MICROSOFT

C# .NET

Srečo Uranič

Kaj je C# in .NET Framework

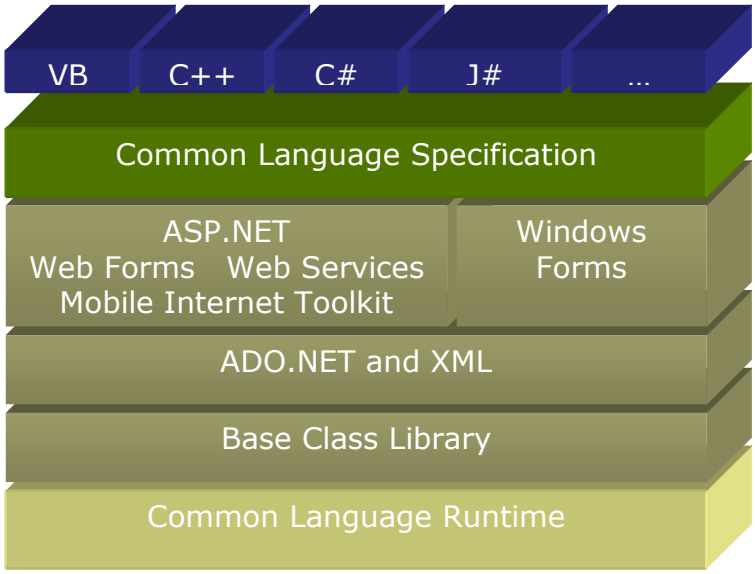
C# je Microsoftov objektno orientiran programski jezik. Jezik je nekakšna kombinacija programske moči jezika C++ z dodanimi dobrotami iz Visual Basica in Jave. Čeprav C# temelji na C++, pa vsebuje številne sestavine značilne za Javo. Razvijalcem Visual C# omogoča razvoj visoko prenosljivih aplikacij.

C# je bil oblikovan za delo z Microsoftovo .NET platformo (.NET Framework). .NET Framework je platforma/knjižnica, ki predstavlja ogrodje za vse .NET orientirana programska orodja in aplikacije za osebne računalnike, dlančnike, pametne telefone, razne vgrajene sisteme,…) Vsebuje pester nabor jezikov (C++, C#, Visual Basic, VBScript, J#, JScript…), omogoča hiter in predvsem lažji razvoj kot tudi izvajanje tako obsežnih in zahtevnih, kot tudi majhnih in enostavnih projektov. Omogoča tudi kreiranje projektov, sestavljenih iz posameznih modulov, zgrajenih z različnimi programskimi jeziki. Obenem omogoča kar najbolj možno prenosljivost programske kode. Vsebuje tudi kar se da optimizirano vgrajeno kodo, standardizirano z najnovejšo tehnologijo kot npr. XML in SOAP.

Razvojno okolje VISUAL Studio .NET omogoča razvijalcem izbor enega od štirih orodij (programskih jezikov) za razvoj tako konzolnih kot tudi vizuelnih aplikacij: **Visual C#**, **Visual C++**, **Visual Basic** in **Visual J#**. Pri instalaciji se lahko odločimo, da instaliramo vse programske jezike, ali pa izberemo (odkljukamo) le tistega, ki ga bomo uporabljali, oz. s katerim bomo razvijal naše aplikacije.

Razvojno okolje **VISUAL Studio** nastopa v več različicah, odvisno od namena uporabe: najbolj znane so različice **Express Edition**, **Standard Edition** in **Professonal Edition**. Verzija **Express Edition** je brezplačna, naložimo pa si jo lahko kar preko spleta. Razlaga in vaje v tej literaturi bodo temeljili večinoma na različici **Express Edition**, nekatere pa tudi na **Standard Edition**.

Zgradba .NET



Izdelava konzolnih aplikacij v VISUAL C# . NET

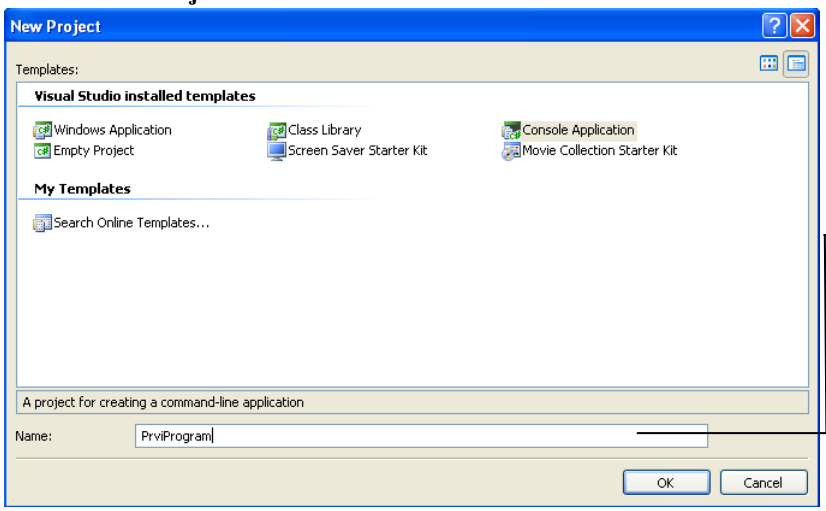
Začetek programiranja v okolju Visual C# - prva konzolna aplikacija

Pri kreiranju novega projekta je majhna razlika glede na to, katero različico razvojnega okolja uporabljamo. Kasneje razlik skoraj ni, še posebej ne pri razvoju konzolnih aplikacij.

Začenjanje novega projekta v različici Express Edition

Za začetek novega projekta v okolju **Visual C# Express Edition** odprimo meni

File -> New Project…



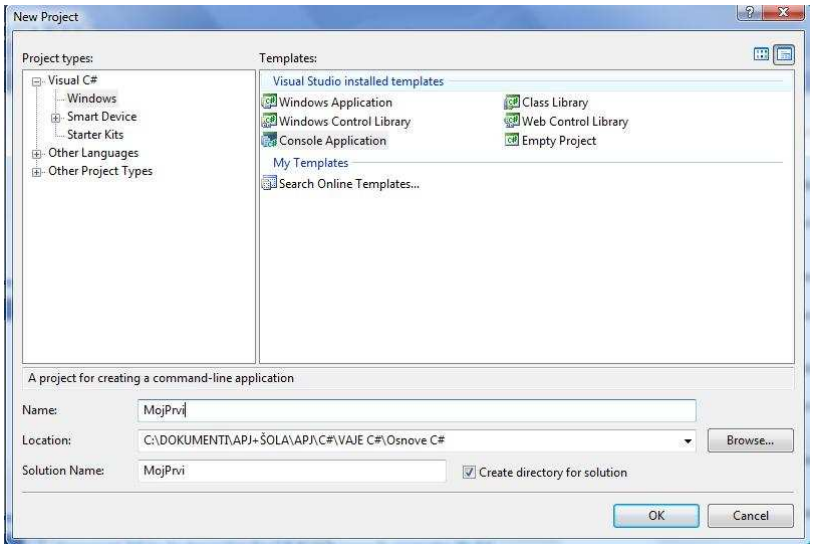
Odpre se novo okno **New Project** v katerem pod **Templates** (Vzorci) izberemo ikono **Console Application**. Na dnu okna napišimo še ime naše prve konzolne aplikacije, npr. *Prvi Program*.

S klikom na gumb OK se ustvari **nov projekt**.

Začenjanje novega projekta v različici Standard Edition

Za začetek novega projekta v okolju **Visual C# Standard Edtion** odprimo meni

File -> New Project…



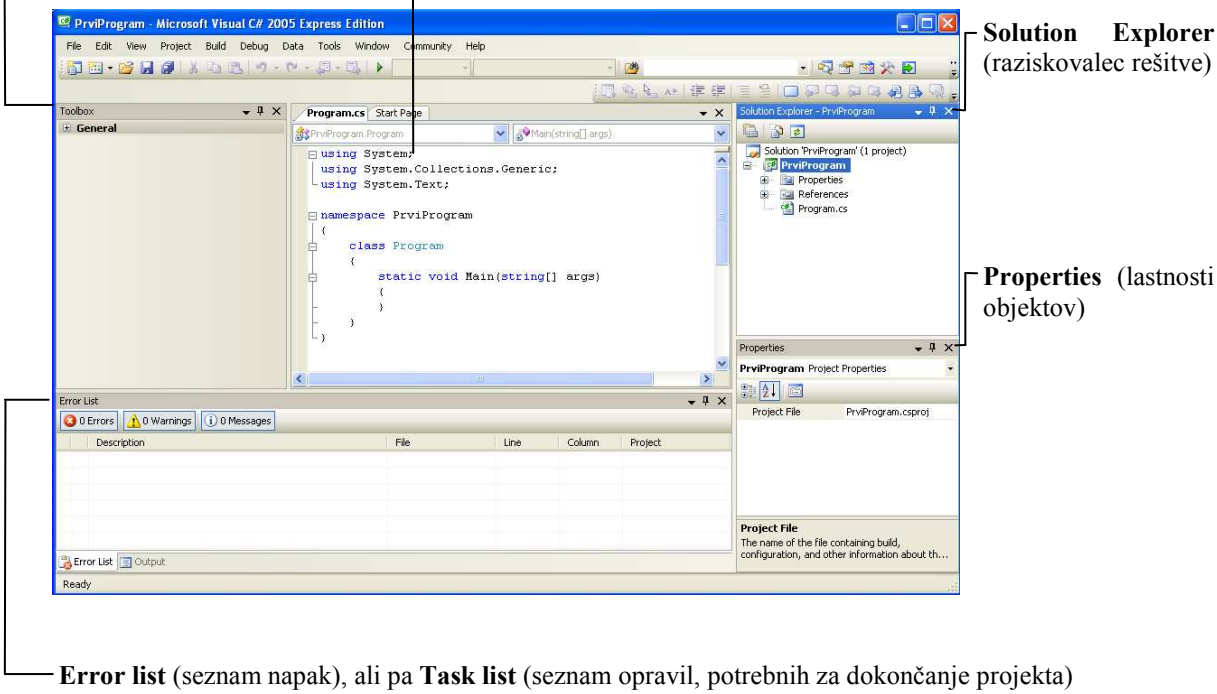
Odpre se novo okno **New Project** v katerem pod **Templates** (Vzorci) izberemo ikono **Console Application**. Na dnu okna napišimo še ime naše prve konzolne aplikacije (**Name**), npr. *Moj Prvi*, določimo mapo v kateri bo shranjen naš projekt (**Location**) in še zapišemo ime naše rešitve (**Solution Name** - priporočljivo je, da sta vsaj v začetnih projektih imeni programa in rešitve enaka).

S klikom na gumb OK se

ustvari **nov projekt**.

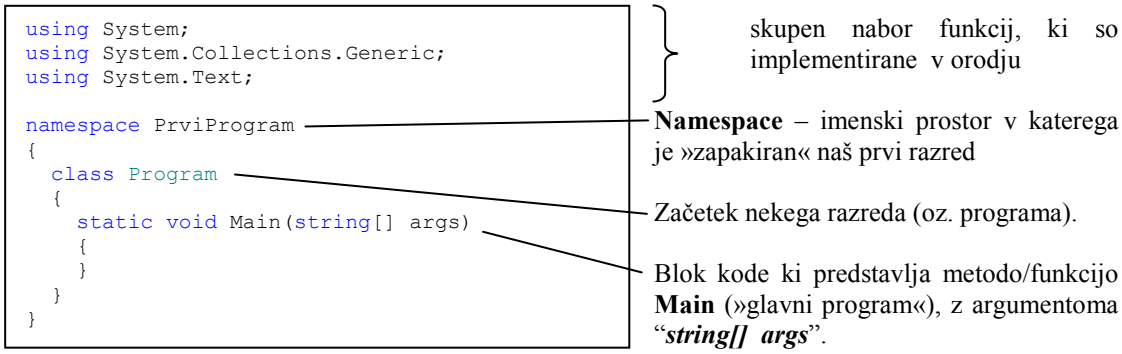
Pisanje programske kode z uporabo IntelliSense tehnologije

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



Kratek opis osnovnih oken razvojnega okolja

- **Solution Explorer** (raziskovalec rešitve): V obliki drevesne strukture nam prikazuje in omogoča dostop do vseh datotek znotraj našega projekta. Datoteke lahko dodajamo, jih spreminjamo in brišemo. Okno je dostopno tudi preko tipkovnice, s kombinacijo tipk **Ctrl+Alt+L**. Na dnu tega okna sta dva jezička: **Solution Explorer** in **ClassView**. Okno **Class View** prikazuje hierarhijo razredov znotraj rešitve. Dostopno je tudi preko tipkovnice, s kombinacijo tipk **Ctrl+Alt+L.** Okno je bolj uporabno kot **Solution Explorer** predvsem pri velikih projektih.
- **Toolbox** (objekti za obrazce): Vsebuje objekte (kontrole) za izdelavo **Windows Forms** (obrazci) ali **Web Forms** aplikacije. Pri izdelave konzolne aplikacije je seveda okno prazno, saj v tem primeru ne delamo z že pripravljenimi objekti. Okno je dostopno tudi preko tipkovnice, s kombinacijo tipk **Ctrl+Alt+X**.
- **Properties** (lastnosti objektov): V tem oknu spreminjamo in nastavljamo lastnosti in dogodke objektom pri izdelavi **Windows Forms** ali **Web Forms** aplikacije. Pri izdelave konzolne aplikacije je seveda okno prazno, saj v tem primeru ne delamo z že pripravljenimi objekti. Okno je dostopno tudi preko tipkovnice, s pomočjo tipke **F4**.
- **Error list** (seznam napak), ali pa **Task list** (seznam potrebnih opravil, ki so potrebna za dokončanje projekta). Vrsta prikazanega okna je odvisna od izbire v meniju **View** (lahko je **Error List**, **TaskList**, Output, …).
- **Editor** (urejevalnik -okno za pisanje/urejanje kode): Okno je namenjeno za pisanje ter urejanje programske kode. Omogoča tudi skrivanje in prikazovanje delov kode (**outlining**), ki smiselno tvorijo neko celoto (+ in – na levi strani urejevalnika). V primeru da je del kode skrit (da levi strani znak -) lahko (ne da bi skriti del odprli s klikom na znak -) vidimo vsebino tudi tako, da se z miško premaknemo čez povzetek skritega dela besedila na desni strani vrstice.



- Opomba: **C#** je **case sensitive** jezik, zato mora biti metoda **Main** napisana z veliko začetnico, sicer bo prevajalnik javil napako.

V **Visual C#** lahko v fazi načrtovanja (razvoja) programa ustvarjamo t.i. **regije** (začetek regije **#region** konec **#endregion**). Del kode, zapisane med ti dve besedici lahko kadarkoli skrijemo in zopet prikažemo, s tem pa pridobimo na preglednosti izvornega programa.

Ko smo projekt ustvarili, nam je **C#** ustvaril datoteko **Program.cs**, ki definira razred imenovan **Program**, le-ta pa vsebuje metodo imenovano **Main**. **Vse metode našega programa morajo biti definirane znotraj razreda imenovanega Program.** Metoda **Main** je posebna, ker je vstopna točka programa, zato **mora biti statična** (besedica **static mora** biti zapisana pred imenom metode). Kaj pa pravzaprav pomeni, da je neka metoda statična, bomo spoznali precej kasneje, pri poglavju o razredih.

Programsko kodo naše prve konzolne aplikacije bomo napisali v metodo **Main**, zgleda pa takole:

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

Console je **razred**, ki je definiran v imenskem prostoru (knjižnici) **System**. Pojem **imenski prostor** je razložen v zadnjem delu tega poglavja, na tem mestu pa moramo napisati nekaj o pojmu **razred**. Pojem **razred** in z njim povezan pojem **objekt**, sta pravzaprav osnova objektnega programiranja. Poenostavljeno rečeno je razred neka kompleksna podatkovna struktura, ki vsebuje množico lastnosti in metod, ki operirajo nad temi lastnostmi oz. opravljajo kako drugo nalogo. Razred je abstrakten pojem, v programih pa jih uporabljamo tako, da iz njih izpeljemo objekte, to je predstavnike razredov. O tem, kako napišemo svoj razred, kako mu določimo lastnosti in metode, ter še veliko drugega kar se tiče razredov, bomo spoznali veliko kasneje. Zaenkrat pa bomo razrede obravnavali tako, da bomo spoznali njihova imena, kako naredimo predstavnika (primerek, instanco) tega razreda (oz. objekt), čemu je kakšen razred namenjen in kako preko objektov uporabljamo njegove lastnosti in metode. Zapomnimo si le to, da do lastnosti in metod razredov (oz. lastnosti in metod objektov) pridemo tako, da za imenom razreda (objekta) zapišemo piko, temu pa sledi ime lastnosti oz. metode.

Ker smo knjižnico **Console** navedli na začetku naše prve konzolne aplikacije (stavek **using System**), se lahko na ta razred sklicujemo samo z imenom. V primeru, da pa na začetku programa ne bi bilo vrstice **using System**, bi do razreda prišli preko knjižnice **System** (torej **System.Console…).**

Razred **Console** je prvi izmed množice razredov, ki jih bomo uporabljali. Zaenkrat moramo o njem vedeti le to, da ta razred vsebuje metode, ki lahko berejo posamezne znake ali pa zaporedja znakov (stavke oz. vrstice) preko konzole (najpogosteje tipkovnice). Vsebuje tudi metode za pretvarjanje podatkov (npr. števila v zaporedja znakov-**stringe** in obratno), metode za formatiranje spremenljivk, ter za formatiran izpis npr. na ekran ali pa v datoteko.

Ko za besedo **Console** napišemo piko, se nam odpre t.i. **IntelliSense** meni (kontekstno občutljiva pomoč). Le-ta nam prikaže imena vseh članov in metod (**atributov**) razreda **Console** (v splošnem pa imena razreda, ki smo ga napisali pred znakom pika). Na levi strani vsakega atributa je ikona, ki označuje tip določenega atributa.

Ikona	Pomen
	method – metoda
	property – lastnost
	class – razred
	struct – struktura
	enum - naštevanje
	interface - vmesnik
	namespace – imenski prostor
	event - dogodek
	delagate - delegat

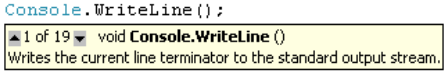
V **IntelliSense** meniju poiščimo metodo **WriteLine** in pritisnimo <**Enter**> ali dvakrat kliknimo z miško na metodo - metoda je s tem dodana programski kodi. Za tem napišimo oklepaj in zaklepaj, znotraj oklepaja pa v narekovajih napišimo besedilo, ki nam ga bo program izpisal na zaslon - v našem primeru "Pozdravljen svet!" . Za oklepajem dodajmo še podpičje, saj s tem zaključimo ta ukaz (dvopičje je znak za konec stavka).

- Opomba: Kadar je na vrhu okna (zavihka oz. okna urejevalnika) v katerem je naša koda znak 

zvezdica (*), to pomeni, da je bila v naši datoteki narejena vsaj ena sprememba, potem, ko smo jo nazadnje shranili. Ročno shranjevanje ni potrebno, saj se shranjevanje izvede avtomatsko takoj ko poženemo ukaz za prevajanje. Seveda pa je ročno shranjevanje priporočljivo takrat, kadar smo zapisali veliko vrstic kode, pa prevajanja še ne želimo pognati.

V splošnem nam **IntelliSense** nudi več vrst pomoči

- **List Members** : kombinacija tipk **Ctrl-J** nam prikaže seznam vseh članov (**members**) nekega atributa, nad katerim se nahaja kazalnik miške.
- **Parameter info** (informacije o parametrih neke metode): z miško kliknimo med oklepaja poljubne metode, in uporabimo kombinacijo tipk **Ctrl-Shift-Space**. V okvirju (seveda če le ta za to metodo sploh obstaja) se prikažejo vse možne variante izbrane metode. V levem zgornjem kotu imamo podatek o tem, koliko različnih vrst te metode obstaja (**preobložene metode!!!**). S klikom na trikotnika se lahko sprehodimo po teh metodah, obenem pa so prikazani podatki o številu in tipu parametrov, v spodnjem delu okna pa je še razlaga metode.



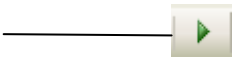
- **Quick Info** (hitra pomoč): S kombinacijo tipk **Ctrl-I** se nam namreč ob kazalniku miške prikaže sličica daljnogleda; če sedaj miško zapeljemo čez poljuben identifikator v urejevalniškem oknu, se v majhnem okencu prikazujejo osnovne informacije o tem identifikatorju.
- **Complete Word** (dokončaj besedo): S kombinacijo tipk **Ctrl-Space** dosežemo, da se neka beseda, ki jo pišemo, avtomatsko izpiše do konca, v primeru, da pa je možnih besed več, pa nam **C#** v okvirčku, v obliki **PopUp** menija, ponudi seznam vseh možnih besed (to je seznam vseh možnih metod, dogodkov, lastnosti, …). Zraven vsakega od njih je ustrezna ikona, ki označuje za kakšno besedo gre (ali je to metoda, lastnost, dogodek,…). Pomen ikon je pojasnjen v posebni tabeli na prejšnji strani.

Prevajanje in zagon projekta

Naš prvi projekt je sedaj pripravljen za prevajanje. Na voljo imamo več možnosti

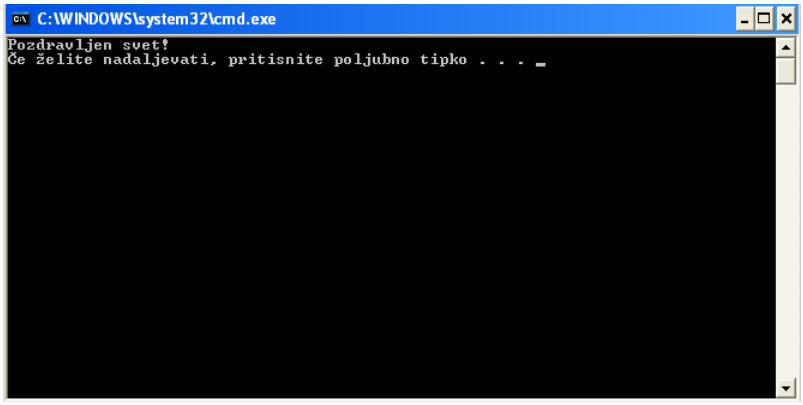
- Če želimo naš projekt le prevesti, ne pa tudi zagnati, potem v meniju **Build** kliknemo opcijo **Build Solution**. V primeru, da smo naredilo kako napako, nam **C#** v oknu **ErrorList** napiše vse potrebne informacije o številu in vrsti napak, pa tudi natančno informacijo o vrstici in stolpcu kjer je napaka;

- Prevajanje in obenem zagon projekta poženemo iz menija **Debug**, opcija **Start Debugging** (tipka **F5**) ali pa **Start Without Debugging** (**Ctrl-F5**). Razlika med načinoma je v tem, da se bo v drugem primeru okno, v katerem se bo izvedla naša aplikacija takoj zaprlo in si eventuelnih rezultatov(v našem primeri izpisa na zaslonu) ne bomo mogli ogledati;

- Kliknimo gumb **Start Debugging** v orodjarni 

- O ostalih opcijah (**Rebuild Solution**, **Step info**, **Step Over**) bo več zapisanega v nadaljevanju.

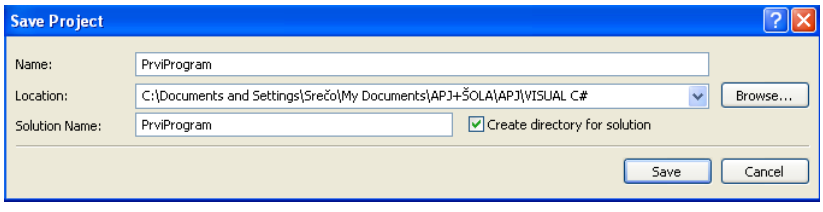
Naš program poženemo torej z opcijo **Ctrl-F5** (oziroma v meniju **Debug** kliknimo **Start Without Debugging**). Če je program brez napak, se odpre novo okno, v katerem steče naš program in se v njem prikažejo rezultati našega programa. Konzolno okno je privzeto črno, napis na oknu prikazuje mapo in ime programa, barva znakov v oknu pa je bela.



Prepričajmo se, da je prikazano okno aktivno (oz. da ima **focus**) in pritisnimo poljubno tipko. Okno se zapre in vrnemo se v okolje **Visual C#**.

Naš projekt se je ob prvem prevajanju tudi v celoti shranil, vendar POZOR! Običajno želimo projekt shraniti v točno določeno mapo, zaradi česar običajno projekt pred prvim zagonom shranimo tja, kamor želimo sami in ne tja kamor so nastavljene privzete nastavitve. To storimo tako, da pred prvim prevajanjem v meniju **File** kliknemo na ikono **Save All** (ali pa kar na ikono **Save All** v orodjarni).

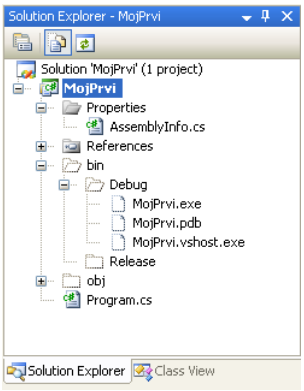
Če uporabljamo različico **Express Edition**, se prikaže okno za nastavitev shranjevanja, v katerem nastavimo (zapišemo) vse potrebne podatke o tem, kam in pod kakšnim imenom želimo naš projekt shraniti:



Ime našega programa lahko sedaj poljubno spremenimo (vrstica **Name**), izberemo mapo v kateri želimo imeti naš projekt (**Location**). Če mapa še ne obstaja, jo lahko ob kliku na gumb **Browse** skreiramo. V vrstici **Solution Name** imamo možnost, da se ime programa razlikuje od same rešitve, a priporočljivo je, da sta ime programa in konkretna rešitev poimenovana enako. Dodana je še možnost kreiranja novega imenika za naš program (znotraj izbranega imenika/mape v vrstici **Location**). Priporočljivo pa je, da za vsak nov projekt okolje **Visual C#** samo kreira svoj imenik oz. mapo, zato da imamo vse potrebne datoteke, ki tvorijo projekt, v svoji mapi.

Pri shranjevanju v različici Standard zgornjega okna ni, saj smo vse potrebne nastavitve naredili že pri odpiranju projekta.

Preglejmo še katere datoteke so nastale ob kreiranju naše prve konzolne aplikacije. V **Solution Explorerju** kliknimo ikono **Show All Files**. Po imenom mape našega projekta (v našem primeru **MojPrvi**) se prikažejo še mape **Properties**, **bin** in **obj**. Te mape se kreirajo takoj, ko poženemo prevajanje programa in vsebujejo izvršilno (**executable**) verzijo našega programa, ter še nekaj datotek, ki so potrebne za razhroščevanje (**debugging**) programa. Če v **Solution Explorerju** kliknimo znak + pred mapo **bin**, se prikaže še vsebovana podmapa **Debug**. Kliknimo še na + pred mapo **Debug** in prikažeta se datoteki **MojPrvi.exe** in **MojPrvi.pdb**.



- Opomba: Vsak izvorni program (**source code**) lahko prevedemo v izvršilno verzijo (**executable code**) tudi neposredno iz komandne vrstice (konzole) z uporabo **C# csc** prevajalnika.

Samo zaradi tega, da se navadimo na to, da imajo razredi v splošnem veliko lastnosti in metod, ki so nam takoj na voljo in jih torej lahko uporabimo v naših aplikacijah, je tule tabela **nekaterih** lastnosti in metod razreda **Console**, ki jih lahko preizkusimo v naslednji vaji.

Lastnost	**R a z l a g a**
BackgroundColor	Pridobivanje oz. nastavljanje barve ozadja konzole.
ForegroundColor	Pridobivanje oz. nastavljanje barve znakov (fontov).
Title	Besedilo, ki je izpisano na vrhu konzolnega okna.
Metode	**R a z l a g a**
Beep	Zvočni signal.
Clear	Brisanje vsebine konzolnega okna.
Read	Branje naslednjega znaka z vhodnega toka, npr,. tipkovnice.
ReadKey	Branje znaka ob uporabnikovem pritisku tipke oz. kombinacije tipk.
ReadLine	Branje vrstice teksta (zaporedja znakov do pritiska tipke <Enter>) z vhodnega toka.
ToString	Pretvorba nekega objekta (npr celega števila) v zaporedje znakov (string).
Write	Pisanje besedila na standardni izhodni tok, npr. na ekran
WriteLine	Pisanje besedila na standardni izhodni tok, npr. na ekran, ki mu sledi še oznaka za konec vrstice.

Nekaj primerov uporabe lastnosti in metod razreda **Console**:

```
//barva ozadja bo bela
Console.BackgroundColor = ConsoleColor.White;
//celo okno bo belo
Console.Clear();
//barva pisave bo črna
Console.ForegroundColor = ConsoleColor.Black;
//napis na oknu
Console.Title = "POZDRAVNO OKNO";
//zvočni signal, s frekvenco 1000 HZ, ki traja 500 milisekund
Console.Beep(1000,500);
```

```
//Pozdravno sporočilo
Console.Write("Dober dan!\n\nVnesi svoje ime in priimek: ");
//zahtevamo vnos podatkov preko tipkovnice. Vnos zaključimo z Enter
string stavek=Console.ReadLine();
Console.WriteLine("\n\n\nPozdravljen " + stavek+"!\n\n\n\n");
```

Še razlaga metode **Console.Readkey()**:

```
Console.Write("Pritisni poljubno tipko ali kombinacijo tipk: ");
//Funkcija ReadKey vrne vrednost tipa ConsoleKeyInfo
ConsoleKeyInfo k;
//Preko lastnosti KeyChar spremenljivke tipa ConsoleKeyInfo dobimo oznako tipke oz. oznako
//kombinacije tipk
k = Console.ReadKey();
Console.WriteLine();
Console.WriteLine("Pritisnil si tipko "+k.KeyChar);
```

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. **Visual C#** pa ima tudi za odpravljanje logičnih oziroma pomenskih napak na voljo posebno orodje, ki pa ga bomo spoznali v drugem delu te literature. To je t.i. **razhroščevalnik** oz. **Debugger**.

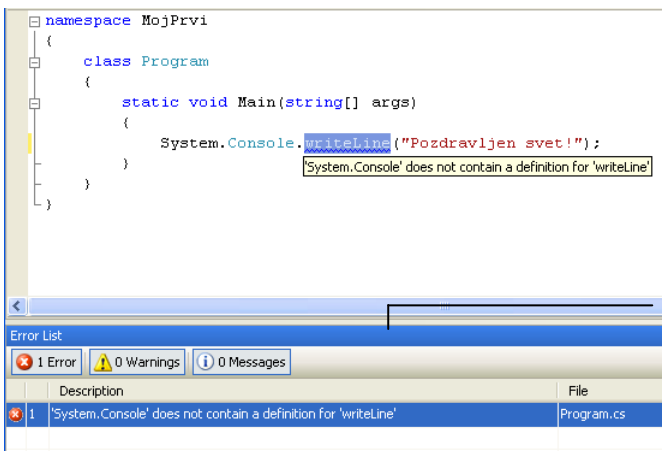
Na primeru naše prve konzolne aplikacije si poglejmo prikaz in odpravljanje prvih dveh vrst napak. Recimo, da smo se pri pisanju programske kode zmotili in namesto pravilnega zapisa

```
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.



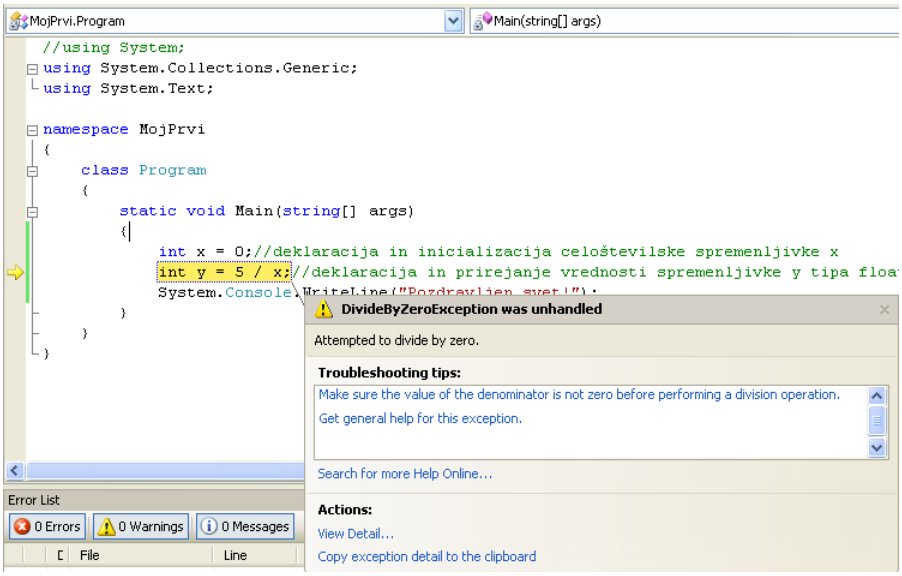
Poleg tega nam **Visual C#** podčrta del kode, kjer se napaka nahaja. Če se z miško premaknemo nad podčrtani del kode, kjer je napaka, nam **Visual C#** v okvirčku pod to besedo izpiše za kakšno napako gre.

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 (npr. z **if** stavkom v katerem preverimo vrednost delitelja).

Imenski prostori

Majhni programčki oz. programi postajajo s časoma vedno večji in pojavita se dva problema. Prvi je ta, da so večji programi težje obvladljivi in razumljivi kot pa manjši programi. Drugi problem pa je v tem, da več kode navadno pomeni več novih imen, več spremenljivk, funkcij, razredov, … S tem, ko se število teh atributov veča, pa slej ko prej naletimo na problem podvajanja imen, kar pa seveda ni dovoljeno. Program ne dela, potrebno je spreminjanje imen, a to opravilo je pri obsežnih programih lahko mukotrpno ali celo neizvedljivo. Problem podvajanja v sodobnih programskih jezikih rešujejo t.i. imenski prostori (**namespaces**), v katere »zapakiramo« našo kodo. Poskrbeti moramo le za to, da je ime našega imenskega prostora unikatno. Imenski prostor je nabor imen, spremenljivk, razredov, …, ki logično pripadajo neki celoti, v kateri velja pravilo neponovljivosti.

Če želimo npr. narediti nov razred z imenom ***MojPrviPozdrav***, je bolje da naredimo razred poimenovan ***Pozdrav*** znotraj imenskega prostora ***MojPrvi*** takole:

```
namespace MojPrvi
{
```

```
    class Pozdrav
    {
      ...
    }
}
```

Na razred ***Pozdrav*** se sedaj lahko sklicujemo kot ***MojPrvi.Pozdrav***. V primeru, da bo razred z enakim imenom kreiral še kdo drug v okviru njegovega imenskega prostora, ter ga instaliral na naš računalnik, bo naša aplikacija še vedno delala brez problemov. Priporočilo .NET platforme je, da vse razrede definiramo v imenskih prostorih, .NET razvojno okolje pa to priporočilo nadgradi s tem, da za ime projekta prevzame ime zunanjega imenskega prostora.

> ❖ Opomba: Izogibajmo se podvajanju imen imenskih prostorov in razredov. Z drugimi besedami, ne kreirajmo razredov z enakim imenom kot imenski prostor. Imenski prostor lahko vsebuje večje število razredov.

Vsak razred živi v svojem imenskem prostoru. Razred **Console** npr. živi znotraj imenskega prostora **System**. To pomeni, da je njegovo polno ime **System.Console**. Da pa nam polnega imena ni potrebno pisati vsakič znova, lahko ta problem rešimo z napovedjo

```
using System;
```

Using stavke lahko napišemo na začetku našega programa, ali pa kot prve stavke znotraj našega imenskega prostora. Z uporabo teh stavkov je poimenovanje razredov bistveno krajše, koda pa bolj pregledna. Takole bi izgledal naš prvi program, če **using** stavka ne bi uporabili (v spodnjem primeru je stavek **using System** označen kot komentar).

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

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

Komentarji

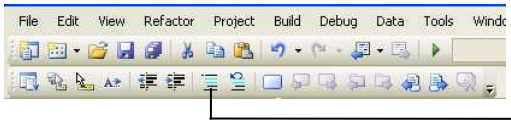
Komentarji so na poseben način označeni deli besedila, ki niso del programske kode. V komentarje zapisujemo razne opazke ali pa jih uporabljamo za lažje iskanje delov programa in za izboljšanje preglednosti programske kode. Prevajalnik komentarje ne prevaja, tako da ti ne vplivajo na velikost izvršne 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!
```

```
*/
```

Visual C# pa nam ponuja še eno zelo priročno možnost, da nek del teksta, ki je že napisan, označimo kot komentar (in ga s tem npr. začasno izključimo iz prevajanja). To nam omogoča hitri gumb **Comment out the selected lines**. Tekst, ki ga želimo zakomentirati (označiti kot komentar) najprej označimo z miško, nato pa kliknemo na hitri gumb **Comment out the selected lines**, ali pa pritisnemo kombinacijo tipk **Crtl+E, C**.



Gumb **Comment out the selected lines**

Podobno lahko delu teksta, ki je označen kot komentar, oznake za komentar umaknemo. To storimo tako, da ta del teksta najprej označimo z miško, nato pa kliknemo na hitri gumb **Uncomment the selected lines**, oziroma pritisnemo kombinacijo tipk **Crtl+E, U**.

Podatkovni tipi v C#

Podatkovni tipi, ki jih uporabljamo v C# temeljijo na skupnem sistemu tipov (**Common Type System - CTS**).

Podatkovne tipe v C# lahko delimo na

- **Vgrajene** podatkovne tipe, ki jih že poznamo. To so podatkovni tipi **int, float** in **char**;
- **Uporabniško definirane** podatkovne tipe (teh še ne poznamo), kot sta npr **struct** in **class**.

Prav tako pa lahko podatkovne tipe delimo v

- **Vrednostne** podatkovne tipe – spremenljivke teh tipov shranjujejo vrednosti;
- **Referenčne** podatkovne tipe – spremenljivke teh tipov hranijo le referenco (oz. naslov) dejanskih podatkov.

Vrednostni podatkovni tipi

Vrednostne podatkovne tipe delimo na:

- **Strukturni tipi**
  - **Numerični tipi** – Numerične tipe že poznamo in jih delimo še na tri podskupine
    - **Celoštevilčni oz. integralni podatkovni tipi**

Tip	Razpon	Razlaga
sbyte	-128 do 127	Predznačeno 8 bitno celo število.
byte	0 do 255	Nepredznačeno 8 bitno celo število.
char	U+0000 do U+ffff	Unicode 16 bitni znak.
short	-32.768 do 32.767	Predznačeno 16 bitno celo število.
ushort	0 do 65.535	Nepredznačeno 16 bitno celo število.
int	-2.147.483.648 do 2.147.483.647	Predznačeno 32 bitno celo število.
uint	0 do 4.294.967.295	Nepredznačeno 32 bitno celo število
long	-9,223.372.036,854.775.808 do 9.223.372.036.854.775.807	Predznačeno 64 bitno celo število
ulong	0 to 18,446,744,073,709,551,615	Nepredznačeno 64 bitno celo število

Nekaj primerov:

```
int i = 100;
```

```
long velikoStevilo = 1000000000;

char char1 = 'Z';        // znak
char char2 = '\x0058';   // Hexadecimalen zapis
char char3 = (char)88;   // Konverzija celega števila v znak
char char4 = '\u0058';   // Unicode
```

- **Realni podatkovni tipi**

Tip	Razpon	Natančnost
float	±1.5e−45 to ±3.4e38	7 cifer
double	±5.0e−324 to ±1.7e308	15-16 cifer

Za inicializacijo spremenljivke tipa float moramo realnemu številu dodati še pripono f ali F.

❖ POZOR: Decimalno ločilo pri inicializaciji spremenljivke tipa float (pa tudi double in decimal) je **decimalna pika**!

Nekaj primerov:

```
float x = 124.80f;
float xx=50.333F;
//cela števila se avtomatsko konvertirajo v realna, oznaka f zato ni
//potrebna
float y = 500;
double z = 3900.89;
```

- **Decimalni podatkovni tip**

Tip	Razpon	Natančnost
decimal	±1.0 × 10e−28 to ±7.9 × 10e28	28-29 cifer

Če hočemo, da bo neka numerična vrednost obravnavana kot decimalna, ji moramo dodati še oznako **m** ali **M**.

Nekaj primerov:

```
decimal znesek=450.60m; //oznaka m ali M obvezna
decimal cena=345.78M;

//cela števila se avtomatsko konvertirajo v decimalna, oznaka m zato ni
//potrebna
decimal visina = 5600;
```

- Logični tip

  Spremenljivka tipa bool lahko zavzame le dve vrednosti: **true** ali **false**.

```
bool logicna=true;
```

- Uporabniško definirani strukturni tipi

- Naštevni tipi: Naštevnih tipov še ne poznamo, spoznali ih bomo v nadaljevanju.

Najpomembnejše metode posameznih podatkovnih tipov

Tip char	Razlaga
IsDigit	Metoda vrne true, če znak predstavlja cifro
IsLetter	Metoda vrne true, če znak predstavlja črko
IsLetterOrDigit	Metoda vrne true, če znak predstavlja črko ali cifro
IsLower	Metoda vrne true, če znak predstavlja malo črko abecede
IsNumber	Metoda vrne true, če znak predstavlja cifro
IsUpper	Metoda vrne true, če znak predstavlja veliko črko abecede
Parse	Pretvarjanje stringa v znak
ToLower	Pretvarjanje črke v malo črko
ToString	Pretvarnaje znaka v string
ToUpper	Pretvarjanje črke v veliko črko

O izpisovanju besedila in podatkov na zaslon bo sicer več govora v naslednjih poglavjih. Da pa si bomo rezultate primerov, ki sledijo, lahko ogledali na zaslonu, se že sedaj naučimo uporabljati metodo **Console.WriteLine()**) za izpisovanje podatkov na konzolo - zaslon. Z metodo **WriteLine** namreč lahko izpišemo poljubno število parametrov (besedilo, spremenljivke, konstante) hkrati. Najenostavnejša uporaba te metode za izpis enega podatka (podatek je lahko poljubno besedilo, ki pa mora biti v dvojnih narekovajih, podatek je lahko poljubna spremenljivka ali pa konstanta) je takale:

```
Console.WriteLine(podatki);
```

V kolikor pa želimo hkrati izpisati več podatkov lahko metodo **WriteLine** uporabimo kar takole:

```
Console.WriteLine(podatek1+podatek2+podatek3+ …);
```

V metodi **WriteLine** torej operator '+' uporabljamo za sestavljanje izpisa. Pri tem ni prav nič važno, kakšnega podatkovnega tipa so posamezni podatki, saj metoda sama poskrbi za avtomatsko konverzijo v izpisovanju!

Nekaj primerov:

```
char ch = '8';
Console.WriteLine(Char.IsDigit(ch));          //izpis True

char ch1 = '8';
Console.WriteLine(Char.IsLetter(ch1));        //izpis False

char ch2 = 'A';
Console.WriteLine(Char.IsLetterOrDigit(ch2)); //izpis False

char ch3 = 'a';
Console.WriteLine(Char.IsLower(ch3));         //Izpis: True

Console.WriteLine(Char.IsNumber('8'));        //Izpis: True

char ch4 = 'A';
Console.WriteLine(Char.IsUpper(ch4));         //izpis True

Console.WriteLine(Char.Parse("A"));           // Izpis: 'A'

char znak = Char.ToLower('A');
Console.WriteLine(znak);  //izpis a
```

```
Console.WriteLine(Char.ToLower('A'));          //izpis 'a'

char znak1 = 'a';
Console.WriteLine(znak1.ToString());           //Izpis : "a"

char zn = Char.ToUpper('x');
Console.WriteLine(zn);                         //izpis 'X'
```

Tip int	Razlaga
Parse	Pretvarjanje stringa v celo število
ToString	Pretvarjanje celega števila v string

Nekaj primerov:

```
int cifra=int.Parse("100");
Console.WriteLine(cifra);   //izpis 100

int celo=550;
string sCelo=celo.ToString();
```

Tip float	Razlaga
Parse	Pretvarjanje stringa v celo število
ToString	Pretvarjanje celega števila v string

Nekaj primerov:

```
float stevilo= float.Parse("100,67");
Console.WriteLine(stevilo);   //izpis 100,67
float znesek = 550.98f;
string sZnesek = znesek.ToString();
Console.WriteLine(sZnesek);   //izpis 550,98
```

Tip double	Razlaga
Parse	Pretvarjanje stringa v celo število
ToString	Pretvarjanje celega števila v string

Nekaj primerov:

```
double stevilo= double.Parse("100,67");
Console.WriteLine(stevilo);   //izpis 100,67
double znesek = 550.98;
string sZnesek = znesek.ToString();
Console.WriteLine(sZnesek);   //izpis 550,98
```

Tip decimal	Razlaga
Parse	Pretvarjanje stringa v decimalno število
Round	Zaokroževanje na najbližje celo število

ToByte	Pretvarjanje decimalnega števila v 8 bitno nepredznačeno celo število
ToDouble	Pretvarjanje decimalnega števila v število tipa double
ToInt16	Pretvarjanje decimalnega števila v 16 bitno predznačeno celo število
ToInt32	Pretvarjanje decimalnega števila v 32 bitno predznačeno celo število
ToInt64	Pretvarjanje decimalnega števila v 64 bitno predznačeno celo število
ToString	Pretvarjanje decimalnega števila v string
Truncate	Pretvarjanje decimalnega števila v celo število – odrežemo decimalke

Nekaj primerov:

```
decimal decimalnoStevilo = Decimal.Parse("1234,98");
Console.WriteLine(decimalnoStevilo);              //izpis 1234,98
//POZOR
decimal Stevilo = Decimal.Parse("1234.98");
Console.WriteLine(Stevilo);                       //izpis 123498

decimal znesek = 900.67m;
decimal zaokrozeno = decimal.Round(znesek);
Console.WriteLine(zaokrozeno);// izpis 901

decimal cena = 500.45m;
cena = decimal.Truncate(cena);
Console.WriteLine(cena);// izpis 500
```

Velikost pomnilnika v bytih, ki ga zasedajo spremenljivke vrednostnih podatkovnih tipov dobimo s pomočjo rezervirane besede **sizeof**.

```
Console.WriteLine("Velikost pomnilnika v bytih za vrednostne podatkovne tipe:\n");
Console.WriteLine("byte: " + sizeof(byte));//1
Console.WriteLine("sbyte: " + sizeof(sbyte));//1

Console.WriteLine("bool: " + sizeof(bool));//2
Console.WriteLine("short: "+sizeof(short));//2
Console.WriteLine("ushort: " + sizeof(ushort));//2
Console.WriteLine("char : "+sizeof(char));//2

Console.WriteLine("int  : "+sizeof(int));//4
Console.WriteLine("uint  : " + sizeof(uint));//4
Console.WriteLine("float : " + sizeof(float));//4

Console.WriteLine("long : "+sizeof(long));//8
Console.WriteLine("ulong : " + sizeof(ulong));//8
Console.WriteLine("double : " + sizeof(double));//8
Console.WriteLine("decimal : " + sizeof(decimal));//16
```

Konverzija tipov

Osnovni tipi so lahko prosto pomešani v izrazih, saj je poskrbljeno, da se opravljajo avtomatske pretvorbe (konverzije), ki ohranjajo informacijo, če je le to mogoče. Pogosto lahko spremenimo tip iz nižjega (zasede manj bitov) v višjega, problemi pa nastopijo, če gremo v obratno smer, saj v tem primeru izgubimo informacijo.

```
byte b = 3;
Console.WriteLine("byte: " + b);//izpis:3
int st = b;
Console.WriteLine("st: " + st);//avtomatska konverzija, izpis:3
```

```
char znak = 'A';
Console.WriteLine("znak: " + znak);//izpis:A
int stevilo = znak;
Console.WriteLine("stevilo: " + stevilo);//avtomatska konverzija, izpis:65

int i = 10;
Console.WriteLine("i : " + i); //izpis:10
float x = i;//avtomatska konverzija iz int v float
Console.WriteLine("x : " + x); //izpis:10

double razdalja = x; //avtomatska konverzija iz float v double
Console.WriteLine("razdalja : " + razdalja); //izpis:10
```

Eksplicitne konverzije (casting)

Marsikdaj želimo napraviti konverzijo, ki sicer ni razvidna oz. jo prevajalnik ne bi opravil. V tem primeru govorimo o **eksplicitni** konverziji. To storimo tako, da pred spremenljivko v oklepaju povemo nov tip.

```
int i1=25;
int i2=10;

float f1 =  i1 / i2;
//ker sta i1 in i2 celi števili se operator "/" obnaša kot operator za celoštevilčno deljenje
Console.WriteLine("f1 : " + f1); //rezultat 2

//če poskrbimo za eksplicitno konverzijo int v float se operator "/" obnaša kot operator za
deljenje realnih števil
f1 = (float) i1 / (float) i2;    //eksplicitna konverzija iz int v float
Console.WriteLine("f1 : " + f1); //rezultat 2,5

double cena=1240.40;
decimal znesek = (decimal)cena; //potrebna eksplicitna konverzija double v decimal
Console.WriteLine("znesek : " + znesek);
```

Vaja:

```
//S pomočjo eksplicitne konverzije tipov lahko zamenjamo vrednosti dveh spremenljivk tipa
char, brez uporabe pomožne spremenljivke!
char st1='A';
char st2='B';
Console.WriteLine("PRED ZAMENJAVO : st1 = "+st1+",  st2 = "+st2);

//izpišimo konvertirane vrednosti še pred konverzijo:
Console.WriteLine("(int)st1 = "+(int)st1+" (int)st2 = "+(int)st2);//izpis int(st1)=65
                                                                  //int(st2)=66
st1=(char)((int)st1+(int)st2);
st2=(char)((int)st1-(int)st2);
st1=(char)((int)st1-(int)st2);
Console.WriteLine("PO ZAMENJAVI : st1 = "+st1+",  st2 = "+st2);
```

Osnovni operatorji v C#

V naslednji tabeli so zbrani osnovni operatorji jezika C#. Zanje velja enaka prioriteta kot je to v matematiki, seveda pa lahko njihov vrstni red določimo s pomočjo okroglih oklepajev (tako kot v matematiki).

Operator	Razlaga

+	seštevanje
-	odštevanje
*	množenje
/	deljenje
%	modulo (ostanek pri celoštevilskem deljenju)
++	inkrement (predimkrement ali poinkrement)
--	dekrement (preddekrement ali podekrement)
?:	pogojni operator
=	prirejanje
+=	seštevanje in prirejanje
-=	odštevanje in prirejanje
*=	množenje in prirejanje
/=	Deljenje in prirejanje

Primeri:

```
int st = 10;
int st1 = 3;
Console.WriteLine(st/st1);//celoštevilsko deljenje, rezultat je 3
Console.WriteLine(st % st1);//ostanek pri celoštevilskem deljenju, rezultat je 1

int i = 1;
i++;  //poinkrement, enakovreden zapis kot i=i+1; i torej dobi vrednost 2

int j = 1; int j1 = 10;
int k = j++ + j1++;  //poinkrement: najprej se izračuna vsota, nato se vrednost
                     //spremenljivkama j in st poveča za 1
Console.WriteLine(k);//izpis 11

int n = 1; int n1 = 10;
int m = ++n + ++n1;  //predinkrement: najprej se vrednost spremenljivkama n in n1 poveča za 1,
                     //nato se izračuna vsota
Console.WriteLine(m);//izpis 13

int stevilo = 10;
stevilo += 5;//enakovreden zapis kot stevilo=stevilo+5;
Console.WriteLine(stevilo);//izpis 15
```

Naloge:

- Napiši program ki na osnovi danih robov kvadra izračuna in izpiše njegovo površino in prostornino.
- Dani sta poljubni celi števili *stevilo1* in *stevilo2*. Ugotovi in izpiši rezultat celoštevilskega deljenja teh dveh števil, ostanek pri celoštevilskem deljenju, rezultat pravega deljenja teh dveh števil, ter celo število, ki ga dobiš, če rezultatu pravega deljenja teh dveh števil odrežeš vse decimalke.
- Dane so deklaracije:

```
int n = 5;
int m = 10;
int k = 0;
```

```
int i = 7;
```

- Kakšno vrednost dobijo spremenljivke *stevilo1*, *stevilo2*, *stevilo3* in *stevilo4* po izvedbi naslednjih stavkov:

```
int stevilo1 = 2 * n++ + jm++;
int stevilo2 = --m + ++k - ++n;

k += 5;
n *= 2;
m /= 3;
i %= 4;

int stevilo3 = n++ -2 / k++ - m + i++;
int stevilo4 = 2 * (n - ++m) + 2 * (i + --j);
```

- Dano je poljubno celo število st. Ugotovi in izpiši njegove enice.

- Dane so deklaracije:

```
double x,y = 2;
int z = 3;
int a = 10;
bool c= true;
```

- Kateri od izrazov NI pravilen?

```
x= y + z;
z= x - y;
c= true;
int b= z = a;
```

- Dane so deklaracije – podatki o začetnem in končnem času nekega tekmovanja. Ugotovi porabljeni čas!

```
int uraStart = 5;
int minutaStart = 45;
int sekundaStart = 33;

int uraCilj = 18;
int minutaCilj = 1;
int sekundaCilj = 11;
```

- Dana sta kota α in β nekega trikotnika, podana v stopinjah, minutah in sekundah. Izračunaj tretji kot (γ), če veš, da vsi trije koti skupaj merijo 180 stopinj!

- Dana je celoštevilska spremenljivka *stirimestno*, njena vrednost pa je neko celo štirimestno število, npr.: `int stirimestno = 1234;` Zamenjaj prvo in zadnjo cifro, za drugo in tretjo cifro pa izračunaj tretjo potenco in iz tako dobljenega števila ohrani samo enice. Primer: iz števila 1234 ustvari 4871 (1 in 4 smo zamenjali, tretja potenca druge cifre 2 je enaka 8, tretja potenca tretje cifre 3 je enaka 27, ohranimo pa le enice, to je 7).

Referenčni podatkovni tipi

Spremenljivke referenčnih podatkovnih tipov hranijo referenco na ustrezen podatek in ne podatka samega. Tega se pri delu sploh ne zavedamo, zaradi česar se uporaba teh spremenljivk ne razlikuje od spremenljivk vrednostnih tipov.

Referenčne podatkovne tipe bomo obravnavali precej kasneje, zaenkrat jih le naštejmo:

- Razred – **class**
- Vmesnik – **interface**

- Delegati – **delegate**

Med referenčne tipe pa štejemo tudi vgrajene referenčne tipe: to pa so tip **string** in tip **object**.

Podatkovni tip **string** (niz) označuje zaporedje znakov. Spremenljivko tega tipa v **C#** lahko inicializiramo tudi takole:

```
string var5;
var5 = "Lokomotiva";
```

Najpomembnejše lastnosti in metode razreda string za delo z zaporedji znakov (stringi)

Indeks	Razlaga
[indeks]	Dostop do znaka na določeni poziciji.
Lastnost	**Razlaga**
Length	Število znakov stringu.
Metoda	**Razlaga**
StartsWith(string)	Vrne logično vrednost, ki označuje, ali se nek string začenja z navedenim stringom.
EndsWith(string)	Vrne logično vrednost, ki označuje, ali se nek string končuje z navedenim stringom.
IndexOf(string[,začetni indeks])	Vrne celo število ki predstavlja pozicijo (indeks) prve pojavitve navedenega stringa v nekem stringu od začetnega indeksa naprej. Če začetni indeks ni naveden, se iskanje začne na začetku stringa. Če navedeni string ni najden, je vrnjena vrednost -1.
Insert(začetni indeks, string)	Vrne string v katerega je na navedeno mesto (začetni indeks) vrinjen navedeni string.
PadLeft(skupna_dolžina)	Vrne string, ki je DESNO poravnan in na levi strani zapolnjen s tolikšnim številom presledkov, da je skupno število znakov enako vrednosti **skupna_dolžina**.
PadRight(skupna_dolžina)	Vrne string, ki je LEVO poravnan in na desni strani zapolnjen s tolikšnim številom presledkov, da je skupno število znakov enako vrednosti **skupna_dolžina**.
Remove(začetni_indeks, N)	Vrne string iz katerega je odstranjeno **N** znakov od pozicije **začetni_indeks** naprej.
Replace(stariString, noviString)	Vrne string, v katerem so vse pojavitve stringa **stariString** zamenjane s stringom **noviString**.
Substring(začetni_indeks[,dolžina])	Vrne del stringa, ki se začne na navedeni poziciji in ima navedeno dolžino. Če dolžina ni navedena metoda vrne vse znake do konca stringa.
ToLower()	Vrne string v katerem so vsi znaki zamenjani z malimi znaki.
ToUpper()	Vrne string v katerem so vsi znaki zamenjani z velikimi znaki.

Trim()	Vrne string iz katerega so odstranjeni vsi vodilni in končni presledki.
Split(razmejitveni_znak)	Vrne tabelo stringov v katerem so vsi elementi deli tega stringa (besede) med seboj razmejeni z znakom **razmejitveni_znak**.

Privzeta vrednost referenčnih tipov spremenljivk (v primeru, da jih ne inicializiramo) je `null`.

Nekaj primerov:

```
//dostop do posameznega znaka v stringu
string znaki = "abcdefg";
char a=znaki[0];     // 'a'
char b=znaki[1];     // 'b'

string beseda="Danes je lep dan!";
//sestavimo nov niz iz nekaterih znakov niza beseda
string znakiInPresledki = beseda[0]+ beseda[3]+ beseda[4]+ beseda[14];
//znakiInPresledki dobi vrednost "Desa"

//Uporaba metod StartsWith in EndsWith
bool zacneZabc = znaki.StartsWith("abc");   // true
bool koncaZabc = znaki.EndsWith("abc");     // false

//Uporaba metode IndexOf
string sola = "Tehniški Šolski Center Kranj";
int index1 = sola.IndexOf(" ");        // 8, ker se string " " pojavi prvič na osmem mestu
int index2 = sola.IndexOf(' ');        // 8, ker se znak ' ' pojavi prvič na osmem mestu
int index3 = sola.IndexOf("Center");   // 16, ker se string "Center" pojavi prvič na 16 mestu
int index4 = sola.LastIndexOf(" ");    // 22, ker se string " " pojavi zadnjič na 22 mestu

//Uporaba metod Remove, Insert in Replace
sola = sola.Remove(0, 9);                          // Šolski center Kranj
sola = sola.Insert(sola.Length, ", Slovenija"); // Šolski center Kranj, Slovenija
sola = sola.Replace("Slovenija", "4000 Kranj"); // Šolski center Kranj, 4000 Kranj

//Uporaba metod Substring, ToUpper in ToLower
string ime = "aNJA";
string prvaCrka = ime.Substring(0, 1).ToUpper(); // A
string drugeCrke = ime.Substring(1).ToLower();  // nja
ime = prvaCrka + drugeCrke;                      // Anja

//Kopiranje enga stringa v drug string
string s1 = "abc";
string s2 = s1;       // string s2 dobi vrednost strinfa s1, torej s2 postane "abc"
s2 = "def";           // string s2 postane "def", string s1 pa se ne spremeni
string s3 = s1 + s2;  // strig s3 dobi vrednost "abcdef"

//Uporaba metode Substring
string polnoIme = "  Edward C Koop    ";     // "  Edward C Koop    "
polnoIme = polnoIme.Trim();                  // "Edward C Koop"
int prvipresledek = polnoIme.IndexOf(" ");   // 6
string lastnoIme = polnoIme.Substring(0,prvipresledek); // "Edward"

//Uporaba razmejitvenih znakov v stringu
string naslov = " |34 Kališka ulica|Slovenija|Kranj|4000 ";
naslov = naslov.Trim();        // "|34 Kališka ulica|Slovenija|Kranj|4000"
naslov.Remove(0,1);            //iz naslova odstranimo prvi znak
string naslov1= naslov.Remove(naslov.Length-1,1);// iz naslova odstranimo zadnji znak

int indeksUlice = naslov.IndexOf("|") + 1;                                    // 1
int indeksDrzave = naslov.IndexOf("|", indeksUlice) + 1;                      // 18
int indeksKraja = naslov.IndexOf("|", indeksDrzave) + 1;                      // 28
int indeksPoste = naslov.IndexOf("|", indeksKraja) + 1;                       // 34

string ulica = naslov.Substring(0, indeksDrzave-1);                        // 34 Kališka ulica
string mesto = naslov.Substring(indeksKraja, indeksPoste - indeksKraja - 1);  // Kranj
string posta = naslov.Substring(indeksPoste);                                 // 4000
```

```
//Uporaba metode Split
string pisatelj = " Edward C Koop  ";   // " Edward C Koop  "
pisatelj = pisatelj.Trim();             // "Edward C Koop"
string[] imena = pisatelj.Split(' ');
string imePisatelja = imena[0];         // "Edward"
string imePisatelja1 = imena[1];        // "C"
string imePisatelja2 = imena[2];        // "Koop"

//Uporaba metode Insert
string telefonskaStevilka = "041827919";                  // "041827919"
telefonskaStevilka = telefonskaStevilka.Insert(3, "-");   // "041-827919"
telefonskaStevilka = telefonskaStevilka.Insert(7, "-");   // "041-827-919"

//Uporaba metode Replace
string datum = "21-08-2007";                  // "21-08-2007"
datum = datum.Replace("-", "/");
```

Naloge:

- Dana je deklaracija `string stavek = "moj stavek";` S pomočjo lastnosti **Length** ter metode **ToUpper** spremeni prvo in zadnjo črko tega stringa v veliko črko. Prvo in zadnjo črko tega stringa spremeni v veliko črko.

- Ugotovi na katerem mestu stringa *stavek* se nahaja prva pojavitev veznika in (uporabi metodo **IndexOf**)?

- Iz stringa *stavek* odstrani vse znake od 10. znaka naprej! Uporabi metodo **Remove**!

- Dana je spremenljivka *stavek* tipa string, spremenljivka ima že neko vrednost (vsebuje več kot 10 znakov).
  - Odstrani vse vodilne in končne presledke;
  - Vse črke v stringu *stavek* spremeni v velike črke angleške abecede;
  - Ugotovi in izpiši prvi in zadnji znak tega stringa;
  - Ugotovi in izpiši koliko znakov je v tem stringu;
  - V spremenljivko *beseda* (string) zapiši prvih pet znakov stringa *stavek*;
  - Zadnje tri znake tega stringa nadomesti s pikicami;
  - Na sredino stringa *stavek* vrini tri podčrtaje;
  - Iz stringa *stavek* odstrani vse znake od šestega znaka naprej.

- V stringu ***stavek*** vse številke nadomesti z besednim opisom (številko 1 zamenjaj z ena, številko 2 z dva, …)

- Dane so trije stringi *st1*, *st2* in *st3*, ki so že inicializirani. Deklariraj string *st4*, ki naj bo sestavljen iz zadnjih znakov stringov *st1*, *st2* in *st3*!

Pretvarjanje osnovnih podatkovnih tipov

Pri delu s spremenljivkami se velikokrat srečamo s potrebo po pretvarjanju osnovnih podatkovnih tipov (npr. zaporedja znakov v celo število ali realno število, znaka v cifro, celo število v string, …). Najpogosteje za pretvarjanje uporabljamo razred **Convert**, ki vsebuje celo vrsto metod za pretvarjanje. Omenimo le nekaj najpomembnejših:

- **ToInt32** – pretvorba določene vrednosti v 32 bitno predznačeno (**signed**) celo število
- **ToInt64** – pretvorba določene vrednosti v 64 bitno predznačeno (**signed**) celo število
- **ToByte** – pretvorba v 8-bitno nepredznačeno celo število
- **ToDouble** – pretvorna določene vrednosti v vrednost tipa **double**
- **ToDecimal** – pretvorba v decimalno število

- **ToSingle** – pretvorba v spremenljivko tipa **float**
- **ToChar** – pretvorba v znakovno vrednost
- **ToBoolean** – pretvorba v logično vrednost
- **ToString** – pretvorba v zaporedje znakov - **string**

Primeri uporabe:

```
int celo stevilo = Convert.ToInt32("12345");
long veliko_celo = Convert.ToInt64("123456789");
float realno = Convert.ToSingle("12.89");//namesto decimalne pike lahko pišemo vejico!!!
double veliko_stevilo = Convert.ToDouble("250000.89");
char znak = Convert.ToChar(65);
bool logicna = Convert.ToBoolean(1);
double vsota = 100.22 + 200.33;
string stavek = Convert.ToString(vsota);
```

Standardne oznake/kode za formatiranje števil

V naslednji tabeli so prikazane standardne kode (oznake - šifre), ki jih uporabimo za formatiranje števil, kadar jih želimo z metodo **ToString** spremeniti v zaporedje znakov – string. Katerokoli od teh kod (znakov) lahko uporabimo znotraj oklepajev metode ToString, obvezno pa jih moramo zapisati med dvojna narekovaja.

Oznaka	Format	Opis
C ali c	Denarna valuta	Formatiranje števila kot denarno valuto z določenim številom decimalnih mest.
P ali p	Procent	Formatiranje števila kot procent, z določenim številom decimalnih mest.
N ali n	Številka	Formatiranje števila z ločilom za tisočice in določenim število decimalnih mest.
F ali f	Float	Formatiranje števila kot decimalnega števila, z določenim število decimalnih mest.
D ali d	Cifre	Formatiranje celega števila z določenim število cifer.
E ali e	Eksponent	Formatiranje števila v eksponentni obliki z določenim število decimalnih mest.
G ali g	Splošno	Formatiranje števila v decimalni ali eksponentni obliki odvosno od tega, katera oblika je bolj primerna.

Nekaj primerov:

```
decimal znesek=1547.20m;
string  noviznesek=znesek.ToString("c"); //Rezultat: €1.547,20

decimal obresti=0.023m;
string obrestnamera=obresti.ToString("p1");//v stringu bo zapisano eno decimalno mesto,
                                           //Rezultat: 2,3%

float vrednost = 15000;
string novavrednost = vrednost.ToString("n0");//v stringu bo zapisano 0 decimalnih mest
                                              //Rezultat: 15.000

decimal skupaj=432818.678m;
string vseskupaj = skupaj.ToString("f2");//v stringu bosta zapisani 2 decimalni mesti, število
                                         //bo zaokroženo. Rezultat: 432818,68
decimal cena = 145345.23m;
string novacena = vrednost.ToString("n2");//izpis bo formatiran na tisočice, v stringu bosta
                                          //zapisani 2 decimalni mesti. Rezultat: 145.345,23
```

Pri formatiranju števil si lahko pomagamo tudi z metodo **Format** razred **String** (pozor, **String** z veliko začetnico!!!).

Nekaj primerov;

```
//Z metodo Format lahko formatiramo lahko več števil hkrati. Zapis 0:c pomeni, da se bo prvo
število (oznaka 0) formatiralo s šifro c
string novacena = String.Format("{0:c}", 1547.2m);    //Rezultat 1.547,20 €

//Zapis 0:c pomeni, da se bo prvo število (oznaka 0) formatiralo s šifro c, zapis 1:p pa
//pomeni, da se bo drugoo število (oznaka 1) formatiralo s šifro p - torej kot procent
//Rezultat naslednjega formata: Znesek: 1.547,20 €,  Procent: 12,00%
string novacena = String.Format("Znesek: {0:c},  Procent: {1:p}", 1547.2m,0.12);

string obrestnamera = String.Format("{0:p1}", .023m); //Rezultat 2,3%
string novavrednost=String.Format("{0:n0}",15000);    //Rezultat 15.000
string vseskupaj = String.Format("{0:f3}", 432.8175); //Rezultat 432,818

//format {0,30} pomeni desno poravnavo teksta, format {0,-30} pa levo poravnavo
string stavek=string.Format("{0,30}{1,9:f1}",beseda,hitrost); //leva poravnava
string stavek1 = string.Format("{0,-30}{1,-9:f1}", beseda, hitrost); //desna poravnava
```

Običajne (custom) oznake/kode za formatiranje števil

Če nam standardno formatiranje števil ne zadošča za oblikovanje formata, ki ga želimo, lahko kreiramo svoj lasten format. Pri takem formatiranju uporabimo običajne (**custom**) šifre za formatiranje.

Tako v metodi **ToString**, kot v metodi **Format** lahko z uporabo treh sekcij, ki so med seboj ločene z znakom podpičje določimo, kako naj bo formatirano pozitivno število, kako negativno število in kako naj bo formatirano število 0.

Seznam običajnih oznak (kod) za formatiranje števil;

Oznaka	Pomen
0	Mesto rezervirano za cifro 0.
#	Mesto rezervirano za cifro.
.	Decimalna pika.
,	Decimalno ločilo.
%	Mesto rezervirano za znak procent.
;	Ločilo posamezne sekcije.

```
decimal znesek = -1547.20m; //negativno število

string znesekS=znesek.ToString("€#,##0.00"); //Rezultat -€1.547,20
string znesekS1=znesek.ToString(" €#,##0.00"); //Rezultat - €1.547,20

znesek = 1547.20m; //pozitivno število

//PRIKAZ SEKCIJ v formatiranju
//če je število, ki ga formatiramo poztivno, se upošteva prvi format (prva sekcija), če je
//negativno druga sekcija, cicer pa tretja sekcija. Ločilo za sekcije je znak ";";

string znesekS1 = znesek.ToString("€#,##0.00;(-#,##0.00);Nič");//Rezultat: €1.547,20
string znesekS2 = String.Format("{0:€#,##0.00;(-#,##0.00);Nič}",znesek); //Rezultat: €1.547,20
```

```
decimal znesek1 = -1547.20m; //negativno število

string znesekS3 = znesek1.ToString("€#,##0.00;(-#,##0.00);Nič");//Rezultat: (-1.547,20)
string znesekS4 = String.Format("{0:€#,##0.00;(-#,##0.00);Nič}", znesek1);
//Rezultat: (-1.547,20)

//Rezultat naslednjega formata: Znesek: €1.547,20,  Procent: 12,00%
string novacena1 = String.Format("Znesek: {0:€#,##0.00},  Procent: {1:p}", 1547.2m, 0.12);

decimal znesek2 = 0;
string znesekS5 = znesek2.ToString("€#,##0.00;(#,##0.00);Nič"); //Rezultat: Nič
```

Naloge:

- Dane so deklaracije

```
string ulica1 = "Trubarjeva ulica";
string ulica2 = "Mali trg";
int st1 = 12,st2 = 2390;
```

Formatiraj stringa *naslov1* in *naslov2* tako , da bo ulica formatirana na 30 mest, hišna številka pa na 5 mest! Ulica naj bo poravnana levo, hišna številka pa desno!

- Dani sta deklaraciji:

```
string stevilo1 = "200";
string stevilo2 = "300";
```

Oba podatka pretvori v celoštevilski vrednosti in izračunaj njuno vsoto

- Dane so spremenljivke:

```
string ime="Andrej";
int starost=20;
double velikost=178.45;
```

Formatiraj izpis tako, da dobiš na zaslonu takle izpis:

```
Naziv         starost          velikost
-----------------------------------------
Andrej             20            178.45
```

- Kakšno vrednost dobi spremenljivka *stavek* po izvedbi naslednjih stavkov?

```
string jezero = "Bohinjsko jezero";
double razdalja=178.45124;
stavek = string.Format("{0,-30}{1,9:f2}", jezero, razdalja);
```

- Kakšno vrednost dobita spremenljivka *znesek1* in *znesek2* po izvedbi naslednjih stavkov?

```
decimal znesek = -1547.20m; //pozitivno število

string znesek1 = znesek.ToString("#,##0.00");//Rezultat: €1.547,20
string znesek2 = znesek.ToString("#,##0.00;(-#,##0.00)");
```

- Kakšno vrednost dobi spremenljivka *obvestilo* po izvedbi naslednjih stavkov?

```
int st = 15;
```

```
int st1 = 6;
decimal procent = .15m;
string obvestilo = String.Format("Od {0,3} izdelkov se jih je kar {1,3} podražilo za
                                  več kot {2:p1}!",st,st1,procent);
```

Vhodni in izhodni stavki (branje in izpis podatkov)

Branje (**Console.ReadLine()**) in izpisovanje (**Console.WriteLine()**) podatkov je v C# izvedeno z branjem in zajemanjem nizov – stringov. Že v prejšnjih primerih smo prikazali nekaj preprostih izhodnih stavkov – izpisovanje rezultatov na konzolo (ekran).

Primer:

```
//izpis teksta, na koncu pa se doda oznaka za prehod v novo vrstico
Console.WriteLine("Pozdravljen C#");
Console.Write("Pozdravljen C#"); //izpis brez dodanega znaka za prehod v novo vrstico
```

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** ali pa metode razred **Convert**. Največkrat bomo uporabljali slednje.

Primer:

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

//če želimo prebrati spremenljivko tipa int, se moramo zavedati, da metoda ReadLine vrne
//string, ki ga moramo spremeniti v int
Console.Write("Vnesi celo število: ");
int stevilo = Convert.ToInt32(Console.ReadLine());

//ali pa

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

Z metodo **WriteLine** lahko izpisujemo tudi več parametrov (spremenljivk, konstant) hkrati. V tem primeru moramo znotraj prvega parametra (ki je neko zaporedje znakov – **string**) za vsak dodatni parameter (spremenljivko) napovedati njegov položaj znotraj izpisa in njegovo zaporedno številko (to storimo s parom znakov **{}**, znotraj oklepajev pa zaporedna številka parametra).

Primeri:

```
decimal znesek = 1297.86m;
Console.WriteLine("Izpis decimalnega števila: {0}", znesek);              //Izpis: 1297,86

decimal procent = 14.5m;
//Rezultat deljenja ni zaokrožen
Console.WriteLine("14.5% od 1297.86 = {0}",(znesek / 100) * procent);
//Izpis: 14.5% od 1297.86 = 188,18970

//Naslednji rezultat v izpisu je zaokrožen na 2 decimalki
Console.WriteLine("14.5% od 1297.86 = {0:F2}",(znesek / 100) * procent);//Izpis na 2 decimalki


//Rezultat zaokrožimo s pomočjo oznake za format
Console.WriteLine("14.5% od 1297.86 = {0:F2}",(znesek / 100) * procent);
//Izpis: 14.5% od 1297.86 = 188,18970

//Rezultat zaokrožimo s pomočjo metode Round razreda Math
decimal rezultat=Math.Round((znesek / 100) * procent,2);
Console.WriteLine("14.5 % od 1297.86 = {0}", rezultat);
//Izpis: 14.5% od 1297.86 = 188,18970

//ali pa kat takole
```

```
Console.WriteLine("14.5% od 1297.86 = {0}", Math.Round((znesek/100) * procent,2));
//Izpis: 14.5% od 1297.86 = 188,18970

decimal znesek = 100.55m;
decimal znesek1 = 166.5m;
decimal znesek2 = 1600.67m;
//izpis formatiran na 15 mest za vsako spremenljivko
Console.WriteLine("{0,15}{1,15}{2,15}",znesek,znesek1,znesek2);
```

V enem od zgornjih primerov smo uporabili metodo razreda **Math**. Razred **Math** vsebuje številne metode za uporabo matematičnih funkcij. Nekatere od njih so zbrane v spodnji tabeli:

Lastnost	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.
Ceiling	Najmanjše celo število večje ali enako od števila, ki nastopa kot argument metode.
Cos	Kosinus kota.
Exp	Eksponentna funkcija (eksponent je iracionalno število e).
Floor	Največje celo število manjše ali enako od števila, ki nastopa kot argument metode.
Log	Logaritem.
Log10	Desetiški logaritem.
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

decimal stevilo = 6.45327m;
decimal cstevilo = Math.Ceiling(stevilo);
Console.WriteLine(cstevilo);  //izpis 7

double gama = Math.PI;
double cosgama = Math.Cos(gama);
Console.WriteLine(cosgama);  //izpis -1

decimal stevilo1 = 6.45327m;
decimal cstevilo1 = Math.Floor(stevilo1);
Console.WriteLine(cstevilo1);  //izpis 6

double st = 100;
double log10st = Math.Log10(st);
Console.WriteLine(log10st);  //izpis 2

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

decimal decimalno = 245.67843m;
Console.WriteLine(Math.Round(decimalno)); //izpis 246
int celo = Convert.ToInt32(Math.Round(decimalno));
Console.WriteLine(celo);  //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

decimal velikost = 165.45m;
int vel = Convert.ToInt32(Math.Truncate(velikost));
Console.WriteLine(vel); //izpis 165
```

Naloge:

- Dana je poljubna spremenljivka tipa decimal, ki je že inicializirana
  - kolikšen je njen celi del
  - kolikšen je njen decimalni del
  - zaokroži jo na dve decimalki natančno
  - izračunaj in izpiši njen kvadratni koren
  - izračunaj in izpiši četrto potenco tega števila
- Dani sta kateti pravokotnega trikotnika. Izračunaj hipotenuzo, in notranje kote tega trikotnika.
- Zapiši v C# naslednji matematični izraz

$$x = \frac{(2a-b)^2 + \sqrt{a3 + b2}}{3xy^3}$$

- Premer kroga znaša 10 cm. Ugotovi obseg in ploščino kroga. Kakšna tetiva pripada središčnemu kotu 33 stopinj? Rezultat zaokroži na dve decimalki natančno!

- V pravokotnem koordinatnem sistemu je točka A(5,12). Ugotovi njeno razdaljo od koordinatnega izhodišča! Rezultat zaokroži na tri decimalke!

- Dani sta stranici romboida in njun vmesni kot podan v stopinjah. Izračunaj obseg, ploščino ter obe diagonali tega romboida.

- Napiši program, ki bo za poljubno količino mleka, podanega v litrih (1 liter je enak enemu kubičnemu decimetru) izračunal in izpisal količino potrebne embalaže oblike tetrapak (dimenzije ene embalaže so: dolžina 1 dm, širina 0.5 dm in višina 2 dm !). Izpis ustrezno oblikuj(formatiraj)!

- Dana je premica $y = 8x + 16$. Ugotovi in izpiši:

  - pod kakšnim kotom seka abscisno os
  - koordinati presečišč z obema osema
  - ploščino pravokotnega trikotnika, ki jo tvori premica z obema osema

Krmilni stavki v C#

Stavek if

Osnovni krmilni stavek v **C#** je **if** stavek. Uporabljamo ga tedaj, ko želimo programski tok razvejiti na dve veji. Poznamo tri oblike tega stavka:

- pogojna izvršitev,
- razvejitev in
- gnezdena oblika.

Pogojna izvršitev

```
if (pogoj P) // pogoj mora biti vedno v oklepajih
{
   Stavek1; //poljuben stavek
   Stavek2; //poljuben stavek
}
```

Stavek **S** se izvede v primeru, da je pogoj **P** izpolnjen, sicer pa se program nadaljuje za **if** stavkom. V primeru, da je znotraj if stavka en sam stavek, lahko zavite oklepaje izpustimo:

```
if (pogoj P) // pogoj mora biti vedno v oklepajih
   Stavek1; //poljuben stavek
```

Razvejitev

```
if (pogoj P)
{
   stavek S1;  //poljuben stavek
}
else
{
   Stavek S2;  //poljuben stavek
}
```

Stavek **S1** se izvede v primeru, da je pogoj **P** izpolnjen, sicer pa se izvede stavek **S2**.

V pogojih lahko uporabljamo relacijske operatorje:

Relacijski Operator	Pomen
<	Manjše.
>	Večje.
==	Enako.

<=	Manjše ali enako.
>=	Večje ali enako.
!=	Različno.

Pogoji so lahko tudi sestavljeni. Sestavljamo jih lahko s pomočjo logičnih operatorjev.

Logični Operator	Pomen
&&	Logični in (**and**).
||	Logični ali (**or**).
!	Negacija (**not**).

Če je združenih več pogojev lahko uporabimo oklepaje. Vrstni red operacij je takšen, kot to velja pri matematiki in ga določajo oklepaji.

Gnezdeni if stavek

Gnezdeni **if** stavek je **if** stavek znotraj **if** stavka. Pri takih stavkih moramo biti pozorni na to, v katerem primeru se izvede **else** veja takega gnezdenega stavka. Pomagamo si seveda z oklepaji **{}**.

```
int stevilo = Convert.ToInt32(Console.ReadLine());
if (stevilo > 10)
{
    if (stevilo % 2 == 0)
       Console.WriteLine("Sodo število večje od 10");
    else
       Console.WriteLine("Liho število večje od 10");
}
else
{
    if (stevilo % 2 == 0)
       Console.WriteLine("Sodo število manjše od 10");
    else
       Console.WriteLine("Liho število manjše od 10");
}
```

Vaja:

```
/* Preberi dvomestno število in izpiši njegove desetice in enice. Če vnešeno število
ni dvomestno, naj program izpiše obvestilo*/

Console.Write("Vnesi število : ");
int stevilo=Convert.ToInt32(Console.ReadLine());
if (stevilo>=10 && stevilo <=100)
{
   Console.WriteLine("desetice : {0}",stevilo/10);
   Console.WriteLine("enice : {0}",stevilo %10); // % = ostanek pri celoštevilčnem deljenju
}
Else
 Console.WriteLine("Število ni v dogovorjenih mejah!");
```

Vaja:

```
/*Program naj zahteva vnos starosti neke osebe, nato pa izpiše za kakšno vrsto osebe gre in
sicer
```

```
           do 2 leti  Dojenček
           3-10 let - Mladoletnik
           11-19 let - Najstnik
           20 in več - Odrasla oseba*/

Console.Write("Starost osebe v letih: ");
int starost=Convert.ToInt32(Console.ReadLine());
if (starost<=2)
   Console.WriteLine("Dojenček!");
else if (starost>=3 && starost<=10)
   Console.WriteLine("Mladoletnik!");
else if (starost>=11 && starost<=19)
   Console.WriteLine("Najstnik!");
else Console.WriteLine("Odrasla oseba!");
```

Vaja:

```
//REŠEVANJE TRIKOTNIKA

Console.Write("Vnesi stranico a : ");
double a = Convert.ToDouble(Console.ReadLine());
Console.Write("Vnesi stranico b : ");
double b = Convert.ToDouble(Console.ReadLine());
Console.Write("Vnesi stranico c : ");
double c = Convert.ToDouble(Console.ReadLine());

if (a + b < c || a + c < b || b + c < a)  // logicni ali zapisemo v C# takole: ||
    Console.WriteLine("Tak trikotnik ne obstaja!");
else
{
    double s = (a + b + c) / 2;
    double plosc = Math.Sqrt(s * (s - a) * (s - b) * (s - c));
    double vc = 2 * plosc / c;
    double va = 2 * plosc / a;
    double vb = 2 * plosc / b;
    //funkcija asin je obratna/inverzna funkcija funkcije sinus
    double alfa = Math.Asin(vc / b) * 180 / Math.PI;
    double beta = Math.Asin(vc / a) * 180 / Math.PI;
    double gama = 180 - alfa - beta;
    double R = a / (2 * Math.Asin(vc / b));
    double r = a * b * c / (4 * plosc);
    Console.WriteLine("Obseg trikotnika       : {0:F}", 2 * s);
    Console.WriteLine("Ploščina trikotnika    : {0:F} ",plosc);
    Console.WriteLine("Višina na stranico c   : {0:F}",vc);
    Console.WriteLine("Višina na stranico a   : {0:F} ", va);
    Console.WriteLine("Višina na stranico b   : {0:F}", vb);
    Console.WriteLine("Kot alfa               : {0:F}", alfa);
    Console.WriteLine("Kot beta               : {0:F}", beta);
    Console.WriteLine("Kot gama               : {0:F} ", gama);
    Console.WriteLine("Polmer včrtanega kroga : {0:F} ", R);
    Console.WriteLine("Polmer očrtanega kroga : {0:F} ", r);
}
```

Naloge:

- Napiši program, ki zahteva vnos stranic trikotnika in ugotovi, ali tak trikotnik sploh obstaja, ali je trikotnik pravokoten, ali je trikotnik enakokrak in ali je trikotnik mogoče enakostraničen.
- Napišim program, ki bo pomagal ugotoviti, ali bo na določeno leto prestopno ali ne. Leto je prestopno, kadar je deljivo s 4 in ne s 100 ali kadar je deljivo s 400.
- Ugotovi pravilnost oz. nepravilnost naslednjega pogoja, če je x = 5

  ((3 > x) || (5 <= x)) && (x! = 8)

- Kakšna je vrednost spremenljivke N po izvedbi naslednjega **if** stavka?

```
int N = 1;
bool B = true;
if ((N < 5) && B)
{
N = N + 1;
}
else
{
N = 0;
}
```

- Preberi poljubno celo število. Ugotovi in izpiši, ali je sodo ali liho!

- Šola organizira izlet za učence na šoli. Cena izleta je 450 EUR, če se prijavi do vključno 30 učencev. Vsaka naslednja prijava zniža ceno na posameznika za 1%. Če se prijavi manj kot 20 učencev izlet ne bo organiziran. Napiši program, ki bo zahteval vnos števila učencev in nato izračunal in izpisal ceno izleta na posameznega učenca, oziroma, da je premalo prijav.

- Sestavi program, ki ugotovi, ali je poljubno trimestno število deljivo s svojo srednjo cifro.

- Na transakcijskem računu imaš določen znesek, dovoljen limit je 500 EUR. Ker ti je zmanjkalo denarja, želiš dvigniti določen znesek. Če limit ni presežen, bankomat izplača izbrani znesek. Napiši program, ki bo v primeru opravljene transakcije izpisal dvignjen znesek, stanje na računu in razpoložljivo stanje na računu. Če transakcija ni dovoljena, bo izpisal le stanje in razpoložljivo stanje na računu. Stanje na računu in znesek dviga naj program prebere.

- Indeks staranja prebivalstva izračunamo tako, da število prebivalcev starejših od 60 let delimo s številom mladih do 20 in količnik pomnožimo s 100. Na podlagi rezultatov lahko ugotovimo tip starostne strukture. Napiši program, ki bo prebral število mladih do 20 let in starejših od 60 let. Na podlagi teh podatkov izračunaj indeks staranja in s pomočjo pogojnih stavkov določi starostni tip prebivalstva ter ga izpiši.

- Napiši program, ki poišče rešitev kvadratne enačbe.

- Realiziraj naslednjo funkcijo :

$$f(x) = \begin{cases} -3 & ; x < -3 \\ -3*x+2 & ; -3 <= x < 0 \\ 7 & ; x = 0 \\ 3*x - 2 & ; 0 < x < 4 \\ 3 & ; x >= 4 \end{cases}$$

Stavek switch

V primeru, ko želimo program razvejiti na več vej, uporabimo **switch** stavek.

Sintaksa:

```
switch (spremenljivka)
{
   case vrednost1: stavek1; break;
   case vrednost2: stavek2; break;
   case vrednost3: stavek3; break;
   .
   .
   case vrednostN: stavekN; break;

   default:
      stavki;
}
```

Stavki, ki so zapisani v veji **default** se izvedejo le v primeru, da ni bil izveden noben **case** stavek. Za razliko od **switch** stavka v **C++**, v **C#** ni dovoljen impliciten prehod iz ene **case** veje v drugo, razen če je veja prazna.

Primer:

```
//ker smo v prvi veji pozabili break stavek, pride pri prevajanju do napake
int n = 2;
switch (n)
{
    case 1: Console.WriteLine("Vrednost n je enaka  1!"); //NAPAKA - manjka break stavek!!!
    case 3:
       Console.WriteLine("Vrednost n je enaka 3.");
       break;
    default:
       Console.WriteLine("Število n je različno od 1 ali 3!");
    break;
}
```

V primeru, da so posamezne veje **switch** stavka prazne, se izvedejo stavki v prvi veji, ki vsebuje stavek **break**.

```
//ker sta prvi dve veji prazni, dobimo na ekrau izpis:  Vrednost n je enaka  1, 2, ali 3.
int n = 2;
switch (n)
{
    case 1:
    case 2:
    case 3:
       Console.WriteLine("Vrednost n je enaka  1, 2, ali 3.");
       break;
    default:
       Console.WriteLine("Število n je različno od 1, 2 ali 3!");
       break; //break na tem mestu OBVEZEN
}
```

Vaja:

```
//Program, ki prebere številko dneva, izpiše pa njegovo ime.
string dan = Console.ReadLine();
switch (dan)
{
    case "1": Console.WriteLine("ponedeljek"); break;
```

```
    case "2": Console.WriteLine("torek"); break;
    case "3": Console.WriteLine("sreda"); break;
    case "4": Console.WriteLine("četrtek"); break;
    case "5": Console.WriteLine("petek"); break;
    case "6": Console.WriteLine("sobota"); break;
    case "7": Console.WriteLine("nedelja"); break;
    default:
        Console.WriteLine("Napačen vnos!");
        break;
}
```

Vaja:

```
//Program namenejn kavnemu avtomatu.

Console.WriteLine("Vrsta kave: 1=Mala 2=Srednja 3=Velika");
Console.Write("Vnesi ustrezno številko: ");
string s = Console.ReadLine();
int n = Convert.ToInt16(s);
int cost = 0;
switch (n)
{
    case 1:
        cost += 50;
        break;
    case 2:
        cost += 25;
        goto case 1;
    case 3:
        cost += 50;
        goto case 1;
    default:
        Console.WriteLine("Napačen vnos. Izberi  1, 2, al 3.");
        break;
}
if (cost != 0)
{
    Console.WriteLine("Vstavite prosim {0} centov.", cost);
}
Console.WriteLine("Hvala!");
```

Naloge:

- Program naj prebere poljubno cifro in jo izpiše z besedo. V primeru napačnega vnosa naj se izpiše ustrezno obvestilo.

- Napiši program, ki zna računati obseg in ploščino geometrijskih likov. Uporabniku ponudi nabor geometrijskih likov (1 - kvadrat, 2 - pravokotnik, 3 - pravokotni trikotnik in 4 - krog). S pomočjo **switch** stavka ugotovi uporabnikovo izbiro, nato pa v odvisnosti od izbranega lika zahtevaj vnos ustreznih podatkov.

- Z uporabo **switch** stavka kreiraj izpis:

  ocena 1 pomeni nezadostno
  ocena 2 pomeni zadostno
  ocena 3 pomeni dobro
  ocena 4 pomeni prav dobro
  ocena 5 pomeni odlično

- Preberi poljuben stavek dolžine pet znakov. Kreiraj nov stavek *sifriranstavek*, v katerem boš s pomočjo **switch** stavka vse samoglasnike nadomestil z njihovo ASCII kodo!

- Preberi poljuben stavek, nato pa s pomočjo **switch** stavka ugotovi, ali je prvi znak tega stavka samoglasnik, številka, ali pa kaj tretjega!

- Napiši program, ki pretvarja poljuben znesek podan v EUR v drugo valuto, npr. USD in obratno. Na ekranu naj bo sporočilo:

  Pretvarjanje v drugo valuto:
  A – EUR v USD, B – USD v EUR
  Izberi: _

  Po uporabnikovi odločitvi zahtevaj vnos zneska za menjavo in ustrezen menjalniški tečaj, nato pa s pomočjo **switch** stavka (upoštevaj vnos velikih in malih črk!) izračunaj in izpiši rezultat.

- Preberi poljuben stavek, nato pa s pomočjo **switch** stavka ugotovi, koliko je v tem stavku pik, vejic in dvopičij!

- Naslednje zaporedje stavkov spremeni tako, da namesto **if** stavka uporabiš **switch** stavek: (spremenljivka a je tipa **int**) :

```
if  (a == 3)
   Console.WriteLine("a =  3");
else if (a>5 && a<10)
   Console.WriteLine ("a med 5 in 10");
else Console.WriteLine("a je napačen");
```

- Napiši program, ki bo prebral poljubno celo število in ugotovil ter izpisal koliko cifer 1 2 3 4 vsebuje.

Zanke

Zanke so programske strukture, za katere so značilni naslednji elementi:

- števec - spremenljivka katere vrednost se spreminja med izvajanjem zanke,
- začetna vrednost je vrednost, ki določa začetno stanje števca,
- končna vrednost je vrednost pri kateri se izvajanje zanke konča in
- korak zanke je vrednost, ki se prišteje števcu v eni ponovitvi zanke.

Zanka For

Zanka **for** je najbolj poznana in največkrat uporabljena zanka v vseh programskih jezikih . Najpogosteje se pri krmiljenju zanke uporablja spremenljivka v obliki števca, katerega vrednost se po korakih povečuje ali zmanjšuje. Spremenljivka, ki se uporablja kot števec mora biti števna (celo število, znak, naštevni tip, …)

Struktura zanke **for** :

```
for (inicializacja; test; korak)
{
    stavek;
    stavek;
   // …
}
```

ali :

```
for ( inicializacija; test ; korak )
  stavek;
```

inicializacija: določimo začetno vrednost števca (npr int i = 0) ali števcev, če jih je več;
test: določimo pogoj, ki naj se testira pri posameznem prehodu zanke;
korak: določimo, kako naj se povečuje ali zmanjšuje števec (oz. števci, če jih je več)

V vsakega izmed treh glavnih delov ***for*** stavka lahko združimo več stavkov, ki morajo biti ločeni z vejico. Pomembno pa je, da se tako zapisani stavki izvajajo od **leve proti desni**.

Vaja:

```
//Izpis tabele kvadratov števil od 1 do 20. Izpis je formatiran na 8 mest, desna poravnava.
Console.WriteLine("Število     Kvadrat");
for (int i = 1; i <= 20; i++)    //i++ je krajša oblika zapisa    i=i+1;
   Console.WriteLine("{0,8}   {1,8}",i,Math.Pow(i, 2));
```

Vaja:

```
//Program ki poišče vsa naravna števila med 1 in 10000, ki so enaka vsoti kubov svojih števk.
//Eno od števil, ki ima zahtevano lastnost je npr. 153: 153 = 13 + 53 + 33.
int i,j,k,l;  //števke
double st,vk;    //število in vsota kubov števk
for (i=0;i<10;i++)
  for (j=0;j<10;j++)
    for (k=0;k<10;k++)
      for (l=0;l<10;l++)
```

```
      {
        st=1000*i+100*j+10*k+l;
        vk = Math.Pow(i, 3) + Math.Pow(j, 3) + Math.Pow(k, 3) + Math.Pow(l, 3);
        if (st==vk)
              Console.Write(st+",    ");
      }
```

Vaja:

```
//Primer for zanke, ki vsebuje dva števca: eden se povečuje, drugi zmanjšuje.
Console.Write("Vnesi mejo: ");
int konec = Convert.ToInt32(Console.ReadLine());
for (int gor = 1, dol = konec; (gor <= konec) && (dol >= 1); gor++, dol--)
   Console.WriteLine("{0,5}  {1,5}",gor,dol);
```

Vaja:

```
/*Napiši  program ki izpiše naslednji vzorec. Primer:  za  n = 5 je izpis takle:
    5
    4 4
    3 3 3
    2 2 2 2
    1 1 1 1 1
*/

int i,j;
Console.Write("Velikost vzorca (1 - 9): ");
int n = Convert.ToInt32(Console.ReadLine());
for(i=n; i>0; i--) {
  for(j=n; j>=i; j--)
     Console.Write(i);
  Console.WriteLine();
}
```

```
/*Napišite program, ki s tipkovnice prebere dve števili, N in R, ter izračuna vsoto vseh
tistih naravnih števil, manjših ali enakih N, ki so deljiva z R.*/

int vs = 0;
Console.Write("Vnesi naravno število N: ");
int N = Convert.ToInt32(Console.ReadLine());
Console.Write("Vnesi delitelj R: ");
int R = Convert.ToInt32(Console.ReadLine());
for (; N >= R; N--)
{
   if ((N % R) == 0) vs = vs + N;
}
Console.WriteLine("\nVsota je {0}", vs);
```

Naloge:

- Kolikšna je vsota vseh naravnih števil med 1 in 1000?
- Tabeliraj linearno funkcijo f(x) = 4x + 3 na intervalu [-5 , 5] s korakom 1!
- Seštej 100 členov vrste: **vrsta = 1 + 1/2+ 1/3 +1/4 + ..**
- Preberi poljubno celo število in izpiši njegov poštevanko!
- Preberi poljubno celo število in ga izpiši navpično!
- Preberi poljuben stavek in ga izpiši navpično in nato še diagonalno (vsaka črka v svoji vrstici, za en znak bolj v desno!.

- Preberi poljubno celo število in ga izpiši z besedami. Celo število **235** tako izpiši kot **dva tri pet**.
- Napiši program, ki vse šumnike *š*, *č* in *ž* nekega stavka nadomesti z znaki *s*, *c* in *z*!
- Napiši program, ki bo izpisal vse kvadrate naravnih števil do n (n preberemo), ki se naprej in nazaj berejo enako. (npr 141, 232, 181, …).
- Napiši program, ki bo prebral dve celi števili, ugotovil, katero je večje in izpisal:
  - vsa števila med najmanjšim in največjim številom.
  - vsa števila med najmanjšim in največjim številom, ki delijo največje število.
  - vsa števila med najmanjšim in največjim številom, ki so soda in delijo največje število.
  - vsa števila med najmanjšim in največjim številom, ki so soda ali delijo največje število.

While zanka

Zanko **while** (ta tudi zanko **do while**) uporabljamo, kadar število ponavljanj zanke ni vnaprej znano, saj je število ponavljanj zanke odvisno od nekega pogoja. Najpomembnejše značilnosti **while** zanke so

- pogoj je testiran na začetku zanke,
- zanka se izvaja, dokler je pogoj izpolnjen,
- če pogoj ni izpolnjen že na začetku, se zanka ne izvede niti enkrat,
- pogoj, ki ga testiramo je lahko sestavljen.

Struktura **while** zanke:

```
while (pogoj)
{
    Stavki  //eden ali več poljubnih stavkov
}
```

Vaja:

```
/*Izračunaj vsoto N členov zaporedja, če poznaš splošni člen a[n]=(n+1)*(n-1); n = 1,2,3,...N
Členi zaporedja so potemtakem: 0,3,8,15,24,35,...
Izpis naj vsebuje tudi člene zaporedja, ter vsoto prvih N členov.*/

int n=1,vsota=0;
Console.WriteLine("Zaporedje ima splosni clen a[n]=(n+1)*(n-1); n =1..N!\n");
Console.Write("Vpisi stevilo clenov zaporedja: ");
int N=Convert.ToInt32(Console.ReadLine());
while (n <= N) // tekoci člen <= n-temu clenu
{
    int clen;
    clen = (n+1) * (n-1); // izračun n-tega člena
    Console.WriteLine(n+". člen zaporedja je:" +clen); //izpis člena
    vsota = vsota+ clen; //vsoto povečamo za n-ti člen
    n=n+1;  //krajši zapis tega stavka je:    n++;
}
Console.WriteLine("\nVsota prvih "+N+" clenov tega zaporedja je "+ vsota);
```

Vaja:

```
/*Napiši program, ki iz prebranega celega števila naredi novo število, v katerem so le sode
          cifredanega števila. Če so vse cifre danega števila lihe, je novo število enako 0.
*/

int novostevilo=0,cifra,faktor=1;
Console.Write("Vnesi polubno celo število: ");
```

```
int stevilo=Convert.ToInt32(Console.ReadLine());
while (stevilo>0){
    cifra=stevilo%10;
    if (cifra%2==0)
    {
        novostevilo=novostevilo+cifra*faktor;
        faktor=faktor*10;
    }
    stevilo=stevilo/10;
}
Console.WriteLine("\nNovo stevilo je {0}\n",novostevilo);
```

Vaja:

```
/*izpis vseh stirimestnih stevil, pri katerih je vsota zadnjih dveh cifer enaka vsoti prvih
dveh izpis po 10 v vrsti. Koliko je takih števil? */

int stevilo = 1000, e, d, s, t, vvrsti = 0, vseh = 0;
Console.Clear();
while (stevilo < 9999)
{
    e = stevilo % 10;    //enice
    d = (stevilo % 100) / 10;  //desetice
    s = (stevilo % 1000) / 100;  //stotice
    t = (stevilo % 10000) / 1000;  //tisočice
    if (e + d == s + t)
    {
        Console.Write("{0}, ", stevilo);
        vvrsti = vvrsti + 1;
        vseh = vseh + 1;
        if (vvrsti % 10 == 0) Console.WriteLine();
    }
    stevilo = stevilo + 1;
}
Console.WriteLine("\n\nVseh takih števil je {0}.\n", vseh);
```

Naloge:

- Preberi poljuben string. S pomočjo while zanke ugotovi in izpiši, koliko presledkov vsebuje.
- Preberi kemijsko formulo in jo izpiši tako, da so številke vrstico nižje. Če npr. prebereš formulo H2SO4, jo izpiši kot

  H SO
  2 4

- Dana sta dva stringa, *ime* in *priimek*. Sestavi nov string *sifra* tako, da najprej obrneš ime in priimek, nato pa izmenično iz zaporednih črk obrnjenega imena in priimka sestaviš nov string. Iz stringov *ime* in *priimek* vzemi le toliko črk, kot znaša dolžina krajšega od obeh stringov.
- Sestavi program *Kopije*, ki prebere niz *beseda* in pozitivno celo število ***k*** ter izpiše niz, ki je sestavljen iz ***k*** kopij niza *stavek*.
- Sestavi program, ki bo prebral naravno število in izračunal produkt njegovih neničelnih števk. Primer: za število 2304701 program vrne 168.
- Preberi poljubno celo število manjše od 100, računalnik pa naj ga ugane. Seveda predpostaviš, da računalnik števila ne pozna in da ga ugiba. Ugotovi in izpiši koliko poskusov bo računalnik potreboval za to, da ugotovil pravo število. Kakšno taktiko pa bo računalnik ubral, je odvisno od tebe. Nekaj primerov:
  - naključno ugiba števila, dokler ne ugane pravega. Pri tem nič ne upošteva podatka o tem, ali je izžrebano število od iskanega manjše ali večje.

- gre po vrsti od 1 do 100. Pri tem nič ne upošteva podatka o tem, ali je izžrebano število od iskanega manjše ali večje.
- naključno ugiba števila, dokler ne ugane pravega. Pri tem upošteva podatek ali je naključno število preveliko ali premajhno število tako, da zoži interval, iz katerega žreba števila.

- Napiši program ki naj izpiše vsa trimestna števila, katerih vsota števk je 20.

- Oče bi si rad kupil avtomobil, zato se je odločil 1 mesec varčevati na prav poseben način. Prvi dan bo dal na stran 1 tolar, drugi dan 2 tolarja, tretji dan 4 tolarje itd., vsak dan torej dvakrat toliko kot prejšnji dan. Program naj izračuna, po koliko dnevih si lahko oče kupi avtomobil. Ceno avtomobila vnesemo sami.

- Leta 2000 je bila dolžina kapnika 3 mm, nato pa se vsakih 10 let poveča za 6 mm. Napiši program, s katerim boš ugotovil in nato izpisal, kolikšna bo višina kapnika leta 2020 in katerega leta bo dosegel višino 1,5 m ?

- Napiši program, ki izpiše vsa naravna števila med 1 in 3000, ki so deljiva s 7 in niso liha!

Do while zanka

Zanko **do while** (tako kot zanko **while**) uporabljamo, kadar število ponavljanj zanke ni vnaprej znano, saj je število ponavljanj zanke odvisno od nekega pogoja. Najpomembnejše značilnosti **do while** zanke so

- pogoj je testiran na koncu zanke,
- zanka se izvaja, dokler je pogoj izpolnjen,
- zanka se v vsakem primeru izvede vsaj enkrat, četudi pogoj ni izpolnjen že na začetku,
- pogoj, ki ga testiramo je lahko sestavljen.

Struktura **do while** zanke:

```
do
{
    Stavki  //eden ali več poljubnih stavkov
}
while (pogoj)
```

Vaja:

```
/*Program, ki iz vnesenih znakov sestavi številko */
Console.Write("Vnesi zaporednje znakov in pritisni Enter: ");
char znak;
int stevilo = 0;
int cifra;
int faktor = 1;

do
{
    znak = Convert.ToChar(Console.Read());   //prestrežemo znak s tipkovnice
    if ((znak >= '0') && (znak <= '9'))  //vneseni znak predstavlja cifro
    {
       cifra = (int)(znak - 48); //znak spremenimo v cifro glede na ASCII tabelo znakov
       Console.Write(cifra);  //izpišemo vneseno cifro na ekran, da jo bo uporabnik videl
       stevilo = stevilo * faktor + cifra; //cifre spreminjamo/dodajamo skupnemu številu
       faktor = 10;
    }
}
while (znak != (char)13);  //vnos se zaključi s pritiskom tipke <Enter>

Console.WriteLine("\nVneseno stevilo : {0}.", stevilo);
```

Vaja:

```
/*Program nariše trikotnik iz samih zvezdic */

int velikost;
do
{
    Console.Clear();
    Console.Write("Vnesi velikost trikotnika ( 1 - 24 ) : ");
    velikost = Convert.ToInt32(Console.ReadLine());
}
while (velikost < 1 || velikost > 24); //vneseni podatek mora biti med 1 in 24

int zvezde = 1;
while (velikost > 0)
{
    int p = velikost;
    while (p != 0)
    {
       Console.Write(" ");
       p = p - 1;
    }
    int z = zvezde;
    while (z != 0)
    {
       Console.Write("*");
       z = z - 1;
    }
    Console.WriteLine();
    velikost = velikost - 1;
    zvezde = zvezde + 2;
}
```

Vaja:

```
/*Napiši program, ki izračunannalednjo vsoto za poljubno število sumandov!
              2    2    2    2
             -- + -- + -- + -- + ...
              2    3    4    5
  Izpisuj vmesne vsote!!!   */

int clenov;
do
{
     Console.Clear();
     Console.Write("Vnesi število sumandov ( 1 ali več) : ");
     clenov = Convert.ToInt32(Console.ReadLine());
}
while (clenov < 1); //vneseni podatek mora biti več ali enako 1
float suma = 0;
int n = 0;
while (n < clenov)
{
     suma = suma + (float)2 / (2 + n);
     Console.WriteLine("\n{0} .  {1}", n + 1, suma); //izpisujem vmesne rezultate
     n++;
}
Console.WriteLine("\nVsota {0}. členov tega zaporedja je {1}", clenov, suma);
```

Vaja:

```
/*Napiši program, ki na zaslon izpiše prvih 50 vrednosti zaporedja, podanega kot f(n) = f(n-1)
+ f(n-2) (n je naravno število n=1,2,3,...      Pri tem velja   f(1) = 1   in   f(2) = 1.
V vsaki vrstici naj b po izpisanih po 5 števil, izpis na 15 mest, desna poravnava  */

int n = 3;
long f, f1 = 1, f2 = 1;
Console.WriteLine("Vrednosti funkcije f(n) = f(n-1) + f(n-2) na intervalu od 1 do 40\n\n");
Console.Write("{0,10}{0,10}", f1, f2);
do
{
    f = f1 + f2;
    f2 = f1;
    f1 = f;
    Console.Write("{0,10}", f);
```

```
    n++;
    if (n % 5 == 1) Console.WriteLine(); //zato, ker se zanka začne z n=3
} while (n <= 40);
```

Naloge:

- S pomočjo **do while** zanke beri znake toliko časa, da je vneseni znak enak pika (.). Prebrane znake nato izpiši v obliki stavka.
- Preberi poljubno celo število in ga pretvori v dvojiški sestav. Pretvorjeno število nato izpiši!
- Poišči največji skupni delitelj dveh celih števil
- Napiši program, ki bo izpisal vse kvadrate naravnih števil do n (n preberemo), ki se naprej in nazaj berejo enako (npr 121, 676, 484, …). Uporabi **do while** zanko.
- Sestavi preprost program, ki bo kodiral in dekodiral kratka telefonska sporočila (sms-e). Program mora v obrniti vrstni red besed v sporočilu in vrstni red črk v besedi.
- Sestavi program, ki prebere n decimalnih števil (tudi n je podatek). Izpiši, koliko od teh števil je manjših od 10, med 10 (vključno) in 100 (vključno) in koliko večjih od 100.
- Preberi naravno število n. Če je liho, izračunaj 3n+1, sicer pa n/2.Isto naredi z novim številom in postopek ponavljaj, dokler ne dobiš 1. Izpiši število korakov postopka!
- Nek izdelek naj bi se vsak teden podražil za 2%. Da bi državljani razumeli, kaj to pomeni, napiši program, ki prebere začetno in končno ceno tega izdelka in izpiše, v kolikšnem času (število tednov) bo cena dosegla ali presegla to končno vrednost.
- Sestavi program, ki prebere velikost paralelograma in ga izpiše na zaslon. Izpis paralelograma za velikost 5 naj bo takle

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

- Največ kolikokrat je potrebno neko število (vneseš ga preko tipkovnice) pomnožiti z 1.1, da bo rezultat še vedno manjši kot 100?

Stavka break in countinue

Stavek **break** uporabljamo tudi v zankah: povzroči izstop iz (najbolj notranje) zanke tipa **for**, **while** ali **do while**.

Stavek **continue** pa ima nasprotno vlogo in se lahko pojavi samo v zankah. Pri zanki **for** izvede spremembo spremenljivk in skoči na začetek zanke. Pri ostalih dveh zankah pa skoči na pogoj zanke, ter sproži ponovno preverjanje pogoja. Če je ta še izpolnjen se izvajanje zanke nadaljuje, sicer pa se zanka zaključi.

Vaja:

```
double stevilo;
while (true)
{
    Console.Write("Vnesi poljubno število :");
    stevilo=Convert.ToDouble(Console.ReadLine());
```

```
    if (stevilo == 0.0)
          continue; //nazaj na začetek zanke
    Console.WriteLine("  Obratna vrednost števila "+stevilo+" je " + 1 / stevilo);
    break; //izstop iz zanke
}
```

Naloge:

- Preberi 5 stavkov. Če je v stavku manj kot 5 znakov, ga ne upoštevaj (stavek **continue**). Na koncu ugotovi in izpiši najdaljši stavek.

- Radi bi tabelirali funkcijo y = 10 sin(5x) na intervalu od 0 do .. Želimo izpisati le tiste točke, ki so po ordinatni vrednosti za 1 večje od sosednje točke. (nasvet: izberemo majhen korak in preračunamo vrednost funkcije. Če je izračunana vrednost premajhna glede na prejšnjo izračunano vrednost, s pomočjo ukaza **continue** ponovno izračunamo y v naslednji točki.

- Sestavi program, ki bo izpisal prvih n decimalk kvocienta a/b. Delaj samo s celimi števili.
  Primer:
  Vneseno število a: 10
  Vneseno število b: 761
  Vneseno število n : 40
  Rešitev: 0.0131406044678055190538764783180026281208

- Program, ki prebere naravno število in izpiše vse kvadrate do vključno prebranega.

- Napiši program, ki meče dve kocki toliko časa, da na obeh padeta šestici. Vsako kocko predstavljajo naključna števila med 1 in 6. Program naj tudi šteje, kolikokrat smo vrgli kocki, da smo dobili dvojno šestico.

- Sod drži 780 litrov. Začel je puščati, ker ga kletar ni popravil. Na minuto je izteklo 6 litrov vina. To je opazil šele čez pol ure. Koliko vina je še ostalo v sodu? Napiši še program, ki za vsako velikost soda izračuna in izpiše, koliko tekočine je izteklo iz njega, če je pogoj nespremenjen, torej izteče 6 litrov na minuto.

- Sestavi enostavni program, ki bo prebral vnešeno število ter izpisoval vsako drugo celo število do izbranega števila.

- Napiši program, ki najprej prebere naravni števili **n** in **m**, nato generira naključna števila med 1 in **m** toliko časa, da se pojavi število **n** in jih izpiše. Izpiše naj tudi število poskusov. Če vpisani podatki onemogočajo rešitev (če je **n > m**), naj se izpiše, da problem ni rešljiv.

- Napiši program, v katerega vnašaš števila in ti na koncu izpiše največjo in najmanšo vnešeno številko. Vnos se zaključi, ko uporabnik vnese število 0!

- Izpiši tista števila med celima številoma a in b (podatka, ki ju prebereš), ki so deljiva z vsoto svojih števk. Npr. 12 je že deljivo z vsoto števk (3). Prav tako 1101. Če sta torej podatka 8 in 14, naj program izpiše: Med 8 in 14 so z vsoto svojih števk deljiva naslednja števila: 8, 9, 10, 12.

Operatorji nad biti v C#

Operacije nad biti: OR(|), XOR(^), AND(&) in NOT(~)

C# pozna naslednje operatorje, ki delujejo nad biti:

- binarni OR(|) operator
- binarni AND(&) operator
- XOR (^) operator
- Not (~) operator

Binarni OR(|) operator

Binarni operator | (logični bitni OR) je namenjen številnim podatkovnim tipom in logičnim tipom. Pri številnih podatkovnih tipih je rezultat *false* samo v primeru, **da sta oba operatorja** *false*.

Primer:

```
byte a = 7;
byte b = 9;
int logicniOr = a | b;
Console.WriteLine(a+" | "+ b+" = "+ logicniOr); //Izpis: 7 | 9 = 15
```

Da bi razumeli, kako pridemo do takega rezultata, se spomnimo, da so podatki v pomnilniku zapisani v binarni obliki. Število 1 nam predstavlja vrednost *true*, število 0 pa vrednost *false*. Tule je bazična tabela, ki predstavlja binarni zapis potence števila 2 od 1 (2^0) do 128 (2^7). Ker je *byte* sestavljen iz 8 *bitov*, potrebujemo tabelo 8 elementov:

128	64	32	16	8	4	2	1	Rezultat
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	1	0	0	4
0	0	0	0	1	0	0	0	8
0	0	0	1	0	0	0	0	16
0	0	1	0	0	0	0	0	32
0	1	0	0	0	0	0	0	64
1	0	0	0	0	0	0	0	128

Če pogledamo tabelo bolj podrobno vidimo, kako se vsako naslednjo vrstico (vsako višjo potenco števila 2) enka premika po tabeli za eno mesto bolj v levo (»**bit shift**«).

Poglejmo sedaj, kako bi v binarni tabeli predstavili števili 7 in 9.

128	64	32	16	8	4	2	1	Izračun	Rezultat
0	0	0	0	0	1	1	1	(4+2+1)	7
0	0	0	0	1	0	0	1	(8+1)	9

Bitni **or** (|) deluje nad posameznimi biti. Ker v našem primeru noben par bitov ni hkrati *false* (vselej je vsaj eden enak 1, torej *true*) je rezultat takle

0	0	0	0	1	1	1	1	**(8+4+2+1)**	**15**

Vaja:

```
//številom med 0 in 10 zamenjajmo prvi bit (0 v 1, oz 1 v 0)

ushort stevilo;
ushort i;
for (i = 0; i <= 10; i++)
{
     stevilo = i;
     Console.WriteLine("Število: " + stevilo);
     stevilo = (ushort)(stevilo | 1); // stevilo | 0000 0001
     Console.WriteLine("Število potem, ko smo zamenjali prvi bit: "+ stevilo + "\n");
}
// Izpis:
Število: 0
Število potem, ko smo zamenjali prvi bit: 1
Število: 1
Število potem, ko smo zamenjali prvi bit: 1
. . .
Število: 10
Število potem, ko smo zamenjali prvi bit: 11
```

Binarni AND(&) operator

Binarni operator & (logični bitni AND) je namenjen številnim podatkovnim tipom in logičnim tipom. Pri številnih podatkovnih tipih je rezultat *true* samo v primeru, **da sta oba operatorja** *true*.

Primer:

```
byte a = 7;
byte b = 9;
int logicniIn = a & b;
Console.WriteLine(a+" & "+ b+" = "+ logicniIn); //Izpis: 7 & 9 = 1
```

Da bi razumeli, kako pridemo do takega rezultata, poglejmo še enkrat v tabelo. Ker bitni **in** (**&**) deluje nad posameznimi biti, v našem primeru pa je le en par bitov hkrati *true* (samo enkrat je par enak 1, torej *true*) je rezultat takle

128	64	32	16	8	4	2	1	Izračun	Rezultat
0	0	0	0	0	1	1	1	(4+2+1)	7
0	0	0	0	1	0	0	1	(8+1)	9
0	0	0	0	0	0	0	1	(1)	1

Vaja:

```
//S pomočjo operatorja & ugotovi, ali je neko število sodo ali liho

byte num;
num = 21;

if ((num & 1) == 1)
    Console.WriteLine(num + " je liho število");

Random naklj = new Random();
```

```
byte stevilo = (byte)naklj.Next(0, 100);//naključno celo število med 0 in 1000

if ((stevilo & 1) == 1)
    Console.WriteLine(stevilo + " je liho število");
else
    Console.WriteLine(stevilo + " je sodo število");
```

Bitni Xor (^) operator

Binarni operator & (logični bitni ekskluzivni OR) je namenjen številnim podatkovnim tipom in logičnim tipom. Pri številnih podatkovnih tipih je rezultat *true* samo v primeru, **da je natanko eden od operatorjev** *true*.

Primer:

```
byte a = 7;
byte b = 9;
int logicniXor = a ^ b;
Console.WriteLine(a+" ^ "+ b+" = "+ logicniXor); //Izpis: 7 & 9 = 14
```

Da bi razumeli, kako pridemo do takega rezultata, poglejmo še enkrat v tabelo. Ker bitni **ekskluzivni ali** (^) deluje nad posameznimi biti, v našem primeru pa je v dveh primerih natanko en od operatorjev enak *true* (dvakrat je natanko eden od parov enak 1, torej *true*) je rezultat takle

128	64	32	16	8	4	2	1	Izračun	Rezultat
0	0	0	0	0	1	1	1	(4+2+1)	7
0	0	0	0	1	0	0	1	(8+1)	9
0	0	0	0	0	0	0	1	(8+4+2)	14

Še en primer uporabe bitnega **XOR** operatorja nad spremenljivkami tipa *char*:

```
char ch1 = 'A';
char ch2 = 'B';
char ch3 = 'C';
int key = 88;

Console.WriteLine("Originalno sporočilo: " + ch1 + ch2 + ch3);//Izpis ABC

// kodirajmo sporočilo
ch1 = (char)(ch1 ^ key);
ch2 = (char)(ch2 ^ key);
ch3 = (char)(ch3 ^ key);

Console.WriteLine("Kodirano sporočilo: " + ch1 + ch2 + ch3);//Izpis ↓→←  (ASCII 25,26 in 27)

// dekodirajmo sporočilo, ki bo enako originalnemu!!!
ch1 = (char)(ch1 ^ key);
ch2 = (char)(ch2 ^ key);
ch3 = (char)(ch3 ^ key);

Console.WriteLine("Dekodirano sporočilo: " + ch1 + ch2 + ch3);//Izpis ABC
```

Bitni not (~) operator

Operator ~ (Bitni NOT) predstavlja komplement operanda, kar v praksi pomeni nasprotno vrednost vsakega bita posebej. Uporabljamo ga lahko nad spremenljivkami tipov *int*, *uint*, *long*, in *ulong*. Operator ~ je tudi **unarni** operator, tako da mu ni potrebno posredovati vrednosti (npr. *~n* ali pa *~a+b*).

Ker binarni operator ~ vrača komplement vsakemu bitu, je rezultat lahko pozitiven ali pa negativen (**negativen seveda le v primeru, da smo uporabili spremenljivko predznačenega tipa** – torej spremenljivko, ki lahko hrani negativno vrednost)

Primer:

```
byte a = 7;
byte b = 9;
int NOTMINUSa = ~-a;//bitni not negativnega števila a
int NOTa = ~a;
int rez=~a+b;

Console.WriteLine(" ~-" + a + " = " + NOTMINUSa);//Izpis: ~ -7 = 6
Console.WriteLine(" ~ " + a + " = " + NOTa);//Izpis: ~ 7 = -8
Console.WriteLine(" ~ " + a + " + " + b + " = " + rez);//Izpis: ~ a + b = 1
```

Poglejmo še, kako pridemo do takih rezultatov.

128	64	32	16	8	4	2	1	Izračun	Rezultat
0	0	0	0	0	1	1	1	(4+2+1)	7
1	1	1	1	1	0	0	0	**~a = -128+64+32+16+8+1**	-8

Pri spremenljivkah predznačenih tipov, je skrajni levi bit namenjen hranjenju predznaka (1= negativen predznak, 0 je pozitiven predznak). **Negativna števila pa so v C# interno predstavljena v dvojiškem komplementarnem formatu: vsak bit dobi najprej nasprotno vrednost, nato pa dodamo še 1.**

128	64	32	16	8	4	2	1	Izračun	Rezultat
0	0	0	0	0	1	1	1	(4+2+1)	7
1	1	1	1	1	0	0	0	**-a= (-128 +64+32+16+8) +1**	-7
0	0	0	0	0	1	1	1	**~-a = 4+2+1**	-8

Vaja:

```
//Program izpiše posamezne bite števila (oz. število izpišemo v binarnem zapisu), nato pa
//to število obrne z operatorjem ~ in ga ponovno izpiše!

sbyte b = 104;
//ispis posameznih bitov števila b
for(int t=128; t > 0; t = t/2)
{
   if((b & t) != 0)
     Console.Write("1 ");
   else
     Console.Write("0 ");
}
Console.WriteLine();

//obrnimo posamezne bite
b = (sbyte) ~b;
//ispis posameznih bitov števila ~b
for (int t = 128; t > 0; t = t / 2)
{
   if ((b & t) != 0)
     Console.Write("1 ");
   else
     Console.Write("0 ");
}
//Izpis:
0 1 1 0 1 0 0 0
```

```
1 0 0 1 0 1 1 1
```

Shift operatorja levi Shift (<<) in desni Shift (>>)

***Levi Shift* operator (<<)** premakne prvi operand v levo za tolikšno število bitov, kot jih navedemo v drugem operandu. Zaradi tega mora biti drugi operand obvezno tipa *int*.

```
int n = 1;
long lg = 1;
//izpis v desetiškem formatu
Console.WriteLine(n << 1);   //izpis: 2
Console.WriteLine(n << 33);  //izpis: 2
Console.WriteLine(lg << 33); //izpis: 8589934592
//POZOR: i<<1 in i<<33 nam data enak rezultat, ker imata 1 and 33 enakih spodnjih pet bitov.
//izpis v heksadecimalnem (šestnajstiškem) formatu
Console.WriteLine("0x{0:x}", n << 1);   //izpis: 0x2
Console.WriteLine("0x{0:x}", n << 33);  //izpis: 0x2
Console.WriteLine("0x{0:x}", lg << 33); //izpis: 0x200000000
```

***Desni Shift* operator (>>)** premakne prvi operand v desno za tolikšno število bitov, kot jih navedemo v drugem operandu. Zaradi tega mora biti drugi operand obvezno tipa *int*.

```
//DESNI Shift operator
int st = 5;//5= 00000101
Console.WriteLine(st >> 1);//Izpis 2, ker je st>>1=00000010=2

st = 16;//16= 00010000
Console.WriteLine(st >> 4);//Izpis 1, ker je st>>4=00000001=1
```

Vaja:

```
//Primer uporabe Shift operatorjev << in >>
Console.WriteLine("Operator <<");
int val = 1;
for (int i = 0; i < 8; i++)
{
   for (int t = 128; t > 0; t = t / 2)
   {
       if ((val & t) != 0) Console.Write("1 ");
       if ((val & t) == 0) Console.Write("0 ");
   }
   Console.WriteLine();
   val = val << 1; // levi shift
}
            /*Izpis:
            0 0 0 0 0 0 0 1
            0 0 0 0 0 0 1 0
            0 0 0 0 0 1 0 0
            0 0 0 0 1 0 0 0
            0 0 0 1 0 0 0 0
            0 0 1 0 0 0 0 0
            0 1 0 0 0 0 0 0
            1 0 0 0 0 0 0 0*/

Console.WriteLine();

Console.WriteLine("Operator <<");
val = 128;

for (int i = 0; i < 8; i++)
{
   for (int t = 128; t > 0; t = t / 2)
   {
       if ((val & t) != 0) Console.Write("1 ");
       if ((val & t) == 0) Console.Write("0 ");
```

```
    }
    Console.WriteLine();
    val = val >> 1; // desni shift
}
            /*Izpis:
            1 0 0 0 0 0 0 0
            0 1 0 0 0 0 0 0
            0 0 1 0 0 0 0 0
            0 0 0 1 0 0 0 0
            0 0 0 0 1 0 0 0
            0 0 0 0 0 1 0 0
            0 0 0 0 0 0 1 0
            0 0 0 1 0 0 0 1
           */
```

Funkcije (metode)

Funkcije uporabljamo zato, da se izognemo večkratnemu ponavljanju istih stavkov. V splošnem dobijo funkcije na vhodu neke parametre in na izhodu nekaj vrnejo.

Funkcija – klic parametrov po vrednosti

Splošna deklaracija funkcije :

```
<static> <tip podatka, ki ga funkcija vrne>   ime funkcije  (<parametri funkcije>)  //Glava funkcije
{
   ...TELO FUNKCIJE…

  return ...   //funkcija vrne neko vrednost. Stavka return ni, če funkcija tipa void
}
```

Primera glav funkcije :

static int sestej (int k, int m) //funkcija tipa int
*static void izpisi_pomoc(int n) // funkcija ne vrne ničesar (zato tip **void**)*

Razen v primeru, ko je funkcija tipa ***void*** (v tem primeru funkcija ne vrača ničesar), funkcija vedno vrača neko vrednost. Vrednost, ki jo funkcija vrača, mora biti enakega tipa, kot je tip funkcije. To pomeni, da če je funkcija npr. tipa *int*, potem mora tudi vrniti celoštevilsko vrednost. Funkcija vrača vrednost s pomočjo rezervirane besede ***return***.

Vse funkcije, ki jih bomo pisali v konzolnih aplikacijah morajo biti **statične (pred tipom funkcije je obvezna besedica static)**. Natančen pomen rezervirane besede **static** bomo spoznali v poglavju o razredih in objektih.

Primer:

```
//Funkcija, ki dobi za parameter dve celi števili in vrne večje izmed njih
static int vecje(int prvo, int drugo)
{
    if (prvo > drugo)
       return prvo;
    else
       return drugo;
}

//Glavni program
static void Main(string[] args)
{
    Console.Write("Prvo število: ");
    int st1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Drugo število: ");
    int st2 = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Večje izmed prebranih števi je " + vecje(st1, st2));//klic funkcije
}
```

Vaja:

```
/*Nariši romb iz samih zvezdic. Število zvezdic, ki tvorijo stranico preberi preko
tipkovnice*/
```

```
//Metoda
static void diamant(int n)
{
   int i, j;
   for( i=0; i<n; i++ )
   {
      for( j=n-i; j>0; j-- )
          Console.Write(" ");
      for( j=0; j<(2*i+1); j++ )
          Console.Write("*");
      Console.WriteLine();
   }

   for( j=0; j<2*n+1; j++ )
      Console.Write("*");
   Console.WriteLine();
   for( i=n-1; i>=0; i-- )
   {
      for( j=n-i; j>0; j-- )
          Console.Write(" ");
      for( j=0; j<(2*i+1); j++ )
          Console.Write("*");
      Console.WriteLine();
   }
}

//Glavni program
static void Main(string[] args)
{
   Console.Write("Vnesi stranico romba (=število zvezdic, ki tvorijo stranico): ");
   int stevilo=Convert.ToInt32(Console.ReadLine());

   if (stevilo>0) diamant(stevilo-1);
   else Console.WriteLine("Stranica ne more biti negativna!");
}
```

Vaja:

```
/*Napiši funkcijo ki na zaslon nariše trikotnik, sestavljen iz znakov *. Velikost trikotnika
določa nenegativno celo število n, ki je podano kot argument funkcije.

                  Primer:   n=0    *        n=1    *        n=1    *        n=3    *
                                             ***             ***             ***
                                                            *****           *****
                                                                           *********
*/

//Metoda
static void trikotnik(int n)
{
    int i, j;
    for( i=0; i<(n+1); i++ )
    {
       for( j=n-i; j>0; j-- )
           Console.Write(" ");
       for( j=0; j<(2*i+1); j++ )
           Console.Write("*");
       Console.WriteLine();
    }
}
//Glavi program
static void Main(string[] args)
{
    Console.Write("Velikost trikotnika: ");
    int velikost = Convert.ToInt32(Console.ReadLine());
    trikotnik(velikost); //klic funkcije
}
```

Vaja:

```
/* Napiši funkcijo, ki dobi dva parametra: poljuben STAVEK in poljuben ZNAK. Funkcija naj
ugotovi in vrne kolikokrat se v stavku pojavi izbrani znak    */


//Metoda
static int kolikokrat(string stavek, char znak)
{
    int skupaj = 0;
    for( int i=0; i<stavek.Length; i++ )
    {
       if (stavek[i] == znak)
           skupaj++;
    }
    return skupaj;
}
//Glavni program
static void Main(string[] args)
{
    Console.Write("Vnesi poljuben stavek: ");
    string stavek = Console.ReadLine();
    Console.Write("Vnesi znak, ki te zanima: ");
    char znak = Convert.ToChar(Console.Read());
    Console.WriteLine("V tem stavku je " + kolikokrat(stavek, znak)+" znakov "+znak+".");
}
```

Vaja:

```
/*Število PI lahko izračunamo tudi kot vsoto vrste 4 - 4/3 + 4/5 - 4/7 + 4/9 - ...  Napiši
funkcijo, ki ugotovi in vrne, koliko členov tega zaporedja moramo sešteti, da se bo tako
dobljena vsota ujemala s konstanto Math.PI do npr. vključno devete decimalke. Funkcija naj ima
za parameter število ujemajočih se decimalk. Rešitev za 9 členov: 1096634169*/

//Metoda
static long stClenov(int clenov)
{
    double pi=4;   //definiramo in inicializiramo začetno vrednost za pi
    double clen;   //tekoči člen zaporedja
    long i=1;      //definicija in inicializija števca členov
    while (Math.Round(pi,clenov)!=Math.Round(Math.PI,clenov))
    {
        clen=4.00/(i*2+1);  //izracun tekocega clena
        if(i%2!=0)
            pi-=clen;       //lihe člene odštejemo od pi, krajši zapis za pi=pi-clen
        else
            pi+=clen;       //sode člene prištejemo k pi, krajši zapis za pi=pi+clen
        i++;
    }
    return i;
}

//Glavni program
static void Main(string[] args)
{
    Console.WriteLine("Računam koliko členov zaporedja 4 - 4/3 + 4/5 - 4/7 + 4/9 - ... je
         potrebno\nsešteti, da bo tako dobljena vsota enaka konstanti Math.PI!");
    Console.WriteLine("\nTrenutek ......\n");
    Console.WriteLine("Število členov: " + stClenov(9)); //ujemanje na 9 decimalk
}
```

V vseh dosedanjih primerih so bili parametri, ki smo jih posredovali funkcijam, posredovani na privzeti način, to pa je **po vrednosti**. To pomeni, da je bila vrednost vsake spremenljivke posredovana ustreznemu parametru v metodi, ki je tako v resnici delala s kopijo originalne spremenljivke. Zaradi tega sprememba parametra v funkciji ni vplivala na velikost spremenljivke, ki smo jo navedli pri klicu funkcije oz. metode.

Funkcija – klic parametrov po referenci

Parametre v funkciji pa lahko kličemo tudi po **referenci**. V tem primeru dobi metoda le referenco na ustrezno spremenljivko, kar dejansko pomeni, da vsaka sprememba parametra v funkciji, pomeni spremembo vrednosti

spremenljivke, ki smo jo uporabili pri klicu metode. Klic po referenci dosežemo s pomočjo rezervirane besede **ref**. Vendar pozor: **besedico ref moramo napisati tako pred tipom parametra v glavi funkcije, kot tudi pred imenom spremenljivke pri klicu funkcije**.

Primer:

```
//Metoda
static void Methoda(ref string s) //parameter s je posredovan po referenci
{
    s = "Spremenjen!";
}

//Glavni program
static void Main()
{
    string stavek = "Originalen stavek";
    Method(ref stavek); //parameter stavek je klican po referenci
   // stavek je sedaj spremenjen
}
```

Vaja:

```
/*Zamenjava vrednosti dveh spremenljivk s pomočjo fukcije, v kateri sta parametra podana po
referenci*/

//Metoda
static void zamenjaj(ref int st1, ref int st2)//parametra podana po referenci
{
    int zacasna = st1;
    st1 = st2;
    st2 = zacasna;
}

//Glavni program
static void Main(string[] args)
{
    int stevilo1 = 200;
    int stevilo2 = 500;
    Console.WriteLine("Prvo število: "+stevilo1+",  drugo število: "+stevilo2);
    zamenjaj(ref stevilo1, ref stevilo2); //klic parametrov po referenci
    Console.WriteLine("Prvo število: " + stevilo1 + ",  drugo število: " + stevilo2);
}
```

Jezik **C#** pozna še en način klica parametrov po **referenci**, to je klic s pomočjo rezervirane besede **out**. Besedica **out** je sicer podobna besedici **ref**, razlika pa je v tem, da klic s pomočjo besedice **ref** zahteva, da je spremenljivka pred klicem že inicializirana. **Tudi pri klicu s pomočjo besedice ref velja, da jo moramo napisati tako pred tipom parametra v glavi funkcije, kot tudi pred imenom spremenljivke pri klicu funkcije**.

Primer:

```
//Metoda
static void Method(out int i)
{
    i = 44;
}
static void Main()
{
    int stevilo;  //spremenljivka še ni inicializirana
    Method(out stevilo);
     //vrednost spremenljivke je 44
}
```

Vaja:

```
static void Method(out int i, out string s1, out string s2) //Parameter klican po referenci
```

```
{
    i = 44;
    s1 = "Danes je lep dan!";
    s2 = null;
}

//Glavni program
static void Main()
{
    int stevilo;
    string str1, str2;
    Method(out stevilo, out str1, out str2);

    // stevilo =  44
    // str1 = Danes je lep dan!"
    // spremenljivka str2 je še vedno neinicilizirana. Njena vrednost je torej še vedno null;
}
```

Vaja:

```
//Napiši funkcijo, ki obrne niz znakov (npr »abeceda« pretvori v »adeceba«).

//Metoda
static void obrni(ref string stavek) //parameter stavek je klican po referenci
{
    string pomozni = "";
    for (int i = stavek.Length - 1; i >= 0; i--)
        pomozni = pomozni + stavek[i];
    stavek = pomozni;
}

//Glavni program
static void Main(string[] args)
{
    Console.Write("Stavek: ");
    string stavek = Console.ReadLine();
    Console.WriteLine("\n\nOriginalni stavek: \n\n" + stavek);
    obrni(ref stavek); //Parameter klican po referenci
    Console.WriteLine("\n\nObrnjeni stavek: \n\n"+stavek+"\n\n");
}
```

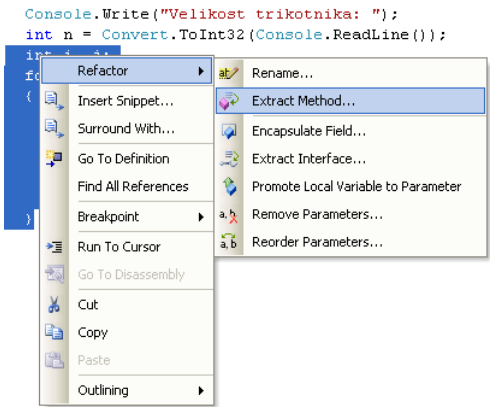
Kreiranje funkcije/metode s pomočjo čarovnika

C# omogoča pisanje funkcij tudi s pomočjo čarovnika. Kot primer vzemimo naslednji program:

```
static void Main(string[] args)
{
    Console.Write("Velikost trikotnika: ");
    int n = Convert.ToInt32(Console.ReadLine());

    int i, j;
    for (i = 0; i < (n + 1); i++)
    {
        for (j = n - i; j > 0; j--)
           Console.Write(" ");
        for (j = 0; j < (2 * i + 1); j++)
           Console.Write("*");
        Console.WriteLine();
    }
}
```

Tole kodo v glavnem programu bi radi nadomestili s funkcijo

Kodo, ki bi jo radi s pomočjo čarovnika zapisali v novo funkcijo, najprej označimo. Nato kliknemo desni miškin gumb in v oknu, ki se odpre izberemo opcijo **Rafactor** in nato **Extract Method..** V oknu, ki se odpre, nato še zapišemo ime funkcije (npr. trikotnik) in ime potrdimo s klikom na gumb **OK**. Čarovnik nam nato sam zgenerira ustrezno funkcijo.



V našem primeru bo rezultat takle:

```
//Glavi program
static void Main(string[] args)
{
     Console.Write("Velikost trikotnika: ");
     int n = Convert.ToInt32(Console.ReadLine());
     Trikotnik(n);   //KLIC nove metode
}

private static void Trikotnik(int n)
{
   int i, j;
   for (i = 0; i < (n + 1); i++)
   {
       for (j = n - i; j > 0; j--)
         Console.Write(" ");
       for (j = 0; j < (2 * i + 1); j++)
         Console.Write("*");
       Console.WriteLine();
   }
}
```

NOVA metoda

Naloge:

- Napiši funkcijo **sekunde**, ki sprejme tri cela števila : ure, minute in sekunde. Funkcija naj izračuna, koliko je to sekund in vrne rezultat.
- Napiši funkcijo, ki izpiše vsa števil med 0 in 1000, katerih vsota števk je enaka številu, ki nastopa kot parameter te funkcije.
- Napiši funkcijo, ki dobi za parameter poljubno decimalno število in ki izračuna in vrne vsoto vseh cifer v tem številu!
- Sestavi metodo ***kopije(String s, int k),*** ki sprejme niz ***s*** in pozitivno celo število ***k*** ter vrne niz, ki je sestavljen iz ***k*** kopij niza ***s***.
- Napiši metodi **zakodiraj** in **odkodiraj**, ki sprejmeta niz in vrneta zakodiran oz. odkodiran niz. Metoda **zakodiraj** naj vsako črko v tekstu zamenja s črko, ki v (angleški) abecedi leži 5 črk za originalno. Menjava je seveda ciklična, tako da črko a zamenjata s črko e, črko z pa s črko d. Upoštevaj, da so črke lahko male ali pa velike. Seveda naj funkcija ločil in ostalih znakov ne kodira!

- Napiši metodo, ki sestavi posebno enostavno križanko in jo izpiše. To pomeni, da uporabnik vnese prvo besedo s poljubnim številom črk in jo napiše navpično navzol. Potem ga program za vsako črko te besede vpraša za besedo, ki se začne na to črko in je dolga 4 črke in jo zapiše vodoravno od tiste črke.

- Napiši metodo **izpisiPiramido**, ki sprejme pozitivno celo število n in na zaslon izriše piramido iz zvezdic, visoko n zvezdic. Na primer za n = 6 naj se izriše

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

- Napiši metodo **donos**, ki ima tri parametre: **cas**, **znesek** in **obresti**. **Cas**: koliko let se denar obrestuje. **Znesek**: koliko denarja položimo na bančni račun. **Obresti**: koliko obresti (v %)se pripiše glavnici vsako leto. Uporabi obrestno obrestni račun in metodo preizkusi.

- Sestavi metodo s pomočjo katere boš izračunal produkt vseh sodih števil med dvema danima pozitivnima celima številoma.

- Napiši metodo, ki sprejme pozitivno celo število **n** in na zaslon izpiše naslednji vzorec: najprej se n-krat izpiše .(pika) in nato n-krat znak #. Prikazan je primer za n = 5:

```
.....#####
```

Sklad in kopica

Pomnilnik, ki je namenjen spremenljivkam, je razdeljen na dva dela: **kopica** (*heap*) in **sklad** (*stack*).

V **sklad** se shranjujejo t.i. **vrednostne** spremenljivke (**value type**). To so spremenljivke tipa **bool**, char, **int**, **float**, double, **decimal**, **struct**, **enum** in spremenljivke, ki se tvorijo ob klicu funkcije. Na skladu se hranijo tudi parametri funkcij, ki se obnašajo tako kot spremenljivke, ki se tvorijo ob klicu funkcij. Pri klicu parametrov ***po vrednosti***, se spremenljivka, ki nastopa kot parameter funkcije prepiše (prekopira) na sklad in v funkciji delamo v bistvu z njeno kopijo. Ob klicih funkcij se torej sklad povečuje, ob zaključku izvajanja funkcij pa se zmanjšuje. Za ponazoritev sklada lahko vzamemo tudi nekaj primerov iz narave: skladovnica knjig (ena knjiga na drugi), PEZ bonboni, … Elemente vstavljamo (**push**) in tudi jemljemo (**pop**) s sklada na enem (istem) koncu. Njihova značilnost je dejstvo, da gre element, ki ga vstavimo v tak sklad prvega ven zadnji. Tak princip imenujemo **LIFO** (**Last In First Out**). Primeri implementacije sklada v računalništvu pa so npr.: obiskane strani v brskalniku, zaporedje UNDO ukazov, rekurzija, …

Sklad običajno imenujemo tudi del pomnilnika, kjer so shranjeni podatki. Običajno dostopamo samo do zadnjega, vrhnjega podatka. Najbolj značilni operaciji za sklad sta

- Postavi na sklad (**push**)
- Vzemi s sklada (**pop**)

V **kopico** pa se zlagajo t.i. **referenčne** spremenljivke (**reference type**), to je spremenljivke ki imajo poleg svoje vrednosti tudi referenco oz kazalec nanjo. To so npr. vse tabelarične, vsi iz razredov izpeljani objekti, … Dva podatka na kopici imata torej lahko isto referenco in operacija na referenci lahko vpliva na več podatkov. To so v bistvu spremenljivke, ki jih tvorimo izrecno. Kopica je torej namenjena vsem dinamično kreiranim spremenljivkam, ki jih tvorimo z ukazom ***new***. **Vse take spremenljivke se samodejno inicializirajo** (takoj, ko se pojavijo, dobijo neko začetno vrednost – pri tabeli celih števil je ta začetna vrednost npr. enaka 0). Značilnost kopice je, da je to podatkovna struktura v katero vstavljamo podatke na repu, brišemo oz. jemljemo pa jih s čela. Zanjo velja princip **FIFO** (**First In First Out**). Spremenljivkam, ki jih ustvarimo s pomočjo operatorja **new** pravimo tudi **objekti**.

Pomen sklada lahko ponazorimo tudi pri funkcijah: pri klicu parametrov ***po referenci*** pa se spremenljivka, ki nastopa kot parameter funkcije prenese v funkcijo in v funkciji dejansko delamo prav s to spremenljivko. Sam prenos je v tem primeru realiziran tako, da se tokrat na sklad prenese **referenca** na spremenljivko in funkcija potem ve, da ima opravka z referenco, ne pa z vrednostjo.

Tabele - POLJA

Potreba po uporabi tabel (oziroma polj) in tabelaričnih spremenljivk se pokaže tedaj, ko imamo opravka z večjim ali velikim število spremenljivk istega tipa. To so npr. členi matematičnega zaporedja, rezultati nekih meritev, vzorčenje, … Tabelarične spremenljivke imajo vse enako ime (npr. tabela), razlikujejo pa se po **indeksu**, zaradi česar vsako od njih obravnavamo kot samostojno spremenljivko.

Deklaracija tabel

Vse spremenljivke, ki smo jih obravnavali doslej so zajemale pomnilniški prostor na **skladu** (**stack**). Ker pa pri programiranju pogosto ne vemo koliko prostora bomo potrebovali, posežemo po dinamičnem zaseganju pomnilnika. Pomnilnik zasežemo z ukazom "**new**" in ga sprostimo z ukazom "**delete**". Že v prejšnjem poglavju smo ugotovili, da se vse dinamično kreirane spremenljivke (objekti) nahajajo v delu pomnilnika, ki ga imenujemo kopica (**heap**). Tudi tabele tvorimo v **C#** dinamično, s pomočjo rezervirane besede **new**, to pa zaradi tega, ker so tabele pravzaprav izpeljane iz razreda **Array**, ki je sestavni del .NET platforme. Ob kreiranju tabele torej pravzaprav naredimo nov objekt razreda **Array** (več o razredih in objektih bomo izvedeli v poglavju **Razredi in objekti**).

Ker vsako tabelo kreiramo s pomočjo operatorja **new** (dinamično), so vse **tabelarične spremenljivke samodejno inicializirane.**

Splošna deklaracija enodimenzionalne tabele:

```
Podatkovni_tip[] ime_tabele = new Podatkovni_tip [dimenzija];
```

Tabelo lahko deklariramo na tri načine:

- Najprej deklariramo tabelo (npr z imenom **tabela)** , kasneje pa ji določimo še velikost:

```
int[] tabela;  //deklaracija tabelarične spremenljivke
tabela = new int[100]; //zaseganje pomnilnika za 100 celih števil
```

- Ob deklaraciji tabelarične spremenljivke z operatorjem **new** takoj zasežemo pomnilnik:

```
int[] tabela = new int[100];
```

- Tabelo najprej deklariramo, z operatorjem **new** zasežemo pomnilnik, nato pa jo še inicializirajmo

```
int[] tabela = new int[5] { 1, 2, 3, 4, 5 };
```

Tudi v prvih dveh primerih se spremenljivke, (deklarirane so na kopici) samodejno inicializirajo (v našem primeru ima vseh 100 elementov tabele vrednost 0.

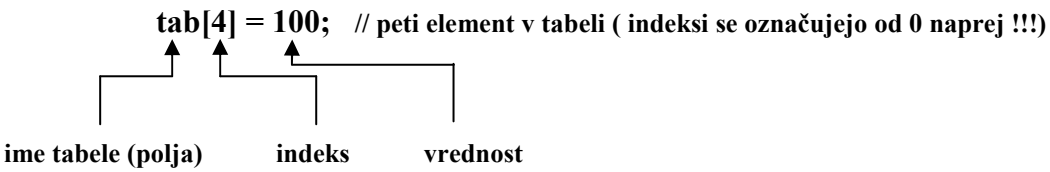
V vseh treh primerih smo deklarirali **enodimenzionalno** tabelo. Obstajajo tudi dvo in večdimenzionalne tabele, ki pa jih bomo obravnavali v nadaljevanju.

Grafično si lahko enodimenzionalno tabelo predstavljamo takole:

```
int[] tab = new int[5];//ime tabelarične spremenljivke je tab
```

Tabela tab

tab[0]	tab[1]	tab[2]	tab[3]	tab[4]



Vsaka tabelarična spremenljivka ima svoj **indeks**, preko katerega ji določimo vrednost (jo inicializiramo) ali pa jo uporabljamo tako kot običajno spremenljivko. Začetni indeks je enak nič (0) končni indeks pa je za ena manjši kot je dimenzija tabele (dimenzija – 1).
Posameznim tabelaričnim spremenljivkam (komponentam) lahko prirejamo vrednost, ali pa jih uporabljamo v izrazih, npr.:

```
tab[0] = 100;
tab[1] = tab[0]-20;      // tab[1] dobi vrednost 80
tab[2] = 300;
tab[3] = 400;
tab[4] = tab[1]+tab[3]; // tab[4] dobi vrednost 500
```

Izgled tabele po izvedbi zgornjih stavkov je torej takle:

100	80	300	400	380

Primer:

Deklarirajmo tabelo 100 decimalnih števil in jo inicializirajmo tako, da bodo imeli vsi elementi tabele vrednost 1:

```
decimal [] tabela=new decimal[100];

for (int i=0;i<100;i++)
{
    tabela[i] = 1; //ker je tabela že inicializirana oznaka M( ali m) za decimalno število ni
                   //več potrebna
}
```

Vaja:

```
//Deklariraj tabelo sedmih strigov in jo inicializiraj tako, da bo vsebovala dneve v tednu.
//Tabelo nato še izpiši, vsak element tabele v svojo vrsto.
string[] dneviVTednu = new string[7] {"Ponedeljek", "Torek", "Sreda", "Četrtek", "Petek",
                                      "Sobota", "Nedelja" };

for (int i=0;i<7;i++)
{
     Console.WriteLine(dneviVTednu[i]);
}
```

Vaja:

```
//Primer enodimenzionalne tabele tipa double
double [] meseci=new double[12];
double letnoPovp=0;

for (int i=0;i<12;i++)
{
     Console.Write("Povprečna temperatura za " + (i + 1) + ". mesec : ");
      meseci[i]=Convert.ToDouble(Console.ReadLine());
      letnoPovp=letnoPovp+meseci[i];
}
Console.WriteLine("Povprečna letna temperatura : "+Math.Round(letnoPovp/12,2));
```

Vaja:

```
//Preberi poljuben stavek in gotovi ter izpiši koliko znakov vsebuje, koliko je v njem
samoglasnikov, koliko cifer in koliko ostalih znakov
char [] samogl= new char [5] {'A','E','I','O','U'}; //deklaracija in inicializacija tabele
                                                    //samoglasnikov
int stsam=0,stcifer=0;
string stavek;

Console.Write("Vnesi poljuben stavek: ");
stavek=Console.ReadLine();
Console.WriteLine();

for (int i=0;i<stavek.Length;i++)
{   //funkcija strlen vrne dolžino stavka
     for (int j=0;j<5;j++)
     {
         char crka=Convert.ToChar(stavek[i]);
         if (char.ToUpper(crka)==samogl[j])
            stsam=stsam+1;
     }
     if (stavek[i]>='0'&& stavek[i]<='9')
         stcifer=stcifer+1;

}
Console.WriteLine("\nŠtevilo vseh znakov v stavku : "+stavek.Length);
Console.WriteLine("Število samoglasnikov        : "+stsam);
Console.WriteLine("Število cifer                : "+stcifer);
Console.WriteLine("Število ostalih znakov       : "+(stavek.Length-stsam-stcifer));
```

Bistvena prednost takega načina kreiranja tabel je, da velikost tabele lahko določimo med samim delovanjem programa. Dimenzijo tabele torej lahko določi uporabnik, glede na svoje potrebe:

Primer:

```
Console.WriteLine("Določi dimenzijo tabele: ");
int dimenzija = Convert.ToInt32(Console.ReadLine());//Dimenzijo tabele določi uporabnik!!!

int[] tabela = new int[dimenzija];
```

Vaja:

```
/*Vsaka premica je podana z enačbo y = k * x + n, kjer je k smerni koeficient premice pa
prosti člen. napiši program, ki zahteva vnos smernega koeficienta in prostega člena, nato pa
to funkcijo tabeliraj na poljubnem intervalu s korakom 1!
Napiši funkcijo, ki ugotovi in izpiše presečišči te premice z obema osema */

//Metoda
static void presecisci(int k, int n)
{
   if (n == 0)
      Console.WriteLine("Premica gre skozi koordinatno izhodišče!");
   else if (k == 0)
      Console.WriteLine("Premica je vzporena x osi! y os seka v točki N(0," + n + ")");
   else
   {
```

```
     //presečišče z x osjo ima drugo koordinato enako 0
     double presx = Math.Round((double)n / k, 2);//zaokroženo na dve decimalki
     Console.WriteLine("Presečišče z x osjo: M( " + presx + " , 0 )");
     //presečišče z y osjo ima prvo koordinato enako 0
     Console.WriteLine("Presečišče z y osjo: N( 0 , " + n+" )");
   }
}

//Glavni program
static void Main(string[] args)
{
     Console.Write("Smerni koeficient premice (celo število): ");
     int k = Convert.ToInt32(Console.ReadLine());
     Console.Write("Prosti člen (celo število): ");
     int n = Convert.ToInt32(Console.ReadLine());
     Console.WriteLine("Enačba premice: y = " + k + " * x + " + n);
     Console.WriteLine("Vnesi interval, na katerem želiš tabelirati to funkcijo!");
     Console.Write("Levi rob intervala: ");
     int levi = Convert.ToInt32(Console.ReadLine());
     Console.Write("Desni rob intervala: ");
     int desni = Convert.ToInt32(Console.ReadLine());
     if (levi > desni)
          Console.WriteLine("Tabeliranje ni možno, meji intervala sta napačni!");
     else
     {
         //vrednosti koordinat bomo shranili v enodimenzionalno tabelo celih števil
          int korakov = desni - levi + 1; //tabelirali bomo tudi v obeh mejnih točkah
           //zgeneriram tako veliko tabelo, kot jo potrebujem
          int[,] tabela = new int[korakov, 2];
          for (int i = 0; i < korakov; i++)
          {
               tabela[i, 0] = levi;
               tabela[i, 1] = k * levi + n;
               levi++;
          }
          Console.WriteLine("\n\nIzpis tabele: \n");
          Console.WriteLine("     x |    y");
          Console.WriteLine("-------------");
          for (int i = 0; i < korakov; i++)
               Console.WriteLine("{0,6} |{1,5}", tabela[i, 0], tabela[i, 1]);
          presecisci(k, n);
     }
}
```

Dvodimenzionalne in večdimenzionalne tabele

Dvodimenzionalne tabele vsebujejo vrstice in stolpce.

Splošna deklaracija dvodimenzionalne tabele:

```
Podatkovni_tip[ , ] ime_tabele = new Podatkovni_tip [vrstic, stolpcev];
```

Tudi dvodimenzionalno tabelo lahko deklariramo na tri načine:

- Najprej deklariramo tabelo (npr z imenom **tabela)** , kasneje pa ji določimo še velikost:

```
int[,] tabela;  //deklaracija tabelarične spremenljivke
tabela = new int[100 , 200]; //dvodimenzionalna tabela 100 vrstic in 200 stolpcev
```

- Ob deklaraciji tabelarične spremenljivke z operatorjem **new** takoj zasežemo pomnilnik:

```
int[ , ] tabela = new int[100 , 200];
```

- Tabelo najprej deklariramo, z operatorjem **new** zasežemo pomnilnik, nato pa jo še inicializiramo

```
int[ , ] tabela = new int[ 2, 3]  { { 1, 1 ,1 },{2, 2, 2 }};
```

Tudi v prvih dveh primerih se spremenljivke, (deklarirane so na kopici) samodejno inicializirajo (v našem primeru ima vseh 100 x 200 (to je 20.000) elementov tabele vrednost 0.

Primer:

Deklariraj dvodimenzionalno tabelo 10 x 10 celih števil in jo inicializiraj tako, da bodo po diagonali same enice, nad diagonalo same dvojke, pod diagonalo pa ničle.

```
int[,] tabela=new int[10,10];  //deklaracija tabelarične spremenljivke
//ob deklaraciji se je avtomatični izvedla še inicializacija: vse tabelarične spremenljivke so
//dobile vrednost 0

for (int i=0;i<10;i++)
{
    for (int j = 0; j < 10; j++)
    {
        if (i == j)
             tabela[i, j] = 1;
        else if (i<j)
             tabela[i, j] = 2;
    }
}
//tabelo še izpišimo v primerni obliki
for (int i = 0; i < 10; i++)
{
     for (int j = 0; j < 10; j++)
         Console.Write(tabela[i, j] + "  ");
     Console.WriteLine();
}
```

Vaja:

```
/*Kreiraj dvodimenzionalno tabelo 10 x 10 celih števil. V posamezni vrstici naj bo poštevanka
števil med 1 in 10. Tabelo na primeren način tudi izpiši.*/

int [,] tabela=new int[10,10];
for (int i=0;i<10;i++)
    for (int j=0;j<10;j++)
        tabela[i,j]=(i+1)*(j+1);

//Še izpis tabele
for (int i=0;i<10;i++)
{
    for (int j=0;j<10;j++)
        Console.Write("{0,4}",tabela[i,j]);
    Console.WriteLine();
}
```

Razred Random

V programih se večkrat pojavi potreba po generiranju oz. uporabi naključnih števil. V **C#** je naključnim številom namenjen razred **Random**. Njegove metode omogočajo generiranje naključnih celih števil v poljubnem obsegu in iz poljubnega intervala, pa tudi generiranje naključnih števil tipa **double**.

Če hočemo delati z naključnimi števili, moramo najprej s pomočjo rezervirane besede **new** ustvariti nov **objekt** za generiranje naključnih števil:

```
Random nakljucno = new Random();//generiranje objekta nakljucno za generiranje naklj.števil
```

Obekt **nakljucno** sedaj lahko uporabimo za generiranje naključnih števil. Za samo generiranje imamo na voljo dve metodi razreda **Random:**

Metoda	Razlaga
Next	Naključno celo število
NextDouble	Naključno število iz intervala 0.0 in 1.0

Primer uporabe:

```
Random nakljucno = new Random();

int naklj = nakljucno.Next();//naključno nenegativno število

int naklj1 = nakljucno.Next(5); //naključno število med vključno 0 in vključno 4

int naklj2 = nakljucno.Next(10, 20);  //naključno število med vključno 10 in vključno 19

double naklj3 = nakljucno.NextDouble(); //naklj.število tipa double na intervalu od 0.0 do 1.0

double naklj4 = nakljucno.NextDouble() * 1000; //naklučno število tipa double na intervalu od
                                               //0.0 do 1000
```

Vaja:

```
//Generirajmo nekaj naključnih celih in naključnih realnih števil
Random nakljucno = new Random();

//Generiramo 10 naključnih celih števil, večjih od 0 in MANJŠIH od 5
for (int j = 0; j < 10; j++)
     Console.Write(" {0,5} ", nakljucno.Next(5)); //izpis formatiran na 3 mesta
Console.WriteLine();

//Generirajmo 6 naključnih realnih števil
for (int j = 0; j < 6; j++)
    Console.Write(" {0,5:F3} ", nakljucno.NextDouble());//izpis na 5 mest, formatiran na 3
                                                        //decimalke
```

Vaja:

```
/*Kreiraj naključne elemente kvadratne matrike (4x4) celih števil med 0 in 10 in nato izpiši
največjo vrednost v tabeli. Ugotovi in izpiši, na katerih mestih se ta največji element pojavi
v tabeli (indeksi!).*/

int[,] mat = new int[4, 4];
int i, j, max = 0;

//generator naključnih vrednosti
Random Nakljucno = new Random();
for (i = 0; i < 4; i++)
{
    for (j = 0; j < 4; j++)
    {
         mat[i, j] = Nakljucno.Next(10);
         if ((i == 0) && (j == 0)) max = mat[i, j]; //iskanje najvecjega
         if (max < mat[i, j]) max = mat[i, j];
    }
}

Console.WriteLine("TABELA 4 x 4\n");
for (i = 0; i < 4; i++)
{ //izpis matrike
    for (j = 0; j < 4; j++)
    Console.Write(mat[i, j] + " ");
    Console.WriteLine();
}
Console.WriteLine("\nNajvecji element: {0}\n", max);
Console.WriteLine("Indeksi najvecjega elementa: ");
for (i = 0; i < 4; i++)  //izpis pozicij/indeksov najvecjega elementa
    for (j = 0; j < 4; j++)
```

```
        if (mat[i, j] == max) Console.WriteLine("[ {0} , {1} ]", i, j);
```

Vaja:

```
/*Deklariraj dvodimenzionalno tabelo celih števil dimenzije 20x20 in jo napolni po pravilu: na
diagonali naj bodo zaporedoma mnogokratniki števila 2 (2,4,6,…), pod diagonalo naj bodo taka
števila kot so na diagonali, nad diagonalo pa naj bodo naključna naravna števila med -5 in +5!

Napiši še funkcijo, ki dobi za parameter poljubno število  N in ki vrne podatek o tem, koliko
števil v zgornji tabeli je enakih N*/

//funkcija ima dva za drugi parameter dvodimenzionalno tabelo
static int enakihN(int N,int [,] stevila)
{
   int skupaj = 0;
   for (int i = 0; i < 20; i++)
      for (int j = 0; j < 20; j++)
         if (stevila[i,j] == N) skupaj++;
   return skupaj;
}

//glavni program
static void Main(string[] args)
{
     Random naklj = new Random();
     int[,] stevila = new int[20, 20];
     for (int i = 0; i < 20; i++)
         for (int j = 0; j < 20; j++)
             if (i == j) stevila[i, j] = (i + 1) * 2;
             else if (i > j) stevila[i,j] = stevila[j,j];
             else stevila[i, j] = naklj.Next(6) - 5;
     for (int i = 0; i < 20; i++)
     {
         for (int j = 0; j < 20; j++)
             Console.Write("{0,3}", stevila[i,j]); //izpis formatiran na 3 mesta
         Console.WriteLine();
     }
     Console.WriteLine("\n\nV tej tabeli je {0} števil ekakih 6!", enakihN(6,stevila));
}
```

Če tabelo ob deklaraciji tudi inicializiramo, je možna tudi deklaracija na krajši način:

```
Podatkovni_tip[] ime_tabele = {elementi tabele ločeni z vejico}];
```

Primer:

```
int[] tab = { 0,1,2,3,4,5};//deklaracija in inicializacija enodimenzionalne tabele 6 elementov
```

Pri tabelaričnih spremenljivkah oz. pri tabelah obstajata dve metodi, s pomočjo katerih lahko ugotovimo število elementov v tabeli in dimenzijo tabele:

Metoda	Razlaga
Length	Število elementov v tabeli
Rank	Dimenzija tabele

Primer:

```
Random naklj = new Random();
int dimenzija = naklj.Next(10000)+1; //dimenzija tabele je naključno število med 1 in 10000

int[] tabela = new int[dimenzija];
```

```
for (int i = 0; i < dimenzija; i++)
   tabela[i] = naklj.Next(1000);//v tabeli so naključna števila med 0 in 1000

Console.WriteLine("V tej tabeli je " + tabela.Length + " elementov, dimenzija tabele pa je " +
        tabela.Rank);
```

Naloge:

- Iz treh naključnih števil med 0 in 9 sestavi naključno tromestno število med 0 in 999!
- Ustvari naključno tabelo 100 celih števil z vrednostmi med 10 in 100 in izpiši le tiste člene, ki so večji od zadnjega elementa v tej tabeli. Program dopolni tako, da ustvariš naključno dolgo tabelo (a ne krajšo od 10 in ne daljšo od 100 elementov).
- Zgeneriraj tabelo naključnih naravnih števil in izpiši vse elemente, ki so deljivi s številom, ki je na sredini tabele. Indeks elementa na sredini tabele je definiran kot (dolzina tabele) / 2.
- Sestavi program, ki bo uporabniku zastavil 10 računov za množenje celih števil. Eden izmed faktorjev naj bo naključno trimestno celo število, drugi izmed faktorjev pa mora biti naključno enomestno število. Števila in uporabnikove rezultate shranjuj v ustrezno tabelo. Tabelo NA KONCU obdelaj (preveri rezultate) in izpiši število pravilnih in število nepravilnih odgovorov.
- Napiši program, ki prebere tri cela števila: *dimenzija*, *spMeja* in *zgMeja*. Program naj nato tabelo velikosti *dimenzija* napolni z naključnimi števili med *spMeja* in *zgMeja*.
- Beri besede in jih shranjuj v tabelo dimenzije do 100 elementov. Ugotovi in izpiši najdaljšo besedo v tej tabeli. Napiši funkcijo, ki dobi za parameter to tabelo in poljubno celo število *N* in ki vrne podatek o tem, koliko besed v tej tabeli vsebuje več kot N znakov!
- Napiši program, ki bo izpisal dvodimenzionalno tabelo naključnih celih števil med 0 in 100 (dimenziji izbere uporabnik programa), poiskal njen največji element ter izračunal, za koliko se največji element razlikuje od povprečne vrednosti vseh elementov.
- Napiši program, ki prešteje število sodih in lihih elementov v naključno generirani tabeli celih števil. Nato iz začetne tabele sestavi dve tabeli. V prvi so samo soda števila, v drugi pa samo liha.
- Sestavi tabelo naključnih celih števil med 1 in 100. Prepiši jih v novo tabelo tako, da bodo v novi tabeli najprej elementi, ki imajo v prvi tabeli indekse 0, 2, 4, ..., potem pa še elementi z lihimi indeksi. Primer: Če ime prvotna tabela elemente 2, 4, 23, 5, 45, 6, 8 so v novi tabeli elementi razporejeni kot 2, 23, 45, 8, 4, 5, 6. 3. Izpiši obe tabeli po 10 v vrsto. Za realizacijo naloge napiši metode: **generiraj** - ustvari tabelo, **prelozi** - preloži v novo tabelo na zahtevan način in **izpisi** - v vsaki vrsti se izpiše 10 elementov
- Dan je zelo dolg niz, sestavljen iz števk. Ugotovi, katerih števk je v nizu največ.

Garbage Collector

Novo **instanco** (nov objekt) nekega razreda kreiramo s pomočjo rezervirane besede **new** (to smo spoznali že pri učenju osnov jezika **C#** oz. **C++** - poglavje razredi in objekti). Za sproščanje pomnilnika, ki ga zasedejo objekti kreirani z operatorjem **new** nam v okolju **Visual Studio .NET** ni potrebno skrbeti. **.NET Platforma** (**Framework**) uporablja za ta namen proces imenovan **garbage collector**, ki avtomatsko sprošča objekte, ki niso več v uporabi in s tem skrbi za upravljanje s pomnilnikom. V večini primerov se ta proces izvede avtomatično in se ga zelo redko sploh zavedamo. Ko sistem **garbage collector** zazna, da sistemu začne primanjkovati pomnilnika, oz. da je nezaposlen, sprosti pomnilnik vseh tistih objektov, ki niso več v funkciji. Preden jih počisti iz pomnilnika pa poskrbi, da se izvede metoda **Finalize** za vsak tak objekt, pa četudi je metoda privzeta, torej prazna (nismo napisali svoje kode, kaj naj se zgodi ob uničenju določenega objekta).

Zanka foreach

Zanka **foreach** je zanka, v kateri se ponavlja množica stavkov za vsak element neke tabele ali množice objektov. Zanke ne moremo uporabiti za spremembo elementov tabele, po kateri se sprehajamo, oz. za spremembo objektov. Na začetku **foreach** zanke je deklarirana spremenljivka poljubnega tipa, ki avtomatično pridobi vrednost posameznega elementa neke tabele, zaradi česar ima ta oblika zanke prednost pred klasičnim **for** stavkom za obdelavo neke tabele. Uporabimo jo lahko le za obdelavo cele tabele, ne pa tudi za obdelavo le njenega dela. Iteracija poteka vedno od indeksa **0** do indeksa **Length -1**, iteracija v obratni smeri pa **NI** možna.

Splošna oblika **foreach** zanke:

```
foreach ( tip spremenljivka in tabela)
{
  ..stavki..  //poljuben stavek ali več stavkov
}
```

Primer:

```
string znaki = "abcdefg";

//foreach zanka za dostop do vseh znakov v stringu
string znakiSPresledki = "";

foreach (char znak in znaki)
    znakiSPresledki += znak + " ";
//znakiSPresledki dobi vrednost "a b c d e f g "
```

Še en primer:

```
//deklaracija in inicializija enodimenzionalne tabele celih števil
int[] tab = { 0,1,2,3,4,5,6,7,8,9,10 };

Console.WriteLine("\nTabela tab vsebuje "+tab.Length+" elementov. \nTa tabela je "+tab.Rank+".
dimenzionalna");

Console.WriteLine(vsebina tabele: ");

foreach(int x in  tab)
    Console.Write(x+", "); //Izpis vsebine tabele
```

Vaja:

```
/*Deklariraj enodimenzionalno tabelo 10 decimalnih števil in jo napolni z naključnimi
decimalnimi števili med 0 in 1, zaokroženimi na 3 decimalke. Vsebino tabele nato izpiši s
pomočjo zanke foreach!*/

decimal[] stevila = new decimal[10];
string stringstevilo = "";
Random naklj = new Random();

for (int i = 0; i < 10; i++)
{
    //generiranje realnega števila med 0 in 1, zaokroženega na 3 decimalke
    stevila[i] = Math.Round((decimal)naklj.NextDouble(), 3);
}
Console.WriteLine("Tabela 10 naklj. realnih števil. Za izpis uporabimo foreach zanko)\n\n");

foreach (decimal stevilo in stevila)
{
    stringstevilo += stevilo.ToString() + "   ";
}
```

```
Console.WriteLine("Vsebina tabele:\n"+stringstevilo);
```

Vaja:

```
/*Deklariraj tabelo 7 stringov in jo ob deklaraciji inicializiraj tako, da bodo v njej dnevi v
tednu. Napiši metodo izpisiTabelo(dneviVTednu), ki dobi za parameter to tabelo in ki s pomočjo
zanke foreach to tabelo izpiše*/

//Metoda
static void izpisiTabelo(string[] tabela)
{
    foreach (string dan in tabela)
        Console.Write( dan+", ");

    Console.WriteLine();
}

//Glavni program
static void Main(string[] args)
{
     // Deklaracija in inicilaizacija tabele
     string[] dneviVTednu = new string[] { "Ned", "Pon", "Tor", "Sre", "Čet", "Pet", "Sob" };

     //Funkcija dobi za parameter tabelo:
     izpisiTabelo(dneviVTednu);
}
```

Vaja:

```
/*Generiraj dvodimenzionalno tabelo 3 x 5 celih števil in jo napolni za naključnimi celimi
števili med 0 in 100. Vsebino tabele nato izpiši s pomočjo zanke foreach. Ugotovi in izpiši
vsoto vseh elementov.*/

int sum = 0;
int[,] tab = new int[3, 5]; //Dvodimenzionalna tabela

Random naklj = new Random();

//inicializacija tabele
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 5; j++)
        tab[i, j] = naklj.Next(100); //Naključna števila med 0 in vključno 99

Console.WriteLine("Izpis vseh elementov dvodimenzionalne tabele s pomočjo zanke foreach!\n");
// zanka foreach za izpis elementov dvodim. tabele
foreach (int stevilo in tab)
{
    Console.Write(stevilo+" ");
    sum += stevilo; //števila sproti seštevamo
}
Console.WriteLine("\n\nSkupna vsota elementov: " + sum+"\n\n");
```

Vaja:

```
/*generiraj tabelo 50 naključnih celih med vključno 0 in 100. S pomočjo foreach zanke
ugotovi in izpiši, katerih števil med 0 in 100 NI v tej tabeli*/

int[] stevila = new int[50];
Random naklj = new Random();

for (int i = 0; i < 50; i++)
     stevila[i] = naklj.Next(101); //naključna števila med 0 in 100

Console.WriteLine("Izpis vsebine tabele naključnih števil: \n(v vsaki vrstici je 10
            števil)\n");
for (int i = 0; i < 50; i++)
{
     Console.Write("{0,3} ",stevila[i]);
     if ((i+1) % 10 == 0) //po 10 števil v vsako vrstico izpisa
     Console.WriteLine();
}
```

```
Console.WriteLine("\n\nštevila med 0 in 100, ki jih NI v tej tabeli so:\n");
for (int st=0;st<=50; st++)
{
     bool found = false;  //logična spremenljivka za iskanje števila
     foreach (int x in stevila)
     {
        if (x == st)
        {
           found = true;  //število najdeno
           break;         //takojšnja prekinitev zanke foreach
        }
     }

     if (!found) //če števila v tabeli ni, ga izpišemo
         Console.Write(st+" ");
}
Console.WriteLine();
```

Naloge:

- S pomočjo **foreach** zanke izpiši najprej vse samoglasnike poljubnega stringa, nato ta stavek izpiši navpično, vsak znak v svojo vrsto!

- Dana je dvodimenzionalna tabela 10 x 10 celih števil. Napiši funkcijo, ki dobi to tabelo za parameter in ki vrne vsoto vseh sodih števil iz te tabele. Za obdelavo tabele uporabi zanko **foreach**.

- Dana je enodimenzionalna tabela 100 znakov. S pomočjo zanke **foreach** ugotovi, ali se v tej tabeli nahaja določen znak.

- Dana je premica y = 2x+3. Napiši program, ki bo najprej zgeneriral tabelo TOCKE 10 naključnih celih števil med -20 in + 20 (POZOR! – števila morajo biti različna) nato pa izpisal koordinate desetih točk, ki ležijo na dani premici, pri čemer boš za prve koordinate vzel vrednosti iz tabele TOCKE.

- Ustvari naključno tabelo 100 celih števil z vrednostmi med 50 in 100 in nato izpiši le tiste člene, ki so večji od zadnjega. Program dopolni še tako, da ustvariš naključno dolgo tabelo (a ne krajšo od 10 in ne daljšo od 1000 elementov).

- Napiši program, ki prebere niz in vsak znak niza razmnoži tolikokrat kot mu naroči uporabnik z vnesenim številskim parametrom. Primer izpisa: Uporabnik je vnesel niz AVTO in zahteval, da se vsak znak ponovi trikrat. Za obdelavo niza uporabi foreach zanko.

  AAAVVVTTTOOO

- Sestavi program, ki prebere celo število. Izpiše naj vsoto vseh števil od 1 do prebranega števila. Izpiše naj tudi vmesne vsote, vsako v svojo vrstico, na koncu pa še končno, skupno vsoto.

  Primer: Če je izbrano število 5, naj program izpiše :

  1
  3
  6
  10
  15

- Sestavi program, s katerim prebereš neko naravno število in ugotoviš, katero naravno število moraš povečat za 9-krat, da dobiš število, ki si ga izbral. Primer: Katero je število, ki ga moraš povečati za 9-krat, da dobiš število 810. To je število 90. Če tako število ne obstaja, naj nas program o tem obvesti!

- Napiši program, ki bo uporabniku omogočal vpis poljubnega stavka. Iz vpisanega stavka program nato izloči samoglasnike. Namesto ostalih znakov pa izpiše podčrtaj (_).

- Napiši program, ki prebere določeno število nizov in jih izpiše v obratnem vrstnem redu. Podatek o tem, koliko nizov bo vnesenih, prebereš na začetku programa. Pomagaj si s tabelo.

Zbirke - Collections

Podobno kot tabele, je tudi **zbirka** (**collection**) objekt, ki lahko vsebuje enega ali pa več elementov. Za razliko od tabel, kjer moramo že ob definiciji napovedati njeno dolžino, pa **zbirka NIMA** fiksne dolžine. Zbirke so torej lahko spremenljive dolžine.

Zbirke so lahko **netipizirane** (vsebujejo elemente različnih tipov) ali pa **tipizirane**: vsi elementi zbirke so enakega tipa.

Netipizirane zbirke

Pred deklaracijo **netipizirane** zbirke moramo poskrbeti za vključitev ustreznega imenskega prostora

```
using System.Collections;
```

Splošna deklaracija zbirke:

```
ArrayList ime_zbirke = new ArrayList();
```

Elemente dodajamo v zbirko s pomočjo metode Add().

```
Ime_zbirke.Add(7); //v zbirko smo dodali nov element - tip elementa ni pomemben
```

Do posameznih elementov zbirke dostopamo preko indeksa (tako kot pri tabelaričnih spremenljivkah). Začetni indeks je enak 0! Za razliko od tabelaričnih spremenljivk pa preko indeksa elementa zbirke ni dovoljena kar poljubna aritmetika!

Primer napačnega prirejanja in pravilnega prirejanja:

```
Ime_zbirke[1]=Ime_zbirke[0]+100;     //NAPAKA!!!!!!!! - aritmetični operatorji niso dovoljeni
Ime_zbirke[1]=Ime_zbirke[0];         //OK
Ime_zbirke[1]= "Danes je lep dan! "; //OK
```

Primer:

```
ArrayList stevila = new ArrayList();
stevila.Add(3);
stevila.Add(7);
stevila.Add("test"); //prevajanje normalno, a pri uporabi v nadaljevanju (for zanka) pride do
                     //napake
int vsota=0;

//Count je metoda zbirke, ki vrne trenutno število elementov zbirke
for (int i=0;i<stevila.Count;i++)
{
    int stevilo=(int)stevila[i];//potrebna eksplicitna konverzija (int)
    vsota+=stevilo;
}
```

Še en primer:

```
// POZOR, preveri, če si vključil imenski prostor: using System.Collections;
ArrayList al = new ArrayList();
```

```
Console.WriteLine("Začetna kapaciteta zbirke: " + al.Capacity);
Console.WriteLine("Začetno število elementov zbirke: " + al.Count);

Console.WriteLine();

Console.WriteLine("Dodajmo 6 elementov");
// Dodajanje elementov v zbirko
al.Add('C');
al.Add('A');
al.Add('E');
al.Add('B');
al.Add('D');
al.Add('F');

Console.WriteLine("Trenutna kapaciteta zbirke: " + al.Capacity);
Console.WriteLine("Trenutno število elementov zbirke: " + al.Count);

//Izpis elementov zbirke s pomočjo indeksa posameznega elementa zbirke.
Console.Write("Trenutna vsebina zbirke (izpis s pomočjo zanke for: ");
for (int i = 0; i < al.Count; i++)
     Console.Write(al[i] + " ");
Console.WriteLine("\n");

Console.WriteLine("Odstanimo 2 elementa");
// Odstranjevanje elementov iz zbirke
al.Remove('F');
al.Remove('A');

Console.WriteLine("Trenutna kapaciteta zbirke: " + al.Capacity);
Console.WriteLine("Trenutno število elementov zbirke: " + al.Count);

// Uporaba zanke foreach za izpis vsebine zbirke.
Console.Write("Vsebina zbirke (izpis s pomočjo zanke foreach): ");
foreach (char c in al)
   Console.Write(c + " ");
Console.WriteLine("\n");

Console.WriteLine("Dodajmo še 20 elementov");

for (int i = 0; i < 20; i++)
    al.Add((char)('a' + i));
Console.WriteLine("Trenutna kapaciteta zbirke: " + al.Capacity);
Console.WriteLine("Število elementov zbirke po dodajanju 20 elementov: " + al.Count);
Console.Write("Vsebina zbirke: ");
foreach (char c in al)
    Console.Write(c + " ");
Console.WriteLine("\n");

// Spremenimo vsebino zbirke s pomočjo tabelaričnega zapisa elementov zbirke.
Console.WriteLine("Spremenimo prve tri elemente zbirke");
al[0] = 'X';
al[1] = 'Y';
al[2] = 'Z';
Console.Write("Vsebina zbirke: ");
foreach (char c in al)
   Console.Write(c + " ");
Console.WriteLine();
```

Vaja:

```
/*Dan je poljuben stavek. Posamezne besede iz tega stavka prepišimo v zbirko z imenom besede,
nato pa te besede izpiši s pomočjo foreach zanke - vsaka beseda v svoji vrstici */

ArrayList besede = new ArrayList();
string stavek = "Danes je pa res en lep dan.";

//metoda split vse znake med dvema presledkoma shrani v ustrezen element objekta besede
besede.AddRange(stavek.Split(' '));

foreach (string beseda in besede) //izpis posameznih besed, vsaka beseda v novi vrstici
                Console.WriteLine(beseda);
```

Vaja:

```
/*Kreiraj zbirko 1000 naključnih realnih števil med 0 in 1000 z dvema decimalkama.
Napiši funkcijo, ki ugotovi in vrne največje število v zbirki
Napiši funkcijo, ki sešteje in vrne samo decimalke vseh števil v zbirki*/

//funkcija, ki ugotovi in izpiše največje število v zbirki
static void najvecje(ArrayList stevila)
{
    double najv=0;
    //Count je metoda zbirke, ki vrne trenutno število elementov zbirke
    for (int i = 0; i < stevila.Count;i++ )
        if ((double)stevila[i] > najv)
            najv = (double)stevila[i];  //vsak element zbirke moramo pretvoriti v double
    Console.WriteLine("Največje število v zbirki je "+najv);
}

//funkcija, ki sešteje in vrne vsoto vseh decimalk zbirke
static double vsotadec(ArrayList stevila)
{
    double vsota = 0;
    for (int i = 0; i < stevila.Count; i++)
    /*decimalke posameznega števila dobimo tako, da od števila odštejemo njegov celi del
    (funkcija truncate vrne celi del števila)*/
    //rezultat še zaokrožimo na dve decimalki
    vsota = vsota + Math.Round((double)stevila[i] - Math.Truncate((double)stevila[i]),2);
    return vsota;
}

static void Main(string[] args)
{
    //POZOR - imenski prostor ->>>>> using System.Collections;
     ArrayList stevila = new ArrayList();
     Random naklj = new Random();
     for (int i=0;i<1000;i++) //v zbirko dodamo 1000 naključnih števil
        stevila.Add(Math.Round(naklj.NextDouble()*1000,2));
     //klic funkcije, ki poišče največje število v zbirki
     najvecje(stevila);
     //klic funkcije, ki vrne vsoto decimalk zbirke
     Console.WriteLine("Vsota vseh decimalk zbirke je " + vsotadec(stevila));
}
```

Tipizirane zbirke

Pred deklaracijo **tipizirane** zbirke moramo poskrbeti za vključitev ustreznega imenskega prostora

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

Tipizirane zbirke imajo dve prednosti pred **netipiziranimi**. Prva je ta, da preverjajo tip vsakega elementa posebej že pri prevajanju, kar zagotavlja, da pri izvajanju programa ne pride do napak tipa **Run Time Error**. Druga prednost pa je seveda ta, da se na ta način zmanjša čas potreben za pretvarjanje podatkov iz zbirke (**casting**).

Tabela najpogosteje uporabljenih zbirk:

Zbirka	Razlaga
List<T>	Zbirka uporablja indeks za dostop do elementov, podobno kot tabela. Zelo je uporabna pri sekvenčnem dostopu do elementov zbirke. Lahko pa je neučinkovita pri vstavljanju elementov na sredino zbirke.
SortedList<K, V>	Uporablja ključ za pridobivanje vrednosti. Ključ je lahko poljuben objekt. Zbirka je lahko neučinkovita pri sekvenčnem dostopanju do posameznih elementov zbirke, je pa zelo učinkovita pri vstavljanju novih elementov na sredino zbirke.
Queue<T>	Uporablja metode za dodajanje in brisanje elementov.

Stack<T>	Uporablja metode za dodajanje in brisanje elementov.

Primer:

```
List<int> stevila = new List<int>();//deklaracija zbirke celih števil
//v zbirko lahko dodajamo le cela števila, njihovo število je poljubno!
stevila.Add(3);
stevila.Add(5);
//stevila.add("Test"); //prevajalnik bi tule javil napako!
int vsota = 0;
//Count je metoda zbirke, ki vrne trenutno število elementov zbirke
for (int i = 0; i < stevila.Count; i++)
{
   int stevilo = stevila[i];//eksplicitna konverzija NI potrebna
   vsota += stevilo;
}
```

Najpogosteje uporabljena zbirka je zbirka **List**. Nekaj primerov deklaracije takšnih zbirk:

```
//zbirka stringov
List <string> naslovi=new List<string>();

//zbirka decimalnih števil
List<decimal> cene = new List<decimal>();

//zbirka treh stringov. Seveda pa to ne pomeni, da stringov ne more biti več. A vsaka
//prekoračitev navedene dimenzije podvoji kapaciteto seznama!
List<string> kratice = new List<string>(3);
```

Najpomembnejše lastnosti in metode zbirke List

Indeks	Razlaga
[indeks]	Dostop do posameznega elementa zbirke. Indeks prvega elementa zbirje je tako kot pri tabelah enak 0.
Lastnost	**Razlaga**
Capacity	Nastavitev števila elementov ki jih lahko zbirka **List** hrani.
Count	Pridobivanje števila elementov v zbirki.
Lastnost	**Razlaga**
Add(objekt)	Dodajanje elementa na konec seznama in vračanje njegovega indeksa.
Clear()	Odstranitev vseh elementov iz seznama, lastnost **Count** postane enaka 0.
Contains(objekt)	Logična vrednost, ki ponazarja, ali seznam vsebuje navedeni objekt.
Insert(indeks,objekt)	Vrivanje elementa v seznam na mesto z navedenim indeksom.
Remove(objekt)	Odstranitev prve pojavitve navedenega objekta iz seznama.
RemoveAt(indeks)	Odstranitev elementa z navedenim indeksom iz seznama.
BinarySearch(objekt)	Iskanje navedenega objekta v seznamu in vračanje njegovega indeksa.
Sort()	Urejanje elementov seznama v naraščajočem zaporedju.

Primer:

```
List<string> priimki = new List<string>(3); //seznam treh stringov
priimki.Add("Ming");
priimki.Add("Lakovič");
priimki.Add("Taylor");
priimki.Add("House");//kapaciteta se podvoji na 6 elementov
priimki.Add("Menendez");
```

```
priimki.Add("Murach");
priimki.Add("Socrates");//kapaciteta se podvoji na 12 elementov
//urejanje seznama
priimki.Sort();

string vsiPriimki = "";
for (int i = 0; i < priimki.Count; i++)
{
   vsiPriimki += priimki[i] + "\n";
}
//urejen seznam vseh priimkov prikažemo v sporočilnem oknu
Console.WriteLine (vsiPriimki);
```

Še nekaj primerov:

```
//definicija tabele decimalnih števil
decimal[] novecene ={3275.68m, 4378.12m, 54123.34m, 100.00m };
//deklaracija seznama decimalnih števil
List<decimal> cene = new List<decimal>();
//v seznam dodamo vsa decimalna števila iz tabele novecene
foreach (decimal d in novecene)
      cene.Add(d);

decimal cena1 = cene[0];//dostop do prvega elementa zbirke
cene.Insert(0, 2345.11m);//vstavljanje nove cene na začetek seznama
cena1 = cene[0];  //cena1 dobi novo vrednost 2345.11

decimal cena2=cene[1];  //cena2 dobi vrednost 3275.68
cene.RemoveAt(1);    //odstranimo DRUGI elemenet (prvi elemeni ima indeks 0!) iz seznama
cena2 = cene[1];  ////cena2 dobi vrednost 4378.12

decimal x = 100.00m;
//če v zbirki najdemo znesek 100.00, da odstranimo iz zbirke
if (cene.Contains(x))
{
     cene.Remove(x);
     Console.WriteLine("Znesek odstranjen iz seznama!");
}
```

Uporaba zbirk tipa **vrsta** (**queue**) in **stack** je podobna, a imata obe vrsti zbirk svoje posebnosti in zaradi tega nista tako pogosto uporabljeni.

Naloge:

- Kreiraj zbirko n naključnih decimalnih števil (tudi n je podatek) večjih od 0 in manjših od 200. Ugotovi in izpiši koliko od teh števil je manjših od 10, med 10 (vključno) in 100 (vključno) in koliko večjih od 100.
- Preberi 10 besed in jih shrani v netipizirano zbirko. Ugotovi in izpiši najdaljšo besedo v tej zbirki.
- Napiši funkcijo, ki dobi za parameter poljuben stavek, vrne pa najdaljšo besedo v tem stavku. Navodilo: s pomočjo metod **AddRange** in **Split** besede iz tega stavka shrani v netipizirano zbirko, ki jo nato obdelaj!

Strukture

Struktura je sestavljeni podatkovni tip, sestavljen največkrat iz enostavnih podatkovnih tipov. V strukturo združimo podatke različnih tipov, ki skupaj tvorijo neko celoto (npr. podatki o določeni osebi, podatki o določenem artiklu, podatki o avtomobilu, …).

Strukture so spremenljivke vrednostnega tipa, kar pomeni, da novo spremenljivko tipa struktura deklariramo tako kot običajno vrednostno spremenljivko (brez uporabe operatorja **new**). To pa hkrati pomeni, da ko neko strukturo deklariramo, **njene komponente še nimajo začetne vrednosti** (niso inicializirane).

Splošna deklaracija strukture :

```
struct <ime strukture>
{
    <public><tip1> <ime prvega člana>;
    <public><tip2> <ime drugega člana>;
    // ….
};  //Podpičje za zaključek strukture ni obvezno
```

Struktura se začne z rezervirano besedo **struct**, ki ji sledi ime strukture, nato pa v zavitih oklepajih naštejemo člane strukture. Besedica **public** naj nam zaenkrat pomeni, da so člani te strukture javni in imamo do njih neomejen dostop.

Primer:

```
struct obcan   //deklaracija MORA biti pred glavnim programom
{
   public string ime;
   public string priimek;
   public string EMSO;
   public long davcna;
};
obcan o; //deklaracija spremenljivke obc tipa obcan
//POZOR: elementi strukture obcan še NISO inicializirani, nimajo začetne vrednosti
```

Do elementov strukture dostopamo s pomočjo operatorja pika ('.'). Nad spremenljivkami izpeljanimi iz neke strukture lahko izvajamo vse operacije, ki jih poznamo nad osnovnimi podatkovnimi tipi. **Pomembno pa je, da strukturo deklariramo pred glavnim programom**.

Primer:

```
struct obcan
{
    public string ime;
    public string priimek;
    public string EMSO;
    public long davcna;
};

static void Main(string[] args)
{
    obcan o;
    Console.Write("Vpiši ime            : ");
    o.ime=Console.ReadLine();
```

```
    Console.Write("      priimek         : ");
    o.priimek=Console.ReadLine();
    Console.Write("      EMŠO            : ");
    o.EMSO=Console.ReadLine();
    Console.Write("      davčna številka : ");
    o.davcna=Convert.ToInt64(Console.ReadLine());
    Console.WriteLine("\nPodatki o občanu : \n");
    Console.WriteLine("    Priimek in ime : "+o.priimek+" "+o.ime);
    Console.WriteLine("    EMŠO           : "+o.EMSO);
    Console.WriteLine("    Davčna številka: "+o.davcna);
    Console.WriteLine();
}
```

Vaja:

```
/*Deklariraj strukturo ulomek z dvema komponentama: števec in imenovalec ulomka. Napiši
funkcijo za vnos podatkov za ta dva ulomka. Ulomka nato seštej in rezultat zapiši v tretji
ulomek, ki ga nato v primerni obliki tudi izpiši. Napiši tudi metodo za krajšanje ulomka*/

struct ulomek
{
    public int stevec;
    public int imenovalec;
};

static void vnos(out ulomek u) //metoda za vnos števca in imenovalca ulomka
{
    Console.WriteLine("Vnesi števec in menovalec ulomka(Imenovalec mora biti različen od 0)");
    Console.Write("Števec: ");
    u.stevec=Convert.ToInt32(Console.ReadLine());
    do
    {
        Console.Write("Imenovalec: ");
        u.imenovalec = Convert.ToInt32(Console.ReadLine());
    }
    while (u.imenovalec == 0);
}

static void izpis(ulomek u) //metoda za izpis ulomka
{
    Console.WriteLine(u.stevec + " / " + u.imenovalec);
}

static void krajsaj(ref ulomek u)//Metoda za krajšanje ulomka. Parameter klican po referenci
{
    for (int i=2;i<=u.stevec;i++)
      if (u.stevec % i == 0 && u.imenovalec % i == 0)
      {
         do  //ker obstaja možnost, da lahko ulomek z istim številom krajšamo večkrat!!!
         {
             u.stevec = u.stevec / i;
             u.imenovalec=u.imenovalec/i;
         }
         while (u.stevec % i == 0 && u.imenovalec % i == 0);
      }
}

static void Main(string[] args)
{
    ulomek u1, u2;//dva nova ulomka
    vnos(out u1); //klic funkcije za vnos števca in menovalca ulomka u1 (klic je po referenci
                  //out, ker ulomka še nista inicializirana)
    vnos(out u2); //klic funkcije za vnos števca in menovalca ulomka u2 (klic je po referenci
                  //out, ker ulomka še nista inicializirana)
    //seštejmo oba ulomka
    ulomek u3;
    u3.imenovalec=u1.imenovalec*u2.imenovalec;
    u3.stevec = u1.stevec * u2.imenovalec + u2.stevec * u1.imenovalec;
    Console.Write("\nVsota dveh ulomkov \n"+ u1.stevec + " / " + u1.imenovalec + " + " +
                   u2.stevec + " / " + u2.imenovalec + " = ");
    izpis(u3); //klic metode za izpis ulomka u3
    //okrajšajmo ulomek u3
    krajsaj(ref u3); //klic metode za krajšanje ulomka - parameter klican po referenci ref,
                     //ker je ulomek u3 ŽE inicializiran
```

```
            Console.WriteLine("Okrajšana vsota: " + u3.stevec + " / " + u3.imenovalec);
}
```

Vaja:

```
/*Napišimo program za vodenje evidence podrtih kegljev za dva tekmovalca. Kreiraj strukturo
tekmovalec, ki ima dve komponenti: naziv tekmovalca in tabelo v katero boš vpisoval posamezne
mete. Preberi imeni tekmovalcev, nato pa še posamezne mete posameznega tekmovalca (po 10
metov). Na koncu ugotovi zmagovalca in napiši metodo, ki vrne povprečno število podrtih
kegljev izbranega tekmovalca.*/

struct tekmovalec
{
    public String naziv;
    public int [] keglji;  //napoved tabele: DIMENZIJA tabele še NI določena!!!
}

static double povp(tekmovalec T,int N)
{
    int suma = 0;
    for (int i = 0; i < N; i++) //seštejemo mete temovalca
        suma += T.keglji[i];
    return (Math.Round((double)suma / N,2));
}

static void Main(string[] args)
{
    tekmovalec A,B;  //dve vrednostni spremenljivki tipa struktura
    int N = 2;//število metov posameznega tekmovalca
    A.keglji=new int[N];   //na kopici ustvarimo tabelo za vnos podatkov o podrtih kegljih
                           //prvega tekmovalca
    B.keglji = new int[N]; //na kopici ustvarimo tabelo za vnos podatkov o podrtih kegljih
                           //drugega tekmovalca
    Console.Write("Naziv prvega tekmovalca: ");
    A.naziv=Console.ReadLine();
    Console.Write("Naziv drugega tekmovalca: ");
    B.naziv=Console.ReadLine();
    Console.WriteLine("Vnos posameznih metov obeh tekmovalcev");
    Console.WriteLine("---------------------------------------");
    Console.WriteLine("{0,-12}              {1,-12}", A.naziv, B.naziv);
    Console.WriteLine("------------              ------------");
    for (int i = 0; i < N; i++)
    {
        A.keglji[i] = Convert.ToInt32(Console.ReadLine());
        /*metoda SetCursorPosition postavi kurzor na pravilno pozicijo za vnos podatkov.
         Metoda ima dva parametra -  oddaljenost od levega roba in oddaljenost od zgornjega
         roba (merjeno v znakih) */
        Console.SetCursorPosition(26, 6 + i);
        B.keglji[i] = Convert.ToInt32(Console.ReadLine());
    }
    int sumaA = 0, sumaB = 0;
    for (int i=0;i<N;i++) //seštejemo mete posameznih temovalcev
    {
         sumaA += A.keglji[i];
         sumaB += B.keglji[i];
    }

    if (sumaA > sumaB)
        Console.WriteLine("Zmagovalec je " + A.naziv + ". Skupaj je podrl " + sumaA + "
                          kegljev!");
    else if (sumaA==sumaB)
        Console.WriteLine("Tekmovalca sta izenačena. Oba sta podrla po " + sumaA + "
                          kegljev!");
    else
        Console.WriteLine("Zmagovalec je " + B.naziv + ". Skupaj je podrl " + sumaB + "
                          kegljev!");

    Console.WriteLine("Povprečno število metov prvega tekmovalca je
                      "+povp(A,A.keglji.Length));
    Console.WriteLine("Povprečno število metov drugega tekmovalca je
                      "+povp(B,B.keglji.Length));
}
```

Spremenljivko tipa struktura (objekt) pa lahko ustvarimo tido na kopici, torej z uporabo operatorja new:

```
struct kontinent
{
   public string naziv;
   public long stprebivalcev;
};

static void Main(string[] args)
{
   kontinent k1;//k1 je vrednostna spremenljivka tipa kontinent
   kontinent k2 = new kontinent();//k2 smo ustvarili na kopici-referenčna spremenlj. (objekt)

   //od tu dalje delamo z obema spremenljivkama enako
   k1.naziv = "Evropa";
   k2.naziv = "Azija";
   Console.Write(k1.naziv+" - število prebivalcev: ");
   k1.stprebivalcev = Convert.ToInt64(Console.ReadLine());
   Console.Write(k2.naziv+" - število prebivalcev: ");
   k2.stprebivalcev = Convert.ToInt64(Console.ReadLine());
}
```

Naloge:

- Napiši program, ki v strukturo podatki prebere podatke o šoli (v strukturi je ime šole, naslov, poštna številka kraj in telefon). Vpisane podatke nato izpiše v obliki: "Hodim v šolo ..., ki je na naslovu: ... Telefonska številka je ..., poštna ..., kraj …!

- Kreiraj strukturo *pravokotnik* z dvema elementoma – dolžino in višino pravokotnika. Ustvari vrednostno spremenljivko *P1* tipa pravokotnik in referenčno spremenljivko objekt) *P2* izpeljano iz strukture pravokotnik. Napiši funkcijo za vnos podatkov v spremenljivki *P1* oz. *P2*. Izračunaj in izpiši obseg in ploščino obeh pravokotnikov!

- Kreiraj strukturo *spricevalo* s tremi komponentami: ime in priimek (string), predmetnik (enodimenzionalna tabela 10 stringov, v katere bomo shranjevali imena predmetov), ter ocene (enodimenzionalna tabela 10 celih števil - za toliko ocen, kot je bilo predmetov). Napiši funkcijo
  - za vnos podatkov v tako strukturo;
  - za izpis spričevala. Na koncu izpiši še uspeh(5-odl, 4-pdb, 3-db, 2-zd, 1-nzd);
  - ki izračuna in vrne povprečno oceno.

Tabele struktur

Struktura lahko nastopa tudi kot tabelarična spremenljivka. Do komponent posamezne spremenljivke dostopamo preko indeksa tabelaričnega elementa in operatorja pika. Tabela struktur tako predstavlja zelo kompaktno podatkovno strukturo in ohranja vse prednosti in jih prinašajo operacije nad tabelami in strukturami.

Primer:

```
struct tekoci    //deklaracija strukture MORA biti pred glavnim programom
{
    public int zapst;
    public string naziv;
    public decimal znesek;
};

static void Main(string[] args)
{
     tekoci[] racuni = new tekoci[10];  //tabela 10 struktur

     Console.WriteLine("Vnos podatkov v tabelo: \n");
     for (int i = 0; i < 10; i++)
     {
```

```
            Console.WriteLine("\nKomitent številka " + (i + 1));
            racuni[i].zapst=i+1;
            Console.Write("Naziv: ");
            racuni[i].naziv = Console.ReadLine();
            Console.Write("Znesek: ");
            racuni[i].znesek = Convert.ToDecimal(Console.ReadLine());
        }
}
```

Vaja:

```
/*Deklariraj strukturo slaščica z dvema komponentama: ime slaščice in cena. Napiši zanko za
vnos podatkov v tabelo N takih struktur. Napiši metodi za iskanje in izpis najdražje slaščice
v tej tabeli in metodo, ki vrne število slaščic dražjih od nekega poljubnega zneska*/

struct slascica  //deklaracija strukture slascice
{
    public string ime;
    public double cena;
};


//metoda, ki poišče in izpiše ime in ceno najdražje slaščice
static void najdrazja(slascica[] slascicarna)
{
    int naj = 0; //predpostavimo, da je najdražja slaščica v tabeli tista, ki ima indeks 0
    for (int i = 1; i < slascicarna.Length; i++)
    {
        if (slascicarna[i].cena > slascicarna[naj].cena)
        naj = i;
    }
Console.WriteLine("\nNajdrazja slascica v tabeli je " + slascicarna[naj].ime + ",  njena cena
                  pa je " + slascicarna[naj].cena);
}

//metoda, ki vrne število slaščic, ki so dražje od 200 sit!
static int drazje(double znesek, slascica[] slascicarna)
{
    int stevilo = 0; //začetna vrednost števila slaščic dražjih od 200 SIT
    for (int i = 1; i < slascicarna.Length; i++)
    {
       if (slascicarna[i].cena > znesek)
          stevilo++;
    }
    return stevilo;
}

static void Main(string[] args)
{
    int N = 5;
    slascica[] slascicarna = new slascica[N];  //tabela 50 slaščic
    //vnos podatkov tabelo
    for (int i = 0; i < N; i++)
    {
        Console.Write("Slaščica: ");
        slascicarna[i].ime = Console.ReadLine();
        Console.Write("Cena: ");
        slascicarna[i].cena = Convert.ToDouble(Console.ReadLine());
    }
    //funkcija, ki poišče in izpiše ime najdražje slaščice v tabeli
    najdrazja(slascicarna);
    //klic funkcije, ki vrne število slaščic, ki so dražje od 200 SIT
    Console.WriteLine("\nŠtev. slaščic, ki so dražje od 200 SIT je " +
                      drazje(200.00,slascicarna));
}
```

Vaja:

```
/*Kreiraj strukturo tocka2D z dvema komponentama - koordinatama tipa int. Napiši funkcije za
vnos podatkov, izračun razdalje posamezne točke od koordinatnega izhodišča in funkcijo za
izračun razdalje med točkama!*/
```

```
struct tocka2D {
    public int x,y;
};

//parameter je klican po referenci, ker pa še ni inicializiran uporabimo operator out
static void vnos(out tocka2D T)
{
    Console.Write("Prva koordinata:  ");
    T.x=Convert.ToInt32(Console.ReadLine());
    Console.Write("Druga koordinata: ");
    T.y=Convert.ToInt32(Console.ReadLine());
}

static double razdalja (tocka2D T)
{
    double pomozna = Math.Sqrt(Math.Pow(T.x, 2) + Math.Pow(T.y, 2));
    return (Math.Round(pomozna,2));
}

static double razdalja(tocka2D T1, tocka2D T2)
{
    double pomozna = Math.Sqrt(Math.Pow(T1.x - T2.x, 2) + Math.Pow(T1.y - T2.y, 2));
    return (Math.Round(pomozna,2));
}

static void najboljoddaljena(tocka2D[] tabela)
{
    int naj = 0; //spremenljivka 'naj' predstavlja INDEKS najbolj oddaljene točke; na začetku
                 določimo da je to kar prva točka v tabeli
    for (int i = 1; i < 100; i++)
    {
       if(razdalja(tabela[i])>razdalja(tabela[naj]))
          naj=i; //če smo našli bolj oddaljeno točko, si zapomnimo njen indeks v tabeli
    }
    Console.WriteLine("Koordinati točke, ki je najbolj oddaljena od izhodišča: ( tabela[naj].x
             +" , "+tabela[naj].x+" )");
}

static void Main(string[] args)
{
    tocka2D A,B,C;
    Console.WriteLine("Vnos koordinat tocke A");
    vnos(out A); //ker struktura A še ni inicializirana, smo za klic po referenci uporabili
                    operator out!!!
    Console.WriteLine("Vnos koordinat tocke B");
    vnos(out B); //ker struktura B še ni inicializirana, smo za klic po referenci uporabili
                    operator out!!!

    Console.WriteLine("Tocka A je od koord. izhodišča oddaljena za "+razdalja(A)+" enot");
    Console.WriteLine("Razdalja med točkama A in B je "+razdalja(A,B)+" enot!");

    /*VAJE:
      Preberi podatke še za točko C! Kolikšen je obseg trikotnika ABC?
      Katera od tock A, B in C je najbolj oddaljena od koord.izhodišča?
      Napiši funkcijo HERON, ki ima za parametre tri točke, vrne pa ploščino trikotnika ki ga
      tvorijo; klic funkcije npr: double P=HERON(A,B,C);
      Heronova formula p=sqrt(s*(s-a)*(s-b)*(s-c)), s=(a+b+c)/2  */

     /*Deklarirajmo še tabelo 100 naključnih točk (koordinati x in y sta naključni števili med
      0 in 100)in s pomočjo funkcije ugotovimo in izpišimo koordinate tiste točke, ki je
      najdalj od izhodišča */

     tocka2D[] tabela = new tocka2D[100];//deklaracija tabele 100 točk tipa tocka2D
     Random naklj = new Random();
     for (int i = 0; i < 100; i++)  //določimo naključne koordinate vsem 100 točkam
     {
        tabela[i].x = naklj.Next(100);
        tabela[i].y = naklj.Next(100);
     }
     najboljoddaljena(tabela);  //klic funkcije, ki ugotovi in izpiše koordinati točke, ki je
                                  najbolj oddaljena od koordinatnega izhodišča
}
```

Naloge:

- Kreiraj strukturo **tocka**, ki predstavlja točko v dvodimenzionalnem koordinatnem sistemu (elementa strukture sta koordinati, obe sta celoštevilskega tipa). Nato deklariraj tabelo 100 struktur tipa **tocka** in jo inicializiraj tako, da bodo koordinate točk naključna cela števila med -100 in +100. Napiši funkcijo, ki dobi za parameter to tabelo in ki
  - ugotovi in v primerni obliki izpiše koordinati točke, ki je najbolj oddaljena od koordinatnega izhodišča (razdalja točke od izhodišča je kvadratni koren iz vsote kvadratov obeh koordinat!);
  - vrne število točk, ki ležijo v prvem kvadrantu (obe koordinati >0).

- Napiši program, s katerim bi za 20 trgovin (tabela trgovin!) vnesel podatke o nazivu trgovine in ceni za 1 kg kruha. Napiši funkcijo
  - ki dobi za parameter to tabelo, vrne pa naziv trgovine z najdražjim kruhom;
  - ki dobi za parameter to tabelo, vrne pa povprečno ceno kruha.

- Deklariraj strukturo, ki naj predstavlja nek ulomek. Nato deklariraj enodimenzionalno tabelo 100 takih struktur. Napiši
  - Stavke, ki vsem ulomkom v tej tabeli priredijo naključne vrednosti števcev in imenovalcev (vrednosti naj bodo med 1 in 10);
  - funkcijo, ki dobi kot argument dva indeksa te tabele, vrne pa vsoto ulomkov na ustreznih mestih tabele (vrednost, ki jo funkcija vrne naj bo tipa **double**);
  - funkcijo, ki dobi za parameter to tabelo, vrne pa največji ulomek te tabele (funkcija torej vrne strukturo!).

- Deklariraj strukturo, s katero boš opisal podatke o nekem rabljenem vozilu (tip vozila, prevoženi km in cena). Kreiraj tabelo 10 takih struktur. Napiši program, ki bo v obliki menija nudil naslednje opcije:
  - Vnos novega vozila v seznam;
  - Izpis vseh podatkov o vozilih vozil dražjih od 10.000 EUR na zaslon.

- Za zaposlenega potrebujejo v podjetju naslednje podatke:

  ime, *priimek*, *šifra* (celo število) in *osebne dohodke za zadnjih 12 mesecev* (tabela 12 realnih števil)

  Napiši:
  - deklaracijo za tako strukturo;
  - deklaracijo spremenljivk *zaposl1* in *zaposl2* za tako strukturo;
  - sklic na plačo z indeksom nič spremenljivke *zaposl1*;
  - deklaracijo za dinamično ustvarjen objekt strukture (ustvarjen na kopici);
  - sklic na priimek dinamično ustvarjene strukture:
  - ukaz za sprostitev pomnilnika, zasedenega z dinamično ustvarjenim objektom strukture;
  - deklaracijo tabele 10 primerkov struktur zgornjega tipa;
  - sklic na ime primerka strukture z indeksom 5.

- Dijak obiskuje 5 predmetov. Šolsko leto ima tri ocenjevalna obdobja. V vsakem ocenjevalnem obdobju dobi dijak pri vsakem predmetu dve oceni. Deklariraj ustrezno podatkovno strukturo in napiši podprogram, za vnos in izpis vseh ocen za enega dijaka!

Gnezdene strukture

Strukture so lahko tudi gnezdene (struktura v strukturi). Najprej deklariramo strukturo, ki bo vgnezdena v drugi strukturi, nato pa še samo strukturo. Do članov strukture dostopamo z operatorjem pika.

Splošna deklaracija gnezdene strukture :

```
struct <ime gnezdene strukture>  //deklaracija strukture, ki bo vgnezdena v
```

```
{                                    //drugi strukturi
    <public><tip1> <ime prvega člana>;
    <public><tip2> <ime drugega člana>;
    // ….
};
```

```
struct <ime strukture>                    //struktura
{
    <public><tip1> <ime prvega člana>;
    <public><tip2> <ime drugega člana>;
    <gnezdena struktura> <ime člana>    //gnezdena struktura
    // ….
};
```

Primer:

```
struct rojen   //struktura, ki bo vgnezdena
{
    public int dan,mesec,leto;
};

struct oseba
{
    public string priimek,ime,naslov;
    public rojen rojstvo;  //gnezdena struktura
};

static void vnos(out oseba os)//metoda za vnos/branje podatkov v strukturo
{
    Console.Write("Priimek: ");
    os.priimek=Console.ReadLine();
    Console.Write("Ime: ");
    os.ime=Console.ReadLine();
    Console.Write("Naslov: ");
    os.naslov=Console.ReadLine();
    Console.Write("Podatki o rojstvu -   Dan: ");
    os.rojstvo.dan=Convert.ToInt32(Console.ReadLine());
    Console.Write("                    Mesec: ");
    os.rojstvo.mesec=Convert.ToInt32(Console.ReadLine());
    Console.Write("                     Leto: ");
    os.rojstvo.leto=Convert.ToInt32(Console.ReadLine());
}

static void Main(string[] args)
{
    oseba nekdo, Mary, Angelika;  //tri spremenljivke tipa oseba (tri nove strukture)
    //podatke za osebo nekdo določimo npr. kar takole:
    nekdo.priimek = "Andersen";
    nekdo.ime = "Tim";
    nekdo.naslov = "Beverly Hills 123";
    nekdo.rojstvo.dan = 12;
    nekdo.rojstvo.mesec = 6;
    nekdo.rojstvo.leto = 2000;

   vnos(out Mary);//podatke za osebo Mary preberemo s pomočjo funkcije
   //Napiši metodo za izpis podatkov o neki spremenljivki tipa oseba (klic npr : izpis(nekdo)
   //Napiši funkcijo, ki dobi za parameter dve osebi in ki vrne starejšo od obeh oseb
}
```

Vaja:

```
/*Deklariraj strukturo dijak, nato pa strukturo tecaj, v kateri je vgnezdena tabela struktur
tipa dijak. Preberi podatke, nato pa izračunaj povprečno starost vseh dijakov!*/

struct dijak              //struktura, ki bo vgnezdena v drugo strukturo
{
   public string ime;
   public string priimek;
```

```
   public int dan;
   public int mesec;
   public int leto;
};

struct krozek
{
   public int Id;
   public string naziv;
   public dijak [] Dijaki;  //tabela dijakov - vsak dijak je gnezdena struktura
};

const int N=10;   //definicija celoštevilske konstante (dimenzija tabele)

static void Main(string[] args)
{
   krozek T;                  //nova spremenljivka tipa krozek (struktura)
   Console.WriteLine("Vnesi podatke o tečaju: ");
   Console.Write("Številka tečaja: ");
   T.Id=Convert.ToInt32(Console.ReadLine());
   Console.Write("Naziv tečaja: ");
   T.naziv=Console.ReadLine();
   Console.WriteLine("Podatki o udeležencih tečaja: ");
   T.Dijaki = new dijak[N];   //inicializacija tabele
   for (int i = 0; i < N; i++)
   {
      Console.WriteLine("\nDijak št. "+(i + 1)+". :");
      Console.Write("          ime: ");
      T.Dijaki[i].ime=Console.ReadLine();
      Console.Write("      priimek: ");
      T.Dijaki[i].priimek = Console.ReadLine();
      Console.Write("  dan rojstva: ");
      T.Dijaki[i].dan = Convert.ToInt32(Console.ReadLine());
      Console.Write("mesec rojstva: ");
      T.Dijaki[i].mesec=Convert.ToInt32(Console.ReadLine());
      Console.Write(" leto rojstva: ");
      T.Dijaki[i].leto = Convert.ToInt32(Console.ReadLine());
    }
    seznam(T,1990);//Funkcija, naj izpiše seznam dijakov rojenih po letu 1990
    //Izračunajmo skupno starost vseh dijakov glede na današnji datum
    DateTime d; //struktura DateTime je namenjena deli z datumi in časi
    d = DateTime.Now;  //Now je lastnost, ki v spremenljivko tipa DateTime vrne trenutni datum
    int starost = 0;
    for (int i = 0; i < N; i++)
       starost = starost + d.Year - T.Dijaki[i].leto;
    Console.WriteLine("\nSkupna starost vseh tečajnikov je "+starost/N +" let.");
}

//Funkcija, ki izpiše seznam dijakov rojenih po letu letnik
static void seznam(krozek T, int letnik) {
    Console.WriteLine("Seznam dijakov rojenih po letu " + letnik + ": \n");
    for (int i = 0; i < N; i++)
       if (T.Dijaki[i].leto > letnik)
          Console.WriteLine(T.Dijaki[i].ime + " " + T.Dijaki[i].priimek);
}
```

Vaja:

```
/*Kreiraj telefonski imenik: sestavlja ga tabela oseb. Oseba je struktura s podatki o določeni
osebi, ter z gnezdeno strukturo, ki predstavlja podatek o področni kodi ter telefonski
številki!*/

struct telefon
{
    public int  pkoda; //področna koda
    public long stevilka;
};

struct oseba
{
    public string ime;
    public telefon TEL; //gnezdena struktura
```

```
};

const int ST=2; //število oseb v imeniku

static void vnos(out oseba OS)
{
   Console.Write("\nIme: ");
   OS.ime=Console.ReadLine();
   Console.Write("TELEFON - Podrocna koda: ");
   OS.TEL.pkoda=Convert.ToInt32(Console.ReadLine());
   Console.Write("          Številka     : ");
   OS.TEL.stevilka=Convert.ToInt32(Console.ReadLine());
}

/*funkcija OBDELAJ ugotovi in izpiše koliko oseb je iz omrežne skupine OR*/
static void OBDELAJ(oseba [] IM,int OMR)
{
   int  skupaj=0;
   for (int i=0;i<ST;i++)
   {
       if (IM[i].TEL.pkoda==OMR) skupaj++;
   }
   Console.WriteLine("\nŠtevilo oseb iz omrežne skupine "+OMR+": "+skupaj);
}

static void Main(string[] args)
{
   oseba[] Imenik = new oseba[ST];
   for (int i = 0; i < ST; i++)   //Vnos podatkov v tabelo
   {
       vnos(out Imenik[i]); //funkcija ima za parameter strukturo (to je element tabele)
   }
   OBDELAJ(Imenik,4);//funkcija ugotovi, koliko oseb je iz omrežne skupine 4
}
```

Naloge:

- Kreiraj telefonski imenik: sestavlja ga tabela oseb (velikost tabele določi sam). Oseba je struktura s podatki o določeni osebi, ter z gnezdeno strukturo, ki predstavlja podatek o področni kodi (omrežni skupini) ter telefonski številki! Napiši funkcijo, ki izpiše seznam vseh oseb iz določene omrežne skupine!

- Kreiraj strukturo *datum*, ki naj vsebuje komponente *dan*, *mesec* in *leto*. Strukturo *datum* nato uporabi v strukturi *oseba*, ki naj vsebuje *ime*, *priimek*, *datum-rojstva*, *datum-prihoda*, *datum-odhoda* in *število-dni*. Nato deklariraj spremenljivko *o1* tipa *oseba*.

    - in napiši funkcijo za vnos podatkov v strukturo *oseba*;
    - napiši stavke za izračun razlike med datumom prihoda in datumom odhoda osebe o1.

Konstruktor

Omenili smo že, da so strukture spremenljivke vrednostnega tipa, kar pomeni, da ob deklaraciji nove spremenljivke tega tipa njene komponente še nimajo začetne vrednosti.

Primer:

```
struct tocka3D {
    public int x,y,z;
};

static void Main(string[] args)
{
```

```
    Tocka3D A; //točka A še nima inicializiranih komponent/koordinat

    Console.WriteLine(A.x + "  " + A.y + "  " + A.z); /*Compile time error,
                                        ker komponente točke A še niso inicializirane!!!*/
}
```

Spremenljivka A je spremenljivka vrednostnega tipa, shranjena na skladu in ob deklaraciji še ni inicializirana. Za inicializacijo komponent seveda lahko poskrbimo sami, npr. takole:

```
tocka3D A; //deklaracija nove vrednostne spremenljivke A, ki je tipa tocka3D (struktura)
A.x = 2;   //inicializacija prve komponente točke A
A.y = 5;   //inicializacija druge komponente točke A
A.z = 4;    //inicializacija tretje komponente točke A

Console.WriteLine(A.x + "  " + A.y + "  " + A.z); //Izpis 2  5  4
```

Tak način je seveda čisto legalen, a zamuden če je takih spremenljivk veliko. Obstaja pa še možnost, da spremenljivko tipa struktura ustvarimo na kopici, seveda s pomočjo operatorja **new**. V tem primeru gre za referenčno spremenljivko, kar pa pomeni, da je taka spremenljivka ob deklaraciji že inicializirana:

```
tocka3D B = new tocka3D();//deklaracija nove REFERENČNE spremenljivke za strukturo tocka3D
/*POZOR: ker je B referenčna spremenljivka (ustvarili smo jo s pomočjo operatorja new), so
komponente točke B ŽE INICIALIZIRANE, imajo torej vrednost 0*/

Console.WriteLine(B.x + "  " + B.y + "  " + B.z); //Izpis 0  0  0
```

Vrednost, ki so jo dobile komponente spremenljivke B, kreirane s pomočjo operatorja **new**, so seveda enake 0 (privzeta začetna vrednost). V kolikor pa bi želeli imeti drugačne začetne vrednosti, lahko to storimo s pomočjo posebne metode znotraj strukture – ta metoda se imenuje **konstruktor**. Konstruktor je metoda znotraj strukture, ki ima enako ime kot struktura, njeni parametri pa so komponente te strukture (OBVEZNO morajo kot parametri nastopati VSE vrednostne komponente strukture). Struktura torej ne more vsebovati konstruktorja brez parametrov. (V poglavju razredi in objekti bomo spoznali bolj kompleksen podatkovni tip **class (razred)**, ki pa za razliko od strukture lahko vsebuje tudi konstruktorje brez parametrov!!!).

```
struct tocka3D
{
    public int x,y,z;

    //KONSTRUKTOR - poskrbi za inicializacijo komponent strukture
    public tocka3D(int koordX,int koordY, int koordZ)
    {
         x = koordX;
         y = koordY;
         z = koordZ;
    }
};

static void Main(string[] args)
{
    tocka3D B = new tocka3D(); //Privzeta deklaracija nove spremenljivke tipa tocka3D
    Console.WriteLine(B.x + "  " + B.y + "  " + B.z); //izpis 0  0  0

    tocka3D C = new tocka3D(3,6,9); //Ob deklaraciji točke C se izvede tudi konstruktor
    Console.WriteLine(C.x + "  " + C.y + "  " + C.z); //izpis 3  6  9
}
```

Iz primera je razvidno, da se konstruktor izvede le v primeru, ko pri kreiranju nove referenčne spremenljivke v oklepaju navedemo tudi vrednosti posameznih komponent. **OBVEZNO** pa moramo navesti želene vrednosti **vseh** komponent strukture – izjema so le tabelarične komponente strukture, ki jih v glavi konstruktorja ne navajamo.

Primer:

Kreirajmo novo referenčno spremenljivko A1 (točko) tipa **tocka3D**, ki bo že ob deklaraciji imela naključne koordinate med 0 in 10.

```
struct tocka3D
{
    public int x,y,z;
    //konstruktor, ki poskrbi za inicializacijo komponent strukture
    public tocka3D(int koordX,int koordY, int koordZ)
    {
         x = koordX;
         y = koordY;
         z = koordZ;
    }
};

static void Main(string[] args)
{
   Random naklj = new Random();
   //novi referenčni spremenljivki A1 posredujemo za paramere 3 naključna števila med 0 in 11
   tocka3D A1 = new tocka3D(naklj.Next(11), naklj.Next(11), naklj.Next(11));
   Console.WriteLine(A1.x + "  " + A1.y + "  " + A1.z); //izpis naključnih koordinat točke A1
}
```

Vaja:

```
/*Deklarirajmo strukturo, ki naj predstavlja kompleksno število. Struktura naj ima svoj
konstruktor. Napišimo še metodo za izpis kompleksnega števila, nato pa kreirajmo tabelo 10
kompleksnih števil in jo inicializirajmo z naključnimi vrednostmi obeh komponent.*/
struct Complex
{
    public Complex(float real, float imag)  //Konstruktor
    {
        realna = real;
        imaginarna = imag;
    }
    public float realna;
    public float imaginarna;
};

static string IzpisiKompleksno(Complex C)//Metoda za izpis kompleksnega števila
{
    return (C.realna+" + ( "+C.imaginarna+" * i )");
}

public static void Main()
{
    Complex[] tabK = new Complex[10]; //tabela 10 kompleksnih števil
    Random naklj = new Random();
    for (int i = 0; i < tabK.Length; i++)
    {
        tabK[i].realna = naklj.Next(-10, +10);
        tabK[i].imaginarna = naklj.Next(-10, +10);
        Console.WriteLine(i+". ti element "+IzpisiKompleksno(tabK[i]));
    }
    /*Napiši funkcijo za seštevanje dvek kompleksnih števil. Funkcija ima za parametra
      kompleksni števili in tudi vrne kompleksno število.
      Napiši funkcijo za množenje dveh kompleksnih števil.  Funkcija ima za parametra
      kompleksni števili in tudi vrne kompleksno število.
      Napiši funkcijo za izračun absolutne vrednosti kompleksnega števila.  Funkcija ima za
      parameter kompleksno število in vrne absolutno vrednost.*/
}
```

Vaja:

```
/* Za vodenje evidence o padavinah v zadnjem letu potrebujemo naslednje podatke: ime kraja in
podatke o količini padavin v zadnjih  12 mesecih (12 števil v polju/tabeli). Kreiraj ustrezno
podatkovno strukturo (ime strukture naj bo padavine).
Napiši funkcijo za vnos podatkov o padavinah za vseh 12 mesecev.
Napiši funkcijo povp(kraj), ki vrne povprečno mesečno količino padavin ustreznega kraja!*/

//deklaracija strukture padavine
struct padavine
```

```
{
    public padavine(string ime) //konstruktor
    {
         ime kraja = ime;
         kolicina = new double[12]; //inicializacija tabele padavin
    }
    public string ime kraja;
    public double[] kolicina; //deklaracija tabele padavin - inicializira jo konstruktur
};

static void vnos(padavine kraj) //metoda za vnos količine padavin kraja
{
    Console.WriteLine("\nVnos mesečne količine padavin za kraj: " + kraj.ime kraja);
    for (int i = 0; i < 12; i++)
    {
        Console.Write("Količina padavin v " + (i + 1) + ". mesecu: ");
        kraj.kolicina[i] = Convert.ToDouble(Console.ReadLine());
    }
}

static double povp(padavine kraj) //metoda za izračun povprečne količ.padavin določenega kraja
{
    double vsota = 0;
    for (int i = 0; i < 12; i++)
       vsota += kraj.kolicina[i];
    return (Math.Round(vsota / 12, 2)); //rezultat zaokrožimo na dve decimalki
}

//glavni program
static void Main(string[] args)
{
    padavine kraj1=new padavine("Kranj"); //struktura kraj1 je ustvarjena na kopici (zaradi
                                          //operatorja new)
    vnos(kraj1); //klic metode za vnos padavin kraja kraj1

    padavine kraj2 = new padavine("Ljubljana");//pa še ena struktura ustvarjena na kopici
    vnos(kraj2);

    Console.WriteLine("Povprečna količina padavin v mestu "+kraj1.ime_kraja+" je bila "
          +povp(kraj1));

    //pa še deklaracija spremenljivke vrednostnega tipa izpeljane iz strukture
    padavine kraj;
    Console.Write("Naziv kraja: ");
    kraj.ime_kraja = Console.ReadLine();//preberemo/vnesemo ime kraja
    Console.WriteLine("\nVnos mesečne količine padavin za kraj: " + kraj.ime_kraja);
    kraj.kolicina = new double[12]; //inicializacija tabele padavin za določen kraj
    for (int i = 0; i < 12; i++)
    {
        Console.Write("Količina padavin v " + (i + 1) + ". mesecu: ");
        kraj.kolicina[i] = Convert.ToDouble(Console.ReadLine());
    }
}
```

Naštevni tipi - enum

Naštevni tip (**enum**) je tip, v katerem je zbrana množica poimenovanih celoštevilskih konstant. Uporabljamo ga torej za imenovane vrednosti, ki so znane med prevajanjem. Temelji na celoštevilčnem osnovnem tipu. Osnovni tip je lahko katerikoli enostaven celoštevilčni tip: **byte**, **short**, **int**, **long** in njihove izvedenke (le **char** in **bool** ne!). Privzeto ima prva konstanta naštevnega tipa vrednost 0 (razen, če ne deklariramo druge vrednosti), vrednost vsake naslednje konstante pa je za eno večja.

Deklaracija in inicializacija naštevnega tipa

```
enum <ime> { vrednost1, vrednost2,  … };
```

Primer:

Deklarirajmo naštevni tip **Dan** s sedmimi konstantami, ki predstavljajo dneve v tednu:

```
enum Dan {Pon, Tor, Sre, Cet, Pet, Sob, Ned};
```

```
//Primer uporabe v glavnem programu
int x = (int)Dan.Ned;
int y = (int)Dan.Pet;
Console.WriteLine("Nedelja = " +x);
Console.WriteLine("Petek = "+ y);

Dan d=Dan.Sre;  //deklaracija in inicializacija nove spremenljivke d, ki je tipa Dan
```

Ker nismo določili drugače, ima konstanta **Pon** vrednost 0, konstanta **Tor** vrednost 1, itd.

Še en primer:

Deklarirajmo naštevni tip s katerim bomo deklarirali in inicializirali množico vseh ocen:

```
enum Ocene {Negativno = 1, Zadostno, Dobro, PravDobro, Odlično};
```

Prvi konstanti (**Negativno**) smo priredili vrednost, zato ima konstanta **Zadostno** vrednost 2, ostalim pa se vrednost povečuje za 1.

Vaja:

```
/*Deklarirajmo naštevni tip Dan, s konstantami, ki predstavljajo dneve v tednu.  */

enum Dan {Pon, Tor, Sre, Cet, Pet, Sob, Ned};

static string izpis(Dan d) //funkcija vrne polno ime dneva
{
    if (d == Dan.Pon) return "Ponedeljek";
    else if (d == Dan.Tor) return "Torek";
    else if (d == Dan.Sre) return "Sreda";
    else if (d == Dan.Cet) return "Četrtek";
    else if (d == Dan.Pet) return "Petek";
    else if (d == Dan.Sob) return "Sobota";
    else return "Nedelja";
}

static void Main(string[] args)
```

```
{
    Dan d;
    Console.WriteLine("Dnevi v tednu so: ");
    for (d = Dan.Pon; d <= Dan.Ned; d++) //operator ++ lahko uporabimo tudi za spremenljivko
                                         //naštevnega tipa
    Console.Write(izpis(d));
    //Elemente naštevnega tipa lahko izpišemo tudi takole
    for (d = Dan.Pon; d <= Dan.Ned; d++)
       Console.Write("Dan "+d+" ima vrednost "+(int)d+"\n");
}
```

Vaja:

```
/*Deklarirajmo naštevni tip Vrata s štirmi konstantami in vsem konstante hkrati tudi
inicializirajmo.*/

enum Vrata { Motor = 0, SportniAvto = 2, Limuzina = 4, HatcBack = 5 };

public static void Main()
{
    Vrata mojAvto = Vrata.SportniAvto;
    Vrata tvojAvto = Vrata.Motor;
    Vrata sosedovAvto = Vrata.Limuzina;

    Console.WriteLine("Ali ima "+ mojAvto+" več vrat kot "+ tvojAvto+("?");
    //Izpis: Ali ima SportniAvto več vrat kot Motor?
    if (mojAvto.CompareTo(tvojAvto) > 0)
       Console.WriteLine("Da\n");
    else Console.WriteLine("Ne\n");

    Console.WriteLine("Ali ima " + mojAvto + " več vrat kot " + sosedovAvto);
    //Izpis: Ali ima SportniAvto več vrat kot Limuzina?
    Console.WriteLine("{0}", mojAvto.CompareTo(sosedovAvto) > 0 ? "Da" : "Ne");
}
```

Vaja:

```
//Še en primer naštevnega tipa

enum Barve { Rdeča, Zelena, Modra, Rumena };

public static void Main()
{
    Barve mojaBarva = Barve.Modra;

    Console.WriteLine("Moja priljubljena barva je " + mojaBarva);
    Console.WriteLine("Vrednost moje priljubljene barve je "+ (int) mojaBarva);

    //ali

    Console.WriteLine("Vrednost moje priljubljene barve je " + Enum.Format(typeof(Barve),
                      mojaBarva, "d"));

    Console.WriteLine("Heksadecimalna vrednost moje priljubljene barve pa je "
                      +Enum.Format(typeof(Barve), mojaBarva, "x"));

    //metoda GetName vrne ime konstante določenega naštevnega tipa
    Console.WriteLine("Četrta vrednost barv v naštevnem tipu je "+ Enum.GetName(typeof(Barve),
                      3));  //izpis Rumena

    foreach (int i in Enum.GetValues(typeof(Barve)))
-       Console.Write(i+"  "); //Izpis: 0 1 2 3 4
    Console.WriteLine();
    Barve barva;
    for (barva=Barve.Rdeča;barva<=Barve.Rumena;barva++)
        Console.Write(barva + "  "); //Izpis: Rdeča Zelena Modra Rumena

    //ali pa

    foreach (string s in Enum.GetNames(typeof(Barve)))
        Console.Write(s); //Izpis: Rdeča Zelena Modra Rumena
}
```

Vaja:

```
/*Deklarirajmo naštevni tip ZaposleniTip z nekah elementi, nato pa strukturo Zaposleni, ki vsebuje podatek o tipu zaposlenega, njegovem imenu in oddelku. Napišimo tudi ustrezen konstruktor, nato pa prikažimo primer deklaracije spremenljivke vrednostnega tipa in spremenljivke ustvarjene na kopici*/

enum ZaposleniTip : byte
{
   Menedžer,
   Razvijalec,
   Administrator,
   Programer
}

struct Zaposleni
{
   public ZaposleniTip naziv;
   public string ime;
   public short oddelek;

   public Zaposleni(ZaposleniTip ZTip,string Zime,short Zodd)//konstruktor
   {
      naziv = ZTip;
      ime = Zime;
      oddelek = Zodd;
   }
}

public static void Main(string[] args)
{
   Zaposleni fred;//fred je vrednostna spremenljivka na skladu (konstruktor se ni izvedel)
   fred.oddelek = 40;
   fred.ime = "Fred";
   fred.naziv = ZaposleniTip.Razvijalec;

   Zaposleni mary = new Zaposleni(ZaposleniTip.Programer, "Mary", 11);//mary je referenčna
                    //spremenljivka, ustvarjena na kopici
   Zaposleni tom = new Zaposleni();//izvede se privzeti konstruktor: numerična polja dobijo
                                   //vrednost 0, polja tipa string pa so prazna
}
```

Vaja:

```
/*Definiraj naštevni tip Izobrazba, ki ima elemente (osnovna, poklicna, srednja, višja, visoka, magisterij, doktorat). Naštevni tip naj se prične s številko 3. Deklariraj še strukturo za delavca z elementi: ime, priimek, končana šola. Deklariraj spremenljivko za tako strukturo, vnesi podatke in jih nato izpiši.(Navodilo: za delavca vnesemo stopnjo izobrazbe s številko izpišemo pa element naštevnega tipa). */

public enum Izobrazba {osnovna = 3, poklicna,srednja, višja, visoka, magisterij, doktorat}

public struct Tdelavec
{
    public string ime;
    public string priimek;
    public Izobrazba KoncanaSola;
    public Tdelavec(string ime, string priimek, Izobrazba KoncanaSola)   //konstruktor
    {
        this.ime = ime;
        this.priimek = priimek;
        this.KoncanaSola = KoncanaSola;
    }
}

static void Main(string[] args)
{
    Tdelavec delavec;
    Console.WriteLine("Podatki o delavcu");
    Console.Write("  Ime: ");
    delavec.ime = Console.ReadLine();

    Console.Write("\n  Priimek: ");
```

```
    delavec.priimek = Console.ReadLine();
    Console.Write("\n  Izobrazba(3,4,5,6,7,8,9): ");

    //eksplicitna konverzija iz int v VrstaIzobrazevanja
    delavec.KoncanaSola = (Izobrazba)Convert.ToInt32(Console.ReadLine());
    Console.WriteLine(delavec.ime);
    Console.WriteLine(delavec.priimek);
    Console.WriteLine("Izobrazba: "+delavec.KoncanaSola);
}
```

Naloge:

- Deklariraj naštevni tip **Glasnost** s tremi elementi (**Tiho**, **Srednje**, **Glasno**). Element **Tiho** naj ima privzeto vrednost 1. Napiši funkcijo za uporabnikov vnos nastavitve glasnosti. Napiši še funkcijo za poimenski in vrednostni izpis elementov naštevnega tipa **Glasnost**.

- Kreiraj strukturo *CD* z elementi naslov, izvajalec, zvrst, založba, letnica in cena *CD*-ja. Element zvrst je naštevni tip, katerega vrednosti določi sam (npr. pop, rock, klasika, …) Napiši funkcijo vnos, ki prebere podatke o *CD*-ju. Vpisane podatke izpiši s pomočjo funkcije izpis. Funkcija vnos ima za parameter referenco na strukturo (klic po referenci), funkcija izpis pa strukturo (prenos po vrednosti). Uporabnik naj vpiše podatke za dva *CD*-ja, ki ju potem izpiši po abecedi po naslovih.

- Deklariraj naštevni tip *vrstaRože* (vrtnica=1, lilija, dalija, nagelj, iris, mešano..), ter naštevni tip *barvaRože* (rdeča=1, bela, rumena, vojoličasta, oranžna, modra, zelena …), nato pa kreiraj strukturo *roža* s komponentami *tip* (*vrstaRože*), *barva* (*barvaRože*), komadi (*int*) ter *cena* (*decimal*);

  - napiši konstruktor, ki določi tip ter barvo rože, za začetno ceno pa določi 0;
  - napiši program, ki od uporabnika zahteva izbiro rože, ter barve;
  - glede na izbiro rože ter barve deklariraj ustrezen objekt *šopek* tipa *enostavenŠopek*, vnesi še število rož ter ustrezno ceno za komad, nato pa izpiši ceno šopka;
  - deklariraj še netipizirano zbirko *šopek*, v kateri boš lahko dodal poljubno število objektov tipa *enostavenŠopek*. Vsebino zbirke *šopek* na koncu izpiši in obenem izračunaj ter izpiši njegovo ceno!

Rekurzija

Rekurzija je sposobnost programskega jezika, ki omogoča funkcijam, da kličejo same sebe. V takih funkcijah največkrat uporabljamo nek parameter, ki se zmanjšuje ali povečuje, in ko doseže neko vrednost, je rekurzije konec.

Da ima rekurzija smisel, pa ne sme klicati samo sebe v vsakem izvajanju, ker se potem izvajanje metode nikoli ne ustavi – to pa so tudi najpogostejše napake pri pisanju rekurzivnih metod. Zato je zelo pomembno dejstvo, da moramo pri pisanju rekurzivnih metod vedno paziti, da se rekurzivni klici ustavijo! Bistvo rekurzije je v tem, da izvajamo postopek na vedno manjšem naboru podatkov. Ker pri rekurziji podprogram oz. metoda kliče samega sebe, se le-ta navadno začne s pogojem (**if**), ki ustavi rekurzijo.

Ideja za take podprograme je podobna kot pri rekurzivnih definicijah v matematiki.

Primer:

Dano je Fibonaccijevo zaporedje : 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ... Vsak člen tega zaporedja (razen prvih dveh) je vsota predhodnih dveh členov. Označimo splošni, n-ti člen tega zaporedja z **F(n)**. Pravilo o tem, kako dobimo n -ti člen , lahko izrazimo rekurzivno takole :

$$F(n) = F(n-1) + F(n-2) \quad n= 3, 4, 5 \ldots.$$

Za prva dva člena zaporedja pa predpišemo še poseben pogoj :

$$F(1) = 1 \ , F(2) = 1$$

Formulacija je **rekurzivna**, ker se v enačbi za splošni člen **F(n)** sklicujemo tudi na člena **F(n-1)** in **F(n-2)**. Rekurzivno definicijsko enačbo lahko uporabimo tudi direktno za izračun določenega člena zaporedja, npr. za n=5. To storimo takole :

$$F(5) = F(4) + F(3)$$

Sedaj uporabimo pravilo za izračun **F(4)** in **F(3)** itn. ;

$$\begin{aligned} F(5)= F(4) + F(3) &= (F(3) + F(2)) + (F(2) + F(1)) = \\ &= ((F(2) + F(1)) + 1) + (1 + 1) = \\ &= ((1 + 1) + 1) + (1 + 1) = 5 \end{aligned}$$

Napišimo sedaj funkcijski podprogram ***int Fib(int N)*** , ki za dani **n** izračuna ustrezno Fibonaccijevo število . Zgornji rekurzivni princip lahko neposredno uporabimo takole :

če je n enako 1 ali 2, potem je Fib(n) = 1,
sicer pa je Fib(n) enako Fib(n-1) + Fib(n-2)

Zapis v C# :

```
static int Fib(int n)
{
    if ((n == 1) || (n == 2)) return 1;
    else return (Fib(n - 1) + Fib(n - 2));
}

//Primer klica te funkcije v glavnem programu
Console.WriteLine(Fib(33));  //Klic rekurzivne funkcije
```

Rekurzivna definicija se v gornjem podprogramu zrcali v tem, da podprogram kliče samega sebe (in to dvakrat) v stavku **`return (Fib(n - 1) + Fib(n - 2));`**

V funkcijskem podprogramu se torej sklicujemo na podprogram, ki še ni v celoti definiran, kar je v nasprotju s splošnim principom, da smemo uporabljati samo take pojme, ki so že definirani. Izkaže pa se, da C# lahko gornji program izvaja brez težav. Zato so taki rekurzivni podprogrami povsem legalni in predstavljajo izjemo glede na splošno načelo, da se smemo sklicevati le na objekte, ki so že definirani.

Takole pa bi napisali podprogram za Fibonaccijeva števila brez rekurzije :

```
static int Fib1(int n)  //Klasična (iterativna)  funkcija za izračun poljubnega člena
Fibonaccijevega zaporedja
{
    if (n == 1) return 1;
    if (n == 2) return 1;
    int F1 = 1, F2 = 1, P;
    for (int i = 0; i < n-2; i++)
    {
        P = F2;
        F2 = F1 + F2;
        F1 = P;
    }
    return F2;
}


//Primer klica te funkcije v glavnem programu
Console.WriteLine(Fib1(33)); //Klic iterativne funkcije
```

Za tako rešitev pravimo, da je formulirana **iterativno**, za razliko od **rekurzivne**.

Rekurzivna rešitev je krajša, enostavnejša in jasnejša, saj se v njej neposredno zrcali originalna matematična definicija Fibonaccijevega zaporedja in je zato tudi lažje razumljiva. Rekurzivna rešitev poleg tega tudi ne uporablja nobenih dodatnih spremenljivk. Glede varčnosti pomnilniškega prostora pa se izkaže, da je pri rekurzivni rešitvi le navidezna. Zahteve po rekurzivnih klicih se pri rekurzivnem podprogramu skrivajo v rekurzivnih klicih. Vsak rekurzivni klic namreč zahteva nekaj prostora, zapomniti si je potrebno tudi mesto, kam se je potrebno vrniti ob povratku iz podprograma. Za vsak rekurzivni klic pa si je potrebno zapomniti tudi vmesne rezultate. V resnici si vsak rekurzivni klic zgradi svoj povratni naslov in svojo verzijo vmesnih rezultatov oz. lokalnih spremenljivk. Ti podatki se nalagajo v pomnilniški sklad (stack). Sklad je enostavno shranjen v pomnilniku in zanj poskrbi sam prevajalnik, tako da programerju o njem ni potrebno razmišljati. Sklad tako raste in pada po potrebi. Izkaže se, da tak sklad navadno zahteva več prostora kot iterativni podprogrami, rekurzivni podprogrami pa so zato nekoliko potratnejši od iterativnih. Podobna je situacija glede hitrosti izvajanja. Klic podprograma traja nekoliko dlje kot enostavne operacije v iterativnih zankah. V našem zgledu je rekurzivni podprogram še posebno počasen, saj zahteva več seštevanj kot iterativni.

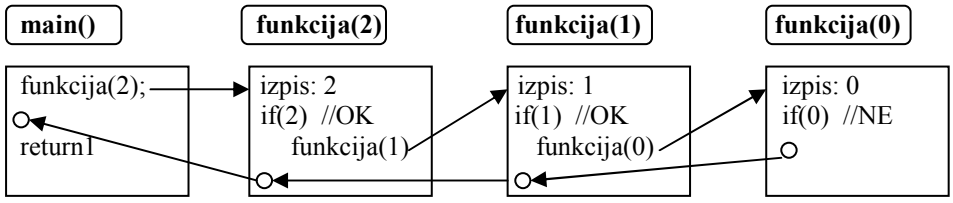
Rekurzivne formulacije imajo torej prednost pred iterativnimi v tem, da je programiranje bolj enostavno, jasno in razumljivo. Nekoliko manj učinkovit pa je rekurzivni podprogram glede časa izvajanja in prostora v pomnilniku. Od konkretnega primera pa je odvisno, za katero pot se bomo odločili.

Še en primer:

```
static void funkcija (int i)
{
    Console.WriteLine("Sem v funkciji s parametrom funkcija("+i+")");
    if ((i--)>0)    //pogoj je izpolnjen dokler i>0!
       funkcija(i);
}

static void Main(string[] args)
{
    funkcija(2);
}
```

Izpis programa : **Sem v funkciji s parametrom funkcija(2)**
Sem v funkciji s parametrom funkcija(1)

Sem v funkciji s parametrom funkcija(0)



Rekurzija je dobra, toda počasna, zato se ji poskušamo izogniti, če je to le mogoče, oz. če s tem ne bi preveč zakomplicirali algoritma.

Vaja:

```
//Rekurzivna funkcija za produkt 2n*(2*n-2)*(2*n-4)* ...4*2
static long produkt (int n)
{
    if (n==1) return 2;
    else return 2*n*produkt(n-1);
}

static void Main(string[] args)
{
    int n;
    Console.Write("Vnesi število členov produkta : ");
    n=Convert.ToInt32(Console.ReadLine());
    long c = produkt(n);
    Console.WriteLine("\nProdukt znaša : "+ c);
}
```

Vaja:

```
//Rekurzivna funkcija za pretvarjanje iz desetiškega v poljuben sestav
static void pretvori (int n,int s)
{
    if (n>0)
    {
        pretvori(n / s,s);
        Console.Write(n%s);
    }
    else Console.Write("\nPretvorjeno število : ");
}

static void Main(string[] args)
{
    int n, s;
    Console.Write("Vnesi število : ");
    n=Convert.ToInt32(Console.ReadLine());
    Console.Write("\nŠtevilski sestav : ");
    s=Convert.ToInt32(Console.ReadLine());
    pretvori(n, s);
    Console.WriteLine();
}
```

Vaja:

```
/*Dano je zaporedje 2, 8, 26, 80, 242. Z uporabo rekurzije napiši program, ki vrne n-ti
element zaporedja!  Napiši podprogram, ki izpiše elemente zaporedja od elementa z zaporedno
številko zac, do elementa z zaporedno številko kon !

Definicija zaporedja :   a( 1 ) = 2
                         a( n + 1 ) = 3 * a( n ) + 2    */

static int vrni(int n)
{
    if (n==1)
      return 2;
```

```
    else return (3 * vrni( n - 1 ) + 2) ;
}

static void izpisi(int zac,int kon)
{
    if  (zac <= kon)
    {
        Console.Write(vrni(zac)+"  ");
        izpisi(zac+1,kon );
    }
}

static void Main(string[] args)
{
    int zac,kon ;
    Console.Write("Vnesi začetni člen zaporedja : ");
    zac = Convert.ToInt32(Console.ReadLine());
    Console.Write("Vnesi končni člen zaporedja : ");
    kon = Convert.ToInt32(Console.ReadLine());
    if ((zac<=kon) &&(zac!=0)&&(kon!=0 ))
       izpisi(zac,kon);
    else Console.WriteLine("Napačen vnos!!!");
}
```

Vaja:

```
//Dana je rekurzivna funkcija.  Kakšen je izpis, če je v glavnem programu stavek
Console.WriteLine(pisem(97)); ?

        static int pisem( int n )
        {
           if (( n==1)||( n==2)) return 10;
           else
           {
              Console.Write(n+"  ");
              return ((n%4) + pisem(n/4 ));
           }
        }

        static void Main(string[] args)
        {
            Console.WriteLine(pisem(97));
        }
```

Vaja:

```
//Rekurzivna funkcija za izpis poljubnega stavka v obratni smeri
static void obrat(string str)
{
    if (str.Length > 0)
        obrat(str.Substring(1, str.Length - 1));
    else
        return;
    Console.Write(str[0]);
}

static void Main(string[] args)
{
    string s = "Tole je testni stavek";
    Console.WriteLine("Originalen stavek: " + s);

    Console.Write("Obratni stavek: ");
    obrat(s);

    Console.WriteLine();
}
```

Vaja:

```
/*Zamisli si, da pobiraš pločevinke in jih zlagaš v vrste po naslednjem pravilu: v prvi vrsti
je ena pločevinka, v drugi sta dve, v tretji tri itd. V vsaki novi vrsti je ena pločevinka več
```

```
kot v prejšnji. Koliko pa je vseh pločevink skupaj? Seveda je to odvisno od števila vrst.
Napiši program,ki bo z rekurzijo izračunal skupno število pločevink za izbrano število vrst.
Skupno število pločevink za izbrano število vrst lahko izračunamo, če poznamo zaporedno
številko vrste (potem vemo, da smo dodali prav toliko pločevink) in vsoto pločevink od prej.
Zapisano v bolj matematičnem jeziku: vsota (n) = vsota (n-1) + n */

static int trikotno (int n)
{
     if (n == 1)
     {
        return 1;
     }
     else
     {
        return n + trikotno (n - 1);
     }
}

static void Main(string[] args)
{
    Console.WriteLine("Število pločevink v petih vrstah: " + trikotno(5));
}
```

Naloge:

- Popravi rekurzivno funkcijo za izračun poljubnega člena Fibonaccijevega zaporedja tako, da bo uporabnik lahko podal poljubni začetni vrednosti :
- Fib(6, 2, 5) – Fibbonacijevo število reda 6, kjer sta začetni vrednosti 2 in 5.
- Napiši rekurzivni podprogram, ki dobi za parameter poljubno celo število (npr. 1234567) vrne pa celo število v katerem so cifre zapisane v obratnem vrstnem redu (v našem primeru 7654321).
- Sestavi rekurzivno funkcijo, ki izračuna največji skupni delitelj dveh pozitivnih števil, če veš, da velja:

  gcd(n, n) = n
  gcd(n, k) = gcd(n - k, k) za n > k
  gcd(n, k) = gcd(k, n) za n < k

- Sestavi program, ki s pomočjo rekurzije izpiše vse permutacije števil od 1 do n. Permutacije naj bodo izpisane v leksikografskem vrstnem redu.

  1 2 3
  1 3 2
  2 1 3
  2 3 1
  3 1 2
  3 2 1

- Napiši rekurzivno funkcijo za izpis **poštevanke** poljubnega števila!
- S pomočjo rekurzije napiši program, ki izračuna vsoto vrste $1^0 + 2^1 + 3^2 + 4^3 + \ldots + x^{(x-1)}$.

Funkcije - nadgradnja

Funkcija Main()

Ko poženemo katerikoli program, se le-ta začne izvajati v funkciji **Main()**, ki je lahko zapisana kjer koli v programu. Funkcija Main() je torej vstopna točka našega programa, zaključek te funkcije pa je obenem tudi zaključek našega programa. Deklarirana je **znotraj razreda** (pojem razreda bomo opisali v nadaljevanju), **mora biti statična** (pojem statične funkcije bo razložen v poglavju o razredih in objektih) in **ne sme biti public**. Vedno lahko obstaja le ena funkcija Main(). V glavi funkcije Main() je najprej deklaracija tipa, ki ga funkcija vrača, temu sledi ime funkcije in tabela parametrov funkcije Main() - **`string[] args`**. Pomen te tabele parametrov je naslednji: če pri zagonu programa za njegovim imenom zapišemo še parametre (če je parametrov več, so med seboj ločeni s presledki), se le-ti shranijo v to tabelo. Tabela parametrov ni obvezna in jo lahko izpustimo. Za razliko od C in C++ prvi argument te tabele **NI** ime programa.

Funkcija **Main()** lahko torej prav tako sprejme parametre. Ker pa v fazi razvoja programa ne moremo vedeti, kakšne parametre bo uporabnik napisal, v funkciji Main() ne moremo preprosto določiti, naj sprejme **int**, **double**, **char** ipd. Vsi parametri funkcije so shranjeni v tabeli, kjer je vsak element tabele nek niz znakov (konkreten parameter). Tako v funkciji Main() potrebujemo samo en parameter – ime tabele, v kateri so shranjeni njeni parametri

Primer :

```
//Napišimo naslednji program in ga shranimo pod imenom Main1
class Program
{
   static void Main(string[] args) //tabeli parametrov je ime args
   {
      //Izpis skupnega števila parametrov ukazne vrstice programa
      System.Console.WriteLine("Skupno število parametrov: "+args.Length);
      Console.ReadLine();
   }
}
```

Če ga poženemo :

Main1 arg1 arg2 arg3 arg4 <ENTER>

dobimo izpis :

```
Skupno število parametrov: 4
```

Še en primer:

```
//Napišimo naslednji program in ga shranimo pod imenom Main2
static void Main(string[] args)
{
   //Izpis skupnega števila parametrov ukazne vrstice programa
   System.Console.WriteLine("Skupno število parametrov: " + args.Length);
   for (int i = 0; i < args.Length; i++)
      Console.WriteLine(args[i]);
   //Lahko pa uporabimo tudi zanko foreach
   /*
   foreach (string s in args)
   {
      System.Console.WriteLine(s);
   }
   /*
```

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

Če ga poženemo :

Main2 Prvi Drugi Tretji <ENTER>

dobimo izpis :

```
Skupno število parametrov: 3
Prvi
Drugi
Tretji
```

V tabelo z imenom **args** se namreč zaporedoma shranijo vsi trije argumenti, ki smo jih napisali v ukazni vrstici ob klicu programa **Main2**. V elementu tabele z indeksom 0 je torej zapisan string ***Prvi***, v elementi z indeksom 1 je zapisan string ***Drugi***, v elementu z indeksom 2 pa string ***Tretji***.

Vsak niz, ki je ločen s presledkom, je za program nov argument. Včasih pa je to ovira, zato lahko argument, ki vsebuje presledke, zapremo med dvojne narekovaje.

Če torej program Main2 poženemo npr. takole:

Main2 "Celoten stavek je en sam argument" <ENTER>

dobimo izpis :

```
Skupno število parametrov: 1
Celoten stavek je en sam argument
```

Ker je **Main()** funkcija, je lahko tipa **void** (ne vrača ničesar) ali pa ima tip, kar pomeni da vrača rezultat. Če ima funkcija Main() tip, je to običajno **celoštevilski tip – int**. Vračanje celoštevilskega rezultata omogoča drugim programom, da le-tega uporabijo, ko se program v C# konča. Najpogosteje se rezultat uporabi npr. v kakih **batch** programih. Ko se program v C# izvaja npr. v Windows okolju, se vsaka vrnjena vrednost funkcije Main() shrani v okoljsko spremenljivko z imenom ERRORLEVEL. S testiranjem le-te, lahko nek **batch** program ugotovi, kakšen je bil rezultat programa v C# (oz. kakšno vrednost je vrnila funkcija Main()). Tradicionalno pomeni vrnjena vrednost **nič** (**0**) uspešen zaključek funkcije **Main()**.

Vaja:

Napiši program, ki bi ga klicali npr. **naravna 10 20** in ki izračuna in izpiše vsoto zapisanih števil (v našem primeru sta števili 10 in 20, vsota pa torej 30)!

```
static void Main(string[] args)
{ //Program sprejme dva parametra - dve celi števili in izračuna ter izpiše njuno vsoto
   if (args.Length == 2)
   {
       int prvo = Convert.ToInt32(args[0]);
       int drugo = Convert.ToInt32(args[1]);
       Console.WriteLine("Vsota: " + (prvo + drugo));
   }
   else Console.WriteLine("Napačno število parametrov!");
}
```

Vaja:

Napiši funkcijo, imenovano **Potenca()**, ki sprejme dva parametra: poljubno realno število in poljubno celo število. Program naj izračuna in izpiše vrednost ustrezne potence – osnovo in eksponent vnesemo v ukazni vrstici ob zagonu programa. Klic programa **Potenca 10 3** naj torej vrne 1000 (ker je 10^3 enako 1000).

```
static double potenca(double st1, int st2)
{
    double temp = st1;
    if (st2 == 0) return 1;
    if (st2 == 1) return st1;
    if (Math.Abs(st2) > 1)
    {
        for (int i = 1; i < Math.Abs(st2); i++)
           st1 = st1 * temp;
    }
    if (st2 > 0) return st1;
    else return 1 / st1;
}

static void Main(string[] args)
{
    if (args.Length == 2)   //preverimo, če smo v ukazni vrstici vnesi dva argumenta
    {
        double st1 = Convert.ToDouble(args[0]);
        int st2 = Convert.ToInt32(args[1]);
        Console.WriteLine("Rezultat: "+potenca(st1, st2));
    }
}
```

Vaja:

```
/*Program za izračun faktoriele poljubnega celega števila. Faktoriela je matematična
operacija, definirana takole:
N!=N * (N-1) * (N – 2) * … * 3 * 2 * 1
Konkretno: 7! = 7 * 6 * 5 * 4 * 3 * 2 * 1
*/
static long Fakt(long i)
{
    return ((i <= 1) ? 1 : (i * Fakt(i - 1)));
}

static int Main(string[] args)
{
   // Test vnosa vhodnega parametra (število, katerega faktorielo želimo izračunati)
   if (args.Length == 0)
   {
        Console.WriteLine("Vnesi prosim numerični argument.");
        Console.WriteLine("Uporaba: Main-Faktoriela <celo število>");
        return 1;
   }
   else
   {
        // Pretvorba vhodnega argumenta v celo število:
        long num = Convert.ToInt64(args[0]);
        Console.WriteLine("Faktoriela od {0} je {1}.", num, Fakt(num));
        return 0;
   }
}
```

Naloge:

- Napiši program, ki mu v ukazni vrstici podamo poljubno število besed, program pa med njimi poišče najdaljšo in najkrajšo besedo!

Preobložene (overloaded) metode

V C# imata lahko dve ali pa več metod enako ime, razlikovati pa se morata v deklaraciji parametrov. Za take metode pravimo da so preobložene – **overloaded**, proces pa se imenuje preobložitev (**overloading**). Preobložene funkcije so torej funkcije z enakim imenom, ki pa se razlikujejo v številu ali pa v tipu parametrov.

Primer:

```
//vse tri funkcije imajo enako ime, a se razlikujejo v številu parametrov - so preobložene

static void Funkcija(string drzava, string padavine, string temp)
{
    Console.WriteLine(drzava + ": " + padavine+" in "+temp);
}

static void Funkcija(string drzava, string padavine)
{
    Console.WriteLine(drzava+": "+padavine);
}

static void Funkcija()
{
    Console.WriteLine("Slovenija: sneg in mraz!");
}

static void Main(string[] args)
{
   Funkcija();                          //Izpis: Slovenija: sneg in mraz!
   Funkcija("Slovenija","dež");         //Izpis: Slovenija: dež!
   Funkcija("Slovenija","dež","toplo"); //Izpis: Slovenija: dež in toplo!
}
```

Vaja:

```
//Preobloženi funkciji za izračun vsote cifer naravnih števil med 0 in n, ter za izračun vsote
cifer med dvema naravnima številoma n in m
static int vsota(int n)
{
    int suma = 0;
    for (int i = 0; i <= n; i++)
       suma += i;
    return suma;
}
static int vsota(int n, int m)
{
    int suma = n;
    for (int i = 0; i <= m; i++)
        suma += i;
    return suma;
}

static void Main(string[] args)
{
    Console.WriteLine("Vsota cifer med 0 in 10 je "+vsota(10));
    Console.WriteLine("Vsota cifer med 5 in 10 je " + vsota(5,10));
}
```

Vaja:

```
//Še en primer preobloženih funkcij in avtomatske konverzije argumentov
static void f(int x)
{
     Console.WriteLine("Znotraj funkcije f f(int): " + x);
}

static void f(double x)
{
     Console.WriteLine("Znotraj funkcije f(double): " + x);
}
```

```
static void Main()
{
    int i = 10;
    double d = 10.1;

    byte b = 99;
    short s = 10;
    float x = 11.5F;

    f(i); // klic f(int)
    f(d); // klic f(double)

    f(b); // klic f(int), avtomatska konverzija byte v int
    f(s); // klic f(int), avtomatska konverzija short v int
    f(x); // klic f(double), avtomatska konverzija float v double
}
```

Varovalni bloki (osnove) – obravnava napak in izjem (Exceptions)

Tako kot vsakdanjem življenju se tudi pri pisanju programov oz. pri njihovem delovanju pojavljajo napake. Nekateri razvijalci, predvsem začetniki, skušajo napake odkriti in popraviti tako, da definirajo eno ali več posebnih globalnih spremenljivk in potem spremljati njihove vrednosti, ter na ta način skušati odkriti napake. Taka rešitev je seveda napačna in se je moramo izogibati.

C#, pa tudi drugi objektno orientirani jeziki kot orodje za odkrivanje napak, pa tudi za izboljšanje delovanja uporabljajo t.i. **izjeme** (**exceptions**). Izjeme so torej objektno orientirana rešitev za obdelavo napak v programih.

Idejna rešitev za obdelavo izjem je v tem, da ločimo kodo, ki predstavlja tok programa in kodo za obdelavo napak. Na ta način postaneta obe kodi lažji za razumevanje, saj se ne prepletata.

Za obdelavo izjem pozna **C#** naslednje bloke:

- blok za obravnavo prekinitev **try…catch …**,
- večkratni varovalni blok **try … catch … catch …**
- brezpogojni varovalni blok **try … finally …** .

Blok za obravnavo prekinitev: *try … catch …*

Kodo, ki bi jo sicer napisali v delu programa ali pa npr. v neki metodi, zapišemo v varovalnem bloku **try** (blok je vsak del kode napisane med oklepajema { in }). Drugi del začenja besedica **catch** in v bloku zapišemo enega ali več stavkov za obdelavo izjem oz. napak (t.i. ***catch* handlers**). Če katerikoli stavek znotraj bloka **try** povzroči izjemo (če pride do napake), se normalni tok izvajanja programa prekine (normalni tok izvajanja pomeni, da se posamezni stavki izvajajo od leve proti desni, stavki pa se izvajajo eden za drugim, od vrha do dna), program pa se nadaljuje v bloku **catch**, v katerem pa moramo seveda napako ustrezno obdelati.

Primer:

```
try
{
   int levo = Convert.ToInt32(Console.ReadLine());
   int desno = Convert.ToInt32(Console.ReadLine());
   float rezultat = levo / desno;
   Console.WriteLine(rezultat);
}
catch (System.Exception caught)
{
   // Vrsto napake izpišemo na zaslonu
   Console.WriteLine(caught.Message);
}
```

običajna programska koda

koda za obdelavo napake

Metode **Convert.ToInt32(..)**, **Convert.ToString(..)** skušajo napraviti ustrezne pretvorbe iz **stringa** v celo število oz iz realnega števila v **string**, pri čemer pa seveda lahko pride do napake (npr. če uporabnik vnese znak ki je različen od znakov '0' do '9'). Do napake lahko pride tudi v stavku, kjer je operacija za deljenje, saj se lahko zgodi, da je drugi operand enak 0. Varovalni blok to napako prestreže in izvede se stavek (oz. stavki) v bloku **catch**.

Če v stavkih znotraj bloka **try** pride do napake (izjeme), se program na tem mestu ne prekine, ampak se ostali stavki znotraj bloka **try** ne izvedejo, začnejo pa se izvajati stavki za obdelavo izjem (napak), ki so znotraj bloka **catch**. Če pa do napake v bloku **try** ne pride, se stavki v bloku **catch** NE izvedejo nikoli!

V zgornjem primeru je uporabljena kar **splošna** metoda za prestrezanje napake (`System.Exception`); le-ta se odzove/odkrije na vsako izjemo (napako), tudi tako, ki se je morda sploh ne zavedamo. V takem primeru lahko blok **catch** uporabimo tudi brez parametrov takole:

```
try
{
   // stavki, kjer lahko pride do napake
}
catch
{
   // obdelava napake . . .
}
```

Seveda pa obstajajo tudi metode za obdelavo posameznih vrst napak: **`System.FormatException`** (to metodo uporabimo na primer za obdelavo uporabnikovih napačnih vnosov podatkov), **`System.DivideByZeroException`**, **`System.EArgument.Exception`**, …). Vsaka od teh metod vrne ustrezno obvestilo o napaki, ki ga lahko izpišemo v konzolnem. Obvestilo o napaki je v tem primeru v angleščini; v kolikor pa želimo uporabnikom ponuditi prijazna obvestila o napaki, jih je potrebno napisati posebej, npr. takole:

```
try
{
   int levo = Convert.ToInt32(Console.ReadLine());
   int desno = Convert.ToInt32(Console.ReadLine());
   float rezultat = levo / desno;
   Console.WriteLine(rezultat);
}
catch
{
   Console.WriteLine("Napaka v podatkih!!!");
}
```

Tak način je za začetnika tudi najbolj priporočljiv in v nadaljevanju ga bomo tudi največkrat uporabljali.

Večkratni varovalni blok za obravnavo prekinitev *try … catch … catch …*

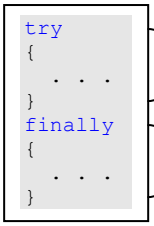
Različne napake seveda proizvedejo različne vrste izjem. Recimo, da imamo operacijo deljenja, pri kateri se lahko zgodi, da je drugi operand enak 0. Izjema, ki se pri tem zgodi, se imenuje **DivideByZeroException**. V takem primeru lahko napišemo večkratni **catch** blok, enega za drugim. Najprej ujamemo najnižji razred izjem, nazadnje pa najvišjega:

```
try
{
   int levo = Convert.ToInt32(textBox1.Text);
   int desno = Convert.ToInt32(textBox2.Text);
   float rezultat = levo / desno;
   textBox4.Text = Convert.ToString(rezultat);
}
catch (System.FormatException caught)
{
   Console.WriteLine("Napaka pri pretvarjanju podatkov!!!");
}
catch (System.DivideByZeroException caught)
{
   Console.WriteLine("Napaka pri deljenju z 0");
}
```

Seznam vseh možnih izjem dobimo, če v **Debug** meniju kliknemo opcijo **Exceptions**.

Brezpogojni varovalni blok *try … finally …*

Stavki v bloku **catch** se izvedejo samo ob prekinitvi (napaki), sicer pa se ne izvedejo. Kadar pa za del kode želimo, da se izvede v vsakem primeru, uporabimo brezpogojni varovalni blok **finally**.



Programska koda, ki bi jo napisali tudi če ne bilo varovalnega bloka;

Blok kode, ki se izvede vedno, ne glede ali je v zgornjem bloku prišlo do napake ali ne!

V praksi pogosto uporabljamo tudi kombinacijo obeh metod: blok za obravnavo prekinitev vgnezdimo v brezpogojni varovalni blok:

```
try
{
   // stavki, kjer lahko pride do napake
}
catch
{
   // obdelava napake
}
finally
{
    // blok, ki se izvede vedno, ne glede  na napako
}
```

Vaja:

```
//Varovalne bloke lahko tudi gnezdimo - na ta način lahko npr. uporabnika učinkovito seznanjamo z napakami pri vnašanju podatkov!
struct Narocilo
{
    public string CustomerName;
    public DateTime datumNarocila;
    public DateTime casNarocila;
    public uint SteviloEnot;
}

static void Main(string[] args)
{
   Narocilo nar1 = new Narocilo();
   Console.Write("Stranka:  ");
   string stranka = Console.ReadLine();
   try
   {
      Console.Write("Vnesi datum naročila(mm/dd/yyyy):  ");
      nar1.datumNarocila=Convert.ToDateTime(Console.ReadLine());
      try
      {
         Console.Write("Vnesi čas naročila(hh:mm AM/PM): ");
         nar1.casNarocila = Convert.ToDateTime(Console.ReadLine());
         try
         {
             Console.Write("Število enot: ");
             nar1.casNarocila = Convert.ToDateTime(Console.ReadLine());
         }
         catch
```

```
            {
                Console.WriteLine("Nepravilen vnos števila enot!");
            }
        }
        catch
        {
            Console.WriteLine("Nepravilen vnos ure naročila!");
        }
    }
    catch (FormatException)
    {
        Console.WriteLine("Nepravilen vnos datuma");
    }
}
```

Razredi in objekti (osnove)

Razred - class

Pojem **razred** (**class**) je temeljni pojem objektno (predmetno) orientiranega programiranja. Kreirati nov razred pomeni sistematično urediti podatke in informacije, ter manipulacije nad njimi v neko pomensko celoto. Takšno urejanje podatkov in informacij je nekaj, kar je običajen pojav tudi v vsakdanjem življenju in ne velja le za programiranje. Kot primer iz vsakdanjega življenja vzemimo pojem avto: vsi avtomobili imajo nekaj skupnih zmožnosti (lahko jih usmerjamo oz. vodimo, lahko jih zaustavimo, pospešujemo, itd.) in nekaj skupnih lastnosti oz. atributov (imajo volanski obroč, pogonski motor, kolesa itd.). Ko torej pomislimo na avto, takoj vemo, da gre za pojem oz. objekte ki si delijo prej omenjene skupne lastnosti in zmožnosti. Klasificiranje oz. razvrščanje pojmov v neko celoto je torej temeljna spretnost, ki je lastna vsem ljudem in brez katere si težko zamišljamo, kako bi ljudje razmišljali in komunicirali med seboj. Prav zaradi tega so tudi programerji prišli na idejo, da bi posamezne pojme, njihove lastnosti (atribute) in operacije nad njimi, združili v neko celoto, ki so jo poimenovali **razred** – **class**. Prav to pa je natanko tisto, kar nam ponujajo moderno zasnovani objektno orientirani programski jeziki, kot je npr. **Microsoft Visual C#**. Omogočajo definicijo novih razredov, ki v sebi združujejo lastnosti (atribute oz. podatke) in operacije nad njimi oz obnašanje (metode).

Sveta trojica predmetno usmerjenega programiranja so pojmi **enkapsulacija**, **dedovanje** in **polimorfizem**

Enkapsulacija

Glavni del pojma enkapsulacija predstavlja besedica kapsula: *enkapsulacija* pomeni postaviti nekaj v kapsulo. Za kapsulo je značilno, da ima vidno (*javno*) površino in nevidno (*zasebno*) notranjost, v kateri se nekaj nahaja. Vsak predmet ali pojem si lahko ponazorimo s kapsulo. Površino predstavljajo podatki in operacije, ki jih lahko od zunaj spreminjamo. Notranjost pa so zasebni deli predmeta, ki od zunaj niso dostopni.

Tudi v svetu programiranja poznamo dve pomembni uporabi oz razlagi pojma enkapsulacija:

- Princip, kako v enoto združimo podatke in metode, ki operirajo nad temi podatki. Gre torej za **kombinacijo metod in podatkov znotraj razreda**, z drugimi besedami **razvrščanju** oz. **klasifikaciji**;
- Notranje podatke in procese navzven skrijemo – gre torej za kontrolo **dostopa** do metod in podatkov v kapsuli, z drugimi besedami za **kontrolo uporabe razreda**.

Razred – **class** je programska struktura, ki zagotavlja sintakso in semantiko za podporo teh dveh dveh temeljnih načel enkapsulacije.

Kot primer kreirajmo razred ***krog***, ki vsebuje eno metodo (za izračun ploščine kroga) in eno lastnost oz. podatek (polmer kroga). Deklaracijo novega razreda začnemo z rezervirano besedico ***class***, ki ji sledi ime razreda, nato pa še telo razreda, zapisano med dvema zavitima oklepajema. V telesu razreda so **metode** (npr. metoda za ploščino), **spremenljivke**, ki jim pravimo tudi polja razreda (npr. polmer kroga) in pa lastnosti (**properties**) – o lastnostih bo več zapisanega kasneje.

```
class krog
{
     double Ploscina()   //metoda razreda
     {
         return Math.PI * polmer * polmer;
     }
     double polmer;      //polje razreda
}
```

Tako deklariran razred **krog** pa nima praktične uporabe, saj nismo upoštevali drugega načela enkapsulacije: **dostopnost**. Potem, ko smo enkapsulirali metode in polje znotraj razreda, smo pred temeljno odločitvijo, kaj naj bo javno, kaj pa zasebno. Vse kar smo zapisali med zavita oklepaja v razredu, spada v notranjost razreda, vse kar pa je pred prvim oklepajem in za zadnjim oklepajem pa je zunaj razreda. Z besedicama **public, private** in **protected** pa lahko kontroliramo, katere metode in polja bodo dostopna tudi od zunaj:

- Metoda ali polje je mišljeno kot privatno (**private**), kadar je dostopno le znotraj razreda. Tako polje oz. metodo deklariramo tako, da pred tipom metode oz. polja postavimo besedico ***private***.
- Metoda ali polje je mišljeno kot javno (**public**), kadar je dostopno tako znotraj, kot tudi izven razreda. Tako polje oz. metodo deklariramo tako, da pred tipom metode oz. polja postavimo besedico ***public.***.
- Metoda ali polje je mišljeno kot zaščiteno (**protected**), kadar je vidno le znotraj razreda, ali pa v **podedovanih** (**izpeljanih**) razredih.

Deklaracijo našega razreda **krog** sedaj napišimo še enkrat. Tokrat bomo metodo **Ploscina** deklarirali kot javno metodo, polje **polmer** pa kot privatno polje. Metodo **Ploscina** bomo v tem primeru lahko poklicali oz. uporabili tudi izven razreda, neposrednega dostopa do polja **polmer** pa ne bomo imeli.

```
class Krog
{
     public double Ploscina()   //javna metoda razreda
     {
         return Math.PI * polmer * polmer;
     }
     private double polmer;     //zasebno polje razreda
}
```

Če metode ali polja ne deklariramo kot **public** ali pa **private**, bo privzeto, da je metoda oz. polje **private**. Seveda pa velja, da če so vse metode in polja znotraj razreda privatne, razred nima praktične uporabe.

Če hočemo razred, ki smo ga definirali tudi uporabiti, moramo kreirati novo spremenljivko tega tipa, oz. kot temu pravimo v svetu objektnega programiranje, kreirati moramo novo **instanco** (primerek) razreda, ki ji pravimo tudi **objekt**. Novo spremenljivko tipa **Krog** lahko deklariramo tako kot vsako drugo spremenljivko. Naredimo primerjavo med kreiranjem in inicializacijo spremenljivk osnovnih (primitivnih) podatkovnih tipov in spremenljivk izpeljanih iz razredov (objektov). Osnovna sintaksa je enaka:

```
int i;   //deklaracija spremenljike i, ki je celoštevilčnega tipa
Krog K   //deklaracija spremenljivke K ki je tipa Krog (razred)
```

Če hočemo uporabiti vrednost katerekoli spremenljivke, moramo biti prepričani, da ta spremenljivka že ima vrednost.

```
int i;   //deklaracija spremenljike i, celoštevilčnega tipa
Console.WriteLine(i); //NAPAKA – uporaba neinicilaizirane spremenljivke
```

To pravilo velja seveda za vse spremenljivke, ne glede na njihov tip.

```
Krog K  //deklaracija spremenljivke k, ki je tipa Krog (razred)
Console.WriteLine(K); //NAPAKA – uporaba neinicilaizirane spremenljivke
```

Spremenljivke osnovnih podatkovnih tipov seveda lahko inicializiramo enostavno takole:

```
int i=10;   //deklaracija in inicilaizacija spremenljike i, celoštevilčnega tipa
Console.WriteLine(i); //OK!!

//ali pa takole

int j;
j = 10;
Console.WriteLine(j); //OK!!
```

Spremenljivkam, ki so izpeljane (ustvarjene) iz razreda pa vrednosti ne moremo prirejati tako kot spremenljivkam osnovnih (vrednostih) tipov. Uporabiti moramo rezervirano besedo ***new*** in za naš razred

poklicati ustrezen **konstruktor**. Novo spremenljivko (nov objekt oz. novo instanco) razreda **Krog** torej ustvarimo s pomočjo rezervirane besede **new** takole:

```
Krog K = new Krog();/*ustvarili smo nov objekt razreda Krog: pri tem smo uporabili privzeti
konstruktor – to je metodo Krog(), ki ima enako ime kot razred. */

//Ali pa takole

Krog K;
K = new Krog();
```

Primer:

```
class Zgradba//deklaracija razreda Zgradba z dvema javnima poljema. Razred nima nobene metode.
{
    public int kvadratura;
    public int stanovalcev;
}

static void Main(string[] args)
{
    Zgradba hiša = new Zgradba();  //nov objekt razreda Zgradba
    Zgradba pisarna = new Zgradba(); //nov objekt razreda Zgradba
    int kvadraturaPP; // kvadratura na osebo

    hiša.stanovalcev = 4;
    hiša.kvadratura = 2500;

    pisarna.stanovalcev = 25;
    pisarna.kvadratura = 4200;

    kvadraturaPP = hiša.kvadratura / hiša.stanovalcev;

    Console.WriteLine("Hiša ima:\n  " +
                      hiša.stanovalcev + " stanovalcev\n  " +
                      hiša.kvadratura + " skupna kvadratura\n  " +
                      kvadraturaPP + " kvadratura na osebo");

    Console.WriteLine();

    kvadraturaPP = pisarna.kvadratura / pisarna.stanovalcev;

    Console.WriteLine("Pisarna ima:\n  " +
                      pisarna.stanovalcev + " stanovalcev\n  " +
                      pisarna.kvadratura + " skupna kvadratura\n  " +
                      kvadraturaPP + " kvadratura na osebo");
}
```

Vaja:

```
/*Kreirajmo razred oseba s štirimi polji (ime, priimek, letnik in višina), ter dvema metodama:
metodo za prirejanje podatkov posameznim poljem in metodo za izpis podatkov*/

class Oseba
{
    //razred ima štiri zasebna polja
    private string ime;
    private string priimek;
    private int letnik;
    private int visina;

    //razred ima tudi dve javni metodi
    public void VpisiOsebo(string ime,string priimek,int letnik,int visina)
    {
       //ker imajo parametri metode enako ime kot polja razreda, smo za dostop do polj razreda
       //uporabili rezervirano besedo this - ta pomeni referenco na konkreten primerek razreda
       //(objekt), oz. referenco na polje konkretnega objekta
       this.ime = ime;
       this.priimek = priimek;
       this. letnik = letnik;
       this.visina = visina;
```

```
    }
    public void IzpisiOsebo()
    {
       Console.Write("Ime: " + ime + "\nPriimek: " + priimek + "\nStarost: " + letnik +
                     "\nVišina: " + visina+"\n");
    }
}

//glavni program
static void Main(string[] args)
{
   Oseba nogometas = new Oseba();//nova instanca (primerek) razreda Oseba
   nogometas.VpisiOsebo("David", "Becham", 1980, 181);
   nogometas.IzpisiOsebo();
   Oseba predsednik = new Oseba();//nova instanca (primerek) razreda Oseba
   predsednik.VpisiOsebo("George", "Bush", 1951, 173);
   predsednik.IzpisiOsebo();
}
```

Vaja:

```
/*Kreirajmo razred avto s štirimi zasebnimi polji (znamka, model, najvecjahitrost in teza),
ter dvema javnima metodama: metodo za inicializacijo (vnos) polj tega razreda, ter javno
metodo za izpis polj tega razreda. Nato kreirajmo dva objekta, ju inicializirajmo ter izpišimo
njune podatke*/

class avto
{
   //zasebna polja razreda avto
   private string znamka;
   private string model;
   private int najvecjahitrost;
   private double teza;

   public void Inicializacija()//javna metoda za vnos podatkov o konkretnem avtomobilu
   {
       Console.Write("\nZnamka: ");
       znamka = Console.ReadLine();
       Console.Write("\nModel: ");
       model = Console.ReadLine();
       Console.Write("\nNajvečja hitrost: ");
       najvecjahitrost = Convert.ToInt32(Console.ReadLine());
       Console.Write("\nTeža vozila: ");
       teza = Convert.ToDouble(Console.ReadLine());
   }
   public void IzpisPodatkov()//javna metoda za izpis podatkov o konkretnem avtomobilu
   {
      Console.WriteLine("\nIzpis podatkov o vozilu:\n");
      Console.WriteLine("Znamka: "+znamka
                        +"\nModel: "+model
                        +"\nNajvečja hitrost: "+najvecjahitrost
                        +"\nteža vozila: "+teza);
   }
}

static void Main(string[] args)
{
    avto mojavto=new avto();  //nov objekt tipa avto (nova instanca razreda)
    mojavto.Inicializacija(); //klic metode za inicializac.(vnos) podatkov za konkreten avto
    mojavto.IzpisPodatkov();  //klic metode za izpis podatkov o avtomobilu
    //kreiramo še drugi objekt, ga inicializiramo in izpišimo še njegove podatke
    avto prijateljevavto = new avto();
    prijateljevavto.Inicializacija();
    prijateljevavto.IzpisPodatkov();
}
```

Vaja:

```
/*Kreirajmo razred Prodajalec, ki bo imel zasebno polje zneski (tabela 12 števil tipa double),
ter javne metode za inicializacijo te tabele: metodo za določanje prodaje za posamezen mesec,
metodo za izračun celoletne prodaje in metodo za izpis celoletne prodaje*/
class Prodajalec
```

```
{
   private double[] zneski;     //zasebna tabela ki hrani mesečne zneske prodaje
   public void inicializacija() //metoda ki inicializira tabelo prodaje
   {
      zneski = new double[12];
   }
   public void doloci prodajo(int mesec, double znesek) //Vnos zneska prodaje za posam. mesec
   {
      if (mesec >= 1 && mesec <= 12 && znesek > 0)
         zneski[mesec - 1] = znesek; //[mesec -1] zato, ker v gredo tabeli indeksi od 0 d0 11
      else
         Console.WriteLine("Napačen mesec ali znesek prodaje");
   }
   public void izpisi letno prodajo()
   {
      Console.WriteLine("Skupna letna prodaja je : " + skupna_letna_prodaja() + " EUR");
   }
   public double skupna_letna_prodaja()   // funkcija ki vrne skupno letno prodajo
   {
      double vsota = 0.0;
      for (int i = 0; i < 12; i++)
         vsota += zneski[i];
      return vsota;
   }
};

static void Main(string[] args)
{
   Prodajalec p = new Prodajalec();   // tvorba objekta p tipa Prodajalec
   p.inicializacija();  //inicializacija tabele zneskov prodaje
   double znesek prodaje;
   for (int i = 1; i <= 12; i++)
   {
       Console.Write("Vnesi znesek prodaje za mesec "+i+": ");
       znesek_prodaje = Convert.ToDouble(Console.ReadLine());
       p.doloci prodajo(i, znesek prodaje);
   }
   p.izpisi letno prodajo();
}
```

Vaja:

```
/*Napišimo razred mojstring, v katerem bomo napisali nekaj javnih metod za delo s stringi, ki
naj pomenijo alternativo obstoječim metodam - npr. Length, Replace, ToUpper, ...*/

class mojstring
{
    private string stavek; //zasebno polje razreda mojstring
    public void DolociStavek(string st) //metoda za inicializacijo polja stavek
    {
        stavek = st;
    }
    public int dolzina()//metoda ki vrne število znakov v stringu
    {
        return stavek.Length;
    }
    //metoda za zamenjavo določenih znakov v stringu z novimi znaki
    public void ZamenjajZnak(string stari, string novi)
    {
        stavek = stavek.Replace(stari, novi);
    }
    public string IzpisStavka()
    {
        return stavek;
    }
    public void VelikeCrke()
    {
        stavek = stavek.ToUpper();
    }
}

static void Main(string[] args)
{
```

```
    mojstring st=new mojstring();
    st.DolociStavek("Razred mojstring: metodam za delo s stringi smo priredili slovenska
                    imena!");
    Console.WriteLine("\nV stavku:\n\n"+st.IzpisStavka()+"\n\nje "+st.dolzina()+" znakov!");
    st.ZamenjajZnak(" ", "X");//presledke nadomestimo z znakom X
    Console.WriteLine(st.IzpisStavka());
    st.ZamenjajZnak("X"," ");//vse znake "X" nadomestimo s presledki
    st.VelikeCrke();  //vse črke v stringu spremenimo v velike črke
    Console.WriteLine(st.IzpisStavka());
}
```

Naloge:

- Napiši razred **Kosarka**, za spremljanje košarkaške tekme. Voditi moraš število prekrškov za vsakega tekmovalca (10 igralcev), število doseženih točk (posebej 1 točka, 2 točki in 3 točke), ter metodo za izpis statistike tekme. Doseganje košev in prekrškov realiziraj preko metode *zadelProstiMet*(), *zadelZa2Tocki*, *zadelZa3Tocke* in *prekrsek*.

- Sestavi razred, ki bo v svoja polja lahko shranil ulico, številko nepremičnine ter vrsto nepremičnine. Ustvari poljuben objekt, ga inicializiraj in ustvari izpis, ki naj zgleda približno takole: Na naslovu Cankarjeva ulica 32, Kranj je blok.

Konstruktor

Konstruktor je metoda razreda, ki se uporablja za kreiranje novega primerka (instance) razreda. **Njegovo ime je vedno enako kot je ime njegovega razreda**. Namen konstruktorja pa je, da poskrbi za inicializacijo polj novo kreiranega objekta. Če konstruktorja ne napišemo sami, ga avtomatično za nas skreira prevajalnik. Z drugimi besedami, originalni razred **Krog**, za katerega smo zgoraj napisali eno samo metodi in eno polje, v resnici vsebuje še eno metodo: to je nevidni konstruktor, ki poskrbi za inicializacijo polja polmer. Ta metoda je povsem enaka, kot če bi napisali celoten razred takole:

```
class Krog
{
     public Krog()
     {
          polmer = 0.0;
     }

     public double Ploscina()
     {
         return Math.PI * polmer * polmer;
     }
     private double polmer;
}
```

Konstruktor, ki ga avtomatično generira prevajalnik, je vedno **public**, nima nobenega tipa (niti void), nima argumentov, vrednosti numeričnih polj postavi na 0, polja tipa **bool** postavi na **false**, vsem referenčnim poljem (spremenljivkam) pa priredi vrednost ***null***.

Ko je nov objekt inicializiran, lahko dostopamo do njegovih polj in uporabljamo njegove metode. Do javnih (**public**) polj (lastnosti razreda) in javnih metod dostopamo s pomočjo operatorja pika, tako kot pri strukturah.

```
Krog K = new Krog();
Console.WriteLine(K.Ploscina()); /*Izpis bo seveda enak 0, ker je privzeti konstruktor polju
polmer avtomatično dodelil vrednost 0!*/
```

Primer:

```
/*Kreiraj razred Pravokotnik z dvema poljema (dolžina in višina pravokotnika) in dvema
metodama (metoda za izračun ploščine in metoda za izračun obsega pravokotnika. Nato ustvari
```

```
nov objekt tipa Pravokotnik, določi stranice pravokotnika in s pomočjo metod razreda
Pravokotnik izračunaj njegovo ploščino in obseg*/

public class Pravokotnik
{
    //deklariramo polja; polja bodo javna (public), da bomo imeli do njih dostop izven razreda
     public int dolzina,visina;  //polji (lastnosti) razreda sta stranici pravokotnika

     public Pravokotnik()   //konstruktor (naš konstruktor je enak privzetemu konstruktorju!)
     {
         dolzina=0;
         visina=0;
     }
     public double ploscina()   //metoda, ki izračuna in vrne ploščino pravokotnika
     {
         return (dolzina*visina);
     }
     public double obseg() //metoda, ki izračuna in vrne obseg pravokotnika
     {
         return (2*dolzina+2*visina);
     }
};

static void Main(string[] args)
{
   Pravokotnik P = new Pravokotnik();  //na kopici ustvarimo nov objekt razreda pravokotnik
   P.dolzina = 10; //določimo dolžino pravokotnika
   P.visina = 8;   //določimo višino pravokotnika
   Console.WriteLine("Ploščina pravokotnika: " + P.ploscina()); //klic metode za izračun
                                                                //ploščine
   Console.WriteLine("Obseg pravokotnika: " + P.obseg());  //klic metode za izračun obsega
}
```

Vaja:

```
/*Kreiraj razred Kocka z enim privatnim poljem (rob kocke) in s tremi javnimi metodami (metoda
za določanje robu kocke, metoda za izpis robu kocke in metoda za izračun prostornine kocke.
Nato ustvari nov objekt tipa Kocka, določi rob kocke in s pomočjo metode razreda Kocka
izračunaj njegovo prostornino */

public class Kocka
{
   private double rob;
   public Kocka()  //konstruktor (enak je privzetemu konstruktorju)
   {
      rob = 0;
   }
   public void Nastavi_Rob(double x)   //metoda, ki omogoča nastavitev roba kocke
   {
      rob = x;
   }
   public double izpis_Roba()
   {
      return rob;
   }
   public double prostornina() //metoda, ki izracuna in vrne prostornino kocke
   {
      return (rob*rob*rob);
   }
};

static void Main(string[] args)
{
   Kocka K = new Kocka();  //na kopici ustvarimo novo instanco oz. nov objekt razreda Kocka
   Random naklj=new Random();
   K.Nastavi_Rob(Math.Round(naklj.NextDouble()*10,1)+1); //Rob kocke bo naključno realno
                                                         //število med 1 in 10
   Console.WriteLine("Rob kocke: " + K.izpis Roba());
   Console.WriteLine("Prostornina kocke: " + K. prostornina ());//klic metode za izračun
                                                               //prostornine
}
```

Vaja:

```
/*Napiši razred Kompleksno z dvema poljema (realna in imaginarna), ki naj predstavljata realno
in imaginarno komponento nekega kompleksnega števila. Napiši konstruktor, ki komponentama
priredi določeni vrednosti. Napiši še metodo, ki v primerni obliki izpiše poljuben objekt tega
razreda!*/

class Kompleksno
{
    private int realna, imaginarna;  //zasebni polji razreda
    public Kompleksno (int real, int imag) // konstruktor, ki poljema priredi celošt.vrednosti
    {
        realna = real;
        imaginarna = imag;
    }
    public void izpisi(string ime)  //metoda za izpis
    {
        Console.WriteLine(ime+" = "+realna + " + " + imaginarna + " * i");
    }
}

static void Main(string[] args)
{
    Kompleksno alfa=new Kompleksno(1,1); //kreiranje novega objekta tipa Kompleksno
    alfa.izpisi("alfa");

    Kompleksno beta=new Kompleksno(6,8); //kreiranje novega objekta tipa Kompleksno
    alfa.izpisi("beta");
}
```

Preobloženi (overloaded) konstruktorji

Poglejmo si še enkrat prvotno deklaracijo razreda Krog, ki ima deklarirano eno privatno polje (polmer) in javno metodo Ploscina().

```
class Krog
{
     public double Ploscina()
     {
         return Math.PI * polmer * polmer;
     }
     private double polmer;
}
```

Deklarirajmo novo spremenljivko tipa Krog, ji priredimo vrednost novega objekta Krog, nato pa pokličimo metodo Ploscina:

```
Krog K = new Krog();
Console.WriteLine(K.Ploscina());
```

Privzeti konstruktor v vsakem primeru postavi polmer na 0 (ker je polmer **private**, ga tudi ne moremo spremeniti), zaradi česar bo tudi ploščina objekta Krog vsakič enaka 0. Rešitev tega problema je v dejstvu, da je konstruktor prav tako metoda (sicer metoda posebne vrste), za metode pa velja, da so lahko preobložene (preobložene metode so metode, ki imajo enako ime, razlikujejo pa se v številu ali pa tipu parametrov). Z drugimi besedami, napišemo lahko svoj lasten konstruktor z vrednostjo polmera kot parametrom. Novi konstruktor ima seveda enako ime kot razred, ne vrača pa ničesar. Razred Krog bo sedaj izgledal takole:

```
class Krog
{
    public Krog(double inicPolmer) //naš lasten konstruktor
    {
       polmer = inicPolmer;
    }
    public double Ploscina()
    {
       return Math.PI * polmer * polmer;
```

```
    }
    private double polmer;
}
```

V primeru, da za nek razred napišemo lasten konstruktor velja, da prevajalnik ne generira privzeti konstruktor. Če pa smo napisali lasten konstruktor, ki sprejme en parameter in bi kljub temu želeli poklicati tudi privzeti konstruktor, ga moramo napisati sami. Pri tem pa moramo biti pozorni na dejstvo, da bodo vrednosti polj, ki jih v konstruktorju ne bomo inicializirali, še vedno ostala implicitno inicializirana na **0**, **false** ali pa **null**.

Destruktorji

Destruktor je metoda, ki počisti za objektom. Destruktor se torej izvede, ko objekt odmre (npr. ko se zaključi nek blok, ali pa se zaključi neka metoda, v kateri je bil objekt ustvarjen) – pokliče ga torej smetar (Garbage Collector). Destruktor torej sprosti pomnilniški prostor objekta (prostor, ki ga zasedajo njegovi podatki).

Destruktor ima enako kot konstruktor, enako ime kot razred sam in nima tipa. Ne sprejema argumentov, za razliko od konstruktorja pa pred destruktorjem stoji še znak '~ '. Med konstruktorjem in destruktorjem pa obstaja še ena velika razlika. Medtem ko je konstruktorjev določenega objekta lahko več, je destruktor vedno samo eden. Pomembno je tudi to, da destruktorji za razliko od konstruktorjev ne obstajajo v strukturah – obstajajo le v razredih.

```
class Krog
{
    ...  //deklaracija polj

    public Krog() //prvi konstruktor
    {
       ...
    }

    public Krog(double inicPolmer) //drugi konstruktor
    {
       ...
    }
    ~Krog() //destruktor
    {
       ...
    }
}
```

Glede destruktorjev je torej pomembno sledeče :

- Destruktor je metoda razreda, ki ima isto ime kot razred in dodan prvi znak '~ '.
- Destruktor je metoda razreda, ki nič ne vrača (pred njo ne sme stati niti void!) in nič ne sprejme.
- Razred ima lahko samo en destruktor.
- Destruktor se izvede, ko se objekt uniči.

Dogovor o poimenovanju

Pri poimenovanju javnih polj in metod se skušajmo držati pravila, ki ga imenujemo **PascalCase** (ker je bilo prvič uporabljeno v programskem jeziku Pascal). **Imena javnih polj in metod naj se začenjajo z veliko črko (**v zgornjem primeru **Ploscina**).

Pri poimenovanju zasebnih polj in metod pa se skušajmo držati pravila, ki ga imenujemo **camelCase. Imena zasebnih polj in metod naj se začenjajo z malo črko** (v zgornjem primeru **polmer)**.

Pri tako dogovorjenem poimenovanju je ena sama izjema. Imena razredov naj bi se začenjala z veliko črko. Ker pa se mora ime konstruktorja natančno ujemati z imenom razreda, se mora torej tudi ime konstruktorja v vsakem primeru začeti z veliko črko, ne glede na to ali je javen ali zaseben.

Primer:

Kot primer napišimo razred **Tocka**, ki naj ima dve privatni polji **x** in **y**, ter dva lastna konstruktorja. Prvi naj bo brez parametrov in v njegovem telesu le zapišimo stavek, ki bo v oknu izpisal, da je bil ta konstruktor poklican. Drugi konstruktor pa naj ima dva parametra, s katerima inicializiramo obe polji razreda. V telesu drugega konstruktorja napišimo stavek, ki izpiše vrednosti obeh polj.

```
class Tocka
{
    public Tocka()//Konstruktor brez parametrov
    {
        Console.WriteLine("Klican je bil privzeti konstruktor!");
    }

    public Tocka(int x, int y) //Preobloženi konstruktor z dvema parametroma
    {
        Console.WriteLine("x:{0} , y:{1}", x, y);
    }
    private int x, y;
}

static void Main(string[] args)
{
    Tocka A = new Tocka();  //Klic konstruktorja brez parametrov
    Tocka B = new Tocka(600, 800); //KLic konstruktorja z dvema parametroma
}
```

Vaja:

```
/*Napišimo razred Tocka, ki nja ima dve zasebni polji (koordinati točke), privzeti
konstruktor, ki obe koordinati postavi na 0, ter konstruktor z dvema parametroma, s katerima
inicializiramo koordinati nove točke. Napišimo še metodo, ki izračuna in vrne razdaljo točke
od neke druge točke*/

class Tocka
{
    public Tocka()//lasten privzeti konstruktor
    {
       x = 0;
       y = 0;
    }

    public Tocka(int initX, int initY) //Preobloženi konstruktor z dvema parametroma
    {
       x = initX;
       y = initY;
    }
    public double RazdaljaOd(Tocka druga) //metoda za izračun razrdalje od poljubne točke
    {
       int xRazd = x - druga.x;
       int yRazd = y - druga.y;
       return Math.Sqrt(Math.Pow(xRazd,2) + Math.Pow(yRazd,2));
    }
    private int x, y;
}

static void Main(string[] args)
{
    Tocka A = new Tocka();  //Klic konstruktorja brez parametrov
    Tocka B = new Tocka(600, 800); //Klic konstruktorja z dvema parametroma
    double razdalja = A.RazdaljaOd(B); //Izračun razdalje obeh točk
    Console.WriteLine("Razdalja točke A od točke B je enaka " + razdalja+" enot!");
}
```

Vaja:

```
/*Napišimo razred Zaposleni z enim samim zasebnim poljem (dohodek) in dvema konstruktorjema.
Prvi konstruktor naj ima kot parameter letni dodek zaposlenega, drugi pa naj imam dva
paramera: tedenski dohodek in število tednov. Napišimo še metodo za izpis skupnega dohodka*/

public class Zaposleni
{
    private double dohodek;
    public Zaposleni(int letniDohodek) //konstruktur
    {
        dohodek = letniDohodek;
    }

    public Zaposleni(int tedenskiDohodek, int številoTednov)//preobloženi konstruktur
    {
        dohodek = tedenskiDohodek * številoTednov;
    }

    public double vrniDohodek() //metoda ki vrne vrednost zasebnega polja dohodek
    {
        return dohodek;
    }
}

static void Main(string[] args)
{
    Zaposleni Janez = new Zaposleni(20000);  //nov objekt, izvede se prvi konstruktor
    Zaposleni Tina = new Zaposleni(550, 54); //nov objekt, izvede se drugi (preobloženi)
                                             //konstruktor
    Console.WriteLine("Janez ima letni dohodek : " + Janez.vrniDohodek()+" EUR" );
    Console.WriteLine("Tina ima letni dohodek : " + Tina.vrniDohodek()+ " EUR");
}
```

Naloge:

- Sestavi razred *denarnica*, ki bo omogočal naslednje operacije: dvig, vlogo in ugotavljanje stanja. Začetna vrednost se naj postavi s konstruktorjem. Ustvari tabelo desetih denarnic z naključno mnogo denarja in jih izpiši.

- Potrebujemo razred, ki bo hranil podatke o objektih tipa *Instrukcije* z naslednjimi komponentami: vrsta inštrukcije, število opravljenih ur in ali je inštrukcija možna. Razred naj ima tudi metode in sicer: inštrukcija se opravlja, inštrukcija se ne opravlja. Z osnovnim konstruktorjem določi vrednost (poljubno) vsem komponentam razreda. Z dodatnim konstruktorjem poskrbi za možnost nastavitve začetne vrednosti za vrsto inštrukcije, opravljene ure ter ali je inštrukcija možna. Na osnovi razreda *Instrukcije* napiši še testni program, ki bo kreiral in izpisal dva objekta razreda *Instrukcije* in sicer tako, da bodo začetne vrednosti pri prvem objektu nastavljene z osnovnim konstruktorjem, pri drugem objektu pa z dodatnim konstruktorjem.

- Sestavi razred, ki predstavlja osnovo za izdelavo programa, s pomočjo katerega bomo pregledovali rezultate nekega športnega tekmovanja. Sestavi razred *Tekmovalec*, ki ima naslednje komponente: startno številko, ime, priimek in klub. Vsa polja so tipa string. Napiši vsaj dva konstruktorja in pripravi ustrezne *get/set* metode za vse te podatke. Z metodo *public String toString()* naj se izpišejo podatki o tekmovalcih (startna številka, ime, priimek, klub).

- Sestavi razred *Pacient*, ki ima tri komponente: *ime*, *priimek*, *krvna_skupina*. Komponenti *ime* in *priimek* naj bosta *public*, *krvna_skupina private*, vse tri pa tipa *string*. Napiši vsaj dva konstruktorja:

  - prazen konstruktor, ki vse tri komponente nastavi na "NI PODATKOV",
  - konstruktor, ki sprejme vse tri podatke in ustrezno nastavi komponente.

  Z metodo *public String toString()* naj se izpišejo podatki o pacientu (ime, priimek, krvna skupina).

Lastnost (Property)

Polja so znotraj razreda običajno deklarirana kot zasebna (private). Vrednosti jim priredimo tako, da zapišemo ustrezen konstruktor (konstruktorje), ali pa da napišemo posebno javno metodo (metode) za prirejanje vrednosti polj. Ostaja pa še tretji način, ki je najbolj razširjen – za vsako polje lahko definiramo ustrezno lastnost (**property**), s pomočjo katere dostopamo do posameznega polja, ali pa z njeno pomočjo prirejamo (nastavljamo) vrednosti polja. Lastnosti (properties) v razredih torej uporabljamo za inicializacijo oziroma dostop do polj razreda (objektov). Lastnost (**property**) je nekakšen križanec med spremenljivko in metodo. Pomen lastnosti je v tem, da ko jo beremo, ali pa vanjo pišemo, se izvede koda, ki jo zapišemo pri tej lastnosti. Branje in izpis (dostop) vrednosti je znotraj lastnosti realizirana s pomočjo rezerviranih besed **get** in **set** – imenujemo ju ***pristopnika*** (**accessors**). **Accessor *get*** mora vrniti vrednost, ki mora biti istega tipa kot lastnost (**seveda pa mora biti lastnost istega tipa kot polje, kateremu je naemnjena**), v **accessor**-ju ***set*** pa s pomočjo implicitnega parametra **value** lastnosti priredimo (nastavimo) vrednost.

Navzven pa so lastnosti vidne kot spremenljivke, zato lastnosti v izrazih in prirejanjih uporabljamo kot običajne spremenljivke.

Primer:

```
class LastnostDemo
{
    int polje;              //zasebno polje

    public LastnostDemo()   //konstruktor
    {
       polje = 0;
    }

    public int MojaLastnost //deklaracija lastnosti (property) razreda LastnostDemo
    {
        get  //metoda get za pridobivaje vrednosti lastnosti polje
        {
            return polje  ;
        }
        set  //metoda set za prirejanje (nastavljanje) vrednosti lastnosti polje
        {
            polje = value;
        }
    }
}

static void Main(string[] args)
{
    LastnostDemo ob = new LastnostDemo();//nov objekt razreda LastnostDemo
    Console.WriteLine("Originalna vrednost ob.MojaLastnost: " + ob.MojaLastnost);

    ob.MojaLastnost = 100;
    Console.WriteLine("Vrednost ob.MojaLastnost: " + ob.MojaLastnost);

    Console.WriteLine("Prirejanje nove vrednosti -10 za ob.MojaLastnost");
    ob.MojaLastnost = -10;
    Console.WriteLine("Vrednost ob.MojaLastnost: " + ob.MojaLastnost);
}
```

Vaja:

```
/*Deklarirajmo razred Oseba z dvema poljema ( ime in priimek) ter javno metodo za
nastavljanje vrednosti obeh polj. Razred naj ima tudi lastnost PolnoIme, za prirejanje in
vračanje polnega imena osebe (imena in priimka). Ustvarimo nov objekt in demonstrirajmo
uporabo lastnosti PolnoIme*/

class Oseba
{
    private string _priimek; //zasebno polje razreda Oseba
    private string _ime;     //zasebno polje razreda Oseba

    public void NastaviIme(string ime, string priimek)  //javna metoda razreda
    {
        _priimek = priimek;
        _ime = ime;
    }
```

```
    public string PolnoIme   //javna lastnost razreda
    {
        get
        {
            return _ime + " " + _priimek;
        }
        set
        {
            string zacasna = value;
            string[] imena = zacasna.Split(' ');  //Celotno ime razdelimo na posamezne besede
                                                  //in jih spravimo  tabelo
            _ime = imena[0];  //za ime vzamemo prvi string v tabeli
             priimek = imena[imena.Length - 1]; //za priimek vzamemo drugi string v tabeli
        }
    }
}

static void Main(string[] args)
{
    Oseba politik = new Oseba();
    politik.NastaviIme("Nelson", "Mandela");
    Console.WriteLine("Polno ime osebe je " + politik.PolnoIme);  //Izpis: Polno ime osebe je
                                                                  // Neslon Mandela
    politik.PolnoIme = "George Walker Bush";
    Console.WriteLine("Polno ime osebe je " + politik.PolnoIme);  //Izpis: Polno ime osebe je
                                                                  // George Bush
    politik.PolnoIme = "France Prešeren";
    Console.WriteLine("Polno ime osebe je " + politik.PolnoIme);   //Izpis: Polno ime osebe je
                                                                   // France Prešeren
}
```

Za posamezno polje lahko napišemo le eno lastnost, v kateri s pomočjo stavkov **get** ali **set** dostopamo oz. nastavljamo vrednosti posameznega polja.

Besedici **get** ali **set** lahko izpustimo in dobimo write-only ali read only lastnosti.

Primer:

```
class MojRazred
{
   double A = 3;  //zasebno polje
   double B = 4;  //zasebno polje

   public double MojaVrednost //MojaVrednost je ReadOnly: vsebuje le get, ne pa tudi set
   {
      get { return A * B; }    //lastnost vrne vrednost produkta obeh polj
   }
}

static void Main(string[] args)
{
    MojRazred c = new MojRazred();   //nov objekt tipa MojRazred
    Console.WriteLine("MojaVrednost: "+c.MojaVrednost); //prikaz vrednosti lastnosti
                                                        //MojaVrednost
}
```

Naloge:

- Sestavi razred *Piramida*, ki predstavlja pokončno piramido, ki ima za osnovno ploskev kvadrat. V razredu hrani podatke o dolžini stranice osnovne ploskve in višino piramide. Obe komponenti naj imata tip dostopa **private**. Višina in stranica nista nujno celi števili. Napiši vsaj dva konstruktorja:
  - prazen konstruktor, ki ustvari piramido višine 1 in s stranico osnovne ploskve dolžine 1
  - konstruktor, ki sprejme podatka o višini in dolžini stranice osnovne ploskve
  - Napiši program, ki ustvari tabelo 50 naključnih piramid in med njimi poišče piramido z največjo višino.

Napiši **get** in **set** metode. V **set** metodah pazi na smiselnost podatkov (višina in dolžina stranice ne smeta biti negativni,...). Napiši tudi metodo *toString*, ki vrne niz s smiselnim izpisom podatkov o piramidi.

- Sestavi razred, kjer boš v objektih te vrste hranil ime države, njeno glavno mesto, površino (v km^2) in število prebivalcev. Pripravi ustrezne *get/set* metode za vse te podatke in vsaj dva konstruktorja. Ne pozabi tudi na smiselno metodo toString.

- Sestavi razred Igrača, ki ima tri privatne spremenljivke: tip igrače(tip *string*), število igrač na zalogi(tip *int*) ter ceno igrače(tip *double*).
  - sestavi prazen konstruktor;
  - sestavi konstruktor s tremi parametri;
  - napiši ustrezne *set* in *get* metode (pazi na smiselne vrednosti spremenljivk);
  - napiši metodo *toString*, ki naj smiselno izpiše, kateri tip igrače, koliko kosov je na zalogi in kakšna je cena;
  - dodaj še metodo *zalogaIgrac*, ki sprejme pozitivno celo število, če se je število igrač povečalo (dobava novih) ter negativno celo število če se je število igrač zmanjšalo (prodane igrače). Temu primerno spremeni število igrač na zalogi in pri tem pazi, da število igrač ne pade pod nič;
  - napiši metodo *znizanaCena*, ki sprejme celo število med 0 in 100 (%), to je za koliko procentov se bo znižala cena igrače, in nastavi novo ceno igrače;

  napiši glavni program, ki bo ustvaril pet tipov igrač, tri izmed njih znižal za 10, 20 in 50% ter dvema spremenil zalogo.

- V kemijskem laboratoriju večkrat preverjamo kislost snovi in jo definiramo s *pH* vrednostjo. Napiši razred Snov, ki bo imel tri komponente: *imeSnovi*, *ion* in *kislost*. Prvi dve sta tipa *string*, tretja pa *int*. Ker je kislost zelo pomemben podatek o snovi, zagotovi, da bo zagotovo pravilen. Zato bo ustrezna spremenljivka imela privaten dostop. Sestavi tudi ustrezni *get* in *set* metodi. Pazi, da boš tudi v konstruktorju s parametri poskrbel, da ne bo prišlo do napačne nastavitve. Če bo uporabnik podal napačno kislost, ustvari objekt, kjer za prva dva podatka napišeš, da sta neobstoječa, kislino pa pustiš tako, kot jo je vnesel uporabnik. Sestavi tudi konstruktor brez parametrov, ki naredi objekt, ki predstavlja vodo. Kreiraj tabelo snovi in napiši stavke za izpis vseh objektov.

Statične metode

Za lažje razumevanje pojma **statična metoda**, si oglejmo metodo **Sqrt** razreda **Math**. Če pomislimo na to, kako smo jo v vseh dosedanjih primerih uporabili (poklicali), potem je v teh klicih nekaj čudnega. Metodo **Sqrt** smo namreč vselej poklicali tako, da smo pred njo navedli ime razreda (**Math.Sqrt**) in ne tako, da bi najprej naredili nov objekt tipa **Math**, pa potem nad njim poklicali metodo **Sqrt**. Kako je to možno?

Pogosto se bomo srečali s primeri, ko metode ne bodo pripadale objektom (instancam) nekega razreda. To so uporabne metode, ki so tako pomembne, da so neodvisne od kateregakoli objekta. Metoda **Sqrt** je tipičen primer take metode. Če bila metoda **Sqrt** običajna metoda objekta izpeljanega iz nekega razreda, potem bi za njeno uporabo morali najprej kreirati nov objekt tipa **Math**, npr takole:

```
Math m = new Math();
double d = m.Sqrt(42.24);
```

Tak način uporabe metode pa bi bil neroden. Vrednost, ki jo v tem primeru želimo izračunati in uporabiti, je namreč neodvisna od objekta. Podobno je pri ostalih metodah tega razreda (npr. Sin, Cos, Tan, Log, …). Razred **Math** prav tako vsebuje polje **PI** (iracionalno število Pi), za katerega uporabo bi potemtakem prav tako potrebovali nov objekt. Rešitev je v t.i. **statičnih poljih oz.metodah**.

V C# morajo biti vse metode deklarirane znotraj razreda. Kadar pa je metoda ali pa polje deklarirano kot **statično** (**static**), lahko tako metodo ali pa polje uporabimo tako, da pred imenom polja oz. metode navedemo ime razreda. Metoda **Sqrt** (in seveda tudi druge metode razreda **Math**) je tako znotraj razreda **Math** deklarirana kot statična, takole:

```
class Math
{
    public static double Sqrt(double d)
    {
       . . .
    }
}
```

Zapomnimo pa si, da statično metodo ne kličemo tako kot objekt. Kadar definiramo statično metodo, le-ta **nima** dostopa do kateregakoli polja definiranega za ta razred. Uporablja lahko le polja, ki so označena kot **static** (statična polja). Poleg tega, lahko statična metoda kliče le tiste metode razreda, ki so prav tako označene kot statične metode. Ne-statične metode lahko, kot vemo že od prej, uporabimo le tako, da najprej kreiramo nov objekt.

Primer:

Napišimo razred točka in v njem statično metodo, katere naloga je le ta, da vrne string, v katerem so zapisane osnovni podatki o tem razredu

```
class Tocka
{
   public static string Navodila()//Statična metoda
   {
      string stavek="Vsaka točka ima dve koordinati/polji: x je abscisa, y je ordinata!";
      return stavek;
   }
}

//Zato, da pokličemo statično metodo oz. statično polje NE POTREBUJEMO OBJEKTA!!!
//Primer klica npr. v glavnem programu
Console.WriteLine(Tocka.Navodila());//Klic statične metode
```

Statična polja

Tako kot obstajajo statične metode, obstajajo tudi statična polja. Včasih je npr. je potrebno, da imajo vsi objekti določenega razreda dostop do istega polja. To lahko dosežemo le tako, da tako polje deklariramo kot statično.

```
class Test
{
    public const double Konstanta = 20;//Statično polje
}

//Zato, da pokličemo statično metodo oz. statično polje NE POTREBUJEMO OBJEKTA!!!
Console.WriteLine(Test.Konstanta);//klic statičnega polja
```

Vaja:

```
class Nekaj
{
    static int stevilo=0;  //Statično polje
    public Nekaj()         // Konstruktor
    {
         stevilo++;        //število objektov se je povečalo
    }

    public void Hello()
    {
        if (stevilo>3)
        {
            Console.WriteLine("Sedaj smo pa že več kot trije! Skupaj nas je že "+stevilo);
        }
        switch (stevilo)
        {
           case 1 : Console.WriteLine("V tej vrsticI  sem sam !!");
```

```
                    break;
            case 2 : Console.WriteLine("Aha, še nekdo je tukaj, v tej vrstici sva dva!!");
                    break;
             case 3 : Console.WriteLine("Opala, v tej vrstici smo že trije.");
                     break;
        }
    }
}

static void Main(string[] args)
{
    Nekaj a=new Nekaj();              //konstruktor za a
    a.Hello();
    {
        Nekaj b=new Nekaj();          //konstruktor za b
        b.Hello();
        {
            Nekaj c=new Nekaj();      //konstruktor za c
            c.Hello();
            {
                Nekaj d=new Nekaj(); //konstruktor za d
                d.Hello();
            }
        }
    }
}  //Konec programa-> Garbage Collector poskrbi za uničenje vseh objektov
```

Polje razreda je lahko statično a se njegova vrednost ne more spremeniti: pri deklaraciji takega statičnega polja zapišemo besedico **const** (const = Constant – konstanta). Besedica **static** pri deklaraciji konstantnega polja **NI** potrebna, pa je polje še vedno statično. Konstantno polje je torej avtomatično statično in je tako dostopno preko imena razreda in ne preko imena objekta! **Vrednost statičnega polja se NE da spremeniti**. Tako je npr. deklarirana konstanta **PI** razreda **Math**.

Primer:

```
class Test
{
    public static double kons1 = 3.14; //Statično polje
    public const double kons = 3.1416; //Konstantno polje je avtomatično tudi statično
    . . .
}

//Zato, da pokličemo statično polje NE POTREBUJEMO OBJEKTA!!!
Console.WriteLine(Test.kons1);//klic statičnega polja
Console.WriteLine(Test.kons);//klic konstantnega statičnega polja
Test.kons1= Test.kons1 + 1; //OK -> Statično polje LAHKO spremenimo
Test.kons= Test.kons + 1; //NAPAKA -> Konstantnega polja ne moremo spremeniti
```

Dedovanje (Inheritance) – izpeljani razredi

Kaj je to dedovanje

Dedovanje (***Inheritance***) je ključni koncept objektno orientiranega programiranja. Smisel in pomen dedovanja je v tem, da iz že zgrajenih razredov skušamo zgraditi bolj kompleksne, ki bodo znali narediti nekaj uporabnega. Dedovanje je torej orodje, s katerim se izognemo ponavljanju pri definiranju različnih razredov, ki pa imajo več ali manj značilnosti skupnih. Opredeljuje torej odnos med posameznimi razredi.

Vzemimo pojem **sesalec** iz biologije. Kot primer za sesalce vzemimo npr. konje in kite. Tako konji kot kiti počnejo vse kar počnejo sesalci nasploh (dihajo zrak, skotijo žive mladiče, …), a prav tako pa imajo nekatere svoje značilnosti (konji imajo npr. štiri noge, ki jih kiti nimajo, imajo kopita, …, obratno pa imajo npr. kiti

plavuti, ki pa jih konji nimajo, ..). V Microsoft C# bi lahko za ta primer modelirali dva razreda: prvega bi poimenovali ***Sesalec***, in drugega ***Konj***, in obenem deklarirali, da je ***Konj*** podeduje (inherits) ***Sesalca***. Na ta način bi med sesalci in konjem vzpostavili povezavo v tem smislu, da so vsi konji sesalci (obratno pa seveda ne velja!). Podobno lahko deklariramo razred z imenom ***Kit***, ki je prav tako podedovan iz razreda ***Sesalec***. Lastnosti, kot so npr, kopita ali pa plavuti pa lahko dodatno postavimo v razred ***Konj*** oz. razred ***Kit***.

Bazični razredi in izpeljani razredi

Sintaksa, ki jo uporabimo za deklaracijo, s katero želimo povedati, da razred podeduje nek drug razred, je takale:

```
class IzpeljaniRazred : BazičniRazred
{
    . . .
}
```

Izpeljani razred deduje od bazičnega razreda. Za razliko od C++, lahko razred v C# deduje največ en razred in ni možno dedovanje dveh ali več razredov. Seveda pa je lahko razred, ki podeduje nek bazični razred, zopet podedovan v še bolj kompleksen razred.

Primer:

Radi bi napisali razred, s katerim bi lahko predstavili točko v dvodimenzionalnem koordinatnem sistemu. Razred poimenujmo ***Tocka***:

```
class Tocka //bazični razred
{
    public Tocka(int x, int y)  //konstruktor
    {
          //telo konstruktorja
    }
    //telo razreda Tocka
}
```

Sedaj lahko definiramo razred za tridimenzionalno točko z imenom Tocka3D, s katerim bomo lahko delali objekte, ki bodo predstavljali točke v tridimenzionalnem koordinatnem sistemu in po potrebi dodamo še dodatne metode:

```
class Tocka3D : Tocka  //razred Tocka3D podeduje razred Tocka
{
   //telo razreda Tocka3D - tukaj zapišemo še dodatne metode tega razreda!
}
```

Klic konstruktorja bazičnega razreda

Vsi razredi imajo vsaj en konstruktor (če ga ne napišemo sami, nam prevajalnik zgenerira privzeti konstruktor). Izpeljani razred avtomatično vsebuje vsa polja bazičnega razreda, a ta polja je potrebno ob kreiranju novega objekta inicializirati. Zaradi tega mora konstruktor v izpeljanem razredu poklicati konstruktor svojega bazičnega razreda. V ta namen se uporablja rezervirana besedica ***base***:

```
class Tocka3D : Tocka  ////razred Tocka3D podeduje razred Tocka
{
    public Toca3D(int z)
           :base(x,y)  //klic bazičnega konstruktorja Tocka(x,y)
    {
       //telo konstruktorja Tocka3D
    }
    //telo razreda Tocka3D
}
```

Če bazičnega konstruktorja v izpeljanem razredu ne kličemo eksplicitno (če vrstice **:base(x,y)** ne napišemo!), bo prevajalnik avtomatično zgeneriral privzeti konstruktor. Ker pa vsi razredi nimajo privzetega konstruktorja (v veliko primerih napišemo lastni konstruktor), moramo v konstruktorju znotraj izpeljanega razreda obvezno najprej klicati bazični konstruktor (rezervirana besedica ***base***). Če klic izpustimo (ali ga pozabimo napisati) bo rezultat prevajanja *compile-time error*:

```
class Tocka3D : Tocka  //razred Tocka3D podeduje razred Tocka
{
    public Tocka3D(int z)
    //NAPAKA - POZABILI smo klicati bazični konstruktor razreda Tocka
    {
       //telo konstruktorja Tocka3D
    }
    //telo razreda Tocka3D
}
```

Določanje oz. prirejanje razredov

Poglejmo še, kako lahko kreiramo objekte iz izpeljanih razredov. Kot primer vzemimo zgornji razred **Tocka** in iz njega izpeljani razred **Tocka3D**.

```
class Tocka //bazični razred
{
   //telo razreda Tocka
}
```

```
class Tocka3D: Tocka //razred Tocka3D podeduje razred Tocka
{
    //telo razreda Tocka3D
}
```

Iz razreda **Tocka** izpeljimo še dodatni razred **Daljica**

```
class Daljica:Tocka  //razred Daljica podeduje razred Tocka
{
    //telo razreda Daljica
}
```

Kreirajmo sedaj nekaj objektov:

```
Tocka A = new Tocka();
Tocka3D B = A;  //NAPAKA - različni tipi


Tocka3D C = new Tocka3D ();
Tocka D = C; //OK!
```

Nove metode

Razredi lahko vsebujejo več ali manj metod in slej ko prej se lahko zgodi, da se pri dedovanju v izpeljanih razredih ime metode ponovi – v izpeljanem razredu torej napišemo metodo, katere ime, število in tipi parametrov se ujemajo z metodo bazičnega razreda. Pri prevajanju bomo zato o tem dobili ustrezno opozorilo - ***warning***. Metoda v izpeljanem razredu namreč v tem primeru prekrije metodo bazičnega razreda. Če npr. napišemo razred **Tocka** in nato iz njega izpeljemo razred **Tocka3D**,

```
class Tocka   //bazni razred
{
     private int x, y;  //polji razreda Tocka

     public  void Izpis()  //metoda za izpis koordinat razreda Tocka
     {
          Console.WriteLine("Koordinate točke:\nx = " + x + "\ny = " + y);
     }
```

```
}

class Tocka3D : Tocka   //razred Tocka3D je izpeljan iz razreda Tocka
{
     private int z;       //dodatno polje razreda Tocka3D

     //Metoda Izpis prepiše istoimensko metodo razreda Tocka
     public void Izpis()
     {
         Console.Write("z = " + z + "\n");
     }
}
```

nam bo prevajalnik zgeneriral opozorilo, s katerim nas obvesti, da metoda **Tocka3D.Izpis** prekrije metodo **Tocka.Izpis**:

Program se bo sicer prevedel in tudi zagnal, a opozorilo moramo vzeti resno. Če namreč napišemo razred, ki bo podedoval razred **Tocka3D**, bo uporabnik morda pričakoval, da se bo pri klicu metode **Izpis** pognala metoda bazičnega razreda, a v našem primeru se bo zagnala metoda razreda **Tocka3D**. Problem seveda lahko rešimo tako, da metodo **Izpis** v izpeljanem razredu preimenujemo (npr. **Izpis1**), še boljša rešitev pa je ta, da v izpeljanem razredu eksplicitno povemo, da gre za NOVO metodo – to storimo z uporabo operatorja ***new***.

```
class Tocka   //bazni razred
{
     private int x, y;  //polji razreda Tocka

     public  void Izpis()  //metoda za izpis koordinat razreda Tocka
     {
          Console.WriteLine("Koordinate točke:\nx = " + x + "\ny = " + y);
     }
}

class Tocka3D : Tocka   //razred Tocka3D je izpeljan iz razreda Tocka
{
     private int z;       //dodatno polje razreda Tocka3D

     //Z operatorjem new napovemo, da ima razred Tocka3D SVOJO LASTNO  metodo Izpis
     new public void Izpis()
     {
         Console.Write("z = " + z + "\n");
     }
}
```

Virtualne metode

Pogosto želimo metodo, ki smo je napisali v bazičnem razredu, v višjih (izpeljanih) razredih skriti in napisati novo metodo, ki pa bo imela enako ime in enake parametre. Eden izmed načinov je uporaba operatorja ***new*** za tako metodo, drug način pa je z uporabo rezervirane besede ***virtual***. Metodo, za katero želimo že v bazičnem razredu označiti, da jo bomo lahko v nadrejenih razredih nadomestili z novo metodo (jo prekriti), označimo kot **virtualno** (***virtual***), npr.:

```
//virtualna metoda v bazičnem razredu - v izpeljanih razredih bo lahko prekrita(override)
public virtual void Koordinate()
{...}
```

V nadrejenem razredu moramo v takem primeru pri metodi z enakim imenom uporabiti rezervirano besedico ***override***, s katero povemo, da bo ta metoda prekrila/prepisala bazično metodo z enakim imenom in enakimi parametri.

```
//izpeljani razred - besedica override pomeni, da smo s to metodo prekrili bazično metodo
public override void Koordinate()
{ ...}
```

Ločiti pa moramo razliko med tem, ali neka metoda prepiše bazično metodo (**override**) ali pa jo skrije. Prepisovanje metode (**overriding**) je mehanizem, kako izvesti drugo, novo implementacijo iste metode – ***virtualne*** in ***override*** metode so si v tem primeru sorodne, saj se pričakuje, da bodo opravljale enako nalogo, a nad različnimi objekti (izpeljanimi iz bazičnih razredov, ali pa iz podedovanih razredov). Skrivanje metode (**hiding**) pa pomeni, da želimo neko metodo nadomestiti z drugo – metode v tem primeru niso povezane in lahko opravljajo povsem različne naloge.

Primer:

Naslednji primer prikazuje zapis virtualne metode **Koordinate()** v bazičnem razredu in **override** metode **Koordinate()** v izpeljanem razredu.

```
class Tocka   //bazični razred
{
    private int x, y;   //polji razreda Tocka

    //metoda Koordinate je virtualna, kar pomeni, da jo lahko prepišemo (override)
    public virtual void Koordinate()  //metoda za izpis koordinat razreda Tocka
    {
        Console.WriteLine( "Koordinate točke:\nx = "+x + "\ny = " + y);
    }
}

class Tocka3D : Tocka   //razred Tocka3D je izpeljan iz razreda Tocka
{
    private int z;      //dodatno polje razreda Tocka3D

   //Metoda Koordinate je označena kot override - prepiše istoimensko metodo razreda Tocka
    public override void Koordinate()
    {
       base.Koordinate();  //klic bazične metode Koordinate razreda Tocka
       Console.Write("z = "+z+"\n");
    }
}
```

Prekrivne (Override) metode

Kadar je bazičnem razredu neka metoda označena kot virtualna (**virtual**), jo torej lahko v nadrejenih razredih prekrijemo/povozimo (***override***). Pri deklaraciji takih metod (pravimo jim **polimorfne** metode) z uporabo rezerviranih besed **virtual** in **override**, pa se moramo držati nekaterih pomembnih pravil:

- Metoda tipa **virtual** oz. **override** NE more biti zasebna (ne mre biti **private**);
- Obe metodi, tako *virtualna* kot *override* morata biti identični: imeti morata enako ime, enako število in tip parametrov in enak tip vrednosti, ki jo vračata;
- Obe metodi morata imeti enak dostop. Če je npr. virtualna metoda označena kot javna (**public**), mora biti javna tudi metoda **override**;
- Prepišemo (prekrijemo/povozimo) lahko le virtualno metodo. Če metoda ni označena kot virtualna in bomo v nadrejenem razredu skušali narediti **override**, bomo dobili obvestilo o napaki;
- Če v nadrejenem razredu ne bomo uporabili besedice **override**, bazična metoda ne bo prekrita. To pa hkrati pomeni, da se bo tudi v izpeljanem razredu izvajala metoda bazičnega razreda in ne tista, ki smo napisali v izpeljanem razredu;
- Če neko metodo označimo kot **override**, jo lahko v nadrejenih razredih ponovno prekrijemo z novo metodo.

Vaja:

Razred **Tocka** in razred **Tocka3D**, ki je izpeljan iz razreda **Tocka** sta implementirana v naslednjem primeru:

```
class Tocka   //bazni razred
{
```

```
    private int x, y;         //polji razreda Tocka

    public Tocka(int x,int y) //konstruktor
    {
        this.x = x;
        this.y = y;
    }
    //metoda Koordinate je virtualna, kar pomeni, da jo lahko preobložimo (naredimo override)
    public virtual void Koordinate()  //metoda za izpis koordinat razreda Tocka
    {
        Console.WriteLine( "Koordinate točke:\nx = "+x + "\ny = " + y);
    }
}

class Tocka3D : Tocka   //razred Tocka3D je izpeljan iz razreda Tocka
{
    private int z;      //dodatno polje razreda Tocka3D

    public Tocka3D(int x,int y,int z) //konstruktor - parametra x in y potrebujemo za
         : base(x,y)                  //dedovanje bazičnega konstruktorja razreda Tocka
    {
       this.z = z;
    }
    //Metoda Koordinate prepiše istoimensko metodo razreda Tocka
    public override void Koordinate()
    {
       base.Koordinate();  //klic bazične metode Koordinate razreda Tocka
       Console.Write("z = "+z+"\n");
    }
}

static void Main(string[] args)
{
    Tocka A = new Tocka(1,1);           //Nov objekt razreda Tocka
    A.Koordinate();                     //klic metode Koordinate razreda Tocka
    Tocka3D A3D = new Tocka3D(1, 2, 3); //Nov objekt razerda Tocka3D
    A3D.Koordinate();                   //klic preobložene metode Koordinate razreda Tocka3D
}
```

Vaja:

```
/*Napišimo razred krog z zasebnim poljem polmer in virtualno metodo ploscina, nato pa še
razred kolobar. Razred kolobar naj deduje razred krog, dodano naj ima še eno zasebno polje
notranjiPolmer in svojo lastno(override) metodo polščina*/
class krog   //bazni razred
{
    private int polmer;   //polje razreda krog

    //metoda ploscina je virtualna, kar pomeni, da jo lahko preobložimo (override)
    public virtual double ploscina()
    {
        return Math.PI * polmer * polmer;
    }
    public int Polmer //lastnost(property) razreda krog, za dostop in inic. polja polmer
    {
        get
        {
           return polmer;
        }
        set
        {
           polmer = value;
        }
    }
}

class kolobar : krog   //razred kolobar podeduje razred krog
{
    //ker kolobar deduje krog, že pozna polje polmer
    private int notranjiPolmer; //dodatno polje razreda kolobar

    //metoda ploscina prepiše istoimensko metodo bazičnega razreda krog
    public override double ploscina()
    {
```

```
        return Math.PI * (Polmer * Polmer - notranjiPolmer*notranjiPolmer);
    }

    //ker kolobar deduje krog, že pozna njegovo lastnost Polmer
    //dodatna lastnost (property) razreda kolobar, za dostop in inic. polja notranjiPolmer
    public int NotranjiPolmer
    {
        get
        {
            return notranjiPolmer;
        }
        set
        {
            notranjiPolmer = value;
        }
    }
}

static void Main(string[] args)
{
    Random naklj = new Random();
    krog k=new krog();              //nov objekt razreda krog
    k.Polmer = naklj.Next(1, 10);   //polje polmer inicializiramo preko lastnosti Polmer

    kolobar kol = new kolobar();    //nov objekt razreda kolobar
    //tudi polje polmer objekta kol inicializiramo preko lastnosti Polmer
    kol.Polmer = naklj.Next(1, 10);
    //polje notranjiPolmer inicializiramo preko lastnosti notranjiPolmer
    kol.NotranjiPolmer = naklj.Next(1, 10);

    //izpis ploščine kroga na 4 decimalke
    Console.WriteLine("Ploščina kroga: {0:F4}",k.ploscina());

    //izpis ploščine kolobarja na 4 decimalke
    Console.WriteLine("Ploščina kolobarja: {0:F4}", kol.ploscina());
}
```

Vaja:

```
//Deklarirajmo razred Oseba, nato pa razred Kmetovalec, ki naj podeduje razred Oseba. Razred
Oseba naj ima polje ime, razred Kmetovalec pa še dodatno polje velikostPosesti. Za oba razreda
napišimo tudi konstruktor in virtualno metodo ToString za izpis podatkov o posameznem
objektu!*/

//glavni program
static void Main(string[] args)
{
    ArrayList osebe = new ArrayList();  //zbirka oseb

    Oseba mike = new Oseba("mike");
    osebe.Add(mike);
    Console.WriteLine("Osebe:");
    //izpišemo seznam vseh oseb
    foreach (Oseba p in osebe)
       Console.WriteLine(p.ToString());

    ArrayList kmetovalci = new ArrayList();  //zbirka kmetovalcev
    Kmetovalec john = new Kmetovalec("john", 10);
    Kmetovalec bob = new Kmetovalec("bob", 20);
    kmetovalci.Add(john);
    kmetovalci.Add(bob);
    Console.WriteLine();
    Console.WriteLine("Kmetovalci:");
    //izpišemo seznam vseh kmetovalcev
    foreach (Kmetovalec f in kmetovalci)
       Console.WriteLine(f.ToString());

    ArrayList vsiSkupaj = new ArrayList();  //zbirka vseh skupaj
    //napolnimo skupno zbirko vseh oseb
    foreach (Oseba o in osebe)
      vsiSkupaj.Add(o);
    foreach (Kmetovalec f in kmetovalci)
      vsiSkupaj.Add(f);
    Console.WriteLine();
```

```
    Console.WriteLine("Vsi skupaj:");
    //izpišemo seznam vseh oseb
    foreach (Oseba p in vsiSkupaj)
       Console.WriteLine(p.ToString());
}

public class Oseba            //bazični razred
{
    public string ime;        //bazično polje

    public Oseba(string ime)  //bazični konstruktor
    {
       this.ime = ime;
    }

    public virtual string ToString()  //virtualna metoda bazičnega polja
    {
        return "Ime=" + ime;
    }
}

public class Kmetovalec : Oseba  //razred Kmetovalec deduje razred Oseba
{
    int velikostPosesti;  //dodatno polje razreda Kmetovalec

    public Kmetovalec(string ime, int velikostPosesti) //konstruktor razreda Kmetovalec
        : base(ime)                                    //podeduje bazično polje ime
    {
        this.velikostPosesti = velikostPosesti;
    }

    public override string ToString()  //metoda ToString prekrije bazično metodo ToString
    {
        return base.ToString() + "; Kvadratnih metorv = " +
        velikostPosesti.ToString();
    }
}
```

Naloge:

- Napiši razred Kocka tako, da bo izpeljan iz razreda Kvadrat. Razreda naj poleg svojih podatkov vsebujeta še privzeti konstruktor, metode za postavitev vrednosti podatkov in metode za izračun površine, obsega in volumna.

Polimorfizem - mnogoličnost

V izpeljanih razredih se srečamo še z enim temeljnim pojmom objektno orientiranega programiranja – to je pojem **polimorfizem** oz. **mnogoličnost**. Pojem **polimorfizem** označuje princip, da lahko različni objekti razumejo isto sporočilo in se nanj odzovejo vsak na svoj način. Pomeni tudi, da je ista operacija lahko implementirana na več različnih načinov oz. zanjo obstaja več metod. Dedovanje in polimorfizem sta lastnosti, ki predstavljata osnovo objektnega programiranja in omogočata hitrejši razvoj in lažje vzdrževanje programske kode.

Kot primer za prikaz polimorfizma deklarirajmo razred **`OsnovniRazred`**, ki ima eno samo metodo z imenom **`Slika`**. Ker želimo, da bo vsak objekt, ki bo izpeljan iz tega razreda (tudi tisti v podedovanih – izpeljanih razredih) ohranil sebi lastno obnašanje ob klicu metode **`Slika`**, moramo uporabiti **polimorfno redefinicijo**. Polimorfno redefinicijo omogočimo z že znano definicijo **virtualne metode**. V telesu te metode bomo za vajo in zaradi enostavnosti prikazali le neko sporočilno okno!

```
public class OsnovniRazred   //temeljni razred
{
   public virtual void Slika()  //polimorfna redefinicija - omogočimo jo z virtualno metodo
   {
        Console.WriteLine("Osnovni objekt!");
   }
```

```
}
```

Ker smo metodo **Slika** definirali kot **virtualno**, smo s tem napovedali, da bodo izpeljani objekti lahko uporabljali **svojo** (**prekrivno** oz. **override**) metodo **Slika**, ki pa bo imela enako ime.

Razred **OsnovniRazred** bo naš temeljni razred, iz katerega bomo tvorili **izpeljane** razrede in v njih tvorili **nove** objekte. Ker je metoda **Slika** virtualna to pomeni, da lahko v izpeljanih razredih to metodo prekrijemo (**override**) z metodo, ki bo imela enako ime a drugačen pomen (drugačno vsebino).

Napišimo sedaj še tri razrede, ki naj bodo izpeljani iz razreda **OsnovniRazred** in ki imajo svojo metodo **Slika**. Pred tipom takih metod mora stati besedice **override**, ki označuje, da se bodo objekti izpeljani iz teh razredov na to metodo odzivali vsak na svoj način. Osnovni pogoj pa je, da imajo take **override** metode enako raven zaščite (npr. vse so public) , enako ime in enake parametre kot jih ima osnovna virtualna metoda v bazičnem razredu.

```
public class Crta : OsnovniRazred  //Crta je razred, ki podeduje razred OsnovniRazred
{
   public override void Slika()    //preobložena metoda razreda Crta
   {
       Console.WriteLine("Črta.");   //Telo override metode je seveda lahko drugačno!!!
   }
}

public class Krog : OsnovniRazred //Tudi Krog je razred, ki podeduje razred OsnovniRazred
{
   public override void Slika()
   {
       Console.WriteLine ("Krog.");  //Telo override metode je seveda lahko drugačno!!!
   }
}

public class Kvadrat: OsnovniRazred //Tudi Kvadrat je razred, ki podeduje razred OsnovniRazred
{
   public override void Slika()     //Telo override metode je seveda lahko drugačno!!!
   {
       Console.WriteLine ("Kvadrat.");
   }
}
```

Poglejmo sedaj, kako bi te štiri razrede sedaj uporabili in na primeru izpeljanih objektov prikazali princip polimorfizma. V ta namen kreirajmo tabelo objektov. Ime tabele je **dObj**, tabela pa naj bo inicializirana tako, da so v njej lahko štiri objekti tipa OsnovniRazred.

```
OsnovniRazred[] dObj = new OsnovniRazred[4];//tabela objektov
```

Ker so razredi **Crta**, **Krog** in **Kvadrat** izpeljani iz bazičnega razreda **OsnovniRazred**, jih lahko priredimo isti tabeli **dObj**. Če te zmožnosti ne bi bilo, bi morali za vsak nov objekt, izpeljan iz kateregakoli od teh štirih razredov, kreirati svojo tabelo. Dedovanje pa nam omogoča, da se vsak od izpeljanih objektov obnaša tako kot njegov bazični razred. Naslednjo kodo lahko zapišemo npr. v dogodek **Click** nekega gumba, ali pa neko opcijo menija.

```
dObj[0] = new Crta();          //konstruktor objekta dObj[0]
dObj[1] = new Krog();          //konstruktor objekta dObj[1]
dObj[2] = new Kvadrat();       //konstruktor objekta dObj[2]
dObj[3] = new OsnovniRazred();//konstruktor objekta dObj[3]

foreach (OsnovniRazred objektZaRisanje in dObj)
{
     objektZaRisanje.Slika();  //klic metode Slika ustreznega objekta
}
```

Ko je tabela inicializirana, lahko npr. s **foreach** zanko pregledamo vsakega od objektov v tabeli. Zaradi načela polimorfizma se v zagnanem programu vsak objekt obnaša po svoje, pač odvisno od tega, iz katerega razreda je bil izpeljan. Ker smo v izpeljanih razredih prepisali virtualno metodo **Slika** , se ta metoda v izpeljanih objektih

izvaja različno, pač glede na njeno definicijo v izpeljanih razredih. Pri vsakem prehodu zanke bomo tako dobili drugačno sporočilno okno.

Uporaba označevalca protected

Uporaba označevalcev **private** in **public** predstavlja obe skrajni možnosti dostopa do članov razreda (oz. objekta). Javna polja in metode so dostopne vsem, zasebna polja in metode pa so dostopne le znotraj razreda. Pogosto pa je koristno, da bazični razred dovoljuje izpeljanim razredom, da le-ti dostopajo do nekaterih bazičnih članov, obenem pa ne dovoljujejo dostopa razredom, ki niso del iste hierarhije. V takem primeru uporabimo za dostop označevalec ***protected***.

Nadrejeni razred torej lahko dostopa do člana bazičnega razreda, ki je označen kot **protected**, kar praktično pomeni, da bazični član, ki je označen kot **protected**, v izpeljanem razredu postane javen (**public**). Javen je tudi v vseh višjih razredih.

V primeru, ko pa nek razred ni izpeljani razred, pa nima dostopa do članov razreda, ki so označeni kot **protected** – znotraj razreda, ki ni izpeljani razred, je torej član razreda, ki je označen kot **protected** enak zasebenemu članu (**private**).

Delo z datotekami in podatkovnimi tokovi

Datoteka je zaporedje podatkov, ki so shranjeni na disku ali kakem drugem mediju (CD-ROM, disketa, …). Vse datoteke, ki so dostopne na računalniku, so organizirane v datotečni sistem, ki je sestavljen iz **direktorijev** ali **map**. Vsaka datoteka ima **ime** in je shranjena v **mapi**. Polno ime datoteke dobimo tako, da zložimo skupaj njeno ime in mapo.

Imenski prostor (knjižnjica) System.IO

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

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

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

Vse te metode so statične metode, zaradi česar jih lahko kličemo direktno preko imena razreda in ne preko imen objektov. Pred uporabo kateregakoli od razredov imenskega prostora **System.IO**, je potrebno ta imenski prostor navesti v stavku **using**.

```
using System.IO;  //dodajanje imenskega prostora
```

Če tega stavka na začetku ne bi bilo, bi se morali na vse razrede imenskega prostora **System.IO** sklicevati preko **imena** tega imenskega prostora.

Najpomembnejše metode razreda Directory

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

Primer uporabe:

Znak \ v nekem stringu je v splošnem napoved neke t.i. **escape (ubežne) sekvence** (**\n** – nova vrstica, **\t** – Tabulator, **\r** – **Return**, \\ - **Backslash**, **\"** - dvojni narekovaj). Če pa pred definicijo **stringa** postavimo znak **@**, potem znak \ **NE** predstavlja začetek neke **escape sekvence**.

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

Najpomembnejše metode razreda File

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

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

Primer uporabe:

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

Vaja:

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

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

    string datoteka = "Vaja1";
    string datoteka1 = "Vaja2";
    //Za datoteko Vaja1 preverimo obstoj v imeniku C:\Vaje C#: če že obstaja,jo prej pobrišimo
    if (File.Exists(dir + "\\" + datoteka))
    {
         Console.WriteLine("Datoteka Vaja1 že obstaja in bo pobrisana!");
         File.Delete(dir + "\\" + datoteka);  //če datoteka obstaja, jo pobrišemo
    }
    //skreirajmo NOVO datoteko Vaja1 v imeniku C:\Vaje C#
    File.Create(dir + "\\" + datoteka);
    //skreirajmo NOVO datoteko Vaja2 v imeniku C:\Vaje C#
    File.Create(dir + "\\" + datoteka1);
    //S pomočjo metode GetFiles imena vseh datotek izbrane mape shranimo v tabelo datoteke
    string[] datoteke = Directory.GetFiles(dir);
    Console.WriteLine("Seznam datotek imenika Vaje C# na disku C:");
    foreach (string imedatoteke in datoteke) //izpišimo imena vseh datotek mape C:\Vaje C#
    Console.WriteLine(imedatoteke);
```

```
}
```

Vaja:

```
/*Napiši program Kopiraj, ki z ukazne vrstice sprejme imeni dveh datotek: program naj prvo
datoteko prekopira v drugo.Pred kopiranjem obvezno preveri obstoj datotek*/

if (args.Length==2)
{
   string izvornaDatoteka = args[0];
   string ponornaDatoteka=args[1];

   if (!File.Exists(izvornaDatoteka))
   {
      Console.WriteLine("Datoteka " + args[0] + " ne obstaja!");
      Environment.Exit(0); //izhod iz programa
   }
   if (File.Exists(ponornaDatoteka))
   {
      Console.WriteLine("Ponorna datoteka " + args[1] + " že obstaja! Kopiranje ni mogoče!");
      Environment.Exit(0); //izhod iz programa
   }
   try
   {
      File.Copy(izvornaDatoteka, ponornaDatoteka);
   }
   catch
   {
      Console.WriteLine("Napaka pri kopiranju!");
   }
}
   else
      Console.WriteLine("Napačno število parametrov!");
```

Naloge:

- Preveri, če na disku D že obstaja pot d:\C#\Vaje\Datoteke. Če še ne obstaja jo kreiraj, nato pa skreiraj datoteki Vaja1.txt in Vaja2.txt. Če pot že obstaja, najprej preveri, če datoteki Vaja1.txt in Vaja2.txt že obstajata – če ne, ju skreiraj.

- Preveri, če na disku C že obstaja mapa s poljubnim imenom, ki ga vneseš preko tipkovnice. Če mapa že obstaja, ugotovi in izpiši koliko datotek vsebuje. Izpiši tudi imena vseh datotek.

- Kreiraj poljubno mapo in v njej datoteko s poljubnim imenom – imeni vnese uporabnik preko tipkovnice. Pred kreiranjem preveri, če mapa oz. datoteka že obstaja.

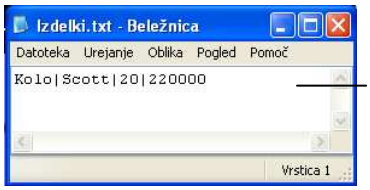
- Kreiraj drevesno strukturo imenikov d:\APJ\C#\VajeC#\Letnik_1

Delo s podatkovnimi tokovi

Ko uporabljamo razrede imenskega prostora **System.IO** za delo z vhodno izhodnimi operacijami, lahko uporabljamo dve vrsti datotek: **tekstovne** datoteke in **binarne** datoteke. V tekstovnih datotekah so zapisani le **berljivi** znaki (črke, številke, presledek, znaki !"#$\%&/()= in podobni znaki), ter znaka za **konec vrste** in **konec datoteke**. Tekstovno datoteko lahko odpremo v vsakem urejevalniku (na primer v Notepadu) ali jo izpišemo na zaslon (ukaz **type** (DOS) oziroma **cat** (Linux)). Binarne datoteke pa vsebujejo podatke v **binarnem** (dvojiškem) zapisu. Najmanjša enota zapisa v binarni datoteki je podatek tipa **int**. Binarne datoteke niso berljive (urejevalniki besedil običajno ne znajo prikazati posebnih znakov, namesto njih prikazujejo »kvadratke«).

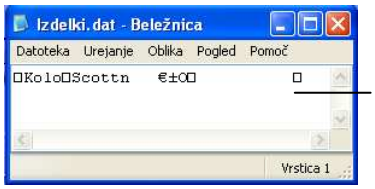
V tekstovnih datotekah so torej vsi podatki shranjeni kot **tekstovni znaki**, ali pa kot **zaporedje znakov** – **stringi**. Pogosto so posamezni sklopi znakov (besede, polja, …) med seboj ločeni s posebnimi ločili (npr

znakom |, ali pa z znakom *vejica* ipd), datoteke pa vsebujejo tudi znake **end of line** (znak za konec vrstice). Tekstovne datoteke imajo torej vrstice.



Izgled vsebine **tekstovne** datoteke odprte z Beležnico (v datoteki je ena sama vrstica).

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



Izgled vsebine **binarne** datoteke, če jo odpremo z Beležnico

Vrste datotek

Tip datoteke	Razlaga
Text – tekstovna datoteka	Datoteka, ki lahko vsebuje le znake (podatkovni tip **char**) ali pa zaporedje znakov (**stringe**). Pogosto so le ti med seboj ločeni z ločili (npr znakom |, znakom *vejica* ipd.
Binary – binarna datoteka	Datoteka, ki vsebuje množico različnih podatkovnih tipov.

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

Tokovi podatkov za delo z datotekami

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

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

Vrste podatkovnih tokov

Razred	Razlaga
Stream	Splošen podatkovni tok

FileStream	Zagotavljanje dostopa do vhodnih in izhodnih datotek (podatkovni tok namenjen datotekam).
StreamReader	Uporablja se za branje tekstovnih podatkov v podatkovni tok (npr. iz tekstovne datoteke).
StreamWriter	Uporablja se za zapisovanje toka tekstovnih podatkov (npr. v tekstovno datoteko).
BinaryReader	Uporablja se za branje binarnih podatkov v podatkovni tok (npr. iz binarne datoteke).
BinaryWriter	Uporablja se za zapisovanje toka binarnih podatkov (npr. v binarno datoteko).

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

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

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

Uporaba razreda FileStream

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

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

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

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

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

Tabela načinov kreiranja datoteke – FileMode

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

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

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

Tabela načinov manipulacije s podatki – FileAccess

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

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

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

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

Primer:

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

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

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

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

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

Najpogostejše metode razreda FileStream

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

Delo s tekstovnimi datotekami

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

Pisanje podatkov v tekstovno datoteko

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Vaja:

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

string dir = @"C: \Tekstovna";

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

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

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

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

Vaja:

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

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

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

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

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

Vaja:

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

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

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

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

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

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

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

Vaja:

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

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

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

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

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

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

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

Naloge:

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

  1._ _ 4
  2._ _ 6
  3._ _ 1
  …

Branje podatkov iz tekstovne datoteke

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

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

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

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

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

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

Primer:

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

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

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

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

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

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

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

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

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

Vaja:

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

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

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

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

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

    //3) Branje vrstice za vrstico
    fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
    f = new FileStream(@"C: \Tekstovna\Padavine2.rez", FileMode.Create,
                       FileAccess.Write);//podatkovni tok za pisanje
    textIn = new StreamReader(fs);//podatke bomo brali v tok
    textOut = new StreamWriter(f);//podatke bom s tokom zapisovali
    while (textIn.Peek() != -1) //iz datoteke beremo podatke dokler jih ne zmanjka
    {
       string stavek = textIn.ReadLine(); //preberemo celo vrstico
       textOut.WriteLine(stavek);         //prebrano vrstico izpišemo v novo datoteko
    }
    textIn.Close();  //zapiranje podatkovnega toka in s tem datoteke
    textOut.Close(); //zapiranje podatkovnega toka in s tem datoteke

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

```
    fs.Close();
}
```

Vaja:

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

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

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

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

Vaja:

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

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

```
            {
                najdaljsa = tabela[i];
            }
        }
    }
     textIn.Close();
     fs.Close();
     Console.WriteLine("najdaljsa beseda: "+najdaljsa);
}
```

Vaja:

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

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

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

static void menjava(string imeDat)
{
   FileStream fs;
   if (!File.Exists(imeDat))//če datoteka še ne obstaja bomo naredili novo
      Console.WriteLine("Datoteka še ne obstaja!");
   else  //če datoteka že obstaja, bomo podatke dodajali
   {
      fs = new FileStream(imeDat, FileMode.Open, FileAccess.Read);
      StreamReader textIn = new StreamReader(fs);
      //preberem celo datoteko, da vem, katere valute so že v njej
      int stevec=1;
      Console.WriteLine();
      //Najprej preberemo vse vrstice, da ugotovimo, koliko podatkov (in katere valute) je že
```

```
      //v datoteki
      while (textIn.Peek() != -1)
      {
         string vrstica = textIn.ReadLine();
         string[] tabela = vrstica.Split('|');
         Console.Write(stevec+") "+tabela[1]+"  "); //oznake valut, ki so že v tabeli v obliki
                                                    //menija izpišemo na zaslon
         stevec++;
      }
      textIn.Close();
      fs = new FileStream(imeDat, FileMode.Open, FileAccess.Read);
      textIn = new StreamReader(fs);
      //uporabnikova izbira valute se ujema z zaporedno številko valute (=vrstice) v datoteki
      Console.Write("\nIzberi ustrezno valuto: ");
      int izb = Convert.ToInt32(Console.ReadLine());
      if (izb > 0 || izb < stevec)
      {
         Console.Write("Znesek za menjavo: ");
         double znesek = Convert.ToDouble(Console.ReadLine());
         int stvrstice=1;
         //poiščemo zaporedno številko vrstice v datoteki, ki se ujema z izbarno valuto
         while (true)
         {
            string vrstica = textIn.ReadLine();
            string[] tabela = vrstica.Split('|');
            if (stvrstice==izb)
            {
               Console.WriteLine();
               Console.WriteLine("Oznaka valute: "+tabela[1]);
               Console.WriteLine("Prodajni tečaj: "+tabela[2]);
               Console.WriteLine("Znesek za menjavo: "+znesek);
               Console.WriteLine("Znesek v valuti: "+znesek*Convert.ToDouble(tabela[2])
                                 +"  "+tabela[1]);
               break;
            }
            else
               stvrstice++;
         }
      }
   }
   Console.ReadLine();
}

//glavni program
static void Main(string[] args)
{
   char izbira;
   do
   {
      Console.Clear();
      Console.WriteLine("V-Vnos oz. dodajanje,I-Izpis,M-Menjava,K-Konec");
      Console.Write("Vnesi izbiro: ");
      izbira=char.ToUpper(Convert.ToChar(Console.ReadLine()));
      switch (izbira)
      {
         case 'V':{
                     vnos(@"c:\Tekstovna\Tecaji.txt");
                     break;
                  }
         case 'I':{
                     izpis(@"c:\Tekstovna\Tecaji.txt");
                     break;
                  }
         case 'M':{
                     menjava(@"c:\Tekstovna\Tecaji.txt");
                     break;
                  }
      }
   }
   while (char.ToUpper(izbira) != 'K');
}
```

Vaja:

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

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

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

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

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

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

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

Vaja:

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

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

   public Izdelek() //konstruktor
   { }

}
```

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

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

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

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

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

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

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

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

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

Vaja:

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

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

string datoteka = "Stevila.txt";

FileStream fs = new FileStream(datoteka, FileMode.OpenOrCreate,FileAccess.ReadWrite);
StreamWriter textOut = new StreamWriter(fs);  //Tekstovni podatkovni tok
Random naklj = new Random();  //generator naključnih števil
//v datoteko zapišemo 10 celih števil
for (int i = 0; i < 10; i++)
```

```
{
    int stevilo = naklj.Next(0, 101);
    textOut.WriteLine(stevilo); //zapis v tekstovno datoteko
    Console.Write(stevilo+", ");
}

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

Naloge:

- Napiši funkcijo, ki dobi za parameter ime poljubne tekstovne datoteke in ki njeno vsebino prikaže na zaslonu!
- Dana je tekstovna datoteka DIJAKI.txt. V vsaki vrstici te datoteke je prvih 30 znakov rezervirano za ime dijaka, naslednjih 15 pa za učni uspeh (od 1 do 5).
  - Koliko dijakov je v datoteki
  - Koliko dijakov ima splošni učni uspeh enak 5
  - Kolikšen je povprečen učni uspeh vseh dijakov
- Kreiraj tekstovno datoteko APJ.TXT. V to datoteko zapiši poljubno število stavkov (bereš jih preko tipkovnice, znak za konec vnosa je prazen stavek!). Vsak stavek naj bo v svoji vrstici, med vrsticami pa naj bo po ena prazna vrstica. Napiši funkcijo, ki dobi za parameter ime te datoteke in ki ugotovi in izpiše, koliko znakov vsebuje celotna datoteka!
- Dana je tekstovna datoteka Naloga.txt. V vsaki vrstici te datoteke so po tri cela števila, med seboj ločena s presledkom. Datoteko obdelaj tako, da za vsako vrstico na ekran izpišeš vsoto vseh treh števil, na koncu pa še skupno vsoto vseh števil!
- V tekstovni datoteki temperature.TXT so shranjeni podatki o temperaturi v določenem kraju. V vsaki vrstici je prvih 20 znakov (desna poravnava) rezerviranih za ime kraja, sledi pa podatek o temperaturi (realno število). Ugotovi povprečno temperaturo, ter izpiši ime kraja z največjo temperaturo!
- V tekstovni datoteki so zapisani podatki o porabi bencina za posamezne tipe vozila. V vsaki vrstici je zapisan tip vozila, nato pa podatek o porabi goriva na 100 km (decimalno število)! Koliko vozil je v datoteki? Ugotovi in izpiši tip vozila z najmanjšo porabo goriva. Podatka otipu vozila in porabi goriva sta razmejena z ločilnim znakom |.
- Napišite program, ki izpiše 10 najdaljših besed v poljubni tekstovni datoteki.

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

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

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

Delo z binarnimi datotekami

Za branje in pisanje podatkov v binarne datoteke uporabljamo razreda **BinaryReader** in **BinaryWriter**.

Pisanje podatkov v binarno datoteko

Za pisanje podatkov v binarno datoteko uporabljamo metodo **Write**, ki pripada razredu **BinaryWriter**. Da lahko pričnemo s pisanjem podatkov v binarno datoteko, moramo najprej ustvariti nov objekt tipa **BinaryWriter**. To storimo npr. takole:

```
string datoteka = @"C: \Tekstovna\Padavine.dat";  //ime in pot do datoteke
//povezava z datoteko na disku
FileStream podatkovni tok = new FileStream(datoteka, FileMode.Create, FileAccess.Write);
//Kreiramo podatkovni tok za pisanje
BinaryWriter binaryOut = new BinaryWriter(podatkovni_tok);
```

Krajši način, takšen kot smo ga spoznali pri tekstovnih datotekah, pri pisanju v binarno datoteko **NE** obstaja!!!

Metode razreda BinaryWriter	Razlaga
Write(podatki)	Zapiše podatke v izhodni tok.
Close()	Zapre objekt tipa **BinaryWriter** in pripadajoči objekt tipa **FileStream**.

Vaja:

```
/*V binarno datoteko Stevila.dat zapiši 10000 naključnih celih števil med -100 in +100 */

string datoteka = @"C: \Binarne\Stevila.dat";
string dir=@"c:\Binarne";
if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
   Directory.CreateDirectory(dir);

//povezava z datoteko na disku
FileStream fs = new FileStream(datoteka, FileMode.Create, FileAccess.Write);
BinaryWriter binaryOut = new BinaryWriter(fs);  //Binarni podatkovni tok
Random naklj=new Random();  //generator naključnih števil

for (int i=0;i<10000;i++)   //zanka za generiranje 10000 naključnih števil
    binaryOut.Write(naklj.Next(-100,101)); //zapis v binarno datoteko

binaryOut.Close(); //zapremo podatkovni tok
fs.Close();
```

V naslednjem primeru bomo uporabili objekt razreda **Izdelek**, ki smo ga deklarirali v prejšnjem primeru. Podatke bomo zapisovali v datoteko z metodo **Write**. Ta pred zapisovanjem najprej preveri tip podatka, ki ga želimo zapisati in nato **TA TIP podatka** zapiše v datoteko takšnega kot je. Če torej metodi **Write** posredujemo podatek, ki je tipa **decimal**, metoda tega podatka ne bo spremenila v **string**, ampak bo v datoteko zapisala decimalni podatek. Ker smo uporabili metodo **`FileMode.Create`** bodo tekoči podatki prepisali prejšnje, če le-ti seveda obstajajo. V datoteko bomo vpisovali vsako polje posebej, saj objekt **BinaryWriter** ne omogoča, da bi v podatkovni tok (in s tem tudi v datoteko) zapisali celotno strukturo naenkrat.

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

   public Izdelek() //konstruktor
   { }
}
```

```
//glavni program
//določimo pot in ime datoteke
string pot = @"C: \Binarne\Datoteke\Izdelki.dat";
//dinamično kreiramo nov podatkovni tok
FileStream fs=new FileStream(pot,FileMode.Create,FileAccess.Write);
//dinamično kreiramo nov objekt tipa BinaryWriter
BinaryWriter binaryOut = new BinaryWriter(fs);

Izdelek Izd = new Izdelek();//nov objekt tipa Izdelek
//določimo vrednosti članom razreda. Seveda bi lahko te podatke vnesel uporabnik, npr. v
//gradnike TextBox nekega obrazca
Izd.naziv = "Kolo";
Izd.proizvajalec = "Scott";
Izd.komadov = 110;
Izd.cena =220000.00m;

//podatke zapišemo v binarno datoteko
binaryOut.Write(Izd.naziv);
binaryOut.Write(Izd.proizvajalec);
binaryOut.Write(Izd.komadov);
binaryOut.Write(Izd.cena);
//zapremo tok podatkov in s tem tudi datoteko
binaryOut.Close();
fs.Close();
```

Naloge:

- V binarno datoteko *PopolnaStevila* zapiši vsa števila med 1 in 1000000, ki imajo to lastnost, da so enaka vsoti svojih deliteljev! Koliko je takih števil?

- Napiši program, ki bo računal vrednost eksponentne funkcije e^x. Program naj se izvaja tako dolgo, dokler se v enem koraku vrednost spremeni za več kot 0.00001. Vrednosti funkcije na vsakem koraku shranjuj v binarno datoteko. Za izračun uporabi matematično vrsto (pravilo):

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

  Pri tem je **3!** enako produktu *3 * 2 * 1*, **n!** pa je enako produktu *n * (n-1) * (n – 2) * ... * 3 * 2 * 1*

- V binarno datoteko *LepaStevila* zapiši vsa števila med 1 in 1000000, ki imajo to lastnost, da so enaka vsoti svojih deliteljev! Koliko je takih števil?

Branje podatkov iz binarne datoteke

Za branje podatkov iz binarne datoteke uporabljamo metode, ki pripadajo razredu **BinaryReader**. Da lahko pričnemo z branjem podatkov iz binarne datoteke, moramo najprej ustvariti nov objekt tipa **BinaryReader**. To storimo npr. takole:

```
string datoteka = @"C: \Tekstovna\Padavine.dat";  //ime in pot do datoteke
//povezava z datoteko na disku
FileStream podatkovni tok = new FileStream(datoteka, FileMode.Open);
//Kreiramo podatkovni tok za branje
BinaryReader binaryIn = new BinaryReader(podatkovni_tok);
```

Razred **BinaryReader** vsebuje kar nekaj metod, ki jih lahko uporabimo pri obdelavi oz. branju binarne datoteke

Lastnostrazreda BinaryReader	**Razlaga**
BaseStream	Omogoča dodatne možnosti dostopa za pripadajoči podatkovni tok, npr. premik po toku (datoteki)
Metode razreda BinaryReader	**Razlaga**
PeekChar( )	Vrne naslednji razpoložljivi znak v vhodni tok, brez premika na naslednjo pozicijo (naslednji znak). Če ni na voljo nobenega znaka več, metoda vrne vrednost -1.
Read( )	Vrne naslednji razpoložljivi znak iz vhodnega toka in napreduje na novo pozicijo v datoteki.
ReadBoolean()	Vrne logično vrednost z vhodnega toka in napreduje naprej od trenutne pozicije v podatkovnem toku za en **byte**.
ReadByte( )	Vrne byte iz vhodnega toka in ustrezno napreduje naprej od trenutne pozicije v podatkovnem toku.
ReadChar( )	Vrne znak iz vhodnega toka in ustrezno napreduje naprej od trenutne pozicije v podatkovnem toku.
ReadDecimal( )	Vrne decimalno vrednost iz vhodnega toka in napreduje naprej od trenutne pozicije v podatkovnem toku za 16 **bytov**.
ReadInt32( )	Vrne 4 **byte** dolgo predznačeno celo število iz vhodnega toka in napreduje naprej od trenutne pozicije v podatkovnem toku za 4 **byte**.
ReadString( )	Vrne **string** iz vhodnega toka in napreduje naprej od trenutne pozicije v podatkovnem toku za toliko, kot je skupno število znakov v tem **stringu**.
Close( )	Zapre objekt **BinaryReader** in pripadajoči objekt **FileStrem**.

Objekta tipa **BinaryWriter** in **BinaryReader** omogočata zapisovanje in branje izključno osnovnih (primitivnih) podatkovnih tipov (to so znaki, cela števila, decimalna števila, logične vrednosti in zaporedja znakov – stringe). V podatkovni tok tako NE MOREMO zapisati ali pa iz njega prebrati npr. celotne strukture. Če hočemo v podatkovni tok (in nato v datoteko) zapisati strukturo, moramo zapisati vsako polje posebej.

Primer:

Datoteko, ki smo jo kreirali v prejšnjem primeru bomo sedaj odprli za branje in jo prebrali. Prebrane podatke bomo izpisali na zaslon!

```
//določimo pot in ime datoteke
string pot = @"C: \Binarne\Izdelki.dat";
//kreiramo nov podatkovni tok za našo datoteko; če datoteke še ne obstaja se zgradi nova
FileStream fs = new FileStream(pot, FileMode.OpenOrCreate, FileAccess.Read);
//dinamično kreiramo nov objekt tipa BinaryReader
```

```
BinaryReader binaryIn = new BinaryReader(fs);

while (binaryIn.PeekChar() != -1) //doler ne zmanjka podatkov (znakov) v datoteki
{
    Izdelek Izd = new Izdelek();//nov objekt tipa Izdelek
    Izd.naziv = binaryIn.ReadString();//podatek o nazivu preberemo kot string
    Izd.proizvajalec = binaryIn.ReadString();//podatek o proizvajalcu preberemo kot string
    Izd.komadov = binaryIn.ReadInt32();//podatek o komadih preberemo kot CELO ŠTEVLO
    Izd.cena = binaryIn.ReadDecimal();//podatek o ceni preberemo kot DECIMALNO ŠTEVLO

    //prebrane podatke prikažemo na zaslonu
    Console.WriteLine("Izdelek: " + Izd.naziv
                            + "\nProizvajalec: " + Izd.proizvajalec
                            + "\nKomadov: " + Izd.komadov
                            + "\nCena: " + Izd.cena);
}
//zapremo tok podatkov in s tem tudi datoteko
binaryIn.Close();
fs.Close();
```

Lastnost **BaseStream** omogoča dodatne možnosti dostopa za pripadajoči podatkovni tok. Ne glede na to, na katerem mestu v datoteki se trenutno nahajamo, se lahko s pomočjo te lastnosti premaknemo naprej in nazaj po podatkovnem toku (datoteki). Če smo podatkovni tok odprli z lastnostjo **`FileMode.OpenOrCreate`**, lahko v datoteko nekaj zapišemo, se pomaknemo kamorkoli znotraj datoteke in iz nje beremo.

Primer:

```
FileStream fs = new FileStream(@"C:\Binarne\temp.dat", FileMode.OpenOrCreate);
BinaryReader binaryIn = new BinaryReader(fs);

//po podatkovnem toku (datoteki) se premaknimo za dve celi števili naprej
binaryIn.BaseStream.Position = sizeof(int)*2;

//po podatkovnem toku (datoteki) se premaknimo za eno celo število nazaj
binaryIn.BaseStream.Position = -sizeof(int);

//skok na začetek podatkovnega toka
binaryIn.BaseStream.Position = 0;
```

Vaja:

```
/*V Binarno datoteko zapišimo 10 naključnih celih števil. S pomočjo lastnosti
Position se nato postavimo na začetek te datoteke (ne da bi zaprl podatkovni tok), preberimo
števila iz te datoteke in jih izpišimo na zaslon!!!!*/

string datoteka = @"D:\Stevila.dat";
FileStream fs = new FileStream(datoteka, FileMode.OpenOrCreate);
BinaryWriter binaryOut = new BinaryWriter(fs);  //Binarni podatkovni tok
Random naklj = new Random();  //generator naključnih števil
//v datoteko zapišemo 10 celih števil
for (int i = 0; i < 10; i++)
  binaryOut.Write(naklj.Next(-100, 101)); //zapis v binarno datoteko
//Datoteko (podatkovni tok) pustimo ODPRT in se postavimo na začetek datoteke
binaryOut.BaseStream.Position = 0;
//preberemo vsebino datoteke in jo izpišemo na zaslon
BinaryReader binaryIn = new BinaryReader(fs);
Console.WriteLine("Vsebina datoteke: \n");
try
{
   for (int i = 0; i < 10; i++)//preberemo 10 števil iz datoteke
   {
      int st = binaryIn.ReadInt32();
      Console.Write(st + " ");
   }
}
catch { }
binaryOut.Close(); //zapremo podatkovni tok
binaryIn.Close();
fs.Close();
```

Vaja:

```
/*Primer zapisovanje različnih osnovnih podatkovnih tipov v binarno datoteko, ter branje iz
take datoteke*/

Console.WriteLine("*Kreiranje binarne datoteke in zapis ter branje binarnih podatkov *");
FileStream fs = new FileStream(@"c:\Binarne\temp.dat", FileMode.OpenOrCreate,
                               FileAccess.ReadWrite);

BinaryWriter binOut = new BinaryWriter(fs);

int mojInt = 9;
float mojFloat = 9.8F;
bool mojBool = false;
char[] mojaTabelaZnakov= { 'P', 'o', 'z', 'd', 'r', 'a', 'v' };

binOut.Write(mojInt);
binOut.Write(mojFloat);
binOut.Write(mojBool);
binOut.Write(mojaTabelaZnakov);

binOut.BaseStream.Position = 0;  //premik na začetek podatkovnega toka

Console.WriteLine("\nBranje binarnih podatkov - tako, kot so bili zapisani v datoteko!\n");
BinaryReader binIn = new BinaryReader(fs);

Console.WriteLine(binIn.ReadInt32());   //izpis: 9
Console.WriteLine(binIn.ReadSingle());  //izpis: 9,8
Console.WriteLine(binIn.ReadBoolean()); //izpis: false
for (int i=0;i<7;i++)
   Console.Write(binIn.ReadChar());    //izpis: Pozdrav

Console.WriteLine("\n\nBranje binarnih podatkov - vsak byte posebej!\n");

binOut.BaseStream.Position = 0;  //premik na začetek podatkovnega toka
try
{
    while (true)
    {
       Console.Write(binIn.ReadByte()+" ");
    }
}
catch { }
//zapiranje podatkovnih tokov in povezave z datoteko na disku
binOut.Close();
binIn.Close();
fs.Close();
```

Vaja:

```
/*V binarno datoteko Stev.dat zapiši 100 naključnih celih števil med 65 in 90
  Datoteko nato preberi vse podatke kot cela števila
  Datoteko preberi še kot datoteko znakov*/

//določimo pot in ime datoteke
string datoteka = @"C: \Binarne\Stev.dat";
string dir = @"c:\Binarne";

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

FileStream fs = new FileStream(datoteka, FileMode.Create, FileAccess.Write);
BinaryWriter binaryOut = new BinaryWriter(fs);  //Binarni podatkovni tok
Random naklj = new Random();            //generator naključnih števil
for (int i = 0; i < 100; i++)           //zanka za generiranje 100 naključnih števil
   binaryOut.Write(naklj.Next(65, 91)); //zapis v binarno datoteko

binaryOut.Close(); //zapremo podatkovni tok

//Iz datoteke berimo podatke kot cela števila
fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);  //povezava z datoteko na disku
```

```
BinaryReader binaryIn = new BinaryReader(fs);  //Binarni podatkovni tok
Console.WriteLine("Vsebina datoteke Stev.dat, iz katere binarno beremo cela števila!");

try //uporabimo varovalni blok - ko bo podatkov konec se bo program nadaljeval za catch
{
   while (true)
   {
      int i = binaryIn.ReadInt32();
      Console.Write(i + "  ");
   }
}
catch
{  }
binaryIn.Close(); //zapremo podatkovni tok

fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);  //povezava z datoteko na disku
binaryIn = new BinaryReader(fs);  //Binarni podatkovni tok
Console.WriteLine("Vsebina datoteke Stev.dat, iz katere binarno beremo znake!");
try  //uporabimo varovalni blok - ko bo podatkov konec se bo program nadaljeval za catch
{
   while (true)
   {
      char znak = binaryIn.ReadChar();//Iz datoteke berimo podatke kot znake
      Console.Write(znak);
   }
}
catch{  }
binaryIn.Close(); //zapremo podatkovni tok
fs.Close();
```

Vaja:

```
/*V binarno datoteko Tocke.dat zapišimo 10 točk z naključnimi koordinatami. Vsaka točka je
struktura z dvema komponentama - x in y koordinato. Datoteko nato odpri in koordinate vseh
točk izpiši na zaslon*/

struct tocka
{
   public int x, y;
   public tocka(int x,int y)
   {
      this.x=x;
      this.y=y;
   }
}

static void Main(string[] args)
{
   //določimo pot in ime datoteke
   string datoteka = @"C:\Binarne\Tocke.dat";//ime datoteke
   string dir = @"c:\Binarne";
   if (!Directory.Exists(dir))  //če imenik še ne obstaja, ga skreiramo
      Directory.CreateDirectory(dir);

   FileStream fs = new FileStream(datoteka, FileMode.Create, FileAccess.Write);
   BinaryWriter binaryOut = new BinaryWriter(fs);  //Binarni podatkovni tok
   Random naklj = new Random();  //generator naključnih števil
   tocka nova;
   for (int i = 0; i < 10; i++)   //zanka za generiranje 10000 naključnih števil
   {
      nova=new tocka(naklj.Next(101),naklj.Next(101));
      binaryOut.Write(nova.x);
      binaryOut.Write(nova.y);
   }
   binaryOut.Close(); //zapremo podatkovni tok

   //Obdelava binarne datoteke: izpis točk, ki so v datoteki
   fs = new FileStream(datoteka, FileMode.Open, FileAccess.Read);
   BinaryReader binaryIn = new BinaryReader(fs);  //Binarni podatkovni tok
   try
   {
       Console.WriteLine("Seznam točk iz datoteke: ");
       while (true)
```

```
        {
            tocka T = new tocka();
            //POZOR! Ker so v datoteki podatki tipa int moramo uporabiti metodo ReadInt32()
            T.x = binaryIn.ReadInt32();//preberemo en podatek - število iz datoteke
            T.y = binaryIn.ReadInt32();//preberemo en podatek - število iz datoteke
            Console.WriteLine("("+T.x + " , " +T.y+")");
        }
    }
    catch { }
    binaryIn.Close();
    fs.Close();
}
```

Naloge:

Kreiraj binarno datoteko 1000 naključnih celih števil med 1 in 10000. Števila iz te datoteke nato prepiši v tekstovno datoteko tako, da bo vsako število v svoji vrsti, pred številom pa bo še z besedo zapisano število mest tega števila. (če je torej število enako 546 bo pred njim besedica tri…).

Urejanje podatkov

Urejanje podatkov je eno najpogostejših opravil v računalniški praksi (urejanje po abecedi, urejanje po velikosti, ipd.). Pojem **urejanje** oz. **sortiranje** podatkov pomeni preurejanje množice podatkov a_1, a_2, a_3, … a_n, v nek nov novi vrstni red a_{1k}, a_{2k}, a_{3k}, …, a_{nk}, tako, da velja $a_{1k} < a_{2k} < a_{3k} < \ldots < a_{nk}$. Kot metodo za urejanje lahko uporabimo eno od številnih že obstoječih metod za urejanje. Namen urejanja podatkov je olajšati kasnejše postopke, npr. iskanje določenega podatka ali skupine podatkov.

Poglavje o urejanju oz. sortiranju ima več namenov: podati pregled najpomembnejših sortirnih metod, narediti primerjave med njimi, opisati njihove zahtevnosti, napisati primere za posamezne metode, ter ugotoviti, kdaj je katera primernejša od druge. Videli bomo tudi, da za urejanje podatkov obstajajo različni algoritmi, od katerih ima vsak svoje dobre in slabe strani. Načini urejanja se lahko ocenjujejo po različnih kriterijih in je pogosto težko povsem nedvoumno reči, kateri algoritem je nasplošno boljši ali slabši. V računalništvu gre največkrat za kompromise med različnimi kriteriji, kot so na primer prostorska - časovna zahtevnost ali primernost metode za majhno ali veliko število podatkov.

Sortirne metode najprej delimo glede na to, ali urejamo podatke, ki so v neki tabeli, ali pa urejamo podatke, ki so v neki datoteki:

- Za podatke, ki so v tabeli je značilno, da so istočasno v hitrem pomnilniku. Da bi lahko urejali čim več podatkov, si postavimo omejitev, da smemo operirati le z eno kopijo podatkov. Primer: sortiranje odprtih kart na mizi;
- Za podatke v datotekah pa je značilno, da so na nekem masovnem mediju za hrambo podatkov (diskih, trakovih,..), ki so navidezno neomejeni. Primer: sortiranje velikega števila kart po kupčkih.

V tem poglavju se bomo omejili na sortiranje polj (tabel). Omejitve, ki se jih pri tem moramo zavedati, oz. jih upoštevati so:

- Prostorske omejitve (ena kopija podatkov, oz. ena tabela);
- Časovne omejitve (preverjanje časovne zahtevnosti – čim manjše število primerjav in premikov podatkov, ki so v tabeli).

V osnovi poznamo dve vrsti sortirnih metod

- **enostavne:** preprosti algoritmi, na njih sloni velika večina sestavljenih algoritmov, zahtevnost pa narašča praviloma s kvadratom števila podatkov;
- **sestavljene:** zahtevnejši algoritmi, pogosto rekurzivni, njihove prednosti se izkažejo pri velikem številu podatkov.

Navadne metode za urejanje podatkov

Sortiranje z navadnim vstavljanjem

Metoda **Navadnega vstavljanja** je uspešna metoda za *malo podatkov* in v posebnih primerih *delno urejenih podatkov*, pa tudi osnova zelo uspešnih sestavljenih metod **Shellsort** in **Quicksort**.

Zaporedje podatkov razdelimo na **urejeni** in **neurejeni** del. Na začetku vzamemo kot urejenega prvi podatek, ostali so neurejeni. Podatke iz neurejenega dela po vrsti vstavljamo v urejeni tako, da zaporedoma primerjamo podatek, ki ga vstavljamo, s podatkom v urejenem zaporedju in ju menjamo, če je novi podatek manjši.

Primer preurejanja elementov v tabeli: prikazano je začetno stanje tabele 8 celih števil in posamezni koraki urejanja, ki nas privedejo do urejene tabele:

12	**5**	**9**	**14**	**6**	**3**	**21**	**10**
5	**12**	**9**	**14**	**6**	**3**	**21**	**10**
5	**9**	**12**	**14**	**6**	**3**	**21**	**10**
5	**9**	**12**	**14**	**6**	**3**	**21**	**10**
5	**6**	**9**	**12**	**14**	**3**	**21**	**10**
3	**5**	**6**	**9**	**12**	**14**	**21**	**10**
3	**5**	**6**	**9**	**12**	**14**	**21**	**10**
3	**5**	**6**	**9**	**10**	**12**	**14**	**21**

Rešitev v C#:

```
static void vstavljanje(int[] tabela)
{
   int i, j, trenutni;
   for (i = 1; i < elementov; i++)
   {
       trenutni = tabela[i];
       j = i - 1 ;
       while ((j>=0)&&(trenutni < tabela[j]))
       {
           tabela[j + 1] = tabela[j];
           j = j - 1;
       }
       tabela[j + 1] = trenutni;
   }
}
```

Število primerjav je konstantno za vse primere. V vsakem koraku i (i=1…n-1) je potrebno podatek iz neurejenega dela tabele vstaviti v urejeni del. Ta metoda za urejanje podatkov je primerna za urejanje do nekaj tisoč podatkov.

Izboljšana vrsta navadnega vstavljanje se imenuje **dvojiško iskanje.**

Urejanje z izbiranjem

V tem primeru se vse zaporedje vzame za neurejeni del, iz katerega se izbira najmanjši podatek. Tega zamenjamo s prvim podatkom v neurejenem delu, ki se s tem uvrsti v urejenega. V primerjavi z navadnim vstavljanjem imamo pri navadnem izbiranju v vsakem koraku le dve zamenjavi, vendar moramo izbrati najmanjši podatek izmed vseh v neurejenem zaporedju. Primer takega urejanja podatkov v vsakdanjem življenju je npr. urejanje kart pri taroku.

Primer preurejanja elementov v tabeli: prikazano je začetno stanje tabele 8 celih števil in posamezni koraki urejanja, ki nas privedejo do urejene tabele:

12	**5**	**19**	**2**	**9**	**17**	**3**	**10**
2	**5**	**19**	**12**	**9**	**17**	**3**	**10**
2	**3**	**19**	**12**	**9**	**17**	**5**	**10**
2	**3**	**5**	**12**	**9**	**17**	**19**	**10**
2	**3**	**5**	**9**	**12**	**17**	**19**	**10**
2	**3**	**5**	**9**	**10**	**17**	**19**	**12**
2	**3**	**5**	**9**	**10**	**12**	**19**	**17**
2	**3**	**5**	**9**	**10**	**12**	**17**	**19**

Rešitev v C#:

```
static void izbiranje(int[] tabela)
{
    int i,j,x,k;
    for (i = 0; i <= N - 2; i++) //N je število elementov tabele
    {
       k = i; x = tabela[i];
       for (j = i + 1; j < N; j++)
       {
          if (tabela[j] < x)
          {
             k = j;
             x = tabela[j];
          }
       }
       tabela[k] = tabela[i]; tabela[i] = x;
    }
}
```

Število primerjav je konstantno za vse primere. V vsakem koraku i (i=1…n-1) je potrebno izbrati najmanjši podatek izmed preostalih. Ta metoda za urejanje podatkov je tako kot metoda vstavljanja primerna za urejanje do nekaj tisoč podatkov.

Urejanje s premenami - Bubblesort

Osnovna značilnost so premene. Podatki se obnašajo kot različno težki mehurčki, ki jih vzgon dviga proti vrhu cevke s tekočino – zato **bubblesort**. V spodnjem primeru so vhodni podatki vpisani v prvem stolpcu.

Primer preurejanja elementov v tabeli: prikazano je začetno stanje tabele 8 celih števil in posamezni koraki urejanja, ki nas privedejo do urejene tabele:

12	**5**	**19**	**2**	**9**	**17**	**3**	**10**
2	**12**	**5**	**19**	**3**	**9**	**17**	**10**
2	**3**	**12**	**5**	**19**	**9**	**10**	**17**
2	**3**	**5**	**12**	**9**	**19**	**10**	**17**

2	3	5	9	12	10	19	17
2	3	5	9	10	12	17	19
2	3	5	9	10	12	17	19
2	3	5	9	10	12	17	19

Rešitev v C#

```
static void mehurcki(int[] tabela)
{
   int x;
   for (int i = 1; i < elementov; i++)
   {
      for (int j = elementov - 1; j >= i; j--)
      {
         if (tabela[j - 1] > tabela[j])
         {
             x = tabela[j - 1];
             tabela[j - 1] = tabela[j];
             tabela[j] = x;
         }
      }
   }
}
```

Izboljšani Bubblesort – Shakersort (metoda stresanja)

Glavna slabost metodo **bubbelsort** je, da so premene na kratko razdaljo (za eno mesto) neugodne; majhni podatki se hitro dvigujejo, veliki počasi padajo. Posodobljena različica te metode se imenuje **metoda stresanja** – **shakesort**. Na vsakem koraku primerjamo dva sosednja elementa v tabeli po njuni vrednosti, ter ju, če še nista v pravilnem vrstnem redu, zamenjamo. Postopek ponavljanja teče izmenično, enkrat z leve strani, drugič z desne strani tabele. Prednost tega algoritma je v tem, da se zmanjša število primerjanj, število zamenjav pa ostane enako kot pri urejanju z mehurčki. Metodo **shakesort** uporabljamo tedaj, kadar sortiramo manjše tabele, saj je časovna zahtevnost takega urejanja precejšnja.

Sestavljene sortirne metode

Sortiranje z vstavljanjem s padajočim prirastkom – Shellsort

Ideja algoritma je enostavna: množico elementov posebej sortiramo po skupinah, pri čemer v isto skupino spadajo elementi, ki so določeno število mest narazen.

Algoritem ne ureja podatkov samostojno, ampak si pomaga z drugimi algoritmi, najpogosteje z navadnim vstavljanjem. Izboljšan je tako, da pri urejanju ne primerja vseh elementov po vrsti, ampak v začetku ureja na "daleč", torej z večjimi koraki. Algoritem je zanimiv zgolj zaradi tega, ker poveča učinkovitost drugih algoritmov. Z grupiranjem podatkov se na nek način zmanjša čas, ki bi bil drugače potreben pri procesu urejanja podatkov. Največ menjav opravimo na začetku, kasneje so skupine večinoma urejene in je potrebno le manjše število korakov. V vsaki skupini je majhno število elementov ali pa so že kar dobro urejeni, zato ni potrebno veliko število korakov. Na koncu uporabimo navadno vstavljanje, ki v najslabšem primeru postori celotno delo in nas pripelje do urejene tabele.

Rešitev v C#:

```
static void shellSort(int[] tabela)
{
   int k = 1; // izračun koraka za prvo fazo
   // določimo največji korak, ki ga lahko izvedemo znotraj tabele
   while (3 * k + 1 < elementov)
       k = 3 * k + 1;
   while (k > 0) // zanka za faze
   {
      for (int i = k; i < elementov; i++)
      {
         int x = tabela[i]; // izberemo zadnji element skupine
         int j = i - k;
         while (j >= 0 && x < (tabela[j]))//če je element x v skupini manjši od el. pred njim
         {
            tabela[j + k] = tabela[j]; // ju zamenjamo
            j = j - k;
         }
         tabela[j + k] = x; // nov trenutni element
      }
      k = k / 3;            // vsi elementi v skupini so urejeni, zmanjšamo korak
   }
}
```

Hitro urejanje – Quicksort

Metoda **quicksort** je najboljša metoda za sortiranje polj. Metoda je rekurzivna in spominja na **bubblesort** - premene na čim večje razdalje.

Ideja urejanja je takale: izberemo sredinski element - **pivot** (najugodneje bi bilo, če bi bil po vrednosti na sredini vseh elementov). Nato z ene strani iščemo podatek, ki je večji, z druge pa takega, ki je manjši od njega. Podatka zamenjamo. Tako dobimo dve zaporedji s podatki, večjimi (ali enakimi) oz. manjšimi od pivota. Za obe zaporedji postopek ponavljamo, dokler niso zaporedja prazna ali vsebujejo en sam podatek.

Rešitev v C#:

```
public static void quickSort(int[] tabela)
{
    quicksort(tabela, 0, elementov - 1);
}

public static void quicksort(int[] tabela, int i, int j)
{
   if (i < j)
   {
      int p = premeci(tabela, i, j);
      quicksort(tabela, i, p - 1);
      quicksort(tabela, p + 1, j);
   }
}

static int premeci(int[] tabela, int i, int j)
{
   int pivot = tabela[i];
   int m = i;
   int n = j;
   while (m < n)
   {
      while (m < n && tabela[m] <= pivot)
         m = m + 1;

      while (m < n && tabela[n] > pivot)
         n = n - 1;
      if (m < n)
```

```
      {
         int t = tabela[m];
         tabela[m] = tabela[n];
         tabela[n] = t;
      }
   }
   if (tabela[m] > pivot)
     m = m - 1;
   tabela[i] = tabela[m];
   tabela[m] = pivot;
   return m;
}
```

Našteli smo le nekaj metod za urejanje. Poleg naštetih metod obstaja seveda še vrsta drugih. Nekatere med njimi so enostavne in inventivne – urejanje je zaradi tega počasno, druge pa ekstremno kompliciranе - urejanje pa je zelo hitro.

Naslednji program prikazuje primerjavo med posameznimi metodami, saj za vsako od metod za urejanje merimo potreben čas za urejanje elementov tabele 10000 celih števil. Preizkus pokaže, da je pri takšnem številu podatkov najpočasnejše urejanje z metodo mehurčkov, sledita metoda izbiranja in vstavljanja, nato pa metoda s padajočim prirastkom - **shellsort**. Pri tako velikem številu podatkov je seveda najhitrejša metoda hitrega urejanja – **quicksort**.

```
static int elementov = 10000;//število elementov tabele
static int [] tabela;
static Random naklj;
static void Main(string[] args)
{   //metode za urejanje podatkov
    char izbira;
    tabela=new int[elementov];
    naklj=new Random();
    for (int i=0;i<elementov;i++)
       tabela[i]=naklj.Next(1000);
    do
    {
        Console.WriteLine("\nA) Urejanje z vstavljanjem");
        Console.WriteLine("B) Urejanje z izbiranjem");
        Console.WriteLine("C) Urejanje z metodo mehurčkov (BubbleSort)");
        Console.WriteLine("D) Shellsort");
        Console.WriteLine("E) Quicksort");
        Console.WriteLine("I) Izpis trenutne tabele");
        Console.WriteLine("N) Nova tabela");
        Console.WriteLine("K) Konec");
        Console.WriteLine("-----------------------------------");
        Console.Write("Vnesi izbiro: ");
        izbira = Convert.ToChar(Console.ReadLine().ToUpper());
        switch (izbira)
        {
            case 'A':
            {
                vstavljanje(tabela);
                break;
            }
            case 'B':
            {
                izbiranje(tabela);
                break;
            }
            case 'C':
            {
                mehurcki(tabela);
                break;
            }
            case 'D':
            {
                shellSort(tabela);
                break;
            }
            case 'E':
            {
                quickSort(tabela);
                break;
```

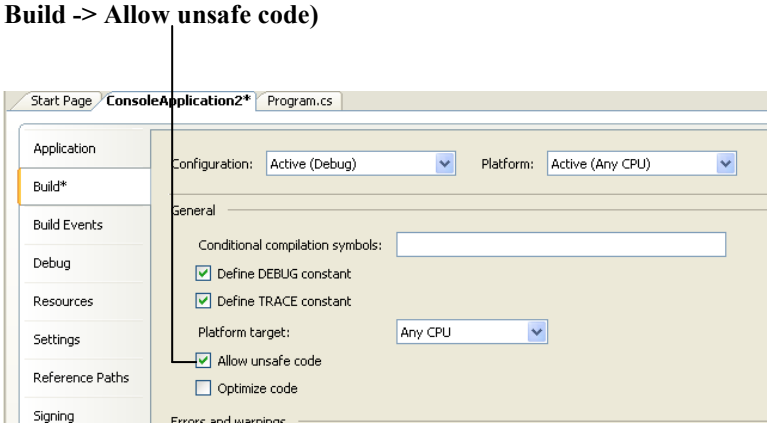
```
            }
            case 'I':
            {
                izpis(tabela);
                break;
            }
            case 'N':
            {
                novaTabela(tabela);
                break;
            }

        }
    } while (izbira != 'K');
}

static void izpis(int[] tabela)
{
   Console.WriteLine("Tabela "+elementov+" celih števil:");
   for (int i=0;i<elementov;i++)
   {
        Console.Write("{0,4}",tabela[i]);
        if ((i+1)%10==0)
           Console.WriteLine();
   }
}

static void vstavljanje(int[] tabela)
{
   int i, j, trenutni;
   DateTime zacetek = DateTime.Now;
   for (i = 1; i < elementov; i++)
   { //element z indeksom 0 uporabimo za pomožno spremenljivko
       trenutni = tabela[i];
       j = i - 1 ;
       while ((j>=0)&&(trenutni < tabela[j]))
       {
          tabela[j + 1] = tabela[j];
          j = j - 1;
       }
       tabela[j + 1] = trenutni;
   }
   DateTime konec = DateTime.Now;
   Console.WriteLine("Potreben čas za urejanje: " + Convert.ToString(konec - zacetek));
}

static void izbiranje(int[] tabela)
{
   int i,j,x,k;
   DateTime zacetek = DateTime.Now;
   for (i = 0; i <= elementov - 2; i++)
   {
      k = i; x = tabela[i];
      for (j = i + 1; j < elementov; j++)
      {
         if (tabela[j] < x)
         {
            k = j;
            x = tabela[j];
         }
      }
      tabela[k] = tabela[i]; tabela[i] = x;
   }
   DateTime konec = DateTime.Now;
   Console.WriteLine("Potreben čas za urejanje: " + Convert.ToString(konec - zacetek));
}

static void mehurcki(int[] tabela)
{
   int x;
   DateTime zacetek = DateTime.Now;
   for (int i = 1; i < elementov; i++)
   {
      for (int j = elementov - 1; j >= i; j--)
      {
```

```
            if (tabela[j - 1] > tabela[j])
            {
               x = tabela[j - 1];
               tabela[j - 1] = tabela[j];
               tabela[j] = x;
            }
         }
      }
      DateTime konec = DateTime.Now;
      Console.WriteLine("Potreben čas za urejanje: " + Convert.ToString(konec - zacetek));
}

static void shellSort(int[] tabela)
{
      DateTime zacetek = DateTime.Now;
      int k = 1; // izračun koraka za prvo fazo
      while (3 * k + 1 < elementov) // določimo največji korak, ki ga lahko izvedemo znotraj
                                    //tabele  (1, 4, 13, 40, 121, ...)
          k = 3 * k + 1;
      while (k > 0) // zanka za faze
      {
          for (int i = k; i < elementov; i++)
          {
             int x = tabela[i]; // izberemo zadnji element skupine
             int j = i - k;
             while (j >= 0 && x < (tabela[j])) //če je element x v skupini manjši od tistega pred
                                               //njim
             {
                tabela[j + k] = tabela[j]; // ju zamenjamo
                j = j - k;
             }
             tabela[j + k] = x; // nov trenutni element
          }
          k = k / 3;       // vsi elementi v skupini so urejeni, zmanjšamo korak  }}
      }
      DateTime konec = DateTime.Now;
      Console.WriteLine("Potreben čas za urejanje: " + Convert.ToString(konec - zacetek));
}

public static void quickSort(int[] tabela)
{
      DateTime zacetek = DateTime.Now;
      quicksort(tabela, 0, elementov - 1);
      DateTime konec = DateTime.Now;
      Console.WriteLine("Potreben čas za urejanje: " + Convert.ToString(konec - zacetek));
}

public static void quicksort(int[] tabela, int i, int j)
{
      if (i < j)
      {
         int p = premeci(tabela, i, j);
         quicksort(tabela, i, p - 1);
         quicksort(tabela, p + 1, j);
      }

}

static int premeci(int[] tabela, int i, int j)
{
      int pivot = tabela[i];
      int m = i;
      int n = j;
      while (m < n)
      {
         while (m < n && tabela[m] <= pivot)
            m = m + 1;
         while (m < n && tabela[n] > pivot)
            n = n - 1;
         if (m < n)
         {
            int t = tabela[m];
            tabela[m] = tabela[n];
            tabela[n] = t;
         }
```

```
    }
    if (tabela[m] > pivot)
    { m = m - 1; }
    tabela[i] = tabela[m];
    tabela[m] = pivot;
    return m;
}

static void novaTabela(int[] tabela)
{
    for (int i=0;i<elementov;i++)
      tabela[i]=naklj.Next(1000);
}
```

Kazalci v C#: perator ->

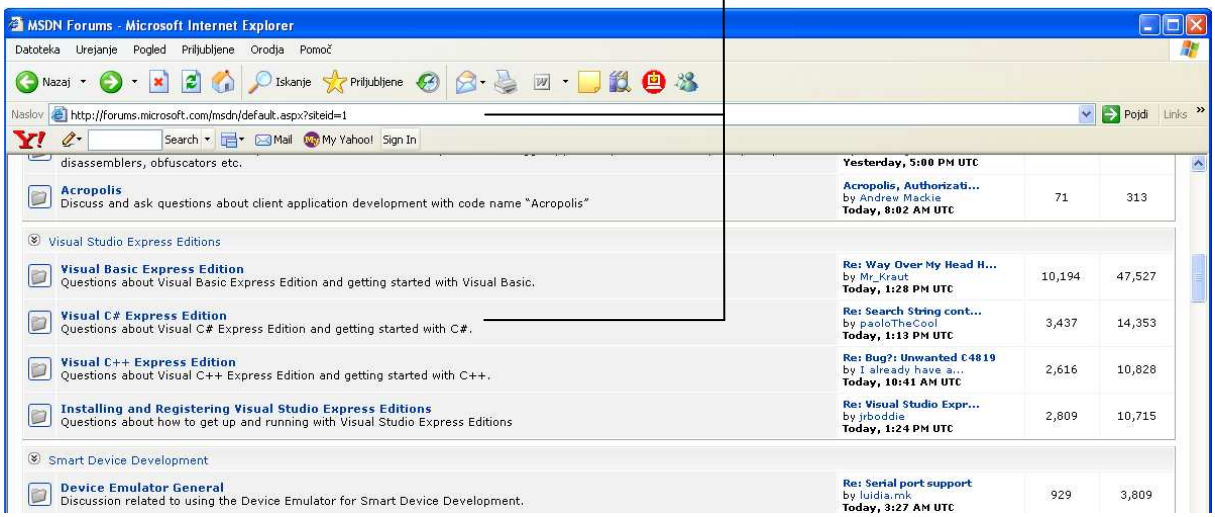
Tudi C# pozna kazalce, a jih uporabljamo redkeje kot v C++. Kodo, ki uporablja kazalce moramo obvezno zapreti v blok ***unsafe***, pri prevajanju pa odkljukati opcijo **Allow unsafe code (Meni Project -> Properties-> Build -> Allow unsafe code)**




```
struct Tocka
{
   public int x;
   public int y;
}

static void Main(string[] args)
{

    unsafe  // začetek bloka unsafe
    {
       Tocka pt = new Tocka();  // nov objekt izpeljan iz strukture Tocka
       Tocka* pp = &pt;   // kazalec pp (ki je tipa Tocka) kaže na objekt pt
       pp->x = 123;  // polje x objekta pt (na objekt kaže kazalec pp) dobi vrednost 123
       pp->y = 456;  // polje y objekta pt (na objekt kaže kazalec pp) dobi vrednost 456
       Console.WriteLine("{0} {1}", pt.x, pt.y); //izpis obeh polj objekta pt
    }  // konec bloka unsafe
}
```

Pomoč

Spoznali smo nekaj osnov pri delu s **C#.** Pri nadaljnjem delu bomo seveda rabili še veliko pomoči, ki nam jo okolje **C#** ponuja v meniju **Help**, ali pa pomoč poiščemo preprosto tako, da z miško kliknemo na besedo, za katero rabimo pomoč in nato na tipkovnici pritisnemo tipko **F1**. Če pa nam pomoč v meniju **Help** ali pa **F1** še ne zadoščata, imamo na voljo številne **msdn** forume!



Dodatek 1: Kode tipk (KeyCode) na tipkovnici

Ikona označuje tipke, ki jih podpira .NET Compact Framework. Zaradi enostavnosti razlaga posamezne tipke ni prevedena!

	Oznaka tipke	Razlaga
✓	**A**	**The A key.**
✓	**Add**	**The add key.**
✓	**Alt**	**The ALT modifier key.**
✓	**Apps**	**The application key (Microsoft Natural Keyboard).**
✓	**Attn**	**The ATTN key.**
✓	**B**	**The B key.**
✓	**Back**	**The BACKSPACE key.**
	BrowserBack	**The browser back key (Windows 2000 or later).**
	BrowserFavorites	**The browser favorites key (Windows 2000 or later).**
	BrowserForward	**The browser forward key (Windows 2000 or later).**
	BrowserHome	**The browser home key (Windows 2000 or later).**
	BrowserRefresh	**The browser refresh key (Windows 2000 or later).**
	BrowserSearch	**The browser search key (Windows 2000 or later).**
	BrowserStop	**The browser stop key (Windows 2000 or later).**
✓	**C**	**The C key.**
✓	**Cancel**	**The CANCEL key.**
✓	**Capital**	**The CAPS LOCK key.**
✓	**CapsLock**	**The CAPS LOCK key.**
✓	**Clear**	**The CLEAR key.**
✓	**Control**	**The CTRL modifier key.**

ControlKey	**The CTRL key.**
Crsel	**The CRSEL key.**
D	**The D key.**
D0	**The 0 key.**
D1	**The 1 key.**
D2	**The 2 key.**
D3	**The 3 key.**
D4	**The 4 key.**
D5	**The 5 key.**
D6	**The 6 key.**
D7	**The 7 key.**
D8	**The 8 key.**
D9	**The 9 key.**
Decimal	**The decimal key.**
Delete	**The DEL key.**
Divide	**The divide key.**
Down	**The DOWN ARROW key.**
E	**The E key.**
End	**The END key.**
Enter	**The ENTER key.**
EraseEof	**The ERASE EOF key.**
Escape	**The ESC key.**
Execute	**The EXECUTE key.**
Exsel	**The EXSEL key.**
F	**The F key.**

F1	**The F1 key.**
F10	**The F10 key.**
F11	**The F11 key.**
F12	**The F12 key.**
F13	**The F13 key.**
F14	**The F14 key.**
F15	**The F15 key.**
F16	**The F16 key.**
F17	**The F17 key.**
F18	**The F18 key.**
F19	**The F19 key.**
F2	**The F2 key.**
F20	**The F20 key.**
F21	**The F21 key.**
F22	**The F22 key.**
F23	**The F23 key.**
F24	**The F24 key.**
F3	**The F3 key.**
F4	**The F4 key.**
F5	**The F5 key.**
F6	**The F6 key.**
F7	**The F7 key.**
F8	**The F8 key.**
F9	**The F9 key.**
FinalMode	**The IME final mode key.**

G	The G key.
H	The H key.
HanguelMode	The IME Hanguel mode key. (maintained for compatibility; use HangulMode)
HangulMode	The IME Hangul mode key.
HanjaMode	The IME Hanja mode key.
Help	The HELP key.
Home	The HOME key.
I	The I key.
IMEAccept	The IME accept key, replaces IMEAceept.
IMEAceept	The IME accept key. Obsolete, use IMEAccept instead.
IMEConvert	The IME convert key.
IMEModeChange	The IME mode change key.
IMENonconvert	The IME nonconvert key.
Insert	The INS key.
J	The J key.
JunjaMode	The IME Junja mode key.
K	The K key.
KanaMode	The IME Kana mode key.
KanjiMode	The IME Kanji mode key.
KeyCode	The bitmask to extract a key code from a key value.
L	The L key.
LaunchApplication1	The start application one key (Windows 2000 or later).
LaunchApplication2	The start application two key (Windows 2000 or later).
LaunchMail	The launch mail key (Windows 2000 or later).
LButton	The left mouse button.

LControlKey	The left CTRL key.
Left	The LEFT ARROW key.
LineFeed	The LINEFEED key.
LMenu	The left ALT key.
LShiftKey	The left SHIFT key.
LWin	The left Windows logo key (Microsoft Natural Keyboard).
M	The M key.
MButton	The middle mouse button (three-button mouse).
MediaNextTrack	The media next track key (Windows 2000 or later).
MediaPlayPause	The media play pause key (Windows 2000 or later).
MediaPreviousTrack	The media previous track key (Windows 2000 or later).
MediaStop	The media Stop key (Windows 2000 or later).
Menu	The ALT key.
Modifiers	The bitmask to extract modifiers from a key value.
Multiply	The multiply key.
N	The N key.
Next	The PAGE DOWN key.
NoName	A constant reserved for future use.
None	No key pressed.
NumLock	The NUM LOCK key.
NumPad0	The 0 key on the numeric keypad.
NumPad1	The 1 key on the numeric keypad.
NumPad2	The 2 key on the numeric keypad.
NumPad3	The 3 key on the numeric keypad.
NumPad4	The 4 key on the numeric keypad.

	NumPad5	**The 5 key on the numeric keypad.**
	NumPad6	**The 6 key on the numeric keypad.**
	NumPad7	**The 7 key on the numeric keypad.**
	NumPad8	**The 8 key on the numeric keypad.**
	NumPad9	**The 9 key on the numeric keypad.**
	O	**The O key.**
	Oem1	**The OEM 1 key.**
	Oem102	**The OEM 102 key.**
	Oem2	**The OEM 2 key.**
	Oem3	**The OEM 3 key.**
	Oem4	**The OEM 4 key.**
	Oem5	**The OEM 5 key.**
	Oem6	**The OEM 6 key.**
	Oem7	**The OEM 7 key.**
	Oem8	**The OEM 8 key.**
	OemBackslash	**The OEM angle bracket or backslash key on the RT 102 key keyboard (Windows 2000 or later).**
	OemClear	**The CLEAR key.**
	OemCloseBrackets	**The OEM close bracket key on a US standard keyboard (Windows 2000 or later).**
	Oemcomma	**The OEM comma key on any country/region keyboard (Windows 2000 or later).**
	OemMinus	**The OEM minus key on any country/region keyboard (Windows 2000 or later).**
	OemOpenBrackets	**The OEM open bracket key on a US standard keyboard (Windows 2000 or later).**
	OemPeriod	**The OEM period key on any country/region keyboard (Windows 2000 or later).**
	OemPipe	**The OEM pipe key on a US standard keyboard (Windows 2000 or later).**

	Oemplus	**The OEM plus key on any country/region keyboard (Windows 2000 or later).**
	OemQuestion	**The OEM question mark key on a US standard keyboard (Windows 2000 or later).**
	OemQuotes	**The OEM singled/double quote key on a US standard keyboard (Windows 2000 or later).**
	OemSemicolon	**The OEM Semicolon key on a US standard keyboard (Windows 2000 or later).**
	Oemtilde	**The OEM tilde key on a US standard keyboard (Windows 2000 or later).**
	P	**The P key.**
	Pa1	**The PA1 key.**
	Packet	**Used to pass Unicode characters as if they were keystrokes. The Packet key value is the low word of a 32-bit virtual-key value used for non-keyboard input methods.**
	PageDown	**The PAGE DOWN key.**
	PageUp	**The PAGE UP key.**
	Pause	**The PAUSE key.**
	Play	**The PLAY key.**
	Print	**The PRINT key.**
	PrintScreen	**The PRINT SCREEN key.**
	Prior	**The PAGE UP key.**
	ProcessKey	**The PROCESS KEY key.**
	Q	**The Q key.**
	R	**The R key.**
	RButton	**The right mouse button.**
	RControlKey	**The right CTRL key.**
	Return	**The RETURN key.**
	Right	**The RIGHT ARROW key.**
	RMenu	**The right ALT key.**

RShiftKey	**The right SHIFT key.**
RWin	**The right Windows logo key (Microsoft Natural Keyboard).**
S	**The S key.**
Scroll	**The SCROLL LOCK key.**
Select	**The SELECT key.**
SelectMedia	**The select media key (Windows 2000 or later).**
Separator	**The separator key.**
Shift	**The SHIFT modifier key.**
ShiftKey	**The SHIFT key.**
Sleep	**The computer sleep key.**
Snapshot	**The PRINT SCREEN key.**
Space	**The SPACEBAR key.**
Subtract	**The subtract key.**
T	**The T key.**
Tab	**The TAB key.**
U	**The U key.**
Up	**The UP ARROW key.**
V	**The V key.**
VolumeDown	**The volume down key (Windows 2000 or later).**
VolumeMute	**The volume mute key (Windows 2000 or later).**
VolumeUp	**The volume up key (Windows 2000 or later).**
W	**The W key.**
X	**The X key.**
XButton1	**The first x mouse button (five-button mouse).**
XButton2	**The second x mouse button (five-button mouse).**

	Y	**The Y key.**
	Z	**The Z key.**
	Zoom	**The ZOOM key.**

Literatura:

Microsoft Visual C# 2005 Step by Step, John Sharp, izdano leta 2005
Microsoft Visual C# 2003 Step by Step, John Sharp, izdano leta 2003
Murach's C# 2005 Training & reference, Joel Murach, Mike Murach & Associates Inc. 2006
Microsoft Visual C# .NET, Mickey Williams, izdano leta 2002
http://www.functionx.com/csharp/
http://penelope.fmf.uni-lj.si/diri0607/index.php/Kategorija:Naloge

Kazalo vsebine