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 8

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 8

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

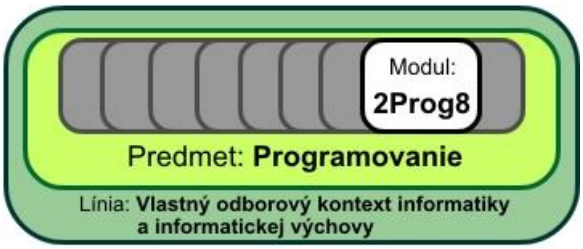
Garant predmetu:

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:

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

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

Abstrakt modulu

Modul sa venuje práci s textovými súbormi. Objasňuje spôsob otvárania a zatvárania súborov, princíp pri zapisovaní a čítaní údajov, testovaním konca riadkov a konca súboru. Predvádza riešenie rôznych úloh pomocou súborov.

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 – **Textové súbory čísel**
   - Vytvárame súbor čísel
   - Čítame súbor čísel
   - Testujeme koniec súboru, koniec riadka
   - Čítame aj zapisujeme
2. tematická jednotka – **Textové súbory znakov**
   - Pracujeme po riadkoch
   - Pracujeme po znakoch
   - Znaky aj čísla v súbore

Cieľ modulu

Po absolvovaní tohto modulu sa od účastníka očakáva, že

- pre danú úlohu bude schopný zadefinovať textový súbor,
- správne zvolí otvorenie súboru buď na čítanie alebo na zápis,
- bude schopný priradiť súborovej premennej vhodný súbor na disku,
- bude schopný zapísať údaje do súboru a potom ich prečítať a spracovať tieto hodnoty,
- bude schopný pre danú úlohu spracovať len časť súboru,
- bude schopný pracovať naraz s viac súbormi.

Vstupné vedomosti

Požadované prerekvizity

Modul Programovanie 7 – štruktúrovaný typ pole.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Predpokladáme, že účastník vzdelávania:

- vie popísať princíp práce s poľom,
- vie popísať princíp indexovania prvkov poľa,
- dokáže analyzovať zadanie a tiež možné chyby v riešení, chyby vie nájsť a opraviť,
- dokáže vytvoriť aplikácie, ktoré využívajú viac polí,
- rozumie rozdielom medzi rôznymi dátovými typmi, vie posúdiť, kedy je vhodné použitie niektorého typu, ich viditeľnosť v kóde.

Preverenie vstupných vedomostí

Aplikácia, ktorá pomocou techniky veľkých čísel prevedie nejaké celé číslo do poľa v zadanej číselnej sústave.

1. tematická jednotka – Textové súbory čísel

Niektoré dátové typy v rôznych programovacích jazykoch majú takú vlastnosť, že svoje hodnoty ukladajú nie v pamäti, ale na nejakom externom médiu - najčastejšie na disku. Funguje to tak, že keď do premennej ukladáme nejaké hodnoty (zapisujeme ich), tie sa automaticky ukladajú do súboru. Keď ich budeme niekedy nabudúce potrebovať získať, prečítajú sa zo súboru.

Premenné typu súbor môžu byť rôznych typov. My sa zoznámime iba s tzv. textovým súborom, ktorý je z nich najjednoduchší. Okrem toho, že sa s ním manipuluje veľmi jednoducho, takéto súbory vieme prezerať a vytvárať aj mimo pascalovského programu. Textové súbory môžeme vytvárať, prezerať a aj upravovať v ľubovoľnom textovom editore (napr. Poznámkový zošit, Notepad a pod.) ale môžeme ich otvoriť aj vo vývojovom prostredí, napr. Lazarus (môžeme ich pretiahnuť do editovacieho okna Lazarusa). Textový súbor na disku (alebo USB kľúči, resp. inom zariadení) má väčšinou príponu **.txt**.

1. Vytvárame súbor čísel

Naučíme sa vytvárať textový súbor, do ktorého zapíšeme nejakú postupnosť čísel. Na prácu so súborom musíme najprv zadeklarovať premennú, pomocou ktorej sa budeme na tento súbor odvolávať. V deklaračnej časti zapíšeme napr.

```
var
  Subor: TextFile;
```

Tým sme si pripravili premennú, ktorá bude slúžiť na ukladanie a vyberanie nejakých hodnôt do, resp. zo súboru.

Do tela programu budeme musieť teraz zapísať príkazy, ktorými najprv priradíme premennej typu **TextFile** konkrétny súbor na disku, aby sme potom mohli zapisovať nejaké hodnoty.

```
AssignFile(premenná, meno súboru);
```

Volaním tejto štandardnej procedúry sa zatiaľ iba zapamätá, že súborová premenná bude asi pracovať s uvedeným súborom. Meno súboru zadávame znakovým reťazcom, buď ako absolútnu cestu (napr. '**c:\udaje\mojsubor.txt**'), alebo ako relatívnu cestu (napr. '**mojsubor.txt**') pre súbor v tom istom priečinku, ako sa nachádza samotná aplikácia. Konkrétne zapíšeme, napríklad

```
AssignFile(Subor, 'mojsubor.txt');
```

Až ďalší príkaz za **AssignFile** prácu s fyzickým súborom naozaj naštartuje. Keďže sa najprv naučíme súbor vytvárať, tak tým príkazom bude

```
Rewrite(Subor);
```

Týmto volaním sa uvedený súbor **otvoril na zápis**. Znamená to, že súbor sa najprv na disku vytvoril ako úplne prázdny (ak už existoval, tak sa vyprázdnil) a nastavil sa režim, v ktorom môžeme do súboru zapisovať nejaké hodnoty. Slovo **otvoriť** pri práci so súbormi vlastne znamená **povoliť**.

Súbor máme otvorený a môžeme do neho zapisovať ľubovoľný počet hodnôt. Na to slúži príkaz

```
WriteLn(premenná, hodnota);
```

Napr. príkazom

```
WriteLn(Subor, 37);
```

zapíšeme do súboru jednu celočíselnú hodnotu **37**. Týchto príkazov zápisu môžeme uviesť aj viac za sebou a vtedy sa uvedené hodnoty budú do súboru zapisovať pod sebou.

Na záver musíme súbor **zatvoriť**, aby sa korektne prerušilo naše napojenie pomocou súborovej premennej a aby operačný systém potom vykonal všetky nevyhnutné záverečné činnosti na disku. Od tohto momentu už nebudeme mať povolenie do neho zapisovať. Súbor zatvárame volaním:

```
CloseFile(Subor);
```

Zapíšme teraz celú postupnosť týchto príkazov do jedného programu. Aplikácia po zatlačení tlačidla najprv otvorí textový súbor, zapíše do neho niekoľko čísel, zatvorí ho a na záver do textovej plochy (**Memo1**) zapíše správu, že skončil:

```
procedure Form1.Button1Click(Sender: TObject);
var
  Subor: TextFile;
begin
  AssignFile(Subor, 'cisla.txt');
  Rewrite(Subor);
  WriteLn(Subor, 37);
  WriteLn(Subor, 2 * 3 * 4 * 5);
  WriteLn(Subor, 000);
  CloseFile(Subor);
  Memo1.Lines.Append('hotovo');
end;
```

Po spustení sa najprv vytvorí textový súbor s menom **cisla.txt** a potom sa do textovej plochy vypíše správa **hotovo**. V programe je práca s týmto súborom realizovaná pomocou premennej **Subor**. Nájdime tento súbor na disku a otvorme ho napríklad v programe **Notepad**. Môžeme vidieť:

```
37
120
0
```

Výrazy, ktoré sme zapisovali do súboru, sa najprv vyhodnotili a potom zapísali do riadkov pod sebou. Uvedomte si, že čísla sú v textovom súbore zapísané ako znakové reťazce, teda sa príkazom **WriteLn** prekonvertovali na čísla.

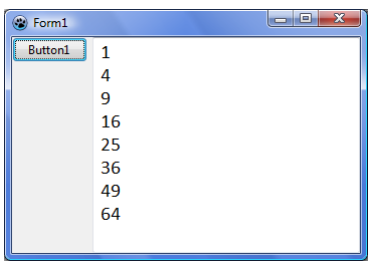
V ďalšom príklade ukážeme využitie for-cyklu a tiež použijeme nový príkaz, ktorým vypíšeme obsah ľubovoľného textového súboru do textovej plochy. Je to príkaz

```
Memo1.Lines.LoadFromFile(meno súboru);
```

Program do súboru zapíše 8 čísel (druhých mocnín čísel od 1 do 8) a potom obsah tohto súboru pre kontrolu vypíše do textovej plochy:

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'cisla8.txt');
  Rewrite(Subor);
  for I := 1 to 8 do
    WriteLn(Subor, I * I);
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('cisla8.txt');
end;
```

Po spustení vidíme, že súbor obsahuje 8 riadkov a v každom je jedno číslo:



Do jedného riadka súboru môžeme zapísať aj viac čísel, ale vtedy ich musíme navzájom oddeliť znakom medzera. Do príkazu **WriteLn** môžeme zadať aj viac hodnôt. Vtedy sa zapíšu do jedného riadka tesne vedľa seba. Napríklad, ak by sme zadali tieto dva príkazy

```
WriteLn(Subor, 10, 12);
WriteLn(Subor, 10, ' ', 12);
```

tak sa do súboru zapíšu tieto dva riadky:

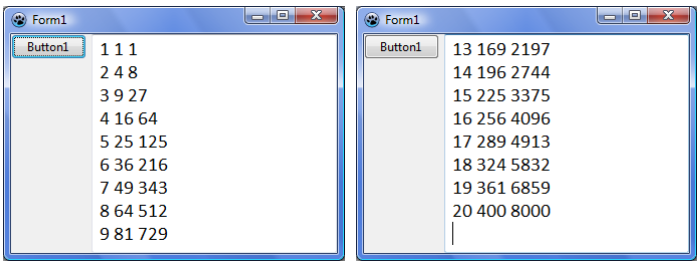
```
1012
10 12
```

V prvom riadku je jedno štvorciferné číslo 1012 a v druhom dve čísla oddelené medzerou. Neskôr, keď budeme zo súboru tieto hodnoty čítať do premenných, z prvého riadka sa prečíta len jedno číslo, pričom sme pravdepodobne chceli zapísať dve samostatné čísla.

V nasledujúcom programe vytvoríme súbor, ktorý bude obsahovať 20 riadkov. V každom zapísané bude postupne číslo od 1 do 20, za ním jeho druhá mocnina a za ním tretia mocnina:

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'cisla20.txt');
  Rewrite(Subor);
  for I := 1 to 20 do
    WriteLn(Subor, I, ' ', I * I, ' ', I * I * I);
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('cisla20.txt');
end;
```

Po spustení pravdepodobne v textovej ploche vidíme vypísaných len niekoľko prvých riadkov súboru. Keď teraz klikneme do textovej plochy myšou a šípkami vyrolujeme jej obsah, tak môžeme vidieť aj nižšie riadky:



Rovnaký obsah by sme videli, keby sme si tento súbor **cisla20.txt** pozreli, napr. v programe **Notepad**.

Už vieme ukladať do súboru čísla do jedného stĺpca, ale aj niekoľko čísel do jedného riadka. Ešte sa naučíme, ako ukladať viac čísel do jedného riadka pomocou for-cyklu. Budeme riešiť takúto úlohu: do jedného riadka zapíšeme 10 nasledujúcich čísel počnúc hodnotou I až po hodnotu I+9. Môžeme to zapísať napríklad takto:

```
WriteLn(Subor, I, ' ', I+1, ' ', I+2, ' ', I+3, ' ', I+4, ' ',
        I+5, ' ', I+6, ' ', I+7, ' ', I+8, ' ', I+9, ' ', I+10);
```

Lenže toto je veľmi nepraktický zápis. Aby sme toto mohli zapísať for-cyklom, musíme pochopiť, ako funguje **WriteLn** a ako ho môžeme rozdeliť na menšie časti. Zoberme len jednu časť z tohto dlhého výpisu:

```
WriteLn(Subor, I, ' ', I+1);
```

Tu sa vypíšu len dve čísla, medzi ktorými sa zapíše medzera. Príkaz **WriteLn** tu zapisuje do súboru tri hodnoty (číslo, medzera, číslo) a ešte k tomu na záver zaznačí, že riadok skončil a ďalšie zápisy do súboru pôjdu už na nový riadok. Toto môžeme prepísať pomocou iného variantu príkazu, ktorý tiež zapisuje do súboru a to príkaz **Write**:

```
Write(Subor, I);
Write(Subor, ' ');
Write(Subor, I+1);
WriteLn(Subor);
```

Príkaz **Write** na rozdiel od **WriteLn** tiež zapíše nejaké hodnoty do súboru, ale tieto hodnoty ostávajú stále v jednom a tom istom riadku. Keď budeme niekedy

potrebovať tento riadok ukončiť, použijeme príkaz **WriteLn** – všimnite si, že tu sme ho použili bez hodnôt, ktoré treba vypísať. Vtedy sa len ukončí riadok a ďalšie výpisy pôjdu na nové riadky.

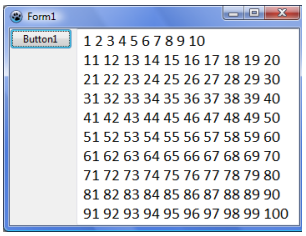
Vytvorme teraz program, ktorý do súboru postupne zapíše 100 čísel. Tieto sa zapíšu tak, že v každom riadku bude 10 čísel. Program sa dá vytvoriť mnohými spôsobmi, napríklad takto:

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'cisla100.txt');
  Rewrite(Subor);
  for I := 1 to 100 do
  begin
    Write(Subor, I, ' ');
    if I mod 10 = 0 then
      WriteLn(Subor);
  end;
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('cisla100.txt');
end;
```

Vždy, keď do súboru zapíšeme nejaké číslo, tak hneď vtedy za neho zapíšeme aj jednu medzeru. Keď to spustíte, zistíte, že táto medzera je aj za posledným číslom v každom riadku. Niekedy to takto nevadí, ale v niektorých situáciách to môže narobiť problémy. Druhý variant tohto programu korektne rieši aj zbytočné medzery na konci riadkov:

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'cisla100.txt');
  Rewrite(Subor);
  for I := 1 to 100 do
  begin
    Write(Subor, I);
    if I mod 10 = 0 then
      WriteLn(Subor)
    else
      Write(Subor, ' ');
  end;
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('cisla100.txt');
end;
```

Po spustení dostávame:



Za každé číslo tu zapíšeme buď medzeru (ak to nie je posledné číslo v riadku), alebo prejdeme na nový riadok pomocou **WriteLn(Subor)**, ak to bolo posledné číslo v riadku.

Okrem vypisovanej hodnoty môžeme ešte určiť aj formátovanie výpisu. Formátovanie znamená, že špecifikujeme, na akú šírku sa má zadaná hodnota zapísať. Formátovanie zapisujeme za znak dvojbodka. Pozrime to na niekoľkých príkladoch:

```
WriteLn(Subor, I:5);
```

označuje, že hodnota premennej **I** sa zapíše na šírku piatich znakov. Ak má toto číslo menej ako päť cifier, tak sa zľava doplní medzerami. Ak je číslo viac ako päťciferné, tak tento **formátovací parameter** sa ignoruje a vypíše sa kompletná hodnota čísla aj bez medzier. Napr. príkaz **WriteLn(Subor, I:5);** pre rôzne **I** do súboru zapíše:

```
pre I = 123          zapíše '  123'
pre I = 7            zapíše '    7'
pre I = 65535        zapíše '65535'
pre I = 1234567      zapíše '1234567'
```

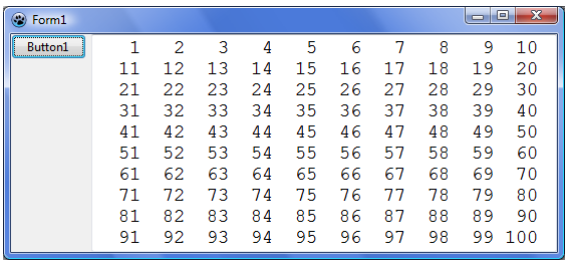
Doteraz sme do súboru zapisovali čísla bez formátovacieho parametra. Môžeme si to

predstaviť tak, že vtedy je formátovacím parametrom 0 a preto, keďže sa zrejme na túto šírku výpis čísla nezmestí, vypíšu sa všetky jeho cifry. Formátovací parameter teda určuje rezervované miesto pre výpis čísla, ktorý bude zarovnaný doprava. Ak to miesto nepostačuje, vypisuje sa aj za toto rezervované miesto.

Úloha, v ktorej sme vytvárali 10-riadkový súbor s číslami od 1 do 100, môže s formátovacím parametrom vyzerať, napr. takto:

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'cisla100.txt');
  Rewrite(Subor);
  for I := 1 to 100 do
  begin
    Write(Subor, I:4);
    if I mod 10 = 0 then
      WriteLn(Subor);
  end;
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('cisla100.txt');
end;
```

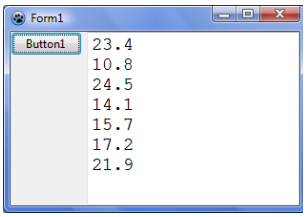
a po spustení:



Formátovací parameter môžeme použiť nielen pri zápise celých čísel, ale aj pri zápise znakov, znakových reťazcov a aj reálnych čísel. Pre reálne čísla je to dokonca odporúčané, keďže bez formátovacieho parametra sa reálne čísla zapisujú v nie najkrajšom tvare (semilogaritmický tvar). Reálne čísla môžu mať aj druhý formátovací parameter, ktorým určujeme počet zapisovaných desatinných miest. Ak máme reálnu premennú **R** s hodnotou 3.14, môžeme vidieť rôzne zápisy:

```
Writeln(Subor, R);      zapíše '3.14000000000000E+000'
Writeln(Subor, R:7);    zapíše ' 3.1E+000'
Writeln(Subor, R:7:3);  zapíše '  3.140'
Writeln(Subor, R:7:1);  zapíše '    3.1'
Writeln(Subor, R:0:2);  zapíše '3.14'
```

Po spustení dostaneme napríklad takéto náhodné hodnoty:



Nasledujúci program ilustruje zápis reálnych čísel do súboru. Program vytvorí súbor **teploty.txt**, ktorý bude obsahovať 7 nameraných hodnôt. Budú to reálne čísla, ktoré vygenerujeme náhodne od 10 do 25 na jedno desatinné miesto.

```
var
  Subor: TextFile;
  I: Integer;
begin
  AssignFile(Subor, 'teploty.txt');
  Rewrite(Subor);
  for I := 1 to 7 do
    WriteLn(Subor, 10+Random(151)/10:0:1);
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('teploty.txt');
end;
```

Čo sme sa naučili

Textový súbor môže obsahovať viac riadkov a v každom môže byť jedno alebo viac čísel oddelených medzerou.

Keď chceme vytvoriť textový súbor, musíme:

- zadeklarovať premennú typu **TextFile**,
- potom musíme priradiť nejaké konkrétne meno súboru pomocou **AssignFile**,
- príkazom **Rewrite** otvoríme tento súbor na zápis,
- príkazy **Write** a **WriteLn** slúžia na zápis hodnôt do súboru,
- čísla v jednom riadku musíme oddeľovať aspoň jednou medzerou,
- zapisované hodnoty môžeme formátovať pomocou formátovacích parametrov.

Úlohy na precvičenie

Zadanie 1	Vytvorte textový súbor **trojuholnik.txt**, ktorý bude obsahovať 20 riadkov: v prvom bude číslo 1, v druhom dve čísla 1 a 2, v treťom tri čísla 1, 2 a 3, atď. až v poslednom v 20 riadku budú čísla od 1 do 20.
Zadanie 2	Program vytvorí textový súbor **domy.txt**, ktorý bude obsahovať 200 náhodných celých čísel z intervalu 0 až 8. Sú to počty obyvateľov v 200 domoch nejakej obce.
Zadanie 3	Vygenerujte súbor **obchod**, ktorý bude obsahovať 50 náhodných celých čísel z intervalu 1 až 100. Sú to nákupy zákazníkov, ktorí vychádzajú z obchodu a my zaznačujeme zaplatenú sumu do súboru.
Zadanie 4	Predchádzajúce zadanie pozmeňte tak, že prvé číslo v súbore **obchod.txt** obsahuje počet zákazníkov (náhodné číslo od 10 do 80) a za tým v súbore nasleduje príslušný počet ich platieb (náhodné čísla od 1 do 100).
Zadanie 5	Napíšte program, ktorý vygeneruje textový súbor **teploty.txt**. Súbor bude obsahovať 7 riadkov pre 7 sledovaných dní. V každom riadku bude náhodný počet nameraných hodnôt (počet od 1 do 5). Teploty budú reálne čísla, ktoré vygenerujete náhodne od 10 do 25 na jedno desatinné miesto.

2. Čítame súbor čísel

V predchádzajúcej časti sme sa naučili vytvárať textové súbory, ktoré sa mohli skladať z viacerých riadkov a každý mohol obsahovať jedno alebo viac čísel. Teraz si ukážeme, ako budeme takéto súbory čítať a ďalej spracovávať. Môžeme napríklad takéto zo súboru prečítané čísla spočítavať, môžeme ich ukladať do poľa, môžeme z nich vytvárať nejakú kresbu v grafickej ploche a pod.

Aby sme mohli z textového súboru čítať, musíme ho otvoriť podobne ako pri zápise. Samozrejme, že nemôžeme použiť volanie **Rewrite**, lebo tento nenávratne vymaže obsah súboru, aby ho pripravil na zápis. Hovoríme, že **Rewrite** urobí otvorenie súboru na zápis. Otvorenie na čítanie robí iný príkaz: volanie **Reset** nastaví súbor na čítanie.

V prvom rade, aby sa dalo zo súboru čítať (dal sa otvoriť na čítanie), súbor musí na disku už existovať, inak program spadne s chybovou správou. Po druhé, keď budeme zostavovať program na čítanie textového súboru, tento by mal byť pripravený korektne. Preto, ak má obsahovať nejakých 20 celých čísel, tak ich tam nesmie byť menej a samozrejme, že v súbore nesmú byť žiadne nevhodné znaky okrem čísel a prípadných medzier medzi číslami v riadkoch. V tejto časti sa naučíme čítať len korektne zadané textové súbory. Neskôr sa naučíme ošetrovať aj prípady, keď bude treba textový súbor nejako zanalyzovať, aby sme z neho mohli "vytiahnuť" nejaké čísla.

Často program, ktorý číta textový súbor, vyzerá približne takto:

```
var
  Subor: TextFile;

begin
  AssignFile(Subor, meno súboru);
  Reset(Subor);

  // čítanie zo súboru

  CloseFile(Subor);
end;
```

Takže najprv otvoríme súbor na čítanie (volanie **Reset**), potom z neho prečítame očakávané čísla a na koniec súbor zatvoríme (volaním **CloseFile**). Opäť existujú dva varianty príkazov na čítanie zo súboru. Začneme príkazom **Read**. Príkaz má tvar

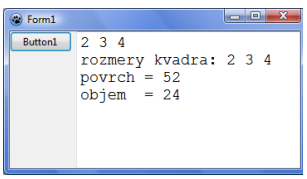
```
Read(Subor, premenná);
```

Parametre má analogické príkazu **Write**: prvým parametrom musí byť súborová premenná, ktorá odkazuje už na otvorený súbor. Druhým parametrom musí byť číselná premenná (**Integer** alebo **Real**, podľa toho, čo sa nachádza v súbore).

Náš prvý program prečíta z textového súboru tri celé čísla - predpokladáme, že sú to tri rozmery strán nejakého kvádra. Program potom do textovej plochy vypíše obsah plášťa a objem kvádra:

```
var
  Subor: TextFile;
  A, B, C: Integer;
begin
  Memo1.Lines.LoadFromFile('kvader.txt');
  AssignFile(Subor, 'kvader.txt');
  Reset(Subor);
  Read(Subor, A, B, C);
  Memo1.Lines.Append('rozmery kvadra: ' + IntToStr(A) + ' ' +
                     IntToStr(B) + ' ' + IntToStr(C));
  Memo1.Lines.Append('povrch = ' + IntToStr(2*(A*B+A*C+B*C)));
  Memo1.Lines.Append('objem  = ' + IntToStr(A*B*C));
  CloseFile(Subor);
end;
```

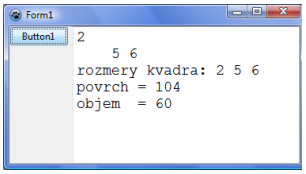
Keď program spustíme s testovacím súborom **kvared.txt**:



Pripravíme si textový súbor **kvader.txt**, do ktorého zapíšeme nejaké tri čísla, napr. čísla 2, 3 a 4.

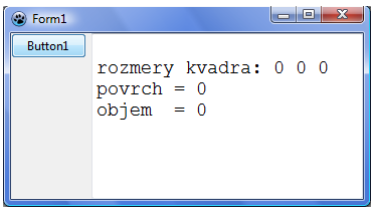
Všimnite si, že program na začiatku vypísal obsah súboru, z ktorého potom číta 3 čísla.

Ak teraz zmeníme obsah súboru **kvader.txt**, a znovu zatlačíme tlačidlo **Button1**, program prečíta nové tri hodnoty a znovu vypíše súbor aj výpočet.



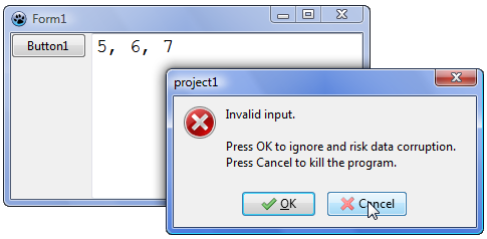
V každej z týchto ukážok môžeme vidieť, že pri čítaní celých čísel vôbec nezáleží, ako sú tieto čísla naformátované.

Zadajme teraz súbor, ktorý obsahuje menej celých čísel ako 3, napr. súbor s jedným prázdnym riadkom:



Dozvedáme sa dôležitú vlastnosť: ak je v súbore menej čísel, ako sa pokúšame čítať, príkaz **Read** do týchto premenných priradí 0.

Ďalší náš pokus, ale spôsobí spadnutie programu. Ak zadáme 3 čísla, ktoré oddelíme napr. čiarkami, program spadne so správou o chybe:

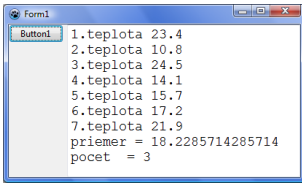


Častejšie budú naše programy čítať viac údajov a my budeme preto potrebovať nejaký cyklus.

Poďme riešiť takúto úlohu: v súbore **teploty.txt** máme 7 nameraných teplôt za celý týždeň. Našou úlohou bude zistiť, aká bola priemerná teplota a koľko dní bolo teplejšie ako tento priemer. Pozrime naše prvé riešenie:

```
var
  Subor: TextFile;
  I, Pocet: Integer;
  Cislo, Sucet, Priemer: Real;
begin
  AssignFile(Subor, 'teploty.txt');
  Reset(Subor);
  Sucet := 0;
  for I := 1 to 7 do
  begin
    Read(Subor, Cislo);
    Memo1.Lines.Append(IntToStr(I) + '.teplota ' + FloatToStr(Cislo));
    Sucet := Sucet + Cislo;
  end;
  Priemer := Sucet / 7;
  Memo1.Lines.Append('priemer = ' + FloatToStr(Priemer));
  Reset(Subor);
  Pocet := 0;
  for I := 1 to 7 do
  begin
    Read(Subor, Cislo);
    if Cislo > Priemer then
      Inc(Pocet);
  end;
  Memo1.Lines.Append('pocet  = ' + IntToStr(Pocet));
  CloseFile(Subor);
end;
```

Po spustení dostávame (použili sme súbor, ktorý sme vygenerovali v predchádzajúcej časti):



Vidíme, že program najprv prečítal všetky hodnoty, vypísal ich a pritom počítal ich súčet. Po vypočítaní priemeru sme opäť otvorili súbor na čítanie (volaním **Reset(Subor)**), aby sme mohli pôvodné čísla zo súboru opäť čítať a porovnávať s priemerom. Vidíme, že súbor môžeme v programe čítať aj viackrát. Stačí ho len znovu otvoriť.

Druhé riešenie tejto istej úlohy využije sedem prvkové pole **Teplota**. Vďaka tomu nemusíme súbor čítať znovu, lebo všetky prečítané hodnoty už máme uložené v pamäti:

```
var
  Subor: TextFile;
  Teplota: array [1..7] of Real;
  I, Pocet: Integer;
  Sucet, Priemer: Real;
begin
  AssignFile(Subor, 'teploty.txt');
  Reset(Subor);
  Sucet := 0;
  for I := 1 to 7 do
  begin
    Read(Subor, Teplota[I]);
    Memo1.Lines.Append(IntToStr(I) + '.teplota ' +
                       FloatToStr(Teplota[I]));
    Sucet := Sucet + Teplota[I];
  end;
  Priemer := Sucet / 7;
  Memo1.Lines.Append('priemer = ' + FloatToStr(Priemer));
  Pocet := 0;
  for I := 1 to 7 do
    if Teplota[I] > Priemer then
      Inc(Pocet);
  Memo1.Lines.Append('pocet  = ' + IntToStr(Pocet));
  CloseFile(Subor);
end;
```

Porovnajte tento program s predchádzajúcim riešením. Všimnite si volanie

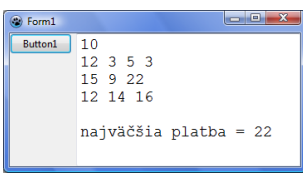
```
Read(Subor, Teplota[I]);
```

ktorým prečítame číslo zo súboru priamo do prvku poľa. Tento program dáva úplne rovnaký výsledok ako predchádzajúce riešenie.

Ďalšia ukážka predvedie, ako spracujeme textový súbor, ktorý na začiatku obsahuje počet hodnôt, ktoré ďalej v tomto súbore nasledujú. Predpokladajme, že máme súbor **obchod.txt**, v ktorom je na začiatku zapísaný počet zákazníkov, ktorí tu v jeden deň nakupovali. Za tým nasleduje príslušný počet celých čísel - súm, ktoré zaplatili za nákupy. Program otvorí tento súbor a zistí, aká suma peňazí bola v ten deň najvyššia:

```
var
  Subor: TextFile;
  I, Platba, Pocet, Max: Integer;
begin
  Memo1.Lines.LoadFromFile('obchod.txt');
  AssignFile(Subor, 'obchod.txt');
  Reset(Subor);
  Read(Subor, Pocet);
  Max := 0;
  for I := 1 to Pocet do
  begin
    Read(Subor, Platba);
    if Platba > Max then
      Max := Platba;
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('najväčšia platba = ' + IntToStr(Max));
end;
```

Po spustení by to mohlo vyzerať, napr. takto:



Program najprv prečíta počet údajov a až potom číta samotné údaje, ktoré ďalej

nasledujú. Vytvorte testovací súbor a otestujte takýto spôsob čítania údajov.

Väčšie množstvo údajov v súbore môžeme organizovať aj inak. Postupnosť hodnôt môže byť ukončená špeciálnou hodnotou, ktorou vieme odlíšiť koniec tejto postupnosti, napr. postupnosť nezáporných čísel ukončíme -1.

Nasledujúci program bude riešiť hlasovanie v súťaži o Superstar. V textovom súbore **hlasovanie.txt** sa nachádzajú hlasy divákov pre 12 súťažiacich, t.j. čísla z intervalu od 1 do 12. Tento zoznam čísel je ukončený hodnotou -1 (niekedy tomu hovoríme zarážka). Program nájde súťažiaceho, ktorý dostal najmenej hlasov:

```
var
  Subor: TextFile;
  Hlas: array [1..12] of Integer;
  I, Cislo, Min: Integer;
begin
  for I := 1 to 12 do
    Hlas[I] := 0;
  AssignFile(Subor, 'hlasovanie.txt');
  Reset(Subor);
  Read(Subor, Cislo);
  while Cislo <> -1 do
  begin
    Inc(Hlas[Cislo]);
    Read(Subor, Cislo);
  end;
  CloseFile(Subor);
  Min := 1;
  for I := 2 to 12 do
    if Hlas[I] < Hlas[Min] then
      Min := I;
  Memo1.Lines.Append('najmenej hlasov dostal súťažiaci s číslom ' +
                     IntToStr(Min));
end;
```

Pri takto organizovanom súbore sme najprv prečítali prvú hodnotu zo súboru a pomocou while-cyklu sme túto hodnotu spracovali a prečítali sme ďalšiu. Takto postupne prečítame a spracujeme všetky hodnoty v súbore, až kým pri čítaní nenarazíme na hodnotu -1. Otestujte program nejakým vhodne pripraveným súborom.

Zamyslite sa, čo sa stane, ak vstupný súbor **hlasovanie.txt** nebude obsahovať koncovú hodnotu -1. Program spadne? Alebo naopak, zacyklí sa?

Čo sme sa naučili

Aby sme mohli čítať údaje z textového súboru, musíme najprv tento otvoriť na čítanie pomocou volania **Reset**. Samotné údaje čítame pomocou príkazu **Read**. Číselné hodnoty čítame do premenných zodpovedajúcich typov. Pritom je dôležité, aby

- čísla boli v samostatných riadkoch,
- alebo boli navzájom oddelené aspoň jednou medzerou,
- a zároveň v súbore neboli žiadne neprípustné znaky, napr. čiarky medzi číslami.

Navyše medzery pred číslami a aj prázdne riadky pred číslami sa pri čítaní čísel ignorujú.

Ak súbor obsahuje väčšie množstvo čísel, tak

- buď vieme ich presný počet,
- ich počet je zapísaný na začiatku súboru,
- alebo takáto postupnosť čísel je ukončená nejakou konkrétnou číselnou hodnotou, ktorá už do postupnosti nepatrí.

Úlohy na precvičenie

Zadanie 1	Riešili sme úlohu o platiacich zákazníkoch, kde sme v súbore **obchod.txt** získali ich zaplatené sumy. Prerobte program tak, aby mohol spracovať evidenciu nielen pre jeden deň, ale aj pre viac dní. Napr. pre dva dni by súbor obsahoval dva bloky informácií: blok začína počtom zákazníkov, za tým nasledujú ich platby a za tým by hneď nasledoval druhý blok údajov o druhom dni. Aj tento by začínal počtom zákazníkov a za tým by boli ich platby. Program spracuje oba dni a pre každý vypíše maximálnu platbu.
Zadanie 2	Riešili sme úlohu o SMS hlasovaní v súťaži o Superstar. V súbore hlasovanie.txt boli hlasy za jedno kolo súťaže. Doplňte program tak, aby spracoval aj druhý deň, t.j. v súbore za údajmi pre prvé kolo, ktoré sú ukončené hodnotou -1, nasledujú informácie o druhom kole hlasovania. Aj tieto sú ukončené hodnotou -1. Program zistí ako dopadlo druhé kolo, ak vypadnutý súťažiaci v prvom kole sa už v druhom nezúčastňuje (jeho hlasy v druhom kole nemajú žiaden vplyv).
Zadanie 3	Detektívna kancelária si vyžiadala od vstupnej kontroly do firmy zoznam všetkých návštevníkov pre dva konkrétne sledované dni. Potrebovala zistiť, ktorí návštevníci boli vo firme v oba sledované dni. Dostali sme k dispozícii textový súbor, ktorý obsahuje čísla preukazov všetkých návštevníkov pre každý z dní – oba zoznamy sú v jednom súbore **navstevy.txt**. Každý z týchto zoznamov je ukončený hodnotou -1. Napíšte program, ktorý vypíše všetky čísla, ktoré sa nachádzajú v oboch zoznamoch.

3. Testujeme koniec súboru, koniec riadka

V predchádzajúcej časti sme videli, že pri čítaní súborov by sme mali, buď presne vedieť, koľko čísel budeme čítať, alebo by mala byť v súbore zapísaná nejaká hodnota (zarážka), ktorá označuje koniec čítanej postupnosti čísel.

Tiež už vieme aj to, že ak sme zo súboru prečítali všetky čísla, tak ďalšie čítanie neexistujúcich hodnôt v súbore prebehne bez upozornenia a do čítanej premennej sa priradí hodnota 0.

Našťastie máme v Pascale dve pomocné logické funkcie, ktoré nám o otvorenom súbore prezradia, či v súbore, resp. v momentálnom riadku máme ešte čo čítať, alebo sme už na konci.

Začneme funkciou **SeekEof** (kde **Eof** je z anglického "end of file", t.j. "koniec súbor"). Táto funkcia nám odpovie, či sme už na konci súboru a teda by sme už žiadne čísla nemali čítať. Pritom sa pozrie, či od momentálneho miesta až do konca súboru nasledujú len prázdne riadky. Vtedy vráti **True**, alebo vráti **False**, ak ešte nie sme na konci súboru a teda môžeme ešte ďalej zo súboru čítať.

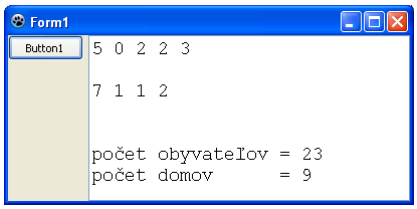
Štandardná logická funkcia **SeekEof** funguje v prostredí Delphi trochu rozdielne.

Použitie **SeekEof** ukážeme na takomto programe: v súbore **domy.txt** máme uložené informácie o počte obyvateľov jednotlivých domov v nejakej obci. Dopredu nevieme koľko je v obci domov. Úlohou je zistiť, koľko obyvateľov žije v obci a koľko je v nej domov.

Program otvorí súbor na čítanie a potom, kým nie sme ešte na konci súboru (kým môžeme ešte čítať), prečítame jednu hodnotu a spracujeme ju (pripočítame ju k počtu obyvateľov a zvýšime počet domov) :

```
var
  Subor: TextFile;
  Cislo, PocetObyvatelov, PocetDomov: Integer;
begin
  Memo1.Lines.LoadFromFile('domy.txt');
  PocetObyvatelov := 0;
  PocetDomov := 0;
  AssignFile(Subor, 'domy.txt');
  Reset(Subor);
  while not SeekEof(Subor) do
  begin
    Read(Subor, Cislo);
    PocetObyvatelov := PocetObyvatelov + Cislo;
    Inc(PocetDomov);
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('počet obyvateľov = '+IntToStr(PocetObyvatelov));
  Memo1.Lines.Append('počet domov      = ' + IntToStr(PocetDomov));
end;
```

Môžeme vidieť, že funkcia **SeekEof** má jeden parameter a to je súborová premenná. Neskôr budeme pracovať s viacerými súbormi súčasne a vtedy sa tu bude rozlišovať, o ktorom súbore chceme vedieť, či sme už na jeho konci. Keďže cyklus **while** pracuje dovtedy kým je splnená nejaká podmienka, museli sme znegovať výsledok **SeekEof**, lebo my chceme pokračovať, kým ešte nie sme na konci. Vytvorili sme si malý testovací súbor **domy.txt**, ktorý sa na začiatku nášho programu celý vypísal do textovej plochy. Vidíme, že na konci súboru boli prázdne riadky a vtedy sa už určite žiadne čísla neprečítali. Program prečítal len 9 čísel:



Veľmi podobná funkcii **SeekEof** je ďalšia pomocná funkcia **SeekEoln** (z **Eoln** je anglického "end of line", t.j. "koniec riadka"). Táto funkcia vráti **True** vtedy, keď v

momentálne čítanom riadku už nie sú ďalšie hodnoty (možno sú tam už len medzery). Rovnako ako **SeekEof**, aj táto funkcia najprv "preskočí" všetky medzery a ak pritom príde na koniec riadka, vráti **True**. Na rozdiel od **SeekEof**, táto funkcia už nepreskakuje na nový riadok, ale čítanie ostane nastavené na konci tohto riadka.

Okrem týchto dvoch pomocných funkcií sa ešte zoznámime s novým príkazom:

```
ReadLn(Subor);
```

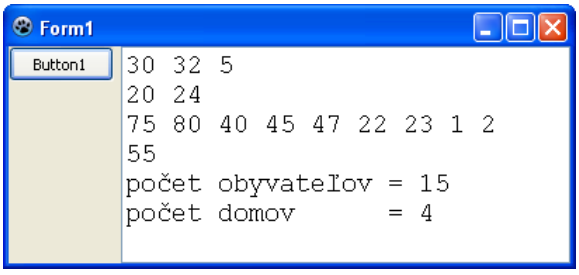
Tento príkaz s jedným parametrom - súborová premenná - preskočí na vstupe zvyšok momentálne čítaného riadka a nastaví ďalšie čítanie na začiatok nasledujúceho riadka. Niekedy sa používa aj vtedy, keď sme pri čítaní nastavení na konci nejakého riadka a my potrebujeme čítať zo začiatku ďalšieho riadka.

Použitie funkcie **SeekEoln** a **ReadLn** ukážeme na takomto príklade: v súbore **domy2.txt** máme pripravené informácie o obyvateľoch nejakej obce. V každom riadku súboru je informácia o obyvateľoch jedného domu: sú to čísla, ktoré označujú vek jeho obyvateľov. Predpokladáme, že žiadny dom nie je prázdny (súbor neobsahuje prázdne riadky). Našou úlohou je zistiť, koľko obyvateľov žije v obci a koľko je v nej domov. :

```
var
  Subor: TextFile;
  Cislo, PocetObyvatelov, PocetDomov: Integer;
begin
  Memo1.Lines.LoadFromFile('domy2.txt');
  PocetObyvatelov := 0;
  PocetDomov := 0;
  AssignFile(Subor, 'domy2.txt');
  Reset(Subor);
  while not SeekEof(Subor) do
  begin
    while not SeekEoln(Subor) do
    begin
      Read(Subor, Cislo);
      PocetObyvatelov := PocetObyvatelov + 1;
    end;
    ReadLn(Subor);
    Inc(PocetDomov);
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('počet obyvateľov = '+IntToStr(PocetObyvatelov));
  Memo1.Lines.Append('počet domov      = ' + IntToStr(PocetDomov));
end;
```

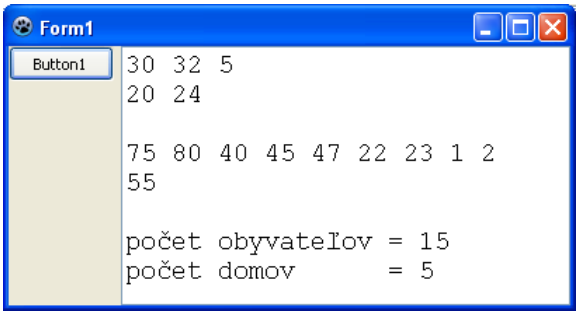
Pozorne si pozrite vnorený while-cyklus. Tento cyklus hovorí presne toto: kým je na momentálnom riadku aspoň jedno číslo, prečítaj ho a zvýš počet obyvateľov obce. Keď tento cyklus skončí (jeden dom sme tým už spracovali), prejdeme na nový riadok príkazom **ReadLn**. Uvedomte si, že ak by sme tento príkaz vynechali (môžete to vyskúšať), program by sa väčšinou zacyklil a teda by nikdy neskončil. Vnorený while-cyklus po prečítaní všetkých údajov pre jeden dom, stále ostáva v tomto jednom riadku a aj nasledovný prechod "veľkým" cyklom stále ostáva v tom istom riadku.

Pripravili sme jednoduchý testovací súbor **domy2.txt** a program sme spustili:



Vidíme, že program pracuje korektne. Doplňme do textového súboru aj dva prázdne riadky (jeden v strede a druhý na konci) a zatlačme **Button1** ešte raz. Dostávame

takýto výstup:

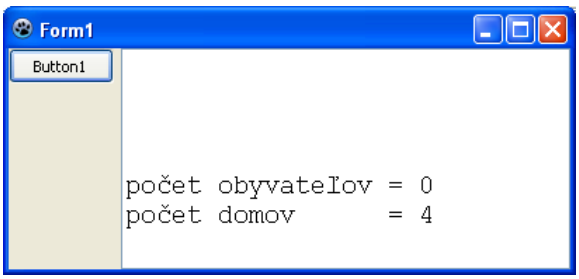


Počet obyvateľov sa nezmenil, ale počet domov sa zvýšil o jedna. Môžeme vidieť, že program pracuje aj pre prázdne domy (pre tretí dom sme zadali prázdny riadok obyvateľov), ale prázdne domy na konci obce (prázdne riadky na konci súboru) sa ignorujú. Spôsobila to funkcia **SeekEof**. O tejto funkcii vieme, že vráti **True** nielen, keď sme v súbore nastavení na úplnom konci, ale aj vtedy, keď v súbore ostali len prázdne riadky.

Tu by sa nám zišla taká funkcia na zisťovanie konca súboru, ktorá nepreskakuje prázdne riadky. Presne na toto slúži logická funkcia **Eof**. Doteraz sme sa ňou nezaoberali, lebo by nám nepomohla pri čítaní čísel na konci súboru. Teraz našu úlohu bez tejto funkcie nevyriešime. V programe nahradíme **SeekEof(Subor)** volaním **Eof(Subor)** a spustíme:

```
while not Eof(Subor) do
```

Teraz môžeme program otestovať aj na obci, v ktorej sú len 4 prázdne domy:

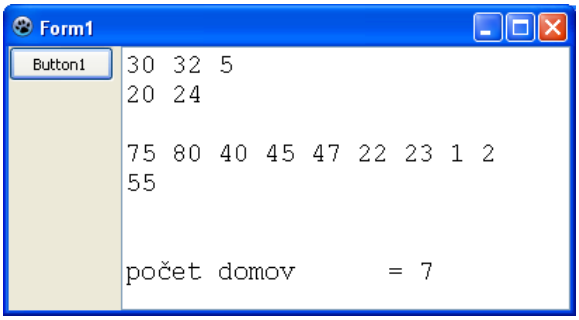


Ukážme ešte riešenie trochu zjednodušenej úlohy s textovým súborom **domy2.txt**. Predpokladajme, že nepotrebujeme vedieť počet obyvateľov obce, ale len počet domov. Potom môžeme program zapísať aj takto:

```
var
  Subor: TextFile;
  PocetDomov: Integer;
begin
  Memo1.Lines.LoadFromFile('domy2.txt');
  PocetDomov := 0;
  AssignFile(Subor, 'domy2.txt');
  Reset(Subor);
  while not Eof(Subor) do
  begin
    ReadLn(Subor);
    Inc(PocetDomov);
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('počet domov      = ' + IntToStr(PocetDomov));
end;
```

Program číta celý súbor len pomocou príkazu **ReadLn(Subor)**, ktorý preskočí celý obsah riadku a nastaví sa na začiatok nasledovného.

Po spustení vidíme:



Program spočítal počet domov korektne.

Čo sme sa naučili

Pri čítaní z textového súboru môžeme používať tri logické funkcie:

- **SeekEof** vráti **True** vtedy, keď sme v súbore nastavení za posledným číslom, t.j. už sme na jeho konci,
- **Eof** vráti **True** len vtedy, keď sme v súbore nastavení na konci posledného riadka, t.j. už sme na jeho konci,
- **SeekEoln** vráti **True** vtedy, keď sme v riadku nastavení za posledným číslom.

Pri čítaní zo súboru, môžeme použiť aj príkaz **ReadLn**, pomocou ktorého preskočíme zvyšok momentálneho riadka a nastavíme čítanie na začiatok ďalšieho riadka.

Úlohy na precvičenie

Zadanie 1	Program prečíta súbor **domy.txt** – počty obyvateľov v jednotlivých domoch obce a zistí, v ktorom dome (jeho poradové číslo) býva najviac obyvateľov. Ak je takýchto domov viac, vypíše ich všetky.
Zadanie 2	Program prečíta súbor **domy2.txt** – vek obyvateľov v jednotlivých domoch obce (v každom riadku sú obyvatelia jedného domu) a zistí, v ktorom dome (jeho poradové číslo) býva najstarší obyvateľ. Ak je takýchto obyvateľov s rovnakým najstarším vekom viac, vypíše ich všetky.
Zadanie 3	V súbore **test.txt** sú zapísané výsledky žiakov z nejakého testu. V každom riadku je bodové ohodnotenie jedného žiaka za jeden príklad, pričom tu môže byť aj viac čísel, ak žiak stihol vypočítať viac príkladov. Zistite, koľko maximálne bodov získal najlepší žiak (teda súčet bodov za príklady, ktoré vyriešil) a koľký to bol žiak v poradí všetkých žiakov v súbore.
Zadanie 4	V súbore **body.txt** sú uložené súradnice bodov v grafickej ploche, ktoré sa tam ukladali nejakým programom počas ťahania myšou (v udalosti **onMouseMove**). Súbor obsahuje viac takýchto kresieb (t.j. postupností súradníc bodov), pričom každá z nich je v samostatnom riadku súboru. Napíšte program, ktorý nakreslí obrazce zadané v tomto súbore (pospája za sebou nasledujúce body). Prvý bod v každej postupnosti sa nakreslí so zdvihnutým perom.
Zadanie 5	Vytvorte súbor **body.txt** z predchádzajúceho zadania. V udalosti **onMouseDown** grafickej plochy sa začne vytvárať nová postupnosť bodov, v udalosti **onMouseMove** sa pridávajú body do tejto postupnosti.
Zadanie 6	V súbore **sprava.txt** sme dostali od priateľa zakódovanú správu. Kódovanie je veľmi jednoduché: každé písmeno (bez diakritiky) prerobíme na číslo, podľa toho, koľké písmeno je to v abecede (napr. a=1, b=2, ..., z=26), medzery medzi slovami sa prepisujú na číslo 0. Napíšte program, ktorý prečíta takýto súbor a do textovej plochy vypíše jej rozkódovanie.
Zadanie 7	Napíšte program, ktorý so zadaného textu (napr. cez vstupný riadok **Edit1**) vytvorí súbor so zakódovaný textom podľa predchádzajúceho zadania.

4. Čítame aj zapisujeme

V tejto časti sa naučíme riešiť úlohy, v ktorých budeme súbory nielen buď čítať alebo len zapisovať, ale program nejaký súbor prečíta a na jeho základe vytvorí nový súbor.

Vo všeobecnosti môžu nastať dva prípady. V prvom prípade nejaký súbor najprv celý prečítame a až potom začneme vytvárať nejaký nový súbor, resp. prepisovať pôvodný súbor novým obsahom. Niekedy ale budeme potrebovať jeden súbor čítať a zároveň pripravovať nejaký ďalší. Vtedy budeme mať súčasne otvorené dva súbory: jeden bude na čítanie a druhý na zápis.

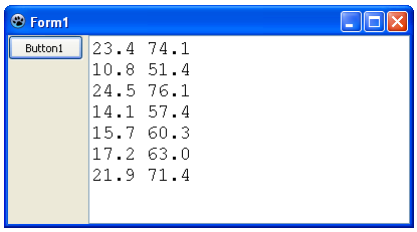
Začnime s prvým prípadom. Najprv nejaký súbor celý prečítame a až potom začneme vytvárať iný. Napíšme program, v ktorom najprv zo súboru **teploty.txt** prečítame 7 nameraných teplôt pre 7 dní v týždni a potom ich zapíšeme do súboru **teplotyF.txt**, ktorý bude obsahovať tieto isté teploty ale aj s pripísanými hodnotami vo Fahrenheitoch. Použijeme vzorec, ktorý sa dá ľahko nájsť aj na internete, napr.

Fahrenheit = Celsius * 1.8 + 32

Program najprv prečítané teploty uloží do poľa a potom ich zapíše, ale s prepočtom, do druhého súboru:

```
var
  Subor: TextFile;
  I: Integer;
  Teplota: array [1..7] of Real;
begin
  AssignFile(Subor, 'teploty.txt');
  Reset(Subor);
  for I := 1 to 7 do
    Read(Subor, Teplota[I]);
  CloseFile(Subor);
  AssignFile(Subor, 'teplotyF.txt');
  Rewrite(Subor);
  for I := 1 to 7 do
    WriteLn(Subor, Teplota[I]:0:1, ' ', Teplota[I]*1.8+32:0:1);
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('teplotyF.txt');
end;
```

Po spustení vidíme v súbore okrem teplôt v stupňoch Celzia, ale aj vo Fahrenheitoch:



Premennú **Subor** sme tu najprv použili pre prečítanie nejakých hodnôt zo súboru a potom po zatvorení súboru, sme ju použili na otvorenie iného súboru na zápis.

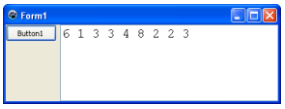
Ďalší program bude pracovať so súborom **domy.txt**, ktorý obsahuje počty obyvateľov v jednotlivých domoch obce. Našou úlohou bude opraviť tento súbor, keďže sme sa dozvedeli, že do každého domu pribudol jeden obyvateľ. Takže čítame aj zapisujeme do toho istého súboru. Budeme predpokladať, že domov v obci nie je viac ako 1000. Program najprv otvorí súbor na čítanie, prečíta z neho všetky hodnoty do poľa **Dom** a zároveň si v premennej **Pocet** uchováva počet týchto čísel. Potom volaním **Rewrite(Subor)** prepne režim práce súboru na zápis a tým sa pôvodný obsah vymaže. Teraz môžeme nové hodnoty zapísať do tohto istého súboru:

```
var
  Subor: TextFile;
  Dom: array [1..1000] of Integer;
  I, Pocet: Integer;
begin
  AssignFile(Subor, 'domy.txt');
  Reset(Subor);
  Pocet := 0;
  while not SeekEof(Subor) do
  begin
    Inc(Pocet);
    Read(Subor, Dom[Pocet]);
  end;
  Rewrite(Subor);
  for I := 1 to Pocet do
    Write(Subor, Dom[I]+1, ' ');
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('domy.txt');
end;
```

Ak predpokladáme, že v súbore pred spustením boli tieto čísla:

```
5 0 2 2 3
7 1 1 2
```

tak po spustení dostávame:



Môžeme vidieť, že to funguje správne.

Všimnite si, že súbor sme po prečítaní nezatvárali, ale iba sme ho otvorili na zápis.

Ďalší program bude trochu náročnejší. Potrebujeme vytvoriť opravenú verziu súboru **domy2.txt**, ktorý obsahuje vek všetkých obyvateľov nejakej obce, pričom v jednom riadku sú obyvatelia jedného domu. Napr. za jeden rok všetci obyvatelia o rok zostarli a my potrebujeme vyrobiť súbor **domy3.txt** s týmito opravenými hodnotami. Nebudeme si najprv všetky hodnoty prečítané zo súboru **domy2.txt** ukladať do nejakého poľa (bolo by to dosť komplikované, lebo si musíme pamäť nielen vek ale aj, v ktorom dome býva). Naprogramujeme to tak, že spolu s prvým súborom otvoríme aj druhý súbor na zápis a do neho budeme každú prečítanú hodnotu kopírovať aj s opravou (zvýšime o 1) :

```
var
  Subor2, Subor3: TextFile;
  Cislo: Integer;
begin
  AssignFile(Subor2, 'domy2.txt');
  Reset(Subor2);
  AssignFile(Subor3, 'domy3.txt');
  Rewrite(Subor3);
  while not Eof(Subor2) do
  begin
    while not SeekEoln(Subor2) do
    begin
      Read(Subor2, Cislo);
      Write(Subor3, Cislo+1, ' ');
    end;
    ReadLn(Subor2);
    WriteLn(Subor3);
  end;
  CloseFile(Subor2);
  CloseFile(Subor3);
  Memo1.Lines.LoadFromFile('domy3.txt');
  Memo1.Lines.Append('*** koniec ***');
end;
```

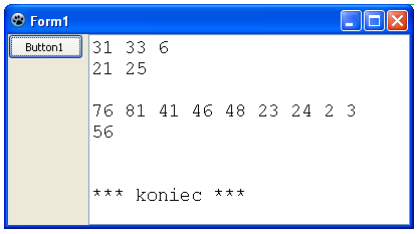
V tomto programe musíme pre oba súbory robiť paralelne veľmi podobné akcie: obom musíme priradiť súbor na disku (**AssignFile**), oba musíme otvoriť (jeden pre čítanie **Reset** a druhý pre zápis **Rewrite**), z jedného čítame (**Read**) a vtedy do druhého zapisujeme (**Write**). V jednom prechádzame na začiatok ďalšieho riadka (**ReadLn**) a vtedy do druhého zapisujeme nový riadok (**WriteLn**). Na záver musíme oba súbory zatvoriť (**CloseFile**).

Ak predpokladáme, že v súbore pred spustením bolo týchto sedem riadkov:

```
30 32 5
20 24

75 80 40 45 47 22 23 1 2
55
```

tak po spustení dostávame správny obsah:



Na záver sme do textovej plochy vypísali koncový riadok, aby boli vidieť aj prázdne riadky na konci súboru.

Čo sme sa naučili

V programoch môžeme pracovať naraz aj s viacerými súbormi. Každý z nich je potom prístupný pomocou svojej súborovej premennej.

Ak v programe pracujeme s nejakým súborom, ktorý zatvoríme (**CloseFile**), môžeme túto premennú použiť na otvorenie iného súboru.

Úlohy na precvičenie

Zadanie 1	Súbor **cisla.txt** obsahuje nejakú postupnosť celých čísel. Napíšte program, ktorý prekopíruje túto postupnosť do súboru **cisla10.txt**, ale tak, že tento nový súbor bude mať v každom riadku (okrem posledného) po 10 čísel.
Zadanie 2	Súbor **cisla.txt** obsahuje nejakú veľkú postupnosť reálnych čísel. Napíšte program, ktorý vytvorí súbor **cisla0.txt**. V tomto novom súbore bude tá istá postupnosť, ale táto nová bude ešte začínať dĺžkou tejto postupnosti. V programe nepoužívajte žiadne pole.
Zadanie 3	Súbor **obchod.txt** obsahuje niekoľko postupností celých nezáporných čísel, z ktorých každá je ukončená číslom -1. Napíšte program, ktorý vytvorí súbor **obchod2.txt** s rovnakým obsahom ako **obchod.txt**, ale v tomto novom súbore bude každá postupnosť v samostatnom riadku (aj s ukončovacím číslom -1).
Zadanie 4	Súbor **domy2.txt** obsahuje informácie o veku obyvateľov nejakej obce: v každom riadku je postupnosť čísel. Niektoré z týchto postupností môžu byť prázdne. Napíšte program, ktorý vytvorí dva súbory **domy18.txt** a **domy0.txt**, v ktorých budú pre každý dom informácie len o dospelých obyvateľoch (v súbore **domy18.txt**) a len o deťoch (súbor **domy0.txt** obsahuje obyvateľov mladších ako 18 rokov).
Zadanie 5	Napíšte program, ktorý z dvoch súborov **domy18.txt** a **domy0.txt** vytvorí súbor **domy2.txt**. Tieto súbory sú popísané v predchádzajúcom zadaní.
Zadanie 6	Okrem súboru **domy2.txt** z predchádzajúcich zadaní máme ešte jeden súbor **prisli.txt**. Tento druhý súbor obsahuje zoznam osôb, ktoré sa prisťahovali do danej obce. Pre každú prisťahovanú osobu súbor obsahuje dvojicu čísel: číslo domu (od 1 do počtu domov) a vek osoby. Napíšte program, ktorý z týchto dvoch súborov vytvorí súbor **domy2novy.txt**. Tento bude obsahovať doplnené informácie o obyvateľoch obce: do správnych riadkov súboru dopíše nových obyvateľov. Mohli by sme zmeniť algoritmus, ak by sme mohli predpokladať, že v súbore **prisli.txt** máme všetky riadky vzostupne usporiadané podľa prvého čísla?

2. tematická jednotka – Textové súbory znakov

Doteraz bol pre nás textový súbor len postupnosťou celých čísel. Táto postupnosť mohla byť na niektorých miestach prerušená novými riadkami. Logické funkcie **SeekEoln**, **SeekEof** a **Eof** nám pomáhali sa rozhodnúť, či sme na konci riadku alebo na konci celého súboru.

V skutočnosti je textový súbor postupnosťou znakov. Môžeme to vidieť aj v ľubovoľnom textovom editore, napr. v programe Notepad. To, že Pascal umožňuje z takéhoto súboru čítať čísla, resp. do súboru zapisovať nejaké čísla, je len malý komfort, ktorý ponúka tento programovací jazyk. Prirodzený spôsob práce so súbormi je práca po znakoch, resp. po riadkoch.

1. Pracujeme po riadkoch

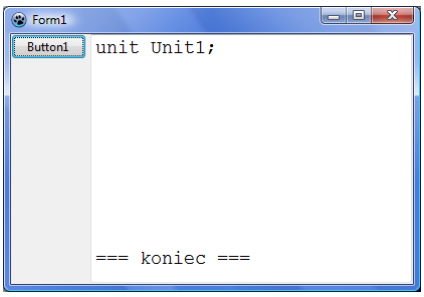
Na textový súbor sa môžeme pozerať ako na postupnosť riadkov, pričom každý riadok obsahuje rôzne dlhý znakový reťazec. Príkaz

```
Read(Subor, Retazec);
```

do premennej **Retazec**, ktorá je typu znakový reťazec, prečíta celý obsah momentálneho riadku. V skutočnosti sa prečítajú znaky od momentálneho miesta v riadku až do konca riadku. Ukážme použitie znakového reťazca na takomto príklade:

```
var
  Subor: TextFile;
  Retazec: string;
  I: Integer;
begin
  AssignFile(Subor, 'unit1.pas');
  Reset(Subor);
  for I := 1 to 10 do
  begin
    Read(Subor, Retazec);
    Memo1.Lines.Append(Retazec);
  end;
  Memo1.Lines.Append('=== koniec ===');
  CloseFile(Subor);
end;
```

Program mal za úlohu vypísať prvých 10 riadkov z nejakého textového súboru. Otvorili sme na čítanie súbor **unit1.pas**, ktorý obsahuje zdrojový kód nášho programu. Toto je totiž tiež textový súbor. Po spustení dostávame takýto výpis:



Program vypísal len prvý riadok súboru a za tým dal 9 prázdnych riadkov napriek tomu, že súbor je v poriadku a sú tam neprázdne riadky.

Chybou v programe je použitie príkazu **Read** - keď čítame znakový reťazec, tak ten sa naozaj prečíta, ale v súbore ostaneme nastavení na konci tohto riadka. Každé ďalšie volanie **Read** číta od momentálnej pozície až do konca radka, t.j. prečíta prázdny reťazec a stále ostáva nastavený koniec toho istého riadka.

Takže chybou v tomto programe je to, že po prečítaní znakového reťazca sa treba nastaviť na začiatok ďalšieho riadka, t.j. treba zavolať **ReadLn**. Musíme teda

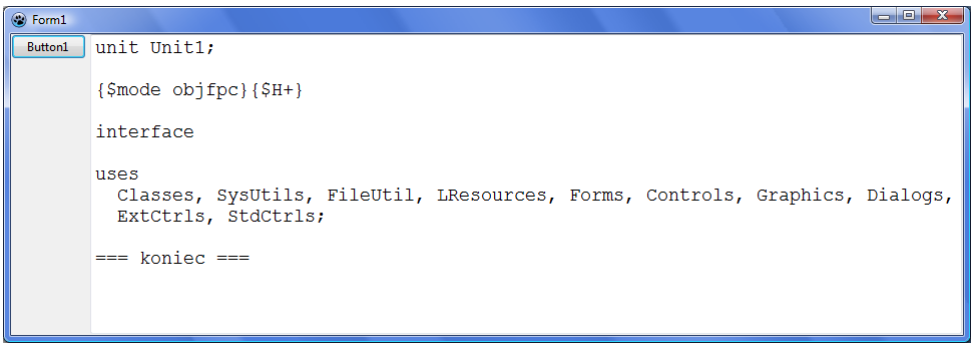
zapísať buď

```
Read(Subor, Retazec);
ReadLn(Subor);
```

alebo to môžeme zapísať dokopy:

```
ReadLn(Subor, Retazec);
```

Ak do príkazu **ReadLn** okrem súborovej premennej uvedieme aj nejaké iné premenné, tak príkaz najprv prečíta hodnoty zo súboru do premenných a potom prejde na ďalší riadok. Program takto opravíme a po spustení už dostaneme:

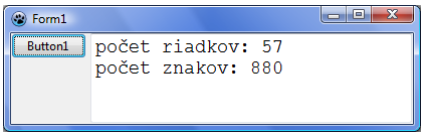


Oknu formulára aj s textovou plochou sme tu trochu zväčšili šírku, aby sa pekne vypísal aj jeden dlhý riadok zo súboru.

Keď už vieme správne prečítať riadky textového súboru, môžeme zistiť nielen počet riadkov súboru ale aj počet všetkých znakov:

```
var
  Subor: TextFile;
  Retazec: string;
  Riadky, Znaky: Integer;
begin
  AssignFile(Subor, 'unit1.pas');
  Reset(Subor);
  Riadky := 0;
  Znaky := 0;
  while not Eof(Subor) do
  begin
    ReadLn(Subor, Retazec);
    Riadky := Riadky + 1;
    Znaky := Znaky + Length(Retazec);
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('počet riadkov: ' + IntToStr(Riadky));
  Memo1.Lines.Append('počet znakov: ' + IntToStr(Znaky));
end;
```

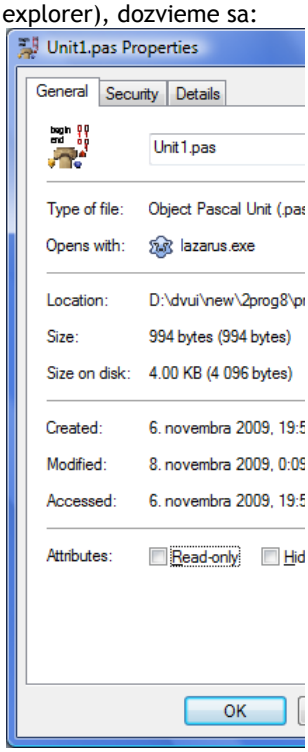
Všimnite si, že sme tu použili logickú funkciu **Eof**. Po spustení programu sme dostali takéto čísla (vám vyjdú pravdepodobne trochu iné):



To, že je takto zistený počet riadkov správny, môžeme prekontrolovať priamo v editore kódu prostredia Lazarus.

Vidíme, že súbor **uit1.**pas zaberá na disku presne 994 bajtov, čo je o 114 bajtov viac, ako sme vypočítali programom. Dôvod, prečo je to tak je jednoduchý. Každý riadok v súbore je ukončený špeciálnou dvojicou znakov, tzv. CR a LF a ich Ascii kódy sú 13 a 10. Keďže riadkov sme napočítali 57, tak na disku súbor naozaj obsahuje o

Ak budeme zisťovať počet znakov napr. v súborovom manažérovi (Windows explorer), dozvieme sa:



2*57 znakov viac ako zistených 880 znakov. Neskôr uvidíme, ako môžeme využiť tieto dva špeciálne znaky.

Nasledujúci program pomocou dvoch súborových premenných skopíruje po riadkoch obsah jedného súboru do druhého:

```
var
  Subor, Subor1: TextFile;
  Retazec: string;
begin
  AssignFile(Subor, 'unit1.pas');
  Reset(Subor);
  AssignFile(Subor1, 'text.txt');
  Rewrite(Subor1);
  while not Eof(Subor) do
  begin
    ReadLn(Subor, Retazec);
    WriteLn(Subor1, Retazec);
  end;
  CloseFile(Subor);
  CloseFile(Subor1);
  Memo1.Lines.Append('hotovo');
end;
```

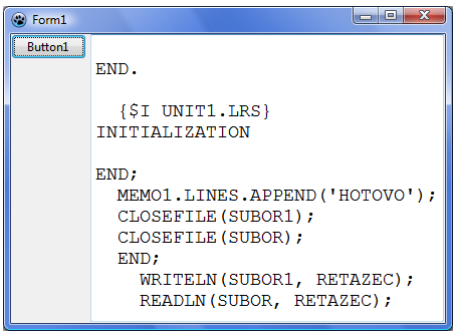
V programe sme použili aj príkaz **WriteLn**, ktorý sme zatiaľ používali len pri zápise čísel a prípadne medzier. Samozrejme, že do súboru môžeme zapisovať aj ľubovoľne dlhé znakové reťazce.

Program najprv otvorí prvý súbor na čítanie (súbor **unit1.pas**) a druhý súbor na zápis (bude vytvárať súbor **text.txt**). Potom, kým neprejdeme všetky riadky prvého súboru, prečítame jeden riadok a hneď ho zapíšeme do druhého súboru. Na záver musíme oba súbory zavrieť. Program na konci vypíše správu "*hotovo*", aby sme videli, že úspešne dokopíroval - bez tejto správy by sme po zatlačení **Button1** nevideli žiadnu reakciu programu.

V ďalšom programe budeme upravovať obsah textového súboru **text.txt** tak, že ho najprv celý prečítame do pamäte (do poľa reťazcov) a potom upravené reťazce zapíšeme namiesto pôvodného obsahu súboru:

```
var
  Subor: TextFile;
  Pole: array [1..1000] of string;
  I, Pocet: Integer;
begin
  AssignFile(Subor, 'text.txt');
  Reset(Subor);
  Pocet:= 0;
  while not Eof(Subor) do
  begin
    Inc(Pocet);
    ReadLn(Subor, Pole[Pocet]);
  end;
  Rewrite(Subor);
  for I := Pocet downto 1 do
    WriteLn(Subor, UpperCase(Pole[I]));
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('text.txt');
end;
```

V programe sme si vyhradili 1000 prvkové pole, lebo predpokladáme, že čítaný súbor nemá viac ako 1000 riadkov. Po prečítaní a uložení všetkých riadkov do poľa máme v premennej **Pocet** momentálny počet riadkov súboru. Teraz súbor **text.txt** otvoríme na zápis (**Rewrite**) a zapíšeme do neho všetky riadky ale od posledného po prvý (riadky teraz budú v opačnom poradí). Zároveň s tým každý reťazec pomocou štandardnej funkcie **UpperCase** prerobíme na veľké písmená:



V tomto výpise vidíme len zopár prvých riadkov upraveného súboru – sú to pôvodne posledné riadky súboru. Zvyšné riadky by sme videli po vyrolovaní textovej plochy.

Už vieme, že každý riadok v súbore končí dvoma znakmi s kódmi 13 a 10. Ak by sme otestovali takýto zápis

```
WriteLn(Subor, 'prvý riadok'+Char(13)+Char(10)+'druhý riadok');
```

mohli by sme vidieť, že sa do súboru zapísali pod seba takéto dva riadky:

```
prvý riadok
druhý riadok
```

Naozaj sme takto v súbore korektne vyrobili dva riadky. Keďže Pascal umožňuje vloženie špeciálnych znakov do znakového reťazca, môžeme to zapísať aj takto úspornejšie:

```
WriteLn(Subor, 'prvý riadok'#13#10'druhý riadok');
```

V Pascale zápis `'abc'#13#10'def'` znamená, že dva špeciálne znaky sú súčasťou reťazca. Môžeme to zapísať aj ako `'abc' + #13#10 + 'def'`

Túto vlastnosť môžeme využiť aj pri prečítaní celého súboru do pamäti. Predtým sme prečítané riadky ukladali do poľa reťazcov. Teraz uvidíme, ako všetky riadky uložíme do jednej reťazcovej premennej:

```
var
  Subor: TextFile;
  Retazec, R: string;
begin
  AssignFile(Subor, 'text.txt');
  Reset(Subor);
  Retazec := '';
  while not Eof(Subor) do
  begin
    ReadLn(Subor, R);
    Retazec := R + #13#10 + Retazec;
  end;
  Rewrite(Subor);
  WriteLn(Subor, UpperCase(Retazec));
  CloseFile(Subor);
  Memo1.Lines.LoadFromFile('text.txt');
end;
```

V programe ku každému prečítanému riadku prilepíme dva znaky **#13#10** a pridáme ich k premennej **Retazec** (v tomto prípade ich ukladáme v opačnom poradí ako prichádzajú). Teraz jediným príkazom **WriteLn** zapíšeme tento reťazec do súboru. Po spustení dostaneme úplne rovnaký výsledok ako predchádzajúci program.

Pri práci s textovými súbormi sa niekedy používa ešte špeciálny znak s kódom 9, t.j. **#9**. Tento znak je kódom pre tabulátor a niektoré aplikácie vyžadujú, aby boli vstupné údaje navzájom oddelené práve týmto znakom. Aj pascalovský **Read** vie pracovať s takýmto oddeľovačom.

Čo sme sa naučili

Pomocou príkazov **Read**, resp. **ReadLn** môžeme do reťazcovej premennej prečítať naraz celý riadok. Pomocou príkazov **Write**, resp. **WriteLn** môžeme do súboru zapisovať aj znakové reťazce. Ak takýto reťazec obsahuje dvojicu špeciálnych znakov s kódmi 13 a 10 (môžeme ich zapísať ako **#13#10**), tieto označujú koniec riadka a teda ďalšie znaky v reťazci už budú v novom riadku.

V tejto časti sme ukázali nasledovné techniky:

- celý textový súbor sme uložili do poľa znakových reťazcov,
- celý textový súbor sme uložili do jednej reťazcovej premennej,
- súbor uložený v jednej reťazcovej premennej môžeme zapísať do súboru jediným príkazom **WriteLn.**

Úlohy na precvičenie

Zadanie 1	Napíšte program, ktorý vypíše najdlhší riadok nejakého súboru. Zároveň spočíta počet prázdnych riadkov súboru.
Zadanie 2	Napíšte program, ktorý prečíta textový súbor **subor1.txt** a vytvorí jeho kópiu do **subor2.txt**. Pritom bude z výstupného súboru vyhadzovať prázdne riadky.
Zadanie 3	Súbor **ziaci.txt** obsahuje mená žiakov v tvare priezvisko, medzera a krstné meno. Napíšte program, ktorý opraví tento súbor tak, že mená začínajú krstným menom a až za tým je priezvisko žiaka. V súbore nie je viac ako 100 žiakov.
Zadanie 4	Napíšte program, ktorý spojí dva textové súbory **prvy.txt** a **druhy.txt** a vytvorí nový súbor **treti.txt**.

2. Pracujeme po znakoch

Čítať a zapisovať do súboru môžeme aj po znakoch. Hoci je tento spôsob najprácnejší, veľakrát je veľmi výhodný.

Nasledujúci program bude po znakoch čítať celý súbor. Ak nájde znak #13, bude predpokladať, že tu skočil riadok a zvýši počítadlo riadkov (premennú **Riadky**) o 1. Zároveň do premennej **Znaky** ukladá počet prečítaných znakov:

```
var
  Subor: TextFile;
  Znak: Char;
  Riadky, Znaky: Integer;
begin
  AssignFile(Subor, 'text.txt');
  Reset(Subor);
  Riadky := 0;
  Znaky := 0;
  while not Eof(Subor) do
  begin
    Read(Subor, Znak);
    Znaky := Znaky + 1;
    if Znak = #13 then
      Riadky := Riadky + 1;
  end;
  CloseFile(Subor);
  Memo1.Lines.Append('počet riadkov: ' + IntToStr(Riadky));
  Memo1.Lines.Append('počet znakov: ' + IntToStr(Znaky));
end;
```

Na rozdiel od programu, ktorý zisťoval počet znakov a riadkov čítaním celých riadkov do reťazcovej premennej, tento program zistí skutočný počet znakov v súbore, t.j. počet aj so špeciálnymi ukončovacími znakmi #13 a #10.

V predchádzajúcej časti sme vytvárali kópiu nejakého súboru tak, že sme čítali celé riadky a celé sme ich zapisovali do druhého súboru. Teraz ukážeme tú istú ideu s kopírovaním po znakoch:

```
var
  Subor, Subor1: TextFile;
  Znak: Char;
begin
  AssignFile(Subor, 'text.txt');
  Reset(Subor);
  AssignFile(Subor1, 'text1.txt');
  Rewrite(Subor1);
  while not Eof(Subor) do
  begin
    Read(Subor, Znak);
    Write(Subor1, Znak);
  end;
  CloseFile(Subor);
  CloseFile(Subor1);
  Memo1.Lines.Append('hotovo');
end;
```

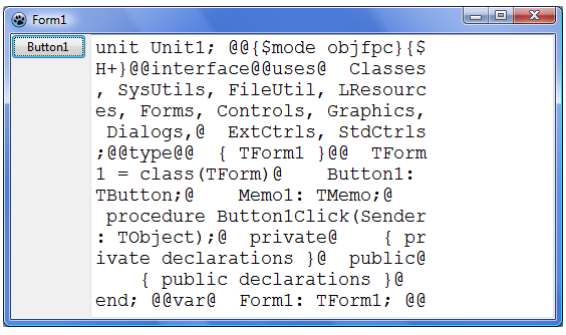
Ak porovnáme navzájom oba spôsoby vytvárania kópie (po celých riadkoch a po samostatných znakoch), bude pravdepodobne toto nové riešenie o kúsok "pomalšie". Časový rozdiel by sme asi videli len vtedy, keby sme takto kopírovali nejaký súbor, ktorý má veľmi veľký počet riadkov.

Pri vytváraní kópie súboru po znakoch, môžeme tieto znaky kontrolovať, upravovať, filtrovať a pod. V nasledujúcom programe budeme počítať počet znakov, ktoré sme už zapísali do súboru, pritom koniec riadku prvého súboru nahradíme znakom '@' a do druhého súboru zapíšeme koniec riadku, len keď sme už zapísali 30 znakov:

```
var
  Subor, Subor1: TextFile;
  Znak: Char;
  Pocet: Integer;
begin
  AssignFile(Subor, 'text.txt');
  Reset(Subor);
  AssignFile(Subor1, 'text1.txt');
  Rewrite(Subor1);
  Pocet := 0;
  while not Eof(Subor) do
  begin
    if Eoln(Subor) then
    begin
      ReadLn(Subor);
      Znak := '@';
    end
    else
      Read(Subor, Znak);
    Write(Subor1, Znak);
    Inc(Pocet);
    if Pocet = 30 then
    begin
      WriteLn(Subor1);
      Pocet := 0;
    end;
  end;
  if Pocet <> 0 then
    WriteLn(Subor1);
  CloseFile(Subor);
  CloseFile(Subor1);
  Memo1.Lines.LoadFromFile('text1.txt');
end;
```

V programe sme použili logickú funkciu **Eoln**, ktorá zisťuje (vráti **True**) či sme v súbore nastavení na konci nejakého riadku. Pripomeňme si, že pri čítaní čísel sme predtým používali funkciu **SeekEoln**, ktorá ale preskakovala medzery na konci riadkov. V tomto našom programe sa nám to nemusí hodiť, preto sme použili príkaz **Eoln**, ktorý medzery nepreskakuje.

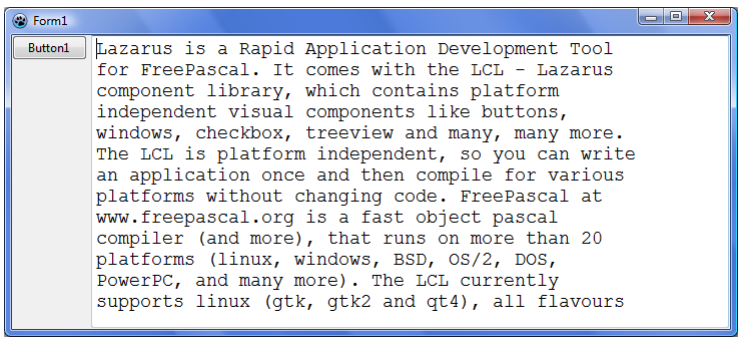
V tomto programe je dôležité si všimnúť, že konce riadkov (vtedy platí **Eoln**) spracovávame inak - nečítame ich ako konce riadkov (t.j. dva špeciálne znaky #13 a #10), ale ich preskočíme pomocou **Readln(Subor)**. Po spustení dostávame takýto súbor



Takto vytvorená kópia nemá pravdepodobne žiaden praktický význam, len ukazuje spôsob zápisu algoritmu. Podobná idea by sa ale dala využiť pri zarovnávaní nejakého bloku textu tak, aby slová nepresahovali nejakú špecifikovanú šírku. Pripravíme si nejaký textový súbor **odsek.txt**, ktorý bude obsahovať nejaký súvislý text a nasledujúci program z neho vytvorí zarovnaný text, ktorého riadky nepresiahnu nejakú zadanú šírku:

```
const
  Sirka = 50;
var
  Subor, Subor1: TextFile;
  Znak: Char;
  Pocet: Integer;
  Slovo: string;
begin
  AssignFile(Subor, 'odsek.txt');
  Reset(Subor);
  AssignFile(Subor1, 'zarovnany.txt');
  Rewrite(Subor1);
  Pocet := 0;
  Slovo := '';
  while not Eof(Subor) do
  begin
    if Eoln(Subor) then
    begin
      ReadLn(Subor);
      Znak := ' ';
    end
    else
      Read(Subor, Znak);
    if Znak <> ' ' then
      Slovo := Slovo + Znak
    else if Slovo <> '' then
    begin
      if Pocet + Length(Slovo) > Sirka then
      begin
        WriteLn(Subor1);
        Pocet := 0;
      end;
      Write(Subor1, Slovo, ' ');
      Pocet := Pocet + Length(Slovo) + 1;
      Slovo := '';
    end;
  end;
  if Slovo <> '' then
    Write(Subor1, Slovo);
  WriteLn(Subor1);
  CloseFile(Subor);
  CloseFile(Subor1);
  Memo1.Lines.LoadFromFile('zarovnany.txt');
end;
```

Po spustení na nejakom pripravenom súbore dostaneme takýto výstup:



Odporúčame, aby ste v programe odsledovali, ako sa odfiltruje viac medzier, ktoré idú v súbore tesne za sebou. Tiež by sme do tohto programu vedeli jednoducho pridať aj napríklad zisťovanie počtu slov v súbore, zisťovanie najdlhšieho slova, nahradzovanie konkrétneho slova (alebo skupiny slov) nejakým reťazcom, napr. nahradenie nejakého konkrétneho nespisovného slova reťazcom '***'.

Čo sme sa naučili

Súbor môžeme čítať, resp. vytvárať aj po znakoch. Zvykneme pritom používať logické funkcie **Eof** a **Eoln**.

V tejto časti sme ukázali nasledovné techniky:

- ako v súbore zisťujeme počet znakov a počet riadkov,
- ako môžeme pri kopírovaní súboru kontrolovať niektoré znaky a prípadne ich zameniť za iné,
- ako kopírujeme súbor po slovách a ako môžeme tieto slová vo výstupnom súbore zarovnávať.

Úlohy na precvičenie

Zadanie 1	Textový súbor obsahuje nejaký text, ktorý obsahuje aj písmená s diakritikou. Napíšte program, ktorý prekopíruje tento súbor a pritom odstráni diakritiku.
Zadanie 2	Napíšte program, ktorý dostáva nejaký väčší textový súbor so slovenským textom. Program zistí frekvenčnú tabuľku výskytov všetkých písmen v texte.
Zadanie 3	Napíšte program, ktorý číta textový súbor a do druhého súboru zapisuje jeho kópiu. Pritom vyhadzuje zbytočné medzery medzi slovami: medzi slovami stačí jedna medzera, na začiatku a konci riadkov medzery byť nemusia.

3. Znaky aj čísla v súbore

V prvej tematickej jednotke sme sa naučili čítať čísla zo súboru. Tento spôsob funguje správne len vtedy, keď takýto súbor obsahuje len čísla a tieto sú navzájom oddelené aspoň jednou medzerou, resp. koncom riadka. V praxi sa ale často stretáme s prípadmi, keď súbor okrem čísel obsahuje aj iné texty, resp. čísla sú navzájom oddelené iným ako medzerovým znakom.

Začnime takýmto príkladom: pani učiteľka si pripravila textový súbor, v ktorom si eviduje mená žiakov a ich známky za celý polrok. V každom riadku súboru **3a.txt** je najprv meno žiaka ukončené znakom dvojbodka. Za tým v riadku nasledujú body z písomiek, ktoré sú navzájom oddelené čiarkou. Úlohou programu bude do súboru **3apolrok.txt** ku každému žiakovi jeho priemerný počet bodov.

Na tomto programe ukážeme použitie ďalšieho variantu podmieneného cyklu, tzv. **cyklus s podmienkou na konci** – repeat-cyklus. Schéma cyklu je nasledovná

```
repeat

  postupnosť príkazov

until podmienka;
```

Príkaz funguje nasledovne:

- najprv sa vykoná *postupnosť príkazov* v cykle medzi rezervovanými slovami **repeat** a **until**,
- potom sa otestuje podmienka za slovom **until** – je to podmienka ukončenia cyklu – ak má podmienka hodnotu **True**, cyklus končí a keď podmienka neplatí, cyklus pokračuje opäť vykonaním postupnosti príkazov v tele cyklu.

Pripomeňme si cyklus **while**: kým platí podmienka, vykonaj telo cyklu – jeden príkaz za slovom **do** (často je to zložený príkaz **begin** - **end**). Cyklus skončí, keď prestane podmienka platiť.

Cyklus **repeat** by sme prečítali: vykonávaj postupnosť príkazov, až kým nenastane podmienka ukončenia cyklu. Cyklus skončí, až keď bude podmienka platiť.

Vidíme teda dôležité rozdiely s **while**-cyklom:

- opačná podmienka rozhoduje o tom, či sa cyklus opakuje alebo končí,
- **while** má podmienku na začiatku a **repeat** na konci,
- ak podmienka už neplatí pre cyklom, **while** sa nevykoná ani raz, **repeat** prejde aspoň raz,
- **while** vykonáva iba jeden príkaz, preto, ak chceme vykonať viac príkazov, musíme ich uzavrieť medzi **begin** a **end**.

Ukážme cyklus **repeat** na konkrétnych príkladoch. Na začiatku riadku v súbore je postupnosť znakov ukončená znakom dvojbodka. Treba ich aj s touto dvojbodkou prekopírovať do druhého súboru. Zapíšeme:

```
repeat
  Read(Subor1, Znak);
  Write(Subor2, Znak);
until Znak = ':';
```

Ak by sme toto isté chceli zapísať while-cyklom, muselo by to vyzerať asi takto:

```
Read(Subor1, Znak);
while Znak <> ':'  do
begin
  Write(Subor2, Znak);
  Read(Subor1, Znak);
end;
Write(Subor2, Znak);
```

Niekedy sa mylne predpokladá, že znak s kódom #26 je znak konca súboru. V skutočnosti je to znak, ktorý sa automaticky generuje príkazom **Read**, keď čítame už za koncom súboru. V skutočnosti sa tento znak v súbore nenachádza.

Oba tieto prípady majú rovnaký nedostatok: ak sa v súbore vôbec nenachádza znak dvojbodka, oba cykly sa zacyklia. Prečítanie znaku (**Read(Subor1, Znak);**) na konci súboru, t.j. keď **Eof(Subor1)** je **True**, nespôsobí žiadnu chybu, ale do premennej **znak** priradí znak s kódom 26, t.j. #26. Keďže to robíme v cykle a táto prečítaná hodnota je stále rôzna od znaku dvojbodka, tento cyklus nikdy neskončí.

Upozorňujeme, že používanie repeat-cyklu je náročnejšie ako while-cyklu a najmä začiatočník s ním robí veľa chýb. V tomto materiáli ho uvádzame preto, že v pascale sa má najväčšie využitie práve pri práci s textovým súborom.

Využiť repeat-cyklus môžeme aj vtedy, keď v riadku vstupného súboru potrebujeme spracovať postupnosť celých čísel, ktoré sú navzájom oddelené znakom čiarka. Opäť predpokladáme, že čiarky sú len medzi číslami a za posledným číslom už čiarka nie je. Tiež predpokladáme, že v riadku je minimálne jedno číslo:

```
repeat
  Cislo := 0;
  Read(Subor1, Znak);
  while (Znak >= '0') and (Znak <= '9') do
  begin
    Cislo := Cislo * 10 + Ord(Znak) - Ord('0');
    Read(Subor1, Znak);
  end;
  // spracuj číslo
until Znak <> ',';
```

Tento cyklus najprv prevedie postupnosť znakov na číslo a ak hneď ďalší znak za týmto číslom nie je čiarka, cyklus končí. Cyklus spracováva postupnosť čísel, kým sú za číslami čiarky. Aj tento repeat-cyklus by sa dal prepísať na while-cyklus, ale nebolo by to zapísané takto elegantne. Môžete to vyskúšať.

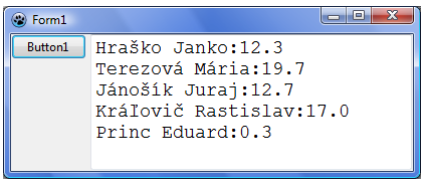
Vráťme sa k riešeniu príkladu, v ktorom budeme počítať priemer bodov žiakov zo súboru **3a.txt**. Pripravme testovací súbor, napr.:

```
Hraško Janko:10,12,13,14
Terezová Mária:20,19,20
Jánošík Juraj:1,5,10,15,20,25
Kráľovič Rastislav:17
Princ Eduard:0,0,0,2,0,0
```

Malý problém by mohol vzniknúť, keby tento súbor obsahoval na konci prázdny riadok. Prázdny riadok znamená, že je to riadok, ktorý neobsahuje znak ':' a takýto riadok nevyhovuje zadaniu – v každom riadku je meno žiaka ukončené dvojbodkou a to by pri niektorom z cyklov mohlo spôsobiť zacyklenie. To je dôvod, prečo namiesto **Eof** použijeme vo vonkajšom while-cykle **SeekEof**:

```
var
  Subor1, Subor2: TextFile;
  Znak: Char;
  Cislo, Pocet, Sucet: Integer;
begin
  AssignFile(Subor1, '3a.txt');
  Reset(Subor1);
  AssignFile(Subor2, '3apolrok.txt');
  Rewrite(Subor2);
  while not SeekEof(Subor1) do
  begin
    repeat
      Read(Subor1, Znak);
      Write(Subor2, Znak);
    until Znak = ':';
    Pocet := 0;
    Sucet := 0;
    repeat
      Cislo := 0;
      Read(Subor1, Znak);
      while (Znak >= '0') and (Znak <= '9') do
      begin
        Cislo := Cislo * 10 + Ord(Znak) - Ord('0');
        Read(Subor1, Znak);
      end;
      Inc(Pocet);
      Sucet := Sucet + Cislo;
    until Znak <> ',';
    WriteLn(Subor2, Sucet/Pocet:0:1);
    ReadLn(Subor1);
  end;
  CloseFile(Subor1);
  CloseFile(Subor2);
  Memo1.Lines.LoadFromFile('3apolrok.txt');
end;
```

Obidva repeat-cykly, ktoré sme tu použili, sme už vysvetlili vyššie. Po spustení programu dostávame takýto výpis:



Záverečný program tejto kapitoly rieši takúto úlohu: v súbore **obrazok.txt** máme popísané príkazy na nakreslenie nejakého obrázka. V každom riadku je jeden z grafických príkazov. Riadok začína jedným z písmen 'm', 'l', 'k', 'o', ktoré označujú príkazy MoveTo, LineTo, Kruh a Obdĺžnik. Prvé dva príkazy majú 2 parametre, Kruh má tri a Obdĺžnik má 4 parametre:

- **m** X, Y - označuje MoveTo(X, Y)
- **l** X, Y - označuje LineTo(X, Y)
- **k** X, Y, A - označuje Ellipse(X-A, Y-A, X+A, Y+A)
- **o** X,Y,A,B - označuje Rectangle(X, Y, X+A, Y+B)

Parametre sú od písmena príkazu oddelené medzerou, parametre sú navzájom oddelené čiarkami. Pripravili sme si nejaký testovací súbor **obrazok.txt**:

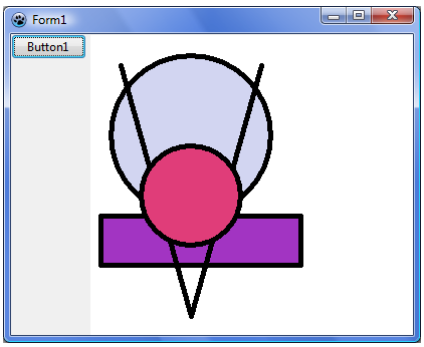
```
k 100,100,80
o 10,180,200,50
m 30,30
l 100,280
l 170,30
k 100,160,50
```

Pozrite, ako sme rozobrali riadok s grafickým príkazom na parametre. Parametre

sme najprv prečítali do znakového reťazca a ten sme potom rozobrali na podreťazce a pomocou **StrToInt** sme ich previedli na čísla:

```
var
  Subor: TextFile;
  Znak: Char;
  Retazec: string;
  X, Y, A, B: Integer;
begin
  AssignFile(Subor, 'obrazok.txt');
  Reset(Subor);
  Image1.Canvas.Pen.Width := 5;
  while not SeekEof(Subor) do
  begin
    ReadLn(Subor, Znak, Retazec);
    Retazec := Retazec + ',';
    X := StrToInt(Copy(Retazec, 1, Pos(',', Retazec)-1));
    Delete(Retazec, 1, Pos(',', Retazec));
    Y := StrToInt(Copy(Retazec, 1, Pos(',', Retazec)-1));
    Delete(Retazec, 1, Pos(',', Retazec));
    if (Znak = 'k') or (Znak = 'o') then
    begin
      A := StrToInt(Copy(Retazec, 1, Pos(',', Retazec)-1));
      Delete(Retazec, 1, Pos(',', Retazec));
    end;
    if Znak = 'o' then
      B := StrToInt(Copy(Retazec, 1, Pos(',', Retazec)-1));
    Image1.Canvas.Brush.Color := Random(256 * 256 * 256);
    case Znak of
      'm': Image1.Canvas.MoveTo(X, Y);
      'l': Image1.Canvas.LineTo(X, Y);
      'k': Image1.Canvas.Ellipse(X-A, Y-A, X+A, Y+A);
      'o': Image1.Canvas.Rectangle(X, Y, X+A, Y+B);
    end;
  end;
  CloseFile(Subor);
end;
```

Po spustení na pripravenom testovacom súbore dostávame:



Čo sme sa naučili

Okrem while-cyklu s podmienkou na začiatku Pascal ponúka aj repeat-cyklus s podmienkou na konci.

Ak čítame zo súboru do znakovej premennej na konci súboru, Pascal nehlási chybu, ale do premennej priradí znak #26.

V tejto časti sme ukázali nasledovné techniky:

- skladanie čísla z cifier prečítaných zo súboru,
- čítanie postupnosti čísel, ktoré sú navzájom oddelené nejakým oddeľovačom,
- čísla zo súboru môžeme prečítať aj tak, že riadok najprv prečítame do znakového reťazca a potom z neho vytiahneme čísla pomocou **StrToInt**.

Úlohy na precvičenie

Zadanie 1	V súbore **ulohy.txt** je v každom riadku jedno zadanie typu: číslo operácia číslo. Všetky čísla sú celé čísla, operáciou je jedna zo znamienok +, -, *. Napíšte program, ktorý vypisuje tieto riadky a vyhodnocuje ich. Napr. pre úlohu **17*11** program vypíše: **17*11=187**.
Zadanie 2	Napíšte program, ktorý najprv do poľa cifier vypočíta 100! Potom toto pole zapíše do textového súboru ale tak, že každú trojicu cifier oddelí medzerou. Napíšte aj druhý program, ktorý tento súbor znovu prečíta do poľa cifier.
Zadanie 3	Napíšte program, ktorý prečíta súbor s pascalovským programom (súbor s príponou **pas**) a zistí, či má tento program dobrý počet výskytov rezervovaných slov **begin** a **end**.
Zadanie 4	Vytvorte aplikáciu na evidenciu známok.
Zadanie 5	Vytvorte aplikáciu, ktorá bude kontrolovať diktáty alebo vyhodnocovať testy (veľa súborov s rovnakou štruktúrou).
Zadanie 6	Vytvorte aplikáciu, ktorá prekonvertuje súbor vo formáte **xls** do formátu **csv.**

Čo sme sa naučili v tomto module

Zhrnutie

Tento modul zoznámil s týmito základnými stavebnými kameňmi programovacieho jazyka:

- textový súbor, testovanie konca riadka a konca súboru
- prečítanie, resp. zápis čísla, znaku alebo znakového reťazca

Okrem toho boli uvedené tieto dôležité koncepty:

- otvorenie súboru na čítanie alebo zápis, zatvorenie súboru
- kopírovanie súboru

Preverenie výstupných vedomostí

Účastník vzdelávania vie naprogramovať takúto aplikáciu:

Textový súbor **subor.txt** obsahuje celé čísla aj nejaké texty. Prekopírujte ho do dvoch súborov takto: do súboru **cisla.txt** zapíšte len všetky čísla a navzájom ich oddeľte medzerami. Do súboru **text.txt** prekopírujte pôvodný text ale už bez čísel, ktoré sme zapísali do prvého výstupného súboru.

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/

Poznámky

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 8

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 978-80-8118-013-2