REPUBLIKA SLOVENIJA | MINISTRSTVO ZA ŠOLSTVO IN ŠPORT
KONZORCIJ ŠOLSKIH CENTROV
Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski sklad za regionalni razvoj

RAČUNALNIŠTVO

PRAKTIČNO PROGRAMIRANJE 3 – PPR

Srečo Uranič

www.bodiprofi.si

munus 2

Uvajanje novih izobraževalnih programov v srednjem poklicnem in strokovnem izobraževanju s področja tehnike za obdobje 2008-2012.
Operacijo delno sofinancira Evropska unija iz Evropskega socialnega sklada in Ministrstvo za šolstvo in šport.

SPLOŠNE INFORMACIJE O GRADIVU

Izobraževalni program: Tehnik računalništva

Ime modula: Praktično programiranje 3 – PPR

Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo:
Osnove metod, varovalni bloki in razhroščevanje. Tabele, zbirke in strukture. Razredi in objekti. Datoteke in načrtovanje zahtevnejših programov.

Avtor: Srečo Uranič

Recenzent: Boštjan Vouk

Lektorica: Milena Ilić

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

Uranič, Srečo
Računalništvo [Elektronski vir] : Praktično programiranje 3 – PPR / Srečo Uranič. - El. knjiga. - Kranj : Konzorcij šolskih centrov, 2010.

Način dostopa (URL): http://munus2.tsc.si. - Projekt MUNUS 2

ISBN xxxxxxxxxxxxx
xxxxxxxxxx

Izdajatelj: Konzorcij šolskih centrov Slovenije v okviru projekta MUNUS 2
Slovenija, avgust 2010

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

POVZETEK/PREDGOVOR

Gradivo ***Praktično programiranje*** je namenjeno dijakom 3. letnika SSI – Tehnik računalništva in dijakom 4 letnika PTI - Tehnik računalništva. Pokriva vsebinski del, naveden v katalogu znanja, pri čemer sem kot izbrani programski jezik izbral programski jezik C#. Osnove programskega jezika C# in razvojnega okolja znotraj razvojnega okolja Microsoft Visual C# 2010 Express. Osnove za izdelavo konzolnih aplikacij gradivo ne vsebuje, saj ju dijaki spoznajo že v predhodnem izobraževanju, dostopne pa so tudi v mojih mapah http://uranic.tsckr.si/ na šolskem strežniku TŠCKR.

V besedilu je veliko primerov programov in zgledov, od najenostavnejših do bolj kompleksnih. Dijake, ki bi o posamezni tematiki radi izvedeli več, vabim, da si ogledajo tudi gradivo, ki je navedeno v literaturi.

Gradivo je nastalo na osnovi številnih zapiskov avtorja, črpal pa sem tudi iz že objavljenih gradiv, pri katerih sem bil "vpleten". V gradivu je koda, napisana v programskem jeziku, nekoliko osenčena, saj sem jo na ta način želel tudi vizualno ločiti od teksta gradiva.

V besedilu so poleg teorije tudi številni zgledi, na koncu poglavij pa tudi vaje. Rešitve vaj, ki so zbrane na koncu celotnega gradiva, predstavljajo le eno izmed možnih rešitev, mogoče niti ne najboljšo in najkrajšo. Ker se programiranja seveda ne da naučiti le s prepisovanjem tujih programov, pričakujem, da bodo dijaki poleg skrbnega študija zgledov in rešitev pisali programe tudi sami. Zato jim predlagam, da rešijo naloge, ki so objavljene v številnih, na spletu dosegljivih, zbirkah nalog.

Bralcem, ki bodo uporabljati ta učbenik pri svojem predmetu, priporočam tudi uporabo spletne učilnice, za katero menim, da je postala nepogrešljiv pripomoček pri poučevanju programiranja.

Gradivo ***Praktično programiranje*** opisuje: osnove razumevanja in pisanja metod, pomen varovalnih blokov in razhroščevanja, delo s tabelaričnimi podatki in zbirkami, osnove objektno orientiranega programiranja, delo z datotekami, ter načrtovanje zahtevnejših programov .

Ključne besede: metode, varovalni bloki, razhroščevanje, tabele, zbirke, naštevanje, strukture, razredi, objekti, konstruktor, preobtežitev, enkapsulacija, polimorfizem, sklad, kopica, smetar, datoteke, knjižnica, rekurzija.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO SLIK:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

KAZALO TABEL

UČNI CILJI

Učenje programiranja je privajanje na algoritmični način razmišljanja. Poznavanje osnov programiranja in znanje algoritmičnega razmišljanja je tudi nujna sestavina sodobne funkcionalne pismenosti, saj ga danes potrebujemo praktično na vsakem koraku. Uporabljamo oz. potrebujemo ga:

- pri vsakršnem delu z računalnikom;
- pri branju navodil, postopkov (pogosto so v obliki "kvazi" programov, diagramov poteka);
- za umno naročanje ali izbiranje programske opreme;
- za pisanje makro ukazov v uporabniških orodjih;
- da znamo pravilno predstaviti (opisati, zastaviti, ...) problem, ki ga potem programira nekdo drug;
- ko sledimo postopku za pridobitev denarja z bankomata;
- ko se odločamo za podaljšanje registracije osebnega avtomobila;
- za potrebe administracije – delo z več uporabniki;
- za ustvarjanje dinamičnih spletnih strani;
- nameščanje, posodabljanje in vzdrževanje aplikacije;
- zbiranje, analiza in dokumentiranje zahtev naročnika, komuniciranje in pomoč naročnikom;
- za popravljanje "tujih" programov.

CILJI

- Spoznavanje oz. nadgradnja osnov programiranja s pomočjo programskega jezika C#.
- Poznavanje in uporaba razvojnega okolja Visual Studio za izdelavo programov.
- Načrtovanje in izdelava preprostih in kompleksnejših programov.
- Privajanje na algoritmični način razmišljanja.
- Manipuliranje s podatki shranjenimi v datotekah na računalniškem mediju.
- Ugotavljanje in odpravljanje napak v procesu izdelave programov.
- Uporaba znanih rešitev na novih primerih.
- Poznavanje dela s tabelaričnimi podatki.
- Spoznavanje osnov sodobnega objektno orientiranega programiranja.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

PAPAJŠČINA

Papajščina je jezik, s katerim so se včasih pogovarjali mladostniki z namenom, da jih ostali (predvsem starši) niso razumeli in so jih tako lahko obirali do obisti. Jezik hitro razume vsak, ki pozna njegove osnove: za vsakim samoglasnikom v stavku dodamo črko 'p' in še ponovimo ta samoglasnik. Stavek *"Govorimo v papajščini"* bi se po papajsko tako glasil *"Go**po**vo**po**ri**pi**mo**po** v pa**pa**pa**pa**jšči**pi**ni**pi**"*.

Radi bi napisali metodo *Papajscina*, ki bo kot vhodni podatek dobila ustrezno besedilo, ki ga bo nato prevedla v *papajščino*. Pri tem bomo spoznali pojem metode, kako so metode sestavljene in zakaj jih sploh uporabljamo. Naučili se bomo napisati svoje metode v jeziku C# in kako si lahko pri pisanju pomagamo z razvojnim okoljem. Obenem bomo spoznali pomen varovalnih blokov in kako si pri odkrivanju in popravljanju napak v programih pomagamo z tehniko, ki jo imenujemo razhroščevanje.

METODE

UVOD

Vsi naši dosedanji programi so bili večinoma kratki in enostavni ter zaradi tega pregledni. Pri zahtevnejših projektih pa postanejo programi daljši, saj zajemajo več kot en izračun, več pogojev, več zank. Če so napisani v enem samem kosu postanejo nepregledni. Poleg tega se zaporedja stavkov pri daljših programih lahko tudi večkrat ponovijo. Vzdrževanje in popravljanje takih programov je težavno, časovno zahtevno, verjetnost za napake pa precejšnja.

Vse to so razlogi za pisanje metod. S pomočjo metod programe smiselno razbijemo na podnaloge, vsako podnalogo pa izdelamo posebej. Te manjše podnaloge imenujemo podprogrami, rešimo jih z lastnimi metodami, zaženemo pa jih v glavnem programu. Program torej razdelimo v več manjših, "neodvisnih" postopkov, in nato obravnavamo vsak postopek posebej. Metodam v drugih programskih jezikih rečemo tudi funkcije, procedure ali podprogrami.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Na ta način bo naš progam krajši, bolj pregleden, izognili se bomo večkratnemu ponavljanju istih stavkov, pa še popravljanje in dopolnjevanje bo enostavnejše. Bistvo metode je tudi v tem, da ko je enkrat napisana, moramo vedeti le še, kako jo lahko uporabimo, pri tem pa nas večinoma ne zanima več, kako je pravzaprav sestavljena. Napisano metodo lahko uporabimo večkrat, seveda pa jo lahko uporabimo tudi v drugih programih.

Metode smo v bistvu že uporabljali pri spoznavanju osnov progarmiranja (modul UPN, ali pa PPR v prejšnjih letnikih), a se tega mogoče nismo zavedali. Za izpisovanje na zaslon smo uporabljali metodo *Console.WriteLine()*, pri pretvarjanju niza v število smo uporabljali metodo *int.Parse()*, pri računanju kvadratnega korena metodo *Math.Sqrt()* in številne druge. Vse te metode so torej že napisane, vse, kar moramo vedeti o njih, pa je, kakšen je njihov namen in kako jih uporabiti.

Seveda pa lahko metode pišemo tudi sami. Pravzaprav smo vsaj eno od metod že ves čas pisali – to je bila metoda *Main()*. Ogrodje zanjo nam je zgeneriralo že razvojno okolje, mi pa smo doslej pisali le njeno vsebino oziroma kot temu rečemo strokovno – **glavo**. Glavo metode nam je zgeneriralo že razvojno okolje, **telo** metode pa smo napisali sami. Metoda *Main()* predstavlja izhodišče našega programa (t.i. **glavni program**) in ima zaradi tega že vnaprej točno predpisano ime, obliko in namen. Metode, ki se jih bomo naučili pisati sami, pa bodo imele prav tako specifičen namen, obliko in ime, vse te lastnosti pa bomo določili mi sami pred in med njihovim pisanjem. Držati pa se moramo pravila, da morajo biti metode, ki jih napišemo, pregledne, če se la da uporabne tudi v drugih programih, biti morajo samozadostne, prav tako jih tudi ne napišemo izrecno za izpisovanje ali branje podatkov, razen če to ni osnovni namen te metode.

DELITEV METOD

V jeziku C# metode v splošnem delimo na statične in objektne. V tem poglavju bomo obravnavali le statične (***static***) metode. Zaenkrat povejmo le, da statične metode kličemo nad razredom, medtem, ko objektne nad objektom. Metode ločimo tudi po načinu dostopa na javne (***public***), privatne (***private***) in zaščitene (***protected***). Ker bomo uporabljali le javne metode, se v način dostopa zaenkrat ne bomo spuščali. Več o razlikah med statičnimi in objektnimi metodami, ter o načinih dostopa, pa bomo spoznali v poglavju o razredih in objektih.

DEFINICIJA METODE

Metodo v program vpeljemo z definicijo. Definicija metode je sestavljena iz **glave** oziroma **deklaracije** metode in **telesa** metode, napišemo pa jo v celoti pred ali pa za glavno metodo *Main* našega programa.

V deklaracijo zapišemo najprej **dostopnost** metode (torej ali je metoda javna - *public*, privatna - *private* ali zaščitena - *protected*), nato **vrsto** metode, s katero povemo, ali je metoda statična (*static*) ali objektna. V tem razdelku bodo vse naše metode vrste ***public static***. Sledi ji **tip** rezultata metode. Tip rezultata metode je poljuben podatkovni tip (npr. *int*, *double[]*, *string*) in pomeni tip vrednosti, ki ga metoda vrača. Nato z **imenom metode** povemo, kako se imenuje

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

metoda. Velja dogovor, da se imena metod začnejo z veliko črko. Metodam damo smiselna imena, ki povedo, kaj metoda dela. Ime metode je poljubno, ne smemo pa za ime metode uporabiti rezerviranih besed (ime metode **ne sme** biti npr. *string*, *do*, *return*, …). V imenih metod **ne sme** biti presledkov in nekaterih posebnih znakov (npr. znakov '/', ')', '.' …), prav tako pa v imenih metod niso zaželeni šumniki. Za imenom metode zapišemo okrogle oklepaje, ki so obvezni del deklaracije. Znotraj oklepajev zapišemo **argumente metode**, ki jih metoda sprejme (imena argumentov), in kakšnega tipa so. Več argumentov med sabo ločimo z vejico. Če metoda ne sprejme nobenih argumentov, napišemo le prazne okrogle oklepaje *()*. Sledi **telo metode**, to je del, ki je znotraj bloka *{}*. V telo metode zapišemo stavke, ki izvršijo nalogo metode.

Posebne metode so tiste, ki ne vračajo nobenega rezultata, ampak le opravijo določene delo (na primer nekaj izpišejo, narišejo sliko, pošljejo datoteko na tiskalnik ...). Pri takih metodah kot tip rezultata navedemo tip ***void***. Metode tipa *void* v svojem telesu nimajo stavka ***return***.

Javna metoda – metoda je dostopna vsem delom našega programa (do nadaljnjega bodo vse metode javne)

Statična metoda (do nadaljnjega bodo vse metode statične – kaj pravzaprav to pomeni, bo jasno šele pri poglavju o razredih in objektih)

Tip metode: določa tip rezultata, ki ga metoda vrača.

```
public static tip_metode Ime_metode(parametri metode)   // Glava metode
{
     ... stavki
     return ... // vračanje vrednosti.                                Telo
metode
     // Stavka return ni, če je metoda tipa void
}
```

Poglejmo si nekaj primerov deklaracij metod:

```
public static int Beri();
```

Metoda *Beri* je javna (*public*), statična (*static*), vrača rezultat tipa *int* in ne sprejme nobenega argumenta (znotraj oklepajev ni argumentov).

```
public static void Nekaj(string stavek);
```

Metoda *Nekaj* je javna (*public*), statična (*static*), ne vrača rezultata (tip *void*) in sprejme en argument, ki se imenuje *stavek* in je tipa *string*.

```
public static int Max(int a, int b);
```

Metoda *Max* je javna (*public*), statična (*static*), vrača rezultat tipa *int* in sprejme dva argumenta, ki sta celi števili (tip *int*), prvi argument se imenuje *a*, drugi *b*.

```
public double Abs();
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Metoda *Abs* je javna (*public*), objektna (ni določila *static*), vrača rezultat tipa *double* in ne sprejme nobenega argumenta (znotraj oklepajev ni argumentov).

Parametri metod niso obvezni, saj lahko napišemo metodo, ki ne sprejme nobenega parametra, take metode v splošnem niso preveč uporabne.

Pozdrav uporabniku

Napišimo metodo, ki prebere naše ime in nas pozdravi (npr. za ime "Janez" izpiše *Pozdravljen Janez*).

```
// ker metoda ne bo vrnila ničesar, je tipa void
public static void PozdravUporabniku()
{
     // preberemo podatek
    Console.Write("Vnesi ime: ");
    string ime = Console.ReadLine();
     // in ga izpišemo
    Console.WriteLine("Pozdravljen " + ime + "!");
}
```

Metodo smo napisali, potrebno pa jo še znati uporabiti oz. jo poklicati. Metode tipa *void* (ki ne vračajo ničesar) uporabimo v samostojnem stavku tako, da napišemo njihovo ime, v oklepajih pa še parametre, če seveda metoda parametre potrebuje, sicer pa napišemo le oklepaj in zaklepaj. Zgornjo metodo bi torej poklicali (npr. kjer koli v glavnem programu *Main*) takole:

```
PozdravUporabniku();
```

VREDNOST METODE

Če je tip metode različen od *void*, moramo vrednost, ki jo vrne metoda, določiti s stavkom

```
return izraz;
```

Vrednost metode je vrednost izraza, ki je naveden za ključno besedo ***return***. V opisu postopka je stavek *return* lahko na več mestih. Kakor hitro se eden izvede, se izvajanje metode konča z vrednostjo, kot jo določa izraz, naveden pri stavku *return*. Če metoda kaj izpisuje ali riše na ekran, to ni rezultat te metode, ampak samo stranski učinek, ki ga ima metoda. Take metode običajno ne vračajo nobenih rezultatov. V tem primeru je rezultat metode tipa *void*.

Vesela gosenica

Napišimo metodo tipa string, ki predstavlja sliko vesele gosenice (slika vesele gosenice dolžine 5 je npr. takale: **(:IIIII**). Gosenica je sestavljena iz glave(oklepaj in dvopičje), ter iz trupa (velika črka **I**). Dolžina trupa gosenice naj bo naključna in sicer med 1 in 10.

```
public static string VeselaGosenica()
{
    Random naklj = new Random(); //generator naključnih števil
    int dolzina = naklj.Next(1, 11);//naključno celo število med 1 in 10
    string gosenica = "(:";  //glava gosenice
    for (int i = 0; i <= dolzina; i++)
        gosenica = gosenica + 'I';  //gosenici dodajamo členke
    return gosenica; //vnemo string, ki predstavlja sliko gosenice
}
```

Metoda *VeselaGosenica* je javna, statična, nima nobenih vhodnih podatkov, vrne pa rezultat tipa *string*. Še primer klica metode v glavnem programu:

```
//ker metoda ni tipa void, jo moramo klicati v stavku
Console.WriteLine(VeselaGosenica());
```

Vsota lihih števil

Napišimo metodo, ki ne sprejme nobenih parametrov, izračuna in vrne pa vsoto vseh dvomestnih števil, ki so deljiva s 5!

```
public static int VsotaDvomestnih()  // glava funkcije
{
     int vsota = 0;  // začetna vsota je 0
     for (int i = 1; i <= 100; i++)
     {
         if (i % 5 == 0)  // če ostanek pri deljenju deljiv s 5
         vsota = vsota + i;  // potem število prištejemo k vsoti
     }
     // metoda vrne rezultat: vsoto vseh dvomestnih števil, deljivih s 5
    return vsota;
}
```

Še klic metode v glavnem programu: ker metoda *VsotaDvomestnih()* vrača rezultat (tip *int*), je ne moremo klicati samostojno, ampak v nekem prireditvenem stavku, lahko pa tudi kot parameter v drugi metodi.

```
int vsota100 = VsotaDvomestnih(); // klic metode v prireditvenem stavku
```

```
Console.WriteLine(VsotaDvomestnih()); // klic metode kot parametra metode
                                      //WriteLine
```

Zgornja metoda je sicer povsem legalna, ker pa nima vhodnih parametrov, je uporabna le za točno določen, ozek namen. Če želimo narediti metodo res uporabno, uporabljamo parametre.

ARGUMENTI METOD – FORMALNI IN DEJANSKI PARAMETRI

V glavi metode lahko uporabimo tudi vhodne podatke - argumente. Pravimo jim tudi parametri metode. Parametre, ki jih napišemo, ko metodo pišemo, imenujemo formalni parametri. Ko pa neko metodo pokličemo, formalne parametre nadomestimo z dejanskimi parametri.

Iskanje večje vrednosti dveh števil

Napišimo metodo, ki dobi za parametra poljubni celi števili, vrne pa večje izmed teh dveh števil.

Ker bo metoda imela za parametra dve celi števili (tip *int*), je večje izmed teh dveh prav gotovo tudi tipa *int*. Metoda bo torej vrnila celo število, zaradi česar bomo za tip metode uporabili tip *int*.

```
public static int Max(int a, int b)  // metoda ima dva parametra, a in b
{
    if (a > b)   // če a večji od b metoda vrne število a
        return a;
    return b;    // sicer pa metoda vrne število b
}
```

Parametra *a* in *b*, ki smo ju napisali v glavi metode, sta formalna parametra. Ko bomo metodo poklicali, pa bomo zapisali dejanska parametra. Metodo lahko uporabimo tako, da v glavnem programu najprej preberemo (določimo, ustvarimo) dve celi števili, nato pa ti dve števili uporabimo za parametra metode *Max*.

```
Console.Write("Prvo število: ");
/* ker ReadLine vrača string, je potrebna pretvorba v celo število: uporabimo
npr. metodo int.Parse*/
int prvo = int.Parse(Console.ReadLine());
Console.Write("Drugo število: ");
int drugo = int.Parse(Console.ReadLine());

int vecje = Max(prvo, drugo);  // parametra prvo in drugo sta dejanska
                               //parametra
Console.WriteLine("Večje od vnesenih dveh števil je : " + vecje);
```

Metodo *Max* bi seveda lahko poklicali tudi takole:

```
Console.WriteLine("Večje od vnesenih dveh števil je : " + Max(prvo, drugo));
```

Pri klicu metode *Max* je parameter *a* dobil vrednost spremenljivke *prvo*, parameter *b* pa vrednost spremenljivke *drugo*. Formalna parametra *a* in *b* smo torej pri klicu metode nadomestili z dejanskima parametroma *prvo* in *drugo*. Namesto spremenljivk bi seveda pri klicu metode lahko uporabili tudi poljubni dve števili ali pa številska izraza, npr.:

```
Console.WriteLine("Večje izmed števil 6 in 11 je : " + Max(6,11);
Console.WriteLine("Večje od vnesenih izrazov je : " + Max(5+2*3,41-3*2));
```

Metodo *Max* pa lahko pa pokličemo tudi takole:

```
Random naklj = new Random();
int prvo = naklj.Next(100);
int drugo = naklj.Next(100);
int tretje = naklj.Next(100);
Console.WriteLine("Največje izmed treh števil je: " + Max(prvo, Max(drugo,
tretje)));
```

Zvezdice

Napišimo metodo, ki nariše vrstico iz samih zvezdic. Število zvezdic naj bo parameter te metode.

```
public static void zvezdice(int n)
{
    for (int i = 0; i < n; i++) //v zanki rišemo n zvezdic, drugo ob drugi
        Console.Write('*');
    Console.WriteLine();
}

static void Main(string[] args)
{
    //nekaj klicev metode Zvezdice
    zvezdice(10);
    zvezdice(80);
    zvezdice(2);

    //Narisali bi radi trikotnik
    //*
    //**
    //***
    //****
    zvezdice(1);
    zvezdice(2);
    zvezdice(3);
    zvezdice(4);
    Console.ReadKey();
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
}
```

Število znakov v stavku

Napišimo metodo, ki dobi dva parametra: poljuben stavek in poljuben znak. Metoda naj ugotovi in vrne, kolikokrat se v stavku pojavi izbrani znak.

```
static int Kolikokrat(string stavek, char znak) // metoda
{
    int skupaj = 0;  // začetno število znakov je 0
    for (int i = 0; i < stavek.Length; i++)
    {
        if (stavek[i] == znak)  // tekoči znak primerjamo z našim znakom
            skupaj++;
    }
    return skupaj;  // metoda vrne skupno število znakov v stavku
}
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 stavku je " + Kolikokrat(stavek, znak) + " znakov  "
+ znak + ".");
}
```

Dopolnjevanje niza

Napišimo metodo *DopolniNiz*, ki sprejme niz *s* in naravno število *n* ter vrne niz dolžine *n*. V primeru, da je dolžina niza s večja od *n*, spustimo zadnjih nekaj znakov. Drugače pa na konec niza *s* dodamo še ustrezno število znakov '*+*'. Preverimo delovanje metode.

```
public static string DopolniNiz(string niz, int n)
{   /*  Metoda DopolniNiz vrne nov niz dolžine n, ki ga dobimo tako, da
bodisi skrajšamo niz niz, ali pa ga dopolnimo z znaki '+'.*/
    int dolzina = niz.Length;  //  Določimo dolžino niza
    string novNiz = "";  //  Nov niz
    if (dolzina > n)
    {  //  Dolžina niza je večja od n
        novNiz = niz.Substring(0, n);
    }
```

```
    else
    {  // dolžina niza je manjša ali enaka n
        novNiz += niz;
        for (int i = dolzina; i < n; i++)
        {
            novNiz += '+';  // dodamo manjkajoče znake'+'
        }
    }
    return novNiz;  // vrnemo nov niz
}
public static void Main(string[] args)
{
    Console.Write("Vnesi niz: ");
    string niz = Console.ReadLine();
    Console.Write("Vnesi n: ");
    int n = int.Parse(Console.ReadLine());
    Console.WriteLine("Nov niz: " + DopolniNiz(niz, n));  // klic metode
}
```

Razlaga

Program začnemo z metodo *DopolniNiz*, ki ji v deklaraciji podamo parametra *niz* in *n*. V metodi določimo dolžino niza *niz*, ki jo shranimo v spremenljivko *dolzina*. Določimo nov niz *novNiz*, ki je na začetku prazen. Preverimo pogoj v pogojnem stavku. Če je izpolnjen (resničen), kličemo metodo *Substring()*, s katero izluščimo podniz dolžine *n*. Rezultat te metode shranimo v spremenljivki *novNiz*. Če pogoj ni izpolnjen (neresničen), potem nizu *novNiz* dodamo niz *niz* in ustrezno število znakov '+'. Te nizu dodamo s pomočjo zanke *for*. Na koncu metode s ključno besedo *return* vrnemo niz *novNiz*.

Delovanje metode preverimo v glavni metodi *Main*. V tej metodi preberemo podatka iz konzole, ju shranimo v spremenljivki *niz* in *n* ter s pomočjo klicane metode *DopolniNiz* izpišemo spremenjeni niz.

KLIC PARAMETOV PO REFERENCI

V vseh dosedanjih primerih so bili parametri, ki smo jih posredovali metodam, posredovani na privzeti način. Takemu načinu pravimo **prenos parametrov 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 vrednosti parametra v metodi ni vplivala na velikost spremenljivke, ki smo jo navedli pri klicu metode.

Parametre pa lahko pri klicu metode kličemo tudi po **referenci**. V tem primeru dobi metoda le referenco na ustrezno spremenljivko, kar dejansko pomeni, da vsaka sprememba tega parametra v metodi, pomeni spremembo vrednosti spremenljivke, ki smo jo uporabili pri klicu te metode. Klic po referenci dosežemo s pomočjo rezervirane besede **ref**. Vendar pozor: besedico **ref** moramo napisati tako pred tipom parametra v glavi metode, kot tudi pred imenom spremenljivke pri klicu metode.

```
//metoda
Public static void Metoda(ref string s) //parameter s je posredovan po
referenci
{
    s = "Novi stavek!";
}
//Glavni program
static void Main()
{
    string stavek = "Originalen stavek";
    Metoda(ref stavek); //parameter stavek je klican po referenci
    Console.WriteLine(stavek);// izpis:Novi stavek!
    Console.ReadKey();
}
```

Zamenjava spremenljivk

Sestavimo metodo, ki naj dobi za parametra dve spremenljivki *st1* in *st2*, metoda pa naj zamenja vrednosti teh dveh spremenljivk.

```
//Metoda
static void Zamenjaj(ref int st1,ref int st2)//parametra klicana po referenci
{
    //ciklična zamenjava vrednosti spremenljivk
    int zacasna = st1;
    st1 = st2;
    st2 = zacasna;
}
//Glavni program
static void Main(string[] args)
{
    int stevilo1 = 200;
    int stevilo2 = 500;
    Console.WriteLine("Število 1: " + stevilo1 + ", število 2: " + stevilo2);
    //Izpis: Število1: 200, število 2: 500
    Zamenjaj(ref stevilo1, ref stevilo2); //klic parametrov po referenci
    Console.WriteLine("Število 1: " + stevilo1 + ", število 2: " + stevilo2);
    //Izpis: Število1: 500, število 2: 200
}
```

Obrni niz

Napišimo metodo, ki obrne niz znakov (npr "abeceda" pretvori v "adeceba").

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
//metoda: parameter stavek je klican po referenci
static void Obrni(ref string stavek)
{
    string pomozni = "";
    //v niz pomozni dodajamo znake prvotnega niza od zadaj naprej
    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: "); //originalni stavek preberemo
    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");
    Console.ReadKey();
}
```

Jezik C# pozna še en način klica parametrov po referenci, to je klic s pomočjo rezervirane besede **out**. Klic po referenci s pomočjo rezervirane besedica *out* je sicer podoben klicu z besedico *ref*, razlika pa je v tem, da klic s pomočjo besedice *ref* zahteva, da je spremenljivka, ki nastopa kot parameter metode, pred klicem že inicializirana. Tudi pri klicu s pomočjo besedice *ref* pa velja, da jo moramo napisati tako pred tipom parametra v glavi metode, kot tudi pred imenom spremenljivke pri klicu metode.

PISANJE METODE S POMOČJO RAZVOJNEGA OKOLJA

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

```
static void Main(string[] args)
{
    Console.Write("Velikost trikotnika: ");
    int n = Convert.ToInt32(Console.ReadLine());
    //Tole kodo v glavnem programu bi radi nadomestili z metodo
    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();
    }
}
```

Temneje osenčeno kodo, ki bi jo radi s pomočjo čarovnika zapisali v novo metodo, 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 metode (npr. *Trikotnik*) in ime potrdimo s klikom na gumb **OK**. Čarovnik nam nato sam ustvari ustrezno metodo.

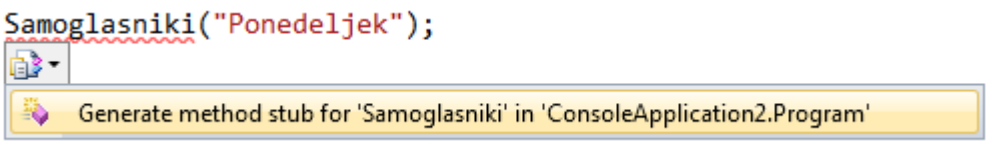
V našem primeru bo rezultat takle:

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

Kadar pa želimo pisati metodo, za katero koda v glavnem programu še ne obstaja, pa postopamo takole: v glavnem programu najprej zapišemo klic še ne obstoječe metode, ki se bo imenovala npr. *Samoglasniki*:

```
Samoglasniki("Ponedeljek");
```

Z miško nato kliknimo na besedico *Samoglasniki* in odprimo spustni seznam pod začetno črko metode:



Slika 1: Ustvarjanje metode s pomočjo razvojnega okolja.

Ostane še klik na vrstico "*Generate method …*" in razvojno okolje nam za glavnim programom ustvari glavo in telo želene metode:

```
private static void Samoglasniki(string p)
```

```
{
    throw new NotImplementedException();
}
```

Telo metode (stavek "*Throw new …*") pobrišemo in napišemo svojo vsebino metode.

POVZETEK

Naučili smo se pisati lastne metode. Prav v vseh programskih jezikih obstaja ogromno število že napisanih metod in preden se odločimo za pisanje nove metode, se raje prepričajmo, če morda taka ali podobna metoda že ne obstaja. Seveda pa bomo slej ko prej naleteli na problem, ko bo pisanje nove metode neizbežno. Naš program bo tako postal bolj pregleden pa še popravljanje ali pa ažuriranje kadarkoli kasneje bo veliko lažje.

VAJE

1. Napiši program za samodejno generiranje gesel. Geslo naj bo naslednje oblike: 1. znak je naključna velika črka angleške abecede, 2. znak je naključna mala črka angleške abecede, 3. znak je naključen samoglasnik, od 4. znaka naprej pa so naključne številke od 0 do 9. Dolžina gesla je vhodni podatek te metode.

2. Napiši metodo Inicialke, ki sprejme dva niza znakov (ime in priimek) in vrne niz sestavljen iz inicialk. Primer: klic metode Inicialke("France", "Prešeren") naj vrne niz "F.P.".

3. Napiši metodo *ZadovoljnaGosenica*, ki nariše zadovoljno gosenico, ki bo imela n členov. En člen gosenice je znak '**I**', glava gosenice pa je sestavljena iz znakov '**(:**'. Število členov je vhodni podatek metode.

4. Na svetovnem spletu uporabljamo več kodnih tabel znakov, od katerih samo nekatere vsebujejo šumnike (č, š, ž). Da se šumniki ne pretvorijo v nerazumljive znake, jih pretvorimo v ustrezne sičnike (c, s, z) in jih take tudi izpišimo. Napiši metodo, ki dobi za parameter vrstico besedila s šumniki in na zaslon izpiše spremenjeno besedilo. Preverjaj tako velike kot tudi male znake!

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

5. Sestavi metodo Produkt s pomočjo katere boš izračunal produkt vseh sodih števil med dvema danima pozitivnima celima številoma. Primer klica:

```
Console.WriteLine("Skupni produkt: " + Produkt(10, 20));
```

6. Napiši metodo, ki dobi za vhodna podatka dve celi števili M in N. Metoda naj riše vrstice sestavljene iz N zvezdic, ter iz M pikic. N se povečuje za ena, M pa zmanjšuje za ena. Slika je gotova ko je M enak 0!

7. Napiši metodo Zeller, ki dobi tri vhodne podatke(dan, mesec in leto), ki predstavljajo nek datum. Metoda naj vrne string, ki predstavlja dan v tednu, ki pade na ta datum. Pomagaj si z Zellerjevo formulo, ki je uporabna če gre za datum v obdobju do leta 4000!

   x:= [(13*mes - 1)/5] + dan + [5*leto/4] + [21*stol/4], kjer je:
   [a] celi del števila a.
   dan ... dan v mesecu
   mes ... mesec - 2, če mesec ni januar ali februar in
   mesec + 10, če je januar ali februar; v tem primeru moramo leto zmanjšati za 1
   leto ... zadnji dve cifri leta
   stol ... prvi dve cifri leta

   Dan v tednu nam da ostanek števila x pri deljenju s 7 :

   0: nedelja, 1 : ponedeljek, 2 : torek, 3 : sreda, 4 : četrtek, 5 : petek, 6 : sobota

8. Verjetno je vsak med nami že kdaj prebral kak strip o Asterixu in Obelixu. Zato vemo, da se imena vseh Galcev zaključijo z "ix", na primer Asterix, Filix, Obelix, Dogmatix, itn. Napiši metodo, ki bo za vhodni podatek dobila ime nekega Rimljana in vrnila string "Galec!", če gre za galsko ime, v nasprotnem primeru pa bo vrnila "Rimljan!". Predpostavimo, da so vnesena imena dolga vsaj tri znake.

9. Število PI lahko izračunamo tudi kot vsoto vrste 4 - 4/3 + 4/5 - 4/7 + 4/9 - ... Napiši metodo, 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. Metoda naj ima za parameter število ujemajočih se decimalk. Za npr. 9 decimalk moramo sešteti 1.096.634.169 členov tega zaporedja, kar lahko traja kar nekaj časa.

10. Napiši metodo, ki iz vhodnega celega števila vrne novo število, v katerem so le sode cifre danega števila. Če so vse cifre danega števila lihe, naj metoda vrne 0.

VAROVALNI BLOKI IN OBRAVNAVA NAPAK

Tako kot v vsakdanjem življenju se tudi pri pisanju programov oz. pri njihovem delovanju pojavljajo napake. Zelo neprijetno je, če se program sredi delovanja "sesuje" (preneha delovati), ker programer ni predvidel, da lahko v določenih izjemnih primerih pride do "čudne" situacije. Na primer, da se uporabi negativni indeks, ali pa da datoteka, kamor naj bi se zapisali rezultati, ne obstaja, da uporabnik kot število, s katerim naj se deli, po pomoti vnese 0 in podobno. Dobro je, če imamo možnost, da v primeru napak program ustrezno reagira, pa če drugega ne, izpiše prijazno obvestilo, da je žal prišlo do take in take napake in da bo zaradi tega program prenehal z delovanjem.

C#, podobno kot drugi sodobni programski jeziki, pozna tako imenovan mehanizem *izjem* (*exceptions*). Če pri delovanju programa pride do napake, se sproži tako imenovana izjema. To izjemo lahko programsko prestrežemo in napišemo ustrezno kodo, kako naj program nadaljuje v tem primeru.

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. 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 lahko napako ustrezno obdelamo, ali pa enostavno le sporočimo uporabniku, da je prišlo do napake.

```
try
{
    // stavki, kjer lahko pride do napake
}
catch (System.Exception caught)
{
    // obdelava napake glede na vrsto napake
}
```

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!

Za obdelavo posameznih vrst napak seveda obstajajo številne metode, v katere pa se zaenkrat ne bom spuščali: *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 oknu.

V zgornjem primeru je uporabljena kar splošna metoda za prestrezanje napake (*System.Exception caught*); le-ta se odzove/odkrije na vsako izjemo (napako), tudi tako, ki se je morda sploh ne zavedamo. V primeru, da pa želimo v *catch* bloku le opozoriti uporabnika, da je prišlo do napake pa lahko blok *catch* uporabimo tudi brez parametrov takole:

```
try
{
    // Stavki, kjer lahko pride do napake
}
catch
{
    //Sporočilo uporabniku, da je prišlo do napake
}
```

Tak način je za začetnika tudi najbolj priporočljiv in v nadaljevanju ga bomo tudi največkrat uporabljali).

Varovalni blok za obravnavo prekinitev pri deljenju

Napišimo varovalni blok, v katerem bomo zaščitili program pred napačnim vnosom uporabnika: bodisi, da uporabnik ne bo vnesel števila, ali pa za delitelja vnesel število 0!

```
try
{
    //Če uporabnik ne vnesel številke, bo pri pretvorbi prišlo do napake,
    //program se bo nadaljeval v bloku catch
    int levo = Convert.ToInt32(Console.ReadLine());
    //Če uporabnik ne vnesel številke, bo pri pretvorbi prišlo do napake
```

```
    int desno = Convert.ToInt32(Console.ReadLine());
    //Če je drugo število enako 0, bo prišlo do napake, program se bo
    //nadaljeval v bloku catch
    float rezultat = levo / desno;
    Console.WriteLine(rezultat);
```

```
}
catch
{
    Console.WriteLine("Napaka v podatkih!!!");
}
```

Varovalni blok pri vnosu numeričnega podatka

Radi bi napisali program, v katerem bo uporabnik vnesel poljuben stavek, nato pa iz stavka vzel nek znak, glede na zaporedno številko.

```
string niz;
Console.Write("Vnesi stavek: ");
niz = Console.ReadLine();
try
{
    Console.Write("Kateri znak po vrsti v tem stavku te zanima: ");
   //pri pretvorbi lahko pride do napake
    int indZnaka = int.Parse(Console.ReadLine());
    Console.WriteLine(indZnaka+".ti znak v "+niz+" je " + niz[indZnaka - 1]);
}
catch
{
  Console.WriteLine("Nisi vnesel števila oz.je število preveliko/premajhno");
}
```

VEČKRATNI VAROVALNI BLOK: 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.

Večkratni varovalni blok pri vnosu podatkov

Napišimo gnezdeni varovalni blok za večkraten vnos podatkov – na ta način lahko npr. uporabnika učinkovito seznanjamo z napakami pri vnašanju podatkov! V praksi bi lahko vsak varovalni blok postavili v neskončno zanko, iz katere bi izšli le v primeru pravilnega vnosa ustreznega podatka.

```
Console.Write("Stranka:  ");
string stranka = Console.ReadLine();
try
{
    Console.WriteLine("Podatki o rojstvu: ");
    Console.Write("    Leto: ");
    int letoRojstva = Convert.ToInt32(Console.ReadLine());
    try
    {
        Console.Write("    Mesec: ");
        int mesecRojstva = Convert.ToInt32(Console.ReadLine());
        try
        {
            Console.Write("    Dan: ");
            int danRojstva = Convert.ToInt32(Console.ReadLine());
        }
        catch
        {
            Console.WriteLine("Nepravilno vnesen dan rojstva!");
        }
    }
    catch
    {
        Console.WriteLine("Nepravilno vnesen mesec rojstva!");
    }
}
catch (FormatException)
{
    Console.WriteLine("Nepravilno vneseno leto rojstva!");
}
Console.ReadKey();
```

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*.

```
try
{
   //Programska koda, ki bi jo napisali tudi če ne bilo varovalnega bloka;
}
finally
{
    //Blok kode, ki se izvede vedno, ne glede ali je v zgornjem bloku prišlo
do napake ali ne!
}
```

V delu *finally* lahko poskrbimo za določanje privzetih vrednosti spremenljivk, spremembe raznih nastavitev, zapremo odprte datoteke, v programih, kjer delamo z objekti pa lahko tudi za sproščanje pomnilnika.

V praksi pogosto uporabljamo tudi kombinacijo obeh metod: blok za obravnavo prekinitev gnezdimo v brezpogojni varovalni blok:

```
try
{
    // stavki, kjer lahko pride do napake
}
catch
{
    // obdelava napake
}
finally
{
    // blok, ki se izvede vedno, ne glede  na napako
}
```

Privzeta vrednost spremenljivke pri napačnem vnosu

V programu od uporabnika zahtevamo vnos decimalnega števila, V primeru njegovega napačnega vnosa (npr. da je namesto cifer vtipkal nek znak), pa bi radi s programom nadaljevali, a namesto uporabnikovega vnosa želimo uporabiti kar privzeto vrednost števila!

```
Console.Write("Vnesi decimalno število: ");
double stevilo=10; //Privzeta vrednost spremenljivke število

try
```

```
{
    // ReadLine vrača string, zato uporabimo pretvorbo v tip double
    stevilo = Convert.ToDouble(Console.ReadLine());
}
catch  // catch del se izvede le če pride do napake
{
    // obvestilo o napačnem vnosu, spremenljivka stevilo bo zaradi tega
    // ohranila začetno vrednost 0
    Console.WriteLine("Napačen vnos števila!");
}
finally // izvede se v vsakem primeru
{
    //V primeru napačnega vnosa bo izpis enak:   Vneseno stevilo = 10
    Console.Write("Vneseno število = "+ stevilo);
}
```

POVZETEK

Mehanizem varovalnih blokov in izjem nam omogoča pisanje programov, ki so veliko bolj odporni na napake, do katerih lahko pride zaradi nepredvidenih vrednosti spremenljiv v programih, še največkrat pa zaradi nepravilnih uporabnikovih vnosov podatkov. V tem razdelku smo si ogledali zgolj najnujnejše.

RAZHROŠČEVANJE

Kreiranje programskega vmesnika oz. zapis programske kode je le prva faza izdelave nekega projekta. Druga faza je prevajanje, povezovanje in izvajanje programa, njegov preizkus ter odpravljanje napak. Če smo pri pisanju kode naredili sintaktično (skladenjsko) napako ali nepravilno uporabili podatkovne tipe, nas na to v večini primerov opozori že prevajalnik (*Compile Time Error* oz. *Run Time Error*). Preveden program je tako že precej očiščen napak, a žal le sintaktičnih, ne pa tudi *semantičnih* oz. *logičnih* napak (pomenskih).

Logičnih napak v programu ne odkrije noben prevajalnik. Tudi sami jih včasih zelo težko odkrijemo in odpravimo. Pri tem pa nam *Visual C#* pomaga z vgrajenim **razhroščevalnikom** (**debuggerjem**). Na voljo imamo nastavljanje prekinitvenih točk, vrednotenje izrazov kar med

samim izvajanjem ter spremljanje in popravljanje vrednosti spremenljivk. Program pa lahko med izvajanjem prekinemo in ga izvajamo po korakih.

KORAČNO IZVAJANJE PROGRAMA

Vsak program lahko zaženemo brez prekinitve (*Ctrl + F5***)**, s klikom na *Debug →Start Debugging* (oziroma *F5*), ali pa s klikom na zeleni gumb *Start Debugging* v orodjarni. Program pa lahko izvajamo tudi stavek za stavkom – koračno.

Pri koračnem izvajanju programa so na voljo opcije

- *Debug → Step Into* (F11) - ob vsakem klicu poljubne lastne metode se poglobimo še v njegovo kodo.
- *Debug → Step Over* (F10) – premikanje preko klicev metod na nov stavek (ne zanima nas notranjost neke metode, torej gremo kar preko!).
- *Run To Cursor* (Ctrl+F10) – za preskok nezanimivih delov kode.
- *Debug → Step Out* (Shift + F10) – za prekinitev razhroščevanja znotraj trenutne metode; metoda se brez prekinitev izvede do konca.

Praviloma ob zagonu programa ne pričnemo takoj s koračnim izvajanjem, ampak program poženemo, ga prekinemo in od te točke naprej izvajamo koračno. Take točke lahko postavimo po različnih mestih znotraj našega programa. Pravimo jim prekinitvene točke (**breakpoints**).

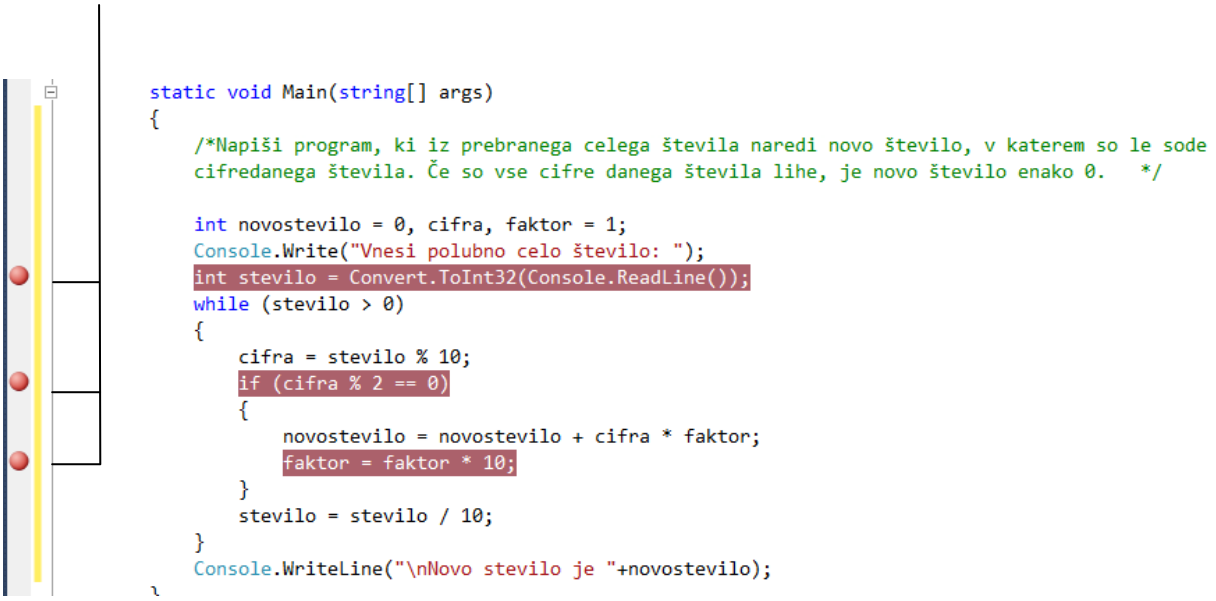
PREKINITVENE TOČKE - breakpoints

Prekinitvene točke lahko postavimo ali zbrišemo na tri načine. Najprej izberemo vrstico, kjer želimo nastaviti prekinitveno točko, nato pa izberemo eno od treh možnosti:

- Debug → Toggle Breakpoint
- Tipka **F9**
- Klik z miško v levem robu ustrezne vrstice s kodo

Na ekranu vidimo prekinitveno točko kot rdečo piko na levem robu okna s kodo, celotna vrstica pa je pobarvana rdeče.

Prekinitvene točke



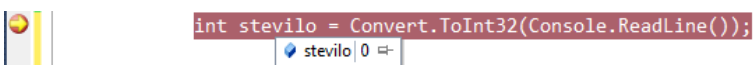
```
static void Main(string[] args)
{
    /*Napiši program, ki iz prebranega celega števila naredi novo število, v katerem so le sode
    cifredanega števila. Če so vse cifre danega števila lihe, je novo število enako 0.   */

    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 "+novostevilo);
}
```

Po končanem nastavljanju prekinitvenih točk program poženemo in v njem nemoteno delamo. Ko pridemo do prekinitvene toče, se izvajanje programa ustavi. Program je še vedno v pomnilniku, le delovanje je ustavljeno za toliko časa, da bomo pregledali vrednosti spremenljivk, oziroma raziskali del programa ki ne dela tako kot smo pričakovali. Od tu dalje ga lahko izvajamo korak za korakom, s pomočjo tipke F11 (*Step Into*) ali F10 (*Step Over*). Na naslednjo prekinitveno točko se prestavimo s tipko F5. Postopek se imenuje *razhroščevanje – debugging.*

RAZHROŠČEVANJE - debugging

Razhroščevanje nam ponuja številne možnosti za spremljanje poteka programa in vrednosti spremenljivk. Poglejmo le nekatere izmed njih:

- Dodatne prekinitvene točke lahko poljubno nastavljamo kar med samim razhroščevanjem. Postopek je enak kot pri začetnem nastavljanju prekinitev.
- Med razhroščevanjem lahko popravljamo kodo in nadaljujemo z razhroščevanjem ne da bi projekt ponovno zagnali.
- *Visualizer*: med razhroščevanjem se lahko postavimo na poljubno spremenljivko, in pod njo se pokaže vrstica z vrednostjo te spremenljivke.



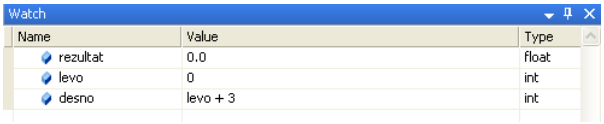
Slika 2: Ogled vrednosti spremenljivke med delovanjem programa.

- Če ima spremenljivka neko dolgo vrednost, npr. nek dolg *string*, se v okvirčku pod spremenljivko pokaže še ikona za povečevanje: če kliknemo nanjo se vrednost spremenljivke pokaže v posebnem oknu.

- Med razhroščevanjem se lahko v tekoči vrstici postavimo na neko spremenljivko in ji poljubno spremenimo vrednost.

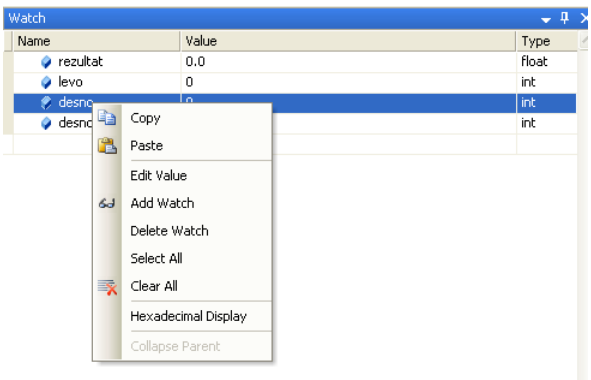
- Med samim razhroščevanjem lahko v okno *Watch* (okno *Watch* odpremo med delovanjem programa s klikom na *Debug* → *Windows* → *Watch*) potegnemo (*drag & drop*) poljubno spremenljivko in v tem oknu nato spreminjamo vrednosti spremenljivk. Opcija je dvosmerna (kar naredimo v *Watch* oknu se odseva v programu in obratno!!!).

- V *Watch* okno lahko pišemo izraze. V oknu *Watch* se postavimo na določeno spremenljivko, kliknemo desni miškin gumb, izberemo opcijo *Edit value* in nato v oknu zapišemo ustrezen izraz.



Slika 3: V oknu *Watch* lahko spremljemo in spreminjamo vrednosti spremenljivk.

- V *Watch* oknu imamo na voljo še nekaj možnosti. Če kliknemo z levo miškino tipko kamorkoli v to okno, lahko nato z desnim klikom miške odpremo meni za kopiranje, lepljenje, urejanje, dodajanje, brisanje ene ali brisanje vseh prekinitev.



Slika 4: V oknu *Watch* lahko odpremo meni za kopiranje, lepljenje, urejanje,..

- V oknu *Call Stack* (*Debug* → *Windows* → *Call Stack*) so navedeni vsi klici metod, ki so bile izvedene preden je prišlo do tekoče prekinitve (breakpointa)

- Okno *Immediate* (*Debug* → *Windows* → *Immediate*) se uporablja za ugotavljanje trenutne vrednosti spremenljivk, vrednosti izrazov oz. izvajanju stavkov, ki jih želimo izvesti kar med samim razhroščevanjem. Če hočemo izvedeti za trenutno vrednost neke spremenljivke, moramo v tekoči vrstici okna *Immediate* najprej obvezno zapisati znak **>** ,

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

nato znak **?** in potem še ime spremenljivke (lahko pa tudi zapišemo poljuben izraz) in končno stisnemo tipko <*Enter*>. V naslednji vrstici se nam prikaže ustrezna vrednost. (namesto znaka **?** lahko napišemo tudi *Debug.Print*).

Slika 5: Okno Immediate.

- Okno *Debug → Windows → Output* lahko uporabimo za prikaz statusnih sporočil za različne posebnosti razvojnega okolja.

Polna verzija *Visual C#* ponuja še nekaj drugih možnosti, a opcije ki jih ponuja Express verzija nam zaenkrat zadoščajo!

POVZETEK

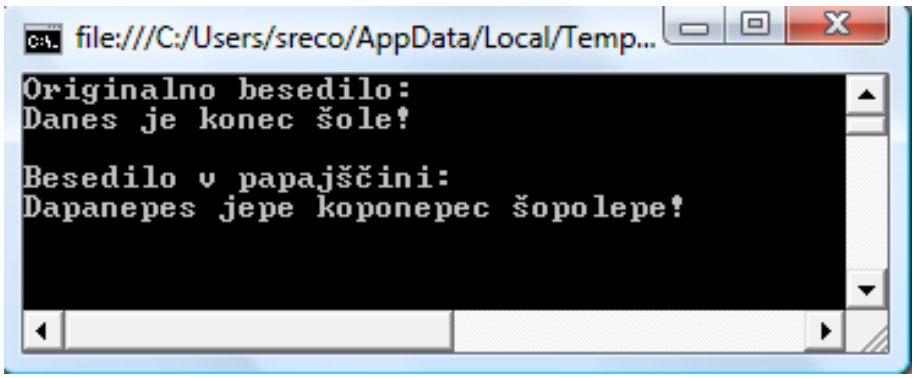
Nekateri razvijalci, predvsem začetniki, skušajo napake v programih odkriti in popraviti tako, da definirajo eno ali več posebnih globalnih spremenljivk, nato spremljajo njihove vrednosti, ter na ta način skušajo odkriti napake. Taka rešitev je seveda napačna in se je moramo izogibati, saj prav vsa sodobna razvojna okolja poznajo boljši način - razhroščevanje. Razhroščevanje je zelo močno in uporabno orodje za odkrivaje logičnih napak in ugotavljanje pravilnosti delovanja našega programa.

Papajščina

Sestavimo končno še metodo *Papajscina*. ki dani niz s pretvori v "papajščino". V metodo bomo vključili tudi varovalni blok (blok za obravnavo prekinitev).

```
// metoda
public static string Papajscina(string s)
{
```

```
    try
    {
        string sPapaj = "";
        for (int i = 0; i < s.Length; i++)
        {
           // metoda bo delovala tudi če so v stavku velike črke
           char znak = char.ToUpper(s[i]);
           if (znak=='A' || znak=='E' || znak=='I' || znak=='O' || znak=='U')
               sPapaj = sPapaj + s[i] + 'p' + s[i];// dodamo znak 'p' in še
                                                   //samoglasnik ki se ponovi
           else sPapaj = sPapaj + s[i];
        }
        return sPapaj;  // vračanje stavka v papajščini
    }
    catch
    {
        //če v bloku try kjerkoli pride do napake, metoda vrne prazen string
        return "";
    }
}

public static void Main(string[] args)
{
    string s = "Danes je konec šole!";
    Console.WriteLine("Originalno besedilo: \n"+s);// \n je skok v novo vrsto
    string papaj = Papajscina(s);
    Console.WriteLine("\nBesedilo v papajščini: \n"+papaj);
    Console.ReadKey();
 }
```



Slika 6: Izpis programa v katerem smo uporabili metodo *Papajscina*.

SEZNAM DRŽAV

Pri geografiji smo obravnavali evropske države, njihova glavna mesta in osnovne podatke o velikosti in številu prebivalcev. Ker moramo na pamet poznati glavna mesta vseh držav, bi radi napisali program, ki bo preveril in na koncu tudi ocenil naše znanje. Za ta namen moramo kreirati tabelo vseh držav, program nam bo naključno izbiral ime države, glavno mesto pa bomo morali uganiti sami. Spoznali bomo tabele in zbirke podatkov, njihove prednosti in slabosti. Za lažje delo s podatki, ki tvorijo neko celoto, bomo spoznali tudi strukture. Naučili se bomo napisati lastno strukturo in uporabljali najpomembnejše, v jezik C# že vgrajene strukture.

TABELE

UVOD

Pogosto se srečamo z večjo količino podatkov istega tipa, nad katerimi želimo izvajati podobne (ali enake) operacije. Takrat si pomagamo s tabelami. Tabelo ali polje (ang. *array*) v jeziku C# uporabljamo torej v primerih, ko moramo na enak način obdelati skupino vrednosti istega tipa. Oglejmo si tak zgled. Tabelarične spremenljivke, kot jih poznamo iz srednješolske matematike, imajo vse enako ime (npr. tabela), razlikujejo pa se po indeksu. Zaradi tega vsako od njih obravnavamo kot samostojno spremenljivko.

Dana je naloga: *Preberi 5 števil in jih izpiši v obratnem vrstnem redu*. Vsa števila moramo prebrati (shraniti), ker jih bomo potrebovali in razvrščali šele potem, ko bodo prebrana vsa. Ustrezen bo na primer tak program:

```
static void Main(string[] args)
{
     //  branje
     Console.WriteLine("Vnesi število:");
     string beri = Console.ReadLine();
     int x1 = int.Parse(beri);
     Console.WriteLine("Vnesi število:");
     beri = Console.ReadLine();
     int x2 = int.Parse(beri);
```

```
        Console.WriteLine("Vnesi število:");
        beri = Console.ReadLine();
        int x3 = int.Parse(beri);
        Console.WriteLine("Vnesi število:");
        beri = Console.ReadLine();
        int x4 = int.Parse(beri);
        Console.WriteLine("Vnesi število:");
        beri = Console.ReadLine();
        int x5 = int.Parse(beri);
        //  izpis
        Console.WriteLine("Števila v obratnem vrstnem redu: ");
        Console.WriteLine(x5);
        Console.WriteLine(x4);
        Console.WriteLine(x3);
        Console.WriteLine(x2);
        Console.WriteLine(x1);
        Console.ReadKey();
}
```

Če malo pomislimo, vidimo, da v našem program pravzaprav ponavljamo dva sklopa ukazov. Prvi je

```
Console.WriteLine("Vnesi število:");
beri = Console.ReadLine();
int x2 = int.Parse(beri);
```

in drugi

```
Console.WriteLine(x3);
```

Vsak sklop ponovimo 5x. Razlikujejo se le po tem, da v njih nastopa enkrat *x2*, drugič *x3*, tretjič *x4* … To nam da misliti, da bi lahko uporabili zanko, namesto običajnih spremenljivk pa moramo uporabiti *tabelarične spremenljivke*.

DEKLARACIJA TABELE

Prvi korak pri ustvarjanju tabele je **najava** le-te. To storimo tako, da za tipom tabele navedemo oglate oklepaje in ime tabele.

```
podatkovniTip[] imeTabele; // Najava tabele
```

Z zgornjim stavkom zgolj napovemo spremenljivko, ki bo hranila ***naslov*** bodoče tabele, ne zasedemo pa še nobene pomnilniške lokacije, kjer bodo elementi tabele dejansko shranjeni. Potrebno količino zagotovimo in s tem tabelo dokončno pripravimo z ukazom ***new***:

```
imeTabele = new podatkovniTip[velikost];  //Deklaracija tabele
```

Ukaz *new* zasede dovolj pomnilnika, da vanj lahko shranimo *velikost* spremenljivk ustreznega tipa in vrne njegov naslov. Velikost tabele je enaka številu elementov, ki jih lahko hranimo v tabeli.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Napoved in zasedanje pomnilniške lokacije lahko združimo v en stavek:

podatkovniTip[] imeTabele = new podatkovniTip[velikost];

Primeri najave tabele:

```
//  tabela 10 celih števil
int[] stevila = new int[10];

//  tabela 3 realnih števil
double[] cena = new double[3];

//  tabela 4 znakov
char[] tabelaZnakov = new char[4];

//  tabela 500 nizov
string[] ime = new string[500];
```

INDEKSI

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).

Deklarirajmo enodimenzionalno tabelo 5 celih števil::

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

Grafično si lahko to tabelo predstavljamo takole:

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

Tabela 1: Grafična predstavitev tabele.

Oglejmo si enega od elementov te tabele in zapišimo:

```
tab[4] = 100; //peti element tabele tab, ker se indeksi začnejo z 0
```

vrednost tabelaričnega elementa

indeks

ime tabele

S posameznimi elementi tabele lahko operiramo enako kot s spremenljivkami tega tipa. Tako je v spodnjem zgledu npr. *tabela[4]* povsem običajna spremenljivka tipa *int*.

```
//  tabelo napolnimo z vrednostmi
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
tab[0] = 10;
tab[1] = 20;
tab[2] = 31;
tab[3] = 74;
tab[4] = 97;
tab[5] = 31;
```

Posameznim tabelaričnim spremenljivkam (komponentam) lahko prirejamo vrednost ali pa jih uporabljamo v izrazih, npr.:

```
int[] tab = new int[5];
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 480
```

Izgled tabele po izvedbi zgornjih stavkov je torej takle:

100	80	300	400	480

Tabela 2: Tabela celih števil.

Pri uporabi indeksov pa moramo seveda paziti, da ne zaidemo izven predpisanih meja.

Poglejmo še, kako bi prebrali vsebino neke tabele oz. izpisali njeno vsebino. Če želimo brati ali izpisati vrednosti elementov, ki so v tabeli, jih moramo obravnavati vsakega posebej. Omenili smo že, da do *i*-tega elementa (natančneje, do elementa z indeksom i) dostopamo z izrazom *imeTabele[i]*. Za branje oz. izpis vseh elementov tabele bomo zato uporabili zanko *for*. Dolžino poljubne že definirane tabele (oz. število elementov, ki jih potrebujemo v glavi zanke) pa dobimo s pomočjo metode *Length*.

```
podatkovniTip[] imeTabele = new podatkovniTip[n];   /*definicija tabele n
elementov*/
for (int i = 0; i < imeTabele.Length; i++)
    Console.Write(imeTabele[i]);  //izpis tabelaričnega elementa z indeksom i
```

Deklaracija tabele in branje podatkov

Deklarirajmo tabelo 5 celih števil, preberimo tabelarične podatke in tabelo zapišimo v obratnem vrstnem redu.

```
static void Main(string[] args)
{
     //  branje
    string beri;
```

```
    int[] x = new int[5];  //deklaracija tabele 5 celih števil
    for (int i = 0; i <= 4; i++)
    {
        Console.WriteLine("Vnesi število:");
        beri = Console.ReadLine();  //preberemo vhodni podatek
        /*vneseno vrednost z metodo int.Parse spremenimo v celo število in jo
        shranimo v tabelarično spremenljivko*/
        x[i] = int.Parse(beri);      }
    for (int i = 4; i >= 0; i--)  //tabelo izpišemo v obratnem vrstnem redu
        Console.WriteLine(x[i]);
    Console.ReadKey();
}
```

Najbolj uporabna lastnost tabel je ta, da jih zlahka uporabljamo v zankah. S spreminjanjem indeksov v zanki poskrbimo, da se operacije izvedejo nad vsemi elementi tabele.

Prirejanje začetne vrednosti elementom neke tabele imenujemo tudi inicializacija.

Tabeli 100 celih števil priredimo naključne začetne vrednosti med 0 in 10

Deklarirajmo tabelo 100 celih števil in jo inicializirajmo tako, da bodo imeli vsi elementi tabele vrednost naključnega celega števila med 0 in 10.

```
int[] tabela = new int[100];
Random naklj = new Random();
for (int i = 0; i < 100; i++)
{
    tabela[i] = naklj.Next(11);//naključne vrednosti med 0 in 10
}
```

Velikost tabele lahko določimo tudi med samim delovanjem programa. Dimenzijo tabele torej lahko določi uporabnik, glede na svoje potrebe:

```
Console.Write("Določi dimenzijo tabele: ");
// Dimenzijo tabele določi uporabnik med izvajanjem!!!
int dimenzija = int.Parse(Console.ReadLine());
int[] tabela = new int[dimenzija];
```

Sočasno z najavo lahko elementom določimo tudi začetne vrednosti. Te zapišemo med zavita oklepaja. Posamezne vrednosti so ločene z vejico.

```
/*deklaracija in inicializacija tabele 9 celih števil
    * elementi tabele dobijo vrednost od 1 do 9*/
int[] stevila = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Ob takem naštevanju vrednosti posameznih elementov pa ne moremo navesti dolžine tabele, saj jo prevajalnik izračuna sam. Paziti moramo, da vsi elementi ustrezajo izbranemu tipu.

Povzemimo vse načine deklaracije tabele:

1. način: Najprej najavimo ime tabele. Nato z operatorjem *new* dejansko ustvarimo tabelo.

```
int[] tab1;  /* Vemo, da je tab1 ime tabele celih števil, tabela tab1
             pa zaenkrat še ne obstaja*/
tab1 = new int[20];  //  tab1 kaže na tabelo 20 celih števil
```

Seveda ni nujno, da si ukaza sledita neposredno drug za drugim. Prav tako ni nujno, da za določanje velikosti uporabimo konstanto. Napišemo lahko poljubni izraz,

```
tab1 = new int[2 + 5]; // tab1 kaže na tabelo 7 celih števil
```

kjer lahko nastopajo tudi spremenljivke, ki imajo v tistem trenutku že prirejeno vrednost,

```
int vel = 10;
tab1 = new int[2 * vel];  //  tab1 predstavlja tabelo 20 celih števil
```

Elemente te tabele lahko preberemo tudi preko tipkovnice:

```
int[] tab = new int[int.Parse(Console.ReadLine())];
```

No, zgornjo "kolobocijo" lahko napišimo bolj razumljivo takole

```
string beri = Console.ReadLine();
int velTab = int.Parse(beri);
int[] tab = new int[velTab];
```

2. način: Oba zgornja koraka združimo v enega.

```
int[] tab1 = new int[2];  //  Vsi elementi so samodejno nastavljeni na 0
double[] tab2 = new double[5]; //Vsi elementi so samodejno nastavljeni na 0.0
```

Tako v 1. kot 2. načinu velja, da se, ko z *new* ustvarimo tabelo, vsi elementi samodejno nastavijo na začetno vrednost, ki jo določa tip tabele (npr. *int* – 0, *double* – 0.0, *bool* – *false*).

3. način: Tabelo najprej najavimo, z operatorjem *new* zasedemo pomnilniško lokacijo, nato pa elementom takoj določimo začetno vrednost.

```
int[] tab3 = new int[] { -7, 5, 65 };  //  Elementi so -7, 5 in 65
```

Lahko pa smo tudi "pridni" in velikost tabele povemo še sami.

```
int[] tab3 = new int[3] { -7, 5, 65 };  //  Elementi so -7, 5 in 65
```

4. način: Tabelo najprej najavimo, nato pa elementom določimo začetno vrednost. Velikosti tabele ni potrebno podati, saj jo prevajalnik izračuna sam.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
char[] tab4 = { 'a', 'b', 'c' };  //  Elementi so 'a', 'b', in 'c'
```

INDEKSI V NIZIH IN TABELAH

Izraz, s katerim dostopamo do *i*-tega elementa tabele (*tab[i]),* se v jeziku C# po videzu ujema z izrazom za dostopanje do *i*-tega znaka v nizu (*niz[i]*). Izraza pa se ne ujemata v uporabi. Pri tabelah uporabljamo izraz za pridobitev in spremembo elementa tabele, medtem ko ga pri nizih uporabljamo le za pridobitev znaka. V primeru, da izraz uporabimo za spremembo določenega znaka v nizu, nam prevajalnik vrne napako.

```
//  Deklaracija tabele in niza
int[] tab = { 1, 6, 40, 15 };
string niz = "Trubarjevo leto 2008.";

//  Pridobivanje
int el = tab[2];  //  Vrednost spremenljivke je 6.
char z = niz[2];  //  Vrednost spremenljivke je koda zanka 'u'.

//  Spreminjanje
tab[1] = 5;  //  Spremenjena tabela je [1, 5, 40, 15].


niz[18] = '9';//NAPAKA! Znaka v nizu ne moremo spremeniti na ta način.
```

Tabela imen

Poglejmo si preprost primer tabele imen. Napolnimo tabelo z imeni in izpišimo njeno dolžino.

```
static void Main(string[] args)
{
    //najava tabele imena tipa string[], tabelo obenem še inicializiramo
    string[] imena={"Jan", "Matej", "Ana", "Grega", "Sanja"};
    //Izpis dolžine tabele
    Console.WriteLine("Dolžina tabele je " + imena.Length + ".");
    /*Izpis elementa, ki se nahaja na sredini naše tabele.
      Izraz imena.length / 2  vrne 2, element, ki pa se nahaja na mestu
      imena[2] je Ana.*/
    Console.Write("Ime na sredini tabele je  ");
    Console.WriteLine(imena[imena.Length/2] + ".");
}
```

Podrobneje si poglejmo, kako indeksiramo podatke v tabeli *imena.*

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Prvi element tabele ima indeks *0* in vsebuje niz *"Jan"*, drugi ima indeks *1* in vsebuje niz *"Matej"*, tretji ima indeks *2* in vsebuje niz *"Ana"*, četrti ima indeks *3* in vsebuje niz *"Grega"* in peti ima indeks *4* in vsebuje niz *"Sanja"*. Tabelo *imena* si lahko predstavljamo takole:

Indeks	0	1	2	3	4
Vrednost	Jan	Matej	Ana	Grega	Sanja

Tabela 3: Indeksi v tabeli *imena*

NEUJEMANJE TIPOV PRI DEKLARACIJI

Kaj se bo zgodilo, če v programu napišemo naslednja stavka?

```
string[] stevila = { 1, 2, 3 };
Console.WriteLine("Dolžina tabele je " +  stevila.Length + ".");
```

Program se ne prevede. Deklarirali smo tabelo nizov, tej tabeli pa smo priredili celoštevilske vrednosti. Ker se tipa ne ujemata, bo prevajalnik javil:

Error 3 Cannot implicitly convert type 'int' to 'string'

Napako popravimo tako, da napišemo ustrezen (pravilen) tip.

```
int[] stevila = { 1, 2, 3 };
Console.WriteLine("Dolžina tabele je " +  stevila.Length + ".");
```

PRIMERJANJE TABEL

Oglejmo si še en primer pogoste napake. Denimo, da želimo primerjati dve tabeli. Zanima nas, če so vsi enakoležni elementi enaki. "Naivna" rešitev z *t1 == t2* nam ne vrne pričakovanega rezultata.

```
static void Main(string[] args)
{
     int[] t1 = new int[] { 1, 2, 3 };
     int[] t2 = new int[] { 1, 2, 3 };
     Console.WriteLine(t1 == t2);
}
```

Opis programa

Čeprav obe tabeli vsebujeta enake elemente, nam izpis (*false*) pove, da tabeli nista enaki. Zakaj? Spomnimo se, da *t1* in *t2* vsebujeta samo naslov, kjer se nahajata tabeli. Ker sta to dve različni tabeli (sicer z enakimi elementi, a vseeno gre za dve tabeli), zato tudi naslova nista enaka. In to nam pove primerjava *t1 == t2*.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Oglejmo si, kako bi pravilno primerjali dve tabeli.

```
static void Main(string[] args)
{
    int[] t1 = new int[] {1, 2, 3};
    int[] t2 = new int[] {1, 2, 3};
    bool je_enaka = true;
    if (t1.Length == t2.Length) //preverimo, če sta tabeli enako veliki
    {
        //z zanko for primerjamo elemente znotraj obeh tabel
        for(int i = 0; i < t1.Length; i++)
        {
            //če naletimo na vsaj en par neenakih števil, tabeli NISTA enaki
            if (t1[i] != t2[i])
               je_enaka = false;
        }
    }
    else je_enaka = false;
    if (je_enaka)
        Console.WriteLine("Tabeli sta enaki.");
    else Console.WriteLine("Tabeli nista enaki.");
    Console.ReadKey();
}
```

Dnevi v tednu

Deklarirajmo tabelo sedmih nizov in jo inicializirajmo tako, da bo vsebovala dneve v tednu. Tabelo nato še izpišimo, vsak element tabele v svojo vrsto.

```
static void Main(string[] args)
{
    string[] dneviVTednu = new string[7] { "Ponedeljek", "Torek", "Sreda",
"Četrtek", "Petek", "Sobota", "Nedelja" };
    for (int i = 0; i < 7; i++)
    {
        Console.WriteLine(dneviVTednu[i]);
    }
    Console.ReadKey();
}
```

Mrzli meseci

Preberimo povprečne temperature v vseh mesecih leta in potem izpišimo mrzle mesece, torej take, ki imajo temperaturo nižjo od povprečne.

```
static void Main(string[] args)
{
    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] = double.Parse(Console.ReadLine());
        letnoPovp = letnoPovp + meseci[i];
    }

    letnoPovp = Math.Round(letnoPovp / 12, 2);
    Console.WriteLine("Povprečna letna temperatura : " + letnoPovp);
    Console.WriteLine("Mrzli meseci: ");

    for (int i = 0; i < 12; i++)
    {
        if (meseci[i] <= letnoPovp) Console.Write((i + 1) + ". mesec  ");
    }
}
```

Delitev znakov v stavku

Preberimo poljuben stavek in izpišimo, koliko znakov vsebuje, koliko je v njem samoglasnikov, koliko števk in koliko ostalih znakov

```
static void Main(string[] args)
{
    string samogl = "AEIOU";

    int stSam = 0, stStevk = 0;
    string stavek;

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

    for (int i = 0; i < stavek.Length; i++)
    {
        char znak = stavek[i];
        if (samogl.IndexOf(znak) > -1)  //  znak je samoglasnik
            stSam = stSam + 1;
```

```
        if ('0' <= znak && znak <= '9')
            stStevk = stStevk + 1;
    }
    Console.WriteLine("\nŠtevilo vseh znakov v stavku : " + stavek.Length);
    Console.WriteLine("Število samoglasnikov        : " + stSam);
    Console.WriteLine("Število cifer                : " + stStevk);
    Console.WriteLine("Število ostalih znakov       : " + (stavek.Length -
stSam - stStevk));
}
```

Zbiramo sličice

Začeli smo zbirati sličice. V album moramo nalepiti 250 sličic. Zanima nas, koliko sličic bomo morali kupiti, da bomo napolnili album. Seveda ob nakupu ne vemo, katero sličico bomo dobili. Ker bi to radi vedeli, še preden se bomo zares spustili po nakupih, bi radi polnjenje albuma simulirali s pomočjo računalniškega programa. In če bomo to simulacijo ponovili dovoljkrat, bomo že dobili občutek o številu potrebnih nakupov. Pri tem seveda naredimo nekaj predpostavk:

- hkrati vedno kupimo le eno sličico,
- vse sličice so enako pogoste, torej je verjetnost, da bomo kupili i-to sličico ravno 1/250,
- podvojenih sličic ne menjamo.

Poglejmo, kako bi sestavili program.

Naš album bo tabela. Če bo vrednost ustreznega elementa *false*, to pomeni, da sličice še nimamo. Da nam ne bo po vsakem nakupu treba prečesati vsega albuma, da bi ugotovili, ali kakšna sličica še manjka, bomo vodili evidenco o tem, koliko sličic v albumu še manjka. V zanki, ki se bo odvijala toliko časa, dokler število manjkajočih sličic ne bo padlo na nič, bomo kupili sličico (naključno generirali število med 0 in 249). Če je še nimamo, bomo zmanjšali število manjkajočih sličic in si v albumu označili, da sličico imamo.

```
static void Main(string[] args)
{
    int st_slicicvalbumu = 250;
    int st_zapolnjenih = 0;
    int st_kupljenihslic = 0;
    Random bob = new Random();        //  boben naključnih stevil
    bool[] album = new bool[st_slicicvalbumu];        //  album
    //  polnimo album
    while (st_zapolnjenih < st_slicicvalbumu)
    {
        st_kupljenihslic = st_kupljenihslic + 1;
        int st_slikic = bob.Next(st_slicicvalbumu);
```

```
        //  jo damo v album
        if (!album[st_slikic])
        {
            album[st_slikic] = true;
            st_zapolnjenih = st_zapolnjenih + 1;
        }
    }
    Console.WriteLine("Da smo napolnili album, smo kupili " +
st_kupljenihslic + " slikic.");
}
```

Opis programa

V uvodu definiramo tri spremenljivke, s katerimi nadzorujemo stanje album. Za polnjenje albuma uporabimo zanko *while*. Ob vsakem vstopu v zanko kupimo sličico, ki ji dodelimo naključno mesto v albumu (naključno smo generirali število med 0 in 249). Preverimo, če že imamo to mesto zapolnjeno (*!album[st_slikic]*). Če ne, jo vstavimo v album in povečamo število zapolnjenih mest v albumu. To ponavljamo toliko časa, da je pogoj v zanki resničen. Na koncu še izpišemo, koliko sličic smo morali kupiti, da smo napolnili album.

Najdaljša beseda v nizu

Napišimo program, ki bo našel in izpisal najdaljšo besedo prebranega niza. Predpostavimo, da so besede ločene z enim presledkom. Da bomo iz niza izluščili besede, si bomo pomagali z metodo *Split*, ki jo najdemo v razredu *string*. Metoda vrne tabelo nizov, v kateri vsak element predstavlja podniz niza, nad katerim smo metodo uporabili. V jeziku C# niz razmejimo na besede z razmejitvenim znakom oziroma znaki, ki jih ločimo z vejico (npr. *Split(',')*). Posamezen razmejitveni znak pišemo v enojnih narekovajih (').

Denimo, da imamo stavek:

```
string stavek = "a/b/c.";
```

Če uporabimo metodo *Split* takole:

```
string[] tab = stavek.Split('/');
```

smo s tem ustvarili novo tabelo *tab* velikosti 3. V tabeli so shranjeni podnizi niza stavek, kot jih loči razmejitveni znak /. Prvi element tabele tab je niz "a", drugi element je niz "b" in tretji element niz "c.".

```
static void Main(string[] args)
{
    //  Vnos niza
    Console.Write("Vnesi niz: ");
    string niz = Console.ReadLine();
```

```
    //  Pretvorba niza v tabelo nizov
    string[] tab = niz.Split(' ');

    //  Iskanje najdaljše besede
    string pomozni = "";  //  Najdaljša beseda (oz. podniz)
    for (int i = 0; i < tab.Length; i++)
    {
        if (pomozni.Length < tab[i].Length)
        {
            pomozni = tab[i];
        }
    }
    //  Izpis najdaljše besede
    Console.WriteLine("Najdaljša beseda: " + pomozni);
}
```

Razlaga

Najprej določimo niz *niz*. Nato s pomočjo klica metode *Split()* določimo tabelo *tab*. V metodi za razmejitveni znak uporabimo presledek (' '). Če je med besedami niza niz več presledkov, se vsi presledki obravnavajo kot razmejitveni znaki. Nato določimo pomožni niz *pomozni*, ki predstavlja trenutno najdaljšo besedo. Z zanko se sprehodimo po tabeli *tab* in poiščemo najdaljšo besedo. Na koncu izpišemo najdaljšo besedo niza *niz* oziroma tabele *tab*.

Če za spremenljivko *niz* določimo prazen niz (v konzoli pri vnosu niza stisnemo le tipko *Enter*), se program izvede in za najdaljši niz izpiše prazen niz. Tabela *tab* je ostala prazna, zato je niz v spremenljivki *pomozni* ostal "".

Uredjanje elementov po velikosti

Sestavimo program, ki bo uredil vrednosti v tabeli celih števil po naslednjem postopku: poiščimo najmanjši element in ga zamenjajmo s prvim. Nato poiščimo najmanjši element od drugega dalje in ga zamenjajmo z drugim, itn. Tabelo velikosti n preberemo in jo po urejanju izpišemo na ekran.

```
public static void Main(string[] args)
{
    //Vnos velikosti tabele: vnese jo uporabnik
    Console.Write("Velikost tabele: ");
    int n = int.Parse(Console.ReadLine());

    // Deklaracija tabele velikosti n
    int[] tab = new int[n];


    Console.WriteLine();
    // Napolnitev tabele: podatke preberemo s tipkovnice
```

```
    for (int i = 0; i < n; i++)
    {
        Console.Write("Vnesi " + (i + 1) + ". element: ");
        tab[i] = int.Parse(Console.ReadLine());
    }

    //Z zanko for se sprehodimo skozi tabelo in jo uredimo
    for (int i = 0; i < n - 1; i++)
    {
        //Kandidat za najmanjši element od indeksa i  do konca tabele
        int min = tab[i];
        int indeks = i;  // Indeks najmanjšega kandidata
        for (int k = i + 1; k < n; k++)
        { //  Poiščimo najmanjši element v tem delu
            if (min > tab[k])
            {
                min = tab[k];
                indeks = k; //  Zapomnimo si indeks najmanjšega elementa
            }
        }
        //  Zamenjava elementov
        int t = tab[i];
        tab[i] = tab[indeks];
        tab[indeks] = t;
    }

    //  Izpis urejene tabele
    Console.WriteLine();
    for (int i = 0; i < n; i++)
    {
        Console.Write(tab[i] + " ");
    }
    //  Prehod v novo vrstico
    Console.WriteLine();
}
```

Manjkajoča števila

Generirajmo tabelo 50 naključnih števil med 0 in 100 (obe meji štejemo zraven). S pomočjo zanke *for* ugotovimo in izpišimo, katerih števil med 0 in 100 ni v tej tabeli. Za kontrolo naredimo izpis elementov tabele.

```
public static void Main(string[] args)
{
    const int vel = 50;  //  Velikost tabele
    const int mejaStevil = 100; //  Zgornja meja naključnih števil
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
    int[] tab = new int[vel];   //  Dekleracija tabele
    Random nak = new Random();  //  Generator naključnih števil
    //  Polnjenje tabele
    for (int i = 0; i < vel; i++)
    {
        tab[i] = nak.Next(mejaStevil + 1);  //  Določitev naključnega števila
    }
    Console.WriteLine("Izpis vsebine tabele naključnih stevil: ");
    //  Izpis elementov tabele
    for (int i = 0; i < tab.Length; i++)
    {
        int el = tab[i];
        Console.Write(el + " ");  //  Izpis elementa, ločenega s presledkom
    }
    Console.WriteLine("\n\nŠtevila med 0 in " + mejaStevil + ", ki niso v tej
tabeli, so: ");
    //  Iskanje števil, ki se ne nahajajo v tabeli
    for (int i = 0; i < mejaStevil + 1; i++)
    {
        bool nasli = false;  //  Spremenljivka za iskanje števil
        for (int j = 0; j < tab.Length; j++)
        {
            int el = tab[j];
            if (el == i)
            {
                nasli = true;  //  Število smo našli
                break;  //  Prekinemo iskanje števila
            }
        }
        //  Ali smo število našli, nam pove spremenljivka nasli
        if (!nasli) Console.Write(i + " ");
    }
    //  Nova vrstica
    Console.WriteLine();
}
```

Če pa si lahko "privoščimo" uporabo dodatne tabele take velikosti, kot je vseh možnih števil (v našem primeru 101), lahko razvijemo tudi boljši in (hitrejši) algoritem.

DVODIMENZIONALNE IN VEČDIMENZIONALNE TABELE

Tabele so lahko tudi dvo ali večdimenzionalne. 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[10,20]; //2-dim tabela 10 vrstic in 20 stolpcev
```

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

```
int[,] tabela = new int[10, 20];
```

- 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, samodejno inicializirajo (v našem primeru ima vseh 10 x 20 (to je 200) elementov tabele vrednost 0.

Dvodimenzionalna tabela naključnih celih števil

Deklarirajmo dvodimenzionalno tabelo 10 x 10 celih števil in jo inicializirajmo tako, da bodo po diagonali same enice, nad diagonalo same dvojke, pod diagonalo pa ničle. Napišimo še metodo za izpis te tabele.

```
static void Main(string[] args)
{
    //deklaracija 2-dim tabele celih števil: spremenljivke dobijo vrednost 0
    int[,] tabela = new int[10, 10];
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (i == j) //diagonalni elementi dobijo vrednost 1
                tabela[i, j] = 1;
            else if (i < j)//elementi nad diagonalo dobijo vrednost 2
                tabela[i, j] = 2;
        }
    }
    //klic metode za izpis tabele
    Izpis(tabela);
    Console.ReadKey();
}
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
//metoda, ki dobi za vhodni podatek dvodim.tabelo celih števil
public static void Izpis(int[,] tabela)
{
 for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
            Console.Write(tabela[i, j] + "  ");
        Console.WriteLine();
    }
}
```

Poštevanka

Kreirajmo dvodimenzionalno tabelo 10 x 10 celih števil. V posamezni vrstici naj bo poštevanka števil med 1 in 10. Tabelo na primeren način tudi izpišimo.

```
static void Main(string[] args)
{
    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(tabela[i, j] + "\t");
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

Največja vrednost v tabeli

Kreirajmo naključne elemente kvadratne matrike (dvodimenzionalna tabela) dimenzije 4 x 4 celih števil med 0 in 10 in nato ugotovimo in izpišimo največjo vrednost v matriki. Ugotovimo in izpišimo še, na katerih mestih v matriki se ta največji element pojavi (indeksi!).

```
static void Main(string[] args)
{
    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 največjega
            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("\nNajvečji element: " + max + "\n");
    Console.WriteLine("Indeksi največjega elementa: ");
    for (i = 0; i < 4; i++)  //izpis pozicij/indeksov največjega elementa
      for (j = 0; j < 4; j++)
        if (mat[i, j] == max) Console.WriteLine("[" + i + "," + j + "]",i,j);
    Console.ReadKey();
}
```

Obstajajo tudi tri in večdimenzionalne tabele. Njihova deklaracija in uporaba je podobna kot pri dvodimenzionalnih tabelah.

Splošna deklaracija tridimenzionalne tabele:

Podatkovni_tip[, ,] ime = new Podatkovni_tip[1.dimenz., 2.dimenz, 3.dimenz.];

Tudi večdimenzionalno tabelo lahko deklariramo na tri načine:

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

```
int[, ,] tab3D;  //deklaracija 3-dim tabelarične spremenljivke
tab3D = new int[10,20,30];
```

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

```
int[, ,] tab3D = new int[10, 20, 30]; //3-dim tabela
```

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

```
int[, ,] tab3D = new int[2, 3, 2] //3-dim tabela
{
    {
        { 1, 1 },
        { 2, 2 },
        { 3, 3 }
    },
    {
        { 0, 0 },
        { 1, 2 },
        { 3, 2 }
    },
};
```

Tridimenzionalna enotska matrika

Kreirajmo tridimenzionalno enotsko matriko: to je tabela, ki ima vse elemente enake 0, elementi, ki pa imajo vse tri indekse enake pa so enaki 1. Tabelo nato tudi izpišimo v primerni obliki!

```
static void Main(string[] args)
{
   int[, ,] tab3D = new int[3, 3, 3];//deklaracija tabele
    //ustvarjanje enotske matrike
   for (int i=0;i<3;i++)
       for (int j=0;j<3;j++)
           for (int k=0;k<3;k++)
               if ((i==j)&&(i==k))
                   tab3D[i,j,k]=1;
               else tab3D[i,j,k]=0;
    //Izpis tabele
   Console.WriteLine("3D enotska matrika\n");
   for (int i = 0; i < 3; i++)
   {
       for (int j = 0; j < 3; j++)
       {
           for (int k = 0; k < 3; k++)
               Console.Write(tab3D[i,j,k]+"  ");
           Console.WriteLine();
       }
       Console.WriteLine();
   }
    Console.ReadKey();
}
```

SKLAD IN KOPICA

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

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 metode. Na skladu se hranijo tudi parametri metod, ki se obnašajo tako kot spremenljivke, ki se tvorijo ob klicu metod. Pri klicu parametrov *po vrednosti*, se spremenljivka, ki nastopa kot parameter metode prepiše (prekopira) na sklad in v metodi delamo v bistvu z njeno kopijo. Ob klicih metod se torej sklad povečuje, ob zaključku izvajanja metod 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*) in *Vzemi s sklada* (*pop*).

Kopica pa je del pomnilnika, kamor se zlagajo t.i. **referenčne** spremenljivke (*reference type*). To so spremenljivke ki imajo poleg svoje vrednosti tudi referenco oz kazalec nanjo. To so npr. vse tabelarične spremenljivke, ali pa vsi iz razredov izpeljani objekti, o katerih bo govora v poglavju o razredih in objektih. 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 kopice lahko ponazorimo tudi pri metodah: pri klicu parametrov *po referenci* se spremenljivka, ki nastopa kot parameter metode prenese v metodo in v metodi dejansko delamo prav s to spremenljivko. Sam prenos je v tem primeru realiziran tako, da se tokrat na sklad prenese referenca na spremenljivko in metoda potem ve, da ima opravka z referenco, ne pa z vrednostjo.

POVZETEK

Bistvena prednost uporabe tabel je, da lahko z njihovo pomočjo na enostaven način obdelujemo množico spremenljivk istega tipa, ki so dostopne preko indeksov te tabele. Velikost tabele lahko določimo med samim delovanjem programa, vse tabelarične spremenljivke pa so si enakovredne ne glede na to, ali se nahajajo na začetku ali pa na koncu tabele. V primeru, da pa je število podatkov preveliko in da jih želimo po zaključku programa tudi shraniti za kasnejše obdelave, pa bomo podatke raje zapisali v datoteke. Slaba stran tabel pa je v tem, da moramo pred njeno deklaracijo vedeti njegovo dolžino, ko pa je tabela definirana, zaseda pomnilnik ne glede na to, ali elemente te tabele potrebujemo ali ne. Pogosto zato namesto tabel raje uporabimo zbirke.

VAJE

1. Ustvari naključno tabelo 100 števil tipa *double* z vrednostmi med 1 in 100 (zaokrožene na dve decimalki) in izpiši le tista števila, ki so večja od zadnjega elementa v tej tabeli.

2. Dana je enodimenzionalna tabela 100 znakov. Ugotovi, ali se v tej tabeli nahaja določen znak.

3. 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 ima 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 tako, da bo v vsaki vrstici po 10 števil.

4. Sestavite program, ki bo prebral podatke v tabelo celih števil, nato pa preveril, ali tabela predstavlja permutacijo števil od 1 do n, kjer je n dolžina tabele. Namig: Tabela dolžine n predstavlja permutacijo, če v njej vsako naravno število od 1 do n nastopa natanko enkrat.

5. Sestavi program, ki bo uporabniku zastavil 10 računov za množenje celih števil. Eden izmed faktorjev naj bo naključno dvomestno celo število, drugi izmed faktorjev pa mora biti naključno enomestno število. Števila in uporabnikove rezultate v obliki stringa (npr. 2;5;10) shranjuj v ustrezno tabelo . Tabelo NA KONCU obdelaj (uporabi metodo Split) in izpiši število pravilnih in število nepravilnih odgovorov.

6. Kreiraj dvodimenzionalno tabelo celih števil med 1 in 1000. Ugotovi in izpiši največji in najmanjši element te tabele, ter indeksa teh dveh elementov.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

7. Deklariraj enodimenzionalno tabelo realnih števil, ki predstavljajo znesek prodaje po posameznih mesecih. Napiši metodo za vnos podatkov tabelo in metodo, ki ugotovi in izpiše mesec v letu, ko je bila prodaja največja. Izpis naj bo v obliki:

   Največja prodaja je bila v mesecu *xxxxxxx* in je znašala *xxxx.xx* €.

8. Trije prijatelji so priredili tekmovanje v metu igralne kocke. Vsak od njih bo metal kocko v treh serijah, v vsaki seriji pa ima po 5 metov. Napiši program, ki bo posamezne mete kocke shranjeval v tridimenzionalno tabelo. Tabelo nato obdelaj in ugotovi, kateri od tekmovalcev je zmagal in kateri od njih je dosegel največ šestic!

9. Trgovina se je odločila, da bo svoje izdelke podražila. Napiši program, ki bo omogočil vnos števila izdelkov, ki jih želijo podražiti, odstotek podražitve, ter staro ceno izdelka. Staro ceno, odstotek podražitve, ter novo izračunano ceno shranjuj v dvodimenzionalno tabelo. Le-to na koncu tudi izpiši v obliki:

Stara cena	Odstotek podražitve	Nova cena
100.00	5 %	105.00
2000.00	10 %	2200.00

10. Kreiraj tabelo števil med 1 in 100. V vsaki vrstici te tabele naj bo najprej število, nato pa še njegov kvadrat, kub, kvadratni koren in tretji koren tega števila (na pet decimalk).

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 prvega do zadnjega elementa, iteracija v obratni smeri pa NI možna.

Splošna oblika *foreach* zanke:

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

Izpis stavka navpično

Poluben stavek s pomočjo zanke foreach izpišimo navično!

```
public static void Main(string[] args)
{
    string znaki = "Tale stavek bo izpisan navpično!";
    foreach (char znak in znaki)
        Console.WriteLine(znak);
    Console.ReadKey();
}
```

Tabela dni v tednu

Deklarirajmo tabelo 7 stringov in jo ob deklaraciji inicializirajmo tako, da bodo v njej dnevi v tednu. Napišimo 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"};

    //Metoda dobi za parameter tabelo:
    IzpisiTabelo(dneviVTednu);
}
```

Rezervirana beseda params

C# omogoča kreiranje metode, ki ji lahko za parameter posredujemo bodisi poljubno število elementov istega tipa, ali pa neko tabelo, ki jo potem v metodi obdelamo s pomočjo zanke *foreach*. To nam omogoča rezervirana beseda ***params***.

```
//v glavi metode uporabimo rezervirano besedo params
/*Metoda ObdelajTabelo lahko sprejme za parameter poljubno število elementov
istega tipa, lahko pa tudi tabelo*/
public static void ObdelajTabelo(params int[] poljubnaTabelaCelihStevil)
{
    int vsota = 0;
    foreach (int i in poljubnaTabelaCelihStevil)
    {
       vsota = vsota + i;
    }
}
//Glavni program
static void Main(string[] args)
{
    // Klic metode obdelajTabelo – metodi posredujemo poljubno število
    // parametrov*/
    ObdelajTabelo(7, 8, 9, 10);
    //deklaracija in inicilaizacija tabele 5 celih števil
    int[] tabelaCelihStevil = new int[5] { 1, 2, 3, 4, 5 };
    //klic metode obdelajTabelo – metodi posredujemo tudi tabelo celih števil
    ObdelajTabelo(tabelaCelihStevil);
    Console.ReadKey();
}
```

POVZETEK

Zanko *foreach* lahko pri običajnih iteracijah nadomestimo s klasičnimi zankami *for*, *while* ali *do while*. Nepogrešljiva ali pa vsaj veliko lažja za uporabo pa postane pri obdelovanju tabel in še posebej pri obdelovanju zbirk. Za razliko od običajnih zank, pa v zanki tipa *foreach* ne moremo spreminjati vrednosti elementov tabele ali zbirke.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

VAJE

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

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

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

4. Ustvari naključno tabelo 100 celih števil z vrednostmi med 50 in 100 in nato v zanki *foreach* 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).

5. 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. (Izpis: AAAVVVTTTOOO).

6. Sestavi program, ki prebere celo število. V zanki *foreach* izpiši vsoto vseh števil od 1 do prebranega števila. Izpiši 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

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

8. 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 in zanko *foreach*.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

9. Preberi poljuben stavek. S pomočjo metode *Split* in zanke *foreach* izpiši vse besede tega stavka, vsako besedo v novo vrsto.

10. Kreiraj je dvodimenzionalno tabela 10 x 10 celih števil. Metodo napolni s števili tako, da bodo nad diagonalo same ničle, pod diagonalo same enice na diagonali pa naključna decimalna števila med -5 in + 5 (tri decimalna mesta)! Koliko enic je v tabeli?

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[0] = (int)Ime_zbirke[0] + 100; //OK!!!!!
```

```
Ime_zbirke[0] = Convert.ToInt32(Ime_zbirke[0]) + 100; //OK
Ime_zbirke[0] = (int)Ime_zbirke[0] + 100;             //OK
Ime_zbirke[1] = Ime_zbirke[0];                        //OK
Ime_zbirke[1] = "Danes je lep dan! ";                 //OK
```

Ker elementi netipizirane zbirke nimajo tipa, moramo pred njihovo uporabo (npr. pri računanju) narediti eksplicitno konverzijo (pretvorbo) v ustrezen tip.

```
ArrayList stevila = new ArrayList();
// V zbirko dodajmo nekaj celih števil
stevila.Add(3);
stevila.Add(7);
stevila.Add(5);
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;
}
//Lahko uporabimo tudi foreach zanko:
foreach (int x in stevila)
{
    vsota += x;
}
```

Delo z zbirkami je enostavno, saj si lahko pomagamo z že obstoječimi metodami in lastnostmi. V naslednji tabeli so le najpomembnejše:

Lastnost	Razlaga
AddRange	Dodajanje več elementov hkrati na konec zbirke.
Capacity	Nastavitev števila elementov ki jih lahko zbirka hrani.
Count	Pridobivanje števila elementov v zbirki.
Clear	Brisanje celotne vsebine zbirke.
Contains	Ugotavljanje, ali je nek element v zbirki.
IndexOf	Iskanje določenega elementa v zbirki: metoda vrne indeks tega elementa znotraj zbirke.
Insert	Vstavljanje novega elementa v zbirko na točno določeno pozicijo.
Remove	Brisanje prve pojavitve določenega elementa iz zbirke.
RemoveAt	Brisanje elementa z določenim indeksom iz zbirke.
Sort	Urejaje zbirke, a vsi elementi morajo biti istega tipa

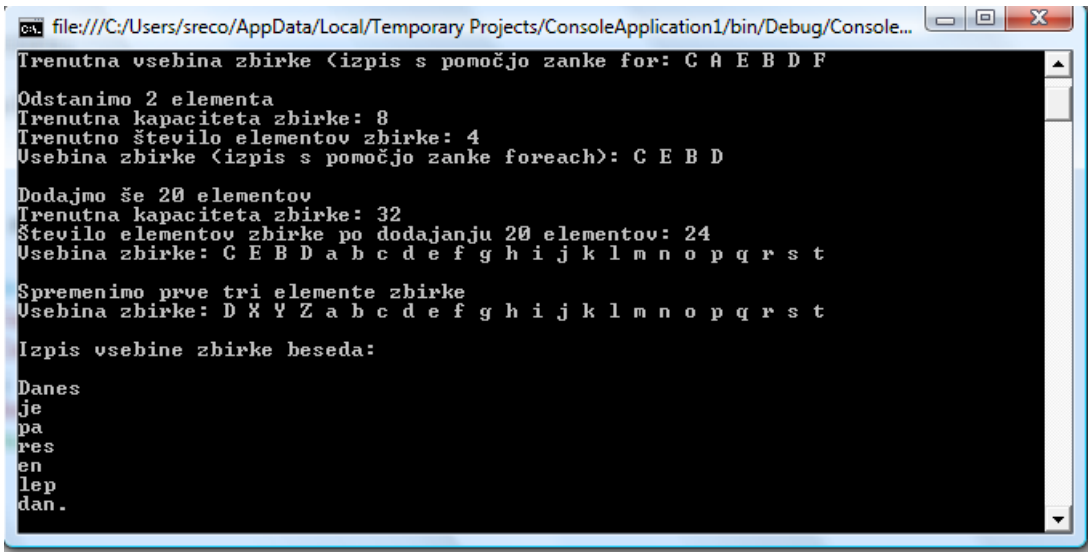
Tabela 4 : Lastnosti in metode netipiziranih zbirk.

Tule je nekaj primerov uporabe metod za delo z netipizirano zbirko:

```
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) // Izpis vsebine zbirke
    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: " + al.Count);
Console.Write("Vsebina zbirke: ");
foreach (char c in al) // Izpis vsebine zbirke
    Console.Write(c + " ");
Console.WriteLine("\n");
// Vsebino zbirke spremenimo tabelaričnega zapisa elementov zbirke.
Console.WriteLine("Spremenimo prve tri elemente zbirke");
al[0] = 'X';
al[1] = 'Y';
al[2] = 'Z';
al.Sort();//urejanje zbirke
Console.Write("Vsebina zbirke: ");
foreach (char c in al) // Izpis vsebine zbirke po urejanju
    Console.Write(c + " ");
ArrayList besede = new ArrayList(); //deklaracija nove zbirke z imenom besede
string stavek = "Danes je pa res en lep dan.";
/*metoda Split vse znake med dvema presledkoma shrani v ustrezen element
zbirke besede*/
```

```
besede.AddRange(stavek.Split(' '));
Console.WriteLine("\n\nIzpis vsebine zbirke beseda: \n");
//izpis posameznih besed, vsaka beseda v novi vrstici
foreach (string beseda in besede)
    Console.WriteLine(beseda);
```

Če bi zgornje stavke zapisali v nek program, bi bil izpis takle:



Slika 7 : Izpis programa, ki dela z zbirkami.

Zbirka decimalnih števil

Kreirajmo zbirko 1000 naključnih realnih števil med 0 in 1000 z dvema decimalkama. Napišimo še metodo, ki ugotovi in vrne največje število v zbirki in metodo, ki sešteje in vrne samo decimalke vseh števil v zbirki*/

```
//Metoda, 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);
}

//metoda, 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 (metoda 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 metode, ki poišče največje število v zbirki
    najvecje(stevila);
    //klic metode, ki vrne vsoto decimalk zbirke
    Console.WriteLine("Vsota vseh decimalk zbirke je " + vsotadec(stevila));
    Console.ReadKey();
}
```

TIPIZIRANE ZBIRKE

Tipizirane zbirke vsebujejo elemente istega podatkovnega tipa, pred deklaracijo pa moramo poskrbeti za vključitev ustreznega imenskega prostora (ta je v C# že privzet).

```
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*).

Tipiziranih zbirk je več vrst (*List*, *SortedList*, *Queue* in *Stack*), izmed katerih pa si bomo ogledali in naredili nekaj primerov le za tipizirane zbirke vrste *List*.

Tipizirana zbirke vrste *List* 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. Tule je nekaj primerov deklaracij takih zbirk:

```
//zbirka stringov
```

```
List<string> naslovi = new List<string>();

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

//Zbirka treh stringov.
List<string> kratice = new List<string>(3);
/*V zbirko kartice pa lahko seveda damo poljubno število stringov, a vsaka
prekoračitev začetne dimenzije podvoji kapaciteto seznama!*/
```

Zbirka celih števil

Ustvarimo tipizirano zbirko celih števil in vanjo dodajmo 100 sodih celih števil med 10 in 100! Izračunajmo vsote teh števil.

```
static void Main(string[] args)
{
    List<int> stevila = new List<int>();//deklaracija zbirke celih števil

    Random naklj=new Random();
    int vsota = 0;
    for (int i = 0; i < 100; i++)
        stevila.Add(naklj.Next(5, 51) * 2);
    //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;
    }
    Console.WriteLine("Vsota elementov zbirke: " + vsota);
    Console.ReadKey();
}
```

V naslednji tabeli so le najpomembnejše lastnosti in metode zbirke *List*:

Indeks	Razlaga
[indeks]	Dostop do posameznega elementa zbirke. Indeks prvega elementa zbirke 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.

Tabela 5: Lastnosti in metode zbirke *List*.

Tipizirana zbirka besed

Ustvarimo tipizirano zbirko stringov, jo napolnimo z nekaj besedami in nati izpišimo njeno vsebino!

```
static void Main(string[] args)
{
    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("Conway");//kapaciteta se podvoji na 12 elementov
    //več elementov hkrati lahko dodamo s pomočjo metode AddRange
    priimki. AddRange(new string[] { "Best", " Socrates" });
    priimki.Sort();//urejanje seznama

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

Tipizirana zbirka decimalnih števil

Vsebino tabele decimalnih števil prepišimo v tipizirano zbirko. Na primeru te zbirke nato demonstrirajmo prirejanje oz. spreminjanje vrednosti elementom zbirke in uporabo metod *Add*, , Insert , *RemoveAT* in *Contains*..

```
static void Main(string[] args)
{
    //definicija tabele decimalnih števil
    decimal[] novecene = { 3275.68m, 4378.12m, 54123.34m, 100.00m };
    //deklaracija zbirke decimalnih števil
    List<decimal> cene = new List<decimal>();
    //v zbirko 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 zbirke
    cena1 = cene[0];  //cena1 dobi novo vrednost 2345.11
    decimal cena2 = cene[1];  //cena2 dobi vrednost 3275.68
    //odstranimo DRUGI element iz zbirke (prvi element ima indeks 0!).
    cene.RemoveAt(1);
    cena2 = cene[1];  ////cena2 dobi vrednost 4378.12
    decimal x = 100.00m;
    //če v zbirki najdemo znesek 100.00, ga odstranimo iz zbirke
    if (cene.Contains(x))
    {
        cene.Remove(x);//element x odstranimo iz zbirke
    }
    Console.ReadKey();
}
```

POVZETEK

Zaradi omejitve prostora smo v tem razdelku prikazali le osnove zbirk. Pri delu s podatki smo velikokrat v dilemi, ali naj uporabimo tabele ali zbirke. Uporaba zbirke ima kar nekaj prednosti: dimenzije nam ni potrebno določati vnaprej, kadarkoli lahko nek element odstranimo iz zbirke na zelo enostaven način, nov element lahko vstavimo v zbirko na poljubno pozicijo, zelo enostavno pa je tudi urejanje zbirke. Za začetnika pa je vsekakor delo s tabelami enostavnejše.

vo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

VAJE

1. 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.

2. Preberi 10 besed in jih shrani v netipizirano zbirko. Ugotovi in izpiši najdaljšo besedo v tej zbirki.

3. Napiši program, ki bo za redne prihodke (12 plač, božičnico, regres) seštel in izračunal povprečni mesečni prihodek ter vse prihodke v letu. Podatke shranjuj v zbirko.

4. V zbirko zapiši naključna cela števila med 1 in 100. Prepiši jih v novi zbirki tako, da bodo v eni zbirki soda števila in v drugi liha. Obe zbirki na koncu izpiši.

5. Napiši metodo, 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 tipizirano zbirko, ki jo nato obdelaj!

6. Napiši program, ki bo na podlagi naključnih števil med -15 in 35 (temperature) izračunal povprečno število (oz. temperaturo) zadnjih nekaj let. Generiraj vsaj 365 števil ali njegov večkratnik. Uporabi tipizirano zbirko celih števil.

7. Bodoča mamica in očka ne vesta, kako bi dala ime svojemu otroku. Priskoči jima na pomoč s programom, ki naj ima v obliki zbirke že shranjenih 10 ženskih in 10 moških imen. Program naj izvede 100000-krat žrebanje tako ženskega kot moškega imena in izpiše število izžrebanj vsakega imena ter katero žensko in moško je bilo največkrat izžrebano in kolikokrat.

8. Dana je (neurejena) zbirka naslovov filmov. Vsak film je treba oceniti z oceno od 1 do 10. Napiši metodo, ki za vse elemente dane zbirke filmov generira naključno oceno in izpiše seznam filmov in oceno posameznega filma.

9. Učitelj bo spraševal za oceno in je učencem naročil, naj se sami dogovorijo za vrstni red spraševanja. Učenci se niso uspeli uskladiti, zato za učitelja napiši program, ki bo prebral imena učencev in jih shranil v zbirko, nato pa s pomočjo žreba napisal vrstni red spraševanja (vsak učenec pride samo enkrat na vrsto).

10. Napiši program, ki zgenerira tisoč naključnih celih števil in jih shrani v zbirko. Nato naj zamenja števila na 1. in 2. mestu, 3. in 4., itd.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

STRUKTURE

UVOD

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 običajno 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* (pravimo jim tudi *komponente* oz. *polja*) strukture. Besedica **public** naj nam zaenkrat pomeni, da so člani te strukture javni in imamo do njih neomejen dostop. Strukturo moramo napisati (deklarirati) izven glavnega programa (izven metode *Main*).

Iz tako deklarirane strukture tvorimo spremenljivke na enak način kot vse spremenljivke vrednostnega tipa. Do elementov strukture pa dostopamo s pomočjo operatorja pika ('.'). Nad spremenljivkami izpeljanimi iz neke strukture lahko izvajamo vse operacije, ki jih poznamo nad osnovnimi podatkovnimi tipi. Pomembno je le, da strukturo deklariramo pred glavnim programom.

Struktura *obcan*

Ustvarimo strukturo *obcan*, s katero bomo opisali nekega občana. Vsebuje naj štiri polja: ime, priimek, EMŠO in davčno številko. V glavnem programu nato ustvari nekaj primerkov te strukture, za eno od spremenljivk preberi podatke in le-te nato izpiši .

```
struct obcan   //deklaracija MORA biti pred glavnim programom
{
    public string ime;
    public string priimek;
    public string EMSO;
    public long davcna;
};
static void Main(string[] args)
{
    obcan o; //deklaracija spremenljivke o tipa obcan
    obcan o1, o2, o3; //še tri spremenljivke tipa obcan

    //POZOR: elementi strukture obcan še NISO inicializirani
    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);
}
```

Struktura ulomek

Deklarirajmo strukturo ulomek z dvema komponentama: števec in imenovalec ulomka. Napišimo metodo za vnos podatkov za dva ulomka. Ulomka nato seštejmo in rezultat zapišimo v tretji ulomek, ki ga nato v primerni obliki tudi izpišimo. Napišimo tudi metodo za krajšanje ulomka.

```
struct ulomek
{
    public int stevec;
    public int imenovalec;
};
//metoda za vnos števca in imenovalca ulomka
static void Vnos(out ulomek u) //out pomeni klic po referenci
{
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
    Console.WriteLine("Vnesi števec in imenovalec ulomka");
    Console.Write("Števec: ");
    u.stevec = Convert.ToInt32(Console.ReadLine());
    //imenovalec mora biti različen od 0
    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);
}
//Metoda za krajšanje ulomka.
static void Krajsaj(ref ulomek u)//ref->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
    Console.WriteLine("Prvi ulomek: \n");
    Vnos(out u1); //klic metode za vnos števca in menovalca ulomka u1
    //out, ker ulomka še nista inicializirana)
    Console.WriteLine("\nDrugi ulomek: \n");
    Vnos(out u2); //klic metode za vnos števca in menovalca ulomka u2
    //out, ker ulomka še nista inicializirana)
    ulomek u3;
    u3.imenovalec = u1.imenovalec * u2.imenovalec;//seštejmo oba ulomka
    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: klic metode za krajšanje ulomka
    //parameter klican po referenci ref, ker je u3 ŽE inicializiran
    Krajsaj(ref u3);
```

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

Predstavnika poljubne strukture lahko ustvarimo tudi na kopici, torej z uporabo operatorja *new*. V tem primeru so komponente že inicializirane (člani tipov *int*, *double*, *float* imajo vrednost 0, znaki in stringi pa so prazni).

```
//ustvarimo novega občana na kopici:
obcan obc = new obcan();//komponente občana obc so ŽE inicializirane.
ulomek u1=new ulomek; //ulomek u1 ustvarimo na kopici
```

Struktura kontinent

Deklarirajmo strukturo kontinent z dvema poljema (*naziv* in *stPrebivalcev*). Iz strukture nato izpeljimo vrednostno spremenljivko *k1*, drug primerek strukture *k2* pa ustvarimo na kopici.

```
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
    //k2 je referenčna spremenljivka (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());
}
```

Keglji

Napišimo program za vodenje evidence podrtih kegljev za dva tekmovalca. Kreirajmo strukturo tekmovalec, ki ima dve komponenti: naziv tekmovalca in tabelo v katero bomo vpisovali posamezne mete. Imeni tekmovalcev in posamezne mete bomo prebrali(npr po 5 metov). Na

koncu ugotovimo zmagovalca in napišimo metodo, ki vrne povprečno število podrtih kegljev izbranega tekmovalca.

```
struct tekmovalec
{
    public String naziv;
    public int[] keglji;
}
//metoda, i vrne povprečno število metov posameznega tekmovalca
static double Povp(tekmovalec T, int N)
{
    int suma = 0;
    for (int i = 0; i < N; i++) //seštejemo mete tekmovalca
        suma += T.keglji[i];
    return (Math.Round((double)suma / N, 2));
}

static void Main(string[] args)
{
    tekmovalec A, B;  //spremenljivki tipa struktura
    int N = 5;//število metov posameznega tekmovalca
    //ustvarimo tabelo podatkov o podrtih kegljih prvega tekmovalca
    A.keglji = new int[N];
    B.keglji = new int[N]; //in še tabelo za 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("----------------------------------");
    //Izpis nazivov levo poravnamo
    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(22, 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));
    Console.ReadKey();
}
```

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.

Tabela tekočih računov

Deklarirajmo strukturo *tekoci*, s tremi komponentami, ki naj predstavlja tekoči račun očana, nato pa tabelo 10 takih struktur. Podatke tabele preberimo preko tipkovnice.

```
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());
    }
}
```

Tabela slaščic

Deklarirajmo strukturo *slaščica* z dvema komponentama: ime slaščice in cena. Napišimo zanko za vnos podatkov v tabelo *N* takih struktur. Napišimo še metodo za iskanje in izpis najdražje slaščice v tej tabeli.

```
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)
{
    //predpostavimo, da je najdražja slaščica tista, ki ima indeks 0
    int naj = 0;
    for (int i = 1; i < slascicarna.Length; i++)
    {
        if (slascicarna[i].cena > slascicarna[naj].cena)
            naj = i;
    }
    Console.WriteLine("\nNajdrazja slaščica v tabeli je " +
slascicarna[naj].ime + ",  njena cena pa je " + slascicarna[naj].cena);
}

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());
    }
    //klic metode, ki poišče in izpiše ime najdražje slaščice v tabeli
```

```
    Najdrazja(slascicarna);
}
```

Tabela točk

Kreirajmo strukturo *tocka2D* z dvema komponentama - koordinatama tipa int. Napišimo metode za vnos podatkov, izračun razdalje posamezne točke od koordinatnega izhodišča in metodo za izračun razdalje med točkama!

```
struct tocka2D
{
    public int x, y;
};
static void Vnos(out tocka2D T) //klic po referenci
{
    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; //'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]))
            //če najdemo bolj oddaljeno točko, si zapomnimo njen indeks
            naj = i;
    }
    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); //out ker ker struktura A še ni inicializirana
    Console.WriteLine("Vnos koordinat tocke B");
    Vnos(out B);
    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!");

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

GNEZDENE STRUKTURE

Strukture so lahko tudi gnezdene (struktura v strukturi). Najprej deklariramo strukturo, ki bo vgnezdena v drugi strukturi, nato pa še samo strukturo. Tudi do članov gnezdene 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>;
    <ime gnezdene struktura> <ime člana>//gnezdena struktura
    // ….
```

```
};
```

Telefonski imenik

Kreirajmo 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 = 10; //število oseb v imeniku

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

/*metoda 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
    {
        //klic metode, ki ima za parameter strukturo(klic po referenci)
        vnos(out Imenik[i]);
    }
    Obdelaj(Imenik, 4);//metoda izpiše, koliko oseb je iz omr.skup. 4
}
```

STRUKTURA DateTime

Za delo z datumi in časi C# uporablja strukturo *DateTime*. Spremenljivke tega tipa hranijo datum in čas obenem. Podatki o letu, mesecu in dnevu so obvezni, podatki o urah, minutah, sekundah in milisekundah pa so le opcijski in torej niso obvezni. Če podatkov o času pri inicializaciji ne navedemo, bo nastavljen privzeti čas, to pa je 12.00 AM (AM = dopoldan).

Deklaracija spremenljivke tipa *DateTime*:

```
DateTime danes; //deklaracija spremenljivke danes, ki je tipa DateTime
```

Spremenljivko tipa *DateTime* lahko inicializiramo na več načinov. Pokažimo nekaj primerov inicializacije:

- Uporaba lastnosti *now* in *Today*

```
//deklaracija spremenljivke danes, ki je tipa DateTime
DateTime danes;

/*v spremenljivko danes bo shranimo trenuten datum in čas s
pomočjo lastnosti now*/
danes = DateTime.Now; //danes dobi današnji datum in trenutni čas
DateTime Datum1 = DateTime.Today; //trenutni datum, brez ure
```

- Inicializacija s pomočjo metode *Convert.ToDateTime*

```
//2. način: v spremenljivko rojstniDan shranimo datum 01.03.2009
DateTime rojstniDan = Convert.ToDateTime("01.3.1992");//Nastavili
smo le datum
```

Pri uporabi metode te metode mora biti seveda string v oklepaju v veljavnem formatu. Veljavni formati za nastavitev datuma so naslednji: 20.08.2010, 20/08/210, 20/08/10, 20-08-2010, 2-7-2010, 2010-01-01, Jan 20 2010, Marec 15, 2010.

```
//Nastavitev ure

DateTime DatumInUra = Convert.ToDateTime("11.04.1999 13:47:44");
```

Veljavni formati za nastavitev časa pa so: 2:15 PM (AM = dopoldan, PM = popoldan), 14:15 ali pa 02:15:30!

- Inicializacija s statično metodo *Parse*

Splošna oblika je takale:

```
DateTime datum=DateTime.Parse(string);
```

Tudi pri uporabi metode *Parse* mora biti seveda *string*, ki je v v oklepaju v veljavnem formatu.

```
DateTime zacetniDatum = DateTime.Parse("20/08/07");

DateTime zacetniDatumInCas=DateTime.Parse("Aug 20,2007,2:15 PM");
```

Za formatiranje datuma in časa uporabljamo metodo **Format** razreda **String**. Metoda deluje podobno kot pri formatiranju števil, pri čemer pa moramo seveda uporabiti standardne ali pa običajne šifre za objekte tipa **DateTime**.

Tabela standardnih formatov za zapis objektov tipa *DateTime*:

Format	Opis
d	Kratka oblika datuma.
D	Dolga oblika datuma.
t	Kratka oblika zapisa časa.
T	Dolga oblika zapisa časa.
f	Dolga oblika zapisa datuma, kratka oblika zapisa časa.
F	Dolga oblika zapisa datuma, dolga oblika zapisa časa.
g	Kratka oblika zapisa datuma, kratka oblika zapisa časa.
G	Kratka oblika zapisa datuma, dolga oblika zapisa časa.
d	Dan v mesecu brez vodilnih ničel.
dd	Dan v mesecu z vodilnimi ničlami.
ddd	Skrajšano ime dneva.
dddd	Polno ime dneva.
M	Mesec brez vodilne ničle.

MM	Mesec z vodilno ničlo .
MMM	Skrajšano ime meseca.
MMMM	Polno ime meseca.
y	Leto zapisano z dvema ciframa brez vodilne ničle .
yy	Leto zapisano z dvema ciframa z vodilni ničlo.
yyyy	Leto zapisano z vsemi štirimi ciframi.
/	Datumsko ločilo.
h	Ure brez vodilne ničle.
hh	Ure z vodilno ničlo.
H	Ura pri 24 urni uri brez vodilnih ničel.
HH	Ura pri 24 urni uri z vodilnimi ničlami.
m	Minute brez vodilnih ničel.
mm	Minute z vodilnimi ničlami.
s	Sekunde brez vodilnih ničel.
ss	Sekunde z vodilnimi ničlami.
f	Deli sekunde (en znak f za vsako decimalno mesto).
t	Prvi karakter za del dneva (dopoldan/popoldan).
tt	Popolni označevalec dela dneva (dopoldan/popoldan).
:	Časovno ločilo.

Tabela 6: Tabela standardnih formatov za zapis objektov tipa *DateTime*.

Nastavitev datuma

Spremenljivki tipa *DateTime* nastavimo trenutni datum in čas.

```
DateTime datum = DateTime.Now;

string danasnji = datum.ToString("d");
```

```
string danasnji1 = datum.ToString("D");
string danasnji2 = datum.ToString("t");
string danasnji3 = datum.ToString("T");
string danasnji4 = datum.ToString("ddd,MMM d,yyyy");
string danasnji5 = datum.ToString("M/d/yy");//Rezutat: 8.20.09
string danasnji6 = datum.ToString("HH/mm/ss");
```

Struktura *DateTime* vsebuje tudi cel kup metod, ki nam olajšajo delo z datumi in časi. Tule je tabela nekaterih lastnosti in metod strukture *DateTime*:

Lastnost	**R a z l a g a**
Date	Vrne vrednost *DateTime* (trenutni datum), čas pa ima privzeto vrednost (12:00:00 AM).
Month	Vrne celo število, ki predstavlja mesec trenutne vrednosti *DateTime*.
Day	Vrne celo število, ki predstavlja dan trenutne vrednosti *DateTime*.
Year	Vrne celo število, ki predstavlja leto trenutne vrednosti *DateTime*.
Hour	Vrne celo število, ki predstavlja uro trenutne vrednosti *DateTime*.
Minute	Vrne celo število, ki predstavlja minute trenutne vrednosti *DateTime*.
Second	Vrne celo število, ki predstavlja sekunda trenutne vrednosti *DateTime*.
TimeOfDay	Vrne vrednost tipa *TimeSpan*, ki predstavlja nek časovni interval.
DayOfWeek	Vrne člana naštevnega tipa *DayOfWeek*, ki predstavlja dan v tednu trenutne vrednosti *DateTime*.
DayOfYear	Vrne celo število, za zaporedno število dneva v tekočem letu glede na trenutno vrednost v *DateTime*.
Metoda	**R a z l a g a**
DaysInMonth(leto, mesec)	Vrne število dni v navedenem letu in mesecu.
IsLeapYear(leto)	Vrne logično vrednost true, če je leto prestopno leto.

Tabela 7: Tabela nekaterih lastnosti in metod strukture *DateTime*.

Nekaj primerov uporabe:

```
//Nastavitev trenutnega datuma in časa
DateTime datum = DateTime.Now;

int mesec = datum.Month;  //trenutni mesec
int ura = datum.Hour;  //ura
```

```
int danVLetu = datum.DayOfYear;//zaporedni dan v letu
int dniVMesecu = DateTime.DaysInMonth(2007, 8);//mesec
bool prestopnoLeto = DateTime.IsLeapYear(2007);//False

//še primer uporabe lastnosti DayOfWeek
DayOfWeek danVTednu = datum.DayOfWeek;
string sporočilo="";
if (danVTednu==DayOfWeek.Saturday||danVTednu==DayOfWeek.Sunday)
   sporočilo="Vikend";
else
sporočilo="Delovni dan";
```

Še tabela operacij za delo z datumi in časom:

Metoda	Razlaga
AddDays(dni)	Dodajanje navedenega števila dni k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
AddMonths(meseci)	Dodajanje navedenega števila mesecev k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
AddYears(leta)	Dodajanje navedenega števila let k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
AddHours(ure)	Dodajanje navedenega števila ur k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
AddMinutes(minute)	Dodajanje navedenega števila minut k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
AddSeconds(sekunde)	Dodajanje navedenega števila sekund k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
Add(timespan)	Dodajanje navedene vrednosti tipa *TimeSpan* k vrednosti objekta *DateTime* in vračanje nove vrednosti *DateTime*.
Subtract(datetime)	Odštevanje navedene vrednosti *DateTime* od vrednosti *DateTime* in vračanje nove vrednosti tipa *TimeSpan*.

Tabela 8: Tabela operacij za delo z datumi in časom.

Pri delu z objekti tipa *DateTime* lahko uporabljamo tudi operatorje +, -, = =, !=, <, <=, > in >=. Za izpisovanje (in tudi formatiranje) spremenljivk tipa *DateTime* pa je na voljo cel kup metod, nekatere izmed njih so prikazane v naslednjih primerih:

```
DateTime danes = Convert.ToDateTime("9.9.2010 13:55:44.345");
Console.WriteLine(danes.ToLongDateString());//Izpis 9. september 2010
Console.WriteLine(danes.ToLongTimeString());//Izpis 13:55:44
```

```
Console.WriteLine("Še milisekunde: " + danes.Millisecond); //Izpis
Še milisekunde: 453

//Glede na svoj rojstni datum, ugotovimo in izpišimo kateri dan v
tednu je bil to
DateTime mojRojstniDan = Convert.ToDateTime("29.09.2009");
DayOfWeek danVtednu = mojRojstniDan.DayOfWeek;
Console.WriteLine(danVtednu.ToString());

DateTime datum = DateTime.Now;              //21.08.2010, 10:50:50 AM
DateTime novidatum = datum.AddMonths(2);    //21.10.2010
DateTime novidatum1 = datum.AddDays(60);    //21.10.2010
DateTime novidatum2 = datum.AddMinutes(30); //21.08.2010, 11:20:50 AM
DateTime novidatum3 = datum.AddHours(12);   //21.08.2010, 22:20:50 AM

DateTime danasnjidatum = DateTime.Today;//21.08.2010
DateTime datumzapadlosti = DateTime.Parse("06/09/2010");//06.09.2010
//Za odštevanje dveh datumov lahko uporabimo tudi operator - (minus)
DateTime danes = DateTime.Today;
DateTime obletnica = DateTime.Parse("25/10/2007");
//datume lahko med seboj tudi primerjamo
bool pretecen = false;
DateTime danes = DateTime.Today;
DateTime obletnica = DateTime.Parse("25/10/2007");
if (danes > obletnica)
   pretecen = true;
```

STRUKTURA TimeSpan

TimeSpan je struktura, ki hrani pretečeni čas, izražen v dnevih, urah, minutah, sekundah in milisekundah, ki ga dobimo kot razliko dveh spremenljivk tipa *DateTime*.

```
DateTime prviDatum = Convert.ToDateTime("27.09.2010 01:00:00");
DateTime drugidatum = Convert.ToDateTime("20.01.2008 07:00:00");
//deklaracija in inicializacija spremenljivke cas, ki bo hranila
časovno razliko med dvema //datumoma.
TimeSpan cas = prviDatum - drugidatum;
Console.WriteLine(cas.ToString());//980.18:00:00

Console.WriteLine("          Dni: " + cas.Days);         //980
Console.WriteLine("           Ur: " + cas.Hours);        //18
Console.WriteLine("        Minut: " + cas.Minutes);      //0
Console.WriteLine("      Sekund: " + cas.Seconds);       //0
Console.WriteLine("  Milisekund: " + cas.Milliseconds);//0
```

Krožek

Deklarirajmo strukturo *dijak*, nato pa strukturo *krozek*, v kateri je vgnezdena tabela struktur tipa *dijak*. Preberimo podatke, nato pa izračunajmo povprečno starost vseh dijakov!

```
struct dijak
{
    public string ime;
    public string priimek;
    public DateTime rojstvo;
};

struct krozek
{
    public int Id;
    public string naziv;
    public dijak[] Dijaki;//tabela dijakov-dijak je gnezdena struktura
};
const int N = 10;//število dijakov

static void Main(string[] args)
{
    krozek T;
    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];
    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("  datum rojstva: ");
        T.Dijaki[i].rojstvo = Convert.ToDateTime(Console.ReadLine());
    }
    //metoda, ki izpiše seznam dijakov rojenih po leto 1990
    Seznam(T, 1990);
    //Izračunajmo skupno starost vseh dijakov glede na današnji datum
    DateTime d;
    d = DateTime.Now; //Now v sprem. tipa DateTime vrne trenutni datum
    int starost = 0;
    for (int i = 0; i < N; i++)
        starost = starost + d.Year - T.Dijaki[i].rojstvo.Year;
```

```
    Console.WriteLine("\nSkupna starost vseh tečajnikov je " + starost / N +
" let.");
}

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].rojstvo.Year > letnik)
            Console.WriteLine(T.Dijaki[i].ime + " " + T.Dijaki[i].priimek);
}
```

KONSTRUKOR

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

Pravilna tristrana pokončna piramida

Napišimo strukturo za tristrano pokončno piramido. Potrebujemo dve polji (osnovni rob in višino piramide).

```
struct piramida
{
    public int rob, visina;
};
static void Main(string[] args)
{
    piramida P1; //P1 še nima inicializiranih komponent
    Console.WriteLine(P1.rob + "  " + P1.visina); /*Compile time error, ker
komponente piramide P1 še niso inicializirane!!!*/
    //poskrbimo za inicializacijo
    P1.rob = 2;   //inicializacija prve komponente točke A
    P1.visina = 5;   //inicializacija druge komponente točke A
    Console.WriteLine(P1.rob + "  " + P1.visina); //Izpis 2  5
}
```

Spremenljivka *P1* iz zgornje vaje je spremenljivka vrednostnega tipa, shranjena na skladu in ob deklaraciji še ni inicializirana. Za inicializacijo komponent smo morali poskrbeti sami. 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. Spremenljivkam, ki jih ustvarimo s pomočjo operatorja *new* pravimo tudi *objekti* (več o objektih pa bomo govorili v poglavju *Razredi in objekti*).

```
//deklaracija nove REFERENČNE spremenljivke P2 za strukturo piramida
```

```
piramida P2 = new piramida();
/*POZOR: ker je P2 referenčna spremenljivka (ustvarili smo jo s pomočjo
operatorja new), sta polji rob in visina ŽE INICIALIZIRANI, imata torej
privzeto vrednost 0*/
```

Vrednost, ki so jo dobile komponente spremenljivke *P2*, 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 nima tipa, 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, *saj le-tega avtomatično ustvari prevajalnik sam*. (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 piramida
{
    public int rob, visina; //polji strukture piramida
    //Konstruktor
    public piramida(int robP, int visinaP)
    {
        rob = robP; visina = visinaP;
    }
}
static void Main(string[] args)
{
   /*nove REFERENČNA spremenljivke P1 za strukturo piramida: P1 se ustvari s
pomočjo privzetega konstruktorja, ki ga avtomatično naredi prevajalnik*/
   piramida P1 = new piramida();//rob in visina piramide P1 imata vrednosti 0
   /*deklaracija nove REFERENČNE spremenljivke P1 za strukturo piramida: P2
se ustvari s pomočjo našega konstruktorja*/
   piramida P2 = new piramida(2, 4);//rob piramide P2 postane 2, visina pa 4
}
```

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.

Padavine

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). Kreirajmo ustrezno podatkovno strukturo (ime strukture naj bo padavine). Napišimo metodo za vnos podatkov o padavinah za vseh 12 mesecev in metodo *Povp(kraj)*, ki vrne povprečno mesečno količino padavin poljubnega kraja!

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
//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
};

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 količine
//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"); //sprem. kraj1 je ustvarjena
    //na kopici (operator new)
    Vnos(kraj1); //klic metode za vnos padavin kraja kraj1

    padavine kraj2 = new padavine("Ljubljana");//še ena sprem. na kopici
    Vnos(kraj2);

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

    //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());
    }
}
```

Struktura zaposleni in tabela zaposlenih

Za zaposlenega potrebujejo v podjetju naslednje podatke: ime, priimek in osebne dohodke za zadnjih 12 mesecev (tabela 12 realnih števil). Napišimo deklaracijo za tako strukturo, lasten konstruktor, ter deklaracijo spremenljivk *Z1* (privzeti konstruktor) in *Z2* (lasten konstruktor) za tako strukturo. Deklarirajmo še tabelo *sindikat* 8 elementov tipa *zaposleni*.

```
struct zaposleni
{
    public string ime, priimek;
    public double[] dohodki;
    public zaposleni(string ime, string priimek)
    {
        this.ime = ime;
        this.priimek = priimek;
        dohodki = new double[12]; //dohodki dobijo privzete vrednosti 0
    }

}
static void Main(string[] args)
{
    //ustvarimo referenčno spremenljivko Z1 za novega zaposlenega
    zaposleni Z1 = new zaposleni();/*ime in priimek zaposlenega Z1 sta prazna
stringa. Elementi tabele dohodkov ŠE NISO INICIALIZIRANI.*/
    Z1.ime = "Tina"; //zaposlenemu Z1 je ime Tina
    Z1.priimek = "Kovač"; //zaposleni Z1 ima priimek Kovač
    //inicializirajmo tabelo dohodkov
    Z1.dohodki = new double[12];
    /*dohodki preko celega leta naj bodo enaki 550 - tabelaričnim elementom
določimo vrednosti 550*/
    for (int i = 0; i < 12; i++)
    {
        Z1.dohodki[i] = 550;
    }

    //ustvarimo referenčno spremenljivko Z2 za novega zaposlenega
```

```
    zaposleni Z2 = new zaposleni("Maja", "Kobal");/*ime zaposlenega Z2 je
Maja, priimek pa Kobal. Elementi tabele dohodkov SO ŽE INICIALIZIRANI in vsi
enaki 0*/

    //določimo npr. da je Majin dohodek v mesecu januarju enak 400!
    Z2.dohodki[0] = 400;

    //ustvarimo še TABELO z imenom sindikat, v kateri je 8 zaposlenih
    zaposleni[] sindikat;/*spremenljivka sindikat označuje da gre za tabelo,
v kateri bomo hranili zaposlene (dejansko gre za NASLOV v pomnilniku, kjer
bomo ustvarili tabelo)*/

    sindikat = new zaposleni[8];/*ustvarili smo tabelo velikosti 8 elementov
(dejansko je to tabela 8 naslovov za objekte tipa zaposleni)*/

    sindikat[0] = new zaposleni("Andreja", "Novak");/*ustvarili smo novega
zaposlenega, ki prestavlja element tabele z indeksom 0: to je Andreja Novak,
njeni celoletni dohodki so zaenkrat enaki 0*/

    Console.ReadKey();
}
```

METODE V STRUKTURAH

Struktura lahko vsebuje tudi metode, ki operirajo nad njenimi polji. Do takih metod dostopamo na enak način kakor do polj, torej s pomočjo operatorja pika!

Akvarij

Sestavimo strukturo *Akvarij*, ki temelji na obliki kvadra, torej vsebuje komponente *dolzino*, *sirino* in *visino*. Sestavimo tudi konstruktor, ki sprejme osnovne stranice v decimalnih številih. Struktura naj vsebuje metodi za izračun prostornine in površine akvarija. Nato sestavi program, ki preveri delovanje strukture in metod, npr. s klicem metode *Akvarij.Volumen* vrne prostornino akvarija v litrih in še metodo *Akvaruj.Povrsina*, ki vrne koliko stekla potrebuješ za izdelavo akvarija!

```
struct Akvarij
{
    public double dolzina, sirina, visina;
    public Akvarij(double dolzina, double sirina, double visina)
    {
        /*ker imajo parametri enaka imena kot so imena polj, uporabimo
parameter this*/
        this.dolzina = dolzina;
        this.sirina = sirina;
        this.visina = visina;
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
    }
    public double Volumen() //Metoda strukture Akvarij za izračun prostornine
    {
        return dolzina * sirina * visina;
    }
    public double Povrsina() //Metoda strukture Akvarij za izračun površine
    {
        return 2 * (dolzina * sirina + dolzina * visina) + dolzina * sirina;
    }
}
//glavni program
static void Main(string[] args)
{
    Akvarij A1 = new Akvarij(65.55, 43.90, 40.61);//dimenzije akvarija A1 v
cm
    Console.WriteLine("Prostornina akvarija je " + Math.Round(A1.Volumen(),
2));
    Console.WriteLine("Za izdelavo tega akvarija potrebujemo " +
Math.Round(A1.Povrsina() / 100, 2) + " dm2 stekla.");

    Akvarij A2; //Akvarij lahko ustvarimo seveda tudi na skladu
    A2.dolzina = 88.45;
    A2.sirina = 66.98; A2.visina = 52.91;
    Console.WriteLine("Prostornina akvarija A2 je "+ Math.Round(A2.Volumen(),
2));
    Console.ReadKey();
}
```

POVZETEK

S pomočjo struktur lahko na enostaven način združimo v neko celoto podatke različnih tipov, tako enostavnih, kot sestavljenih. Vanje lahko pišemo tudi metode, ki operirajo nad polji znotraj strukture. Spremenljivke tipa struktura lahko ustvarimo na skladu ali pa na kopici (v tem primeru si lahko pomagamo tudi s konstruktorjem), do njihovih komponent in metod pa dostopamo tako, da napišemo ime spremenljivke, ki ji sledi operator pika, nato pa ime polja oz. komponente ali pa ime metode te strukture. Strukture so v bistvu predhodnik najkompleksnejšega podatkovnega tipa *razred* (*class*), ki pa ga bomo spoznali v naslednjem poglavju!

vo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje
ropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

VAJE

1. 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 …!

2. 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 metodo za vnos podatkov v spremenljivki *P1* oz. *P2*. Izračunaj in izpiši obseg in ploščino obeh pravokotnikov!

3. Napiši program, s katerim bi za 20 trgovin (tabela trgovin!) vnesel podatke o nazivu trgovine in ceni za 1 kg kruha. Posamezna trgovina je struktura z dvema poljema (naziv in cena). Napiši metodo, ki dobi za parameter to tabelo, vrne pa naziv trgovine z najdražjim kruhom in metodo, ki dobi za parameter to tabelo, vrne pa povprečno ceno kruha.

4. Deklariraj strukturo, s katero boš opisal podatke o nekem rabljenem vozilu (tip vozila, prevoženi km in cena). Kreiraj zbirko takih struktur. Napiši program, ki bo v obliki menija nudil naslednje opcije:

    a. Vnos novega vozila v seznam.
    b. Izpis vseh podatkov o vozilih vozil dražjih od 10.000 EUR na zaslon.

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 metode, za vnos in izpis vseh ocen za enega dijaka!

6. Deklariraj strukturo, ki naj predstavlja kompleksno število. Struktura naj ima svoj konstruktor. Napiši še metodo za izpis kompleksnega števila, nato pa kreiraj tabelo 10 kompleksnih števil in jo inicializiraj z naključnimi vrednostmi obeh komponent.

7. Deklariraj strukturo *ulomek*, in strukturo *dvojni_ulomek*, ki naj vsebuje dve strukturi *ulomek*. Na kopici ustvari spremenljivko tipa *dvojni_ulomek*, števci in imenovalci naj bodo naključna števila med 1 in 10. Izpiši ta dvojni ulomek, nato pa še izračunaj vrednost tega dvojnega ulomka.

8. Stavnica na hipodromu vodi evidenco o tem, koliko denarja je bilo stavljenega na vsakega konja in kakšno je razmerje med vplačilom in dobitkom. Sestavi strukturo *Stava*, ki vsebuje polja *ime_konja*, *vplacana_stava* in *razmerje* (stavno razmerje med

posameznimi konji). Definiraj tudi ustrezni konstruktor. Pokaži, kako se uporabi konstruktor za strukturo *Stava*, če naj bo konju ime *Divja Strela*, zanj je bilo vplačanih 3450 €, stavno razmerje pa je 2 : 9! Za vajo ustvari še tabelo 5 konjev, vse podatke pa naj vnese uporabnik preko tipkovnice.

9. Pred začetkom sezonske razprodaje so se v eni od trgovin odločili, da bodo na artikle, na katerih je že napisana cena, dopisali le višino popusta v odstotkih in ne tudi znižane cene. Sestavi strukturo *Izdelek*, ki vsebuje polja *naziv*, *cena* in *popust*. Struktura naj vsebuje tudi konstruktor in metodo *NovaCena*, ki glede na popust vrne novo, znižano ceno izdelka. Napiši tudi metodo *Izpis*, ki izpiše ceno in popust izdelka. Ustvari še tabelo izdelkov, podatke naj vnese uporabnik.

10. Definiraj rstrukturo *Registracija*, ki vsebuje podatke o registrski številki avtomobila. V Sloveniji je registrska številka sestavljena iz dveh nizov. Prvi niz je območje in vsebuje dva znaka, drugi niz pa je registracija in vsebuje pet poljubnih znakov. Na primer, registrska številka LJ V1-02E sestoji iz območja LJ in registracije V102E. Napiši tudi ustrezen konstruktor, ki sprejme oba niza. Struktura naj vsebuje tudi metodo *VeljavnoObmocje()*, ki vrne *true*, če ima registrska številka veljavno območje. V Sloveniji so veljavna območja LJ, KR, KK, MB, MS, KP, GO, CE, SG, NM in PO. Napiši tudi primer uporabe te metode!

NAŠTEVNI TIP - enum

Naštevni tip (*enum*) je tip, v katerem je zbran seznam poimenovanih celoštevilskih konstant. Seznam se imenuje *naštevni seznam*, njegova vsebina pa so *naštevni identifikatorji*. Naštevni tip uporabljamo za imenovane vrednosti, ki so znane med prevajanjem. Temelji na celoštevilčnem osnovnem tipu.

Naštevanje (*enumeracija*) je bistvu alternativa konstantam. Naštevni tipi so zato vrednostni podatkovni tip, ki ga sestavlja množica konstant.

Recimo, da želimo deklarirati in inicializirati nalsednje konstante:

```
const int TockaZmrzovanja = 0;
const int TockaVrenja = 100;
const int TemperaturaZraka = 30;
const int TemperaturaVode = 25;
const int TemperaturaHladilnika = 4;
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Tak način deklaracije in inicializacije konstant je sicer legalen, a med temi konstantami ni nobene logične povezave, čeprav vse predstavljajo podatke o temperaturah. S pomočjo naštevanja bi bila deklaracija naslednja:

```
enum Temperature
{
    TockaZmrzovanja = 0,
    TemperaturaHladilnika = 4,
    TemperaturaVode = 25,
    TemperaturaZraka = 30,
    TockaVrenja = 100,
}
```

Celotno deklaracijo zapišemo izven glavnega programa. Na posamezno konstanto se sklicujemo preko imena naštevnega tipa, operatorjem pika in imenom konstante, npr.;

```
Console.WriteLine("Voda zavre pri {0} stopinj Celzija!",
Temperature.TockaVrenja);
//izpis: Voda zavre pri TockaVrenja stopinj Celzija!

Console.WriteLine("Voda zavre pri {0} stopinj Celzija!",
(int)Temperature.TockaVrenja);
//izpis: Voda zavre pri 100 stopinj Celzija!
Console.ReadKey();
```

Privzeto so vrednosti konstant tipa *int*, lahko pa določimo tudi katerikoli drug številni tip: *byte*, *sbyte*, *short*, *ushort*, *uint*, *long* ali *ulong* (tipa *char* in *bool* nista dovoljena!). npr.:

```
enum Velikost : uint
{
    majhen = 1,
    srednji = 2,
    velik = 3,
}
```

Prva konstanta naštevnega tipa ima privzeto vrednost 0 (razen, če ne deklariramo druge vrednosti), vrednost vsake naslednje konstante pa je za eno večja.

Zapišimo še splošno deklaracijo in inicializacijo naštevnega tipa:

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

Kot primer deklarirajmo naštevni tip s katerim bomo deklarirali in inicializirali množico vseh ocen. Prvi konstanti (*Negativno*) priredimo 1.

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

Ker ostalim konstantam vrednosti nismo priredili, bodo njihove vrednosti zaporedoma od 2 do 5!

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

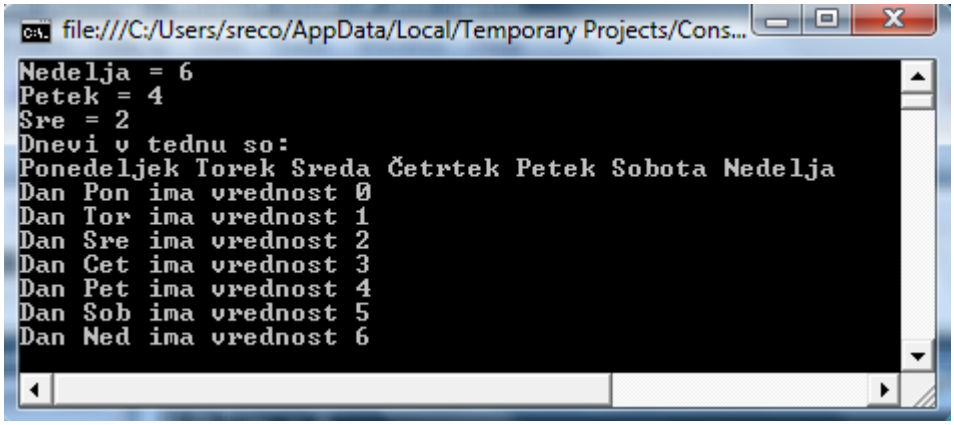
Naštevni tip Dan

Deklarirajmo naštevni tip *Dan* s sedmimi konstantami, ki predstavljajo dneve v tednu. V glavnem programu zapišimo nekaj stavkov, v katerih bomo uporabili tip *Dan*.

```
enum Dan { Pon, Tor, Sre, Cet, Pet, Sob, Ned };
static void Main(string[] args)
{
    //Primer uporabe v glavnem programu
    int x = (int)Dan.Ned;
    int y = (int)Dan.Pet;
    Console.WriteLine("Nedelja = " + x);//Izpis: Nedelja = 6
    Console.WriteLine("Petek = " + y);  //Izpis: Petek = 4

    Dan d = Dan.Sre;  /*deklaracija in inicializacija nove spremenljivke d,
ki je tipa Dan*/
    Console.WriteLine(d+" = " + (int)d);//Izpis: Sre = 2

    Dan d1;
    Console.WriteLine("Dnevi v tednu so: ");
    for (d1 = Dan.Pon; d1 <= Dan.Ned; d1++) /*operator ++ lahko uporabimo
tudi za spremenljivko naštevnega tipa*/
        Console.Write(Izpis(d1)+" ");
    Console.WriteLine();
    //Elemente naštevnega tipa lahko izpišemo tudi takole
    for (d1 = Dan.Pon; d1 <= Dan.Ned; d1++)
        Console.Write("Dan " + d1 + " ima vrednost " + (int)d1 + "\n");

    Console.ReadKey();
}
static string Izpis(Dan d) //metoda 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";
}
```

Ker nismo določili drugače, ima konstanta *Pon* vrednost 0, konstanta *Tor* vrednost 1, itd. Izpis zgornjega programa je takle:

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.



Slika 8: Naštevanje

Naštevanje poklicev

Deklarirajmo naštevni tip *Poklici* z nekaj 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 Poklici : byte
{
    Menedžer,
    Razvijalec,
    Administrator,
    Programer
}

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

public static void Main(string[] args)
{
    //Izpišimo vse poklice
```

```
    Console.WriteLine("Seznam vseh poklicev: ");
    for (Poklici p = Poklici.Menedžer; p <= Poklici.Programer; p++)
        Console.Write(p + " ");

    Zaposleni Z;//Z je vrednostna spremenljivka na skladu, vrednosti določimo
    Z.oddelek = 40;
    Z.ime = "Fred";
    Z.naziv = Poklici.Razvijalec;
    //Nov zaposleni naj dobi ime Marija, poklic programer, oddelek 11
    Zaposleni Z1 = new Zaposleni(Poklici.Programer, "Mary", 11);/*Z1 je
referenčna spremenljivka, ustvarjena na kopici*/
    Console.WriteLine("Podatki o delavcu: ");
    Console.WriteLine("Ime: " + Z1.ime + ", poklic: " + Z1.naziv + ",
oddelek: " + Z1.oddelek);
    Zaposleni Z2 = new Zaposleni();/*izvede se privzeti konstruktor:
numerična polja dobijo vrednost 0, polja tipa string pa so prazna*/
    Console.ReadKey();
}
```

POVZETEK

Naštevanje je krajši in bolj pregleden način za deklaracijo celoštevilskih konstant, ki po neki logiki spadajo skupaj. S pomočjo naštevnega seznama lahko kreiramo številne zelo koristne naštevne tipe, npr. za imena mesecev (Januar, Februar, …), tipe povezav (TCPIP, UDP, …), načine odpiranja datotek (SamoZaBranje, SamoZaPisanje, BranjeInPisanje, Dodajanje, …).

VAJE

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

2. 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 pa izpišemo element naštevnega tipa).

3. 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 metodo Vnos, ki prebere podatke o *CD*-ju. Vpisane podatke izpiši s pomočjo metode Izpis. Metoda Vnos ima za parameter referenco na strukturo (klic po referenci), metoda Izpis pa strukturo (prenos po vrednosti). Uporabnik naj vpiše podatke za dva *CD*-ja, ki ju potem izpiši.

4. Deklariraj naštevni tip *vrstaRože* (*vrtnica*=1, *lilija*, *dalija*, *nagelj*, *iris*, *mešano*..), ter naštevni tip *barvaRože* (*rdeča*=1, *bela*, rumena, *vijoličasta*, *oranžna*, *modra*, *zelena* …) Kreiraj strukturo *enostavenŠopek* s komponentami *tip* (*vrstaRože*), *barva* (*barvaRože*), *komadi* (*int*) ter *cena* (*decimal*). Napiši konstruktor, ki določi tip ter barvo rože. 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 *Šopki*, v katero boš lahko dodal poljubno število objektov tipa *enostavenŠopek*. Vsebino zbirke *Šopki* na koncu izpiši in obenem izračunaj ter izpiši njegovo ceno!

Seznam držav

Rešimo še nalogo za obdelavo evropskih držav, ki smo si jo zastavili na začetku kot učno situacijo. Za njeno rešitev potrebujemo tako znanje tabel, kot tudi zbirk in struktur.

```
struct Drzava
{
    //najprej deklaracija polj strukture drzava
    public string ime, glavno_mesto;
    public int prebivalci;
    public double povrsina;
    //konstruktor
    public Drzava(string ime, string glavno, int preb, double pov)
    {
        this.ime = ime;
        glavno_mesto = glavno;
        prebivalci = preb;
        povrsina = pov;
    }
    public double Gostota()//Metoda vrne gostoto prebivalcev
```

```
    {
        return Math.Round(prebivalci / povrsina,3);
    }
}
//glavni program
static void Main(string[] args)
{
    Drzava[] Evropa; //napoved tabele Evropa, ki bo vsebovala države
    Evropa = new Drzava[6];//za vajo bo v tabeli le 6 držav
    Evropa[0] = new Drzava("Slovenija", "Ljubljana",2000000,20000);
    Evropa[1] = new Drzava("Avstrija", "Dunaj",8100000,83858);
    Evropa[2] = new Drzava("Belgija", "Bruselj",10200000,30158);
    Evropa[3] = new Drzava("Francija", "Pariz",60400000,550000);
    Evropa[4] = new Drzava("Italija", "Rim",57600000,301263);
    Evropa[5] = new Drzava("Nemčija", "Berlin",82000000,356854);

    //ustvarimo zbirko že izbranih držav
    ArrayList izbrane = new ArrayList();
    //generator naključnih števil
    Random naklj = new Random();
    int pravilnih=0;//števec pravilnih odgovorov
    for (int i = 0; i < Evropa.Length ; i++)//obdelamo vse države v tabeli
    {
        while (true)
        {
            int indeks = naklj.Next(6);//naključno število med 0 in 5
            /*če to število v zbirki še ne obstaja, ga dodamo v zbirko,
uporabnika vprašamo za glavno mesto in gremo ven iz zanke while*/
            if (!izbrane.Contains(indeks))
            {
               izbrane.Add(indeks);
               Console.Write("Glavno mesto " + Evropa[indeks].ime + ": ");
               string odgovor=Console.ReadLine();
               if (odgovor == Evropa[indeks].glavno_mesto)
                   pravilnih++;
               break;//skok iz while zanke
            }
            /*če naključni indeks v zbirki izbrane že obstaja, gremo na
začetek zanke while*/
        }
    }
    Console.WriteLine("Število pravilnih odgovorov: "+pravilnih+" od
"+Evropa.Length);

    /*izračunajmo in izpišimo še povprečno gostoto prebivalcev posameznih
Evropskih držav: uporabimo metodo Gostota strukture Drzava*/
    Console.WriteLine("\nGostota prebivalcev evropskih držav: \n");
    for (int i=0;i<Evropa.Length;i++)
        Console.WriteLine(Evropa[i].ime+": "+ Evropa[i].Gostota());
    Console.ReadKey();
```

```
}
```

FARMA ZAJCEV

Odločili smo se, da bomo gojili zajce. Ker pa je zakonodaja zelo stroga, moramo voditi natančno evidenco o vsakem zajcu posebej (serijska številka, spol, masa), kakor tudi o celotni "farmi" zajcev. V vsakem trenutku bi radi imeli pregled nad številčnim stanjem in maso vseh zajcev, pa seveda vodili evidenco o najtežjem zajcu, številu zajkelj, masi najtežjega samca, ipd. Da bomo lahko napisali ustrezen program, bomo spoznali najkompleksnejšo podatkovno strukturo, ki se imenuje *razred* (*class*). Spoznali bomo pojem enkapsulacije, naučili se bomo prekrivati metode in še kaj!

RAZREDI IN OBJEKTI

UVOD

Pri programiranju se je izkazalo, da je zelo koristno, če podatke in postopke nad njimi združujemo v celote, ki jih imenujemo **objekte**. Programiramo potem tako, da objekte ustvarimo in jim naročamo, da izvedejo določene postopke. Programskim jezikom, ki omogočajo tako programiranje, rečemo, da so objektno (ali z drugim izrazom predmetno) usmerjeni. Objektno usmerjeno programiranje nam omogoča, da na problem gledamo kot na množico objektov, ki med seboj sodelujejo in vplivajo drug na drugega. Na ta način lažje pišemo sodobne programe. Objektno usmerjenih je danes večina sodobnih programskih jezikov.

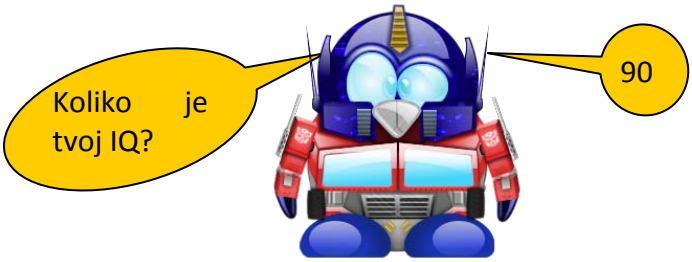
OBJEKTNO PROGRAMIRANJE – KAJ JE TO

Pri objektno usmerjenem programiranju se ukvarjamo z … objekti seveda. Objekt je nekakšna črna škatla, ki dobiva in pošilja sporočila. V tej črni škatli (objektu) se skriva tako koda (torej zaporedje programskih stavkov) kot tudi podatki (informacije, nad katerimi se izvaja koda). Pri klasičnem programiranju imamo kodo in podatke ločene. Tudi mi smo do sedaj programirali več ali manj na klasičen način. Pisali smo metode, ki smo jim podatke posredovali preko parametrov. Parametri in metode niso bili prav nič tesno povezani. Če smo npr. napisali metodo, ki je na primer poiskala največji element v tabeli števil, tabela in metoda nista bili združeni – tabela nič ni vedela o tem, da naj bi kdo npr. po njej iskal največje število.

V objektno usmerjenem programiranju (OO) pa so koda in podatki združeni v nedeljivo celoto – *objekt*. To prinaša določene prednosti. Ko uporabljamo objekte, nam nikoli ni potrebno pogledati v sam objekt. To je dejansko prvo pravilo OO programiranja – uporabniku ni nikoli potrebno vedeti, kaj se dogaja znotraj objekta. Gre dejansko zato, da nad objektom izvajamo različne metode. In za vsak objekt se točno ve, katere metode zanj obstajajo in kakšne rezultate vračajo. Recimo, da imamo določen objekt z imenom *robotek*. Vemo, da objekte take vrste, kot je *robotek,* lahko povprašamo o tem, koliko je njihov IQ. To storimo tako, da nad njimi izvedemo metodo

```
robotek.KolikoJeIQ()
```

Slika 9: Objekt robotek



Slika 10: Objekt sporoča

Objekt odgovori s sporočilom (metoda vrne rezultat), ki je neko celo število.

Tisto, kar je pomembno, je to, da nam, kot uporabnikom objekta, ni potrebno vedeti, kako objekt izračuna IQ (je to podatek, ki je zapisan nekje v objektu, je to rezultat nekega preračunavanja …?). Kot uporabnikom nam je važno le, da se z objektom lahko pogovarjamo o njegovem IQ. Vse "umazane podrobnosti" o tem, kaj je znotraj objekta, so prepuščene tistim, ki napišejo kodo, s pomočjo katere se ustvari objekt.

Seveda se lahko zgodi, da se kasneje ugotovi, da je potrebno "preurediti notranjost objekta". In tu se izkaže prednost koncepta objektnega programiranja. V programih, kjer objekte uporabljamo, nič ne vemo o tem, kaj je "znotraj škatle", kaj se na primer dogaja, ko objekt povprašamo po IQ. Z objektom "komuniciramo" le preko izvajanja metod. Ker pa se klic metode ne spremeni, bodo programi navkljub spremembam v notranjosti objekta še vedno delovali.

Če torej na objekte gledamo kot na črne škatle (zakaj črne – zato, da se ne vidi, kaj je znotraj!) in z njimi ravnamo le preko sporočil (preko klicev metod), naši programi delujejo ne glede na morebitne spremembe v samih objektih.

Omogočanje dostopa do objekta samo preko sporočil in onemogočanje uporabnikom, da vidijo (in brskajo) po podrobnostih, se imenuje skrivanje informacij ali še bolj učeno **enkapsulacija**. In zakaj je to pomembno? Velja namreč, da se programi ves čas spreminjajo. Velika večina programov se dandanes ne napiše na novo, ampak se spremeni obstoječ program. In večina napak izhaja iz tega, da nekdo spremeni določen delček kode, ta pa potem povzroči, da drug del

programa ne deluje več. Če pa so tisti delčki varno zapakirani v kapsule, se spremembe znotraj kapsule ostalega sistema prav nič ne tičejo.

Pravzaprav smo se že ves čas ukvarjali z objekti, saj smo v naših programih uporabljali različne objekte, ki so že pripravljeni v standardnih knjižnicah jezika C#. Pa tudi v poglavju o strukturah smo pojem objekta že omenili, ko smo spremenljivke tipa strukture ustvarjali na kopici. Oglejmo si dva primera objektov iz standardne knjižnice jezika:

Objekt *System.Console* predstavlja *standardni izhod*. Kadar želimo kaj izpisati na zaslon, pokličemo metodo *Write* na objektu *System.Console*.

Objekt tipa *Random* predstavlja generator naključnih števil. Metoda *Next(a, b)*, ki jo izvedemo nad objektom tega tipa, nam vrne neko naključno celo število med *a* in *b*.

In če se vrnemo na razlago v uvodu – kot uporabniki čisto nič ne vemo (in nas pravzaprav ne zanima), kako metoda *Next* določi naključno število. In tudi, če se bo kasneje "škatla" *Random* spremenila in bo metoda *Next* delovala na drug način, to ne bo "prizadelo" programov, kjer smo objekte razreda *Random* uporabljali. Seveda, če bo sporočilni sistem ostal enak. V omenjenem primeru to pomeni, da se bo metoda še vedno imenovala *Next*, da bo imela dva parametra, ki bosta določala meje za naključna števila in da bo odgovor (povratno sporočilo) neko naključno število z želenega intervala.

Naučili smo se torej uporabnosti objektnega programiranja, nismo pa se še naučili izdelave svojih razredov. Seveda nismo omejeni le na to, da bi le uporabljali te "črne škatle", kot jih pripravi kdo drug (npr. jih dobimo v standardni knjižnici jezika). Objekti so uporabni predvsem zato, ker lahko programer definira nove razrede in objekte, torej sam ustvarja te nove črne škatle, ki jih potem on sam in drugi uporabljajo.

RAZRED

Razred je abstraktna definicija objekta (torej opis, kako je naša škatla videti znotraj). Z razredom opišemo, kako je neka vrsta objektov videti. Če na primer sestavljamo razred *zajec*, opisujemo, katere so tiste lastnosti, ki določajo vse zajce.

Če želimo nek razred uporabiti, mora običajno obstajati vsaj en *primerek* razreda. Primerek nekega razreda imenujemo *objekt*. Ustvarimo ga s ključno besedo *new*.

```
imeRazreda primerek = new imeRazreda();
```

Lastnosti objektov določene vrste so zapisane v razredu (***class***). Ta opisuje katere *podatke* hranimo o objektih te vrste in katere so *metode* oz. odzivi objektov na sporočila. Stanje objekta torej opišemo s spremenljivkami (rečemo jim tudi *polja ali komponente*, enako kot pri strukturah), njihovo obnašanje oz. odzive pa z metodami.

Oglejmo si primer programa v C#, ki uporablja objekte, ki jih napišemo sami.

```
public class MojR
{
    private string mojNiz;
    public MojR(string nekNiz)
    {
        mojNiz = nekNiz;
    }
    public void Izpisi()
    {
        Console.WriteLine(mojNiz);
    }
}
public static void Main(string[] arg)
{
    MojR prvi;                                  // oznaka (ime) objekta
    prvi=new MojR("Pozdravljen, moj prvi objekt v C#!"); //kreiranje objekta
    prvi.Izpisi();                              //ukaz objektu
}
```

Definicija razreda

Na kratko razložimo, "kaj smo počeli". Pred glavnim programom smo definirali nov razred z imenom *MojR*. Ta je tak, da o vsakem objektu te vrste poznamo nek niz, ki ga hranimo v njem. Vse znanje, ki ga objekti tega razreda imajo je, da se znajo odzvati ne metodo *Izpis()*, ko izpišejo svojo vsebino (torej niz, ki ga hranimo v njem).

Glavni program *Main* (glavna metoda) je tisti del rešitve, ki opravi dejansko neko delo. Najprej naredimo primerek objekta iz razreda *MojR* (prvi) in vanj shranimo niz *"Pozdravljen, moj prvi objekt v C#!"*. Temu objektu nato naročimo, naj izpiše svojo vsebino.

Povejmo še nekaj o drugem načelu združevanja (***enkapsulacija***), o pojmu **dostopnost**. Potem, ko smo metode in polje združili 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. Z besedicami *public*, *private* in *protected*, ki jih napišemo pred ime metode ali polja, lahko kontroliramo, katere metode in polja bodo dostopna tudi od zunaj:

Metoda ali polje je **privatno** (*private*), kadar je dostopno le znotraj razreda.

Metoda ali polje je **javno** (*public*), kadar je dostopno tako znotraj kot tudi izven razreda.

Metoda ali polje je **zaščiteno** (*protected*), kadar je vidno le znotraj razreda ali pa v *podedovanih* (*izpeljanih*) razredih.

Obstajajo tudi druge dostopnosti, a se z njimi ne bomo ukvarjali. Prav tako ne bomo uporabljali zaščite *protected*. Omejili se bomo le na privatni (*private*) in javni dostop (*public*). Velja pa pravilo, da če oznako dostopnosti ne napišemo, je polje zasebno!

USTVARJANJE OBJEKTOV

V prejšnjem primeru smo iz razreda *MojR* tvorili objekt prvi.

```
MojR prvi;  // prvi je NASLOV objekta
```

Z naslednjim ukazom v pomnilniku naredimo objekt tipa *MojR*, po pravilih, ki so določena z opisom razreda *MojR*. Novo ustvarjeni objekt se nahaja na naslovu *prvi*.

```
prvi = new MojR("Pozdravljen, moj prvi objekt v C#!");
```

Razred je tako v bistvu **šablona**, ki definira spremenljivke in metode, skupne vsem objektom iste vrste, objekt pa je **primerek** (srečali boste tudi "čuden" izraz *instanca*) nekega razreda.

NASLOV OBJEKTA

Sicer vedno rečemo, da v spremenljivki *prvi* hranimo objekt, a zavedati se moramo, da to ni čisto res. V spremenljivki *prvi* je shranjen **naslov** objekta. Zato po

```
MojR prvi, drugi;
```

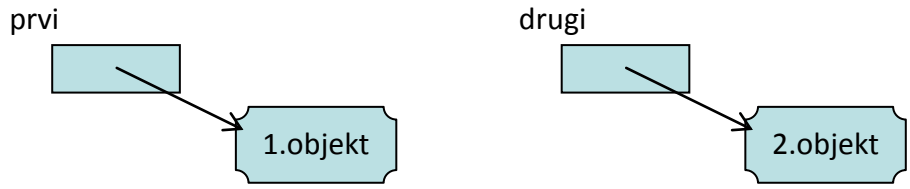
nimamo še nobenega objekta. Imamo le dve spremenljivki, v katerih lahko shranimo naslov, kjer bo operator *new* ustvaril nov objekt. Rečemo tudi, da spremenljivki *prvi* in *drugi* kažeta na objekta tipa *MojR*. Če potem napišemo

```
prvi = new MojR("1.objekt");
```

smo v spremenljivko *prvi* shranili naslov, kjer je novo ustvarjeni objekt. Ustvarimo še en objekt in njegov naslov shranimo v spremenljivko *drugi*

```
drugi = new MojR("2. objekt");
```

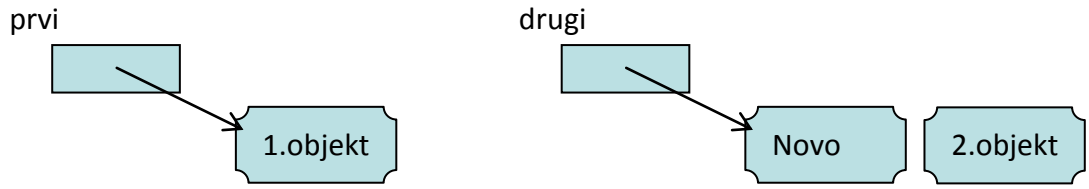
Poglejmo, kakšno je sedaj stanje v pomnilniku. S puščico v spremenljivkah *prvi* in *drugi* smo označili ustrezen naslov objektov. Dejansko je na tistem mestu napisano nekaj v stilu *h002a22* (torej naslov mesta v pomnilniku)



In če sedaj napišemo

```
drugi = new MojR("Novo");
```

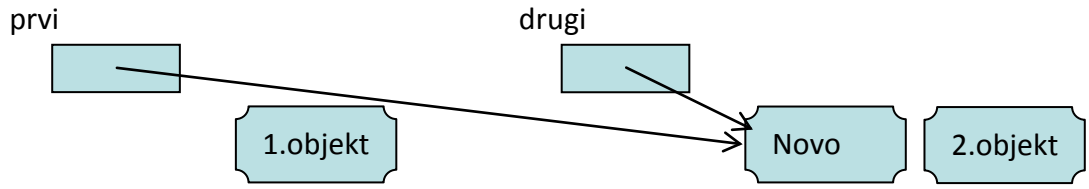
je vse v redu. Le do objekta z vsebino "2. objekt" ne moremo več! Stanje v pomnilniku je



in če naredimo še

```
prvi = drugi;
```

smo s tem izgubili še dostop do objekta, ki smo ga prvega ustvarili in pomnilnik bo videti takole.



Sedaj tako spremenljivka *prvi* in *drugi* vsebujeta naslov istega objekta. Do objektov z vsebino "1. objekt" in "2. objekt" pa ne more nihče več. Na srečo bo kmalu prišel smetar in ju pometel proč. Smetar je poseben program v C#, za katerega delovanje nam ni potrebno skrbeti in poskrbi, da se po pomnilniku ne nabere preveč objektov, katerih naslova ne pozna nihče več.

Sicer bomo vedno govorili kot: *... v objektu mojObjekt imamo niz, nad objektom zajecRjavko izvedemo metodo ...,* a vedeti moramo, da sta v spremenljivkah *mojObjekt* in *zajecRjavko* naslova in ne dejanska objekta.

Kako se torej lotiti načrtovanja rešitve s pomočjo objektnega programiranja?

- Z analizo ugotovimo, kakšne objekte potrebujemo za reševanje.
- Pregledamo, ali že imamo na voljo ustrezen razred (standardne knjižnice, druge knjižnice, naši stari razredi).
- Sestavimo ustrezen razred (določimo polja in metode našega razreda).
- Sestavimo "glavni" program, kjer s pomočjo objektov rešimo problem.

Razred Krog

Radi bi napisali program, ki bo prebral polmer kroga in izračunal ter izpisal njegovo ploščino. "Klasično" bi program napisali takole:

```
static void Main(string[] args)
{
    //  vnos podatka
```

```
     Console.Write("Polmer kroga: ");
     int r = int.Parse(Console.ReadLine());//Parse pretvori string v int
      //  izračun
     double ploscina = Math.PI * r * r;
      //  izpis
     Console.WriteLine("Ploščina kroga: " + ploscina);
}
```

Sedaj pa to naredimo še "objektno". Očitno bomo potrebovali razred *Krog*. Objekt tipa *Krog* hrani podatke o svojem polmeru. Njegovo znanje pa je, da "zna" izračunati svojo ploščino.

```
class Krog
{
    public double polmer;    // javno polje razreda Krog
    public double Ploscina() // javna metoda razreda krog
    {
        return Math.PI * polmer * polmer;
    }
}
public static void Main(string[] arg)
{
    Krog k = new Krog();  // naredimo nov objekt tipa krog
    Console.Write("Polmer: ");
    // do polj in metod objekta k dostopamo z operatorjem pika
    k.polmer = double.Parse(Console.ReadLine());  // polmer preberemo
    Console.WriteLine(k.Ploscina());   // izpis ploščine
}
```

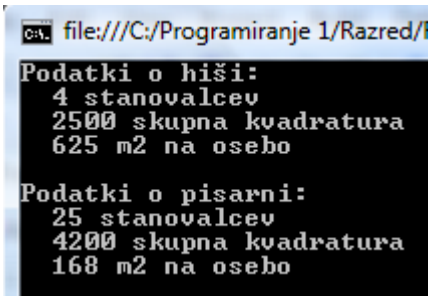
Zgradba

Pogosto razrede uporabimo le zato, da v celoto združimo podatke o neki stvari. Takrat v razred "zapremo" le polja, ki imajo vsa javni dostop, metod pa v razredu ni. Napišimo razred *Zgradba* s poljema *kvadratura* in *stanovalcev*.

```
class Zgradba // deklaracija razreda Zgradba z dvema javnima poljema.
{
    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;  //  spremenljivka za izračun kvadrature na osebo
    // določimo začetne vrednosti prvega objekta
```

```
    hiša.stanovalcev = 4;
    hiša.kvadratura = 2500;
    // določimo začetne vrednosti drugega objekta
    pisarna.stanovalcev = 25;
    pisarna.kvadratura = 4200;

    // izračun kvadrature za prvi objekt
    kvadraturaPP = hiša.kvadratura / hiša.stanovalcev;

    Console.WriteLine("Podatki o hiši:\n  " + hiša.stanovalcev + "
stanovalcev\n  " +
                hiša.kvadratura + " skupna kvadratura\n  " + kvadraturaPP + "
m2 na osebo");

    Console.WriteLine();
    // izračun kvadrature za drugi objekt
    kvadraturaPP = pisarna.kvadratura / pisarna.stanovalcev;

    Console.WriteLine("Podatki o pisarni:\n  " + pisarna.stanovalcev + "
stanovalcev\n  " +
        pisarna.kvadratura + " skupna kvadratura\n  " + kvadraturaPP + " m2
na osebo");
}
```

Izpis, ki ga dobimo, je naslednji:



Slika 11: Uporaba razreda *Zgradba*

Razred *Zgradba* je sedaj nov tip podatkov, ki ga pozna C#. Uporabljamo ga tako kot vse druge v C# in njegove knjižnice vgrajene tipe. Torej lahko napišemo

```
Zgradba[] ulica;   // ulica bo tabela objektov tipa Zgradba
```

Pri tem pa moramo sedaj paziti na več stvari. Zato bomo o tabelah objektov spregovorili v posebnem razdelku.

Evidenca članov kluba

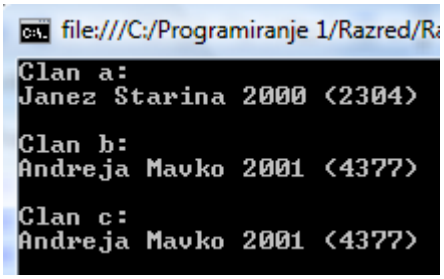
Napišimo program, ki vodi evidenco o članih športnega kluba. Podatki o članu obsegajo ime, priimek, letnico vpisa v klub in vpisno številke (seveda je to poenostavljen primer). Torej objekt, ki predstavlja člana kluba, vsebuje štiri podatke. Ustrezni razred je:

```
public class Clan
{
    public string ime;
    public string priimek;
    public int leto_vpisa;
    public string vpisna_st;
}
```

S tem smo povedali, da je vsak *objekt* tipa *Clan* sestavljen iz štirih komponent: *ime*, *priimek*, *leto_vpisa* in *vpisna_st*. Če je *a* objekt tipa *Clan*, potem lahko dostopamo do njegove komponente *ime* tako, da napišemo *a.ime*. Tvorimo nekaj objektov:

```
static void Main(string[] args)
{
    Clan a = new Clan();
    a.ime = "Janez";
    a.priimek = "Starina";
    a.leto_vpisa = 2000;
    a.vpisna_st = "2304";
    Clan b = new Clan();
    b.ime = "Mojca";
    b.priimek = "Mavko";
    b.leto_vpisa = 2001;
    b.vpisna_st = "4377";
    Clan c = b;
    c.ime = "Andreja";
    Console.WriteLine("Član a:\n" + a.ime + " " + a.priimek +
                                          " " + a.leto_vpisa + " (" +
a.vpisna_st + ")\n");
    Console.WriteLine("Član b:\n" + b.ime + " " + b.priimek +
                                          " " + b.leto_vpisa + " (" +
b.vpisna_st + ")\n");
    Console.WriteLine("Član c:\n" + c.ime + " " + c.priimek +
                                          " " + c.leto_vpisa + " (" +
c.vpisna_st + ")\n");
}
```

Ko program poženemo, dobimo naslednji izpis:



Slika 12: Objekti tipa *Clan*

Zakaj sta dva zadnja izpisa enaka? V programu smo naredili dva nova objekta razreda *Clan*, ki sta shranjena v spremenljivkah *a* in *b*. A spomnimo se, da imamo v spremenljivkah *a* in *b* shranjena *naslova* objektov, ki smo ju naredili. Kot vedno nove objekte naredimo z ukazom *new*. Spremenljivka *c* pa **kaže na isti objekt kot** *b*, se pravi, da se v vrstici *Clan c = b; ni* naredila nova kopija objekta *b*, ampak se spremenljivki *b* in *c sklicujeta* na isti objekt. In zato smo v vrstici *c.ime = "Andreja";* vplivali tudi na ime, shranjeno v objektu *b (*bolj točno, na ime, shranjeno v objektu, katerega naslov je shranjen v *b*)!

V zgornjem primeru smo naredili objekt *a* (naredili smo objekt in nanj pokazali z *a*) in nastavili vrednosti njegovih komponent. Takole nastavljanje je v praksi dokaj neprimerno, ker se zlahka zgodi, da kako komponento pozabimo nastaviti. Zato C# omogoča, da delamo nove objekte na bolj praktičen način s pomočjo **konstruktorjev**.

KONSTRUKTOR

Ob tvorbi objekta bi radi hkrati nastavili začetno stanje polj. To nam omogočajo konstruktorji(podobno kot pri strukturah, a razlike so kar precejšnje). **Konstruktor** je metoda, ki jo pokličemo ob tvorbi objekta z operatorjem *new*. Je brez tipa rezultata. Ne smemo jo zamenjati z metodami, ki rezultata ne vračajo (te so tipa *void*). Konstruktor tipa rezultata sploh nima, tudi *void* ne. Prav tako smo omejeni pri izbiri imena te metode. Konstruktor mora imeti ime nujno tako, kot je ime razreda. Če konstruktorja ne napišemo, ga "naredi" prevajalnik sam (a metoda ne naredi nič). Vendar pozor: kakor hitro napišemo vsaj en svoj konstruktor, C# ne doda svojega.

Največja omejitev, ki loči konstruktorje od drugih metod, je ta, da konstruktorja ne moremo klicati na poljubnem mestu (kot lahko ostale metode). Kličemo ga izključno skupaj z *new*, npr. *new Drzava()*. Uporabljamo jih za vzpostavitev začetnega stanja objekta.

Nepremičnine

Sestavimo razred, ki bo v svoja polja lahko shranil ulico, številko nepremičnine ter vrsto nepremičnine. Ustvarimo poljuben objekt, ga inicializirajmo in ustvarimo izpis, ki naj zgleda približno takole: Na naslovu Cankarjeva ulica 32, Kranj je blok.

```
class Nepremicnina
{
    public string ulica;
    public int stevilka;
    public string vrsta;
    // konstruktor
    public Nepremicnina(string kateraUlica, int hisnaStevilka, string
vrstaNep)
    {
        this.ulica = kateraUlica;
        this.stevilka = hisnaStevilka;
        this.vrsta = vrstaNep;
    }
}

static void Main(string[] args)
{
    // za kreiranje novega objekta uporabimo konstruktor
    Nepremicnina nova = new Nepremicnina("Cankarjeva ulica 32, Kranj", 1234,
"blok");
    // še izpis podatkov o nepremičnini
    Console.WriteLine("Na naslovu " + nova.ulica + " je " + nova.vrsta);
    Console.ReadKey();
}
```

Rezervirano besedo ***this*** smo spoznali že pri strukturah: *this* označuje objekt, ki ga "obdelujemo". V konstruktorju je to objekt, ki ga ustvarjamo. *this.ulica* se torej nanaša na lastnost/komponento spol objekta, ki se ustvarja (ki ga je ravno naredil *new*).

Denimo, da imamo razred *Zajec* in v nekem programu napišemo

```
Zajec rjavko = new Zajec();
Zajec belko = new Zajec();
```

Pri prvem klicu se ob uporabi konstruktorja *Zajec() this* nanašal na *rjavko*, pri drugem na *belko*.

Na ta način (z *this*) se lahko sklicujemo na objekt vedno, kadar pišemo metodo, ki jo izvajamo nad nekim objektom. Denimo, da smo napisali

```
Random ng = new Random();
Random g2 = new Random();
Console.WriteLine(ng.Next(1, 10));
```

Kako so programerji, ki so sestavljali knjižnico in v njej razred *Random*, lahko napisali kodo metode, da se je vedelo, da pri metodi *Next* mislimo na uporabo generatorja *ng* in ne na *g2*?

Denimo, da imamo razred *MojRazred* (v njem pa komponento *starost*) in v njem metodo *MetodaNeka*. V programu, kjer razred *MojRazred* uporabljamo, ustvarimo dva objekta tipa *MojRazred*. Naj bosta to *objA* in *objC*. Kako se znotraj metode *MetodaNeka* sklicati na ta objekt (objekt, nad katerim je izvajana metoda)? Če torej metodo *MetodaNeka* kličemo nad obema objektoma z *objA.MetodaNeka()* oziroma z *objC.MetodaNeka()*, kako v postopku za *metodaNeka* povedati, da gre:

prvič za objekt z imenom *objA* in

drugič za objekt z imenom *objC*

Če se moramo v kodi metode *metodaNeka* sklicati recimo na komponento *starost* (jo recimo izpisati na zaslon), kako povedati, da naj se ob klicu *objA.MetodaNeka()* uporabi starost objekta *objA*, ob klicu *objC.MetodaNeka()* pa starost objekta *objC*?

```
Console.WriteLine("Starost je: " + ?????.starost);
```

Ob prvem klicu so ???? *objA*, ob drugem pa *objC*. To "zamenjavo" dosežemo z *this*. Napišemo

```
Console.WriteLine("Starost je: " + this.starost);
```

Ob prvem klicu *this* pomeni *objA*, ob drugem pa *objC*.

Torej v metodi na tistem mestu, kjer potrebujemo konkretni objekt, nad katerim metodo izvajamo, napišemo *this*.

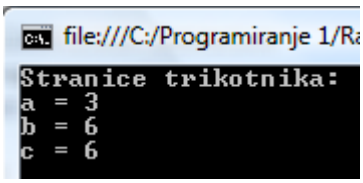
Trikotnik

Napišimo razred *Trikotnik*, ki bo vseboval konstruktor za generiranje poljubnih stranic trikotnika, pri čemer bomo upoštevali trikotniško pravilo: vsota poljubnih dveh stranic trikotnika mora biti daljša od tretje stranice.

```
class Trikotnik // deklaracija razreda Trikotnik
{
    public int a, b, c;   // stranice trikotnika
    public Trikotnik()    // konstruktor
    {
        Random naklj = new Random();
        this.a = naklj.Next(1, 10);
        this.b = naklj.Next(1, 10);
        /*tretjo stranico delamo toliko časa, da stranice zadoščajo
trikotniškemu pravilu*/
        while (true) // neskončna zanka
        {
```

```
            this.c = naklj.Next(1, 10);
            // sestavljeni pogoj za preverjanje stranic
            if ((a + b) > c && (a + c) > b && (b + c) > a)
                break;  // če dolžina ustreza, izstopimo iz zanke
        }
    }
}

static void Main(string[] args)
{
    Trikotnik trik1 = new Trikotnik();
    Console.WriteLine("Stranice trikotnika: \na = "+trik1.a+"\nb = "+ trik1.b
    + "\nc = " + trik1.c);
}
```

Izpis:



Slika 13: Razred Trikotnik

V pogojnem stavku

```
if ((a + b) > c && (a + c) > b && (b + c) > a)  /* sestavljeni pogoj za
preverjanje stranic*/
```

nismo pisali *this.a*, *this.b* in *this.c*,

```
if ((this.a + this.b) > this.c && ...
```

a se prevajalnik ni pritožil. Namreč če ni možnosti za zamenjavo, *this*. lahko izpustimo. A vsaj začetnikom bo verjetno lažje, če bodo *this*. vsaj v začetku vedno pisali.

Če potrebujemo več načinov, kako nastaviti začetne vrednosti polj nekega objekta, moramo pripraviti več konstruktorjev. A pri tem imamo težavo. Namreč vsi konstruktorji se morajo imenovati tako kot razred. Kako pa bo C# potem vedel, kateri konstruktor mislimo? Če si ogledamo zgled, kjer smo razred *Trikotnik* razširili še z enim konstruktorjem, vidimo, da očitno to izvedljivo. Težav ni, ker C# podpira **preobteževanje** metod, ki jo bomo spoznali v naslednjem razdelku.

```
class Trikotnik // deklaracija razreda Trikotnik
{
    public int a, b, c;   // stranice trikotnika
    // konstruktor
    public Trikotnik() //privzeti konstruktor
    {
```

```
        Random naklj = new Random();
        this.a = naklj.Next(1, 10);
        this.b = naklj.Next(1, 10);
        /*tretjo stranico delamo toliko časa, da stranice zadoščajo
trikotniškemu pravilu*/
        while (true)   // neskončna zanka
        {
            this.c = naklj.Next(1, 10);
            if ((a + b) > c && (a + c) > b &&
                    (b + c) > a)  // sestavljeni pogoj za preverjanje stranic
                break;  // če dolžina ustreza, izstopimo iz zanke
        }
    }
    public Trikotnik(int a, int b, int c) //preobteženi konstruktor
    {
        this.a = a;
        this.b = b;
        this.a = c;
    }
}

static void Main(string[] args)
{
    Trikotnik trik1 = new Trikotnik();
    Trikotnik trik2 = new Trikotnik(3, 4, 5);
    Console.WriteLine("Stranice trikotnika: \na = " + trik1.a + "\nb = " +
trik1.b
                                        + "\nc = " + trik1.c);
    Console.WriteLine("Stranice trikotnika: \na = " + trik2.a + "\nb = " +
trik2.b
                                        + "\nc = " + trik2.c);
}
```

V tem zgledu vidimo še primer, ko je *this*. nujno uporabljati. Namreč pri

```
this.a = a;
```

moramo razlikovati med spremenljivko *a*, ki označuje parameter metode (konstruktorja), in med poljem *a*. Z uporabo *this.a* dileme ni. Če pa *this.* ne uporabimo, se vedno uporabi "najbližja" definicija spremenljivke. V našem primeru bi to bil parameter *a*.

OBJEKTNE METODE

Svoj pravi pomen dobi sestavljanje razredov takrat, ko objektu pridružimo še metode, torej znanje nekega objekta. Kot že vemo iz uporabe vgrajenih razredov, metode nad nekim objektom kličemo tako, da navedemo ime objekta, piko in ime metode. Tako, če želimo izvesti metodo *WriteLine* nad objektom *Console*, napišemo

```
Console.WriteLine("To naj se izpiše");
```

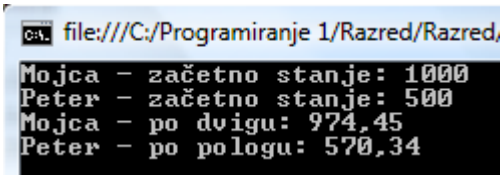
Če želimo izvesti metodo *Equals* nad nizom *besedilo*, napišemo

besedilo.Equals(primerjava);

Denarnica

Sestavimo razred *denarnica*, ki bo omogočal naslednje operacije: dvig, vlogo in ugotavljanje stanja. Začetna vrednost naj se postavi s konstruktorjem.

```
public class denarnica
{    // razred ima dve polji: ime osebe in stanje v denarnici
    public string ime;
    public double stanje;
    // konstruktor za nastavljanje začetnih vrednosti
    public denarnica(string ime, double znesek)
    {
        this.ime = ime;
        stanje = znesek;
    }
    public void dvig(double znesek)
    {
        stanje = stanje - znesek;
    }
    public void polog(double znesek)
    {
        stanje = stanje + znesek;
    }
}
static void Main(string[] args)
{
    // tvorimo dva objekta – uporabimo konstruktor
    denarnica Mojca = new denarnica("Mojca", 1000);
    denarnica Peter = new denarnica("Peter", 500);
    // izpis začetnega stanja v denarnici za oba objekta
    Console.WriteLine("Mojca – začetno stanje: " + Mojca.stanje);
    Console.WriteLine("Peter – začetno stanje: " + Peter.stanje);
    Mojca.dvig(25.55);  // Mojca zapravlja
    Peter.polog(70.34);  // Peter ja zaslužil
    // izpis novega stanja v denarnici za oba objekta
    Console.WriteLine("Mojca – po dvigu: " + Mojca.stanje);
    Console.WriteLine("Peter – po pologu: " + Peter.stanje);
}
```



Slika 14: Objekta razreda *Denarnica*

Prodajalec

Prodajalcu napišimo program, ki mu bo pomagal pri vodenju evidence o mesečni in letni prodaji.

```
class Prodajalec
{
    public double[] zneski; // zasebna tabela ki hrani mesečne zneske prodaje
    public Prodajalec()  // konstruktor ki inicializira tabelo prodaje
    {
        zneski = new double[12];
    }
    // metoda za izpis letne prodaje
    public void IzpisiLetnoProdajo()
    {
        Console.WriteLine("Skupna prodaja: " + SkupnaLetnaProdaja()+" EUR");
    }
    // metoda za izračun skupne letne prodaje
    public double SkupnaLetnaProdaja()
    {
        double vsota = 0.0;
        for (int i = 0; i < 12; i++)
            vsota += zneski[i];
        return vsota;
    }
}
static void Main(string[] args)
{
    //  tvorba objekta p tipa Prodajalec, obenem se izvede tudi konstruktor
    Prodajalec p = new Prodajalec();
    //  zanka za vnos prodaje po mesecih
    for (int i = 1; i <= 12; i++)
    {
        Console.Write("Vnesi znesek prodaje za mesec " + i + ": ");
        p.zneski[i - 1] = Convert.ToDouble(Console.ReadLine());
    }
    p.IzpisiLetnoProdajo(); // objekt p pozna metodo za izpis letne prodaje
```

```
}
```

V zgornjem primeru smo v razredu napovedali tabelo *zneski*. Za inicializacijo te tabele smo napisali konstruktor, seveda pa bi lahko tabelo inicializirali že pri deklaraciji s stavkom:

```
public double[] zneski = new double[12]; /*vsi elementi tabele dobijo
vrednost 0*/
```

PREOBTEŽENE METODE

Jezik C# podpira tako imenovano preobteževanje (*overloading*) metod. Tako imamo lahko znotraj istega programa, ali pa znotraj istega razreda, več metod z enakim imenom. Metode se morajo razlikovati ne v imenu, ampak podpisu. **Podpis** metode je sestavljen iz imena metode in tipov vseh parametrov. V razredu *Trikotnik* ima tako npr. konstruktor *Trikotnik()* in podpis enostavno *Trikotnik*, konstruktor *public Trikotnik (int a, int b, int c)* pa podpis *Trikotnik _int_int_int*. Tip rezultata (*return tip*) **ni** del podpisa!

Preobtežene so lahko tudi "navadne" metode (ne le konstruktorji). Zakaj je to uporabno? Denimo, da imamo metodo *Ploscina*, ki kot parameter dobi objekt iz razreda *Kvadrat*, *Krog* ali *Pravokotnik*. Kako jo začeti?

```
public static double Ploscina(X lik)
```

Ker ne vemo, kaj bi napisali za tip parametra, smo napisali kar *X*. Če preobteževanje ne bi bilo možno, bi morali napisati metodo, ki bi sprejela parameter tipa *X*, kjer je *X* bodisi *Kvadrat*, *Krog* ali *Pravokotnik*. Seveda to ne gre, saj prevajalnik nima načina, da bi zagotovil, da je *X* oznaka bodisi za tip *Kvadrat*, tip *Krog* ali pa tip *Pravokotnik*. Pa tudi če bi šlo – kako bi nadaljevali? V kodi bi morali reči: če je lik tipa *Kvadrat*, uporabi to formulo, če je lik tipa *Krog* spet drugo. S preobteževanjem problem enostavno rešimo. Napišemo tri metode

```
public static double Ploscina(Kvadrat lik) { }
public static double Ploscina(Krog lik) { }
public static double Ploscina(Pravokotnik lik) { }
```

Ker za vsak lik znotraj ustrezne metode točno vemo, kakšen je, tudi ni težav z uporabo pravilne formule. Imamo torej tri metode z enakim imenom, a različnimi podpisi. Seveda bi lahko problem rešili brez preobteževanja, recimo takole:

```
public static double PloscinaKvadrata(Kvadrat lik) { }
public static double PloscinaKroga(Krog lik) { }
public static double PloscinaPravokotnika(Pravokotnik lik) { }
```

A s stališča uporabnika metod je uporaba preobteženih metod enostavnejša. Če ima nek lik *bla*, ki je bodisi tipa *Kvadrat*, tipa *Krog* ali pa tipa *Pravokotnik*, metodo pokliče z *Ploscina(bla)*, ne da bi mu bilo potrebno razmišljati, kakšno je pravilno ime ustrezne metode ravno za ta lik. Prav tako je enostavneje, če dobimo še četrti tip objekta – v "glavno" kodo ni potrebno posegati –

naredimo le novo metodo z istim imenom (*Ploscina*) in s parametrom, katerega tip je novi objekt. Klic je še vedno *Ploscina(bla)*!

TABELE OBJEKTOV

Rekli smo že, da smo s tem, ko smo sestavili nov razred, sestavili dejansko nov tip podatkov. Ta je "enakovreden" v C# in njegovim knjižnicam vgrajenim tipom. Torej lahko naredimo tudi tabelo podatkov, kjer so ti podatki objekti.

Denimo, da smo sestavili razred *Zajec*

```
public class Zajec
{
    public string serijska;
    public bool spol;
    public double masa;
}
```

Zaradi enostavnosti ga ne bomo opremili s konstruktorji, saj to nič ne spremeni naše razlage.

Sedaj pišemo program, v katerem bomo potrebovali 100 objektov tipa *Zajec* (denimo, da pišemo program za upravljanje farme zajcev). Ker bomo z vsemi zajci (z vsemi objekti tipa *Zajec*) počeli enake operacije, je smiselno, da uporabimo tabelo.

```
Zajec[] tabZajci;
```

Spemenljivka *tabZajci* nam označuje torej tabelo, v kateri hranimo zajce. A bodimo tokrat še zadnjič pri izražanju zelo natančni. Natančno rečeno nam spremenljivka *tabZajci* označuje *naslov*, kjer bomo ustvarili tabelo, v kateri bomo hranili naslove objektov tipa *Zajec*. Ko torej napišemo,

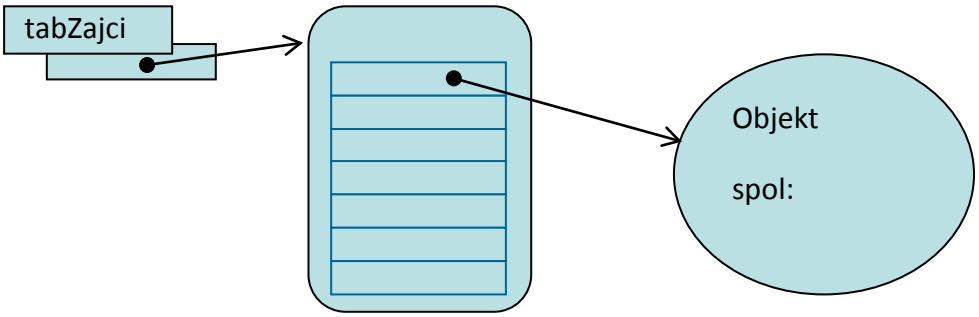
```
tabZajci = new Zajec[250];
```

smo s tem ustvarili tabelo velikosti 250. Kje ta tabela je, smo shranili v spremenljivko *tabZajci*. V tej tabeli lahko hranimo podatek o tem, kje je objekt tipa *Zajec*. V tem trenutku še nimamo nobenega objekta tipa *Zajec*, le prostor za njihove naslove. Šele ko napišemo,

```
tabZajci[0] = new Zajec();
```

smo s tem ustvarili novega zajca (nov objekt tipa *Zajec*) in podatke o tem, kje ta novi objekt je (naslov), shranili v spremenljivko *tabZajci[0]*.

Naslov tabele *Tabela z naslovi objektov*



Slika 15: Tabela objektov v pomnilniku

Seveda pa bomo še vedno govorili, da imamo zajca shranjenega v spremenljivki *tabZajci[0]*.

Zgoščenka

Pesem na zgoščenki je predstavljena z objektom razreda *Pesem*:

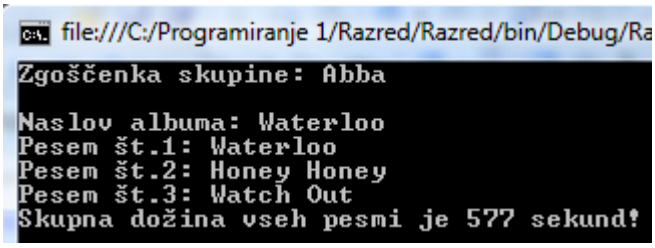
```
public class Pesem
{
    public string naslov;
    public int minute;
    public int sekunde;
    public Pesem(string nasl, int min, int sek)
    {
        naslov = nasl; minute = min; sekunde = sek;
    }
}
```

Objekt *new Pesem("Echoes",15,24)* predstavlja pesem “Echoes”, ki traja 15 minut in 24 sekund. Sestavimo razred *Zgoscenka*, ki vsebuje naslov zgoščenke, ime izvajalca in tabelo pesmi na zgoščenki. Razredu *Zgoscenka* bomo dodali še objektno metodo *Dolzina()*, ki vrne skupno dolžino vseh pesmi na zgoščenki, izraženo v sekundah.

```
public class Zgoscenka
{
    public string avtor;
    public string naslov;
    public Pesem[] seznamPesmi;  // Napoved tabele pesmi
    // s konstruktorjem določimo avtorja, naslov in vse pesmi
    public Zgoscenka(string avtor, string naslov, Pesem[] seznamPesmi)
    {
        this.avtor = avtor;
        this.naslov = naslov;
```

```
        this.seznamPesmi = new Pesem[seznamPesmi.Length];  /* inicializacija
tabele pesmi*/
        for (int i = 0; i < seznamPesmi.Length; i++)
        {
            // tabelo pesmi določimo s pomočjo parametra (tabele) seznamPesmi
            this.seznamPesmi[i] = new Pesem(seznamPesmi[i].naslov,
seznamPesmi[i].minute,
                                                    seznamPesmi[i].sekunde);
        }
    }
    public int Dolzina() // metoda za izračun skupne dolžine vseh skladb
    {
        int skupaj = 0;
        for (int i = 0; i < seznamPesmi.Length; i++)
            skupaj = skupaj + seznamPesmi[i].minute * 60 +
seznamPesmi[i].sekunde;
        return skupaj;
    }
}

static void Main(string[] args)
{
    Zgoscenka CD=new Zgoscenka("Abba","Waterloo",new Pesem[] {new
Pesem("Waterloo", 3, 11), new Pesem("Honey Honey", 3, 4),
new Pesem("Watch Out", 3, 22)});
    // Izpis albuma
    Console.WriteLine("Zgoščenka skupine: " + CD.avtor + "\n\nNaslov albuma:
" + CD.naslov);
    for (int i = 0; i < CD.seznamPesmi.Length; i++)  /*izpišemo celoten
seznam (tabelo) pesmi*/
        Console.WriteLine("Pesem št." + (i + 1) + ": " +
CD.seznamPesmi[i].naslov);
    /*pokličemo še metodo Dolzina objekta CD, ki nam vrne skupno dolžino vseh
skladb*/
    Console.WriteLine("Skupna dožina vseh pesmi je " + CD.Dolzina() + "
sekund!");
}
```

Izpis:



Slika 16: Razred *Zgoscenka*

DOSTOP DO STANJ OBJEKTA

Možnost, da neposredno dostopamo do stanj/lastnosti objekta ni najboljši! Glavni problem je v tem, da na ta način ne moremo izvajati nobene kontrole nad pravilnostjo podatkov o objektu! Tako lahko za število prebivalcev razreda *Drzava* napišemo npr.

```
d1.prebivalci = -400;
```

Ker ne moremo vedeti, ali so podatki pravilni, so vsi postopki (metode) po nepotrebnem bolj zapleteni oziroma so naši programi bolj podvrženi napakam. Ideja je v tem, da naj objekt sam poskrbi, da bo v pravilnem stanju. Seveda moramo potem tak neposreden dostop do spremenljivk, ki opisujejo objekt, preprečiti.

Dodatna težava pri neposrednem dostopu do spremenljivk, ki opisujejo lastnosti objekta, se pojavi, če moramo kasneje spremeniti način predstavitve podatkov o objektu. S tem bi "podrli" pravilno delovanje vseh starih programov, ki bi temeljili na prejšnji predstavitvi.

Torej bomo, kot smo omenjali že v uvodu, objekt res imeli za "črno škatlo", torej njegove notranje zgradbe ne bomo "pokazali javnosti". Zakaj je to dobro? Če malo pomislimo, uporabnika razreda pravzaprav ne zanima, kako so podatki predstavljeni. Če podamo analogijo z realnim svetom: ko med vožnjo avtomobila prestavimo iz tretje v četrto prestavo, nas ne zanima, kako so prestave realizirane. Zanima nas le to, da se stanje avtomobila (prestava) spremeni. Ko kupimo nov avto, nam je načeloma vseeno, če ima ta drugačno vrsto menjalnika, z drugačno tehnologijo. Pomembno nam je le, da se način prestavljanja ni spremenil, da še vedno v takih in drugačnih okoliščinah prestavimo iz tretje v četrto prestavo.

S "skrivanjem podrobnosti" omogočimo, da če pride do spremembe tehnologije, se za uporabnika stvar ne spremeni. V našem primeru programiranja v C# bo to pomenilo, da če spremenimo razred (popravimo knjižnico), se za uporabnike razreda ne bo nič spremenilo.

Denimo, da so v novi verziji C# spremenili način hranjenja podatkov za določanje naključnih števil v objektih razreda *Random*. Ker dejansko nimamo vpogleda (neposrednega dostopa) v te spremenljivke, nas kot uporabnike razreda *Random* ta sprememba nič ne prizadene. Programe še vedno pišemo na enak način, kličemo iste metode. Skratka – ni potrebno spreminjati programov, ki razred *Random* uporabljajo. Še posebej je pomembno, da programi, ki so delovali prej, še vedno delujejo (morda malo bolje, hitreje, a bistveno je, da delujejo enako).

Avto

Napišimo razred *Avto* s tremi polji (*znamka*, *letnik* in *registrska*). Začetne vrednosti objektov nastavimo s pomočjo konstruktorja. Polje *letnik* naj bo zasebno, zato napišimo metodo, ki bo letnik spremenila le v primeru, da bo le-ta v smiselnih mejah. Napišimo še metodo za izpis podatkov o določenem objektu.

```
public class Pesem
{
    public string naslov;
    public int minute;
    public int sekunde;
    public Pesem(string nasl, int min, int sek)
    {
        naslov = nasl; minute = min; sekunde = sek;
    }
}

public class Avto
{
    public string znamka;
    /* polje letnik  je zasebno, zato ga lahko določimo le s konstruktorjem,
spremenimo pa le z metodo SpemeniLetnik*/
    private int letnik;
    public string registrska;

    // konstruktor
    public Avto(string zn, int leto, string stevilka)
    {
        znamka = zn;
        letnik = leto;
        registrska = stevilka;
    }

    public bool SpremeniLetnik(int letnik)
    {
        if ((2000 <= letnik) && (letnik <= 2020))
        {
            this.letnik = letnik;
            return true;  // leto je smiselno, popravimo stanje objekta in
                          //vrnemo true
        }
        return false; //leto ni smiselno, zato ne spremenimo nič,vrnemo false
    }
    // metoda za izpis podatkov o določenem objektu
    public override string ToString()
    {
        return ("Znamka: " + znamka + ", Letnik: " + letnik + ", Registrska
številka: "
                  + registrska);
    }
}
static void Main(string[] args)
{
    // novemu objektu nastavimo začetne vrednosti preko konstruktorja
    Avto novAvto = new Avto("Citroen", 1999, "KR V1-02E");
    Console.WriteLine(novAvto.ToString()); // v izpisu bo letnik  1999
```

```
    novAvto.SpremeniLetnik(208); // letnik 208 NI dovoljen, objektu se letnik
                                 // NE spremeni
    Console.WriteLine(novAvto.ToString());  // v izpisu bo letnik ostal 1999
    novAvto.SpremeniLetnik(2008); // letnik 2008 JE dovoljen, objektu se
                                  //letnik spremeni
    Console.WriteLine(novAvto.ToString()); // v izpisu bo letnik 2008
    Console.ReadKey();
}
```

V programu opazimo novo neznano besedo – **override**. Uporabiti jo moramo zaradi "**dedovanja**". Zaenkrat moramo vedeti o dedovanju le to, da smo z dedovanjem avtomatično pridobili metodo *ToString()*. To je razlog, da je ta metoda vedno na voljo v vsakem razredu, tudi če je ne napišemo. Če želimo napisati svojo metodo, ki se imenuje enako kot podedovana metoda, moramo napisati besedico *override*. S tem "povozimo" obstoječo metodo.

Seveda bi lahko napisali tudi neko drugo metodo, na primer *Opis*, ki bi prav tako vrnila niz s smiselnim opisom objekta.

Uporabnikom lahko torej s pomočjo zasebnih polj (*private*) preprečimo, da "kukajo" v zgradbo objektov ali pa da jim določajo nesmiselne vrednosti. Če bodo želeli dostopati do lastnosti (podatkov) objekta (zvedeti, kakšni podatki se hranijo v objektu) ali pa te podatke spremeniti, bodo morali uporabljati metode. In v teh metodah bo sestavljavec razreda lahko poskrbel, da se s podatki "ne bo kaj čudnega počelo". Potrebujemo torej:

- Metode za dostop do stanj (za dostop do podatkov v objektu).
- Metode za nastavljanje stanj (za spreminjanje podatkov o objektu).

Prvim pogosto rečemo tudi "*get*" metode (ker se v razredih, ki jih sestavljajo angleško usmerjeni pisci, te metode pogosto pričnejo z *get* (Daj)). Druge pa so t.i. "*set*" metode (*set* / nastavi). Več o teh metodah pa v poglavju *Lastnost/Property*.

Razred Tocka

Kreirajmo razred *Tocka*, z dvema zasebnima poljema, koordinatama *x* in *y*. Napišimo tudi dva konstruktorja: privzetega in še enega za nastavljanje vrednosti koordinat. Ker sta koordinati zasebni (nimamo neposrednega dostopa), napišimo še metodi za spreminjanje vrednosti vsake od koordinat. Dodali bomo še metodo za izračun razdalje med dvema točkama.

```
class Tocka
{
    private int x, y;  // koordinati točke sta zasebni polji
    public Tocka()  // osnovni konstruktor
    {
        x = 0;
        y = 0;
```

```
    }
    public Tocka(int x, int y) // preobteženi konstruktor
    {
        this.x = x;
        this.y = y;
    }
    public void SpremeniX(int x) // metoda za spreminjanje koordinate x
    {
        this.x = x;
    }
    public void SpremeniY(int y) // metoda za spreminjanje koordinate y
    {
        this.y = y;
    }
    public double RazdaljaOd(Tocka druga) // metoda za izračun razdalje med
                                          //dvema točkama
    {
        int xRazd = x - druga.x;
        int yRazd = y - druga.y;
        return Math.Sqrt(Math.Pow(xRazd, 2) + Math.Pow(yRazd, 2));
    }
}
```

```
static void Main(string[] args)
{
    Tocka tA = new Tocka();   // Klic konstruktorja brez parametrov
    Tocka tB = new Tocka(6, 8);  //Klic konstruktorja z dvema parametroma
    double razdalja = tA.RazdaljaOd(tB);//klic metode za izračun razd. A do B
    Console.WriteLine("Razdalja točke A od točke B je enaka " + razdalja + "
enot!");
    tB.SpremeniX(4);  // točki B spremenimo prvo koordinato
    tB.SpremeniY(3);  // točki B spremenimo drugo koordinato
    Console.WriteLine("Razdalja točke A do točke B je  " + tA.RazdaljaOd(tB)
+ " enot!");
    Console.ReadKey();
}
```

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 katerega koli

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 nekako takole:

```
class Math
{
    public static double Sqrt(double d)
   {
         . . .
   }
}
```

Zapomnimo pa si, da statično metodo ne kličemo tako kot objektno. Kadar definiramo statično metodo, le-ta **nima** dostopa do katerega koli 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.

Razred Bla

Napišimo razred *Bla* in v njem statično metodo, katere naloga je le ta, da vrne *string*, v katerem so zapisane osnovni podatki o tem razredu

```
class Bla
{
    public static string Navodila() // Statična metoda
    {
        string stavek = "Poklicali ste statično metodo razreda Bla!";
        return stavek;
    }
}
static void Main(string[] args)
```

```
{
    // Zato, da pokličemo statično metodo oz. statično polje NE POTREBUJEMO
OBJEKTA!!!
    // Primer klica npr. v glavnem programu
    Console.WriteLine(Bla.Navodila()); // Klic statične metode
}
```

STATIČNA POLJA

Tako, kot obstajajo statične metode, obstajajo tudi statična polja. Včasih je npr. potrebno, da imajo vsi objekti določenega razreda dostop do istega polja. To lahko dosežemo le tako, da tako polje deklariramo kot statično. Za dostop do statičnega polja ne potrebujemo objektov, saj do takega polja dostopamo preko imena razreda.

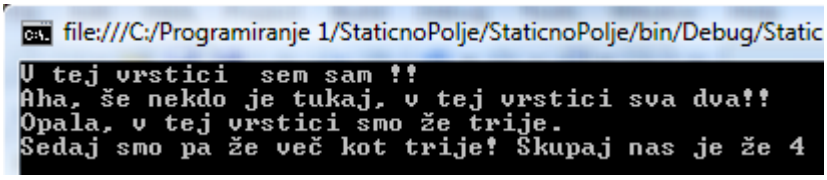
Razred Nekaj

V razredu *Nekaj* bomo prikazali definicijo in uporabo statičnega polja.

```
class Nekaj
{
    static int stevilo = 0;   // Statično polje
    public Nekaj()          //  Konstruktor
    {
        /* ko se izvede konstruktor (ob kreiranju novega objekta), se
statično polje poveča polje stevilo torej šteje živeče objekte*/
        stevilo++;
    }
    public void Pozdrav()
    {
        if (stevilo == 1)
            Console.WriteLine("V tej vrstici  sem sam !!");
        else if (stevilo == 2)
            Console.WriteLine("Aha, še nekdo je tukaj, v tej vrstici sva
dva!!");
        else if (stevilo == 3)
            Console.WriteLine("Opala, v tej vrstici smo že trije.");
        else
            Console.WriteLine("Sedaj smo pa že več kot trije! Skupaj nas je
že " + stevilo);
    }
}
static void Main(string[] args)
{
    Nekaj a = new Nekaj();  // konstruktor za a (prvi objekt tipa Nekaj)
```

```
    a.Pozdrav();
    Nekaj b = new Nekaj();  // konstruktor za b (drugi objekt tipa Nekaj)
    b.Pozdrav();
    Nekaj c = new Nekaj();  // konstruktor za c (tretji objekt tipa Nekaj)
    c.Pozdrav();
    Nekaj d = new Nekaj();  // konstruktor za d (četrti objekt tipa Nekaj)
    d.Pozdrav();
    Console.ReadKey();
}
```

Izpis:



Slika 17: Štejemo živeče objekte

LASTNOST/PROPERTY

Polja so znotraj razreda običajno deklarirana kot zasebna (*private*). Vrednosti smo jim doslej prirejali tako, da smo zapisali enega ali več konstruktorjev, ali pa smo napisali posebno javno metodo (imenovali smo jo *get/set* metodo) za prirejanje vrednosti polj. Ostaja pa še tretji način, ki je namenjen izkušenim programerjem – 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 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**: *get* mora vrniti vrednost, ki mora biti istega tipa kot lastnost (seveda pa mora biti lastnost istega tipa kot polje, kateremu je namenjena), v *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.

Sintakso lastnosti prikažimo na primeru razreda *Demo*:

```
class Demo
{
    private int polje;  //zasebno polje
    //POZOR! Za imenom lastnosti NI oklepajev tako kot pri metodah
    public int polje_Lastnost //deklaracija lastnosti(property) razreda Demo
    {
        get  //metoda get za pridobivaje vrednosti lastnosti polje
```

```
        {
            return polje;
        }
        set//metoda set za prirejanje(nastavljanje) vrednosti lastnosti polje
        {
            polje = value;
        }
    }
}
```

Za posamezno polje lahko napišemo le eno lastnost, v kateri s pomočjo stavkov *get* ali *set* dostopamo oz. nastavljamo vrednosti posameznega polja.

Artikel

Deklarirajmo strukturo *Artikel* z dvema komponentama: *naziv* (javno polje) in *cena*(zasebno polje). Napišimo tudi ustrezen konstruktor, objektno metodo *Izpis*, za izpis podatkov o določenem artiklu, ter lastnost *Cena* za polje *cena*. V glavnem programu ustvarimo objekt tipa *Artikel*, mu nato spremenimo ceno in ga izpišimo!

```
class Artikel
{
    public string naziv;
    private double cena;
    public Artikel(string naziv, double cena)
    {
        this.naziv = naziv;
        this.cena = cena;
    }
    public double Cena  //lastnost oz. Property
    {
        get
        {
            return cena;
        }
        set
        {
            cena = value;
        }
    }
    public void Izpis() //objektna meoda
    {
        Console.WriteLine("Naziv artikla: " + naziv + ", cena: " + cena);
    }
}
```

```
static void Main(string[] args)
{
    Artikel A1 = new Artikel("Zvezek", 5.45);
    //Ker je naziv artikla javno polje ga lahko spremenimo
    A1.naziv="Zvezek - A4";
    //polje cena je zasebno, zato ga lahko spremenimo preko lastnosti Cena
    A1.Cena=6.77;
    //Izpis podatkov objekta A1
    A1.Izpis();
    Console.ReadKey();
}
```

Oseba

Deklarirajmo razred *Oseba* z dvema poljema (ime in priimek) ter ustrezen konstruktor. 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
{
    //dostopnosti NISMO napisali, kar pomeni, da sta polji ZASEBNI
    string priimek; //zasebno polje razreda Oseba
    string ime;     //zasebno polje razreda Oseba

    public Oseba(string ime, string priimek)  //konstruktor
    {
        this.priimek = priimek;
        this.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("Nelson", "Mandela");
    Console.WriteLine("Polno ime osebe je " + politik.PolnoIme);  /*Izpis:
Polno ime osebe je Neslon Mandela*/
    politik.PolnoIme = "France Prešeren";/uporaba lastnosti za spremembo obeh
polj, tako imena kot priimka*/
    Console.WriteLine("Polno ime osebe je " + politik.PolnoIme);  /*Izpis:
Polno ime osebe je France Prešeren*/

    Console.ReadKey();
}
```

Katerokoli od rezerviranih besed *get* ali *set* lahko znotraj lastnosti izpustimo in dobimo *write-only* ali *read-only* lastnost.

Instrukcije

Potrebujemo razred, ki bo hranil podatke o objektih tipa *Instrukcije* s komponentama *predmet* (zasebno polje) in *ure* - število opravljenih ur(zasebno polje). Razred naj ima tudi konstruktor in ustrezno lastnost. Na osnovi razreda *Instrukcije* napišimo še testni program, ki bo kreiral dva objekta razreda *Instrukcije*.

```
class Instrukcije
{
    private string predmet;
    public int ure;
    public Instrukcije(string predmet)
    {
        this.predmet = predmet;
        ure = 0;//niti ni potrebno, ker je vrednost 0 privzeta
    }
    public string Predmet //lastnost je Read_Only
    {
        get
        {
            return predmet;
        }
    }
}
//testni program
static void Main(string[] args)
{
    //nova objekta tipa Instrukcije
    Instrukcije I1 = new Instrukcije("MAT");
    Instrukcije I2 = new Instrukcije("ANJ");
```

```
    //določimo ure za oba objekta
    I1.ure = 5;
    I2.ure = 3;
    Console.WriteLine("Stanje:");
    //do imena predmeta lahko pridemo le preko lastnosti Predmet
    Console.WriteLine(I1.Predmet + ": " + I1.ure);//Izpis: MAT: 5
    Console.WriteLine(I2.Predmet + ": " + I2.ure);//Izpis: ANJ: 3
    Console.ReadKey();
}
```

DESTRUKTOR

Destruktor je metoda, ki počisti za objektom. Izvede se, ko objekt odmre (to pa je enkrat potem, ko se zaključi nek blok, ali pa ko se zaključi neka metoda, v kateri je bil objekt ustvarjen). Pokliče ga torej proces, ki se imenuje smetar (***garbage collector***), o katerem bo govora v naslednjem poglavju. Destruktor torej sprosti pomnilniški prostor objekta (prostor, ki ga zasedajo njegovi podatki).

Destruktor ima tako kot konstruktor, enako ime kot razred sam in nima tipa. Ne sprejema argumentov, za razliko od konstruktorja pa pred destruktorjem stoji še znak '~' (na SLO tipkovnici je to Alt Gr "1"). 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() //konstruktor
    {
       ...
    }

    ~ Krog() //destruktor: znak '~' na tipkovnici je  Alt Gr 1
    {
       ...
    }
}
```

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.

V naših programih običajno destruktorjev ne bomo pisali, ampak bomo sproščanje z objekti zasedenega pomnilnika v celoti prepustili procesu *garbage collector*.

GARBAGE COLLECTOR

Novo *instanco* (nov objekt) nekega razreda kreiramo s pomočjo rezervirane besede *new*. Za sproščanje pomnilnika, ki ga zasedejo objekti kreirani z operatorjem *new* pa 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.

S stališča programerja je uporaba besedice *new* pogosto že kar avtomatična – omogoči pač kreiranje novega objekta. Vendar pa je kreiranje objekta v resnici dvofazni proces. Najprej mora operacija *new* alocirati (zasesti) nekaj prostega pomnilnika na kopici – na ta del kreiranja novega objekta nimamo prav nobenega vpliva. Drugi del operacije *new* pa je v tem, da mora zasedeni pomnilnik pretvoriti v objekt – mora inicializirati objekt. To drugo fazo operacije *new* pa seveda lahko kontroliramo z uporabo konstruktorja. Ko je nov objekt kreiran, lahko do njegovih članov (polj, metod, ..) dostopamo z uporabo operatorja pika.

Iz nekega objekta lahko v nadaljevanju tvorimo poljubno število novih referenc, a vse te reference je potrebno slediti, oz. imeti nad njimi popoln nadzor. Če nek konkreten objekt izgine (zaključek bloka, zaključek metode, …), pa so lahko njegova polja še vedno dostopna. Življenjska doba objekta torej ni vezana na določeno referenčno spremenljivko. Objekt je lahko uničen šele tedaj, ko izginejo vse njegove reference.

Tako kot je dvofazni proces kreiranje novega objekta, je dvofazni proces tudi njegovo uničenje – je njegova zrcalna slika. Prvi del »čiščenja« opravimo s pisanjem *destruktorja*. Drugi del faze pa je v tem, da je potrebno ob uničenju objekta sprostiti (alocirati) del pomnilnika, ki ga je objekt zasedal in ga vrniti kopici v nadaljnjo prosto uporabo. Nad to drugo fazo, tako kot pri kreiranju objekta, nimamo neposrednega vpliva. Proces uničenja objekta in vračanje pomnilnika kopici je poznano pod imenom *garbage collection*.

V okolju Visual C# objektov nikoli ne moremo uničiti sami, saj za to opravilo ne obstaja nikakršna sintaksa (tako kot je npr. v C++ temu namenjena metoda *delete*). Kar nekaj pomembnih razlogov botruje temu, da nam C# ne omogoča eksplicitno uničenje objektov, saj bi v primeru, da bi bila odgovornost nad uničenjem objektov prepuščena nam, slej ko prej prišlo do ene od naslednjih situacij:

- pozabili bi uničiti nek objekt. To bi pomenilo, da se objektov destruktor ni izvedel, čiščenje pomnilnika se ni izvedlo, s tem pa pomnilnik na kopici, ki ga objekt zaseda, ne bi bil sproščen. Na ta način bi kmalu ostali brez pomnilnika.

- poskusili bi uničiti aktivni objekt, kar pa bi bila katastrofa za vse objekte z referenco na ta uničeni objekt.
- isti objekt bi lahko poskušali uničiti večkrat, kar bi bila prav tako lahko katastrofalno, odvisno od destruktorja.

Zgornji problemi so torej nesprejemljivi, zaradi tega je za uničevanje objektov odgovoren proces *garbage collector*. Ta proces nam zagotavlja, da bo vsak objekt zagotovo uničen, obenem pa se bo izvedel njegov destruktor.

Če napišemo *destruktor*, se bo ta zagotovo izvedel, ne vemo pa kdaj, saj o uničenju objektov odloča *garbage collector*. Ko se zaključi nek program, bodo zagotovo uničeni tudi vsi v programu kreirani objekti. Obenem proces zagotavlja, da bo vsak objekt uničen natanko enkrat, uničen pa bo samo takrat, ko postane nedostopen – to pa pomeni tedaj, ko ne obstaja več nobena referenca na ta objekt.

Ostane pa še vprašanje, kdaj pa se čiščenje (*garbage collection*) sploh izvede. Prav gotovo je to tedaj, ko objekt ni več potreben, to pa se ne zgodi vedno neposredno za tem, ko objekta ne potrebujemo več (npr. neposredno po zaključku nekega bloka). Zažene se tedaj, ko zazna, da sistemu začne primanjkovati pomnilnika, oz. da je prezaposlen. Tedaj sprosti pomnilnik vseh tistih objektov, ki niso več v funkciji.

Teta Rozi in njene sadike

Teta Rozi je izvrstna v vzgoji vrtnih sadik. Odločila se je, da bo sadike vzgajala za prodajo. Rada pa bi svojo prodajo vodila s pomočjo programa v C#. Naredi ji razred *Sadika*, kjer bo hranila štiri lastnosti posamezne sadike: naziv, število sadik, ceno vzgoje in prodajno ceno. Poleg potrebuje tudi ustrezne *get/set* metode, kjer lahko spreminja vse lastnosti.

```
using System.Collections; //uporabili bomo netipizirano zbirko
//naštevni tip za vse vrste sadik
public enum VrsteSadik{Endivija,Koleraba,Radič,Paradižnik,Paprika};
public class Sadika
{
    public static int skupajSadik = 0; //statično polje
    public VrsteSadik vrsta;
    public int stevilo;  //število sadik
    double cenaVzgoje, prodajnaCena; //zasebni polji
    public Sadika() //privzeti konstruktor, ki pa ga ne bomo uporabili
    { }//ker je privzeti konstruktor prazen, so numerične vrednosti 0
    //preobteženi konstruktor
    public Sadika(VrsteSadik vrsta,int stevilo,double cenaVzgoje,double
prodajnaCena)
    {
```

```
        this.vrsta=vrsta;
        this.stevilo=stevilo;
        this.cenaVzgoje=cenaVzgoje;
        this.prodajnaCena=prodajnaCena;
        skupajSadik = skupajSadik + stevilo;/*pri vzgoji se skupno število
sadik poveča*/
    }
    public Sadika(VrsteSadik vrsta, int stevilo, double trenutnaCena)
    {
        this.vrsta = vrsta;
        this.stevilo = stevilo;
        prodajnaCena = trenutnaCena;
        skupajSadik = skupajSadik - stevilo;/*pri prodaji se skupno število
sadik zmanjša*/
    }
    ~Sadika() //DEMO destruktor - izvede se ob uničenju objekta
    {}
    public double Cena //lastnos polja cenaVzgoje
    {
        get { return cenaVzgoje; }
        set
        {
            if (Cena>=0)  //ceno vzgoje nastavimo le, če je pozitivna
                cenaVzgoje = value;
        }
    }
    public double Prodajna  //lastnost polja prodajnaCena
    {
        get { return prodajnaCena; }
        set
        {
            if (Prodajna >= 0)//prodajno ceno nastavimo le, če je pozitivna
                prodajnaCena = value;
        }
    }
    public void Izpis()//objektna metoda za izpis sadike
    {
        Console.WriteLine("Naziv sadike: " + vrsta + ", število sadik: " +
stevilo);
        Console.WriteLine(" cena vzgoje: "+cenaVzgoje+", prodajna cena:
"+prodajnaCena);
    }
    /*preobtežena metoda, ki izpiše naziv in število sadike le v primeru, da
je število komadov manjše od nekega vhodnega parametra*/
    public void Izpis(int komadi)
    {
            if (stevilo<komadi)
                Console.WriteLine("  Naziv sadike: " + vrsta + ", število
sadik: " + stevilo);
    }
```

```
}

static void Main(string[] args)
{
    Sadika[] Sadike = new Sadika[5];
    //inicializacija tabele s pomočjo konstruktorja: začetno stanje tete Rozi
    Sadike[0] = new Sadika(VrsteSadik.Endivija, 50, 0.12, 0.55);
    Sadike[1] = new Sadika(VrsteSadik.Koleraba, 40, 0.18, 0.77);
    Sadike[2] = new Sadika(VrsteSadik.Paprika, 30, 0.44, 1.12);
    Sadike[3] = new Sadika(VrsteSadik.Paradižnik, 50, 0.77, 2.33);
    Sadike[4] = new Sadika(VrsteSadik.Radič, 100, 0.09, 0.81);

    //izpis vseh podatkov o sadikah
    for (int i = 0; i < Sadike.Length; i++)
        Sadike[i].Izpis();
    //celotno prodajo shranjujemo v zbirko Promet
    ArrayList Promet = new ArrayList();
    //najprej je prodala 10 sadik paradižnika
    Sadika prodaja = new Sadika(VrsteSadik.Paradižnik, 10,
Sadike[3].Prodajna);
    Sadike[3].stevilo = Sadike[3].stevilo - prodaja.stevilo;/*zmanjšamo
število sadik na zalogi*/
    Promet.Add(prodaja);//sadike dodamo med prodane
    //nato pa še 30 sadik radiča
    prodaja = new Sadika(VrsteSadik.Radič, 30, Sadike[4].Prodajna);
    Sadike[4].stevilo = Sadike[4].stevilo - prodaja.stevilo;/*zmanjšamo
število sadik na zalogi*/
    Promet.Add(prodaja);//sadike dodamo med prodane
    //zaradi recesije je nato vse sadike podražila za 15%
    for (int i = 0; i < Sadike.Length; i++)
        Sadike[i].Prodajna = Math.Round(Sadike[i].Prodajna * 1.15,2);

    //nato pa prodala še 20 sadik endivije
    prodaja = new Sadika(VrsteSadik.Endivija, 20, Sadike[0].Prodajna);
    Sadike[0].stevilo = Sadike[0].stevilo - prodaja.stevilo;/*zmanjšamo
število sadik na zalogi*/
    Promet.Add(prodaja);//sadike dodamo med prodane
    //izračunajmo njen izkupiček
    double cena = 0;
    for (int i = 0; i < Promet.Count; i++) //izračun skupne prodaje
    {
        /*ker je zbirka Promet netipizirana, je potrebna eksplicitna
konverzija (casting) */
        Sadika prodana = (Sadika)Promet[i];
        cena = cena + prodana.stevilo * prodana.Prodajna;
    }

    Console.WriteLine("Skupna prodaja: "+cena+" EUR"); /*Izpis: Skupna
prodaja: 60,2 EUR*/
    //Izpis: Na zalogi še 210 sadik
```

```
    Console.WriteLine("Na zalogi še " + Sadika.skupajSadik+" sadik!");
    Console.WriteLine("\nIzpis sadik, ki jih je na zalogi manj od 40!");
    for (int i = 0; i < Sadike.Length; i++)
        Sadike[i].Izpis(40);

    Console.ReadKey();
}
```

V nalogi smo uporabili kar nekaj objektov: tabela sadik in nato še zbirka sadik. Vse te objekte bo iz pomnilnika počistil smetar!

POVZETEK

V poglavju so zapisane le osnove razredov in objektov, ter nekaj osnovnih primerov, ki omogočajo kasnejši vpogled v ključna pojma objektno orientiranega programiranja – *dedovanje* in *polimorfizem*. Več o tem pa je zapisano v literaturi z nazivom *Načrtovanje programskih aplikacij – NPA*. Seveda pa je za to, da bo objekten pristop k programiranju postal domač, potrebno napisati kar največ programov.

VAJE

1. Janezek se v šoli uči računati ploščino in obseg pravilnega šestkotnika. Ker se mu ne ljubi tako veliko nalog računati v zvezek, bi si rad sestavil razred, ki mu bo vse to izračunal samo z vnosom dolžine stranice.

2. Sestavi razred *Letalo*, ki naj vsebuje naslednje lastnosti: doseg, maksimalna višina, ki jo lahko letalo doseže, maksimalna hitrost, ki jo lahko letalo doseže, tip letala in nosilnost letala. Napiši ustrezne konstruktorje (tudi praznega) in *get* in *set* metode. V glavnem programu kreiraj nekaj objektov in jim določi vrednosti polj.

3. Napiši razred *Kolo*, s katerim predstavite kolesa. Vsako kolo ima neko število prestav, v kateri prestavi se trenutno nahaja kolo, barvo, trenutno hitrost, maksimalno hitrost in za koliko ljudi je namenjeno. Napiši osnovni konstruktor (sam določi privzete vrednosti), dva dodatna konstruktorja in vsaj 5 javnih metod. Uporabi ta razred v nekem primeru.

4. Kreiraj razred Kocka z enim javnim poljem (rob kocke) in s tremi metodami: metodo, ki izračuna in vrne prostornino kocke, metodo, ki izračuna in vrne površino kocke, ter metodo ki vrne string s podatki o konkretni kocki (string naj ima takole obliko: Kocka: rob a= xx, Prostornina: xxx.xx, Površina: xxx.xx.) Nato ustvari nov objekt tipa *Kocka*, določi rob kocke (npr. preberi ga preko tipkovnice) in nato na ustrezen način preizkusi vse tri metode razreda Kocka!

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

5. Deklariraj razred *Complex*, ki naj predstavlja kompleksno število. Razred naj ima dva konstruktorja (privzetega in še enega za nastavljanje realne in imaginarne komponente). Napiši še metodo za izpis kompleksnega števila, nato pa kreiraj tabelo 10 kompleksnih števil in jo inicializiraj z naključnimi vrednostmi obeh komponent.

6. Kreiraj razred *Polinom*, ki hrani polinome do stopnje dva ($ax^2 + bx + c$). Polinom naj ima tri polja (koeficiente polinoma), pozna pa naj tudi tri metode: metodo za seštevanje dveh polinomov, metodo, ki vrne vrednost polinoma v poljubni točki in metodo *ToString*, ki ta polinom predstavi (vrne) v obliki stringa.

7. Napiši razred *Zival*, ki vsebuje tri zasebna polja (število nog, vrsta in ime) in metode za nastavljanje in spreminjanje le-teh. Vsebuje naj tudi konstruktor brez parametrov, ki postavi spremenljivke na smiselno začetno vrednost. V razredu napiši metodo *override public string ToString()*, ki naj izpiše podatke o objektih v obliki: *#Ime* je vrste *#vrsta* in ima *#steviloNog* nog. Pri tem seveda zamenjaj *#vrsta* in *#steviloNog* z vrednostmi v istoimenskih spremenljivkah objekta. Nato napišite testni program, kjer ustvariš vsaj 5 živali in jim nastaviš smiselno vrsto in imena ter število nog. Izpiši jih!

8. Napiši razred *Kosarka*, za spremljanje košarkaške tekme. Voditi moraš število prekrškov za vsakega tekmovalca (12 igralcev), število doseženih točk (posebej 1 točka, 2 točki in 3 točke), ter metodo za izpis statistike tekme. Doseganje košev in prekrškov realiziraj preko metod *ZadelProstiMet*(), *ZadelZa2Tocki*, *ZadelZa3Tocke* in *Prekrsek*.

9. Sestavi razred *Pacient*, ki ima tri komponente: *ime* in *priimek* naj bosta obe *public*, *krvna_skupina* pa *private*, vse tri pa tipa string. Napiši metodo, ki vse tri komponente nastavi na "NI PODATKOV". Napiši metodo, ki sprejme vse tri podatke in ustrezno nastavi komponente. Z metodo *public string ToString()* naj se izpišejo podatki o pacientu (ime, priimek, krvna skupina). Ker je *krvna_skupina* zasebno/private polje, napiši še ustrezno *lastnost/property* ali pa metodo, ki sprejme podatek o krvni skupini in to vrednost priredi polju krvna_skupina.

10. Sestavi razred *Majica*, ki hrani osnovne podatke o majici: velikost (število med 1 in 5), barvo (poljuben niz) in ali ima majica kratke ali dolge rokave. Vse spremenljivke razreda morajo biti privatne, razred pa mora tudi poznati metode za nastavljanje in branje teh vrednosti. Podpisi teh metod naj bodo: *public int VrniVelikost()* ter *public void SpremeniVelikost(int velikost)*, *public string VrniBarvo()* ter *public void SpremeniBarvo(string barva)*, *public bool ImaKratkeRokave()* ter *public void NastaviKratkeRokave(boolean imaKratkeRokave).* Metoda za nastavljanje velikosti majice naj poskrbi tudi za to, da bo velikost vedno ostala znotraj sprejemljivih meja! Če uporabnik poskusi določiti napačno velikost, naj se obdrži prejšnja vrednost.

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Farma zajcev

Z znanjem, ki smo ga pridobili v poglavju o razredih in objektih sedaj napišimo program, za vodenje farme zajcev, ki smo si ga zadali na začetku! Voditi hočemo natančno evidenco o vsakem zajcu posebej (serijska številka, spol, masa), kakor tudi o celotni "farmi" zajcev želimo imeti pregled nad številčnim stanjem in maso vseh zajcev, pa seveda voditi evidenco o prodajni vrednosti zajcev, najtežjem zajcu, ipd. Nalogo smo rešili s pomočjo zbirke, lahko pa seveda uporabili tudi tabelo! V rešitvi je najprej napisan razred, nato glavni program, na koncu pa še statične metode, klicane v glavnem programu.

```
public class Zajec
{
    public static double cenaZaKg = 3.56;
    public string serijska;
    private bool spol;
    private double masa;

    //Konstruktor za nastavljanje vseh treh polj
    public Zajec(string serijskaStev, bool spol, double masa)
    {
        this.serijska = serijskaStev;
        this.masa=masa;
        this.spol = spol;
    }

    public bool SpremeniMaso(double novaMasa)
    {
        if ((0 < novaMasa) && (novaMasa <= 10))//smiselna masa je med 0 in 10
        {
            masa = novaMasa;
            return true; // sprememba uspela
        }
        return false; // v nasprotnem primeru mase NE spremenimo
    }

    public double Masa
    {
        get {  return masa;}
        set
        {
            // smiselna nova teža je le med 0 in 10 kg
            if ((0 < value) && (value <= 10))
            {
                masa = value;
            }
        }
```

```
    }

    public bool Spol
    {
        /*ker se spol naknadno NE spremeni, spreminjanje spola (set) sploh ne
ponudimo uporabniku!*/
        get{ return spol;}
    }

    public override string ToString() //izpis podatkov o posameznem zajcu
    {
        string sp ="samec";
        if (spol==false)
            sp="samica";

        return "Zajec: " + serijska + ", masa: " + masa + ", spol: " + sp ;
    }
    public double Vrednost() //metoda vrne vrednost zajca
    {
        return Math.Round(cenaZaKg * masa,2);
    }
    public double Vrednost(bool sp) /*preobtežena metoda vrne vrednost za
posamezni spol*/
    {
        if (spol == sp) //spol se ujema
            return Math.Round(cenaZaKg * masa, 2);
        else return 0;
    }
}

public static void Main(string[] ar)
{
    ArrayList Farma = new ArrayList();//zbirka zajcev
    string izbira;
    do
    {
        Console.WriteLine("\nV-Vnos začetnega stanja, I-Izpis stanja na
farmi, S-Skupno število zajcev");
        Console.Write("P-Povečaj maso,          C-Cenik, sprememba cene  M-
Sprememba mase\n\nIzbira: ");
        izbira = Console.ReadLine().ToUpper();
        switch (izbira)
        {
            case "V": Zacetno(Farma); break;
            case "I": Izpis(Farma); break;
            case "S": Skupaj(Farma); break;
            case "P": Povecaj(Farma); break;
            case "C": Cenik(Farma); break;
            case "M": Sprememba(Farma); break;
        }
```

```
    }
    while (izbira != "K");
}

private static void Sprememba(ArrayList Farma)
{
    try
    {
        Console.Write("Serijska številka zajca: 1238-12-");
        int stev=Convert.ToInt32(Console.ReadLine());
        //Če ta številka obstaja, ponudimo vnos nove mase
        Zajec z;
        bool obstaja=false;
        for (int i=0;i<Farma.Count;i++)
        {
            z=(Zajec)Farma[i]; //element zbirke spremenimo v objekt
            if (z.serijska==("1238-12-"+stev))
            {
                obstaja=true;
                Console.WriteLine("Sedanja masa: " + z.Masa);
                Console.Write("Nova masa (v mejah med 0 in 10 kg): ");
                double masa = Convert.ToDouble(Console.ReadLine());
                z.Masa = masa;
                Farma[i] = z;
                break;
            }
        }
        if (!obstaja) //če pa serijska številka ne obstaja
        {
            Console.WriteLine("Zajec s to serijsko številko ne obstaja!");
        }

    }
    catch { Console.WriteLine("Napaka pri vnosu!");}
}

private static void Cenik(ArrayList Farma)
{
    double cenaSkupaj = 0, cenaSamci = 0, cenaSamice = 0;
    for (int i = 0; i < Farma.Count; i++)
    {
        Zajec z = (Zajec)(Farma[i]);//elem. zbirke spremenimo v objekt Zajec
        cenaSkupaj += z.Vrednost();
        cenaSamci += z.Vrednost(true);
        cenaSamice += z.Vrednost(false);
    }
    Console.WriteLine("Skupna vrednost vseh zajcev na farmi: " + cenaSkupaj+"
EUR");
    Console.WriteLine("Skupna vrednost vseh samcev: " + cenaSamci+" EUR");
    Console.WriteLine("Skupna vrednost vseh samic: " + cenaSamice+" EUR");
```

```
    //cenaZaKg je statična spremenljivka, zato do nje dostopamo preko razreda
    Console.Write("Sedanja cena za 1 kg mesa: " + Zajec.cenaZaKg + "
EUR.\nNova cena (brez spremembe <Enter>): ");
    try
    {
        Zajec.cenaZaKg = Convert.ToDouble(Console.ReadLine());
    }
    catch{}

}

private static void Zacetno(ArrayList Farma)
{
    try  //varovalni blok
    {
        /*ob dobavi moramo zajcem določiti serijske številke,
določiti/prebrati masa vseh zajcev, polovica je samcev, polovica samic*/
        Console.Write("Število zajcev: ");
        int st = Convert.ToInt32(Console.ReadLine());
        Console.Write("Masa v kg: ");
        double masa = Convert.ToDouble(Console.ReadLine());
        bool spol;
        for (int i = 0; i < st; i++)
        {
            string serijska = "1238-12-" + i;
            if (i < (st / 2))
                spol = true; //prva polovica so samci
            else
                spol = false; //druga polovica so samice
            Zajec nov = new Zajec(serijska, spol, masa);
            Farma.Add(nov);
        }
    }
    catch
    { Console.WriteLine("NAPAKA PRI VNOSU PODATKOV!\n\n"); }
}

//metoda izpiše podatke o vseh zajcih na farmi
private static void Izpis(ArrayList Farma)
{
    for (int i = 0; i < Farma.Count; i++)
    {
        Console.WriteLine(((Zajec)Farma[i]).ToString());/*klic objektne
metode za izpis podatkov o posameznem zajcu*/
    }
}

private static void Skupaj(ArrayList Farma)
{
    double masa = 0;
```

```
    int najM = 0;//indeks zajca z največjo maso
    for (int i = 0; i < Farma.Count; i++)
    {
        Zajec z = (Zajec)(Farma[i]);//elem. zbirke spremenimo v objekt Zajec
        masa = masa + z.Masa;
        if (z.Masa>((Zajec)Farma[najM]).Masa) //če najdemo zajca z večjo maso
            najM = i;                         //si zapomnimo njegov indeks
    }
    Console.WriteLine("Skupno število zajcev na farmi je " + Farma.Count + ",
skupna masa v kg: " + masa);
    Console.WriteLine("Največja masa zajca: " + ((Zajec)Farma[najM]).Masa+"
kg: ");
    Console.ReadKey();
}

private static void Povecaj(ArrayList Farma)
{
    Console.WriteLine("Za koliko kg so se zajci zredili: ");
    double pov = Convert.ToDouble(Console.ReadLine());
    for (int i= 0;i<Farma.Count;i++)
    {
        Zajec z=((Zajec)Farma[i]); //element zbirke spremenimo v objekt Zajec
        z.Masa=z.Masa+pov; //objektu povečamo maso
        Farma[i]=z; //zapišemo ga nazaj na isto mesto v zbirko
    }
}
```

NADLEŽNI STARŠI

Denimo, da ste stari 16 let in imate zelo sitne starše, ki želijo brati vaša sporočila. Zato se s prijateljem dogovorita, da si bosta sporočila pošiljala zakodirana. Napisati morate program, ki bo sporočilo zakodiral. Zakodirano sporočilo naj bo sestavljeno najprej iz znakov, ki so bili v originalnem sporočilu na sodih mestih in nato iz znakov, ki so bili v originalnem sporočilu na lihih mestih. Tako originalno kot zakodirano sporočilo naj bo shranjeno v poljubni datoteki poljubne enote (disk, USB,..) ! Primer: Originalno sporočilo Greva na pijačo?, se zakodira v sporočilo Gean iaorv apjč?

DATOTEKE

UVOD

V vseh dosedanjih programih smo podatke brali s standardnega vhoda (tipkovnica) in jih pisali na standardni izhod (zaslon). Ko se je program zaključil, so bili vsi podatki, ki smo jih v program vnesli ali pa jih je zgeneriral program sam (npr. naključna števila), izgubljeni. Slej ko prej pa se pri delu s programi pojavi potreba po shranjevanju podatkov, saj jih le na ta način lahko pri ponovnem zagonu programa pridobimo nazaj in nadaljujemo z njihovo obdelavo.

Podatke shranjujemo (zapisujemo) v datoteke, iz datotek pa lahko podatke tudi pridobimo (beremo) nazaj − naučiti se moramo torej kako delati z datotekami.

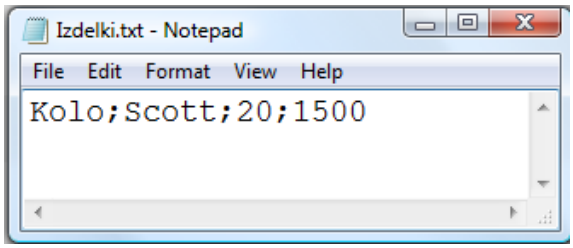
KAJ JE DATOTEKA

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

- **tekstovne** datoteke in
- **binarne** datoteke

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

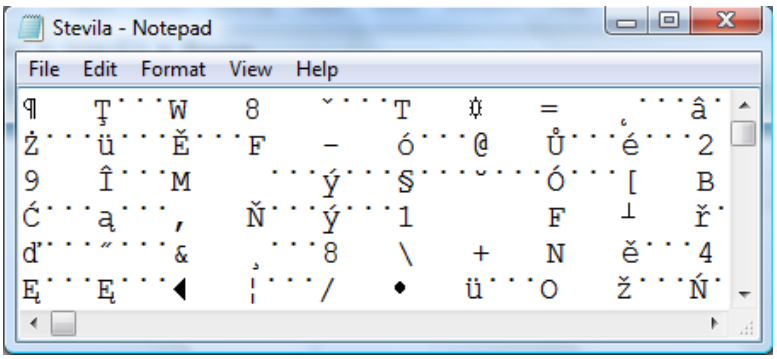
Na sliki je prikazan izgled vsebine tekstovne datoteke odprte z Beležnico (v datoteki je ena sama vrstica).



Slika 18: Tekstovna datoteka odprta z Beležnico

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

Binarne datoteke vsebujejo podatke zapisane v binarnem zapisu. Zato je vsebina le-teh datotek ljudem neberljiva (urejevalniki besedil običajno ne znajo prikazati posebnih znakov, namesto njih prikazujejo "kvadratke"). Tudi podatki v binarnih datotekah lahko vsebujejo znake tako, kot je to v tekstovnih datotekah, a so podatki med seboj ločeni s posebnimi znaki, zaradi česar je vsebina take datoteke, če jo odpremo z nekim urejevalnikom, skoraj neberljiva. Poleg tega binarne datoteke ne vsebujejo vrstic. Na sliki je prikazan izgled vsebine tekstovne datoteke, odprte z Beležnico (v datoteki NI vrstic).



Slika 19: Binarna datoteka odprta z beležnico

IMENA DATOTEK, IMENSKI PROSTOR

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

D:\olimpijada\Peking\atletika.txt

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

Imena imenikov so v operacijskem sistemu Windows ločena z znakom \. Kadar želimo, da je znak \ sestavni del niza, moramo napisati kombinacijo \\. Spomnimo se še, da v jeziku C# znak \ v nizu lahko izgubi posebni pomen, kadar pred nizom uporabimo znak @.

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

```
using System.IO;
```

Z navedbo tega imenskega prostora poenostavimo klice metod za delo z datotekami in upravljanje z imeniki in potmi do datotek. V naslednji tabeli so prikazani glavni trije razredi 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.

Tabela 9: Razredi za delo z imeniki, datotekami in potmi do datotek.

Vse metode teh razredov so statične, zaradi česar jih lahko kličemo neposredno preko imena razreda in ne preko imen objektov.

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
GetDirectories(pot)	Pridobivanje imen map navedene poti
GetCreationTime(pot)	Pridobivanje datuma kreiranja mape
GetDirectoryRoot(pot)	Pridobivanje imena logične enote, na kateri se nahaja določena mapa
GetCurrentDirectory()	Pridobivanje poti do tekoče mape

Tabela 10: Najpomembnejše metode razreda *Directory*.

Metode razreda Directory

Preverimo, če že na disku C že obstaja mapa *Vaje C#*. Če še ne obstaja jo ustvarimo. Prikažimo uporabo nekaterih metod razreda *Directory*.

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

    /*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);
    try
    {
        //Skušamo pobrisati mapo C:\Vaje C#
        Directory.Delete(dir);
        Console.WriteLine("\nMapa c:Vaje C # uspešno pobrisana!");
    }
    catch { Console.WriteLine("\nBrisanje neuspešno, ker mapa vsebuje
datoteke"); }
    DateTime datum = Directory.GetCreationTime(@"c:\Windows");
    Console.WriteLine(@"Mapa C:\Windows je bila kreirana " +
datum.ToShortDateString());

    //Ugotovimo, katere podmape vsebuje mapa Program Files
    string[] mape = Directory.GetDirectories(@"C:\Program Files");
    foreach (string ime in mape)//izpišimo imena vseh datotek mape C:\Vaje C#
        Console.WriteLine(ime);
```

```
    string mapa = @"Program Files";
    Console.WriteLine("Mapa '" + mapa + "' se nahaja na disku " +
Directory.GetDirectoryRoot(@"c:\Program Files"));

    //ugotovimo in izpišimo celotno pot do tekoče mape
    string tekocamapa = Directory.GetCurrentDirectory();
    Console.WriteLine(tekocamapa);//izpis tekoče mape
    //Kreiranje novega imenika C# 2010 in v njem podimenika z imenom Datoteke
    string dir1 = "C: \\C# 2010\\Datoteke\\"; //ali @"C: \C# 2005\Datoteke\"

    if (!Directory.Exists(dir1))  //če imenik še ne obstaja, ga skreiramo
        Directory.CreateDirectory(dir1);
    Console.ReadKey();
}
```

Najpomembnejše metode razreda *Path*:

Metoda	Razlaga
GetFullPath(pot)	Pridobivanje celotne poti do datoteke

Tabela 11: Metoda razreda *Path*.

```
string ime = "Datoteka.txt";
//celotno pot do datoteke dobimo s pomočjo metode GetFullPath razreda Path
string celotnaPot = Path.GetFullPath(ime);
Console.WriteLine(celotnaPot);//izpis celotne poti do datoteke
```

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*).
CreateText(pot)	Kreiranje ali odpiranje tekstovne datoteke za pisanje. Parameter *pot* vsebuje pot do datoteke in njeno ime
OpenText(pot)	Odpiranje obstoječe tekstovne datoteke za branje. Parameter *pot* vsebuje pot do datoteke in njeno ime
OpenRead(pot)	Odpiranje obstoječe datoteke za branje. Parameter *pot* vsebuje pot do datoteke in njeno ime
OpenWrite(pot)	Odpiranje obstoječe datoteke za pisanje. Parameter *pot* vsebuje pot do datoteke in njeno ime
AppendText(pot)	Odpiranje obstoječe tekstovne datoteke za pisanje in dodajanje

	teksta v to datoteko. Če datoteka še ne obstaja, se le-ta najprej ustvari. Parameter *pot* vsebuje pot do datoteke in njeno ime
ReadAllText(pot)	Odpiranje obstoječe tekstovne datoteke, branje vseh vrstic iz te datoteke, nato pa še zapiranje datoteke. Parameter *pot* vsebuje pot do datoteke in njeno ime
WriteAllText(pot,stavek)	Kreiranje nove datoteke, zapis stavka v datoteko in zapiranje datoteke. Če datoteka s tem imenom že obstaja, bo prepisana.
WriteAllLines(pot,stavki)	Kreiranje nove datoteke, zapis stavkov (npr. iz tabele stavkov) v datoteko in zapiranje datoteke. Če datoteka s tem imenom že obstaja, bo prepisana.
AppendAllText(pot,stavek)	Zapis stavka v datoteko. Če datoteka še ne obstaja, bo skreirana nova.

Tabela 12: Metode razreda File.

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#.

Metode razreda File

Preveri, če na disku C že obstaja mapa z imenom *Vaje C#*. Če obstaja, v mapi zgeneriraj datoteki *Vaja1* in *Vaja2* (datoteki bosta seveda le ustvarjeni, njuni vsebini pa 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
    if (!Directory.Exists(dir))
    {
        Directory.CreateDirectory(dir);
    }
    string datoteka = "Vaja1";
    string datoteka1 = "Vaja2";
    /*Preverimo obstoj datoteke Vaja 1 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 brišemo
    }
    //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);
    string stavek = "Danes je torek!";
    //Kreirajmo NOVO datoteko, vanjo zapišimo stavek in datoteko zaprimo!
    //POZOR: Če datoteka s tem imenom že obstaja, bo njena vsebina prepisana
    //Datoteka bo v mapi Bin tekočega projekta
    File.WriteAllText("Primer.txt", stavek);
    //Izpiši vsebino datoteke Primer.txt!
    string vsebinaDat = File.ReadAllText("Primer.txt");/vsebino shranimo v
začasen string*/
    Console.WriteLine(vsebinaDat);//izpis vsebine začasnega stringa
    //V datoteko Primer.txt dodajmo še stavek Jutri bo pa sreda
    File.AppendAllText("Primer.txt", "Jutri bo pa sreda!");

    //ustvarimo tabelo stringov, jo inicializirajmo in prepišimo v datoteko
    string[] stavki = new string[5];
    stavki[0] = "Danes je torek.";
    stavki[1] = "Ura je 13.30.";
    stavki[2] = "Učimo se enostavnega dela z datotekami.";
    stavki[3] = "Uporabljamo razred File.";
    stavki[4] = "Podatkovne tokove bomo obravnavali kasneje.";
    File.WriteAllLines("APJ.txt",stavki);//v dat. Zapišemo TABELO stringov
}
```

PODATKOVNI TOKOVI

Kratica *IO* v imenskem prostoru *System.IO* predstavlja vhodne (*input*) in izhodne (*output*) tokove (*streams*). S pomočjo njih lahko opravljamo želene operacije nad datotekami (npr. branje, pisanje). Pisalni ali izhodni tok v jeziku C# predstavimo s spremenljivko tipa *StreamWriter*. Bralni ali vhodni tok pa je tipa *StreamReader*.

V podrobnosti glede tokov se ne bomo spuščali in bomo predstavili zelo poenostavljeno zgodbo. Podatkovne tokove si lahko predstavljamo takole. Lahko si mislimo, da je datoteka v imeniku jezero, reka, ki teče v jezero (ali iz njega), pa podatkovni tok, ki prinaša oziroma odnaša vodo. Če potrebujemo reko, ki v jezero prinaša vodo, bomo v C# potrebovali spremenljivko tipa *StreamWriter* (pisanje toka), če pa potrebujemo reko, ki vodo (podatke) odnaša, pa *StreamReader* (branje toka).

TEKSTOVNE DATOTEKE – BRANJE IN PISANJE

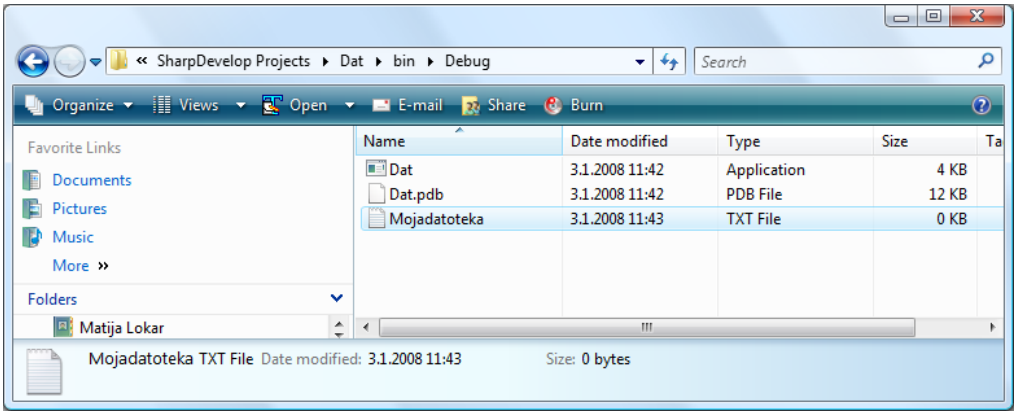
Tekstovne datoteke vsebujejo nize znakov oziroma zlogov, ki so ljudem berljivi. Katere znake vsebujejo, je odvisno od uporabljene kodne tabele.

Poglejmo si najenostavnejši način, kako ustvarimo tekstovno datoteko. "Čarobna" vrstica z ukazom za kreiranje datoteke je

```
File.CreateText("Mojadatoteka.txt");
```

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

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



Slika 20: Vsebina mape z novo ustvarjeno tekstovno datoteko *MojaDatoteka.txt*.

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

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

Pred nizom lahko seveda uporabimo znak @. S tem povemo, da se morajo vsi znaki v nizu jemati dobesedno. Zgornji zgled bi torej lahko napisali kot

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

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

Pred ustvarjanjem nove datoteke je zato smiselno, da prej preverimo, če datoteka že obstaja. Ustrezna metoda je,

File.Exists(ime)

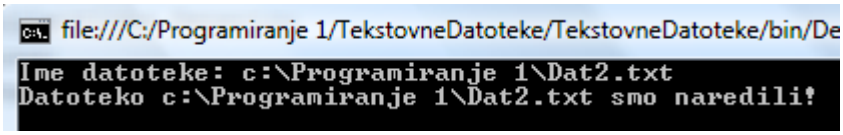
ki vrne *true*, če datoteka obstaja, sicer pa *false*.

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

Ustvari tekstovno datoteko

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

```
public static void Main(string[] args)
{
    Console.Write("Ime datoteke: ");
    string ime = Console.ReadLine();
    if (File.Exists(ime))
    {
        Console.WriteLine("Datoteka " + ime + " že obstaja!");
    }
    else
    {
        File.CreateText(ime);
        Console.WriteLine("Datoteko " + ime + " smo naredili!");
    }
}
```



Slika 21: Ustvarili smo novo datoteko

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

Zagotovo ustvari datoteko

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

```
public static string NovoIme()
{
    Console.Write("Ime datoteke: ");
    string ime = Console.ReadLine();
```

```
    while (File.Exists(ime))
    {  //  če datoteka že obstaja
        Console.WriteLine("Datoteka " + ime + " že obstaja!");
        Console.WriteLine("Vnesi novo ime!\n\n");
        Console.Write("Ime datoteke: ");
        ime = Console.ReadLine();
    }
    return ime;
}
public static void Main(string[] args)
{
    string imeDatoteke = NovoIme();
    File.CreateText(imeDatoteke);
    Console.WriteLine("Ustvarili smo datoteko " + imeDatoteke + "!");
    Console.ReadLine();
}
```

PISANJE V DATOTEKO

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

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

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

```
StreamWriter pisi; // tok za pisanje običajno poimenujemo pisi
pisi = File.CreateText(imeDatoteke);
```

In če sedaj napišemo

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

smo sedaj napisali omenjeno besedilo na datoteko, ki smo jo prej odprli in povezali z oznako. Če pred imenom datoteke ne navedemo poti (oznako diska in mape), bo datoteka shranjena v mapi *Bin* tekočega projekta.

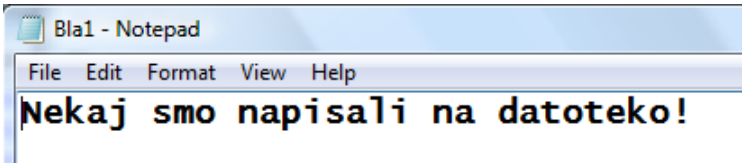
Morebitne nejasnosti bo razjasnil zgled (v zgledu kličemo metodo *NovoIme* iz prejšnjega primera):

```
string imeDatoteke = NovoIme();
StreamWriter pisi;
pisi = File.CreateText(imeDatoteke);
pisi.WriteLine("Nekaj smo napisali na datoteko!");
```

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

```
string imeDatoteke = NovoIme();
StreamWriter pisi;
pisi = File.CreateText(imeDatoteke);
pisi.WriteLine("Nekaj smo napisali na datoteko!");
pisi.Close();
```

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



Slika 22: Vsebina datoteke Bla1

bomo v njej našli omenjeni stavek.

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

Osebni podatki

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

```
public static void Main(string[] args)
{
    // preverimo obstoj mape
    string mapa=@"C:\Tekstovna";
    if (!Directory.Exists(mapa)) // če mapa še ne obstaja, jo ustvarimo
    {
        Directory.CreateDirectory(mapa);
    }
    // pot in ime datoteke
    string ime = @"C:\Tekstovna\Naslov.txt";
    // preverimo obstoj datoteke
    if (File.Exists(ime))
```

```
    {
        Console.WriteLine("Datoteka že obstaja.");
        return; // če datoteka že obstaja, zaključimo program
    }
    // ustvarimo podatkovni tok do nove datoteke
    StreamWriter pisi = File.CreateText(ime);
    // zapišemo osebne podatke
    pisi.WriteLine("Marija Novak");
    pisi.WriteLine("Triglavska 142");
    pisi.WriteLine("Vrba na Gorenskem");
    // zapremo datoteko za pisanje
    pisi.Close();
}
```

Zapis 100 vrstic

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

```
public static void Main(string[] args)
{
    StreamWriter pisi;
    pisi = File.CreateText(@"c:\temp\StoVrstic.txt");
    for (int i = 1; i <= 100; i++)
    {
           pisi.WriteLine(i + ". vrstica");
    }
    pisi.Close();
}
```

Seveda na tekstovno datoteko lahko pišemo tudi s pomočjo metode *Write*. V ta namen si oglejmo še en zgled.

Datoteka naključnih števil

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

12 134 23 22 78

167 12 1 134 45

13 9

Naša metoda bo imela 4 parametre: *n*: koliko števil bo v datoteki, *spMeja*, *zgMeja*: meji za nakključna števila in *kolikoVVrsti*: koliko števil je v vrsti. Rezultat metode bo *void*, saj bo metoda imela le učinek (ustvarjeno datoteko).

```
// metoda
public static void UstvariDat(int n, int spMeja, int zgMeja, int
kolikoVVrsti)
{
    string osnovaImena = @"C:\temp\NakStev";
    int katera = 1;// indeks v imenu datoteke
    string ime = osnovaImena + katera + ".dat";// sestavimo ime datoteke
    while (File.Exists(ime))
    {  //  če datoteka že obstaja
        katera++;
        ime = osnovaImena + katera + ".dat";
    }
    // našli smo "prosto" ime, zato ustvarimo podatkovni tok
    StreamWriter pisi;
    pisi = File.CreateText(ime);
    Random genNak = new Random();// generator naključnih števil
    int stevec = 0;
    while (stevec < n)// n je število podatkov v datoteki
    {
        int nakStev = genNak.Next(spMeja, zgMeja);
        pisi.Write("\t " + nakStev);  // s tabulatorji poskrbimo za poravnavo
        stevec++; // zapisali smo še eno število
        if (stevec % kolikoVVrsti == 0)
        {
            pisi.WriteLine();  // zaključimo vrstico
        }
    }
    if (stevec % kolikoVVrsti != 0)// če je v vrstici že kolikoVVrsti stevil
    {
        pisi.WriteLine();  // zaključimo vrstico
    }
    pisi.Close();
}
public static void Main(string[] args)
{
    UstvariDat(32, 1, 1000, 5); //klic metode v glavnem programu
}
```

BRANJE TEKSTOVNIH DATOTEK – PO VRSTICAH

Tekstovno datoteko odpremo za branje z metodo *File.OpenText()*, ki vrne oznako tako imenovanega podatkovnega toka za branje. To oznako shranimo v spremenljivko tipa *StreamReader*.

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

```
public static void IzpisDatoteke(string ime)
{
    Console.WriteLine("Na datoteki " + ime + " piše: \n\n\n");
    // odpremo datoteko za branje
    StreamReader beri; // tok za branje običajno poimenujemo beri
    beri = File.OpenText(ime);
    // preberemo prvo vrstico in jo izpišemo na zaslon
    Console.WriteLine(beri.ReadLine());
    // preberemo z datoteke naslednji dve vrstici in ju izpišimo na zaslon
    Console.WriteLine(beri.ReadLine());
    Console.WriteLine(beri.ReadLine());
    // preberemo še preostale vrstice
    Console.WriteLine(beri.ReadToEnd());
    beri.Close(); // zapremo podatkovni tok
}
```

Kot smo videli, lahko celo vrstico datoteke beremo z metodo *ReadLine()*. Obstaja pa tudi metoda *ReadToEnd()*, ki prebere vse vrstice v datoteki od trenutne vrstice dalje pa do konca. Tudi datoteke, s katerih beremo, se spodobi zapreti, čeprav pozabljeni *Close* tu ne bo tako problematičen kot pri datotekah, na katere pišemo.

Pri branju vrstic iz datoteke pa moramo natančni. Recimo da bi z zgornjo metodo *IzpisDatoteke* skušali prebrati datoteko, v kateri sta le dva stavka. Izpis na ekranu pa bi vseboval kar 4 vrstice. Namreč, če z metodo *ReadLine()* beremo potem, ko datoteke ni več (neobstoječe vrstice), metoda vrne kot rezultat neobstoječ niz (vrednost *null*). Če tak neobstoječ niz izpišemo, se izpiše kot prazen niz (""). Zato tretji klic izpisa *Console.WriteLine(beri.ReadLine());* izpiše prazen niz. Prav tako tudi metoda *ReadToEnd()* vrne neobstoječ niz, če vrstic ni več.

BRANJE TEKSTOVNIH DATOTEK – VSAK ZNAK POSEBEJ

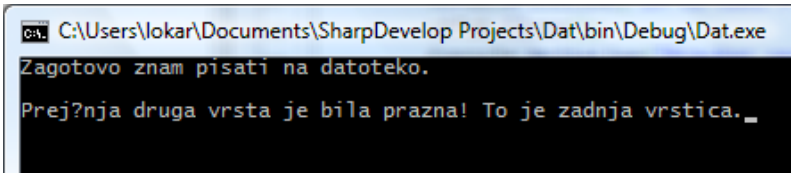
Tekstovne datoteke lahko beremo tudi znak po znak. Poglejmo si kar metodo, ki datoteko znak po znak prepiše na zaslon.

```
public static void IzpisDatotekePoZnakih(string ime)
{
    StreamReader beri = File.OpenText(ime);
    int prebrano;
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

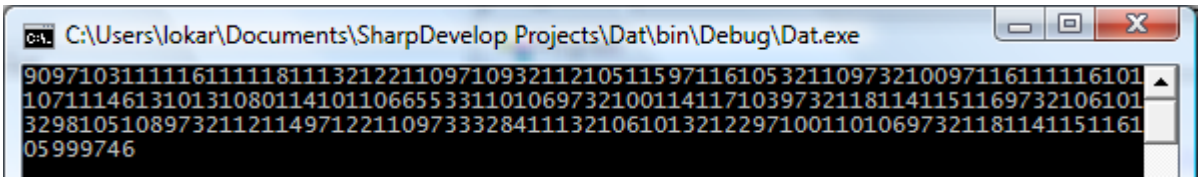
```
    prebrano = beri.Read();
    while (prebrano != -1)  //  konec datoteke
    {
        Console.Write((char)prebrano);  //  izpisujemo ustrezne znake
        prebrano = beri.Read();
    }
    beri.Close();
}
```

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



Slika 23: Pravilen izpis tekstovne datoteke

dobili izpis (če bi namesto *Console.Write(**(char)**beri);* napisali *Console.Write(beri);*)



Slika 24: Kodiran izpis tekstovne datoteke

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

Uporaba metode Split pri branju podatkov iz datoteke

V datoteki c:\Tekstivna\Info.txt je zapisan stavek *"Zala;Polajnar;Ulica Martina cankarja 33;4000;Kranj;Slovenija"*. Med posameznimi podatki je torej ločilni znak dvopičje. Kako bi tako datoteko prebrali in vsak podatek zapisai v svojo vrstico?

```
public static void Main(string[] args)
{
    StreamReader beri = File.OpenText(@"c:\tekstovna\info.txt");
    string info = beri.ReadLine();
    string[] tabInfo;
    // določimo znake, ki predstavljajo ločila med podatki
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

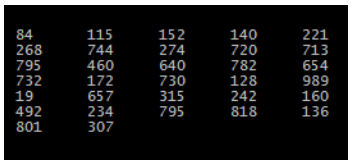
```
    char[] locila = { ';' };  // za ločilo smo vzeli le ;
    tabInfo = info.Split(locila);/*metoda Split vse znake pred naslednjim
ločilom shranjuje v tabelo tabInfo*/
    /*če pa vemo, da je ločilo en sam znak lahko enostavneje napišemo kar
tabInfo = info.Split(';');*/
    for (int x = 0; x < tabInfo.Length; x++) /*izpis tabele stringov, vsakega
v svojo vrstico*/
        Console.WriteLine(tabInfo[x]);

    Console.ReadKey();
}
```

Če pa bi v tabelo locila dodali še več znakov, bo metoda *Split* upoštevala kateregakoli od teh znakov.

Pri obdelavi tekstovne datoteke je potrebno poznati strukturo datoteke, da znamo določiti razmenitveni znak (ločila)!

Branje datoteke števil med seboj ločenih s tabulatorji

Dana je datoteka, kjer je v vrstici več števil, med seboj ločenih s tabulatorji. Števila iz te datoteke bi radi prepisali v tabelo celih števil. Zaenkrat predpostavimo, da je v datoteki 100 števil.



84	115	152	140	221
268	744	274	720	713
795	460	640	782	654
732	172	730	128	989
19	657	315	242	160
492	234	795	818	136
801	307			

Slika 25: Izgled datoteke

```
public static int[] IzDatStevilTabela(string vhod)
{
    StreamReader beri = File.OpenText(vhod);
    int[] tabela = new int[100];
    int koliko = 0;
    string vrstica = beri.ReadLine();// skušamo prebrati vrstico iz datoteke
    // če je branje uspelo je strung vrstica različen od null
    while (vrstica != null)    {
        // vrstico moramo predelati v tabelo posameznih števil
        string[] tabInfo;
        // določimo znake, ki predstavljajo ločila med podatki
        char[] locila = { ' ', '\t' };  //  ločilo je presledek in tabulator
        tabInfo = vrstica.Split(locila);
        for (int x = 0; x < tabInfo.Length; x++)
```

```
        {
            if (tabInfo[x] != "")
            {  //  če nismo dobili le prazni niz
                tabela[koliko] = int.Parse(tabInfo[x]);
                koliko++;
            }
        }
        vrstica = beri.ReadLine();// skušamo prebrati naslednjo vrstico
    }
    beri.Close();
    return tabela;
}

//primer klica metode v glavnem programu
public static void Main(string[] args)
{
    // klic metode IzDatStevilTabela
    int[] tabela = IzDatStevilTabela(@"C:\Tekstovna\Stevila.txt");
    // za vajo še izpis tabele
    for (int i=0;i<tabela.Length;i++)
        Console.Write(tabela[i]+" ");
    Console.ReadKey();
}
```

Najpogostejša napaka pri branju s tekstovnih datotek je ta, da poskušamo z metodo *OpenText* odpreti datoteko, ki je ni! A to že znamo preprečiti. Spomnimo se metode *File.Exists(ime)*,ki smo jo v razdelku o pisanju na datoteke uporabili zato, da nismo slučajno ustvarili datoteke z enakim imenom, kot jo ima že obstoječa datoteka (ker bi staro vsebino s tem izgubili). Torej je smiselno, da pred odpiranjem datoteke preverimo njen obstoj.

V naslednji tabeli so zbrane najpomembnejše metode , ki jih srečamo pri delu s tekstovnimi datotekami.

Razred	**Pomen**
StreamReader	Tip spremenljivke za branje toka.
StreamWriter	Tip spremenljivke, ki ga uporabimo pri spremenljivkah, v katerih je shranjena oznaka podatkovnega toka.
Metoda	**Pomen**
File.Exists()	Metoda, ki jo uporabimo zato, da preverimo obstoj datotek.
File.OpenText()	Metoda, ki izhodni tok poveže z datoteko, s katero delamo.
File.CreateText()	Metoda, ki ustvari datoteko in vrne oznako podatkovnega toka, na katerega pišemo.
File.AppendText()	Metoda, ki vrne oznako podatkovnega toka, preko katerega želimo v že obstoječo datoteko dodajati besedilo. Če datoteka še ne obstaja se

	ustvari nova.
ReadLine()	Metoda, ki prebere celo vrstico v datoteki.
ReadToEnd()	Metoda, ki prebere vse vrstice v datoteki, od trenutne vrstice pa do konca.
Read()	Metoda, ki nam vrne kodo znaka, ki ga preberemo iz datoteke.
WriteLine()	Metoda, ki zapiše dani niz na datoteko, po zapisu se premaknemo v novo vrsto.
Write()	Metoda, ki zapiše dani niz na datoteko.
Close():	Metoda, s katero zapremo datoteko.

Tabela 13: Razreda in osnovne metode za delo z datotekami

Kopija že obstoječe datoteke (po vrsticah in po znakih)

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

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

Naredimo sedaj še kopijo datoteke po znakih. Postopek bo enak, le da bomo brali z *Read* in konec preverjali s kodo znaka -1. Pravzaprav, če dobro premislimo – metoda bo v bistvu

kombinacija prejšnje metode in metode *izpisDatotekePoZnakih*, kjer smo datoteko po znakih izpisali na zaslon. Sedaj bomo počeli enako, le da bomo namesto na zaslon pisali na datoteko.

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

Primerjava vsebine dveh datotek

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

```
public static bool Primerjava(string ime1, string ime2)
{
    // odpremo obe datoteki za branje
    StreamReader dat1 = File.OpenText(ime1);
    StreamReader dat2 = File.OpenText(ime2);
    // preberemo vsebino prve in druge datoteke
    string beri1 = dat1.ReadToEnd();// v beri1 je vsebina prve datoteke
    string beri2 = dat2.ReadToEnd();// v beri2 je vsebina druge datoteke
    // zapremo oba tokova
    dat1.Close();
    dat2.Close();
    // primerjamo vsebini
    return (beri1.Equals(beri2));
}
```

Datoteka padavin

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 '|'. Napišimo metodo za izpis vsebine datoteke na zaslon, metodo, ki dobi za parameter to datoteko in ki ugotovi ter izpiše skupno količino vseh padavin, ter metodo, ki vrne naziv kraja z največ padavinami. Na koncu napišimo še metodo *Dodaj*, ki na konec datoteke doda še dodatno vrstico: podatke o imenu kraja in količini padavin naj vnese uporabnik!

```
//ime datoteke naj bo statična spremenljivka, ki vidna tudi v vseh metodah
static string datoteka = @"c:\Tekstovna\Padavine.txt";
static void Main(string[] args)
{
    if (!File.Exists(datoteka)) //preverimo obstoj datoteke
        Console.WriteLine("Datoteka ne obstaja!");
    else
    {
        izpis(datoteka); //klic metode za izpis vsebine datoteke
        skupajPadavin(datoteka);/*klic metode, ki vrne skupno količ.padavin*/
        //klic metode, ki vrne ime kraja z največ padavinami
        Console.WriteLine("Kraj z največ padavinami: " +
najvecPadavin(datoteka));
        Dodaj(datoteka);//
    }
    Console.ReadKey();
}

static void izpis(string datoteka)
{
    //vzpostavimo povezavo z datoteko na disku
    StreamReader beri = File.OpenText(datoteka);
    Console.WriteLine("Vsebina datoteke Padavine.txt");
    string vrstica = beri.ReadLine(); //skušamo prebrati celo vrstico
    while (vrstica!=null)
    {
        Console.WriteLine(vrstica);//vrstico izpišemo na zaslon
        vrstica = beri.ReadLine(); //skušamo brati naslednjo vrstico
    }
    beri.Close();//zapremo podatkovni tok
}

static void skupajPadavin(string datoteka)
{
    //vzpostavimo povezavo z datoteko na disku
    StreamReader beri = File.OpenText(datoteka);
    string vrstica = beri.ReadLine();
    double vsota = 0;
```

```
    while (vrstica!=null)
    {
        /*stavek razbijemo na posamezne dele, glede na ločilni znak |*/
        string[] tabela = vrstica.Split('|');
        //tabela ima dve celici: v celici z indeksom 0 je ime kraja, v celici
z indeksom 1 pa količina padavin*/
        vsota = vsota + Convert.ToDouble(tabela[1]);
        vrstica = beri.ReadLine();
    }
    Console.WriteLine("Skupna količina padavin: " + vsota);
    beri.Close();
}

static string najvecPadavin(string datoteka)
{
    StreamReader beri = File.OpenText(datoteka);
    string najKraj = "";
    double najPad = 0;
    string vrstica = beri.ReadLine();
    while (vrstica != null)
    {
        string[] tabela = vrstica.Split('|');
        //preverimo, če smo našli kraj z večjo količino padavin
        if (Convert.ToDouble(tabela[1]) > najPad)
        {
            najPad = Convert.ToDouble(tabela[1]);/*shranimo največjo količino
padavin*/
            najKraj = tabela[0];//shranimo ime kraja
        }
        vrstica = beri.ReadLine(); //preberemo stavek iz datoteke
    }
    beri.Close();
    return najKraj;//metoda vrne ime kraja
}

private static void Dodaj(string datoteka)
{
    try
    {
        StreamWriter pisi = File.AppendText(datoteka);/*datoteko odpremo za
dodajanje*/
        //od uporabnika zahtevamo vnos kraja in količine padavin
        Console.Write("Ime kraja: ");
        string kraj = Console.ReadLine();
        Console.Write("Količina padavin: ");
        int kolicina = Convert.ToInt32(Console.ReadLine());/*pretvorba v celo
število*/
        //zapis v datoteko, med podatkoma pa je ločilni znak |
        pisi.WriteLine(kraj+'|'+kolicina);
        pisi.Close();//na koncu obvezno zapiranje podatk. toka (datoteke)
```

```
    }
    catch
    {
        Console.WriteLine("Napaka!");
    }
}
```

Odstranitev števk

Sestavimo metodo *PrepisiBrez*, ki sprejme imeni vhodne in izhodne datoteke. Metoda na izhodno datoteko prepiše tiste znake iz vhodne datoteke, ki niso števke. Predpostavimo, da sta imeni datotek ustrezni (torej, da datoteka za branje obstaja, datoteka za pisanje pa ne (oziroma da njeno morebitno prejšnjo vsebino lahko izgubimo)). Najprej bomo odprli ustrezni datoteki. Prvo datoteko odpremo za branje in drugo za pisanje. Brali bomo posamezne znake iz vhodne datoteke. Če prebran znak ne bo števka, ga bomo prepisali v izhodno datoteko. To bomo počeli toliko časa, dokler ne bomo prebrali znaka s kodo -1 (torej znak EOF).

```
public static void PrepisiBrez(string imeVhod, string imeIzhod)
{
    StreamReader beri;  // ustvarimo podatkovni tok za branje na datoteki
    beri = File.OpenText(imeVhod);
    StreamWriter pisi;  // ustvarimo tok za pisanje na datoteko
    pisi = File.CreateText(imeIzhod);
    // prepis znakov iz ene datoteke na drugo datoteko
    int znak = beri.Read();  // preberemo en znak
    while (znak != -1)
    {
        // primerjamo ali je prebrani znak števka
        if (!('0' <= znak && znak <= '9'))
            pisi.Write("" + (char)znak);  // če ni števka, zapišemo znak
        znak = beri.Read();
    }
    // zapremo podatkovna tokova
    beri.Close();
    pisi.Close();
}
```

Pri obdelavi (branju) podatkov iz tekstovne datoteke včasih uporabimo tudi metodo *Peek()*. Metoda prebere naslednji znak v datoteki, brez premika datotečnega kazalca. Kadar metoda vrne *-1* vemo, da je konec datoteke!

Gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Obdelava tekstovne datoteke

Za poljubno tekstovno datoteko (ime vnese uporabnik) ugotovimo koliko vrstic vsebuje, koliko je vseh znakov v datoteki, koliko samoglasnikov vsebuje in katera je najdaljša vrstica v datoteki.

```
static void Main(string[] args)
{
    string datoteka;
    //ime datoteke vnese uporabnik.
    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 podatkovniTok = new FileStream(datoteka, FileMode.Open,
FileAccess.Read);//tok za branje
    //kreiramo nov objekt tipa StreamReader za zapisovanje v podatkovni tok
    StreamReader beri = new StreamReader(podatkovniTok);/*podatke bomo brali
v podatkovni tok*/

    int znakov = 0, vrstic = 0, samoglasnikov = 0;
    string najdaljsi = "";
    while (beri.Peek() != -1) /*iz datoteke beremo podatke dokler jih ne
zmanjka*/
    {
        string stavek = beri.ReadLine();     //preberemo cel stavek
        if (stavek.Length > najdaljsi.Length)/*preverimo, če je ta stavek
daljši od doslej najdaljšega*/
            najdaljsi = stavek;
        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++;
        }
    }
    beri.Close();  //zapiranje podatkovnega toka in s tem datoteke
    podatkovniTok.Close();
    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);
    Console.ReadKey();
}
```

NAPREDNO DELO S PODATKOVNIMI TOKOVI

Za delo z datotekami obstajajo tudi bolj napredne metode, ki nudijo še več kontrole nad delom z datotekami. Zaenkrat nam sicer povsem zadoščajo metode razreda *File*, a le za informacijo bomo prikazali še en način.

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

```
FileStream podatkovniTok = 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 podatke 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:

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.

Tabela 14: Tabela načinov kreiranja datoteke – *FileMode*

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:

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.

Tabela 15: Tabela načinov manipulacije s podatki – *FileAccess*

S tretjim argumentom *share* 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:

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.

Tabela 16: Tabela načinov porazdelitev (dostopa) podatkov z drugimi aplikacijami – *FileShare*

Kreiranju objekta tipa *FileStream* nato sledi objekt za branje oziroma pisanje v datoteko. Če je datoteka tekstovna, sta tule oba primera:

```
StreamWriter pisi=new StreamWriter(podatkovniTok);/*objekt pisi za
pisanje*/
StreamReader beri=new StreamReader(podatkovniTok); // objekt beri za branje
```

Met kocke

Kreirajmo tekstovno datoteko *Kocka.txt*, v katero želimo shraniti rezultate 1000 metov kocke. V vsako vrstico te datoteke zapišimo zaporedno številko vrstice in naključno število med 1 in 6 (met kocke).

```
static void Main(string[] args)
{
    Random naklj=new Random();//generator naključnih števil
    //kreiramo podatkovni tok tipa FileStream
    FileStream podatkovniTok = new FileStream("Kocka.txt", FileMode.Create,
FileAccess.Write);
    StreamWriter pisi=new StreamWriter(podatkovniTok);
    /*OBA STAVKA LAHKO ZDRUŽIMO V ENEGA:
     StreamWriter PISI = new StreamWriter("Kocka.txt"); tak način je PRIVZETI
     način za odpiranje nove datoteke: stara vsebina datoteke
     bo prepisana z novo vsebino, ne glede na prejšnjo vsebino datoteke!!!!*/

    //v datoteko zapišimo 1000 naključnih števil, rezultatov metov kocke
    for (int i=0;i<1000;i++)
    {
        int stevilo = naklj.Next(6) + 1;//met kocke: naklj.število med 1 in 6
        pisi.WriteLine((i+1)+".\t"+stevilo);//zapis v datoteko
    }
    pisi.Close();
    Console.ReadKey();
}
```

Najdaljša beseda v datoteki

Za poljubno tekstovno datoteko ugotovi in nato izpiši najdaljšo besedo v tej datoteki.

```
static void Main(string[] args)
{
    Console.Write("ime datoteke: ");
    string ime = Console.ReadLine();
    if (File.Exists(ime))
    {
        FileStream podatkovniTok = new FileStream(ime, FileMode.Open);
        StreamReader beri = new StreamReader(podatkovniTok);
        string najdaljsa = "";
        string vrstica = beri.ReadLine();/*vnesti moramo celotno ime
datoteke, tudi končnico*/
```

```
        while (vrstica!=null) //dokler ni konec datoteke
        {
            //z metodo Split posamezne besede iz vrstice shranimo v tabelo
            string[] tabela = vrstica.Split(' ');/*besede so ločene s
presledki*/
            for (int i = 0; i < tabela.Length; i++)
            {
                if (tabela[i].Length > najdaljsa.Length)
                {
                    najdaljsa = tabela[i];
                }
            }
            vrstica = beri.ReadLine(); //preberemo naslednjo vrstico
        }
        beri.Close();
        podatkovniTok.Close();
        Console.WriteLine("Najdaljša beseda: " + najdaljsa);
    }

    Console.ReadKey();
}
```

BINARNE DATOTEKE – BRANJE IN PISANJE

Za branje in pisanje podatkov v binarne datoteke uporabljamo razreda **BinaryReader** in **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 podatkovniTok = new FileStream(datoteka, FileMode.Create,
FileAccess.Write);
//Kreiramo podatkovni tok za pisanje
BinaryWriter PisiBinarno = new BinaryWriter(podatkovniTok);
```

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**.

Tabela 17: Osnovni metodi za delo z binarno datoteko

Števila v binarni datoteki

V binarno datoteko *Stevila.dat* zapišimo 10000 naključnih celih števil med -100 in +100!

```
static void Main(string[] args)
{
    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 podatkovniTok = new FileStream(datoteka, FileMode.Create,
FileAccess.Write);
    BinaryWriter pisiBinarno = new BinaryWriter(podatkovniTok); //binarni
podatkovni tok
    Random naklj = new Random();  //generator naključnih števil

    for (int i = 0; i < 10000; i++)   //zanka za generiranje 10000 naključnih
števil
        pisiBinarno.Write(naklj.Next(-100, 101)); //zapis naključnih števil
med -100 in 100 v binarno datoteko

    pisiBinarno.Close();
    podatkovniTok.Close();
    Console.ReadKey();
}
```

Binarna datoteko izdelkov

V naslednjem primeru bomo uporabili objekt razreda *Izdelek*. Podatke bomo zapisovali v binarno 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 celoten objekt naenkrat.

```
public class Izdelek  //razred Izdelek
{
```

```
    //za naš primer potrebujemo le nekaj polj in konstruktor
    public string naziv;
    public string proizvajalec;
    public int komadov;
    public decimal cena;
    public Izdelek(string naz, string proizv, int kom, decimal cen)
    {
        naziv=naz;
        proizvajalec=proizv;
        komadov=kom;
        cena=cen;
    }
}

//glavni program
static void Main(string[] args)
{
    try
    {
        //določimo pot in ime datoteke
        string pot = @"C: \Binarne\Izdelki.dat";
        //dinamično kreiramo nov podatkovni tok
        FileStream podatkovniTok = new FileStream(pot, FileMode.Create,
FileAccess.Write);
        //dinamično kreiramo nov objekt tipa BinaryWriter
        BinaryWriter pisiBinarno = new BinaryWriter(podatkovniTok);
        //ustvarimo nov objekt tipa Izdelek
        Izdelek Izd = new Izdelek("Kolo", "Scott", 110, 2200.00m);
        /*podatke zapišemo v binarno datoteko in sicer vsako polje objekta
posebej*/
        pisiBinarno.Write(Izd.naziv);
        pisiBinarno.Write(Izd.proizvajalec);
        pisiBinarno.Write(Izd.komadov);
        pisiBinarno.Write(Izd.cena);
        //zapremo tok podatkov in s tem tudi datoteko
        pisiBinarno.Close();
        podatkovniTok.Close();
        Console.ReadKey();
    }
    catch { Console.WriteLine("Napaka!"); }
}
```

POVZETEK

Datoteke bomo v svojem programu uporabili tedaj, kadar obdelujemo veliko količino podatkov oz. kadar hočemo, da se podatki ohranijo tudi po zaključku izvajanja programa. Razložene so bile le osnovne operacije s tekstovnimi datotekami, ki jih uporabljamo za shranjevanje teksta. Dostop do podatkov je tu sekvenčen. Kadar pa bomo želeli v datoteko shraniti velike količine podatkov, ki jih želimo kasneje tudi obdelovati, bomo raje uporabili binarne datoteke. Podatki, ki jih pišemo na binarno datoteko, se ne pretvarjajo v znake, ampak se zlogi zapišejo tako, kot so shranjeni v pomnilniku.

VAJE

1. Sestavite metodo, ki za vhodni parameter sprejme tekstovno datoteko in ki ustvari novo datoteko s končnico bak, v kateri so med znaki originalne datoteke presledki. Če je npr. besedilo datoteke *Lepa Anka kolo vodi,* bo vsebina nove datoteke *L e p a A n k a k o l o v o d i .*

2. Napiši program, ki bo v zanki bral podatke o določenem kraju in njegovi poštni številki. Podatke zapisuj v tekstovno datoteko, vsak kraj v svojo vrstico. Zanka (vnos podatkov) se zaključi, ko uporabnik vnese prazen kraj!

3. Napiši metodo, ki dobi za parameter dve tekstovni datoteki Imena.txt in Priimki.txt. Metoda naj ustvari novo datoteko *ImenaInPriimki.txt*, v kateri bodo imena in priimki združeni! Število imen v prvi datoteki je enako številu priimkov drugi datoteki.

4. 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 0.0001. Vrednosti funkcije na vsakem koraku shranjuj v tekstovno 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.*

5. Numerologi računajo "osebno število" iz datuma rojstva osebe tako, da seštevajo števke rojstnega datuma, dokler ne dobijo enomestnega števila. Na primer, za rojstni datum

19.8.1985 bi dobili število 5, saj je 1+9+8+1+9+8+5 = 41, 4+1 = 5. Sestavi metodo, ki bo v tekstovno datoteko Osebno.txt zapisala osebna števila za vse dni v letu 2010!

6. Napišimo program, za pisanje telegrama. Uporabnik vnaša stavke in ko pritisne Enter, naj se stavek za piše v datoteko, zraven pa še besedica *STOP*. Če uporabnik vnese besedico KONEC je telegram zaključen - datoteko zapremo. Ime datoteke naj določi uporabnik!

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

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

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

Najmanjša je Cindy Crawford, ki meri 179 cm.

Največja je Nina Gazibara, ki meri 185 cm.

8. Napišite program, v katerem napolnite tekstovno datoteko s 100 naključnimi štirimestnimi števili. Ta števila potem preberite iz datoteke in izpišite povprečje vseh lihih števil.

9. Kreiraj tekstovno datoteko *EU.txt* in binarno datoteko *EU.dat*, v katere boš zapisoval države evropske unije. Napiši program, ki bo omogočil vnos, pregled, dodajanje in brisanje držav iz teh dveh datotek. Ločilni znak med posameznimi podatki v tekstovni datoteki naj bo podpičje!

10. V binarno datoteko *Place.txt* zapiši 100 naključnih decimalnih števil med 0 in 10000, zaokroženih na 2 decimalki. Datoteko nato obdelaj tako, da izračunaš in izpišeš povprečno vrednost vseh števil v datoteki.

METODA Main()

Ko poženemo katerikoli program, se le-ta začne izvajati v metodi **Main()**, ki je lahko zapisana kjer koli v programu. Metoda *Main()* je torej vstopna točka našega programa, zaključek te metode pa je obenem tudi zaključek našega programa. Deklarirana je znotraj razreda in mora biti statična. Vedno lahko obstaja le ena metoda *Main()*. V glavi te metode je najprej deklaracija tipa, ki ga metoda vrača, temu sledi ime metode in tabela parametrov - *string[] args*. Pomen tabele *args* 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.

Metoda *Main**()*** lahko torej lahko prav tako sprejme parametre. Ker pa v fazi razvoja programa ne moremo vedeti, kakšne parametre bo uporabnik napisal, v metodi *Main()* ne moremo preprosto določiti, naj sprejme *int*, *double*, *char* ipd. Vsi parametri metode so zato shranjeni v tabeli, kjer je vsak element tabele nek niz znakov, ki predstavlja vhodni parameter. Tako v metodi *Main()* potrebujemo samo en parameter – ime tabele, v kateri so shranjeni njeni parametri.

Tabelo *args* lahko v programu uporabimo tako kot vsako drugo tabelo: z metodo *Length* lahko preverimo njeno dolžino (če je le-ta enaka 0, potem uporabnik pri zagonu programa ni vnesel nobenega vhodnega parametra), s pretvorbo v drug podatkovni tip lahko z elementi tabele računamo ipd.

Napišimo naslednji program in ga shranimo pod imenom *Main1*

```
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 program iz ukazne vrstice ali pa s pomočjo bližnjice zaženemo takole:

Main1 arg1 arg2 arg3 arg4 <ENTER>

dobimo izpis :

```
Skupno število parametrov: 4
```

Poleg imena programa smo zapisali še štiri parametre (arg1 arg2 arg3 arg4 - med posameznimi parametri je presledek).

ıdivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Seznam vhodnih parametrov

Napišimo naslednji program V*hodniParam*, v katerem ugotovimo in izpišimo število in imena vhodnih parametrov.

```
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 program zaženemo npr takole :

VhodniParam 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 *VhodniParam*. V elementu tabele z indeksom 0 je torej zapisan parameter *Prvi*, v elementu z indeksom 1 je zapisan parameter *Drugi*, v elementu z indeksom 2 pa parameter *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 *VhodniParam* poženemo npr. takole:

VhodniParam "Celoten stavek je en sam argument" <ENTER>

dobimo izpis :

```
Skupno število parametrov: 1
Celoten stavek je en sam argument
```

Ker je *Main()* metoda, je lahko tipa *void* (ne vrača ničesar) ali pa ima tip, kar pomeni da vrača rezultat. Če ima metoda *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 metode *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 metoda *Main()*). Tradicionalno pomeni vrnjena vrednost nič (0) uspešen zaključek metode *Main()*.

Vsota vhodnih parametrov

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)
{
    try
    {
        if (args.Length == 2)//preverimo, če je uporabnik vnesel 2 parametra
        {
            //v args[0] je prvi v args[1] pa drugi parameter
            int prvo = Convert.ToInt32(args[0]);//pretvorba v celo število
            int drugo = Convert.ToInt32(args[1]);//pretvorba v celo število
            Console.WriteLine("Vsota: " + (prvo + drugo));//izpis vsote
        }
        else Console.WriteLine("Napačno število parametrov!");
    }
    catch
    {
        Console.WriteLine("Napaka pri vnosu parametrov!");
    }
    Console.ReadKey();
}
```

Izračun potence

Napiši program, imenovan *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).

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
static void Main(string[] args)
{
    //varovalni blok, da se program pri napačnih parametrih ne sesuje
    try
    {
        if (args.Length == 2) /*preverimo, če smo v ukazni vrstici vnesli dva
argumenta */
        {
            double st1 = Convert.ToDouble(args[0]);
            int st2 = Convert.ToInt32(args[1]);
            double rezultat = 1, temp = st1;
            if (st2 == 0) rezultat = 1; //če je eksp. 0, je potenca 1
            else if (st2 == 1) rezultat = st1; //če je eksp. 0 je potenca st1
            else if (Math.Abs(st2) > 1)
            {
                for (int i = 1; i < Math.Abs(st2); i++)
                    st1 = st1 * temp;  //izračun potence
            }
            if (st2 > 0) rezultat = st1;
            else rezultat = 1 / st1; //eksponent nagativen
            Console.WriteLine("Rezultat: " + rezultat);
            Console.ReadKey();
        }
    }
    catch
    {
       Console.WriteLine("Napaka!");
    }
}
```

POVZETEK

Ugotovili smo, da je tudi glavni program (*Main()*) metoda, ki lahko sprejme parametre. Le-ti se privzeto zapisujejo v tabelo stringov z imenom *args*, seveda pa lahko to tabelo poimenujemo tudi drugače (v tem primeru le ime *args* nadomestimo npr. z *TabelaParametrov*, nato pa v programu uporabljamo svoje ime tabele).

VAJE

1. Napiši program, ki izračuna in izpiše *faktorielo* poljubnega celega števila, ki nastopa kot parameter programa. *Faktoriela* je matematična operacija, definirana takole:

   N!=N * (N-1) * (N – 2) * … * 3 * 2 * 1

   Konkretno: 7! = 7 * 6 * 5 * 4 * 3 * 2 * 1

2. Napiši program, ki mu v ukazni vrstici podamo poljubno število besed, program pa med njimi poišče najdaljšo in najkrajšo besedo!

3. Pitagora je imenoval dve celi števili prijateljski, če je vsota deliteljev prvega števila enaka drugemu številu in obratno. Napišite program, ki bo za vhodni dve celi števili preveril, ali sta števili prijateljski in izpisal rezultat v obliki:

   Primer: števili 220 in 284 sta si prijateljski:

   220 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
   284 = 1 + 2 + 4 + 71 + 142 = 220

4. Napiši metodo ki na zaslon nariše trikotnik, sestavljen iz znakov *. Velikost trikotnika določa nenegativno celo število n, ki je podano kot parameter glavnega programa.

5. Napiši program *Obrni*, ki naj dobi za vhodna podatka dva stringa, ime in priimek. Program naj iz teh dveh stringov sestavi nov string tako, da najprej obrne ime in priimek, nato pa izmenično iz zaporednih črk obrnjenega imena in priimka sestavi nov string. Iz stringov ime in priimek vzemi le toliko črk, kot znaša dolžina krajšega od obeh stringov.

6. Sestavi program *Nenicelne*, ki za naravno število n, ki nastopa kot parameter programa, izračuna in izpiše produkt njegovih neničelnih števk.

   Primer: za število 2304701 naj metoda izpiše 168!

7. Napiši program *Ločila*, ki dobi za vhodna podatka poljuben stavek in poljuben znak (ločilo). Program naj prešteje in na koncu v primerni obliki izpiše skupno število ustreznih ločil v tem stavku!

8. Napiši program *Premica*, ki dobi za vhodna podatka smerni koeficient k in prosti člen n premice y = k x + n. Program naj ugotovi in izpiše pod kakšnim kotom premica seka

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

abscisno os, kolikšni sta koordinati presečišč z obema osema in ploščino pravokotnega trikotnika, ki jo tvori premica z obema osema.

9. Napiši program *Prestopno*, ki dobi za parameter poljubno letnico in ki ugotovi in izpiše, če je leto prestopno. Leto je prestopno, kadar je deljivo s 4 in ne s 100 ali kadar je deljivo s 400.

10. Napiši program *Luknje*, ki izpiše koliko lukenj je v naravnem številu n, ki nastopa kot parameter programa. Z luknjami so mišljene "luknje" v grafični podobi posameznih števk: števke 0, 4, 6, 9 imajo po eno luknjo, števka 8 ima dve luknji in preostale števke nimajo nobene luknje. Primer: Število 6854 ima 4 luknje.

REKURZIJA

Kar nekaj znanja o metodah že imamo: znamo napisati lastne metode, poznamo objektne metode, pa statične metode, metode znamo tudi preobtežiti. Znamo jih tudi klicati oz. jih uporabiti. V tem poglavju pa bomo spoznali še rekurzivne metode.

Rekurzivni klic metode je močno programersko orodje. Velikokrat se s pomočjo rekurzije algoritmi lažje in pregledneje izrazijo. O rekurzivnem klicu oz. rekurzivnih metodah govorimo takrat, ko metoda kliče samo sebe. V takih metodah največkrat uporabljamo nek parameter, ki se zmanjšuje ali povečuje, ko pa 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 metoda kliče samo sebe, se le-ta navadno začne s pogojem (*if*), ki ustavi rekurzijo.

```
rekurzivna_metoda (problem)
{
      if (problem majhen)
            return resitev;
      else
        razdeli problem na enega ali več manjših podproblemov enake vrste
        za vse podprobleme
      rekurzivna_metoda (manjši problem);
}
```

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

Rekurzijo si oglejmo na primertu Fibonaccijevega zaporedja: 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) n= 3, 4, 5 ....

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 neposredno 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. ;

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

Napišimo sedaj metodo *int Fib(int N)* , ki za dani *n* izračuna in vrne 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));
}

static void Main(string[] args)
{
    //Primer klica te metode v glavnem programu
    Console.WriteLine(Fib(33));  //Klic rekurzivne metode Fib
    Console.ReadKey();
}
```

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 metodi 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.

Prav vsako rekurzivno metodo pa seveda lahko napišemo tudi na klasičen način, iterativno. V primeru Fibonaccijevega zaporedja bi bila iterativna rešitev taka:

```
static int Fib1(int n)  /*Klasična (iterativna)  metoda za izračun poljubnega člena Fibonaccijevega zaporedja*/
{
    if (n == 1) return 1; //prvi člen zaporedja
    if (n == 2) return 1;  //drugi člen zaporedja
    int F1 = 1, F2 = 1, P;
    for (int i = 0; i < n - 2; i++)
    {
        P = F2;
        F2 = F1 + F2;
        F1 = P;
    }
    return F2;
}
```

Pa še primer klica te metode v glavnem programu

```
Console.WriteLine(Fib1(33));  //Klic iterativne metode Fib1
```

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. Zasedenost pomnilnika se pri rekurzivnih metodah skriva v rekurzivnih klicih. Vsak rekurzivni klic namreč zahteva nekaj prostora, zapomniti si je potrebno tudi mesto, kam se je potrebno vrniti ob povratku iz metode. 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*) - delu pomnilnika, za katerega 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 bolj potratni od iterativnih. Podobna je situacija glede hitrosti izvajanja. Klic metode traja nekoliko dlje kot enostavne

operacije v iterativnih zankah. V našem zgledu je rekurzivni podprogram še posebno počasen, saj zahteva več seštevanj kot iterativni.

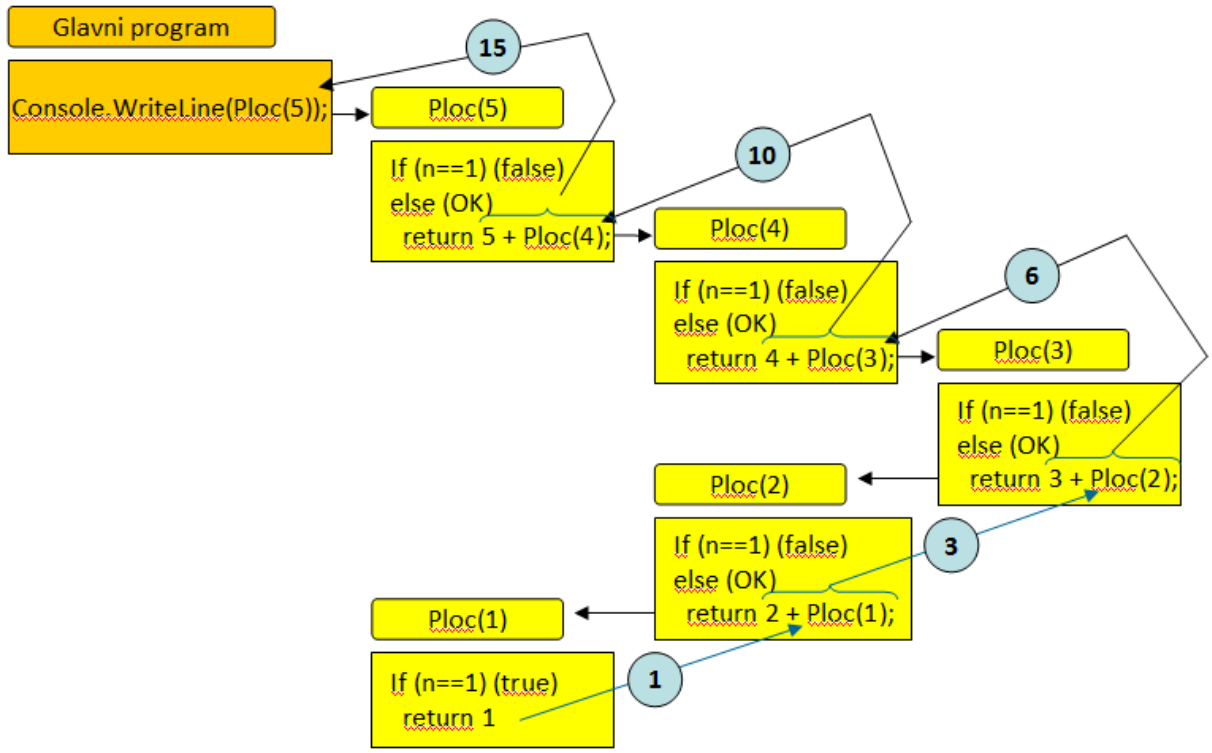
Rekurzivne formulacije imajo torej prednost pred iterativnimi v tem, da je programiranje bolj enostavno, jasno in razumljivo. Nekoliko manj učinkovit pa je rekurzivni podprogram glede časa izvajanja in prostora v pomnilniku. Od konkretnega primera pa je odvisno, za katero pot se bomo odločili. Rekurzija je torej dobra, a počasna, zato se ji poskušamo izogniti, če je to le mogoče, oz. če s tem ne bi preveč zakomplicirali algoritma.

Pločevinke

Zamislimo si, da pobiramo pločevinke in jih zlagamo 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. Napišimo rekurzivno metodo *Ploc*, ki bo izračunala 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 Ploc(int n) //rekurzivna metoda
{
    if (n == 1) //če je vrsta ena sama, metoda vrne 1
    {
        return 1;
    }
    else
    {
        return n + Ploc(n - 1);//rekurzivni klic, parameter je za 1 manjši
    }
}
// glavni program
static void Main(string[] args)
{
    //klic rekurzivne metode
    Console.WriteLine("Število pločevink v petih vrstah: " + Ploc(5));
    Console.ReadKey();
}
```

Poglejmo si še pomnilniško sliko za rekurzijo *Ploc* pomenu, da se trenutna metoda prekine in se začne izvajati nova metoda z enakim imenom, a manjšim parametrom (*Ploc(5)→Ploc(4) →Ploc(3))→Ploc(2))→Ploc(1)*). Metoda *Ploc(1)* vrne metodi *Ploc(2)* nazaj vrednost 1, ta vrne metodi *Ploc(3)* vrednost 3 (ker je *2 +Ploc(1)* enako 3) in tako naprej. Na koncu metoda *Ploc(5)* vrne glavnemu programu vrednost 15, stavek *Console.WriteLine* pa to vrednost izpiše.



Slika 26: Pomnilniška slika ob klicu rekurzije *Ploc(5)*.

Številski sestavi

Napišimo rekurzivno metdo za pretvarjanje iz desetiškega v poljuben sestav.

```
//rekurzivna metoda
static void pretvori(int stevilo, int sestav)
{
    if (stevilo > 0)
    {
        //stevilo/sestav je celoštevilsko deljenje
        pretvori(stevilo / sestav, sestav);
        Console.Write(stevilo % sestav);//% je oznaka za ostanek pri deljenju
    }
    else Console.Write("\nPretvorjeno število : ");
}

static void Main(string[] args)
{
    Console.Write("Vnesi število : ");
    int stevilo = Convert.ToInt32(Console.ReadLine());
    Console.Write("\nŠtevilski sestav : ");
```

```
    int sestav = Convert.ToInt32(Console.ReadLine());
    pretvori(stevilo, sestav); //klic rekurzije
    Console.WriteLine();
    Console.ReadKey();
}
```

Izpis stavka v obratni smeri

Poljuben stavek s pomočjo rekurzivne metode izpišimo v obratni smeri.

```
static void obrat(string str)
{
    if (str.Length > 0)
        obrat(str.Substring(1, str.Length - 1)); //vsi znaki, razen zadnjega
    else
        return;
    Console.Write(str[0]); //izpis prvega znaka
}
static void Main(string[] args)
{
    string s = "Tole je testni stavek";
    Console.WriteLine("Originalen stavek: " + s);
    Console.Write("Obratni stavek: ");
    obrat(s); //klic rekurzije
    Console.WriteLine();
}
```

POVZETEK

Ko govorimo o rekurziji pri programiranju, mislimo na to, da pri sestavljanju algoritma metode za reševanje nekega problema v rešitvi uporabimo zopet klic (enega ali več) te metode. Rešitev problema je torej podana s samim problemom, le nad manjšim ali pa drugačnim obsegom podatkov. V opisu postopka rešitve torej uporabimo kar ta postopek. Če želimo priti do rešitve, seveda ne moremo nadaljevati v nedogled, saj se potem naš postopek ne bi nikoli končal. Zato je pri rekurzivnih algoritmih zelo pomembno določiti ustavitveni pogoj. Ta pove, daj v postopku ne uporabimo ponovno istega postopka (ne pokličemo te metode). Običajno je to takrat, ko so podatki (parametri metode) taki, da je problem "majhen" (enostaven).

VAJE

1. 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* !

2. Dana je rekurzivna metoda *Pisem*. 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));
}
```

3. 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).

4. S pomočjo rekurzije napiši program, ki izračuna vsoto vrste $1^0 + 2^1 + 3^2 + 4^3 + \ldots + x^{(x-1)}$

5. Napiši rekurzivno metodo za izračun potence števila. Nariši pomnilniško sliko za potenco z osnovo 3 in eksponentom 4!

6. Napiši rekurzivno metodo, ki izračuna vsoto števil večjih od nekega naravnega števila n in manjših od 1000, ki so deljiva z 8!

7. Napiši rekurzivno metodo za izpis poštevanke poljubnega števila!

8. Nekega dne je Ančka vsa obupana prosila brata Jureta, naj ji napiše program za domačo nalogo. Ta bi moral rekurzivno izračunati vsoto prvih n naravnih števil. A ker se je Juretu mudilo na vsakodnevni žur, je moral zelo hiteti in je zato v programu naredil nekaj napak. Jih najdeš?

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
public static int Vsota(int n)
{
    if (n > 0)
        return 1;
    else
        return vsota(n - 1);
}
```

9. Dana je naslednja rekurzivna metoda. Ugotovi, kaj dela. Šele potem!! jo prenesi v testni program in zaženi ter preveri, da res dela tisto, kar si si predstavljal.

```
public static string KajDelam(int stevec)
{
    if (stevec <= 0)
        return "";
    else
        return "" + stevec + ", " + KajDelam(stevec - 1);
}
```

Metodo nato prepiši tako, da bo vrnila niz s števili v obratnem vrstnem redu kot prvotna.

10. Dan je naslednji rekurzivni program:

```
public static int Puzzle(int baza, int limit)
{
    //base in limit sta nenegativni števili
    if (baza > limit)
        return -1;
    else
    {
        if (baza == limit)
            return 1;
        else
            return baza * Puzzle(baza + 1, limit);
    }
}
```

Program si najprej oglej, nato pa BREZ uporabe računalnika odgovori na spodnja vprašanja:

a. kateri del metode *Puzzle* je ustavitveni pogoj?
b. kje se izvede rekurzivni klic?
c. kaj izpišejo sledeči stavki:
    - *Console.WriteLine(Puzzle(14,10));*
    - *Console.WriteLine(Puzzle(4,7));*
    - *Console.WriteLine(Puzzle(0,0));*

Nadležni starši

Na koncu pa še rešitev za kodiranje sporočil, ki si jih izmenjujemo s prijateljem. Rešitev je sestavljena tako, da pri zagonu programa zapišemo (kot parameter) še ime datoteke s sporočilom, ki ga želimo zakodirati. Program bo sporočilo zakodiral in ga napisal v datoteko z enakim imenom, končnica pa bo *.kod*.

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

    if (args.Length == 1)/*preverimo, če je uporabnik vnesel parameter=ime
datoteke*/
    {
        string ime=args[0];//ime datoteke je enako prvemu parametru
        string novoIme=ime.Substring(0,(ime.Length -4))+".kod";/*novo ime
datoteke=staro ime s končnico kod*/
        if (File.Exists(ime)) //preverimo, če datoteka obstaja
        {
            try //varovalni blok
            {
                StreamReader beri = File.OpenText(ime); //tok za branje
                StreamWriter pisi = File.CreateText(novoIme);//tok za pisanje
                string lihiZnaki = "";
                string vrstica = beri.ReadLine();//skušamo brati prvo vrstico
                while (vrstica != null)
                {
                    //V novo datoteko najprej zapišemo znake z sodimi indeksi
                    for (int i=0;i<vrstica.Length;i=i+2)
                        pisi.Write(vrstica[i]);
                    //znake z lihimi indeksi dodajamo v pomožni string
                    for (int i = 1; i < vrstica.Length; i = i + 2)
                        lihiZnaki += vrstica[i];
                    //dodamo še znak za konec vrstice
                    pisi.WriteLine();
                    lihiZnaki = lihiZnaki + "\r\n";//še znak za novo vrstico
                    vrstica=beri.ReadLine();//beremo naslednjo vrstico
                }
                beri.Close(); //zapremo tok za branje
                pisi.WriteLine(lihiZnaki);/*v izhodno datoteko zapišimo še
vse znake z lihimi indeksi*/
                pisi.Close(); //zapremo tok za pisanje
            }
            catch
```

```
            {
                Console.WriteLine("Napaka!");
            }
        }
        else Console.WriteLine("Datoteka s tem imenom ne obstaja!");
    }
    else Console.WriteLine("Napačno število parametrov!");

    Console.ReadKey();
}
```

LITERATURA IN VIRI

- Sharp, J. *Microsoft Visual C# 2005 Step by Step*. Washington: Microsoft Press, 2005.
- Lokar, M., in Uranič, S. *Programiranje 1*.Ljubljana: MŠŠ, 2008.
- Murach, J., in Murach, M. *Murach's C# 2008*. Mike Murach @ Associates Inc. London, 2008.
- Petric, D. *Spoznavanje osnov programskega jezika C#*, diplomska naloga. Ljubljana: UL FMF, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Objektno programiranje*. Ljubljana: B2, 2008.
- Jerše, G., in Lokar, M. *Programiranje II*, *Rekurzija in datoteke*. Ljubljana: B2, 2008.
- Kerčmar, N. *Prvi koraki v Javi, diplomska naloga*. Ljubljana: UL FMF, 2006.
- Lokar, M. *Osnove programiranja: programiranje – zakaj in vsaj kaj*. Ljubljana: Zavod za šolstvo, 2005.
- Uranič, S. *Microsoft C#.NET*, (online). Kranj: TŠC, 2008. (citirano 10. 10. 2008). Dostopno na naslovu: *http://uranic.tsckr.si/C%23/C%23.pdf*
- *C# Practical Learning 3.0*, (online). 2008. (citirano 1.12.2008). Dostopno na naslovu: http://www.functionx.com/csharp/
- Lokar, M., et.al. *Projekt UP – Kako poučevati začetni tečaj programskega jezika, sklop interaktivnih gradiv*, (online). 2008. (citirano 1. 12. 2008). Dostopno na naslovu: http://up.fmf.uni-lj.si/
- Lokar, M. *Wiki C#*, (online). 2008 (citirano 2.12.2008). Dostopno na naslovu http://penelope.fmf.uni-lj.si/C_sharp
- Coelho, E. *Crystal Clear Icons*, (online). 2008 (citirano 1.12.2008). Dostopno na naslovu: http://commons.wikimedia.org/wiki/Crystal_Clear
- Mayo, J. *C# Station Tutorial*, (online). 2008 (citirano 1.12.2008). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *C# Station*, (online). 2008 (citirano 1.12.2008). Dostopno na naslovu: http://www.csharp-station.com/Tutorial.aspx
- *CSharp Tutorial*, (online). 2008. (citirano 2. 12. 2008). Dostopno na naslovu: http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm
- *FunctionX Inc*, (online). 2008. (citirano 1.12.2008). Dostopno na naslovu: http://www.functionx.com/csharp/
- *SharpDevelop, razvojno okolje za C#*, (online). 2008. (citirano 7. 12. 2008). Dostopno na naslovu: http: // www.icsharpcode.net/OpenSource/SD/.
- WIKI DIRI 06/07, (online). 2008. (citirano 10. 10. 2008). Dostopno na naslovu: http://penelope.fmf.uni-lj.si/diri0607/index.php/Kategorija:Naloge

REŠITVE VAJ

OPOZORILO: Zaradi enostavnosti in krajše kode, v večini rešitev NISO uporabljeni varovalni bloki. Uporaba le–teh, z nekaj primeri, je razložena v poglavju VAROVALNI BLOKI!

METODE

1.

```
static void Main(string[] args)
{
    Console.WriteLine(Geslo(4));
    Console.ReadKey();
}

private static string Geslo(int p)
{
    // Ustvarimo generator naključnih števil
    Random naklj = new Random();
    // Določimo naključno veliko črko angleške abecede
    char znak1 = (char)naklj.Next('A', 'Z' + 1);
    // Določimo naključno malo črko angleške abecede
    char znak2 = (char)naklj.Next('a', 'z' + 1);
    //Določimo naključen samoglasnik (velika ali pa mala črka!)
    string samoglasniki = "AaEeIiOoUu";
    int stevka = naklj.Next(samoglasniki.Length);//naključno število med 0 in
število znakov v stringu samoglasniki (teh znakov je 10)
    char znak3 = samoglasniki[stevka]; //za tretji znak vzamemo naključen iz
stringa samoglasniki
    string geslo = znak1.ToString() + znak2.ToString() + znak3.ToString();
    if (p > 3)
    {
        //Nadaljnji znaki so naključne števke
        for (int i = 3; i < p; i++)
        {
            stevka = naklj.Next(6);
            geslo = geslo + stevka; //ker geslo že vsebuje znake, pretvorba
stevke v string ni več potrebna
        }
    }
    return geslo;
}
```

2.

```
static void Main(string[] args)
{
    //Klic metode
```

```
    Console.WriteLine("Inicialke: " + Inicialke("France", "Preseren"));
    Console.ReadKey();
}
public static string Inicialke(string ime, string priimek)
{
    return (ime[0] + "." + priimek[0] + ".");
}
```

3.

```
static void Main(string[] args)
{
    //Klic metode KI NARIŠE gosenico
    zadovoljnaGosenica(60);
    //Klic metode KI VRNE gosenico kot string
    Console.WriteLine(zadovoljnaGosenica1(60));

    Console.ReadKey();
}
//Metoda ki NARIŠE gosenico
public static void zadovoljnaGosenica(int n)//glava gosenice
{ //telo metode
    //najprej narišemo TELO gosenice
    for (int i = 0; i < n; i++)
        Console.Write("I");
    //narišemo še GLAVO gosenice
    Console.WriteLine(":)");
}
public static string zadovoljnaGosenica1(int n)//glava gosenice
{   //telo metode
    string telo = "";
    //najprej naredimo TELO gosenice
    for (int i = 0; i < n; i++)
        telo += "I";
    //dodamo še GLAVO gosenice in vrnemo celo gosenico
    return (telo + ":)");
}
```

4.

```
static void Main(string[] args)
{
    //Klic metode
    Console.WriteLine(Sumniki("Iz Ježce čez cesto v Stožce!"));


    Console.ReadKey();
}

private static string Sumniki(string stavek)
{
```

```
    string noviStavek = "";

    for (int i = 0; i < stavek.Length; i++)
    {
        if (stavek[i] == 'š')
            noviStavek = noviStavek + 's';
        else if (stavek[i] == 'Š')
            noviStavek = noviStavek + 'S';
        else if (stavek[i] == 'č')
            noviStavek = noviStavek + 'c';
        else if (stavek[i] == 'Č')
            noviStavek = noviStavek + 'C';
        else if (stavek[i] == 'ž')
            noviStavek = noviStavek + 'z';
        else if (stavek[i] == 'Ž')
            noviStavek = noviStavek + 'z';
        else noviStavek = noviStavek + stavek[i];
    }
    return noviStavek;
}
```

5.

```
static void Main(string[] args)
{
     //Primer klica:
     Console.WriteLine("Skupni produkt: " + Produkt(10, 20));
     Console.ReadKey();*/
}

public static long Produkt(int prvo, int drugo)
{
    if (prvo > drugo)
        return 0;
    int produkt = 1;
    while (prvo <= drugo)
    {
        if (prvo % 2 == 0)
            produkt = produkt * prvo;
        prvo++;
    }
    return produkt;
}
```

6.

```
static void Main(string[] args)
{
    Zvezdice(22, 11); //primer klica metode
    Console.ReadKey();
}
```

```
private static void Zvezdice(int M, int N)
{
    for (; M > 0; M--, N++)
    {
        for (int i = 1; i <= N; i++) Console.Write("*");
        for (int i = 1; i <= M; i++) Console.Write(".");
        Console.WriteLine();
    }
}
```

7.

```
static void Main(string[] args)
{
    Console.WriteLine(Zeller(4, 1, 1955));
    Console.ReadKey();
}

private static string Zeller(int dan, int mesec, int leto)
{
    int stol, x;
    if (mesec > 2)
        mesec = mesec - 2;
    else
    {
        mesec = mesec + 10;
        leto = leto - 1;
    }
    leto = leto % 100;
    stol = leto / 100;
    x = (13 * mesec - 1) / 5 + dan + (5 * leto) / 4 + (21 * stol) / 4;
    x = x % 7;
    Console.Write("Dan: ");
    string danVTednu;
    switch (x % 7)
    {
        case 0: danVTednu = "Nedelja";
            break;
        case 1: danVTednu = "Ponedeljek";
            break;
        case 2: danVTednu = "Torek";
            break;
        case 3: danVTednu = "Sreda";
            break;
        case 4: danVTednu = "Četrtek";
            break;
        case 5: danVTednu = "Petek";
            break;
        default: danVTednu = "Sobota";
```

```
            break;
    }
    return danVTednu;
}
```

8.

```
static void Main(string[] args)
{
    Console.Write("Ime: ");
    string ime = Console.ReadLine();//preberemo ime
    Console.WriteLine("Ugotovitev prve metode : " + Rimljan(ime));//klic prve
metode
    Console.WriteLine("Ugotovitev druge metode: " + Rimljan1(ime)); //Klic
druge metode (Rešitev s pomočjo metod Substring in Equals)

    Console.ReadKey();
}

private static string Rimljan(string ime)
{
    if (ime.Length > 3)
    {
        //ugotovimo, kakšni sta zadnji dve črki imena
        char zadnjiZnak = ime[ime.Length - 1];
        char predzadnjiZnak = ime[ime.Length - 2];
        string zadnjiDveCrki = predzadnjiZnak.ToString() +
zadnjiZnak.ToString();

        //ali
        string zadnjiCrki = (ime[ime.Length - 2]).ToString() +
(ime[ime.Length - 1].ToString());

        //Console.WriteLine(zadnjiDveCrki + "   " + zadnjiCrki);
        if (zadnjiDveCrki == "ix")
            return "Galec!";
        else
            return "Rimljan!";
    }
    else
        return "Ime je prekratko!";
}
private static string Rimljan1(string ime)
{
    //v string podniz shranimo vse znake stringa ime, od predzadnjega naprej
- torej zadnja dva znaka
    string podniz = ime.Substring(ime.Length - 2);
    podniz = podniz.ToLower();//Da ne bo težav z vel.črkami
    // Glede na ime izpišimo ustrezen tekst
    if (podniz.Equals("ix"))
```

```
        // ali if (podniz == "ix")
        return "Galec!";
    else
        return "Rimljan!";
}
```

9.

```
//Metoda
static long StClenov(int decimalk)
{
    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, decimalk) != Math.Round(Math.PI, decimalk))
    {
        clen = 4.00 / (i * 2 + 1);  //izračun tekočega člena
        if (i % 2 != 0)
            pi = pi - clen; //lihe člene odštevamo
        else
            pi = pi + clen; //sode člene prištevamo
        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");

    //ujemanje na 9 decimalk
    Console.WriteLine("Število členov: " + StClenov(9));
    Console.ReadKey();
}
```

10.

```
static void Main(string[] args)
{
    long stevilo = 1276350;
    Console.WriteLine("Staro število: " + stevilo + ", novo število: " +
NovoStevilo(stevilo));
    Console.ReadKey();
}

private static long NovoStevilo(long stevilo)
```

```
{
    long novostevilo = 0;
    int cifra, faktor = 1;
    while (stevilo > 0)
    {
        cifra = (int)stevilo % 10;
        if (cifra % 2 == 0)
        {
            novostevilo = novostevilo + cifra * faktor;
            faktor = faktor * 10;
        }
        stevilo = stevilo / 10;
    }
    return novostevilo;
}
```

TABELE

1.

```
static void Main(string[] args)
{
    double [] stevila=new double[100]; //definicija tabele
    Random naklj = new Random();//generator naključnih števil
    for (int i = 0; i < stevila.Length; i++) //tabelo napolnimo s števili
        stevila[i] = Math.Round(naklj.NextDouble() * 100,3);/*naključneo
realno število med 0 in 100, zaokroženo na 3 decimalke*/

    //izpišemo le tista števila, ki so večja od zadnjega
    double zadnje=stevila[99];
    for (int i=0;i<stevila.Length;i++)
        if (stevila[i]>zadnje)
            Console.Write(stevila[i]+"\t");//za številko še tabulator
    Console.ReadKey();
}
```

2.

```
static void Main(string[] args)
{
    //ustvarimo tabelo 100 naključnih znakov
    char[] Znaki = new char[100];
    Random naklj = new Random();//generator naključnih števil
    /*v tabelo zapišimo naključne znake iz tabele znakov, npr. znake z
zaporednimi številkami od 33 do 255*/
    for (int i = 0; i < Znaki.Length; i++)
            Znaki[i]=(char)(naklj.Next(33,256));

//ugotovimo, ali se v tabeli nahaja npr. znak '#', ter koliko je takih znakov
```

```
    int st=0;
    for (int i = 0; i < Znaki.Length; i++)
            if (Znaki[i]=='#')
                st++;
    if (st==0)
        Console.WriteLine("V tej tabeli ni nobenega znaka #");
    else
        Console.WriteLine("Znak # se v tabeli pojavi "+st+" krat.");
    Console.ReadKey();
}
```

3.

```
static void Main(string[] args)
{
    //Ustvarimo tabelo naključnih števil med 1 in 100
    int[] tabStevil = new int[100];
    Random naklj = new Random();
    for (int i = 0; i < tabStevil.Length; i++)
        tabStevil[i] = naklj.Next(1, 101);
    //definicija nove tabele
    int[] SodaLiha = new int[100];
    int indeks=0;//indeks števil v novi tabeli
    //prenos števil iz prve v drugo tabelo: najprej števila z sodimi indeksi
    for (int i = 0; i < tabStevil.Length; i = i + 2)
    {
        SodaLiha[indeks] = tabStevil[i];
        indeks++;
    }
    //prenos števil iz prve v drugo tabelo: še števila z lihimi indeksi
    for (int i = 1; i < tabStevil.Length; i = i + 2)
    {
        SodaLiha[indeks] = tabStevil[i];
        indeks++;
    }
    //še izpis obeh tabel, po 10 v vrsto
    Console.WriteLine("\n\nPrva tabela: \n");
    for (int i = 0; i < tabStevil.Length; i++)
    {
        Console.Write(tabStevil[i] + "  ");/*za vsakim število sta dva
presledka*/
        if ((i+1) % 10 == 0)
            Console.WriteLine(); /*če je že 10 števil v vrstici, gremo v novo
vrsto*/
    }
    //še izpis druge tabele
    Console.WriteLine("\n\nDruga tabela: \n");
    for (int i = 0; i < SodaLiha.Length; i++)
    {
```

```
        Console.Write(SodaLiha[i] + "  ");/*za vsakim število sta dva
presledka*/
        if ((i+1) % 10 == 0)
            Console.WriteLine(); /*če je že 10 števil v vrstici, gremo v novo
vrsto*/
    }
    Console.ReadKey();
}
```

4.

```
static void Main(string[] args)
{
    //v tabelo celih števil preberemo cela števila: dimenzija tabele je npr.5
    int[] tabStevil = new int[5];
    Console.WriteLine("Vnos števil v tabelo: ");
    for (int i = 0; i < tabStevil.Length; i++)
    {
        Console.Write("Število " + (i + 1) + ": ");
        /*ker metda ReadLine() vrača string, ga moramo spremeniti v celo
število*/
        tabStevil[i] = Convert.ToInt32(Console.ReadLine());
    }
    /*začasna logična spremenljivka: če ostane true potem gre za permutacijo,
sicer pa ne */
    bool permutacija=true;
    //preverimo, če tabela predstavlja permutacijo števil od 1 do n
    for (int i = 0; i < tabStevil.Length; i++)
    {
        if (tabStevil[i] != i + 1) /*indeks tabele je ravno za 1 manjši od
števila za parmutacijo*/
        {
            permutacija = false; //če ne gre za permutacijo zaključimo zanko
            break;
        }
    }
    if (permutacija) //če spremenljivla permutacija enaka true
        Console.WriteLine("Tabela predstavlja permutacijo števil med 1 in " +
tabStevil.Length);
    else
        Console.WriteLine("Tabela NE predstavlja permutacijo števil med 1 in
" + tabStevil.Length);
    Console.ReadKey();
}
```

5.

```
static void Main(string[] args)
{
    //tabela za shranjevanje računov
    string[] Racuni = new string[3];
```

```
    Random naklj = new Random();//generator naključnih števil
    int prvo,drugo,rezultat;
    Console.WriteLine("Izračunaj 10 produktov: \n");
    for (int i = 0; i < Racuni.Length; i++)
    {
        prvo=naklj.Next(0,10);//naključni števili med 0 in 100
        drugo=naklj.Next(0,10);
        Console.Write((i+1)+": "+prvo+"  *  "+drugo+" = ");//izpis računa
        rezultat=Convert.ToInt32(Console.ReadLine());/*uporabnikov vnos
shanimo s spremenljivko rezultat*/
        //v tabelo zapišemo račun in odgovor
        Racuni[i]=prvo+";"+drugo+";"+rezultat;
    }
    //obdelajmo tabelo (ugotovimo, koliko odgovorov je pravilnih)
    int pravilnih=0;
        for (int i = 0; i < Racuni.Length; i++)
    {
            /*metoda Split glede na ločilo ';'string npr. 3;5;15 razbije na
tri dele in jih kot stringe shrani v tabelo Stevila, vsakega v svojo celico*/
            string [] Stevila=Racuni[i].Split(';');
            prvo=Convert.ToInt32(Stevila[0]); //v 1. celici je prvo število
            drugo=Convert.ToInt32(Stevila[1]); //v 2. celici je drugo število
            rezultat=Convert.ToInt32(Stevila[2]); //v 3. celici je rezultat
            if (prvo*drugo==rezultat) //preverimo, če je rezultat pravilen
                pravilnih++;  //če je, povečamo število pravilnih odgovorov
    }
    Console.WriteLine("Pravilnih odgovorov: "+pravilnih);
    Console.ReadKey();
}
```

6.

```
static void Main(string[] args)
{
    int[] Stevila = new int[1000];
    Random naklj = new Random();//generator naključnih števil
    for (int i = 0; i < Stevila.Length; i++)
        Stevila[i] = naklj.Next(1, 1001);//naključna števila med 1 in 1000

    //Poiščimo največji in najmanjši element: zapomnimo si le njuna indeksa
    int indMax = 0;
    int indMin = 0;
    for (int i = 0; i < Stevila.Length; i++)
    {
        if (Stevila[i] > Stevila[indMax])/*če najdemo večji element, si
zapomnimo njegov indeks*/
            indMax = i;
        if (Stevila[i] < Stevila[indMin])/*če najdemo manjši element, si
zapomnimo njegov indeks*/
            indMin = i;
```

```
    }
    //izpis števil in indeksov
    Console.WriteLine("Največji element v tabeli je " + Stevila[indMax] + ",
njegov indeks pa je " + indMax);
    Console.WriteLine("Najmanjši element v tabeli je " + Stevila[indMin] + ",
njegov indeks pa je " + indMin);
    Console.ReadKey();
}
```

7.

```
static void Main(string[] args)
{
    double[] Prodaja = new double[12];//definicija tabele decimalnih števil
    Vnos(Prodaja); //klic metode za vnos podatkov tabelo
    Obdelava(Prodaja);//klic metode za obdelavo tabele
    Console.ReadKey();
}
//metoda za vnos podatkov v tabelo
private static void Vnos(double[] Prodaja)
{
    Console.WriteLine("Vnos podatkov v tabelo: ");
    for (int i = 0; i < Prodaja.Length; i++)
    {
        Console.Write("Mesec " + (i + 1) + ": ");
        Prodaja[i] = Convert.ToDouble(Console.ReadLine());
    }
}

//metoda za obdelavo tabele
private static void Obdelava(double[] Prodaja)
{
    //ugotovimo mesec, ko je bila prodaja največja
    int mesec = 0;
    for (int i = 0; i < Prodaja.Length; i++)
        if (Prodaja[i] > Prodaja[mesec])/*če najdemo mesec z večjo prodajo,
si zapomnimo njegov indeks*/
            mesec = i;
    //izpis rezultata
    string imeMeseca = "";
    switch (mesec)
    {
        case 0: imeMeseca = "januar"; break;
        case 1: imeMeseca = "februar"; break;
        case 2: imeMeseca = "marec"; break;
        case 3: imeMeseca = "april"; break;
        case 4: imeMeseca = "maj"; break;
        case 5: imeMeseca = "junij"; break;
        case 6: imeMeseca = "julij"; break;
        case 7: imeMeseca = "avgust"; break;
```

```
        case 8: imeMeseca = "september"; break;
        case 9: imeMeseca = "oktober"; break;
        case 10: imeMeseca = "november"; break;
        case 11: imeMeseca = "december"; break;
    }
    Console.WriteLine("Največja prodaja je bila v mesecu " + imeMeseca + " in
je znašala " + Prodaja[mesec] + " EUR");
}
```

8.

```
static void Main(string[] args)
{
    Random naklj = new Random();/*generator neključnih števil(če ne bomo
vnašali rezultatov)*/
    //tridimenzionalna tabela
    int[, ,] Meti = new int[5, 5, 5];
    /*prvi indeks tabele (i) predstavla številko igralca, drugi indeks (j)
predstavlja serijo, tretji indeks (k)pa zaporedno številko meta*/
    Console.WriteLine("Mečemo kocko!");
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
            for (int k = 0; k < 5; k++)
            {
                Console.Write("\nIgralec številka " + (i + 1) + ", serija " +
(j + 1) + ", met številka " + (k + 1)+": ");
                int met = Convert.ToInt32(Console.ReadLine());
                /*za ugotavljanje pravilnosti deloavnja programa lahko
namesto prejšnjih dveh stavkov uporabite naslednji stavek, kjer bo met
naključno celo število med 1 in 6*/
                //int met=naklj.Next(1,7);
                Meti[i, j, k] = met;
            }
    }
    //Obdelava tabele: za posameznega igralca seštejemo vsa števila v tabeli,
pri enakem prvem indeksu
    int suma1=0,suma2=0,suma3=0,sestice1=0,sestice2=0,sestice3=0;
    for (int j = 0; j < 5; j++)
    {
        for (int k = 0; k < 5; k++)
        {
            suma1 = suma1 + Meti[0, j, k];//suma za prvega igralca
            if (Meti[0, j, k] == 6)
                sestice1++;
            suma2 = suma2 + Meti[1, j, k];//suma za drugega igralca
            if (Meti[1, j, k] == 6)
                sestice2++;
            suma3 = suma3 + Meti[3, j, k];//suma za tretjega igralca
```

```
            if (Meti[2, j, k] == 6)
                sestice3++;
        }
    }
    //poiščemo zmaovalca
    int zmagovalec = suma1;//predpostavimo, da je zmagovalec prvi
    string ime = "prvi";
    if (suma2 > zmagovalec)
    {
        zmagovalec = suma2;
        ime = "drugi";
    }
    if (suma3 > zmagovalec)
    {
        zmagovalec = suma3;
        ime = "tretji";
    }
    Console.WriteLine("\nZmagovalec je " + ime + ". Skupaj je dosegel " +
zmagovalec + " pik!");
    //preverimo še šestice
    ime = "prvi";
    int sesticeMax = sestice1;
    if (sestice2 > sesticeMax)
    {
        ime = "drugi";
        sesticeMax = sestice2;
    }
    if (sestice3 > sesticeMax)
    {
        ime = "drugi";
        sesticeMax = sestice3;
    }
    Console.WriteLine("\nNajveč šestice je dosegel " + ime + " igralec in
sicer " + sesticeMax);
    Console.ReadKey();
}
```

9.

```
static void Main(string[] args)
{
    Console.Write("Število izdelkov, ki se bodo podražili: ");
    int n = Convert.ToInt32(Console.ReadLine());
    double [,] Cene=new double[n,3];
    Console.WriteLine("\nVnos starih cen in odstotka podražitve:");
    for (int i=0;i<n;i++)
    {
        Console.Write("Stara cena izdelka številka " + (i + 1) + ": ");
        double cena = Convert.ToDouble(Console.ReadLine());
        Console.Write("Odstotek podražitve: ");
```

```
        double odstotek = Convert.ToDouble(Console.ReadLine());
        double novaCena = cena * (1+odstotek/100);
        Cene[i, 0] = cena;//shranim staro ceno
        Cene[i, 1] = cena;//shranim odstotek podražitve
        Cene[i, 2] = cena;//shranim novo ceno
    }
    //izpis tabele
    Console.WriteLine("Stara cena        Odstotek podražitve     Nova cena");
    Console.WriteLine("-------------------------------------------------");
    for (int i = 0; i < n; i++)
    {
        //izpis na določeno število mest in na 2 decemalki
        Console.WriteLine("{0,10:F2}   {1,15:F2}   {2,20:F2}", Cene[i, 0],
Cene[i, 1], Cene[i, 2]);
    }
    Console.ReadKey();
}
```

10.

```
static void Main(string[] args)
{
    double[,] Stevila = new double[100, 5];
    for (int i = 0; i < 100; i++)
    {
        Stevila[i, 0] = i + 1;  //v prvem stolpcu je število
        Stevila[i, 1] = Math.Pow(i + 1, 2);//kvadrat
        Stevila[i, 2] = Math.Pow(i + 1, 3);//kub
        Stevila[i, 3] = Math.Sqrt(i + 1);//kvadratni koren
        Stevila[i, 4] = Math.Pow(i + 1, 1.0/3);//tretji koren
    }
    //izpis tabele
    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 5; j++)
            Console.Write(Math.Round(Stevila[i, j],3) + "\t");//izpis na 3
decimalke + tabulator
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

ZANKA Foreach

1.

```
static void Main(string[] args)
{
    int[,] tabStevil = new int[10, 10];//2-dimenzionalna tabela celih števil
    Random naklj = new Random();//generator naključnih števil
```

```
    //tabelo napolnimo z naključnimi število
    for (int i = 0; i < 10; i++)
        for (int j = 0; j < 10; j++)
            tabStevil[i, j] = naklj.Next(0, 101);/*naključna cela števila med
0 in 100*/
    Console.WriteLine("Vsota sodih števil iz ta tabele je
"+Vsota(tabStevil));//klic metode Vsota
    Console.ReadKey();
}
//metoda, ki sešteje vsa soda števila tabele tabStevil
private static int Vsota(int[,] tabStevil)
{
    int suma=0;//začetna vsota
    foreach (int stevilo in tabStevil)
    {
        if (stevilo%2==0)  //seštevamo le soda števila
            suma += stevilo;
    }
    return suma; //metoda vrne vsoto
}
```

2.

```
static void Main(string[] args)
{
    char[] tabZnakov = new char[100];
    //tabelo napolnimo z naključnimi znaki velike angleške abecede
    Random naklj = new Random();//generator naključnih števil
    for (int i = 0; i < tabZnakov.Length; i++)
    {
        int stevilo = naklj.Next(65, 92); //naključno število med 65 in 91
        tabZnakov[i] = (char)stevilo;  /*število pretvorimo v znak in
zapišemo v tabelo*/
    }

    //s pomočjo zanke foreach ugotovimo, ali se v tabeli nahaja znak 'W'
    bool obstaja = false;
    foreach (char zn in tabZnakov)
    {
        if (zn == 'W')  //če najdemo znak W
        {
            obstaja = true;  //logično spr. nastavimo na true
            break;           //in gremo iz zanke ven
        }
    }
    if (obstaja) /*spremenljivka obstaja je logična, zato ni potrebno
primerjanje*/
        Console.WriteLine("Znak W je v tej tabeli!");
    else
        Console.WriteLine("Znaka W NI tej tabeli!");
```

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

3.

```
static void Main(string[] args)
{
    int[] Koordinate = new int[10];
    /*tabelo napolnimo z naključnimi celimi števili med -20 in 20, pri čemer
se števila ne smejo ponoviti*/
    Random naklj = new Random();
    for (int i = 0; i < Koordinate.Length; i++)
    {
        int stevilo = 0;
        while (true)  /*neskončna zanka, ki se ponavlja toliko časa, da
novega števila še ni v tabeli*/
        {
            stevilo = naklj.Next(-20, 21);
            //preverimo, če je to število že v tabeli
            bool obstaja = false;
            for (int k = 0; k < Koordinate.Length; k++)/*preverimo če je novo
število že v tabeli*/
            {
                if (Koordinate[k] == stevilo)
                {
                    obstaja = true;/*če je število že v tabeli, gremo iz FOR
zanke ven*/
                    break;
                }
            }
            if (!obstaja)/*če števila v tabeli še ni, gremo iz WHILE zanke
ven*/
                break;
        }
        Koordinate[i] = stevilo;//število zapišemo v tabelo
    }
    //Izpis koordinat na premici y = 2x+3
    Console.WriteLine("Izpis koordinat točk na premici y = 2x+3\n");
    Console.WriteLine(" x      |        y");
    Console.WriteLine("-------------------");
    foreach (int x in Koordinate)
        Console.WriteLine(x + "\t|\t" + (2 * x + 3) + "\t");/*med
koordinatama in znakom | je tabulator*/
    Console.ReadKey();
}
```

4.

```
static void Main(string[] args)
{
    double[] stevila = new double[100]; //definicija tabele
```

```
    Random naklj = new Random();//generator naključnih števil
    for (int i = 0; i < stevila.Length; i++) //tabelo napolnimo s števili
        stevila[i] = naklj.Next(50,101);/*naključneo celo število med 50 in
100*/
    Console.WriteLine("Zadnje število v tabeli je " + stevila[99]+"\n");
    Console.WriteLine("Vsa števila iz te tabele ki so večja od " +
stevila[99] + " so\n");
    //izpišemo le tista števila, ki so večja od zadnjega
    foreach (int stevilo in stevila)
        if (stevilo > stevila[99])
            Console.Write(stevilo + "\t");//za številko še tabulator
    Console.ReadKey();
}
```

5.

```
static void Main(string[] args)
{
    Console.Write("Vnesi niz: ");
    string niz = Console.ReadLine();
    Console.Write("Kolikokrat razmnožim vsak znak: ");
    int kolikokrat = Convert.ToInt32(Console.ReadLine());
    string zacasni = "";
    foreach (char znak in niz)
    {
        for (int j = 0; j < kolikokrat; j++)
            zacasni = zacasni + znak;
    }
    Console.WriteLine("Novi niz: " + zacasni);
    Console.ReadKey();
}
```

6.

```
static void Main(string[] args)
{
    Console.Write("Vnesi celo število: ");
    int stevilo = Convert.ToInt32(Console.ReadLine());
    /*ustvarimo tabelo celih števil med 0 in stevilo. Tabelo potrebujemo za
obdelavo z foreach zanko*/
    int[] tabela = new int[stevilo];
    for (int i = 0; i <tabela.Length; i++)
        tabela[i] = (i+1);
    foreach (int x in tabela) //obdelajmo tabelo
    {
        int vsota = 0;
        for (int i = 1; i <=x; i++)//izračun delnih vsot
            vsota += i;
        Console.WriteLine(vsota);//izpis delne vsote
    }
    Console.ReadKey();
}
```

7.

```
static void Main(string[] args)
{
    Console.Write("Vnesi stavek: ");
    string stavek=Console.ReadLine();
    string samoglasniki="AaEeIiOoUu";//niz vseh samoglasnikov
    foreach (char znak in stavek)
    {
        if (!samoglasniki.Contains(znak))  /*preverimo če je ta znak
samoglasnik*/
            Console.Write('-');/*če tekoči znak NI samoglasnik izpišemo znak
'-', sicer pa nič!*/
    }
    Console.ReadKey();
}
```

8.

```
static void Main(string[] args)
{
    Console.Write("Koliko nizov želiš prebrati: ");
    int n = Convert.ToInt32(Console.ReadLine());
    string [] tabNizov=new string[n];//tabela nizov
        /*tabelo polnimo od zadaj naprej, sicer z zanko foreach ne bomo mogli
nizov izpisati v obratnem vrstnem redu*/
    for (int i = n-1; i >=0; i--)
    {
        Console.Write("Niz št. " + (i+1)  + ": ");
        tabNizov[i]=Console.ReadLine();
    }
    //nize izpišimo v obratnem vrstnem redu
    foreach (string niz in tabNizov)
        Console.WriteLine(niz);
    Console.ReadKey();
}
```

9.

```
static void Main(string[] args)
{
    Console.Write("Vnesi stavek: ");
    string stavek = Console.ReadLine();
    /*ker je ločilo med besedami presledek, bo metoda split vse besede
shranila v tabelo tabBesed*/
    string[] tabBesed = stavek.Split(' ');
    /*string[] tabBesed = stavek.Split(' ','.',',',':',';'); //če bi hoteli
vključiti še ostala ločila*/
    foreach (string beseda in tabBesed)
        Console.WriteLine(beseda);
    Console.ReadKey();
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
}
```

10.

```
static void Main(string[] args)
{
    Random naklj=new Random();
    int[,] tabela = new int[10, 10];
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (i > j)
                tabela[i, j] = 1;//pod diagonalo
            else if (i == j)
                tabela[i, j] = naklj.Next(-5, 6);//naključno št. med -5 in +5
            else tabela[i, j] = 0; //nad diagonalo
        }
    }
    //Preštejmo enice v tabeli
    int enic = 0;
    foreach (int stevilo in tabela)
    {
        if (stevilo == 1)
            enic++;
    }
    Console.WriteLine("Skupno število enic v tabeli je " + enic);
    Console.ReadKey();
}
```

ZBIRKE

1.

```
using System.Collections;

static void Main(string[] args)
{
    ArrayList Decimalna = new ArrayList();//netipizirana zbirka
    Random naklj = new Random();//generator naključnih števil
    int n=1000;  //skupno število podatkov
    for (int i = 0; i < n; i++)
    {
        int stevilo = naklj.Next(0, 201);//naključno število med 0 in 200
        Decimalna.Add(stevilo);//število dodamo v zbirko
    }
    //Obdelava zbirke
    int manjOd10 = 0, med10In100 = 0, vecOd100 = 0;
    foreach (int stevilo in Decimalna)
    {
```

```
        if (stevilo < 10)
            manjOd10++;
        if (stevilo >= 10 && stevilo <= 100)
            med10In100++;
        if (stevilo > 100)
            vecOd100++;
    }
    Console.WriteLine("Manjših od 10: " + manjOd10);
    Console.WriteLine("Med 10 in 100: " + med10In100);
    Console.WriteLine("Večjih od 100: " + vecOd100);
    Console.ReadKey();
}
```

2.

```
using System.Collections;

static void Main(string[] args)
{
    ArrayList Besede = new ArrayList(); //netipizirana zbirka
    Console.WriteLine("Vnesi 10 besed: ");
    for (int i = 0; i < 10; i++)
    {
        Console.Write((i + 1) + ". beseda: ");
        string beseda=Console.ReadLine();  //branje besede
        Besede.Add(beseda); //besedo damo v zbirko
    }
    //elementi zbirke so OBJEKTI, zato moramo red uporabo narediti konverzijo
(string)
    string najdaljsa = (string)Besede[0];/*predpostavimo, da je najdaljša
beseda kar prva v zbirki*/
    //obdelava zbirke - uporabimo npr. zanko for
    for (int i = 0; i < Besede.Count; i++)/*Count vrne število podatkov v
zbirki*/
    {
        if (((string)Besede[i]).Length > najdaljsa.Length)/*preverimo, če je
trenutna beseda najdaljša*/
            najdaljsa = (string)Besede[i];/če je, si jo zapomnimo
    }
    Console.WriteLine("Najdaljša beseda v zbirki: " + najdaljsa);
    Console.ReadKey();
}
```

3.

```
using System.Collections;

static void Main(string[] args)
{
    ArrayList Dohodki = new ArrayList();
    double znesek;
```

```
    Console.WriteLine("Vnesi dohodke za 12 mesecev + božičnica + regres:
\n");
    for (int i = 0; i < 12; i++)
    {
        Console.Write((i + 1) + ". mesec: ");
        znesek = Convert.ToDouble(Console.ReadLine());
        Dohodki.Add(znesek);
    }
    Console.Write("Božičnica: ");
    znesek = Convert.ToDouble(Console.ReadLine());
    Dohodki.Add(znesek);
    Console.Write("Regres: ");
    znesek = Convert.ToDouble(Console.ReadLine());
    Dohodki.Add(znesek);
    //obdelava zbirke
    double suma=0;
    foreach (double zn in Dohodki)
    {
        suma+=zn;  //krajši zapis za suma=suma+znesek;
    }
    Console.WriteLine("\nSkupni dohodki: "+suma+", povprečje na mesec:
"+Math.Round(suma/12));
    Console.ReadKey();
}
```

4.

```
using System.Collections;

static void Main(string[] args)
{
    ArrayList Nakljucna = new ArrayList();//netipizirana zbirka vseh števil
    ArrayList Soda = new ArrayList();//netipizirana zbirka sodih števil
    ArrayList Liha = new ArrayList();//netipizirana zbirka lihih števil
    Random naklj = new Random();
    for (int i=0;i<100;i++)
        Nakljucna.Add(naklj.Next(0,101)); /*v zbirko zapišemo 100 naključnih
števil med 0 in 100*/
    //soda števila prenesemo v zbirko Soda, liha pa v zbirko Liha
    foreach (int stevilo in Nakljucna)
    {
        if (stevilo%2==0)
            Soda.Add(stevilo);
        else
            Liha.Add(stevilo);
    }
    //vsebini obeh zbirk še izpišimo
    Console.WriteLine("Zbirka sodih števil: \n");
    foreach(int stevilo in Soda)
        Console.Write(stevilo+"\t");//za vsakim številom je tabulator
```

```
    Console.WriteLine("\n\nZbirka lihih števil: \n");
    foreach(int stevilo in Liha)
        Console.Write(stevilo+"\t");//za vsakim številom je tabulator
    Console.ReadKey();
}
```

5.

```
using System.Collections;

static void Main(string[] args)
{
    Console.Write("Vnesi poljuben stavek: ");
    string stavek=Console.ReadLine();
    Console.WriteLine("Najdaljša beseda v stavku: "+ Obdelaj(stavek));//klic
metode
    Console.ReadKey();
}
//metoda Obdelaj vrne najdaljšo besedo v stavku
private static string Obdelaj(string stavek)
{
    //besede iz stavka s pomočjo metode Split najprej spravimo v tabelo
stringov
    string[] Besede = stavek.Split(' ', ',', ';', ':');/*med besedami je
presledek, vejica, podpičje ali pa dvopičje*/
    ArrayList Zbirka = new ArrayList();
    Zbirka.AddRange(Besede);//celotno tabelo Besede prenesemo v zbirko
    string najdaljsa="";/*začetna vrednost najdaljše besede naj bo prazna*/
    foreach (object beseda in Zbirka)
    {
        if (((string)beseda).Length > najdaljsa.Length) //če najdem daljšo
besedo
            najdaljsa = (string)beseda; //si jo zapomnim
    }
    return najdaljsa;//metoda vrne najdaljšo besedo
}
```

6.

```
using System.Collections;

static void Main(string[] args)
{
    ArrayList Temperature = new ArrayList();//netipizirana zbirka temperatur
    Random naklj = new Random();
    const int dni=365; //skupno število dni
    for (int i=0;i<dni;i++)
    {
        /*naklj.Next(-15,16) je naključno celo število med -15 in 15
            *naklj.NextDouble() je naključno realno število med 0 in 1
            *njun produkt pa naključno realno število med -15 in +15*/
```

```
            double stevilo=naklj.Next(-
15,16)*Math.Round(naklj.NextDouble(),2);
            Temperature.Add(stevilo); //naključno temperaturo dodamo v zbirko
    }
    //obdelava zbirke
    double suma=0;  //seštevali bomo temperature
    for (int i = 0; i < Temperature.Count; i++)  /*Count pove število
elementov v zbirki*/
    {
        //ker so elementi zbirke objekti, je potrebna onverzija (double)
        suma += (double)Temperature[i];
    }
    //rezultat zaokrožimo na dve decimalki
    Console.WriteLine("\n\nPovprečna temperatura zadnjih let: " +
Math.Round(suma/dni,2));
    Console.ReadKey();
}
```

7.

```
using System.Collections;

static void Main(string[] args)
{
    //ustvarimo tabelo 50 imen
    string[] imenaZ =
{"Anja","Jerneja","Petra","Jana","Simona","Andreja","Zala","Tina","Danica","T
eja"};
    string[] imenaM=
{"Marko","Srečko","Tilen","Janez","Peter","Franci","Jože","Urban","Tomo","Gre
gor" };
    ArrayList ImenaZ=new ArrayList();
    ImenaZ.AddRange(imenaZ);//ženska imena iz tabele prenesemo v zbirko
    ArrayList ImenaM=new ArrayList();
    ImenaM.AddRange(imenaM);//moška imena iz tabele prenesemo v zbirko
    /*rezultate žrebanja bomo shranjevali v tabelo celih števil; ker je v
zbirki 10 imen, bomo žrebali številko med 0 in 9, ter ustrezne rezultate
shranjevali v tabelo*/
    int [] rezultati=new int[10];
    Random naklj=new Random();
    for (int i = 0; i < 10000; i++)//10000 žrebanj
    {
        int nakljucno = naklj.Next(0, 10);
        rezultati[nakljucno]++; //posamezne izide beležimo
    }
    int max = 0,indeks=0;
    /*poiščimo kateri element tabele je največji: ta predstavlja tudi indeks
v zbirkah ImenaZ in ImenaM*/
    for (int i = 0; i < 10; i++)
    {
```

```
        if (rezultati[i] > max)//če najdem večji izid
        {
            max = rezultati[i];//si ga zapomnim
            indeks = i;//zapomnim si tudi njegov indeks
        }
    }
    Console.WriteLine("Izžrebano žensko ime: " + (string)ImenaZ[indeks]);
    /*žrebanje bi morali sicer za tabelo moških imen ponoviti, a za namen te
naloge to ni potrebno*/
    Console.WriteLine("Izžrebano moško ime: " + (string)ImenaM[indeks]);
    Console.ReadLine();
}
```

8.

```
using System.Collections;

static void Main(string[] args)
{
    List<string> Filmi = new List<string> /*tipizirana zbirka 10 filmov in
njihovih režiserjev*/
    {
        "This Side of the Truth - Ricky Gervais and Matthew Robinson ",
        "State of Play - Kevin Macdonald",
        "Up - Pete Docter and Bob Peterson",
        "Nine - Rob Marshall",
        "The Informant - Steven Soderbergh",
        "Shutter Island - Martin Scorsese",
        "The Lovely Bones - Peter Jackson",
        "The Road - John Hillcoat",
        "Watchmen - Zack Snyder",
        "Avatar - James Cameron"
    };
    //za filme ustvarimo naključne ocene od 1 do 10
    ArrayList Ocene = new ArrayList(); //zbirka ocen
    /*za filme ustvarmo naključne ocene od 0 do 9, ocena se seveda ne sme
ponoviti*/
    Random naklj = new Random();
    for (int i = 0; i < 10; i++)
    {
        int ocena = 0;
        while (true)//neskonča zanka
        {
            ocena = naklj.Next(0, 10); //naključno število med 0 in 9
            if (!Ocene.Contains(ocena)) /*če v zbirki Ocene še ni te ocene
gremo iz zanke while ven, sicer pa na začetek while zanke*/
                break;
        }
        Ocene.Add(ocena); //oceno za i-ti film dodamo v zbirko Ocene
    }
```

```
    //izpis seznama filmov glede na ocene
    Console.WriteLine("Seznam filmov glede na ocene: \n");
    for (int i = 0; i < 10; i++)
    {
        int oc = (int)Ocene[i];/*zaporedna ocena: ker je zbirka Ocene
netipizirana, je potrebna konverzija (int)*/
        Console.WriteLine((i+1) + ". " + (string)Filmi[oc]);
    }
    Console.ReadKey();
}
```

9.

```
static void Main(string[] args)
{
    //imena učencev preberemo in jih shranimo v tipizirano zbirko stringov
    List<string> Ucenci = new List<string>();
    Console.WriteLine("Vnesi imena učencev! Zaključek vnosa je prazen
string!");
    int n=1;
    while (true) //neskončna zanka - iz nje bomo šli pri praznem vnosu
    {
        Console.Write(n + ". ime: ");
        string ime=Console.ReadLine();
        if (ime.Length != 0)//preverimo dolžino imena
            Ucenci.Add(ime); /*če ime ni prazno, ga dodamo v zbirko*/
        else break;//če je ime prazno, se zanka while zaključi
        n++;//povečamo indeks za ipis
    }
    Random naklj = new Random();
    n = 1;
    Console.WriteLine("Vrstni red spraševanja: ");
    //žrebamo tako, da iz zbirke zaporedoma žremabo imena, dokler zbirka ni
prazna
    while (Ucenci.Count != 0)/*eveda bi lahko uporabili tudi for zanko*/
    {
        int izžrebana=naklj.Next(Ucenci.Count); //žrebamo med preostalimi
učenci
        Console.WriteLine(n+". "+Ucenci[izžrebana]);//izpis izžrebanega
        Ucenci.RemoveAt(izžrebana);//izžrebanega odstranimo iz zbirke
        n++; //povečamo indeks za izpis
    }
    Console.ReadKey();
}
```

10.

```
static void Main(string[] args)
{
    Random naklj=new Random();
    List<int> Stevila = new List<int>();//tipizirana zbirka celih števil
```

```
    for (int i = 0; i < 1000; i++)//v zbirko zapišemo 1000 števil
        Stevila.Add(naklj.Next(1000));//naključno celo število med 0 in 999
    //izpis prvotne zbirke
    Console.WriteLine("Prvotna zbirka:\n");
    foreach (int stevilo in Stevila)
        Console.Write(stevilo + "\t");//izpis števila in še tabulator
    //zamenjava
    for (int i = 0; i < 500; i=i+2)
    {
        int zacasna = Stevila[i+1];
        Stevila[i + 1] = Stevila[i];
        Stevila[i] = zacasna;
    }
    //izpis nove zbirke
    Console.WriteLine("\nNova zbirka:\n");
    foreach (int stevilo in Stevila)
        Console.Write(stevilo + "\t");//izpis števila in še tabulator
    Console.ReadKey();
}
```

STRUKTURE

1.

```
struct podatki
{
    public string ime,naslov;
    public int posta;
    public string kraj, telefon;
}
static void Main(string[] args)
{
    podatki P;
    Console.Write("Ime šole: ");
    P.ime = Console.ReadLine();
    Console.Write("Naslov: ");
    P.naslov = Console.ReadLine();
    Console.Write("Poštna številka: ");
    P.posta =  Convert.ToInt16(Console.ReadLine());
    Console.Write("Kraj: ");
    P.kraj = Console.ReadLine();
    Console.Write("Telefon: ");
    P.telefon = Console.ReadLine();

    Console.WriteLine("Hodim v šolo " + P.ime + ", ki je na naslovu: " +
P.naslov + ". Telefonska številka: " + P.telefon + ", pošta: " + P.posta + ",
kraj: " + P.kraj);
    Console.ReadKey();
```

```
}
```

2.

```
static void Main(string[] args)
{
    Pravokotnik P1; //nov pravokotnik na skladu (vrednostna spremenljivka)
    Pravokotnik P2 = new Pravokotnik(); /*nov Pravokotnik na kopici
(referenčna spremenljivka - objekt)*/
    Vnos(out P1);//Ker P1 še ni inicializirana, potreben klic po referenci
out
    Vnos(out P2);//Klic po referenci
    Console.WriteLine("\nPloščina pravokotnika P1: " + P1.ploscina()); //klic
metode za izračun ploščine
    Console.WriteLine("Ploščina pravokotnika P2: " + P2.ploscina()); //klic
metode za izračun ploščine
    Console.WriteLine("Obseg pravokotnika P1: " + P1.obseg());       //klic
metode za izračun obsega
    Console.WriteLine("Obseg pravokotnika P2: " + P2.obseg());       //klic
metode za izračun obsega
    Console.ReadKey();
}
//metoda za vnos podatkov o strukturi
private static void Vnos(out Pravokotnik P)//parameter je klican po referenci
{
    Console.WriteLine("\nVnos podatkov o pravokotniku: \n");
    Console.Write("Dolžina: ");
    P.dolzina=Convert.ToInt32(Console.ReadLine());
    Console.Write("Višina: ");
    P.visina=Convert.ToInt32(Console.ReadLine());
}
```

3.

```
struct trgovina
{
    public string naziv;
    public double cena;
    public trgovina(string naziv,double cena)//konstruktor
    {
        this.naziv=naziv;
        this.cena=cena;
    }
}
static void Main(string[] args)
{
    trgovina [] Trgovine=new trgovina[2];
    //podatke preberemo
    Console.WriteLine("Podatki o trgovinah in ceni kruha: ");
    for (int i=0;i<Trgovine.Length;i++)
    {
```

```
        Console.Write("Naziv trgovine št. "+(i+1)+": ");
        string naziv=Console.ReadLine();
        Console.Write("Cena kruha: ");
        double cena=Convert.ToDouble(Console.ReadLine());
        Trgovine[i]=new trgovina(naziv,cena);//uporabimo konstruktor
    }
    Console.WriteLine("Naziv trgovine z najdražjim kruhom: " +
Najdrazji(Trgovine));
    Console.WriteLine("Povprečna cena kruha: " + Povp(Trgovine));
    Console.ReadKey();
}

private static double Povp(trgovina[] Trgovine)
{
    double suma = 0;
    for (int i = 0; i < Trgovine.Length; i++)
    {
        suma += Trgovine[i].cena;//seštevamo cene
    }
    return suma / Trgovine.Length;//vrnemo naziv trgovine z najdražjim kruhom
}

private static string Najdrazji(trgovina[] Trgovine)
{
    int indeks = 0;//predpostavimo, da je najdražji kruh kar v prvi trgovini
    for (int i = 0; i < Trgovine.Length; i++)
    {
        if (Trgovine[i].cena > Trgovine[indeks].cena)//če najdemo dražji kruh
            indeks = i;//si zapomnimo indeks trgovine
    }
    return Trgovine[indeks].naziv;//vrnemo naziv trgovine z najdražjim kruhom
}
```

4.

```
struct rabljeno
{
    public string tip;
    public int prevozeni;
    public double cena;
    public rabljeno(string tip, int prevozeni, double cena)//konstruktor
    {
        this.tip = tip;
        this.prevozeni = prevozeni;
        this.cena = cena;
    }
}
static void Main(string[] args)
{
```

```
    List<rabljeno> Vozila = new List<rabljeno>();/*tipizirana zbirka
rabljenih vozil*/
    char izbira;
    do
    {
        Console.WriteLine("\nVnos vozila-V, Izpis dražjih od 10.000 EUR-I,
Konec-K");
        Console.Write("Izbira: ");
        /*uporabnikov vnos opcije menija*/
        izbira = Convert.ToChar(Console.ReadLine().ToUpper());
        switch (izbira)
        {
            case 'V':Vnos(Vozila);break;//klic metode za vnos vozila v zbirko
            case'I':Izpis(Vozila);break;//klic metode za ipis
        }
    }
    while (izbira != 'K');
}
//metoda za vnos novega vozila v zbirko
private static void Vnos(List<rabljeno> Vozila)
{
    Console.WriteLine("\nVnos podatkov o novem vozilu:\n ");
    Console.Write("Tip vozila: ");
    string tip = Console.ReadLine();
    Console.Write("prevoženi kolimetri: ");
    int prevozeni = Convert.ToInt32(Console.ReadLine());
    Console.Write("Cena: ");
    double cena = Convert.ToDouble(Console.ReadLine());
    /*kreiramo nov objekt: uporabimo konstruktor*/
    rabljeno R = new rabljeno(tip, prevozeni, cena);
    Vozila.Add(R);//vozilo dodamo v zbirko
}
//metoda za izpis vozil dražjih od 10000 EUR
private static void Izpis(List<rabljeno> Vozila)
{
    Console.WriteLine("Izpis vseh vozil dražjih od 10.000 EUR:\n");
    foreach (rabljeno R in Vozila)
    {
        if (R.cena > 10000)
        {
            Console.WriteLine("Tip: " + R.tip + ", prevoženi km: " +
R.prevozeni + ", cena: " + R.cena);
        }
    }
}
```

5.

```
struct dijak   //struktura dijak
{
```

```
    public string naziv;
    public int[,] prvoObdobje;   //tabela 5 x 2 (pet predmetov, dve oceni)
    public int[,] drugoObdobje;  //tabela 5 x 2 (pet predmetov, dve oceni)
    public int[,] tretjeObdobje; //tabela 5 x 2 (pet predmetov, dve oceni)
    public dijak(string naziv) //prazen konstruktor
    {
        this.naziv = naziv;
        prvoObdobje = new int[5, 2];
        drugoObdobje = new int[5, 2];
        tretjeObdobje = new int[5, 2];
    }

}
static void Main(string[] args)
{
    /*ustvarimo novega dijaka na kopici: naziv bo Polonca, ocene vse enake 0
(konstruktor)*/
    dijak d = new dijak("Polonca");
    Vnos(d);//metoda za vnos naziva dijaka in vseh ocen
    Izpis(d);//izpis vseh ocen za dijaka
    Console.ReadKey();
}

private static void Izpis(dijak d)
{
    Console.WriteLine("\nIzpis ocen - "+d.naziv+"\n");
        for (int i = 0; i < 5; i++) //pet predmetov
        {
           Console.WriteLine("\n"+(i + 1) + ". predmet: ");
           for (int j = 0; j < 2; j++) //dve oceni
           {
              Console.WriteLine("\t1. ocenjevalno obdobje, "+(j+1)+". ocena:
"+d.prvoObdobje[i, j]);
              Console.WriteLine("\t2. ocenjevalno obdobje, "+(j+1)+". ocena:
"+d.drugoObdobje[i, j]);
              Console.WriteLine("\t3. ocenjevalno obdobje, "+(j+1)+". ocena:
"+d.tretjeObdobje[i, j]);
           }
        }
}

private static void Vnos(dijak d)
{
    Console.WriteLine("Vnos ocen - "+d.naziv+"\n");
    for (int i = 0; i < 5; i++) //pet predmetov
    {
        Console.WriteLine((i + 1) + ". predmet: ");
        for (int j = 0; j < 2; j++) //dve oceni za vsak predmet
        {
            Console.Write("\t1. ocenjevalno obdobje, "+(j+1) + ". ocena: ");
```

```
            d.prvoObdobje[i, j] = Convert.ToInt32(Console.ReadLine());
            Console.Write("\t2. ocenjevalno obdobje, "+(j+1) + ". ocena: ");
            d.drugoObdobje[i, j] = Convert.ToInt32(Console.ReadLine());
            Console.Write("\t3. ocenjevalno obdobje, "+(j+1) + ". ocena: ");
            d.tretjeObdobje[i, j] = Convert.ToInt32(Console.ReadLine());
        }
    }
}
```

6. uz

```
struct Complex
{
    public float realna;     //realna komponenta
    public float imaginarna; //imaginarna komponenta
    public Complex(float real, float imag)  //konstruktor
    {
        realna = real;
        imaginarna = imag;
    }
};

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+". kompleksno število:
"+IzpisiKompleksno(tabK[i]));
    }
    Console.ReadKey();
}
```

7.

```
struct ulomek
{
    public int stevec, imenovalec;
    public ulomek(int stevec, int imenovalec)//konstruktor
    {
        this.stevec = stevec;
```

```
        this.imenovalec = imenovalec;
    }
}
struct dvojniUlomek
{
    public ulomek u1, u2;//gnezdeni strukturi ulomek
    public dvojniUlomek(ulomek u1, ulomek u2)//konstruktor
    {
        this.u1 = u1;
        this.u2 = u2;
    }
}
static void Main(string[] args)
{
    Random naklj=new Random();//generator naključnih števil
    /*ustvarimo nov ulomek u1-števec in imenovalec sta naključni celi števili
med 1 in 10*/
    ulomek u1 = new ulomek(naklj.Next(1,11), naklj.Next(1,11));
    /*ustvarimo nov ulomek u2-števec in imenovalec sta naključni celi števili
med 1 in 10*/
    ulomek u2 =  new ulomek(naklj.Next(1,11), naklj.Next(1,11));
    //ustvarimo še nov dvojni ulomek
    dvojniUlomek dv1 = new dvojniUlomek(u1,u2);
    //izračunamo vrednost prvega dvojnega ulomka
    double vrednost = ((double)dv1.u1.stevec / dv1.u1.imenovalec) /
(dv1.u2.stevec / dv1.u2.imenovalec);
    Console.WriteLine("Ulomek u1: " + u1.stevec + " / " + u1.imenovalec);
    Console.WriteLine("Ulomek u2: " + u2.stevec + " / " + u2.imenovalec);
    Console.WriteLine("\nDvojni ulomek: \n    " + u1.stevec + "  \n  ---\n   "
+ u1.imenovalec+"\n  ===\n   "+u2.stevec+"\n  ---\n   "+u2.imenovalec);
    Console.WriteLine("\nVrednost dvojnega ulomka: " + vrednost);
    Console.ReadKey();
}
```

8.

```
struct Stava
{
    string ime_konja;
    double vplacana_stava;
    string razmerje;
    //konstruktor
    public Stava(string ime, double vplacano, string razmerje)
    {
        ime_konja = ime;
        vplacana_stava = vplacano;
        this.razmerje = razmerje;
    }
    public void Izpis()
    {
```

```
        Console.WriteLine("Konj:    " + ime_konja + "\n\tVplačana stava: " +
vplacana_stava + " EUR\n\tRazmerje: " + razmerje);
    }
}
static void Main(string[] args)
{
    //kreirajmo objekt: konju bo ime Divja Strela
    Stava S1=new Stava("Divja strela",3450,"2:9");
    S1.Izpis(); //klic metode Izpis
    Stava[] Konji = new Stava[5];//tabela 5 konj
    //vnos podatkov
    for (int i = 0; i < Konji.Length; i++)
    {
        Console.WriteLine("Podatki o konju št. " + (i + 1));
        Console.Write("\tIme konja: ");
        string ime=Console.ReadLine();
        Console.Write("\tVplačano: ");
        double vplacano=Convert.ToDouble(Console.ReadLine());
        Console.Write("\tStavno razmerje: ");
        string razmerje=Console.ReadLine();
        //podatek o stavi zapišemo v tabelo Konji
        Konji[i]=new Stava(ime,vplacano,razmerje);
    }
    //še izpis vseh stav
    Console.WriteLine("\nIZPIS VSEH STAV\n");
    for (int i = 0; i < Konji.Length; i++)
    {
        Konji[i].Izpis();
    }
    Console.ReadKey();
}
```

9.

```
struct Izdelek
{
string naziv;//zasebno polje
double cena,popust;
//konstruktor
public Izdelek(string naziv,double cena,double popust)
{
    this.naziv=naziv;
    this.cena=cena;
    this.popust=popust;
}
//objektna metoda za izračun cene s popustom
public double NovaCena()
{
    return cena*(1-(double)popust/100);
}
```

```
//objektna metoda za izpis podatkov o izdelku
public void Izpis()
{
    Console.WriteLine("Stara cena: "+cena+" ... popust: "+popust+" % ... nova
cena: "+NovaCena());
}
}

class Program
{
static void Main(string[] args)
{
    Izdelek[] Izdelki = new Izdelek[5]; //tabela 5 izdelkov
    Vnos(Izdelki);//klic metode za vnos izdelkov v tabelo
    Izpis(Izdelki);//klic metode za izpis tabele izdelkov
    Console.ReadKey();
}

private static void Izpis(Izdelek[] Izdelki)
{
    Console.WriteLine("Izpis izdelkov, cen in popustov");
    for (int i = 0; i < Izdelki.Length; i++)
        Izdelki[i].Izpis();
}
//metoda za vnos podatkov tabelo izdelkov
private static void Vnos(Izdelek[] Izdelki)
{
    Console.WriteLine("Vnos izdelkov v tabelo");
    for (int i = 0; i < Izdelki.Length; i++)
    {
        Console.Write((i + 1) + ". izdelek: ");
        string naziv = Console.ReadLine();
        Console.Write("\tCena: ");
        double cena = Convert.ToDouble(Console.ReadLine());
        Console.Write("\tPopust: ");
        double popust = Convert.ToDouble(Console.ReadLine());
        Izdelki[i] = new Izdelek(naziv, cena,popust);
    }
}
```

10.

```
//zbirka območij
public static List<string> obmocja=new List<string> {"LJ", "KR", "KK", "MB",
"MS", "KP", "GO", "CE", "SG", "NM", "PO"};
struct Registracija
{
    string obmocje;
    string reg;
    public Registracija(string obmocje,string reg)
```

```
    {
        this.obmocje = "";//privzeta nastavitev
        this.reg = "";//privzera nastavitev
        if (obmocja.Contains(obmocje))
            this.obmocje = obmocje.ToUpper();
        //še registerska številka
        if (reg.Length==5)
            this.reg = reg;
    }
    //metoda preveri veljavnost območja
    public bool VeljavnoObmocje(string obm)
    {
        if (obmocja.Contains(obm))
            return true;
        else return false;
    }
    //metoda za izpis registracije
    public void Izpis()
    {
        Console.WriteLine(obmocje + " " + reg);
    }
    //metoda za dostop do območja
    public string Obmocje
    {
        get { return obmocje; }
    }
}
static void Main(string[] args)
{
    Registracija R1 = new Registracija("KR", "A2345");
    R1.Izpis();
    if (R1.VeljavnoObmocje(R1.Obmocje))
        Console.WriteLine("Območje " + R1.Obmocje + " JE veljavno!");
    else
        Console.WriteLine("Območje " + R1.Obmocje + " NI veljavno!");
    Console.ReadLine();
}
```

NAŠTEVNI TIP

1.

```
enum Glasnost { Tiho=1, Srednje, Glasno };

static void Main(string[] args)
{
    Glasnost radio;
    NastaviGlasnost(out radio);//klic metode za nastavitev glasnosti
```

```
    Console.WriteLine("Nastavljena glasnost je: " + radio);/*Izpis
nastavljene glasnosti*/
    PoimenskiIzpis(); //klic metode za poimenski izpis glasnosti
    VrednostniIzpis();//klic metode za vrednostni izpis glasnosti
    Console.ReadKey();
}

//metoda za nastavitev glasnosti: parameter je klican po referenci
private static void NastaviGlasnost(out Glasnost radio)
{
    Console.Write("Nastavi ustrezno glasnost (od 1 do 3): ");
    try  //varovalni blok
    {
        /*beremo uporabnikov vnos*/
        int vnos = Convert.ToInt32(Console.ReadLine());
        if (vnos >= 1 && vnos <= 3)//pravilen vnos, zato nastavimo glasnost
            radio = (Glasnost)vnos;
        else
        {
            /*Če venešeno celo število ni bilo med 1 in 3, bo glasnost
nastavljena na tiho*/
            Console.WriteLine("Napačen vnos!. Glasnost bo nastavljena na
minimalno vrednost!");
            radio = Glasnost.Tiho;
        }
    }
    /*če je bil uporabnikov vnos napačen (vnešen je bil npr. znak) bo
glasnost nastavljena na glasno*/
    catch
    {
        Console.WriteLine("Napačen vnos! Glasnost bo nastavljena na maximalno
vrednost 1!");
        radio = Glasnost.Glasno;
    }
}
//metoda za poimenski izpis glasnosti
private static void PoimenskiIzpis()
{
    Console.WriteLine("\nPoimenski izpis glasnosti");
    for (Glasnost i = Glasnost.Tiho; i <= Glasnost.Glasno; i++)
        Console.Write(i + "  ");
}
//metoda za vrednostni izpis glasnosti
private static void VrednostniIzpis()
{
    Console.WriteLine("\nVrednostni izpis glasnosti");
    for (Glasnost i = Glasnost.Tiho; i <= Glasnost.Glasno; i++)
        Console.Write((int)i + "  ");
}
```

2.

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

public struct Delavec
{
    public string ime;
    public string priimek;
    public Izobrazba KoncanaSola;
}

static void Main(string[] args)
{
    Delavec delavec;//vrednostna spremenljivka tipa 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): ");
    //vnos izobrazbe (celo število) in pretvorba v naštevnega tip
    try  //varovalni blok
    {
        int izobr = Convert.ToInt32(Console.ReadLine());
        if (izobr >= 3 && izobr <= 9)
            delavec.KoncanaSola = (Izobrazba)izobr; //eksplicitna konverzija
iz int v VrstaIzobrazevanja
        else //če je bil vnos nepravilen, nastavimo osnovno izobrazbo
            delavec.KoncanaSola = Izobrazba.osnovna;
    }
    /*Če vnos napačen (vnešen npr. string namesto številke) priredimo osnovno
izobrazbo*/
    catch
    {
       delavec.KoncanaSola = Izobrazba.osnovna;
    }
    //izpis vseh vnešenih podatkov o delavcu
    Console.WriteLine(delavec.ime);
    Console.WriteLine(delavec.priimek);
    Console.WriteLine("Izobrazba: "+delavec.KoncanaSola);
    Console.ReadKey();
}
```

3.

```
//naštevni tip
```

```
enum zvrst { pop=1, rock, jazz, blues, country, folk, funk, punk, metal,
reggae };
struct CD
{
    public string naslov, izvajalec, založba;
    public zvrst vrsta;
    public int letnica;
    public double cena;
    //objektna metoda za izpis vseh podatkov o CD-ju
    public void Izpis()
    {
        Console.WriteLine("Podatki o CD - ju: \n");
        Console.WriteLine("Naslov: "+naslov+", izvajalec: "+izvajalec+",
zvrst glasbe: "+vrsta);
        Console.WriteLine("Založba: "+založba+", letnica: "+letnica+", cena:
"+cena);
    }
}
static void Main(string[] args)
{
    CD CD1 = new CD();// nov CD z imenom CD1
    CD CD2 = new CD();// nov CD z imenom CD2
    //meztoda za branje podatkov o CD-ju
    Vnos(ref CD1);//klic metode za vnos prvega CD-ja(klic po referenci)
    Vnos(ref CD2);//klic metode za vnos drugega CD-ja(klic po referenci)
    CD1.Izpis();  //Izpis prvega CD-ja
    CD2.Izpis();  //Izpis drugega CD-ja
    Console.ReadKey();
}
//metoda za vnos podatkov o novem CD-ju (klic po referenci)
private static void Vnos(ref CD nov)
{
    Console.WriteLine("Vnos podatkov o novem CD-ju");
    Console.Write("Naslov: ");
    nov.naslov = Console.ReadLine();
    Console.Write("Izvajalec: ");
    nov.izvajalec = Console.ReadLine();
    Console.Write("Založba: ");
    nov.založba = Console.ReadLine();
    Console.WriteLine("Glasbene zvrsti (od 1 do 10): pop, rock, jazz, blues,
country, folk, funk, punk, metal, reggae");
    Console.Write("Glasbena zvrst: ");
    int vrsta = Convert.ToInt32(Console.ReadLine()); //vnos vrednosti med 1
in 10
    nov.vrsta = (zvrst)vrsta;//konverzija v naštevni tip
    Console.Write("Leto izida: ");
    nov.letnica = Convert.ToInt32(Console.ReadLine());
    Console.Write("Cena: ");
    nov.cena = Convert.ToDouble(Console.ReadLine());
}
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

4.

```
enum vrstaRože {vrtnica=1, lilija, dalija, nagelj, iris, mešano, amarilis,
gerbera, marjetica};
enum barvaRože {rdeča=1, bela, rumena, vijoličasta, oranžna, modra, zelena};
struct enostavenŠopek
{
    public vrstaRože vrsta;
    public barvaRože barva;
    public int komadi;
    public decimal cena;//cena posamezne rože
    public enostavenŠopek(vrstaRože vrsta, barvaRože barva, int komadi,
decimal cena)
    {
        this.vrsta = vrsta;
        this.barva = barva;
        this.komadi = komadi;
        this.cena=cena;
    }
    public void Izpis()//metoda za izpis podatkov o enostavnem šopku
    {
        Console.WriteLine("Vrsta rože: " + vrsta + ", barva: " + barva + ",
število rož: " + komadi + ", cena šopka: " + komadi * cena);
    }

}

static void Main(string[] args)
{
    Console.WriteLine("Kreiranje šopka!");
    Console.Write("Vrsta rože (od 1 do 9: vrtnica, lilija, dalija, nagelj,
iris, mešano, amarilis, gerbera, marjetica): ");
    int vrsta=Convert.ToInt32(Console.ReadLine());
    vrstaRože IzbranaRoža=(vrstaRože)vrsta;
    Console.Write("Barva (od 1 do 7: rdeča, bela, rumena, vijoličasta,
oranžna, modra, zelena): ");
    int barva=Convert.ToInt32(Console.ReadLine());
    barvaRože izbranaBrava=(barvaRože)barva;
    /*enostaven šopek: vrsto in barvo rože je vnesel uporabnik, v šopku pa so
3 rože  po ceni 2.55 EUR*/
    enostavenŠopek šopek = new enostavenŠopek(IzbranaRoža, izbranaBrava, 3,
2.55m); /*m za zneskom zato, ker je cena decimal*/
    //izpis izbranega šopka
    Console.WriteLine("\nIzbrani šopek: ");
    šopek.Izpis();
    //izpis cene šopka
    Console.WriteLine("Cena izbranega šopka: " + šopek.komadi * šopek.cena+"
EUR");
    //netipizirana zbirka Šopki
```

```
    ArrayList Šopki = new ArrayList();
    //v zbirko dodajmo nekj enostavnih šopkov
    Šopki.Add(new enostavenŠopek(vrstaRože.lilija, barvaRože.rumena, 5,
1.29m));
    Šopki.Add(new enostavenŠopek(vrstaRože.gerbera, barvaRože.oranžna, 7,
3.11m));
    Šopki.Add(new enostavenŠopek(vrstaRože.vrtnica, barvaRože.rdeča, 9,
4.05m));
    //izpis zbirke Šopki in skupne cene vseh enostavnih šopkov v njej
    decimal suma = 0m;
    Console.WriteLine("\nIzpis vseh šopkov: ");
    for (int i = 0; i < Šopki.Count; i++)
    {
        ((enostavenŠopek)Šopki[i]).Izpis();
        suma = suma + ((enostavenŠopek)Šopki[i]).komadi *
((enostavenŠopek)Šopki[i]).cena;
    }
    Console.WriteLine("\nSkupna cena vseh šopkov: " + suma);
    Console.ReadKey();
}
```

RAZREDI IN OBJEKTI

1.

```
class sestkotnik
{
    public int stranica;
    public sestkotnik(int a)
    {
        stranica = a;
    }
    public int Obseg()
    {
        return 6*stranica;
    }
    public double Ploscina()
    {
        //ploščina šestkotnika je 6 ploščin enakostraničnih trikotnikov
        return 6 * ((stranica * stranica) / 4) * Math.Sqrt(3);
    }

}
static void Main(string[] args)
{
    Console.Write("Jenezek, vnesi stranico šestkotnika (celo število): ");
    int a = Convert.ToInt32(Console.ReadLine());
    sestkotnik s=new sestkotnik(a);//nov objekt s tipa sestkotnik, uporabimo
konstruktor
    Console.WriteLine("\nObseg šestkotnika: " + s.Obseg());
```

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

```
    Console.WriteLine("\nPloščina šestkotnika: " +
Math.Round(s.Ploscina(),3));/*ploščino zaokrožimo na 3 decimalke*/
    Console.Read();
}
```

2.

```
class letalo
{
    //polja so zasebna
    double doseg, maxVisina, maxHitrost;
    string tip;
    double nosilnost;
    public letalo()//pazen, privzeti konstruktor
    {}
    //preobteženi konstruktor
    public letalo(double dos, double maxV, double maxH, string tip, double
nos)
    {
        doseg = dos;
        maxVisina = maxV;
        maxHitrost = maxH;
        this.tip = tip;
        nosilnost = nos;
    }
    public double Doseg  //lastnost za polje doseg
    {
        get { return doseg; }
        set {doseg=value; }
    }
    public double MaxVisina  //lastnost za polje maxVisina
    {
        get { return maxVisina; }
        set { maxVisina = value; }
    }
    public double MaxHitrost  //lastnost za polje maxHitrost (enota je M)
    {
        get { return maxHitrost; }
        set { maxHitrost = value; }
    }
    public string Tip  //lastnost za polje tip
    {
        get { return tip; }
        set { tip = value; }
    }
    public double Nosilnost  //lastnost za polje nosilnost
    {
        get { return nosilnost; }
        set { nosilnost = value; }
    }
```

```
}
static void Main(string[] args)
{
    /*ustvarimo dva objekta in jim določimo vrednosti polj s pomočjo
konstruktorja*/
    letalo l1=new letalo(16000,12500,830,"Airbus A340",190000);
    letalo l2=new letalo(3045,12496,860,"Bombardier CRJ200 ",9400);
}
```

3.

```
enum barve{rumena,modra,rdeča,zelena,bela,črna,siva,oranžna};
class kolo
{
    int prestav, trenutnaPrestava;
    barve barva;
    double trenutnaHitrost,maxHitrost;
    int stevilo;
    public kolo()//privzeti konstruktor
    {
        prestav=3;
        trenutnaPrestava=1;
        barva=barve.bela;
        trenutnaHitrost=0;
        maxHitrost=55;
        stevilo=1;
    }
    //dodatni konstruktor za nastavljanje vseh polj
    public kolo(int prestav, int trenutnaPrestava,barve barva,double
trenutnaHitrost,double maxHitrost,int stevilo)
    {
        this.prestav=prestav;
        this.trenutnaPrestava=trenutnaPrestava;
        this.barva=barva;
        this.trenutnaHitrost=trenutnaHitrost;
        this.maxHitrost=maxHitrost;
        this.stevilo=stevilo;
    }
    //dodatni konstruktor za nastavljanje barve in maximalne hitrosti
    public kolo( barve barva, double maxHitrost)
    {
        this.prestav=18;//privzeto število prestav
        this.trenutnaPrestava=1;
        this.barva=barva;
        this.trenutnaHitrost=0;
        this.maxHitrost=maxHitrost;
        this.stevilo=1;
    }
    //1. metoda: kolo damo v višjo prestavo
```

```
    public void PrestaviGor(int n)
    {
        for (int i = 0; i < n; i++)
        {   /*navzgor lahko prestavimo tolikokrat, da pridemo do najvišje
prestave*/
            if (trenutnaPrestava < prestav)
                trenutnaPrestava++;
        }
    }
    //2. metoda: kolo damo v manjšo prestavo
    public void PrestaviDol(int n)
    {
        for (int i=0;i<n;i++)
        {   /*navzdol lahko prestavimo tolikokrat, da pridemo do 1. prestave
            if (trenutnaPrestava > 1)
                trenutnaPrestava--;
        }
    }
    //3. metoda: trenutni podatki o kolesu
    public void Izpis()
    {
        Console.WriteLine("Trenutna prestava: "+trenutnaPrestava+" od
"+prestav+", barva kolesa: "+barva);
        Console.WriteLine("Trenutna hitrost: "+trenutnaHitrost+", maximalna
hitrost: "+maxHitrost);
        Console.WriteLine("Število seb, za katere je kolo namenjeno:
"+stevilo);
    }
    //4. metoda: vrni trenutno hitrost
    public double Trenutna()
    {
        return trenutnaHitrost;
    }
    //5. metoda:  počajmo hitrost
    public void PovečajHitrost(double hitrost)
    {
        trenutnaHitrost += hitrost;
    }
}
static void Main(string[] args)
{
    Console.WriteLine("1.kolo: \n");
    //ustvarimo kolo, uporabimo privzeti konstruktor
    kolo K1 = new kolo();
    K1.Izpis();//izpis podatkov o prvem kolesu
    Console.WriteLine("\n2.kolo: \n");
    //novo kolo, uporabimo drugi konstruktor
    kolo K2 = new kolo(24, 12, barve.oranžna, 0, 80, 1);
    K2.PovečajHitrost(25);//drugo kolo pospeši za 25 km/h
    K2.PrestaviDol(4);//štiri prestave navzdol
```

```
    Console.WriteLine("Trenutna hitrost: " + K2.Trenutna());
    K2.Izpis();//izpis podatkov o drugem kolesu
    Console.WriteLine("\n3.kolo: \n");
    //novo kolo, uporabimo tretjei konstruktor
    kolo K3 = new kolo(barve.črna, 100);
    K3.Izpis();//izpis podatkov o tretjem kolesu

    Console.ReadKey();
}
```

4.

```
public class Kocka
{
    public double rob;
    public Kocka()//privzeti konstruktor
    {
        rob = 0;
    }
    public Kocka(double rob) //preobteženi konstruktor
    {
        this.rob = rob;
    }
    //objektna metoda za izračun prostornine kocke
    public double Volumen()
    {
        return (rob * rob * rob);
    }
    //objektna metoda za izračun površine kocke
    public double Povrsina()
    {
        return (6 *rob * rob);
    }
    //objektna metoda za izpis podatkov o kocki
    public void Izpis()
    {
        Console.WriteLine("KOCKA:");
        Console.WriteLine("Rob kocke: " + rob + ", prostornina: " +
Math.Round(Volumen(),2) + ", površina: " + Math.Round(Povrsina(),2));
    }
};
static void Main(string[] args)
{
    /*na kopici ustvarimo novo instanco razreda Kocka, uporabimo privzeti
konstruktor*/
    Kocka K1 = new Kocka();
    //preberemo rob kocke K1
    Console.Write("Rob kocke: ");
    K1.rob=Convert.ToDouble(Console.ReadLine());
    K1.Izpis();//klic objektne metode za izpis podatkov o kocki K1
```

```
    /*na kopici ustvarimo novo instanco razreda Kocka, uporabimo preobteženi
konstruktor*/
    Random naklj = new Random();
    Kocka K2 = new Kocka(Math.Round((naklj.NextDouble()+1)*10,3));/*rob kocke
je naključno realno število med 1 in 10 (tri decimalke)*/
    K2.Izpis();//klic objektne metode za izpis podatkov o kocki K2

    Console.ReadKey();
}
```

5. fd

```
public class Complex
{
    public float realna;
    public float imaginarna;
    public Complex() // privzeti konstruktorr
    {
        realna = 0;
        imaginarna = 0;
    }
    //preobteženi konstruktor za nastavljanje vrednosti polj
    public Complex(float real, float imag)
    {
        realna = real;
        imaginarna = imag;
     }
    //objektna metoda za izpis komponent kompleksnega števila
    public string ToString(Complex C)
    {
        return (C.realna + " + ( " + C.imaginarna + " * i )");
    }

}


public static void Main()
{
    Complex[] tabK = new Complex[10]; //tabela 10 kompleksnih števil
    Random naklj = new Random();//generator naključnih števil
    for (int i = 0; i < tabK.Length; i++)
    {
        //inicializacija tabele z naključnimi komponentami
        tabK[i] = new Complex(naklj.Next(-10, +10), naklj.Next(-10, +10));
        Console.WriteLine("Kompleksno število št " + (i + 1) + ": " +
tabK[i].ToString(tabK[i]));
    }
    Console.ReadLine();
}
```

6.

```
public class Polinom
{
        public double a, b, c;//koeficienti polinoma
        public Polinom()//privzeti konstruktor
        { }
        //preobteženi konstruktor
        public Polinom(double a, double b, double c)
        {
            this.a = a;
            this.b = b;
            this.c = c;
        }
        //Metoda za seštevanje dveh polinomov
        public Polinom Sestej(Polinom p)
        {
            Polinom zacasni=new Polinom();
            zacasni.a=a + p.a;
            zacasni.b=b + p.b;
            zacasni.c=c + p.c;
            return zacasni;
        }
        //metoda ki izračuna in vrne vrednost polinoma v točki x
        public double vrednostPolinoma(double x)
        {
            return a*x*x+b*x+c;
        }
        //metoda za izpis polinoma v obliki a * x2 + b * x + c
        public string ToString()
        {
            return a + " * x2 + " + b + " * x+ " + c;
        }
}
public static void Main(string[] args)
{
    Polinom P1=new Polinom(); //Prvi polinom - uporabimo privzeti konstruktor
    P1.a = 2;
    P1.b=3;
    P1.c=5;
    Console.WriteLine("Prvi polinom: "+P1.ToString());//Izpis prvega polinoma

    Polinom P2 = new Polinom(1.4,5.66,7.9);//Drugi polinom - //Prvi polinom -
uporabimo preobteženi konstruktor
    Console.WriteLine("Drug polinom: " + P2.ToString());//Izpis drugega
polinoma
    //S pomočjo metode Sestej objekta P1 polinomu P1 prištejmo polinom P2 in
rezultat izpišimo s pomočjo metode ToString
    Console.WriteLine("Vsota polinomov: " + P1.Sestej(P2).ToString());
    Console.ReadLine();
```

```
}
```

7.

```
class Zival
{
    string ime, vrsta;
    int steviloNog;
    public Zival()//privzeti konstruktor
    {
        ime = "Nedoloceno";
        vrsta="Nedlocena";
        steviloNog=0;
    }
    //preobteženi konstruktor
    public Zival(string ime,string vrsta,int steviloNog)
    {
        this.ime = ime;
        this.vrsta = vrsta;
        this.steviloNog = steviloNog;
    }
    public string Ime //lastnost polja ime
    {
        get { return ime;}
        set {ime=value;}
    }
    public string Vrsta //lastnost polja vrsta
    {
        get { return vrsta;}
        set {vrsta=value;}
    }
    public int SteviloNog //lastnost polja steviloNob
    {
        get { return steviloNog;}
        set {steviloNog=value;}
    }
    public override string ToString()
    {
        return ime+" je vrste "+vrsta+" in ima "+SteviloNog+" nog";
    }

}
static void Main(string[] args)
{
    //ustvarimo nov objekt Z1, uporabimo privzeti konstruktor
    Zival Z1 = new Zival();
    //Izpis podatkov o objektu Z1
    Console.WriteLine(Z1.ToString());
    //ustvarimo še nekaj objektov Z2 do Z5, uporabimo preobteženi konstruktor
    Zival Z2 = new Zival("Kanarček", "Ptica", 2);
```

```
    Zival Z3 = new Zival("Žaba", "Dvoživka", 4);
    Zival Z4 = new Zival("Črna vdova", "Pajki", 8);
    Zival Z5 = new Zival("Citronček", "Žuželke", 6);
    //Izpišimo podatke objketa Z5
    Console.WriteLine(Z5.ToString());

    Console.ReadKey();
}
```

8.

```
class Kosarka
{
    string naziv;
    int prekrski;
    int [] tocke;//tabela doseženih točk
    public Kosarka(string naziv)//konstruktor
    {
        this.naziv = naziv;
        this.prekrski = 0;//začetno število prekrškov
        /*tabela točk: prva celica meti za 1 točko, druga celica za 2 točki,
tretja pa za 3 točke*/
        tocke = new int[3];
    }
    public void ZadelProstiMet()
    {
        tocke[0]++;
    }
    public void ZadelZa2Tocki()
    {
        tocke[1] = tocke[1] + 2;
    }
    public void ZadelZa3Tocke()
    {
        tocke[2] = tocke[2] + 3;
    }
    public void Prekrsek()
    {
        prekrski++;
    }
    public void Izpis()
    {
        Console.WriteLine("\nIgralec: " + naziv);
        Console.WriteLine("\t Število prekrškov: " + prekrski);
        Console.WriteLine("\t Doseženi koši:\n\t\tza eno točko;
"+tocke[0]+"\n\t\tza dve točki: "+tocke[1]+"\n\t\tza tri točke: "+tocke[2]);
    }
}
static void Main(string[] args)
{
```

```
    Kosarka[] SLOEkipa = new Kosarka[12]{new Kosarka("Slokar"),new
Kosarka("Laković"),new Kosarka("Udrih"),new Kosarka("Bečirović"),new
Kosarka("Klobučar"),
    new Kosarka("Nachbar"),new Kosarka("Dragić"),new Kosarka("Jagodnik"),new
Kosarka("Zupan"),new Kosarka("Vidmar"),new Kosarka("Brezec"),new
Kosarka("Rizvić")};  //tabela 12 košarkašev
    //nekaj primerov klicev objektnih metod
    SLOEkipa[1].Prekrsek();//Laković je naredil mprekršek
    SLOEkipa[6].ZadelZa3Tocke();//Dragič je zadel za 3 točke
    SLOEkipa[10].ZadelZa2Tocki();//Brezec je zadel za 2 točki
    SLOEkipa[11].Prekrsek();//Rizvič je naredil mprekršek
    //klic objektne metode Izpis
    for (int i = 0; i < SLOEkipa.Length; i++)
        SLOEkipa[i].Izpis();

    Console.ReadKey();
}
```

9.

```
class pacient
{
    public string ime, priimek;
    string krvna_skupina;//zasebno polje
    public pacient()//konstruktor
    {
        ime = "ni podatkov";
        priimek = "ni podatkov";
        krvna_skupina = "ni podatkov";
    }
    //metoda za nastavitev vseh treh podatkov
    public void Nastavi(string ime,string priimek,string krvna_skupina)
    {
        this.ime = ime;
        this.priimek = priimek;
        this.krvna_skupina = krvna_skupina;
    }
    //Metoda za izpis podatkov o pacientu
    public override string ToString()
    {
        return "Pacient: \n\tIme: "+ime+"\n\tPriimek: "+priimek+"\n\tKrvna
skupina: "+krvna_skupina;
    }
    //get/set metoda
    public string Krvna_skupina
    {
        get { return krvna_skupina;}
        set { krvna_skupina = value;}
    }
}
```

```
static void Main(string[] args)
{
    //ustvarimo novega pacienta
    pacient P1 = new pacient();
    //izpis njegovih podatkov: ti še niso določeni
    Console.WriteLine(P1.ToString());
    //pacientu določimo podatke
    P1.Nastavi("Anka", "Dolinar", "AB+");
    //ponoven izpis podatkov: ti so sedaj določeni
    Console.WriteLine(P1.ToString());
    Console.ReadKey();
}
```

10.

```
class Majica
{
    int velikost;//od 1 do 5
    string barva;
    string rokavi;//dolgi ali kratki
    public Majica(int v, string b, string r)//konstruktor
    {
        if (v >= 1 && v <= 5)
            velikost = v;
        else velikost = 0;//velikost še nedoločena
        barva = b;
        if (r.ToLower() == "kratki" || r.ToLower() == "dolgi")
            rokavi = r.ToLower();
    }
    //metode
    public int VrniVelikost()
    {
        return velikost;
    }
    public void SpremeniVelikost(int velikost)
    {
        //če velikost ni v predpisani mejah, se ne spremeni
        if (velikost >= 1 && velikost <= 5)
            this.velikost = velikost;
    }
    public string VrniBarvo()
    {
        return barva;
    }
    public void SpremeniBarvo(string barva)
    {
        this.barva = barva;
    }
    public void NastaviKratkeRokave(bool imaKratkeRokave)
    {
```

```
        if (imaKratkeRokave)
            rokavi="kratki";
        else rokavi="dolgi";
    }
    public bool ImaKratkeRokave()
    {
        if (rokavi == "kratki")
            return true;
        else return false;
    }
}
static void Main(string[] args)
{
    //še primer uporabe: kreirajmo nov objekt tipa Majica
    Majica M1 = new Majica(4, "modra", "dolgi");
    M1.NastaviKratkeRokave(true);//nastavimo kratke rokave
    M1.SpremeniBarvo("rumena");//sprememba barve
    M1.SpremeniVelikost(11);//ker je velikost prevelika se ne spremeni
    M1.SpremeniVelikost(2);//sprememba velikosti OK
    Console.WriteLine("Barve majice: " + M1.VrniBarvo());
    if (M1.ImaKratkeRokave())
        Console.WriteLine("Majica ima kratke rokave!");
    else
        Console.WriteLine("Majica ima dolge rokave!");
    Console.WriteLine("Velikost majice: " + M1.VrniVelikost());
    Console.ReadKey();
}
```

DATOTEKE

1.

```
using System.IO;

static void Main(string[] args)
{
    //Datoteka LepaAnka.txt se mora nahajati v mapi kjer je .exe datoteka
    Presledki("LepaAnka.txt");/*metoda presledki dobi za parameter ime
tekstovne datoteke*/
}
//metoda ki doda presledek za vsak znak
private static void Presledki(string imeDat)
{
    try
    {
        StreamReader beri;
        StreamWriter pisi;
        /*preverimo obstoj datoteke: ta se mora nahajati v isti mapi kot
datoteke .exe*/
```

```
        if (File.Exists(imeDat))
        {
            beri = File.OpenText(imeDat);
            //preberemo kar celo datoteko naenkrat
            string vsebina = beri.ReadToEnd();
            string novaVsebina = "";
            for (int i = 0; i < vsebina.Length; i++)
                novaVsebina = novaVsebina + vsebina[i] + ' ';/*v novo vsebino
dodamo še presledek*/
            /*iz starega imena odstranim končnico .txt in dodam novo končnico
.bak*/
            pisi=File.CreateText(imeDat.Substring(0,imeDat.Length-4)+".bak");
            //v datoteko zapišem vsebino prejšnje datoteke s presledki
            pisi.WriteLine(novaVsebina);
            //zaprem obe datoteki
            beri.Close();
            pisi.Close();
        }
        else
            Console.WriteLine("Datoteka " + imeDat + " v tekočem imeniku na
obstaja!");
    }
    catch
    {
        Console.WriteLine("Napaka pri obdelavi datoteke!");
    }
}
```

2.

```
using System.IO;

static void Main(string[] args)
{
    StreamWriter pisi = null;//podatkovni tok na začetku ni inicaliziran
    try
    {
        //kreirajmo podatkovni tok za pisanje v tekstovno datoteko
        pisi = File.CreateText("Kraji.txt");
        Console.WriteLine("Vnašaj kraje, zaključek vnosa prazen vnos!");
        while (true) //neskončna zanka
        {
            Console.Write("Kraj: ");
            string kraj = Console.ReadLine();
            if (kraj.Length > 0)
            {
                Console.Write("Pošta: ");
                //ker je poštna št. celo število, je potrebna konverzija
                int posta = Convert.ToInt32(Console.ReadLine());
```

```
                pisi.WriteLine(kraj + ";" + posta);/*vpis v datoteko, ločilni
znak je podpičje*/
            }
            else
                break;//zaključek vnosa, gremo ven iz zanke
        }
        Console.WriteLine("Datoteka uspešno kreirana in shranjena v tekočo
mapo!");
        Console.ReadKey();
    }
    catch//če pride do kakršnekoli napake
    {   Console.WriteLine("Napaka pri delu z datoteko!");}
    finally
    {
        /*ne glede na to, ali smo s pisanjem uspeli ali ne, moramo zapreti
podatkovni tok*/
        pisi.Close();
    }
}
```

3.

```
using System.IO;

static void Main(string[] args)
{
    StreamWriter pisi = null;
    StreamReader beri = null;
    StreamReader beri1 = null;
    try
    {
        pisi = File.CreateText("ImenaInPriimki.txt");
        beri = File.OpenText("Imena.txt");
        beri1 = File.OpenText("Priimki.txt");
        string ime = beri.ReadLine();//skušamo brati prvo ime
        while (ime != null)
        {
            string priimek = beri1.ReadLine();//skušamo brati prvi priimek
            pisi.WriteLine(ime+" "+priimek);/*združimo ime in priimek in  ju
zapišemo v novo datoteko*/
            ime=beri.ReadLine();//skušamo brati naslednje ime
        }
    }
    catch //blok za sporočilo in obdelavo napak
    { Console.WriteLine("Napaka pri delu z datotekami!");}
    finally //zaključek varovalnega bloka
    {
        pisi.Close();
        beri.Close();
        beri1.Close();
```

```
        Console.WriteLine("Datoteka imen in priimkov je kreirana!");
    }
    Console.ReadKey();
}
```

4.

```
using System.IO;

static void Main(string[] args)
{
    StreamWriter pisi = File.CreateText("Eksponentna.txt");
    int i = 0;
    //pa naj bo npr. x enak 5
    int x = 5;
    double eNaX= 1;//začetna vsota je enaka 1
    while (true) //neskončna zanka
    {
        i++;
        double noviClen = Math.Pow(x, i) / F(i);/*novi člen, F(i) je klic
metode F za izračun faktiriele x!*/
        //izračunam razliko med sedanjo in naslednjo vsoto
        double razlika = (Math.Abs(eNaX - (eNaX + noviClen)));
        pisi.WriteLine(eNaX);//novo vsoto zapišemo v datoteko
        //če je razlika med sedanjo vsoto in naslednjo <0,0001
        if (razlika < 0.0001)
            break;//gremo iz zanke ven
        eNaX = eNaX + noviClen;//sicer pa izračunamo novo vsoto

    }
    pisi.Close();//na koncu zapremo podatkovni tok
}
//Metoda za izračun faktoriele n!
private static long F(int n)
{
    long rezultat=1;
    if (n==1)
        return 1;
    else
    {
        for (int i=1;i<=n;i++)
            rezultat=rezultat*i;
        return rezultat;
    }
}
```

5.

```
using System.IO;

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

```
{
    //podatkovni tok za pisanje - ustvarimo novo datoeko
    StreamWriter pisi = File.CreateText("OsebnaStevila.txt");
    pisi.WriteLine("Datum     Osebno število");
    DateTime datum = Convert.ToDateTime("1.1.2010");//začetni datum
    DateTime koncniDatum = Convert.ToDateTime("31.12.2010");//končni datum
    while (datum <= koncniDatum)
    {
        int dan = datum.Day;    //iz datuma izluščimo dan
        int mesec = datum.Month;//iz datuma izluščimo mesec
        int leto = datum.Year;  //iz datuma izluščimo leto
        /*osebno število je vsota vseh cifer-vsoto posameznih cifer računa
metoda VsotaCifer*/
        int osebnoStevilo = VsotaCifer(dan) + VsotaCifer(mesec) +
VsotaCifer(leto);
        /*zapis v datoteko je formatiran: datum bo izpisan na 14 mest, leva
poravnava, osebno število pa na tri mesta, desna poravnava*/
        pisi.WriteLine("{0,-
14}{1,3}",datum.ToShortDateString(),osebnoStevilo);
        datum=datum.AddDays(1);//povečamo datum za 1 dan
    }
    pisi.Close();//zapremo podatkovni tok
    Console.WriteLine("Datoteka osebnih števil je skreirana");
    Console.ReadKey();
}
//metoda, ki izračuna in vrne vsoto cifer celega števila
private static int VsotaCifer(int stevilka)
{
    int suma = 0;
    while (stevilka != 0)//dokler je še kaj cifer
    {
        /dodamo enice števila stevilka: znak % je ostanek pri celoštevilskem
deljenju*/
        suma = suma + (stevilka % 10);
        /*iz števila stevilka odstanimo enice: znak / pomeni celoštevilsko
deljenje*/
        stevilka = stevilka / 10;
    }
    return suma;
}
```

6.

```
using System.IO;

static void Main(string[] args)
{
    Console.WriteLine("Vnos telegrama!\n");
    Console.Write("Naslov telegrama (ime datoteke, končnico txt bo dodal
program sam): ");
```

```
    string imeDat=Console.ReadLine();
    /*uporabnik vnese ime telegrama, ki je obenem ime datoteke; končnico .txt
doda program sam*/
    StreamWriter pisi = File.CreateText(imeDat+".txt");
    Console.WriteLine("Vnašaj stavke + <Enter>, konec telegrama je besedica
KONEC+<Enter>\n");
    while (true)//neskončna zanka
    {
        string stavek = Console.ReadLine();
        //če uporabnik vnese "KONEC" ali pa "konec" je telegram zaključen
        if (stavek.ToUpper()!="KONEC")
            pisi.WriteLine(stavek+"STOP");/*zapis v datoteko, dodamo še
besedico "STOP"*/
        else
            break; //prekinitev zanke
    }
    pisi.Close();//zapremo datoteko
}
```

7.

```
using System.IO;

static void Main(string[] args)
{
    if (File.Exists("Manekenke.txt"))//preverimo obstoj datoteke
    {
        StreamReader beri = File.OpenText("Manekenke.txt");
        string vrstica = beri.ReadLine();//skušamo brati prvo vrstico
        string imeMax = "", imeMin = "";
        int visinaMax = 0, visinaMin = 300;/*začetno maksimalno in minimalno
višino smi izmislimo (smiselno)*/
        while (vrstica != null) //dokler so še vrstice
        {
            /*iz prebrane vrstice s pomočjo metod eSplit izluščimo posamezne
podatke*/
            string[] podatki = vrstica.Split(':');/*podatki so ločeni s
podpičjem*/
            /*višina manekenk je v celici z indeksom 0*/
            if (visinaMax < Convert.ToInt32(podatki[0]))
            {
                visinaMax = Convert.ToInt32(podatki[0]);/*zapomnim si večjo
višino*/
                //zapomnim se še ime in priimek največje menekenke
                imeMax = podatki[1] + " " + podatki[2];
            }
            if (visinaMin > Convert.ToInt32(podatki[0]))
            {
                visinaMin = Convert.ToInt32(podatki[0]);/*zapomnim si manjšo
višino*/
```

```
                //zapomnim se še ime in priimek najmanjše menekenke
                imeMin = podatki[1] + " " + podatki[2];
            }
            vrstica = beri.ReadLine();//skušamo brati naslednjo vrstico
        }
        beri.Close();//zapremo datoteko
        Console.WriteLine("Najmanjša je " + imeMin + ", ki meri " + visinaMin
+ " cm.");
        Console.WriteLine("Največja je " + imeMax + ", ki meri " + visinaMax
+ " cm.");
    }
    else
        Console.WriteLine("Datoteka ne obstaja!");
    Console.ReadKey();
}
```

8.

```
using System.IO;

static void Main(string[] args)
{
    StreamWriter pisi = File.CreateText("Stirimestna.txt");
    Random naklj = new Random();
    for (int i = 0; i < 100; i++)
    {
        pisi.WriteLine(naklj.Next(1000,10000));//naključno štirimestno
število
    }
    pisi.Close();
    //datoteko sedaj preberimo in izračunajmo poprečje
    int suma=0,n=0;
    StreamReader beri=File.OpenText("Stirimestna.txt");
    string stevilo=beri.ReadLine(); //število preberemo kot string
    while (stevilo!=null)
    {
        //konverzija v celo število in dodam k vsoti
        suma+=Convert.ToInt32(stevilo);
        n++;
        stevilo=beri.ReadLine(); //število preberemo kot string
    }
    beri.Close();
    Console.WriteLine("Povprečje vseh števil:
"+Math.Round((double)suma/n,2));
    Console.ReadKey();
}
```

9.

```
using System.IO;
```

```
static void Vnos(string imeDat)
{
    Console.Write("Ime države: ");
    string drzava = Console.ReadLine();
    //najprej vpis v tekstovno datoteko
    StreamWriter pisi = File.AppendText(imeDat + ".txt");
    pisi.Write(drzava+";");//vpis v tekstovno datoteko
    pisi.Close();//zapremo tekstovno datoteko
    //sedaj pa še v binarno: najprej ustvarimo binrani podatkovi tok
    FileStream podatkovniTok = new FileStream(imeDat + ".bin",
FileMode.Append, FileAccess.Write);
    BinaryWriter pisiBinarno = new BinaryWriter(podatkovniTok);
    pisiBinarno.Write(drzava);//zapis v binarno datoteko
    pisiBinarno.Close();//zapiranje binarnega toka
    podatkovniTok.Close();
}
public static int vsehDrzav;
public static string[] drzave;
static void Izpis(string imeDat)
{
    //najprej izpis vsebine tekstovne datoteke
    if (File.Exists(imeDat+".txt"))
    {
        Console.WriteLine("\nSeznam držav iz tekstovne datoteke:\n");
        string vsebina = File.ReadAllText(imeDat + ".txt");
        drzave = vsebina.Split(';');//vse države shranimo v tabelo drzave
        for (int i = 0; i < drzave.Length - 1; i++)/*izpišemo vse celice,
razen zadnje*/
                Console.Write((i+1)+". "+drzave[i]+"\t");
        vsehDrzav = drzave.Length - 1;
    }
    else
        Console.WriteLine("\nTekstovna datoteka ne obstaja!");
    //najprej izpis vsebine tekstovne datoteke
    if (File.Exists(imeDat + ".bin"))
    {
        Console.WriteLine("\n\nSeznam držav iz binarne datoteke:\n");
        FileStream podatkovniTok = new FileStream(imeDat +
".bin",FileMode.OpenOrCreate);
        //še binarni podatkovni tok
        BinaryReader beriBinarno = new BinaryReader(podatkovniTok);
        try  /*uporabimo varovalni blok – ko bo podatkov konec se bo program
nadaljeval za catch*/
        {
            int n = 1;
            while (true)
            {
                string drzava = beriBinarno.ReadString();/*iz datoteke berimo
string*/
                Console.Write(n+". "+drzava+"\t"); //izpis države na zaslon
```

```
                n++;
            }
        }
        catch{  }
        beriBinarno.Close(); //zapremo podatkovni tok
        podatkovniTok.Close();
    }
    else
        Console.WriteLine("\nBinarna datoteka ne obstaja!");
}
static void Brisanje(string imeDat)
{
    Izpis("Drzave");
    Console.WriteLine("\nVseh držav je " + vsehDrzav);
    Console.Write("Vnesi številko države, ki jo želiš brisari iz datoteke:
");
    int st = Convert.ToInt32(Console.ReadLine());

    if (st > 0 && st < drzave.Length - 1)
    {
        //Datoteko prepišemo z vsemi državami, razen izbrane
        StreamWriter pisi = File.CreateText(imeDat + ".txt");/*podatkovni tok
za tekstovno datoteko*/
        //še tok za binarno datoteko
        FileStream podatkovniTok = new FileStream(imeDat + ".bin",
FileMode.Create, FileAccess.Write);
        BinaryWriter pisiBinarno = new BinaryWriter(podatkovniTok);
        for (int i = 0; i < drzave.Length - 1; i++)
            if (i != (st - 1)) /*(i-1) zaradi tega, ker se v tabeli drzave
indeksi začenjajo z 0*/
            {
                pisi.Write(drzave[i] + ";");/*zapis države v tekstovno
datoteko*/
                pisiBinarno.Write(drzave[i]);//zapis v binarno datoteko
            }
        pisi.Close(); // zapremo tekstovno datoteko
        pisi.Close(); //zapremo še binarno datoteo
        podatkovniTok.Close();
    }
}
//glavni program
static void Main(string[] args)
{
    char izbira;
    do
    {
        Console.WriteLine("\n\nV-Vnos oz. dodajanje,I-Izpis,B-Brisanje,K-
Konec");
        Console.Write("Vnesi izbiro +<Enter>: ");
        izbira = char.ToUpper(Convert.ToChar(Console.ReadLine()));
```

```
        switch (izbira)
        {
            case 'V': Vnos("Drzave"); break;
            case 'I': Izpis("Drzave");break;
            case 'B': Brisanje("Drzave"); break;
        }
    }
    while (char.ToUpper(izbira) != 'K');
}
```

10.

```
using System.IO;

static void Main(string[] args)
{
    string ime = "Decimalna.bin";
    FileStream podatkovniTok = new FileStream(ime, FileMode.Create,
FileAccess.Write);
    BinaryWriter pisiBinarno = new BinaryWriter(podatkovniTok);
    Random naklj = new Random();
    for (int i = 0; i < 100; i++)
    {
        double stevilo = Math.Round(naklj.NextDouble() * 1000,2);
        pisiBinarno.Write(stevilo);
    }
    pisiBinarno.Close();
    podatkovniTok.Close();
    //datoteko sedaj obdelajmo
    podatkovniTok = new FileStream(ime, FileMode.Open, FileAccess.Read);
    BinaryReader beriBinarno = new BinaryReader(podatkovniTok);
    double suma = 0;
    int n = 0;
    try
    {
            while (true)
        {
            //iz datoteke preberemo število tipa double
            double stevilo = beriBinarno.ReadDouble();
            suma += stevilo;//povečamo vsoto vseh prebranih števil
            n++;//povečamo število prebranih števil
        }
    }
    catch {}
    Console.WriteLine("Povprečna vrednost števil v binarni datoteki:
"+Math.Round(suma/n,2));
    Console.ReadKey();
}
```

METODA Main

1.

```
static void Main(string[] args)
{
    // Testiramo, če je uporabnik vnesel vhodni parameter (število, katerega
faktorielo želi izračunati)
    if (args.Length == 0)
    {
        Console.WriteLine("Nisi vnesel celoštevilskega argumenta!");
        Console.WriteLine("Uporaba: Main-Faktoriela <celo število>");
    }
    else
    {
        // Pretvorba vhodnega argumenta v celo število:
        long num = Convert.ToInt64(args[0]);
        Console.WriteLine("Faktoriela od {0} je {1}.",num, Fakt(num));//klic
metode
    }
    Console.ReadKey();
}
//Metoda za izračun faktoriele n!
private static long Fakt(long n)
{
    long rezultat = 1;
    if (n == 1)
        return 1;
    else
    {
        for (int i = 1; i <= n; i++)
            rezultat = rezultat * i;
        return rezultat;
    }
}
```

2.

```
static void Main(string[] args)
{
    // Testiramo, če je uporabnik vnesel stavek za vhodni parameter
    if (args.Length == 0)
    {
        Console.WriteLine("Nisi vnesel stavka!");
    }
    else
    {
        /*na začetku predpostavimo, da je prva beseda obenem najkrajša in
najdaljša*/
        string najd = args[0], najkr = args[0];
```

```
        for (int i = 0; i < args.Length; i++)
        {
            if (args[i].Length > najd.Length)
                najd = args[i];
            if (args[i].Length < najkr.Length)
                najkr = args[i];
        }
        Console.WriteLine("Najdaljša beseda: " + najd);
        Console.WriteLine("Najkrajša beseda: " + najkr);
    }
    Console.ReadKey();
}
```

3.

```
static void Main(string[] args)
{
    // Testiramo, če je uporabnik vnesel dve števili
    if (args.Length != 2)
    {
        Console.WriteLine("Nisi vnesel parametrov - dveh števil!");
    }
    else
    {
        ArrayList prvaZbirka = new ArrayList();/*zbirka deliteljev prvega
števila*/
        ArrayList drugaZbirka = new ArrayList();/*zbirka deliteljev drugega
števila*/
        int prvoStevilo=Convert.ToInt32(args[0]);
        Delitelji(prvaZbirka, prvoStevilo);
        int sumaPrvi = 0, sumaDrugi = 0;
        Console.Write("Delitelji prvega števila: \n"+prvoStevilo+" = ");
        /*ker želimo za vsakim delitelje izpisati še znak '+' (razen za
zadnjim), moramo uporabiti zanko for*/
        for (int i = 0; i < prvaZbirka.Count;i++ )
        {
            if (i!=prvaZbirka.Count-1)
                Console.Write((int)prvaZbirka[i] + " + ");/*izpis delitelja
in znaka '+'*/
            else
                Console.Write((int)prvaZbirka[i]); /*za zadnjim deliteljem ni
znak '+'*/
            sumaPrvi += (int)prvaZbirka[i];
        }
        Console.WriteLine(" = "+sumaPrvi);
        int drugoStevilo = Convert.ToInt32(args[1]);
        Delitelji(drugaZbirka, drugoStevilo);
        Console.Write("\nDelitelji drugega števila: \n"+drugoStevilo+" = ");
        for (int i = 0; i < drugaZbirka.Count; i++)
        {
```

```
            if (i != drugaZbirka.Count - 1)
                Console.Write((int)drugaZbirka[i] + " + ");
            else
                Console.Write((int)drugaZbirka[i]);
            sumaDrugi += (int)drugaZbirka[i];
        }
        Console.WriteLine(" = " + sumaDrugi);
        if (sumaPrvi == drugoStevilo && sumaDrugi==prvoStevilo)
            Console.WriteLine("\nŠtevili STA prijatelski!");
        else
            Console.WriteLine("\nŠtevili NISTA prijatelski!");
    }
    Console.ReadKey();
}
//metoda, ki v zbirko Zbirka doda vse delitelje števila n
private static void Delitelji(ArrayList Zbirka, int n)
{
    for (int i = 1; i <= (n/2); i++)
        if (n % i == 0)  //če je ostanek pri deljenju enak 0
            Zbirka.Add(i); //našli smo delitelja, damo ga v zbirko
}
```

4.

```
static void Main(string[] args)
{
    if (args.Length == 0)
    {
        Console.WriteLine("Nisi vnesel parametra - višine trikotnika!");
    }
    else
    {
        int visina = Convert.ToInt32(args[0]);
        for (int i = visina; i > 0; i--)
        {// za vsako vrstico (vnešene višine)
            for (int j = i; j > 0; j--) // tekoča vrstica
                Console.Write("*");
            Console.WriteLine();
        }
    }
    Console.ReadKey();
}
```

5.

```
static void Main(string[] args)
{
    if (args.Length != 2)
    {
        Console.WriteLine("Nisi vnesel parametrov - imena in priimka");
    }
```

```
    else
    {
        string ime = Obrni(args[0]);//zmetoda zamenja znake v obratnem redu
        string priimek = Obrni(args[1]);/*zmetoda zamenja znake v obratnem
redu*/
        //predpostavimo, da je ime krajše od priimka
        string krajsi=ime, daljsi=priimek;
        if (ime.Length > priimek.Length)//sicer pa ju zamenjamo
        {
            krajsi = priimek;
            daljsi = ime;
        }
        string kombiniran = "";
        //v novo besedo dodamo izmenoma znake iz krajšega in daljšega stringa
        for (int i = 0; i < krajsi.Length; i++)
            kombiniran = kombiniran + krajsi[i] + daljsi[i];
        //dodamo še preostale znake daljšega stringa
        kombiniran = kombiniran + daljsi.Substring(krajsi.Length - 1,
(daljsi.Length - krajsi.Length));
        Console.WriteLine("Izpis: " + kombiniran);
    }
    Console.ReadKey();

}
//metoda obrne string
private static string Obrni(string beseda)
{
    string obratni="";
    for (int i = beseda.Length - 1; i >= 0; i--)
        obratni = obratni + beseda[i];
    return obratni;
}
```

6.

```
static void Main(string[] args)
{
    if (args.Length == 0)
    {
        Console.WriteLine("Nisi vnesel števila!");
    }
    else
    {
        try
        {
            long stevilo = Math.Abs(Convert.ToInt32(args[0]));
            long novoStevilo = 1;
            if (stevilo == 0)
                novoStevilo = 0;
            else
```

```
            {
                while (stevilo > 0)  //dokler je še kaj cifer
                {
                    long cifra = stevilo % 10;  //iz številke odstranimo
enice
                    if (cifra > 0)
                        novoStevilo = novoStevilo * cifra;
                    stevilo = stevilo / 10;//odrežemo enice
                }
                Console.WriteLine("Produkt neničelnih števk: " +
novoStevilo);
            }
        }
        catch
        {
            Console.WriteLine("Napaka pri pretvorbi ali pa preveliko
število!");
        }
    }
    Console.ReadKey();
}
```

7.

```
static void Main(string[] args)
{
    if (args.Length != 2)
    {
        Console.WriteLine("Nisi vnesel ustrezno število parametrov, ali pa je
vnos napačen!");
        Console.WriteLine("Stavek mora biti v navednicah, ločilo pa ne!");
        //vnos parametrov: najprej stavek (v dvojnih narekovajih, nato
presledek in končno še poljubno ločilo (npr. vejica)
        Console.WriteLine("Primer vnosa: Ločila.exe \"Jabolka, hruške,
marelice, slive in breskve!\" ,");
    }
    else
    {
        try
        {
            string stavek = args[0];//prvi argument je stavek
            char znak = Convert.ToChar(args[1]);//drugi argument je znak
            int skupaj = 0;
            for (int i = 0; i < stavek.Length; i++)//preštejemo znake
                if (stavek[i] == znak)
                    skupaj++;
            Console.WriteLine("V tem stavku je " + skupaj + " znakov " +
znak);
        }
        catch
```

```
        { Console.WriteLine("Napaka v parametrih!"); }
    }
    Console.ReadKey();
}
```

8.

```
static void Main(string[] args)
{
    if (args.Length != 2)
    {
        Console.WriteLine("Nisi vnesel ustrezno število parametrov, ali pa je
vnos napačen!");
    }
    else
    {
        try
        {
             //prvi argument je smerni koeficient premice
             int k = Convert.ToInt32(args[0]);
            int n = Convert.ToInt32(args[1]);/*prvi argument je prosti člen
premice*/
            Console.WriteLine("Premica: y = " + k + " * x + " + n);
            if (k == 0) Console.Write("Slika funkcije je vzporednica x osi in
seka os y v točki N(0," + n + ")");
            else
            {
                //Presečišče z x osjo: druga kordinata je enaka 0
                float x = (float)(-n) / k;
                Console.WriteLine("\nPresečišče z x osjo je točka M({0,3} ,
0 )", x);
                Console.WriteLine("Presečišče z y osjo je točka N(  0 ,{0,3}
)\n", n);
                Console.WriteLine("Ploščina trikotnika, ki ga tvori premica z
obema osema je "+Math.Abs(x)*Math.Abs(n)/2);
                double fi = Math.Atan(k) * (180 / Math.PI);
                Console.WriteLine("Premica seka abscisno os pod kotom " +
Math.Round(fi, 2) + " stopinj");
            }
        }
        catch
        {
            Console.WriteLine("Napaka!");
        }
    }
    Console.ReadKey();
}
```

9.

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

```
{
    if (args.Length != 1)
        Console.WriteLine("Napačno število parametrov");
    else
    {
        try
        {
            int letnica = Convert.ToInt32(args[0]);
            //če je letnica deljiva s 4 in ne s 100 ali pa je deljiva s 400
            if ((letnica % 4==0 && (letnica % 100)!=0) || (letnica % 400==0))
                Console.WriteLine("Leto " + letnica + " JE prestopno!");
            else Console.WriteLine("Leto NI prestopno!");
        }
        catch
        {
            Console.WriteLine("Napaka!");
        }
    }
    Console.ReadKey();
}
```

10.

```
static void Main(string[] args)
{
    if (args.Length == 0)
    {
        Console.WriteLine("Nisi vnesel števila!");
    }
    else
    {
        try
        {
            //poskus pretvorbe argumenta v celo številu
            int stevilo = Math.Abs(Convert.ToInt32(args[0]));
            int luknje = 0;
            //iz stevila jemljem cifre in štejem luknje
            while (stevilo != 0)
            {
                int enice = stevilo % 10; //iz števila odvzamemo enice
                switch (enice) //in pregledamo, koliko lukenj ima ta cifra
                {
                    //po eno luknjo imajo
                    case 0:
                    case 4:
                    case 6:
                    case 9: luknje++; break;
                    // cifra 8 pa ima dve luknji
                    case 8: luknje = luknje + 2; break;
                }
```

```
                /*preiskavo nadaljujemo na preostalih cifrah, zato iz števila
odstarnimo enice*/
                stevilo = stevilo / 10;
            }
            Console.WriteLine("Število lukenj v številu " +
Convert.ToInt32(args[0]) + " je " + luknje + "!");
        }
        catch { Console.WriteLine("Napaka!"); }
    }
    Console.ReadKey();
}
```

REKURZIJA

1.

```
/*Definicija zaporedja je:   a( 1 ) = 2  : prvi člen
                              a( n + 1 ) = 3 * a( n ) + 2  : ostali členi
*/
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());
    //prevrimo uporabnikov vnos
    if ((zac <= kon) && (zac != 0) && (kon != 0))
        Izpisi(zac, kon); //klic rekurtzivne metode za izpis zaporedja
    else Console.WriteLine("Napačen vnos!!!");
    Console.ReadKey();
}
//rekurzivna metoda za izračun novega člena
static int Vrni(int n)
{
    if (n == 1)
        return 2;
    else return (3 * Vrni(n - 1) + 2);
}
//rekurzivna metoda za izpis zaporedja
static void Izpisi(int zac, int kon)
{
    if (zac <= kon)
    {
        /*klic rekurzivne metode za izračun naslednjega člena zaporedja*/
        Console.Write(Vrni(zac) + "  ");
        Izpisi(zac + 1, kon);
    }
}
```

2. Izpis: 97 24 6 13

3.

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

    //število bomo prebrali kot string
    Console.Write("Vnesi poljubno število: ");
    string s = Console.ReadLine();

    Console.Write("Cifre v obratni smeri : ");
    //klic rekurzivne metode Obrat
    Obrat(s);
    Console.ReadKey();
}
//Rekurzivna funkcija za izpis poljubnega stavka v obratni smeri
static void Obrat(string str)
{
    if (str.Length > 0)//če je znakov (cifer) več od 0
        Obrat(str.Substring(1, str.Length - 1));//rekurziven klic
    else
        return;
    Console.Write(str[0]);
}
```

4.

```
static int Vsota(int n)  //rekurzivna funkcija
{
    if (n == 1)//če je člen en sam, je vsota enaka 1
        return 1;
    else if (n > 1)
        return Vsota(n - 1) +  (int)Math.Pow(n,n-1);
    else //če členov ni, je vsota 0
        return 0;
}
static void Main(string[] args)
{
    int eksponent = 0;//začetni člen vsote
    Console.Write("Koliko členov vsote želih sešteti: ");
    int clenov=Math.Abs(Convert.ToInt32(Console.ReadLine()));
    //klic rekurzivne metode vsota
    Console.WriteLine("Vsota vrste je " + Vsota(clenov));
    Console.ReadKey();
}
```

5.

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

```
{
    Console.Write("Osnova: ");
    int osnova = Convert.ToInt32(Console.ReadLine());
    Console.Write("Eksponent: ");
    int eksponent = Convert.ToInt32(Console.ReadLine());
    //rekurziven klic
    Console.WriteLine("Vrednost potence: " + Potenca(osnova, eksponent));
    Console.ReadKey();
}
//rekurzivna metoda za izračun potence
public static double Potenca(double y, int n)
{
    if (n == 0) return 1;
    if (n % 2 == 0) //če eksponent sodo število
    {
        double pom = Potenca(y, n / 2);
        return pom * pom;
    }
    return y * Potenca(y, n - 1); //če eksponent liho število
}
```

6.

```
static void Main(string[] args)
{
    Console.Write("Vnesi začetno število: ");
    int n=Convert.ToInt32(Console.ReadLine());
    Deljiva_z_8(n, 0);//klic rekurzivne metode
        Console.ReadLine();
}
//rekurzivna metoda
public static void Deljiva_z_8(int n, int suma)
{
    if (n % 8 == 0)
    {
        //izpis števi, ki nso deljiva z 8
        Console.Write(n + ", "); //izpis trenut. števila, ki je deljivo z 8
        suma += n; //vsoto povečamo za delitelja števila 8
    }
    n++;
    if (n >= 1000)
    {
        Console.Write("\nSkupna vsota števil, večjih od vnesenega števila in
manjših od 1000, ki so deljiva z 8 je " + suma);
    }
    else Deljiva_z_8(n, suma);//rekurziven klic
}
```

7.

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

```
{
    /*prvi parameter pove, do kam naj gre poštevanka, drugi pa število, ki ga
množimo*/
    Postevanka(20,15);//rekurzija( 20 krat množimo število 15)
    Console.ReadKey();
}
//rekurzivna metoda za izpis poštevanke
private static void Postevanka(int n,int stevilo)
{
    if (n == 1)
        Console.WriteLine("1\t x\t "+stevilo+"\t =\t " + stevilo);
    else
    {
        Postevanka(n - 1, stevilo);//rekurziven klic
        Console.WriteLine(n+"\t x\t "+stevilo+"\t =\t " + n*stevilo);
    }
}
```

8. Popravljena rekurzija in primer klica v glavnem programu:

```
static void Main(string[] args)
{
    const int n=4;
    Console.WriteLine("Vsota prvih " + n + " naravnih števil: " + Vsota(n));
    Console.ReadKey();
}
public static int Vsota(int n)
{
    if (n > 1)
            return n+Vsota(n - 1);
    else
        return 1;
}
```

9. Prvotna rekurzija je povsem napačna! Tule je popravljena, obenem pa še primer klica v glavnem programu: cifre se izpišejo v obratnem vrstnem redu, med njimi pa so vejice in presledek:

```
static void Main(string[] args)
{
    Console.WriteLine(KajDelam(1234567890));
    Console.ReadKey();
}
public static string KajDelam(int stevec)
{
    if (stevec <= 0)
        return "";
    else
    {
```

```
        return "" + stevec % 10 + ", " + KajDelam(stevec / 10);
    }
}
```

10. Ustavitvena pogoja metode *Puzzle* sta dva: to sta stavka

```
if (baza > limit)
    return -1;
```

in

```
if (baza == limit)
    return 1;
```

Rekurziven klic se izvede v stavku

```
return baza * Puzzle(baza + 1, limit);
```

Stavek	*Console.WriteLine(Puzzle(14, 10));*	izpiše -1
Stavek	*Console.WriteLine(Puzzle(4, 7));*	izpiše 120
Stavek	*Console.WriteLine(Puzzle(0,0));*	izpiše 1

Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.