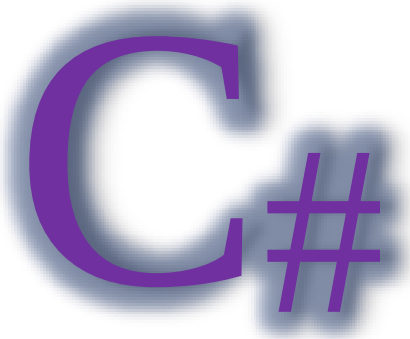
Elektrotehniško-računalniška strokovna šola in gimnazija Ljubljana

OSNOVE C#

e-knjiga



Melita Kompolšek

Elektrotehniško-računalniška strokovna šola in gimnazija Ljubljana

OSNOVE C#

e-knjiga

Melita Kompolšek

Ljubljana, 2016

Melita Kompolšek, Elektrotehniško-računalniška strokovna šola in gimnazija Ljubljana

OSNOVE C#

e-knjiga

Strokovni recenzenti:

doc. dr. Samo Simončič

Jezikovna recenzentka:

mag. Monja Pust

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004.43(075.8)(0.034.2)

KOMPOLŠEK, Melita
Osnove C# [Elektronski vir] : e-knjiga / Melita Kompolšek. - El. knjiga. - Ljubljana : Elektrotehniško-računalniška strokovna šola in gimnazija, 2016

Način dostopa
(URL): http://moodle.vegova.si/pluginfile.php/38463/mod_resource/content/1/Osnove%20Csharp.pdf

ISBN 978-961-93617-3-3 (pdf)

286860288

KAZALO

PREDGOVOR

E-knjiga Osnove C# je namenjena vsem začetnikom, ki želijo usvojiti osnove programskega jezika C# v razvojnem okolju Visual Studio. S pomočjo gradiva bomo spoznali temelje programiranja, na katerih lahko gradimo tehnološko znanje in veščine, da postanemo odličen programer.

Gradivo je namenjeno vsakomur, tudi popolnim začetnikom, ki želijo spoznati ali izpopolniti svoje programerske veščine. S tem namenom je vsebina razdeljena na logična poglavja, kjer je jasno razvidno, kaj je tematika obravnavanega poglavja. Za razumevanje vsebine ni potrebno predznanje, le nekaj osnovne računalniške pismenosti in želja po znanju programiranja.

Avtorica

1 UVOD V PROGRAMSKE JEZIKE

V gradivu je uporabljen programski jezik *C#*, ki je objektno usmerjen programski jezik in izvira pretežno iz programskih jezikov C++, Visual Basic in Java ter je oblikovan za delo z *Microsoftovo platformo .NET* ter razvojnim okoljem *Microsoft Visual Studio*, ki ga je podjetje Microsoft razvilo za pisanje programov v jeziku C#, C++ in Visual Basic.

Uporabne spletne povezave:

- Visual Studio: https://msdn.microsoft.com/en-us/library/dd831853.aspx
- C#: https://msdn.microsoft.com/en-us/library/kx37x362.aspx

1.1 Faze življenjskega cikla programske opreme

Prva faza življenjskega cikla programske opreme je *analiza problema in specifikacija želenega obnašanja grajenega sistema*. V tej fazi se raziskovalec ukvarja z analiziranjem problema, kaj bo potreboval za rešitev problema in čemu bo sistem namenjen.

Druga faza življenjskega cikla programske opreme je *načrtovanje sistema in njegovih delov*. Raziskovalec se v tej fazi ukvarja s pisanjem kode programskega jezika.

Tretjo fazo imenujemo *testiranje delov sistema*. V fazi testiranja raziskovalec testira posamezne dele programa in s tem najde ter odstrani napake v programu.

Četrta faza življenjskega cikla je *testiranje sistema*. Raziskovalec v tej fazi najprej testira program kot celoto, nato pa ga testira še v konkretnem okolju, s čimer ugotovi ali program v tem okolju ustrezno deluje.

Zadnja faza življenjskega cikla programske opreme je *uporaba in vzdrževanje opreme*, kjer raziskovalec programsko opremo in njeno dokumentacijo preda uporabniku v uporabo. V primeru pomanjkljivosti, raziskovalec tekom uporabe opreme sistem popravi oziroma ga po potrebi nadgradi.

1.2 Vloga programskega jezika C#

C# je sodoben, objektno usmerjen, splošno namenski programski jezik, ki ga je skupaj s *platformo .NET* razvilo podjetje Microsoft. S programskim jezikom C# in platformo .NET lahko razvijamo raznoliko programsko opremo: pisarniške aplikacije, spletne aplikacije, spletne strani, namizne aplikacije, mobilne aplikacije, igre in še mnoge druge aplikacije.

Nakov (et al., 2013, 18–19) opredeljuje programski jezik C# kot enega izmed najbolj priljubljenih programskih jezikov. Uporablja ga na milijone razvijalcev po celem svetu. Ker je programski jezik C# razvilo podjetje Microsoft kot del njihove sodobne platforme za razvoj in izvedbo aplikacij .NET Framework, je jezik zelo razširjen med privrženci Microsofta.

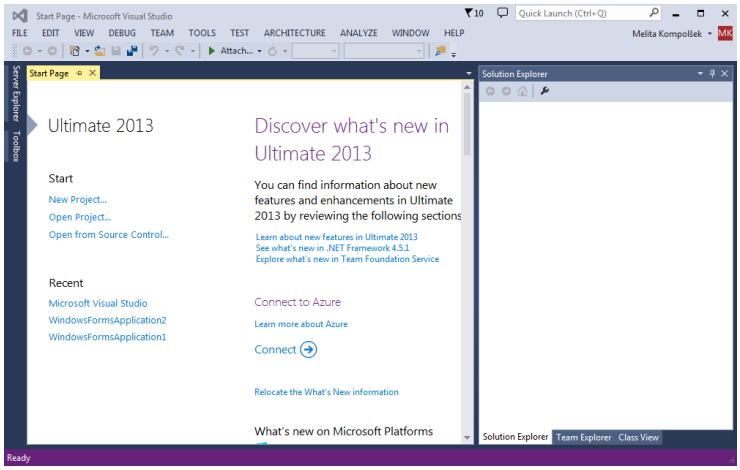
1.3 Potrebna programska oprema

1.3.1 Visual Studio

Visual Studio je integrirano razvojno okolje, ki združuje urejevalnik kode, prevajalnik, razhroščevalnik, orodja za dokumentacijo programov in druga orodja, ki so nam v pomoč pri pisanju programskih aplikacij. Podpira različne programske jezike (C#, VB.NET, C++) in različna razvojna okolja (Win32, COM, ASP.NET, ADO.NET Entity Framework, Windows Forms, WPF, Silverlight, Windows Store aplikacije in druge Windows in .NET tehnologije).

Razvoj novega projekta

Ob zagonu razvojnega okolja *Visual Studio* se prikaže programsko okno, podobno programskemu oknu na Slika 1 (v odvisnosti od različice Visual Studio se lahko nekoliko razlikuje).

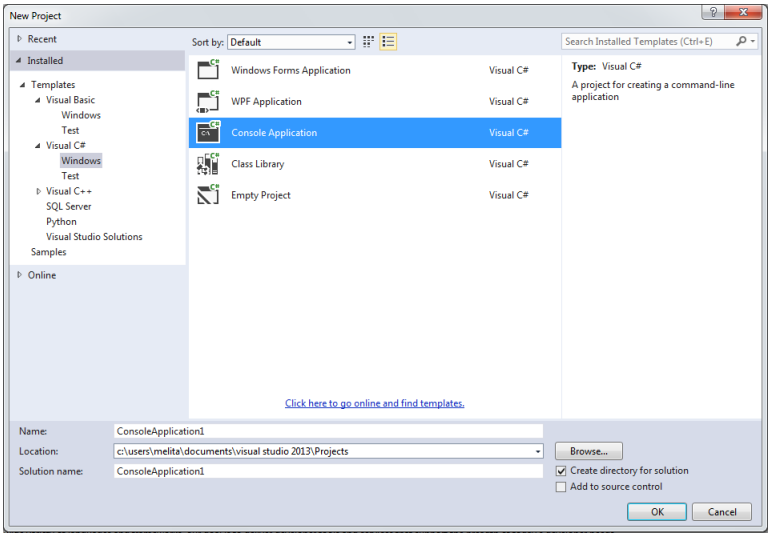


Slika 1: Vstopno okno razvojnega okolja Microsoft Visual Studio

Vir: Lasten (2015)

Nov projekt

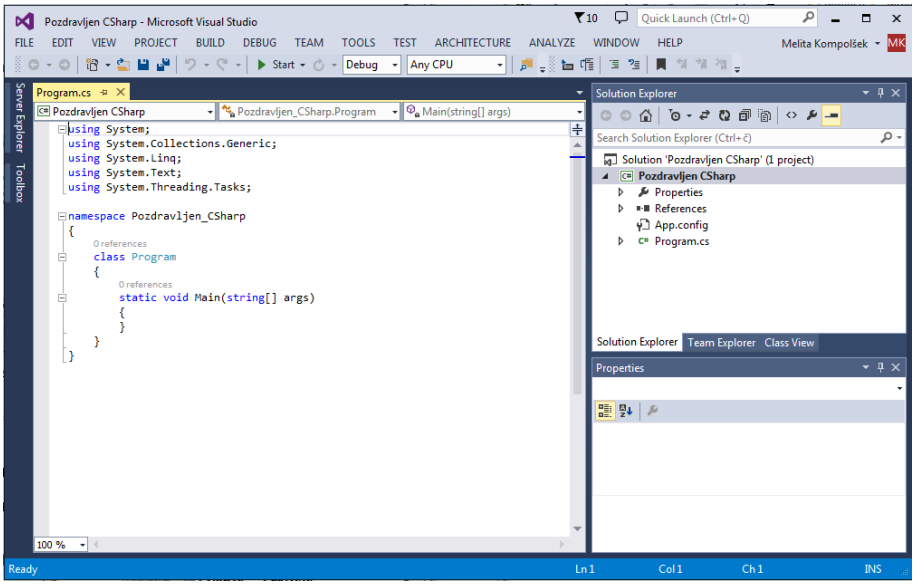
Nov projekt ustvarimo s pomočjo pogovornega okna *New Project*, ki ga najdemo na zavihku *File*.



Slika 2: Pogovorno okno New Project v razvojnem okolju Microsoft Visual Studio

Vir: Lasten (2015)

Med ponujenimi možnostmi izberemo tip projekta *Console Application* in programski jezik *C#*. Na dnu okna lahko konzolno aplikacijo tudi preimenujemo (*Name*) in spremenimo njeno lokacijo shranjevanja (*Location*). Predno s klikom na gumb OK ustvarimo nov projekt, še preverimo, ali je obkljukana možnost *Create directory for solution*. S pritiskom na gumb OK se prikaže osnovno programsko okno razvojnega okolja Microsoft Visual Studio.



Slika 3: Osnovno programsko okno razvojnega okolja Microsoft Visual Studio

Vir: Lasten (2015)

Osnovno programsko okno vsebuje več podoken:

- *Start page* (začetna stran) – iz začetne strani lahko enostavno odpremo kateregakoli od naših zadnjih projektov ali začnemo nov projekt, da ustvarimo svoj prvi C# program.
- *Code Editor* (urejevalnik – okno za pisanje oz. urejanje kode) – ohranja izvorno kodo programa in omogoča odpiranje in urejanje več datotek.
- *Error List* (seznam napak) – prikaže morebitne napake v programu.
- *Solution Explorer* (raziskovalec rešitve) – okno prikaže strukturo našega projekta, in sicer vse datoteke, ki jih projekt vsebuje (če ni naloženega nobenega projekta, je to okno prazno).
- *Properties* (lastnosti) – vsebuje seznam lastnosti trenutnega objekta.

Okostje programa

Ob zagonu novega projekta se prikaže *okostje* programa.

PRIMER 1: Okostje programa:

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

```
using System.Text;
using System.Threading.Tasks;

namespace Pozdravljen_CSharp
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }
}
```

Pomen posameznih delov okostja trenutno ni pomemben. Kot je razvidno iz samega okostja, imamo razred z imenom *class Program* in v njem metodo *Main*. Potrebno se je zavedati, da moramo vse, kar bomo dopisali v kodo programa, napisati znotraj metode *Main* med zavite oklepaje. Metoda *Main* mora imeti samo en parameter *string[]* in ime, ki je v primeru 1 *args*.

1.3.2 .NET Framework

Programski jezik C# ni samostojen izdelek, pač pa predstavlja del Microsoftove platforme *.NET Framework*. Le ta je običajno sestavljen iz okolja za razvoj in izvajanje programov, napisanih v C# ali kakem drugem programskem jeziku, ki je združljiv z .NET (VB.NET, C++, J# ali F#).

Sestoji iz:

- *.NET programskih jezikov* (C#, VB.NET in drugi),
- okolja *Common Language Runtime*, ki izvede program, zapisan v jeziku C#,
- *nabora razvojnih orodij*,

- *nabora standardnih knjižnic*, kot na primer ADO.NET, ki omogočajo dostop do baz podatkov (kot so MS SQL Server ali MySQL) in WCF, ki povezuje aplikacije preko standardnih komunikacijskih okvirjev in protokole, kot so HTTP, REST, JSON in SOAP.

1.4 Avtorske pravice

Zakon o avtorskih in sorodnih pravicah opredeljuje avtorske pravice kot pravice avtorjev na njihovih delih s področja književnosti, znanosti in umetnosti ter pravice izvajalcev, proizvajalcev fonogramov, filmskih producentov, radijskih ali televizijskih organizacij, založnikov in izdelovalcev podatkovnih baz.

Avtorska pravica ni absolutna, pod nekaterimi pogoji je mogoče avtorsko delo celo prosto uporabljati. Avtorsko delo je mogoče uporabiti za citiranje, pri čemer je navedba avtorja obvezna, ali pa ga uporabiti za izobraževalne namene, itd.

Avtor programske opreme ima avtorske pravice nad to programsko opremo ter s tem možnost omejitve nekaterih pravic uporabnikov te opreme. Morebitne omejitve pravic uporabnika programske opreme so zapisane v javnem dokumentu imenovanem *licenca*.

Glede na omejitve pavic uporabnika obstaja več vrst programskih oprem:

- *Brezplačna programska oprema* je programska oprema, ki je uporabnikom na voljo brezplačno.
- *Prosta programska oprema* je programska oprema za katero velja uporabnikova svoboda, tj. uporaba za katerikoli namen, preučevanje načina delovanja, pravica do redistribucije izvorne kode in pravica do izboljšav izvorne kode.
- *Lastniška programska oprema* je programska oprema, ki je uporabnikom na voljo brezplačno ali za manjšo denarno protivrednost.
- *Plačljiva programska oprema* je programska oprema, ki jo proizvajajo podjetja v zameno za denarno plačilo njenih uporabnikov.

Posamezniki se pri plagiatorstvu računalniških programov pogosto izgovarjajo, da nelegalne kopije programov in multimedijskih vsebin uporabljajo zgolj v izobraževalne namene in da z njimi ne opravljajo pridobitvene dejavnosti, vendar pa je potrebno omeniti, da zakonodaja precej natančno določa kaj so izobraževalni nameni, poleg tega pa prosta uporaba ni povsem

brez zakonskih omejitev, saj zakon tudi v teh primerih postavlja nekatere omejitve. Pri računalniškem plagiatorstvu ima zakonodaja še posebej stroge omejitve, saj določa, da lahko upravičeni uporabnik računalniškega programa reproducira *največ dva varnostna primerka programa* in da je *javno posojanje računalniških programov in baz podatkov izključna pravica njihovega avtorja*.

Kazni za kršitve avtorskih pravic so opredeljene v Kazenskem zakoniku, ki v skladu s 158. členom *Kršitev avtorske pravice* določa, da se tistega, ki s svojim imenom ali z imenom koga drugega objavi, prikaže, izvede ali prenese tuje avtorsko delo ali njegov del, ali dovoli to storiti, kaznuje z denarno kaznijo ali z zaporom do enega leta in tistega, ki skazi, okrni ali kako drugače neupravičeno poseže v tuje avtorsko delo, se kaznuje z denarno kaznijo ali z zaporom do šestih mesecev.

1.5 Vprašanja in naloge z namigi in rešitvami

1. Naložite in se spoznajte z razvojnim okoljem Microsoft Visual Studio.

   Namig: Na spletni povezavi https://msdn.microsoft.com/en-us/library/dd831853.aspx poiščite brezplačno različico Visual Studio Expres.

2. Opišite razliko med C# in .NET Framework.

   Namig: Razlike poiščite na internetu (npr. Wikipedia) in jih podrobneje preučite. Ugotovili boste, da je C# programski jezik, medtem ko je .NET Framework platforma za razvoj in izvršitev .NET kode.

3. Sestavite seznam najpopularnejših programskih jezikov.

   Namig: Ugotovite, kateri so najbolj priljubljeni jeziki in preučite nekaj vzorčnih programov, napisanih v teh jezikih. Te programske jezike primerjajte z jezikom C#. Podrobneje si oglejte jezike C, C++, Java, C#, VB.NET, PHP, JavaScript, Perl, Python in Ruby.

1.6 Povzetek

V tem poglavju so predstavljene in opisane faze življenjskega cikla programske opreme in pomen enega najpogosteje uporabljenih programskih jezikov C#. V nadaljevanju poglavja je predstavljeno integrirano razvojno okolje Visual Studio, ki vsebuje urejevalnik kode, razhroščevalnik, prevajalnik, orodja za dokumentacijo programov in druga orodja za pisanje programskih aplikacij. Pomemben del tega poglavja je tudi opis platforme .NET Framework, del katere je programski jezik C#. Na koncu poglavja so naštete in predstavljene avtorske pravice programske opreme.

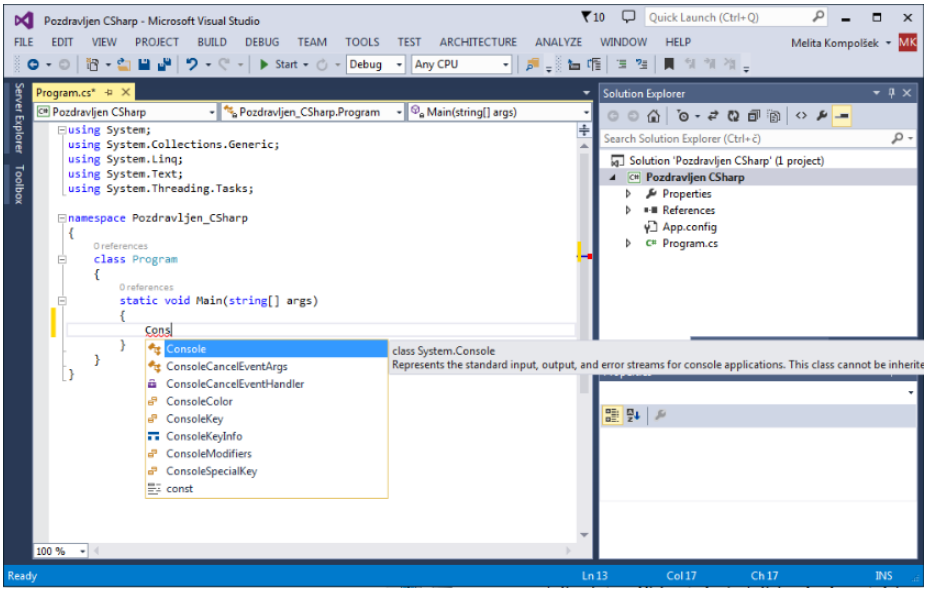
2 OSNOVE PROGRAMIRANJA

2.1 Zgradba programa

Sharp (2013, 8–14) poudarja, da datoteka Program.cs definira *razred* (angl. class) (v našem primeru smo ga poimenovali Program) in vsebuje metodo, imenovano *Main*. V programskem jeziku C# mora biti vsa programska koda opredeljena v *metodah* in vse metode morajo pripadati *razredu*.

2.2 Izpis v konzoli

Program za izpis v konzoli sestavimo tako, da v okno, ki prikazuje datoteko Program.cs postavimo kazalko v glavno metodo in takoj za zavitim uklepajem ({) pritisnemo tipko Enter ter s tem v programu ustvarimo novo vrstico. V novo vrstico vnesemo besedo *Console*, ki predstavlja razred za izpis sporočila na oknu konzole. Pri pisanju besede Console opazimo lastnost urejevalnika, ki nam glede na zapisane črke ponudi možne izbire v obliki spustnega seznama (Slika 4). Programski jezik C# je občutljiv pri uporabi velikih in malih črk, zato moramo paziti, da napišemo črke metod točno tako, kot je navedeno.



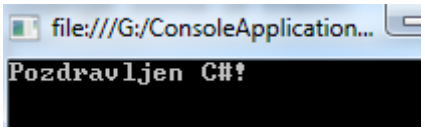
Slika 4: Možne izbire pri vpisu besede Console

Vir: Lasten (2015)

Za besedo Console zapišemo piko in ponovno se pojavi nov spustni seznam z možnimi metodami, med katerimi izberemo *WriteLine*. Za *Console.WriteLine* vpišemo še oklepaj ter v narekovajih besedilo, za katero želimo, da se izpiše na konzoli, in zaključimo z zaklepajem in podpičjem. Kodo zaključimo še s *Console.Read.Key()* ali *Console.Read.Line()*, ki poskrbita, da se konzola zapre, ko pritisnemo tipko oz. tipko Enter.

PRIMER 2: Program, ki izpiše stavek "Pozdravljen C#!":

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



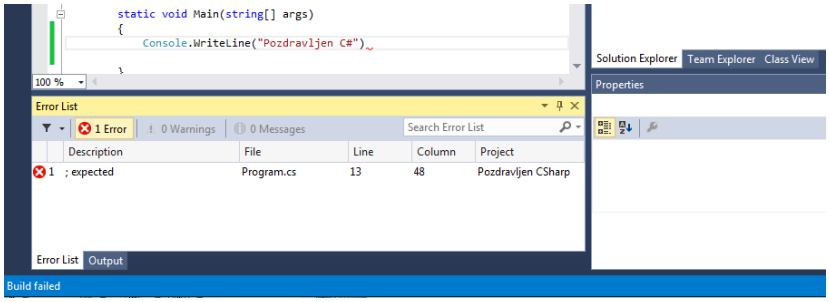
Slika 5: Izpis programa pri primeru 2
Vir: Lasten (2015)

Večkrat v kodi programa srečamo dve poševnici (//), katerima sledi običajno besedilo. Ti dve poševnici nakazujeta, da za njima stoji *komentar*, ki ga prevajalnik kode ignorira in služi zgolj komentiranju kode. Zapis *večvrstičnega komentarja* zapišemo med znakoma /* in */.

PRIMER 3: Program, ki vsebuje komentar:

```
static void Main(string[] args)
        {
            Console.ReadLine(); // Čaka uporabnika, da pritisne
            tipko Enter
        }
```

Zagon programa izvedemo s pritiskom na gumb *Start*. V primeru, da se v zapisu kode samega programa nahaja kakšna napaka, se le-ta izpiše v podoknu *Error list* (Slika 6).



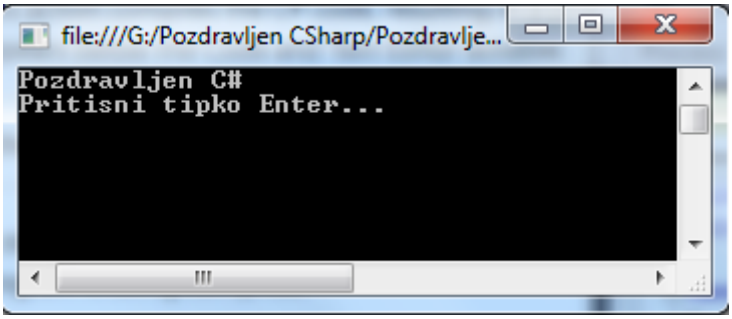
Slika 6: Error list

Vir: Lasten (2015)

Ob pritisku na gumb *Start* se pri programu brez napak odpre konzola in program se izvede.

PRIMER 4: Program, ki izpiše stavek "Pozdravljen C#!" in "Pritisni tipko Enter …":

```
static void Main(string[] args)
        {
            Console.WriteLine("Pozdravljen C#");
            Console.WriteLine("Pritisni tipko Enter...");
            Console.ReadLine();
        }
```



Slika 7: Izpis programa pri primeru 4

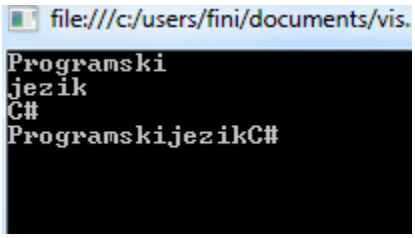
Vir: Lasten (2015)

2.2.1 Metodi Write() in WriteLine()

Osnovni način izpisovanja v konzolo je z metodama *Console.Write()* in *Console.WriteLine()*. Obe metodi izpišeta besedilo, ki se nahaja med narekovaji (") znotraj okroglih oklepajev, razlika med njima je le, da metoda *Console.WriteLine()* po izpisu besedila prestavi položaj začetka naslednjega izpisa na začetek nove vrstice, medtem ko metoda *Console.Write()* po izpisu besedila položaj začetka naslednjega izpisa pusti v isti vrstici, desno od zadnjega izpisanega znaka.

PRIMER 5: Program, ki izpiše besedilo s pomočjo metode Console.Write() in Console.WriteLine():

```
static void Main(string[] args)
        {
            Console.WriteLine("Programski");
            Console.WriteLine("jezik");
            Console.WriteLine("C#");
            Console.Write("Programski");
            Console.Write("jezik");
            Console.Write("C#");
            Console.ReadKey();
        }
```



Slika 8: Izpis programa pri primeru 5

Vir: Lasten (2015)

2.3 Deklaracija spremenljivk in podatkovni tipi programskega jezika

Spremenljivke so lokacije za skladiščenje podatkov, na podlagi vrste teh podatkov pa spremenljivki določimo podatkovni tip.

Osnovni *podatkovni tipi* v programskem jeziku C# so razdeljeni v naslednje vrste:

- cela števila: *int*;
- realna (decimalna) števila: *double*;
- logične operacije: *bool*;
- znak: *char*;
- niz: *string*;
- objekt: *object*.

Obravnavani podatkovni tipi so osnovni in so vgrajeni v C# jeziku na najnižji ravni. Spodnja tabela predstavlja zgoraj navedene vrste podatkovnih tipov, njihov obseg in njihove privzete vrednosti:

Podatkovni tipi	Privzeta vrednost	Minimalna vrednost	Maksimalna vrednost
sbyte	0	-128	127
byte	0	0	255
short	0	-32768	32767
ushort	0	0	65535
int	0	-2147483648	2147483647
uint	0u	0	4294967295
long	0L	-9223372036854775808	9223372036854775807
ulong	0u	0	18446744073709551615
float	0.0f	±1.5×10-45	±3.4×1038
double	0.0d	±5.0×10-324	±1.7×10308
decimal	0.0m	±1.0×10-28	±7.9×1028
bool	false	Two possible values: **true** and **false**	
char	'\u0000'	'\u0000'	'\uffff'
object	null	-	-
string	null	-	-

Spremenljivke moramo vedno najaviti z *deklaracijskim stavkom*:

podatkovniTip imeSpremenljivke;

PRIMER 6: Deklariranje spremenljivk:

```
static void Main(string[] args)
        {
            int stevilo;
            double a, b, c;
            string niz;
        }
```

Spremenljivki lahko ob najavi priredimo tudi začetno vrednost:

podatkovniTip imeSpremenljivke = začetnaVrednost;

Pri tem je potrebno poudariti, da se desna vrednost vedno shranjuje v levo spremenljivko oz. konstanto.

PRIMER 7: Prireditev začetne vrednosti spremenljivki:

```
class Program
    {
        static void Main(string[] args)
        {
            int stevilo = 5;
            double a = 0.5;
            string niz = "Začetni niz";
        }
```

2.3.1 Cela števila in aritmetične operacije

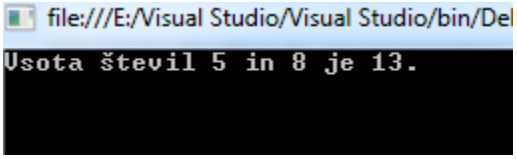
Najpogostejši tip za hranjenje celih števil v spremenljivki je *int*. Seveda je potrebno poudariti, da pri tem ne gre za poljubna cela števila, kot jih poznamo pri matematiki, saj v pomnilnik računalnika ne moremo shraniti neskončno mnogo števil. Spremenljivke tipa int lahko zavzamejo le vrednosti od -2147483648 do 2147483647.

Nad celimi števili lahko uporabljamo vse aritmetične operacije:

- *množenje* (*),
- *celoštevilsko deljenje* (/),
- *seštevanje* (+),
- *odštevanje* (-),
- *celoštevilski ostanek pri deljenju* (%).

PRIMER 8: Program, ki sešteje dve celi števili in vsoto števil (spremenljivko podatkovnega tipa int) izpiše v konzoli:

```
static void Main(string[] args)
        {
            int stevilo1 = 5;
            int stevilo2 = 8;
            int vsota = stevilo1 + stevilo2;
            Console.WriteLine("Vsota števil " + stevilo1 + " in "
            + stevilo2 + " je " + vsota + ".");
            Console.ReadLine();
        }
```



Slika 9: Izpis programa pri primeru 8

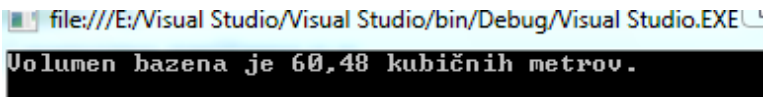
Vir: Lasten (2015)

2.3.2 Realna (decimalna) števila

Realna (decimalna) števila najpogosteje shranjujemo v spremenljivkah tipa *double*. Pri zapisu števil pazimo, da uporabljamo *decimalno piko*, ki ločuje celi del in decimalni del števila. Tudi nad realnimi števili lahko izvajamo vse osnovne računske operacije.

PRIMER 9: Program, ki izračuna volumen bazena z 10,5 m dolgo, 3,6 m široko in 1,6 m visoko stranico in izpiše spremenljivko »volumen« podatkovnega tipa double:

```
static void Main(string[] args)
        {
            double dolžina = 10.5;
            double širina = 3.6;
            double višina = 1.6;
            double volumen = dolžina * širina * višina;
            Console.WriteLine("Volumen bazena je " + volumen + "
            kubičnih metrov.");
            Console.ReadLine();
        }
```



Slika 10: Izpis programa pri primeru 9
Vir: Lasten (2015)

2.3.3 Logične operacije

Logične spremenljivke so spremenljivke, ki lahko zavzamejo le dve vrednoti, pravilno (*true)* ali nepravilno (*false)*. Označimo jih s tipom *bool*.

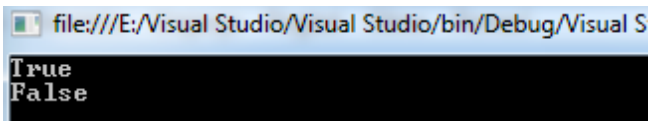
Nad logičnimi spremenljivkami lahko uporabljamo naslednje osnovne operacije:

- && *logični in (and*) – vrne *true*, kadar sta oba pogoja resnična;
- || *logični ali* (*or*) – vrne *true*, kadar je vsaj en pogoj resničen;

- ! *negacija* (*not*) – vrne *false*, če je spremenljivka *true* in *true*, če je spremenljivka *false*;
- == *enako* – pogoj je izpolnjen, če sta vrednosti obeh izrazov enaki;
- != *različno* – pogoj je izpolnjen, če sta vrednosti obeh izrazov različni;
- > *večje* – pogoj je izpolnjen, če je vrednost levega izraza večja od vrednosti desnega izraza;
- >= večje ali enako – pogoj je izpolnjen, če je vrednost levega izraza večja ali enaka od vrednosti desnega izraza;
- < *manjše* – pogoj je izpolnjen, če je vrednost levega izraza manjša od vrednosti desnega izraza;
- <= *manjše ali enako* – pogoj je izpolnjen, če je vrednost levega izraza manjša ali enaka od vrednosti desnega izraza.

PRIMER 10: Program, ki izpiše vrednost spremenljivk podatkovnega tipa bool:

```
static void Main(string[] args)
        {
            int a = 1;
            int b = 2;
            bool manjsi = (a < b);
            bool enak = (a == 0);
            Console.WriteLine(manjsi);
            Console.WriteLine(enak);
            Console.ReadLine();
        }
```



Slika 11: Izpis programa pri primeru 10

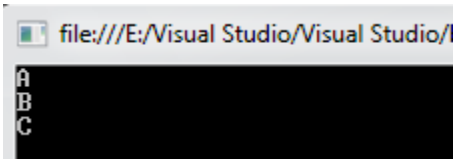
Vir: Lasten (2015)

2.3.4 Znak

Spremenljivko, v kateri želimo hraniti posamezen znak, označimo s tipom *char*. Kot znak se šteje vsaka mala ali velika črka, številka ter posebni (pika, vejica, podpičje ...) in nevidni znaki (tabulator, prelom vrstice ...). Vse možne znake je moč najti v tako imenovani ASCII-tabeli. Znake pišemo med enojnimi narekovaji.

PRIMER 11: Program, ki izpiše spremenljivke podatkovnega tipa char:

```
static void Main(string[] args)
        {
            char znakA = 'A';
            char znakB = 'B';
            char znakC = 'C';
            Console.WriteLine(znakA);
            Console.WriteLine(znakB);
            Console.WriteLine(znakC);
            Console.ReadLine();
        }
```



Slika 12: Izpis programa pri primeru 11

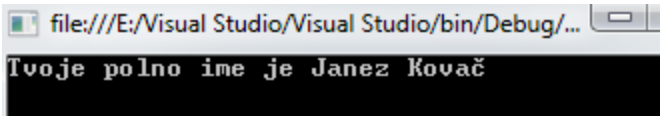
Vir: Lasten (2015)

2.3.5 Niz

V spremenljivke lahko hranimo zaporedje več znakov. Takšno spremenljivko označimo s tipom *string*.

PRIMER 12: Program, ki izpiše spremenljivko podatkovnega tipa string:

```
static void Main(string[] args)
        {
            string ime = "Janez";
            string priimek = "Kovač";
            string polnoIme = ime + " " + priimek;
            Console.WriteLine("Tvoje polno ime je " + polnoIme);
            Console.ReadLine();
        }
```



Slika 13: Izpis programa pri primeru 12

Vir: Lasten (2015)

Dostop do znakov v nizu

Niz je lahko prazen ali pa sestavljen iz več znakov. Vsak znak v nizu ima določen svoj *indeks*. Do posameznega znaka v nizu dostopamo s pomočjo *oglatih oklepajev*, v katere zapišemo indeks znaka. Indeksiranje znakov se na skrajni levi strani niza začne z 0 in se konča z dolžino niza – 1. S pomočjo indeksov znake v nizu preberemo, ne moremo pa jih spreminjati.

PRIMER 13: Program, ki izpiše šesti znak v nizu:

```
static void Main(string[] args)
        {
            string niz = "Pozdravljen C#";
            Console.WriteLine(niz[5]);
            Console.ReadLine();
        }
```

Dolžina niza

Podatek o dolžini niza pridobimo s pomočjo ukaza *Length*.

PRIMER 14: Program, ki vrne dolžino niza:

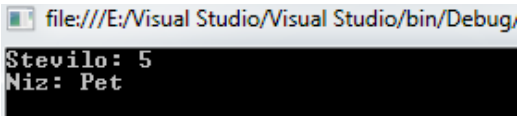
```
static void Main(string[] args)
        {
            string niz = "Pozdravljen C#";
            int dolzinaNiza = niz.Length;
            Console.WriteLine("Dolžina niza: " + dolzinaNiza);
            Console.ReadLine();
        }
```

2.3.6 Objekt

Objekt je poseben podatkovni tip, saj združuje vse ostale tipe. V spremenljivko tipa *object* lahko hranimo kakršnokoli vrsto podatka.

PRIMER 15: Program, ki izpiše spremenljivko podatkovnega tipa object:

```
static void Main(string[] args)
        {
            object stevilo = 5;
            object niz = "Pet";
            Console.WriteLine("Stevilo: " + stevilo);
            Console.WriteLine("Niz: " + niz);
            Console.ReadLine();
        }
```



Slika 14: Izpis programa pri primeru 15

Vir: Lasten (2015)

2.4 Matematične funkcije

Programski jezik C# ponuja pri računanju absolutnih vrednosti, kvadratnih korenov, zaokroževanju in operacijah razred *Math*, ki vsebuje številne metode za uporabo matematičnih funkcij.

Lokar in Uranič (2009, 27–28) navajata nekaj pomembnejših metod in konstant:

Abs ..... vrne absolutno vrednost števila

Acos ..... vrne kot v radianih, katerega kosinus je argument metode

Asin ..... vrne kot v radianih, katerega sinus je argument metode

Atan ..... vrne kot v radianih, katerega tangens je argument metode

Cos ..... vrne kosinus kota

Cosh ..... vrne hiperbolični cosinus kota

Log ..... vrne logaritem

Max ..... vrne večje od dveh števil v argumentu metode

Min ..... vrne manjše od dveh števil v argumentu metode

Pow ..... vrne izračunano potenco števila

Round ..... vrne zaokroženo število

Sin ..... vrne sinus kota

Sinh ..... vrne hiperbolični sinus kota

Sqrt ..... vrne kvadratni koren števila

Tan ..... vrne tangens kota

Tanh ..... vrne hiperbolični tangens kota

Truncate ..... vrne celi del števila

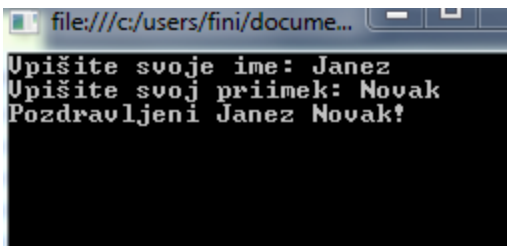
PI..... vrne vrednost konstante iracionalnega števila PI

2.5 Branje podatkov

Za branje iz konzole uporabljamo metodo *Console.ReadLine()*. Omenjena metoda od uporabnika zahteva vnos podatka preko konzole. Ob koncu vnosa podatka v konzolo uporabnik pritisne tipko Enter in s tem uporabnik končal z vnosom.

PRIMER 16: Program, ki od uporabnika zahteva ime in priimek:

```
static void Main(string[] args)
        {
            Console.Write("Vpišite svoje ime: ");
            string ime = Console.ReadLine();
            Console.Write("Vpišite svoj priimek: ");
            string priimek = Console.ReadLine();
            Console.WriteLine("Pozdravljeni " + ime + " " +
            priimek + "!");
            Console.ReadKey();
        }
```



Slika 15: Izpis programa pri primeru 16

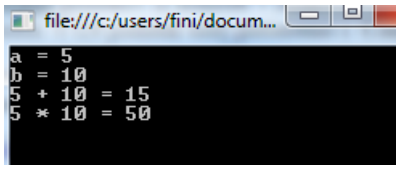
Vir: Lasten (2015)

2.6 Pretvarjanje prebranega niza v drug podatkovni tip

Branje števil iz konzole v programskem jeziku C# ne poteka direktno, saj je prebrani niz podatkovnega tipa (*string*), ki ga je potrebno pretvoriti v število oziroma podatkovni tip, ki ga potrebujemo. Za pretvarjanje iz podatkovnega tipa string v nek drugi podatkovni tip uporabljamo metodo *Parse* ali metodo *Convert*.

PRIMER 17: Program, ki pretvori podatkovni tip string v podatkovni tip int:

```
static void Main(string[] args)
        {
            Console.Write("a = ");
            int a = int.Parse(Console.ReadLine());
            Console.Write("b = ");
            int b = int.Parse(Console.ReadLine());
            Console.WriteLine(a + " + " + b + " = " + (a + b));
            Console.WriteLine(a + " * " + b + " = " + (a * b));
            Console.ReadKey();
        }
```



Slika 16: Izpis programa pri primeru 17

Vir: Lasten (2015)

2.7 Pogojni stavek (if in if-else)

Pogojne stavke uporabljamo, ko želimo, da se neko dejanje zgodi le v primeru, da je izpolnjen pogoj.

2.7.1 Pogojni stavek if

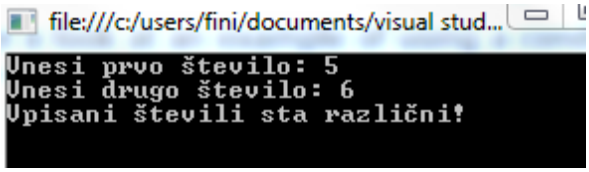
Pogojni stavek if se izvede, če je izpolnjen *pogoj*. Pogoj zapišemo v okroglih oklepajih in gre za nek logičen izraz, ki vrne vrednost *true* ali *false*. Pogoju sledi stavek ali več stavkov, ki jih želimo izvesti v primeru, da je pogoj izpolnjen. Te stavke zapišemo znotraj zavitega oklepaja ({}). Če je vrednost pogoja *true*, se izvede telo pogojnega stavka, če je vrednost *false*, bodo vse operacije v telesu pogojnega stavka preskočene.

if (pogoj)
{
    stavek$_1$;
    stavek$_2$;
    ...
    stavek$_n$;
}

PRIMER 18: Program, ki uporabnika vpraša po dveh številih, ki ju nato primerja. Če sta števili različni, program izpiše "Vpisani števili sta različni!", v nasprotnem primeru se ne zgodi nič:

```
static void Main(string[] args)
        {
            Console.Write("Vnesi prvo število: ");
            int stevilo1 = int.Parse(Console.ReadLine());
            Console.Write("Vnesi drugo število: ");
            int stevilo2 = int.Parse(Console.ReadLine());
            if (stevilo1 != stevilo2)
            {
                Console.WriteLine("Vpisani števili sta
```

```
                različni!");
                Console.ReadKey();
            }
        }
```



Slika 17: Izpis programa pri primeru 18

Vir: Lasten (2015)

2.7.2 Pogojni stavek if – else

Pogojni stavek if – else uporabimo, če želimo, da se v primeru izpolnjenega pogoja izvede del kode, ki pripada danemu pogoju in obratno.

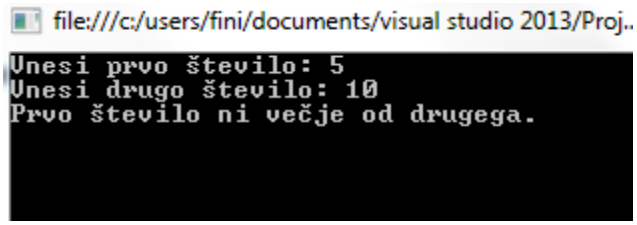
if (pogoj)
{
    stavek$_{p1}$;
    stavek$_{p2}$;
    …
    stavek$_{pn}$;
}
else
{
    stavek$_{r1}$;
    stavek$_{r2}$;
    …
    stavek$_{rm}$;
}

V primeru, da bo pogoj izpolnjen, se bodo izvedli stavki $stavek_{p1}$, $stavek_{p2}$, … ,$stavek_{pn}$; v primeru, da pogoj ne bo izpolnjen, pa stavki $stavek_{r1}$, $stavek_{r2}$, …, $stavek_{rm}$. Če je v bloku le en stavek, lahko zavite oklepaje izpustimo:

if (pogoj) $stavek_1$;
else $stavek_2$;

PRIMER 19: Program, ki iz konzole prebere dve števili, ju primerja in izpiše ali je prvo število večje od drugega:

```
static void Main(string[] args)
        {
            Console.Write("Vnesi prvo število: ");
            int stevilo1 = int.Parse(Console.ReadLine());
            Console.Write("Vnesi drugo število: ");
            int stevilo2 = int.Parse(Console.ReadLine());
            if (stevilo1 > stevilo2)
                Console.WriteLine("Prvo število je večje od
                drugega.");
            else
                Console.WriteLine("Prvo število ni večje od
                drugega.");
            Console.ReadKey();
        }
```



Slika 18: Izpis programa pri primeru 19

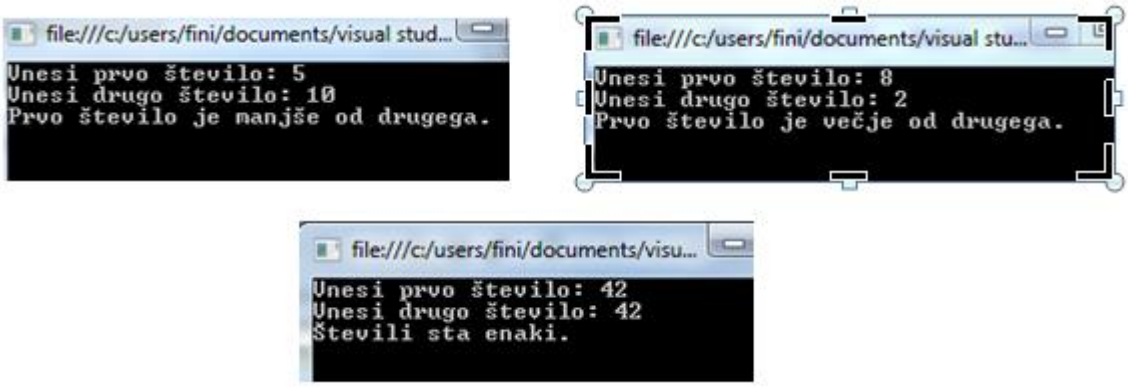
Vir: Lasten (2015)

2.7.3 Gnezdeni stavek

Včasih potrebujemo več pogojnih stavkov if ali if – else znotraj eden drugega, kar imenujemo *gnezdenje*. Torej gnezdenje pomeni, da se pogojni stavek if ali if – else nahaja v telesu nekega drugega pogojnega stavka if ali if – else.

PRIMER 20: Program, ki prebere dve števili in ugotovi, ali sta njuni vrednosti enaki, ter to izpiše:

```
static void Main(string[] args)
        {
            Console.Write("Vnesi prvo število: ");
            int stevilo1 = int.Parse(Console.ReadLine());
            Console.Write("Vnesi drugo število: ");
            int stevilo2 = int.Parse(Console.ReadLine());
            if (stevilo1 == stevilo2)
                Console.WriteLine("Števili sta enaki.");
            else
            {
                if (stevilo1 > stevilo2)
                    Console.WriteLine("Prvo število je večje od
                    drugega.");
                else
                    Console.WriteLine("Prvo število je manjše od
                    drugega.");
            }
            Console.ReadKey();
        }
```



Slika 19: Izpis programa pri primeru 20

Vir: Lasten (2015)

2.8 Zanka (while in for)

Lokar (2005, 56–66) navaja, da *zanke* uporabljamo, ko želimo enega ali več stavkov večkrat ponoviti. Najpogosteje se uporabljata zanki *for* in *while*.

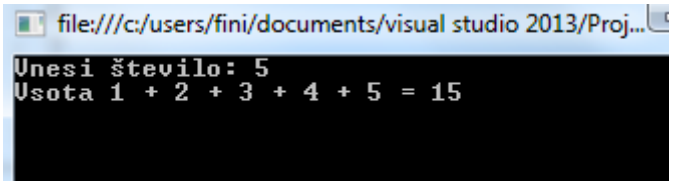
2.8.1 While

Zanka while se uporablja najpogosteje, saj lahko z njo zapišemo katerokoli zanko. Uporabimo jo, ko želimo, da se nek stavek ponavlja, dokler je pogoj izpolnjen.

```
while (pogoj)
{
        stavek1;
        stavek2;
        ...
        stavekn;
}
```

PRIMER 21: Program, ki sešteje števila od 1 do števila, ki ga uporabnik vpiše v konzolo. V programu najprej določimo vrednost spremenljivkama "stevilo" in "vsota". V spremenljivki "stevilo" shranjujemo trenutno število, ki ga pri vsaki ponovitvi zanke povečamo za 1 in ga prištejemo k spremenljivki "vsota":

```
static void Main(string[] args)
        {
            Console.Write("Vnesi število: ");
            int n = int.Parse(Console.ReadLine());
            int stevilo = 1;
            int vsota = 1;
            Console.Write("Vsota 1");
            while (stevilo < n)
            {
                stevilo++;
                vsota += stevilo;
                Console.Write(" + " + stevilo);
            }
            Console.WriteLine(" = " + vsota);
            Console.ReadKey();
        }
```



Slika 20: Izpis programa pri primeru 21

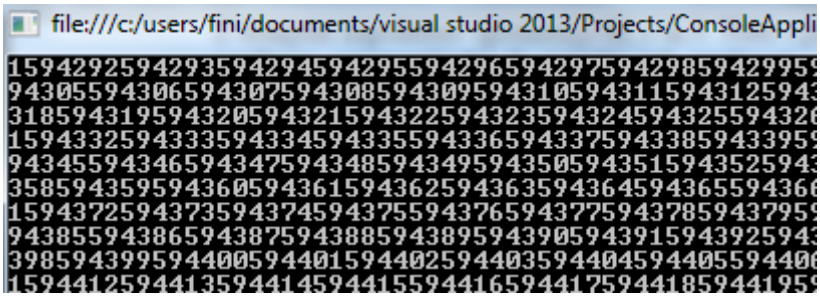
Vir: Lasten (2015)

Neskončna zanka

Pri zankah, še posebno pri zanki while, je potrebno paziti, da je zanka končna, torej da bo pogoj enkrat dosegel vrednost false. Ob nepravilnem definiranju pogoja zanke namreč lahko pride do *neskončnega števila ponovitev* zanke, kar pomeni, da se program nikoli ne bo ustavil.

PRIMER 22: Program z neskončno zanko:

```
static void Main(string[] args)
        {
            int n = 1;
            while (n >= 1)
            {
                Console.Write(n);
                n++;
            }
            Console.ReadKey();
        }
```



Slika 21: Izpis neskončne zanke pri primeru 22

Vir: Lasten (2015)

Stavka break in continue

Operator *break* se uporablja, ko želimo zanko končati predčasno. Stavek *continue* pa sproži ponovno preverjanje pogoja. Primer uporabe je moč videti v primeru 28.

2.8.2 For

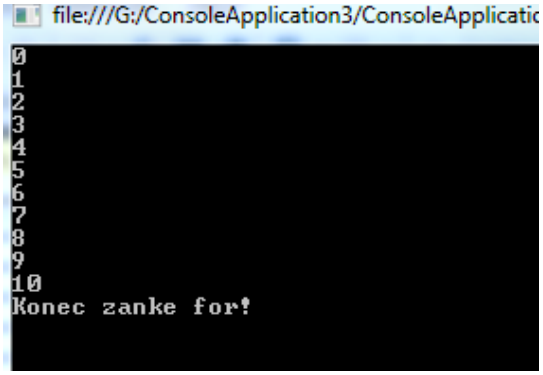
Zanka for je malo zapletenejša od zanke while, vendar pa lahko z njeno pomočjo zapišemo program z manj kode. Zanka for se običajno uporablja, kadar vemo kolikokrat se bo neka zanka ponovila.

for (stavekPredZanko; pogoj; stavekPoKoraku)
{
stavek$_1$;
stavek$_2$;
…
stavek$_n$;
}

Zanka najprej izvede stavekPredZanko, nato preveri pogoj in če je pogoj izpolnjen, izvede stavek$_1$, stavek$_2$, …, stavek$_n$ in na koncu ponovitve zanke še stavekPoKoraku. Potem ponovno preveri pogoj ter če je izpolnjen, ponovno ponovi zgoraj opisan postopek. Zanka se konča takoj, ko pogoj ni izpolnjen.

PRIMER 23: Program, ki izpiše števila od 0 do 10. Vsako število izpiše v svoji vrstici, na koncu pa izpiše še "Konec zanke for! ":

```
static void Main(string[] args)
        {
            for (int n = 0; n <= 10; n++)
            {
                Console.WriteLine(n);
            }
            Console.WriteLine("Konec zanke for!");
            Console.ReadKey();
        }
```

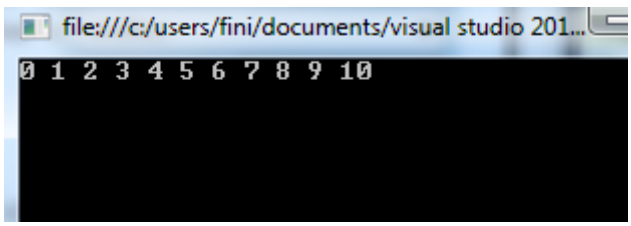


Slika 22: Izpis programa pri primeru 23

Vir: Lasten (2015)

PRIMER 24: Program, ki izpiše števila od 0 do 10:

```
class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.Write(i + " ");
            }
            Console.ReadLine();
        }
    }
```



Slika 23: Izpis programa pri primeru 24

Vir: Lasten (2015)

2.9 Tabele, vektorji in matrike

Tabela je razpredelnica, v kateri se nahajajo podatki, ki jih imenujemo *elementi*. Vsak element je oštevilčen s številkami 0, 1, 2, …, *n*-1, ki jih imenujemo *indeksi*, pri čemer je *n* velikost tabele. V isti tabeli se lahko nahajajo samo podatki istega podatkovnega tipa. Tabele so lahko različnih dimenzij, pri čemer sta najpogostejši enodimenzionalna tabela, imenovana *vektor*, in dvodimenzionalna tabela, imenovana *matrika*.

2.9.1 Deklaracija tabele

Tabelo moramo *deklarirati* z določitvijo podatkovnega tipa oglatih oklepajev ([]) in izbiro imena tabele ter z besedo *new* določimo velikost.

podatkovniTip[] imeTabele = new podatkovniTip[velikost];

PRIMER 25: Program, ki vstavi števila od 0 do 4:

```
static void Main(string[] args)
        {
            Console.WriteLine();
            int[] tabela = new int[5];
            for (int i = 0; i < tabela.Length; i++)
            {
                tabela[i] = i;
            }
            Console.ReadKey();
        }
```

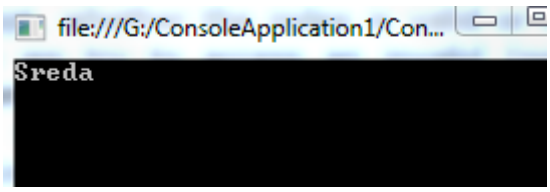
2.9.2 Izpis tabele

Posamezen element tabele izpišemo tako, da uporabimo metodo:

Console.Writeline(imeTabele[indeksElementa]);

PRIMER 26: Program, v katerega shranimo tabelo z dnevi v tednu in izpisuje tretji dan v tednu:

```
static void Main(string[] args)
        {
            string[] dneviTedna = {"Ponedeljek", "Torek", "Sreda",
            "Četrtek", "Petek", "Sobota", "Nedelja" };
            Console.WriteLine(dneviTedna[2]);
            Console.ReadKey();
        }
```



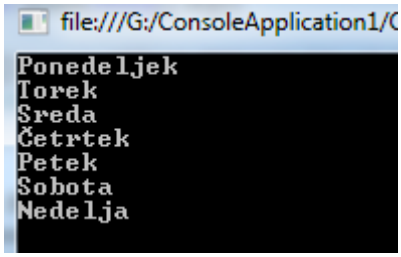
Slika 24: Izpis programa pri primeru 26

Vir: Lasten (2015)

Elemente celotne tabele izpišemo tako, da dostopamo do vsakega elementa posebej.

PRIMER 27: Program, ki izpiše vse elemente tabele:

```
static void Main(string[] args)
        {
            string[] dneviTedna = {"Ponedeljek", "Torek", "Sreda",
            "Četrtek", "Petek", "Sobota", "Nedelja" };
            for (int i = 0; i < dneviTedna.Length; i++)
                Console.WriteLine(dneviTedna[i]);
            Console.ReadKey();
        }
```



Slika 25: Izpis programa pri primeru 27

Vir: Lasten (2015)

2.9.3 Branje elementov tabele iz konzole

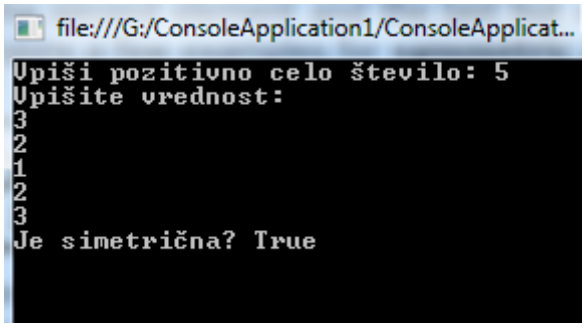
Branje podatkov preko konzole, ki jih želimo hraniti v tabelo, izvedemo tako, da podatke hranimo v spremenljivko, ki ji spremenimo podatkovni tip (iz podatkovnega tipa string v ustrezni podatkovni tip).

podatkovniTip imeSpremenljivke = PodatkovniTip.Parse(Console.ReadLine());
podatkovniTip[] imeTabele = new podatkovniTip[imeSpremenljivke];
Nato uporabimo zanko, ki ob vsaki ponovitvi prebere vpisan podatek in ga hrani v tabelo.

PRIMER 28: Program, ki od uporabnika zahteva velikosti tabele ter njune elemente, nato pa izpiše vrednost true, če je tabela simetrična, in vrednost false, če tabela ni simetrična:

```
static void Main(string[] args)
        {
            Console.Write("Vpiši pozitivno celo število: ");
            int n = int.Parse(Console.ReadLine());
            int[] tabela = new int[n];
            Console.WriteLine("Vpišite vrednost: ");

            for (int i = 0; i < n; i++)
            {
                tabela[i] = int.Parse(Console.ReadLine());
            }
            bool simetricen = true;
            for (int i = 0; i < tabela.Length / 2; i++)
            {
                if (tabela[i] != tabela [n - i - 1])
                {
                    simetricen = false;
                    break;
                }
            }
            Console.WriteLine("Je simetrična? " + simetricen);
            Console.ReadKey();
        }
```



Slika 26: Izpis programa pri primeru 28

Vir: Lasten (2015)

2.10 Vpeljava metod

Halvorsen (2014, 24–26) *navaja, da je metoda* osrednji del programa, ki rešuje konkreten primer tako, da iz podanih parametrov vrne rezultat. Gre torej za nekakšen podprogram, ki neodvisno od glavnega programa rešuje določen problem. Metode uporabljamo predvsem zaradi boljše strukturiranosti programa in boljše berljive kode ter s tem poskrbimo, da ne pride do ponavljanja kode.

2.10.1 Definicija metode

Definicija metode je sestavljena iz *deklaracije* in *telesa metode* in jo napišemo v celoti pred ali za glavno metodo Main.

Z *deklaracijo* metode na nek način registriramo metodo v programu. V deklaracijo zapišemo, ali je metoda javna (*public*), privatna (*private*) ali zaščitena (*protected*). Nato določimo vrsto metode, torej ali je metoda statična ali objektna, ter podatkovni tip rezultata metode. Sledi določitev imena metode, ki se po dogovoru vedno začne z veliko začetnico in ne sme vsebovati presledkov in nekaterih posebnih znakov. Za imenom metode sledita *argument metode* in *podatkovni tip argumenta* zapisana v oklepaju. V primeru več argumentov le-te ločimo z vejicami.

PRIMER 29: Deklaracija metode Min. Metoda Min je javna (public), vrača rezultat v obliki celega števila (tip int) in sprejme dva argumenta, ki sta prav tako celi števili (tip int), prvi argument se imenuje a, drugi pa b:

```
public int Min(int a, int b);
```

Telo metode se nahaja znotraj zavitih oklepajev ({}), v katere vpišemo stavke, katere želimo, da metoda izvede.

PRIMER 30: Metoda Primerjava je statična (static), vrača rezultat v obliki celega števila (tip int) in prejme dve celi števili (tip int), ju primerja ter če je prvo število večje od drugega, vrne število 1, če sta števili enaki, vrne število 0 in če je prvo število manjše od drugega števila, vrne število -1.

```
static int Primerjava(int stevilo1, int stevilo2)
        {
            if (stevilo1 > stavilo2);
            {
                return 1;
            }
            if (stevilo1 == stavilo2);
            {
                return 0;
            }
            else
            {
                return -1;
            }
        }
```

V primeru, da metoda ne vrne nobenega rezultata, pač pa le opravi neko delo, kot podatkovni tip rezultata navedemo *void*. V tem primeru metoda ne vsebuje stavka return.

PRIMER 31: Metoda Izpis je statična (static) metoda in tipa void, saj ne vrne nobenega rezultata, pač pa v konzolo izpiše "Programski jezik C#":

```
static void Izpis()
        {
            Console.WriteLine("Programski jezik C#");
        }
```

2.11 Vprašanja in naloge z namigi in rešitvami

1. Opišite razliko med metodama Console.WriteLine() in Console.Write().

Namig: Pomagajte si z razlago v tem poglavju.

2. Spremenite kodo programa iz tega poglavja za izpis različnih pozdravov, na primer "Dober dan!".

Namig: Uporabite kodo preprostega programa, zapisanega v tem poglavju, in izpišite pozdrav.

3. Napišite kodo programa, ki bo v konzoli izpisal vaše ime in priimek.

Namig: Uporabite metodo Console.Write().

4. Napišite program, ki bo v konzoli izpisal števila 1, 101, 1001, in sicer vsako v svojo vrstico.

Namig: Uporabite metodo Console.WriteLine().

5. Definirajte spremenljivko, ki lahko vsebuje le cela števila.

Namig: Uporabite podatkovni tip int.

6. Definirajte spremenljivko Spol podatkovnega tipa bool.

Namig: Uporabite podatkovni tip bool.

7. Izračunajte, koliko barve potrebujete, da prepleskate sobo, dolgo 5 m, široko 4 m in visoko 2,5 m. V sobi imate tudi okno, ki meri 2 m^2. Z enim kilogramom barve lahko prebarvate 10 m^2.

Namig: Izračunamo površino vseh sten in stropa (tal ne bomo barvali, zato jih pri izračunu površine ne upoštevamo) ter odštejemo površino okna. Dobljeno površino izpišemo. Z enim kilogramom barve prepleskamo 10 m^2, zato delimo z 10 in za vsak primer prištejemo 1 l barve.

Rešitev:

```
static void Main(string[] args)
  {
      int dolzina = 5;
      int sirina = 4;
      int visina = 2;
      int okno = 2;
      int prekrivnost = 10;
      int povrsina = dolzina * sirina + visina * sirina * 2
      + visina * dolzina * 2 - okno;
      int kolicina = (povrsina / prekrivnost + 1);
      Console.WriteLine("Povrsina, ki jo je potrebno
      prebarvati: " + povrsina);
      Console.WriteLine("Potrebna kolicina barve " +
      kolicina);
      Console.ReadLine();
  }
```

8. Zapišite dve spremenljivki tipa int in sestavite program, ki bo vrednosti spremenljivk zamenjal.

Namig: Vpeljemo še tretjo spremenljivko.

Rešitev:

```
static void Main(string[] args)
 {
      int a = 5;
      int b = 7;
      int staria = a;
      a = b;
```

```
        b = staria;
        Console.WriteLine("a = " + a);
        Console.WriteLine("b = " + b);
        Console.ReadLine();
    }
```

9. Zapišite niz, ki vsebuje angleško abecedo, nato izpišite dolžino niza ter 6., 12. in 18. črko abecede.

Namig: Indeksiranje znakov v nizu se začne z 0.

Rešitev:

```
static void Main(string[] args)
    {
        string abeceda = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        int dolzinaAbecede = abeceda.Length;
        char sestaCrka = abeceda[5];
        char dvanajstaCrka = abeceda[11];
        char osemnajstaCrka = abeceda[17];
        Console.WriteLine("Dolžina angleške abecede je " +
        dolzinaAbecede);
        Console.WriteLine("Šesta črka angleške abecede je "
        + sestaCrka);
        Console.WriteLine("Dvanajsta črka angleške abecede
        je " + dvanajstaCrka);
        Console.WriteLine("Osemnajsta črka angleške abecede
        je " + osemnajstaCrka);
        Console.ReadLine();
    }
```

10. Sestavite program, ki prebere tri števila in jih sešteje.

Namig: Uporabite metodi Console.ReadLine() in Parse.

11. Sestavite program, ki prebere polmer kroga in izračuna njegov obseg.

Namig: Uporabite Math.PI.

12. Vsako podjetje ima naziv, naslov, telefonsko številko, spletno stran in direktorja. Sestavite program, ki bo prebral vse te podatke in jih izpisal v konzolo.

Namig: Uporabite metodo Console.ReadLine()

13. Sestavite program, ki bo preko konzole prebral trimestno število in ga izpisal v obrnjenem vrstnem redu.

Namig: Pazite, da prebrani podatek pretvorite v ustrezni podatkovni tip.

Rešitev:

```
static void Main(string[] args)
        {
            Console.Write("Vnesi trimestno celo število:
            ");
            int stevilo = int.Parse(Console.ReadLine());
            int enice = stevilo % 10; ;
            int dvomestnoSt = stevilo / 10;
            int desetice = dvomestnoSt % 10;
            int stotice = dvomestnoSt / 10;
            Console.WriteLine("Število: " + stevilo);
            Console.WriteLine("Obrnjeno število: " + enice
            + desetice + stotice);
            Console.ReadKey();
}
```

14. Sestavite program, ki iz konzole prebere dve števili in zamenja njuni vrednosti v primeru, da je prvo število večje od drugega.

Namig: Uporabite pogojni stavek if.

15. Sestavite program, ki prebere tri števila in izpiše največjo število.

Namig: Uporabite gnezdenje if stavka. Program naj najprej preveri, katero od prvih dveh števil je večje ter le-to število primerja s tretjim številom.

16. Sestavite program, ki uredi tri števila po velikosti.

Namig: Program najprej poišče najmanjše od treh števil in ga zamenja s prvo številko. Potem preverite, če je druga številka večja kot tretja in če je, naj tudi njiju zamenja.

17. Zapišite program, ki iz konzole prebere koeficiente a, b in c kvadratne enačbe: $ax^2 + bc + c$ in izračuna ter izpiše njene realne ničle.

Namig: Kvadratna enačba lahko ima 0, 1 ali 2 ničli. Da izračunamo realne ničle, moramo najprej izračunati diskriminanto (D) po formuli: $D = b^2 - 4ac$. Če je diskriminanta enaka nič, bo imela kvadratna funkcija eno dvojno realno ničlo, ki jo izračunamo po formuli:

$x_{1,2} = \frac{-b}{2a}$. Če je vrednost diskriminante pozitivna, ima kvadratna funkcija dve različni realni ničli, ki ju izračunamo po enačbi: $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. Če je diskriminanta negativna, kvadratna funkcija nima nobene realne ničle.

18. Sestavite program, ki bo od uporabnika zahteval vpis števila in nato v konzolo izpisal števila od 1 do tega števila.

Namig: Uporabite zanko for.

19. Sestavite program, ki bo od uporabnika zahteval vnos števila, nato pa preveril, ali je to število deljivo s 3.

Namig: Uporabite zanko for in operator % za iskanje ostanka pri deljenju (stevilo % 3 == 0).

20. Sestavite program, ki bo od uporabnika zahteval vpis številke, od katere bo nato izračunal in izpisal fakulteto tega števila.

Namig: Uporabite zanko while.

Rešitev:

```
static void Main(string[] args)
     {
         Console.Write("Vnesi število: ");
         int n = int.Parse(Console.ReadLine());
         int fakulteta = 1;
         while (n >= 1)
         {
             fakulteta *= n;
             n--;
         }
         Console.WriteLine("n! = " + fakulteta);
         Console.ReadKey();
     }
```

21. Sestavite program, ki iz konzole prebere število n in izpiše vsoto prvin n števil Fibonaccijevega zaporedja: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 …

Namig: Fibonaccijeva števila se začnejo z 0 in 1, vsako naslednje število pa je definirano z vsoto njunih dveh predhodnikov. Program za izpis prvih n Fibonaccijevih števil lahko

sestavite s pomočjo zanke for od 1 do n, ki na vsakem koraku izračuna naslednje Fibonaccijevo število s seštevkom prejšnjih dveh števil.

22. Sestavite program, ki ustvari tabelo z 20 elementi podatkovnega tipa int in v vsak element tabele hrani številko, ki je enaka petkratniku indeksa.

Namig: Uporabite zanko for.

23. Sestavite program, ki prebere dve tabeli iz konzole, in preverite, če sta enaki.

Namig: Dve tabeli sta enaki, če sta enako dolgi in imata enake vse svoje elemente. Drugi pogoj preverite s pomočjo zanke for.

24. Sestavite program, ki najde in izpiše vsa praštevila od 1 do 100.

Namig: Praštevilo je število, ki ima samo dva delitelja, število 1 in samega sebe.

25. Definirajte metodo, ki za dano ime izpiše "Pozdravljen, <ime>".

Namig: Uporabite metodo s parametrom podatkovnega tipa string.

26. Definirajte metodo Max() z dvema parametroma podatkovnega tipa int, ki vrne večjo od dveh števil. Nato sestavite program, ki prebere tri števila iz konzole in izpiše največjo od teh števil. Pri sestavljanju programa si pomagajte z metodo Max().

Namig: Uporabite izraz: $Max(a, b, c) = Max(Max(a, b), c)$.

27. Definirajte metodo, ki ugotovi, kolikokrat se neko število pojavi v dani tabeli.

Namig: Metoda kot parameter predvideva tabelo z elementi, ki so cela števila (int[]).

28. Definirajte metodo, ki preveri, če je element iz danega indeksa tabele večji kot njegova soseda.

Namig: Prvi in zadnji element tabele bomo primerjali le z njegovim levim oziroma desnim sosedom. Ostale elemente primerjamo z levim in desnim elementom.

2.12 Povzetek

V današnjem času so postali računalniki nenadomestljivi. Uporabljamo jih za reševanje kompleksnih problemov na delovnem mestu, z njihovo pomočjo se lahko tudi zabavamo in komuniciramo. Kljub dejstvu, da so računalniki tako razširjeni, pa le malo ljudi ve, kako v resnici delujejo. V resnici niso računalniki tisti, ki so zares pomembni, pač pa programi (programska oprema), ki delujejo na njih. Za razvoj programske opreme je potrebno znanje programiranja.

V tem poglavju smo obdelali zgradbo programa, deklaracijo spremenljivk ter podatkovne tipe int, double, bool, char, string in object. Podrobneje so povzeti načini za izpis podatkov v konzoli in metode za branje podatkov iz konzole. Nadalje sta predstavljena pogojna stavka if in if-else ter zanki while in for. Na koncu poglavja smo se dotaknili še tabel in metod v programskem jeziku. S tem smo zajeli bistvene elemente programskega jezika in tako osvojili osnove programskega jezika C#.

3 ZAKLJUČEK

Programski jezik C# je eden izmed najpogosteje uporabljenih programskih jezikov, zato so v e-knjigi predstavljene osnove le-tega. V vsakem izmed poglavji je na kratko predstavljena osnovna teorija izbrane tematike, ki je v nadaljevanju podkrepljena z različnimi praktičnimi primeri. Na koncu vsakega poglavja so zbrane naloge z rešitvami in namigi za pomoč pri samem reševanju. Predstavljene tematike so bile izbrane na osnovi mojih večletnih izkušenj omenjenega področja. V osnovi je knjiga namenjena vsem začetnikom in prav tako vsem tistim, ki bi želeli spoznati osnove programskega jezika C# oz. ponoviti osnove le-tega. Za učinkovito razumevanje obravnavanega programskega jezika je priporočljivo rešiti čim večje število nalog znotraj vsakega poglavja. Potrebno se je zavedati, da se programiranja ne učimo samo z branjem teorije, temveč z njegovo uporabo, ki jo lahko osvojimo le tako, da rešujemo raznolike naloge.

4 LITERATURA IN VIRI

C# Practical Learning 3.0, (online). 2008. (citirano 30. 5. 2015). Dostopno na naslovu: http://www.functionx.com/csharp/.

C# Station, (online). 2008 (citirano 30. 5. 2015). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx.

CSharp Tutorial, (online). 2008. (citirano 30. 5. 2015). Dostopno na naslovu: http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm.

Halvorsen, H. *Introduction to Visual Studio and C#.* Porsgrunn, 2014.

Jerše, G., in Lokar, M. *Programiranje II, Objektno programiranje*. Ljubljana: B2, 2008.

Lokar, M. *Osnove programiranja: programiranje – zakaj in vsaj kaj*. Ljubljana: Zavod za šolstvo, 2005.

Lokar, M. in Uranič, L. *Programiranje 1*. Ljubljana: Zavod IRC, 2009.

Mayo, J. *C# Station Tutorial*, (online). 2008 (citirano 30. 5. 2015). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx.

Murach, J., in Murach, M. *Murach's C# 2008*. Mike Murach @ Associates Inc. London, 2008.

Nakov, S. & Co. *Fundamentals of computer programming with C#*, Trnovo: Faber, 2013.

Sharp, J. *Microsoft Visual C# 2013 Step by Step*. Washington: Microsoft Press, 2013.