Programski jezik C#

osnovni koraki

Matija Lokar in Srečo Uranič

V 0.9

oktober 2008

Predgovor

Omenjeno gradivo predstavlja prvi del gradiv, namenjenih predmetu Programiranje 1 na višješolskem študiju Informatika. Pokriva le res osnovne začetke programiranja v programskem jeziku C# in sicer sestavljanje enostavnih programov z izpisovanjem, prireditvenim stavkom in branjem vrednosti s tipkovnice.
Pri sestavljanju gradiva sva imela v mislih predvsem začetnike, ki se s programskimi jeziki srečujejo prvič. Zato je veliko zgledov, primerov programov ... Prav tako (namerno) zamolčiva marsikaj (npr. cel kup različnih številskih tipov, dokumetacijske komentarje, prevajanje med tipi z metodami iz družine Convert ...), vrsto stvari pa poenostaviva. Izkušnje namreč kažejo, da predvsem za začetnika te stvari niso pomembne. Še več, pogosto le motijo, saj preusmerjajo pozornost na manj važne stvari.

Gradivo vsekakor ni dokončano in predstavlja delovno različico. V njem so zagotovo napake (upava, da čimmanj), za katere se vnaprej opravičujeva. 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.

Matija Lokar in Srečo Uranič

Kranj, oktober 2008

Zgodovina sprememb

6. 10. 2008: Različica V0.8 – prva, ki je na voljo javno
13. 10. 2008: Različica V0.9

KAZALO

Zakaj učenje programiranja

Pogosta trditev, ki odvrača od učenja programiranja je mnenje, da uporabniku računalnika danes ni potrebno znati programirati. Pravijo: *»Saj imamo za vse, kar želimo narediti z računalnikom, na voljo ustrezna orodja.«* Znanje programiranja je po mnenju mnogih povsem odveč. Razmišljajo: »*Programiranje je le zelo specialistično znanje skupinice strokovnjakov, ki pišejo programe, ki jih potem običajni uporabniki uporabljamo.«*

Vendar tako razmišljanje ni utemeljeno. Razlogov, zakaj je vseeno dobro poznati vsaj osnove programiranja, je veliko:

- učenje programiranja je privajanje na algoritmični način razmišljanja. Mirno lahko trdimo, da je znanje algoritmičnega razmišljanja nujna sestavina sodobne funkcionalne pismenosti. Danes ga potrebujemo praktično na vsakem koraku. Uporabljamo ga:
  - ko sledimo postopku za pridobitev denarja z bankomata;
  - pri vsakakršnem delu z računalnikom;
  - ko se odločamo za podaljšanje registracije osebnega avtomobila;
  - omogoča nam branje navodil, postopkov (pogosto so v obliki "kvazi" programov, diagramov poteka);
  - potrebujemo ga za umno naročanje ali izbiranje programske opreme;
  - da znamo pravilno predstaviti (opisati, zastaviti ...) problem, ki ga potem sprogramira nekdo drug;

- osnovne programske tehnike potrebujemo za pisanje makro ukazov v uporabniških orodjih
- za potrebe administracije – delo z več uporabniki
- za kreiranje dinamičnih spletnih strani
- za popravljanje "tujih" programov

Splošno znani pregovor pravi: »*Več jezikov znaš, več veljaš!«*. Zakaj med "tujimi" jeziki ne bi bil tudi kak programski jezik? Če bi se tolikokrat "pogovarjali" s Kitajcem, kot se z računalnikom, se ne bi naučili nekega skupnega jezika, ki bi ga "obvladala" oba?

"napredni uporabnik" se slej kot prej sreča s programiranjem. In prav z učenjem programiranja najlažje osvojimo ta način razmišljanja.

Zaradi vsega tega je danes za izobraženca skoraj nujno, da je med jeziki, ki jih obvlada, tudi nek programski jezik. Teh je zelo veliko. Med njimi je tudi programski jezik C#, o katerem bo govora tukaj.

Kaj je C# in .NET Framework

C# je objektno orientiran programski jezik (karkokoli pač že to pomeni), ki ga je razvil Microsoft. Jezik izvira pretežno iz programskih jezikov C++, Visual Basic in Java. Trenutno obstaja v različici 3. Namenjen je pisanju programov v okolju .NET in je primeren za razvoj najzahtevnejše programske opreme. Poleg tega je zasnovan na tak način, da ga je mogoče preprosto izboljševati in razširjati, brez nevarnosti, da bi s tem izgubili združljivost z obstoječimi programi.

C# je bil oblikovan za delo z Microsoftovo .NET platformo (.NET Framework). .NET Framework je platforma/knjižnica, ki predstavlja ogrodje za vsa .NET orientirana programska orodja in aplikacije za osebne računalnike, dlančnike, pametne telefone, razne vgrajene sisteme … To ogrodje je sestavni del operacijskega sistema Windows, oziroma ga lahko temu operacijskemu sistemu dodamo. Ogrodje je sestavljeno iz velikega števila rešitev za različna programska področja kot so gradnja uporabniških vmesnikov, dostopanje do podatkov, kriptografija, razvoj mrežnih aplikacij, numerični algoritmi, omrežne komunikacije … Programerji pri razvoju programov metode, ki so uporabljene v teh rešitvah, kombinirajo z lastno kodo.

Razvoj programske rešitve v okolju Microsoft Visual Studio .Net

Za pisanje programov v jeziku C# (in tudi ostalih jezikih v okolju .NET) je Microsoft razvil razvojno okolje, ki se imenuje **Microsoft Visual Studio.NET**. Združuje zmogljiv urejevalnik kode, prevajalnik, razhroščevalnik, orodja za dokumentacijo programov in druga orodja, ki pomagajo pri pisanju programskih aplikacij. Poleg tega okolje nudi tudi podporo različnim programskim jezikom, kot so na primer C#, C++ in Visual Basic.

Microsoft Visual Studio.Net obstaja v več različicah. Za spoznavanje z jezikom C# zadošča brezplačna različica Visual C# Express Edition. Ta podpira le razvoj programov v jeziku C#. Prenesemo jo lahko z Microsoftovih spletnih strani. Ker se točen naslov, na katerem to različico dobimo, občasno spreminja, naj naslov bralec poišče kar sam. Če v poljubni iskalnik vpišemo ključne besede "Visual Studio Express download", bomo različico zagotovo našli. V besedilu bomo več ali manj opisovali v trenutku pisanja najnovejšo različico tega okolja, Visual C# 2008 Express Edition. Vendar praktično vse povedano velja tudi za starejše različice tega okolja. Seveda obstajajo tudi druga razvojna okolja za pisanje programov v C#, na primer okolje **SharpDevelop** (http://www.icsharpcode.net/OpenSource/SD/).

Programske rešitve običajno ne sestavlja le ena datoteka z izvorno kodo, ampak je datotek več. Skupek datotek in nastavitev, ki rešujejo določen problem, imenujemo **projekt**.

Glede na to, za kakšno vrsto programske rešitve gre, imamo v okolju Visual C# Express vnaprej pripravljenih več različnih tipov projektov:

- **Console Application** (ali konzolne aplikacije) – namenjene gradnji aplikacij, ki ne potrebujejo grafičnega vmesnika. Izvajajo se preko ukaznega okna ali kot mu tudi rečemo, konzole (t.i. »DOS-ovskih« aplikacij).
- **Windows Forms Application** (ali namizne aplikacije) – namenjene za gradnjo namiznih aplikacij s podporo grafičnih gradnikov.
- **Windows Presentation Foundation (WPF) Application** – namenjen za gradnjo programov, ki tečejo v okolju Windows, zasnovanih na uporabi najnovejših gradnikov okolja WPF.
- **Windows Presentation Foundation (WPF) Browser Application** – namenjen za programiranje programov, ki tečejo v spletnih brskalnikih (npr. Internet Explorer, Firefox).
- **Class Library** (ali knjižnice) – namenjene gradnji knjižnic razredov.
- **Empty Project** (ali prazen projekt) – namenjen gradnji aplikacij brez vnaprej določenega vzorca.

Izbira projekta določa, kakšno bo vnaprej pripravljeno ogrodje programov, katere knjižnice se bodo naložile in podobno.

Za spoznavanje osnov programskega jezika C# bomo uporabljali več ali manj tip projekta **Console Application**. V nadaljevanju bomo opiali, kako tak projekt ustvarimo.

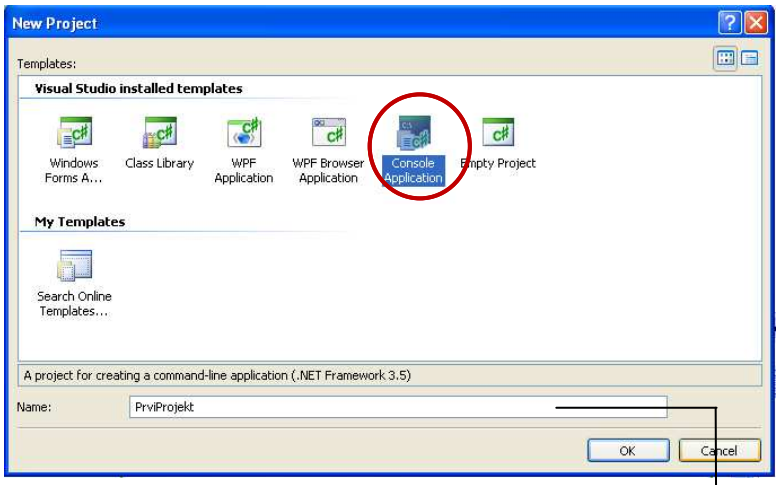
Izdelava programov v VISUAL C#

Razvoj novega projekta v različici Express Edition

Najprej na znan način poženemo okolje **MVCEE** (Microsoft Visual C# 2008 Express Edition) (torej tako, kot poganjamo vse programe v okolju Windows). Po zagonu se odpre začetna stan okolja MVCEE. V zgornjem levem delu je okno, v katerem je seznam projektov, s katerimi smo nazadnje delali. Preostala okna v tem okolju nam služijo za pomoč pri delu in za informiranje o novostih, povezanih z okoljem MVCEE.

Za začetek novega projekta v okolju Visual C# Express Edition odprimo meni **File** -> **New Project**…

Po izvedbi drugega koraka se odpre okno **New Project**. V tem oknu med **Templates** (vzorci) izberemo (kliknemo) tip projekta **Console Application** ter določimo **ime** (Name) projekta.



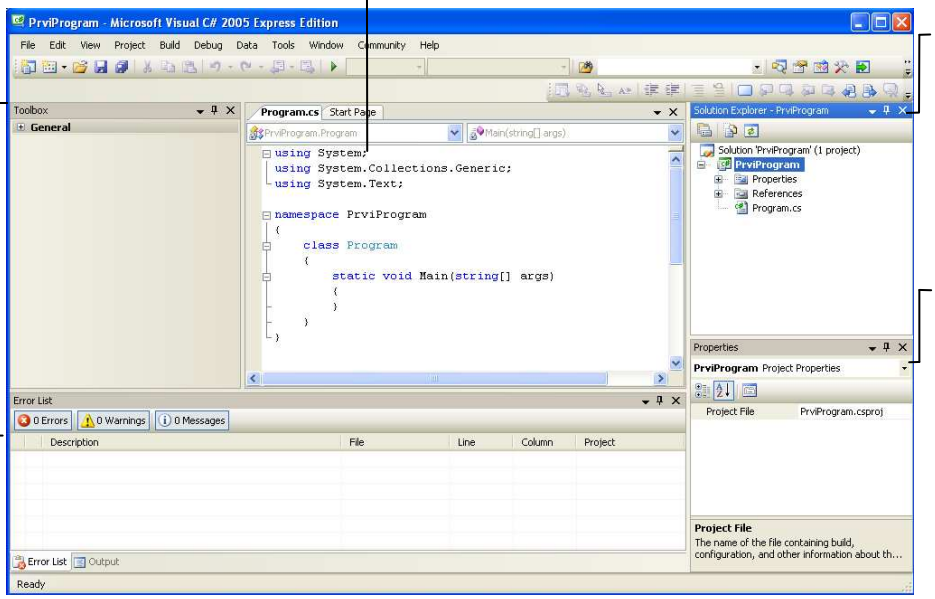
Na dnu okna napišimo še ime naše prve konzolne aplikacije, npr. **Prvi Projekt**. S klikom na gumb OK se ustvari nov projekt.

Pisanje programske kode

Ko smo ustvarili nov projekt, se odprejo **osnovna** okna za urejanje naše prve konzolne aplikacije.

Toolbox (objekti za obrazce)

Editor (urejevalnik -okno za pisanje/urejanje kode)



Solution Explorer (raziskovalec rešitve)

Properties (lastnosti objektov)

Error list (seznam napak), ali pa **Task list** (seznam opravil, potrebnih za dokončanje projekta)

Kratek opis osnovnih delov razvojnega okolja:

- **Editor** ali **urejevalnik** je okno, namenjeno pisanju in urejanju programske kode. Pri pisanju daljših delov programov je zelo priročna načrtovalna črta (outlining). Črto sestavljajo + in – na levi strani urejevalnika. Kliki na te oznake omogočajo skrivanje in prikazovanje delov kode. Pri tem so sestavni deli logični deli kode, kot so posamezne metode, razredi in podobno. V primeru, da je del kode skrit (+ na levi), je naveden le njegov povzetek (začetek, ki mu sledijo …). Pregled vsebine lahko vidimo tako, da se z miško premaknemo na ta povzetek. Če na + kliknemo, se koda prikaže. Obratno s klikom na – kodo skrijemo.

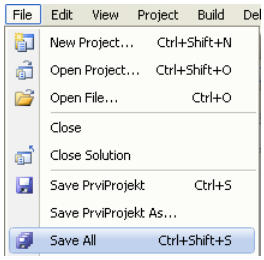
- **Soution Explorer** ali **raziskovalec rešitve** je okno, ki prikazuje in omogoča dostop do vseh datotek znotraj projekta. Datoteke lahko dodajamo, jih brišemo ali spreminjamo.

- **Properties** ali **lastnosti objektov** je okno, v katerem lahko spreminjamo in nastavljamo lastnosti objektov. Pri izdelavi konzolne aplikacije je to okno prazno in ga ne potrebujemo.

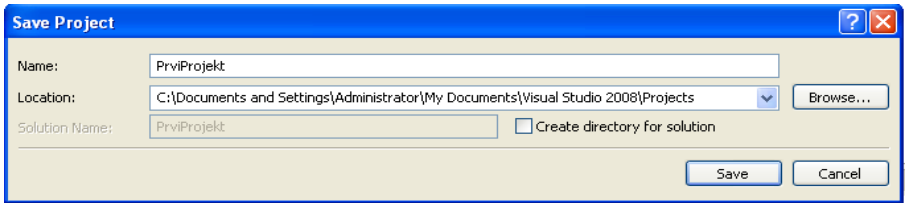
- **Output** ali **izpis** je okno, kjer dobimo sporočila o morebitnih napakah ali opozorilih, ki so se pojavili med preverjanjem kode.

Kot vidimo, nam samo okolje v skladu z izbranim tipom projekta že zgradi osnovno "okostje" programa. V to "okostje" lahko začnemo zapisovati kodo programa.

Priporočljivo je, da nov projekt takoj shranimo. To storimo tako, da se postavimo na meni **File** in kliknemo izbiro **Save All**.



Prikaže se okno **Save Project**. V tem oknu določimo imenik (Location), kjer bodo datoteke novega projekta. Poleg imenika lahko v tem oknu določimo tudi novo ime projekta. To naredimo tako, da v okence **Name** vnesemo novo ime.



Po nastavitvi imena (**Name**) in imenika (**Location**) kliknemo na gumb **Save**. Projekt se shrani in okno **Save** Project se zapre. Po zaprtju okna **Save Project** se ponovno nahajamo v osnovnem oknu.

Sedaj v pripravljeno okostje programa dodamo našo kodo. Nato ponovno shranimo vse skupaj, pri čemer seveda ponujenih nastavitev (ime, imenik …) ni smiselno spreminjati. Nato moramo kodo programa samo še prevesti in zagnati.

"Okostje" programov

Ko v okolju Microsoft Visual C# 2008 Express Edition ustvarimo nov projekt, se v oknu Editor prikaže osnovna zgradba programa v jeziku C#.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Pomen posameznih delov z nas v tem trenutku ni pomemben. Važno je le to, da si zapomnimo, da moramo vse, kar bomo pisali, napisati znotraj metode **Main**, med zavite oklepaje.

```
static void Main(string[] args)
{
  Console.WriteLine("Pozdravljen svet!");
}
```

S tem smo napisali ukaz, ki na zaslon izpiše sporočilo kot je med dvojnimi narekovaji. Pri pisanju lahko opazimo zanimivo lastnost urejevalnika, ki nam je lahko v veliko pomoč. Ko za besedo **Console** napišemo piko, se nam odpre t.i. **IntelliSense** meni (kontekstno občutljiva pomoč). Le-ta nam prikaže ustrezne možnosti, ki so nam na

voljo. S puščicami (če je možnosti preveč, se splača natipkati še prvih par črk ukaza, v našem primeru npr. **WriteLine**) izberemo ustrezno izbiro in pritisnemo na <Enter> ali dvakrat kliknimo z miško na metodo - metoda je s tem dodana programski kodi. Za tem napišimo oklepaj in zaklepaj, znotraj oklepaja pa v narekovajih napišimo besedilo, ki nam ga bo program izpisal na zaslon - v našem primeru ''Pozdravljen svet!''. Za oklepajem dodajmo še podpičje, saj s tem zaključimo ta ukaz (dvopičje je znak za konec stavka).

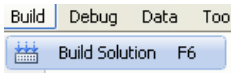
Paziti moramo, da napišemo črke točno tako, kot je navedeno, saj je C# "občutljiv" glede uporabe velikih in malih črk.

Prevajanje in zagon programa

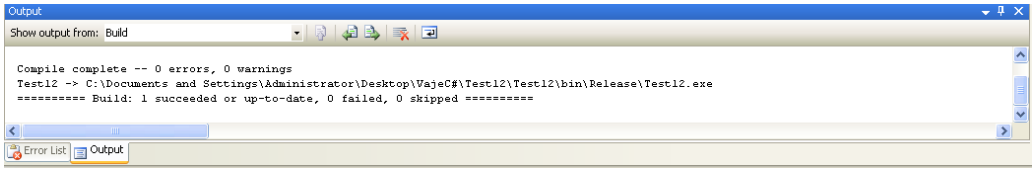
Pred zagonom programa moramo program prevesti. Poglejmo, kako to naredimo v okolju Microsoft Visual C# 2008 Express Edition.

Prevajanje programov

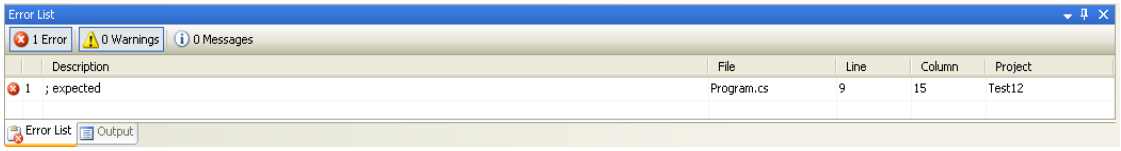
Program po zapisu prevedemo. To storimo tako, da se postavimo na meni **Build** in izberemo **Build Solution**.



Po končanem prevajanju se v oknu **Output** izpiše povzetek.



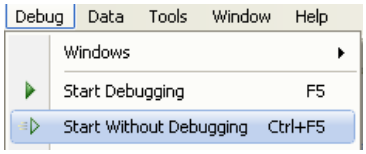
V povzetku je zapisan podatek o številu napak (errors). Če je število napak 0, je program sintaktično pravilen. V primeru, da so v programu napake, se prevajanje prekine in prevajalnik nam sporoči opis sintaktične nepravilnosti in številko vrstice, kjer se napaka nahaja.



Sporočilo o napaki je vidno v oknih **Error List** in **Output**.

Zagon programa

Program, ki je preveden brez sintaktičnih napak, lahko zaženemo. To storimo tako, da se postavimo na meni **Debug** in izberemo opcijo **Start Without Debugging**.



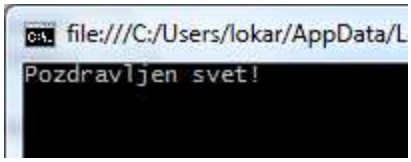
S tem zaženemo program. Če smo dovolj hitri, bomo opazili, da se prikaže konzolno okno, v katerem se prikazujejo rezultati napisanega programa. Ko se namreč program izvede do konca, takoj zapre konzolno okno, ki ga je prej odprl, da bi izpisal rezultate. To nas seveda moti. Zato dopolnimo program tako, da bo tik pred zaključkom "počakal" na pristisk na poljubno tipko. Zapomnimo si, da bomo na koncu vsakega programa (torej tik pred obema }) napisali še

```
  Console.ReadKey();
```

Celoten program je torej

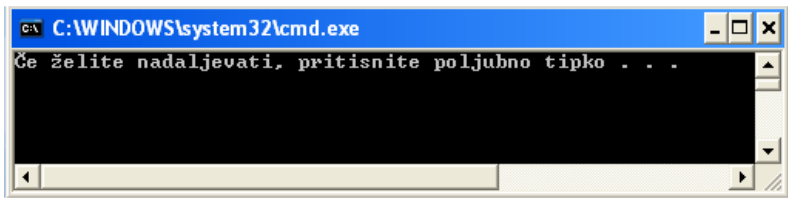
```
static void Main(string[] args)
{
  Console.WriteLine("Pozdravljen svet!");
  Console.ReadKey();
}
```

Ko tak program poženemo, izvede vse ukaze (v našem primeru le tistega, ki nekaj izpiše), razen zadnjega. Zato vidimo sliko



Zadnji ukaz pa zahteva pritisk poljubne tipke. Ko jo pritisnemo, je tudi ta ukaz in s tem tudi program končan. Zato se konzolno okno takrat zapre.

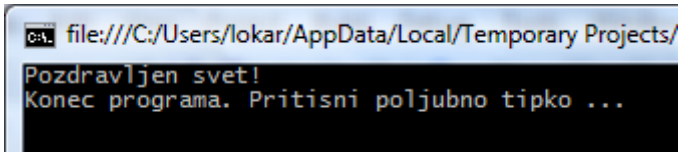
V določenih razvojnih okoljih pa se potem, ko se program izvede do konca, avtomatsko (brez da bi napisali kakšne ukaze) izpiše še obvestilo "Če želite nadaljevati, pritisnite poljubno tipko ...". Sedaj moramo tako kot v našem primeru . Izpis obvestila še nekoliko drugačen, ali pa ga, kot je privzeto v okolju Visual C# 2008 Express Edition sploh ni.



Konzolno okno zapremo s pritiskom poljubne tipke na tipkovnici.

Če želimo, da bi tudi mi dobili ustrezno obvestilo, potem pred ukaz Console.ReadKey(); dodamo še Console.WriteLine("Konec programa. Pritisni poljubno tipko ...");

```
static void Main(string[] args)
{
  Console.WriteLine("Pozdravljen svet!");
  Console.WriteLine("Konec programa. Pritisni poljubno tipko ..." );
  Console.ReadKey();
}
```



Povzemimo in si še enkrat poglejmo, kako dosežemo, da se naš projekt prevede v izvršno datoteko .exe oz. samostojen program. Kliknimo na meni **Build → Build Solution**. Če se bo naša programska koda pravilno prevedla, se bo v statusni vrstici skrajno levo spodaj izpisalo »Build succedded«. Izvršna datoteka se nahaja v bin\Debug mapi v mapi, kamor smo shranili naš projekt. Da pa nam vsakič, ko želimo prevesti kodo, ni potrebno odpirati izvršne datoteke lahko našo aplikacijo zaženemo preprosto s klikom na gumb z zeleno puščico, ki je približno pod izbiro **Tools**. Pred nami se pokaže okno naše aplikacije, ki ga lahko zapremo, minimiziramo, maksimiziramo ali razširjamo. Izvajanje aplikacije prekinemo s klikom na gumb z modro obarvanim štirikotnikom na IDE-ju.

Napake pri prevajanju (Compile Time Error) in napake pri izvajanju (Run Time Error)

Pri pisanju kode se seveda lahko zgodi, da naredimo kakšno napako. V svetu programiranja napakam v programu rečemo tudi **hrošči** (**bugs**). Na srečo ima okolje Visual C# sposobnost, da nekatere vrste napak odkrije.

Poznamo tri vrste napak:

- Napake pri prevajanju (**Compile Time Error**);
- Napake pri izvajanju (**Run Time Error**);
- Logične napake (**Logical Error**).

Prvi dve vrsti napak prevajalnik oziroma okolje odkrije in nam pomaga pri njihovem odpravljanju. Te vrste napak zaradi tega navadno niso problematične, saj jih zaradi pomoči prevajalnika popravimo sorazmerno enostavno in hitro. Bolj problematične so tretje vrste napak – te napake so pomenske, njihovo odpravljanje pa zaradi tega veliko težje in dolgotrajnejše.

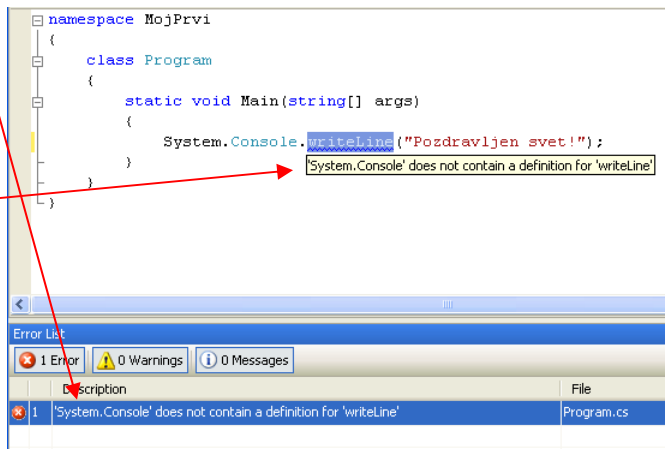
Na primeru naše prve konzolne aplikacije si poglejmo prikaz in odpravljanje prvih dveh vrst napak. Recimo, da smo se pri pisanju programske kode zmotili in namesto pravilnega zapisa

```
Console.WriteLine("Pozdravljen svet!!");
```

zapisali stavek **WriteLine** z malo začetnico takole:

```
Console.writeLine("Pozdravljen svet!!");
```

Ko poženemo prevajanje s klikom na opcijo **Debug → Start Debugging** (tipka F5) ali pa **Start Without Debugging** (Ctrl-F5), nam Visual C# v oknu **Error list** (seznam napak) prikaže za kakšno napako gre in kje se le-ta nahaja (oz. kakšne so napake in kje se nahajajo, če jih je več). Taki vrsti napake pravimo napaka pri prevajanju (**Compile Time Error**), ker se naš program zaradi napake sploh ni mogel prevesti.



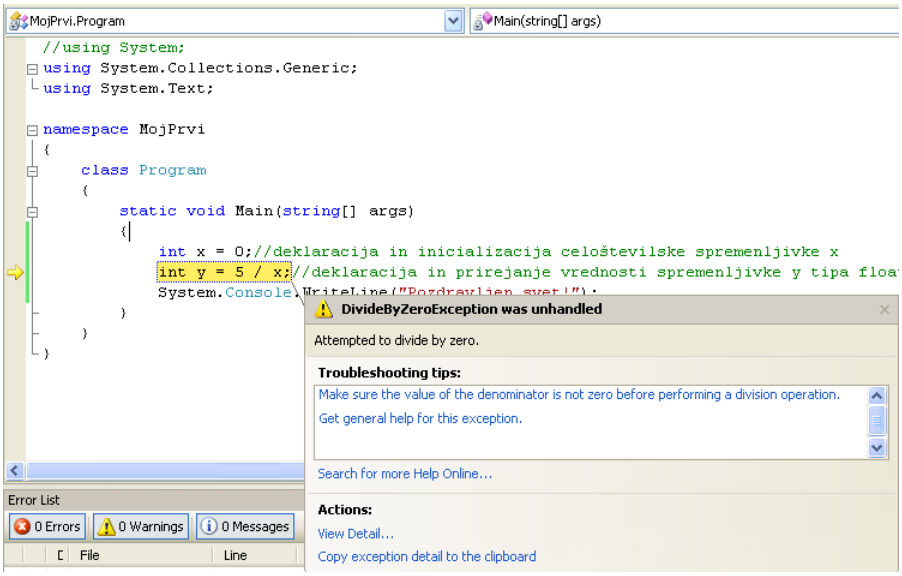
Poleg tega nam Visual C# podčrta del kode, kjer se napaka nahaja. Če se z miško premaknemo nad podčrtani del kode, kjer je napaka, nam Visual C# v okvirčku pod to besedo izpiše za kakšno napako gre.

Napako popravimo in sprožimo novo prevajanje. V kolikor napak ni več, se bo naš program prevedel in zagnal, sicer pa v oknu **Error List** dobimo nov seznam napak. Postopek ponavljamo, dokler ne odpravimo vseh napak.

Zgodi pa se, da se naš program prevede, a kljub temu pride do napake pri samem izvajanju programa. Kot primer take napake poglejmo klasično napako pri deljenju z nič. V našo začetno aplikacijo dodajmo še dva stavka takole:

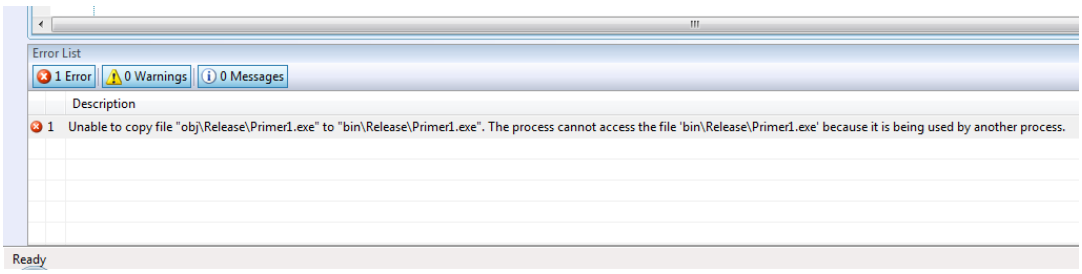
```
int x = 0;//deklaracija in inicializacija celoštevilske spremenljivke x
int y = 5 / x;//deklaracija in prirejanje vrednosti celoštevilčne spremenljivke y
System.Console.WriteLine("Pozdravljen svet!");
```

Ko poženemo prevajanje se program sicer prevede in se prične izvajati. V stavku int y = 5 / x pa smo zahtevali deljenje z 0 (ker je pač vrednost spremenljivke x enaka 0), kar pa je strogo prepovedana operacija. Izvajanje programa se zaradi tega ustavi in na ekranu dobimo približno takole sliko z obvestilom o napaki.



Taki napaki pravimo napaka med izvajanjem programa (Run Time Error). Program moramo seveda popraviti tako, da se izognemo takim operacijam.

Preden nadaljujemo, si oglejmo zanimivo napako. Gremo v urejavalnik, poskusimo prevesti in zagnati program, sistem pa javi napako. Gledamo, gledamo, a napake ne vidimo. Vse je videti OK. A pri poskusu prevajanja sistem vztraja in izpisuje napako.

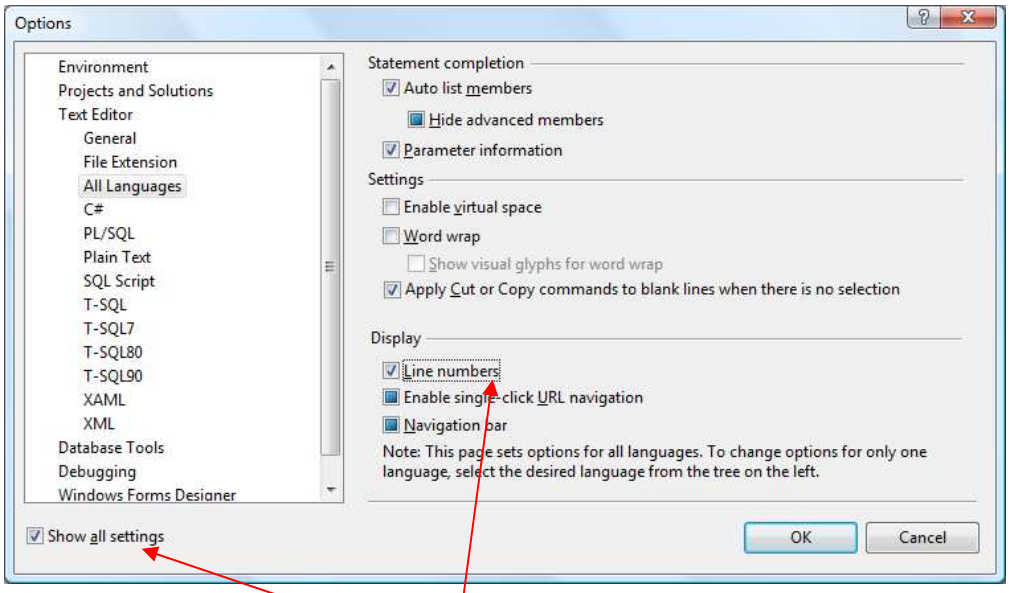


Aha! Vzrok je enostavno ta, da smo pozabili zapreti konzolno okno. Zato se prejšnje različica tega programa "še izvaja" in datoteke exe ne moremo prepisati z novo različico.

Če torej sistem javi tako napako, preverite, če nimate slučajno odprtega kakega nepotrebnega konzolnega okna!

Številke vrstic

Pogosto je koristno, da se ob robu urejevalnika izpisujejo še številke vrstic. Te številke niso del kode. Privzeto se ne prikazujejo, zato moramo okolje ustrezno nastaviti. To storimo tako, da si izberemo Tools>Options



Tam potem odkljukamo **Show All settings** in v izbiri **Text Editor** kliknemo na **All Languages**. V izbirniku na desni strani poskrbimo, da je pred **Line numbers** kljukica.

Komentarji

Komentarji so na poseben način označeni deli besedila, ki niso del programske kode. V komentarje zapisujemo razne opazke ali pa jih uporabljamo za lažje iskanje delov programa in za izboljšanje preglednosti programske kode. Prevajalnik komentarje ne prevaja, tako da ti ne vplivajo na velikost izvršne datoteke. Komentarje v C# označujemo na dva načina:

- s paroma znakov /* in */ - večvrstični komentar;
- z dvema poševnicama // - enovrstični komentar;

```
//Tole je enovrstični komentar;

/*

  Tole pa je večvrstični komentar!

*/
```

Vsak "spodoben" program vsebuje komentarje. Na ta način si olajšamo razumevanje programa, iskanje napak in kasnejše morebitno spreminjanje programa. Komentarje pišemo sproti ob kodi in vedno napišemo kaj v določenem delu programa želimo narediti in zakaj. Kvalitetni komentarji so zelo pomembni, zato bomo večkrat omenili, kako jih napišemo.

Izpisovanje na zaslon v C#

Sedaj, ko vemo, kako okvirno napišemo program, si čimprej oglejmo nekaj enostavnih programov. Pri tem bomo spustili zgornje vrstice (z using in namespace), saj nam bodo vedno ustrezale točno take, kot jih ponudi že samo okolje. Prav tako ne bomo navajali zadnje vrstice z }. Če nam torej okolje ponudi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Primer1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

bomo v naših zgledih sivo označeni del spustili in napisali le

```
    class Izpis
    {
        static void Main(string[] args)
        {
        }
    }
```

ali podobno. Če bo potrebno (zaradi razlage), bomo navedli še številke vrstic, ki jih seveda ne pišemo. Prav tako bomo običajno spustili še dodatek

```
  Console.WriteLine("Konec programa. Pritisni poljubno tipko ..." );
  Console.ReadKey();
```

ki ga moramo v določenih okoljih navesti, da se konzolno okno ne zapre prehitro.

Izpisovanje nizov

Osnovni način izpisovanja na zaslon je z metodo *WriteLine()* oziroma z metodo *Write()*. Obe metodi izpišeta tisto, kar je podano med narekovaji (") znotraj okroglih oklepajev. V našem prvem programu se je tako izpisalo *Pozdravljen, svet!*. Edina razlika med metodama *WriteLine()* in *Write()*je, da prva po izpisu prestavi izpisno mesto (položaj začetka naslednjega izpisa) na začetek nove vrstice, pri drugi pa ostane izpisno mesto v isti vrstici desno od zadnjega izpisanega znaka.

Spremenimo naš prvi program tako, da bo namesto *Pozdravljen, svet!* izpisal *Programski jezik C#*. Spremeniti moramo le tekst med " pri metodi *WriteLine()*.

```
 class Izpis {
   static void Main(string[] args){
   // Program, ki na zaslon izpiše niz "Programski jezik C#".
```

```
        Console.WriteLine("Programski jezik C#");
    } // Main
} // Izpis
```

Prevedemo in poženemo:

```
Programski jezik C#
```

Ne pozabimo – če rezultati prehitro izginejo, program dopolnimo v

```
class Izpis {
  static void Main(string[] args){
  // Program, ki na zaslon izpiše niz "Programski jezik C#".
     Console.WriteLine("Programski jezik C#");
     Console.WriteLine("Konec programa. Pritisni poljubno tipko..." );
     Console.ReadKey();
  } // Main
} // Izpis
```

Kako pa bi napisali vsako besedo v svojo vrstico? Izpisati želimo:

```
Programski
jezik
C#
```

```
1:    class Izpis1 {
2:      static void Main(string[] args) {
3:        Console.WriteLine("Programski");
4:        Console.WriteLine("jezik");
5:        Console.WriteLine("C#");
6:      } // Main
7:    } // Izpis1
```

Prevedemo in poženemo:

```
Programski
jezik
C#
```

Prvi ukaz (stavek), ki se izvede, je stavek v tretji vrstici. Tu metoda *WriteLine()* poskrbi, da se na zaslon izpiše beseda *Programski* in izpisno mesto prestavi na začetek naslednje vrstice. Potem se izvede četrta vrstica, ki izpiše ustrezno besedo in izpisno mesto postavi v naslednjo vrstico. Nato je na vrsti peta vrstica, ki spet izpiše ustrezno besedilo in izpisno mesto prestavi v naslednjo vrstico.

Kaj pa se bo izpisalo v tem primeru?

```
1:    public class Izpis2 {
2:      public static void Main(string[] args) {
3:        Console.Write("Programski ");
4:        Console.Write("jezik");
5:        Console.Write( "C#");
6:      } // Main
7:    } // Izpis2
```

Če niste preskočili razlage o metodi *Write()*, ne bi smelo biti dvomov, kaj se zgodi. Izpiše se:

```
Programski jezikC#
```

Program izpiše vse tri nize enega za drugim, saj smo uporabili metodo *Write()*. Ta ne naredi nič drugega, kot da izpiše željen niz (izpisno mesto ostane za koncem niza). Enak učinek bi dobili, če bi uporabili stavek

```
Console.Write("Programski jezikC#");
```

Zakaj pa med besedama jezik in C# ni presledka? Rekli smo, da metoda *Write()* (in *WriteLine()*) izpiše tisto, kar je med narekovaji. In ker v peti vrstici presledek ni znotraj narekovajev (je med (in "), ga prevajalnik pač ne upošteva in presledek se ne izpiše.

Poskusimo v metodi *Write()* izpisati hkrati več stvari.

```
1:    public class Izpis3 {
2:      public static void Main(string[] args) {
3:        Console.Write("Programski ", "jezik ", "C#");
4:      } // Main
5:    } // Izpis3
```

Prevedemo, a izpiše se le beseda Programski!

Metoda *Write()* (kot tudi *WriteLine())* pričakuje, da bo med oklepaji dobila le en niz! Iz zadrege se rešimo takole:

```
1:    public class Izpis3 {
2:      public static void Main(string[] args) {
3:        Console.Write("Programski " + "jezik " + "C#");
4:      } // Main
5:    } // Izpis3
```

Prevedemo in poženemo:

```
Programski jezik C#
```

Tukaj vidimo, da lahko nize združujemo (stikamo) s tako imenovanim združitvenim operatorjem *'+'*. Na ta način pogosto razdelimo nize v bolj pregledne, krajše nize. Poglejmo primer.

```
1:    public class DolgNiz {
2:      public static void Main(string[] args) {
3:        Console.WriteLine("Peter gre jutri na morje. " +
4:                          "Mislim, da gre z avtomobilom.");
5:      } // Main
6:    } // DolgNiz
```

Prevedemo in poženemo:

```
Peter gre jutri na morje. Mislim, da gre z avtomobilom.
```

Ker prevajalnik pri prevajanju izloči tako imenovane "bele znake" (to so presledki, tabulatorji, prehodi v novo vrsto...), ki niso del nizov, obravnava tretjo in četrto vrstico tako, kot bi jo napisali v isti vrstici.

```
"Peter gre jutri na morje. " + "Mislim, da gre z avtomobilom."
```

Če znotraj oklepajev v metodi *WriteLine()* napišemo izraz (uporabimo torej operatorje – zaenkrat poznamo le operator stika *+*), se najprej izračuna vrednost tega izraza. V našem primeru je to sestavljeni niz. Rezultat izraza metoda *WriteLine()* izpiše na zaslon.

Poglejmo, kako bi z metodo *Write()* izpisali vsako besedo v svoji vrstici. V tem primeru si pomagamo s posebnim znakom1 \n. Ko ta znak "izpisujemo", se to pozna tako, da se na tistem mestu izvede prehod v novo vrsto.

```
1:    public class IzpisNovaVrsta {
2:      public static void Main(string[] args) {
3:        Console.Write("Programski\n");
4:        Console.Write("jezik\nC#\n");
5:      } // Main
6:    } // IzpisNovaVrsta
```

Prevedemo in poženemo:

```
Programski
jezik
C#
```

Prvi ukaz (stavek), ki se izvede, je v tretji vrstici. Tu metoda *Write()* poskrbi, da se na zaslon izpiše beseda *Programski*. Znak \n na koncu niza pomeni, da se izvede preskok v novo vrsto. Naslednja se izvede četrta vrstica. Na zaslon se izpiše beseda *jezik*, zaradi znaka\n pa skočimo v novo vrsto. Nato se na zaslon izpiše beseda *C#*, znak \n pa povzroči še preskok v novo vrsto.

Poglejmo si še naslednji zgled.

Sestavimo program, ki bo na zaslon izpisal naslednji simbol

```
 xxxxxx
x  /\  x
x /  \ x
x/ xx \x
x\ xx /x
x \  / x
x  \/  x
 xxxxxx
```

Ker v konzoli lahko izpisujemo le od leve proti desni in od zgoraj navzdol, bomo simbol izpisali po vrsticah. Uporabili bomo metodo WriteLine(). Pomagali si bomo tudi s kombinacijo \\, s pomočjo katere bomo na zaslon izpisali znak '\'.

```
public class Simbol
{
    public static void Main(string[] args)
    {
        Console.WriteLine(" xxxxxx");
        Console.WriteLine("x  /\\  x");
        Console.WriteLine("x /  \\ x");
        Console.WriteLine("x/ xx \\x");
        Console.WriteLine("x\\ xx /x");
        Console.WriteLine("x \\  / x");
        Console.WriteLine("x  \\/  x");
        Console.WriteLine(" xxxxxx");
    }
}
```

1 Posebni znaki so znaki, ki prek tipkovnice niso dostopni ali pa imajo v kodi poseben pomen. Prikažemo jih s simbolom \ (poševnica). Med pomembnejšimi so znak za prehod v novo vrsto \n, tabulator \t, poševnica \\ in dvojni narekovaj \".

V kodi jezika C# lahko znak '\' zapišemo tudi kot običajen znak, če pred nizom uporabimo znak '@'. S tem znakom povemo, da se morajo vsi znaki v nizu jemati dobesedno. Zgornji program bi torej lahko napisali tudi na naslednji način.

```
C1:   public class Simbol {
C2:       public static void Main(string[] args)    {
C3:           Console.WriteLine(" xxxxxx");
C4:           Console.WriteLine(@"x  /\  x");
C5:           Console.WriteLine(@"x /  \ x");
C6:           Console.WriteLine(@"x/ xx \x");
C7:           Console.WriteLine(@"x\ xx /x");
C8:           Console.WriteLine(@"x \  / x");
C9:           Console.WriteLine(@"x  \/  x");
C10:          Console.WriteLine(" xxxxxx");
C11:      }
C12:  }
```

Razlaga. Ker smo v vrsticah 4 - 9 pred nizom uporabili znak '@', smo s tem povedali, da znak '\' ni posebni znak, ampak so vsi znaki v nizu "navadni". Seveda bi znak '@' lahko uporabili tudi pri izpisu v 3 in 10.

Izpisovanje števil

V prejšnjih primerih smo si pogledali, kako izpisujemo nize. Povejmo še enkrat, da so to poljubna zaporedja znakov med dvojnimi narekovaji. Poglejmo še, kako bi izpisali število.

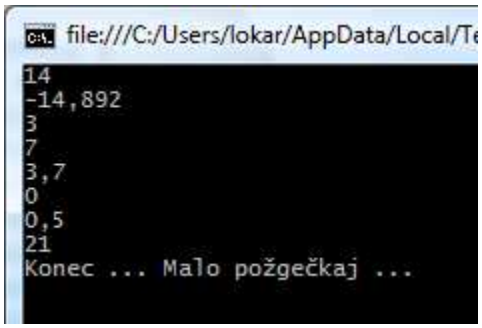
```
1:    public class IzpisStevila {
2:      public static void Main(string[] args) {
3:        Console.WriteLine("12");
4:        Console.WriteLine(12);
5:      } // Main
6:    } // IzpisStevila
```

Prevedemo in poženemo:

```
12
12
```

S tretjo vrstico izpišemo niz, s četrto pa število. Na zaslonu je izpis enak, a če želimo izpisati niz, ga zapišemo med narekovaji. Če pa izpisujemo število, ga navedemo brez narekovajev. Oglejmo si naslednji program:

```
class Stevila {
    static void Main(string[] args) {
      Console.WriteLine(14);
      Console.WriteLine(-14.892);
      Console.WriteLine(1 + 2);
      Console.WriteLine(1 + 2 * 3);
      Console.WriteLine(1.2 + 2.5);
      Console.WriteLine(1 / 2);
      Console.WriteLine(1.0 / 2);
      Console.WriteLine((1 + 2) * (3 + 4));
      Console.WriteLine("Konec ... Malo požgečkaj ...");
      Console.ReadKey();
    } // main
} // Stevila
```



Kot vidimo, lahko s pomočjo programov tudi kaj izračunamo! Pri številih uporabljamo decimalno piko, za množenje je potrebno uporabiti *, / je včasih "čudno" (1/2 je 0, 1.0/2 pa 0.5) ... Podrobnosti si bomo ogledali kasneje, ko si bomo ogledali t.i. podatkovne tipe.

Do razlike pri izpisovanju števil in nizov (ki so števila, npr "12") pride takrat, ko sta število oziroma niz, v katerem je zapisano število, sestavni del nekega izraza. Videli smo že, da je argument metode *WriteLine* (tisto, kar je med oklepaji), lahko tudi izraz. Če napišemo *"12" + "13"* bomo dobili niz *"1213"*. Če pa *+* uporabimo med števili, ima učinek, ki smo ga navajeni iz računstva – sešteje števili. Izračunajmo (in izpišimo) vsoto števil *12* in *13*.

```
1:    public class Racun {
2:      public static void Main(string[] args) {
3:        Console.WriteLine("12 + 13");
4:        Console.WriteLine(12 + 13);
5:        Console.WriteLine("12" + "13");
6:      } // Main
7:    } // Racun
```

Prevedemo in poženemo:

```
12 + 13
25
1213
```

V 3. vrstici izpišemo niz, ki je med oklepaji (*"12 + 13"*). V 4. vrstici pa imamo izraz. Zato se najprej izvede računska operacija (seštevanje), šele nato se izpiše dobljena vrednost (število 25). Tudi v 5. Vrstici imamo izraz, a tokrat gre za stikanje dveh nizov.

Operator *'+'* torej nastopa v dveh vlogah – kot operator seštevanja in za stikanje (združevanje) nizov. Če sta oba operanda števili ali če sta oba operanda niza, je pomen jasen. Kaj pa, če je eden od operandov niz in drugi število? Takrat se število pretvori v niz (*12* v niz *"12"*) in *'+'* je operator združevanja nizov. Za lažje razumevanje uporabe si poglejmo naslednji primer.

```
1:    public class IzpisNizStevilo {
2:      public static void Main(string[] args) {
3:        Console.WriteLine("Stevilo" + 1 + 2);
4:        Console.WriteLine(1 + 2 + "Stevilo");
5:        Console.WriteLine(1 + "Stevilo" + 2);
6:      } // Main
7:    } // IzpisNizStevilo
```

Prevedemo in poženemo:

```
Stevilo12
3Stevilo
```

```
1Stevilo2
```

Poglejmo, zakaj je izpis tak:

3. vrstica. V izrazu nastopata dva operatorja *'+'*. Kot smo navajeni, izvedemo izračun od leve proti desni. Najprej se bo izračunala vrednost (pod)izraza *"Stevilo" + 1*. Ker je prvi operand niz (*"Stevilo"*), drugi pa število *1*, prevajalnik poskrbi, da se število *1* pretvori v niz *"1"*. Dobimo *"Stevilo" + "1"* . Zaradi operatorja za združevanje se ta dva niza stakneta. Rezultat je torej niz *"Stevilo1"*. Sedaj moramo izračunati še *"Stevilo1" + 2*. Ker "seštevamo" niz in število, prevajalnik ponovno poskrbi, da se število *2* pretvori v niz *"2"*. Dobimo *"Stevilo1" + "2"*. Niza se stakneta in rezultat celotnega izraza je *"Stevilo12"*, ki ga metoda *WriteLine* izpiše.

4. vrstica. Ker sta na začetku izraza števili *1* in *2*, operator *'+'* pomeni operacijo seštevanja. Izračuna se vsota *1 + 2*. Dobimo število *3*, ki mu "prištevamo" niz *"Stevilo"*. Prevajalnik poskrbi, da se število *3* pretvori v niz *"3"* in niza se stakneta. Dobimo niz *"3Stevilo"*, ki se izpiše.

5. vrstica. Ker je med številom (*1*) in nizom (*"Stevilo"*) operator *'+'*, prevajalnik število *1* pretvori v niz *"1"*. S stikanjem dobimo niz *"1Stevilo"*. Ker imamo med nizom (*"1Stevilo"*) in številom (*2*) zopet operator *'+'*, prevajalnik število *2* pretvori v niz *"2"* . Zaradi operatorja *'+'* se niza stakneta. Dobimo niz *"1Stevilo2"*, ki se nato izpiše.

Kaj torej velja za vrednost izraza *x + y?*
Če sta x in y niza,
se niza stakneta in dobimo niz.
Če je *x* število in *y* niz,
prevajalnik pretvori število x v niz. Nato se pretvorjeni niz in niz y stakneta in dobimo niz.
Če je *x* niz in *y* število,
prevajalnik najprej pretvori število *y* v niz. Niza se stakneta, dobimo niz.
Če sta *x* in *y* števili,
se števili seštejeta in dobimo število.

Semantične napake

Že prej smo omenili t.i. semantične napake. Program je napisan pravilno (prevajalnik se ne pritoži) in program lahko izvedemo, a rezultati niso taki, kot jih pričakujemo. Poglejmo naslednji program

```
class Stevila {
  static void Main(string[] args) {
    Console.WriteLine("Vsota stevil 2 in 3 je " + 2 * 3);
    Console.WriteLine("Konec ... Malo požgečkaj ...");
    Console.ReadKey();
  } // main
} // Stevila
```

Ko ga izvedemo, dobimo izpis

```
Vsota števil 2 in 3 je 6
```

Kar je očitno narobe. Kje je napaka? Je res to, da smo namesto + napisali *? Kaj pa, če smo želeli v resnici izračunati produkt? Potem napako popravimo z

```
Console.WriteLine("Produkt stevil 2 in 3 je " + 2 * 3);
```

No, kaj pa, če smo res želeli izračunati vsoto? Hitro popravimo v

```
Console.WriteLine("Vsota stevil 2 in 3 je " + 2 + 3);
```

A izpis nas znova preseneti, saj dobimo

```
Vsota števil 2 in 3 je 23
```

Očitno so se inženirji pri Intelu (pa še pri AMD, saj tudi na računalnikih s takim procesorjem dobimo tak rezultat) pošteno zmotili, ko so sestavljali vezje, zadolženo za računanje! Po drugi strani pa, če premislimo razlago, ki smo jo navedli pred tem zgledom, lahko ugotovimo, da smo ga polomili kar mi. In če vrstico popravimo v

```
Console.WriteLine("Vsota stevil 2 in 3 je " + (2 + 3));
```

končno dobimo

```
Vsota števil 2 in 3 je 5
```

Zakaj? Poglejmo, kako sistem "obdela" oba zapisa:

```
"Vsota stevil 2 in 3 je " + 2 + 3
"Vsota stevil 2 in 3 je 2" + 3
"Vsota stevil 2 in 3 je 23"

"Vsota stevil 2 in 3 je " + (2 + 3)
"Vsota stevil 2 in 3 je " + 5
"Vsota stevil 2 in 3 je 5"
```

In nauk te zgodbe? No, v bistvu sta dva:

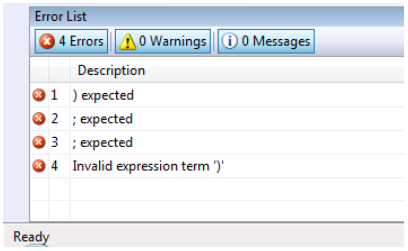
- Ne verjemimo takoj temu, kar računalnik izpiše, ampak premislimo, če je izpisani rezultat smiselen
- Semantične (pomenske) napake je težje odkriti kot sintaktične, saj nihče ne pove, kaj in kje je nekaj narobe

Zgled

Kaj izpiše naslednji program?

```
 1:   class Program {
 2:      static void Main(string[] args) {
 3:       // program nekaj izpiše
 4:        Console.WriteLine(1 + 2 + " krat" +
 5:                          " se je obrnil svet"
 6:                          " za " +  3 +
 7:                          (10 + 50) +
 8:                          " stopinj.");
 9:      }
10:   }
```

Rešitev: Če si dobro pogledal program, si lahko opazil, da ga ne moreš prevesti zaradi sintaktične napake. Prevajalnik bi vrnil naslednje sporočilo



Toda če popraviš program v 5. vrstici tako, da na koncu vrste dodaš +, bi program izpisal »3 krat se je obrnil svet za 360 stopinj.«. Popravljeni program bi se pravilno glasil:

```
 1:   public class Program {
 2:       public static void main(String[] args) {
 3:           // program nekaj izpiše
 4:           Console.WriteLine(1 + 2 + " krat" +
 5:                             " se je obrnil svet" +
 6:                             " za " +  3 +
 7:                             (10 + 50) +
 8:                             " stopinj.");
 9:       }
10:   }
```

Komentar: Metoda Console.WriteLine() pričakuje kot argument (tisto, kar bo izpisala) en niz. Zato se najprej izračuna izraz v oklepajih. Ker so vsi operatorji "+", se računa od leve proti desni. Najprej sešteje 1 + 2. Ker je to celoštevilsko seštevanje, je rešitev 3. Nato poskušamo številu 3 prišteti niz. V tem primeru se 3 spremeni v niz (ni več število) in stakne (zlepi) skupaj z " krat". Tako dobimo niz "3 krat". Tega seštejemo (torej stakemo) z nizom "se je obrnil svet" v niz "3 krat se je obrnil svet". Na podoben način se zgodi z ostalimi deli od 5. do 8. vrstice. Pozorni moramo bit na 7. vrstico na izraz "(10+50) + ". V tem primeru računalnik najprej izračuna kar je v oklepaju, preden to prišteje nizu. Torej program prišteje nizu za izpis vrednost 60.

Oblika programov

Sedaj smo napisali že cel kup programov. Pri tem smo vse pisali v določeni obliki. Tako smo na določenih mestih uporabili presledke, prehode v novo vrsto, zamikanje, znake okoli operatorjev …
Pisanje programov "lepo" ima določen pomen. Je sicer povsem nepomembno za prevajalnik, a zelo pomembno za človeka. Če programe pišemo na določen način (uporabljamo določen stil), se v programu lažje znajdemo, koda nam je bolj pregledna …

Kar se prevajalnika tiče, bi program

```
class Stevila {
  static void Main(string[] args) {
    Console.WriteLine("Vsota stevil 2 in 3 je " + 2 * 3);
    Console.WriteLine("Konec ... Malo požgečkaj ...");
    Console.ReadKey();
  } // main
} // Stevila
```

lahko napisali tudi kot

```
class Stevila {
  static void
Main(string[] args
) {
        Console.WriteLine
("Vsota stevil 2 in 3 je " + 2
* 3);Console.WriteLine("Konec ... Malo požgečkaj ...");      Console.ReadKey()
```

```
; } }
```

ali še kako drugače. A zgornja oblika je za programerja veliko bolj berljiva. Sveda ima vsak pri programiranju svoj stil. Tako bi nekateri npr. zavite oklepaje napisali v svoji vrstici, nekateri okoli operatorjev (+, *, ...) ne spuščajo presledkov, spet tretji med imenom metode in oklepaji naredijo presledek ...

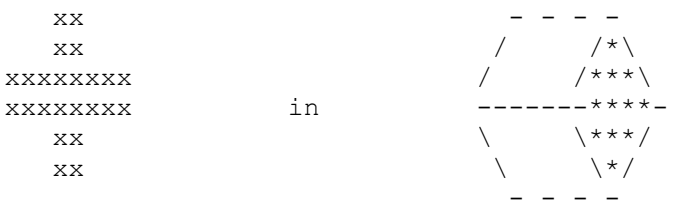
```
class Stevila
{
  static void Main (string[] args)
  {
    Console.WriteLine ("Vsota stevil 2 in 3 je "+2*3);
    Console.WriteLine ("Konec ... Malo požgečkaj ...");
    Console.ReadKey ();
  } // main
} // Stevila
```

Vse te možnosti so stvar osebnega okusa. Dobro pa je, če se vedno držimo svojih "pravil" in vedno pišemo na en način. Tako nam bo lažje pri "branju" programov.

Naloge za utrjevanje znanja

1. Napišite program, ki bo izpisal »Pozdravljena Slovenija ...«.

2. Sestavite program, ki bo na zaslon izpisal naslednja simbola:



3. Napiši program, ki bo izpisal tvoj naslov, kjer sta ime in priimek izpisana v svoji vrstici, ulica in hišna številka v svoji, poštna številka in pošta pa v svoji vrstici.

4. Sestavite program, ki osebne podatke določene osebe na zaslon izpiše v naslednji obliki:

```
Ime in priimek
   Ulica in hišna številka
          Kraj in poštna številka
                 Država
```

   Namig: Pri oblikovanju izpisa osebnih podatkov si pomagajte s tabulatorskim znakom" \t.

5. Napišite program v C#, ki napiše vaši inicialki "na veliko". Npr. takole:

```
*       *   *
**     **   *
* *   * *   *
* * *   *   *
*   *   *   *
*       *   *
*       *   ******
```

6. Poišči napake v programu

```
class Program {
    static void Main(string[] args) {
        Console.WriteLine("Janez Novak");
        Console.WriteLine("Ulica eho 2")
        Console.Writeline("1020 Breskvaland");
    }
}
```

   Namig: Bodi pozoren na podpičja, pravilnost uporabe ukaza za izpis, ... Pri prevajanju naloge ti bo prevajalnik povedal na katerem mestu je opazil napako. Napaka je bodisi tam, bodisi je narejena že prej. Še to – v programu so tri napake.

7. Na http://www.chris.com/ascii/ najdete cel kup zanimivih slik, ki jih lahko naredite s pomočjo izpisovanja z ukazi WriteLine(). Izberite si kakšno sliko in jo narišite.

8. Kaj naredijo naslednji ukazi:

```
Console.WriteLine("To je niz");
Console.WriteLine("Presledek      je del niza, ce je med narekovaji");
```

```
Console.WriteLine ("Presledek      je del niza, le ce
        je med narekovaji"); // napisano v dveh vrsticah!!
Console.WriteLine(     "Matija");
Console.WriteLine("Matija");
Console.WriteLine("Izpis \" narekovaja");
Console.WriteLine("Kaj\nje\nto");
```

9. Poišči napake v programu.

```
class Program {
    static void main(string[] args) {
      Console.WriteLine("Jaz sem pa malo narobe")
      Console.WriteLine("Do sem sta že dve napaki";
      // Od tu dalje je vse OK!
      Console.Write("Press any key to continue . . . ");
      Console.ReadKey();
    }
  }
```

10. Napiši program, ki bo izpisal stavek "Brez C# mi živeti ni!", vsako besedo v svojo vrsto.
    a) Program popravi tako, da bo med prvo in drugo ter drugo in tretjo vrstico še ena prazna vrstica.
    b) Popravi program tako, da boš v njem uporabil samo en ukaz za izpis.

11. Zato, da spečemo 1 kg belega kruha potrebujemo 1 kg bele moke, žličko soli, žličko sladkorja, 42 g kvasa, 3 žlice olja, 100 ml mleka in vodo po potrebi. Želimo sestaviti program, ki prebere, koliko kruha želimo speči (kot decimalno število) in izračuna količino vseh potrebnih sestavin. Tako npr. naj, če uporabnik reče, da bi rad spekel 2.5 kg kruha, program izpiše, da potrebujemo 2.5 kg bele moke, 2.5 žličk soli, 2.5 žličk sladkorja, 105 g kvasa, 7.5 žlic olja, 250 ml mleka in vodo po potrebi.Sestavi program, ki izpiše, koliko sekund ima en teden.

12. Napiši program, ki bo izpisal naslednje vrstice (vsa števila izpisuj kot vrednosti in jih ne vpisuj kot znake. Namesto yy oz. xxxxx napiši ustrezne izraze)

```
Delamo v programskem jeziku "C#".
Danes smo 1\3\1998.
Vsota stevil 12 in 21 je yy.
Do leta 2009 manjka manj kot 80 * 24 * 60 minut, kar je xxxxx.
```

13. Kobe Bryant je visok 6 čevljev in 7 inčev. Koliko je to centimetrov, če veš, da je en čevelj 30.48cm in 1 inč 2.54 cm.

14. Nekateri pravijo, da je prava višina Kobeja 6 čevljev in 6 inčev. Koliko milimetrov pa je potem velik?

Spremenljivke in podatkovni tipi v C#

Spremenljivke

Programi upravljajo s podatki, ki so shranjeni v pomnilniku. V strojnem jeziku se podatki lahko prikličejo samo s klicem na numerični naslov pomnilniške lokacije, kjer je podatek shranjen. V C# se za naslove podatkov uporabljajo imena namesto številk. Programer si mora zapomniti ime, računalnik pa poskrbi za mesto, kje je podatek shranjen. Takšno ime (naslov), ki služi za klic podatkov shranjenih v spominu, se imenuje **spremenljivka**.

Spremenljivko si lahko predstavljamo kot tablo, na kateri je zapisan nek podatek. Spremenljivka se neposredno nanaša na tablo in posredno na podatek. Med izvajanjem programa se podatek na tabli lahko spreminja. Na tabli (v spremenljivki) bodo različne vrednosti (podatki), toda vedno bo to ista tabla. Vedno lahko pogledamo, kakšno vrednost ima podatek na tabli.

Vsako spremenljivko pred prvo uporabo najavimo, ali, kot rečemo, deklariramo. S tem v pomnilniku rezerviramo prostor ustrezne velikosti, ki je določen glede na tip spremenljivke. Kaj je tip, si bomo ogledali v nadaljevanju. Zaenkrat povejmo le, da z njim določimo, kakšne vrednosti lahko hranimo v spremenljivki.

Vsako v programu uporabljeno spremenljivko moramo najaviti oziroma deklarirati. Navadno vse spremenljivke deklariramo skupaj, pred ostalimi stavki (takoj za začetkom metode main).

Deklaracijski stavek

Spremenljivke najavimo z **deklaracijskim stavkom**. Ta je oblike

```
podatkovniTip imeSpremenljivke;
```

Primeri:

```
int stevilo;
double stranica;
string niz;
```

Ko pišemo programe, običajno na začetku deklariramo vse spremenljivke, za katere predvidevamo, da jih bomo potrebovali. Če določene deklarirane spremenljivke nikoli ne uporabimo, ob prevajanju prevajalnik za jezik C# izpiše opozorilo.

```
c:\documents and settings\administrator\my documents\visual studio projects\
consoleapplication1\avtomobili.cs(11,8): warning CS0168: The variable 'avto' is declared but never
used
```

Opozorila nam sicer ni potrebno upoštevati (program se je vseeno prevedel), vendar praviloma nima smisla, da imamo v programih nepotrebne spremenljivke.

Spremenljivke istega tipa lahko najavimo z istim deklaracijskim stavkom.

```
podatkovniTip spremenljivka1, spremenljivka2, spremenljivka3;
```

Primeri:

```
int enice, desetice, stotice;
double stranica_a, stranica_b, stranica_c;
string niz1, niz2, niz3;
```

Spremenljivki ob najavi lahko priredimo začetno vrednost.

```
podatkovniTip spremenljivka1 = začetnaVrednost;
```

Primeri:

```
int stevilo = 0;
char znak = 'A';

string niz = "Vhodni niz";
```

Spremenljivko deklariramo samo enkrat. Kot smo omenili, to običajno naredimo na samem začetku, ni pa to nujno. Obvezno pa mora biti spremenljivka deklarirana pred prvo uporabo, drugače nam prevajalnik javi napako.

Imena spremenljivk

Vsem spremenljivkam moramo določiti ime, da se lahko kasneje nanj sklicujemo in tako pridemo do vrednosti, ki jo posamezna spremenljivka vsebuje. Sama imena spremenljivk morajo biti smiselna. To pomeni, da je iz imena razvidno, kaj pomeni podatek, ki ga hranimo v spremenljivki. Uporaba takih imen nam olajša razumevanje programske kode.

Prva črka v imenu spremenljivke naj bo poljubna **mala** črka abecede[2]. Tej črki sledi poljubno zaporedje črk (malih in velikih), števk in podčrtajev (_). Tako so imena npr. *vhodni_niz, izhodniNiz, niz1, niz_2* ipd. Še enkrat povejmo, da spremenljivkam damo **smiselna imena, ki povedo, kaj je tisto, kar hranimo v spremenljivki.**

Primeri:

```
stranica
znak
naslov
```

Če je ime sestavljeno iz več besed, ne smemo uporabljati presledkov. Zato ga zaradi boljše čitljivosti zapišemo bodisi tako, da vsako naslednjo besedo začnemo z veliko črko (*dolgoIme*), ali pa besede povežemo s podčrtajem *(dolgo_ime)*. Znotraj programa zaradi preglednosti uporabljajmo samo en način.

Primeri:

```
dolzinaNiza
barvaStranice
ploscina_trikotnika
```

Ne pozabimo, da C# loči med malimi in velikimi črkami. Zato so *ime*, *Ime*, *iMe*, *IME* štiri različna imena.

Pri imenovanju spremenljivk moramo upoštevati določena pravila. Tu bomo navedli poenostavljeno različico teh pravil:

- prvi znak mora biti črka;

[2] Pravila jezika C# dopuščajo določene druge znake, a ... .

- ime je lahko sestavljeno le iz črk angleške abecede[3], števk in znaka podčrtaj;
- ne sme biti enako drugemu imenu;
- ne sme biti enako rezerviranim besedam[4].

V obeh jezikih velja, da se pri imenih upošteva (razlikuje), ali je črka mala ali velika. Navedimo še nekaj dodatnih nasvetov za poimenovanje spremenljivk v jeziku C#:

Ime je sestavljeno iz ene besede. Če je besed več, jih zlepimo. Zlepimo jih tako, da naslednjo besedo začnemo z veliko začetnico, nap. vsotaSodihStevil.
Podajanje tipa spremenljivk v imenu (t.i. madžarska notacija, na primer intVsota ali int _vsota) ni zaželeno.
Ne uporabljamo imen, ki se med seboj razlikujejo le v uporabljenih velikih oz. malih črkah. Tako ni dobro, da bi imeli v programu tako spremenljivko Vsota kot tudi spremenljivko vsota.

Prireditveni stavek

Vrednost, zapisano v spremenljivki, spreminjamo (ali nastavimo prvič, če tega ob deklaraciji nismo naredili) s pomočjo prireditvenega stavka. Prireditveni stavek vsebuje prireditveni operator (=). Ta priredi vrednost desne strani spremenljivki na levi strani.

```
x = 10;
starost = 25 + 2 * 8;
```

Prireditven stavek je oblike

```
spremenljivka = izraz;
```

Tu je *izraz* lahko konstanta, ki jo želimo prirediti spremenljivki ali pa predstavlja izraz z operatorji, metodami, ...

Izračuna se vrednost izraza. Dobljena vrednost se shrani v spremenljivko. Če spremenljivka nastopa v izrazu, pomeni, da vzamemo vrednost, ki jo hranimo v spremenljivki. Denimo, da v spremenljivki x hranimo število.

```
y = 3 * x + 5;
```

Pomen tega stavke je:

Izračunamo izraz: 3 krat število, ki je shranjeno v x in to povečamo za 5. Dobljeni rezultat shranimo v y.

Primer:

```
1  x = 5;
2  y = x + 7;
3  z = x + 8 + y;
```

1. vrstica. Spremenljivki *x* priredimo vrednost *5.*
2. vrstica. Ta prireditveni stavek priredi spremenljivki *y* vrednost izraza na desni. Najprej izračunamo desni del. Ker smo v prejšnji vrstici priredili spremenljivki *x* vrednost *5*, se k *5* prišteje *7*. Na desni strani dobimo *12*. Ta vrednost se zapiše v spremenljivko *y* ali kot tudi rečemo, se priredi spremenljivki *y.*

3 V imenu spremenljivk se po dogovoru ne uporabljajo šumniki, čeprav C# podpira poljubne znake v kodi UNICODE.
4 Rezervirane besede so besede, ki imajo v programskem jeziku vnaprej določen pomen, na primer `else`, `if`, `class`, `void` ...

3. vrstica. Tako kot v drugi vrstici se tudi tu najprej izračuna vrednost izraza na desni strani. Vrednosti *5* se prišteje *8* in tej vrednost spremenljivke *y.* Ta je 12. Vrednost izraza na desni strani prireditvenega stavka je torej *25* in spremenljivki *z* se priredi vrednost *25*.

Seveda smo morali prej vse spremenljivke (x, y in z) deklarirati in povedati, da v njih hranimo števila.

Če torej uporabimo ime spremenljivke na desni strani (v izrazu), mislimo na vrednost, shranjeno v tej spremenljivki. Tako *a = b* ne pomeni, da je *a* enako *b*, ampak da vrednost shranjeno v *b* priredimo *a*–ju. V spremenljivko *a* torej shranimo vrednost, kot je v spremenljivki *b*.
Če napišemo

```
x = x + 1;
```

to pomeni, da vrednost shranjeno v x povečamo za 1. Zakaj? Poglejmo, kako se ta stavek izvede. Najprej izračunamo izraz, torej tisto, kar je shranjeno v x, povečamo za 1. Dobljeni rezultat shranimo v x. To, da je prej pri računanju nastopala spremenljivka x, sploh ni pomembno.

Izpisovanje spremenljivk

Kaj se zgodi, če napišemo

```
Console.WriteLine(x);
```

kjer je x neka spremeljivka, v kateri hranimo število?
Ukaz pomeni izpiši vrednost izraza. Vrednost izraza x pa je vrednost spremenljivke x, torej število. Že prej pa smo videli, da se ob izpisovanju število pretvori v niz (zapis števila kot zaporedja znakov). Zato gornji ukaz izpiše kot niz tisto število, ki je v tem trenutku v spremenljivki x.
Če napišemo

```
Console.WriteLine(x + 1);
```

se najprej izračuna se vrednost izraza (tisto, kar je v x, povečamo za 1). Dobljena vrednost je število. To se pretvori v niz, ki se izpiše.
Seveda lahko v izrazih kombiniramo nize, spremenljivke in števila. Če torej napišemo

```
int x = 10;
Console.WriteLine(x + " + " + " 1 = " + x + 1);
```

bo torej izpis

```
10 + 1 = 11
```

Zgledi

Oglejmo si nekaj enostavnih zgledov. Uporabljali bomo spremenljivke tipa int, v katere lahko shranimo cela števila. Več o tem tipu bomo povedali v nadaljevanju.

Pleskanje stanovanja

To jesen ste porabili veliko časa in denarja za preurejanje svojega stanovanja in do popolnosti vam manjka le še na novo prepleskana dnevna soba. Ta meri 6 x 3m. Je pa precej visoka, kar 3m. V sobi imate tudi dve veliki okni s skupno površino 5 m2 (ki jih ne bomo pleskali in moramo njihovo površino odšteti od površine vseh sten in

stropa). Z enim kilogramom barve prepleskate 8 kvadratnih metrov. Koliko kilogramov barve morate najmanj kupiti, če nameravamo nanesti dva nanosa barve?

Namig: Najprej bomo izračunali površino vseh sten in stropa. Od tega bomo odšteli površino oken. To bomo izpisali. Nato bomo površino množili s številom nanosov in delili z 8. Za vsak primer bomo še prišteli 1l barve.

Rešitev:

```
class Program {
    public static void Main(string[] args) {
        // podatki
        int visina = 3;
        int sirina = 3;
        int dolzina = 6;
        int povrsinaOken = 5;
        int prekrivnostBarve = 8; // koliko m2 z enim litrom
        int steviloNanosov = 2;

        //izračun površine za barvanje
        int povrsina = dolzina * sirina + //strop
            visina * sirina * 2 + //dve ožji steni
            visina * dolzina * 2 //dve širši steni
            - povrsinaOken; // oken ne barvamo

        Console.WriteLine("Površina sten za barvanje: " + povrsina);

        //izpis in izračun količine barve
        int kolicinaBarve = povrsina * steviloNanosov / prekrivnostBarve + 1;
            // +1 - da zaokrožimo navzgor
        Console.WriteLine("Potrebna količina barve za dva premaza: " +
                          kolicinaBarve + " litrov");
    }
}
```

Hišnik čisti bazen

Hišnik mora vsak mesec očistiti bazen. Da ga lahko očisti, mora najprej iz njega izčrpati vodo. Ker je bolj lene sorte, bi med iztakanjem vode (ki traja kar nekaj časa) raje odšel v bližnjo kavarno na pogovor s prijateljem, namesto da bi stražil bazen. Napiši mu program, ki bo za bazen velikosti 2.3m x 1.6m x 9.5m izračunal, koliko sekund se bo praznil bazen, če vsako sekundo izteče iz bazena 23l vode. Če si že pozabili: 1dm3 vode = 1l vode.

Namig: Izračunali bomo volumen bazena kar v kubičnih decimetrih. Če bomo volumen delili s 23, bomo dobili število sekund.

Rešitev:

```
class Program{
    public static void Main(string[] args) {
         Console.WriteLine("Praznjenje bazena");

         int sirina = 23; //mere v dm
         int globina = 16;
         int dolzina = 95;
         int prazni = 23; // koliko litrov na sekundo

         int volumen = sirina * globina * dolzina;
         int sekunde = volumen / prazni; // koliko sekund se prazni

         Console.WriteLine("\n   Bazen se prazni " + sekunde
                           + " sekund.");

         Console.Write("\n\nTipka za nadaljevanje . . . ");
```

```
            Console.ReadKey();
        }
    }
```

Konstante

V določenih primerih potrebujemo spremenljivke, katerih vrednost se med izvajanjem programa ne spreminja. Gre za spremenljivke s konstantno vrednostjo ali krajše konstante. Namesto konstant bi lahko na tistem mestu v izrazu zapisali to vrednost. Toda tega se izogibamo. Na ta način namreč vemo, za kaj pri določeni vrednosti gre. Tako lahko ločimo med različnimi uporabami te konstante (npr. 20, ki pomeni starost in 20, ki pomeni ceno izdelka). Konstantne spremenljivke se od ostalih spremenljivk ločijo v deklaraciji.
Če torej spremenljivki ne želimo spreminjati vrednosti, že ob deklaraciji povemo, da je **konstanta**. To naredimo tako, da ob deklaraciji pred tipom spremenljivke dodamo besedo *const*.
Imena konstant pogosto napišemo z velikimi tiskanimi črkami, zato da jih ločimo od spremenljivk.

```
const podatkovniTip IME_KONSTANTE = vrednostKonstante;
```

Primera:

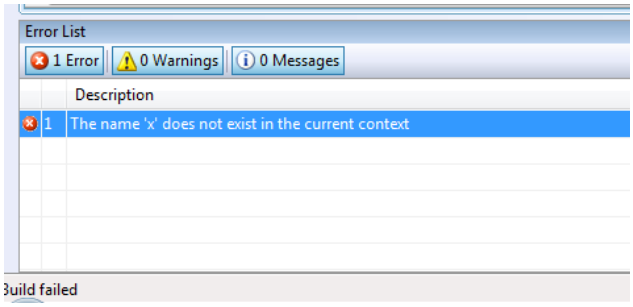
```
const int POVECAJ_ZA = 4;
const double PI = 3.14;
```

Napake pri uporabi spremenljivk in konstant

Poglejmo si naslednji program.

```
1:    class Spremenljivka {
2:      static void Main(string[] args) {
3:        Console.WriteLine(x);
4:      } // main
5:    } // Spremenljivka
```

Ko ga prevedemo, prevajalnik javi:



Ker spremenljivke *x* pred uporabo nismo najavili, prevajalnik javi napako, saj je ne pozna.

Popravimo program.

```
1:    class Spremenljivka {
2:      static void Main(string[] args) {
3:        int x;
4:
5:        Console.WriteLine(x);
6:      } // main
7:    } // Spremenljivka
```

Ko program skušamo prevesti, prevajalnik še vedno javlja napako:

```
Error     1     Use of unassigned local variable 'x'
```

Zakaj se program ne prevede? V 3. vrstici smo za spremenljivko *x* rezervirali prostor in določili, da je tipa *int*. Prevajalnik se pritožuje, ker ji pred uporabo (v 5. vrstici) nismo določili vrednosti. Program popravimo tako, da spremenljivki določimo vrednost.

```
1:    class Spremenljivka {
2:      static void Main(string[] args) {
3:        int x;
4:        x = 5;
5:
6:        Console.WriteLine(x);
7:      } // main
8:    } // Spremenljivka
```

Program prevedemo in poženemo:

```
5
```

Spremenljivki *x* iz prejšnjega programa priredimo novo vrednost *7*.

```
1:    class Spremenljivka {
2:      static void Main(string[] args) {
3:        int x;
4:        x = 5;
5:
6:        int x = 7;
7:        Console.WriteLine(x);
8:      } // main
9:    } // Spremenljivka
```

Ko prevedemo ta program, prevajalnik javi:

```
Error 1 A local variable named 'x' is already defined in this scope
```

Ker smo v 6. vrstici ponovno definirali spremenljivko *x*, ki je že definirana v 3. vrstici, se prevajalnik pritoži. Program popravimo takole:

```
1:    class Spremenljivka {
2:      static void Main(string[] args) {
3:        int x = 5;
4:        Console.WriteLine(x);
5:
6:        x = 7;
7:        Console.WriteLine(x);
8:      } // main
9:    } // Spremenljivka
```

Program prevedemo in poženemo:

```
5
7
```

V 3. vrstici deklariramo spremenljivko *x* in ji priredimo vrednost *5*. Nato v 4. vrstici izpišemo trenutno vrednost spremenljivke *x*. V 6. vrstici priredimo spremenljivki *x* novo vrednost *7*. V 7. vrstici se izpiše vrednost spremenljivke *x*, ki je sedaj *7*.

Spremenimo programa tako, da bo spremenljivka konstanta.

```
1:    class Konstanta {
2:                                          static void main(String[] args) {
3:        const int X = 5;
4:
5:        Console.WriteLine(X);
6:      } // main
7:    } // Konstanta
```

Program prevedemo in poženemo:

```
5
```

Tu je bilo vse v redu. Kaj pa drugi program?

```
1:    class Konstanta {
2:      static void Main(string[] args) {
3:        const int X = 5;
4:        Console.WriteLine(X);
5:
6:        X = 7;
7:        Console.WriteLine(X);
8:      } // main
9:    } // Konstanta
```

Ko program prevedemo, prevajalnik javi:

```
Error 1 The left-hand side of an assignment must be a variable property
or indexer
```

Zakaj v tem primeru prevajalnik javi napako? Kot smo povedali na začetku poglavja, konstante uporabljamo, ko spremenljivkam ne želimo spreminjati vrednosti. Če to kljub temu storimo, nas prevajalnik na to opozori.

Podatkovni tipi

Podatkovni tip pove prevajalniku, koliko pomnilnika naj rezervira za spremenljivko in katere so dovoljene operacije, ki jih lahko s temi spremenljivkami izvajamo.

Nizi

Če želimo kak niz (torej zaporedje znakov med dvema dvojnima narekovajema ") shraniti v spremenljivko, to spremenljivko deklariramo, da je tipa string. Podatkovni tip string (niz) torej označuje zaporedje znakov.

```
string tipVozila;
tipVozila = "Lokomotiva";
string imeOsebe = "Andrej";
```

Kot vemo, lahko z operatorjem + nize stikamo. Če se torej izvedejo stavki

```
string regObmocje = "KR";
string stevilka = "12-13U";
string registrska = regObmocje + " " + stevilka;
Console.WriteLine("Moj avto ima registrsko številko:" + registrska);
```

se bo izpisalo

```
Moj avto ima registrsko številko:KR 12-13U
```

Cela števila

Cela števila (ang. integer) označimo s tipom **int**. Seveda ne gre za poljubna cela števila, kot jih poznamo iz matematike, saj v končni pomnilnik računalnika ne moremo shraniti neskončno velikega števila. Spremenljivke tipa *int* v Javi tako lahko zavzamejo vrednosti med približno -2 milijardi in + 2 milijardi (natančneje med -2.147.483.648 in 2.147.488.647).

Nad celimi števili lahko izvajamo vse osnovne računske operacije (seštevanje, odštevanje, množenje in deljenje). Za množenje uporabljamo znak *. Vsota, razlika in produkt dveh celih števil so definirani na običajen način, deljenje pa je celoštevilsko. To pomeni, da pri deljenju dveh celih števil C# zanemari vsa decimalna mesta (tako je kvocient dveh celih števil vedno celo število). Poleg deljenja pozna C# še operator **%**, ki vrne ostanek pri deljenju (zopet celo število).

OPERATOR	POMEN	PRIMER	REZULTAT
+	vsota	3 + 2	5
-	razlika	3 - 2	1
*	množenje	3 * 2	6
/	celoštevilsko deljenje	3 / 2	1
%	celoštevilski ostanek pri deljenju	3 % 2	1

Zgledi

Sledenje delu programa

Kakšno vrednost imajo spremenljivke a, b in c, ko se izvedejo vsi stavki?

```
1:   int a = 1;
2:   int b = 2;
3:   int c;
4:
5:   c = 3 + 5;
6:   a = b + 1;
7:   b = a + c;
```

Rešitev: Na koncu ima a vrednost 3, b ima vrednost 11 in c vrednost 8.

Komentar: V 1. in 2. vrstici sta deklarirani spremenljivki a in b, kjer a dobi vrednost 1 in b dobi vrednost 2. V 3. vrstici je deklarirana spremenljivka c, vendar ji ni bila prirejena nobena vrednost. V 5. vrstici je spremenljivka c dobila vrednost izraza 3 + 5, torej 8. V 6. vrstici je določena nova vrednost a, ki je vrednost, shranjena v spremenljivki b, povečana za 1. To pomeni, da a dobi vrednost 3 (2 + 1). V 7. vrstici spremenljivka b dobi vrednost vsote števil, shranjenih v a in c oziroma 3 + 8 = 11.

Če pogledamo izvajanje programa s pomočjo tabele, bi ta izgledala takole.

vrstica	ukaz	a	b	c
1:	int a = 1;	1		
2:	int b = 2;	1	2	
3:	int c;	1	2	
4:		1	2	
5:	c = 3 + 5;	1	2	8
6:	a = b + 1;	3	2	8
7:	b = a + c;	3	11	8

Tromestno število izpisano po vrsticah

Tromestno število 376 izpišimo tako, da bodo stotice izpisane v svoji vrstici , nato v svoji vrstici desetice in nato še enice. Programa napišimo tako, da bo za enak izpis nekega drugega tromestnega števla potrebno spremeniti le število 376 v drugo, ostale spremembe v programu pa ne bodo potrebne.

Namig: S pomočjo celoštevilskega deljenja z 10 in 100 in ostanka pri deljenju z 10 ali 100 bomo iz števila izluščili posamezno števko.

Rešitev:

```
// trimestno število izpišimo navpično

class Trimestno {
    static void main(String[] args) {
        int stevilo = 376;
        // izračunamo in izpišemo stotice
        int stotice = stevilo / 100;
        Console.WriteLine(stotice);
        // izračunamo in izpišemo desetice
        int desetice = (stevilo % 100) / 10;
        Console.WriteLine(desetice);
        // izračunamo in izpišemo enice
        int enice = stevilo % 10;
        Console.WriteLine(enice);
    }
}
```

Obrnjeno število

Napišimo program, ki bo celo število 123 izpisal obrnjeno, torej kot 321. Želimo, da izpis zgleda tako:

```
Stevilo 123
Obrnjeno stevilo 321
```

Seveda bomo to naredili tako, da bo program deloval pravilno, če bomo uporabili poljubno trimestno število, in ne le za število 123.

```
1:    class ObrnjenoStevilo {
2:      static void Main(string[] args) {
3:        int trimestnoStevilo = 123;
4:        int enice = trimestnoStevilo % 10;
5:        int dvomestnoStevilo = trimestnoStevilo / 10;
6:        int desetice = dvomestnoStevilo % 10;
7:        int stotice = dvomestnoStevilo / 10;
8:
```

```
 9:         Console.WriteLine("Stevilo " + trimestnoStevilo);
10:         Console.WriteLine("Obrnjeno stevilo "
11:                           + enice + desetice + stotice);
12:       } // main
13:     } // ObrnjenoStevilo
```

Kaj bo program izpisal? Pa poglejmo:

Spremenljivki *tromestnoStevilo* priredimo vrednost *123*, torej število, ki ga želimo izpisati v obratnem vrstnem redu.
Sedaj moramo priti do stotic, desetic in enic(vrstice 4. – 7.). Pomagamo si z operatorjema */* in *%*. Najlažje pridemo do enic. Izračunamo ostanek pri deljenju števila z *10* (vrstica 4). Sedaj enic ne potrebujemo več. Zato jih "odrežemo" in dobljeno shranimo v spremenljivko *dvomestnoStevilo* (vrstica 5). Desetice prvotnega števila so v novem številu enice. Te pa že znamo "izluščiti" – zopet si pomagamo z ostankom pri deljenju z *10*. Stotice lahko izračunamo bodisi iz dvomestnega števila z deljenjem z *10* (vrstica 7) ali pa iz trimestnega števila z deljenjem s *100* (*trimestnoStevilo / 100*).
Na koncu še dobljene vrednosti izpišemo.

Seveda zapisani način ni edini. Lahko bi napisali tudi takle program:

```
 1:   public class ObrnjenoStevilo1 {
 2:     public static void Main(string[] args) {
 3:       int trimestnoStevilo = 123;
 4:       int stotice = trimestnoStevilo / 100;
 5:       int dvomestnoStevilo = trimestnoStevilo % 100;
 6:       int desetice = dvomestnoStevilo / 10;
 7:       int enice = dvomestnoStevilo % 10;
 8:
 9:       Console.WriteLine("Stevilo " + trimestnoStevilo);
10:       Console.WriteLine("Obrnjeno stevilo "
11:                         + enice + desetice + stotice);
12:     } // main
13:   } // ObrnjenoStevilo1
```

Če želimo dobiti stotice, število delimo s *100* (vrstica 4). Uporabili smo celoštevilsko deljenje, saj bomo tako dobili *1*. Torej *123 / 100 = 1*. Nato smo poiskali ostanek števila *123* pri deljenju s *100* (vrstica 5). Dobimo *23*, ki ga delimo z *10* in dobimo *2* desetici (vrstica 6). Sedaj nam mankajo samo še enice. V spremenljivki *dvomestnoStevilo* je *23* , ostanek števila pri deljenju z *10* nam da vrednost *3*.

Načinov, kako sestaviti program, ki reši dani problem, je seveda več. Želimo, da program deluje za poljubno trimestno število. Pomembno je , da program napišemo tako, da lahko spremenimo le vrstico 3 in v spremenljivko *trimestnoStevilo* shranimo kakšno drugo vrednost kot *123.* Na ta način bo naš program deloval, za poljubno trimestno število. Vendar nas moti, da moramo spremenjeni program in ga na novo prevesti. Kako bi vnesli vrednost med samim izvajanjem programa? To si bomo ogledali malo kasneje.

Zamenjava spremenljivk

Velikokrat želimo zamenjati vrednost dveh spremenljivk med sabo. Kako to izvedemo, pokažimo na primeru.

```
 1:   class ZamenjavaSpremenljivk {
 2:     static void Main(string[] args) {
 3:       int x = 4;
 4:       int y = 5;
 5:
 6:       int t = x;
 7:       x = y;
 8:       y = t;
 9:
```

```
10:         Console.WriteLine("Sedaj je x = " + x);
11:         Console.WriteLine("Sedaj je y = " + y + "\n");
12:
13:         x = y;
14:         y = x;
15:
16:         Console.WriteLine("Sedaj je x = " + x);
17:         Console.WriteLine("Sedaj je y = " + y + "\n");
18:       } // main
19:     } // ZamenjavaSpremenljivk
```

Program prevedemo in poženemo:

```
Sedaj je x = 5
Sedaj je y = 4

Sedaj je x = 4
Sedaj je y = 4
```

Od 6. do 8. vrstice je zamenjava spremenljivk *x* in *y*. Najprej deklariramo novo začasno spremenljivko *t*. Tej priredimo vrednost spremenljivke *x*, tako da *t* dobi vrednost *4* (*x* in *y* ostaneta nespremenjeni). Vrednost spremenljivke *x* imamo shranjeno v spremenljivki *t*. Tako lahko spremenljivki *x* priredimo vrednost spremenljivke *y* in s tem "povozimo" staro vrednost, shranjeno v *x* (*t = 4*, *y = 5*). Spremenljivka *x* ima že ustrezno vrednost. Preostane nam še, da spremenljivki *y* priredimo vrednost *t* in s tem končamo zamenjavo.
V 13. in 14. vrstici smo poskusili zamenjati vrednosti spremenljivk "kar na hitro". Želeli smo, da bi v spremenljivki *x* shranjena vrednost *4* in v spremenljivki *y* vrednost *5*. A to ni rezultat naše zamenjave. Poglejmo, zakaj ne. Vrednost spremenljivke *y* priredimo spremenljivki *x*. V *x* s tem shranimo *4*. To smo tudi želeli. V 14. vrstici pa **trenutno** vrednost spremenljivke *x* priredimo *y*. Torej tudi *y* postane *4*, saj smo v 13. vrstici v *x* shranili *4*.

Sekunde do novega leta

Nekaj dni pred novim letom je tvoja priljubljena televizijska postaja začela predvajati števec, ki je povedal, koliko sekund je še potrebno počakati do novega leta. Ker te je zelo motilo, da nisi točno vedel, koliko je to dni in ur, si se odločili, da boš za naslednje novo leto napisali program, ki bo pretvoril sekunde v dneve, ure, minute in sekunde. Kot pripravo najprej sestavi program, ki dano število sekund (23145) pretvori v dneve, ure, minute in sekunde.

Namig: Če čas v sekundah delimo s 60, dobimo, koliko je to minut. Ostanek pri deljenju s 60 pa so tiste "prave" sekunde, ki se niso pretvorile v minute. In tako naprej še za minute, ure in dneve. Torej bomo najprej izračunali (operator %), koliko bo res ostalo sekund. Nato bomo čas pretvorili v (cele) minute. Spet nam bo % pomagal, da bomo ugotovili "prave" minute. Nato bomo minute pretvorili v ure, ...

Rešitev:

```
 1:   // Koliko dni, ur, ... je 23145 sekund
 2:
 3:   public class Stevec  {
 4:       public static void main(String[] args) {
 5:           // vnesemo koliko sekund je do novega leta
 6:           int cas = 23145;
 7:
 8:           // RAČUNAMO
 9:           // pogledamo koliko "pravih"sekund
10:           int sekund = cas % 60;
11:           cas = cas / 60; // koliko je to minut
12:           // pogledamo koliko minut
13:           int minut = cas % 60;
```

```
14:            cas = cas / 60;
15:            // pogledamo koliko ur
16:            int ur = cas % 24;
17:            cas = cas / 24;
18:            // pogledamo koliko dni
19:            int dni = cas;
20:            // izpišemo rešitev
21:            Console.WriteLine("Do novega leta je se " + dni
22:                              + " dni " + ur + " ur " + minut
23:                              + " minut " + sekund + " sekund.");
24:        }
25:    }
```

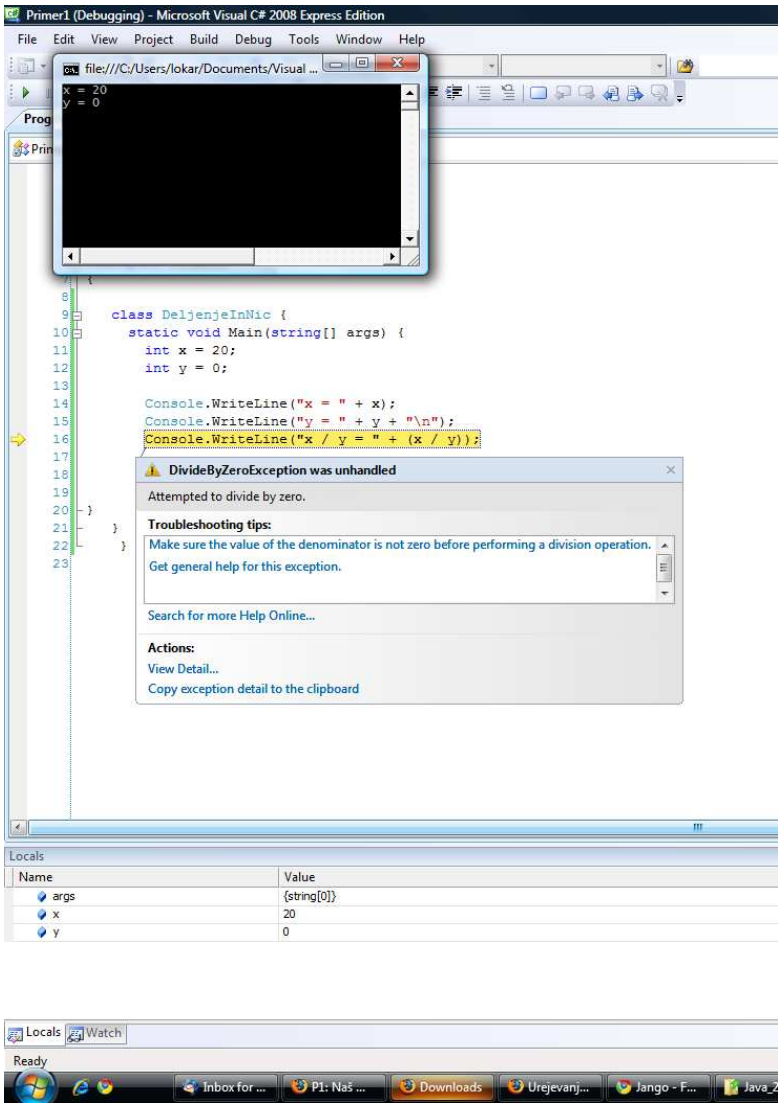
Napake pri delu s tipom int

Deljenje z 0

Paziti moramo, da v programu ne pride do deljenja z nič, saj se v tem primeru izvajanje programa prekine. Izvajalno okolje nam javi, da je prišlo do izjeme (ang. exception).

```
 1:    class DeljenjeInNic {
 2:      static void Main(string[] args) {
 3:        int x = 20;
 4:        int y = 0;
 5:
 6:        Console.WriteLine("x = " + x);
 7:        Console.WriteLine("y = " + y + "\n");
 8:        Console.WriteLine("x / y = " + (x / y));
 9:      } // main
10:    } // DeljenjeInNic
```

Program prevedemo in poženemo:



Po vrsti se izvedejo vse vrstice do 8. V tej pride do deljenja z nič, zato izvajalno okolje prekine izvajanje in sporoči, da se je zgodila tako imenovana izjema.

Preveliko število

Kot smo omenili, lahko v spremenljivkah tipa *int* hranimo cela števila velikosti do približno 2 milijard. Kaj se zgodi, če to mejo prekoračimo?

```
1:    class DolgoStevilo {
2:      static void main(String[] args) {
3:        int stevilo1 = 2147483647;
4:        int stevilo2 = 10;
5:        int vsotaStevil = stevilo1 + stevilo2;
6:
7:        Console.WriteLine("2147483647 + 10 = " + vsotaStevil);
8:      } // main
9:    } // DolgoStevilo
```

Prevedemo in poženemo:

```
2147483647 + 10 = -2147483639
```

Dobimo napačen rezultat brez predhodnega opozorila. Izvajalno okolje ne preverja, če so rezultati izrazov v smiselnih mejah. Na to moramo paziti sami.

Kadar nam obseg tipa *int* ne zadošča, uporabimo podatkovni tip *long*, a se v podrobnosti ne bomo spuščali.

Realna (decimalna) števila

Realna števila shranjujemo v spremenljivkah tipa **double**. Sestavljena so iz celega dela in decimalne vrednosti, ki ju loči **decimalna pika** (ne vejica!).

Primeri:

```
double dolzinaX = 3.4;
double cenaEnote = 1.9;
double faktorPodrazitve = 1.1;
double novaCenaEnote = cenaEnote * faktorPodrazitve;
```

Spremenljivke tipa *double* lahko zavzamejo zelo majhne (do okvirno 5.0 × 10-324) in zelo velike vrednosti (1.7 × 10308) z natančnostjo približno 15 mest. Zapišemo jih lahko tudi v eksponentni obliki z znakom E, npr. 12.16 E20, kar pomeni število 12.16 * 1020. V podrobnosti se ne bomo spuščali.

Tudi nad realnimi števili lahko izvajamo vse osnovne operacije (seštevanje, odštevanje, množenje in deljenje). Vsota, razlika, produkt in deljenje dveh realnih števil so definirani na običajen način. V realnih aritmetičnih izrazih nikoli ne pride do napake med izvajanjem (izjeme), saj način kodiranja omogoča zapis neskončnih in neizračunljivih vrednosti, ki se pripetijo pri deljenju z ničlo, neskončnost pa se uporabi tudi pri prekoračitvi obsega.
Če operator (+, -, /, *) povezuje celo in realno število, se celo število najprej pretvori v realno. Rezultat je realno število (tudi v primeru, ko je decimalni del enak nič!).

Primeri:

```
12.9 / 3 = 4.3
13.4 / 6.7 = 2.0
6.7 - 4.6 = 2.1000000000000005
1.4 * 0 = 0.0
3.6 / 0 = Infinity
```

Ker se realna števila hranijo v dvojiškem zapisu in zaradi omejenega prostora pride pri delu z njimi do zaokrožitvenih napak. Pri kakršni koli uporabi realnega tipa moramo te napake vzeti v zakup (kot se je v zgornjem primeru izraz 6.7 - 4.6 izračunal v 2.1000000000000005 namesto v 2.1).

OPERATOR	RAZLAGA	UPORABA	REZULTAT
+	vsota	6.7 + 2.3	9.0
-	razlika	6.7 - 2.3	4.4
*	množenje	6.7 * 2.3	15.409999999999998
/	deljenje	6.7 / 2.3	2.91304347826087

Uporabo operacij si bomo pogledali na nekaj zgledih. Še prej pa si oglejmo matematične funkcije v jeziku C#.

Funkcije – razred Math

Pogosto moramo izračunati na primer kvadratni koren nekega decimalnega števila, ali pa absolutno vrednost števila. Jezik C# pozna cel kup različnih matematičnih funkcij. Zbrane so v t.i. razredu Math. Razred Math vsebuje številne metode za uporabo matematičnih funkcij. Nekatere od njih so zbrane v spodnji tabeli:

Konstanta	Pomen
PI	Iracionalno število PI.
Metoda	Pomen
Abs	Absolutna vrednost.
Acos	Arkus Kosinus – kot (v radianih) , katerega kosinus je argument metode.
Asin	Arkus Sinus – kot, katerega sinus je argument metode.
Atan	Arkus Tangens – kot, katerega tangens je argument metode.
Cos	Kosinus kota.
Max	Metoda vrne večjega izmed dveh števil, ki nastopata kot argument metode.
Min	Metoda vrne manjšega izmed dveh števil, ki nastopata kot argument metode.
Pow	Metoda za izračun potence.
Round	Zaokroževanje.
Sqrt	Kvadratni koren.
Tan	Tangens.
Truncate	Celi del števila (odrežemo decimalke, rezultat je celo število!).

Nekaj primerov:

```
int negativno = -300;
int pozitivno = Math.Abs(negativno);
Console.WriteLine(pozitivno);  //izpis 300

double alfa = Math.Acos(-1);
Console.WriteLine(alfa);  //izpis 3,141592... = iracionalno število PI

double beta = Math.Asin(0.5);
Console.WriteLine(beta);  //izpis 0,52359..... = iracionalno število PI/6

double gama = Math.PI / 2;
double cosgama = Math.Cos(gama);
Console.WriteLine(cosgama);  //izpis 0

double vecje = Math.Max(235.8, 100.7);
Console.WriteLine(vecje);  //izpis 235.8

double manjse = Math.Min(235.8, 100.7);
Console.WriteLine(manjse);  //izpis 100.7

double eksponent = 3;
double osnova = 5;
double potenca = Math.Pow(osnova,eksponent);
Console.WriteLine(potenca);  //izpis 125

double decimalno = 245.67843;
Console.WriteLine(Math.Round(decimalno)); //izpis 246
Console.WriteLine(Math.Round(decimalno,2)); //izpis 245,68
Console.WriteLine(Math.Round(decimalno,3)); //izpis 245,678

double kvadrat = 625;
Console.WriteLine(Math.Sqrt(kvadrat)); //izpis 25
```

Zgledi

Dolžina poti

Prekolesarili smo dve petini poti, do cilja je še 21 kilometrov. Izračunajmo dolžino celotne poti.

Za izračun celotne poti uporabimo enačbo:
nePrevozeno = 3.0 / 5;
dolzinaPoti = 21 / nePrevozen;

```
class DolzinaPoti {
    static void Main(string[] args) {
        double delNePrevozeno = 3.0 / 5; // Delež neprevožene poti
        int prevozeno = 21; // Prevožena pot

        // Izračun dolžine prekolesarjene poti
        double dolzinaPoti = prevozeno / delNePrevozeno;

        // Izpis celotne poti
        Console.WriteLine(dolzinaPoti);
    }
}
```

Zapis na zaslonu:

```
35
```

Kolesarja

Zoran in Franci sta v treh dneh prikolesarila do kraja A. Zoran je prvi dan prekolesaril 32,1 kilometrov, drugi dan 21,8 in tretji dan 12,4 kilometrov. Franci pa je vsak dan prekolesaril tretjino celotne poti. Izračunajmo, koliko kilometrov je Franci prevozil vsak dan.

Potrebovali bomo dve realni spremenljivki. V eno bomo shranili dolžino poti do kraja A (vsoto Zoranovih dnevno prevoženih kilometrov), drugo pa potrebujemo za izračun Francijeve dnevno prevožene poti, ki je tretjina celotne poti.

```
 1:    class Kolesarja {
 2:      static void Main(string[] args) {
 3:        // razdalja do kraja A
 4:        double dolzinaPoti = 32.1 + 21.8 + 12.4;
 5:
 6:        // dolzina Francijeve enodnevne prekolesarjene poti
 7:        double tretjinaPoti = dolzinaPoti / 3;
 8:
 9:        Console.WriteLine("Franci je dnevno prevozil " +
10:                          tretjinaPoti + " km.");
11:      } // main
12:    } // Kolesarja
```

Program prevedemo in poženemo:

```
Franci je dnevno prevozil 22.1 km.
```

Plačilo bencina

Bencin se je podražil za 0,2 evra. Pred podražitvijo, ko je bil bencin še 1,17 evra, smo za poln tank plačali 40,5 evrov. Izračunajmo, koliko bomo po podražitvi plačali za poln tank.

Novo ceno bomo izračunali tako, da bomo volumen tanka pomnožili z novo ceno za liter bencina. Volumen tanka dobimo tako, da staro ceno polnega tanka delimo s staro ceno za liter bencina.

```
 1:    public class PlaciloBencina {
 2:      public static void main(String[] args) {
 3:        double staraCena = 1.17;
 4:        double novaCena = staraCena + 0.2;
 5:        double stariZnesek = 40.5;
 6:        double noviZnesek;
 7:        double volumenTanka = stariZnesek / staraCena;
 8:        // cena bencina po podrazitvi
 9:        noviZnesek = volumenTanka * novaCena;
10:
11:       Console.WriteLine("Za poln tank bomo placali " +
12:                         noviZnesek + " evrov.");
13:      } // main
14:    } // PlaciloBencina
```

Barvanje sob

Soba v obliki kvadra je dolga 5,6 m, široka 4,2 m in visoka 4,5 m. Pobarvati želimo stene in strop. Kilogram barve stane 6.99 € in zadošča za 5 m2. Izračunajmo, koliko denarja bomo potrebovali za barvanje sobe.

Za izračun površine ki jo moramo prebarvati uporabimo naslednjo enačbo:

*povrsina = dolzinaSobe * sirinaSobe + 2* (dolzinaSobe * visinaSobe) + 2*(sirinaSobe* visinaSobe)*

oz. če izpostavimo skupni faktor v drugem delu enačbe dobimo:

*povrsina = dolzinaSobe * sirinaSobe + 2 * visinaSobe * (dolzinaSobe + sirinaSobe).*

```
 1:    class BarvanjeSobe {
 2:      static void Main(string[] args) {
 3:        double dolzinaSobe = 5.6;
 4:        double sirinaSobe = 4.2;
 5:        double visinaSobe = 4.5;
 6:        double povrsina, cenaBarve;
 7:        const double CENAKGBARVE = 6.99;
 8:        const int IZDATNOST = 5;//za koliko k. metrov je kg          barve
 9:        // povrsina sobe brez tal
10:        povrsina = dolzinaSobe * sirinaSobe + 2 * visinaSobe *
11:                   (dolzinaSobe + sirinaSobe);
12:
13:        // cena barve
14:        cenaBarve = CENAKGBARVE * povrsina / IZDATNOST;
15:
16:        Console.WriteLine("Za barvanje sobe potrebujemo " +
17:                          cenaBarve + " tolarjev.");
18:      } // main
19:    } // BarvanjeSobe
```

Popravimo program tako, da bo znesek zaokrožili na tolar natančno. Pomagali si bomo z metodo *Round()* iz razreda *Math*. Več o tem razredu bomo izvedeli v nadaljevanju, sedaj pa povejmo le, da metoda *Round()* zaokroži realno število na najbližje celo število, pokličemo pa jo z

Math.Round(steviloZaZaokrozanje).

```
1:    class BarvanjeSobe1 {
2:      ... (glej program BarvanjeSobe)
```

```
15:
16:         Console.WriteLine("Za barvanje sobe potrebujemo " +
17:                           Math.Round(cenaBarve) + " tolarjev.");
18:       } // main
19:     } // BarvanjeSobe1
```

Plačilo mesečne vozovnice

Mesečna vozovnica za avtobus stane 68,6€ tolarjev. Izračunajmo, koliko bo stala po 5–odstotni podražitvi.

```
 1:     public class MesecnaVozovnica {
 2:       public static void main(String[] args) {
 3:         double cenaMesecne = 68.6;
 4:         double podrazitev = 5 / 100;
 5:         double novaCena;
 6:
 7:         // cena mesecne po podrazitvi
 8:         novaCena = cenaMesecne + cenaMesecne * podrazitev;
 9:
10:         Console.WriteLine("Mesecna bo stala " +
11:                           novaCena + " €.");
12:       } // main
13:     } // MesecnaVozovnica
```

Program prevedemo in poženemo:

```
Mesecna bo stala 68.6 €.
```

Ker je rezultat enak prvotni ceni mesečne vozovnice pred podražitvijo, nekaj ni v redu. Kje smo se zmotili?

Razlog tiči v četrti vrstici. V prireditvenem stavku se najprej izračuna desna stran. Izračunamo *5 / 100.* Ker sta števili celi, je to *0*. Spremenljivka *podrazitev* je realno število, zato se celo število (*0*) pretvori v realno, to je v *0.0*. Ta vrednost se priredi spremenljivki *podrazitev*.

Kadar želimo dve celi števili deliti na običajen način (da dobimo realni kvocient), je pred deljenjem potrebno vsaj eno od števil pretvoriti v realno. To pa lahko naredimo na več načinov.

```
5 / 100.0 // celemu številu dodamo decimalno vrednost (.0)
(double)5 / 100 // pred število, ki ga želimo pretvoriti,
                // napišemo željen tip v oklepaju
```

Več o spremembah med tipi si bomo ogledali v posebnem razdelku, sedaj pa popravimo naš program.

```
 1:     public class MesecnaVozovnica {
 2:       public static void main(String[] args) {
 3:         double cenaMesecne = 68.6;
 4:         double podrazitev = 5.0 / 100;
 5:         double novaCena;
 6:
 7:         // cena mesecne po podrazitvi
 8:         novaCena = cenaMesecne + cenaMesecne * podrazitev;
 9:
10:         Console.WriteLine("Mesecna bo stala " +
11:                           novaCena + " €.");
12:       } // main
13:     } // MesecnaVozovnica
```

Program prevedemo in poženemo:

```
Mesecna bo stala 72 €.
```

Dolžina poti

Prehodili smo 3/5 poti, do cilja je še 16 kilometrov. Izračunajmo dolžino celotne poti.

Za izračun celotne poti uporabimo enačbo:

delezPoti = 2 / 5.0 (ne prevožene)
*delezPoti * dolzinaPoti = 16* oz.
dolzinaPoti= 16 / delezPoti

```
 1:    public class DolzinaPoti {
 2:      public static void main(String[] args) {
 3:        // delez poti do cilja
 4:        double delezPoti = 2 / 5.0;
 5:        int ostane = 16;
 6:        double dolzinaPoti;
 7:
 8:        // dolzina prehojene poti
 9:        dolzinaPoti = ostane / delezPoti;
10:
11:        Console.WriteLine(dolzinaPoti);
12:      } // main
13:    } // DolzinaPoti
```

Program prevedemo in poženemo:

```
40.0
```

Pretvarjanje med vgrajenimi podatkovnimi tipi

Večkrat tip vrednosti ne ustreza željnemu. Zato ga je potrebno pretvoriti v drugi tip. Včasih to stori prevajalnik, včasih pretvorbo zahteva programer. Avtomatične pretvorbe (tiste, ki jih opravi prevajalnik) iz tipa double in tipa int v tip string ter iz tipa int v tip double smo že srečali v več zgledih.

Pretvarjanje iz tipa int v tip double

Če želimo celo število pretvoriti v realno, napišemo pred celim številom v okroglih oklepajih *double*.

```
(double)celoStevilo
```

Število tipa *int* se pretvori v število tipa *double* tudi tako, da se zadaj doda .0 (decimalni del).

Zgled - plačilo sira

V Estonji v trgovini stane 1 kilogram sira 1150 kron. Manja kupi 20 dekagramov sira. Izračunajmo koliko bo Manja plačala za sir.

```
 1:    class CenaSira {
 2:      static void Main(string[] args) {
 3:        int teza1 = 100; // teza v dekagramih
 4:        int cena1 = 1150; // cena za 1 kilogram
 5:        int teza2 = 20; // teza v dekagramih
 6:
 7:        // cena sira
 8:        double cena2 = teza2 * cena1 / (double)teza1;
 9:
10:        Console.WriteLine(teza2 + " dag sira stane " +
11:                          cena2 + " kron.");
12:      } // main
13:    } // CenaSira
```

Program prevedemo in poženemo:

```
20 dag sira stane 230.0 kron.
```

Ceno sira dobimo tako, da ceno za 1 dekagram (*cena1 / (double)teza1 = 11.5*) pomnožimo s težo kupljenega sira. Če želimo pri deljenju dveh celih števil dobiti realno število*(11.5)*, moramo vsaj eno od števil spremeniti v realno. Z *(double)teza1* smo celoštevilčno vrednost, ki je shranjena v spremenljivki *teza1,* spremenili v realno število. Ko se izračuna desni del, se priredi realni spremenljivki *cena2*.

Na koncu še izpišemo vrednosti spremenljivk *teza2* in *cena2* opremljeni z ustreznim besedilom.

Pretvarjanje iz tipa double v int

Če želimo realno število pretvoriti v celo, napišemo pred realnim številom v okroglih oklepajih *int*.

```
(int)realnoStevilo
```

Sprememba tipa iz realnega v celo število se opravi tako, da se preprosto odreže decimalni del.

Višina vode

V valjast sod s premerom 100 centimetrov nalijemo 100 litrov vode. Izračunajmo, kolikšno višino doseže voda. Rezultat naj bo zaokrožen na eno decimalno mesto.

Pri računanju si pomagamo s konstanto *PI* ter metodo *Pow()*, ki ju najdemo v razredu *Math*.
V konstanti *PI* je shranjena vrednost π, uporabimo jo z ukazom *Math.PI*. Metoda pow( ) vrne eksponent števila. Z ukazom *Math.Pow(osnova, eksponent)* dobimo število *osnovaeksponent*.
Da bomo število zaokrožili na eno decimalko število najprej pomnožimo z 10, ga zaokrožimo ter ga spremenimo v tip int. S tem smo odrezali odvečne decimalne vrednosti. Nato število delimo z 10 in tako dobimo število, ki je zaokroženo na eno decimalno mesto.

```
 1:    class VisinaVode {
 2:      public static void Main(string[] args) {
 3:        double volumenVode = 100000; // volumen vode v cm3
 4:        double polmerSoda = 50; // polmer soda v cm
 5:        double visinaVode; // visina vode v cm
 6:        double ploscinaKroga; //  ploscina sodovega dna v cm2
 7:
 8:        // visina vode
 9:        ploscinaKroga = Math.PI * Math.Pow(polmerSoda, 2);
10:        visinaVode = volumenVode / ploscinaKroga;
11:
```

```
12:         // zaokrozimo na eno decimalno mesto
13:         visinaVode = (int)(Math.Round(visinaVode * 10));
14:         visinaVode = visinaVode / 10;
15:
16:         Console.WriteLine("Voda stoji " + visinaVode +
17:                           " centimetrov visoko.");
18:       } // main
19:     } // VisinaVode
```

Program prevedemo in poženemo:

```
Voda stoji 12.7 centimetrov visoko.
```

Višino vode izračunamo s pomočjo formule za volumen valja. Volumen vode delimo s ploščino sodovega dna. Ploščino dna izračunamo po formuli za ploščino kroga z *Math.PI * Math.Pow(polmerSoda, 2)*. *Math.PI* vrne vrednost π, *Math.Pow(polmerSoda, 2)* vrne *polmerSoda2*. Ko se izračuna desna stran, se vrednost priredi spremenljivki *visinaVode*.
Sedaj moramo še "odrezati" odvečne decimalke. To naredimo na naslednji način. Realno število *visinaVode* najprej pomnožimo s 10. S tem smo decimalno piko prestavili za eno mesto v desno. Dobljeno vrednost zaokrožimo z Math.Round() in decimalni del "odrežemo" tako, da realno število pretvorimo v celo število z *(int)*. Sedaj moramo deliti še z 10, da dobimo nazaj prvotno vrednost, vendar zaokroženo na eno decimalko natančno.
Nato še izpišemo vrednost spremenljivke *visinaVode* z ustreznim tekstom.

Dolžina ograje

Ploščina kvadratnega vrta meri 119,40 metrov. Izračunajmo, koliko metrov ograje bomo potrebovali za ograditev vrta. Rezultat naj bo zaokrožen na dve decimalni mesti.

Tokrat bomo potrebovali metodo *Sqrt()*, ki jo prav tako najdemo v razredu *Math*. Metoda *Sqrt()* vrne kvadratni koren števila. Z ukazom *Math.Sqrt(število)* torej dobimo kvadratni koren števila *število*.
Pri zaokroževanju na določeno število mest si pomagamo podobno kot v prejšnjem primeru. Število najprej pomnožimo s *100* ter ga celoštevilsko zaokrožimo. S tem smo odrezali odvečne decimalne vrednosti. Nato število delimo s *100* in dobimo število, ki je zaokroženo na dve decimalni mesti.

```
 1:    class DolzinaOgraje {
 2:      static void Main(string[] args) {
 3:        double ploscinaVrta = 119.40;
 4:        double dolzinaOgraje;
 5:        double a; // dolzina (oziroma sirina) vrta
 6:
 7:        // izracunamo dolzino (oziroma sirino) vrta
 8:        a = Math.Sqrt(ploscinaVrta);
 9:
10:        // izracunamo dolzino ograje
11:         dolzinaOgraje = 4 * a;
12:
13:        // zaokrozimo na dve decimalni mesti
14:        dolzinaOgraje = (int)(Math.Round(dolzinaOgraje * 100));
15:        dolzinaOgraje = dolzinaOgraje / 100;
16:
17:        Console.WriteLine("Potrebovali bomo " + dolzinaOgraje +
18:                          " metrov ograje.");
19:      } // main
20:    } // DolzinaOgraje
```

Program prevedemo in poženemo:

```
Potrebovali bomo 43.71 metrov ograje.
```

Pretvarjanje iz niza (string) v celo število (int)

Če so v nizu zapisane le števke (in kot prvi znak morebiti – ali +), ga v C# lahko pretvorimo v število na dva načina:

- z metodo `int.Parse();`
- z eno od metod `Convert.ToInt16(),Convert.ToInt32() in Convert.ToInt64`()(odvisno od tega, s kako velikim celim številom imamo opravka) .

Zamenjava števk

Podan imamo niz "8304". Spremenimo ga tako, da bo njegova oblika "3084".

Pomagali si bomo s števkami števila 8304. Števke števila bomo pridobili s pomočjo aritmetičnih operatorjev / in %.

```
1:    class SpremeniNiz {
2:        static void Main(string[] args)
3:        {
4:            // Podan niz
5:            string niz = "8304";
6:
C7:           // Niz pretvorimo v celo število
8:            int stevilo = int.Parse(niz);
9:           //ali int stevilo = Convert.ToInt32(niz);
10:
11:           // Določimo števke števila
12:           int e = stevilo % 10; // enice
13:           int d = stevilo / 10 % 10; // destice
14:           int s = stevilo / 100 % 10; // stotice
15:           int t = stevilo / 1000; // tisočice
16:
17:           // V niz shranimo novo obliko
18:           niz = "" + s + d + t + e;
19:
20:           // Izpišemo novo število
21:           Console.WriteLine(niz);
22:       }
      }
```

Zapis na zaslon:

```
3084
```

Razlaga. Niz niz s klicem metode int.Parse() pretvorimo v celo število, ki ga priredimo spremenljivki stevilo. V vrsticah od C11 do C14 izračunamo števke. Po izračunu števk jih v ustreznem vrstnem redu shranimo v niz (C17). Na začetku smo navedli "" z namenom, da vrednosti navedenih spremenljivk (s, d, t in e) avtomatsko pretvorimo v nize in jih združimo v skupen niz. Ta niz predstavlja spremenjen niz niz, ki smo ga želeli dobiti. Pri tem si pomagamo z združevalnim operatorjem +.

Razlika števil

V dveh celoštevilskih spremenljivkah imamo shranjeni dve števki. Izračunajmo razliko med največjim in najmanjšim številom, ki ju lahko sestavimo iz teh dveh števk.

Pri računanju si pomagamo z metodo *Abs()*, ki jo najdemo v razredu *Math*. Metoda *Abs()* vrne absolutno vrednost števila.

```
 1:    class RazlikaStevil {
 2:      static void Main(string[] args) {
 3:        int prvaStevka = 2;
 4:        int drugaStevka = 4;
 5:
 6:        // sestavimo oz. staknemo stevili
 7:        string prviNiz = "" + prvaStevka + drugaStevka;
 8:        string drugiNiz = "" + drugaStevka + prvaStevka;
 9:
10:        // niza pretvorimo v celi stevili
11:        int prvoStevilo = int.Parse(prviNiz);
12:        int drugoStevilo = int.Parse(drugiNiz);
13:
14:        // izracunamo razliko
15:        int razlika = prvoStevilo - drugoStevilo;
16:
17:        Console.WriteLine("Razlika sestavljenih stevil je " +
18:                          Math.Abs(razlika) + ".");
19:      } // main
20:    } // RazlikaStevil
```

Program prevedemo in poženemo:

```
Razlika sestavljenih stevil je 18.
```

Iz dveh števk sestavimo število tako, da ju staknemo, pri tem si pomagamo z združitvenim operatorjem *+*. S tem dobimo niz (prviNiz = "24" in drugiNiz = "42"). V 11. vrstici in 12. vrstici iz niza dobimo celo število s pomočjo metode *int.Parse()*. Niz *prviNiz* se v vrstici 11 najprej pretvori v celo število, ki se priredi celoštevilski spremenljivki *prvoStevilo*. Niz *drugiNiz* se s klicem metode *int.Parse(drugiNiz)* pretvori v celo število, ki se priredi celoštevilski spremenljivki na levi strani *drugoStevilo* (vrstica 12). Sedaj lahko izračunamo razliko med številoma. Razliko priredimo spremenljivki *razlika*. Sledi še izpis absolutne vrednosti spremenljivke *razlika,* opremljen z ustreznim tekstom.

Pri pretvarjanju iz številskih vrednosti v niz si pomagamo z združitvenim operatorjem *+* kot smo to že večkrat naredili. Če torej število seštejemo s praznim nizom (""), bomo dobili ustrezno predstavitev števila kot niz.

Pretvarjanje iz niza v realno število

Če je v nizu zapisano realno število, ga v jeziku C# pretvorimo v število z metodo `double.Parse()`, ali pa z metodo `Convert.ToDouble()`.

Razlika števil II

V dveh celoštevilskih spremenljivkah imamo shranjeni dve enomestni števili. Izračunajmo razliko med največjim in najmanjšim številom, ki ju lahko sestavimo iz teh dveh števk, če eno od teh pomeni enice, drugo pa desetinke.

Pri računanju razlike dveh realnih števil si pomagamo z metodo Abs(), ki se nahaja v razredu `Math`.

```
1:    class RazlikaStevil {
2:        static void Main(string[] args)
3:        {
4:            // Števki števil
5:            int prvaStevka = 9;
6:            int drugaStevka = 5;
7:
8:            // Sestavimo niza
9:            string prviNiz = prvaStevka + "," + drugaStevka;
```

```
10:             string drugiNiz = drugaStevka + "," + prvaStevka;
11:
12:             // Niza pretvorimo v realni števili
13:             double prvoStevilo = double.Parse(prviNiz);
14:             //ali double prvoStevilo = double.Parse(prviNiz);
15:             double drugoStevilo = double.Parse(drugiNiz);
16:             //ali double drugoStevilo = double.Parse(drugiNiz);
17:
18:             // Izračun razlike
19:             double razlika = prvoStevilo - drugoStevilo;
20:
                // Izpis
21:             Console.WriteLine("Razlika dveh sestavljenih stevil je " +
22:                 Math.Abs(razlika) + ".");
            }
        }
```

Zapis na zaslonu:

```
Razlika dveh sestavljenih stevil je 3,6.
```

Razlaga. Iz dveh števk in decimalne vejice sestavimo število tako, da jih ustrezno staknemo z združevalnim operatorjem +. S tem v vrstici C9 dobimo niz prviNiz in v vrstici C10 niz drugiNiz. V vrsticah C13 in C14 s pomočjo metode double.Parse() iz niza dobimo realno število. Sedaj lahko izračunamo razliko med realnima številoma. Izračunano razliko priredimo spremenljivki razlika (C17). Na koncu še izpišemo absolutno vrednost spremenljivke razlika, opremljeno z ustreznim besedilom.

Opomba: Metoda **double.Parse(niz)** v nizu ne prepozna decimalne pike (jo enostavno ignorira), zato v nizu uporabimo decimalno vejico. Tako obnašanje prevajalnika za C# je odvisno od nastavitev operacijskega sistema. Okolja .NET namreč ločilo decimalnega dela števila privzeto prevzeme iz operacijskega sistema. Zato pri običajnih nastavitvah operacijskega sistema (pri nas v Sloveniji), vejica (,) pomeni ločilo med celim in decimalnim delom števila. Znak . pa operacijski sistem uporablja le za vidno ločevanje skupin števk pri večjih številih. Zato se . v nizih, ki jih v jeziku C# pretvarjamo z metodo Parse, kar preskoči.

Primeri:

```
// V nizu uporabimo decimalno vejico
string prviNiz = "23,4";
double prvoStevilo = double.Parse(prviNiz); // Vrednost spremenljivke je 23,4

// V nizu uporabimo decimalno piko
string drugiNiz = "1.5";
double drugoStevilo = double.Parse(drugiNiz); // Vrednost spremenljivke je 15

// V nizu uporabimo decimalno piko in decimalno vejico
string tretjiNiz = "1.235,4";
double tretjeStevilo = double.Parse(tretjiNiz); // Vrednost spremenljivke je 1235,4
```

Rešene naloge

Dopolni program

Na označenih mestih dopolni stavke tako, da bo prevajalnik ta del progama prevedel brez opozoril. Kakšne vrednosti zavzamejo spremenljivke po izvedbi teh ukazov?

```
/* dopolni tukaj */ a = 3.14;
int b = /* dopolni tukaj */ a;
/* dopolni tukaj */ c = (int) (a * b);
/* dopolni tukaj */ d = "" + (a + (int) a + b) + "a";
```

Rešitev: Dopolnjeni izrazi:

```
double a = 3.14;
int b = (int)a;
int c = (int)(a * b);
string d = "" + (a + (int)a + b) + "a";
```

Spremenljivke imajo na koncu vrednost

a = 3.14
b = 3
c = 9
d = 9.14a

Komentar: Če želiš v spremenljivko a shraniti realno število, potem mora biti spremenljivka tipa double. Glede na to, da b mora biti celo število, potem moraš poskrbeti, da je izraz za enačajem tudi celo število. To storiš tako, da decimalno vrednost spremenljivke a z (int) pretvoriš v tipa int. Podoben razmislek velja za spremenljivko c, ki naj bi hranila celi del zmnožka a in b. No, resnici na ljubo, bi bila spremenljivka c lahko tudi tipa double. Takrat bi dobila vrednost 9.0. Spremenljivka d mora biti tipa string, ker poleg seštevanje vrednosti na začetku in na koncu prištejemo še niza.

Pretvorba funtov v kg

Napiši program FuntVKg, ki 34.78 funtov pretvori v kilograme. Izračunano vrednost izpiši na dve decimalki natančno. Pri pisanju programa tega napiši tako, da ga bo mogoče zlahka spremeniti za drugo težo.

Namig: En funt je enako 0.45359 kilograma.

Rešitev:

```
1:   class FuntVKg {
2:        static void main(String[] args) {
3:           // povemo koliko kilogramov je en funt
4:           const double funtVkg = 0.45359;
5:           // določimo funte
6:           double funti = 34.78;
7:           // pretvorimo v kilograme
8:           double kilogrami = funti * funtVkg;
9:           // kilograme pripravimo na dve decimalki natančno
10:          double kg_2dec = ((int) (kilogrami * 100)) / 100.0;
11:          // izpišemo rešitev
12:          Console.WriteLine(funti + " funtov je enako "
13:                            + kg_2dec + " kg.");
14:       }
15:  }
```

Komentar: V programu smo najprej nastavili konstanto (določilo const) za pretvorbo iz funtov v kilograme (4. vrstica). Sledila je pretvorba iz funtov v kilogram (8. vrstica). Naloga je tudi zahtevala, da rešitev zapišemo na dve decimalki natančno. To smo v 10. vrstici naredili tako, da smo najprej pomnožili rešitev s 100 in z uporabo operatorja za pretvorbo tipa (int) pretvorili v celo število. Tako smo dosegli, da smo vse nepotrebne decimalke »izgubili«. Ostane nam le še, da popravljene kilograme delimo s 100 in izpišemo rešitev.

Sledenje programu

Kakšno vrednost imajo na koncu zaporedja stavkov spremenljivke a, b, c in d?

```
 1:   int a;
 2:   int b;
 3:   double c;
 4:   double d;
 5:
 6:   a =  5;
 7:   b = -3;
 8:   c = a + b;
 9:   d = (a + 1) / b;
10:   a = a / b;
```

Rešitev: Na koncu tega dela programa ima a vrednost -1, b ima vrednost -3, c vrednost 2.0 in d vrednost -1.

Komentar: V 1. in 2. vrstici sta deklarirani celoštevilski spremenljivki a in b, v 3. in 4. vrstici pa realni spremenljivki c in d. V 6. vrstici dobi a vrednost 5, v 7. vrstici pa b dobi vrednost -3. V 8. vrstici dobi spremenljivka c vrednost vsote a in b. Pri tem moramo biti pozorni, da je c realno število in da shrani vrednost, zapisano z decimalko. Vrednost c v tej vrstici po izvedbi ukaza je 2.0. Podobno se zgodi v 9. vrstici, kjer spremenljivka d dobi vrednost -2.0. V 10. vrstici pa a dobi novo vrednost in sicer a / b. Ker sta oba operanda pri deljenju celi števili, je deljenje celoštevilsko.

Poglejmo izvajanje programa s pomočjo tabele:

vrstica	ukaz	a	b	c	d
1:	int a;				
2:	int b;				
3:	double c;				
4:	double d;				
5:					
6:	a = 5;	5			
7:	b = -3;	5	-3		
8:	c = a + b;	5	-3	2.0	
9:	d = (a + 1) / b;	5	-3	2.0	-2.0
10:	a = a / b;	-1	-3	2.0	-2.0

Decimalni del števila

Tonček si pri realnih številih težko predstavlja, kateri je celi in kateri decimalni del števila. Zato mu napiši program, kjer bo lahko dobil izpis celega in izpis decimalnega dela števila. Seveda naj se izpiše tudi število. Tvoj program naj dela za število 23.576, napisan pa naj bo tako, da je to število lahko poljubno.
Namigi

Program naj izgleda takole:

* Preberi število.
* Izračunaj celi in decimalni del.
* Izpiši celi in decimalni del.

Spomni se, kako realno število spremenimo v celo. Kakšna je zveza med celim in decimalnim delom realnega števila?

Če je double d decimalno število, pretvorba v celo število int celo = (int)d odseka decimalni del. Tako iz npr. 3.56 dobimo 3.

Decimalni del števila dobimo tako, da najprej izračunamo celi del števila, ki ga potem odštejemo od števila. Razlika je ravno decimalni del.

```
class CeliInDecimalni {
    static void Main(string[] args) {
        double vnos = 23.576;
        int celiDel = (int)vnos;
        double ostanek = vnos - celiDel;

        //izpisemo
        Console.WriteLine(stevilo);
        Console.WriteLine("Celi del: " + celiDel);
        Console.WriteLine("Decimalni del: " + ostanek);
    }
}
```

Branje

Prej smo že napisali program, s katerim smo iz števila 138 naredili število 831 (mu torej obrnili števki). Pri tem smo pazili, da smo program napisali tako, da smo, če smo želeli uporabiti drugo trimestno število, naredili popravek le na enem mestu (tam, kjer smo število 138 shranili v spremenljivko). A vseeno, če smo želeli delati z drugim številom, je bilo potrebno:

- popraviti program
- ponovno prevajanje programa
- izvedba programa

Uporabnik našega programa torej potrebuje izvorno kodo, kot tudi znanje popravljanja kode in prevajanja.

Če želimo, da uporabnik sam vnaša določene podatke v naš program, moramo spoznati način, kako lahko spremenljivki priredimo vrednost, ki jo uporabnik vnese s pomočjo tipkovnice. Najprej si bomo pogledali, kako bi prebrali in izpisali niz, ki ga vpiše uporabnik.

Za branje podatkov iz konzole smo v jeziku C# uporabimo metodo ReadLine(), pripada razredu Console. V C# podatek torej preberemo z

```
Console.ReadLine();
```

Rezultat te metode je niz, ki vsebuje tisto zaporedje znakov, ki ga natipkamo. Shranili ga bomo torej v spremeljivko tipa string.

```
string bla = Console.ReadLine();
```

Ko se izvaja ta stavek, se program "ustavi" in čaka, da nekaj natipkamo. Ko natipkamo določene znake in pritisnemo na tipko Enter, se natipkani znaki (brez Enter) shranijo v niz (v našem primeru v spremenljivko bla).

Da vemo, da program čaka na nas, pred branjem z metodo Write (ali WriteLine) na zaslon izpišemo ustrezno opozorilo:

```
Console.Write("Vnesi ime avtorja: ");
//preberemo stavek in ga shranimo v spremenljivko tipa string
string avtorIme = Console.ReadLine();
```

Branje podatkov iz konzole je možno le preko tipa niz (string). Vse podatke je potrebno kasneje pretvoriti iz niza v obliko, ki jo potrebujemo. Za pretvarjanje lahko porabimo metodo Parse .

Primer:

```
Console.Write("Vnesi še eno celo število: ");
int st = int.Parse(Console.ReadLine());
```

Lahko pa gremo tudi korak za korakom in podatek najprej preberemo v spremenljivko tipa string.

Primer:

```
Console.Write("Vnesi še eno celo število: ");
string branje = Console.ReadLine();
int st = int.Parse(branje);
```

Sedaj že vemo dovolj, da lahko program, s katerim smo trimestno število izpisali obrnjeno, spremenimo tako, da bomo število prebrali. Na ta način bo naš program uporaben za obračanje poljubnega trimestnega števila, saj se bo šele ob zagonu programa "odločilo", katero trimestno število obračamo.

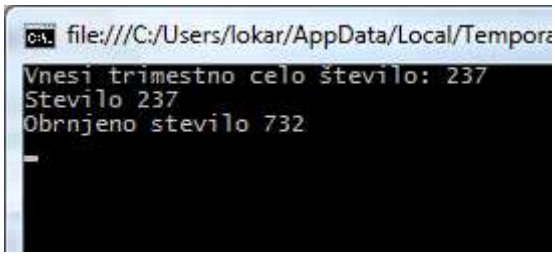
Navedimo stari program ponovno in označimo dele, ki jih bomo spremenili

```
 1:    class ObrnjenoStevilo {
 2:      static void Main(string[] args) {
 3:        int trimestnoStevilo = 123;
 4:        int enice = trimestnoStevilo % 10;
 5:        int dvomestnoStevilo = trimestnoStevilo / 10;
 6:        int desetice = dvomestnoStevilo % 10;
 7:        int stotice = dvomestnoStevilo / 10;
 8:
 9:        Console.WriteLine("Stevilo " + trimestnoStevilo);
10:        Console.WriteLine                          ("Obrnjeno stevilo "
11:                              + enice + desetice + stotice);
12:      } // main
13:    } // ObrnjenoStevilo
```

Spremeniti bo potrebno le tretjo vrstico. To bomo nadomestili z

```
Console.WriteLine("Vnesi trimestno celo število: ");
string beri = Console.ReadLine();
int trimestnoStevilo = int.Parse(beri);
```

Vidimo, da smo dodali vrstici, kjer smo prebrali število kot niz. Sledi še popravek v (stari) 3. vrstici kjer namesto števila 123 v spremenljivko *trimestnoStevilo* shranili število, ki ga dobimo s pretvorbo niza *beri* v tip *int*. Vse ostalo je ostalo nespremenjeno. To pove, da smo prvotni program res napisali tako, da je deloval za poljubno trimestno število.



Od največjega do najmanjšega

Uredimo tri števila od najmanjšega do največjega. Napišimo program, ki bo prebral tri cela števila in jih izpisal v padajočem vrstnem redu.

Pri razporeditvi števil si pomagajmo z metodama **Min()** in **Max()**, ki ju najdemo v razredu **Math**.

```
 1:    public class PadajoceUrejanje {
 2:        public static void Main(string[] args)
 3:        {
 4:            // Vnos podatkov
 5:            Console.Write("Vnesi prvo stevilo: ");
 6:            int prvo = int.Parse(Console.ReadLine());
 7:            Console.Write("Vnesi drugo stevilo: ");
 8:            int drugo = int.Parse(Console.ReadLine());
 9:            Console.Write("Vnesi tretje stevilo: ");
10:            int tretje = int.Parse(Console.ReadLine());
11:
12:            // Določimo razporeditev števil
13:            int min = Math.Min(Math.Min(prvo, drugo), tretje);
14:            int max = Math.Max(Math.Max(prvo, drugo), tretje);
15:            int srednje = prvo + drugo + tretje - max - min;
16:
```

```
17:             // Izpis
18:             Console.WriteLine();
19:             Console.WriteLine("Urejena stevila od najmanjsega do" +
20:                               "najvecjega so: ");
21:             Console.WriteLine(min + " " + srednje + " " + max);
22:         }
23:     }
```

Zapis na zaslon:

```
Vnesi prvo stevilo: 59
Vnesi drugo stevilo: -46
Vnesi tretje stevilo: 23

Urejena stevila od najmanjsega do najvecjega so:
-46 23 59
```

Razlaga. Preberemo podatke in jih pretvorimo v cela števila (C5 – C10). Nato določimo razporeditev števil s pomočjo metod Math.Min() in Math.Max() (C13 – C15). Po določitvi razporeditve števil izpišemo ustrezno besedilo (C19 – C21).

Naloge

1. Spremeni naslednja navodila v ukaze v C#
   - Deklariraj celoštevilčno spremenljivko x z začetno vrednostjo 12.
   - Deklariraj celoštevilčno spremenljivko y z začetno vrednostjo -715.
   - Deklariraj celoštevilčno spremenljivko z z začetno vrednostjo 0.
   - Nastavi z na vsoto spremenljivk x in y.
   - Odštej 7 od x.
   - Nastavi y na produkt x in z.

2. Kakšno vrednost ima spremenljivka potem, ko se izvede prireditveni stavek:

```
a) int x = (int)9.2 * 6;                          x  _
b) string d = "1 + 3 = " + 1 + 3;                 d  _
c) double e = 16/5 - 8%5;                         e  _
d) bool f = !(3 <= 5 && 6 != 7);                  f  _
e) bool g = (3 < 2) || (3 < 5 && 6 > 3);          g  _
```

3. Kakšna je vrednost spremenljivke x po vsakem od naslednjih zaporedij ukazov:

```
a) int x = (int)(14.0 / 3 + 2.0 / 3);             x  [     ]
b) double x = (double)(12 / 5 - 3 % 2);           x  [     ]
c) string x = 1 + 2 + "bla" + 1 + 2;              x  [     ]
č) int y = 2;
   int x=y*y;
   y=3;                                           x  [     ]
d) double x = 0;
   double y = 1;
   y = x;
   x = y;                                         x  [     ]
```

4. Kakšno vrednost imajo spremenljivke niz1, niz2, niz3 in niz4 (tipa string) potem, ko se izvede zaporedje stavkov:

```
string niz1 = "bla1";
       niz1 = "niz1" + "abc";

string niz2 = 1 + "abc" + 1 + 2;
string niz3 = "lo\" + \"bb";
string niz4 = 1 + 2 + niz1;
string niz5 = niz1 + (1 + 2 + "abc");

niz1 [              ] niz2 [              ] niz3 [ ]

niz4 [              ] niz5 [              ] .
```

5. V C# napiši izraze za:

a) celi del decimalnega števila zmanjšan za celi del desetkratnika decimalnega dela istega števila

_____

b) koren iz vsote kvadratov spremenljivk a in b

_____

c) Za 1 povečano število, ki ga dobimo tako, da v celo število pretvorimo niz, ki ga dobimo, če nizu "234" na koncu dodamo še enice števila, ki je v spremenjivki x tipa int.

_____

6. Kateri podatkovni tip bi uporabili, da bi vanj shranili vrednost izraza

```
int.Parse("123") / 2.0
```
_____

7. Kakšno vrednost ima spremenljivka potem, ko se izvede prireditveni stavek:

a) `int x = (int)2.5 * 6;` x _____

b) `string d = 1 + 3 + "1 + 3 = ";` d _____

c) `double e = 16 / 5 - 16.0 / 5.0;` e _____

8. Kakšna je vrednost spremenljivke x po vsakem od naslednjih zaporedij ukazov:

a)
```
int x = (int)(14.0 / 3);
```
x _____

b)
```
double x = (double)(12 / 5 - 3 % 2);
```
x _____

c)
```
int y = 2;
int x = y * y;
y = 3;
x = x;
```
x _____

d)
```
double x = 1.7;
double y = x + 2.3;
y = x;
x = y;
```
x _____

9. Kakšno vrednost bo imela spremenljivka x (tipa int) potem, ko se izvede stavek:

a) `x = 12 / 5 + 2 * 3;` x _

b) `x = 2 – 5 * 3;` x _

c) `x = 12 % 5 + 2 % 3;` x _

10. Kakšno vrednost bosta imeli spremenljivki x (tipa int) in a (tipa double) potem, ko se izvede zaporedje stavkov:

a)
```
x = (int) (6 / 5.0);
a = (int) (6 / 5);
```
x =
a =

b)
```
x = (int)(6 / 2.0);
a = (double)(6 % x);
```
x =
a =

c)
```
x = 7 / 2;
a = 6.2 + ((double)((x * 3) - 4));
```
x =
a =

11. Napišite v C# izraze, ki ustrezajo naslednjim matematičnim zapisom

a) $\sin x + \cos y$ _____

b) $\sqrt{(b^2 - 4ac)}$ _____

c) $|3 - 5^3|$ _____

12. Kakšno vrednost bodo imele spremenljivke, ko se izvedejo stavki:

```
a) double  w = (int)(14.0 / 5) + 14.0 / 5 ;            w      _____
b) string s = 2 + 3 + " = " + 2 + 3;       s           _____
c) int r = (int)12.0 / 5 + (int)(12.0 / 5 + 2 / 5);    r          _____
```

13. Določite tip in vrednost (int, double, string, ...) izrazov:

```
13.a).      (int)(2.0 / 3.0) + int.Parse("234")  _____
13.b).      1 + 2 + "3"                          _____
13.c).      (int) 2 + 3.0 - 3 + 6                _____
```

Naloge iz sestavljanja programov

1. Napišite program, ki prebere cela števila a, b, c in x ter izračuna vrednost polinoma $ax^2 + bx + c$. Poskrbite za lep izpis.
2. Za zrak se nam zdi, kot da nima teže. Vendar pa je količina zraka v nekem prostoru kar velika in to na koncu prinese tudi določeno težo. Napišite program, ki bo izračunal, kolikšna je masa zraka v sobi, ki ima obliko kvadra. Vse mere sobe vnašajte preko tipkovnice na cm natančno, rezultat izpišite na dag natančno. Gostota zraka je 1,3kg/m3.
3. Trgovsko podjetje Trgovec s.p. je hotelo veliko zaslužiti, zato je podražilo izdelke za 30%. Ker blaga po tako visoki ceni ni moglo prodati, je ceno znižalo za 25%. Kolikšna je nova cena izdelka? Začetno ceno izdelka vnesite preko tipkovnice.
4. Fiziki se veliko ukvarjajo s pretvarjanjem enot in se pri tem seveda veliko motijo. Da se ne bi zgodilo kaj katastrofalnega (predstavljaj si, da bi napačna pretvorba povzročila napako, zaradi katere bi se podrl novozgrajeni most ali nebotičnik!) jim napiši program, ki od uporabnika prebere volumen telesa v kubičnih milimetrih in jih pretvori v kubične decimetre, centimetre in milimetre (npr. 128021480 mm^3 je 128 dm^3 21 cm^3 480 mm^3)
5. Napišite program, ki izračuna obseg in ploščino pravokotnega trikotnika. Dolžini katet vnesemo v program na začetku preko tipkovnice.
   Namig: Pomagajte si z metodo Sqrt() iz razreda Math.
6. Sestavite program, ki bo prebral podatke o času odhoda in prihoda vlaka (oba podatka sta podana v urah in minutah). Izračunajte čas potovanja. Predpostavite lahko, da ima vlak odhod in prihod na isti dan. Seveda je predpostavka tudi, da se pri vnosu ne zmotimo in so vsi vneseni podatki smiselni (torej, da so minute in ure v ustreznih mejah in da je odhod pred prihodom).
7. Napišite program, ki obrne dano trimestno celo število (npr. iz 415 naredi število 514) in izpiše razliko prvotnega in obrnjenega števila.
8. Napišite program, ki dolžino, izraženo v centimetrih, pretvori v dolžino v palcih. En palec je 2.54 cm. Rezultat naj bo izpisan na dve decimalki.
9. Pleskarji pleskajo plavalni bazen, ki ima obliko kvadra (zgornje ploskve bazen seveda nima). Pomagajte jim in napišite program, ki bo izračunal, kolikšno površino sten bazena morajo prebarvati. Dolžino, širino in globino bazena vnesimo preko tipkovnice na dm natančno. Globino bazena merimo od roba bazena do dna.
10. Napiši program ki na osnovi danih robov kvadra izračuna in izpiše njegovo površino in prostornino.
11. Manca ima v šoli težave z mešanjem različnih alkoholnih raztopin. Ne ve, na primer, koliko vode naj dolije, da bi iz 80% alkohola dobila 65% alkohol. Njena mama pa bi doma potrebovala za rumov lonec 60% rum, na voljo ima pa 40%-nega in 75%-nega. Kako zmešati? Sestavi program, ki bo prebral, koliko imamo prve raztopine (v kg) ter koliko % alkohola je v prvi, v drugi in koliko naj ga bo v željeni raztopini. Na koncu naj izpiše, koliko druge tekočine je potrebno doliti, da dosežemo željeni odstotek alkohola. Upoštevajte tudi možnost, da z dolivanjem druge raztopine včasih ni mogoče doseči željene mešanice. V takšnem primeru naj program to jasno izpiše.
12. Tonček si pri realnih številih težko predstavlja, kateri je celi in kateri decimalni del števila. Zato mu napiši program, kjer bo lahko po vnosu števila dobil izpis celega in izpis decimalnega dela števila. Seveda naj se izpiše tudi vneseno število.
13. Zaradi utrujenosti se je učiteljica Nada nekajkrat zmotila pri že tako preprosti operaciji, kot je izračun povprečja dveh ocen. Da se ji to v prihodnje ne bi dogajalo, je prosila šolskega računalničarja naj ji sestavi program, ki izračuna poprečje dveh ocen (ocena je celo število med 1 in 5). Ker je računalničar zbolel, ji program napiši ti.
14. Stara mama rada šiva. Pri tem uporablja blago v traku in ker ga pogosto razreže, je želela, da bi ji napisal program, ki bi ji povedal, koliko metrov blaga ji še ostane, potem ko odreže njegov del. Želi vnesti dolžino celotnega traku in dolžino, ki jo bo odrezala. Seveda bi to lahko storila na pamet, vendar je že stara in težko računa z decimalnimi števili. Želi, da ji program izpiše dolžino celotnega traku, dolžino odrezanega dela in dolžino ostanka.

15. Oglej si naslednji program. Popravi ga tako, da bo deloval za trimestna (zelo lahko) in nato naredi še ne nov program, ki bo delal za petmestna števila (malo težje).

```
class Izpis {
  static void Main(string[] args)
   {
      int n = 5412;
      Console.WriteLine("n = " + n);
      Console.WriteLine("------------------");
      Console.WriteLine("enice | " + n % 10 );
      Console.WriteLine("------------------");
      Console.WriteLine("desetice | " + n / 10 % 10 );
      Console.WriteLine("------------------");
      Console.WriteLine("stotice | " + n / 100 % 10 );
      Console.WriteLine("------------------");
      Console.WriteLine("tisocice | " + n / 1000 );
      Console.WriteLine("------------------");
      Console.ReadLine();
    }
}
```

16. Kaj izpiše naslednji program

```
class NapacenIzpis {
  static void Main(string[] argc)   {
      int a = (1/2) * 4;
      double f = (3/2);
      //izpisemo rezultat
      System.Console.WriteLine("a=" + a + ", f=" + f);
   }
}
```

Popravi ga tako, da bo izpisal kot rezultat 2 in 1.5. Dopolni ta program tako, da boš v označenem delu napisal ukaze, s katerimi zamenjaš vrednosti spremenljivk a in b.

```
class ZamenjavaSpremenljivk
{
   static void Main(string[] argc)
   {
      int a = 234;
      int b = 23;
      System.Console.WriteLine("a = " + a);
      System.Console.WriteLine("b = " + b + "\n\n");
      // Zamenjamo vrednosti v a in b

        ...

      //izpisemo ponovno
      System.Console.WriteLine("a = " + a);
      System.Console.WriteLine("b = " + b + "\n\n");
      System.Console.ReadLine();
   }
}
```

17. Preberite menjalniški tečaj dolar/evro. Preračunajte koliko eurov boste dobili za določen znesek dolarjev.
18. Sestavi program, ki nam izračuna kota, ki ga opišeta urni in minutni kazalec v času, ki ga vpišemo. Ta čas je manjši kot 12h!
19. Ker nam je zmanjkalo denarja, smo morali porabiti celotni limit. Preberi višino limita, ki naj bo pozitivno celo število ter koliko mesecev ne bomo položili denarja na račun, da bi pokrili minus na računu. Napišite program, ki bo izračunal kakšne obresti nam zaračuna banka, če je mesečna obrestna mera 10,85% in računajo po navadnem obrestovanju (to je obresti se ne obrestujejo).

20. Tekač vsak dan teče v okolici svoje hiše. Pri tem meri čas od začetka do konca teka. Teče s konstantno hitrostjo 8 km/h. Nikoli pa ne ve, koliko km je dejansko pretekel. Sestavi program, ki bo tekaču pomagal pri izračunu dolžine poti. Preberi čas v urah in minutah npr. če je tekač tekel 1 uro in 45 minut, naj to zapiše kot 1,45 .
21. Doma imamo velik akvarij. Tako velik je, da so vse njegove mere podane kar v metrih. Zanima nas, koliko nas stane zamenjava vode, če kubični meter vode stane 0.42€. Sestavi program, ki prebere mere akvarija (kot decimalna števila), nato pa izpiše, koliko nas stane voda za ta akvarij.
22. Podjetnik Srečko ima kar nekaj podjetij, zato bi želel imeti program, ki bi mu ob koncu leta izračunal enostavne pokazatelje uspešnosti poslovanja podjetja. Pridobil je podatke za eno od svojih podjetij, ki proizvaja pivo. Količina proizvodov je 30.000 steklenic, cena ene steklenice piva je 0.5€, število delavcev je 15, celotni stroški proizvodnje so znašali 11500€, njegov vloženi kapital pa znaša 42000€. Sestavi program, ki bo izračunal ekonomičnost, produktivnost in rentabilnost za to podjetje in po potrebi tudi druga podjetja. (za neekonomiste: ekonomičnost je razmerje med vrednostjo produkcije in celotnimi stroški; vrednost produkcije je količina krat cena; produktivnost dela je razmerje med številom proizvodov in številom delavcev; rentabilnost pa izražamo s profitno mero, ki jo izračunamo tako, da profit ali dobiček delimo s kapitalom in pomnožimo z 100; profit dobimo, ko od vrednosti produkcije odštejemo celotne stroške).
23. Zanima nas, kako se je spremenilo število prebivalcev v enem letu. V statističnih podatkih najdemo podatke o številu umrlih in rojenih ter priseljenih in odseljenih. Na podlagi teh podatkov sestavi program, s katerim bomo ugotovili, za koliko ljudi se je spremenilo število prebivalcev v enem letu.
24. Družina se bo povečala za enega družinskega člana, zato smo se odločili, da bomo eno od sob v stanovanju preuredili v otroško sobo. Bele stene se nam zdijo dolgočasne, zato smo se odločili, da jih oblepimo s tapetami. Sestavi program, ki prebere mere otroške sobe v metrih in površino oken in vrat pa v kvadratnih metrih. Le teh ne bomo oblepili s tapetami, zato jih moraš odšteti od površine sten sobe. Z eno rolo tapet oblepiš 5 kvadratnih metrov sobe. Koliko rol tapet moraš kupiti?
25. Bliža se zima in v hotelu, kjer preživljaš zimski dopust, rabijo program, ki bo prebral število odraslih družinskih članov, število otrok ter izračunal skupno vrednost zimskega dopusta za vse vnesene osebe (odrasle in otroke) in vse to za določeno število dni. Število dni naj program prebere.

    Program naj izpiše v prvo vrstico število odraslih oseb ter ceno aranžmaja za enega odraslega za en dan, v drugo vrstico število otrok in ceno za enega otroka za en dan, v tretjo vrstico naj izpiše skupno vrednost ter na koliko dni se nanaša izračunana vrednost.
    Pomagaj jim – nagrada je enodnevna brezplačna smučarska karta za tvorca programa!

    Namig:
    - branje števila odraslih oseb, števila otrok ter števila dni (kot niz)
    - pretvorba nizov v števila in shranjevanje v spremenljivko
    - izračun vrednosti
    - izpis

26. Pri fiziki se je tvoj profesor domislil, da bi temperaturo izražali namesto v °C v °F. Ker ti je pretvarjanje vzelo preveč časa, si se odločil, da si boš za to napisal program, ki bo to počel namesto tebe. Program boš napisal tako, da bo temperaturo v °C najprej pretvoril v °F in nato spet nazaj v °C.

    Obrazec za pretvorbo °C v °F: `T(°F) = 1.8*T(°C) + 32`
    Obrazec za pretvorbo °F v °C: `T(°C) = (T(°F) - 32) / 1.8.`