mal obsahovať otázku Opakovať program/výpočet/hru? a podľa odpovede sa buď zopakujú príkazy v repeat-cykle, alebo sa ukončí.

Schéma na opakovanie programu

```

...
var Odpoved: string;
begin
  príkazy, ktoré stačí vykonať raz, napríklad randomize;
repeat
  Príkazy programu
  Odpoved:= InputBox ('Schéma', 'Opakovať výpočet (Áno -> A)', 'A' )
until Odpoved <> 'A';
end;

```

☛ V podmienke repeat-cyklu sa testuje, či premenná Odpoved obsahuje veľké písmeno A. Keďže je predvolenou hodnotou v InputBoxe, stačí, keď ju užívateľ len potvrdí. Čo však, ak užívateľ vloží malé písmeno a. Mohlo by nám napadnúť použiť funkciu UpCase (Odpoved), na čo by však prekladač zahlásil chybu. Problém je v tom, že výsledkom InputBoxu nie je typ char ale typ string a pre ten funkcia UpCase nefunguje. Zmeniť v deklarácii (var ...) typ string na char tiež nie je riešením, „nepáčilo“ by sa to funkcii InputBox, ktorej výsledkom musí byť typ string. Ďalšie možnosti „nesprávnej“ odpovede sú Áno, áno, ano,... - tento problém si necháme na neskoršie riešenie ☺.

📄 Doplňte niektoré predchádzajúce programy o schému na opakovanie programu.

📄 Príklad 3.1.10 pozmeňte tak, aby sa po každom príklade opýtal: Ďalší príklad?

Príklad 3.3.3

Vráťme sa k príkladu 3.1.3, v ktorom sme zisťovali, či zadané prirodzené číslo je prvočíslo. Už vtedy sme upozornili, že algoritmus je neefektívny, keďže sa proces delenia nezastaví hneď, ako nájde deliteľa zadaného čísla ($1 < \text{deliteľ} < \text{zadané číslo}$). Použitý for-cyklus to neumožňoval, cyklus s podmienkou ukončenia je však niečo iné ☺.

Podstatná časť programu

```

procedure TForm1.JePrvocisloClick(Sender: TObject);
var Cislo, Delitel: integer;
begin
  Cislo:= StrToInt (Edit1.Text);
  if ( Cislo < 2 ) or (( Cislo > 2 ) and ( Cislo mod 2=0 ))
  then Label2.Caption:='Nie je prvočíslo'
  else begin
    Delitel:= 1;
    repeat
      Delitel:= Delitel + 1
    until Cislo mod Delitel = 0;
    if Delitel < Cislo
    then Label2.Caption:= 'nie je prvočíslo'
    else Label2.Caption:= 'je prvočíslo'
  end;
end;
end;

```

 Program doplňte o výpis deliteľa, ak zadané číslo nie je prvočíslo a je väčšie ako 2.

Príklad 3.3.4

Vytvorte program na výpočet aritmetického priemeru vopred neznámeho počtu reálnych čísel.

Analýza

Príklad 3.1.9 má podobné zadanie, v ňom je však počet čísel vopred známy. Pri neznámom počte čísel zrejme musíme použiť cyklus s podmienkou ukončenia. Ukončenie cyklu je podmienené vložením dohodnutej koncovkej hodnoty. Keďže minimálne jednu hodnotu treba zadať (môže byť hneď koncová) vyhovuje repeat-cyklus. While-cyklus by zrejme musel obsahovať jeden vstup nad príkazom cyklu.


Okrem sčítovania zadávaných čísel musíme sledovať aj počet zadaných čísel, aby sme mohli, po skončení cyklu, vypočítať aritmetický priemer zo zadaných čísel (samozrejme bez koncovkej hodnoty!).

Dohodnime sa, že v našom príklade bude koncovou hodnotou číslo nula.

Zaujímavosťou je, že formulár neobsahuje žiaden vstupný komponent Edit.

Podstatná časť programu

```
procedure TForm1.PocitajClick(Sender: TObject);
const KoncovaHodnota = 0;
var   PocetCisel: integer;
      Cislo, Sucet: real;
begin
Label1.Caption:= "";           // „vyčistiť“ výstup pri novom kliknutí na tlačidlo Počítaj
PocetCisel:= 0;
Sucet:= 0;
repeat
    Cislo:= StrToFloat ( InputBox ( 'Priemer' , 'Zadaj reálne číslo (Koniec -> 0) ' , '' ) );
    if Cislo <> KoncovaHodnota then begin inc(PocetCisel); Sucet:= Sucet + Cislo end;
until Cislo = KoncovaHodnota;
if PocetCisel > 0
then Label1.Caption:= 'Priemer = ' + FloatToStr ( Sucet/PocetCisel )
else Label1.Caption:= 'Nebolo zadané ani jedno platné číslo!'
end;
```

 Príklad 3.3.4 zrealizujte aj s while-cyklom.

Príklad 3.3.5

Limonádový Joe z príkladu 3.1.5 sa rozhodol spestriť si svoju cestu do baru. Keďže z domu do baru má presne 100 m, každý deň prejde polcestu (50 m od domu aj baru) a začne hádzať mincou. Ak padne rub, vráti sa 10 m k domu, ak líce, posunie sa 10 m k baru. Nakoniec skončí doma alebo v bare. Simulujte Joeovo hádzanie mincou a vytvorte program, ktorý vypíše, koľkokrát za rok skončil doma a koľkokrát v bare.

Analýza

Ako vyzerá jeden Joeov deň? Joe prejde z domu smerom k baru 50 m. Potom hodí mincou a posunie sa o 10 m k domu alebo k baru. Hádzanie opakuje, kým neskončí doma alebo v bare preto cyklus (opakuje sa hod mincou) s neznámym počtom opakovaní - nevieme, po koľkých hodoch skončí doma alebo v bare; musí uskutočniť niekoľko hodov (minimálne 5), preto môžeme použiť aj repeat-cyklus (nie je podmienkou).


Ako vyzerá celý Joeov rok? Do baru sa vyberie každý deň, preto treba simuláciu jedného dňa zopakovať 365-krát, preto cyklus s pevným počtom opakovaní, for-cyklus. Každý deň, keď skončí doma, treba zvýšiť hodnotu v premennej napríklad Doma o 1. Koľkokrát skončil za rok v bare je zrejmé, 365 - Doma.


Podstatná časť programu

```

procedure TForm1.ZistiClick(Sender: TObject);
var Doma, Den, Prejdene: integer;
begin
  randomize;
  Doma:= 0; // počet skončení doma, na začiatku je nula
  for Den:= 1 to 365 do // opakovať pre každý deň, t.j. 365-krát
  begin
    Prejdene:= 50; // začína v strede cesty
    repeat // opakuje hody mincou
      if random(2) = 0 then Prejdene:= Prejdene - 10 // padol rub
      else Prejdene:= Prejdene + 10 // padlo líce
    until (Prejdene = 0) or (Prejdene = 100); // hádzanie končí doma alebo v bare
    if Prejdene = 0 then Doma:= Doma + 1; // ak skončil doma, počet zvýšiť o 1
  end;
  Memo1.Lines.Add ('Doma skončil ' + IntToStr (Doma) + '-krát ');
  Memo1.Lines.Add ('V bare skončil ' + IntToStr (365-Doma) + '-krát ');
  Memo1.Lines.Add ('-----');
end;

```

 Program doplňte o zistenie, koľko metrov priemerne prešiel denne za celý rok (súčet všetkých prejdených metrov/365).

 V programe sa môže vyskytovať viacej cyklov. Takým príkladom je aj „Limonádový Joe“, kde sú dva cykly – for a repeat.

Cykly môžu byť **susedné**, t.j. jeden skončí a až potom začína druhý cyklus, napríklad

```

// vstup hodnôt pre výpočet mocniny x na n-tú
repeat
  X:= StrToFloat (InputBox (...));
  N:= StrToInt (InputBox (...));
  if (X=0) and (N=0) then ShowMessage ('0 na 0-tú nedefinované, skús znova!');
until (X<>0) or (N<>0); // negácia predchádzajúcej podmienky z if !

// výpočet mocniny pomocou cyklu
Sucin:= 1;
for i:= 1 to N do Sucin:= Sucin * X;
...
Najprv skončí cyklus repeat a až potom začína cyklus for.

```

Ak je do jedného - vonkajšieho cyklu vložený druhý cyklus - vnútorný, cykly sú **vnorené**. Ukážkou vnorených cyklov je práve „Limonádový Joe“. Konštrukcia vnútorného cyklu musí skončiť skôr, ako konštrukcia vonkajšieho cyklu! Situácie, keď vo vnútornom cykle skončí vonkajší cyklus, nie je dovolená. Teda napríklad nie je dovolená konštrukcia


```
repeat
    while podmienka2 do
        begin
            príkazy;
        until podmienka1;
    end; // while
```

Kedy ktorý cyklus?

Keď už máme základnú predstavu o fungovaní jednotlivých cyklov a im zodpovedajúcich príkazov, môžeme si poznatky trochu zosystematizovať:

- ak vieme počet opakovaní príkazov v cykle a premenná, ktorá riadi počet prechodov cyklom, sa môže meniť na nasledovníka (krok +1), použijeme for-cyklus
- ak nevieme počet opakovaní príkazov v cykle alebo premenná, ktorá riadi počet prechodov cyklom, nenadobúda „pekné“ hodnoty (hodnoty „+1“), použijeme while-cyklus alebo repeat-cyklus, pričom:
 - príkaz while, t.j. s podmienkou na začiatku použijeme, ak môže nastať situácii, že príkazy v cykle sa nemajú vykonať ani raz
 - príkaz repeat, t.j. s podmienkou na konci použijeme, ak sa príkazy v cykle majú vykonať aspoň raz
- while-cyklus je najuniverzálnejší, t.j. môže nahradiť for aj repeat-cyklus
- rozdiely vo vykonaní príkazov while a repeat:
 - pokiaľ je podmienka splnená, vykonávajú sa príkazy vo while-cykle; naopak v repeat-cykle, v ktorom, ak je podmienka splnená, cyklus sa ukončí
 - pri použití while-cyklu, ak sa má opakovane vykonávať viacej príkazov, musia byť uzavreté medzi begin a end; u repeat-cyklu to nie je potrebné
 - nedá sa napísať repeat-cyklus v ktorom by sa príkazy v cykle nemali vykonať ani raz; while-cyklus taký existuje.

Dovolené, ale prakticky nepoužiteľné, tvary cyklov:

```
for i:= 1 to 10 do; príkaz; // za „do“ je prázdny príkaz, 10-krát sa zopakuje „nič“, príkaz len 1x
while TRUE do príkaz; // nekonečný cyklus, ukončiť program možno Run – Program Reset
while FALSE do príkaz; // príkaz sa nevykoná ani raz
repeat príkazy until TRUE; // príkazy sa vykonajú práve raz
repeat príkazy until FALSE; // nekonečný cyklus
Test:= TRUE;
while Test do Test:= not Test; // príkaz v cykle sa vykoná 1x
Test:= TRUE;
repeat Test:= not Test until Test; // príkaz v cykle sa vykoná 2x
```

Uvedomte si, že v každom cykle musí byť premenná, ktorá riadi počet prechodov daným cyklom! Kontrolujte, či sa jej hodnoty zväčšujú alebo znižujú a či sú ohraničené podmienkou ukončenia cyklu, ináč cyklus nikdy neskončí.

CYKLY V ŽIVOTE


Ani bez cyklov sa nezaobídeme v živote, stačí, keď sa chceme niekde dostať pešo, ved' kráčať znamená opakovať krok striedavo pre ľavú a pravú nohu :)

Ďalšie príklady:


umývanie zubov:	pozri na hodiny; //počiatočný stav opakuj 10x pohni kefkou po zuboch zhora dole; (for-cyklus) 10x pohni kefkou na zuboch vľavo - vpravo; ...
	kým neprejdú 2 minúty; //odporúčané zubnými lekármi
prechod so semaforom:	pozri na semafor; //počiatočný stav pokiaľ svieti červená rob stoj a pozri na semafor; prejdi;
varenie:	rozbi vajíčka; //počiatočný stav opakuj miešaj; kým hmota nezhustne;
beh:	postav sa na štart; //počiatočný stav pokiaľ štartér nedal pokyn rob stoj; opakuj krok; kým nebudeš v cieľi;
uvoľnenie skrutky:	nasad' skrutkovač; //počiatočný stav pokiaľ je skrutka v materiáli rob otoč skrutkovač proti smeru chodu hodín;
hra Človeče nehnevaj sa!	hod' kockou; //počiatočný stav pokiaľ padla šestka rob hod' kockou; alebo opakuj hod' kockou; kým nepadne iné ako šestka; presnejšie súčet bodov je nula; //počiatočný stav opakuj hod' kockou; body pripočítaj k súčtu; kým nepadne iné ako šestka; presuň panáčika o súčet políčok;
platba:	opakuj podaj bankovku; kým si nezaplatil; ak si preplatil //vetvenie tak čakaj na vydanie inak postúp ďalej;


Pohrajme sa s reťazcami

Reťazec je skupina znakov uzavretá v apostrofoch. **Prázdny reťazec** sú len samotné apostrofy vložené hneď za sebou. Pri reťazcoch možno zisťovať ich dĺžku (počet znakov), meniť malé písmená na veľké a opačne, vložiť alebo vyrezat' podreťazec z reťazca, zmeniť číselný reťazec na číslo a naopak, zistiť, či reťazec obsahuje zadaný znak prípadne na ktorom mieste, zistiť, či je reťazec symetrický, vymeniť znaky v reťazci atď. Programovanie procedúr na prácu s reťazcami je vďačnou a poučnou problematikou umožňujúcou dobre si precvičiť cykly.

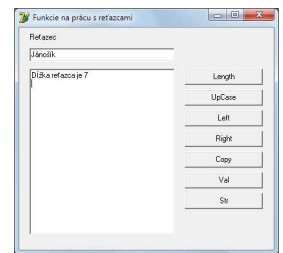
 Znaky v reťazcoch sú znaky z použitej kódovacej tabuľky.

Môžeme si pripomenúť (pozri príklad 3.1.7): znaky 0 (nula) až 9 idú za sebou a majú poradové čísla od 48 po 57, za nimi sú veľké písmená A až Z, majú poradové čísla 65 až 90 (je ich 26) a nakoniec malé písmená, majú poradové čísla 97 až 122. Poradové číslo znaku v tabuľke získame pomocou funkcie `ord` a znak, zodpovedajúci zadanému poradovému číslu, funkciou `chr`.

 Dôležité je vedieť, že pomocou zápisu `reťazec[i]` vieme pracovať s i-tým znakom reťazca. Prvý znak reťazca má index 1, druhý 2 atď.

 Funkcia **length (reťazec)** vracia celé číslo - dĺžku reťazca, t.j. počet jeho znakov. Napríklad `length ('Jánošík') = 7`, `length ('') = 0`, `length ('Jano' + 'Fero') = 8`.

Príklady, ktorými sa teraz budeme zaoberať, nemusíte ukladať ako samostatné programy (projekty), ale len ako „tlačidlá“ - procedúry v jednom formulári (pozri obrázok vpravo). Pre všetky môžete použiť na zadanie reťazca jeden komponent Edit a výstup napríklad Memo.



Príklad 3.4.1

Naprogramujte funkciu `Length`, t.j. vytvorte podprogram, ktorý zistí dĺžku zadaného reťazca.

Analýza

Problémom je, ako zistiť, ktorý znak v reťazci je posledný, za ktorým už „pre nás nič nie je, teda tam nemáme, presnejšie nesmieme, nazrieť“. Môžeme použiť jednoduchý trik, za koniec reťazca vložiť dohodnutý znak (nárazník), ktorý nám bude signalizovať, že sme dosiahli - presnejšie prekročili, koniec reťazca. Najuniverzálnejšie je asi vložiť nulový znak, ktorý sa zapisuje `#0` (mriežku `#` vložíte ako ľavé `Alt+35`). Zistiť dĺžku reťazca znamená zistiť počet jeho znakov po `#0`. Keďže nevieme, kedy narazíme na nulový znak, musíme použiť cyklus s podmienkou ukončenia.

Verzia s `while`-cyklom:

```
procedure TForm1.BtLengthClick(Sender: TObject);
var Ret: string; Dlzka: integer;
begin
  Ret:= Edit1.Text; // vloženie reťazca z Edit1 do premennej Ret
  Ret:= Ret + #0; // vloženie nárazníka za koniec reťazca
  Dlzka:= 0; // počiatková dĺžka reťazca je nula
  while Ret[Dlzka + 1] <> #0 do Dlzka:= Dlzka + 1; // pokiaľ sme nenarazili na nárazník, zvyšuj
  // hodnoty v premennej Dlzka o 1
  Memo1.Lines.Add ( 'Dĺžka reťazca je ' + IntToStr ( Dlzka ));
end;
```

V prípade použitia repeat-cyklu
 repeat

```
Dlzka:= Dlzka + 1;
until Ret[Dlzka + 1] = #0;
```

musíme buď samostatne ošetriť prázdny reťazec (použiť úplný príkaz if) alebo zvoliť pre počiatočnú hodnotu premennej Dlzka číslo -1 (Dlzka:= -1;).

Príklad 3.4.2

Funkcia LeftStr vráti zo zadaného reťazca zadaný počet znakov zľava. Naprogramujte ju.

☛ Pozor, funkciu LeftStr môžete použiť, len ak do uses v Unit1 dopíšete StrUtils.

Analýza

Napríklad LeftStr ('Janosik' , 4) vráti reťazec Jano. Všeobecne LeftStr (Ret, Kolko) vytvorí nový reťazec, do ktorého prenesie z Ret prvých Kolko znakov. Teda 1. znak, potom 2. atď. až Kolko. znak preto cyklus s pevným počtom opakovaní, for-cyklus. Nový reťazec vznikne pridaním prvého znaku Ret[1] k prázdnomu reťazcu, potom druhého s hodnotou Ret[2],..., vo všeobecnosti i-teho s hodnotou Ret[i].

```
procedure TForm1.BtLeftStrClick(Sender: TObject);
var Ret, novýRet: string;
    Kolko, i: integer;
begin
Ret:= Edit1.Text; // vloženie reťazca z Edit1 do premennej Ret
Kolko:= StrToInt (InputBox ('Funkcia LeftStr' , 'Kolko znakov vrátiť' , '')); // zadanie Kolko
novýRet:= ''; // nový reťazec je na začiatku prázdny
for i:=1 to Kolko do novýRet:= novýRet + Ret[i]; // pridávanie znakov z Ret do novýRet
Memo1.Lines.Add (novýRet); // vypísanie hodnoty novýRet
end;
```

📖 V predchádzajúcom príklade sme nový reťazec vytvárali postupným pridávaním znakov k pôvodne prázdnomu reťazcu. Tento postup musíme zvoliť, ak nepoznáme dĺžku novovytváraného reťazca. Ak ju poznáme, môžeme použiť inú stratégiu. Príkaz **SetLength** (premenná , číslo); nastaví dĺžku premennej typu string na zadané číslo a tým vytvorí priestor v pamäti na jeho zapamätanie. V predchádzajúcom príklade môžeme teda namiesto príkazu novýRet:= ''; použiť príkaz SetLength (novýRet , Kolko); a v cykle už len vložiť potrebné znaky z Ret do novýRet príkazom for i:= 1 to Kolko do novýRet[i]:= Ret[i]; (je nevyhnutné, aby i nadobúdalo hodnoty od 1!). Využitie príkazu setlength je obmedzené a dobre zväžte jeho použitie. Nemožno ho použiť napríklad ani v riešeniach nasledujúceho príkladu.

Príklad 3.4.3

Funkcia RightStr vráti zo zadaného reťazca zadaný počet znakov sprava. Naprogramujte ju.

☛ Pozor, funkciu RightStr môžete použiť, len ak do uses v Unit1 dopíšete StrUtils.

Analýza

Napríklad RightStr ('Janosik' , 5) vráti reťazec nosik. Všeobecne RightStr (Ret, Kolko) vytvorí nový reťazec, do ktorého prenesie z Ret posledných Kolko znakov. Napadne nám možno viacej riešení. Zátvorky v príkazoch for nie sú nevyhnutné, použili sme ich kvôli ľahšiemu pochopeniu príkazov.


```
Prvé: Dlzka:= length (Ret);
      novýRet:= '';
      for i:= (Dlzka – Kolko + 1) to Dlzka do novýRet:= novýRet + Ret[i];
```

```
Druhé: Dlzka:= length (Ret);
```

```
novyRet:= "";  
i:= 0;  
while i < Kolko do  
begin  
    novyRet:= Ret[Dlзка - i] + novyRet;    // záleží na poradí spájania znak + reťazec!  
    i:= i + 1;  
end;
```

Toto riešenie si asi vyžaduje komentár. Z Ret zoberieme posledný znak s indexom Dlзка-0 a pridáme ho do (pred) prázdny reťazec, potom predposledný s indexom Dlзка-1 a pridáme ho pred posledný atď. Cyklus končí po pridaní Kolko znakov.

```
Tretie: Dlзка:= length (Ret) + 1;    // pridali sme +1!  
novyRet:= "";  
for i:=1 to Kolko do novyRet:= Ret[Dlзка - i] + novyRet;
```

 Zvoľte si jedno z riešení, prípadne vymyslíte ďalšie, a odladíte ho na počítači.


Príklad 3.4.4

Funkcia Copy skopíruje zo zadaného reťazca, od zadaného čísla znaku zadaný počet znakov. Naprogramujte ju.

Napríklad Copy ('Janosik' , 2 , 3) vráti ano, Copy ('Janosik' , 3 , 3) vráti nos.

Riešenie bez komentára

```
procedure TForm1.BtCopyClick(Sender: TObject);  
var Ret, novyRet: string;  
    Od, Kolko, i: integer;  
begin  
    Ret:= Edit1.Text;  
    Od:= StrToInt (InputBox ('Funkcia Copy' , 'Od koľkého znaku' , ''));  
    Kolko:= StrToInt (InputBox ('Funkcia Copy' , 'Koľko znakov' , ''));  
    novyRet:= "";  
    for i:= Od to (Od + Kolko - 1) do novyRet:= novyRet + Ret[i];  
    Memo1.Lines.Add (novyRet);  
end;
```

 Funkcia Delete odstráni zo zadaného reťazca, od zadaného čísla znaku zadaný počet znakov. Napríklad Delete ('Janosik' , 4 , 2) vráti Janik. Naprogramujte ju.

 Funkcia Insert vloží zadaný reťazec do zadaného reťazca, od zadaného čísla znaku. Napríklad Insert ('ce' , 'Janosik' , 7) vráti Janosicek. Naprogramujte ju. Odporúčame použiť funkciu Copy.

Príklad 3.4.5

Funkcia UpCase zmení malé písmeno anglickej abecedy na veľké, ostatné znaky nemení. Naprogramujte ju.

Analýza

Podstatou riešenia tohto problému je poznanie kódovacej tabuľky a využitie postavenia malých a veľkých písmen anglickej abecedy v nej.

Algoritmus

Ak je znak malé písmeno angl. abecedy, posuň sa od neho doľava o vzdialenosť medzi malými a veľkými písmenami v tabuľke. Dostaneš hľadané veľké písmeno.

Nakreslite si vodorovnú os, dôležité znaky na nej podľa poradových čísel a ujasnite!

Presnejší zápis

posun $\leftarrow \text{ord}('a') - \text{ord}('A')$; // vzdialenosť medzi zodpovedajúcimi malými a veľkými písmenami
 ak $(\text{Znak} \geq 'a')$ a zároveň $(\text{Znak} \leq 'z')$ // ak je znak malé písmeno
 tak $\text{Znak} \leftarrow \text{chr}(\text{ord}(\text{Znak}) - \text{posun})$; // zmenši poradové číslo znaku o posun a ulož nový znak

Naprogramovanú funkciu UpCase sme použili v nasledujúcom príklade, kde si môžete pozrieť príslušnú časť programu. Premenná Znak je nahradená premennou Ret[i], teda i-tým znakom reťazca Ret.

Príklad 3.4.6

Funkcia UpperCase zmení v zadanom reťazci malé písmená anglickej abecedy na veľké, ostatné znaky nemení. Naprogramujte ju.

Analýza

Ak vieme zmeniť malé písmeno anglickej abecedy na veľké, vykonať takúto zmenu v reťazci je už jednoduché. Stačí prejsť reťazcom znak po znaku a všetky malé písmená anglickej abecedy meniť na veľké. Keďže vieme zistiť počet znakov v reťazci, môžeme použiť for-cyklus.

```
procedure TForm1.BtUpperCaseClick(Sender: TObject);
const POSUN = ord('a') - ord('A'); // vzdialenosť medzi malými a veľkými písmen. v kód. tabuľke
var Ret, novyRet: string;
    i, posun: integer;
begin
Ret:= Edit1.Text;
novyRet:= "";
for i:=1 to length(Ret) do
    if (Ret[i] >= 'a') and (Ret[i] <= 'z') // ak i-ty znak je malé písmeno
    then novyRet:= novyRet + chr(ord(Ret[i]) - posun) // pridaj veľké
    else novyRet:= novyRet + Ret[i]; // inak pridaj pôvodné
Memo1.Lines.Add (novyRet);
end;
```

☛ Podmienku $(\text{Ret}[i] \geq 'a')$ and $(\text{Ret}[i] \leq 'z')$ možno nahradiť aj množinovým zápisom $\text{Ret}[i] \text{ in } ['a'..'z']$ („in“ čítame „je prvkom množiny, patrí“, matematický symbol \in).

📖 Funkcia LowerCase zmení v zadanom reťazci veľké písmená anglickej abecedy na malé, ostatné znaky nemení. Naprogramujte ju.

📖 Vytvorte podprogram, ktorý vypíše koľko číslíc, veľkých a malých písmen anglickej abecedy a iných znakov je v zadanom reťazci.

Príklad 3.4.7

Vytvorte podprogram, ktorý zistí miesto prvého výskytu zadaného znaku v reťazci. Ak sa znak v reťazci nenachádza, nech to oznámi.

Analýza

Funkciu nazvime PosChar. Napríklad PosChar ('Janosik', 'o') vráti číslo 4. Postup je jednoduchý, postupne porovnávame prvý znak reťazca s hľadaným znakom, potom druhý atď. až kým nenájdeme hľadaný znak alebo sa nedostaneme na koniec reťazca. Ak sme sa dostali až na koniec reťazca, ešte to neznamená, že znak sa v ňom nenachádza! Môže byť na poslednom mieste v reťazci. V každom prípade je potrebný cyklus s podmienkou ukončenia. Môžeme použiť zaujímavú „fintu“, tzv.

nárazník. Na koniec reťazca vložíme hľadaný znak a tým máme zaručené, že sa v reťazci nachádza. Ak ho cyklus nenájde skôr ako na poslednom mieste, znak sa pôvodne v reťazci nenachádzal.

```
procedure TForm1.PosCharClick(Sender: TObject);
var Ret: string;
    Znak: char;
    i: integer;
begin
Ret:= Edit1.Text;
Znak:= InputBox ('PosChar' , 'Hľadať znak' , '')[1];           // vysvetlené pod procedúrou
Ret:= Ret + Znak;                                             // vloženie nárazníka!
i:= 0;
repeat
i:= i + 1
until Ret[i] = Znak;
if i = length(Ret)
then Memo1.Lines.Add ('Znak sa v reťazci nevyskytuje')        // cyklus sa zastavil na nárazníku
else Memo1.Lines.Add ('Prvý výskyt znaku na indexe ' + IntToStr(i)) // znak sa našiel skôr
end;
```

Priradenie `Znak:= InputBox ('PosChar' , 'Hľadať znak' , '')`; prekladač nedovolí, lebo výsledkom `InputBox`-u je typ `string` a ten sa do typu `char` „nezmestí“. Situáciu možno zachrániť priradením len prvého znaku z reťazca (veď len ten sme zadali) zápisom `InputBox(...)[1]`.

Príklad 3.4.8

Vytvorte podprogram, ktorý odstráni zo zadaného reťazca všetky znaky okrem číslíc a písmen anglickej abecedy.

Riešenie bez komentára

```
procedure TForm1.OdstranClick(Sender: TObject);
var Ret, novyRet: string;
    i: integer;
begin
Ret:= Edit1.Text;
novyRet:= '';
for i:= 1 to length(Ret) do // alternatíva podmienky nižšie Ret[i] in ['0'..'9','A'..'Z','a'..'z']
    if (Ret[i]>='0')and(Ret[i]<='9') or (Ret[i]>='A')and(Ret[i]<='Z') or (Ret[i]>='a')and(Ret[i]<='z')
    then novyRet:= novyRet + Ret[i];
Memo1.Lines.Add (novyRet);
end;
```

Príklad 3.4.9

Vytvorte podprogram, ktorý uloží do nového reťazca znaky v opačnom poradí, ako sú v zadanom reťazci.

Analýza

Ak do premennej `Dlžka` uložíme dĺžku reťazca, zrejme treba vymeniť

- | | | | | |
|---------|-----------------------------|------------------------|---|----------------------------|
| 1. znak | s posledným, jeho index je: | <code>Dlžka</code> | = | <code>Dlžka - 1 + 1</code> |
| 2. | s predposledným – index: | <code>Dlžka - 1</code> | = | <code>Dlžka - 2 + 1</code> |
| 3. | | <code>Dlžka - 2</code> | = | <code>Dlžka - 3 + 1</code> |
| ... | | | | |
| i. | | | = | <code>Dlžka - i + 1</code> |

...
 ...
 Dĺžka

```
procedure TForm1.OtocClick(Sender: TObject);
var Ret, novýRet: string;
    i, Dĺžka: integer;
begin
Ret:= Edit1.Text;
Dĺžka:= length(Ret);
setlength(novýRet, Dĺžka);
for i:= 1 to Dĺžka do novýRet[i]:= Ret[Dĺžka - i + 1];
Memo1.Lines.Add(novýRet);
end;
```

Príklad 3.4.10

Vytvorte podprogram, ktorý zistí, či zadaný reťazec je symetrický. Symetrické sú napríklad reťazce ABBA, radar, jelenovipivonelej.

Analýza

Pre symetrický reťazec zrejme platí: prvý znak sa rovná poslednému (má index dĺžka reťazca!), druhý predposlednému atď. až do stredu reťazca (stačí, aby sa ľavá polovica reťazca rovnala pravej polovici). Opäť by sme mohli použiť for-cykľus a „prepínač“, ako v prípade zisťovania prvočísla (Príklad 3.1.3). Efektívnejšie však je použiť cyklus s podmienkou ukončenia, ktorý sa ukončí hneď, ako sa nájdu nezodpovedajúce znaky alebo sme už otestovali všetky znaky reťazca.

Treba testovať, či

1. znak	=?	Dĺžka (posledný)	= Dĺžka - 1 + 1
2.	=?	Dĺžka - 1 (predposledný)	= Dĺžka - 2 + 1
3.	=?	Dĺžka - 2	= Dĺžka - 3 + 1
...	=?	...	
i.	=?	Dĺžka - i + 1	

až po stredný index, čo je zrejme Dĺžka div 2.

Riešenie

Komentár si zaslúži použitie príkazu if. Vždy sa snažte v podmienke testovať to, čo je pre riešenie úlohy prioritné! V tomto príklade je prioritné zistenie, či sa rovnajú príslušné znaky. Teda ak skončil cyklus a aj posledne testované znaky sa rovnajú, reťazec musí byť symetrický. Postaviť podmienku v if na zistení, či sme už v strede reťazca, by mohlo byť zradné.

```
procedure TForm1.SymetrickyClick(Sender: TObject);
var Ret: string;
    i, Dĺžka: integer;
begin
Ret:= Edit1.Text;
Dĺžka:= length(Ret);
i:= 1;
while ( Ret[i] = Ret[Dĺžka-i+1] ) and ( i < Dĺžka div 2 ) do i:= i + 1;
if Ret[i] = Ret[Dĺžka-i+1] then Memo1.Lines.Add ('Je symetrický')
else Memo1.Lines.Add ('Nie je symetrický')
end;
```

Príklad 3.4.11

Vytvorte program, ktorý zo zadaného rodného čísla
 a) odstráni lomku, ak bola vložená

- b) zistí, či to je správne rodné číslo
- c) zistí pohlavie
- d) vypíše dátum narodenia.

Analýza

RR znamená posledné dvojčísle roku narodenia (predpokladajme, že máme v evidencii len osoby narodené po roku 1910), MM mesiac a DD deň. Rodné číslo u muža obsahuje RRRMMDDXXXX, napríklad 7206189199 znamená dátum narodenia 18. jún 1972. Posledné štvorčísle je volené tak, aby rodné číslo bolo deliteľné 11. Pri rodnom čísle ženy je k MM pripočítané číslo 50, napríklad 8761137088 znamená dátum narodenia 13. november 1987. Často sa rodné číslo píše aj s lomkou pred posledným štvorčísliem. Tú ľahko odstránime funkciou Delete.

Rodné číslo je správne, ak obsahuje len cifry a je deliteľné 11 (správnosť cifier zanedbáme). Tu už nastáva problém, lebo desaťciferné rodné číslo je pre typ integer „trochu veľké“ (chybové hlásenie vpravo). Pomôcť si môžeme typom real, pričom využijeme funkciu $\text{frac}(x:\text{real})$, ktorá vracia desatinnú časť z reálneho čísla x . To sa nám hodí, lebo ak desatinná časť čísla $\text{StrToFloat}(RC) / 11$ sa rovná nule, číslo musí byť deliteľné 11, a teda správne.

Ak 3. znak v RC je znak 0 alebo 1, je to muž, ináč žena.

Dátum narodenia musíme postupne poskladať, ako sa nám to podarilo, pozri nižšie.



```
procedure TForm1.RCClick(Sender: TObject);
```

```
var RC, DatNar: string;
```

```
    i: integer;
```

```
begin
```

```
RC:= Edit1.Text;
```

```
// zadanie rodného čísla
```

```
if length(RC) > 10 then Delete(RC,7,1);
```

```
// odstránenie lomky, ak treba
```

```
i:=1; while (RC[i] >= '0') and (RC[i] <= '9') and (i<10) do i:= i + 1;
```

```
// Kontrola znakov v RC
```

```
if (RC[i] < '0') or (RC[i] > '9')
```

```
then Memo1.Lines.Add ('Nedovolený znak v rodnom čísle')
```

```
else if frac( StrToFloat(RC) / 11) <> 0
```

```
// Kontrola deliteľnosti 11
```

```
    then Memo1.Lines.Add ('Zlé rodné číslo')
```

```
    else begin // Rodné číslo je správne, zistenie pohlavia a dátumu narodenia
```

```
        if RC[3] < '2' then Memo1.Lines.Add ('Muž')
```

```
        else Memo1.Lines.Add ('Žena');
```

```
        // Vytvorenie dátumu narodenia
```

```
DatNar:= Copy( RC , 5 , 2 ) + '.';
```

```
// deň
```

```
if ( RC[3] = '1' ) or ( RC[3] = '6' ) then DatNar:= DatNar + '1';
```

```
// mesiac
```

```
DatNar:= DatNar + RC[4] + '.';
```

```
// mesiac
```

```
if Copy( RC , 1 , 2 ) <= '10' then DatNar:= DatNar + '20' + Copy( RC , 1 , 2 )
```

```
else DatNar:= DatNar + '19' + Copy( RC , 1 , 2 );
```

```
Memo1.Lines.Add ('Dátum narodenia: ' + DatNar);
```

```
end;
```

```
end;
```

☞ Program upravte tak, aby nevypisoval nulu, keď je dátum dňa narodenia jednociferný.

☞ Využite množinový zápis v programe, t.j. výrazy $(RC[i] \geq '0')$ and $(RC[i] \leq '9')$ nahraďte zápisom $RC[i]$ in $['0'..'9']$ a pod.

Príklad 3.4.12

Vytvorte program, ktorý umožní zašifrovať zadaný reťazec posunutím každého znaku o jednu pozíciu doprava.

Analýza

Napríklad z reťazca „A posuň na B.“ vznikne „B!qptvó!ob!C/“. Ako? Zoberie sa prvý znak a zmení na jeho nasledovníka (funkcia succ(znak)), potom druhý,..., až posledný znak v pôvodnom reťazci preto cyklus s pevným počtom opakovaní.

Program sme doplnili ponukou Dešifruj, ktorá posunie všetky znaky vstupného reťazca o jednu pozíciu doľava (funkcia pred(znak)). V oboch procedúrach možno použiť aj príkaz setlength pre nový reťazec, my sme ho použili len v procedúre DesifrujClick.

Zvolili sme takú kombináciu príkazov, aby umožňovala viackrát za sebou použiť tlačidlo Šifruj a tak isto tlačidlo Dešifruj (komponenty Edit slúžia ako vstupné aj výstupné!).

Pre začiatok si odľad'te základnú verziu, zvýraznenú v procedúrach!

Procedúru TForm1.Edit1Click sme vložili dvojklikom do udalosti (Events) OnClick pre komponent Edit1! Mysleli sme na to, že po kliknutí do Edit1 chce užívateľ zmeniť vstupný text a teda „starý“ text v Edit2.Text treba zmazať. Tiež nemožno dešifrovať nový vstupný text, preto sa tlačidlo Dešifruj „znevíditeľní“. Pred prvým spustením programu by ste mali vlastnosť Visible u tlačidla Dešifruj nastaviť na False.

```
procedure TForm1.SifrujClick(Sender: TObject);
```

```
var Ret, novyRet: string;
```

```
    i: integer;
```

```
begin
```

```
if Edit2.Text<>" then Edit1.Text:= Edit2.Text;
```

```
Ret:= Edit1.Text;
```

```
novyRet:= "";
```

```
for i:= 1 to length(Ret) do novyRet:= novyRet + succ(Ret[i]);
```

```
Edit2.Text:= novyRet;
```

```
// zobraz zašifrovaný text v Edit2
```

```
Desifruj.Visible:= True;
```

```
// zviditeľní tlačidlo Dešifruj
```

```
end;
```

```
procedure TForm1.DesifrujClick(Sender: TObject);
```

```
var Ret, novyRet: string;
```

```
    i: integer;
```

```
begin
```

```
Edit1.Text:= Edit2.Text;
```

```
// zašifrovaný text sa stane vstupným textom
```

```
Edit2.Text:= "";
```

```
// zmaže Edit2
```

```
Ret:= Edit1.Text;
```

```
// vstupný text do premennej Ret
```

```
setlength (novyRet, length(Ret));
```

```
// vyhrad' miesto v pamäti pre novyRet
```

```
for i:= 1 to length(Ret) do novyRet[i]:= pred(Ret[i]); // posuň a ulož do novyRet znak po znaku
```

```
Edit2.Text:= novyRet;
```

```
// zobraz v Edit2
```

```
end;
```

```
procedure TForm1.Edit1Click(Sender: TObject);
```

```
begin
```

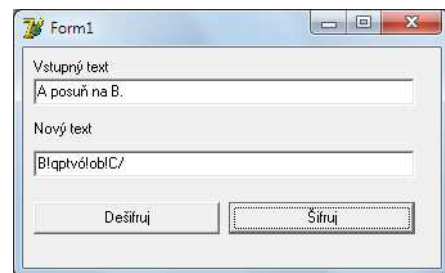
```
Edit2.Text:= "";
```

```
// zmaže Edit2
```

```
Desifruj.Visible:= False;
```

```
// znevíditeľní tlačidlo Dešifruj
```

```
end;
```



☞ Program upravte tak, aby posúval len písmená anglickej abecedy a číslice a to tak, že $Z \rightarrow A$, $z \rightarrow a$, $9 \rightarrow 0$ a pri dešifrovaní opačne $A \rightarrow Z$, $a \rightarrow z$ a $0 \rightarrow 9$.

Príklad 3.4.13

Naprogramujte funkciu `IntToStr`, t.j. vytvorte program, ktorý zmení zadané celé nezáporné číslo na reťazec.

Analýza

Nezáporné celé číslo zmeníme na reťazec jeho „rozobratím“ na cifry a ich prevodom na znaky. Rozobrať číslo na cifry pomocou funkcií `mod` a `div` sme sa učili napríklad pri výpočte ciferného súčtu zadaného čísla. Zmeniť cifru na znak vieme, ak poznáme princíp kódovacej tabuľky a funkcie `chr` a `ord` na prácu so znakmi. K cifrám sa dostávame od jednotiek čísla, takže pozor na poradie vkladania znakov do vytváraného reťazca.

Princíp výpočtu pre číslo 53

Cislo	Cifra	<code>ord('0') + Cifra</code>	<code>chr(ord('0') + Cifra)</code>	<code>chr(ord('0')+Cifra) + CisloStr</code>
53	$53 \bmod 10$	$48 + 3$	<code>chr(51)</code>	'3' + ''
5	$5 \bmod 10$	$48 + 5$	<code>chr(53)</code>	'5' + '3'
0				'53'

Riešenie

```
procedure TForm1.BtIntToStrClick(Sender: TObject);
var Cislo, Cifra: integer;
    CisloStr: string;
begin
Cislo:= StrToInt (Edit1.Text);           // zadanie celého čísla
CisloStr:= "";                           // „číselný“ reťazec je na zač. prázdny
repeat                                   // opakuj
    Cifra:= Cislo mod 10;                 // získanie poslednej cifry z čísla
    CisloStr:= chr ( ord('0') + Cifra ) + CisloStr; // prevod cifry na znak a pridanie do reť.
    Cislo:= Cislo div 10;                 // odstránenie spracovanej cifry z čísla
until Cislo = 0;                          // pokiaľ sme nespracovali všetky cifry
Memo1.Lines.Add (CisloStr);              // vypíš „číselný“ reťazec
end;
```

☞ Podobne ako funkcie `IntToStr` a `FloatToStr` pracuje aj príkaz `Str`, pozrite si ho v pomocníkovi.

Príklad 3.4.14

Naprogramujte funkciu `StrToInt`, ktorá zadaný reťazec zmení na nezáporné celé číslo.

Analýza

Predpokladajme, že zadaný reťazec obsahuje len korektné (správne) nezáporné celé číslo.

Proces bude opačný, ako v predchádzajúcom príklade, t.j. znaky budeme meniť na cifry a skladať z nich celé číslo. Počet znakov vieme zistiť funkciou `length`. Zmeniť znak na cifru možno úvahou: znak napríklad '5' je vzdialený v kódovacej tabuľke od znaku '0' o 5 pozícií, alebo $\text{ord}('5') - \text{ord}('0') = 53 - 48 = 5$. Všeobecne pre „číselný“ znak dostávame $\text{ord}(\text{znak}) - \text{ord}('0')$ = hľadanej cifre.


Riešenie

```
procedure TForm1.BtStrToIntClick(Sender: TObject);
var CisloStr: string;
    Cislo, Cifra, i: integer;
begin
CisloStr:= Edit1.Text;                   // zadanie „číselného“ reťazca
```

```

Cislo:= 0; // počiatková hodnota
for i:= 1 to length(CisloStr) do // spracovať prvý znak, druhý,...
begin
    Cifra:= ord( CisloStr[i] ) – ord('0'); // vzdialenosť i-teho čísel. znaku od znaku nula
    Cislo:= Cislo*10 + Cifra; // pridanie cifry na miesto jednotiek
end;
Memo1.Lines.Add (IntToStr(Cislo)) // vypísanie výsledku
end;

```

 Podobne ako funkcie StrToInt a StrToFloat pracuje aj príkaz Val. Má tvar Val (reťazec, premenná, pozícia); pričom reťazec prekonvertuje na celé alebo reálne číslo (podľa toho, akého typu je premenná) a ak nastala chyba, v premennej pozícia je 0, ak nastala chyba pri konverzii, v premennej pozícia je poradové číslo prvého chybného znaku.

Napríklad nech platí var Cislo: real; Poz: integer;

```

Použitie príkazu Val (Edit1.Text, Cislo, Poz);
if Poz > 0 then ShowMessage ('Chybný ' + IntToStr(Poz) + '. znak!')
else // výpočet

```

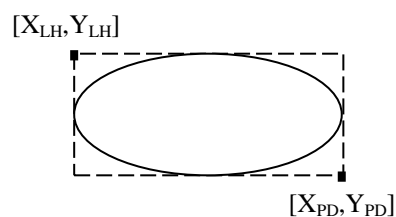
Úvod do grafiky

Na kreslenie sa používa komponent Image (komponent v záložke Additional) a jeho vlastnosť Canvas - plátno. Nemusíme však použiť komponent Image, môžeme kresliť aj na akési virtuálne plátno, čo má ale určité obmedzenia (napr. obrázok nevieme uložiť, aj krátkodobým prekrytím obrázka iným oknom prichádzame o prekrytú časť obrázka). Nepoužitie komponentu Image však zjednodušuje zápis a v našom úvode sa bez neho zaobídeme.

Súradnicová sústava je preklopená okolo osi x, t.j. kladné hodnoty na osi ypsilon „idú“ smerom nadol!



Obrázok k časti **Útvary**



Útvary:

MoveTo (X,Y)

LineTo (X,Y)

Rectangle (XLH,YLH,XPD,YPD)

Ellipse (XLH,YLH,XPD,YPD)

Polygon([Point(x1,y1), Point(x2,y2),...])

TextOut (X,Y, 'reťazec')

Pixels [X,Y] := clFarba

nastaví kreslenie do bodu so súradnicami X,Y

nakreslí úsečku z aktuálnej pozície do bodu X,Y

nakreslí obdĺžnik

nakreslí vpísanú elipsu

nakreslí N-uholník s vrcholmi $[x_i, y_i]$, $i = 1, 2, \dots, N$

vypíše reťazec od bodu X,Y

vykreslí bod zadanej farby napr. clRed, clBlue,...

pozri vlastnosť Color v Properties ľub. komponentu

Príklad G1a

Vytvorte program, ktorý, po kliknutí na tlačidlo Mriežka, vykreslí štyri vodorovné a štyri zvislé čiary vo vzdialenostiach po 100 bodov (obrázok vpravo).

Návod

Kreslenie čiary začína v bode $[x_0, y_0]$, do ktorého môžeme nastaviť pero príkazom Canvas.MoveTo(x_0, y_0) a končí v bode, ktorého súradnice treba zadať príkazom Canvas.LineTo(x, y).

Riešenie

```
procedure TForm1.btMriezkaClick(Sender: TObject);
```

```
begin
```

```
Canvas.MoveTo(100,0); Canvas.LineTo(100,400);
```

```
Canvas.MoveTo(200,0); Canvas.LineTo(200,400);
```

```
Canvas.MoveTo(300,0); Canvas.LineTo(300,400);
```

```
Canvas.MoveTo(400,0); Canvas.LineTo(400,400);
```

```
Canvas.MoveTo(0,100); Canvas.LineTo(400,100);
```

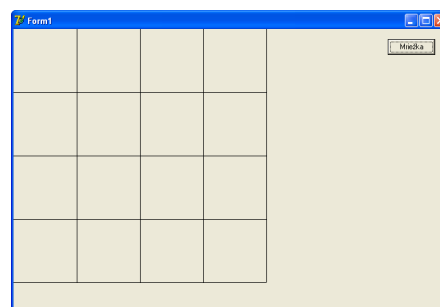
```
Canvas.MoveTo(0,200); Canvas.LineTo(400,200);
```

```
Canvas.MoveTo(0,300); Canvas.LineTo(400,300);
```

```
Canvas.MoveTo(0,400); Canvas.LineTo(400,400);
```

```
end;
```

Mriežka nám pomôže orientovať sa pri programovaní zložitejších obrázkov. Vidíme, že ak nezmeníme veľkosť formulára, pracovnou oblasťou je približne štvorec 0,0 až 400,400 bodov. Všimnite si, že napriek tomu, že počty bodov vo vodorovnom a zvislom smere sú rovnaké, nedostávame presné štvorce.



Príklad G1b

 Vytvorte program, ktorý nakreslí poľskú zástavu.

Návod

Poľská zástava sa skladá z dvoch vodorovných pruhov, horný biely a spodný červený.

Ak si pomôžeme mriežkou, ako vhodné sa ukazuje vyfarbiť obdĺžniky so súradnica 100,100,400,200 a 100,200,400,300.

Riešenie

```
procedure TForm1.btPolskaClick(Sender: TObject);
```

```
begin
```

```
with Canvas do
```

```
begin
```

```
    Brush.Color:= clWhite;           //nastavenie farby výplne
```

```
    Rectangle(100,100,400,200);     //vykreslenie obdĺžnika, farba okraja - pera štand. čierna
```

```
    Brush.Color:= clRed;            //nastavenie farby výplne
```

```
    Rectangle(100,200,400,300);    //vykreslenie obdĺžnika
```

```
end;
```

```
end;
```

V príklade sme použili príkaz `with Canvas do begin príkazy end;` ktorý nám umožňuje skrátiť zápis príkazov začínajúcich na `Canvas`. Medzi `begin` a `end` nemusíme písať `Canvas`, program ho „dosadí za nás“.

Často sa nám bude hodiť zmazať formulár, použijeme na to procedúru:

```
procedure TForm1.btZmazClick(Sender: TObject);           //procedúra tlačidla Zmaž
```


```
begin
```

```
Canvas.Brush.Color:= Form1.Color;                       //nastavenie farby výplne na farbu formulára
```

```
Canvas.FillRect(ClientRect);                             //prekreslenie plátna farbou formulára-pozadia
```

```
end;
```

Príklad G1c

 Vytvorte program, ktorý nakreslí zástavu Monaka, v ktorej sú len farebné pruhy v opačnom poradí, ako na poľskej zástave.

Príklad G1d

 Vytvorte program, ktorý nakreslí zástavu SRN.

Zástava SRN má tri vodorovné pruhy s farbami čierna, červená a žltá (najnižšie).

Príklad G1e

 Vytvorte program, ktorý nakreslí zástavu Belgicka.

Zástava Belgicka sa skladá z troch zvislých pruhov, čierneho, žltého a červeného (najviac vpravo).

Riešenie

Zástavu sme umiestnili do začiatku súradnicovej sústavy 0,0.

Posuňte ju do stredu mriežky!

```
procedure TForm1.btBelgickaClick(Sender: TObject);
```

```
begin
```

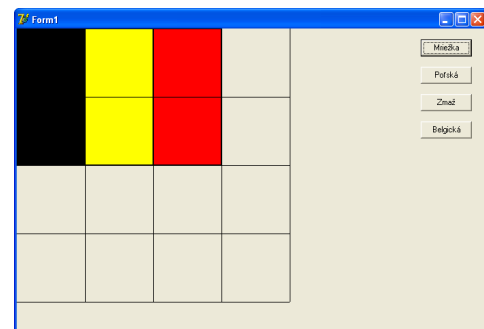
```
with Canvas do
```

```
begin
```

```
    Brush.Color:= clBlack;
```

```
    Rectangle(0,0,100,200);
```

```
    Brush.Color:= clYellow;
```



```
Rectangle(100,0,200,200);  
Brush.Color:= clRed;  
Rectangle(200,0,300,200);
```

```
end;  
end;
```

Príklad G1f



  Vytvorte program, ktorý nakreslí zástavu Švajčiarska.

Zástava Švajčiarska je tvorená bielym krížom umiestneným v červenom štvorci (biely kríž nezasahuje až po okraj štvorca).

Riešenie

- neúplné
procedure TForm1.btSvajciarskaClick(Sender: TObject);
begin
with Canvas do
begin
Brush.Color:= clRed;
Rectangle(100,100,400,400);
Brush.Color:= clWhite;
Pen.Color:= clWhite; //najprv vytvorte obrázok bez tohto príkazu!
Rectangle(200,100,300,400);
Rectangle(100,200,400,300);
end;
end;
- úplné
pozmeňte parametre obrázka tak, aby nezmenený kríž bol obklopený červenou farbou.

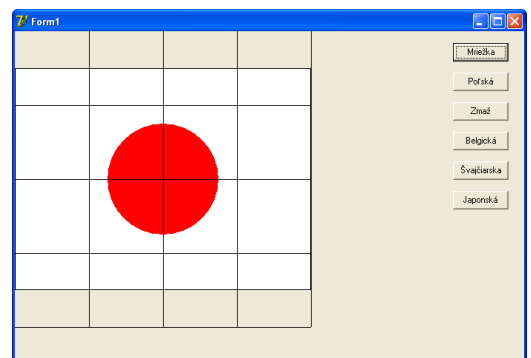
Príklad G1g



  Vytvorte program, ktorý vykreslí zástavu Japonska („vychádzajúce Slnko na bielej oblohe“).

Jedno z možných riešení:

```
procedure TForm1.btJaponskaClick(Sender: TObject);  
begin  
with Canvas do  
begin  
Brush.Color:= clWhite;  
Rectangle(0,50,400,350);  
Brush.Color:= clRed;  
Pen.Color:= clRed;  
Ellipse(125,125,275,275);
```

```
end;  
end;
```



  Na internete si nájdite zástavy štátov a pokúste sa ich vykreslenie naprogramovať (Francúzska, Dánska, USA bez hviezdíčiek atď.).

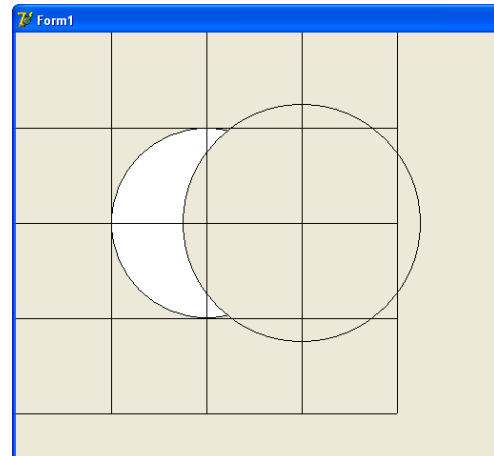
Príklad G1h

☞ ☞ Často sa na zástavách štátov nachádza polmesiac s rôznym vykrojením. Ako ho možno naprogramovať (inšpirujte sa obrázkom)?

Riešenie

```

procedure TForm1.btPolmesiacClick(Sender: TObject);
begin
with Canvas do
begin
    Brush.Color:= clWhite;
    Pen.Color:= Form1.Color;
    Ellipse(100,100,300,300);
    Brush.Color:= Form1.Color;
//    Ellipse(200,100,400,300); // 1.pokus
    Ellipse(175,75,425,325); //toto sa nám pozdávalo
end;
end;
  
```



Príklad G1i

☞ ☞ Čo vykreslí procedúra s príkazom Polygon – mnohoúholník?

```

procedure TForm1.btCeskaClick(Sender: TObject);
begin
with Canvas do
begin
    Brush.Color:= clWhite;
    Rectangle(100,100,400,200);
    Brush.Color:= clRed;
    Rectangle(100,200,400,300);
    Brush.Color:= clBlue;
    Polygon([Point(100, 100), Point(200, 200), Point(100, 300)]);
end;
end;
  
```

Takže teraz už ľahko nakreslíte hviezdu, prípadne javorový list.

☞ Farby možno namiešať aj zápisom **RGB (číslo, číslo, číslo)** kde číslo je celé číslo od 0 po 255 a R znamená red/červená, G - green/zelená a B - blue/modrá; preto napr. clBlack = RGB(0,0,0); clWhite = RGB(255,255,255); clRed = RGB(255,0,0); clYellow = RGB(255,255,0) atď.
 Pozri Skicár – Upraviť farby: Červená – Zelená – Modrá.

☞ Veľmi pekné obrázky vznikajú aj náhodným generovaním vlastností obrázkov (farby, umiestnenia, rozmerov, hrúbky čiary,...), čo umožňuje funkcia random.
 Funkcia **random** (\mathcal{N}) vráti náhodne vybrané prirodzené číslo z intervalu $\langle 0, \mathcal{N}-1 \rangle$, kde \mathcal{N} je prirodzené číslo.

Funkcia **random** vráti náhodne vybrané reálne číslo z intervalu $\langle 0, 1 \rangle$.


Pred funkciou random sa zvykne jedenkrát použiť príkaz randomize (znáhodní výber aj prvého čísla). Najjednoduchšie je umiestniť príkaz randomize do časti initialization, t.j na záver unitu uviesť

```


initialization
randomize;
end.
  
```


Preto nebudeme v jednotlivých procedúrach uvádzať príkaz randomize, predpokladáme jeho umiestnenie v časti initialization, ktorej príkazy sa vykonajú hneď po spustení programu.


Príklad G2a

 Vytvorte program, v ktorom sa, po každom kliknutí na tlačidlo ČIARA, vykreslí úsečka so začiatkom v bode [200,200] s náhodne vybranou hrúbkou čiary (pera) od 1 po 10, s náhodne vybranou farbou čiary a s náhodne vybraným koncovým bodom kreslenej úsečky od 0 po 400.

```
procedure TForm1.btCiaraClick(Sender: TObject); //po každom kliknutí na tlačidlo nakreslí čiaru
begin
Canvas.Pen.Width:= random(10) + 1; //náhod.výber hrúbky pera
Canvas.Pen.Color:= RGB(random(256), random(256), random(256)); //náhod.výber farby pera
Canvas.MoveTo( 200, 200 ); //začiatok čiary [200,200]
Canvas.LineTo( random(401), random(401) ); //náhod.koniec čiary so súradnicami od 0 po 400
end;
```

 Ak chceme použiť ľubovoľnú zo všetkých možných farieb, stačí použiť výraz $\text{random}((255*255*255)+1)$, ktorý náhodne vyberie číslo farby a Delphi toto číslo „premení“ na farbu zvoleného útvaru príkazom `útvár.Color:= random((255*255*255)+1);`

Príklad G2b


 Vytvorte program, v ktorom sa, po kliknutí na tlačidlo BOD, zobrazí bod s náhodne vybranými súradnicami od 0 po 499 a s náhodne vybranou farbou.

Riešenie

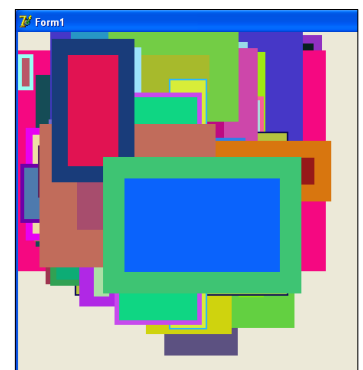
Riešenie je v prvom riadku príkazovej časti procedúry (v poznámke). Tieto body sú veľmi malé, na ďalších riadkoch ukazujeme „fintu“, ako možno zväčšiť veľkosť „bodu“. Keďže v príkazoch `Canvas.MoveTo(x,y)`; a `Canvas.LineTo(x,y)`; potrebujeme použiť rovnaké súradnice x aj y, musíme ich najprv vygenerovať do premenných x a y a až následne použiť.

```
procedure TForm1.btBodClick(Sender: TObject);
var x,y: integer;
begin
// Canvas.Pixels[random(500), random(500)]:= random((255*255*255)+1);
Canvas.Pen.Width:= 10; //nastavenie hrúbky pera na 10
Canvas.Pen.Color:= random((255*255*255)+1); //vygenerovanie náhod. farby pera - „bodu“
x:= random(500); y:= random(500); //vygenerovanie súradnice x a y
Canvas.MoveTo(x,y);
Canvas.LineTo(x,y); //x musí byť rovnaké ako v predchádzajúcom riadku, analogicky pre y
end;
```

Príklad G2c

 Vytvorte program, v ktorom sa, po kliknutí na tlačidlo OBDĽŽNIK, vykreslí obdĺžnik s náhodne vybranou farbou výplne (`Brush.Color`) aj okraja (`Pen.Color`), s náhodne vybranou hrúbkou okraja (`Pen.Width`) aj súradnicami X_{LH} , Y_{LH} , X_{PD} a Y_{PD} (obrázok vpravo).

```
procedure TForm1.btObdlznikClick(Sender: TObject);
begin
with Canvas do
begin
```



```


Brush.Color:= random((255*255*255)+1);
Pen.Width:= random(50)+1;
Pen.Color:= random((255*255*255)+1);
Rectangle(random(300), random(300), random(400), random(400));

```

```
end;
```

```
end;
```

Príklad G2d

 Vytvorte program, v ktorom sa, po kliknutí na tlačidlo ELIPSA, vykreslí elipsa s náhodne vybranou farbou výplne (Brush.Color) aj okraja (Pen.Color), s náhodne vybranou hrúbkou okraja (Pen.Width) aj súradnicami X_{LH} , Y_{LH} , X_{PD} a Y_{PD} .

Riešenie

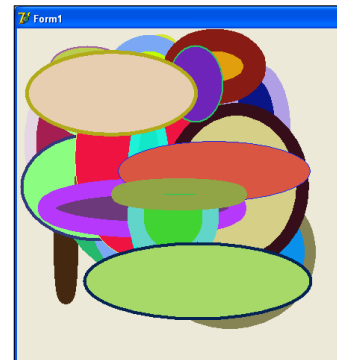
```

procedure TForm1.btElipsaClick(Sender: TObject);
begin
with Canvas do
begin
    Brush.Color:= random((255*255*255)+1);
    Pen.Width:= random(30)+1;
    Pen.Color:= random((255*255*255)+1);
    Ellipse(random(300), random(300), random(400), random(400));

```

```
end;
```

```
end;
```



Príklad G2e

 Čo musíme zmeniť, aby sme v príklade G.2c namiesto obdĺžnika dostali štvorec?

Riešenie

Po kliknutí na tlačidlo ŠTVOREC sa vykreslí štvorec s náhodne vybranou farbou výplne (Brush.Color) aj okraja (Pen.Color), s náhodne vybranou hrúbkou okraja (Pen.Width) aj súradnicami X_{LH} , Y_{LH} . Súradnice X_{PD} a Y_{PD} však už nemôžu byť náhodne vybrané čísla, pretože vo štvorci musí byť vzdialenosť z bodu $[X_{LH}, Y_{LH}]$ do bodu $[X_{PD}, Y_{PD}]$ rovnaká v smere osí x aj y (strana štvorca). Veľkosť strany štvorca však môže byť ľubovoľné kladné celé číslo.

```


procedure TForm1.btStvorecClick(Sender: TObject);
var x, y, strana: integer;
begin
with Canvas do
begin
    Brush.Color:= random((255*255*255)+1);
    Pen.Width:= random(20)+1;
    Pen.Color:= random((255*255*255)+1);
    x:= random(300); y:= random(300); strana:= random(200)+1;
    Rectangle(x, y, x + strana, y + strana);

```

```
end;
```

```
end;
```

Príklad G2f

 Vytvorte program, ktorý po kliknutí na tlačidlo KRUH, vykreslí kruh a nie elipsu.

Riešenie


```

procedure TForm1.btKruhClick(Sender: TObject);
var x, y, priemer: integer;

```

```
begin
with Canvas do
begin
    Brush.Color:= random((255*255*255)+1);
    Pen.Width:= random(10)+1;
    Pen.Color:= random((255*255*255)+1);
    x:= random(300); y:= random(300); priemer:= random(200)+1;
    Ellipse(x, y, x + priemer, y + priemer);
end;
end;
```


Príklad G2g

 Kružnica je určená aj svojim stredom a polomerom. Upravte predchádzajúcu procedúru tak, aby sa generoval stred a polomer kružnice.

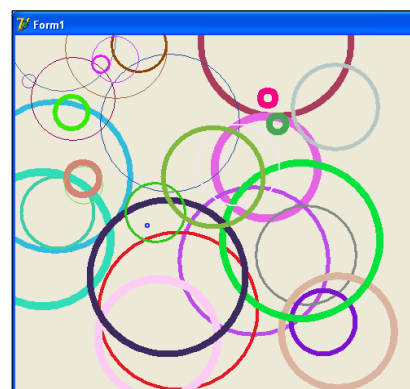
Riešenie

```
procedure TForm1.btKruznicaClick(Sender: TObject);
var Sx, Sy, polomer: integer;
begin
with Canvas do
begin
    Pen.Width:= random(10)+1;
    Pen.Color:= random((255*255*255)+1);
    Sx:= random(400); Sy:= random(400); polomer:= random(100)+1;
    Ellipse(Sx - polomer, Sy - polomer, Sx + polomer, Sy + polomer);
end;
end;
```

Je len na vás, ktorý zo spôsobov na kreslenie kružníc a kruhov budete používať (pokiaľ to riešenie úlohy umožňuje).

 Určite ste si všimli, že sme vynechali príkaz `Brush.Color:= random((255*255*255)+1);` aby sme dostali kružnicu a nie kruh. Napriek tomu pri viacnásobnom kliknutí na tlačidlo KRUŽNICA vidíme, že „vnútro kružnice“ je vyplnené farbou formulára (premazávajú sa prekrývajúce sa kruhy). Tento nedostatok možno odstrániť pridaním príkazu `Brush.Style:= bsClear;` ktorý vypína vykresľovanie „vnútra kružnice“. Štandardne je zrejme hodnota vlastnosti `Style` výplne `Brush` nastavená na `bsSolid`, čo znamená úplná výplň, niekedy však potrebujeme „žiadna výplň“ - `bsClear`. Všimnite si, že príkaz `Brush.Style := bsClear;` vyradí príkaz `Brush.Color := random((255*255*255)+1);` „z hry“.

Vpravo obrázok príkladu G2g po použití príkazu `Brush.Style:= bsClear;` a viacnásobnom kliknutí na tlačidlo KRUŽNICA.



📖 Pomocou posledných procedúr si môžeme trocha objasniť aj príkaz `randomize`. Ak ho nevediete, vždy po spustení programu a kliknutí na tlačidlo sa nakreslí rovnaký prvý útvar napriek tomu, že mnohé jeho vlastnosti sú generované náhodne. Po pridaní príkazu `randomize` sa tak už diať nebude.

- ✍️ 🖥️ Naprogramujte vykreslenie kruhového terča pre lukostreľbu s piatimi rôznofarebnými kruhmi.
- ✍️ 🖥️ Naprogramujte vykreslenie „čokoládovej“ hviezdy Orion.
- ✍️ 🖥️ Naprogramujte jednotlivé steny hracej kocky (body od 1 po 6).

Príklad G2h*

Vytvorte program simulujúci hod hracou kockou (môže padnúť 1, 2, 3, 4, 5 alebo 6).

Ak padne 6, nech program počká 2 sekundy a vygeneruje nový hod.

a) Číslo, ktoré padlo, nech sa zobrazí ako zväčšená číslica vo formulári.

b) Číslo, ktoré padne, nech sa vykreslí ako stena hracej kocky s príslušným počtom bodov.

Riešenie

Odporúčanie: Pred počítaním súradníc bodov hracej steny si nakreslite osem vodorovných a zvislých rovnako širokých pruhov, každý šírky 50 bodov plátna.

implementation

{\$R *.dfm}

```
procedure TForm1.btHodClick(Sender: TObject);
```

```
var Padla: integer;
```

```
begin
```

```
  Memo1.Clear;
```

```
  repeat //opakovanie príkazov
```

```
    Padla:= random(6)+1;
```

```
    Memo1.Lines.Add(IntToStr(Padla));
```

```
    if Padla = 6 then sleep(2000);
```

```
  until Padla<>6; //až kým nepadne iné číslo ako 6
```

```
end;
```

```
procedure Jednotka;
```

```
begin
```

```
with Form1.Canvas do
```

```
begin
```

```
  Ellipse(200,200,250,250);
```

```
end;
```

```
end;
```

```
procedure Trojka;
```

```
begin
```

```
with Form1.Canvas do
```

```
begin
```

```
  Ellipse(100,300,150,350);
```

```
  Ellipse(200,200,250,250);
```

```
  Ellipse(300,100,350,150);
```

```
end;
```

```
end;
```

```
procedure Dvojka;
```

```
begin
```

```
with Form1.Canvas do
```

```
begin
```

```
  Ellipse(100,300,150,350);
```

```
  Ellipse(300,100,350,150);
```

```
end;
```

```
end;
```

```
procedure Stvorka;
```

```
begin
```

```
with Form1.Canvas do
```

```
begin
```

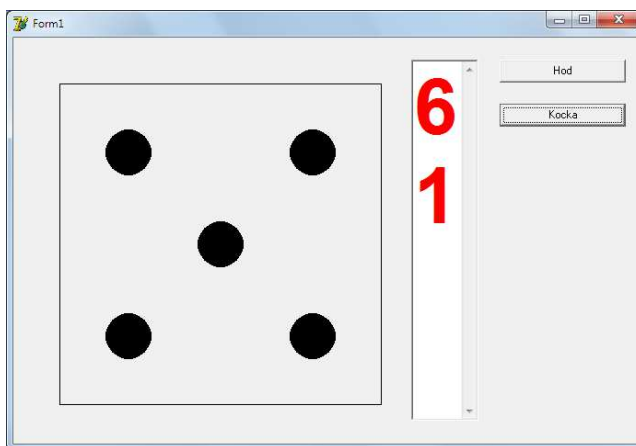
```
  Ellipse(100,100,150,150);
```

```
  Ellipse(300,100,350,150);
```

```
  Ellipse(100,300,150,350);
```

```
  Ellipse(300,300,350,350);
```

```
end; end;
```




```
procedure Patka;  
begin  
with Form1.Canvas do  
begin  
    Ellipse(100,100,150,150);  
    Ellipse(100,300,150,350);  
    Ellipse(200,200,250,250);  
    Ellipse(300,100,350,150);  
    Ellipse(300,300,350,350);  
end;  
end;
```

```
procedure Sestka;  
begin  
with Form1.Canvas do  
begin  
    Ellipse(100,100,150,150);  
    Ellipse(200,100,250,150);  
    Ellipse(300,100,350,150);  
    Ellipse(100,300,150,350);  
    Ellipse(200,300,250,350);  
    Ellipse(300,300,350,350);  
end;  
end;
```

```
procedure HOD;  
var Padla: integer;  
begin  
with Form1.Canvas do  
begin  
    Brush.Color:= Form1.Color;  
    Pen.Color:= clBlack;  
    Rectangle(50,50,400,400);  
    Brush.Color:= clBlack;  
end;  
Padla:= random(6)+1;  
beep; beep; beep;  
case Padla of  
    1: jednotka;           //vykresli stenu hracej kocky „s jedným bodom“  
    2: dvojka;            //vykresli stenu hracej kocky „s dvoma bodmi“  
    3: trojka;  
    4: stvorka;  
    5: patka;  
    6: begin  
        sestka;           //vykresli stenu hracej kocky „so šiestimi bodmi“  
        sleep(2000);     //počkaj dve sekundy  
        HOD;             //rekurzia, padla 6-ka, opakuj hod  
    end;  
end;  
end;
```

```
procedure TForm1.btKockaClick(Sender: TObject);  
begin  
HOD;  
end;
```

```
initialization  
randomize;  
end.
```

 V programe sme použili podmienený príkaz **case**, ktorý realizuje n-árne vetvenie ($n \geq 2$).

Má tvar:	case v of	kde v je tzv. výberový výraz ordinálneho typu,
	h1 : p1;	h1, h2 až hn sú hodnoty rovnakého typu ako výberový výraz
	h2 : p2;	a p, p1, p2 až pn sú príkazy
	...	
	hn : pn	
	[else p]	do hranatých zátvoriek sa umiestňuje nepovinná časť
	end	príkaz case končí vyhradeným slovom end

Vykonanie: Vyhodnotí sa výberový výraz a vykoná príkaz, predznačený hodnotou, ktorú nadobudol výberový výraz. Ak výraz v nenadobudne ani jednu z hodnôt h1 až hn a príkaz case obsahuje časť else, vykoná sa príkaz p; ak príkaz case neobsahuje časť else, príkaz case je bez účinku.

Napríklad

case Cislo of	case Znak of
1 : mena:= 'euro';	'A'..'Z','a'..'z': Label1.Caption:= 'písmeno angl. abecedy';
2..4: mena:= 'eurá';	'0'..'9': Label1.Caption:= 'číslica';
else mena:= 'eur';	else Label1.Caption:= 'iný znak'
end;	end;

For-cyklus v grafických príkladoch

V príkladoch G2 sme väčšinou opakovaně stlačili kláves vykresľujúci obrazec. Príkaz for nám ľahko zabezpečí viacnásobné vykreslenie obrazca. Jednotlivé procedúry stačí doplniť nasledovnými riadkami:

1. medzi hlavičku procedúry začínajúcu slovami procedure TForm1... a slovo begin treba vložiť var i: integer;
ktorý počítaču hovorí, že v procedúre je použitá premenná s názvom i a jej obsahom bude celé číslo
2. ak sa majú príkazy vykresľujúce obrazec opakovať napríklad 100-krát, treba príkazovú časť začínajúcu slovom begin doplniť nasledovne
begin
for i := 1 to 100 do //zabezpečí vykonanie príkazov zapísaných medzi begin a end stokrát
begin
with Canvas do //kresli do plátna
begin
príkazy vykresľujúce obrázok
end;
end;
end;
end; // koniec celej procedúry, k nemu prislúchajúci begin je nad príkazom randomize

☛ Poznámka

Ak chceme vidieť, ako sú postupne vykresľované jednotlivé obrazce, je potrebné po vykreslení obrazca zastaviť vykonávanie programu na zvolený čas. Príkaz sleep (*milisekundy*); zastaví beh programu na zadaný počet milisekúnd. Všetko si ozrejníme v nasledujúcom príklade.

Príklad G3a

Čo nakreslí nižšie uvedená procedúra?

Cez príkaz InputBox (nie je komponent, nevkladá sa do formulára, stačí napísať!) môžeme zadať počet opakovaní pre for-cyklus. Pozor na apostrofy, to sú tie „čiarky“ okolo 'Ružica', 'Počet lúčov', '50' a 'Skončil som!'. Pri slovenskej klávesnici sa vkladajú cez ľavé Alt+39, v anglickej sú pod „anglickými“ úvodzovkami, vedľa „slovenského“ paragrafu.

```
procedure TForm1.btRuzicaClick(Sender: TObject);
var i, Pocet: integer; //v procedúre použijeme dve premenné i a Pocet, obe celočíselné
begin
Pocet:= StrToInt ( InputBox ( 'Ružica' , 'Počet lúčov' , '50' )); //do premennej Pocet zadáme cez
for i:= 1 to Pocet do //príkaz InputBox počet opakovaní pre for
begin
with Canvas do //vykreslí sa obrázok
begin
Pen.Color:= random(65536);
Pen.Width:= random(5)+1;
MoveTo(200,200);
LineTo(random(400), random(400));
end;
sleep(200); //počká sa 200 milisekúnd
end;
Canvas.TextOut(400,10,'Skončil som!'); //vypíše sa Skončil som!
end;
```

Príklad G3b

Ak sa pozrieme na príklad G1a - mriežku, vidíme, že sa opakujú príkazy MoveTo a LineTo so zmenenými súradnicami. Keďže zmeny súradníc sú násobky čísla 100 (100, 200, 300 a 400), vieme použiť for-cyklus. Tu je riešenie:

```

procedure TForm1.btMriezkaClick(Sender: TObject);
var i: integer;
begin
with Canvas do
begin
//najprv vodorovné čiary, sú štyri, preto
for i:= 1 to 4 do
begin
MoveTo(0, i*100);
LineTo(400, i*100);
sleep(500); //aby sme videli, ako sa "to" kreslí
end;
//teraz zvislé čiary
for i:= 1 to 4 do
begin
MoveTo(i*100, 0);
LineTo(i*100, 400);
sleep(500); //aby sme videli, ako sa "to" kreslí
end;
end;
end;
end;

```

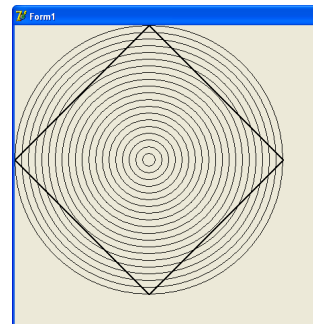
Príklad G3c

Aj v príklade Očný klam (obrázok vpravo) vidíme opakované vykresľovanie kružníc s pravidelne sa zmenšujúcim priemerom. Lahôdka pre príkaz for, ale treba spraviť dobrú analýzu problému.

Analýza

- kružnice sú postupne vykresľované do štvorcov so súradnicami

0, 0	400, 400	hodnoty v druhom stĺpci možno zapísať tiež	400 - 0
10, 10	390, 390		400 - 10
20, 20	380, 380		400 - 20
30, 30	370, 370		400 - 30
...
180, 180	220, 220		400 - 180
190, 190	210, 210		400 - 190
- teda RiadiacaPremennáCyklu môže nadobúdať hodnoty 0, 1, 2, 3,..., 18, 19 a po jej vynásobení desiatimi dostávame požadované súradnice 0, 10, 20, 30,..., 180, 190 resp. odčítaním od štyristo dostávame 400, 390, 380, 370,...






```

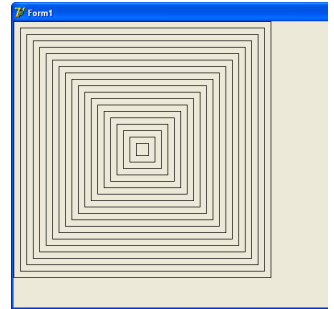
procedure TForm1.btKlamClick(Sender: TObject);
var Pocitadlo, Pocet: integer; //namiesto označenia i sme zvolili slovo Pocitadlo
begin
Pocet:= 19;
with Canvas do
begin

```



```
for Pocitadlo:= 0 to Pocet do
begin
    Ellipse(10*Pocitadlo,10*Pocitadlo,400-10*Pocitadlo,400-10*Pocitadlo);
    //doplňte sleep(500);
end;
Pen.Width:= 2; // nastaví hrúbku pera na 2 a v ďalšom vykreslí otočený štvorec
MoveTo(200,0); LineTo(400,200); LineTo(200,400); LineTo(0,200); LineTo(200,0);
end;
end;
```

- ✎  Naprogramujte terč s piatimi kruhmi pomocou príkazu for.
- ✎  Naprogramujte obrázok vpravo. Štvorce kreslite od najmenšieho po najväčší, aby vám menšie štvorce „nemizli“, použite príkaz `Brush.Style:= bsClear;` (popísaný vyššie).
- ✎  V príklade G3c sa pokúste naprogramovať kružnice od najmenšej po najväčšiu. Je nevyhnutné použiť príkaz `Brush.Style:= bsClear;`



Poznámky:

Udalosti pri práci s myšou

Ako sme napísali už v úvode zbičky, beh programu v Delphi je riadený udalosťami, ktoré vyvoláva užívateľ. Pri práci s myšou v aplikácii môže užívateľ vyvolať najmä tieto udalosti, a teda programátor využiť pri tvorbe programu:

OnClick	užívateľ klikol na ľavé tlačidlo myši s kurzorom nad komponentom (túto udalosť ButtonClick sme využívali aj doteraz)
OnDbClick	užívateľ dvojklíkol na ľavé tlačidlo myši nad komponentom
OnMouseMove	pohybuje myšou nad komponentom pri stlačení niektorom tlačidlo myši
OnMouseDown	stlačil tlačidlo myši
OnMouseUp	uvoľnil tlačidlo myši

Tieto udalosti si ozrejmime pri kreslení obrázkov, takže použijeme komponent Image zo záložky Additional. Veľmi jednoducho a rýchlo sa môžeme dopracovať až k možnosti kresliť obrázky ako ten vpravo. Ale poďme postupne.



Príklad M1

Vytvorte program, ktorý po kliknutí na ľavé tlačidlo myši v komponente Image vypíše na pozíciu kurzora slovo Začiatok a po uvoľnení tlačidla slovo Koniec.

Riešenie

Po vložení komponentu Image1 má programátor v záložke Events k dispozícii udalosť – obrázok vpravo. Po dvojklíku do bieleho poľa vpravo od udalosti OnMouseDown sa do programu vloží procedúra

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

```
end;
```

Z parametrov uvedených v hlavičke procedúry nás zaujíma parameter Shift, ktorý do procedúry dovezie informáciu o stlačených tlačidlách na myši. Môžu nastať situácie, že sa

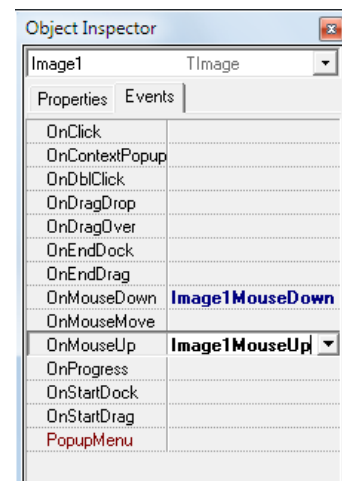
Shift = []	nebolo stlačené žiadne tlačidlo myši
Shift = [ssLeft]	bolo stlačené ľavé tlačidlo myši
Shift = [ssRight]	bolo stlačené pravé tlačidlo myši
Shift = [ssLeft, ssRight]	boli stlačené naraz obe tlačidlá

(zápis možno čítať „Shift sa rovná množine hodnôt...“).

Zároveň parametre X a Y dovezú do procedúry aktuálnu pozíciu kurzora v plátne.

Keď medzi begin a end dopíšeme príkaz `Image1.Canvas.TextOut(X, Y, 'Začiatok');` docielime, že po stlačení tlačidla myši sa na pozíciu kurzora v plátne vypíše slovo Začiatok.


Po vložení procedúry prislúchajúcej k udalosti OnMouseUp môžeme docieľiť, že na mieste uvoľnenia tlačidla myši sa vypíše slovo Koniec.



```

procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
begin
Image1.Canvas.TextOut(X, Y, 'Koniec' );
end;

```

 Vyskúšajte si stlačenie ľavého, pravého aj oboch tlačidiel myši súčasne, prípadne uvoľnenie najprv ľavého a až potom pravého tlačidla, klik a pod.

Príklad M2

Vytvorte program, ktorý umožní kresliť čiary voľnou rukou pri stlačení ľavom tlačidle myši.

Analýza

Použijeme udalosť `OnMouseMove`, pri ktorej, ak užívateľ drží stlačené ľavé tlačidlo myši, kreslia sa úsečky s koncovými bodmi `[X, Y]` – aktuálna poloha kurzora myši. Počiatočný bod novej úsečky je koncovým bodom predchádzajúcej úsečky.

```

procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
if Shift = [ssLeft] then Image1.Canvas.LineTo(X, Y);
//ak je stlačené ľavé tlačidlo myši, nakresli úsečku s koncovým bodom [X,Y]
end;

```

Ak si vyskúšate uvedenú procedúru, zistíte, že problémom je vždy začiatok kreslenia, ktorý je po spustení programu v bode `[0,0]` a pri každom novom stlačení ľavého tlačidla myši v poslednej pozícii kurzora pri predchádzajúcom kreslení. My však potrebujeme dosiahnuť situáciu, aby počiatočný bod nového kreslenia bol vždy tam, kde stlačíme ľavé tlačidlo myši. Tento problém nám rieši udalosť `OnMouseDown`, resp. nasledujúca procedúra.

```

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

```

Navrhnete udalosť, ktorá by bola vhodná na zmazanie plátna a doplňte do programu.

Riešenie

Nám sa pozdáva dvojklik ľavým tlačidlom myši do plátna (udalosť `OnDbClick`), čo znamená doplniť program procedúrou

```

procedure TForm1.Image1DbClick(Sender: TObject);
begin
Image1.Canvas.FillRect(Image1.ClientRect);
end;

```

Príklad M3

Predchádzajúci program doplňte o procedúru, ktorá umožní pri stlačení pravom tlačidle myši mazať časť obrazu pod kurzorom myši.

Riešenie

Keďže sa opäť využíva udalosť `OnMouseMove`, len stlačené je tentoraz pravé tlačidlo myši, doplníme už vytvorenú procedúru o rozhodovanie, či je stlačené ľavé alebo pravé tlačidlo myši. Pri stlačení ľavom tlačidle myši sa má kresliť čiernou farbou, pri stlačení pravom tlačidle myši sa má „kresliť“ farbou pozadia, čo zabezpečí mazanie pod kurzorom myši. Aby mazanie bolo pohodlnejšie, zväčšíme šírku pera pri mazaní na 10.

```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
with Image1.Canvas do
begin
if Shift = [ssLeft]
then begin
Pen.Color:= clBlack;
Pen.Width:= 1;
LineTo(X, Y);
end;
if Shift = [ssRight]
then begin
Pen.Color:= Brush.Color;
Pen.Width:= 10;
LineTo(X, Y);
end;
end;
end;
```

Zrejme rovnaký výsledok by sme dostali, keby sme použili jeden úplný príkaz if (aj keď treťou možnosťou je, že sú súčasne stlačené obe tlačidlá myši).

Všimnite si, že príkaz LineTo(X,Y) je použitý dvakrát, nebolo by jednoduchšie uviesť ho len raz po príkazoch if (nastavení parametrov pera)?

K dokonalejšiemu grafickému editoru nám chýba najmä možnosť meniť farbu a šírku pera. Tu je možné elegantné riešenie.

Príklad M4

Program z predchádzajúceho príkladu doplňte o tlačidlo ZMEŇ FARBU s využitím komponentu ColorDialog, ktorý umožňuje meniť farbu pera a o komponent SpinEdit, ktorý umožňuje meniť šírku pera.

Riešenie

Komponent ColorDialog nájdeme v záložke Dialogs a komponent SpinEdit v záložke Samples.

Podstatná časť programu:

```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
with Image1.Canvas do
begin
if Shift = [ssLeft]
then begin
Pen.Color:= ColorDialog1.Color;           //nastaví sa farba pera z ColorDialog
Pen.Width:= SpinEdit1.Value;             //nastaví sa šírka pera zo SpinEdit
LineTo(X, Y);
end;
if Shift = [ssRight]
then begin
Pen.Color:= Brush.Color;                 //nastaví sa farba pera na farbu pozadia
Pen.Width:= SpinEdit1.Value + 3;         //nastaví sa šírka zo SpinEdit zväčšená o 3
LineTo(X, Y);
end;
end;
end; end;
```

```

procedure TForm1.btZmenFarbuClick(Sender: TObject);
begin
Form1.ColorDialog1.Execute;           //zobrazí ColorDialog
Image1.Canvas.Pen.Color:= ColorDialog1.Color; //priradí farbu peru podľa výberu
end;

procedure TForm1.TrackBar1Change(Sender: TObject);
begin
Image1.Canvas.Pen.Width:= SpinEdit1.Value; //nastaví šírku pera podľa hodnoty v SpEd
end;

```

„Bonusová“ procedúra nám umožní, po kliknutí na tlačidlo ULOŽ DO SÚBORU, uložiť nakreslený obrázok do aktuálneho priečinka pod zadaným názvom s príponou bmp.

```

procedure TForm1.btUlozDoSuboruClick(Sender: TObject);
begin
Image1.Picture.SaveToFile( InputBox( 'Uložiť obrázok' , 'Názov súboru' , 'obr-1' ) + '.bmp' );
end;

```

Príklad M5

Udalosť OnMouseMove - pohyb myšou na komponent môžeme využiť rôznymi spôsobmi. Vtipným variantom je skrytie komponentu, v našom prípade tlačidla Button, keď naň užívateľ presunie kurzor. Vytvorte vizuálny návrh formulára podobný nášmu vpravo. Nech program neumožňuje odpovedať – stlačiť tlačidlo Nie. Docielime to zmenou vlastnosti btNie1.Visible na False po presune kurzora na toto tlačidlo. Ak sa zároveň zmení vlastnosť Visible analogického tlačidla btNie2 na True, docielime efekt, ako by tlačidlo Nie uskakovalo pred stlačením užívateľom.

Riešenie

```

procedure TForm1.FormCreate(Sender: TObject);
begin
Image1.Picture.LoadFromFile('portret.jpg');
btNie2.Visible:= False;
end;

procedure TForm1.btAnoClick(Sender: TObject);
begin
ShowMessage('Správna odpoveď');
Application.Terminate;
end;

procedure TForm1.btNie1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
btNie1.Visible:= False;
btNie2.Visible:= True;
end;

procedure TForm1.btNie2MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
btNie2.Visible:= False;
btNie1.Visible:= True;
end;

```



Časovač (Timer)

Ak programátor potrebuje, aby sa v pravidelných časových intervaloch vykonávali určité príkazy, môže použiť komponent Timer, ktorý nájde v záložke SYSTEM. Vo vlastnostiach komponentu Timer môže nastaviť tzv. Interval, v ktorom určí, po koľkých milisekundách sa má opäť vykonať procedúra Timer1Timer (udalosť OnTimer), resp. príkazy v nej uvedené. Programátor tiež cez vlastnosť Enabled, s hodnotami True alebo False, môže riadiť spúšťanie a zastavovanie časovača. Všetko si ozrejmime na príkladoch.

Príklad T1

Príklad G3a prerobte tak, aby úlohu for-cyklu prevzal časovač. Vykresľovanie nech sa začne hneď po spustení programu a ukončí zastavením programu.

Riešenie

Po vložení komponentu Timer do formulára naň dvojklikneme, čo spôsobí vloženie procedúry procedure TForm1.Timer1Timer(Sender: TObject); Do jej príkazovej časti skopírujeme príkazy, ktoré kreslia obrázok z príkladu G3a (pozri nižšie) a úloha je vyriešená. Keďže Timer na vlastnosť Enabled – aktivovať, zapnutú na True, hneď po spustení sa začnú v sekundových intervaloch (Interval = 1000 ms) vykonávať príkazy uvedené medzi begin a end. Pre zrýchlenie vykresľovania môžeme nastaviť interval napríklad na 300.

```
procedure TForm1.Timer1Timer(Sender: TObject);  
begin  
with Image1.Canvas do  
begin  
    Pen.Color:= random(65536);  
    Pen.Width:= random(5)+1;  
    MoveTo(300,300);  
    LineTo(random(600), random(600));  
end;  
end;
```

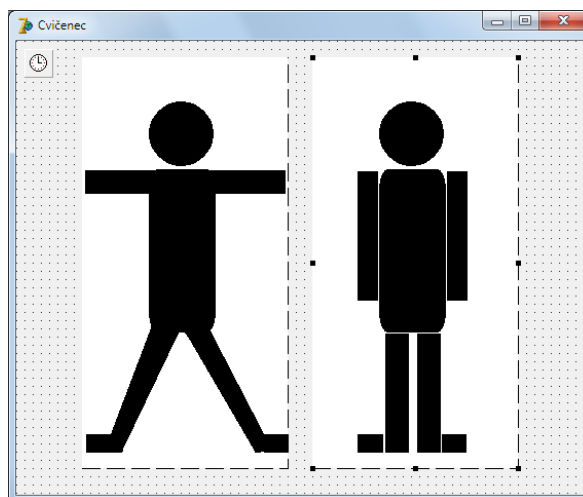
Program doplníte o tlačidlá ŠTART a STOP, ktoré umožnia spustiť a zastaviť vykresľovanie po spustení programu. Nezabudnite pred spustením programu nastaviť vlastnosť Enabled časovača na False.

Príklad T2

Vytvorte program – animáciu cvičenia cvičenca, ktorý bude rozpažovať a upažovať ruky striedavo v pravidelných časových intervaloch.

Riešenie

Podstatou animácie je striedanie dvoch obrázkov, ktoré si môžeme nakresliť napríklad v Skicári (obrázok vpravo) a uložiť do priečinka s programom. Oba obrázky musia mať rovnaké rozmery a cez Image1 a Image2 vlastnosť Picture – Load... ich umiestnime na seba (nie ako v obrázku, kde sú vedľa seba). Predpokladajme, že obrázok vložený do Image2 je v pozadí, teda obrázok v Image1 je nad obrázkom vloženým do Image2. Teraz už len stačí v pravidelných časových intervaloch zapínať a vypínať zobrazovanie horného obrázka. To nám zabezpečí komponent Timer, resp. jeho udalosť OnTimer. Potrebujeme, aby na každý „tik“ časovača sa striedavo zapínalo a vypínalo



zobrazovanie horného obrázka. Globálna premenná prvý typu boolean, vykonávaním príkazu prvý:= not prvý; striedavo nadobúda hodnoty false a true, čo v spojení s vlastnosťou Image1.Visible zabezpečuje vypínanie a zapínanie zobrazovania obrázka.

```
implementation
{$R *.dfm}

var prvý:boolean; //prvý musí byť globálna premenná

procedure TForm1.Timer1Timer(Sender: TObject);
begin
if prvý
then Image1.Visible:= True
else Image1.Visible:= False;
prvý := not prvý
end;
```

V tomto prípade ani nevaďí, že sme nenastavili počiatočnú hodnotu premennej prvý. Program doplňte o tlačidlo Štart/Stop, pomocou ktorého spúšťate a zastavujete cvičenie cvičenca. Uvádzame jedno z možných riešení, nezabudnite pred spustením programu nastaviť vlastnosť Enabled v Timer1 na False.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if Button1.Caption = 'Štart'
then begin
    Timer1.Enabled:= True;
    Button1.Caption:= 'Stop'
    end
else begin
    Timer1.Enabled:= False;
    Button1.Caption:= 'Štart'
    end
end;
```

Príklad T3

Naprogramujte digitálne stopky. Program nech umožňuje spustiť, zastaviť a vynulovať stopky.

Riešenie

Riešenie uvádzame bez komentára, h znamená hodiny, m minúty, s sekundy a s100 znamená stotiny sekúnd. Interval v Timer1 je teoreticky nastavený na 10.

```
var h, m, s, s100: integer;

procedure TForm1.FormActivate(Sender: TObject);
begin
h:= 0; m:= 0; s:= 0; s100:= 0
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
s100:= s100 + 1;
if s100 = 100
then begin
```




```
s:= s + 1; s100:= 0;
end;
if s = 60
then begin
    m:= m + 1; s:= 0;
    end;
if m = 60
then begin
    h:= h + 1; m:= 0;
    end;
Label1.Caption:= Format( '%2d : %2d : %2d : %2d' , [ h , m , s , s100 ] );
end;

procedure TForm1.btStartClick(Sender: TObject);
begin
Timer1.Enabled:=True
end;

procedure TForm1.btStopClick(Sender: TObject);
begin
Timer1.Enabled:=False
end;

procedure TForm1.btNulujClick(Sender: TObject);
begin
s:= 0; m:= 0; h:= 0; s100:= 0;
Label1.Caption:= Format( '%2d : %2d : %2d : %2d' , [ h , m , s , s100 ] );
end;
```

Príklad T4

Vytvorte program simulujúci prácu semafora.

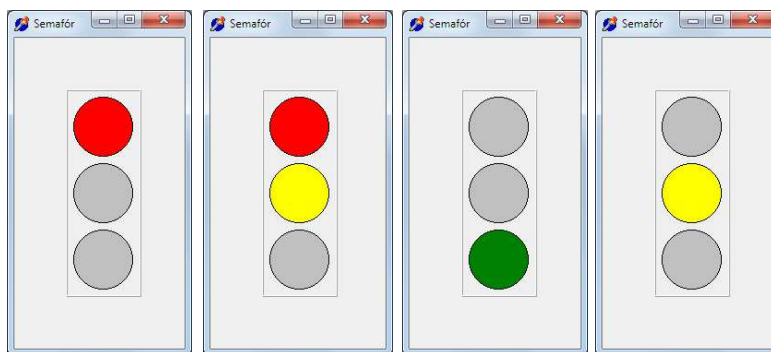
Analýza

Na svetlách semafora sa pravidelne cyklicky opakujú štyri stavy, ktoré sme znázornili na obrázkoch vpravo.

Na každý klik časovača sa má vypnúť aktuálny stav a zapnúť nasledujúci stav. Zapnúť alebo vypnúť svetlo znamená vhodne zmeniť jeho farbu. Použili sme komponent Shape zo záložky Additional, u ktorého sme zmenili vlastnosť Shape na `stCircle`.

Svetlá (Shape) sme rámovali komponentom Bevel.

Prvý stav, svieti červená, sme označili ako stav nula. Po uplynutí intervalu nastaveného v časovači sa má stav nula zmeniť na stav jedna, čo možno dosiahnuť pričítaním jednotky k hodnote premennej Stav. Po stave jedna nasledujú postupne stavy dva a tri. Po stave tri má opäť nastav stav nula, čo sa ľahko dosiahne príkazom `Stav:= Stav mod 4`; Takže potrebujeme rozlíšiť štyri situácie, preto sme použili príkaz `case` vysvetlený v príklade G2h*.



Riešenie

```

var Stav: integer;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
case Stav of
0:begin
    Shape1.Brush.Color:=clRed;
    Shape2.Brush.Color:=clSilver;
    Shape3.Brush.Color:=clSilver;
    end;
1:begin
    Shape1.Brush.Color:=clRed;
    Shape2.Brush.Color:=clYellow;
    Shape3.Brush.Color:=clSilver;
    end;
2:begin
    Shape1.Brush.Color:=clSilver;
    Shape2.Brush.Color:=clSilver;
    Shape3.Brush.Color:=clGreen
    end;
3:begin
    Shape1.Brush.Color:=clSilver;
    Shape2.Brush.Color:=clYellow;
    Shape3.Brush.Color:=clSilver;
    end;
end;
Stav:= Stav + 1;
Stav:= Stav mod 4;
end;

initialization
Stav:= 0;
end.

```

Program upravte tak, aby pred zhasnutím zeleného a rozsvietením žltého svetla blikalo určitý čas zelené svetlo. Návod: treba pridať ďalší stav aj komponent Timer.

Príklad T5

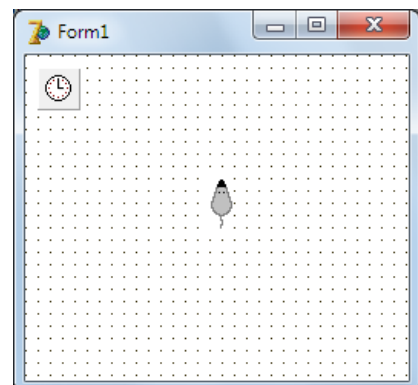
Na ukážku ovládania pohybu „myši“ kurzorovými šípkami vo formulári sme v Skicári postupne nakreslili štyri podoby myši



(mysL.bmp, mysH.bmp, mysD.bmp a mysP.bmp) a naprogramovali jej pohyb.

Prvá procedúra (vlastnosť OnKeyDown formulára) sleduje stláčanie kurzorových šípok a podľa toho nahráva obrázok a nastavuje smer.

Druhá procedúra (udalosť OnTimer komponentu Timer) zabezpečuje pohyb vo zvolenom smere a testuje, či „myš nevybehla z formulára“.



```
const MYS = 34;           // rozmer myši v px
      KROK = 10;
var   smer: char;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
case Key of
      vk_up: begin                // Key obsahuje kód stlačeného klávesu
                // ak bola stlačená šípka nahor
                Image1.Picture.Bitmap.LoadFromFile('mysH.bmp');
                smer:='h'
            end;
      vk_down: begin              // ak bola stlačená šípka nadol
                Image1.Picture.Bitmap.LoadFromFile('mysD.bmp');
                smer:='d'
            end;
      vk_right: begin             // ak bola stlačená šípka doprava
                Image1.Picture.Bitmap.LoadFromFile('mysP.bmp');
                smer:='p'
            end;
      vk_left: begin              // ak bola stlačená šípka doľava
                Image1.Picture.Bitmap.LoadFromFile('mysL.bmp');
                smer:='l';
            end;
end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
case smer of
      'h': Image1.Top:= Image1.Top - KROK;
      'd': Image1.Top:= Image1.Top + KROK;
      'p': Image1.Left:= Image1.Left + KROK;
      'l': Image1.Left:= Image1.Left - KROK;
end;
If (Image1.Top < 0) or (Image1.Top > ClientHeight - MYS)
  or (Image1.Left < 0) or (Image1.Left > ClientWidth - MYS)
then begin
      Timer1.Enabled:=False;
      ShowMessage('O H R A D A !!!');
      Application.Terminate;
  end
end;
end;
```

Poznámky:

Ťahák z grafiky

Canvas – plátno

Súradnicová sústava:



Útvary:

MoveTo(X,Y)

LineTo(X,Y)

Rectangle(X_{LH},Y_{LH},X_{PD},Y_{PD})

Ellipse(X_{LH},Y_{LH},X_{PD},Y_{PD})

TextOut(X,Y, 'reťazec')

Pixels[X,Y] := clFarba

Polygon([Point(x₁,y₁), Point(x₂,y₂),...]) nakreslí N-uholník s vrcholmi [x_i, y_i], i = 1, 2,...,N

Polyline([Point(x₁,y₁), Point(x₂,y₂),...]) nakreslí lomenú čiaru s vrcholmi [x_i, y_i], i = 1, 2,...,N

FillRect(X_{LH},Y_{LH},X_{PD},Y_{PD})

FrameRect(X_{LH},Y_{LH},X_{PD},Y_{PD})

RoundRect(X_{LH},Y_{LH},X_{PD},Y_{PD},S,V)

Chord(X_{LH},Y_{LH},X_{PD},Y_{PD},X_L,Y_L,X_P,Y_P)

Pie(X_{LH},Y_{LH},X_{PD},Y_{PD},X_Z,Y_Z,X_K,Y_K)

Arc(X_{LH},Y_{LH},X_{PD},Y_{PD},X_Z,Y_Z,X_K,Y_K)

Využívajte kontextovú pomoc:

1. napíšte aspoň začiatok neznámeho príkazu
 2. umiestnite do neho kurzor
 3. stlačte F1
 4. vyhľadajte aj Delphi example – príklad
1. umiestnite kurzor do vlastnosti objektu
 2. kliknite pri stlačení Ctrl
 3. zobrazia sa všetky možnosti danej vlastnosti

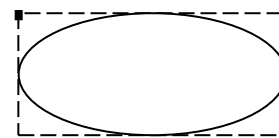
zmení aktuálnu pozíciu pera na X,Y

nakreslí úsečku z aktuálnej pozície pera do bodu X,Y

nakreslí obdĺžnik

[X_{LH},Y_{LH}]

nakreslí vpísanú elipsu



vypíše reťazec

vykreslí bod zadanej farby,

napr. clRed, clBlue, clYellow

nakreslí N-uholník s vrcholmi [x_i, y_i], i = 1, 2,...,N

nakreslí lomenú čiaru s vrcholmi [x_i, y_i], i = 1, 2,...,N

nakreslí obdĺžnik bez obrysov (iba výplň)

nakreslí iba obrysy obdĺžnika (farbou Brush)

nakr. obdĺž. so zaoblenými rohmi (S, V – šírka, výška elipsy)

nakreslí elipsový odsek určený spojnicou [X_L,Y_L], [X_P,Y_P]

napr. Chord(0,0,200,200,0,100,200,100); vykreslí dolný polkruh

nakreslí elipsový výsek, stred spojený s [X_Z,Y_Z] a [X_K,Y_K]

napr. Pie(0,0,200,200,100,0,200,100); vykreslí 3/4 kruh

nakreslí časť vpísanej elipsy „od“ [X_Z,Y_Z] „po“ [X_K,Y_K]

napr. Arc(0,0,200,200,100,0,200,100); vykreslí 3/4 kružnicu

Farby možno namiešať aj zápisom **RGB (číslo, číslo, číslo)** kde číslo je celé číslo od 0 po 255 a R znamená red, G - green a B - blue; preto napr. clBlack = RGB(0,0,0); clWhite = RGB(255,255,255); clRed = RGB(128,0,0); clYellow = RGB(255,255,0) atď.

Funkcia random (*prírodné číslo*) vráti celé číslo z intervalu <0, prírodné číslo - 1>.

Funkcia random vráti reálne číslo z intervalu <0,1>.

Pred funkciou random sa zvykne jedenkrát použiť príkaz randomize.

```
procedure TForm1.btZmazClick(Sender: TObject); //zmaže plátno
begin
Canvas.Brush.Color:= Form1.Color; //nastavenie farby výplne na farbu formulára
Canvas.FillRect(ClientRect);
end;
```

Vlastnosť:

Pen - pero

Canvas.Pen.Color

farba pera, napr. Canvas.Pen.Color:= clFuchsia;

Canvas.Pen.Width

šírka pera, napr. Canvas.Pen.Width:= 10;

Canvas.Pen.Style

štýl kreslenia - pozri nižšie Konštanty vlastností

Canvas.Pen.Mode

kresliaci režim - pozri nižšie Konštanty vlastností

Brush - štetec

Canvas.Brush.Color
 Canvas.Brush.Style

farba výplne, napr. Canvas.Brush.Color:= clSilver;
 kresliaci režim - pozri nižšie Konštanty vlastností

Font - znak

Canvas.Font.Color
 Canvas.Font.Height
 ...

farba písma, napr. Canvas.Font.Color:= clNavy;
 veľkosť písma, napr. Canvas.Font.Height:= 32;

Konštanty vlastností:**Pen.Style:**

psSolid	plná čiara
psDash	čiarkovaná čiara
psDot	bodkovaná čiara
psDashDot	-.-.-.
psDashDotDot	-..-..
psClear	nie je nič kreslené
psInsideFrame	čiary vo vnútri rámu uzavretých tvarov

Brush.Style:

bsSolid	úplná výplň
bsClear	žiadna výplň
bsBDiagonal	šrafovanie vpravo hore pod 45°
bsFDiagonal	šrafovanie vľavo hore pod 45°
bsCross	mriežka
bsDiagCross	mriežka pod 45°
bsHorizontal	horizontálne šrafovanie
bsVertical	vertikálne šrafovanie

Pen.Mode:

pmBlack	vždy čierna
pmWhite	vždy biela
pmNop	nezmenená (výsledok je neviditeľný)
pmNot	inverzná k farbe pozadia
pmCopy	určená vlastnosťou Pen.Color
pmNotCopy	inverzná k farbe Pen.Color

OBSAH

Úvodné príklady.....	3
SEKVENCIA	7
VETVENIE.....	15
Podmienený príkaz if.....	15
CYKLUS.....	25
Cyklus s pevným počtom opakovaní.....	25
Príkaz for	25
Vyhľadávanie v skupine údajov	33
Cyklus s podmienkou ukončenia.....	40
Príkaz while	40
Príkaz repeat	44
Kedy ktorý cyklus?.....	49
Pohrajme sa s reťazcami.....	51
Úvod do grafiky	61
For-cyklus v grafických príkladoch.....	71
Udalosti pri práci s myšou	75
Časovač (Timer).....	79
Ťahák z grafiky	85

Použitá literatúra:

Kalaš I. a kol.: Informatika pre stredné školy (učebnica), SPN

Blaho A.: Programovanie v Delphi (učebnica), SPN