Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Programovanie 7

Predmet: Programovanie

Línia: Vlastný odborový kontext informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

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

Programovanie 7

Identifikácia modulu

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

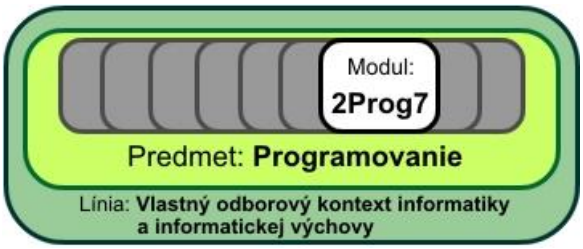
Garant predmetu:

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

Autori:

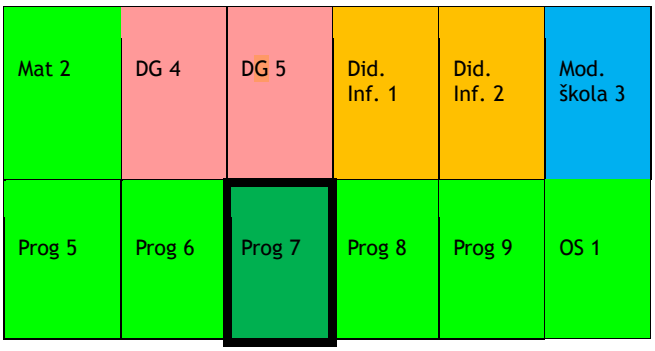
RNDr. Andrej Blaho
KAI FMFI UK, Bratislave
RNDr. Ľubomír Salanci, PhD.
KZVI FMFI UK, Bratislave

Zaradenie modulu



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

Druhý semester vzdelávania:



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

Abstrakt modulu

Modul sa venuje štruktúrovanému typu pole. Objasňuje spôsob deklarovania, princíp práce s prvkami pomocou indexov, použitím for-cyklu. Predvádza riešenie rôznych úloh pomocou polí.

Obsah

Úvod

Modul sa skladá z dvoch tematických jednotiek každá približne rozsahu 3 až 5 vyučovacích hodín. Obe jednotky pokrývajú tieto témy:

1. tematická jednotka – **polia**
   - jednorozmerné pole
   - index poľa
   - pole v cykle
   - zisťuj v poli
   - zisťuj medzi prvkami
   - len časť poľa
2. tematická jednotka – **algoritmy s poľami**
   - ďalšie pole v programe
   - ďalšie pole iného typu
   - pomocné pole
   - Eratostenove sito
   - vkladanie do poľa na správne miesto

Cieľ modulu

Po absolvovaní tohto modulu sa od účastníka očakáva, že

- pre danú úlohu bude schopný zadefinovať jednorozmerné pole,
- správne pre pole zvolí rozsah indexu a typ prvkov,
- bude schopný priradiť hodnotu do prvku poľa,
- bude schopný pristupovať k prvkom v poli,
- bude schopný zaplniť pole údajmi a spracovať tieto hodnoty,
- bude schopný pre danú úlohu spracovať len časť poľa,
- bude schopný pracovať naraz s viac poľami,
- bude schopný definovať a použiť pole s indexom, ktorý má nejaký sémantický význam.

Vstupné vedomosti

Požadované prerekvizity

Modul Programovanie 6 – znakové reťazce a udalosti s myšou.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie popísať princíp fungovania znakových reťazcov,
- vie popísať princíp spracovania udalostí od myši v grafickej ploche,
- dokáže analyzovať zadanie a tiež možné chyby v riešení, chyby vie nájsť a opraviť,
- dokáže vytvoriť aplikácie, ktoré využívajú udalosti od myši,
- rozumie rozdielom medzi rôznymi dátovými typmi, vie posúdiť, kedy je vhodné použitie niektorého typu, ich viditeľnosť v kóde.

Preverenie vstupných vedomostí

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

1. tematická jednotka – Polia

1. Jednorozmerné pole

Doteraz sme pracovali s premennými rôznych typov:
- celočíselné – **Integer**
- reálne - **Real**
- logické - **Boolean**
- znakové - **Char**
- znakové reťazce - **string**

Typy premenných nám striktne určujú, aké typy hodnôt môžeme v týchto premenných uchovávať. Takto máme zabezpečené, že keď chceme pracovať s nejakou premennou, napríklad v aritmetických výrazoch či pri volaní procedúr, táto premenná bude obsahovať iba hodnoty daného typu.

V tomto materiáli sa zaoberáme len jednorozmernými statickými poľami. FreePascal umožňuje vytvárať nielen viacrozmerné polia, ale aj dynamické polia, ktoré môžu meniť počas behu programu svoju veľkosť.

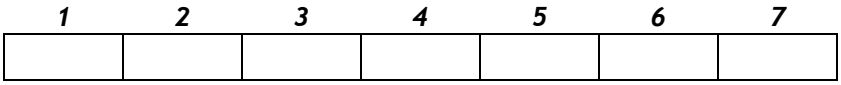
Niekedy sa stretávame s úlohami, v ktorých potrebujeme pracovať aj s väčšou skupinou premenných. Doteraz by sme pre takéto úlohy definovali veľké množstvo tzv. jednoduchých premenných a v programe by sa niektoré časti museli opakovane zapísať pre všetky takéto premenné.

Môžeme si vytvoriť špeciálnu premennú, ktorá môže v sebe obsahovať hoci aj veľké množstvo iných premenných. Tomuto hovoríme štruktúrovaný typ premennej. Konkrétne sa naučíme pracovať so štruktúrovaným typom **pole**, ktorý je charakteristický tým, že všetky premenné, ktoré sú obsiahnuté v poli, musia byť rovnakého typu.

Doteraz sme si premenné v počítači predstavovali ako pamäťové miesta, ktoré môžu v jednom momente obsahovať len nejakú jednu hodnotu. Každá premenná musí mať svoje meno – identifikátor premennej. Priraďovacím príkazom zmeníme hodnotu premennej a tá pôvodná hodnota sa tým zabúda, stráca. Premenné sme si pritom kreslili, napr. ako rámik, okienko, škatuľka, kartička, políčko na kalkulačke a pod., ktoré mali niekde nad sebou uvedené svoje meno.

Štruktúrované premenné budeme kresliť podobne, len v jednom rámiku toho bude výrazne viac, keďže do jedného rámika sa zmestí naraz viac premenných. Všetky premenné, ktoré sú súčasťou jedného poľa, sú nejako očíslované. Tomuto očíslovaniu hovoríme **index** prvku. Index slúži na to, aby sme jednoznačne vedeli určiť, s ktorou premennou (prvkom) poľa potrebujeme pracovať.

Premenná pole sa najlepšie nakreslí ako tabuľka, kde každé políčko má svoje číslo (už vieme, že sa to volá index), napr.



Takto vyzerá pole, ktoré má 7 prvkov (premenných) s indexmi 1 až 7.

Prvým príkladom, na ktorom predvedieme, ako funguje pole a ako sa s ním pracuje, bude príklad s tabuľkou nameraných teplôt: Žiaci dostali úlohu, aby každý deň v jednom týždni zapísali vonkajšiu teplotu u nich doma. Týchto 7 nameraných hodnôt treba zapísať do tabuľky. Okrem toho treba zistiť, aká bola v ten týždeň priemerná teplota. Prvé riešenie bude zatiaľ bez štruktúry pole, len pomocou siedmich premenných:

```
var
  Teplota1, Teplota2, Teplota3, Teplota4, Teplota5,
  Teplota6, Teplota7, Priemer: Real;
begin
  Teplota1 := 20.3;
  Teplota2 := 22.0;
  Teplota3 := 17.7;
  Teplota4 := 18.1;
  Teplota5 := 16.7;
  Teplota6 := 17.3;
  Teplota7 := 19.0;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(10, 70, IntToStr(Teplota1));
  Image1.Canvas.TextOut(30, 70, IntToStr(Teplota2));
  Image1.Canvas.TextOut(50, 70, IntToStr(Teplota3));
  Image1.Canvas.TextOut(70, 70, IntToStr(Teplota4));
  Image1.Canvas.TextOut(90, 70, IntToStr(Teplota5));
  Image1.Canvas.TextOut(110, 70, IntToStr(Teplota6));
  Image1.Canvas.TextOut(130, 70, IntToStr(Teplota7));
  Priemer := (Teplota1 + Teplota2 + Teplota3 + Teplota4 +
              Teplota5 +  Teplota6 + Teplota7) / 7;
  Image1.Canvas.TextOut(10, 100, 'priemer = ' + FloatToStr(Priemer));
end;
```

Namerané hodnoty zapíšeme do tabuľky:

1	*2*	*3*	*4*	*5*	*6*	*7*
20.3	**22.0**	**17.7**	**18.1**	**16.7**	**17.3**	**19.0**

Aby sme túto úlohu mohli vyriešiť pomocou poľa, musíme sa najprv naučiť, ako zadeklarovať štruktúrovanú premennú. Nazvime ju **Teplota** a indexmi budú čísla od 1 do 7. Zapíšeme to takto:

```
var
  Teplota: array [1..7] of Real;
```

Tento zápis hovorí presne toto:

- definujeme premennú **Teplota**, ktorá je poľom (teda tabuľkou),
- obsahuje 7 prvkov s indexmi 1 až 7,
- každá z týchto 7 premenných bude reálneho typu.

Keď budeme v programe chcieť pracovať s niektorým prvkom tabuľky, t.j. s príslušnou premennou, musíme zapísať, napr.

```
  Teplota[1] := 20.3;
```

Teda index zapisujeme do hranatých zátvoriek. Prepíšme teraz kompletný program pomocou poľa:

```
var
  Teplota: array [1..7] of Real;
  Priemer: Real;
begin
  Teplota[1] := 20.3;
  Teplota[2] := 22.0;
  Teplota[3] := 17.7;
  Teplota[4] := 18.1;
  Teplota[5] := 16.7;
  Teplota[6] := 17.3;
  Teplota[7] := 19.0;
  Image1.Canvas.Font.Height := 30;
  Image1.Canvas.TextOut(10, 70, IntToStr(Teplota[1]));
  Image1.Canvas.TextOut(30, 70, IntToStr(Teplota[2]));
  Image1.Canvas.TextOut(50, 70, IntToStr(Teplota[3]));
  Image1.Canvas.TextOut(70, 70, IntToStr(Teplota[4]));
  Image1.Canvas.TextOut(90, 70, IntToStr(Teplota[5]));
  Image1.Canvas.TextOut(110, 70, IntToStr(Teplota[6]));
  Image1.Canvas.TextOut(130, 70, IntToStr(Teplota[7]));
  Priemer := (Teplota[1] + Teplota[2] + Teplota[3] + Teplota[4] +
              Teplota[5] +  Teplota[6] + Teplota[7]) / 7;
  Image1.Canvas.TextOut(10, 100, 'priemer = ' + FloatToStr(Priemer));
end;
```

Čo sa tu teraz zmenilo:

- namiesto siedmich premenných **Teplota1**, **Teplota2**, ... **Teplota7** sme zadeklarovali jedno sedemprvkové pole **Teplota**,
- namiesto výskytu každej premennej v programe **Teplota1**, **Teplota2**, ... používame zápis **Teplota[1]**, **Teplota[2]**, ...

Program teraz funguje rovnako ako so siedmimi samostatnými premennými. Neskôr uvidíme, že použitie poľa má mnoho ďalších významných výhod.

Čo sme sa naučili

V programoch môžeme zoskupovať premenné do tzv. štruktúrovanej premennej pole. Premenné typu pole majú tieto vlastnosti:

- všetky prvky poľa sú rovnakého typu,
- k prvkom poľa pristupujeme pomocou indexu – zapisujeme ho do [ ] zátvoriek,
- pole si môžeme najlepšie predstaviť ako tabuľku, ktorá má očíslované prvky od 1 do počet prvkov.

Úlohy na precvičenie

Zadanie 1	V škole je 9 tried. Zapíšte do prvkov poľa **Trieda** počty žiakov, vypíšte ich a spočítajte počet všetkých žiakov školy.
Zadanie 2	Jeden žiak má 5 sporiteľničiek. V prvej mal 5 euro. V druhej mal dvojnásobok prvej. V každej ďalšej mal dvojnásobok predchádzajúcej. Zapíšte tieto hodnoty do poľa, vypíšte ich a tiež celkovú ich sumu. Uvažujte nad tým, čo sa bude meniť, keď sa zmení prvá zadaná hodnota?

2. Index poľa

V príklade so siedmimi nameranými teplotami sme **indexovali** číslom od 1 do 7 - indexom bola konštanta. V skutočnosti môže byť indexom aj komplikovaný aritmetický výraz. Jediné, čo by sme mali zaručiť, aby jeho hodnota nebola menšia ako 1 a väčšia ako 7. Nasledujúce priradenia len ilustrujú rôzne možnosti zápisov indexov:

```
  Teplota[1+1+1] := 11.1;            // Teplota[3] := 11.1;
  Teplota[2*3*4-4*5] := 22.2;        // Teplota[4] := 22.2;
  Teplota[4+1] := Teplota[4-1] + 1;  // Teplota[5] := Teplota[3]+1;
```

Polia môžu mať aj oveľa viac prvkov ako len 7. Nasledujúci príklad bude ilustrovať iné použitie poľa. Na miestnom úrade menšej obce v počítači evidujú počty obyvateľov v jednotlivých domoch. Domy majú popisné čísla a tieto začínajú číslom 1 a končia posledným domom s číslom 200. V nasledujúcom programe nebudeme priraďovať hodnoty pre všetkých 200 domov, ukážeme len deklaráciu a jednoduché použitie:

```
var
  Dom: array [1..200] of Integer;
begin
  // tu sa priradili hodnoty pre všetkých 200 domov

  Dom[100] := 8;
  Dom[29] := Dom[29] + 2;
  Dom[123] := Dom[123] + Dom[122];
  Dom[122] := 0;
end;
```

Prvé priradenie **Dom[100] := 8;** označuje, že sme v tabuľke o počtoch obyvateľov opravili prvok s indexom 100 - priradili sme sem hodnotu 8 (v dome býva 8 obyvateľov). Druhé priradenie **Dom[29] := Dom[29] + 2;** označuje, že sme do domu číslo 29 pridali dvoch obyvateľov (asi tam pribudli dvojčatá). Ďalšie dve priradenia popisujú takúto situáciu: obyvatelia z domu 122 sa kompletne presťahujú do domu 123 (asi idú rekonštruovať dom). To znamená, že najprv k počtu obyvateľov domu 123 pripočítame počet z domu 122 (**Dom[123] := Dom[123] + Dom[122];**) a potom vynulujeme počet v dome 122 (**Dom[122] := 0;**).

V praxi sa vyskytujú podobné úlohy, ale v nich môžeme index prvkov poľa vyjadriť pomocou premenných. Zapíšme časť programu, ktorý rieši takéto úlohy:

- z domov číslo **I**, **I+7**, **I+8** odišlo po jednom študentovi študovať do mesta
- ulica, na ktorej je 5 domov, začína domom číslo **X** (ďalšie čísla nasledujú za ním); na ulici opravujú vodovod; zistite, koľkým obyvateľom bude treba voziť vodu,
- v každom desiatom dome počnúc číslom **D** sa zdvojnásobil počet obyvateľov - zapíšte to len pre 6 takýchto domov.

```
  Dom[I] := Dom[I] - 1;
  Dom[I+7] := Dom[I+7] - 1;
  Dom[I+8] := Dom[I+8] - 1;

  PocetBezVody := Dom[X] + Dom[X+1] + Dom[X+2] + Dom[X+3] + Dom[X+4];

  Dom[D] := 2 * Dom[D];
  Dom[D+10] := 2 * Dom[D+10];
  Dom[D+20] := 2 * Dom[D+20];
  Dom[D+30] := 2 * Dom[D+30];
  Dom[D+40] := 2 * Dom[D+40];
  Dom[D+50] := 2 * Dom[D+50];
```

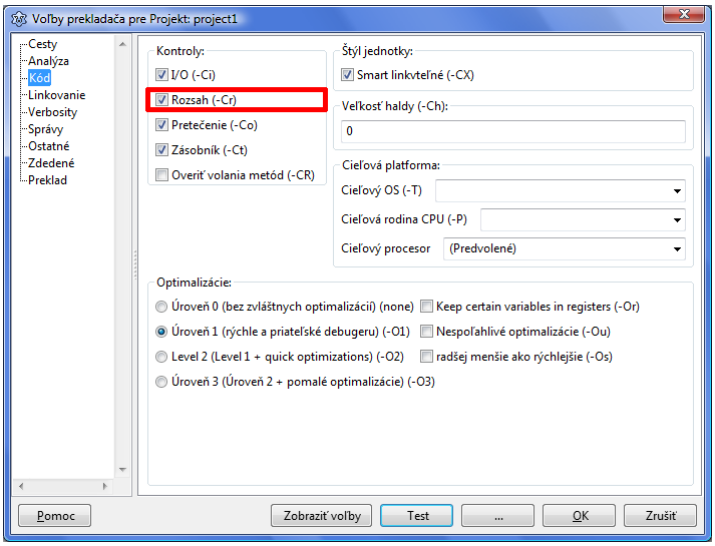
Už sme spomínali, že počítač by mal mať pri práci s prvkami poľa vždy istotu, že index je v správnom rozsahu. V našom príklade s počtami obyvateľov v domoch to

znamená, že index nemá menšiu hodnotu ako 1 a väčšiu ako 200. Pre každé z priradení s premennými **I**, **X** a **D** určime, aké sú povolené hodnoty týchto premenných. Zrejme všetky tri musia mať minimálne hodnotu 1. Ich maximálna hodnota je ale rôzna:

- pre premennú **I** by indexovanie **Dom[I+7]** a **Dom[I+8]** nemalo presiahnuť 200, preto musí **I+8 <= 200**, preto **I <= 192**,
- pre premennú **X** nesmie **X+4** presiahnuť 200 a preto musí **X <= 196**,
- pre premennú **D** nesmie **D+50** presiahnuť 200 a preto musí **D <= 150**.

Čo by sa stalo v prípade, že by index bol mimo povoleného rozsahu? Program by spadol a dostali by sme chybovú správu "**Range check error**" (t.j. Chyba pri kontrole rozsahu indexu).

Pre kontrolu tejto chyby musí byť v kompilátore nastavená kontrola rozsahov:



Čo sme sa naučili

Indexom poľa môže byť

- konštanta
- aritmetický výraz
- premenná alebo výraz s premennou

Index nesmie presiahnuť povolený rozsah

3. Pole v cykle

Peknou vlastnosťou polí je to, že k jeho prvkom môžeme pristupovať vo for-cykle. Totiž for-cyklus funguje tak, že počítadlo cyklu postupne mení hodnoty, napr. od 1 do 7. Túto premennú môžeme využiť ako index poľa a teda s každým prvkom poľa vykonáme nejakú operáciu. Napr. v príklade so 7 nameranými teplotami môžeme zapísať:

```
var
  Teplota: array [1..7] of Real;
  I: Integer;
begin
  for I := 1 to 7 do
    Teplota[I] := 21.0;
```

Lebo jeden zo žiakov nameral na svojom teplomere každý deň presne 21 stupňov. Ďalší so žiakov nameral prvý deň 21, druhý 22, tretí 23, atď., každý ďalší deň o stupeň viac:

```
var
  Teplota: array [1..7] of Real;
  I: Integer;
begin
  for I := 1 to 7 do
    Teplota[I] := 20 + I;
```

Tretí zo žiakov si domácu úlohu zjednodušil a nechal počítač náhodne vygenerovať nejaké stupne od 16 do 25:

```
var
  Teplota: array [1..7] of Real;
  I: Integer;
begin
  for I := 1 to 7 do
    Teplota[I] := 16 + Random(10);
```

Tieto zápisy nám ukazujú, že ak vieme popísať nejaký predpis (napríklad popis vzniku nameraných hodnôt), tak jedným for-cyklom môžeme priradiť hodnoty prvkov ľubovoľného poľa.

V príklade s nameranými hodnotami sme tieto hodnoty ešte aj vypisovali do plochy a počítali ich priemer. Aj na toto môžeme elegantne využiť for-cykly. Prepíšeme kompletný program a to tak, že teploty pritom budú nepárne čísla od 25 nižšie:

```
var
  Teplota: array [1..7] of Real;
  I: Integer;
  Sucet, Priemer: Real;
begin
  for I := 1 to 7 do
    Teplota[I] := 27 - 2 * I;
  Image1.Canvas.Font.Height := 30;
  for I := 1 to 7 do
    Image1.Canvas.TextOut(20*I-10, 70, IntToStr(Teplota[I]));
  Sucet := 0;
  for I := 1 to 7 do
    Sucet := Sucet + Teplota[I];
  Priemer := Sucet / 7;
  Image1.Canvas.TextOut(10, 100, 'priemer = ' + FloatToStr(Priemer));
end;
```

Pristavme sa podrobnejšie pri týchto nových dvoch for-cykloch. Najprv cyklus, ktorý vypisuje prvky poľa do plochy. Pôvodná verzia bez cyklu vyzerala takto:

```
  Image1.Canvas.TextOut(10, 70, IntToStr(Teplota[1]));
  Image1.Canvas.TextOut(30, 70, IntToStr(Teplota[2]));
  Image1.Canvas.TextOut(50, 70, IntToStr(Teplota[3]));
  Image1.Canvas.TextOut(70, 70, IntToStr(Teplota[4]));
  Image1.Canvas.TextOut(90, 70, IntToStr(Teplota[5]));
  Image1.Canvas.TextOut(110, 70, IntToStr(Teplota[6]));
  Image1.Canvas.TextOut(130, 70, IntToStr(Teplota[7]));
```

A my sme X-ovú súradnicu výpisu zapísali ako **20*I-10**. Druhý cyklus počítal priemer tak, že najprv spočítal súčet teplôt (k premennej **Sucet** postupne pripočítaval **I**-tu teplotu) a na záver sa tento súčet vydelil 7.

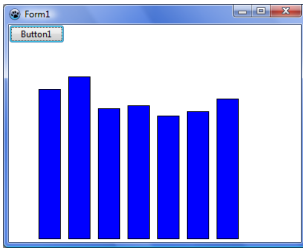
Ešte by sa nám hodilo, keby sme vedeli do poľa priradiť hodnoty nie for-cyklom, ale nejakým pekným priradením. Nie postupne po jednom, ale nejako šikovnejšie. Existuje možnosť, tzv. **inicializácia poľa**, keď už počas deklarácie vymenujeme všetky hodnoty prvkov poľa. Zapisujeme to takto:

```
var
  Teplota: array [1..7] of Real =
      (20.3, 22.0, 17.7, 18.1, 16.7, 17.3, 19.0);
```

Všimnite si, že za typ prvku zapisujeme znak rovná sa a za tým, v okrúhlych zátvorkách sú vymenované všetky hodnoty – musí byť ich presne toľko, koľko prvkov je v zadeklarovanom poli. Samozrejme, že takto sa zvyknú priraďovať prvky len menších polí. Priraďovať takto niekoľko stovák prvkov by bolo dosť nepraktické.

Dokončime príklad s teplotami tak, že nakreslíme stĺpcový graf: Výška každého stĺpca bude vyjadrovať teplotu v jednotlivých dňoch. Keďže teplota je reálne číslo, nesmieme zabudnúť túto hodnotu previesť na celé číslo (príkaz **Rectangle** funguje len s celými číslami):

Po spustení dostávame v grafickej ploche:




```
var
  Teplota: array [1..7] of Real =
      (20.3, 22.0, 17.7, 18.1, 16.7, 17.3, 19.0);
  I: Integer;
begin
  Image1.Canvas.Brush.Color := clBlue;
  for I := 1 to 7 do
    Image1.Canvas.Rectangle(40*I, 290,
                            40*I+30, 290-Round(Teplota[I]*10));
end;
```

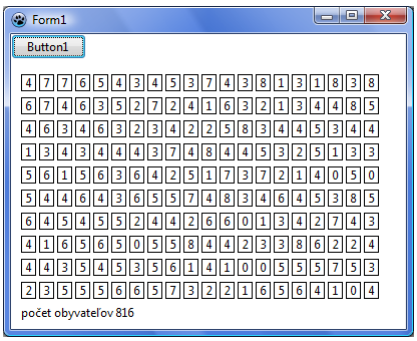
Vytvorme teraz program, ktorý vytvorí 200-prvkové pole počtov obyvateľov nejakej obce. Pole nejako zaplní náhodnými hodnotami, vykreslí ho do plochy (napr. ako obdĺžniky s číslom počtu obyvateľov) a na záver vypíše ich celkový počet:

Vedeli by ste povedať, či hodnota výrazu **Random(9)** dáva rovnaké výsledky, ako keby sme namiesto toho zapísali **Random(5) + Random(5)**?

```
var
  Dom: array [1..200] of Integer;
  I, X, Y, Sucet: Integer;
begin
  for I := 1 to 200 do
    Dom[I] := Random(9);
  Sucet := 0;
  for I := 1 to 200 do
  begin
    X := 18 * ((I-1) mod 20) + 10;
    Y := 23 * ((I-1) div 20) + 40;
    Image1.Canvas.Rectangle(X, Y, X + 15, Y + 17);
    Image1.Canvas.TextOut(X+5, Y+1, IntToStr(Dom[I]));
    Sucet := Sucet + Dom[I];
  end;
  Image1.Canvas.TextOut(10, 270, 'počet obyvateľov '+IntToStr(Sucet));
end;
```

Všimnite si, že súčet všetkých prvkov poľa (počet obyvateľov) počítame v tom istom

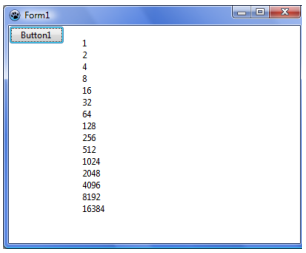
for-cykle, v ktorom aj kreslíme všetky domy. Po spustení:



V ďalšom príklade vytvoríme a vypíšeme tabuľku mocnín čísla 2. Tieto hodnoty budeme ukladať do tabuľky (jednorozmerné pole) :

```
const
  N = 15;
var
  Mocnina: array [1..N] of Integer;
  I: Integer;
begin
  Mocnina[1] := 1;
  for I := 2 to N do
    Mocnina[I] := Mocnina[I-1] * 2;
  for I := 1 to N do
    Image1.Canvas.TextOut(100, 16*I, IntToStr(Mocnina[I]));
end;
```

Po spustení dostávame mocniny 2 od 2^0 až po 2^{14}:



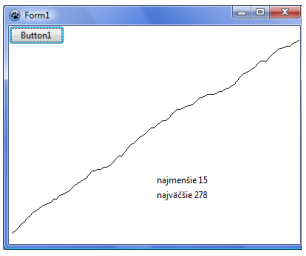
V tomto príklade je poučné použitie konštanty **N**. Tiež vidíme, ako počítame mocniny 2: najprv sa do Mocniny[1] priradí hodnota 1, a potom sa **I**-ta mocnina počíta ako dvojnásobok predchádzajúcej hodnoty.

Na záver tejto časti náhodne vygenerujeme 100 prvkové pole, ale tak, aby každý ďalší prvok bol buď rovný predchádzajúcemu, alebo bol o trochu väčší. Takto sa vytvorí vzostupná postupnosť čísel. Z tejto postupnosti na záver vykreslíme graf:

```
var
  Cislo: array [1..100] of Integer;
  I: Integer;
begin
  Cislo[1] := 10 + Random(10);
  for I := 2 to 100 do
    Cislo[I] := Cislo[I-1] + Random(6);
  Image1.Canvas.MoveTo(4, Image1.Height - Cislo[1]);
  for I := 2 to 100 do
    Image1.Canvas.LineTo(4 * I, Image1.Height - Cislo[I]);
  Image1.Canvas.TextOut(200, 200, 'najmenšie '+IntToStr(Cislo[1]));
  Image1.Canvas.TextOut(200, 220, 'najväčšie '+IntToStr(Cislo[100]));
end;
```

Samotný program nepotreboval na túto úlohu pole, ale takto vytvorené pole môžeme využiť v rôznych iných algoritmoch.

Môžeme vidieť, že postupnosť čísel v poli je neklesajúca, pričom vidíme aj prvé z nich (najmenšie) aj posledné (najväčšie):



Čo sme sa naučili

Prvky poľa môžeme prechádzať v cykle. Vtedy je počítadlo cyklu indexom poľa. Najčastejšie dolná a horná hranica cyklu sa nastaví na rozsah indexov poľa.

V tejto časti sme ukázali nasledovné techniky:

- priemer prvkov poľa,
- nakreslenie stĺpcového grafu z prvkov poľa
- vytvorenie poľa mocnín dvojky
- vygenerovanie náhodnej rastúcej postupnosti

Úlohy na precvičenie

Zadanie 1	Vygenerujte 100 prvkové pole tak, že v poslednom prvku bude 1, v predchádzajúcom 2, pred ním 3, atď. až v prvom bude 100.
Zadanie 2	Vygenerujte 20 prvkové pole tak, že v prvom prvku poľa bude 1, v druhom bude o 3 viac, v ďalšom o 5 viac od druhého, atď. v každom ďalšom bude o ďalšie nepárne číslo viac ako v predchádzajúcom. Pole vypíšte.
Zadanie 3	V poli **Vyska** máme uložené výšky všetkých 15 žiakov v triede. Do poľa sme tieto výšky zadávali v poradí od najmenšieho po najväčšieho. Vtedy sme zistili, že najmenší meral 155 cm a každý ďalší v rade bol o 2 cm vyšší. Zaplňte celé pole a potom tieto výšky vypíšte.
Zadanie 4	Do poľa **Vahy** sme uložili odvážené hmotnosti detí, ktoré sme zistili na začiatku letného tábora. Po skončení tábora deti pribrali približne od 5% do 15%. Opravte toto pole.
Zadanie 5	V poli **Platy** sú uložené platy učiteľov. Chceme im náhodne zvýšiť plat buď o 5%, alebo o10% alebo o 15%. Naprogramujte to.

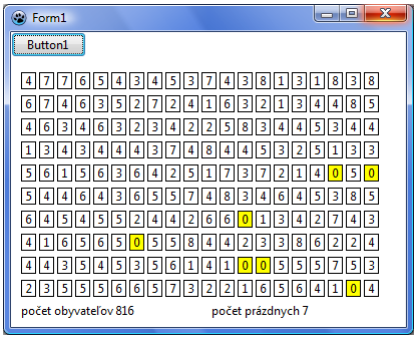
4. Zisťuj v poli

Už sme sa naučili, ako je užitočné vedieť prechádzať prvky poľa v cykle. Pritom sme buď do týchto prvkov niečo priraďovali, alebo sme ich spočítavali, alebo vykresľovali. Lenže cyklus môžeme využiť aj vtedy, keď chceme o prvkoch poľa len niečo zistiť. Napr. nás môže zaujímať, či sa v poli nachádzajú nulové prvky (či je v obci prázdny dom), alebo, v ktorom prvku je najväčšie číslo (kedy bolo najteplejšie, v ktorom dome býva najviac obyvateľov).

Pri takomto zisťovaní nejakých informácií o poli využijeme podmienený príkaz **if**. Každý prvok prekontrolujeme na nejakú vlastnosť. Začnime zisťovaním počtu nulových prvkov v poli. Použijeme generátor náhodných čísel pre zaplnenie poľa:

```
var
  Dom: array [1..200] of Integer;
  I, X, Y, Sucet, Pocet: Integer;
begin
  for I := 1 to 200 do
    Dom[I] := Random(9);
  Sucet := 0;
  for I := 1 to 200 do
  begin
    X := 18 * ((I-1) mod 20) + 10;
    Y := 23 * ((I-1) div 20) + 40;
    if Dom[I] = 0 then
      Image1.Canvas.Brush.Color := clYellow
    else
      Image1.Canvas.Brush.Color := clWhite;
    Image1.Canvas.Rectangle(X, Y, X + 15, Y + 17);
    Image1.Canvas.TextOut(X+5, Y+1, IntToStr(Dom[I]));
    Sucet := Sucet + Dom[I];
  end;
  Image1.Canvas.TextOut(10, 270, 'počet obyvateľov '+IntToStr(Sucet));
  Pocet := 0;
  for I := 1 to 200 do
    if Dom[I] = 0 then
      Inc(Pocet);
  Image1.Canvas.TextOut(200, 270, 'počet prázdnych '+IntToStr(Pocet));
end;
```

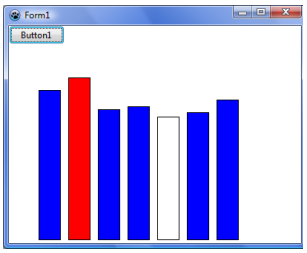
Všimnite si, že už pri vykresľovaní poľa sme využili podmienený príkaz - nulové prvky sme zafarbili na žlto. Okrem toho ďalší cyklus počíta počet nulových prvkov: najprv dá do premennej **Pocet** nulu a potom pri každom výskyte 0, tento počet zvýši o 1. Aj tento cyklus sa mohol vykonať už počas kreslenia prvkov poľa. Tu sme to úmyselne oddelili, aby bolo lepšie vidieť zisťovanie počtu prvkov s nejakou vlastnosťou (sú rovné 0). Po spustení dostávame:



Ďalšie ukážka bude zisťovať maximálny a minimálny prvok poľa. Ukážeme to na príklade s tabuľkou nameraných teplôt.

```
var
  Teplota: array [1..7] of Real =
      (20.3, 22.0, 17.7, 18.1, 16.7, 17.3, 19.0);
  I: Integer;
  Min, Max: Real;
begin
  Max := Teplota[1];
  Min := Teplota[1];
  for I := 2 to 7 do
  begin
    if Teplota[I] > Max then
      Max := Teplota[I];
    if Teplota[I] < Min then
      Min := Teplota[I];
  end;
  for I := 1 to 7 do
  begin
    if Teplota[I] = Max then
      Image1.Canvas.Brush.Color := clRed
    else if Teplota[I] = Min then
      Image1.Canvas.Brush.Color := clWhite
    else
      Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Rectangle(40*I, 290,
                            40*I+30, 290-Round(Teplota[I]*10));
  end;
end;
```

Aj do samotného vykresľovania stĺpcového grafu sme vložili podmienený príkaz - ak máme vykresliť maximálnu hodnotu, tak sa vykreslí červenou farbou, minimálny prvok bielou a všetky ostatné modrou.



Pozorne si pozrite, ako hľadáme maximum, resp. minimum v poli. Ešte pred cyklom do premennej **Max** priradíme hodnotu prvého prvku. V tomto momente predpokladáme, že možno práve prvý prvok je maximálny. Potom postupne prechádzame všetky prvky poľa (začíname od 2., lebo prvý už kontrolovať nemusíme) a porovnávame ich s doteraz najlepším maximom. Ak nájdeme väčší prvok, tento sa stáva novým maximom. Úplne analogicky funguje aj hľadanie minimálneho prvku.

Zisťovať minimum a maximum môžeme aj trochu inak. V tomto predchádzajúcom riešení sme v premenných **Max** a **Min** mali uloženú maximálnu a minimálnu hodnotu. V niektorých prípadoch ale môžeme potrebovať nielen túto hodnotu, ale aj koľký v poradí je to prvok. Upravíme program tak, že v premenných **Max** a **Min** nie sú hodnoty, ale indexy do poľa (aj ich typy nie sú **Real** ale **Integer**):

```
var
  Teplota: array [1..7] of Real =
      (20.3, 22.0, 17.7, 18.1, 16.7, 17.3, 19.0);
  I, Min, Max: Integer;
begin
  Max := 1;
  Min := 1;
  for I := 2 to 7 do
  begin
    if Teplota[I] > Teplota[Max] then
      Max := I;
    if Teplota[I] < Teplota[Min] then
      Min := I;
  end;
```

```
  for I := 1 to 7 do
  begin
    if I = Max then
      Image1.Canvas.Brush.Color := clRed
    else if I = Min then
      Image1.Canvas.Brush.Color := clWhite
    else
      Image1.Canvas.Brush.Color := clBlue;
    Image1.Canvas.Rectangle(40*I, 290,
                            40*I+30, 290-Round(Teplota[I]*10));
  end;
end;
```

Uvedomte si, že ak bolo v poli viac prvkov, ktoré mali maximálnu alebo minimálnu hodnotu, toto riešenie farebne vyznačí len prvú z týchto hodnôt.

Podmienený príkaz môžeme využiť aj vtedy, keď chceme meniť hodnoty niektorých prvkov. Napríklad, do všetkých prázdnych domov nasťahujeme nové 2-členné rodiny. Okrem toho v každom dome, kde bývajú aspoň 4, sa narodí bábätko. Koľko takto pribudne nových obyvateľov?

```
var
  Dom: array [1..200] of Integer;
  I, Pribudne: Integer;
begin
  ...
  Pribudne := 0;
  for I := 1 to 200 do
  begin
    if Dom[I] = 0 then
    begin
      Dom[I] := 2;
      Pribudne := Pribudne + 2;
    end;
    if Dom[I] >= 4 then
    begin
      Inc(Dom[I]);
      Inc(Pribudne);
    end;
  end;
  Image1.Canvas.TextOut(200, 270, 'pribudne '+IntToStr(Pribudne));
end;
```

Namiesto
Pribudne := Pribudne + 2;
môžeme zapisovať aj
Inc(Pribudne, 2);

Ďalšou skupinou úloh sú také, keď potrebujeme zistiť nielen napr. maximum alebo priemer, ale zároveň potrebujeme vedieť počet prvkov, ktoré sú rovné maximu, alebo počet prvkov, ktoré sú väčšie ako priemer. Obe tieto úlohy majú svoje špecifiká, preto ich budeme riešiť zvlášť. Najprv zistíme počet nadpriemerných prvkov, t.j. koľko dní v meranom týždni bolo teplejších ako priemerná teplota?

```
var
  Teplota: array [1..7] of Real;
  I, Pocet: Integer;
  Sucet, Priemer: Real;
begin
  ...
  Sucet := 0;
  for I := 1 to 7 do
    Sucet := Sucet + Teplota[I];
  Priemer := Sucet / 7;
  Pocet := 0;
  for I := 1 to 7 do
    if Teplota[I] > Priemer then
      Inc(Pocet);
  Image1.Canvas.TextOut(100, 10, 'nadpriemerných ' + IntToStr(Pocet));
end;
```

Všimnite si, že počet nadpriemerných prvkov musíme počítať dvoma cyklami: prvým vypočítame súčet prvkov a potom samotný priemer. Druhým cyklom znovu

prechádzame všetky prvky a zisťujeme koľko z nich je väčších ako priemer.

Ďalšou úlohou bude spočítať počet maximálnych prvkov, t.j. koľko dní v meranom týždni bolo najteplejšie (bola rovnaká teplota ako v najteplejší deň)? Opäť to vyriešme dvoma cyklami:

```
var
  Teplota: array [1..7] of Real;
  I, Pocet: Integer;
  Max: Real;
begin
  ...
  Max := Teplota[1];
  for I := 2 to 7 do
    if Teplota[I] > Max then
      Max := Teplota[I];
  Pocet := 0;
  for I := 1 to 7 do
    if Teplota[I] = Max then
      Inc(Pocet);
  Image1.Canvas.TextOut(100, 10, 'maximálnych ' + IntToStr(Pocet));
end;
```

Prvý cyklus nájde maximálny prvok. Druhým cyklom znovu prechádzame všetky prvky a zisťujeme koľko z nich je rovnakých ako toto maximum.

Na rozdiel od zisťovania nadpriemerných prvkov, kde sme naozaj potrebovali dva cykly, dá sa zapísať algoritmus, ktorý zistí tento počet už v prvom cykle: vždy keď nájdeme nové maximum, začneme počet maximálnych počítať od jedna. Vždy, keď nájdeme rovnakú hodnotu, ako maximum, len zvýšime počet:

```
var
  Teplota: array [1..7] of Real;
  I, Pocet: Integer;
  Max: Real;
begin
  ...
  Max := Teplota[1];
  Pocet := 1;
  for I := 2 to 7 do
    if Teplota[I] > Max then
    begin
      Max := Teplota[I];
      Pocet := 1;
    end
    else if Teplota[I] = Max then
      Inc(Pocet);
  Image1.Canvas.TextOut(100, 10, 'maximálnych ' + IntToStr(Pocet));
end;
```

Uvedomte si, že všetky algoritmy na zisťovanie maxima, priemernej hodnoty, počtu nadpriemerných prvkov aj počtu maximálnych hodnôt, fungujú aj pre ľubovoľne veľké pole. Stačí len zmeniť hornú hranicu cyklov zo 7 na skutočnú veľkosť poľa.

Čo sme sa naučili

Počas prechádzania prvkov poľa v cykle môžeme niektoré akcie robiť len vtedy, keď prvok spĺňa nejakú podmienku.

V tejto časti sme ukázali nasledovné techniky:

- zistenie počtu nulových prvkov poľa,
- hľadanie minimálneho a maximálneho prvku,
- počet minimálnych, maximálnych a nadpriemerných prvkov.

Úlohy na precvičenie

Zadanie 1	V poli **Vysky** máme uložené odmerané výšky žiakov. Zistite, koľkí z nich sú najvyšší, prípadne sa od najvyššieho nelíšia o viac ako 5 cm. Podobne zistite počet minimálnych výšok tiež s toleranciou 5 cm.
Zadanie 2	V poli znakových reťazcov máme menný zoznam žiakov. Vypíšte najkratšie a najdlhšie meno. Ak je takých viac vypíšte ich všetky.
Zadanie 3	V poli máme uložené bodové výsledky testu žiakov. Zistite, aké bolo dosiahnuté minimum a koľko bodov získal, druhý najslabší test.

5. Zisťuj medzi prvkami

Doteraz sme pri prechádzaní prvkov poľa tieto porovnávali buď s nejakou konštantou (koľko bolo nulových) alebo s hodnotou nejakej premennej (koľko nadpriemerných, ktorý je maximálny prvok, koľko je maximálnych prvkov).

Teraz budeme zisťovať vzťahy medzi prvkami poľa. Zistíme, či sa každý deň oproti predchádzajúcemu otepľovalo, t.j. či bolo stále teplejšie a teplejšie. Pýtame sa teda, či má pole vzostupné hodnoty prvkov. Využijeme tu logickú hodnotu. V nej si budeme pamätať, či pri prechádzaní prvkov poľa je stále ešte splnená požadovaná podmienka.

```
var
  Teplota: array [1..7] of Real;
  I: Integer;
  Vzostupne: Boolean;
begin
  ...
  Vzostupne := True;
  for I := 1 to 6 do
    if Teplota[I] >= Teplota[I+1] then
      Vzostupne := False;
  if Vzostupne then
    Image1.Canvas.TextOut(100, 10, 'vzostupné hodnoty')
  else
    Image1.Canvas.TextOut(100, 10, 'nie sú vzostupné');
end;
```

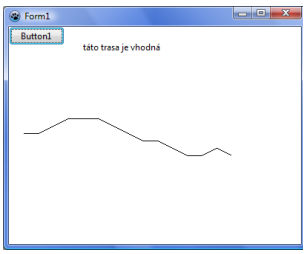
Logická premenná **Vzostupne** má na začiatku hodnotu **True**, lebo zatiaľ sme *optimisti* a predpokladáme, že pole bude spĺňať podmienku. Potom postupne prechádzame prvky poľa a porovnávame ich s ich nasledovníkmi. Ak nájdeme dvojicu prvkov, v ktorej je prvý väčší alebo rovný ako druhý, je jasné že už celé pole nemôže spĺňať očakávanú podmienku. Preto v tomto prípade nastavíme logickú premennú **Vzostupne** na **False**.

Všimnite si v programe veľmi dôležitý detail: for-cyklus nejde od 1 do 7, teda cez všetky prvky, ale od 1 iba do 6. Dôvod je ten, že v tele cyklu pracujeme nielen s **I**-tym prvkom poľa **Teplota[I]**, ale aj s **I+1**, t.j. **Teplota[I+1]**. Keby cyklus bežal až po 7, v poslednom prechode by sme chceli porovnávať siedmy prvok s ôsmym, ale na tomto by program spadol na "Range checking error".

V ďalšom programe budeme riešiť takúto úlohu: v poli **Trasa** máme zaznačené nadmorské výšky prechádzanej trasy nášho výletu. Predpokladáme, že tieto výšky sme odmerali v pravidelných úsekoch cesty a medzi týmito bodmi ide cesta veľmi rovnomerne. Keďže na výlet chceme zobrať starú mamu, potrebujeme zistiť, či to zvládne a teda, či na trase nie sú prudké stúpania, či klesania. Určite ju zvládne, ak sa nadmorská výška nelíši o viac ako +1 alebo -1. Napíšme program, ktorý dostane v poli zadanú trasu, vykreslí ju a vypíše správu, či je nenáročná aj pre starú mamu.

```
var
  Trasa: array [1..15] of Integer =
       (15, 15, 16, 17, 17, 17, 16, 15, 14, 14, 13, 12, 12, 13, 12);
  I: Integer;
  Vhodna: Boolean;
begin
  Image1.Canvas.MoveTo(20, Image1.Height - Trasa[1]*10);
  for I := 2 to 15 do
    Image1.Canvas.LineTo(20*I, Image1.Height - Trasa[I]*10);
  Vhodna := True;
  for I := 1 to 14 do
    if Abs(Trasa[I] - Trasa[I+1]) > 1 then
      Vhodna := False;
  if Vhodna then
    Image1.Canvas.TextOut(100, 20,'táto trasa je vhodná')
  else
    Image1.Canvas.TextOut(100, 20,'táto trasa nie je vhodná');
end;
```

Po spustení programu dostávame:



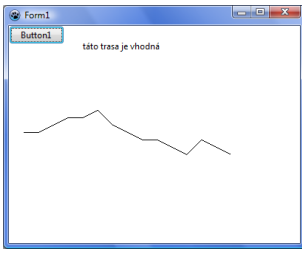
Vidíme tu dva cykly: prvý cyklus vykresľuje trasu do grafickej plochy a preto do

prvého vrcholu presunie grafické pero ešte pred cyklom a potom cyklus ide až od druhého prvku poľa. Druhý cyklus kontroluje podmienku pre trasu a preto musí byť skrátený po predposledný prvok. Všimnite si, ako počítame sklon trasy: **Abs(Trasa[I] - Trasa[I+1])**.

Starý otec je v lepšej forme a zvládne aj náročnejšiu trasu ako stará mama. Vyhlásil, že na výlete zvládne aj dva úseky, ktorých prevýšenie je 2 metre. Zrejme tým myslel, že ostatné úseky nemajú väčšie prevýšenie ako 1 meter. Logickú premennú **Vhodna** nahradíme počítadlom **PocetTazkych**, v ktorom budeme počítať počet náročnejších úsekov:

```
var
  Trasa: array [1..15] of Integer =
      (15, 15, 16, 17, 17, 18, 16, 15, 14, 14, 13, 12, 14, 13, 12);
  I, PocetTazkych: Integer;
begin
  Image1.Canvas.MoveTo(20, Image1.Height - Trasa[1]*10);
  for I := 2 to 15 do
    Image1.Canvas.LineTo(20*I, Image1.Height - Trasa[I]*10);
  PocetTazkych := 0;
  for I := 1 to 14 do
    if Abs(Trasa[I] - Trasa[I+1]) > 1 then
      if Abs(Trasa[I] - Trasa[I+1]) = 2 then
        Inc(PocetTazkych)
      else
        PocetTazkych := 100;
  if PocetTazkych <= 2 then
    Image1.Canvas.TextOut(100, 20,'táto trasa je vhodná')
  else
    Image1.Canvas.TextOut(100, 20,'táto trasa nie je vhodná');
end;
```

Po spustení:



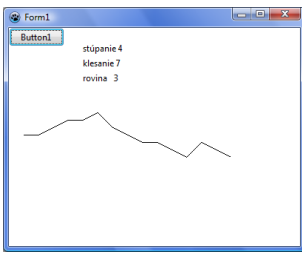
Pozorne preštudujte podmienky v podmienených príkazoch. Rozumiete, prečo je tu priradenie **PocetTazkych := 100**?

Takže tento program namiesto logickej hodnoty pracuje s počítadlom nejakých situácií a na záver tento počet zhodnotil.

Využijeme to v ďalšej úlohe: zistíme charakteristiky trasy, t.j. aká časť trasy je stúpanie, aká časť je klesanie a aká časť je rovinka:

```
var
  Trasa: array [1..15] of Integer =
      (15, 15, 16, 17, 17, 18, 16, 15, 14, 14, 13, 12, 14, 13, 12);
  I, Stupanie, Klesanie, Rovina: Integer;
begin
  Image1.Canvas.MoveTo(20, Image1.Height - Trasa[1]*10);
  for I := 2 to 15 do
    Image1.Canvas.LineTo(20*I, Image1.Height - Trasa[I]*10);
  Stupanie := 0;
  Klesanie := 0;
  Rovina := 0;
  for I := 1 to 14 do
    if Trasa[I] < Trasa[I+1] then
      Inc(Stupanie)
    else if Trasa[I] > Trasa[I+1] then
      Inc(Klesanie)
    else
      Inc(Rovina);
  Image1.Canvas.TextOut(100, 20,'stúpanie ' + IntToStr(Stupanie));
  Image1.Canvas.TextOut(100, 40,'klesanie ' + IntToStr(Klesanie));
  Image1.Canvas.TextOut(100, 60,'rovina   ' + IntToStr(Rovina));
end;
```

Po spustení program vypíše informácie o trase:



Opäť musíme použiť cyklus od 1 do 14, t.j. o jedna kratší ako počet prvkov poľa. Tiež si všimnite, ako sú za sebou zaradené tri sledované prípady: stúpanie, klesanie a rovinka.

Čo sme sa naučili

Pri prechádzaní prvkov poľa v cykle môžeme kontrolovať aj nejaké vzťahy medzi za sebou nasledujúcimi prvkami. Z tohto dôvodu má väčšinou cyklus o 1 menej prechodov ako je počet prvkov poľa.

V tejto časti sme ukázali nasledovné techniky:

- zistenie, či je pole usporiadané (vzostupne, zostupne),
- zistenie, či hodnoty v poli na nejakom úseku príliš nestúpajú alebo neklesajú

Úlohy na precvičenie

Zadanie 1	V poli znakových reťazcov máme menný zoznam žiakov, Zistite, či je tento zoznam utriedený.
Zadanie 2	V poli **Dom** máme priradené počty obyvateľov v jednotlivých domoch obce. • Zistite, pri ktorom prázdnom dome je dom, v ktorom býva čo najviac obyvateľov. • Nájdite trojicu susediacich domov, ktoré majú najväčší súčet obyvateľov.

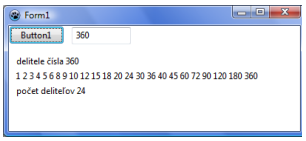
6. Len časť poľa

V istých situáciách nevieme dopredu, aké veľké pole budeme potrebovať, ale vieme odhadnúť jeho maximálnu veľkosť. Pri práci s takýmto poľom si musíme v nejakej premennej pamätať, koľko prvkov sme už do tohto poľa vložili. Aby sme vedeli, koľko prvkov poľa môžeme vypísať, na ktorý index môžeme pridať novú hodnotu, prípadne, ak chceme poslednú hodnotu vyhodiť. Musíme si dobre zvážiť, aké veľké pole pre takéto úlohy zvolíme.

Prvý program bude hľadať a ukladať do poľa všetky delitele nejakého celého čísla. Algoritmus už poznáme z predchádzajúcich častí kurzu, tu len uvidíme spôsob ukladania do poľa:

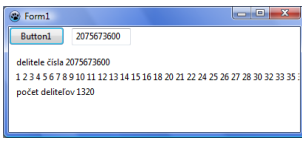
```
var
  Delitel: array [1..2000] of Integer;
  I, Cislo, X: Integer;
  Retazec: string;
begin
  Cislo := StrToInt(Edit1.Text);
  X := 0;
  for I := 1 to Cislo do
    if Cislo mod I = 0 then
    begin
      Inc(X);
      Delitel[X] := I;
    end;

  Retazec := '';
  for I := 1 to X do
    Retazec := Retazec + IntToStr(Delitel[I]) + ' ';
  Image1.Canvas.TextOut(10, 40,'delitele čísla ' + IntToStr(Cislo));
  Image1.Canvas.TextOut(10, 60, Retazec);
  Image1.Canvas.TextOut(10, 80,'počet deliteľov ' + IntToStr(X));
end;
```

Po spustení dostávame napr.



Na tomto príklade si všimnite, ako pracujeme s poľom: na začiatku je premenná **X**, ktorá označuje počet uložených hodnôt do poľa, nulová. Vždy, keď chceme pridať novú hodnotu, zvýšime premennú **X** o jedna a na toto nové miesto (prvok s indexom **X**) priradíme nový deliteľ.

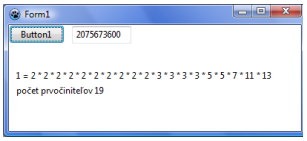
Pole deliteľov sme zadeklarovali na 2000 prvkov. Nižšie vidíme, že existuje celé číslo, ktoré má viac ako 1000 deliteľov a je možné, že ich môže byť ešte aj výrazne viac (zrejme menších ako 2^{31}).



V druhom príklade budeme ukladať všetky prvočinitele nejakého zadaného čísla. Našťastie rôznych prvočiniteľov pre čísla menšie ako 2^{31} nebude viac ako 31, preto aj stačí zadeklarovať menšie pole:

```
var
  Delitel: array [1..31] of Integer;
  I, Cislo, X: Integer;
  Retazec: string;
begin
  Cislo := StrToInt(Edit1.Text);
  X := 0;
  I := 2;
  while Cislo > 1 do
    if Cislo mod I = 0 then
    begin
      Inc(X);
      Delitel[X] := I;
      Cislo := Cislo div I;
    end
    else
      Inc(I);
```

Po spustení zadáme nejaké veľké číslo, napr. 2075673600:

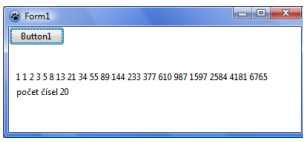



```
  Retazec := IntToStr(Cislo) + ' = ' ;
  for I := 1 to X do
    Retazec := Retazec + IntToStr(Delitel[I]) + ' * ';
  SetLength(Retazec, Length(Retazec) - 3);
  Image1.Canvas.TextOut(10, 60, Retazec);
  Image1.Canvas.TextOut(10, 80,'počet prvočiniteľov ' + IntToStr(X));
end;
```

Známym príkladom, v ktorom využívame už existujúce prvky poľa na to, aby sme pridali nový prvok, je Fibonacciho postupnosť. Prvé dve čísla do neho vložíme a ďalšie a každé ďalšie sa počíta ako súčet dvoch posledne priradených. Program vytvorí tabuľku všetkých Fibonacciho čísel, ktoré nie sú väčšie ako 10000:

```
var
  Fib: array [1..100] of Integer;
  I, N: Integer;
  Retazec: string;
begin
  Fib[1] := 1;
  Fib[2] := 1;
  N := 2;
  while Fib[N] <= 10000 do
  begin
    Fib[N+1] := Fib[N] + Fib[N-1];
    Inc(N);
  end;
  Dec(N);
  Retazec := '';
  for I := 1 to N do
    Retazec := Retazec + IntToStr(Fib[I]) + ' ';
  Image1.Canvas.TextOut(10, 60, Retazec);
  Image1.Canvas.TextOut(10, 80,'počet čísel ' + IntToStr(N));
end;
```

Po spustení dostávame:



Všimnite si, že cyklus skončí vtedy, keď sme vypočítali väčšie Fibonacciho čislo ako 10000. Preto už posledné číslo nemalo byť v tabuľke. Po skončení cyklu vidíme príkaz **Dec(N)**, ktorým toto posledné číslo z tabuľky *vyhodíme*. Zamyslite sa, či by ste vedeli upraviť podmienku while-cyklu tak, aby sa do tabuľky zbytočne nepripisovalo ďalšie číslo a cyklus by skončil správne.

Ďalší program bude hľadať prvočísla, ktoré nie sú väčšie ako 1000 a bude ich ukladať do poľa. Všimnite si, že na zisťovanie, či je nejaké číslo prvočíslom, využívame už skonštruovanú časť poľa:

```
var
  Prvocislo: array [1..300] of Integer;
  Cislo, I, N: Integer;
  Retazec: string;
begin
  N := 0;
  for Cislo := 2 to 1000 do
  begin
    I := 1;
    while (I <= N) and (Cislo mod Prvocislo[I] <> 0) do
      Inc(I);
    if I > N then
    begin
      Inc(N);
      Prvocislo[N] := Cislo;
    end;
  end;

  Retazec := '';
  for I := 1 to N do
    Retazec := Retazec + IntToStr(Prvocislo[I]) + ' ';
  Image1.Canvas.TextOut(10, 60, Retazec);
  Image1.Canvas.TextOut(10, 80,'počet prvočísel ' + IntToStr(N));
end;
```

Tento program už nie je až taký jednoduchý, ako boli tie predchádzajúce. Použitie

kompletnej už vytvorenej časti poľa pre ďalšie prvky si už vyžaduje istú programátorskú skúsenosť.

Čo sme sa naučili

Prvky poľa môžeme postupne vytvárať rôznymi algoritmami. Musíme mať pritom pomocnú premennú, ktorá si pamätá index posledne vloženého prvku.

V tejto časti sme ukázali nasledovné techniky:

- kopírovanie prvkov s nejakou vlastnosťou do samostatného poľa,
- vytvorenie poľa prvočiniteľov,
- vytvorenie poľa Fibonacciho čísiel,
- vytvorenie poľa prvočísiel do 1000.

Úlohy na precvičenie

Zadanie 1

Do poľa uložte takúto postupnosť: prvý prvok obsahuje 0, druhý vznikne tak, že zoberieme predchádzajúci a pričítame k nemu 1. Ďalší prvok vznikne ako súčet predchádzajúceho a čísla 3. Každé ďalšie číslo v postupnosti vzniká ako súčet predchádzajúceho ďalšieho nepárneho čísla, t.j. ďalej pripočítavame 5, 7, 11, ...

Vytvorte prvých 100 členov tejto postupnosti.

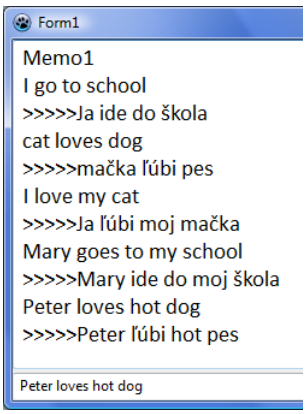
2. tematická jednotka – Algoritmy s poľami

1. Ďalšie pole v programe

Od úloh s jedným poľom teraz prejdeme k úlohám, ktoré pracujú s viacerými poľami. Prvá úloha bude robiť jednoduchý preklad z angličtiny do slovenčiny. Do formuláru vložíme textovú plochu **Memo1**, vstupný riadok **Edit1** a jedno tlačidlo **Button1**, ktorého zatlačenie vykoná:

```
var
  Angl: array [1..10] of string =
    ('I', 'love', 'my', 'dog', 'cat', 'go', 'school', 'to',
     'goes', 'loves');
  Slov: array [1..10] of string =
    ('Ja', 'ľúbi', 'moj', 'pes', 'mačka', 'ide', 'škola', 'do',
     'ide', 'ľúbi');
  Veta, Slovo, PSlovo, Preklad: string;
  I: Integer;
begin
  Veta := Edit1.Text + ' ';
  Memo1.Lines.Append(Veta);
  Preklad := '';
  while Veta <> '' do
  begin
    Slovo := Copy(Veta, 1, Pos(' ', Veta) - 1);
    Delete(Veta, 1, Pos(' ', Veta));
    PSlovo := Slovo;
    for I := 1 to 10 do
      if Slovo = Angl[I] then
        PSlovo := Slov[I];
    Preklad := Preklad + PSlovo + ' ';
  end;
  Memo1.Lines.Append('>>>>>' + Preklad);
end;
```

Program otestujeme na niekoľkých jednoduchých vetách:



V programe pracujeme s dvoma poľami. Prvé obsahuje 10 anglických slov. Druhé pole obsahuje 10 slovenských slov, ktoré presne zodpovedajú anglickým slovám. Napríklad, keď budeme chcieť preložiť anglické slovo "dog", nájdeme ho v prvom poli. Jeho index 4, preto jeho prekladom je 4. slovo v druhom poli, t.j. "'pes".

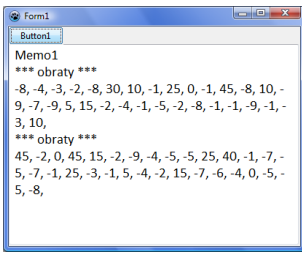
V programe využívame to, že už vieme rozobrať vetu na slová. Pre jednoduchosť programu budeme predpokladať, že slová sú navzájom oddelené práve jednou medzerou. Zápis **(Veta, 1, Pos(' ', Veta) - 1)** nájde prvý výskyt medzery (tá označuje koniec prvého slova vo vete) a vyberie podreťazec od 1. znaku až po túto medzeru. Nasledujúci príkaz **Delete(Veta, 1, Pos(' ', Veta))** toto prvé slovo z vety vyhodí. Všimnite si, že už pre prečítaní vstupného riadku do premennej **Veta** sme sem na koniec reťazca pridali jednu medzeru. Táto nám zjednoduší nasledovné rozoberanie vety na slová.

Všimnite si, že tie slová, ktoré nie sú v slovníku, sa neprekladajú, ale ostávajú bez zmeny. V našom teste to boli krstné mená "Mary", "Peter" a slovo "hot".

V ďalšom programe si budeme robiť evidenciu toho, ako míňame peniaze na našom účte. Každý deň si zapisujeme, koľko sme minuli, resp. koľko sme na účet pridali. Vytvárame si tabuľku, kde pre každý deň (od 1 do 31) máme zapísané buď kladné číslo (na účet sme pridali) alebo záporné číslo (z účtu sme vybrali). Vo formulári teraz máme jedno tlačidlo **Button1** a textovú plochu **Memo1**. Zapíšme najprv náhodné vygenerovanie tejto tabuľky:

```
var
  Obraty: array [1..31] of Integer;
  I: Integer;
  R: string;
begin
  Randomize;
  for I := 1 to 31 do
    if Random(5) = 0 then
      Obraty[I] := 5 * Random(10)
    else
      Obraty[I] := - Random(10);
  Memo1.Lines.Append('*** obraty ***');
  R := '';
  for I := 1 to 31 do
    R := R + IntToStr(Obraty[I]) + ', ';
  Memo1.Lines.Append(R);
end;
```

Po spustení a niekoľkých zatlačeniach tlačidla dostávame rôzne zaplnené pole **Obraty**, napr.

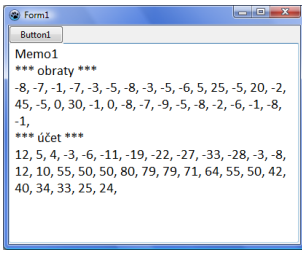


Príkaz **Randomize** na začiatku programu "zamieša generátor náhodných čísel" tak, aby každé spustenie aplikácie náhodne vygenerovalo iné náhodné čísla. Všimnite si podmienený príkaz **if** v tele cyklu. V každom prechode cyklu vygenerujeme jeden prvok tabuľky. Tu sa najprv rozhodneme, či to bude kladné alebo záporné číslo. Použili sme tu podmienku **Random(5) = 0**, ktorá označuje, že v pätine prípadov (keď jedna z piatich náhodne vygenerovaných hodnôt je 0) priradíme kladné číslo (alebo aj nulu) a vo zvyšných prípadoch (teda približne v 80%) to bude záporné číslo.

Teraz budeme vytvárať druhé pole - pole, ktoré uchováva každodenný stav účtu (premenná **Ucet**).

```
var
  Obraty, Ucet: array [1..31] of Integer;
  I, Stav: Integer;
  R: string;
begin
  Randomize;
  for I := 1 to 31 do
    if Random(5) = 0 then
      Obraty[I] := 5 * Random(10)
    else
      Obraty[I] := - Random(10);
  Memo1.Lines.Append('*** obraty ***');
  R := '';
  for I := 1 to 31 do
    R := R + IntToStr(Obraty[I]) + ', ';
  Memo1.Lines.Append(R);
  Stav := 20;
  for I := 1 to 31 do
  begin
    Ucet[I] := Stav + Obraty[I];
    Stav := Ucet[I];
  end;
  Memo1.Lines.Append('*** účet ***');
  R := '';
  for I := 1 to 31 do
    R := R + IntToStr(Ucet[I]) + ', ';
  Memo1.Lines.Append(R);
end;
```

Po spustení, dostávame približne takýto výstup:



V tomto programe sme zadeklarovali ďalšie pole **Ucet**, ktoré je rovnakého typu ako pole **Obraty**. Pri jeho zapĺňaní hodnotami, používame pomocnú premennú **Stav**, ktorá má na začiatku hodnotu 20 (tu sme si zvolili počiatočnú hodnotu stavu účtu ako 20€).

S takto pripravenými poľami s informáciami o každodenných obratoch a stavoch na účte môžeme riešiť množstvo rôznych úloh, napr.

- koľko dní bol účet v mínuse?
- koľko dní bolo na účte viac ako 100€?
- koľkokrát sa z účtu vyberalo, pritom bol v mínuse?

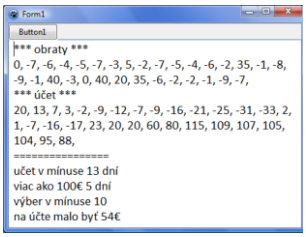
Asi by ste zvládli vytvoriť aj krajší, čitateľnejší výstup.

- akú minimálnu čiastku sme mali mať na účte na začiatku mesiaca, aby bolo na účte každý deň aspoň 1€?

Nasledujúca časť programu ilustruje, ako môžeme zapísať riešenie týchto štyroch úloh.

```
var
  Obraty, Ucet: array [1..31] of Integer;
  I, Stav: Integer;
  R: string;
  VMinuse, Viac100, VyberVMinuse, Min: Integer;
begin
  ...
  VMinuse := 0;
  Viac100 := 0;
  VyberVMinuse := 0;
  Min := Ucet[1];
  for I := 1 to 31 do
  begin
    if Ucet[I] < 0 then
      Inc(VMinuse);
    if Ucet[I] >= 100 then
      Inc(Viac100);
    if (Obraty[I] < 0) and (Ucet[I] < Obraty[I]) then
      Inc(VyberVMinuse);
    if Ucet[I] < Min then
      Min := Ucet[I];
  end;
  Memo1.Lines.Append('================');
  Memo1.Lines.Append('účet v mínuse ' + IntToStr(VMinuse) + ' dní');
  Memo1.Lines.Append('viac ako 100€ ' + IntToStr(Viac100) + ' dní');
  Memo1.Lines.Append('výber v mínuse ' + IntToStr(VyberVMinuse));
  Memo1.Lines.Append('na účte malo byť ' + IntToStr(21-Min) + '€');
end;
```

Po spustení môžeme dostať takýto výpis:



Uvedený konkrétny výpis na bočnom paneli označuje, že účet bol 13 dní v mínuse, ale 5 dní na ňom bolo viac ako 100€. Z účtu sa vyberalo, napriek tomu, že bol v mínuse, až 10-krát. Ale keby na ňom na začiatku nebolo iba 20€, ale až 54€, tak celý mesiac by na ňom bolo aspoň 1€.

V ďalšom príklade budeme ukladať pohyby myši po grafickej ploche do dvoch polí **XX** a **YY**. Premenná **Pocet** bude uchovávať počet uložených bodov v týchto poliach. V aplikácii bude ešte jedno tlačidlo, ktoré prekreslí zapamätanú kresbu náhodnou farbou.

```
var
  XX, YY: array [1..10000] of Integer;
  Pocet: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Image1.Canvas.FillRect(Image1.ClientRect);
  Image1.Canvas.Pen.Width := 3;
end;
```

Premenné **XX**, **YY** a **Pocet** musia byť globálne. Udalosti **onMouseDown** a **onMouseMove** okrem kreslenia ukladajú súradnice bodov do polí **XX** a **YY**:

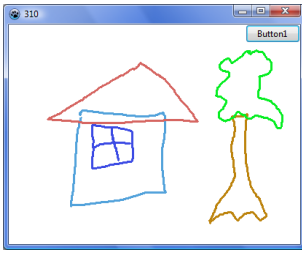
```
procedure TForm1.Image1MouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Pocet := 1;
  XX[1] := X;
  YY[1] := Y;
  Image1.Canvas.Pen.Color := clBlack;
  Image1.Canvas.MoveTo(X, Y);
end;
```

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

Zatlačenie **Button1** prekreslí posledne vytvorenú kresbu nejakou náhodnou farbou. Keď tlačidlo zatlačíme viackrát za sebou, zakaždým sa kresba kreslí novou farbou.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  Caption := IntToStr(Pocet);
  Image1.Canvas.Pen.Color := Random(256 * 256 * 256);
  Image1.Canvas.MoveTo(XX[1], YY[1]);
  for I := 2 to Pocet do
    Image1.Canvas.LineTo(XX[I], YY[I]);
end;
```

Spustíme program a otestujeme, napríklad:



Polia môžeme využiť aj na uloženie cifier nejakého veľkého čísla, napr. 20-ciferného. Preto si pripravíme takéto pole:

array [1..20] of Integer

Každý prvok poľa označuje jednu cifru veľkého čísla. Cifra s indexom 1 označuje najnižší rád. Ak budeme mať v takomto poli napr. trojciferné číslo 765, tak **Cislo[1]** bude 5, **Cislo[2]** bude 6, **Cislo[3]** bude 7 a všetky ostatné cifry budú 0. Takéto veľké čísla budeme vypisovať ako postupnosť cifier, pričom úvodné nuly sa pri výpise nahradia medzerami.

Teraz chceme vytvoriť procedúru, ktorá dostane ako **parameter** veľké číslo, t.j. dvadsať prvkové pole. Procedúra vypíše toto veľké číslo do textovej plochy. V pascale platí jedno špeciálne pravidlo, ktoré sme zatiaľ spomínať nemuseli: ak je parametrom pole, tak toto pole treba zadeklarovať ako nový typ. Pozrite nasledujúci program: zadefinovali sme nový typ s menom **VelkeCislo** a potom sme vytvorili procedúru **Vypis**, ktorá vypíše veľké číslo.

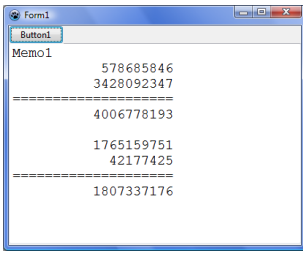
```
type
  VelkeCislo = array [1..20] of Integer;

procedure Vypis(Cislo: VelkeCislo);
var
  I: Integer;
  Retazec: string;
begin
  Retazec := '';
  I := 20;
  while (I > 1) and (Cislo[I] = 0) do
  begin
    Retazec := Retazec + ' ';
    Dec(I);
  end;
  While I >= 1 do
  begin
    Retazec := Retazec + IntToStr(Cislo[i]);
    Dec(I);
  end;
  Memo1.Lines.Append(Retazec);
end;
```

Tento typ aj procedúru **Vypis** deklarujeme vo vnútri procedúry **Button1Click**.

Keď už máme definovaný nový typ, vytvoríme dve premenné tohto typu **Cislo1** a **Cislo2**, vygenerujeme do nich nejaké náhodné čísla a vypíšeme ich. Potom obe veľké čísla sčítame (tak, ako by sme to robili ručne na papieri) a znovu vypíšeme:

```
var
  Cislo1, Cislo2: VelkeCislo;
  I, J, Prenos, Sucet: Integer;
begin
  J := Random(10) + 5;
  for I := 20 downto J+1 do
    Cislo1[I] := 0;
  for I := J downto 1 do
    Cislo1[I] := Random(10);
  J := Random(10) + 5;
  for I := 20 downto J+1 do
    Cislo2[I] := 0;
  for I := J downto 1 do
    Cislo2[I] := Random(10);
  Vypis(Cislo1);
  Vypis(Cislo2);
  Memo1.Lines.Append('====================');
  Sucet := 0;
  Prenos := 0;
  for I := 1 to 20 do
  begin
    Sucet := Cislo1[I] + Cislo2[I] + Prenos;
    Cislo1[I] := Sucet mod 10;
    Prenos := Sucet div 10;
  end;
  Vypis(Cislo1);
  Memo1.Lines.Append('');
end;
```

Program spustíme a každé zatlačenie tlačidla vygeneruje iné dve dvojice čísel, ktoré sčíta. Môžeme dostať napr.



Obe náhodné veľké čísla generujeme tak, že najprv náhodne vygenerujeme, koľko bude platných cifier (premenná **J**), cifry od 20 do **J** vynulujeme a zvyšne opäť generujeme náhodne. Všimnite si, ako sme použili premennú Prenos pri sčítavaní cifier dvoch čísel.

Úlohy na precvičenie

Zadanie 1	Program bude simulovať prechod autobusu zastávkami MHD. Na každej zo zastávok čaká niekoľko cestujúcich, pričom každý sa chce odviesť presne 2 zastávky. Kapacita autobusu je 20 cestujúcich. Vypíšte, ako sa zmenia počty čakajúcich na zastávkach, keď cez ne prejde autobus: na každej môže niekoľko ľudí vystúpiť a aj nastúpiť.
Zadanie 2	Program umožní kresliť v grafickej ploche ťahaním myši. Popritom si bude zapamätávať súradnice kresby. Kliknutie do plochy pravým tlačidlo myši znovu nakreslí celú kresbu, ale už od novej pozície.
Zadanie 3	Využite ideu veľkých čísel a zapíšte algoritmus na výpočet mocniny N^K. Najprv do veľkého čísla vložte cifry čísla **N** (nemusí byť veľmi veľké) a potom toto veľké číslo **K-1** krát násobte číslom **N**.
Zadanie 4	Využite ideu veľkých čísel a vypočítajte faktoriál nejakého väčšieho čísla.

2. Ďalšie pole iného typu

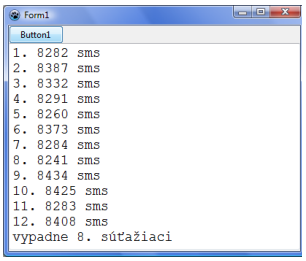
Doteraz naše programy pracovali s jedným, prípadne s viacerými poľami, ale v jednom programe bol iba jeden typ poľa. Nasledujúci príklad ukáže situáciu, keď budeme potrebovať viac polí rôznych typov.

Ukážme to na takomto príklade. Získali sme informácie o SMS hlasovaní v televíznej súťaži *Superstar*. Do hlasovania prišlo 100000 SMS, pričom v každej sa mohol dať jeden hlas jednému z 12 súťažiacich. Našou úlohu bude zistiť, ktorý súťažiaci dostal najmenej hlasov a teda zo súťaže vypadne.

Hlasovacie SMS uložíme do poľa, pričom každý jeho prvok bude obsahovať jedno z čísel od 1 do 12. Okrem toho zadefinujeme 12-prvkové pole, v ktorom budeme spočítavať počty hlasov pre jednotlivých súťažiacich.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SMS: array [1..100000] of Integer;
  Hlas: array [1..12] of Integer;
  I, Min: Integer;
begin
  for I := 1 to 100000 do
    SMS[I] := Random(12) + 1;
  for I := 1 to 12 do
    Hlas[I] := 0;

  for I := 1 to 100000 do
    Inc(Hlas[SMS[I]]);

  Min := 1;
  for I := 1 to 12 do
  begin
    Memo1.Lines.Append(IntToStr(I) + '. ' + IntToStr(Hlas[I])+' sms');
    if Hlas[I] < Hlas[Min] then
      Min := I;
  end;
  Memo1.Lines.Append('vypadne ' + IntToStr(Min) + '. súťažiaci');
end;
```

Po spustení môžeme dostať takýto výpis.



Najzaujímavejšia časť programu je tento príkaz: **Inc(Hlas[SMS[I]])**. Pre začiatočníka toto môže byť ťažké prečítať. Rozložme ho na niekoľko krokov:

```
  ItaSMS := SMS[I];
  Hlas[ItaSMS] := Hlas[ItaSMS] + 1;
```

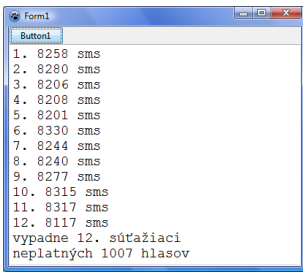
Znamená: zoberme I-tu SMS. Jej hodnota je číslo od 1 do 12 a to je indexom do druhej tabuľky, v ktorej evidujeme počty hlasov. Takže v tejto druhej tabuľke zväčšíme o jedna práve ten riadok, ktorého hlas sme zistili v SMS.

Takémuto typu algoritmu hovoríme **frekvenčná tabuľka**. Pri frekvenčných tabuľkách sa niekedy môže stať, že sa objavia aj správy, ktoré sú neplatné (obsahujú nejaké iné číslo ako od 1 do 12) a preto ich treba špeciálne ošetriť. Upravme predchádzajúci program, aby správne spracoval aj neplatné hlasy.

```
var
  SMS: array [1..100000] of Integer;
  Hlas: array [1..12] of Integer;
  I, Min, Neplatne: Integer;
begin
  for I := 1 to 100000 do
    if Random(100) = 0 then
      SMS[I] := Random(100) + 13
    else
      SMS[I] := Random(12) + 1;
  for I := 1 to 12 do
    Hlas[I] := 0;
```

```
  Neplatne := 0;
  for I := 1 to 100000 do
    if (SMS[I] >= 1) and (SMS[I] <= 12) then
      Inc(Hlas[SMS[I]])
    else
      Inc(Neplatne);

  Min := 1;
  for I := 1 to 12 do
  begin
    Memo1.Lines.Append(IntToStr(I) + '. ' + IntToStr(Hlas[I])+' sms');
    if Hlas[I] < Hlas[Min] then
      Min := I;
  end;
  Memo1.Lines.Append('vypadne ' + IntToStr(Min) + '. súťažiaci');
  Memo1.Lines.Append('neplatných ' + IntToStr(Neplatne) + ' hlasov');
end;
```

Po spustení už vidíme, že program spracoval aj neplatné hlasy, napr.:



Čo sme sa naučili

V programe môžeme používať viac polí rôznych typov.

V tejto časti sme ukázali nasledovné techniky:

- veľké čísla uložené v poli po cifrách, súčet takýchto čísel,
- frekvenčná tabuľka,

Úlohy na precvičenie

Zadanie 1	Napíšte program, ktorý bude zaznačovať u žiakov v triede, koľkokrát sa medzi nimi vyskytli, ktoré záujmy. Program najprv vypíše legendu: 1=známky, 2=modelár, 3=hudba, 4=tanec, 5=šport, 6=kniha, 7=domáce zviera a očakáva zadávanie čísel od 1 do 7. Pritom si eviduje počet výskytov týchto čísel. Na záver (po stlačení **Button2**) program vypíše, ktorý koníček bol najčastejší a ktorý najzriedkavejší.
Zadanie 2	Napíšte program, do ktorého budeme cez vstupný riadok zadávať krstné mená žiakov. Tieto sa budú vypisovať do textovej plochy a zároveň sa bude evidovať ich počet výskytov. Na záver (po stlačení **Button2**) program vypíše najčastejšie sa vyskytujúce mená.
Zadanie 3	Program bude pre lotériu generovať 10 náhodných čísel od 1 do 100. Treba zabezpečiť, aby to boli náhodné čísla. Použite pomocné pole, v ktorom si budete ukladať už vygenerované čísla. Pri každom ďalšom skontrolujte, či sa už medzi skôr vylosovanými nenachádza.
Zadanie 4	Riešte predchádzajúcu úlohu ale tak, že použijete pomocné logické pole veľkosti 100. V tomto poli budete značiť už vylosované čísla tak, že príslušným políčkam zmeníte logickú hodnotu.

3. Pomocné pole

Nasledujúci príklad ilustruje použitie poľa pre zjednodušenie vypisovania údajov. V príklade budeme vypisovať zadaný text zakódovaný Morseovou abecedou. Použijeme pomocné pole, v ktorom pre každé malé písmeno anglickej abecedy (je to 26 písmen od 'a' do 'z') dostaneme znakový reťazec z bodiek a čiarok – práve Morseove písmeno. Teda radi by sme, aby indexom nemuselo byť číslo z nejakého intervalu, ale písmeno od 'a' do 'z'. Našťastie Pascal umožňuje ako index definovať ľubovoľný interval čísel, znakov alebo logických hodnôt. Napr. toto sú korektné zápisy polí:

```
var
  Pole1: array [0..99] of Integer;
  Pole2: array [100..200] of Integer;
  Pole3: array [-5..5] of Integer;
  Pole4: array ['A'..'Z'] of Integer;
  Pole5: array ['0'..'9'] of Integer;
  Pole6: array [Char] of Integer;
  Pole7: array [Boolean] of Integer;
```

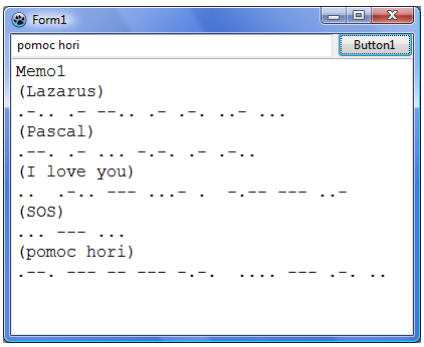
- **Pole1** označuje 100-prvkové pole celých čísel, pričom povolené indexy sú od 0 do 99.
- **Pole2** je 101 prvkové pole s indexmi od 100 do 200.
- **Pole3** je 11 prvkové pole, kde indexmi môžu byť aj záporné čísla.
- **Pole4** je 26 prvkové pole, indexmi sú len veľké písmená.
- **Pole5** je 10 prvkové pole, kde indexmi sú len znaky cifier od '1' do '9'.
- **Pole6** je 256 prvkové pole, indexujeme ľubovoľným znakom.
- **Pole7** je dvojprvkové pole, indexovať môžeme buď hodnotou **False** alebo **True**.

V našom programe budeme potrebovať indexovať malými písmenami, preto deklarované pole reťazcov bude mať indexy z intervalu **'a'..'z'**:

```
var
  Morse: array ['a'..'z'] of string =
    ('.-', '-...', '-.-.', '-..', '.', '..-.', '--.', '....', '..',
     '.---', '-.-', '.-..', '--', '-.', '---', '.--.', '--.-', '.-.',
     '...',  '-', '..-', '...-', '.--', '-..-', '-.--', '--..');
  Vstup, Vystup: string;
  Znak: Char;
  I: Integer;
begin
  Vstup := LowerCase(Edit1.Text);
  Vystup := '';
  for I := 1 to Length(Vstup) do
  begin
    Znak := Vstup[I];
    if (Znak >= 'a') and (Znak <= 'z') then
      Vystup := Vystup + Morse[Znak];
    Vystup := Vystup + ' ';
  end;
  Memo1.Lines.Append('(' + Edit1.Text + ')');
  Memo1.Lines.Append(Vystup);
end;
```

Kódy Morseovej abecedy môžeme nájsť na internete (napr. na Wikipédii) a priamo v deklarácii poľa **Morse** nimi inicializujeme celé toto pole. Aplikácia prečíta zo vstupného reťazca (**Edit1**) nejakú vetu, rozoberie ju na písmená a každé jedno prerobí na Morse postupnosť bodiek a čiarok. Výstupný reťazec potom vypíše do textovej plochy (**Memo1**).

Program spustíme a zadáme nejaké testovacie slová a vety:



Inicializované pole môžeme využiť na vygenerovanie, napríklad náhodných farieb:

```
var
  Farba: array [0..5] of TColor =
    (clRed, clBlue, clGreen, clYellow, clFuchsia, clAqua);
begin
  ...
  Image1.Canvas.Brush.Color := Farba[Random(6)];
  Image1.Canvas.Pen.Color := Farba[Random(4)];
  ...
end;
```

Tiež pole môžeme využiť aj pri výpise logickej hodnoty, napr.

```
var
  Log: array [Boolean] of string
    ('Pravda', 'Nepravda');
  A, B: Boolean;
begin
  ...
  Memo1.Lines.Append(Log[A] + ' or ' + Log[B] + ' = ' + Log[A or B]);
end;
```

Ak napr. **A** je **True** a **B** je **False**, program tu vypíše: **Pravda or Nepravda = Pravda**.

Môžeme skonštruovať frekvenčnú tabuľku, v ktorej budeme zisťovať, ktoré z písmen bolo v nejakej vete najčastejšie:

```
var
  Tab: array [Char] of Integer;
  Veta: string;
  I: Integer;
  Znak, Max: Char;
begin
  for Znak := Char(0) to Char(255) do
    Tab[Znak] := 0;
  Veta := Edit1.Text;
  for I := 1 to Length(Veta) do
    Inc(Tab[Veta[I]]);
  Max := Char(0);
  for Znak := Char(0) to Char(255) do
    if Tab[Znak] > Tab[Max] then
      Max := Znak;
  Memo1.Lines.Append('najčastejší znak vo vete je "' + Max + '"');
end;
```

Opäť je tu zaujímavá konštrukcia **Inc(Tab[Veta[I]])**. Preštudujte ju.

Posledný príklad v tejto časti pre zadaný dátum vypíše, ktorý je to deň v týždni. Využijeme na to štandardné funkcie **DayOfWeek** a **EncodeDate**, pričom volanie **DayOfWeek(EncodeDate(*Rok*, *Mesiac*, *Deň*))** vypočíta poradový deň v týždni, t.j. číslo od 1 do 7 (1 pre nedeľu, 2 pre pondelok, ...).

```
var
  DMeno: array [1..7] of string =
    ('nedeľa', 'pondelok', 'utorok', 'streda', 'štvrtok', 'piatok',
     'sobota');
  MMeno: array [1..12] of string =
    ('január', 'február', 'marec', 'apríl', 'máj', 'jún', 'júl',
     'august', 'september', 'október', 'november', 'december');
  R: string;
  Den, Mesiac, Rok, DenVTyzdni: Integer;
begin
  R := Edit1.Text;
  Den := StrToInt(Copy(R, 1, Pos('.', R) - 1));
  Delete(R, 1, Pos('.', R));
  Mesiac := StrToInt(Copy(R, 1, Pos('.', R) - 1));
  Delete(R, 1, Pos('.', R));
  Rok := StrToInt(R);
  DenVTyzdni := DayOfWeek(EncodeDate(Rok, Mesiac, Den));
  Memo1.Lines.Append(IntToStr(Den) + '. ' + MMeno[Mesiac] + ' ' +
    IntToStr(Rok) + ' ... ' + DMeno[DenVTyzdni]);
end;
```

Program očakáva, že do vstupného riadka **Edit1** zadáme nejaký dátum, napr. v tvare 19.10.2009. Potom tento dátum rozoberie do troch celočíselných premenných **Den**, **Mesiac**, **Rok**, zistí deň v týždni a všetko to vypíše do textovej plochy **Memo1** v tvare **19. október 2009 ... pondelok**.

Čo sme sa naučili

Indexy polí nemusia byť len intervaly čísel začínajúce 1 po nejaké iné celé číslo. Indexom môže byť ľubovoľný interval aj čísel a logických hodnôt. Výhodne sa v takýchto úlohách využívajú inicializované hodnoty polí.

Úlohy na precvičenie

Zadanie 1

Napíšte program, ktorý bude počítať prevod nejakej sumy v cudzej mene na eurá. Vo vstupnom riadku zadáme nejakú sumu a názov meny, napr. **15.50 GBP** a program vypíše sumu zaokrúhlenú na 2 desatinné miesta. Názvy mien môžu byť napr. CZK, GBP, HUF, CHF, PLN, USD, ...

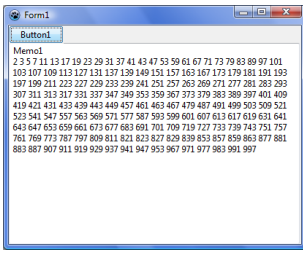
4. Eratostenove sito

Eratostenovo sito je algoritmus, ktorý hľadá všetky prvočísla do nejakej hodnoty metódou, ktorú vymyslel grécky matematik ešte približne 2 storočia pred našim letopočtom. Princíp algoritmu je nasledovný: vypíšeme si všetky čísla od 2 do nejakej maximálnej hodnoty (napr. 1000). Zoberieme prvé z nich (teda 2) a všetky jeho násobky označíme, že už určite prvočíslami nebudú. Zoberieme ďalšie najmenšie ešte neoznačené číslo (teda 3) a opäť označíme všetky jeho násobky. Takto pokračujeme so všetkými neoznačenými číslami. To čo ostane neoznačené, sú hľadané prvočísla.

Pre tento algoritmus použijeme 1000-prvkové logické pole. Hodnota **True** bude označovať, že je to kandidát na prvočíslo. Hodnota **False** znamená, že sme už príslušné číslo označili ako nejaký násobok a teda už to prvočíslo určite nebude. Program najprv inicializuje celé pole - okrem 1 všetkým prvkom nastavíme **True**. Potom postupne prechádzame všetky prvky a ak majú hodnotu **True**, tak vo while-cykle označíme všetky jeho násobky ako **False**.

```
var
  ErSito: array [1..1000] of Boolean;
  I, J: Integer;
  Retazec: string;
begin
  ErSito[1] := False;
  for I := 2 to 1000 do
    ErSito[I] := True;
  for I := 1 to 1000 do
    if ErSito[I] then
    begin
      J := 2 * I;
      while J <= 1000 do
      begin
        ErSito[J] := False;
        Inc(J, I);
      end;
    end;

  Retazec := '';
  for I := 1 to 1000 do
    if ErSito[I] then
      Retazec := Retazec + IntToStr(I) + ' ';
  Memo1.Lines.Append(Retazec);
end;
```

Po spustení programu dostávame:



Záverečná časť programu pripravuje zoznam všetkých prvočísel pre výpis. Samozrejme, že nevypisujeme hodnoty poľa, ale indexy tých prvkov, ktorých hodnota je **True**.

Program môžeme ešte vylepšiť. Matematici by iste navrhli, aby sme zbytočne neoznačovali násobky všetkých prvočísel, ale malo by stačiť len tých, ktoré nie sú väčšie ako druhá odmocnina z 1000, t.j len čísla do 31. Preto základný for-cyklus nahradíme while-cyklom.

Okrem toho tu použijeme ešte jednu novinku. Keďže program hľadá prvočísla do 1000, na viacerých miestach v programe sa vyskytuje konštanta 1000. Keby sme to chceli zmeniť, aby sa hľadali prvočísla, napr. do 2500, museli by sme všetky konštanty 1000 nahradiť číslom 2500. Keďže nie vždy takéto nahrádzanie môže byť takto jednoduché, odporúča sa namiesto 1000 radšej písať pomenovanú konštantu napr. **N (const N = 1000;)** Hoci existuje aj iný spôsob, ako urobiť program univerzálnejší.

Hoci nie vždy je to také jednoduché - v niektorých iných programoch by 1000 mohlo znamenať aj niečo iné ako dĺžku poľa. A zase v programe by nejaká 999 mohla znamenať dĺžku poľa - 1, ktorú by sme si mohli nevšimnúť.

Programy, ktoré pracujú s poľami, nemusia používať číselné konštanty ako horné hranice indexu, ale pomocou štandardnej funkcie **High** sa môžu odvolávať na túto hornú hranicu.

Takže namiesto

for I := 2 to 1000 do

môžeme zapísať

for I := 2 to High(ErSito) do

Takýto zápis teraz označuje, že by mohol správne fungovať pre ľubovoľne zadeklarované pole. Upravme teraz program pomocou funkcie **High**.

```
var
  ErSito: array [1..10000] of Boolean;
  I, J: Integer;
  Retazec: string;
begin
  ErSito[1] := False;
  for I := 2 to High(ErSito) do
    ErSito[I] := True;
  I := 2;
  while I * I <= High(ErSito) do     // I <= Sqrt(High(ErSito))
  begin
    if ErSito[I] then
    begin
      J := 2 * I;
      while J <= High(ErSito) do
      begin
        ErSito[J] := False;
        Inc(J, I);
      end;
    end;
    Inc(I);
  end;
  Retazec := '';
  for I := 1 to High(ErSito) do
    if ErSito[I] then
      Retazec := Retazec + IntToStr(I) + ' ';
  Memo1.Lines.Append(Retazec);
end;
```

Ešte ukážme, ako z takto vytvoreného Eratostenovho sita (pole logických hodnôt) vytvoríme pole prvočísel:

```
var
  ErSito: array [1..10000] of Boolean;
  Prvocislo: array [1..2000] of Integer;
  Pocet: Integer;
  ...
begin
  ...
  Pocet := 0;
  for I := 1 to High(ErSito) do
    if ErSito[I] then
    begin
      Inc(Pocet);
      Prvocislo[Pocet] := I;
    end;
  Retazec := '';
  for I := 1 to Pocet do
    Retazec := Retazec + IntToStr(Prvocislo[I]) + ' ';
  Memo1.Lines.Append(Retazec);
  Memo1.Lines.Append('počet = ' + IntToStr(Pocet));
end;
```

5. Vkladanie do poľa na správne miesto

Pani učiteľka má v poli **Ziak** znakových reťazcov menný zoznam žiakov. Samozrejme, že tento zoznam má usporiadaný podľa abecedy. Keďže dopredu počítala s tým, že počet žiakov sa môže meniť, vyhradila väčšie pole, ako je terajší počet žiakov. Preto má okrem neho aj pomocnú premennú **Pocet**, ktorá určuje aktuálny počet žiakov v poli.

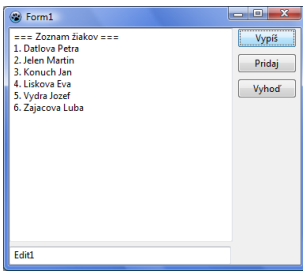
Budeme pripravovať aplikáciu, v ktorej umožníme toto pole vypísať, pridať do neho nové meno alebo nejaké meno zo zoznamu vyhodiť. Samotné pole **Ziak** a aj premenná **Pocet** musia byť globálne premenné. Vložme do formulára textovú plochu (**Memo1**), vstupný riadok (**Edit1**) a tri tlačidlá (**Button1**, **Button2**, **Button3**).

Radi by sme do poľa **Ziak** priradili nejaké hodnoty už pri štarte programu. Vieme inicializovať pole, ale len vtedy, keď tým zaplníme všetky prvky. Lenže my by sme radi zaplnili, napr. len prvých šesť hodnôt. Pole preto budeme inicializovať už v udalosti **FormCreate** (tam kde zvykneme zapisovať vymazanie grafickej plochy):

```
var
  Ziak: array [1..50] of string;
  Pocet: Integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Ziak[1] := 'Datlova Petra';
  Ziak[2] := 'Jelen Martin';
  Ziak[3] := 'Konuch Jan';
  Ziak[4] := 'Liskova Eva';
  Ziak[5] := 'Vydra Jozef';
  Ziak[6] := 'Zajacova Luba';
  Pocet := 6;
end;
```

Mená sme úmyselne zapísali bez diakritiky, nakoľko porovnávanie znakových reťazcov, ktoré obsahujú diakritiku, dáva úplne iné výsledky, ako by sme očakávali. Doplníme vypísanie obsahu poľa. Mohlo by to robiť zatlačenie tlačidla **Button1**:

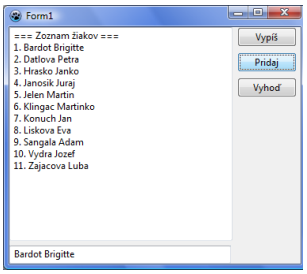
Spustíme a otestujeme výpis poľa zatlačením prvého tlačidla:




```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  Memo1.Clear;
  Memo1.Lines.Append('=== Zoznam žiakov ===');
  for I := 1 to Pocet do
    Memo1.Lines.Append(IntToStr(I) + '. ' + Ziak[I]);
end;
```

Ďalším krokom tejto aplikácie bude tlačidlo, ktoré vsunie nové meno do tabuľky mien. Procedúra najprv prečíta vstupný riadok do premennej **Novy**, potom nájde správne miesto, kam v poli zaradí tento reťazec, priradí ho tam, zväčší počet žiakov a vypíše zoznam (zavolá zatlačenie tlačidla **Button1**, t.j. **Button1.Click**):

Funkčnosť preveríme pridaním niekoľkých nových mien:




```
procedure TForm1.Button2Click(Sender: TObject);
var
  I: Integer;
  Novy: string;
begin
  Novy := Edit1.Text;
  I := Pocet;
  while (I >= 1) and (Ziak[I] > Novy) do
  begin
    Ziak[I+1] := Ziak[I];
    Dec(I);
  end;
  Ziak[I+1] := Novy;
  Inc(Pocet);
  Button1.Click;
end;
```

Pozorne preštudujte while-cykus, ktorý "rozsúva" prvky poľa tak, aby sme na správne

miesto zaradili nové meno žiaka.

Posledným krokom aplikácie bude vyhodenie zvoleného mena z tabuľky. Aby sme mohli zadané meno vyhodiť, musíme najprv nájsť jeho index a potom všetky prvky v poli za týmto indexom presunúť na nižšie indexy. Na hľadanie mena v tabuľke pripravíme pomocnú procedúru **Hladaj**. Zadefinujeme ju ako globálnu:

```
var
  Nasiel: Integer;

procedure Hladaj(R: string);
begin
  Nasiel := 1;
  while (Nasiel <= Pocet) and (R > Ziak[Nasiel]) do
    Inc(Nasiel);

  if (Nasiel > Pocet) or (R <> Ziak[Nasiel]) then
    Nasiel := 0;
end;
```

Tento podprogram by sa elegantnejšie riešil pomocou funkcie. Použitie procedúry a globálnej premennej je len "náhradné" riešenie.

Funkciám sa budeme venovať v 9. module.

Táto procedúra postupne prechádza od prvého prvku až po hľadanú hodnotu, resp. skončí vtedy, keď nájde taký, ktorý je v abecede až za ním. Dobre preštudujte podmienku while-cyklu: kým sme neprešli všetky prvky a porovnávaný prvok je (abecedne) stále menší, ako ten čo hľadáme, presúvaj sa na ďalší. Tu je dôležité si uvedomiť, pri akej podmienke tento cyklus môže skončiť. Je to vždy negácia podmienky while-cyklu, t.j.

(Nasiel > Pocet) or (R <= Ziak[Nasiel])

A v tomto konkrétnom programe to znamená, že sme buď neúspešne prešli celé pole, alebo sme našli prvok, ktorý sa buď rovná hľadanému alebo je už (abecedne) väčší. Preto máme záverečný test, ktorý vylúči neúspešné prípady.

Samotné vyhodenie prvku poľa potom vyzerá takto:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  Hladaj(Edit1.Text);
  if Nasiel > 0 then
  begin
    I := Nasiel;
    while I < Pocet do
    begin
      Ziak[I] := Ziak[I+1];
      Inc(I);
    end;
    Dec(Pocet);
    Button1.Click;
  end;
end;
```

Po spustení asi rýchlo zistíme takéto správanie:

- keď nejaké meno zadáme správne, tak sa korektne vyhodí zo zoznamu,
- keď meno zadáme chybne, program nič nevyhodí, ale ani nič neoznámi,
- chybou je aj, keď prvé písmeno mena nebude veľké ale malé.

Pokúsime sa vylepšiť procedúry **Hladaj** aj **Button1Click** tak, aby program oznámil neúspešne zadané meno, ale aby tiež pri hľadaní mena v zozname nerozlišoval malé a veľké písmená:

```
var
  Nasiel: Integer;

procedure Hladaj(R: string);
begin
  R := UpperCase(R);
  Nasiel := 1;
  while (Nasiel <= Pocet) and (R > UpperCase(Ziak[Nasiel])) do
    Inc(Nasiel);
  if (Nasiel > Pocet) or (R <> UpperCase(Ziak[Nasiel])) then
    Nasiel := 0;
end;
```

Štandardná funkcia **UpperCase** vráti zadaný reťazec tak, že všetky malé písmená v ňom prerobí na veľké. Napríklad **UpperCase('jeLEN aDAm') = 'JELEN ADAM'**

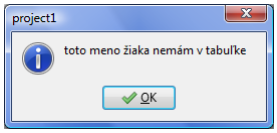
V tejto procedúra sa najprv prerobí zadaný reťazec na veľké písmená (**UpperCase**) a

potom sa porovná so všetkými reťazcami v zozname. Aj tieto sa pri porovnávaní prerábali na veľké písmená.

Do procedúry **Button3Click** sme pridali výpis správy, v prípade, keď sa zadaný reťazec nenašiel v tabuľke:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  I: Integer;
begin
  Hladaj(Edit1.Text);
  if Nasiel = 0 then
    ShowMessage('toto meno žiaka nemám v tabuľke')
  else
  begin
    I := Nasiel;
    while I < Pocet do
    begin
      Ziak[I] := Ziak[I+1];
      Inc(I);
    end;
    Dec(Pocet);
    Button1.Click;
  end;
end;
```

Nový príkaz **ShowMessage**, ktorý sme tu použili, vypíše zadanú správu a program potom ďalej pokračuje. V našom prípade sa objaví takáto správa:



Keď už sme sa naučili príkaz **ShowMessage** a máme aj pomocnú procedúru **Hladaj**, môžeme vylepšiť aj vkladanie nového mena do tabuľky. Pridáme test na zaplnenie tabuľky (ak **Pocet** = 50, t.j. **High(Ziak)**, tabuľka je plná a preto sa už do nej nedá pridávať), ale aj test, aby sa nedalo to isté meno pridať viackrát.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  I: Integer;
  Novy: string;
begin
  if Pocet = High(Ziak) then
    ShowMessage('tabuľka je už plná')
  else
  begin
    Novy := Edit1.Text;
    Hladaj(Novy)
    if Nasiel > 0 then
      ShowMessage('toto meno už je v tabuľke')
    else
    begin
      I := Pocet;
      while (I >= 1) and (UpperCase(Ziak[I]) > UpperCase(Novy)) do
      begin
        Ziak[I+1] := Ziak[I];
        Dec(I);
      end;
      Ziak[I+1] := Novy;
      Inc(Pocet);
      Button1.Click;
    end;
  end;
end;
```

Keďže procedúra **Button2Click** volá **Hladaj**, definíciu tejto procedúry (aj s premennou **Nasiel**) musíme presťahovať ešte pred **Button2Click** - možno najlepšie hneď pod deklarácie poľa **Ziak** a premennej **Pocet**.

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- štruktúrovaný typ pole
- interval rozsahu indexu

Okrem toho boli uvedené tieto dôležité koncepty:

- chyba rozsahu indexu poľa
- frekvenčná tabuľka
- algoritmus Eratostenove sito

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Pomocou techniky veľkých čísel prevedie nejaké celé číslo do poľa v zadanej číselnej sústave.

Literatúra a použité zdroje

Základné materiály:

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

Internetové zdroje pre prostredie Delphi

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

Internetové zdroje pre prostredie Lazarus

[7] Lazarus - vysokoškolské prednášky na FMFI UK, http://www.pascal.input.sk/
[8] Lazarus wiki, http://wiki.lazarus.freepascal.org/Main_Page/sk
[9] Lazarus Documentation/sk, http://wiki.lazarus.freepascal.org/Lazarus_Documentation/sk
[10] Free pascal, http://www.freepascal.org/

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

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

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

Podnázov Programovanie 7

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti Doc. RNDr. Gabriela Andrejková, CSc.
RNDr. Peter Gurský, PhD.

Počet strán 40

Náklad 300 ks

Prvé vydanie, Bratislava 2009

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

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-8118-009-5