Programski jezik
C#

Delo z datotekami

Gregor Jerše, Matija Lokar in Srečo Uranič

V 0.92

februar 2009

Predgovor

Omenjeno gradivo pokriva delo s tekstovnimi datotekami v C# . Je del gradiv, namenjenih predmetu Programiranje 1 na višješolskem študiju Informatika. Glede na program naj bi študenti tu že poznali osnove programiranja v jeziku C# od osnovnih tipov, pogojnih stavkov, zabk tabel in vse do (statičnih) metod.

Pri sestavljanju gradiva smo imeli v mislih predvsem začetnike, ki se s programskimi jeziki srečujejo prvič. Zato je veliko zgledov, primerov programov ...

Samo gradivo je nastalo na osnovi zapiskov avtorjev, črpali pa smo tudi iz že objavljenih gradiv, pri katerih smo bili »vpleteni«. Osnova ta gradiva so bila:

- *Nina Kerčmar, Prvi koraki v Javi, diplomska naloga, UL FMF, 2006*
- *Danica Petric, Spoznavanje osnov programskega jezika C#, diplomska naloga, UL FMF, 2008*
- *Srečo Uranič, Microsoft C#.NET*
- *Matija Lokar, Osnove programiranja : programiranje – zakaj in vsaj kaj, Zavod za šolstvo, Ljubljana, 2005*
- *Gregor Jerše in Matija Lokar, Programiranje II, Objektno programiranje, B2, 2008*
- *Gregor Jerše in Matija Lokar, Programiranje II, Rekurzija in datoteke, B2, 2008*

Gradivo vsekakor ni dokončano in predstavlja delovno različico. V njem so zagotovo napake (upamo, da čimmanj), za katere se vnaprej opravičujemo. Da bo lažje spremljati spremembe, obstaja razdelek Zgodovina sprememb, kamor bova vpisovala spremembe med eno in drugo različico. Tako bo nekomu, ki si je prenesel starejšo različico, lažje ugotoviti, kaj je bilo v novi različici spremenjeno.

Gregor Jerše, Matija Lokar in Srečo Uranič

Kranj, november 2008

Zgodovina sprememb

11. 11. 2008: Različica V0.8 – več ali manj zlitje (še neurejeno) Matijevega in Sašovega gradiva. Premisliti je še potrebno kaj in kako.
25. 11. 2008: Različica V0.81 – popravki prvega dela do strani 46. Nadaljni del je potrebno še ustrezno umestiti.
24. 12. 2008: Različica V0.9 – manjši redakcijski popravki pred tiskanjem nekaj primerkov.
10. 2. 2009: Različica V0.92 – odstranjen del z binarnimi datotekami. Zadnji del označen kot dodatno gradivo.

KAZALO

Datoteke

Kaj je datoteka

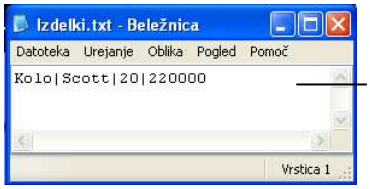
V Slovarju slovenskega knjižnega jezika (SSKJ) piše :
datoteka -e ž (e) elektr. urejena skupina podatkov pri (elektronskem) računalniku, podatkovna zbirka: vnesti nove podatke v datoteko / datoteka s programom.

Datoteka (ang. file) je zaporedje podatkov na računalniku, ki so shranjeni na disku ali kakem drugem pomnilniškem mediju (npr. CD-ROM, disketa, ključek). V datotekah hranimo različne podatke: besedila, preglednice, programe, ipd.

Glede na tip zapisa jih delimo na:

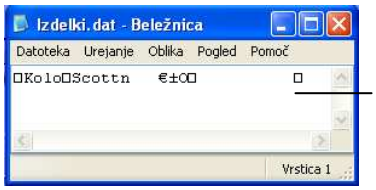
a) ***tekstovne*** datoteke
Tekstovne datoteke vsebujejo nize znakov oziroma zlogov, ki so ljudem berljivi. Poleg teh znakov so v tekstovnih datotekah zapisani določeni t.i. kontrolni znaki. Najpomembnejša tovrstna znaka sta znaka za konec vrstice (end of line) in konec datoteke. Najpomembnejše je, da vemo, da so tekstovne datoteke **razdeljene na vrstice**. Odpremo jih lahko v vsakem urejevalniku (npr. WordPad) ali jih izpišemo na zaslon.

V tekstovnih datotekah so tudi vsi številski podatki shranjeni kot zaporedje znakov (števk in morda decimalne pike). Zato jih moramo, če jih želimo uporabiti v aritmetičnih operacijah, spremeniti v številske podatke. V tekstovnih datotekah so torej vsi podatki shranjeni kot **tekstovni znaki**. Pogosto so posamezni sklopi znakov (besede, polja, …) med seboj ločeni s posebnimi ločili (npr znakom |, ali pa z znakom *vejica* ipd), datoteke pa vsebujejo tudi znake **end of line** (znak za konec vrstice). Tekstovne datoteke imajo torej vrstice.
Na sliki je prikazan izgled vsebine **tekstovne** datoteke odprte z Beležnico (v datoteki je ena sama vrstica).



b) ***binarne*** datoteke
Binarne datoteke vsebujejo podatke zapisane v binarnem zapisu. Zato je vsebina le teh datotek ljudem neberljiva (urejevalniki besedil običajno ne znajo prikazati posebnih znakov, namesto njih prikazujejo "kvadratke"). Binarne datoteke ne vsebujejo vrstic.

Tudi podatki v binarnih datotekah lahko vsebujejo znake, tako kot je to v tekstovnih datotekah, a so podatki med seboj ločeni s posebnimi znaki, zaradi česar je vsebina take datoteke, če jo odpremo z nekim urejevalnikom, skoraj neberljiva. Poleg tega binarne datoteke ne vsebujejo vrstic. Izgled vsebine **binarne** datoteke, če jo odpremo z Beležnico



Ukvarjali se bomo več ali manj le s tekstovnimi datotekami.

Imena datotek
Poimenovanje datoteke je odvisno od operacijskega sistema. V tej nalogi se bomo omejili na operacijske sisteme družine Windows.
Vsaka datoteka ima ime in je v neki mapi (imeniku). Polno ime datoteke dobimo tako, da zložimo skupaj njeno ime (t.i. kratko ime) in mapo.

Primer:
`D:\olimpijada\Peking\atletika.txt`
Če povemo z besedami: Ime datoteke je `atletika.txt` na enoti `D:`, v imeniku `Peking`, ki je v podimeniku imenika `olimpijada`.

Imena imenikov so v operacijskem sistemu Windows ločena z znakom \. Spomnimo se, da ima znak \, če ga v jeziku java in C# zapišemo kot sestavni del niza, poseben pomen. Šele v kombinaciji z naslednjim znakom predstavljajo nek znak (npr. \n - nova vrstica, \t - tabulator, \r - return). Zato, kadar želimo, da je znak \ sestavni del niza, moramo napisati kombinacijo \\. Spomnimo se še, da v jeziku C# znak \ v nizu lahko izgubi posebni pomen, kadar pred nizom uporabimo znak @.

Knjižnica (imenski prostor)
Za delo z datotekami v jeziku C# so zadolžene metode iz razredov v imenskem prostoru `System.IO`. Zato na začetku vsake datoteke, v kateri je program, ki dela z datotekami, napišemo

```
using System.IO;
```

```
1 using System;
2 using System.IO;
```

Navedemo torej uporabo omenjenega imenskega prostora. S tem poenostavimo klice teh metod.

Podatkovni tokovi
Kratica `IO` v imenskem prostoru `System.IO` predstavlja vhodne (*input*) in izhodne (*output*) tokove[1] (*streams*). S pomočjo njih lahko opravljamo želene operacije nad datotekami (npr. branje, pisanje). V podrobnosti glede tokov se ne bomo spuščali in bomo predstavili poenostavljeno zgodbo. Pisalni ali izhodni tok v jeziku C# predstavimo s spremenljivko tipa `StreamWriter`. Bralni ali vhodni tok pa je tipa `StreamReader`.

Podatkovne tokove si lahko predstavljamo takole. Lahko si mislimo, da je datoteka v imeniku jezero, reka, ki teče v jezero (ali iz njega), pa podatkovni tok, ki prinaša, oziroma odnaša vodo. Če potrebujemo reko, ki v jezero prinaša vodo, bomo v C# potrebovali spremenljivko tipa StreamWriter (pisanje toka), če pa potrebujemo reko, ki vodo (podatke) odnaša, pa StreamReader (branje toka).

Branje in pisanje na tekstovne datoteke

Tekstovne datoteke
Tekstovne datoteke vsebujejo nize znakov oziroma zlogov, ki so ljudem berljivi. Katere znake vsebujejo, je odvisno od uporabljene kodne tabele.

Ustvarimo datoteko
Poglejmo si najenostavnejši način, kako ustvarimo tekstovno datoteko. "Čarobna" vrstica z ukazom je #8.

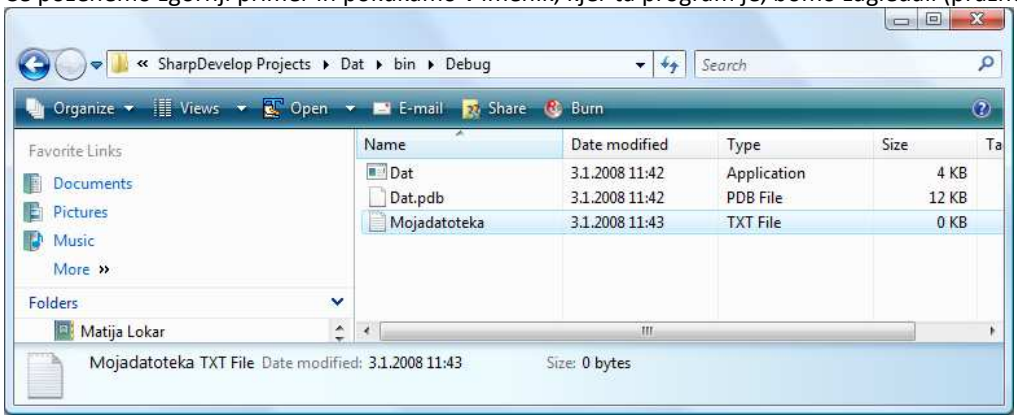
```
1:      using System;
2:      using System.IO;
```

[1] Tok ponazarja potek informacij od enega objekta k drugemu objektu.

```
3:      public class Program
4:      {
5:        public static void Main(string[] args)
6:        {
7:          File.CreateText("Mojadatoteka.txt");
8:        }
9:      }
```

"Zanimiva" je le vrstica 7. Če pokličemo metodo *File.CreateText("nekNiz"),* ustvarimo datoteko z imenom *nekNiz*. V tem nizu mora seveda biti na pravilen način napisano ime datoteke, kot ga dovoljuje operacijski sistem.

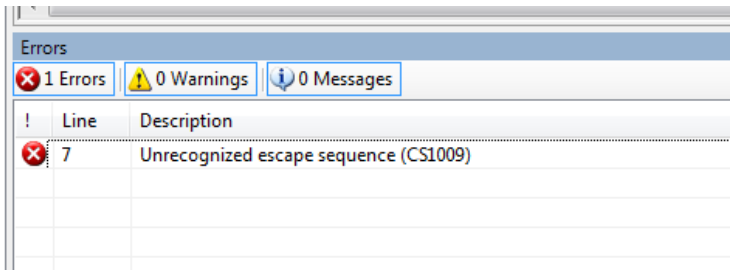
Če poženemo zgornji primer in pokukamo v imenik, kjer ta program je, bomo zagledali (prazno) datoteko.



Če bi želeli datoteko ustvariti v kakšnem drugem imeniku, moramo seveda napisati polno ime datoteke. Seveda takole

```
1:      using System;
2:      using System.IO;
3:      public class Program
4:      {
5:        public static void Main(string[] args)
6:        {
7:          File.CreateText("C:\temp\Mojadatoteka.txt");
8:        }
9:      }
```

ne gre, saj se prevajalnik hitro oglasi z



Spomnimo se, da ima znotraj niza znak '\' poseben pomen – napoveduje, da v nizu prihaja nek poseben znak. Če želimo dobiti \, moramo napisati dva. Torej

```
File.CreateText("C:\\temp\\Mojadatoteka.txt");
```

Sedaj ni težav in v želenem imeniku dobimo ustrezno datoteko.
Ker pa je tovrstni zapis s podvojenimi \ morda malo nepregleden, lahko pred nizem uporabimo znak @. S tem povemo, da se morajo vsi znaki v nizu jemati dobesedno. Zgornji zgled bi torej lahko napisali kot

```
File.CreateText(@"C:\temp\Mojadatoteka.txt");
```

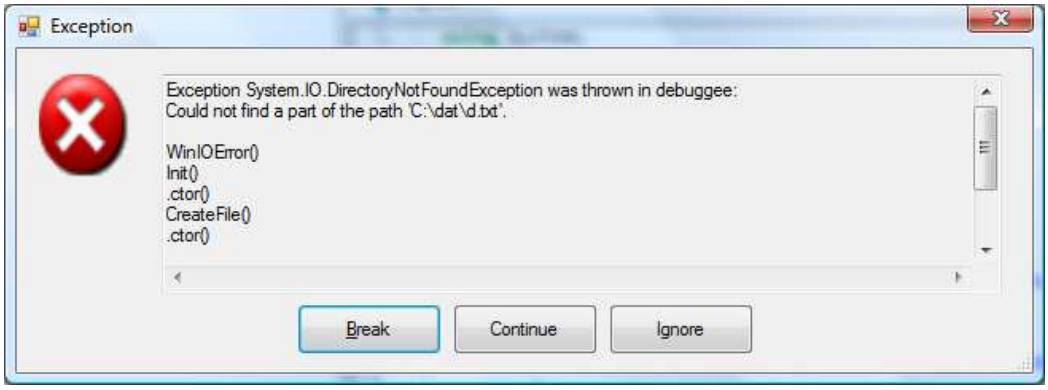
Vendar moramo biti pri uporabi tega ukaza previdni. Namreč, če datoteka že obstaja, jo s tem ukazom "povozimo". Izgubimo torej staro vsebino in ustvarimo novo, prazno datoteko.

Zato je smiselno, da prej preverimo, če datoteka že obstaja. Ustrezna metoda je

```
File.Exists(ime)
```

ki vrne true, če datoteka obstaja in false sicer.

Če imenika, kjer želimo narediti datoteko ni, se program "sesuje"

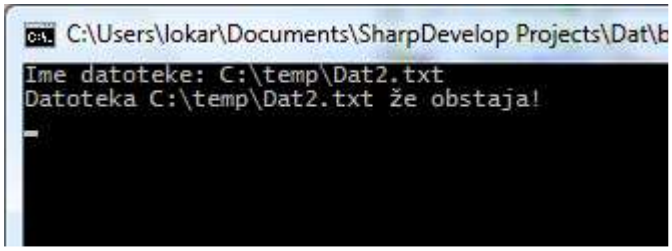


Zgled: Ustvari datoteko

Naredimo zgled, ki uporabnika vpraša po imenu datoteke. Če datoteke še ni, jo naredi, drugače pa izpiše ustrezno obvestilo.

```
10:  using System;
1:   using System.IO;
2:   public class Program
3:   {
4:     public static void Main(string[] args)
5:     {
6:       Console.Write("Ime datoteke: ");
7:       string ime = Console.ReadLine();
8:       if (File.Exists(ime)) {
9:         Console.WriteLine("Datoteka " + ime + " že obstaja!");
10:      } else {
11:        File.CreateText(ime);
12:        Console.WriteLine("Datoteko " + ime + " smo naredili!");
13:      }
14:      Console.ReadLine();
15:    }
```

```
16:  }
```



Iz slike vidimo še eno stvar. Ko tipkamo ime datoteke, \ navajamo običajno (enkratno), saj C# ve, da vnašamo "običajne" znake.

Razlaga programa: V 7. in 8. vrstici od uporabnika preberemo ime datoteke. V 9. vrstici preverimo, če datoteka obstaja. Če da, v 10. vrstici le izpišemo ustrezno obvestilo, drugače pa jo v 12. vrstici ustvarimo in uporabnika ustrezno obvestimo.

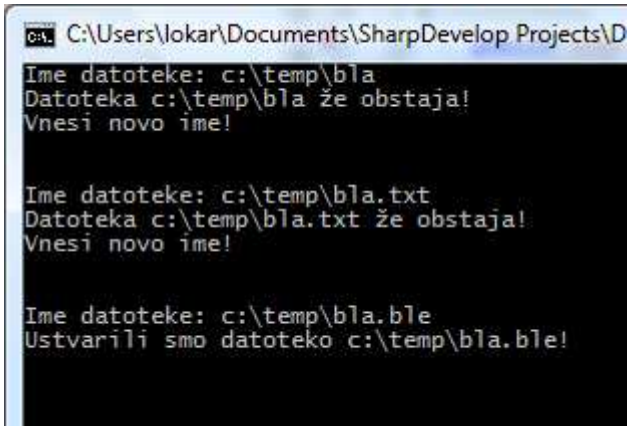
Zgled: Zagotovo ustvari datoteko

Denimo, da pišemo program, s katerim bomo nadzirali varnostne kamere v našem podjetju. Naš program mora ob vsakem zagonu začeti na novo pisati ustrezno dnevniško datoteko. Ker gre za izjemno skrivno operacijo, ni mogoče, da bi se datoteke ustvarjale kar v nekem fiksnem imeniku. Zato mora ime datoteke vnesti kar operater. Seveda morajo prejšnje datoteke ostati nespremenjene. Datotek se bo hitro nabralo zelo veliko, zato bo operater izgubil pregled nad imeni, ki jih je ustvaril. Zato mu pomagaj in napiši metodo, ki bo uporabnika tako dolgo spraševala po imenu datoteke, dokler ne bo napisal imena, ki zagotovo ni obstoječa datoteka. To ime naj metoda vrne kot rezultat.

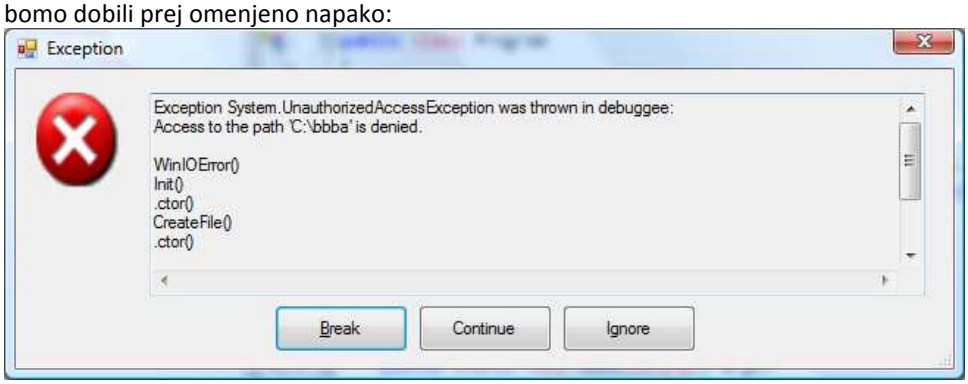
Ideja:

Metoda se bo začela z `public static string NovoIme()`. V metodi bomo prebrali ime datoteke in to potem ponavljali toliko časa, dokler metoda File.Exist ne bo vrnila false.

```
1:   using System;
2:   using System.IO;
3:   public class Program
4:   {
5:     public static string NovoIme() {
6:       Console.Write("Ime datoteke: ");
7:       string ime = Console.ReadLine();
8:       while (File.Exists(ime)) { // èe datoteka že obstaja
9:         Console.WriteLine("Datoteka " + ime + " že obstaja!");
10:        Console.WriteLine("Vnesi novo ime!\n\n");
11:        Console.Write("Ime datoteke: ");
12:        ime = Console.ReadLine();
13:      }
14:      return ime;
15:    }
16:    public static void Main(string[] args)
17:    {
18:      string imeDatoteke = NovoIme();
19:      File.CreateText(imeDatoteke);
20:      Console.WriteLine("Ustvarili smo datoteko " + imeDatoteke + "!");
21:      Console.ReadLine();
22:    }
23:  }
```



Seveda pa rešitev še ni "idealna". Namreč, če nimamo pravice, da bi datoteko v določenem imeniku ustvarili, bomo dobili prej omenjeno napako:



Temu se lahko izognemo na ta način, da preverimo pravice, ki jih izvajalni program ima. A to že presega kontekst naše zgodbe.

Pisanje na datoteko

Datoteko torej znamo ustvariti. A kako bi nanjo kaj zapisali? Kot vemo, lahko izpisujemo z metodo WriteLine (in z Write). A zaenkrat znamo pisati le na izhodno konzolo z

```
Console.WriteLine("nekaj izpišimo");
```

Zgodba pri pisanju na datoteko je povsem podobna. Metoda File.CreateText(), ko ustvari datoteko, namreč vrne oznako tako imenovanega podatkovnega toka. To oznako shranimo v spremenljivko tipa StreamWriter. In če sedaj napišemo

```
oznaka.WriteLine("Mi znamo pisati v datoteko");
```

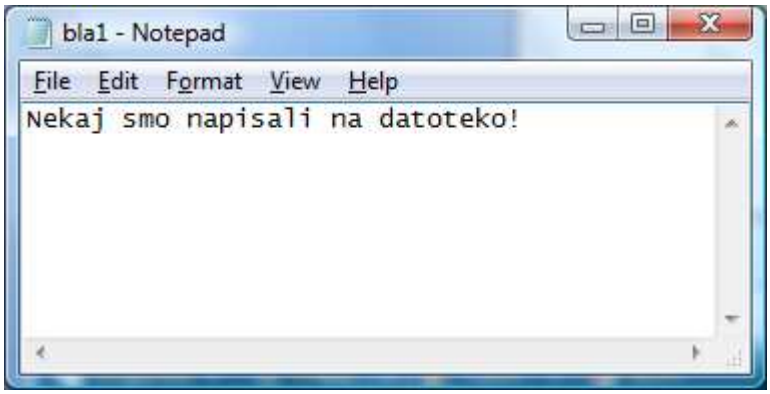
smo sedaj napisali omenjeno besedilo na datoteko, ki smo jo prej odprli in povezali z oznako.
Morebitne nejasnosti bo razjasnil zgled.

```
1:      public static void Main(string[] args)
2:      {
3:        string imeDatoteke = NovoIme();
4:        StreamWriter oznaka;
5:       oznaka = File.CreateText(imeDatoteke);
6:        oznaka.WriteLine("Nekaj smo napisali na datoteko!");
7:      }
```

Poženimo program in pokukajmo v imenik, kjer je nova datoteka. A glej. Datoteka sicer je tam, a je prazna. Zakaj? Vedno, ko nekaj počnemo z datotekami, jih je potrebno na koncu zapreti. To storimo z metodo Close. Če torej program spremenimo v

```
1:      public static void Main(string[] args)
2:      {
3:        string imeDatoteke = NovoIme();
4:        StreamWriter oznaka;
5:        oznaka = File.CreateText(imeDatoteke);
6:        oznaka.WriteLine("Nekaj smo napisali na datoteko!");
7:        oznaka.Close();
8:      }
```

in si sedaj ogledamo datoteko, vidimo, da ni več dolga 0 zlogov. Če jo odpremo v najkoristnejšem programu Beležnici:



bomo na njej našli omenjeni stavek.

Če podatkovnega toka po pisanju podatkov ne zapremo, je možno, da v nastali datoteki manjka del vsebine, oziroma vsebine sploh ni. Namreč, da napisan program pospeši operacije s diskom, se vsebina ne zapiše takoj v datoteko na disku, ampak v vmesni polnilnik. Šele ko je ta poln, se celotna vsebina medpomnilnika zapiše na datoteko. Metoda `Close` v C# poskrbita, da je medpomnilnik izpraznjen tudi, če še ni povsem poln.

Seveda bi program lahko napisali tudi tako, da bi vrstici 4 in 5 združili

```
1:      public static void Main(string[] args)
2:      {
3:        string imeDatoteke = NovoIme();
4:        StreamWriter oznaka = File.CreateText(imeDatoteke);
5:        oznaka.WriteLine("Nekaj smo napisali na datoteko!");
6:        oznaka.Close();
7:      }
```

Oglejmo si sedaj nekaj zgledov programov, kjer pišemo na datoteke.

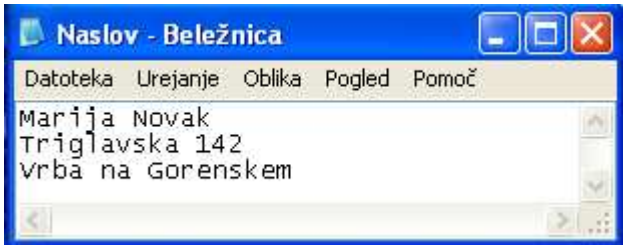
Osebni podatki

Na enoti `C` v korenskem imeniku ustvarimo mapo `Tekstovna`. V tej podmapi ustvarimo tekstovno datoteko `Naslov.txt` in vanjo zapišimo svoj naslov. To naredimo le, če datoteke še ni.

```
C1:   using System;
C2:   using System.IO;
C3:   public class OsebniPodatki{
```

```
C4:       public static void Main(string[] args){
C5:          // Pot in ime
C6:          string ime = @"C:\Tekstovna\Naslov.txt";
C7:
C8:          // Preverimo obstoj datoteke
C9:          if(File.Exists(ime)){
C10:            Console.WriteLine("Datoteka ze obstaja.");
C11:            return;
C12:         }
C13:
C14:         // Ustvarimo novo datoteko
C15:         StreamWriter dat = File.CreateText(ime);
C16:
C17:         // Zapišemo osebne podatke
C18:         dat.WriteLine("Marija Novak");
C19:         dat.WriteLine("Triglavska 142");
C20:         dat.WriteLine("Vrba na Gorenskem");
C21:
C22:         // Zapremo datoteko za pisanje
C23:         dat.Close();
C24:      }
C25:   }
```

Zapis na datoteki Naslov.txt:



Razlaga. Najprej določimo niz, ki predstavlja polno ime datoteke (vrstica C6). Če bi za niz določili le kratko ime, bi se datoteka ustvarila tam, kjer bi izvedli program `OsebniPodatki.cs`. V vrstici C9 preverimo obstoj datoteke. Če datoteka obstaja, izpišemo obvestilo (vrstica C10) in končamo izvajanje programa (vrstica C11). Nato v vrstici C15 odprem datoteko za pisanje. S klicem metode `dat.WriteLine()` podatke zapišemo na datoteko (vrstice C18 - C20). Po končanem zapisu datoteko zapremo (vrstica C23). Če tega ne bi storili, datoteka ne bi imela vsebine.

Števila

V tekstovno datoteko `Stevila.txt` zapišimo prvih `n` sodih števil, ki niso deljiva s šest. Števila izpišimo ločena z znakom '`/`', torej na primer kot:

```
1/2/3/8/14/16/20/22/
```

Ideja.
Napišemo metodo, ki ima dva parametra:

- *ime* - ime datoteke
- *n* - koliko števil bo v datoteki

Rezultat metode bo *void*, saj ima metoda le učinek ustvarila bo datoteko.

```
private static void Stevila(string ime, int n)
```

Najprej preverimo, če datoteka *ime* obstaja. V primeru obstoja izpišimo opozorilo in prekinimo izvajanje metode. Lepše bi bilo, če bi namesto izpisa obvestila metoda vrgla izjemo. A ker se z izjemami v jeziku C# ne bomo ukvarjali, bomo napako javili z izpisom obvestila.

```
if (File.Exists(ime)){
   Console.WriteLine("Napaka.");
   return;
```

```
}
```

Nato ustvarimo datoteko in jo povežemo s tokom za pisanje.

```
StreamWriter dat;
dat = File.CreateText(ime);
```

Ustvarimo še števec za štetje v datoteko zapisanih števil in spremenljivko, ki bo hranila tekoče sodo število.

```
int stevec = 1;
int soda = 0;
```

Nato naredimo zanko, ki se bo izvajala toliko časa, dokler ne bo *stevec* postal večji kot *n*. V zanki:

- Preverimo, če vrednost spremenljivke *soda* ni deljiva s *6*.
  - Če je pogoj izpolnjen, vrednost spremenljivke *soda* zapišemo na datoteko.
  - Povečamo spremenljivko *stevec* za 1.
- Spremenljivko *soda* nastavimo na naslednje sodo število..

```
while (stevec <= n){
   if (soda % 6 != 0){
      dat.Write(soda + "/");
      stevec++;
   }
   soda = soda + 2;
}
```

Zatem zapremo podatkovni tok.

```
dat.Close();
```

Na koncu še napišemo metodo *Main*. V metodi:

- napišemo ime datoteke,
- preberemo število *n* in
- pokličemo metodo *Stevila*.

```
public static void Main(string[] args){
   // Ime datoteke
   string datoteka = "Stevilo.txt";
   // Število števil v datoteki
   Console.Write("Vnesi n: ");
   int n = int.Parse(Console.ReadLine());
   Stevila(datoteka,n); // Klic metode
}
```

Sestavimo zapisane dele programa v celoto.

```
C1:   using System;
C2:   using System.IO;
C3:   public class Program{
C4:      private static void Stevila(string ime, int n){
C5:         // Preverjanje obstoja datoteke
C6:         if (File.Exists(ime)){
C7:            Console.WriteLine("Napaka.");
C8:            return;
C9:         }
C10:
C11:        StreamWriter dat; // Ustvarimo tok za pisanje na datoteko
C12:        dat = File.CreateText(ime);
C13:
C14:        // Pomožni spremenljivki
C15:        int stevec = 1;
C16:        int soda = 0;
C17:
C18:        // Zapisujemo števila na datoteko
C19:        while (stevec <= n){
C20:           if (soda % 6 != 0){
C21:              dat.Write(soda + "/");
```

```
C22:                stevec++;
C23:             }
C24:             soda = soda + 2;
C25:          }
C26:
C27:          // Zapremo datoteko za pisanje
C28:          dat.Close();
C29:       }
C30:
C31:       public static void Main(string[] args){
C32:          // Ime datoteke
C33:          string datoteka = "Stevilo.txt";
C34:          // Število števil v datoteki
C35:          Console.Write("Vnesi n: ");
C36:          int n = int.Parse(Console.ReadLine());
C37:          Stevila(datoteka,n); // Klic metode
C38:       }
C39:    }
```

Zapis 100 vrstic

Napišimo sedaj program, ki bo v datoteko StoVrstic.txt zapisal 100 vrstic, denimo takih: 1. vrstica, 2. vrstica, ..., 100. vrstica

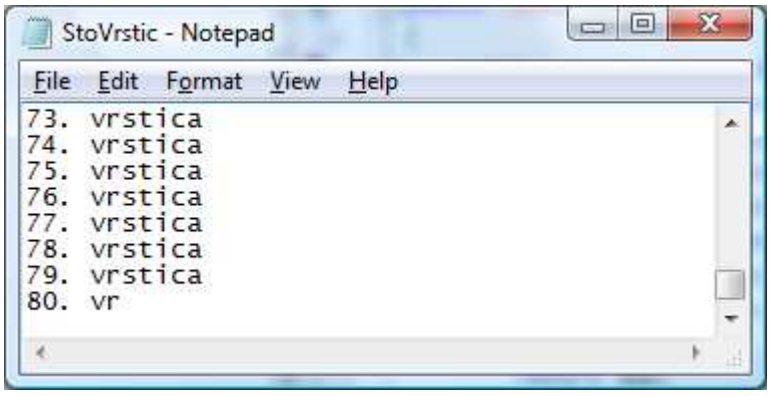
Naloga je enostavna:

- Ustvarimo datoteko StoVrstic.txt in jo povezeno z pisalnim podatkovnim tokom
- V zanki izpišemo števec in besedilo ". vrstica"

Nato ... Nič! To je vse.

```
1:      public static void Main(string[] args)
2:      {
3:        StreamWriter oznaka;
4:        oznaka = File.CreateText(@"c:\temp\StoVrstic.txt");
5:        for (int i = 1; i <= 100; i++) {
6:          oznaka.WriteLine(i + ". vrstica");
7:        }
8:      }
```

Poglejmo datoteko. Na začetku je vse lepo in prav, a kaj pa je s koncem:



20 vrstic manjka! Razlog je v tem, da smo pozabili na Close. Če torej dodamo

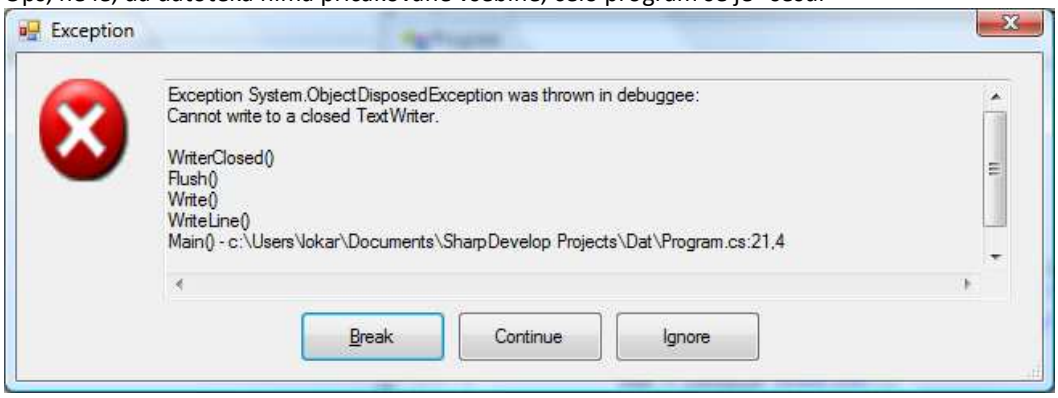
```
oznaka.Close()
```

bo datoteka StoVrstic.txt taka, kot pričakujemo.

```
1:      public static void Main(string[] args)
```

```
2:      {
3:        StreamWriter oznaka;
4:        oznaka = File.CreateText(@"c:\temp\StoVrstic.txt");
5:        for (int i = 1; i <= 100; i++) {
6:          oznaka.WriteLine(i + ". vrstica");
7:          oznaka.Close();
8:        }
9:      }
```

Ops, ne le, da datoteka nima pričakovane vsebine, celo program se je "sesul"



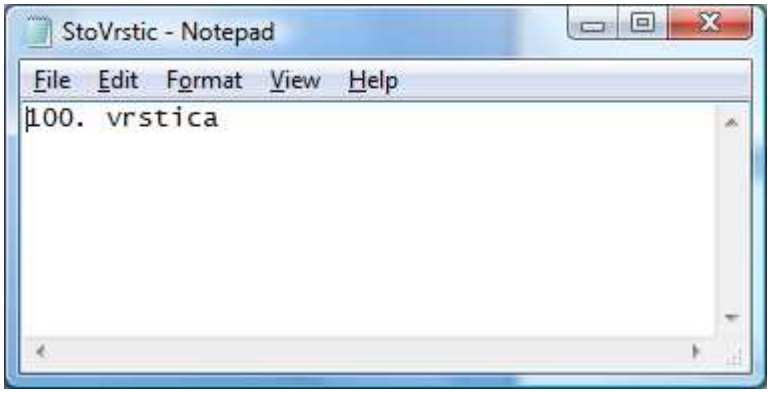
Razlog nam lepo pojasni obvestilno okno. Piše namreč:

Cannot write to a closed TextWriter.

Datoteko smo torej zaprli na napačnem mestu, v zanki, ko bi morala biti še odprta. Popravimo program

```
1:    public static void Main(string[] args)
2:      {
3:        StreamWriter oznaka;
4:        for (int i = 1; i <= 100; i++) {
5:          oznaka = File.CreateText(@"c:\temp\StoVrstic.txt");
6:          oznaka.WriteLine(i + ". vrstica");
7:          oznaka.Close();
8:        }
9:      }
```

Tako. Program sedaj deluje lepo in prav. A ko odpremo datoteko SttoVrstic.txt nas čaka presenečenje:



V datoteki je le zadnja vrstica. Seveda – vsako odpiranje datoteke pobriše staro vsebino. In zato smo sproti pobrisali to, kar smo na prejšnjem koraku napisali. No, ko program spremenimo v

```
1:    public static void Main(string[] args)
2:      {
3:        StreamWriter oznaka;
4:        oznaka = File.CreateText(@"c:\temp\StoVrstic.txt");
5:        for (int i = 1; i <= 100; i++) {
6:          oznaka.WriteLine(i + ". vrstica");
7:        }
8:        oznaka.Close();
9:      }
```

je datoteka končno taka, kot smo pričakovali.
Bralcem se za "neumne" napake seveda opravičujemo. A dejstvo je, da so prav take napake zelo pogoste v začetniških programih. Zato si jih je dobro ogledati in razumeti, zakaj se zadeva obnaša na tak način. Tako se bomo lažje izognili napakam v naših programih.
Seveda na tekstovno datoteko lahko pišemo tudi s pomočjo metode Write. V ta namen si oglejmo še en zgled.

Datoteka naključnih števil

Sestavi metodo, ki ustvari datoteko NakStev**X**.dat z n naključnimi števili med a in b. X naj bo prvo "prosto" naravno število. Če torej obstajajo datoteke NakStev1.dat, NakStev2.dat in NakStev3.dat, naj metoda ustvari datoteko NakStev4.dat.
Števila izpiši levo poravnana po k v vsaki vrsti, torej npr. kot:

```
12    134    23     22     78
167   12     1      134    45
13    9
```

Ideja:
Naša metoda bo imela 4 parametre:

- n: koliko števil bo v datoteki
- spMeja, zgMeja: meji za nakključna števila
- kolikoVVrsti: koliko števil je v vrsti

Rezultat metode bo void, saj bo metoda imela le učinek (ustvarjeno datoteko).

```
public static void UstvariDat(int n, int spMeja, int zgMeja,
                              int kolikoVVrsti)) {
```

Najprej bomo z zanko, ki bo zaporedoma preverjala, če datoteke z imeni NakStev**X**.dat obstajajo., poskrbeli, da bomo ustvarili primerno datoteko:

```
string osnovaImena = @"C:\temp\NakStev";
int katera = 1;
string ime = osnovaImena + katera + ".dat";
while (File.Exists(ime)) { // èe datoteka že obstaja
      katera++;
      ime = osnovaImena + katera + ".dat";
}
```

Nato bomo datoteko ustvarili in povezali s tokom za pisanje.

```
StreamWriter oznaka;
oznaka = File.CreateText(ime);
```

Ustvarili bomo še generator naključnih števil.

Nato naredimo zanko, ki se bo izvedla n-krat. V njej

- Ustvarimo naključno število
- Ga zapišemo na datoteko. S tabulatorjem "\t" poskrbuimo za poravnavo.
- Če smo izpisali že večkratnik kolikoVVrsti števil, gremo v novo vrsto

```
int nakStev = genNak.Next(spMeja, zgMeja);
pisiDat.Write("\t " + nakStev);
    // s tabulatorji poskrbimo za poravnavo
stevec++; // zapisali smo še eno število
if (stevec % kolikoVVrsti == 0) {
       pisiDat.WriteLine(); // zaključimo vrstico
}
```

Na koncu le še zaključimo vse skupaj:

- Po potrebi gremo še v novo vrsto
- Zapremo podatkovni tok

```
if (stevec % kolikoVVrsti != 0) {
          // če prej nismo ravno končali z vrstico
       pisiDat.WriteLine(); // zaključimo vrstico
}
pisiDat.Close();
```

Poglejmo sedaj še celotni program

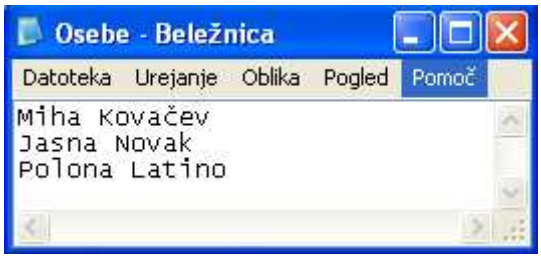
```
1:    using System;
2:    using System.IO;
3:    public class Program
4:    {
5:      public static void UstvariDat(int n, int spMeja, int zgMeja,
6:                                    int kolikoVVrsti) {
7:        string osnovaImena = @"C:\temp\NakStev";
8:        int katera = 1;
9:        string ime = osnovaImena + katera + ".dat";
10:       while (File.Exists(ime)) { // če datoteka že obstaja
11:         katera++;
12:         ime = osnovaImena + katera + ".dat";
13:       }
14:       // našli smo "prosto" ime
15:       StreamWriter pisiDat;
16:       pisiDat = File.CreateText(ime);
17:       Random genNak = new Random();
18:       int stevec = 0;
19:       while (stevec < n) {
20:         int nakStev = genNak.Next(spMeja, zgMeja);
21:         pisiDat.Write("\t " + nakStev);
22:               // s tabulatorji poskrbimo za poravnavo
23:         stevec++; // zapisali smo še eno število
24:         if (stevec % kolikoVVrsti == 0) {
25:           pisiDat.WriteLine(); // zaključimo vrstico
26:         }
27:       }
```

```
28:       if (stevec % kolikoVVrsti != 0) {
29:           // èe prej nismo ravno konèali z vrstico
30:         pisiDat.WriteLine(); // zaključèimo vrstico
31:       }
32:       pisiDat.Close();
33:     }
34:
35:     public static void Main(string[] args)
36:     {
37:       UstvariDat(32, 1, 1000, 5);
38:     }
39:   }
```

Kako v datotekah hranimo podatke

V datoteki hranimo podatke po nekem pravilu, imenovanem format zapisa, ki nam pove, kako so podatki zapisani. Pri formatu zapisa moramo biti precej pozorni, sicer se preproste zadeve hitro zapletejo. Oglejmo si primer.

Imamo datoteko v katero zapisujemo ime in priimek poljubnih oseb.



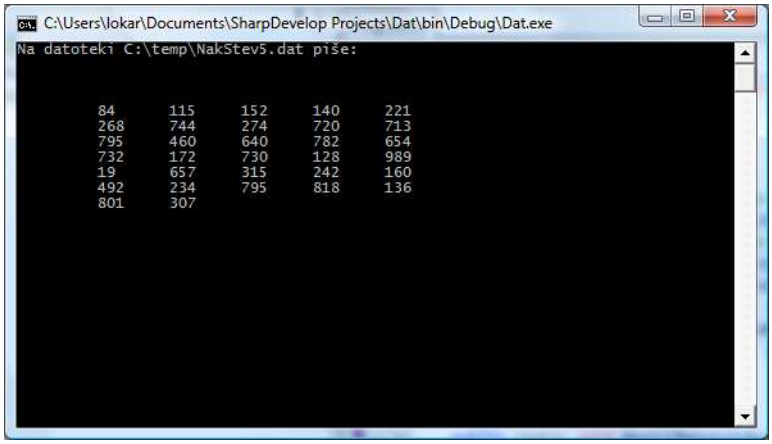
Problem nastopi, ko vpisujemo osebo, ki ima dve ali več imen. Na primer Ana Marija Težava. Kako ravnati v primeru, ko imamo med besedama Ana in Marija presledek? Sicer s samim zapison ne bo probelma. Ta pa se bo pojavil, ko bomo tak podatek brali. Ali sedaj Marija spada pod priimek in kam potem spada njen priimek Težava?

Da se izognemo takšnim težavam, je priporočljivo, da naredimo pred zapisovanjem načrt formata za obstoječe podatke. Ko delamo načrt, se držimo nekaterih pomembnih nasvetov:

- Format zapisa mora biti načrtovan tako, da ne more priti do zmede. Na primer, če so podatki med seboj ločeni z presledki, potem polj v zapisu ne smemo ločiti s presledki.
- Format zapisa mora biti tak, da programerju olajša pisanje podatkov iz programa na datoteko ter branje iz datoteke v program.
- Dobro premislimo, v kakšnem vrstnem redu naj bodo podatki zapisani.
- Pomembno je, da so zapisani podatki pregledni, da jih ljudje razumejo. Še pomembneje pa je, da so podatki shranjeni tako, da jih je lahko brati in pisati s programi.

Branje tekstovnih datotek

Poglejmo si sedaj, kako bi prebrali tisto, kar smo v prejšnjem razdelku napisali na datoteko.



Branje po vrsticah

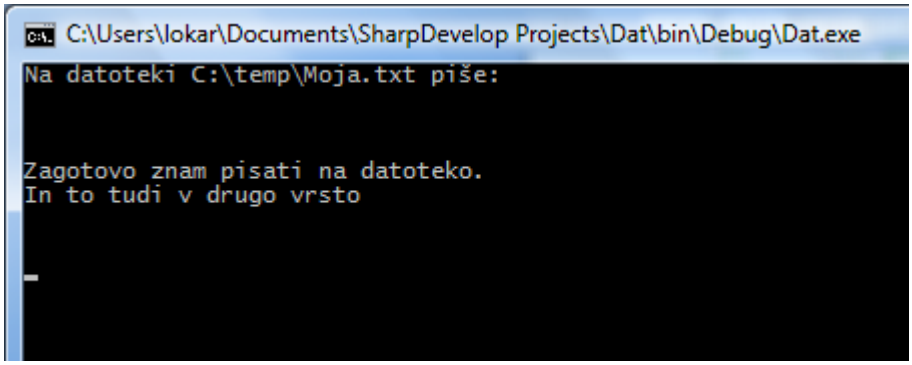
S tekstovnih datotek najležje beremo tako, da preberemo kar celo vrstico hkrati. Poglejmo si naslednjo metodo

```
1:      public static void IzpisDatoteke(string ime) {
2:        Console.WriteLine("Na datoteki " + ime + " piše: \n\n\n");
3:        // odpremo datoteko za
4:        StreamReader izhTok;
5:        izhTok = File.OpenText(ime);
6:        // preberemo prvo vrstico in jo izpišemo na zaslon
7:        Console.WriteLine(izhTok.ReadLine());
8:        // preberi z datoteke naslednji dve vrstici in ju izpiši na zaslon
9:        Console.WriteLine(izhTok.ReadLine());
10:       Console.WriteLine(izhTok.ReadLine());
11:       // preberi preostale vrstice
12:       Console.WriteLine(izhTok.ReadToEnd());
13:       // zaprimo tok
14:       izhTok.Close();
15:     }
```

Razlaga. V 2. vrstici najprej na zaslon izpišemo obvestilo in spustimo tri prazne vrstice. Nato v vrstici 4 določimo spremenjlivjko, ki bo označevala izhodni tok. V v. 5 z metodo OpenText izhodni tok povežemo z datoteko z imenom ime. Kot vidimo, podatke beremo z ReadLine(). S tem preberemo kompletno vrstico. Obstaja tudi metoda ReadToEnd(), ki prebere vse vrstice v datoteki od trenutne vrstice dalje pa do konca. Tudi datoteke s katerih beremo se spodobi zapreti, čeprav pozabljeni Close tu ne bo tako problematičen kot pri datotekah na katere pišemo.

Branje datoteke z dvema vrsticama

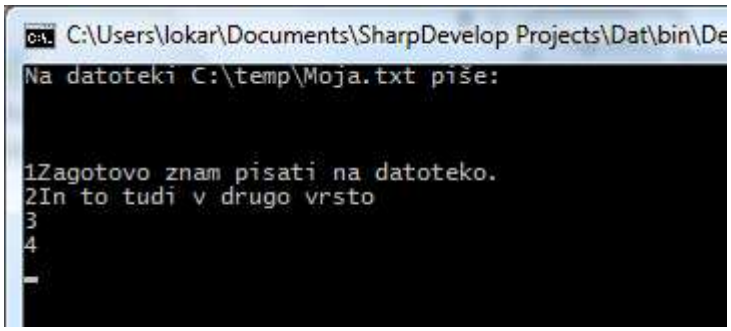
Z zgornjo metodo preberimo datoteko, ki ima le dve vrstici.



Kot vidimo, smo izpisali štiri vrstice! Kako to? Če si z Beležnico ogledamo datoteko Moja.txtx vidimo, da ima le dve vrstici. Od kje potem preostali dve? Če popravimo kodo metode tako, da se spredaj izpiše tudi ustrezna številka

```
// preberemo prvo vrstico in jo izpišemo na zaslon
Console.WriteLine(1 + izhTok.ReadLine());
// preberi z datoteke naslednji dve vrstici in ju izpiši na zaslon
Console.WriteLine(2 + izhTok.ReadLine());
Console.WriteLine(3 + izhTok.ReadLine());
// preberi preostale vrstice
Console.WriteLine(4 + izhTok.ReadToEnd());
```

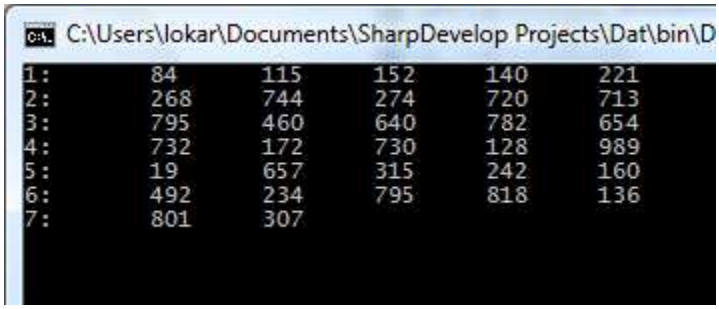
na zaslon dobimo



Namreč, če z metodo ReadLine() beremo potem, ko datoteke ni več (neobstoječe vrstice), metoda vrne kot rezultat neobstoječ niz (vrednost null). Če tak neobstoječ niz izpišemo, se izpiše kot prazen niz (""). Zato tretji klic izpisa `Console.WriteLine(3 + izhTok.ReadLine());` izpiše 3 in prazen niz. Prav tako tudi metoda ReadToEnd() vrne neobstoječ niz, če vrstic ni več.
Dejstvo, da metoda ReadLine() vrne neobstoječ niz, bomo uporabili v naslednjem zgledu.

Branje in številčenje vrstic

Denimo, da želimo prebrati neko tekstovno datoteko in jo izpisati z oštevilčenimi vrsticami. Npr. takole:



Ali pa, če je vsebina datoteke `Vrba.txt`

```
O Vrba! srečna, draga vas domača,
kjer hiša mojega stoji očeta;
de b' uka žeja me iz tvojga sve'ta
speljala ne bila, goljfiva kača!

Ne vedel bi, kako se v strup prebrača
vse, kar srce si sladkega obeta;
mi ne bila bi vera v sebe vzeta,
ne bil viharjov no'tranjih b' igrača!

Zvesto' srce in delavno ročico
za doto, ki je nima miljonarka,
bi bil dobil z izvoljeno devico;

mi mirno plavala bi moja barka,
pred ognjam dom, pred točo mi pšenico
bi bližnji sosed va'roval - svet' Marka.
```

potem naj program izpiše

```
1       O Vrba! srečna, draga vas domača,
2       kjer hiša mojega stoji očeta;
3       de b' uka žeja me iz tvojga sve'ta
4       speljala ne bila, goljfiva kača!
5
6       Ne vedel bi, kako se v strup prebrača
7       vse, kar srce si sladkega obeta;
8       mi ne bila bi vera v sebe vzeta,
9       ne bil viharjov no'tranjih b' igrača!
10
11      Zvesto' srce in delavno ročico
12      za doto, ki je nima miljonarka,
13      bi bil dobil z izvoljeno devico;
14
15      mi mirno plavala bi moja barka,
16      pred ognjam dom, pred točo mi pšenico
17      bi bližnji sosed va'roval - svet' Marka.
```

Glavni "igralec" tukaj bo zanka, ki bo izpisala posamezno vrstico in prebrala naslednjo vrstico. To bomo počeli toliko časa, dokler prebrana vrstica ne bo neobstoječ niz.

```
while (vrstica != null) {
  Console.WriteLine(stevec + ": " + vrstica);
  vrstica = izhTok.ReadLine();
```

```
    stevec++;
  }
```

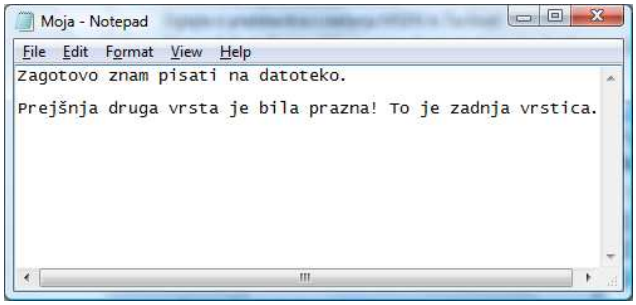
In še celotna metoda

```
public static void IzpisDatoteke2(string ime) {
  StreamReader izhTok;
  izhTok = File.OpenText(ime);
  // preberemo prvo vrstico in jo izpišemo na zaslon
  string vrstica = izhTok.ReadLine();
  int stevec = 1;
  while (vrstica != null) {
    Console.WriteLine(stevec + ": " + vrstica);
    vrstica = izhTok.ReadLine();
    stevec++;
  }
  // zaprimo tok
  izhTok.Close();
}
```

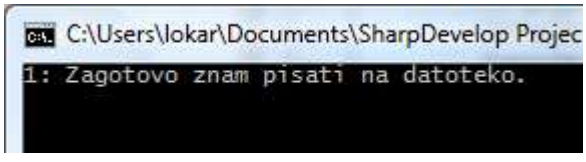
Moramo pa paziti. Če bi namreč zanko napisali kot

```
  while (vrstica != "") {
    Console.WriteLine(stevec + ": " + vrstica);
    vrstica = izhTok.ReadLine();
    stevec++;
  }
```

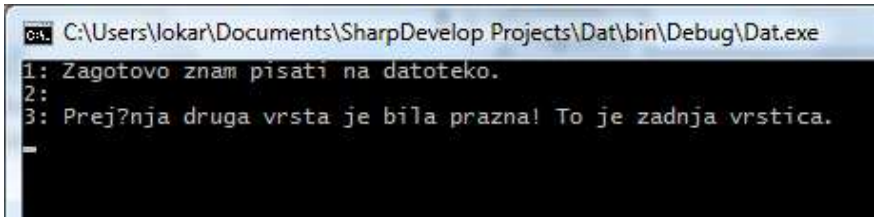
bi npr. pri datotekah, ki vsebujejo kakšno prazno vrstico, dobili izpisane vrstice le do te prazne vrstice. Npr. če metodo izvedemo nad



dobimo pri spremenjeni kodi



namesto



Pogosto bomo metodo za branje datoteke po vrsticah videli zapisano v obliki

```
public static void IzpisDatoteke1(string ime) {
  StreamReader s = File.OpenText(ime);
  string beri = null;
  while ((beri = s.ReadLine()) != null)
  {
    Console.WriteLine(beri);
  }
  s.Close();
}
```

"Čudna" je le oblika `while ((beri = s.ReadLine()) != null)`. Gre za uporabo dejstva, da C# pozna tudi tako imenovan **prireditveni izraz**. To je "izraz z učinkom".
Če napišemo

```
x = 3;
```

je to prireditveni stavek, ki v spremenljivko x shrani 3. Če pa napišemo

```
x = 3;
```

(torej brez podpičja), gre za izraz. Ta izraz ima enak učinek, kot prireditveni stavek (torej v x shrani 3), poleg tega pa ima tako kot vsi izrazi tudi vrednost. Vrednost prireditvenega izraza je tista vrednost, ki se je priredila. V našem primeru torej 3. Z uporabo prireditvenih stavkov lahko napišemo npr.

```
a = b = 0;
```

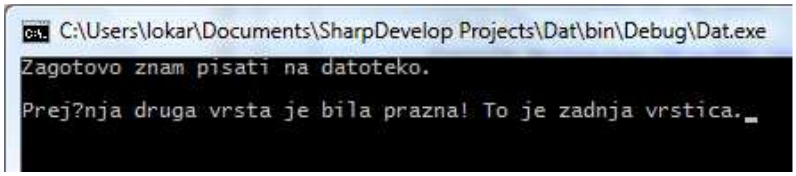
Zakaj gre? Napisali smo prireditveni stavek, ki v spremenljivko a shrani ... kaj? Vrednost izraza, kaj pa drugega. In kaj je izraz – gre za prireditveni izraz (b = 0). Ta v b shrani 0 (ima učinek) in kot svoj rezultat vrne vrednost, ki smo jo shranili v b, torej 0. V a se bo torej tudi shranila 0.
Sedaj lahko razumemo, zakaj gre pri `beri = s.ReadLine()) != null` To je pogojni izraz (torej nekaj, kar ima vrednost true ali false), ki sprašuje, ali je nekaj različno (!=) od null. Tisto nekaj pa je vrednost prireditvenega izraza `beri = s.ReadLine()`. Vrednost tega izraza je tisto, kar se shrani v spremenljivko `beri`, torej to, kar prebere metoda `ReadLine`. Z (beri = s.**ReadLine**()) torej preberemo vrstico iz vhodnega toka s. Prebrano shranimo v spremeljivko beri in zraven še primerjamo, če je ta shranjena vrednost različna od null.
Ko se torej preverja pogoj `((beri = s.ReadLine()) != null)` sta dve možnosti. Če v vhodnem toku ni več vrstic, bomo v `beri` shranili vrednost null in pogoj bo imel vrednost `false`. Če pa vrstice še obstajajo, bomo tekočo vrstico prebrali in shranili v `beri`. Pogoj pa se bo torej izračunal v `true` (in se bo zanka nadaljevala). Ko smo torej v telesu zanke, smo ravno prebrali tekočo vrstico, ki obstaja (ni `null`). Zanka pa se preneha izvajati, ko ravno "preberemo" neobstoječo vrstico (`ReadLine` vrne `null`, kar shranimo v `beri`).

Branje po znakih

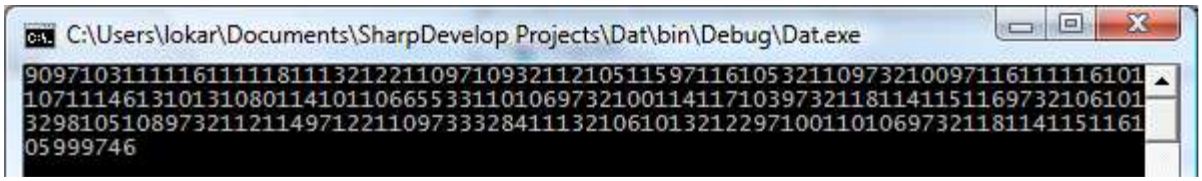
Tekstovne datoteke lahko beremo tudi znak po znak. Poglejmo si kar metodo, ki datoteko znak po znak prepiše na zaslon.

```
public static void IzpisDatotekePoZnakih(string ime) {
  StreamReader s = File.OpenText(ime);
  int beri;
  beri = s.Read();
  while (beri != -1) // konec datoteke
  {
    Console.Write((char)beri); // izpisujemo ustrezne znake
    beri = s.Read();
  }
  s.Close();
}
```

Ko beremo datoteko po znakih uporabljamo metodo Read. Ta nam vrne kodo znaka, ki ga preberemo. Ko bomo naleteli na konec datoteko, bo prebrana koda -1, ki jo uporabimo za to, da vemo, kdaj smo pri koncu.
Seveda ga moramo pri izpisu lepo pretvoriti v znak, saj bi drugače namesto



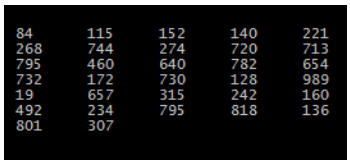
dobili (če bi bila 7. vrstica torej `Console.Write(beri);`)



Če verjamete ali ne, gre za isto datoteko, le da so druga poleg druge izpisane kode znakov namesto njihovih grafičnih podob.

Iz vrstic izlušči posamezne znake

Kako iz datoteke, kjer je v vrstici več števil,



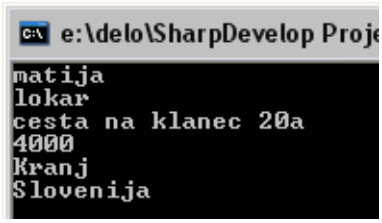
dobiti posamezna števila.
V C# imamo na voljo metodo `split`, ki zna niz razdeliti na posamezne dele. Ta del kode bo povedal vse:

```
string info = "matija;lokar;cesta na klanec 20a;4000;Kranj;Slovenija";
string[] tabInfo;
//določimo znake, ki predstavljajo ločila med podatki
char[] locila  = {';'}; // mi smo za ločilo vzeli le ;

tabInfo = info.Split(locila);
for(int x = 0; x < tabInfo.Length; x++)
```

```
    {
      Console.WriteLine(tabInfo[x]);
    }
```

Če izvedemo to kodo, dobimo 6 vrstic, saj je v nizu pet ločil ;.



Če pa bi med ločila dodali še npr. presledek

```
char[] locila  = {';', ' '}; // locili sta presledek in ;
```

bi dobili dodatne tri vrstice

Seveda pa moramo paziti. Če npr. niz spremenimo v (med matija in ; smo dodali presledek, prav tako smo med cesta in na dodali dodatni presledek

```
string info = "matija ;lokar;cesta  na klanec 20a;4000;Kranj;Slovenija";
```

bomo dobili še dve vrstici

Poznati je potrebno strukturo datoteke, da znamo določiti separatorski znak (ločila)! V našem primeru bosta to presledek in tabulatorski znak ('\t'), ker bo datoteka tista, ki jo ustvari metoda **`UstvariDat`**, ki smo jo napisali prej. Ker pa je med števili lahko več teh ločil, bomo prazne nize prezrli!

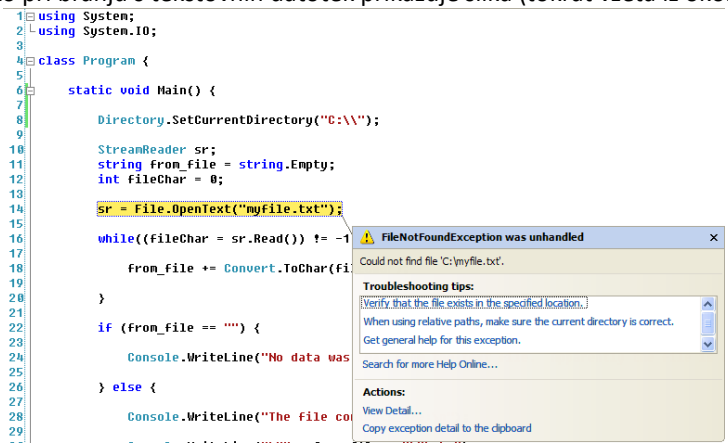
Predpostavimo spet, da v datoteki ni več kot 100 števil.

```
public static int[] IzDatStevilTabela(string vhod){
```

```
    StreamReader beri = File.OpenText(vhod);
    int[] tabela = new int[100];
    int koliko = 0;
    string vrst = beri.ReadLine();
    while(vrst != null){
      // vrstico moramo predelati v tabelo posameznih števil
      string[] tabInfo;
      //določimo znake, ki predstavljajo ločila med podatki
      char[] locila  = {' ', '\t'}; // ločilo je presledek in tabulator
      tabInfo = vrst.Split(locila);
      for(int x = 0; x < tabInfo.Length; x++)
      {
        if (tabInfo[x] != "") { // èe nismo dobili le prazni niz
          tabela[koliko] = int.Parse(tabInfo[x]);
          koliko++;
        }
      }
      vrst = beri.ReadLine();
    }
    beri.Close();
    return tabela;
  }
```

Datoteke ni

Najbolj pogosto napako pri branju s tekstovnih datotek prikazuje slika (tokrat vzeta iz okolja Visual Studio)



Gre za to, da smo z OpenText odpirali datoteko, ki je ni! A to že znamo preprečiti. Spomnimo se metode `File.Exists(ime)`, ki smo jo v razdelku o pisanju na datoteke uporabili zato, da nismo slučajno ustvarili datoteke z enekim imenom, kot jo ima že obstoječa datoteka (ker bi staro vsebino s tem izgubili). Torej je smiselno, da pred odpiranjem datoteke to preverimo. Popravimo eno od prejšnjih metod

```
1:    public static void IzpisDatoteke4(string ime) {
2:       if (!File.Exists(ime)) {
3:         Console.WriteLine("Datoteka " + ime + " ne obstaja!");
4:       } else {
5:         StreamReader izhTok;
6:         izhTok = File.OpenText(ime);
7:         // preberemo prvo vrstico in jo izpišemo na zaslon
```

```
8:          string vrstica = izhTok.ReadLine();
9:          int stevec = 1;
10:         while (vrstica != "") {
11:           Console.WriteLine(stevec + ": " + vrstica);
12:           vrstica = izhTok.ReadLine();
13:           stevec++;
14:         }
15:         // zaprimo tok
16:         izhTok.Close();
17:       }
18:     }
```

Ponovitev

Ponovimo pomembne metode, ki jih srečamo pri delu s tekstovnimi datotekami:

- `StreamReader`: Tip spremenljivke za branje toka.
- `StreamWriter`: Tip spremenljivke, ki ga uporabimo pri spremenljivkah, v katerih je shranjena oznaka podatkovnega toka.
- `File.Exists()`: Metoda, ki jo uporabimo zato, da preverimo obstoj datotek.
- `File.OpenText()`: Metoda, ki izhodni tok poveže z datoteko, s katero delamo.
- `File.CreateText()`: Metoda, ki ustvari datoteko in vrne oznako podatkovnega toka na katerega pišemo.
- `ReadLine()`: Metoda, ki prebere celo vrstico v datoteki.
- `ReadToEnd()`: Metoda, ki prebere vse vrstice v datoteki od trenutne vrstice pa do konca.
- `Read()`: Metoda, ki nam vrne kodo znaka, ki ga preberemo iz datoteke.
- `WriteLine() in Write()`: Metodi, ki zapišeta dani niz na datoteko
- `Close()`: Metoda, s katero zapremo datoteko.

Zgledi

Prepiši datoteko

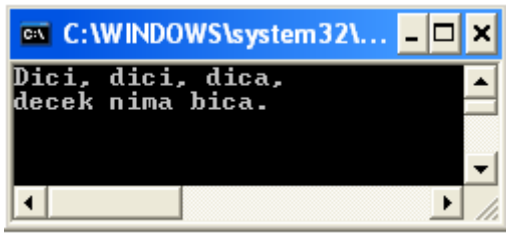
Podano imaš datoteko Decek.txt, ki vsebuje le dve vrstici. Napiši metodo, ki za vhodni podatek sprejme ime datoteke in na zaslon izpiše dani vrstici.

```
1:    using System;
2:    using System.IO;
3:    namespace Branje
4:    {
5:      class Branje
6:      {
7:        public static void IzpisNaZaslon1(string ime){
8:           if(!File.Exists(ime)){
9:                Console.WriteLine("Datoteka ne obstaja");
10:          } else{
11:                 StreamReader dat;
12:                 dat = File.OpenText(ime);
```

```
13:                 Console.WriteLine(dat.ReadLine());
14:                 Console.WriteLine(dat.ReadToEnd());
15:                 dat.Close();
16:             }
17:         }
18:     }
19: }
```

- V 1. vrstici napovemo uporabo imenskega prostora, ki vsebuje osnovne tipe (int, double, …)
- V 2. vrstici vidimo ukaz using System.IO, s katerim C# povemo, da bomo uporabili razrede in metode iz imenskega prostora System.IO. Ta imenski prostor vsebuje vse potrebno za delo z datotekami.
- V 8. vrstici vidimo, da z metodo File.Exists() preverimo, ali dana datoteka sploh obstaja. Zelo smiselno je, da pred odpiranjem datotek preverimo, če datoteka obstaja. Če želene datoteke ni, se program sicer sesuje. V primeru, da datoteka obstaja, ukaz File.Exists() vrne true, sicer false.
- V 9. vrstici izpišemo opozorilo, ki se izpiše kadar iskana datoteka ne obstaja.
- V 11. vrstici vidimo, kako določimo spremenljivko, ki bo označevala izhodni tok.
- V 12. vrstici z metodo File.OpenText() povežemo izhodni tok z datoteko z imenom ime.
- V 13. vrstici z metodo ReadLine() preberemo celotno vrstico in jo z ukazom Console.WriteLine() izpišemo na zaslon.
- V 14. vrstici smo za branje uporabili tudi metodo ReadToEnd(). Ta prebere vse vrstice v datoteki od trenutne vrstice dalje (torej od druge) pa do konca. V našem primeru bi bilo povsem vseeno, če bi povnovno uporabili kar metodo ReadLine(), saj je v datoteki le še ta vrstica.
- V 15. vrstici vidimo, da na koncu datoteko zapremo z metodo Close(). To naredimo vedno, ko končamo z uporabo datoteke.

Rezultat izpisa:



Opozorili:

- V splošnem ne vemo vnaprej, koliko vrstic ima podana datoteka.
- Če z metodo ReadLine() ali ReadToEnd() beremo potem, ko datoteka nima več obstoječih vrstic (trenutne vrstice ni več, oziroma smo na koncu datoteke), metoda kot rezultat vrne null (neobstoječ niz). Neobstoječi niz je ob izpisu na zaslon viden kot prazen niz (prazna vrstica).

Odstrani prazne vrstice

Na datoteki Dekle.txt je pesem zapisana po kiticah. To pomeni, da so med kiticami prazne vrstice. Napiši metodo, ki bo za dano ime datoteke vse kitice združila in jih izpisala na zaslon.

Vsebina datoteke Dekle.txt:	Izpis na zaslon:
Dekle je po vodo šlo *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*	*Dekle je po vodo šlo* *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*

Koda metode:

```
public static void IzpisNaZaslon(string ime)
      // preverimo, če datoteka obstaja
      if (!File.Exists(ime)) {
           Console.WriteLine("Datoteka ne obstaja");
      } else {
         // odpremo datoteko za branje
         StreamReader dat = File.OpenText(ime);
         //pomožna spremenljivka,ki nam bo predstavljala eno vrstico
         string vrsta = dat.ReadLine();
             // zanka, ki teče dokler je še kakšen obstoječ niz
             while(vrsta != null){
               // če niz ni prazen (ni šlo za prazno vrstico) ga izpišemo
               if(vrsta.Length != 0){
                  Console.WriteLine(vrsta);
               }
               //preberemo novo vrstico
               vrsta = dat.ReadLine();
         }
             //zapremo datoteko
             dat.Close();
          }
      }
```

Glavna "trika" v tej metodi sta:

- zanka, ki se izvaja toliko časa, dokler je v datoteki še kakšna neprebrana vrstica oziroma obstoječ niz.

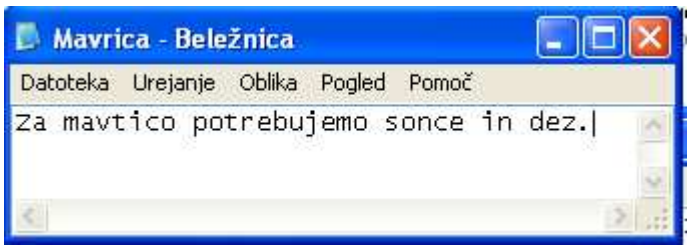
```
while(vrsta != null){…}
```

- pogojni stavek, ki pregleduje, ali je prebrana vrstica prazna (brez znakov). V primeru, da je vrstica polna, jo izpišemo. Pri preverjanju polnosti vrstice se spomnimo, da so vrstice v resnici nizi. Za nize pa vemo, da je niz prazen, če je dolžina niza enako 0.

```
if(vrsta.Length != 0) {…}
```

Izpiši datoteko na zaslon znak po znak

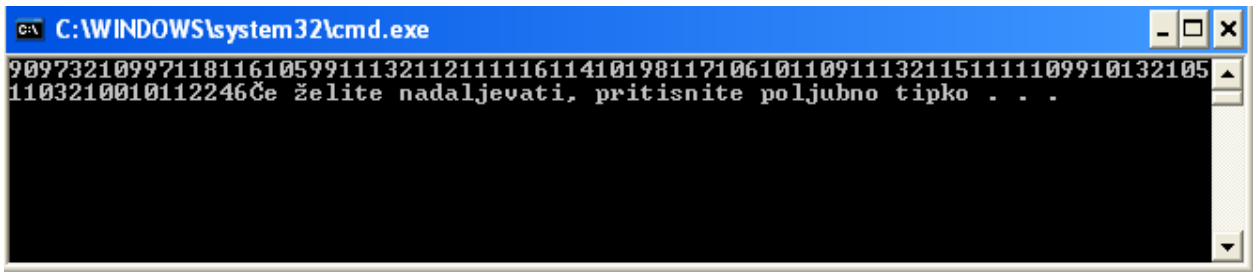
Podano imaš datoteko Mavrica.txt. Napiši metodo, ki na zaslon izpiše besedilo datoteke. Besedilo prepiši na zaslon tako, da pišeš znak po znak na zaslon.



Koda metode:

```
public static void IzpisNaZaslon(string ime){
      if(!File.Exists(ime)){
          Console.WriteLine("Datoteka ne obstaja");
      }else{
          StreamReader dat = File.OpenText(ime);
          int beri = dat.Read();
          while(beri != -1){
            Console.Write((char)beri);
            beri = dat.Read();
          }
             dat.Close();
           }
       }
```

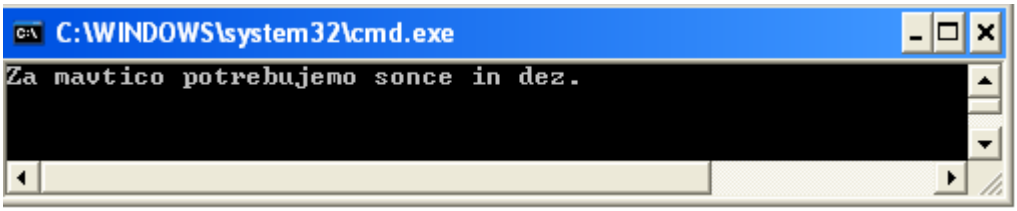
Glavni trik v tej metodi je uporaba metode `Read()`, ki jo uporabimo za branje po znakih. Značilnost te metode je, da znak prebere v obliki njegove kode. Primer datoteke Mavrica zapisane v kodi:



Tak izpis bi dobili, če bi vrstico 8 napisali kot

```
Console.Write(beri);
```

Seveda te kode navadni smrtniki ne znamo brati. Zato moramo besedilo pri izpisu prevesti nazaj v znake. Primer datoteke Mavrica zapisan v obliki, ki smo jo vajeni:



Opomba: Kako računalnik ve, kdaj je konec zapisa na datoteki? Zadnji znak datoteke je oznaka EOF, katere koda je -1. Ko torej preberemo kodo -1 vemo, da smo na koncu besedila zapisanega v datoteki.

Kopija datoteke (po vrsticah)

Sestavimo metodo, ki bo naredil kopijo datoteke.

1. Najprej bomo odprli dve datoteki. Prvo za branje (File.OpenText), drugo za pisanje File.CreateText)
2. potem bomo brali s prve datoteke vrstico po vrstico (metoda ReadLine) in jih sproti zapisovali na izhodno datoteko.
3. To bomo počeli tako dolgo, dokler prebrani niz ne bo null.

```
public static void Kopija(string vhod, string izhod){
    StreamReader beri = File.OpenText(vhod);
    StreamWriter pisi = File.CreateText(izhod);
    string vrst = beri.ReadLine();
    while(vrst != null){
      Console.WriteLine(vrst);
      pisi.WriteLine(vrst);
      vrst = beri.ReadLine();
    }
    beri.Close();
    pisi.Close();
  }
```

Kopija datoteke (po znakih)

Naredimo sedaj enako, le da tokrat kopirajmo datoteko znak po znak. Postopek bo enak, le da bomo brali z Read in konec preverjali z kodo znaka -1. Pravzaprav, če dobro premislimo – metoda bo v bistvu kombinacija prejšnje metode in metode izpisDatotekePoZnakih, kjer smo datoteko po znakih izpisali na zaslon. Sedaj bomo počeli enako, le da bomo namesto na zaslon pisali na datoteko.

```
public static void KopijaDatotekePoZnakih(string vhod, string izhod){
    StreamReader beri = File.OpenText(vhod);
    StreamWriter pisi = File.CreateText(izhod);
    int prebranZnak;
    prebranZnak = beri.Read();
    while (prebranZnak!= -1) // konec datoteke
    {
      pisi.Write((char)prebranZnak); // izpisujemo ustrezne znake
      prebranZnak = beri.Read();
    }
    beri.Close();
    pisi.Close();
  }
```

Datoteka s števili (vsako v svoji vrsti), ki jih preberemo v tabelo

Dana je datoteka, na kateri so zapisana cela števila. Vemo, da na datoteki ni več kot 100 števil. Vsako število je v svoji vrstici. Sestavimo metodo, ki za danio ime datoteke vrne tabelo števil.

```
public static int[] IzDatTabela(string vhod){
  StreamReader beri = File.OpenText(vhod);
  int[] tabela = new int[100];
  int koliko = 0;
  string vrst = beri.ReadLine();
  while(vrst != null){
    tabela[koliko] = int.Parse(vrst);
    koliko++;
    vrst = beri.ReadLine();
  }
  beri.Close();
  return tabela;
}
```

Kaj če ne vemo koliko je števil:

- najprej en prehod preko datoteke in preštejemo število vrstic

Kaj, če želimo vrniti ravno tabelo "idealne" velikosti, torej točno tako veliko, kot je minimalno potrebno (kot je število vrstic v datoteki)

- Če ne vemo, koliko je podatkov – rešitev prej
- Če vemo, koliko jih je največ:
  - beremo kot prej
  - ko smo naredili tabelo, naredimo novo tabelo ustrezne velikosti in prepišemo le ustrezni del

Zgled: zapis tabele na datoteko

Sestavimo metodo, ki tabelo celih števil prepiše v datoteko, določeno z nizom, ki je tudi parameter metode. Vsak element tabele naj bo zapisana v svoji vrstici.
Primer tabele:

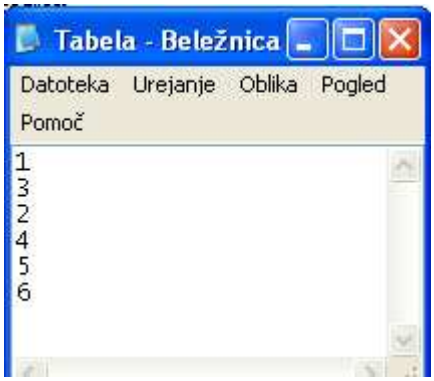
$$[1 \quad 3 \quad 2 \quad 4 \quad 5 \quad 6]$$

Koda metode:

```
using System;
using System.IO;
namespace Pisanje{
class Pisanje{
     public static void IzpisovanjeNaDatoteko(int[] m, string ime ){
         // preverimo, če že obstaja datoteka z enakim imenom
         if(File.Exists(ime)){
            Console.WriteLine("Datoteka s takšnim imenom že obstaja");
         }else{
         // odpremo datoteko v katero bomo zapisovali podatke
         StreamWriter oznaka = File.CreateText(ime);
         // sprehodimo se čez tabelo
         for(int i = 0; i< m.Length; i++){
             //posamezni element tabele zapišemo v datoteko in to v svojo vrsto
```

```
            oznaka.WriteLine(m[i]);
        }
        // zapremo datoteko po uporabi
        oznaka.Close();
    }
  }
}
```

Prikaz podatkov iz datoteke Tabela.txt:



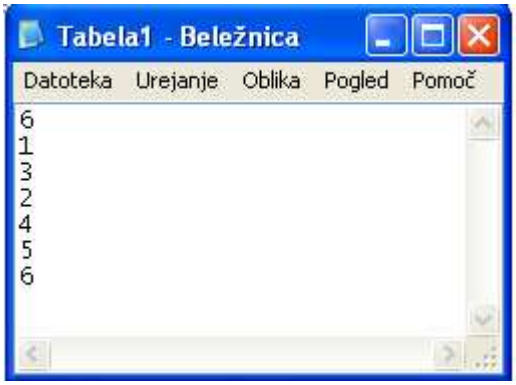
Napisano imamo datoteko z imenom Tabela.txt. Ali sedaj znamo zapisati te iste podatke iz datoteke nazaj v tabelo? Sicer bi šlo tako, kot smo si ogledali v prejšnjem zgledu. Najprej napišemo metodo, ki vrne število vrstic naše datoteka. To metodo potem uporabimo zato, da ustvarimo primerno veliko tabelo in se potem lotimo branja. Toda ta način ni dober za primere, ki imajo po več sto vrstic, saj bi bil porabljen čas prevelik.

Boljši način je, da na kaj takega kot je naknadno branje, pomislimo vnaprej in spremenimo format zapisa. V prvo vrstico datoteke zapišemo velikost tabele. V ostalih vrsticah pa zapišemo elemente tabele tako, kot smo jih v prvem primeru.

Popravljena metoda:

```
public static void IzpisovanjeNaDatoteko(int[] m, string ime ){
     if(File.Exists(ime)){
        Console.WriteLine("Datoteka s takšnim imenom že obstaja");
     }else{
        StreamWriter oznaka = File.CreateText(ime);
     // V prvo vrstico datoteke vpišemo velikost tabele
     oznaka.WriteLine(m.Length);
     for(int i = 0; i< m.Length; i++){
        oznaka.WriteLine(m[i]);
     }
     oznaka.Close();
   }
```

Ponoven prikaz datoteke Tabela1.txt:



Ponovitev branje iz tekstovne datoteke

Podatke na datoteke shranjujemo za kasnejšo uporabo. Če želimo podatke iz tekstovne datoteke prebirati, moramo datoteko odpreti v načinu za branje. To nalogo v jeziku C# prevzeme stavek

```
StreamReader vhodTok = File.OpenText(ime_datoteke);
```

Pri odpiranju datotek za branje pogosto navedemo ime neobstoječe datoteke. To povzroči napako med izvajanjem programa. Napako znamo preprečiti. Spomnimo se izraza `File.Exists(ime_datoteke`. V razdelku o pisanju smo ga uporabili zato, da nismo ustvarili datoteke z enakim imenom, kot jo ima že obstoječa datoteka. Torej je tudi pred odpiranjem datoteke za branje smiselno, da prej preverimo, ali datoteka z določenim imenom obstaja.

Da iz datoteke preberemo podatke, so na voljo metode, ki so prikazane v spodnji tabeli. Najdemo jih v razredu `StreamReader`.

C#	Razlaga
ReadLine()	Prebere naslednjo vrstico podatkov z datoteke in jo vrne kot niz.
Read()	Prebere naslednji razpoložljiv znak z datoteke. ***Pozor:*** Metoda vrne kodo prebranega znaka.
ReadToEnd()	Prebere podatke v datoteki od trenutnega mesta v datoteki pa do konca. Podatke vrne kot niz. Navadno to metodo uporabljamo, da preberemo celotno vsebino datoteke.
Peek()	Vrne naslednji znak v datoteki, brez premika na naslednji znak. Če ni na voljo nobenega znaka več, metoda vrne vrednost `-1`.

Odstranimo prazne vrstice

Na datoteki je pesem zapisana po kiticah. Med kiticami so prazne vrstice. Napišimo metodo, ki za dano ime datoteke vse kitice izpiše na zaslon. Pri tem prazne vrstice izpusti.

Vsebina datoteke `Dekle.txt`:	Izpis na zaslon:
Dekle je po vodo šlo *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*	*Dekle je po vodo šlo* *na visoke planine.* *Vodo je zajemala,* *je ribico zajela.* *Ribica jo je prosila:* *oj, pusti me živeti.* *Dekle b'la je usmiljena,* *je ribico spustila.* *Ribica je zaplavala,* *je dekle poškropila.*

Koda:

```
C1:   public static void Dekle(string ime){
C2:      // Preverimo obstoj datoteke
C3:      if(!File.Exists(ime)){
C4:         Console.WriteLine("Datoteka " + ime + "ne obstaja.");
C5:         return;
C6:      }
C7:
C8:      // Odpremo datoteko za branje
C9:      StreamReader dat = File.OpenText(ime);
C10:
C11:     // Beremo in izpisujemo (neprazne) vrstice datoteke
C12:     while(dat.Peek() != -1){ // Beremo, dokler ne preberemo kode oznake EOF
C13:        string vrstica = dat.ReadLine();
C14:        if(vrstica.Length != 0){ // Prebrana vrstica ni prazna
C15:          Console.WriteLine(vrstica);
C16:        }
C17:     }
C18:
C19:     // Zapremo datoteko za branje
C20:     dat.Close();
C21:  }
```

Razlaga. Najprej preverimo obstoj datoteke (C3). Če datoteka ne obstaja, izpišemo obvestilo (C4) in prekinemo izvajanje metode (C5). Odpremo datoteko za branje (C9). V vrstici C12 naredimo zanko, ki ponavlja vrstice od C13 do C15 toliko časa, dokler ne pridemo do konca datoteke. To prepoznamo po tem, da je na vrsti za branje znak EOF[2]. Ta znak ima kodo -1. V C13. vrstici beremo posamezno vrstico datoteke. Če je dolžina vrstice različna od 0 (če torej vrstica ni prazna), vrstico izpišemo (C15). Po končanem branju datoteko zapremo (C20).
Zadnji znak datoteke je oznaka EOF, katerega koda je -1. Ko je na vrsti znak s to kodo, vemo, da smo na koncu besedila, zapisanega na datoteki. Oznake EOF ob odprtju datoteke ne vidimo.

Opomba: Tudi datoteke, s katerih v C# beremo, se spodobi zapreti. Sicer neposrednih posledic (kot pri pisanju) ne bo. A če bomo odprto datoteko poskusili odpreti ponovno, se bo program sesul. Prav tako odprte datoteke trošijo sistemske vire računalnika. Zato je dobro, da se navadimo datoteke, potem ko jih ne potrebujemo več, vedno zapreti.

Odstranitev števk

Sestavimo metodo `PrepisiBrez`, ki sprejme imeni vhodne in izhodne datoteke. Metoda na izhodno datoteko prepiše tiste znake iz vhodne datoteke, ki niso števke. Predpostavimo, da sta imeni datotek ustrezni (torej, da

[2] Kratica EOF pomeni End-Of-File oziroma konec datoteke.

datoteka za branje obstaja, datoteka za pisanje pa ne (oziroma da njeno morebitno prejšnjo vsebino lahko izgubimo)).

Koraki programa:

- Najprej bomo odprli ustrezni datoteki. Prvo datoteko odpremo za branje in drugo za pisanje.
- Brali bomo posamezne znake iz vhodne datoteke.
  - Če prebran znak ne bo števka, ga bomo prepisali v izhodno datoteko.
- To bomo počeli toliko časa, dokler ne bomo prebrali znaka s kodo -1 (torej znak EOF).

Zapis v C#:

```
C1:   public static void PrepisiBrez(string imeVhod, string imeIzhod){
C2:      StreamReader datZaBranje; // Ustvari podatkovni tok za branje na datoteki
C3:      datZaBranje = File.OpenText(imeVhod);
C4:      StreamWriter datZaPisanje; // Ustvari tok za pisanje na datoteko
C5:      datZaPisanje = File.CreateText(imeIzhod);
C6:
C7:      // Prepis znakov iz ene datoteke na drugo datoteko
C8:      int znak = datZaBranje.Read(); // Prebere en znak
C9:      while (znak != -1){
C10:        // Primerjamo ali je prebrani znak števka
C11:        if (!('0' <= znak && znak <= '9'))
C12:           datZaPisanje.Write("" + (char)znak); // Če ni števka, zapišemo znak
C13:        znak = datZaBranje.Read();
C14:     }
C15:
C16:     // Zapremo tokova datotek
C17:     datZaBranje.Close();
C18:     datZaPisanje.Close();
C19:  }
```

Primerjava vsebine dveh datotek

Napišimo metodo `Primerjaj`, ki sprejme imeni dveh datotek in primerja njuno vsebino. Če sta vsebini datotek enaki, metoda vrne vrednost `true`, sicer vrne `false`. Predpostavimo, da sta imeni datotek ustrezni (torej, da datoteki za branje obstajata).

Zapis v C#:

```
C1:   public static bool Primerjava(string ime1, string ime2){
C2:      // Odpremo obe datoteki za branje
C3:      StreamReader dat1 = File.OpenText(ime1);
C4:      StreamReader dat2 = File.OpenText(ime2);
C5:
C6:      // Preberemo vsebino prve in druge datoteke
C7:      string beri1 = dat1.ReadToEnd();
C8:      string beri2 = dat2.ReadToEnd();
C9:
C10:     // Zapremo obe datoteki za branje
C11:     dat1.Close();
C12:     dat2.Close();
C13:
C14:     // Primerjamo vsebini
C15:     return (beri1.Equals(beri2));
C16:  }
```

Razlaga. Najprej odpremo datoteki (C3 – C4). V niz `beri1` s pomočjo metode `ReadToEnd()` preberemo celotno vsebino datoteke `dat1` (C7). V niz `beri2` shranimo celotno vsebino datoteke `dat2` (C8). Zapremo datoteki (C11 – C12). Na koncu vrnemo rezultat, ki ga dobimo pri primerjavi niza `niz1` z nizom `niz2`.

Zamenjava

Napišimo program, ki preko tipkovnice prebere ime vhodne datoteke in ime izhodne datoteke. Nato vhodno datoteko prepiše na izhodno, pri čemer vse znake 'a' nadomesti z nizom "`apa`".

Ideja programa:

- preberemo ime vhodne in izhodne datoteke,
- preverimo obstoj datotek,

- odpremo vhodno datoteko za branje in izhodno datoteko za pisanje,
- v zanki izpisujemo znake iz vhodne datoteko v izhodno datoteko. Če je znak 'a', ga nadomestimo z "apa",
- vhodno in izhodno datoteko zapremo.

Koda:

```
C1:   using System;
C2:   using System.IO;
C3:   public class Zamenjava{
C4:      public static void Main(string[] args){
C5:         // Vnos imen datotek
C6:         Console.Write("Vnesi ime vhodne datoteke: ");
C7:         string vhod = Console.ReadLine();
C8:         Console.Write("Vnesi ime izhodne datoteke: ");
C9:         string izhod = Console.ReadLine();
C10:
C11:        // Preverjanje obstoja: vhodna mora obstajati, izhodna pa ne
C12:        if (!File.Exists(vhod) || File.Exists(izhod)) {
C13:           Console.WriteLine("Napaka v imenu datotek.");
C14:           return;
C15:        }
C16:
C17:        // Odprtje datotek
C18:        StreamReader branje = File.OpenText(vhod);
C19:        StreamWriter pisanje = File.CreateText(izhod);
C20:
C21:        // Prenos podatkov
C22:        while (branje.Peek() != -1) { // Do konca datoteke
C23:           char znak = (char)branje.Read();
C24:           if (znak == 'a') {
C25:              pisanje.Write("ap"); // Zadnji 'a' bo dodal naslednji stavek
C26:           }
C27:           pisanje.Write(znak);
C28:        }
C29:
C30:        // Zapremo za datoteke
C31:        branje.Close();
C32:        pisanje.Close();
C33:     }
C34:  }
```

Razlaga. Preberemo ime vhodne in izhodne datoteke (C6 - C9). Nato preverimo obstoj datotek (C12). Če vhodna datoteka ne obstaja ali izhodna datoteka obstaja, izpišemo obvestilo (C13) ter prekinemo izvajanje metode (C14). Datoteki odpremo za branje oziroma pisanje (C18 - C19). Nato z zanko `while` beremo posamezne znake iz vhodne datoteke in jih prepisujemo na izhodno datoteko (C22 - C28). Če je prebran znak 'a' (C24), pred njim na izhodno datoteko zapišemo še niz "ap" (C25). Znak 'a' iz vhodne datoteke na ta način zapišemo na izhodno datotako z nizom "apa". Zanko končamo, ko preberemo kodo oznake EOF. Po koncu prepisovanja znakov datoteki zapremo (C31 - C32).

Kopiranje datotek

Sestavimo metodo `public static void Kopiranje(string vhod, string izhod)`, ki vsebino ene datoteke prepiše na drugo datoteko. Ime prve datoteke naj predstavlja prvi argument, ime druge datoteke pa naj predstavlja drugi argument. Predpostavimo, da je obstoj datotek že preverjen. Kopiranje izvedemo tako, da prepisujemo vrstico po vrstico.

Da preverimo, če smo že na koncu datoteke, lahko uporabimo tudi rezultat metode `ReadLine()`. Če omenjeno metodo uporabimo na neobstoječi vrstici (če smo torej že na koncu datoteke), nam metoda vrne vrednost `null`.

Zapis v C#:

```
C1:   public static void Kopija(string vhod, string izhod){
C2:      // Odpremo datoteko za branje in pisanje
C3:      StreamReader beri = File.OpenText(vhod);
C4:      StreamWriter pisi = File.CreateText(izhod);
```

```
C5:
C6:       // Beremo in kopiramo posamezne vrstice
C7:       string vrsta = beri.ReadLine();
C8:       while(vrsta != null){// Beremo toliko časa, dokler ne preberemo
C9:                            // vseh vrstic
C10:         pisi.WriteLine(vrsta);
C11:         vrsta = beri.ReadLine();
C12:      }
C13:
C14:      //Zapremo datoteko za branje in pisanje
C15:      beri.Close();
C16:      pisi.Close();
C17:   }
```

Razlaga. Najprej datoteki odpremo za branje oziroma pisanje (C3 - C4). Nato z zanko `while` beremo posamezne vrstice vhodne datoteke in jih prepisujemo v izhodno datoteko (C7 - C12). Zanko končamo, ko prebrane vrstice ni (ustrezni niz dobi vrednost `null`). Po končani zanki datoteki zapremo (C15 - C16).
Če naloga ne bi zahtevala prepisovanje po vrsticah, bi seveda lahko naenkrat prebrali kar celotno vsebino datoteke. Prepisovanje pa bi lahko izvedli tudi znak po znak.

Vaje

1. Sestavi metodo, ki tabelo nizov prepiše na datoteko, določeno z nizom, ki je tudi parameter metode. Vsak niz naj bo v svoji vrstici.
2. Dana je tekstovna datoteka. Izpiši jo tako, da zamenjaš sode in lihe vrstice.
3. Dana je metoda, ki naj bi preštela število vrstic v datoteki. A v njej so napake. Odpravi jih!

```
1:     public static int PrestejVrstice(string ime){
2:        StreamReader dat = File.OpenText(ime);
3:        int vrstice = 0;
4:        string vrst = dat.ReadLine();
5:        while(vrst != ""){
6:           vrstice = vrstice + 1;
7:        }
8:        dat.Close();
9:        return vrstice;
10:    }
```

4. Z metodo

```
1.  public static int PrestejZnake(string ime){
2.        StreamReader dat = File.OpenText(ime);
3.        int znaki = 0;
4.
5.        while(dat.ReadLine() != null ){
6.           znaki = znaki + dat.ReadLine().Length;
7.           //dat.readLine() je niz,
8.           // pristejemo njegoco dolzino
9.        }
10.       dat.Close();
11.       return znaki;
12.   }
```

naj bi prešteli število znakov v datoteki. Žal se metoda sicer prevede, očitno pa ne dela prav. Odpravi napake!

5. Sestavi metodo, ki "obrne" dano datoteko (zadnja vrstica postane prva, predzadnja druga …). Prepodstavi, da datoteka zagotovo nima več kot 100 vrstic.
6. Arheologi so v jami blizu Gize našli čuden papirus. Besedilo na njem je bilo videti zelo čudno, a po drugi strani zelo podobno našemu. Dolgo so poskušali vse, da bi ga razvozlali. Na koncu se je oglasil 6 letni Mihec, sin glavnega arheologa: "Oči, zakaj si pa na ta papir pisal v napačno smer?" In res. Šlo je za povsem običajni tekst, le besedilo je bilo zapisano od desne proti levi. Ker pa je nam tako besedilo malček zporno brati, pomagaj arheologom in sestavi metodo, ki za dano ime datoteke sestavi novo datoteko z ravno obrnjenim imenom in enako končnico (iz bla.txt torej naredi alb.txt, iz mojaDat.cs pa taDajom.cs). Ta nova datoteka naj ima ravno obrnjene vrstice prvotne datoteke.
7. Na spletnem naslovu http://www.ljse.si/ najdeš podatke o gibanju cen različnih vrednostnih papirjev (delnic) - glej arhiv enotnih tečajev. Izberi si delnico, poberi s spletne strani vse možne podatke o gibanju njenega tečaja. Podatke dobiš tako, da najprej izbereš delnico, v koledarju izbereš obdobje, za katere te gibanje delnice zanima (denimo 1.1.2007 - 1.11.2007). Nato klikneš na Shrani. Podatki se shranijo na datoteko v formatu CSV (comma separated values) v taki obliki: `3.10.2001;1.983,67;100` Podatki so ločeni s ;. Prvi podatek je datum, drugi SBI (slovenski borzni indeks), tretji pa vrednost delnice. Prvi in drugi podatek nas NE zanimata. Pripravi metodo, ki bo iz podatkov kot jih boš pobral s spletne strani (torej iz datoteke), ustvarila ustrezno tabelo z vrednostmi delnice.
8. Napiši metodo BrisiKomentarje, ki sprejme imeni vhodne in izhodne datoteke. Nato na izhodno datoteko prepiše tiste vrstice iz vhodne datoteke, ki se ne začnejo z znakom '%'.
9. Napiši metodo, ki prepiše datoteko imevhodna v datoteko imeizhodna. Pri tem vse znake 'a' zamenja z znaki 'e'.
10. Sestavite program, ki iz ukazne vrstice prebere pot do datoteke, jo odpre in na zaslon izpiše prvih deset vrstic njene vsebine (če je datoteka krajša, jih lahko izpiše tudi manj).
11. Napiš metodo Primerjaj, ki sprejme imeni dveh datotek in primerja njuno vsebino. Če sta datoteki enaki, metoda vrne vrednost true, sicer vrne false. Pozor: datoteki nista nujno enako dolgi. Pri tem si lahko pomagaš z naslednjo metodo, ki pa ima napake!

```
public static bool Primerjaj(string ime1, string ime2) {
  if (ime1 == ime2) return true; // gre za isto datoteko
  StreamReader dat1 = File.OpenText(ime1);
  StreamReader dat2 = File.OpenText(ime2);
  bool enaki = true;
  bool konec = false;

  while(!konec){
    string s1 = dat1.ReadLine();
    string s2 = dat2.ReadLine(); // ce je 2. datoteke ze konec,
                                 // ima s2 vrednost null
    if(!(s1.equals (s2))){
      enaki = false;
     konec = true;
    }
  }

  if(s2 != null){  // 2. datoteka je daljsa od prve
    enaki = false;
  }
  dat1.Close();
```

```
    dat2.Close();
    return enaki;
  }
```

12. Generiraj N (podatek) naključnih števil med 1 in 100. Rezultate zapiši v datoteko `rezultatN.txt`. V datoteki naj bo za vsak korak zapisana vrstica `#zap št. #vrednost (npr. 1 23)`

13. Na datoteki manekenke.dat so shranjeni podatki o velikosti manekenk. V vsaki vrsti je zapisana velikost manekenke v centimetrih (celo število), nato pa sledita ime in priimek. Polja so ločena z dvopičji. Primer datoteke:

```
179:Cindy:Crawford
182:Naomi:Campbel
185:Nina:Gazibara
180:Elle:Mac Perhson
180:Eva:Herzigova
```

Napiši program, ki prebere datoteko manekenke.dat in na zaslon izpiše višino, ime in priimek največje in najmanjše manekenke. V zgornjem primeru bi program deloval takole:

```
Najmanjsa je Cindy Crawford, ki meri 179 cm.
Najvecja je Nina Gazibara, ki meri 185 cm.
```

14. Dana je metoda PrestejBesede, ki sprejme niz znakov in vrne število besed v nizu. Za besedo šteje poljubno neprazno zaporedje znakov med dvema presledkoma. Za presledke štejemo tudi tabulator '\t' in znak za novo vrsto '\n'. Na primer, niz znakov `"Kdor ne delja, naj ne je."` ima 6 besed, niz znakov `"++ !! ax"` ima 3 besede, niz `"Ena dva,tri stiri pet"` pa ima 4 besede. (Več zaporednih presledkov seveda šteje kot en presledek.)

```
public static int PrestejBesede(string s){
  int steviloBesed=0;
  bool presledki=true; //pove, ali se trenutno nahajamo v besedi
  for(int i = 0; i < s.Length; i = i + 1){
    if(presledki){
      if(!Char.IsWhiteSpace(s[i])){
          // iz presledkov smo prisli v besedo
        steviloBesed++;
        presledki = false;
      }
    }
    else
      if(Char.IsWhiteSpace(s[i])){
        // iz besede smo prisli v presledke
      presledki = true;
    }
  }
  return steviloBesed;
}
```

S pomočjo te metode preštej število besed v pesnitvi *Krst pri Savici*, ki ga najdeš na primer na http://haka.fmf.uni-lj.si/pra-racunalnistvo-1/arhiv-2003/lekcija08/krst.txt

15. Napiš program Tezisce, ki iz datoteke z imenom "podatki.dat" prebere koordinate točk in izpiši na zaslon koordinate njihovega težišča. Koordinate vsake točke so napisane v svoji vrstici in ločne s presledkom. Na primer, datoteka

```
1.2 3.41
0.0 1.8
9.0 3.19
2.2 2.2
```

vsebuje podatke, ki predstavljajo toče s koordinatami (1.2, 3.41), (0.0, 1.8), (9.0, 3.19) in (2.2, 2.2). Za ta primer ima težišče koordinate (3.1, 2.65).
Nauk: koordinate težišča so povprečne vrednosti koordinat točk.

16. Janko in Metka sta za sporazumevanje izumila posebno kodo: Vsako črko v sporočilu sta zamenjala s črko, ki v (angleški) abecedi leži 6 črk za originalno. Menjava je seveda ciklična, tako da črko *a* zamenjata s črko *f*, črko *z* pa s črko *e*. Napiši metodi `zakodiraj` in `odkodiraj`, ki sprejmeta ime originalne in kodirane datoteke. Seveda ločil in ostalih znakov ne kodirata – pomagaj si z metodami `Char.IsLetter(char c)`, ki za dani znak pove, če je črka. `Char.IsLower(char c)`, ki za dani znak pove, če je mala črka, `Char.IsUpper(char c)`, ki za dani znak pove, če je velika črka.

17. Marko je agent pri varnostni agenciji ČUK. Poskrbeti mora, da bodo datoteke varno prišle od ene agenture do druge. V ta namen bo napisal metodo Razdeli(ime, koliko), ki iz datoteke z imenom ime.cuk ustvari koliko datotek z imenom ime_1.cuk, ime_2.cuk, ..., ime_koliko.cuk. Na prvi datoteki (ime_1.cuk) je 1, 1 + koliko, 1 + 2*koliko, … - ti znak iz vsake vrstice datoteke ime.cuk, na drugi 2, 2 + koliko, 2 + 2*koliko, … - ti znak iz vsake vrstice, …

    Prav tako je napisal še metodo Zdruzi(ime, koliko), ki izvaja obratno operacijo. A kot zakleto se je koda metod zgubila. Pomagaj Marku in metodi napiši na novo.

18. Napiši metodo, ki prepiše datoteko vhodna v datoteko izhodna. Pri tem naj vse samoglasnike podvoji.

19. Dana je tekstovna datoteka. Izpiši jo tako, da bodo vrstice zapisane v obratnem vrstnem redu.

    Primer:

Tekstovna datoteka:	Izpis:
Velik korak za človeka, mali korak za človečnost.	, akevolč az karok koleV .tsončevolč az karok ilam

    Namig: Napiši metodo, ki bo obrnila poljuben niz.

20. Sestavi metodo, ki niz prepiše na datoteko, tako da bo vsak znak niza v svoji vrstici. Niz in ime datoteke sta vhodna podatka.

21. Napiši metodo BrišiVrstice, ki sprejme ime vhodne in izhodne datoteke. Nato na izhodno datoteko prepiše tiste vrstice iz vhodne datoteke, ki se ne začnejo s samoglasnikom.

22. Poznamo imeni dveh datotek. Zanima nas, ali imata ti dve datoteki enako število znakov. Napiši metodo Primerjaj, ki vrne true, če je število znakov enako, sicer false.

23. Napiši metodo Potroji, ki sprejme ime vhodne in izhodne datoteke. Nato na izhodno datoteko prepiši vse vrstice vhodne datoteke in sicer tako, da se bo vsaka vrstica zaporedoma potrojila.

    Primer:

Vhodna datoteka:	Izhodna datoteka:
Velik korak za človeka, mali korak za človečnost.	Velik korak za človeka, Velik korak za človeka, Velik korak za človeka, mali korak za človečnost.

	mali korak za človečnost. mali korak za človečnost.

24. Dana je tekstovna datoteka. Izpiši vsa števila, ki se pojavijo v njej.
25. Tekstovna datoteka naj bo sestavljena iz vrst, napolnjenih s celimi števili, ločenimi s presledki. Sestavi statično metodo MinMax(ime datoteke), ki vrne maksimum od minimumov števil v vsaki vrsti v datoteki.

    Primer:
    1 2 3 4
    5 2
    0 1 2 -1

    Rezultat primera: 2
26. Napišite program, v katerem napolnite tekstovno datoteko s 100 števili. Ta števila potem preberite iz datoteke in izpišite povprečje vseh lihih števil.
27. Napišite program, v katerem v tekstovni datoteki z besedilom zamenjate vsak samoglasnik z * in tvorite novo datoteko s tako zamenjanimi črkami.
28. Napiši metodo, ki za vhodni podatek sprejme ime vhodne in izhodne datoteke. Metoda naj prepiše vrstice v vhodni datoteki na izhodno datoteko, pri čemer združi vse vrstice v eno vrstico. Na primer, če je na datoteki Dat1.txt zapisano

```
prva vrstica,
druga vrstica,
tretja vrstica,
četrta vrstica,
peta vrstica.
```

    potem je zapis na datoteki Dat2.txt

```
prva vrstica, druga vrstica, tretja vrstica, četrta vrstica, peta
vrstica.
```

29. Napiši program Tezisce, ki iz datoteke z imenom "Podatki.txt" prebere koordinate točk in izpiše na zaslon koordinate njihovega težišča. Koordinate vsake točke so napisane v svoji vrstici in ločene s presledkom. Na primer, datoteka

```
1.2 3.41
0.0 1.8
9.0 3.19
2.2 2.2
```

    vsebuje podatke, ki predstavljajo točke s koordinatami (1.2, 3.41), (0.0, 1.8), (9.0, 3.19) in (2.2, 2.2). Za ta primer ima težišče koordinate (3.1, 2.65).

30. Na datoteki Manekenke.txt so shranjeni podatki o velikosti manekenk. V vsaki vrsti je zapisana velikost manekenke v centimetrih (celo število), nato pa sledita ime in priimek. Polja so ločena s presledki. Primer datoteke:

```
179 Cindy Crawford
182 Naomi Campbel
185 Nina Gazibara
180 Elle Mac Perhson
180 Eva Herzigova
```

Napiši metode, ki za vhodni podatek sprejmejo ime metode, kot izhodni podatek pa nam vrnejo:

- Število, ki nam pove koliko manekenk je v datoteki.
- Povprečno telesno višino manekenk.
- Višino tiste manekenke, ki ima najdaljše ime.
- Vse priimke manekenk (pozor, priimek četrte manekenke je Mac Perhson!)

Rešitve za podani primer: 5, 181.2, 179, "Crawford, Campbel, Gazibara, Mac Perhson, Herzigova"

31. Sestavi metodo, ki za vhodni podatek sprejme ime datoteke, na kateri so zapisane koordinate točk v ravnini. Na primer, vsebina takšne datoteke bi lahko bila:

```
-1.060 -3.456
2.850 0.056
0.000 2.718
-9.023 6.003
2.540 9.023
```

V vsaki vrstici sta zapisani koordinati x in y ene točke, ločeni s presledkom. Metoda naj vrne število točk v prvem kvadrantu (to so tiste točke, ki imajo obe koordinati strogo pozitivni). Na primer, za zgornjo datoteko, program izpiše 2, ker sta točki (2.850, 0.056) in (2.540, 9.023) v prvem kvadrantu.

32. Direktor podjetja je dobil datoteko Priporocilo.txt na kateri je pisalo:

Spoštovani gospod direktor,

Janeza Novaka, mojega asistenta pri delu, vedno vidite, kako trdo dela v svoji mali pisarni. Janez dela neodvisno in ne lenari ali se pogovarja s sodelovci. Nikoli se ne zgodi, da bi zavrnil kakšnega sodelovca, ki potrebuje pomoč. Do sedaj je vedno končal z delom pravočasno. Zelo pogosto si vzeme podaljšan delovni čas, da konča svoje delo, pri čemer včasih preskoči odmor. Janez je takšen delavec, ko nima absolutno nobenega spodrslajaja pri opravljenih delih, ima visoke dosežke in je širokega znanja na njegovem področju. Moje mnenje je, da ga lahko takoj uvrstimo med tiste najbolj vzorne delovce, ki jih nikoli ne odpustimo. Prav tako vam vljudno predlagam, da je moj predlog o napredovanju tega izjemnjega, vzornega in nepogrešljivega delavca izvršen kakor hitro je mogoče.

Lep pozdrav!

Že se je spravil pisati predlog za napredovanje, ko je po e-pošti prispel dopis:

> *Direktor!*
> *Ta idiot je stal za menoj, ko sem pisal prejšnje priporočilo.*
> *Prosim znova preberite vsako drugo vrstico tega pisma.*

Direktor je sedaj povsem zmeden. Pomagaj mu in sestavi program, ki na datoteko NovoPriporocilo.txt napiše vsako drugo vrstico prvotnega priporočil!

33. Sestavi metodo, ki izpiše seznam vseh datotek v imeniku (directory) skupaj z vsemi podimeniki. V ta namen poišči ustrezne metode v C#, s pomočjo katerih za dano ime datoteke ugotovi, ali gre za "navadno" datoteko ali imenik.

Primer:
klic DirR (@"D:\j2sdk1.4.1") izpiše (... označujejo spuščene vrstice):

```
D:\j2sdk1.4.1\bin\appletviewer.exe
D:\j2sdk1.4.1\bin\extcheck.exe
...
D:\j2sdk1.4.1\bin\tnameserv.exe
D:\j2sdk1.4.1\COPYRIGHT
D:\j2sdk1.4.1\demo\applets\Animator\Animation.class
...
D:\j2sdk1.4.1\demo\applets\Animator\Animator.java
D:\j2sdk1.4.1\demo\applets\Animator\audio\0.au
...
D:\j2sdk1.4.1\demo\applets\Animator\audio\spacemusic.au
D:\j2sdk1.4.1\demo\applets\Animator\DescriptionFrame.class
...
D:\j2sdk1.4.1\demo\applets\Animator\example4.html
D:\j2sdk1.4.1\demo\applets\Animator\images\Beans\T1.gif
...
D:\j2sdk1.4.1\demo\applets\Animator\images\Beans\T9.gif

(izpuščenih je še približno 20 strani datotek)
```

34. Globina podimenikov
Uporabi idejo prejšnje naloge in ugotovi, kako "globoko" segajo podimeniki danega imenika. Torej za imenik brez poimenikov bo rezultat 0. Če bo dani imenik vseboval podimenik, ki bo vseboval podimenik, ki bo vseboval podimenik, ki bo vseboval le običajne datoteke, bo rezultat 3.
35. Sestavi metodo

```
public static void PrepisBrez(string imeVhod, string imeIzhod)
```

ki tekstovno datoteko, katere ime je v `imeVhod` *prepiše na novo datoteko z imenom, kot ga določa* `imeIzhod`*. Pri tem naj spusti vse številke, struktura vrstic pa naj se ohrani. Primer:*

Vhodna datoteka	Izhodna datoteka
Triglav je visok 2864m. To je visoko. *Dne 25.3.2008 pišemo izpit iz* *predmeta Programiranje 2.* ***To je predzadnja vrstica.*** ***Cela datoteka ima 5 vrstic.***	*Triglav je visok m. To je visoko.* *Dne .. pišemo izpit iz* *predmeta Programiranje .* *To je predzadnja vrstica.* *Cela datoteka ima vrstic.*

36. Dana je tabela znakov. V tabeli so znaki bodisi presledki, bodisi znaki `'*'`. Tabela je napisana na datoteki in sicer tako, da je v prvi vrstici datoteke podana dimenziji tabele (najprej cello število, kipredstavlja število vrstic in nato presledek in nato celo število, ki predstavlja število stolpcev), nato pa vrstica tabele ustreza vrstici na datoteki. Sestavi rekurzivno metodo

```
Zapolni(string ImeVhodDat, string imeIzhodDat, int vr, int st)
```

ki prebere podatke z vhodne datoteke in ustvari datoteko, ki predstavlja tabelo, kjer z znakom '+' "zapolni" zaprt del lika, katerega koordinati (indeksa) sta tretji in četrti parametra metode. Indekse štejemo od 0 dalje.

Tako klic `Zapolni("Vh.txt", "Izh.txt", 2, 8)` ustvari naslednjo datoteko

Vhodna datoteka Vh.txt

```
10 9
* * * * * * * * * *
*         * *     *
*     *   *       *
*     * * * * *   *
* * * *           *
*     *   *   *   *
* *   *   * * *   *
*     *       *   *
* * * * * * * * * *
```

Rezultat (Izh.txt)

```
10 9
* * * * * * * * * *
*         * * + + *
*     *   * + + + *
*     * * * * * + *
* * * * + + + + + *
*     * + * + * + *
* *   * + * * * + *
*     * + + + * + *
* * * * * * * * * *
```

Dva elementa tabele sta med seboj povezana po štirih smereh (in ne morda osmih). Predpostavi, da imamo na robovih tabele vedno znake '*'. Če je na ustreznem mestu znak #, se seveda tabela ne spremeni (torej je izhod nova datoteka z identično vsebino kot jo ima vhodna datoteka)! Prav tako dobimo izhodno datoteko z identično vsebino kot je vhodna datoteka, če so koordinate izven tabele.

37. Napišite program, v katerem tekstovno datoteko napolnite s 100 poljubnimi celimi števili. Ta števila potem preberite iz datoteke in izpišite povprečje vseh lihih.

38. Napišite program `Sestej`, ki prebere ime datoteke, v kateri so zapisana cela števila, vsako v svoji vrsti. Program naj na zaslon izpiše vsoto števil iz datoteke. Denimo, da je v datoteki `Stevila.txt` zapisano

```
10
4
100
12
```

Spodnji okvir ponazarja izvajanje programa:

```
Ime datoteke: Sestej.txt
Vsota: 126
```

39. Napišite program, ki prebere imeni vhodne in izhodne datoteke ter vhodno datoteko prepiše v izhodno. Pri tem naj vse pojavitve znaka 'e' zamenja z znakom 'E'.

40. Napišite program `VrednostPolinoma`, ki iz datoteke "`polinom.txt`" prebere celo število `t` in celoštevilske vrednosti polinoma $p(x) = a_0 + a_1x + \ldots + a_nx^n$ ter izpiše na zaslon vrednost `p(t)`. Datoteka vsebuje dve vrstici: v prvi vrstici je zapisano celo število `t`, v drugi vrstici pa so koeficienti polinoma `p`, ločeni s presledki.

Primer:
Zapis v datoteki
```
2
1 2 0 3 4
```
Za ta primer mora program izpisati na zaslon število `29`, ker podatki predstavljajo `t=2` in polinom $p(x)=1+2x+4x^3+3x^4$, od koder dobimo $p(t)=1+2\cdot2+4\cdot2^3+3\cdot2^4=29$.

41. Tekstovna datoteka naj bo sestavljena iz vrst, napolnjenih s celimi števili, ločenimi s presledki. Sestavite statično metodo `MinMax(imeDatoteke)`, ki vrne maksimum minimumov števil v vsaki vrstici v datoteki.

Primer: Če je v datoteki `Stevila.txt` zapisano

```
2 3 4
5 -1
8 9 6 3
```

je rezultat metode `MinMax("Stevila.txt")` enak `3`.

42. Ustvarite tekstovno datoteko `APJ.txt`. V to datoteko zapišite poljubno število stavkov (berete jih preko tipkovnice, znak za konec vnosa je `*`). Vsak stavek naj bo na datoteki v svoji vrstici, med vrsticami pa naj bo po ena prazna vrstica. Napišite metodo, ki dobi za parameter ime te datoteke. Vrne naj, koliko znakov vsebuje celotna datoteka! Prazna vrstica seveda ne vsebuje nobenega znaka.

43. Dana je tekstovna datoteka `Naloga.txt`. V vsaki vrstici te datoteke so po tri cela števila, med seboj ločena s presledkom. Datoteko obdelajte tako, da za vsako vrstico na zaslon izpišete vsoto vseh treh števil, na koncu pa še skupno vsoto vseh števil!

44. V tekstovni datoteki so zapisani podatki o porabi bencina za posamezne tipe vozila. V vsaki vrstici je zapisan tip vozila, nato pa podatek o porabi goriva na 100 km (realno število). Koliko vozil je v datoteki? Ugotovite in izpišite tip vozila z najmanjšo porabo goriva. Podatka o tipu vozila in porabi goriva sta razmejena z ločilnim znakom |.

45. Napišite program, ki v poljubni tekstovni datoteki prešteje vse števke in na koncu izpiše, kolikokrat se vsaka števka pojavi v tej datoteki. Ime datoteke programu podamo preko konzolnega okna.

46. V datoteki `realna.txt` so zapisana realna števila (vsako v svoji vrstici). Napišite program, ki ugotovi in izpiše, koliko je vseh števil v datoteki in kakšen je procent števil 0.

Primer izpisa:
```
V datoteki je 20 stevil, od tega 25 procentov nicel.
```

Dodatno gradivo

Tu je navedeno gradivo, ki ga načeloma v sklopu predavanj in vaj ne bomo uporabljali. Navedene so določene dodatne metode. Prav tako so uporabljeni posamezni prijemi iz "stare" snovi, ki jih nismo obravnavali. Ta del je zgolj informativne narave.

Dodatne metode imenskega prostora System.IO

Kot smo omenili, za delo z datotekami v C# potrebujemo imenski prostor **System.IO**, ki vsebuje kopico **razredov**, za upravljanje z imeniki (direktoriji), datotekami in potmi do datotek. Razredi tega imenskega prostora pa vsebujejo cel kup metod za raznorazne vhodno-izhodne operacije, med katerimi so za nas najpomembnejše metode za kreiranje, zapisovanje, branje in na splošno manipuliranje s tekstovnimi in binarnimi datotekami.

Razredi imenskega prostora System.IO za delo z imeniki, datotekami in potmi do datotek

Razred	Razlaga
Directory	Uporablja se za kreiranje, urejanje, brisanje ali pridobivanje informacij o imenikih.
File	Uporablja se za kreiranje, urejanje, brisanje ali za pridobivanje informacij o datotekah.
Path	Uporablja se za pridobivanje informacij o poteh do datotek.

Najpomembnejše metode razreda Directory

Metoda	Razlaga
Exists(path)	Vrne logično vrednost, ki ponazarja ali nek imenik obstaja ali ne.
CreateDirectory(path)	Kreira imenike v navedeni poti.
Delete(path)	Brisanje imenika in njegove vsebine.
GetFiles(path)	Pridobivanje imen datotek navedene poti

Primer uporabe:

```
//Kreiranje novega imenika C# 2005 in v njem še podimenika datoteke
string dir ="C: \\C# 2005\\Datoteke\\";
//ali
//znak @ pred definicijo pomeni, da znak \ v stringu ne predstavlja escape sekvence
string dir1 = @"C: \C# 2005\Datoteke\";
if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
    Directory.CreateDirectory(dir);
```

Najpomembnejše metode razreda File

Metoda	Razlaga
Exists(path)	Vrne logično vrednost, ki ponazarja ali neka datoteka obstaja ali ne.
Delete(path)	Brisanje datoteke.
Copy(source, dest)	Kopiranje datoteke iz izvorne poti (**source**) do končne poti (**dest**).
Move(source, dest)	Premik datoteke iz izvorne poti (**source**) do končne poti (**dest**).

Napisanih je le nekaj najpomembnejših metod razredov **Directory** in **File**. Ostale metode in njihovo uporabo lahko dobimo v sistemu pomoči, ki je sestavni del okolja C#.

Primer uporabe:

```
string dir = @"c:\C# 2005\Datoteke\";
string pot = dir + "Izdelki.txt";
//preverimo obstoj datoteke Izdelki.txt v navedenem imeniku (c:\C# 2005\Datoteke\)
if (File.Exists(pot))
    File.Delete(pot);  //če datoteka obstaja, jo pobrišemo
```

```
/*Preveri, če na disku C že obstaja mapa z imenom Vaje C#. Če še ne obstaja, jo kreiraj in v
mapi zgeneriraj datoteki Vaja1 in Vaja2 (datoteki bosta seveda PRAZNI!).*/

static void Main(string[] args)
{
    string dir = "C: \\Vaje C#";
    //ALI PA:  string dir = @"C:\Vaje C#";
    //Preverimo, če imenik C:\Vaje C# že obstaja - če še ne obstaja, ga skreiramo
    if (!Directory.Exists(dir))
        Directory.CreateDirectory(dir);

    string datoteka = "Vaja1";
    string datoteka1 = "Vaja2";
    //Za datoteko Vaja1 preverimo obstoj v imeniku C:\Vaje C#: če že obstaja,jo prej pobrišimo
    if (File.Exists(dir + "\\" + datoteka))
    {
         Console.WriteLine("Datoteka Vaja1 že obstaja in bo pobrisana!");
         File.Delete(dir + "\\" + datoteka);  //če datoteka obstaja, jo pobrišemo
    }
    //skreirajmo NOVO datoteko Vaja1 v imeniku C:\Vaje C#
    File.Create(dir + "\\" + datoteka);
    //skreirajmo NOVO datoteko Vaja2 v imeniku C:\Vaje C#
```

```
    File.Create(dir + "\\" + datoteka1);
    //S pomočjo metode GetFiles imena vseh datotek izbrane mape shranimo v tabelo datoteke
    string[] datoteke = Directory.GetFiles(dir);
    Console.WriteLine("Seznam datotek imenika Vaje C# na disku C:");
    for (int i = 0; i < datoteke.Length; i++) {
       string imedatoteke = datoteke[i]; //izpišimo imena vseh datotek mape C:\Vaje C#
       Console.WriteLine(imedatoteke);
    }
}
```

Dodatne naloge z datotekami

1. Preveri, če na disku D že obstaja imenik d:\C#\Vaje\Datoteke. Če še ne obstaja, ga ustvari, nato pa v njem ustvari datoteki Vaja1.txt in Vaja2.txt. Če imenik že obstaja, najprej preveri, če datoteki Vaja1.txt in Vaja2.txt že obstajata – če ne, ju skreiraj.

2. Preveri, če na disku C že obstaja imenik z imenom, ki ga vneseš preko tipkovnice. Če imenik že obstaja, ugotovi in izpiši, koliko datotek vsebuje. Izpiši tudi imena vseh datotek.

3. Ustvari poljubni imenik in v njej datoteko s poljubnim imenom – imeni vnese uporabnik preko tipkovnice. Pred kreiranjem preveri, če imenik oz. datoteka že obstaja.

4. Ustvari drevesno strukturo imenikov d:\P1\C#\VajeC#\Letnik_1

Delo s podatkovnimi tokovi

Ko uporabljamo razrede imenskega prostora **System.IO** za delo z vhodno izhodnimi operacijami, lahko

Za delo z vhodno-izhodnimi (**I/O** – **Input/Output**) operacijami s tekstovnimi in binarnimi datotekami, **.NET Framework** uporablja tokove (**streams**). Tok (**stream**) si lahko predstavljamo kot pretakanje podatkov iz ene lokacije na drugo. Izhodni tok (**otput stream**) si torej predstavljamo kot tok podatkov z internega pomnilnika aplikacije v datoteko na disku, vhodni tok (**input stream**) pa kot tok podatkov z diska v interni pomnilnik. Pri delu s tekstovnimi datotekami uporabljamo tekstovni tok podatkov (**text stream**), pri delu z binarnimi datotekami pa binarni tok (**binary stream**).

Tokovi podatkov za delo z datotekami

Podatkovni tok	Razlaga
Text	Uporablja se za prenos tekstovnih podatkov.
Binary	Uporablja se za prenos binarnih podatkov.

V tem razdelku bodo prikazani razredi imenskega prostora **System IO**, ki jih uporabljamo za delo s tokovi in datotekami. Za kreiranje toka, ki nas poveže z datoteko, tako npr. uporabimo razred **FileStream**. Za branje podatkov iz datoteke preko tekstovnega toka uporabimo npr. razred **StreamReader**, za branje podatkov iz binarne datoteke preko binarnega toka pa razred **BinaryReader**.

Vrste podatkovnih tokov

Razred	Razlaga
Stream	Splošen podatkovni tok
FileStream	Zagotavljanje dostopa do vhodnih in izhodnih datotek (podatkovni tok namenjen

	datotekam).
StreamReader	Uporablja se za branje tekstovnih podatkov v podatkovni tok (npr. iz tekstovne datoteke).
StreamWriter	Uporablja se za zapisovanje toka tekstovnih podatkov (npr. v tekstovno datoteko).
BinaryReader	Uporablja se za branje binarnih podatkov v podatkovni tok (npr. iz binarne datoteke).
BinaryWriter	Uporablja se za zapisovanje toka binarnih podatkov (npr. v binarno datoteko).

Razred **Stream** je osnovni razred za vse podatkovne tokove. Predstavljamo si ga lahko kot sekvenco/zaporedje zlogov (bytov), kot npr. datoteka, podatki vneseni preko tipkovnice, podatki, ki smo jih poslali na zaslon. Razred **Stream** torej omogoča splošen oz. enoličen pogled in obdelavo podatkov ne glede na njihov različen izvor, tako da se programerjem ni potrebno ukvarjati s posebnostmi operacijskega sistema in pripadajočo opremo.

V binarne datoteke lahko shranimo prav vse vgrajene numerične podatkovne tipe, zaradi česar so binarne datoteke bolj primerne za aplikacije, ki operirajo z numeričnimi podatki. V nasprotju, pa so vsi numerični podatki v tekstovnih datotekah shranjeni kot zaporedje znakov, zaradi česar jih moramo, če jih hočemo uporabiti v aritmetičnih operacijah, spremeniti v numerične podatke.

Ko shranimo nek tekst v tekstovno datoteko, lahko za to datoteko uporabimo poljubno končnico (ekstenzijo). Bolj naravno pa je (tako bomo delali tudi v nadaljevanju), da za tekstovne datoteke uporabimo končnico ***txt***, za binarne datoteke pa končnico ***dat***. Tako nam že same končnice datotek povedo, ali gre za tekstovne ali pa za binarne datoteke.

Uporaba razreda FileStream

Za kreiranje podatkovnega toka, ki nas poveže z neko datoteko na disku, uporabimo razred **FileStream**.

Sintaksa za kreiranje novega objekta razreda **FileStream**:

```
FileStream fs = new FileStream(pot, mode [, access [, share]])
```

V prvem parametru povemo, kako se datoteka imenuje, lahko pa zapišemo tudi pot do te datoteke (imenike in podimenike), z drugim parametrom pa povemo, kako bomo to datoteko odprli (ali bomo kreirali novo datoteko, ali bomo datoteko le odprli ali pa bomo podatke dodajali k obstoječim v datoteki, ipd.). Prva dva parametra (pot in način kreiranja – **FileMode**) sta obvezna, druga dva pa le opcijska. Če tretji parameter (**access**) ni naveden, lahko podate v datoteko tako zapisujemo, kot tudi beremo iz nje. Za kodiranje (zapis) argumentov `mode`, `access` in `share` uporabljamo zaporedoma naštevne tipe **FileMode**, **FileAccess** in **FileShare**.

Če želimo npr. kreirati datoteko, ki še ne obstaja, bomo uporabili način **FileMode.Create** in tako kreirali novo datoteko. Če pa datoteka z imenom, ki smo jo navedli v prvem parametru (pot) že obstaja, bo njena vsebina prepisana z novo vsebino. Če pa seveda nočemo prepisati vsebine že obstoječe datoteke, bomo raje uporabili način **FileMode.CreateNew**. Naslednja tabela prikazuje vse možne načine odpiranja datoteke:

Tabela načinov kreiranja datoteke – FileMode

Način kreiranja	Razlaga
Append	Odpiranje datoteke, če le-ta obstaja in obenem se postavimo na njen konec. Če datoteka ne obstaja, se skreira nova. **Ta način kreiranja datoteke lahko uporabimo le kadar želimo v datoteko pisati**, ne pa tudi kadar želimo iz nje samo brati podatke.
Create	Kreiranje nove datoteke. **Če datoteka že obstaja, bo njena vsebina prepisana**.
CreateNew	Kreiranje nove datoteke. Če datoteka že obstaja, pride do izjeme (napake - **exception**).
Open	Odpiranje že obstoječe datoteke. Če datoteka še ne obstaja, pride do izjeme (**exception**).

OpenOrCreate	Odpiranje datoteke, če le ta obstaja, oziroma kreiranje nove datoteke, če le-ta še ne obstaja.
Truncate	Odpiranje obstoječe datoteke in jo skrajšati (izprazniti), tako da je njena dolžina nič bytov.

Z drugim parametrom **`access`** povemo, ali bomo podatke iz datoteke brali, jih vanjo zapisovali ali pa oboje. Ta parameter lahko izpustimo, a v tem primeru bo vzeta privzeta vrednost – v datoteko bomo lahko podatke zapisovali in jih hkrati brali iz nje. Naslednja tabela opisuje vse tri možne načine manipulacije z neko datoteko:

Tabela načinov manipulacije s podatki – FileAccess

Način manipulacije	Razlaga
Read	Podatke iz datoteke lahko le beremo, ne pa tudi zapisujemo.
ReadWrite	Podatke iz datoteke lahko beremo in tudi zapisujemo vanjo. **Ta način je privzet**.
Write	Podatke lahko v datoteko le zapisujemo, ne pa tudi beremo.

S tretjim **`share`** argumentom povemo, ali bodo imeli dostop do te datoteke tudi drugi uporabniki in kakšne pravice za dostop bodo imeli. V naslednji tabeli so prikazani vsi možni načini:

Tabela načinov porazdelitev (dostopa) podatkov z drugimi aplikacijami – FileShare

Način porazdelitve	Razlaga
None	Datoteko ne more odpreti nobena druga aplikacija.
Read	Omogoča, da datoteko lahko odprejo tudi druge aplikacije, a le za branje.
ReadWrite	Omogoča, da datoteko lahko odprejo tudi druge aplikacije in to za branje in pisanje.
Write	Omogoča, da datoteko lahko odprejo tudi druge aplikacije, a le za branje.

Primer:

Kreirajmo nov objekt izpeljan iz razreda **FileStream** in ga poimenujmo **fs**. Objektu smo s tretjim parametrom priredili le možnost **pisanja** v datoteko.

```
string pot = @"C:\Datoteke\Izdelki.txt";
FileStream fs = new FileStream(pot, FileMode.Create, FileAccess.Write);
```

V primeru smo uporabili metodo **`FileMode.Create`** za kreiranje nove datoteke (oz. za prepis vsebine že obstoječe datoteke). Če pa smo v prvem parametru **`pot`** uporabili pot, ki ne obstaja (npr. navedli smo neobstoječi imenik), bo prišlo do izjeme (napake) tipa **DirectoryNotFoundException** (več o izjemah je razloženo v poglavju **Varovalni bloki – obravnava izjem**).

Kreirajmo nov objekt izpeljan iz razreda **FileStream** in ga poimenujmo **fs**. Objektu smo s tretjim parametrom priredili le možnost **branja** v datoteko. Tudi v tem primeru, tako kot v prejšnjem, smo uporabili metodo **`FileMode.Create`** za kreiranje nove datoteke (oz. za prepis vsebine že obstoječe datoteke). Velja tako kot za prvi primer: če smo v prvem parametru **`pot`** uporabili pot, ki ne obstaja (npr. navedli neobstoječi imenik), bo prišlo do izjeme (napake) tipa **DirectoryNotFoundException**.

```
string pot = @"C: \C# 2005\Datoteke\Izdelki.txt";
FileStream fs = new FileStream(pot, FileMode.Create, FileAccess.Read);
```

Najpogostejše metode razreda FileStream

Metoda	Razlaga
Close( )	Zapiranje podatkovnega toka in sproščanje pomnilnika za vse vire vezane na ta tok.
Seek( )	Nastavitev trenutnega položaja potkavnega toka na določeno vrednost
Flush( )	Izpraznitev podatkovnega toka in dokončen zapis vseh podatkov iz toka npr. na disk

Delo s tekstovnimi datotekami

Za branje in pisanje podatkov v tekstovne datoteke uporabljamo razreda **StreamReader** in **StreamWriter**.

Pisanje podatkov v tekstovno datoteko

Za pisanje podatkov v tekstovno datoteko uporabljamo metodi **Write** in **WriteLine** ki pripadata razredu **StreamWriter**. Pri uporabi metode **WriteLine** je v datoteko avtomatsko dodan še znak za konec tekoče vrstice. V kolikor v datoteko shranjujemo posamezne podatke (in ne cele stavke), lahko le-te med seboj ločimo z ustreznim ločilom (npr znakom **|**).

Da lahko pričnemo s pisanjem podatkov v tekstovno datoteko, moramo najprej ustvariti nov objekt tipa **StreamWriter**. To storimo s stavkom:

```
StreamWriter textOut=new StreamWriter(podatkovni_tok);
```

Pri tem je **`podatkovni_tok`** objekt (spremenljivka) tipa **FileStream**, ki smo ga ustvarili že prej (npr. s stavkom `FileStream podatkovni_tok = new FileStream(@"C:\APJ\Vaja.txt", FileMode.Create);`

Obstaja pa seveda tudi krajši način vzpostavljanja podatkovnega toka za pisanje v neko tekstovno datoteko.

```
string datoteka = @"D:\APJ\Vaja.txt";  //ime datoteke, ki jo želimo ustvariti in vanjo pisati
StreamWriter textOut = new StreamWriter(datoteka); /*ker smo uporabili PRIVZETI način za
odpiranje nove datoteke, bo stara vsebina datoteke prepisana z novo vsebino, ne glede na
prejšnjo vsebino datoteke!!!!*/
```

Nastavitve so v tem primeru torej **privzete** – ustvarila se bo **nova** datoteka (ne glede na to ali datoteka s tem imenom že obstaja), v to datoteko bomo pisali, **če pa datoteka s tem imenom že obstaja, bo njena vsebina prepisana z novo vsebino brez opozorila**!)

Metode razreda StreamWriter	Razlaga
Write(podatki)	Zapiše podatke v izhodni tok.
WriteLine(podatki)	Zapiše podatke v izhodni tok in na koncu doda še znak za konec vrstice (običajno znaka **\r\n – carriage return** in **line feed**).
Close( )	Zapre objekt tipa **StremWriter** in pripadajoči objekt **FileStream**.

Kot primer uporabe napišimo kodo, ki v tekstovno datoteko zapiše eno samo vrstico s tremi podatki, ki so med seboj ločeni z ločilom **|**.

```
//Najprej deklariramo string, ki označuje pot do datoteke in njeno ime.
//Pot (imenik in podimenik) MORA že obstajati, sicer pride do napake
string pot = @"C:\Tekstovna\Izdelki.txt";
```

```
//dinamično kreiramo podatkovni tok: napovemo pot in ime datoteke, način kako jo bomo kreirali
//(FileMode.create) in kakšna bo manipulacija s podatki (FileAccess.write)
FileStream fs=new FileStream(pot,FileMode.Create,FileAccess.Write);

//dinamično kreiramo nov objekt tipa StreamWriter za zapisovanje v podatkovni tok - datoteko
StreamWriter textOut=new StreamWriter(fs);

textOut.Write("Kolo"+"|"); //zapis prvega podatka v datoteko, za njim pa še ločilnega znaka |
//lahko bi zapisali tudi textOut.Write("Scott|");
textOut.Write("Scott"+"|");

int komadov = 20;
textOut.Write(komadov + "|"); //enakovredno kot textOut.Write(komadov.Tostring() + "|");

//podatek 220000 je sicer numeričen, a zaradi avtomat. konverzije v string ne pride do napake
textOut.WriteLine(220000);
//enakovreden zapis zadnjega stavka bi bil tudi textOut.WriteLine("220000");

textOut.Close();  //zapiranje podatkovnega toka in s tem datoteke
fs.Close(); //zapremo podatkovni tok fs, da bo datoteka na voljo drugim uporabnikom
```

V zgornjem primeru smo v tekstovno datoteko zapisali tudi numerična podatka. Pri vpisovanju pride do avtomatske konverzije numeričnega podatka v **string**, zaradi česar dodatna uporaba metode **ToString()** ni potrebna. Za konverzijo poskrbi kar sama metoda **Write** oz. **WriteLine**.

Zapomnimo si: pri delu s tekstovno datoteko moramo najprej ustvariti ustrezen **podatkovni tok** (kjer navedemo ime in pot do datoteke, način kreiranja datoteke – **FileMode** in pa način dostopa do datoteeke – **FileAccess**), nato pa moramo kreirati še ustrezen **objekt za zapisovanje podatkov v podatkovni tok** (za zapisovanje v datoteko je to objekt tipa **StreamWriter** , za branje podatkov iz datoteke pa objekt tipa **StreamReader**).

Vaja:

```
/*Na disku C kreiraj mapo Tekstovna. V tej podmapi skreiraj tekstovno datoteko Naslov.txt in
vanjo zapiši svoje osebne podatke*/

string dir = @"C: \Tekstovna";

if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
    Directory.CreateDirectory(dir);
else Console.WriteLine("Mapa že obstaja! Vsebina datoteke bo prepisana!");

string datoteke = dir+@"\Izdelki.txt";

/*ustvarimo podatkovni tok za povezavo z datoteko na disku. Uporabimo opcijo Create (Če
datoteka že obstaja, bo njena vsebina prepisana z novo. Tretji parameter nastavimo na
FileAccess.Write - v datoteko lahko podatke zapisujemo.*/
FileStream fs = new FileStream(datoteke, FileMode.Create, FileAccess.Write);

//kreiramo nov objekt tipa StreamWriter za zapisovanje v podatkovni tok
StreamWriter textOut = new StreamWriter(fs);
//Podatkovni tok je pripravljen za pisanje v datoteko
textOut.WriteLine("France Prešeren"); //zapis prvega stavka v datoteko
textOut.WriteLine("Triglavska 123"); //zapis drugega stavka v datoteko
textOut.WriteLine("Vrba na Gorenjskem");//zapis tretjega stavka v datoteko
textOut.Close();  //zapiranje podatkovnega toka in s tem datoteke
fs.Close(); //zapremo podatkovni tok fs, da bo datoteka na voljo drugim uporabnikom
```

Vaja:

```
/*tekstovno datoteko "c:\Tekstovna\Nakljucna.txt" zapiši 500 naključnih celih števil med 0 in
1000. V vsaki vrstici naj bo natanko 20 števil!*/

string dir = @"C: \Tekstovna";  //imenik in pot

//preverimo obstoj imenika
if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
Directory.CreateDirectory(dir);
string datoteke = dir + @"\Nakljucna.txt";
```

```
/*ustvarimo podatkovni tok za povezavo z datoteko na disku. Uporabimo opcijo Create (Če
datoteka že obstaja, bo njena vsebina prepisana z novo. Tretji parameter nastavimo na
FileAccess.Write - v datoteko lahko podatke zapisujemo.*/
FileStream fs = new FileStream(datoteke, FileMode.Create, FileAccess.Write);

//kreiramo nov objekt tipa StreamWriter za zapisovanje v podatkovni tok
StreamWriter textOut = new StreamWriter(fs);
//Podatkovni tok je pripravljen za pisanje v datoteko
Random naklj = new Random();
for (int i = 0; i < 500; i++)
{
   //Zapis formatiramo tako, da za vsako število predvidimo natanko 5 mest, desna poravnava
   textOut.Write("{0,5}",naklj.Next(1001));
   //textOut.Write("{0,-5}", naklj.Next(1001));//TAKOLE pa bi izgledala LEVA poravnava števila
   if ((i + 1) % 20 == 0)  //v vsaki vrstici naj bo le po 20 števil
      textOut.WriteLine(); //skok v novo vrstico
}
textOut.Close();  //zapiranje podatkovnega toka in s tem datoteke
fs.Close(); //zapremo podatkovni tok fs, da bo datoteka na voljo drugim uporabnikom
```

Vaja:

```
/*v tekstovno datoteko, katere ime določi uporabnik (nahaja pa se v mapi "c:\Tekstovna" zapiši
poštevanko števila, ki ga prebereš preko tipkovnice!*/

string dir = @"C: \Tekstovna";  //imenik in pot

if (!Directory.Exists(dir))   //če imenik še ne obstaja, ga skreiramo
    Directory.CreateDirectory(dir);

Console.Write("Ime datoteke (brez končnice): ");
string ime = Console.ReadLine();                    //Preberemo ime datoteke
ime = dir + @"\" + ime + ".txt";
Console.Write("Število za poštevanko: ");
int stevilo = Convert.ToInt32(Console.ReadLine()); //Preberemo ime število za poštevanko
/*ustvarimo podatkovni tok za povezavo z datoteko na disku. Uporabimo opcijo Create (Če
datoteka že obstaja, bo njena vsebina prepisana z novo. Tretji parameter nastavimo na
FileAccess.Write - v datoteko lahko podatke zapisujemo.*/

FileStream fs = new FileStream(ime, FileMode.Create, FileAccess.Write);

//kreiramo nov objekt tipa StreamWriter za zapisovanje v podatkovni tok
StreamWriter textOut = new StreamWriter(fs);

for (int i = 1; i < 11; i++)
{
    //Zapis v datoteko formatiramo
    textOut.WriteLine("{0,3} * {1,3} = {2,5}", i,stevilo,stevilo*i);
}
textOut.Close();  //zapiranje podatkovnega toka in s tem datoteke
fs.Close(); //zapremo podatkovni tok fs, da bo datoteka na voljo drugim uporabnikom
```

Vaja:

```
/*Kreiraj dvodimenzionalno tabelo 50 x 20 naključnih celih števil med vključno -100 in +100.
Tabelo nato zapiši v tekstovno datoteko Tabela2D.txt*/

Random naklj = new Random();
//zgenerirajmo tabelo in jo napolnimo z naključnimi števili
int [,]tabela2D=new int[50,20];
for (int i = 0; i < 50; i++)
  for (int j = 0; j < 20; j++)
     tabela2D[i,j] = naklj.Next(-100, 101);

//tabelo sedaj prepišimo v tekstovno datoteko c:\Tekstovna\Tabela2D.txt
string dir = @"C: \Tekstovna";  //imenik in pot

//preverimo obstoj imenika
if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
  Directory.CreateDirectory(dir);

string datoteke = dir + @"\Tabela2D.txt"; //datoteka, v katero bomo prepisali našo 2D tabelo
```

```
/*ustvarimo podatkovni tok za povezavo z datoteko na disku. Uporabimo opcijo Create (Če
datoteka že obstaja, bo njena vsebina prepisana z novo. Tretji parameter nastavimo na
FileAccess.Write - v datoteko lahko podatke zapisujemo.*/
FileStream fs = new FileStream(datoteke, FileMode.Create, FileAccess.Write);

//kreiramo nov objekt tipa StreamWriter za zapisovanje v podatkovni tok
StreamWriter textOut = new StreamWriter(fs);
//vsebino tabele prepišemo v datoteko
for (int i = 0; i < 50; i++)
{
   for (int j = 0; j < 20; j++)
   {
      //Zapis v datoteko formatiramo na 6 mest
      textOut.Write("{0,6}",tabela2D[i, j]);
   }
   textOut.WriteLine(); //skok v novo vrstico
}
textOut.Close();  //zapiranje podatkovnega toka in s tem datoteke
fs.Close(); //zapremo podatkovni tok fs, da bo datoteka na voljo drugim uporabnikom
```

Naloge:

- V tekstovno datoteko NASLOV.TXT zapiši svoje osebne podatke (1. vrstica: ime in priimek, 2. vrstica: naslov, 3. vrstica naj bo prazna, 4.vrstica: pošta in kraj)!
- Napiši program, ki bo v zanki bral podatke o določenem kraju in njegovi poštni številki. Podatke zapisuj v tekstovno datoteko, za vsak kraj v svojo vrstico. Zanka (vnos podatkov) se zaključi, ko uporabnik vnese prazen kraj!
- Kreiraj tekstovno datoteko OCENE.TXT in vanjo zapiši štiri vrstice: v vsaki vrstici naj bo naziv predmeta in tvoja ocena pri tem predmetu. Vpis naj bo formatiran (za naziv predmeta natanko 40 znakov – leva poravnava, za oceno pa natanko 2 znaka - desna poravnava!)
- Napiši program, ki bere stavke vnesene preko tipkovnice in stavke zapisuje v tekstovno datoteko v obratnem vrstnem redu.
- V tekstovno datoteko Stevila.txt zapiši prvih 100 sodih števil, ki niso deljiva s 6! Med števila zapiši poljuben ločilni znak!
- Napiši program za pisanje/dodajanje podatkov v tekstovno datoteko DRZAVE.TXT. Podatke vpisuj/dodajaj s močjo funkcije, ki naj na začetku preveri, ali datoteka sploh obstaja – če datoteka še ne obstaja naj jo funkcija najprej ustvari. Vsaka vrstica v datoteki naj bo sestavljena iz imena države (40 znakov) in števila prebivalcev v njej (celo število, formatirano na 15 mest)
- Kreiraj tekstovno datoteko KOCKA.TXT, v katero želiš shraniti rezultate 1000 metov kocke. V vsako vrstico te datoteke nato zapiši zaporedno številko vrstice in naključno število med 1 in 6 (met kocke) – vpis meta kocke formatiraj na 3 mesta. Izgled datoteke:

  1._ _ 4
  2._ _ 6
  3._ _ 1
  …

Branje podatkov iz tekstovne datoteke

Za branje podatkov iz tekstovne datoteke je na voljo več metod razreda **StreamReader**, najpomembnejši pa sta metodi **Read** in **ReadLine**.

Da lahko pričnemo z branjem podatkov iz tekstovne datoteke, moramo najprej ustvariti nov objekt tipa **StreamReader**. To storimo npr. takole:

```
string datoteka = @"C: \Tekstovna\Padavine.txt";  //ime in pot do datoteke
//najprej ustvarimo objekt za povezavo z datoteko na disku. Uporabimo opcijo Open.
FileStream podatkovni_tok = new FileStream(datoteka, FileMode.Open);
//kreiramo nov objekt tipa StreamReader za branje v podatkovni tok
StreamReader textIn = new StreamReader(podatkovni_tok);
```

Obstaja pa tudi krajši način za odpiranje datoteke za branje, brez predhodnega kreiranja objekta tipa **FileStream**. V tem primeru so nastavitve ob odpiranju datoteke privzete (datoteko odpremo za branje, vsebina datoteke pa bo drugim uporabnikom nedostopna vse dokler je ne bomo zaprli).

```
string datoteka = @"C: \Tekstovna\Padavine.txt";  //ime in pot do datoteke
StreamReader textIn = new StreamReader(datoteka); //privzeto odpiranje datoteke za branje
```

Metode razreda StreamReader	Razlaga
Peek( )	Vrne naslednji razpoložljivi znak v vhodni tok, brez premika na naslednjo pozicijo (naslednji znak). Če ni na voljo nobenega znaka več, metoda vrne vrednost -1.
EndOfStream()	Metoda vrne **true**, če smo že na koncu podatkovnega toka, sicer pa vrne vrednost **false**.
Read( )	Bere naslednji razpoložljivi znak z vhodnega toka. **POZOR**: metoda vrne **CELO ŠTEVILO**, ki predstavlja **ASCII** kodo tega znaka.
ReadLine( )	Bere naslednjo vrstico podatkov z vhodnega toka in jo vrne kot **string**.
ReadToEnd( )	Bere podatke s trenutne pozicije v vhodnem toku, vse do konca toka in podatke vrne kot **string**. Navadno se ta metoda uporablja za branje celotne vsebine datoteke.
Close( )	Zapre objekt tipa **StremReader** in pripadajoči objekt **FileStream**.
Lastnost	**Razlaga**
EndOfStream	Pridobivanje vrednosti ki nam pove, ali smo že na koncu podatkovnega toka. V tem primeru je lastnost enaka **true**, sicer pa **false**.

Primer:

```
/*Dana je tekstovna datoteka c:\Tekstovna\Padavine.txt. Izpišimo jo na zaslon. Uporabili bomo
vse tri načine:
1) Branje vsakega znaka posebej
2) Branje vrstice za vrstico
3) Enkratno branje celotne datoteke*/

string datoteka = @"C: \Tekstovna\Padavine.txt";  //ime in pot do datoteke

if (File.Exists(datoteka))  //preverimo obstoj datoteke
{
   /*ustvarimo podatkovni tok za povezavo z datoteko na disku. Uporabimo opcijo Open. Tretji
     parameter nastavimo na FileAccess.Read = branje datoteke*/
   FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   //kreiramo nov objekt tipa StreamReader za zapisovanje v podatkovni tok
   StreamReader textIn = new StreamReader(fs);

   //1) Branje vsakega znaka posebej
   while (textIn.Peek() != -1) //iz datoteke beremo posamezne znake dokler jih ne zmanjka
   {
      char znak = (char)textIn.Read(); //preberemo vsak znak posebej
      Console.Write(znak);             //prebrani znak izpišemo na zaslon
   }
   textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
```

```
   Console.WriteLine("\n-------------------------------------------------------------------");

   //2) Branje vrstice za vrstico
   fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   textIn = new StreamReader(fs);
   while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
   /*Lahko bi zapisali tudi:
     while (!textIn.EndOfStream)   //dokler NI konec toka
        {…}

   ali pa:
     string vrstica;
     //dokler metoda ReadLine vrača vrednost različno od null
      while ((vrstica = branje.ReadLine()) != null)
     {…}
   */

   {
      string stavek = textIn.ReadLine(); //preberemo celo vrstico
      Console.WriteLine(stavek);         //prebrano vrstico izpišemo na zaslon
   }
   textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke

   Console.WriteLine("\n-------------------------------------------------------------------");

   //3) Enkratno branje cele datoteke
   fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   textIn = new StreamReader(fs);
   string vsebinaDatoteke = textIn.ReadToEnd(); //naenkrat preberemo celotno vsebino datoteke
   Console.WriteLine(vsebinaDatoteke);          //prebrano vsebino datoteke izpišemo na zaslon
   textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
   fs.Close();
}
```

Vaja:

```
/*Dana je tekstovna datoteka c:\Tekstovna\Padavine.txt. Prekopiraj jo v datoteko
Padavine.rez. Nalogo reši na štiri načine:
   1) Datoteko prekopiraš z metodo Copy razreda File
   2) Iz datoteke bereš vsak znak posebej in znake sproti zapisuješ v novo datoteko
   3) Iz datoteke bereš stavke in te stavke sproti zapisuješ v novo datoteko
   4) Naenkrat prebereš celotno vsebino datoteke in to vsebino nato zapišeš v novo datoteko*/

string datoteka = @"C: \Tekstovna\Padavine.txt";  //ime in pot do datoteke

if (File.Exists(datoteka))  //preverimo obstoj datoteke
{
    //1)Metoda Copy
    if (!File.Exists(datoteka))
    File.Copy(datoteka, @"C: \Tekstovna\Padavine.rez");

    //2) Branje vsakega znaka posebej
    FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);//tok za branje
    FileStream f = new FileStream(@"C: \Tekstovna\Padavine1.rez", FileMode.Create,
                                  FileAccess.Write);//podatkovni tok za pisanje
    //kreiramo nov objekta tipa StreamReader in StreamWriter
    StreamReader textIn = new StreamReader(fs);//podatke bomo brali v tok
    StreamWriter textOut = new StreamWriter(f);//podatke bom s tokom zapisovali

    while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
    {
        char znak = (char)textIn.Read(); //preberemo vsak znak posebej
        textOut.Write(znak);             //prebrani znak izpišemo v novo datoteko
    }
    textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
    textOut.Close(); //zapiranje podatkovnega toka in s tem datoteke

    //3) Branje vrstice za vrstico
    fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
    f = new FileStream(@"C: \Tekstovna\Padavine2.rez", FileMode.Create,
                       FileAccess.Write);//podatkovni tok za pisanje
    textIn = new StreamReader(fs);//podatke bomo brali v tok
    textOut = new StreamWriter(f);//podatke bom s tokom zapisovali
```

```
    while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
    {
       string stavek = textIn.ReadLine(); //preberemo celo vrstico
       textOut.WriteLine(stavek);         //prebrano vrstico izpišemo v novo datoteko
    }
    textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
    textOut.Close(); //zapiranje podatkovnega toka in s tem datoteke

    //4) Enkratno branje cele datoteke
    fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
    f = new FileStream(@"C: \Tekstovna\Padavine3.rez", FileMode.Create,
                       FileAccess.Write);//podatkovni tok za pisanje
    textIn = new StreamReader(fs);//podatke bomo brali v tok
    textOut = new StreamWriter(f);//podatke bom s tokom zapisovali
    string vsebinaDatoteke = textIn.ReadToEnd(); //naenkrat preberemo celotno vsebino datoteke
    textOut.WriteLine(vsebinaDatoteke);          //prebrano vsebino izpišemo v novo datoteko
    textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
    textOut.Close(); //zapiranje podatkovnega toka in s tem datoteke
    fs.Close();
}
```

Vaja:

```
/*Za poljubno tekstovno datoteko (ime vnese uporabnik) ugotovi
  1) Koliko vrstic vsebuje
  2) Koliko je vseh znakov v datoteki
  3) Koliko samoglasnikov vsebuje
  4) Najdaljši stavek v datoteki*/

string datoteka;
while (true)  //neskončna zanka
{
    Console.Write("Ime datoteke: ");
    datoteka = Console.ReadLine();
    if (File.Exists(datoteka))
      break;  //če datoteka obstaja, gremo iz zanke ven
    else Console.WriteLine("Datoteka s tem imenom ne obstaja!");
}

FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);//tok za branje
//kreiramo nov objekt tipa StreamReader in StreamWriter za zapisovanje v podatkovni tok
StreamReader textIn = new StreamReader(fs);//podatke bomo brali v tok

int znakov = 0,vrstic=0,samoglasnikov=0;
string najdaljsi="";
while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
{
    string stavek = textIn.ReadLine();    //preberemo cel stavek
    if (stavek.Length > najdaljsi.Length) //preverimo, če je ta stavek daljši od doslej
        najdaljsi = stavek;               //najdaljšega
    vrstic++;                            //povečamo število vrstic
    znakov = znakov + stavek.Length;    //povečamo število vseh znakov
    for (int i = 0; i < stavek.Length; i++)
    {
        char znak = char.ToUpper(stavek[i]); //vsak znak spremenimo v veliko črko (če znak ni
                                             //črka, se ne zgodi ničesar)
        if (znak == 'A' || znak == 'E' || znak == 'I' || znak == 'O' || znak == 'U')
           samoglasnikov++;
    }
    //Namesto zgornje rešitve bi lahko brali tudi vsak znak posebej
    //char znak = (char)textIn.Read(); //preberemo vsak znak posebej
    //znaka za konec vrstice sta (char)13 in (char)10
    //znak za konec datoteke pa je textIn.EndOfStream
    //Preverimo, če smo na koncu vrstice oz na koncu datoteke
    //if ((znak==(char)10)||(znak==(char)13) || (textIn.EndOfStream))
}
textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
Console.WriteLine("Skupno število znakov v datoteki: " + znakov);
Console.WriteLine("Skupno število vrstic v datoteki: " + vrstic);
Console.WriteLine("Skupno število samoglasnikov v datoteki: " + samoglasnikov);
Console.WriteLine("Najdaljši stavek v datoteki: " + najdaljsi);
```

Vaja:

```
/*Za poljubno tekstovno datoteko ugotovi in nato izpiši najdaljšo besedo v tej datoteki*/

Console.Write("ime datoteke: ");
string ime=Console.ReadLine();
if(File.Exists(ime))
{
   FileStream fs = new FileStream(ime, FileMode.Open);
   StreamReader textIn = new StreamReader(fs);
   string najdaljsa = "";
   while (textIn.Peek() != -1) //dokler ni konec datoteke
   {
       string vrstica = textIn.ReadLine();//preberemo celo vrstico
       //z metodo Split posamezne besede iz vrstice shranimo v tabelo
       string[] tabela = vrstica.Split(' ');
       for (int i = 0; i < tabela.Length; i++)
       {
           if (tabela[i].Length > najdaljsa.Length)
           {
               najdaljsa = tabela[i];
           }
       }
   }
    textIn.Close();
    fs.Close();
    Console.WriteLine("najdaljsa beseda: "+najdaljsa);
}
```

Vaja:

```
/*PREPROSTA MENJALNICA!  Program, za vnos novega tečaja, izpis tečajne liste in menjavo
denarja. Podatki so v datoteki Tecaji.txt - to je datoteka deviznih tečajev (v vsaki vrstici
je oznaka države, oznaka valute in trenutni prodajni tečaj (realno število).*/
static void vnos(string imeDat)
{
   FileStream fs;
   if (!File.Exists(imeDat))//če datoteka še ne obstaja bomo naredili novo
      fs = new FileStream(imeDat, FileMode.Create, FileAccess.Write);
   else  //če datoteka že obstaja, bomo podatke dodajali
      fs = new FileStream(imeDat, FileMode.Append, FileAccess.Write);

   StreamWriter textOut=new StreamWriter(fs);
   //vnos podatkov o novi valuti
   Console.Write("Država: ");
   string drzava = Console.ReadLine();
   Console.Write("Oznaka valute: ");
   string valuta = Console.ReadLine();
   Console.Write("Trenutni tečaj: ");
   double tecaj = Convert.ToDouble(Console.ReadLine());
   textOut.WriteLine(drzava+"|"+valuta + "|" + tecaj);//zapis v datoteko, med podatki je
                                                      //ločilni znak "|"
   textOut.Close();
   fs.Close();
}

static void izpis(string imeDat)
{
   FileStream fs;
   if (!File.Exists(imeDat))//če datoteka še ne obstaja bomo naredili novo
      Console.WriteLine("Datoteka še ne obstaja!");
   else  //če datoteka že obstaja, bomo podatke dodajali
   {
      fs = new FileStream(imeDat, FileMode.Open, FileAccess.Read);
      StreamReader textIn = new StreamReader(fs);
      Console.WriteLine("Država              Oznaka valute    Tečaj");
      Console.WriteLine("-----------------------------------------");
      while (textIn.Peek() != -1) //podatke beremo dokler ne pridemo do konca datoteke
      {
         string vrstica=textIn.ReadLine();
         string [] tabela=vrstica.Split('|');  //ker so podatki v prebrani vrstici razmejeni z
                                               //znakom "|", jih lahko ločimo z metodo split
         Console.WriteLine("{0,-20}{1,-17}{2,-10:5}",tabela[0],tabela[1],tabela[2]);
      }
```

```
      textIn.Close();
      fs.Close();
   }
   Console.ReadLine();
}

static void menjava(string imeDat)
{
   FileStream fs;
   if (!File.Exists(imeDat))//če datoteka še ne obstaja bomo naredili novo
      Console.WriteLine("Datoteka še ne obstaja!");
   else  //če datoteka že obstaja, bomo podatke dodajali
   {
      fs = new FileStream(imeDat, FileMode.Open, FileAccess.Read);
      StreamReader textIn = new StreamReader(fs);
      //preberem celo datoteko, da vem, katere valute so že v njej
      int stevec=1;
      Console.WriteLine();
      //Najprej preberemo vse vrstice, da ugotovimo, koliko podatkov (in katere valute) je že
      //v datoteki
      while (textIn.Peek() != -1)
      {
         string vrstica = textIn.ReadLine();
         string[] tabela = vrstica.Split('|');
         Console.Write(stevec+") "+tabela[1]+"  "); //oznake valut, ki so že v tabeli v obliki
                                                    //menija izpišemo na zaslon
         stevec++;
      }
      textIn.Close();
      fs = new FileStream(imeDat, FileMode.Open, FileAccess.Read);
      textIn = new StreamReader(fs);
      //uporabnikova izbira valute se ujema z zaporedno številko valute (=vrstice) v datoteki
      Console.Write("\nIzberi ustrezno valuto: ");
      int izb = Convert.ToInt32(Console.ReadLine());
      if (izb > 0 || izb < stevec)
      {
         Console.Write("Znesek za menjavo: ");
         double znesek = Convert.ToDouble(Console.ReadLine());
         int stvrstice=1;
         //poiščemo zaporedno številko vrstice v datoteki, ki se ujema z izbarno valuto
         while (true)
         {
            string vrstica = textIn.ReadLine();
            string[] tabela = vrstica.Split('|');
            if (stvrstice==izb)
            {
               Console.WriteLine();
               Console.WriteLine("Oznaka valute: "+tabela[1]);
               Console.WriteLine("Prodajni tečaj: "+tabela[2]);
               Console.WriteLine("Znesek za menjavo: "+znesek);
               Console.WriteLine("Znesek v valuti: "+znesek*Convert.ToDouble(tabela[2])
                                 +"  "+tabela[1]);
               break;
            }
            else
               stvrstice++;
         }
      }
   }
   Console.ReadLine();
}

//glavni program
static void Main(string[] args)
{
   char izbira;
   do
   {
      Console.Clear();
      Console.WriteLine("V-Vnos oz. dodajanje,I-Izpis,M-Menjava,K-Konec");
      Console.Write("Vnesi izbiro: ");
      izbira=char.ToUpper(Convert.ToChar(Console.ReadLine()));
      switch (izbira)
      {
         case 'V':{
```

```
                    vnos(@"c:\Tekstovna\Tecaji.txt");
                    break;
                }
            case 'I':{
                    izpis(@"c:\Tekstovna\Tecaji.txt");
                    break;
                }
            case 'M':{
                    menjava(@"c:\Tekstovna\Tecaji.txt");
                    break;
                }
        }
    }
    while (char.ToUpper(izbira) != 'K');
}
```

Vaja:

```
/*Dana je tekstovna datoteka PADAVINE.TXT.  V njej je neznano število stavkov s podatki o
količini padavin v določenem kraju. Vsaka vrstica je sestavljena iz imena kraja in količine
letnih padavin. Med imenoom kraja in količino padavin je ločilni znak |.
Napiši funkcijo za izpis vsebine datoteke na zaslon.
Napiši funkcijo, ki dobi za parameter to datoteko in ki ugotovi ter izpiše skupno količino
vseh padavin!
Napiši še funkcijo, ki vrne naziv kraja z največ padavinami*/

static void Main(string[] args)
{
   string datoteka=@"c:\Tekstovna\Padavine.txt";
   if (!File.Exists(datoteka))
      Console.WriteLine("Datoteka ne obstaja!");
   else
   {
      izpis(datoteka);
      skupajPadavin(datoteka);
      Console.WriteLine("Kraj z največ padavinami: "+najvecPadavin(datoteka));
   }
}

static void izpis(string datoteka)
{
   //vzpostavimo povezavo z datoteko na disku
   FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   StreamReader textIn = new StreamReader(fs);//podatkovni tok za branje

   Console.WriteLine("Vsebina datoteke Padavine.txt");
   string vrstica;
   while (textIn.Peek() != -1)
   {
      vrstica = textIn.ReadLine(); //preberemo stavek iz datoteke
      Console.WriteLine(vrstica);//vrstico izpišemo na zaslon
   }
   textIn.Close();
   fs.Close();
}

static void skupajPadavin(string datoteka)
{
   FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   StreamReader textIn = new StreamReader(fs);//podatkovni tok za branje

   string vrstica;
   double vsota=0;
   while (textIn.Peek() != -1)
   {
      vrstica = textIn.ReadLine(); //preberemo stavek iz datoteke
      string []tabela = vrstica.Split('|'); //stavek razbijemo na posamezne dele, glede na
                                            //ločilni znak |
      vsota = vsota + Convert.ToDouble(tabela[1]);
   }
   Console.WriteLine("Skupna količina padavin: "+vsota);
}

static string najvecPadavin(string datoteka)
```

```
{
   FileStream fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   StreamReader textIn = new StreamReader(fs);//podatkovni tok za branje
   string vrstica, najKraj="";
   double najPad = 0;
   while (textIn.Peek() != -1)
   {
      vrstica = textIn.ReadLine(); //preberemo stavek iz datoteke
      string[] tabela = vrstica.Split('|');
      if (Convert.ToDouble(tabela[1]) > najPad)
      {
         najPad = Convert.ToDouble(tabela[1]);
         najKraj=tabela[0];
      }
   }
   return najKraj;
}
```

Vaja:

V naslednji vaji bomo prebrali podatke iz prej kreiranje tekstovne datoteke **Izdelki.txt**. Podatke bomo shranili v tabelo izdelkov. Vsak izdelek je objekt razreda Izdelek, ki ga moramo seveda najprej deklarirati. Deklariramo ga seveda izven vseh metod, a znotraj imenskega prostora, ali pa znotraj razreda, ki pripada obrazcu, ki ga trenutno obdelujemo.

```
public class Izdelek
{
   public string naziv;
   public string proizvajalec;
   public int komadov;
   public decimal cena;

   public Izdelek() //konstruktor
   { }

}
```

Še koda za branje podatkov iz datoteke in zapis v tabelo objektov tipa Izdelek:

```
string pot = @"C: \Tekstovna\Izdelki.txt";
FileStream fs=new FileStream(pot,FileMode.OpenOrCreate,FileAccess.Read);
StreamReader textIn = new StreamReader(fs);

//enodimenzionalna tabela objektov tipa izdelek
Izdelek [] tabelaizdelkov=new Izdelek[10];

int indeks = 0;//zaporedna štev. izdelka in hkrati zaporedna vrstica v tabeli tabelaizdelkov

while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
{
    string vrstica = textIn.ReadLine();//preberemo celo vrstico

    //v tabelo stolpci zaporedoma zložimo zaporedja znakov med ločili |
    string[] stolpci = vrstica.Split('|');

    Izdelek Izd = new Izdelek();//nov objekt tipa Izdelek

    //objektu izd, ki je izplejan iz razreda Izdelek priredimo vrednosti, ki smo jih izluščili
    //iz prebrane vrstice. To nam je uspelo zato, ker so bili podatki ločeni z ločilom |
    Izd.naziv = stolpci[0];
    Izd.proizvajalec = stolpci[1];
    Izd.komadov = Convert.ToInt32(stolpci[2]);
    Izd.cena = Convert.ToDecimal(stolpci[3]);

    tabelaizdelkov[indeks] = Izd;//objekt izd zapišemo v tabelo izdelkov z ustreznim indeksom
    indeks++;  //povečamo indeks
}
```

Vaja:

V naslednji vaji je prikazana uporaba metode **Seek** razreda **Filestrem**, ki omogoča, da se premikamo po podatkovnem toku. Metoda ima dva parametra. S prvim parametrom povemo, kolikšen nja bo relativni odmik (**offset**) od pozicije, ki jo določimo z drugim parametrom. Drugi parameter (**origin**) določa, ali želimo odmik (ki je podan s prvim parametrom) izvesti od začetka podatkovnega toka, od konca podatkovnega toka ali pa od trenutne pozicije. Izberemo lahko torej eno izmed treh vrednosti, ki pripadajo naštevnemu tipu SeekOrigin, ki ima tri vrednosti: **SeekOrigin.Begin**, **SeekOrigin.Current** in **SeekOrigin.End**.

```
/*V Tekstovno datoteko zapišimo 10 naključnih celih števil. S pomočjo  metode Seek se nato
postavimo na zečetek toka in preberemo zapisane podatke*/

string datoteka = "Stevila.txt";

FileStream fs = new FileStream(datoteka, FileMode.OpenOrCreate,FileAccess.ReadWrite);
StreamWriter textOut = new StreamWriter(fs);  //Tekstovni podatkovni tok
Random naklj = new Random();  //generator naključnih števil
//v datoteko zapišemo 10 celih števil
for (int i = 0; i < 10; i++)
{
    int stevilo = naklj.Next(0, 101);
    textOut.WriteLine(stevilo); //zapis v tekstovno datoteko
    Console.Write(stevilo+", ");
}

//Poskrbimo za fizičen zapis podatkov v toku na disk: podatki se namreč
//dokončno zapišejo na disk šele ko zapremo podatkovni tok, ali pa ko
//uporabimo metodo Flush()
textOut.Flush();  //Fizičen zapis podatkov v toku na disk!!!
StreamReader textIn = new StreamReader(fs);
//z metodo Seek se postavimo na začetek toka fs - SeekOrigin.Begin (TA JE
//OSTAL ODPRT), offset(odmik) od začetka pa je enak 0.
fs.Seek(0, SeekOrigin.Begin);
Console.WriteLine("\nVsebina datoteke: \n");
//while (!textIn.EndOfStream) - while zanke NE moremo uporabiti, ker
//dejansko šele metoda close "zapiše" KONEC datoteke
for (int i = 0; i < 10; i++)
{
    int st = Convert.ToInt32(textIn.ReadLine());
    Console.Write(st + ", ");
}
textOut.Close(); //zapremo podatkovne tokove
textIn.Close();
fs.Close();
```

Naloge:

- Napiši funkcijo, ki dobi za parameter ime poljubne tekstovne datoteke in ki njeno vsebino prikaže na zaslonu!
- Dana je tekstovna datoteka DIJAKI.txt. V vsaki vrstici te datoteke je prvih 30 znakov rezervirano za ime dijaka, naslednjih 15 pa za učni uspeh (od 1 do 5).
  - Koliko dijakov je v datoteki
  - Koliko dijakov ima splošni učni uspeh enak 5
  - Kolikšen je povprečen učni uspeh vseh dijakov
- Kreiraj tekstovno datoteko APJ.TXT. V to datoteko zapiši poljubno število stavkov (bereš jih preko tipkovnice, znak za konec vnosa je prazen stavek!). Vsak stavek naj bo v svoji vrstici, med vrsticami pa naj bo po ena prazna vrstica. Napiši funkcijo, ki dobi za parameter ime te datoteke in ki ugotovi in izpiše, koliko znakov vsebuje celotna datoteka!

- Dana je tekstovna datoteka Naloga.txt. V vsaki vrstici te datoteke so po tri cela števila, med seboj ločena s presledkom. Datoteko obdelaj tako, da za vsako vrstico na ekran izpišeš vsoto vseh treh števil, na koncu pa še skupno vsoto vseh števil!

- V tekstovni datoteki temperature.TXT so shranjeni podatki o temperaturi v določenem kraju. V vsaki vrstici je prvih 20 znakov (desna poravnava) rezerviranih za ime kraja, sledi pa podatek o temperaturi (realno število). Ugotovi povprečno temperaturo, ter izpiši ime kraja z največjo temperaturo!

- V tekstovni datoteki so zapisani podatki o porabi bencina za posamezne tipe vozila. V vsaki vrstici je zapisan tip vozila, nato pa podatek o porabi goriva na 100 km (decimalno število)! Koliko vozil je v datoteki? Ugotovi in izpiši tip vozila z najmanjšo porabo goriva. Podatka otipu vozila in porabi goriva sta razmejena z ločilnim znakom **|**.

- Napišite program, ki izpiše 10 najdaljših besed v poljubni tekstovni datoteki.

- Napiši program, ki v poljubni tekstovni datoteki prešteje vse cifre (znake med 0 in 9) in na koncu izpiše, kolikokrat se vsaka cifra pojavi v tej datoteki. Ime datoteke programu podamo kot parameter ukazne vrstice.

- Napiši program **BrisiKomentarje**, ki z ukazne vrstice sprejme ime datoteke v kateri je zapisan nek izvorni program v **C#**. Program naj v datoteko z enakim imenom, a s končnico **rez** prepiše tiste vrstice iz vhodne datoteke, ki se ne začnejo z znakoma za enovrstični komentar '//'.

- Za poljubno tekstovno datoteko ugotovi in izpiši vse besede iz te datoteke in kolikokrat se posamezna beseda pojavi v tej datoteki. Pri tem ne delaj razlike med malimi in velikimi črkami!