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

Upravljanje s programirljivimi napravami

Aleš Hvasti

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: **Upravljanje s programirljivimi napravami - UPN**

Naslov učnih tem ali kompetenc, ki jih obravnava učno gradivo:
Spremenljivke v programskem jeziku, vnos podatkov in njihov prikaz na zaslonu, pogojni stavki in zanke v algoritmu, način programiranja s podprogrami, uporaba tabelaričnih spremenljivk.

Ključne besede:

Algoritem, pomnilnik, spremenljivka, podatkovni tipi, deklaracija, operatorji, izrazi, stavki, stavčni blok, konzolno okno, knjižnica, razred, metoda, pogojni stavek, zanka, argumenti metode, indeks, števec, referenca, tabela.

Avtor: Aleš Hvasti

Recenzentka: Manja Sovič Potisk

Lektorica: Maja Jošt

Izdajatelj: Konzorcij šolskih centrov Slovenije v okviru projekta MUNUS 2
Slovenija, september 2011

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 je namenjeno delu z dijaki pri modulu Upravljanje s programirljivimi napravami (UPN) in pokriva vsebinski del tega modula glede na vsebine, ki jih določa katalog znanja, v splošnem pa je uporabno pri poučevanju dijakov, ki se kot začetniki prvič srečajo z načrtovanjem programskih aplikacij.

Nastalo je na osnovi gradiva, ki sem ga imel pripravljenega kot pripomoček za izvajanje vaj pri poučevanju osnov algoritmov in programskih jezikov.

Posamezno vajo sem uporabil kot učno situacijo, skozi katero dijaka z raznimi namigi, razlagami postopkov in predstavitvami novih pojmov vodim do rešitve. Gradivo ni namenjeno samostojnemu učenju programiranja in je sestavljeno tako, da mora učitelj kot mentor pomagati dijakom priti do rešitve dane učne situacije. Ker se programiranja ne da naučiti s prepisovanjem že rešenih programov, gradivo končnih rešitev ne vsebuje, s tem pa poskuša dijaka motivirati, da se sam loti pisanja programske kode. Dijaki nato na osnovi rešitev, do katerih pridejo med poukom in s pomočjo gradiva, v katerem je opisan postopek reševanja, ponovijo in utrdijo učno snov.

Učne situacije si sledijo tako, da je v kasnejših učnih situacijah že aplicirano znanje, ki so ga dijaki pridobili v prejšnjih, zato je v takšnih primerih posredovan le namig na način reševanja, ponovne razlage pa niso vključene.

V gradivu sem kot programski jezik izbral jezik C#, za razvojno okolje pa je uporabljen Microsoftov Visual C# 2008 Express. Samo razvojno okolje je v gradivu le delno predstavljeno, več o tem si dijaki lahko ogledajo v gradivu na spletni strani http://uranic.tsckr.si/ , ki je objavljeno na strežniku TŠC Kranj.

Obravnavane teme v gradivu so vnos in izpis podatkov in njihovo shranjevanje v delovnem pomnilniku, oblikovanje izpisa podatkov v konzolnem oknu, pisanje izrazov in uporaba matematičnih funkcij v izrazih, generiranje naključnih vrednosti, pisanje algoritmov, ki vsebujejo pogojne stavke in zanke, delo s tekstovnimi podatki, načini programiranja s podprogrami in postopki pri komuniciranju med podprogrami ter shranjevanje podatkov v obliki tabel.

Dijakom in tudi ostalim uporabnikom gradiva želim čim več uspeha pri delu in učenju.

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.

VNOS IN IZPIS PODATKOV
– uporaba standardne vhodne in izhodne enote

Računalnik bi radi uporabili kot napravo, ki bi za nas rešila določeno nalogo. Vzemimo, da bi želeli, da računalnik za nas reši preprost elektrotehniški primer, na primer izračun skupne upornosti dveh vzporedno vezanih uporov.
Naučili ga bomo, da bo od nas najprej zahteval vnos podatkov, to sta upornosti obeh uporov. Te bomo vnesli preko tipkovnice, računalnik pa jih bo shranil v svoj pomnilnik. Nato ga bomo naučili, kako se izračuna skupna upornost, na koncu pa bi želeli, da nam dobljeni rezultat še prikaže.
Na ta način lahko računalnik pripravimo za reševanje marsikatere podobne in tudi zahtevnejše naloge in ga uporabljamo kot zelo koristen in priročen pripomoček.

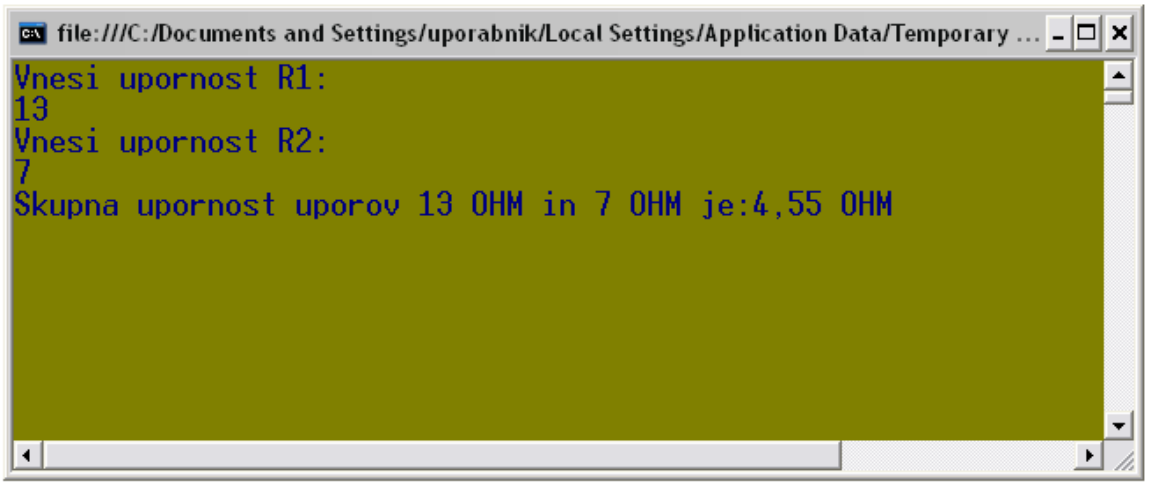
Pri vaji bomo spoznali metodi za vnos podatkov s tipkovnice in vpis v pomnilnik in izpis podatkov v konzolno (pogovorno) okno. V programskem jeziku C# sta to metoda *ReadLine()* za vnos in metoda *WriteLine()* za izpis. Pri uporabi teh dveh metod si bomo ogledali:

- deklaracijo nove spremenljivke
- decimalni podatkovni tip
- uporabo nekaterih osnovnih operatorjev
- uporabo pogovornega okna
- uporabo okna s spiskom napak
- oblikovanje programske kode v tekstovni obliki
- uporabo tekstovnega urejevalnika in nekaterih pripomočkov, ki jih vsebuje
- pisanje komentarjev.

Metodi *ReadLine()* in *WriteLine()* najdemo v razredu ***Console***.

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.

Napiši program za izračun skupne upornosti dveh vzporedno vezanih uporov.
Uporabnik vnese vrednosti obeh uporov, program pa na koncu izpiše izračunano vrednost.



Slika 1: prikaz zgleda konzolnega okna

Pričetek dela v konzolnem načinu

Ko odpremo nov projekt v konzolnem načinu, se nam v tekstovnem urejevalniku, ki je namenjen pisanju programske kode, pojavi osnutek programske kode. Ta vsebuje vključitev nekaterih *knjižnic* ter deklaracije *imenskega prostora*, *razreda* in glavne metode *Main()*.

```
using System;                          //vključitev knjižnic
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1          //deklaracija imenskega
                                       //prostora
{
    class Program                      //deklaracija razreda
    {
        static void Main(string[] args)//deklaracija
                                              //glavne metode
        {
            //mesto, kjer začnemo s pisanjem kode
        }
    }
}
```

Metode in razredi

Program, ki ga pišemo, predstavlja nek nov razred (*class*). Vsak razred se nahaja v nekem imenskem prostoru oz. knjižnici (*namespace*). Znotraj razreda pa se nahajajo metode za izvedbo posameznih nalog in glavna metoda *Main*, ki je edina obvezna. Zaenkrat se bomo zadovoljili s tem, da bomo vse postopke (celotno programsko kodo) napisali v metodi *Main*.

Deklaracija nove spremenljivke

V programu podatki nastopajo v obliki spremenljivk. Z deklaracijo spremenljivke v pomnilniku rezerviramo prostor za nek podatek. Vrednost tega podatka se lahko med izvajanjem programa spreminja.

```
double R1 = 0;          //primer deklaracije spremenljivke
double Rs = 0;
```

- Prva beseda pove tip spremenljivke, s katerim opišemo vrsto podatka (celo število, decimalno število, znak ...).
- Na drugem mestu sledi ime spremenljivke (prevajalnik je občutljiv na uporabo velikih ali malih črk).
- Sledi operator = , ki predstavlja operacijo vnosa vrednosti v pomnilnik (RAM).
- Za operatorjem navedemo še vrednost (ta je lahko navedena kot konstanta ali pa napišemo izraz, po katerem se vrednost izračuna).

V našem primeru bomo uporabljali decimalna števila, zato bomo uporabljali spremenljivke tipa *double*. Potrebovali bomo 3 takšne spremenljivke, eno za R1, drugo za R2 in tretjo za Rs. Imena spremenljivk izberemo sami, običajno tako, da nam ta imena povedo nekaj o vrednostih, ki se skrivajo za njimi. Imena spremenljivk se ne smejo podvajati!
Ko je spremenljivka enkrat deklarirana, jo lahko uporabljamo, kar pomeni, da pred njenim imenom ne navajamo več podatkovnega tipa. Deklaracijo vsake spremenljivke naredimo samo enkrat!

Pisanje stavkov in izrazov

Za izračun upornosti napišemo izraz, v katerem uporabljamo različne operatorje (npr. + za seštevanje, * za množenje, / za deljenje …). Izraz za izračun skupne upornosti zapišemo v stavku:

*Rs = (R1 * R2) / (R1 + R2);*

- Na začetku zapišemo ime spremenljivke, ki prevzame izračunano vrednost. Pod imenom Rs se bo izračunana vrednost shranila v pomnilnik.
- Z oklepaji določimo prioriteto operacije.
- Pri deljenju moramo biti pozorni na tip spremenljivke. Pri deljenju dveh celih števil bo tudi rezultat deljenja celo število, zato je priporočljiva uporaba spremenljivk tipa *double*.
- Stavek vedno zaključimo s podpičjem.

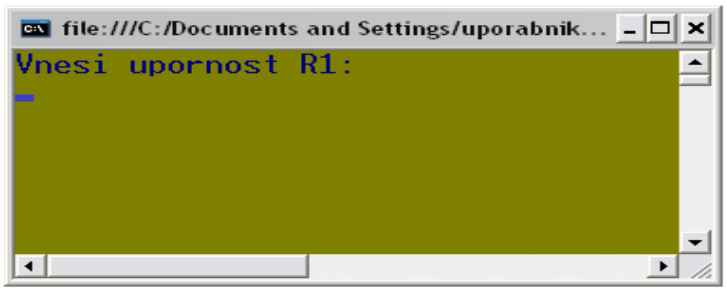
Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.

Pogovorno oziroma konzolno okno

Vnos podatka opravimo s stavkom:

```
R1 = Convert.ToDouble(Console.ReadLine());
```

Metoda *Console.ReadLine()* pobere podatek s tipkovnice v obliki teksta (znakovni niz oz. *string*). Ker mi potrebujemo številčnim podatek (*double*), moramo prebrani tekst pretvoriti v številko. To storimo s pomočjo ustrezne metode, ki jo najdemo v razredu *Convert*. (Metoda *ToDouble()* pretvarja v podatkovni tip *double*, imamo pa še druge metode npr. metodo *ToInt32()*, ki pretvarja v celoštevilčni tip *int*…)
Pri izvršitvi metode *Console.ReadLine()* se odpre pogovorno okno, v katerega vnesemo vrednost. Izvajanje programa se zaustavi in program čaka na vnos podatkov.



Slika 2: konzolno okno čaka na vnos podatka

Vnos se konča s pritiskom na <ENTER> in izvajanje programa se nadaljuje. Pred vnosom podatkov je priporočljivo uporabniku podati navodila, kaj pričakujemo od njega. To storimo z izpisom teksta na zaslon. S stavkom

```
Console.WriteLine("Vnesi vrednost za upor R1: ");
```

dosežemo izpis teksta, ki je v narekovajih, v pogovorno okno. Postopek ponovimo še za upor R2.
Izpis rezultata naredimo s stavkom

```
Console.WriteLine(Rs);
```

Podatki se na zaslonu prikažejo v pogovornem oknu, ki pa se takoj samodejno zapre, če ga ne zaustavimo. To lahko storimo z metodo za vnos podatkov (npr. *ReadLine()*) in prikazane podatke lahko preberemo. Priporočljivo je izpis opremiti s tekstom

```
Console.WriteLine("Skupna upornost uporov "+R1+" in "+R2+"
je:"+Rs);
```

Argument metode *WriteLine()*, ki ga navedemo v oklepajih, mora biti v obliki teksta, operator + pa na tem mestu sestavlja posamezne dele v celoto (ne predstavlja operacije seštevanja).

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.

Napiši program za izračun in izpis površine in volumna kvadra. Uporabnik vnese velikost vseh treh stranic.
Program opremi z ustreznimi komentarji.

Komentarji

Komentarji so del programske kode, ki se ne prevajajo v izvršilno obliko. Namenjeni so pojasnjevanju in označevanju določenih delov programske kode. Na začetku programske kode kot komentar lahko navedemo nekatere podatke o programu in avtorjih …

```
double c = Convert.ToDouble(Console.ReadLine());
//vnos nekega števila
```

Zgoraj je primer enovrstičnega komentarja, ki pojasnjuje, kaj naredi zgornji stavek. Začetek komentarja označimo z dvema poševnicama, konec pa določimo s pritiskom na <ENTER>, ki nas pomakne v novo vrstico.

```
    double c = Convert.ToDouble(Console.ReadLine());
/*Zgornji stavek predstavlja deklaracijo decimalne
spremenljivke in vnos nekega podatka, ki ga pretvorimo v
decimalno število. Podatekm se pod imenom c shrani v RAM.*/
```

Zgoraj imamo primer večvrstičnega komentarja.

1. Kaj so spremenljivke in zakaj jih potrebujemo?
2. Kaj moramo navesti pri deklaraciji spremenljivke?
3. Kje se nahajajo metode in kateri operator moramo uporabiti, da pridemo do določene metode?
4. Katero metodo uporabljamo za vnos podatkov na standardnem vhodu in katero za izpis podatkov na standardni izhod?
5. Kateri operator pomeni shranjevanje vrednosti v pomnilnik?
6. Čemu so namenjeni komentarji in kako jih obravnava prevajalnik?

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.

OBLIKOVANJE IZPISA V KONZOLNEM OKNU – vsebina razreda Console

Ko računalnik uporabljamo kot orodje, nam mora biti vedno razumljivo, kaj v danem trenutku od nas pričakuje. Pri tem si pomagamo z besedilom, ki ga izpisujemo na zaslon. V nekaterih primerih moramo izpisati večje število podatkov in bi zato želeli njihov prikaz narediti preglednejši. V takšnem primeru lahko uporabimo različne barve ozadja in pisave, na podatek opozorimo z zvočnim signalom, brišemo nepotrebne podatke … Določimo lahko tudi mesto izpisa podatka ali nekega besedila in oblikujemo preprosto preglednico.

Pri tej vaji bomo spoznali nekaj načinov za oblikovanje izpisa v konzolnem pogovornem oknu. Pomagali si bomo z metodami razreda *Console*. Pri uporabi teh metod si bomo ogledali:

- določanje širine polja za izpis
- določanje poravnave izpisa v polju
- določanje števila decimalnih mest
- določanje izgleda pogovornega okna
- postopek za postavitev začetka izpisa na katero koli mesto v oknu
- brisanje vsebine pogovornega okna
- vnos celega in decimalnega števila
- ASCII kodo znakov
- neposredno pretvorbo med podatkovnimi tipi

Napiši program, ki prebere dve celi števili, nato pa izračuna njuno vsoto, njuno razliko, njun produkt in njun količnik ter dobljene vrednosti izpiši. Prikaz podatkov oblikuj tako, da zamenjaš

- **barvo ozadja,**
- **barvo teksta,**
- **napis na konzolnem oknu,**
- **določiš velikost okna,**
- **določiš število mest za izpisane vrednosti in izbereš poravnavo (desno ali levo).**

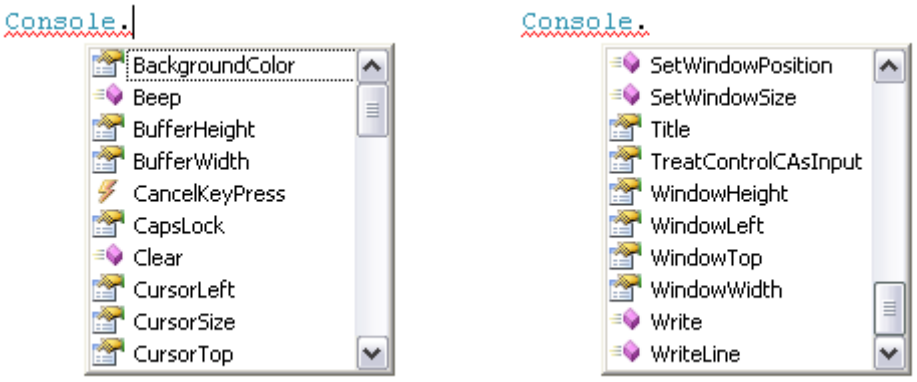
Na koncu s pritiskom na <ENTER> pobrišemo vsebino okna. Konzolno okno naj zgleda približno tako:

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 3: prikaz rezultatov

Vsebina razreda *Console*

V razredu *Console* imamo zbrane metode in lastnosti za oblikovanje konzolnega okna in njegove vsebine. Metodi za vnos in izpis podatkov smo že spoznali, sedaj pa si na seznamu oglejmo še ostalo vsebino (vijolična kocka predstavlja metodo, list z roko pa lastnost).



Slika 4: vsebina razreda Console

Argumente metodam vedno podajamo v oklepajih, ki sledijo imenu metode, lastnostim pa vedno lahko podamo le en argument, ki pa ga ne pišemo v oklepajih, ampak uporabimo operator = .

Barva ozadja in barva teksta

V konzolnem oknu lahko izbiramo med 16 različnimi barvami, standardna nastavitev pa je bel tekst na črni podlagi. Barvi lahko zamenjamo z lastnostma *BackgroundColor* (ozadje) in *ForegroundColor* (tekst).

```
Console.BackgroundColor = ConsoleColor.DarkYellow;
Console.ForegroundColor = ConsoleColor.DarkBlue;
Console.Clear();
```

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.

Če na koncu uporabimo še metodo *Clear()*, se bo obarvalo celotno okno, sicer pa le del neposredno pod tekstom.

Velikost konzolnega okna

Standardna velikost konzolnega okna je 80 stolpcev in 24 vrstic. Velikost lahko programsko spreminjamo z lastnostjo *SetWindowSize*.

```
Console.SetWindowSize(80,10);
```

V zgornjem primeru smo število vrstic spremenili na 10, število stolpcev pa je ostalo 80.

Napis na konzolnem oknu

Določimo ga lahko z lastnostjo *Title*.

```
Console.Title = "VSOTA, RAZLIKA, ZMNOŽEK IN KOLIČNIK DVEH
ŠTEVIL";
```

Vnos podatkov

Do sedaj smo vnašali le decimalna števila (*double*), tokrat pa bi radi delali s celimi števili. Potrebovali bomo spremenljivko tipa *int*.

```
Console.WriteLine("Vnesi prvo število: ");   //navodilo…
                                             //…uporabniku
int a = Convert.ToInt32(Console.ReadLine());
```

Pri pretvorbi vnesenega niza znakov v ustrezen podatkovni tip smo uporabili metodo *ToInt32()*, ki ustreza podatkovnemu tipu *int*. (Številka 32 pomeni število bitov, ki jih spremenljivka zasede v pomnilniku.) Metoda *ToInt16()* ustreza tipu *short*, ki se uporablja za manjša števila, metoda *ToInt64()* pa tipu *long*, ki ga uporabimo za delo z zelo velikimi števili.

Izpis izračunanih vrednosti

Oglejmo si formatiran način izpisa v konzolno okno. Na voljo imamo metodi *Write()* in *WriteLine()*. Pri prvi se izpis nadaljuje v isti vrstici, pri drugi pa na koncu izpisa skočimo v novo vrsto. Argument je pri obeh metodah nek tekst, zato ga označimo z dvojnim narekovajem. V tekst pa lahko vrivamo vrednosti spremenljivk. Mesto, kjer bomo vrivali, označimo z zavitimi oklepaji {}, spremenljivke, ki bodo uporabljene, pa naštejemo za tekstom in jih ločimo z vejicami. Oglejmo si primer:

```
int ena = 1, dva = 2, tri = 3;     //deklariramo tri
                                   //spremenljivke
Console.Write("Prva vrednost je {0}, druga vrednost je
{1}, tretja vrednost je {2}.", ena, dva, tri);
```

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.

V zavite oklepaje vpišemo zaporedno številko spremenljivke glede na vrstni red, v katerem so navedene. Spremenljivka ena ima zaporedno številko 0, spremenljivka dva ima številko 1… V našem primeru bi izpis izgledal takole:

```
Console.WriteLine("Vsota: {0,-8} razlika: {1,-8} zmnožek:
{2,-8} količnik: {3,-8}", a+b, a-b, a*b, kolicnik);
```

V zavitih oklepajih je dodan še en parameter, ki predstavlja število mest, ki so na voljo za izpis vrednosti (v našem primeru 8). Če pred ta parameter dodamo znak -, to pomeni levo poravnavo znotraj rezerviranega števila mest, če znaka ni, pa to pomeni desno poravnavo. Za tekstom smo namesto spremenljivk navedli kar izraze za izračun vsote, razlike in zmnožka, razen pri količniku.

Izračun količnika

Rezultat deljenja je običajno neko decimalno število. Operacija deljenja pa je v jeziku C# izvedena tako, da pri deljenju dveh celih števil nikoli ne dobimo decimalk (odreže jih, ne zaokrožuje!!). Decimalke dobimo, če med seboj delimo decimalna števila
Če v izrazu nastopa vsaj eno decimalno število, se vsa ostala cela števila v izrazu avtomatično pretvorijo v decimalna in rezultat bo imel decimalke.
Zato v našem primeru eno od števil pretvorimo v decimalno. Ker je izraz za izračun nekoliko daljši, naredimo za količnik posebno spremenljivko. V izraz dodamo še zaokrožitev na 3 decimalke.

```
double kolicnik = Math.Round((double)a / b, 3);
```

(double) a //pretvorba spremenljivke a v decimalno število

Brisanje ekrana

Za brisanje ekrana uporabimo metodo *Clear()*. Pred klicem te metode bi bilo smiselno zaustaviti izvajanje programa, da lahko s konzolnega okna preberemo podatke. To storimo z eno od metod za vnos podatkov (*Read(), ReadLine(), ReadKey()*).

```
Console.Write("\nPritisni enter za brisanje!");
Console.Read();    //zaustavitev izvajanja programa
Console.Clear();
```

S pritiskom na katerokoli tipko na tipkovnici (tudi <ENTER>) se izvajanje programa nadaljuje.

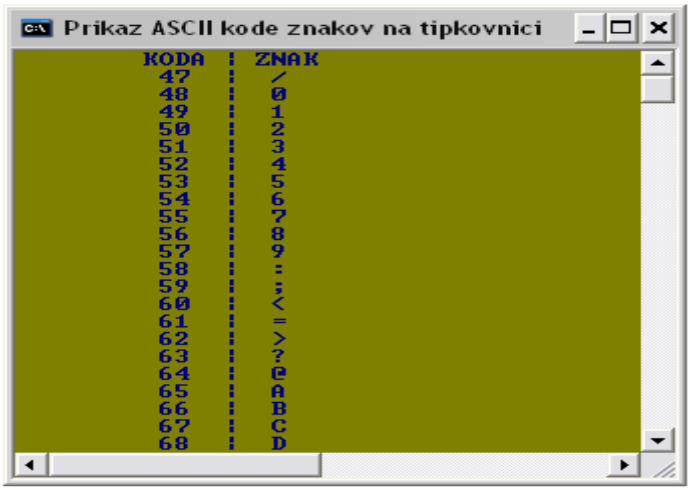
"\n" //ukaz za skok v novo vrstico

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.

Napiši program, ki prebere nek znak, nato pa na sredini pogovornega okna ta znak izpiše, v naslednji vrstici pod njim pa izpiše še njegovo ASCII kodo.

ASCII koda

Vsak znak, ki ga najdemo na tipkovnici, ima svojo številsko vrednost (kodo). Te kode so standardizirane po ASCII standardu, potrebne pa so zato, ker računalnik lahko operira le s številkami.



Slika 5: prikaz ASCII tabele v konzolnem oknu

Vnos znaka s tipkovnice

Znake v jeziku C# predstavlja podatkovni tip *char*. Ustvarimo ustrezno spremenljivko in prebrani podatek z metodo *ToChar()* pretvorimo v ustrezen podatkovni tip.

```
char znak = Convert.ToChar(Console.ReadLine());
```

Nastavitev položaja izpisa

To storimo z metodo *SetCursorPosition()*. Kot argumenta v oklepaju nastopata koordinati, na katerih želimo začeti izpis. Upoštevati moramo, da ima pogovorno konzolno okno 80 stolpcev in 25 vrstic in na osnovi tega določiti sredino okna. Metodo *SetCursorPosition()* vedno uporabimo neposredno pred metodo za izpis!

```
Console.SetCursorPosition(40, 12); //postavimo se v
                                   //vrstico 12
Console.Write(znak);               //izpis znaka
Console.SetCursorPosition(40, 13); //postavimo se v
                                   //vrstico 13
```

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.Write((int)znak);          //izpis ASCII kode
```

Do ASCII kode pridemo s pretvorbo znakovne spremenljivke v celo število.

7. Katerim opravilom so namenjene metode razreda *Console*?
8. Na kaj moramo biti pozorni pri uporabi metode *Clear()*?
9. Kakšna je razlika med postopkom uporabe lastnosti in postopkom uporabe metode?
10. Kakšna je standardna velikost konzolnega okna?
11. Katera dva načina za pomik kurzorja v novo vrstico poznaš?
12. Zakaj je potrebno pri računanju količnika podatke pretvoriti v decimalna števila in kako smo izvedli pretvorbo?

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.

UPORABA MATEMATIČNIH FUNKCIJ V IZRAZIH - vsebina razreda Math

Če želimo, da računalnik za nas opravi nek določen izračun, na primer izračun ploščine kroga, mu moramo podati navodilo, kako se ploščina izračuna. Navodilo podamo v obliki matematičnega izraza, pri pisanju teh pa moramo pogosto uporabljati različne matematične operacije, kot so potenca, korenjenje, kotne funkcije … Ker klasičnih matematičnih simbolov za te operacije ni na tipkovnici, se moramo naučiti, na kakšen način jih lahko podajamo računalniku in kako v programskem jeziku pišemo matematične izraze.

Pri tej vaji bomo spoznali nekatere metode, ki so shranjene v razredu ***Math***. Pri uporabi teh metod se bomo naučili

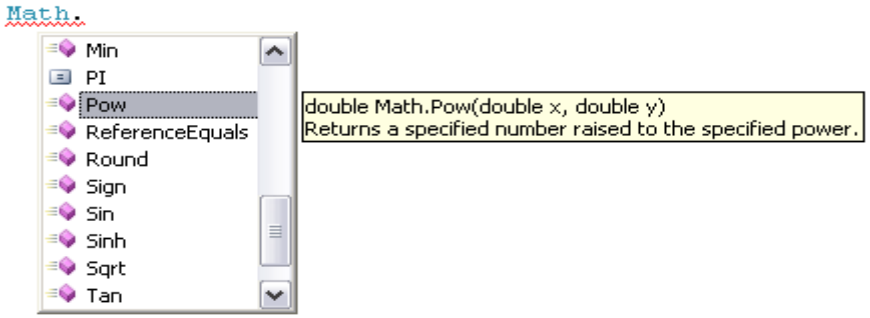
- poklicati metodo,
- podati argumente, ki jih metoda potrebuje za svoje delo (izračun),
- nekaj o (izračunani) vrednosti, ki jo metoda vrne,
- uporabljati neformatirano obliko izpisa.

Ogledali si bomo, kako so metode v razredu deklarirane.

Napiši program za izračun ploščine kroga.
Uporabnik programa naj vnese velikost radija, program pa izračuna in izpiše izračunano vrednost, ki naj ima v izpisu 3 decimalke.
Za kvadriranje uporabimo metodo za izračun potence *Pow()*, ki jo najdemo v razredu *Math*. Vrednost konstante π je prav tako shranjena v razredu *Math* pod imenom *PI*.

Deklaracija metode *Pow()*

Do metode pridemo tako, da najprej navedemo ime razreda in na koncu dodamo operator pika (v našem primeru *Math.*). Odpre se spustni seznam, ki prikazuje vsebino razreda.

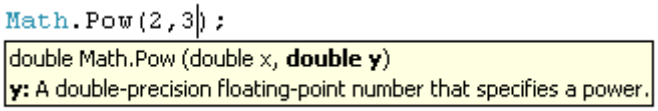


Slika 6: spustni seznam z vsebino razreda Math

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 njem poiščemo ustrezno metodo, jo označimo in s pritiskom na <ENTER> ali pa z dvoklikom prenesemo v tekstovni urejevalnik.

Deklaracijo metode *Pow()* si lahko ogledamo, če se z miško postavimo na ime metode in izgleda takole:



Slika 7: prikaz deklaracije metode Pow()

- Prva beseda je tip metode in pove, kakšno vrednost metoda vrača. *Double* pomeni, da bo izračunana vrednost decimalno število.
- Druga beseda je ime metode (pred imenom vedno stoji ime razreda, v katerem je metoda definirana, v razred vstopimo z operatorjem pika). Ko v kodi navedemo ime metode, jo s tem pokličemo.
- V oklepaju so navedeni argumenti (podatki), ki jih metoda potrebuje za svoje delo (x predstavlja osnovo, y pa potenco). Metoda kot argumente pričakuje decimalna števila (možna so tudi cela števila, ker podatkovni tip *double* to omogoča).

Klic metode

Ko metodo pokličemo, ta na klicno mesto vrne izračunano vrednost, ki jo lahko:

1. izpišemo ali
2. shranimo v pomnilnik .

V prvem primeru metoda *Pow()* predstavlja argument metode za izpis:

```
Console.WriteLine(Math.PI * Math.Pow( radij, 2));
```

V drugem primeru moramo ustvariti novo spremenljivko, ki prevzame izračunano vrednost

```
double ploscina = Math.PI * Math.Pow( radij, 2 );
```

Zaokroževanje

Ker podatkovni tip *double*, ki smo ga uporabili, podpira 15 decimalnih mest, jih je toliko tudi v izpisu. Število decimalk lahko zmanjšamo z zaokrožitvijo, ki jo naredimo z metodo *Round()*.

```
ploscina = Math.Round(ploscina, 3);
```

Metodi povemo, katero spremenljivko zaokrožujemo in na koliko decimalk (v našem primeru 3).

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.

Napiši program za izračun hipotenuze pravokotnega trikotnika (Pitagorov izrek). Uporabnik vnese velikosti obeh katet preko tipkovnice, program pa izpiše rezultat v obliki: *Pri vnesenih vrednostih katet 3 in 4 je velikost hipotenuze 5.*

Izračun kvadratnega korena

Za izračun kvadratnega korena uporabi metodo *Sqrt()*, ki jo najdeš v razredu *Math*. Vrednost, ki jo korenimo, predstavlja argument metode.

```
Console.WriteLine(Math.Sqrt(3)); //izpiše koren števila 3
```

Oblikovanje izpisa

Metoda *WriteLine()* kot argument zahteva nek tekst. Tega lahko sestavimo iz več delov. Za sestavljanje uporabimo operator +. Če kateri od delov ni v tekstovni obliki (je npr. število ali znak), se pri sestavljanju avtomatično pretvori v tekst.

```
Console.Write("Pri vnesenih vrednostih katet " + a + " in
" + b + " je  velikost hipotenuze " + c);
```

Zgornji tekst je sestavljen iz šestih delov.

POZOR!
V programski kodi vrstice ne smemo deliti v več vrstic. Dolžina vrstice v kodi ni omejena.

Napiši program za izračun katete pravokotnega trikotnika z uporabo kotnih funkcij. Uporabnik vnese velikost hipotenuze in kota ß , kateto pa izračuna z izrazom *a = c*cos(ß)* . Pri vnosu podatkov moramo biti pozorni na enoto, v kateri vnašamo velikost kota ß (oglej si deklaracijo metod *sin()* in *cos()* !). Program sestavi tako, da boš velikost kota lahko vnesel v kotnih stopinjah namesto v radianih

Vnos kota v radianih

Pri izračun katete je treba upoštevati, da je 180 stopinj enako 2π radianov, zato vneseno vrednost kota delimo s 180 in množimo z 2π.

```
double c = Convert.ToDouble(Console.ReadLine());
double beta = Convert.ToDouble(Console.ReadLine());
beta = beta * Math.PI / 180;       //pretvorba iz
                                   //stopinj v radiane
double a = c * Math.Cos(beta);     //izračun katete
```

V izrazu za pretvorbo stopinj v radiane smo dobili novo vrednost spremenljivke beta in ta se v naslednjem stavku uporabi v izračunu katete.

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.

13. Katere podatkovne tipe poznaš in katere vrste podatkov predstavljajo?
14. Kaj predstavlja tip metode in kaj so argumenti metode?
15. Koliko argumentov ima lahko metoda?
16. Kateri operator uporabimo, če želimo sestaviti besedilo iz več delov?
17. Pri uporabi kotnih funkcij smo morali podatke vnesti v radianih. Kakšen je bil postopek za pretvorbo iz stopinj v radiane?

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.

NAKLJUČNE VREDNOSTI – uporaba metod razreda Random

Pogosto naletimo na primere, kjer potrebujemo veliko število številskih podatkov. Da bi vse te podatke vnesli v računalnik preko tipkovnice, bi nam vzelo precej časa. Če vrednosti teh podatkov niso pomembne, lahko uporabimo naključna števila, pri tem pa nam lahko pomaga računalnik, ker ima orodje za ustvarjanje naključnih podatkov. Oglejmo si, kako to orodje deluje in kako ga uporabljamo.

Pri tej vaji bomo spoznali razred ***Random***, ki omogoča, da program naključno ustvari neko vrednost. V tem razredu imamo na voljo dve metodi, eno za ustvarjanje celih, drugo pa za ustvarjanje decimalnih števil. Ob vsakem klicu ene od metod nam ta vrne neko naključno izbrano pozitivno število, nabor vrednosti, znotraj katerega metoda izbira, pa lahko določimo sami. Ogledali si bomo:

- podajanje argumentov, ki določajo nabor vrednosti za izbiro naključnega števila (zgornja in spodnja meja)
- naključno generiranje decimalnih in negativnih števil
- naključno generiranje znakov
- oblikovanje izpisa s pomočjo tabulatorja

Napiši program, ki v pomnilnik vnese štiri naključna števila velikosti med 0 in 100. Nato ta števila izpiši v isti vrstici s presledkom velikosti enega tabulatorja.

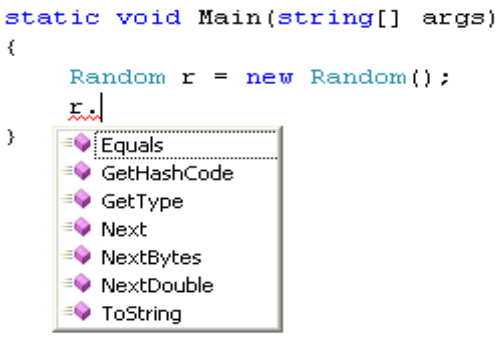
Deklaracija objekta razreda *Random*

Ker sta obe metodi, ki jih bomo potrebovali za delo, dinamični, moramo ustvariti objekt razreda, ker do dinamične metode lahko dostopimo le preko objekta. Objekt ustvarimo z naslednjim stavkom, ki ga zapišemo na začetku programske kode:

```
Random r = new Random();
```

Črka *r* predstavlja ime objekta (ime lahko izberemo sami), če imenu dodamo še operator pika (*r.*), se nam odpre seznam z vsebino razreda *Random*.

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.



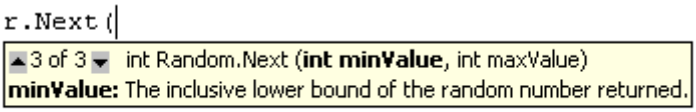
Slika 8: vsebina razreda Random

Naključno celo število

Če želimo ustvariti naključno celo število, iz seznama izberemo metodo *Next()*. To metodo lahko uporabimo na 3 načine:

1. brez argumenta – vrne celo število med vrednostma 0 in cca. 2000000000,
2. z enim argumentom – vrne celo število med vrednostma 0 in navedenim številom (to število ni vključeno),
3. z dvema argumentoma – vrne celo število med navedenima vrednostma (prvo število je vključeno, drugo pa ne).

Pri ogledu deklaracije metode vidimo, da morata biti argumenta metode celi števili (*int*):



Slika 9: deklaracija metode Next()

Vnos v pomnilnik

Za vnos štirih števil v pomnilnik moramo ustvariti štiri spremenljivke. Ob vsakem klicu metode ena od spremenljivk prevzame vrnjeno naključno vrednost:

```
int s1 = r.Next(101); //ustvarimo vrednost med 0 in 100
```

Izpis tabulatorja

Med vsakim številom moramo izpisati še tabulator, kar pomeni, da bo za vsako vrednost v izpisu namenjeno 8 mest (če je vrednost trimestno število, bodo cifre zasedle zadnja tri mesta, ostalih pet mest pa ostane praznih). Tabulator izpišemo tako, da v metodi za izpis uporabimo

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.

posebno konstanto, ki je rezervirana za izpis tabulatorja. Ker je to znakovna konstanta, jo moramo izpisati v dvojnih narekovajih!

newline	\n	nova vrstica
horizontal tab	\t	horizontalni tabulator
vertikal tab	\v	vertikalni tabulator
backspace	\b	briši prejšnji znak
carriage return	\r	na začetek vrstice
alert	\a	zvonček
backslash	\\	znak \
question mark	\?	vprašaj
single quote	\'	enojni narekovaj
double quote	\"	dvojni narekovaj

Slika 10: preglednica nekaterih znakovnih konstant

Primer izpisa spremenljivk z imenoma a in b, ki sta razmaknjeni za tabulator:

```
Console.WriteLine(a + "\t" + b);
```

Napiši program, ki ustvari in izpiše štiri naključna števila in naključno izbrano veliko tiskano črko. Prvo število naj bo dvomestno, drugo število naj bo negativno število do velikosti 200, tretje število naj bo decimalno število do velikosti 10 s tremi decimalkami, četrto število pa naj bo decimalno število v naboru med vrednostma -25 in +25. Vsa števila shrani v pomnilnik, njihove vrednosti pa prikaži v konzolnem oknu.

Dvomestno število

Dvomestna števila so števila med številoma 10 in 99. Potrebno je le ustrezno postaviti argumente metode *Next()*.

Negativno število

Negativno število dobimo, če vrnjeno vrednost pomnožimo s faktorjem -1 ali pa postavimo meje med -200 in 0.

Decimalno število

Decimalno število pa ustvarimo s pomočjo metode *NextDouble()*. Ta metoda ne dovoli vnosa argumenta kar pomeni, da ne moremo določiti zgornje in spodnje meje nabora naključnih vrednosti. Spodnja mejna vrednost 0 in zgornja mejna vrednost 1 sta že vnaprej določeni.

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.

Števila, ki so večja od 1, dobimo tako, da vrednost, ki jo ustvari metoda *NextDouble()*, pomnožimo z ustreznim faktorjem. V našem primeru je ta faktor število 10.

```
double d = r.NextDouble() * 10;
```

Pri decimalnih številih moramo uporabiti spremenljivko tipa *double*, ker tudi metoda pripada istemu tipu, ustvarjena vrednost pa ima 15 decimalk. Če želimo imeti le 3 decimalke, to dosežemo z zaokrožitvijo s pomočjo metode *Round()*, ki jo najdemo v nam že znanem razredu *Math*.

```
d = Math.Round(d, 3);
```

Decimalno število med -25 in +25

Ker pri metodi *NextDouble()* ne moremo določiti meja, si pomagamo s premikom po številski osi. Ustvarimo števila med 0 in 50 in jih nato na številski osi za 25 mest premaknemo v levo (odštejemo vrednost 25).

Naključno izbrana črka

Metoda, ki bi neposredno ustvarila črko, ne obstaja. Črka predstavlja znak (*char*), ti pa so kodirani s pomočjo ASCII tabele (vsak znak ima svojo desetiško kodo). Kode za velike tiskane črke angleške abecede se začnejo pri vrednosti 65 (za veliki tiskani A), črk pa je 26. Črko bomo ustvarili tako, da bomo s pomočjo metode *Next()* ustvarili naključno celo število, ki bo predstavljalo ASCII kodo neke črke (meje so med 65 in 91) in jo nato pretvorili v znak. Pretvorbo lahko naredimo s pomočjo ustrezne metode v razredu *Convert*

Convert.ToChar(r.Next())

ali pa z neposredno pretvorbo (cast)

(char) r.Next() // v oklepaju pred vrednostjo navedemo želeni //podatkovni tip

---

18. Kaj pri programiranju predstavljajo naključne vrednosti?
19. Katera metoda se uporablja za ustvarjanje celih naključnih števil in katera za ustvarjanje decimalnih naključnih števil? Pri kateri lahko nastavimo spodnjo in zgornjo mejo?
20. Napiši deklaracijo objekta razreda *Random*! Zakaj ta objekt potrebujemo?
21. Na kakšen način lahko ustvarimo naključno črko?
22. Kje se v ASCII tabeli nahajajo velike in kje male črke?
23. Navedi vsaj dva načina za pretvorbo iz številskega v znakovni podatkovni tip.

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.

POGOJNI STAVEK – uporaba stavka if

Računalnika pa ne bomo naučili le računanja. Naučili ga bomo, da v določenih primerih sprejema tudi odločitve in sicer na osnovi podatkov, ki jih dobi. Če na primer v računalnik vnesete neko število, lahko računalnik preveri, če je to število pozitivno ali negativno in se na podlagi te ugotovitve odloči, kaj bo storil. Na izbiro ima dve možnosti. Če je število pozitivno, bo na primer izračunal koren števila, če je negativno, pa bo na primer sporočil, da korenov negativnih števil ne računa. Vsebino obeh možnosti seveda določimo sami, prev tako pa tudi pogoj, na osnovi katerega se računalnik odloča.

Pri tej vaji si bomo na podlagi primerov ogledali uporabo stavka **if** in **if- else**. Spoznali bomo:

- nekatere operatorje, ki jih uporabljamo pri zapisu pogoja
- obliko pogojnega stavka
- način za zapis več pogojev hkrati

Napiši program, ki prebere število med 0 in 999 in ugotovi, ali je to število enomestno, dvomestno ali trimestno, ter ugotovitev izpiše. V primeru, da je število preveliko, naj program izpiše, da število ne ustreza zahtevam. (Pri vaji se bomo omejili na pozitivna števila.)

Oblika pogojnega stavka

Pogojni stavek oziroma ***if***-stavek ima v jeziku C# naslednjo obliko:

if (pogoj) { stavki } *else* { stavki }

Če je pogoj izpolnjen, se izvršijo stavki, ki sledijo besedi *if*, sicer pa se izvršijo stavki, ki sledijo besedi *else*. Če v delu za besedo *else* ni nobenega stavka, lahko ta del izpustimo. Tako dobimo naslednjo obliko:

if (pogoj) { stavki }

Stavki, ki se izvršijo pod pogojem, so označeni z zavitima oklepajema. Če je stavek samo eden, oklepaji niso potrebni.

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.

Zapis pogoja

Pogoj moramo zapisati s pomočjo operatorjev, ki jih imamo na voljo. Enomestna števila so števila, ki so manjša od deset, kar v sintaksi jezika C# zapišemo v obliki:

stev < 10 ali *stev <= 9*

Pri tem *stev* predstavlja ime spremenljivke, ki smo jo ustvarili, da smo lahko vnesli število. Če prebrano število ustreza temu pogoju, sledi izpis znakovnega niza (teksta):

```
if (stev < 10)
{
     Console.WriteLine("Število je enomestno.");
}
```

Zapis dveh pogojev hkrati

Dvomestna so vsa števila med 10 in 99, kar bi v matematiki zapisali *9 < stev < 100*. V tem zapisu hkrati nastopata dva pogoja in da trditev drži, morata biti izpolnjena oba hkrati. V jeziku C# moramo oba pogoja zapisati ločeno, združimo pa jih s pomočjo operatorja **&&** (logični IN, ker morata veljati oba pogoja hkrati). Zapis pogoja zgleda takole:

stev > 9 **&&** *stev < 100* *ali pa* *stev >=10* **&&** *stev <= 99*

Oba zapisa predstavljata isti pogoj.

Napiši program, ki prebere tri stranice trikotnika (a in b sta kateti, c pa hipotenuza) in ugotovi, če podatki ustrezajo pogojem trikotnika.
Če so podatki ustrezni, program najprej to ugotovitev izpiše in nato preveri še, če je trikotnik enakokrak ali pravokoten, sicer pa izpiše, da podatki ne ustrezajo.

Preverjanje veljavnosti trikotnika

Nek lik je lahko trikotnik, če velja, da je dolžina stranice manjša od vsote dolžin ostalih dveh. Pogoj mora hkrati veljati za vse tri stranice!

Uporaba stavka if-else

Pri preverjanju veljavnosti trikotnika uporabimo stavek *if-else*, ki ima dva dela. Za del, ki sledi besdi *if* , velja, da je pogoj izpolnjen, za del, ki sledi desedi *else*, pa velja, da pogoj ni izpolnjen. Besedi *if* zato sledi blok stavkov, v katerem so:

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.

- stavek za izpis, da podatki ustrezajo,
- preverjanje enakokrakosti in izpis ugotovitve,
- preverjanje pravokotnosti in izpis ugotovitve.

Besedi *else* sledi stavek za izpis teksta, da podatki ne ustrezajo.

Začetek in konec stavčnega bloka označimo z zavitima oklepajema {}. Če stavčni blok tvori en sam stavek, oklepaja nista potrebna.

Preverjanje enakokrakosti

Kot pogoj zapišemo trditev, da sta po dve kateti enaki. Ker imamo 3 stranice, imamo 3 možnosti

$a = b$ *ali* $a = c$ *ali* $b = c$

Za enakokrakost zadostuje že eden od pogojev, zato je med pogoji logični ALI, kar v jeziku C# zapišemo

```
a == b || a == c || b == c     //pogoj za enakokrakost
```

Preverjanje pravokotnosti

Kot pogoj zapišemo trditev, da velja Pitagorov izrek. V obeh primerih uporabimo stavek *if-else*, da lahko izpišemo, če trditev velja ali ne velja:

```
if (c == Math.Sqrt(Math.Pow(a, 2) + Math.Pow(b, 2)))
    Console.WriteLine(" je pravokoten ");
else
    Console.WriteLine(" ni pravokoten ");
```

POZOR!

- Beseda *else* ob sebi nima pogoja, ker se nanaša na pogoj ob besedi *if*.
- Pogojni stavek predstavlja neko zaključeno celoto, zato za pogojem (v prvi vrstici) ni podpičja, ker bi s tem pogojni stavek na tem mestu zaključili. Podpičja so na koncu posameznih stavkov v stavčnem bloku.

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.

Napiši program za izračun ploščine kvadrata, pravokotnika, kroga in trikotnika. Program naj ima na začetku možnost izbire, za kateri lik bo opravil izračun.

Izbira lika

Izbiro naredimo s pomočjo menija, v katerem bo nastopila spremenljivka znakovnega tipa (vrednost znakovnih spremenljivk vedno navajamo v enojnih narekovajih):

```
char izbira = 'A';//začetna vrednost spremenljivke je 'A'
Console.WriteLine("Izberi: A-kvadrat B-pravokotnik
C-trikotnik D-krog");
izbira = Convert.ToChar(Console.ReadLine());
```

V zadnji vrstici smo prebrali vrednost izbire, sedaj pa bomo za vsako od možnosti napisali pogojni stavek, ki bo preverili, kakšna je vnesena vrednost.

```
if (izbira == 'A')
{
     //Na tem mestu napišemo kodo za vnos podatkov in
     //izračun ploščine kvadrata.
}
if (izbira == 'B')
{
     //Na tem mestu napišemo kodo za vnos podatkov in
     //izračun ploščine pravokotnika.
}
```

Enako storimo še za možnosti 'C' in 'D'.

Napačna izbira

Če želimo, da program ugotovi napačen vnos podatka za izbiro lika, moramo uporabiti stavke *if-else*. Če (*if*) je vrednost izbire enaka črki A, sledi izračun ploščine kvadrata, sicer (*else*) preverimo, če je vrednost izbire črka B …

```
if (izbira == 'A')
{
     //kvadrat
}
else if (izbira == 'B')
{
     //pravokotnik
}
else           //nič od zgornjega
Console.WriteLine("Napačna izbira");
```

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.

Mala in velika črka

Program je sedaj narejen tako, da je izbira pravilna le ob vnosu velike tiskane črke, kar je za uporabnika neprijetno. Če želimo, da bo izbira delovala tudi za male črke, problem rešimo tako, da pri preverjanju pogoja vneseni znak vedno pretvorimo v veliko črko. To nam omogoča metoda *ToUpper()* iz razreda *Char*. Preverjanje pogoja v pogojnem stavku bi bilo v tem primeru

```
Char.ToUpper(izbira)=='A'
```

Metoda *ToUpper()* spremeni male črke v velike, ostale znake pa pusti nespremenjene. Ravno nasprotno od metode *ToUpper()* deluje metoda *ToLower()*.

24. Kdaj uporabimo pogojni stavek *if* in kdaj njegovo razširjeno obliko *if-else*?
25. Katere operatorje uporabljamo v pogoju za primerjavo dveh vrednosti in kakšno primerjavo posamezni operator predstavlja?
26. Ali na koncu vrstice, ki se začne z besedo *if*, pišemo podpičje? Zakaj?
27. Kako označimo začetek in konec pogojnega stavka in v katerem primeru to ni potrebno?
28. Ali stavek *else* lahko nastopa brez stavka *if*?
29. Kako napišemo pogoj, ki bi hkrati veljal za izbrano veliko in malo črko?
30. Ali lahko v istem pogojnem stavku uporabimo več pogojev hkrati? Kateri dve možnosti imamo na izbiro?

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.

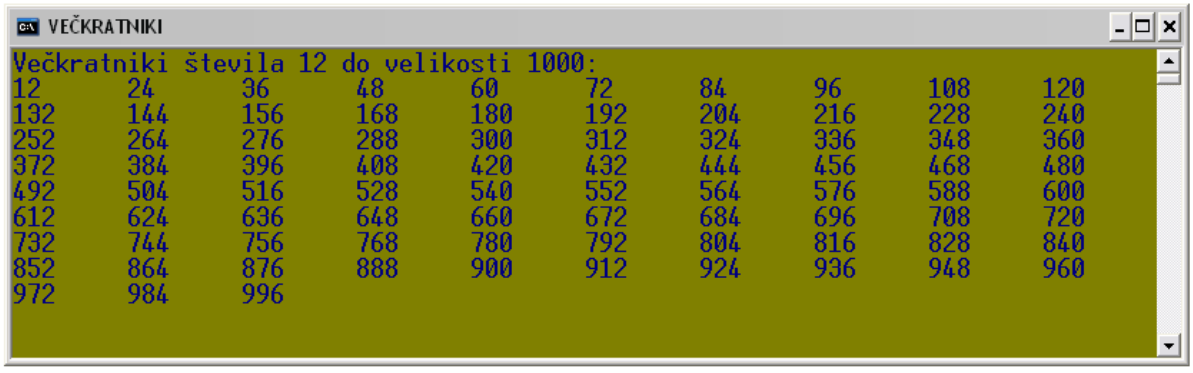
UPORABA ZANK
– zanki while in do-while

Pogosto naletimo na primere, ko se nek izračun ali postopek zelo velikokrat ponovi. Vzemimo na primer, da želimo poiskati in izpisati 500 večkratnikov nekega števila. Pri tem se postopek izračuna večkratnika in izpisa ponovi petstokrat in če bi to tolikokrat zapisali v program, bi bilo pisanje zamudno, program pa po nepotrebnem predolg. Situacijo rešimo s pomočjo zanke, kjer postopek napišemo samo enkrat, potem pa ga ponavljamo tako dolgo, kot je to potrebno, v našem primeru na primer petstokrat. Tudi v tem primeru se mora računalnik, podobno kot pri pogojnem stavku, na osnovi nekega pogoja odločiti, kdaj bo ponavljanje končal.

Na osnovi primerov bomo spoznali obliko zank *while* in *do-while* ter načine in možnosti za njuno uporabo. Ogledali si bomo

- razlike med zankama
- zapis pogoja za izhod iz zanke
- nekatere operatorje, ki jih uporabljamo pri zapisu pogoja
- napake pri zapisu pogoja
- oblikovanje tabele pri izpisu podatkov

Napiši program, ki izpiše vse večkratnike izbranega števila do velikosti 1000. Število uporabnik programa vnese preko tipkovnice. Izpis večkratnikov oblikuj s pomočjo tabulatorja.



Slika 11: prikaz izpisa večkratnikov

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.

Zanki *while* in *do-while*:

Osnovni zanki v jeziku C# sta zanki ***while*** in ***do-while***. Zanka pomeni, da se del programa večkrat ponovi, če je za to izpolnjen določen pogoj.

Zanka ***while***:

while (pogoj) { stavki }

Stavki se izvajajo toliko časa, dokler je pogoj izpolnjen.

Zanka ***do-while***:

do { stavki } *while* (pogoj);

Tudi tu se stavki izvajajo toliko časa, dokler je pogoj izpolnjen. Razlika med zankama je v tem, da se pri *do-while* stavki izvedejo vsaj enkrat, pri *while* pa to ni nujno, ker je pogoj preverjen na začetku zanke.
Če predstavlja vsebino zanke samo en stavek, zaviti oklepaji niso potrebni.
Pri obeh zankah moramo paziti, da pride med izvajanjem enkrat do **neizpolnitve** pogoja, sicer smo ustvarili neskončno zanko, s tem pa tudi program, ki se ne konča.

Večkratniki

Dobimo jih lahko na dva načina:

1. s prištevanjem prebranega števila k prejšnjemu večkratniku
2. ustvarimo števec, ki se povečuje za 1 in ga množimo s prebranim številom

Oba navedena postopka je potrebno ponavljati v zanki. Uporabimo zanko *while*.

Prvi način:

```
int veck = stevilo;           //pred vstopom v zanko
          //postavimo vrednost prvega večkratnika
veck = veck + stevilo;        //ta stavek ponavljamo,
                              //zato bo v zanki
```

Spremenljivka *stevilo* predstavlja podatek, ki ga vnese uporabnik. Spremenljivka *veck* pa bo predstavljala naš večkratnik. Njena začetna vrednost naj bo enaka vnesenemu številu. V zanki se bo vrednost te spremenljivke na novo izračunala, vsakič, ko se zanka izvede, bomo prejšnjemu večkratniku prišteli vrednost spremenljivke *stevilo*. Ko večkratnik izračunamo, ga še izpišemo (v zanki).

Drugi način:

```
//pred zanko:
```

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 veck = 0;       //ustvarimo spremenljivko veck
int i = 1;          //števec z začetno vrednostjo 1
//v zanki:
veck = i * stevilo;      //izračun večkratnika
i = i + 1;               //povečamo števec za 1
```

Spremenljivko *veck* deklariramo pred zanko ker jo bomo potrebovali pri preverjanju pogoja. Njena nova vrednost se v tem primeru ne bo izračunavala na osnovi njene stare vrednosti kot pri prvem načinu. Ko večkratnik izračunamo, ga izpišemo in števec povečamo za 1.

Pogoj:

Zanka se ponavlja, dokler je za to izpolnjen pogoj. Ta je odvisen od velikosti večkratnika. Pogoj naj bo izpolnjen, dokler je večkratnik manjši od 1000. V jeziku C# to zapišemo

veck <= 1000

POZOR!
Zapis pogoja *veck == 1000* ni primeren, ker število 1000 ni nujno med večkratniki izbranega števila. V tem primeru bi ustvarili neskončno zanko, ker obstaja možnost, da pogoj nikoli ne bo izpolnjen!

Napiši program, ki omogoča vnos več znakov s tipkovnice. Vnos se zaključi, če je vneseni znak črka Q (velja za malo in veliko črko Q). Ko se vnos konča, naj program izpiše, koliko znakov je bilo vnesenih.

Izbira zanke

Ker moramo prebrati več znakov, izvedemo branje v zanki. Pogoj za končanje zanke pa je odvisen od prebranega znaka, zato se mora zanka izvršiti vsaj enkrat, preden testiramo pogoj. V takšnih primerih je primernejša raba zanke *do-while*.

```
char c = 'a';
do
{
    Console.WriteLine("Vnesi znak: "); //navodilo uporabniku
    c = Convert.ToChar(Console.ReadLine());   //vnos znaka
}
while(//pogoj);
```

Zanko *do-while* končamo s podpičjem.

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.

Štetje znakov

Za štetje znakov ustvarimo dodatno spremenljivko (celo število), ki jo imenujemo števec. To moramo storiti pred zanko, njena začetna vrednost naj bo 0, ker še nismo vnesli nobenega znaka. V zanki takoj za vnosom znaka moramo vrednost števca povečati za 1. Ko se zanka konča, pa izpišemo vrednost števca, ker ta predstavlja število prebranih znakov.

```
int i = 0;       //deklaracija števca
i++;         //povečanje za 1 (krajši zapis za: i=i+1;)
Console.Write("Vnesli smo " + i + " znakov.");  //izpis
```

Velika ali mala črka (pogoj)

Problem lahko rešimo na dva načina:

1. Na mestu, kjer v zanki testiramo pogoj, lahko zapišemo hkrati dva pogoja, ki jih povežemo z operatorjem ***logični in***, kar pomeni, da se izvajanje zanke konča, če sta hkrati veljavna oba pogoja:

znak != *'q'* && *znak* != *'Q'*

Pri zgornjem pogoju se zanka ponavlja, če je vneseni znak hkrati različen ob male in velike črke Q.

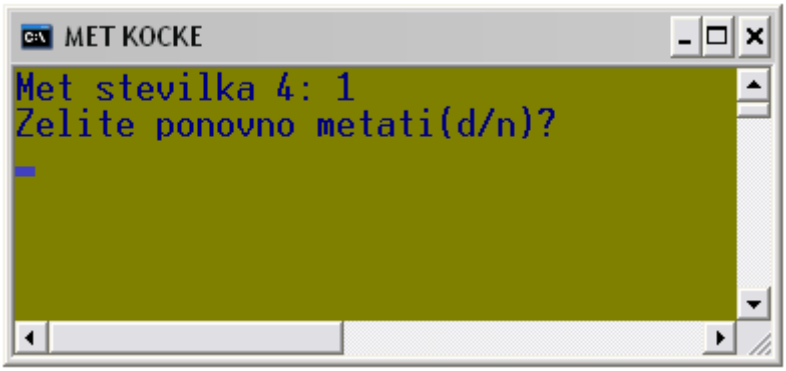
2. uporabimo metodo *ToUpper()*, ki pretvori malo črko v veliko. V tem primeru preverjamo pogoj le za veliki Q:

Char.ToUpper(znak)!= *'Q'*

V programu preizkusi oba načina!

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.

Napiši program za simulacijo meta kocke. Po vsakem metu vas program vpraša, če želite ponoven met, ter nato pobriše zaslon. V izpisu naj bo vedno tudi zaporedna številka meta. Na koncu program izpiše še povprečno vrednost vseh metov.



Slika 12: prikaz meta kocke v konzolnem oknu

Met kocke

Simuliramo ga z uporabo metode za ustvarjanje naključnih celih števil.

```
int met = 0;   //pred zanko pripravimo spremenljivko,
               //ki bo beležila vrednost meta
met = r.Next(1,7);  //v zanki pokličemo metodo Next(),
          //ki vrne naključno celo število,
          //spremenljivka r je objekt razreda Random
```

Spremenljivka za kontrolo izbire ponovnega meta

Za kontrolo izbire ponovnega meta potrebujemo spremenljivko znakovnega tipa, ki ji damo začetno vrednost (npr. 'd' , kar pomeni da). V primeru, da se vrednost te spremenljivke spremeni, se izvajanje zanke konča.

```
//pred zanko:
char ponovi = 'd';  //spremenljivko ustvarimo
//v zanki:
Console.WriteLine("Zelite ponovno metati(d/n)?");
ponovi = Console.ReadKey().KeyChar;
                    //vnos vrednosti 'd' ali 'n'
Console.Clear();    //brisanje vsebine konzolnega okna
```

Pri vnosu vrednosti spremenljivke *ponovi* smo uporabili metodo *ReadKey()*. Ta metoda s tipkovnice prebere samo 1 znak, pri tem pa ne čaka na tipko <ENTER>, vnos se zgodi že s

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.

pritiskom na izbran znak na tipkovnici. Pretvorbo vnesenega podatka v ustrezen podatkovni tip naredimo z lastnostjo *KeyChar*.
Brisanje ekrana opravi metoda *Clear()*. Priporočljivo je, da jo uporabimo v naslednji vrstici za vnosom podatkov. Takrat se namreč izvajanje programa ustavi in lahko preberemo vsebino konzolnega okna, preden jo pobrišemo.
Pogoj za vstop v zanko bo odvisen od spremenljivke *ponovi*. Če je njena vrednost 'd', se zanka ponovi, ob vseh drugih vrednostih pa zanko končamo.

ponovi == 'd' // pogoj

Povprečna vrednost

V zanki je potrebno sproti seštevati vrednosti metov, pred izpisom pa je dobljeno vsoto potrebno deliti s številom metov. Pri deljenju bodite pozorni na tipe spremenljivk! Če želimo dobiti izpis v decimalkah, bo potrebo uporabiti tip *double*.

```
double povp = 0;   //spremenljivka za izračun povprečja

povp = povp + met; //spremenljivko povp bomo začasno
         //uporabili za shranjevanje trenutne vsote vseh
         //metov, novo vrednost dobimo //tako, da stari
         //vsakič prištejemo vrednost zadnjega meta

povp = Math.Round(povp / i,3));  //po izstopu iz zanke
         //izračunamo povprečje metov tako, da vsoto
         //delimo s številom metov, //število metov beleži
         //spremenljivka z imenom i (števec), z metodo
         //Round povprečje zaokrožimo na 3 decimalke
Console.WriteLine("Povprecna vrednost " + i + " metov je " +
povp);
         //ko povprečje itračunamo, ga izpišemo
```

Napiši program, ki po vrsti izpiše vse velike črke angleške abecede. ASCII kodo črke 'A' ugotovi s pomočjo programa. Upoštevaj, da je črk 26.

Izpis ASCII kode črke a

```
Console.Write("ASCII koda črke A je " + (int)'A');
```

Izpis abecede

Napišemo zanko, ki se izvede 26-krat. Pred zanko ustvarimo števec, katerega vrednost bo na začetku 0. V telesu zanke števec vedno povečamo za 1, v pogoju pa preverjamo, če je vrednost števca manjša on 26.

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.

Za črko pa potrebujemo še dodatno spremenljivko (*crka*), katere začetna vrednost bo ASCII koda črke A.

```
int crka = 65; //deklaracija pred zanko
```

V zanki to spremenljivko izpišemo.

```
Console.Write((char)crka);    //izpis v zanki
```

Pred izpisom spremenljivko pretvorimo iz številčne vrednosti v znakovno, sicer bomo na ekranu videli kodo črke. V naslednji vrstici za izpisom pa vrednost spremenljivke povečamo za 1, kar nas premakne v abecedi na naslednjo črko. Ta se bo izpisala ob naslednji izvedbi zanke.

31. Kaj je zanka in kateri dve osnovni zanki pozna jezik C#? Kakšna je razlika med njima?
32. V katerih primerih programer uporabi zanko?
33. Kako označimo začetek in konec zanke in kje uporabimo podpičje?
34. Kaj nam omogoča metoda *ReadKey()*?
35. Kaj v zanki predstavlja števec in čemu služi?
36. Opiši postopek (algoritem) za izračun povprečne vrednosti dvajsetih 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.

ZANKA FOR
– posebna oblika zanke while

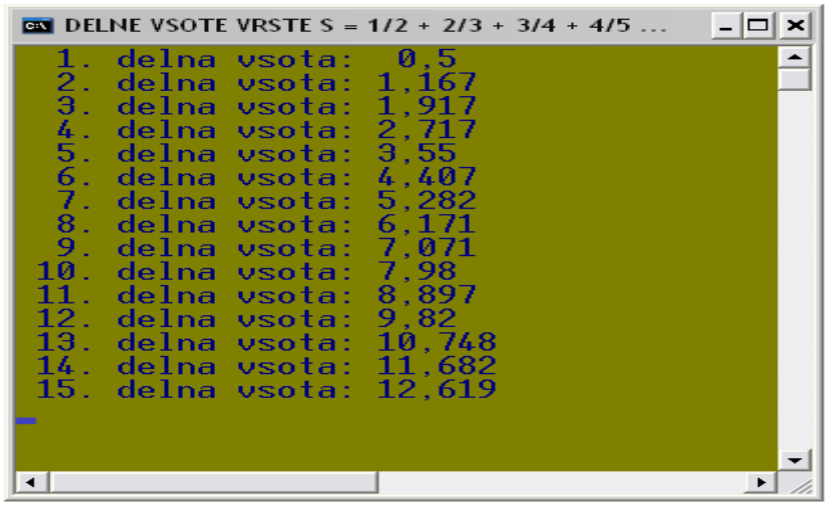
Kadar se nek postopek ponavlja, pogosto že vnaprej poznamo število ponovitev, zato štejemo, kolikokrat se je zanka ponovila in v pogoju preverjamo, če smo že dosegli želeno število. Za takšne primere lahko uporabimo posebno obliko zanke, ki nam omogoča, da že v začetku nastavimo začetno vrednost štetja, končno vrednost štetja in vrednost, za katero se bo števec spreminjal. Takšna oblika je preglednejša od običajne zanke.

Pri tej vaji si bomo na podlagi primerov ogledali uporabo zanke ***for***. Ob tem bomo spoznali:

- v katerih primerih uporabiti zanko *for*
- obliko zanke *for*
- uporabo logične spremenljivke tipa *bool*
- uporabo števca tipa *int*, *char* in *double* v glavi zanke *for*
- uporabo ukaza za prekinitev zanke
- inkrement in dekrement vrednosti spremenljivke
- prekoračitev največje vrednosti določenega podatkovnega tipa

---

Napiši program, ki izračuna in izpiše delne vsote vrste S = 1/2 + 2/3 + 3/4 + 4/5 ... za prvih 15 členov.



Slika 13: prikaz delnih vsot v konzolnem oknu

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.

Zanka *for*

Zanka *for* je posebna oblika zanke *while*, pri kateri vnaprej poznamo število ponovitev zanke. Zato moramo v glavi te zanke vedno navesti spremenljivko, ki ima vlogo števca števila ponovitev. Oblika zanke *for*:

```
for ( inicializacija števca ; pogoj;  korak števca )        //glava zanke
 { vsebina zanke }                                          //telo zanke
```

- Inicializacija števca – postavimo začetno vrednost števca. Števec lahko deklariramo na tem mestu ali pa pred zanko. Če ga deklariramo na tem mestu, bo veljaven le znotraj zanke, kar pomeni, da ga izven zanke ne moremo uporabljati. Števec lahko pripada tipom *int*, *char* ali *double*.
- Pogoj – s pogojem povemo, pri kateri vrednosti števca se zanka konča.
- Korak števca – vrednost števca se v zanki enakomerno spreminja (povečuje ali zmanjšuje). Korak števca pove, za koliko se ob vsaki izvedbi zanke števec spremeni.
- Vsebina zanke – del kode, ki se ponavlja.

Dva primera zanke:

```
for (int i = 0; i < 20; i++)        //zanka teče od vrednosti
                                    //števca 0 do 19
{
     //vsebina zanke
}


for (int i = 20; i >= 0; i--)       //zanka teče od vrednosti
                                    //števca 20 do 0
{
      //vsebina zanke
}
```

Oblike korakov

```
i = i + 2;       //povečanje števca i za 2
st = st - 4;     //zmanjšanje števca st za 4
```

Pogosto se vrednost koraka spreminja za 1. V tem primeru uporabimo krajši zapis:

```
i++;       //enako kot i = i + 1; ta zapis imenujemo
           //inkrement
```

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.

```
a--;        //enako kot a = a - 1; ta zapis imenujemo
            //dekrement
```

Delna vsota:

Delno vsoto n-členov dobimo tako, da seštejemo vrednosti vseh členov do člena z zaporedno številko n:

prva delna vsota je 1/2
druga delna vsota je 1/2 + 2/3
tretja delna vsota je 1/2 + 2/3 + 3/4 …

Za izračun vrednosti posameznih členov potrebujemo splošni člen vrste. V našem primeru je to *n /(n + 1)* . Če uporabimo zanko *for*, spremenljivka n lahko nastopi hkrati tudi kot števec v zanki.

```
for (int n = 1; n <= 15; n++)      //glava zanke
```

Če primerjamo prvi člen vrste z izrazom za splošni člen, ugotovimo, da je v prvem členu vrednost faktorja n enaka 1. Zato bo začetna vrednost spremenljivke n tudi enaka 1. V drugem členu je vrednost spremenljivke n enaka 2, v tretjem 3… Od tod lahko sklepamo, da se bo n povečeval za 1. Ker želimo delne vsote za prvih 15 členov, v pogoju zapišemo, da n ne sme preseči vrednosti 15.
V zanki nato izračunamo vrednost člena za trenutno vrednost spremenljivke n. Pri deljenju moramo biti pozorni na tip spremenljivke, če želimo dobiti decimalke!

```
double clen = (double)n / (n + 1);
```

Spremenljivko *clen* lahko deklariramo v zanki, ker je izven zanke ne potrebujemo.
Delne vsote dobimo tako, da k prejšnji delni vsoti prištevamo izračunani člen.

```
double vsota = 0;          //deklariramo pred zanko
vsota = vsota + clen;      //clen prištejemo prejšnji vsoti
                           //in dobimo novo delno vsoto
```

Zgornjemu stavku sledi še izpis delne vsote

```
Console.WriteLine("{0,3}. delna vsota: {1,4}",n, Math.Round(vsota, 3));
```

Tako izračun kot tudi izpis delne vsote morata biti v zanki.

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.

Napiši program, ki izpiše najprej vse velike, nato pa še vse majhne črke angleške abecede. Izpis oblikuj tako, da bo po 7 črk v vsaki vrstici.

Izpis črke

Ker si črke v ASCII tabeli sledijo po vrsti, jezik C# pa omogoča nekatere računske operacije s podatki tipa *char*, lahko uporabimo zanko *for* s števcem tipa *char*. Začetna vrednost števca je prva črka abecede:

```
for( char c = 'A'; c <= 'Z'; c++ )
```

Če števec pripada znakovnemu podatkovnemu tipu (*char*), pri koraku lahko uporabimo le inkrement in dekrement.

Po 7 v vrsto

Skok v novo vrsto izvedemo pod pogojem, da smo že izpisali 7 črk. Potrebujemo torej stavek *if*, pogoj pa bomo najlaže postavili, če bomo uporabili še dodatno spremenljivko, ki šteje črke. Ob izpisu vsake črke jo povečamo za 1in v primeru, ko je ta spremenljivka deljiva s 7, naj se zgodi skok v novo vrsto.

```
if(/* števec i je deljiv s 7 */)
{
     Console.WriteLine();    //skok v novo vrsto
}
```

Pogoj v stavku *if*

Število je deljivo z nekim drugim številom, kadar je ostanek po deljenju enak 0. Kadar med seboj delimo dve celi števili (*int*), dobimo njun ostanek s pomočjo operatorja % (npr. 8 % 3 je 2).
V našem primeru želimo, da je ostanek po deljenju števca i s številom 7 enak 0, kar zapišemo

```
i % 7 == 0
```

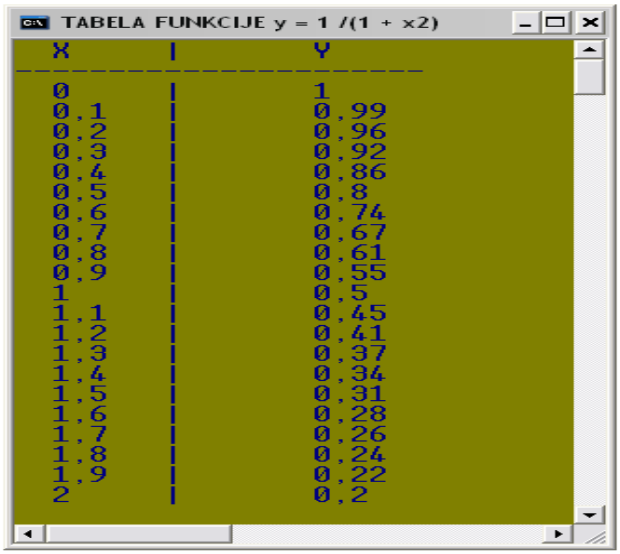
Če je ta pogoj izpolnjen, skočimo v novo vrsto.

Male črke

Za izpis malih črk napišemo še eno enako zanko *for*, ki se razlikuje le v začetni in končni vrednosti števca. Ti morata biti mali *'a'* in *'z'* ali pa uporabimo metodo *ToLower()*, ki spremeni velike črke v male.

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.

Napiši program za tabeliranje matematične funkcije $y = 1/(1+x^2)$ s korakom 0.1 do vrednosti x = 2.



Slika 14: tabeliranje funkcije

Korak

Korak, s katerim se povečuje spremenljivka x, je 0.1, zato bo ta spremenljivka (ki obenem nastopa tudi kot števec v zanki *for*) tipa *double*, korak pa zapišemo:

x = x + 0.1;

Glave tabele oblikujemo z znaki, ki so nam na voljo.

```
Console.Title = "TABELA FUNKCIJE y = 1 /(1 + x2)";
Console.WriteLine("  X\t|\tY\n----------------------");
```

Poravnavo v tabeli naredimo s pomočjo tabulatorja (`"\t"`).

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.

Napiši program, ki prebere število in ugotovi, če je vneseno število praštevilo.

Praštevilo

To je število, ki ni deljivo z nobenim številom, razen z 1 in samim seboj. Program bo po vrsti preveril vse možne delitelje prebranega števila. Najprej preverimo deljivost s številom 2, nato s številom 3 in tako naprej delitelj povečujemo za 1. Največji možni delitelj ne more biti večji od polovice vnesenega števila.

Delitelji

Možne delitelje bo predstavljala spremenljivka, katere začetna vrednost bo 2 in se bo enakomerno povečevala za 1, dokler ne bo dosegla vrednosti polovice vnesenega števila. Takšno spremenljivko lahko ustvarimo v glavi zanke *for*

```
for (int del = 2; del <= stevilo / 2; del++)
```

Spremenljivka *stevilo* predstavlja vneseno število. V zanki bomo za vsak delitelj preverili, če deli prebrano število.

Preverjanje deljivosti

Če pri vnesenem številu najdemo vsaj enega delitelja, si moramo to informacijo zapomniti, ker bo na osnovi te informacije program ugotavljal, če je neko število praštevilo. Zato potrebujemo neko spremenljivko, pri kateri pa zadostuje, da ima le dve vrednosti. Takšne so spremenljivke, ki pripadajo logičnemu podatkovnemu tipu *bool*.

```
bool je_prastevilo = true;
```

Spremenljivka tipa *bool* ima lahko le vrednost *true* ali *false*. V našem primeru jo uporabimo tako, da ima pred preverjanjem deljivosti (pred zanko) začetno vrednost *true*. Če v zanki najdemo vsaj enega delitelja, spremenimo vrednost spremenljivke *je_prastevilo* v *false*.

```
if (//število je deljivo)
{
     je_prastevilo = false;
     break;
}
```

Ko enkrat najdemo delitelja, je nadaljnje iskanje nesmiselno, ker predstavlja izgubo procesorskega časa. Zato zanko prekinemo s stavkom *break* (ker se pogojni stavek *if* nahaja v zanki *for*, bo stavek *break* prekinil to zanko).

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.

Pogoj za deljivost

Pogoj zapišemo na osnovi trditve »če je vneseno število (*stevilo*) deljivo s trenutnim deliteljem (*del*), je ostanek po deljenju enak 0. V jeziku C# to zapišemo

stevilo % del == 0

Izpis

Ko je iskanje delitelja zaključeno (po izhodu iz zanke), s pogojnim stavkom preverimo stanje spremenljivke *je_prastevilo* in izpišemo, če je število praštevilo.

```
if (je_prastevilo == true)
     Console.WriteLine(stevilo+" je praštevilo.");
else Console.WriteLine(stevilo+" ni praštevilo.");
```

V pogoju spremenljivko *je_prastevilo* primerjamo z vrednostjo *true*. Ta primerjeva je v jeziku C# pri vseh pogojih določena avtomatično, zato zapis pogoja lahko skrajšamo.

```
if (je_prastevilo)  //se primerja s true
```

Če želimo primerjavo z vrednostjo *false*, pa uporabimo negacijo, ki jo predstavlja operator *!* .

```
if (!je_prastevilo) //se primerja s false
```

Napiši program, ki prebere 10 števil in med njimi poišče najmanjše in največje število.

Iskanje najmanjšega števila

Potrebujemo spremenljivko, ki si bo zapomnila trenutno najmanjše število. Če bo naslednje prebrano število manjše, bo ta spremenljivka prevzela njegovo vrednost, sicer ostane njena vrednost nespremenjena. Začetna vrednost te spremenljivke mora biti zelo velika, pri nalogi se omejimo na števila tipa *short*.

Podatkovni tip short

Spremenljivka tipa *short* zasede v pomnilniku 16 bitov oz. 2 byta. Največje vrednosti, ki so pri takšnih spremenljivkah dovoljene, so od minus 2 na petnajsto potenco do plus 2 na petnajsto potenco, kar je približno med minus in plus 32000.
Ta vrednost naj zato predstavlja zgornjo mejo pri vnosu števil in hkrati začetno vrednost spremenljivke, ki si zapomni najmanjše število.

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.

```
short min = 32000;
```

Tudi pri vnosu števil s tipkovnice moramo biti pozorni, da uporabimo metodo za pretvorbo v številski tip *ToInt16()*, uporabnika pa v navodilih opozorimo na največjo dovoljeno vrednost.

Zanka za vnos iskanja najmanjšega števila

Za vneseno število bomo uporabili eno samo spremenljivko, zato bomo števila preverjali sproti. Vnos in preverjanje bosta zato v isti zanki.

```
for (int i = 0; i < 10; i++) //zanka, ki se ponovi 10-krat
{
     //vnos števila
     //preverjanj
}
```

Preverjanje

Če je prebrano število (*stevilo*) manjše od trenutno najmanjšega števila (*min*), naj *min* postane *stevilo*. Zgornji stavek bi v jeziku C# zapisali:

```
if (stevilo < min)
{
     min = stevilo;
}
```

Ko se zanka po vnosu 10 števil konča, bo v spremenljivki *min* ostalo shranjeno najmanjše število, ki ga na koncu izpišemo.

Iskanje največjega števila

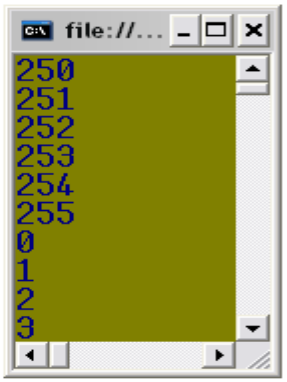
Pri iskanju največjega števila je postopek enak, le začetna vrednost spremenljivke, ki si zapomni največje število (*max*), mora biti 0, če delamo le s pozitivnimi števili, ali pa zelo majhna, če delamo tudi z negativnimi števili (pri tipu *short* bi bilo to približno -32000). Iskanje največjega števila izvedemo v isti zanki kot iskanje najmanjšega, ker ga bomo iskali v istem naboru desetih števil.

Prekoračitev največje vrednosti določenega podatkovnega tipa

Oglejmo si izpis programa

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.

```
class Program
{
    static void Main(string[] args)
    {
        byte x=250;
        for (int i = 0; i < 10; i++)
            Console.WriteLine(x++);
        Console.Read();
    }
}
```



Slika15: program in njegov izpis

Spremenljivka tipa *byte* zasede v pomnilniku 8 bitov, zato ima na voljo 2 na osmo potenco (256) različnih vrednosti. Ker je tip *byte* namenjen pozitivnim celim številom, bodo to števila med 0 in 255. Ko dosežemo največjo vrednost, se števila začnejo ponovno od začetka in če na to nismo pozorni, bo program deloval narobe.
Isti program preveri še za spremenljivko x, ki naj bo tipa *sbyte*, začetna vrednost pa naj bo 125. Komentiraj dobljeni izpis.

37. V katerih primerih je zanka *for* primernejša od zanke *while*?
38. Kaj moramo navesti v glavi zanke *for* in v katerem vrstnem redu?
39. Kaj predstavljata operaciji inkrement in dekrement?
40. Kaj je korak zanke?
41. Kakšen je postopek za oblikovanje izpisa po 7 elementov v vrstico?
42. S katerim namenom smo uporabili spremenljivko logičnega tipa *bool*?
43. Kakšen je postopek za iskanje najmanjšega števila in kaj v postopku spremenimo, če iščemo največje število?
44. Kaj je prekoračitev vrednosti pri določenem podatkovnem tipu?

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.

PODATKOVNI TIP STRING – delo s tekstom

Računalnik lahko uporabimo tudi kot napravo za pisanje in urejanje besedil, pri urejanju pa si pogosto pomagamo z različnimi programi za iskanje in zamenjavo določenih znakov, besed ali delov besedila. V takšnih primerih namesto številskih uporabljamo tekstovne podatke, ki jih lahko potegnemo v naš program iz različnih elektronskih virov. Računalnik lahko namesto nas opravi obsežno delo iskanja podatkov ali odkrivanja napak, ki jih zna tudi odpraviti.

Pri tej vaji si bomo uporabili spremenljivko, ki pripada podatkovnemu tipu *string* in je namenjena delu z znakovnimi nizi. Pri tem si bomo ogledali:

- nekatere metode za vnos znakov in načine za vnos niza znakov v pomnilnik
- nekatere metode za urejanje znakovnega niza
- dostop do posameznega znaka v nizu
- kombinacijo zanke in pogojnega stavka
- primerjavo dveh nizov
- dodajanje nove knjižnice
- uporaba sistemskega ukaza v programski kodi
- sestavljanje niza iz posameznih znakov ali manjših nizov.

Sestavi program, ki s tipkovnice prebere niz znakov (npr. besedo) in ga nato izpiše navpično (vsak znak v svojo vrstico).

Znakovni niz

Tekstovne podatke računalnik obravnava kot znakovni niz. Ustrezen podatkovni tip za takšne podatke je v jeziku C# tip *string*. Oglejmo si primer deklaracije tekstovne spremenljivke

```
string text = "Tole je nek niz znakov: #$%& 765.";
```

V niz sodijo vsi znaki, ki so kodirani v ASCII tabeli in jih najdemo na tipkovnici (tudi presledek). Začetek in konec znakovnega niza označimo z dvojnim narekovajem, ki pa ni del niza.

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.

Vnos niza s tipkovnice

Za vnos uporabimo metodo ReadLine(), ki podatke že vrne v obliki znakovnega niza, zato pretvorba ne bo potrebna.

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

Indeks

Izpisati moramo vsak znak posebej, ker je po vsakem izpisu potrebno skočiti v novo vrstico. Do posameznega znaka v nizu pridemo s pomočjo indeksa, ki ga pišemo v oglatih oklepajih ob imenu znakovnega niza. Indeks predstavlja zaporedno številko znaka v nizu in na ta način kaže na določen znak.

```
Console.Write(beseda);      //izpis celotnega niza
//če dodamo indeks:
Console.Write(beseda[3]);   //izpis četrtega znaka
```

Kadar ne uporabimo indeksa, se sklicujemo na celoten niz, sicer pa le na en znak. Prvi znak v nizu ima indeks 0, vsi ostali indeksi pa sledijo po vrsti.

Dolžina niza

Ker so besede lahko različno dolge, se dolžina vpisanega niza spreminja. Zaradi tega ne poznamo indeksa zadnjega znaka v nizu. Tega lahko program poišče sam s pomočjo lastnosti za ugotavljanje dolžine niza. Indeks zadnjega znaka je dolžina niza, zmanjšana za 1, dolžina niza pa je število znakov v nizu. Oglejmo si izpis zadnjega znaka iz niza *beseda*, ki smo ga ustvarili zgoraj

```
int zadnji = beseda.Length - 1; //zadnji indeks
Console.Write("zadnji znak: " + beseda[zadnji]);
```

Kot indeks smo uporabili spremenljivko, ki pa mora pripadati tipu *int*, ker je indeks celo število.

Izpis

Ponavljal se bo postopek izpisa enega znaka, zato uporabimo zanko. Pri delu z nizi zelo pogosto pride v poštev zanka *for*, pri kateri se števec v glavi zanke uporabi kot indeks. Če za začetno vrednost števca uporabimo 0, v pogoju pa rečemo, da števec teče do dolžine niza in se povečuje za 1 in potem ta števec uporabimo kot indeks, se bomo ob vsaki izvedbi zanke po vrsti sklicevali na enega od znakov v nizu.

```
string s = "Kalkulator.";          //niz dobi vrednost
for (int i = 0; i < s.Length; i++)
     Console.WriteLine(s[i]); //izpis posameznega
                              //znaka iz niza
```

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.

Dolžina niza *s* je 11, ker ima niz 11 znakov (šteje tudi pika). Največja vrednost števca glede na pogoj *i* < *s.Length* bo 10, kar je tudi indeks zadnjega znaka v nizu.

Utrjevanje snovi

1. Preveri, kaj izpiše naslednji program in dobljeni izpis komentiraj.

```
string stavek = "Vaja dela mojstra, mojster pa veajo.";
for (int k = 3; k < stavek.Length - 5; k = k + 4)
     Console.Write(stavek[k]+" ");
Console.ReadLine();
```

2. Napiši program, ki ob deklaraciji vnesen niz (stavek) izpiše tako, da bo vsaka beseda v svoji vrsti. (Namig: Znake izpisujemo v zanki po enega. Vsak znak preverimo in če je presledek, skočimo v novo vrsto.)

Napiši program, ki s tipkovnice prebere nek stavek, nato pa v njem prešteje vse samoglasnike in rezultat štetja izpiše.

Štetje samoglasnikov

Za štetje samoglasnikov potrebujemo spremenljivko, ki prevzame vlogo števca. Stavek nato vnesemo in s pomočjo zanke *for* preverimo vsak znak v nizu. V pogojnem stavku ga primerjamo z vsemi petimi samoglasniki. Pogoj v pogojnem stavku zapišemo

```
stavek[i]=='a'||stavek[i]=='e'||stavek[i]=='i'|| …
```

Če je pogoj izpolnjen, je znak samoglasnik in vrednost števca povečamo za 1.

Kombinacija zanke in pogojnega stavka

V tem programu bomo pogojni stavek uporabili znotraj zanke. Zanka po vrsti izbira znake iz niza, pogojni stavek pa jih preverja.

```
for(/*zanka gre čez celoten stavek*/)
{
     if(/*pogoj*/)
     {
          //povečamo števec
     }
}
```

Ko se zanka konča, izpišemo vrednost števca.

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.

POZOR!
Kadar pišemo pogojni stavek ali zanko, je priporočljivo najprej z oklepaji {} označiti začetek in konec, nato pa vsebino vrivamo, ker bo v tem primeru tekstovni urejevalnik poskrbel za pravilno obliko programske kode.

Napiši program za preverjanje veljavnosti gesla. Pri vnosu posameznega znaka naj uporabnik na zaslonu vidi zvezdico. Geslo naj bo dolgo 8 znakov. Ob pravilnem vnosu gesla naj se zažene Internet explorer.

Slika 16: okno za vnos gesla

Potrebujemo dva znakovna niza. V prvem bo shranjeno geslo, ki ga ugibamo in ga bomo imenovali *geslo*. V drugega, ki ga bomo imenovali *pass*, pa bomo shranili geslo, ki ga vnese uporabnik. Program bo po vnosu preveril, če sta gesli enaki.

Vnos pravega gesla

Tega vnesemo v sami programski kodi ob inicializaciji spremenljivke tipa *string* z imenom *geslo*:

```
string geslo = "geslo123";
```

Geslo je dolgo 8 znakov, določi ga programer, uporabnik ga ne more spremeniti.

Vnos uporabnikovega gesla

Tega vnašamo po posameznih znakih, da je možna nadomestitev znakov z zvezdico. Dolžina gesla naključnemu uporabniku ne sme biti poznana, zato uporabimo pri vnosu znakov zanko *for*, ki ima večje število ponovitev, kot je znakov v geslu (npr. 12). Po vnosu 12 znakov se bo vnos samodejno prekinil.

```
for (int i = 0; i < 12; i++)
{
      char znak = Console.ReadKey(true).KeyChar;
}
```

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.

Za vnos posameznega znaka uporabimo metodo *ReadKey().KeyChar*. Metoda *ReadKey()* prebere znak brez čakanja na tipko <ENTER>. Če ji v oklepaj damo argument *true*, bodo vneseni znaki nevidni, kar je pri vnosu gesla zaželeno. Lastnost *KeyChar* pretvori vneseni znak v podatkovni tip *char*.

Prekinitev vnosa

Vnos znakov končamo s pritiskom na <ENTER>. To pomeni, da moramo vsak znak preveriti. Tipko <ENTER> lahko preverjamo le preko njene ASCII kode, ki je 13. Če je vneseni znak <ENTER>, prekinemo zanko za vnos znakov.

```
if ((int)znak == 13)
     break;    //prekine zanko
```

Znak moramo pred primerjavo pretvoriti v število, ki prav tako predstavlja njegovo ASCII kodo.

Dodajanje znakov v geslo

Ustvarimo še znakovni niz *pass*. Njegova vsebina naj bo na začetku prazen niz, v zanki pa mu bomo dodajali vnesene znake.

```
string pass = "";   //prazen znakovni niz
```

Znake lahko v niz dodajamo z operatorjem +. To storimo po tem, ko smo se prepričali, da vneseni znak ni <ENTER>.

```
pass = pass + znak; //dodamo znak v niz
Console.Write("*"); //izpis zvezdice
```

Po vnosu znaka izpišemo še zvezdico.

Primerjava vnesenega gesla s pravim in zagon Internet explorerja

Primerjavo naredimo s pomočjo pogojnega stavka, kjer s pomočjo operatorja == primerjamo enakost dveh nizov. Če primerjava uspe, pokličemo zagonski ukaz za zagon Internet explorerja, sicer pa uporabnika obvestimo, da je geslo napačno.

```
if (pass == geslo)
     Process.Start("IExplore");  //klic sistemskega
                 //ukaza za zagon Internet explorerja
else Console.WriteLine("GESLO NI PRAVILNO!");
```

Za klic metode *Start()* moramo vključiti sistemsko knjižnico (*namespace*) Diagnostics, v kateri se nahaja razred Process. V tem razredu je namreč definirana metoda *Start()*. Knjižnico vključimo z ukazom *using* v zaglavju programa.

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.

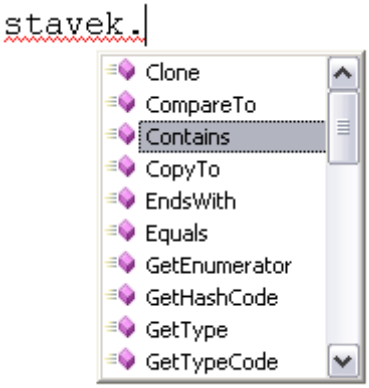
```
using System.Linq;
using System.Text;
using System.Diagnostics;   //dodamo knjižnico
```

Uporaba metod za delo z znakovnimi nizi

1. **Program naj s tipkovnice prebere dva znakovna niza, nek stavek in neko besedo, nato pa naj ugotovi, če se ta beseda nahaja v prebranem stavku.**

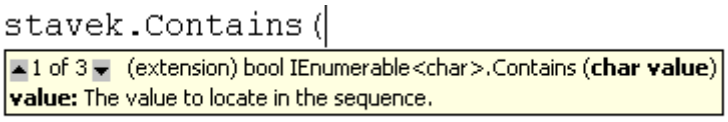
Iskanje besede v stavku

Pomagamo si z metodo *Contains()*. Ustvarimo tekstovno spremenljivko z imenom stavek in si oglejmo seznam metod za delo z znakovnimi nizi:



Slika 17: seznam metod za delo z znakovnimi nizi

Iz seznama z dvoklikom izberemo metodo Contains() in si ogledamo njeno deklaracijo.



Slika 18: deklaracija metode Contains()

Metoda v nizu, na katerem jo uporabimo (v našem primeru je to niz z imenom *stavek*), poišče nek znak ali nek niz znakov, ki ga podamo kot argument metode. Deklarirana je kot tip *bool*, kar pomeni, da vrne vrednost *true*, če je iskanje uspešno in vrednost *false*, če iskanje ni uspešno.

Izpis ugotovitve

Na osnovi vrnjene vrednosti *true* ali *false* naredimo izpis. Metodo kličemo lahko kar v pogoju *if* stavka.

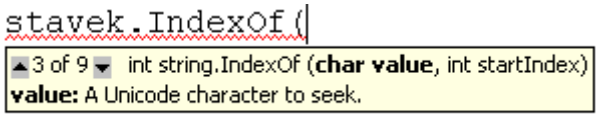
```
if(stavek.Contains(beseda))
     Console.Write("Beseda " + beseda + " je v stavku.");
else
     Console.Write("Besede " + beseda + " ni v stavku.");
```

V pogoju je primerjava z vrednostjo *true* že privzeta primerjava in je ni potrebno izrecno navesti.

2. V prebranem stavku naj program ugotovi, na katerih mestih se nahajajo presledki in to sporoči v konzolnem oknu.

Določitev mesta presledka

Mesto nekega znaka v nizu določa njegov indeks, upoštevati moramo le, da se indeksi začnejo z vrednostjo 0, mi pa bomo šteli mesta od 1 naprej. Uporabili bomo metodo *IndxOf()*, ki v nizu poišče nek znak (ali znakovni niz), vrne pa njegov indeks. Če je v nizu več enakih znakov, vedno poišče le prvega.



Slika 19: deklaracija metode IndexOf()

Iskanje presledkov

Ker je presledkov več, metoda pa najde le prvega, bomo morali uporabiti zanko, da se bo iskanje nadaljevalo. V metodi lahko nastavimo začetno mesto iskanja (*startIndex*), iskanje pa nadaljujemo na mestu, kjer smo našli zadnji presledek.

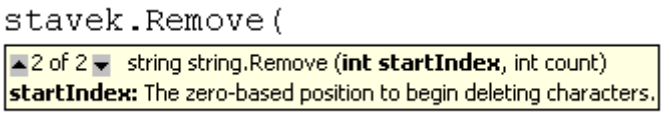
```
int index = stavek.IndexOf(' ');
                          //indeks prvega presledka
while (index > -1)        //v zanki poiščemo še ostale
{
     Console.Write((index + 1) + " ");
     index = stavek.IndexOf(' ', index+1);
}
```

V zanki najprej izpišemo indeks prvega presledka, povečan za 1, nato pa nadaljujemo iskanje pri naslednjem znaku (*index+1*). Če metoda ne najde iskanega znaka, vrne vrednost -1, kar uporabimo pri pogoju za konec iskanja. Iskanje se nadaljuje, dokler je vrednost spremenljivke *index* večja od 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.

3. Program iz prebranega stavka odstrani vse samoglasnike. (Isto nalogo bomo rešili s pomočjo metod Remove() in Replace().)

Uporaba metode *Remove():*



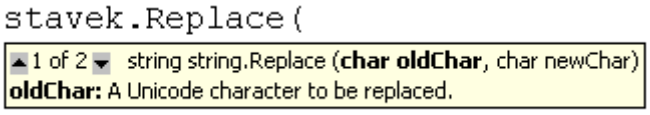
Slika 20: deklaracija metode Remove()

Metoda *Remove()* iz izbranega znakovnega niza odstrani del tega niza od nekega indeksa dalje (argument *startIndex*). Število odstranjenih znakov lahko izberemo (argument *count*), če tega ne storimo, pa odstrani vse znake do konca niza.
V našem primeru bomo morali s pomočjo pogojnega stavka *if* poiskati vse samoglasnike. Ko samoglasnik najdemo, ga odstranimo in nadaljujemo iskanje pri naslednjem znaku, ki sledi v nizu.

```
for (//zanka gre od začetka do konca stavka)
{
     if (//trenutni znak je samoglasnik)
     {
          stavek = stavek.Remove(i, 1); //odstrani zank
          i--;                          //korekcija indeksa
     }
}
```

Ko odstranimo samoglasnik iz stavka, moramo na novo dobljeni stavek shraniti. Pri tem se število znakov v stavku zmanjša za 1 in vsi nadaljnji indeksi se spremenijo, zato je potrebna korekcija indeksa.

Uporaba metode *Replace()*



Slika 21: Deklaracija metode Replace()

Ta metoda izbrani znakovni niz v stavku nadomesti z drugim izbranim znakovnim nizom. V našem primeru bomo samoglasnik nadomestili s praznim nizom.

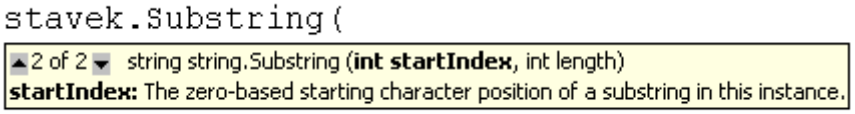
```
stavek = stavek.Replace("a","");
```

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.

V zgornjem primeru bo metoda *Replace()* nadomestila vse znake `"a"` v stavku, zato zanka *for* ni potrebna. Ker je samoglasnikov pet, moramo enako, kot smo storili za a, narediti še za ostale štiri.

Utrjevanje snovi

1. Podobno kot pri ostalih metodah za delo z znakovnimi nizi na osnovi deklaracije razišči še metodo *Substring()*.



Slika 22: deklaracija metode Substring()

S pomočjo te metode vzemi iz stavka nek niz znakov in ga shrani pod imenom *podniz*.

2. Program naj iz sledečega teksta izpiše vse besede, ki se začnejo na črko *s*.

Zvečer, ko so ljudje po hišah zaspali, so se Mišmašu živo zaiskrile oči, vzel je leščerbo in se odpravil po strmih stopnicah v klet. Z velikim ključem je odklenil težka, škripajoča vrata in stopil v podzemno dvorano. Tu so ob stenah iz rezanega kamna stale visoke police, na njih pa vreče z moko, zdrobom, soljo in sladkorjem, malo dalje so stali veliki vrči smetane in mleka, zavitki kvasa vseh vrst in vrečice z dišavami. Na sredi dvorane so bila tri lesena korita za gnetenje testa, zadaj pa je žarela stara krušna peč. ***Vir: Svetlana Makarovič, Pekarna MIšmaš***

Namig: Poišči indeks črke b in presledka, ki mu sledi in ju uporabi kot argumenta metode *Substring()*. Indeks črke *s* prestavlja *startindex*, *length* pa dobimo z razliko obeh indeksov. Postopek ponavljaš v zanki *for*.

45. Čemu je namenjen podatkovni tip *string*?
46. Kako označujemo začetek in konec znakovnega niza?
47. Kaj je dolžina znakovnega niza in kako jo lahko ugotovimo s pomočjo programa? Ali presledki v nizu vplivajo na dolžino?
48. V čem je razlika, če neko število shranimo kot *string* ali kot *int*?
49. Kaj pri znakovnem nizu predstavlja indeks?
50. Na kakšen način lahko dodajamo znake v znakovni niz?
51. Katere metode za delo z znakovnimi nizi smo spoznali v tem poglavju in za kaj se uporabljajo?
52. Kakšen je postopek za vključitev sistemskega ukaza v programsko kodo?

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.

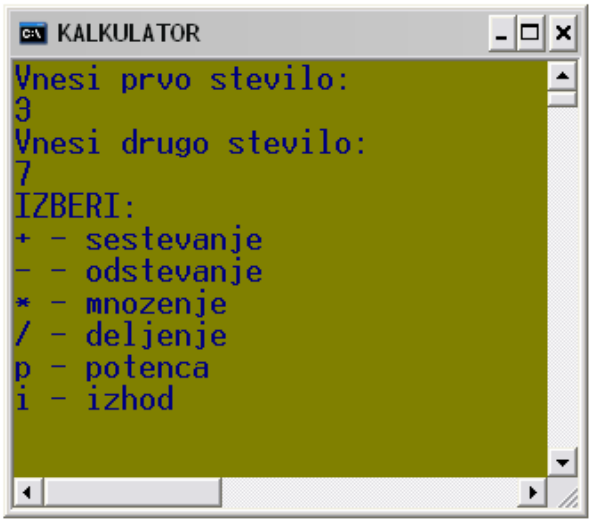
STAVEK SWITCH
– pogojni stavek z več možnostmi

Spoznali smo že, da računalnik zna izbrati med dvema možnostma, ki sta na voljo. V nekaterih primerih pa je teh možnosti več in uporabiti bi morali več zaporednih pogojnih stavkov ali pa poseben stavek, ki lahko to zaporedje pogojnih stavkov nadomesti. Ta stavek omogoča, da med ponujenimi možnostmi eno izberemo.

Pri tej vaji si bomo ogledali uporab pogojnega stavka, ki ima na izbiro več možnosti. Pri tem si bomo ogledali:

- obliko stavka *switch*
- izhod iz programa z ukazom za izhod
- uporabo neskončne zanke in njeno prekinitev.

Sestavi program, ki bo predstavljal preprost kalkulator za izračun vsote, razlike, zmnožka, količnika in potence dveh števil. Program naj uporabniku ponudi preprost meni za izbiro operacije, po izvršeni operaciji naj se vrne v meni za izbiro nove operacije.



Slika 23: primer preprostega menija

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.

Spremenljivka za izbiro operacije

Za izbiro operacije moramo izbrati enega od znakov, ki jih ponuja meni. Uporabimo znakovno spremenljivko, ki bo nato v stavku *switch* odločala, katera možnost bo izbrana in jo vnesemo takoj za izpisom menija.

```
char izbira = Console.ReadKey().KeyChar;
```

Stavek *switch*

V tem stavku imamo na izbiro več možnosti, od katerih pa bo izbrana le ena. O izbiri odloča spremenljivka, katere ime navedemo v glavi stavka. Če se njena vrednost ujema z vrednostjo, ki je navedena pri kateri od možnosti v telesu stavka, je ta možnost izbrana. Vrednosti so lahko cela števila (*int*), znaki (*char*) ali znakovni nizi (*string*), ne morejo pa biti decimalna števila.

```
switch (/*spremenljivka za izbiro možnosti*/)
{
     case /* vrednost 1 */:
          //vsebina prve možnosti
          break;
     case /* vrednost 2 */:
          //vsebina druge možnosti
          break;
     case /* vrednost 3 */:
          //vsebina tertje možnosti
          break;
     default:
          //vsebina možnosti default
          break;
}
```

Z besedo *case* začnemo posamezno možnost, nato sledi vrednost, na osnovi katere bo ta možnost izbrana in dvopičje.
Vsebina posamezne možnosti je programska koda, ki pove, kaj se pri izbiri te možnosti zgodi. Lahko ima več vrstic, končamo pa jo s stavkom break, ki nas vrže na konec stavka *switch*. S tem se njegovo izvajanje konča in zato je lahko izbrana le ena od možnosti.
Število možnosti v stavku *switch* ni omejeno, zadnja možnost pa je možnost *default*, ki se izvede, če nobena od ostalih možnosti ni bila izbrana. Ta možnost ni obvezna in jo lahko tudi izpustimo.

Primer za seštevanje in odštevanje:

Telo stavka na začetku in koncu vedno označimo z zavitimi oklepaji.

```
switch (izbira)     //glava stavka
{
```

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.

```
            case '+':
                 Console.WriteLine("Vsota je: "+(a+b));
                 break;
            case '-':
                 Console.WriteLine("Razlika je: "+(a-b));
                 break;
    //Tu na enak način dodamo še možnosti za množenje,
    //delejnje in potenco ter možnost za izhod iz programa.
    //Na koncu dodamo še možnost default.
     }
```

Vsota bo izbrana z vnosom znaka '+', razlika pa z vnosom znaka '-'. Ta dva znaka predstavljata vrednosti, ki jih navedemo ob besedah case. Ker so vrednosti znakovne, morajo biti v narekovajih. (Pri številčnih vrednostih, narekovajev ni, pri znakovnih nizih pa bi uporabili dvojne narekovaje.) Vsebina možnosti bo izpis vsote oziroma razlike vnesenih števil. Možnost za izračun vsote bo izbrana, kadar bo vrednost spremenljivke *izbira* enaka vrednosti '+'.

Izhod iz programa

Ta se zgodi, če izberemo znak *'i'*, ki ga kot vrednost navedemo ob besedi *case*.

```
case 'i':
     Console.WriteLine("Pritisni ENTER za izhod!");
     Console.ReadLine();  //program čaka na <ENTER>
     Environment.Exit(1); //izhod iz programa
     break;
```

Možnost default

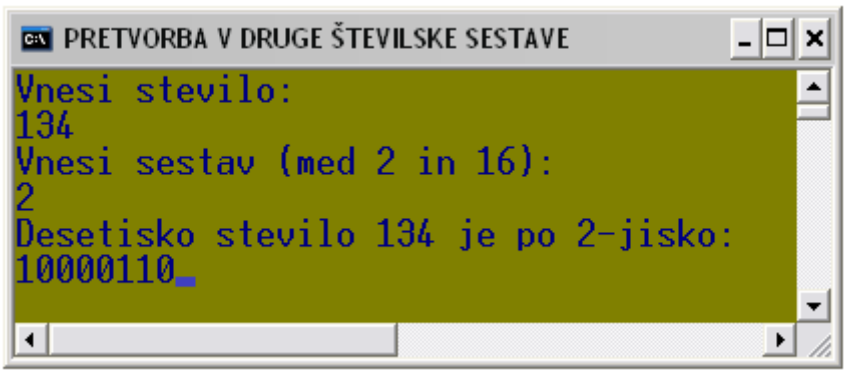
To možbnost bomo izkoristili v primeru, ko uporabnik izbere znak, ki ga meni ne ponuja, da mu to sporočimo.

```
default:
     Console.WriteLine("NAPACNA IZBIRA!!");
     break;
```

Pri tej možnosti se vrednost ne navaja, prav tako kot ostale pa jo moramo zaključiti s stavkom *break*.

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.

Sestavi program, ki vneseno desetiško število pretvori v drug številski sestav. Pri tem imamo na izbiro vse številske sestave od dvojiškega do šestnajstiškega



Slika 24: primer izpisa

Postopek pretvorbe iz desetiškega v nek drug sestav

Število pretvorimo tako, da ga delimo s številom, ki predstavlja nek sestav (če pretvarjamo npr. v osmiški sestav, delimo z 8), tako dolgo, da to število postane enako 0. Postopek deljenja se torej ponavlja v zanki, zanimajo pa nas ostanki po deljenju. Prebrani v obratnem vrstnem redu nam dajo rezultat pretvorbe.
Primer postopka:

222 deljeno z 8 je 27, ostane 6
27 deljeno z 8 je 3, ostane 3
3 deljeno z 8 je 0, ostane 3

Ker je rezultat deljenja 0, je pretvorbe konec, iz ostankov pa dobomo število 336.

Pretvorba v dvojiški sestav

Najprej bomo sestavili program za pretvorbo v dvojiški sestav in ga nato razširili tako, da bo deloval še za ostale sestave.
Uporabili bomo zanko while, v kateri najprej poiščemo ostanek po deljenju števila z 2, nato pa število delimo z 2, da dobimo novo vrednost števila. Zanko ponavljamo, dokler je vrednost števila večja od 0.

```
while (stevilo > 0)
{
     int ostanek = stevilo % 2;   //izračun ostanka
     stevilo = stevilo / 2;       //število delimo z 2
```

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 tem mestu moramo ostanek še shraniti
    }
```

Shranjevanje ostanka

Ostanke moramo shraniti v pomnilnik, da jih bomo na koncu lahko izpisali v obratnem vrstnem redu. Najprimerneje bo, če jih shranimo kot znakovni niz in ob vsakem izračunu ostanka tega v obliki znaka dodamo v niz. Pri tem si pomagamo s stavkom *switch*. V primeru, da je vrednost ostanka 1, shranimo znak `"1"`, v primeru, da je vrednost ostanka 0, pa shranimo znak `"0"`.

```
switch (ostanek)
{
    case 0: rezultat = rezultat + "0"; break;
    case 1: rezultat = rezultat + "1"; break;
}
```

Možnost *default* je tu odveč, ker drugačne vrednosti kot 0 in 1 niso možne. Pri dvojiškem sestavu bi situacijo lahko rešili tudi s stavkom *if*, pri ostalih sestavih pa bo možnosti ostanka več in bo stavek *switch* primernejši.

Izpis v obratnem vrstnem redu

Uporabimo zanko for, izpisovati začnemo pri zadnjem znaku in končamo pri prvem. Indeks zadjega znaka je vedno za 1 manjši od dolžine znakovnega niza.

```
for(int i = rezultat.Length - 1; i >= 0; i--)
    Console.Write(rezultat[i]);
```

Ostali številski sestavi

Uporabniku moramo na začetku dati možnost, da izbere sestav. Tega naj izbere v obliki številke.

```
Console.WriteLine("Vnesi sestav (med 2 in 16): ");
int sestav = Convert.ToInt32(Console.ReadLine());
```

Spremenljivko *sestav* bomo pri pretvorbi uporabili namesto števila 2.

```
int ostanek = x % sestav;    //izračun ostanka
x = x / sestav;              //število delimo
```

Če delimo s številom, ki je večje od 2, bomo imeli večji nabor možnih ostankov in zato bo imel stavek *switch* več možnosti. Pri šestnajstiškem sestavu je teh možnosti 16 in ker su tu zajeti tudi vsi možni ostanki za ostale sestave, bomo priredili stavek *switch* šestnajstiškemu sestavu.

```
switch (ostanek)
{
```

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.

```
            case 0: rezultat = rezultat + "0"; break;
            case 1: rezultat = rezultat + "1"; break;
            case 2: rezultat = rezultat + "2"; break;
            case 3: rezultat = rezultat + "3"; break;
            case 4: rezultat = rezultat + "4"; break;
            case 5: rezultat = rezultat + "5"; break;
            case 6: rezultat = rezultat + "6"; break;
            case 7: rezultat = rezultat + "7"; break;
            case 8: rezultat = rezultat + "8"; break;
            case 9: rezultat = rezultat + "9"; break;
            case 10: rezultat = rezultat + "A"; break;
            case 11: rezultat = rezultat + "B"; break;
            case 12: rezultat = rezultat + "C"; break;
            case 13: rezultat = rezultat + "D"; break;
            case 14: rezultat = rezultat + "E"; break;
            case 15: rezultat = rezultat + "F"; break;
    }
```

53. V katerih primerih uporabimo stavek *switch*?
54. Na kakšen način v stavku *switch* izberemo eno od možnosti?
55. Kdaj se izvede možnost *default* in kaj se zgodi, če te možnosti ni?
56. Katerim podatkovnim tipom lahko pripada spremenljivka, ki je v stavku *switch* namenjena izbiri možnosti?
57. Kaj predstavlja operator %?
58. Kakšen je postopek za izpis znakovnega niza v obratnem vrstnem redu?

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.

METODE
– razdelitev programske kode na manjše gradnike

Pri pisanju programa daljše programe običajno razdelimo na manjše gradnike, ki jih imenujemo metode. Tak program je preglednejši in omogoča, da v naš program vključujemo tudi metode, ki so jih napisali drugi programerji. Načine vključevanja takšnih metod smo že spoznali, sedaj pa si moramo ogledati še, kako naredimo svojo metodo.

Naučili se bomo, kako v jeziku C# sami napišemo neko metodo. Pri tem si bomo ogledali:

- deklaracijo metode
- klic metode
- določitev parametrov, ki jih pošljemo neki metodi
- uporabo stavka *return*
- orodje za kreiranje nove metode.

Napiši program za izračun prostornine valja. Podatke o višini in radiju osnovne ploskve vnesemo v glavni metodi *Main()*, nato pa pokličemo metodo *Volumen()* za izračun volumna, ki jo napišemo sami. Rezultat naj metoda *Volumen()* vrne glavni metodi, ki ga nato izpiše.

Razdelitev programske kode na več metod

Naš program vedno oblikujemo kot razred (*class*) z imenom *Program*. Vsak razred pa lahko vsebuje več metod, od katerih ima vsaka svojo nalogo, nad njimi pa bdi glavna metoda *Main()*, ki je edina obvezna. Ko program poženemo, se vedno začne izvajati na začetku te metode in konča na koncu te metode. Vse ostale metode se izvedejo vmes, vendar le če jih pokličemo.

Glavna metoda Main()

Ko poženemo *Visual Studio* v konzolnem načinu, se nam v tekstovnem urejevalniku že pojavi ogrodje glavne metode.

```
static void Main(string[] args) //glava metode
{
    //telo metode
}
```

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.

Glava metode predstavlja deklaracijo metode, telo metode pa je programska koda, ki pove, kaj metoda naredi.

Deklaracija metode

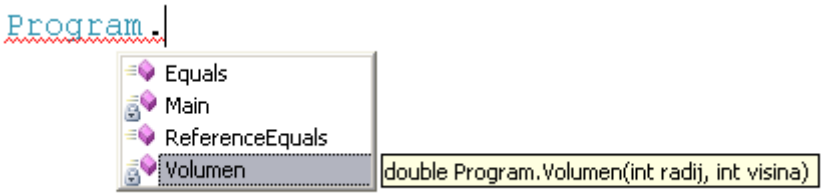
Metode vedno deklariramo eno za drugo (nikoli eno znotraj druge!), pri tem vrstni red ni pomemben. Pri deklaraciji navedemo tip metode, ime metode in spremenljivke, ki jih bomo potrebovali za prevzem argumentov.

- **Tip metode** pove, kakšno vrednost metoda vrne. Tu imamo dve možnosti:
  1. Metoda ne vrača vrednosti in pripada tipu *void*. Takšne metode so običajno namenjene izpisu podatkov ali urejanju izpisa.
  2. Metoda vrne vrednost, ki jo izračuna. V tem primeru izberemo enega od podatkovnih tipov, ki jih uporabljamo pri deklaracijah spremenljivk (*int*, *double*, *char*…) glede na izračunano vrednost. Metoda lahko vrne samo eno vrednost.
- **Ime metode** izberemo sami glede na to, čemu je metoda namenjena. Pri imenih metod praviloma uporabljamo veliko začetnico.
- **Argumenti metode** so podatki, ki jih metoda dobi od zunaj, da lahko opravi svojo nalogo. (Npr. metoda za izračun kvadratnega korena mora kot argument dobiti število, ki ga bo korenila, metoda za zaokroževanje mora dobiti število, ki ga bo zaokrožila in število decimalk…)

Deklaracija metode za izračun volumna:

```
static double Volumen(int radij, int višina)
{
      //koda za izračun volumna
}
```

Najprej napišemo besedo *static*, katere pomen bomo spoznali kasneje. Sledi tip metode, v našem primeru je to *double*, ker pričakujemo kot rezultat izračuna decimalno število. Za ime smo izbrali besedo *Volumen*, ker je metoda namenjena izračunu volumna. Ob imenu v oklepajih deklariramo spremenljivke, ki bodo prevzele argumente. V našem primeru bosta to radij osnovne ploskve in višina valja, ki bosta oba celi števili, zato tu deklariramo dve spremenljivki tipa *int*. Ko smo metodo deklarirali, jo lahko poiščemo na spustnem seznamu vsebine razreda *Program*.



Slika 25: vsebina razreda Program

Vidimo, da je pomembno, kakšna imena izberemo za argumente, ker se ta imena pojavijo v oknu za pomoč in na osnovi tega programer, ki bo uporabil metodo, ve, katere parametre potrebuje.

Klic metode

Metodo pokličemo, ko jo potrebujemo. Ob klicu navedemo ime metode, v oklepaju pa navedemo vrednosti, ki jih pošiljamo. V našem primeru bomo metodo Volumen poklicali iz glavne metode, ko bomo imeli pripravljene podatke.

```
static void Main(string[] args) //glava metode
{
     //vnos radija in višine:
     int r = Convert.ToInt32(Console.ReadLine());
     int v = Convert.ToInt32(Console.ReadLine());
     Volumen(r, v);    //klic metode
     //izpis rezultata
}
```

Ker smo pri vnosu radia in višine njune vrednosti shranili v pomnilnik, bomo kot argument navedli imena spremenljivk, kjer sta ti vrednosti shranjeni.
Vrednosti parametrov se prenašajo po vrstnem redu, kot so navedeni v oklepaju in po takšnem vrstnem redu jih sprejmejo spremenljivke, ki so deklarirane v glavi metode!

Vrnjena vrednost

Ko metoda opravi svoje delo, vrne izračunano vrednost. To stori s pomočjo rezervirane besede *return*, ki predstavlja zadnjo vrstico v telesu metode. (Metode tipa *viod* nimajo stavka *return*, ker ne vračajo vrednosti!)

```
static double Volumen(int radij, int visina)
{
     //koda za izračun volumna:
     double volumen = Math.PI*Math.Pow(radij,2)*visina;
     return volumen;     //vrne izračunano vrednost
}
```

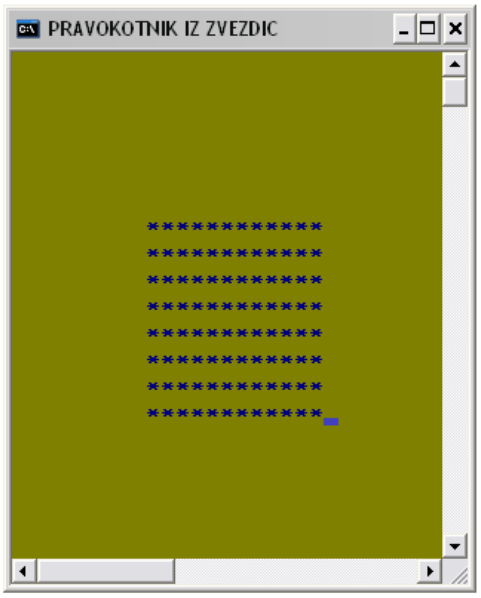
Na mestu, kjer smo metodo klicali, moramo vrnjeno vrednost sprejeti. Imamo dve možnosti:

- naredimo novo spremenljivko in vrednost shranimo in jo nato izpišemo:
  ```
  double volumen = Volumen(r, v);
  Console.Write("Prostornina valja: " + volumen);
  ```
- ali pa vrnjeno vrednost izpišemo:
  ```
  Console.Write("Prostornina valja: " + Volumen(r, v));
  ```

V drugem primeru predstavlja klic metode *Volumen()* argument metode *Write()*.

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.

Napiši program, ki v sredino pogovornega okna izriše pravokotnik iz zvezdic. Velikost stranic pravokotnika vnesi v glavni metodi, za risanje pa napiši posebno metodo, ki dobi podatka o velikosti kot argumenta. Program nato dopolni tako, da bo izrisal le zunanji rob pravokotnika, notranjost pa bo prazna.



Slika 26: izris pravokotnika na sredino

Tip metode

Ker je metoda namenjena risanju, ne bo nobene izračunane vrednosti, ki bi jo na koncu vrnila. Zato bo pripadala tipu *void*.

Risanje

Za risanje uporabimo **gnezdeno zanko** *for* in metodo za nastavitev položaja izpisa *SetCursorPosition()*. Gnezdena zanka pomeni zanko, ki se nahaja znotraj druge zanke. V našem primeru notranja zanka riše vrstice, zunanja zanka pa nas premakne v novo vrstico. **Pri gnezdenih zankah morajo biti imena števcev v zankah različna!**

Primer gnezdene zanke *for*:

```
for (int y = 0; y < 10; y++)          //ponovimo 10-krat
{
     for (int x = 0; x < 20; x++)     //izpis 20 zvezdic
```

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.

```
        {
            //izpis zvezdice
        }
    }
```

V zgornjem primeru se notranja zanka izvede 20-krat in predstavlja izpis ene vrstice, ker pa se nahaja v zanki, se bo to zgodilo 10-krat.

Nastavitev položaja izpisa

Metoda *SetCursorPosition()* nas v konzolnem oknu postavi v določeno točko. Okno si predstavljamo kot koordinatni sistem, mi pa določimo odmik v desno (x) in odmik navzdol (y), x in y pa sta parametra metode *SetCursorPosition()*. Privzeta velikost konzolnega okna je 80 znakov v smeri x in 24 v smeri y. Na sredino bi se postavili z ukazom

```
Console.SetCursorPosition(40, 12);
```

Izris v levi zgornji kot

Če v metodi za nastavitev položaja izpisa kot argumenta postavimo števca iz gnezdene zanke *for*, dobimo izris v levi zgornji kot, ker števca tečeta od vrednosti 0 naprej.

```
for (int y = 0; y < b; y++)
{
    for (int x = 0; x < a; x++)
    {
        Console.SetCursorPosition(x, y);
        Console.Write("*");
    }
}
```

V pogojih za konec izvajanja zanke moramo upoštevati vrednosti, ki smo jih dobili kot parametre. V ta namen bomo v glavi metode za risanje ustvarili spremenljivko *a* za širino in *b* za višino lika. Deklaracija metode za risanje:

```
static void Risanje(int a, int b)
{

}
```

Premik na sredino

Začetek izpisa moramo premakniti za 40 mest v desno, nato pa še za polovico stranice a v levo. To pomeni, da spremenljivki *x* v notranji zanki *for* prištejemo 40 in odštejemo polovico spremenljivke *a* za določitev začetka vrstice, za določitev konca pa prištejemo 40 in polovico stranice *a*.

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.

```
for (int x = 40-a/2; x < 40+a/2; x++)
{
     Console.SetCursorPosition(x, y);
     Console.Write("*");
}
```

Enako storimo tudi v zunanji zanki z upoštevanjem, da je srednja vrstica dvanajsta in da je višina lika stranica b.

Risanje pravokotnika s prazno notranjostjo

Problem rešimo tako, da preko pravokotnika iz zvezdic narišemo še pravokotnik iz presledkov, ki pobriše zvezdice. Stranici drugega pravokotnika morata biti za dve točki manjši od stranic prvega. (Na spodnji sliki ima višina zunanjega pravokotnika 5 zvezdic, višina notranjega pa 3 presledke. Presledke vidimo kot prazen prostor.)

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

Slika 27: pravokotnik s prazno notranjostjo

Za risanje lahko uporabimo isto metodo, ki jo moramo zato ponovno poklicati. Metodi dodamo še en argument, ki bo predstavljal znak, ki ga metoda uporabi pri risanju. To omogoči metodi izbiro znaka pri risanju, ker bo znak v metodi *Write()* nastopil kot spremenljivka. Ob prvem klicu metodi pošljemo zvezdico, ob drugem pa presledek.

```
//spremenjena deklaracija metode:
static void Risanje(int a, int b, char znak)
{
}
```

Spremeniti moramo tudi vrednosti ostalih dveh parametrov.

```
//primer klica metode za risanje zvezdic:
Risanje(a, b, '*');
//primer klica metode za risanje presledkov:
Risanje(a - 2, b - 2, ' ');
```

Spremenljivki a in b predstavljata vnesene vrednosti širine in višine, v drugem primeru sta zmanjšani za 2.

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.

Napiši metodo, ki v nekem stavku prešteje število izbranih znakov. Stavek in izbrani znak vnese uporabnik v glavni metodi in ju nato kot argumenta pošlje metodi. Program nato spremeni tako, da bo stavek shranjen kot statična spremenljivka.

Statične in dinamične spremenljivke

Spremenljivke, ki smo jih srečali do sedaj, so bile dinamične. Ustvarjene so bile znotraj nekega stavčnega bloka (metoda, zanka …) in so bile zato veljavne le znotraj tega bloka. Statično spremenljivko pa ustvarimo na nivoju celotnega razreda in bo zato veljavna v vseh metodah, ki so deklarirane v tem razredu.

```
class Program
{
     static int stat = 10;
     static void Metoda()
     {
          int din1 = 10;
          for (/*.....*/)
          {
               int din2 = 10;
          }
    }
}
```

Spremenljivka *stat* je ustvarjena na nivoju razreda *Program*, spremenljivka *din1* na nivoju metode, spremenljivka *din2* pa na nivoju zanke *for* (izven te zanke ne obstaja). Pri deklaraciji statične spremenljivke moramo uporabiti besedo *static*.

Stavek kot dinamična spremenljivka

Spremenljivki stavek in znak ustvarimo v glavni metodi.

```
static void Main(string[] args)
{
     string stavek = Console.ReadLine();
     char znak = Convert.ToChar(Console.ReadLine());
     //klic metode za štetje znakov:
     int stevilo = Prestej(znak, stavek);
     Console.Read();
}
static int Prestej(char znak, string stavek)
```

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 prešteje in vrne število znakov
}
```

Ker izven glavne metode nista vidni (ne obstajata), ju moramo kot argumenta poslati metodi *Prestej()*, ki ju potrebuje za svoje delo. Kot argument se prenese le vrednost spremenljivke, zato imamo v glavi metode (v oklepaju) deklarirane spremenljivke, ki te vrednosti prevzamejo. Pri poslanih vrednostih in spremenljivkah, ki te vrednosti sprejmejo, se mora ujemati:

1. število vrednosti in spremenljivk,
2. vrstni red, v katerem so navedene ene in druge,
3. podatkovni tip (znakovno vrednost naj pričaka spremenljivka tipa *char* …).

Stavek kot statična spremenljivka

Deklaracijo spremenljivke *stavek* premaknemo pred glavno metodo (ne pozabi besede *static*). V tem primeru je ta spremenljivka vidna v vseh metodah, ki so v razredu, kjer je ustvarjena, in je ni potrebno pošiljati kot argument metode.

```
static string stavek = Console.ReadLine();
static void Main(string[] args)
{
      char znak = Convert.ToChar(Console.ReadLine());
      //klic metode za štetje znakov:
      int stevilo = Prestej(znak);
      Console.Read();
}
static int Prestej(char znak)
{
      //metoda prešteje in vrne število znakov
}
```

Štetje znakov v stavku

V tem primeru bi lahko uporabili zanko *foreach*. Ta zanka se od zanke *for* razlikuje po tem, da ne moremo:

1. nastaviti začetne vrednosti števca (ta je vedno 0),
2. ne moremo določiti pogoja (zanko *foreach* lahko uporabimo le na nekem objektu, kot je npr. znakovni niz in vedno teče do zadnjega znaka v nizu),
3. ne moremo določiti koraka povečevanja števca (vedno se povečuje za 1).

V našem primeru potrebujemo prav takšno zanko. Oblika zanke *foreach*:

```
foreach(/*deklaracija znakovne spremenljivke*/ in /*ime
stringa*/)
{

}
```

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.

V glavi zanke deklariramo znakovno spremenljivko, ki po vrsti prevzema znake iz znakovnega niza in preko nje lahko pridemo do posameznega znaka. Za besedi in nato navedemo še ime znakovnega niza, iz katerega pobiramo znake. Metoda za štetje znakov:

```
static int Prestej(char znak)
{
     int i = 0;          //števec, ki šteje znake
     foreach(char z in stavek)   //za vsak znak v stavku:
     {
          if (z == znak) //če je trenutni znak enak
          i++;           //izbranemu, preštejemo znak
     }
     return i;
}
```

Sestavi program, ki mu vnesemo nek stavek, nato pa na osnovi izdelanega menija odloča med izvedbo naslednjih metod:

- **izpisa stavka po diagonali,**
- **nadomestitvijo vseh presledkov z znakom '-',**
- **štetjem številk v stavku,**
- **odstranitvijo vseh presledkov iz stavka**

Stavek preberemo v glavni metodi, kjer izdelamo tudi meni za izbiro metode.

Meni:

Naredimo ga s pomočjo stavka *switch*, kjer v vsaki od možnosti kličemo eno od metod.

```
Console.Write("Izberi:");
Console.Write("\n1-diagonalni izpis");
Console.Write("\n2-zamenjava presledkov z -");
          // ...
int izbira = Convert.ToInt32(Console.ReadLine());
switch (izbira)
{
     case 1://klic metode diagonalni izpis
          break;
     case 2://klic metode za zamenjavo presledkov
          break;
          // ...
     default://napačna izbira
          break;
}
```

Pri klicu vsake metode stavek pošljemo kot argument. Pred klicem metode je priporočljivo brisanje ekrana.

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.

Izpis po diagonali

Uporabimo zanko *for* in metodo *SetCursorPosition()* razreda *Console*. V diagonali sta vrednosti koordinat x in y enaki, njune vrednosti pa se gibljejo od vrednosti 0 pa do zadnjega indeksa v stavku. Zato kot argument metode *SetCursorPosition()* lahko uporabimo kar števec iz glave zanke *for*.

```
for (int i = 0; i < stavek.Length; i++)
{
     Console.SetCursorPosition(i, i);
     Console.Write(stavek[i]);
}
```

Metoda bo opravila izpis v konzolno okno in ne bo vrnila nobene vrednosti, zato bo pripadala tipu *void*.

Nadomestitev presledkov z znakom '-'

Na stavku uporabimo metodo *Replace()*, ki opravi zamenjavo znakov. Dobljeni stavek vrnemo glavni metodi, ki ga izpiše.

```
static string Zamenjaj(string stavek)
{
     stavek = stavek.Replace(' ', '-');
     return stavek;
}
```

Metoda v tem primeru vrne stavek in pripada tipu *string*. Klic metode pa naj bo kar argument metode *WriteLine()*.

Štetje številk

Metoda kot parameter dobi stavek in vrne število, zato bo tipa *int*. Za štetje bomo potrebovali števec, ki ga povečamo, če je trenutni znak med '0' in '9' vključno s tema vrednostma. Rezultat štetja vrnemo glavni funkciji, ki ga izpiše. Za premikanje po stavku lahko uporabimo zanko *foreach*.

```
foreach (char znak in stavek)
     if (znak >= '0' && znak <= '9')
     {
          i++;    //povečamo števec
     }
```

Odstranitev presledkov

Uporabimo metodo *Remove()* in dobljeni niz vrnemo glavni metodi, ki ga izpiše.

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.

Napiši program, v katerem deklariramo tekstovno spremenljivko in vanjo vnesemo nek stavek (ali več stavkov). Nato pokličemo metodo, ki vse črke v stavku razvrsti in izpiše po abecednem redu. Pri razvrščanju ni pomembno, ali so črke velike ali male.

Postopek razvrščanja:

Uporabimo gnezdeno zanko *for*. Zunanja zanka teče od črke 'a' do črke 'z', notranja pa od začetka do konca niza. Če se trenutni znak v nizu ujema s črko, ki je na vrsti v zunanji zanki, ga shranimo (dodamo) v neko novo, za to pripravljeno, znakovno spremenljivko.
V primeru, da naletimo na veliko tiskano črko, jo s pomočjo metode *ToLower()* med preverjanjem enakosti spremenimo v malo črko.

```
string stavek = Console.ReadLine();
string urejen = "";
for (char crka = 'a'; crka <= 'z'; crka++)
     for (int i = 0; i < stavek.Length; i++)
          if(Char.ToLower(stavek[i]) == crka)
               urejen = urejen + crka;
```

Zunanja zanka nam določa vrstni red iskanja črk, najprej v stavku iščemo vse a-je, nato b-je …
Kadar bo v stavku več enakih črk, se bodo vse te črke pojavile tudi v urejenem nizu. Če želimo, da bi se vsaka črka pojavila le enkrat, moramo iskanje prekiniti, ko neko črko najdemo. To storimo s stavkom *break*, ki prekine notranjo zanko.

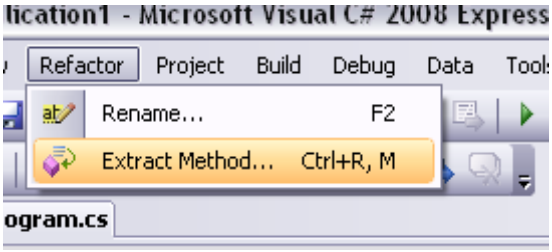
```
for (char crka = 'a'; crka <= 'z'; crka++)
     for (int i = 0; i < stavek.Length; i++)
          if(Char.ToLower(stavek[i]) == crka)
          {
               urejen = urejen + crka;
               break;
          }
```

Stavek *break* vedno prekine le zanko, v kateri se neposredno nahaja. V tem primeru je to notranja zanka *for*.

Kreiranje metode s pomočjo orodja okolja *Visual Studio*

V zgornji orodni vrstici imamo v možnosti *Refactor* orodje *Extract Method*.

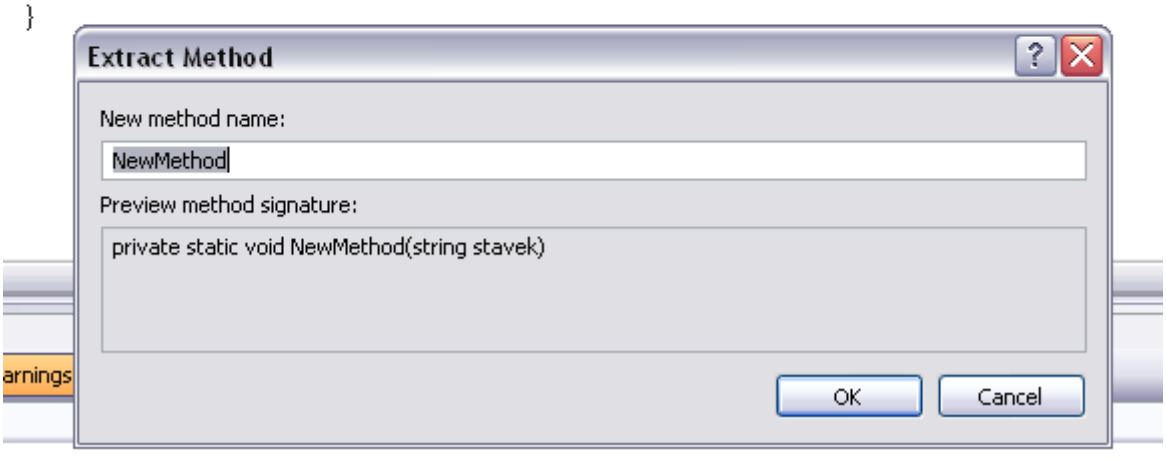
Učno gradivo je nastalo v okviru projekta Munus 2. Njegovo izdajo je omogočilo sofinanciranje Evropskega socialnega sklada Evropske unije in Ministrstva za šolstvo in šport.



Slika 28: možnost Refactor v orodni vrstici

Mi smo našo programsko kodo sedaj napisali v glavni metodi. Dodamo še stavek za izpis urejenega niza v konzolno okno in z miško označimo del kode, ki ga želimo preoblikovati v lastno metodo (na sliki št. 29 je označeni del obarvan sivo). Nato kliknemo na *Extract Method* in pojavi se okno za določitev imena metode.

```
static void Main(string[] args)
{
    string stavek = Console.ReadLine();
    string obrnjen = "";
    for (char crka = 'a'; crka <= 'z'; crka++)
        for (int i = 0; i < stavek.Length; i++)
            if(Char.ToLower(stavek[i]) == crka)
                obrnjen = obrnjen + crka;
    Console.WriteLine(obrnjen);
    Console.ReadLine();
}
```



Slika 29: označen del kode in okno za poimenovanje nove metode

Izberemo ime (npr. Razvrsti, ker bo to metoda za razvrščanje) in stisnemo tipko OK. Pod glvno metodo se pojavi nova metoda, na označenem mestu pa imamo klic te metode.

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

```
{
    string stavek = Console.ReadLine();
    Razvrsti(stavek);   //klic metode Razvrsti()
    Console.ReadLine();
}
//deklaracija metode Razvrsti():
private static void Razvrsti(string stavek)
{
     //vsebina metode
}
```

Metoda pripada tipu *void*, kot argument pa mora iz glavne metode dobiti stavek. (Besedo *private* bomo spoznali kasneje in jo lahko tudi pobrišemo.)

59. Zakaj programsko kodo (razred) razdelimo na manjše gradnike in kako te gradnike imenujemo?
60. Kaj vsebuje deklaracija metode?
61. Kateremu podatkovnemu tipu, ki pri spremenljivkah ne obstaja, lahko pripada metoda? Kakšne so metode, ki pripadajo temu tipu?
62. Čemu je namenjen stavek *return* in kje se v metodi nahaja?
63. Ali lahko ustvarimo (deklariramo) metodo znotraj metode?
64. Od česa je odvisno število in vrstni red pošiljanja argumentov pri klicu neke metode?
65. S pomočjo orodja *Refactor* ustvari metodo za izpis števil od 10 do 50!
66. Kako imenujemo zanko, ki se nahaja znotraj druge zanke?
67. V čem se zanka *foreach* razlikuje od zanke *for*?

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.

REFERENCE
– povezave med spremenljivkami

Nekatere metode lahko obdelajo večje količine podatkov, katerih vrednosti se zato spremenijo. Te spremenjene podatke želimo kasneje uporabiti v nekem drugem delu programa, zato ustvarimo povezavo med podatki. Če so podatki v dveh metodah povezani z referenco, se sprememba vrednosti odraža v obeh metodah.

Namen sklopa je spoznati uporabo referenčnih spremenljivk pri prenosu argumentov med metodami. Ogledali si bomo:

- deklaracijo in inicializacijo referenčne spremenljivke,
- deklaracijo metode, če so argumenti referenčne spremenljivke,
- razliko med operatorjema *ref* in *out*.

Napiši program, ki prebere tri števila, nato pa pokliče metodo, ki vsa negativna števila med njimi spremeni v pozitivna. Pri podajanju argumentov metodi uporabi referenčni način. V glavni metodi nato s pomočjo izpisa preveri pravilnost delovanja programa.
Nato spremeni način prenosa parametrov. Namesto prenosa po referenci uporabi prenos po vrednosti in opazuj razlike v izpisu.

Referenčne spremenljivke

Referenčna spremenljivka je spremenljivka, ki v pomnilniku nima shranjene neke konkretne vrednosti, temveč se navezuje na neko drugo spremenljivko. Njena vrednost je tako vrednost te druge spremenljivke, referenca pa je povezava med njima. Uporaba referenčnih spremenljivk pride v poštev pri prenosu argumentov med metodami.

Deklaracija referenčne spremenljivke

Ker jih uporabljamo pri prenosu argumentov, so običajno deklarirane v glavi metode, pri deklaraciji pa moramo uporabiti operator *ref* ali *out*. Operator *ref* naredi povezavo s spremenljivko, ki že ima neko vrednost (je inicializirana), operator *out* pa naredi povezavo s spremenljivko, ki še nima vrednosti (inicializacija se bo izvršila preko reference). Primer dveh referenčnih spremenljivk si oglejmo na programu.

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

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 a = 11;
        int b;      //spremenljivka b nima začetne vrednosti
        Metoda(ref a, out b);   //klic metode
        Console.WriteLine("spremenljivka b: " + b);
        Console.Read();
    }
    static void Metoda(ref int a1,out int b1)
    {
        b1 = 22;    //tukaj spremenljivka b1 dobi vrednost
        Console.WriteLine("spremenljivka a: " + a1);
    }
```

V *Metodi()* se spremenljivka *a1* naveže na spremenljivko *a*, v pomnilniku obe spremenljivki kažeta na isto vrednost. Spremenljivka *b1* pa se naveže na spremenljivko *b*, ki pa še nima vrednosti, zato referenco ustvarimo z operatorjem *out*. Vrednost *b1* dobi kasneje v metodi, s tem pa dobi vrednost tudi spremenljivka *b*, ker sta povezani. Operatorja *ref* in *out* moramo uporabiti tudi pri klicu metode.

Primerjava vrednostnega in referenčnega prenosa argumentov

Pri klasičnem prenosu (prenos konkretne vrednosti) v metodi ustvarimo nove spremenljivke, ki te vrednosti prevzamejo. Med izvajanjem metode se te vrednosti lahko spreminjajo, spremembe pa na klicno mesto lahko sporočimo le s stavkom *return*, ta pa je lahko le en sam. Pri referenčnem prenosu pa so spremenljivke povezane in vsaka sprememba v metodi se odraža tudi na originalnih spremenljivkah (v metodi, iz katere smo klicali našo metodo).

Program za spremembo negativnih vrednosti v pozitivne

Najprej naredimo prenos argumentov z referenčnimi spremenljivkami.

```
    static void Main(string[] args)
    {
        int a, b, c;
        //tukaj vnesemo vrednosti spremenljivk
        Metoda(ref a, ref b, ref c);   //klic metode
        Console.WriteLine("V glavnem programu:");
        Console.WriteLine("a: {0},b: {1},c: {2}",a,b,c);
        Console.Read();
    }

    static void Spremeni(ref int a, ref int b, ref int c)
    {
        //1. Način - če je št. negativno, ga množimo z -1:
        if (a < 0)
            a = a * (-1);
        //2. Način - poiščemo absolutno vrednost števila:
        b = Math.Abs(b);
        //enako storimo še za spremenljivko c
```

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("V metodi:");
        Console.WriteLine("a: {0},b: {1},c: {2}", a, b, c);
    }
```

V tem primeru bosta oba izpisa enaka, ker so spremenljivke povezane z referenco. Sedaj pa naredimo še prenos argumentov po vrednosti. V zgornjem programu moramo odstraniti vse operatorje *ref*. Program poženemo in ob primerjavi obeh izpisov vidimo, da se spremembe v metodi na spremenljivkah v glavnem programu ne odražajo.

Napiši program, ki prebere tri števila in jih razvrsti po velikosti od največjega do najmanjšega. Program za razvrščanje naj uporabi metodo, ki med seboj zamenja dve števili, če je prvo število manjše od drugega. Delovanje programa preverimo z izpisom števil v glavni metodi.

Metoda za zamenjavo dveh števil

Metoda dobi kot argument dve števili in če je prvo število manjše od drugega, ju zamenja (spremenljivka *x* dobi vrednost spremenljivke *y* in obratno). Pri postopku zamenjave moramo uporabiti dodatno spremenljivko, kamor začasno shranimo vrednost ene od spremenljivk, ker bi jo sicer izgubili.

```
static void Zamenjaj(ref int x,ref int y)
{
    if (x < y)
    {
        int temp = x;  //začasno shranimo vrednost x
        x = y;
        y = temp;
    }
}
```

Ob zamenjavi dobita spremenljivki *x* in *y* novi vrednosti, ki ju moramo vrniti nazaj na klicno mesto. S stavkom *return* dveh vrednosti ne moremo vrniti, zato uporabimo referenčni prenos argumentov in spremenljivke povežemo.

Postopek razvrščanja

Postopek, ki nas vedno pripelje do rešitve, je tak, da najprej zamenjamo prvi dve števili, nato drugi dve in potem spet prvi dve. Zamenjavo opravimo le, če je prvo število manjše od drugega. Primer – vzemimo števila 2, 8 in 6:

a	b	c	
2	8	6	
---	---	---	
8	2	6	1. menjava – a in b zamenjamo, ker je 2 manjše od 8, c pustimo

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.

8	6	2	2. menjava – b in c zamenjamo, ker je 2 manjše od 6, a pustimo
8	6	2	3. menjava – a in b ne zamenjamo, c pa pustimo

Števila so razvrščena od največjega do najmanjšega. Postopek moramo sedaj zapisati v obliki programa. Uporabimo metodo za zamenjavo dveh števil, ki jo kličemo trikrat. Najprej metodi pošljemo prvi dve števili, ta pa postavi večje število pred manjše. Ob naslednjem klicu pošljemo drugo in tretje število, nato pa ponovno pošljemo prvi dve števili.

```
int a = 2, b = 8, c = 6;
Zamenjaj(ref a, ref b);  //1. klic
Zamenjaj(ref b, ref c);  //2. klic
Zamenjaj(ref a, ref b);  //3. Klic
```

Zgornji program nato popravimo, da bo uporabnik lahko sam vnesel 3 izbrane vrednosti in dodamo izpis rezultata razvrščanja.

68. Kaj je referenčna spremenljivka?
69. Kdaj je smotrno kot argumente metode uporabiti referenčne spremenljivke?
70. V katerih primerih uporabimo operator *ref* in v katerih operator *out*?
71. Kaj pri formatirani obliki izpisa vnesemo v zaviti oklepaj? Kje so navedene vrednosti, ki v besedilu nadomestijo zavite oklepaje?
72. Zakaj pri zamenjavi vrednosti dveh spremenljivk potrebujemo dodatno spremenljivko?

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 IN INICIALIZACIJA TABELE – tabelarične spremenljivke

Večje količine podatkov si lažje predstavljamo, kadar so urejeni v tabelah. Poleg tega tabele omogočajo tudi avtomatično obdelavo večje količine podatkov, saj se s programom preprosto premikamo po tabeli, iz nje jemljemo podatke, jih obdelamo in vrnemo na njihovo mesto v tabeli.

Namen vaje je spoznati, kako v jeziku C# ustvarimo tabelo ter načine za njeno inicializacijo in urejanje. Ogledali si bomo:

- deklaracijo tabele
- uporabo zanke *for* in *foreach* pri vnosu podatkov v tabelo
- način inicializacije ob deklaraciji
- nekatere postopke za urejanje znakovnih tabel
- pošiljanje tabel kot argumentov drugim metodam

Napiši program, v katerem ustvariš tri tabele. Prva naj bo tabela znakov, druga naj bo tabela decimalnih števil, tretja pa tabela znakovnih nizov.

1. **Tabele napolnimo takoj ob deklaraciji, v prvo vnesemo 8 decimalnih števil, v drugo 5 znakov, v tretjo pa 4 besede. Nato izpišemo njihovo vsebino v konzolno okno.**

Deklaracija in inicializacija tabele

Tabela je spremenljivka, v katero lahko shranimo večje število vrednosti istega tipa. Število vrednosti v tabeli predstavlja velikost tabele. Kadar tabelo takoj ob deklaraciji tudi inicializiramo (napolnimo), se velikost tabele prilagodi številu vrednosti, ki jih vnesemo v tabelo.
Oglejmo si primere deklaracije in inicializacije tabel različnih podatkovnih tipov.

```
//tabela decimalnih števil:
double[] dec = {1.22,2.33,3.44,4.55,5.66,6.77,7.88,8.99};
//tabela znakov:
char[] znaki = {'a','#','2',' ','A'};
//tabela znakovnih nizov:
string[] nizi = {"lonec", "2011", "a+b=c", "ABC"};
```

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.

Kadar deklariramo tabelarično spremenljivko, ob vrsti podatkovnega tipa vedno napišemo oglati oklepaj. Vrednosti, ki jih naštejemo v zavitih oklepajih, se shranijo pod imenom tabele v takšnem vrstnem redu, kot smo jih navedli. Podobno kot v znakovnem nizu tudi tukaj vsaka vrednost dobi svoj indeks, s pomočjo katerega jo lahko v tabeli najdemo in ga prav tako navajamo v oglatih oklepajih.

Velikost tabele

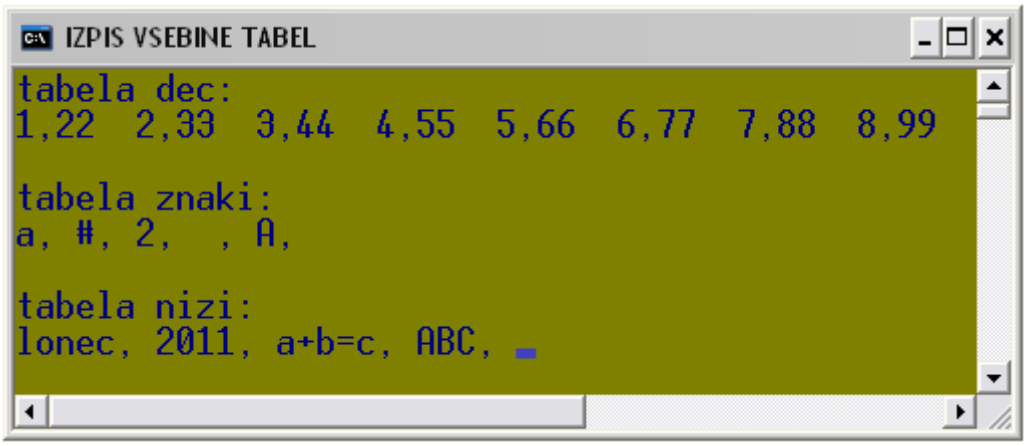
Tudi velikost tabele ugotavljamo na enak način kot pri znakovnem nizu s pomočjo lastnosti *Length*. Poglejmo si velikost tabele *dec* v konzolnem oknu.

```
Console.WriteLine("tabela dec: " + dec.Length);
```

Izpiše se število 8, ker je v njej osem vrednosti.

Izpis vsebine tabele v konzolno okno

Vsebino tabel bi radi prikazali v konzolnem oknu.



Slika 28: prikaz vsebine tabel

Potrebno bo izpisati vsak element tabele (vrednost) posebej in ker je teh običajno veliko, si bomo pomagali z zanko. Najprimernejši sta zanki *for* in *foreach*.

```
//izpis z zanko for:
for (int i = 0; i < dec.Length; i++)
     Console.Write(dec[i] + "  ");
for (int i = 0; i < znaki.Length; i++)
          Console.Write(znaki[i] + ", ");
for (int i = 0; i < nizi.Length; i++)
          Console.Write(nizi[i] + ", ");
```

Števec iz zanke uporabimo kot indeks za določitev elementa, ki ga izpisujemo. Ker želimo izpisati vse elemente tabele, začnemo z vrednostjo 0 in gremo do največjega indeksa, ki je za 1 manjši od velikosti tabele.

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.

```
//izpis z zanko foreach:
foreach (double d in dec)
     Console.Write(d + "  ");
```

Pri tej zanki ne uporabljamo indeksa, ker do vrednosti v tabeli dostopamo preko posebne spremenljivke, ki jo ustvarimo v glavi zanke. Ta spremenljivka mora biti istega tipa kot vrednosti v tabeli. V našem primeru je to spremenljivka z imenom *d*, ki je tipa *double*, ker je tabela dec, katere vrednosti izpisujemo, tipa *double*.

2. **Spremenimo vsebino tabel. V prvo vnesimo števila od 1 do 15, v drugo vnesimo nek znakovni niz, v tretjo pa prenesimo nek stavek, vsaka beseda v stavku pa naj predstavlja en element tabele. Stavek naj uporabnik vnese s tipkovnice.**

Ker tabele že imamo, deklaracije ne bodo potrebne, spremeniti pa bo potrebno velikost tabel.

Tabela števil

V tabeli *dec*, ki jo bomo uporabili za števila, imamo 8 elementov, potrebujemo pa tabelo za 15 števil. Zato nastavimo velikost tabele na 15.

```
dec = new double[15];    //v oklepaju je velikost tabele
```

Zgornji ukaz nam v tabelo z imenom *dec* vnese 15 ničel (tabela ni prazna!), ki se smatrajo kot decimalna števila, ker imamo tip *double*. Sedaj moramo namesto ničel vnesti želene vrednosti, torej števila od 1 do 15. Najlaže to storimo z zanko *for*, kjer vrednost števca ob vsaki izvedbi zanke vnesemo v tabelo. Pri vnosu vrednosti moramo biti pozorni, da ne prekoračimo velikosti tabele, zato morajo biti vrednosti števca zanke enake indeksom v tabeli (od 0 do velikosti tabele), vrednosti, ki jih vnašamo, pa ustrezno preračunamo.

```
for (int i = 0; i < dec.Length; i++)
     dec[i] = i + 1;     //števec moramo povečati za 1
```

Tabela znakov

Znake bomo vnesli v obliki znakovnega niza in si pri tem pomagali z metodo *ToCharArrey()*.

```
znaki = "abcdefghijklmn".ToCharArray();
```

Metoda *ToCharArrey()* poskrbi tudi za prilagoditev velikosti in nam za to ni potrebno skrbeti. Niz znakov lahko vnesemo tudi preko tipkovnice.

```
znaki = Console.ReadLine().ToCharArray();
```

Tabela znakovnih nizov

V razredu *String* imamo posebno metodo, ki znakovni niz razdeli na več delov glede na določen znak. V našem primeru bomo stavek razdelili glede na presledek.

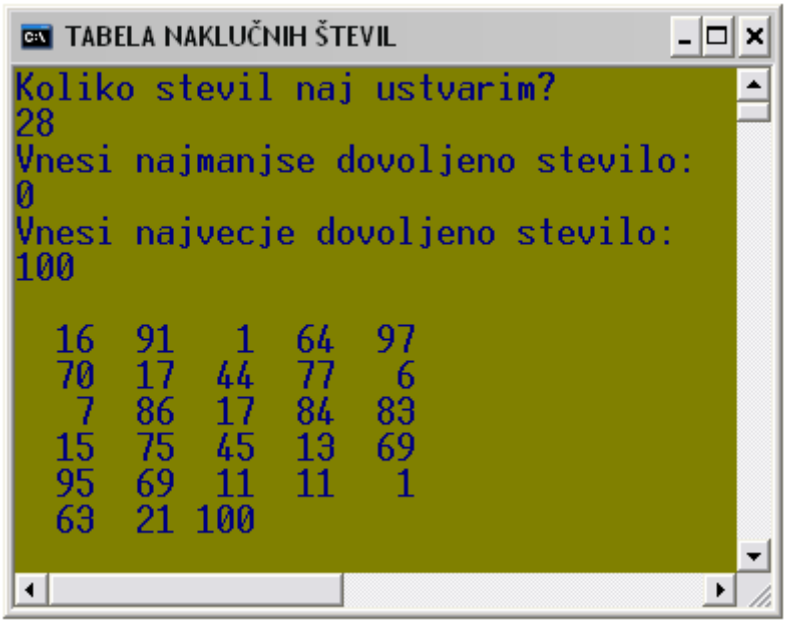
```
string stavek = Console.ReadLine(); //preberemo stavek
```

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.

```
nizi = stavek.Split(' ');  //argument metode je presledek
```

Vneseni stavek, ki ga metoda *Split()* razdeli na besede, prenesemo v tabelo *nizi*.
Izpis v konzolno okno naredimo enako kot v prvem delu naloge s pomočjo zanke.

Napiši program, ki napolni tabelo z naključnimi celimi števili. Uporabnik naj ima možnost izbire, koliko bo teh števil in kako velika naj bodo (lahko določi spodnjo in zgornjo mejo). Program naj v konzolnem oknu prikaže vsebino tabele, v izpisu naj bo v vsaki vrstici po pet števil, med števili pa naj bodo presledki.



Slika 29: izpis vsebine tabele

Priprava spremenljivk za velikost tabele, spodnjo in zgornjo mejo

Kadar ustvarimo tabelo, moramo vedno izvesti tudi inicializacijo. V našem primeru bomo v tabelo vnesli neke naključne vrednosti, katerih števila vnaprej ne poznamo, saj ga določi šele uporabnik. Zato bomo v tabelo vpisali ničle, pred tem pa moramo s tipkovnice prebrati velikost tabele.

```
Console.WriteLine("Koliko števil naj ustvarim?");
int stevilo = Convert.ToInt32(Console.ReadLine());
int[] tab = new int[stevilo];
```

Spremenljivka *stevilo* predstavlja velikost tabele in mora biti tipa *int*. Enako preberemo še spremenljivki, ki omejita velikost naključnih števil.

```
Console.WriteLine("Vnesi najmanjše dovoljeno število: ");
```

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 sp_meja = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Vnesi največje dovoljeno število: ");
int zg_meja = Convert.ToInt32(Console.ReadLine());
```

Kasneje ju bomo uporabili kot argumenta metode za ustvarjanje naključnih vrednosti.

Vnos podatkov v tabelo

Ker se mora program prilagajati uporabniku, bomo za vnos podatkov uporabili zanko, ki se bo ponovila tolikokrat, kot je velikost tabele. V zanki ustvarimo naključno vrednost in jo vnesemo v tabelo.

```
Random r = new Random();
for (int i = 0; i < tab.Length; i++)
     tab[i] = r.Next(sp_meja, zg_meja + 1);
```

Pri zgornji meji moramo upoštevati, da jo metoda *Next()* ne vključuje, zato ji prištejemo enko.

Izpis vsebine tabele

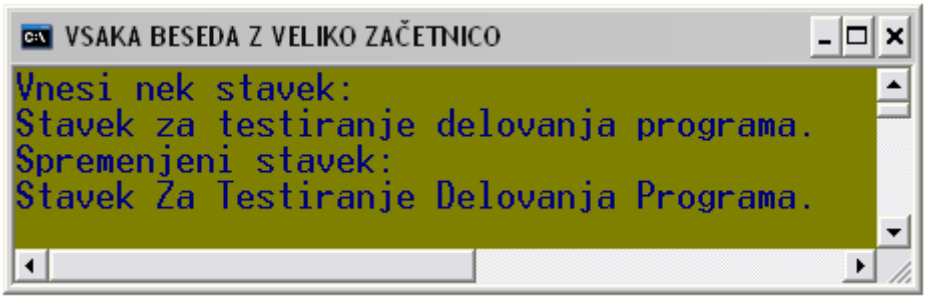
Enako zanko kot pri vnosu vrednosti, bomo uporabili tudi pri izpisu. Določiti moramo še, da je v vrstici le pet števil, zato dodamo še pogojni stavek, ki nas po petih, desetih, petnajstih …, številih vrže v novo vrsto. Vsa omenjena števila so deljiva s 5, kar uporabimo pri zapisu pogoja.

```
for (int i = 0; i < tab.Length; i++)
{
     if (i % 5 == 0)     //če je ostanek po deljenju s 5
                         //enak 0…
          Console.WriteLine();     //…skoči v novo vrsto
     Console.Write("{0,4}",tab[i]);   //izpis elementa
}
```

Pri izpisu določimo 4 mesta za širino izpisa enega elementa tabele.

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.

Program naj s tipkovnice prebere nek stavek in ga vnese v znakovno tabelo. Nato naj pokliče metodo, ki bo prvo črko v vsaki besedi spremenila iz male v veliko. Spremenjeni stavek naj na koncu glavne metode izpiše v konzolno okno.



Slika 30: primer izpisa programa

Vnos stavka in izpis

Metoda *ReadLine()* prebere stavek s tipkovnice, metoda *ToCharArray()* pa ga pretvori v obliko, ki je primerna za znakovno tabelo. Pri izpisu znakovne tabele ni potrebno v zanki izpisovati vsak znak posebej, podobno kot pri znakovnem nizu lahko izpišemo kar celotno tabelo.

```
Console.WriteLine("Vnesi nek stavek: ");
char[] stavek = Console.ReadLine().ToCharArray();   //vnos
//na tem mestu bomo poklicali metodo
Console.WriteLine("Spremenjeni stavek: ");
Console.WriteLine(stavek);  //izpis celotne tabele
```

Metoda za pretvorbo malih črk v velike

Metodo poimenujmo *VelikeCrke()*. Kot argument ji bomo poslali tabelo, ki je po naravi referenčna spremenljivka, zato je pri pošiljanju argumentov avtomatično izbran referenčni način. Operatorja *ref* ni potrebno pisati.

```
//klic metode v glavni metodi:
VelikeCrke(stavek);

//metoda:
static void VelikeCrke(char[] s)
{
     //vsebina metode
}
```

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.

V glavi metode deklariramo tabelo z imenom *s*, ki se poveže s tabelo *stavek*, ki jo pošljemo kot argument ob klicu. Karkoli bomo s tabelo *s* storili v metodi, se bo zaradi reference odražalo tudi na tabeli *stavek*, čeprav je ta deklarirana v glavni metodi. Metodi zato ni potrebno vračati nobenih vrednosti in bo pripadala tipu *void*.

```
//vsebina metode:
for (int i = 0; i < s.Length; i++)
     if (s[i] == ' ')    //če je tekoči znak presledek…
          s[i + 1] = Char.ToUpper(s[i + 1]);
```

Vsebina metode predstavlja postopek za iskanje prve črke v besedi in njeno pretvorbo v veliko črko. Prva črka se nahaja takoj za presledkom, če ima presledek indeks *i*, ima prva črka indeks *i+1*.
Program ne deluje za prvo črko v stavku, ker ta ne stoji za presledkom. Da to pomanjkljivost odpravimo, moramo napisati v metodi še stavek, ki v veliko črko spremeni znak z indeksom 0.

Napiši program za preverjanje veljavnosti gesla. Pri vnosu posameznega znaka naj uporabnik na zaslonu vidi zvezdico. Geslo naj bo dolgo 8 znakov, program naj uporabniku v konzolno okno sporoči, če je geslo veljavno.

Vnos znakov

V znakovno tabelo znakov ne moremo dodajati s pomočjo operatorja + kot pri znakovnih nizih. Znak vnesemo tako, da vsak element tabele prevzame vrednost, ki jo vrne metoda *ReadKey()*. Postopek vnosa ponavljamo v zanki, število ponovitev zanke naj se ne ujema s številom znakov v geslu, da uporabnik ne more ugotoviti pravega števila.

```
//skrito geslo:
char[] tab1 = "geslo123".ToCharArray();
//tbela za geslo, ki ga bo vnesel uporabnik:
char[] tab2 = new char[12];
for (int i = 0; i < tab2.Length; i++)
     tab2[i] = Console.ReadKey().KeyChar;
```

Ob vnosu ne pozabimo na nevidnost znakov, izpis zvezdice in prekinitev zanke ob pritisku na <ENTER>.

Primerjava dveh znakovnih tabel

Ko je vnos gesla končan, program preveri, če je vsebina obeh tabel enaka. Najlaže to storimo tako, da vsebino obeh tabel pretvorimo v znakovni niz in preverimo, če sta niza enaka.

```
if (tab1.ToString() == tab2.ToString())
```

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.

Napiši program, ki prebere 5 stavkov in ugotovi, kateri stavek je najdaljši. Postopek iskanja najdaljšega stavka napiši v metodi, ki ta stavek tudi izpiše.

Tabela stavkov

Stavke v zanki vnesemo v tabelo stavkov, vsak stavek dobi svoj indeks.

```
string[] stavki = new string[5];
for(/*se izvede 5-krat*/)
     stavki[i] = Console.ReadLine();
```

Metoda za iskanje najdaljšega stavka

V metodi najprej ustvarimo spremenljivki *najd* in *index*, prva si zapomni največjo dolžino (njena začetna vrednost naj bo 0), druga pa indeks stavka, ki je do tega trenutka najdaljši. Z zanko *for* gremo skozi tabelo in ugotovimo dolžine posameznih stavkov. Sproti preverjamo, če je trenutni stavek daljši od do tega trenutka najdaljšega ter si zapomnimo njegov indeks, spremenljivka *najd* pa vedno prevzame vrednost dolžine trenutno najdaljšega stavka.

```
if(stavki[i].Length > najd)
{
     najd = stavki[i].Length;
     index = i;      //zapomnimo si indeks
}
```

Ker imamo indeks najdaljšega stavka, ga na koncu lahko izpišemo.

73. Ali so v tabeli lahko shranjeni elementi različnih podatkovnih tipov?
74. Napiši primer deklaracije tabele dvajsetih znakov!
75. Kaj ugotavljamo z lastnostjo *Length* in kako jo uporabimo?
76. Ali lahko vsebino tabele izpišemo na zaslon brez uporabe zanke? Katera zanka je najprimernejša za izpis celotne vsebine?
77. Kako v tabeli najdemo določen podatek?
78. Za katero opravilo se uporabljata metodi *ToCharArray()* in *ToString()*?

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.

DVODIMENZIONALNE TABELE – tabele z več vrsticami

Tabele imajo lahko tudi več dimenzij. Dvodimenzionalna tabela je urejena tako, da imamo več vrstic z enakim številom elementov, kar pri velikih tabelah omogoča večjo preglednost in hitrejši dostop do določenega podatka v tabeli. Pri prikazu vsebine tabele bomo program napisali tako, da bodo podatki izpisani v obliki dvodimenzionalne tabele.

Namen vaje je spoznati princip urejenosti 2-D in s tem tudi večdimenzionalnih tabel. Pri tem si bomo ogledali:

- inicializacijo 2-D tabele
- vnos in izpis elementov s pomočjo gnezdene zanke for
- dostop do posameznega stolpca in vrstice.

Program ustvari 2-D znakovno tabelo velikosti 5 x 8 in jo napolni z naključno izbranimi velikimi črkami angleške abecede. V nadaljevanju program izpiše vsebino tabele, nato pa izpiše še drugo vrstico in četrti stolpec tabele.

Slika 31: izpis programa

Deklaracija dvodimenzionalne tabele

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.

Dvodimenzionalna (2-D) tabela pomeni, da so podatki v tej tabeli razvrščeni v več enako dolgih vrstic, kar je zelo uporabno pri velikih tabelah. Do nekega podatka v tabeli pridemo tako, da najprej poiščemo vrstico, v kateri se nahaja, nato pa podatek poiščemo še znotraj vrstice. Takšne tabele so bolj pregledne, dostop do, podatka pa je hitrejši.
Ker moramo najprej navesti vrstico podatka, nato pa še njegovo mesto v vrstici, mora tak podatek imeti dva indeksa. Vsak indeks predstavlja eno dimenzijo, zato je to 2-D tabela. Deklariramo jo na podoben način kot navadno enodimenzionalno tabelo, le v oglati oklepaj dodamo vejico, ko podajamo velikost, pa navedemo obe dimenziji.

```
char[,] crke = new char[5,8];
```

Jezik C# pozna tudi večdimenzionalne tabele, ki jih ustvarimo na enak način, razlika je le v številu vejic v oglatem oklepaju (tridimenzionalna ima dve vejici).

Vnos znakov

Za vnos elementov v 2-D tabelo je najprimernejša uporaba dvojne (gnezdene) zanke *for*. Notranja zanka napolni vrstico, zunanja pa nas postavi v novo vrstico. Števec v zunanji zanki gre do velikosti 5, ker imamo 5 vrstic, števec v notranji zanki pa gre do velikosti 8, ker je v vrstici 8 elementov.

```
for (int v = 0; v < 5; v++)
     for (int s = 0; s < 8; s++)
          crke[v,s] = (char)r.Next(65,65+26);
```

Ne pozabimo, da je *r* objekt razreda *Random*, ki omogoča dostop do metode *Next()*, ta pa ustvari naključno celo število v določenih mejah, navedenih v oklepaju. Do črk pridemo preko njihove kode v ASCII tabeli, veliki 'A' ima kodo 65, kar predstavlja spodnjo mejo pri izbiri naključnih števil. Ker je črk v abecedi 26, bo zgornja meja za 26 večja od spodnje.

Izpis vsebine

Tudi pri izpisu moramo uporabiti dvojno zanko *for*, če želimo dobiti obliko izpisa v dveh dimenzijah. Notranja zanka nam izpiše eno vrstico tabele, ko je izpis vrstice končan, pa skočimo v novo vrsto.

```
//ponovimo tolikokrat, kot je vrstic:
for (int v = 0; v < 5; v++)
{
     //izpis vrstice:
     for (int s = 0; s < 8; s++)
          Console.Write(crke[v, s] + " ");
     //skok v novo vrsto:
     Console.WriteLine();
}
```

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.

Izpis vrstice

Ker smo določili le eno vrstico, se indeks *v* ne bo spreminjal. Pri drugi vrstici bo njegova vrednost ve čas enaka 1, zato na njegovo mesto v oglati oklepaj vpišemo enko. Potrebovali bomo zanko, ki bo izpisala vrstico, v kateri je 8 elementov.

```
for (int s = 0; s < 8; s++)
     Console.Write(crke[1, s] + " ");
```

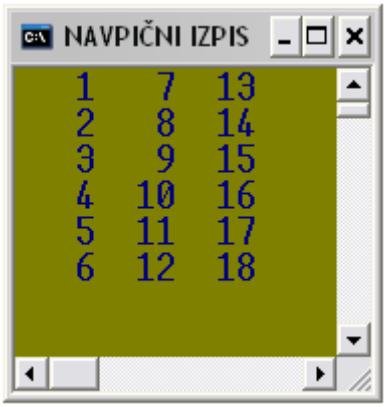
Izpis stolpca:

Pri izpisu stolpca je podobno, le da je v tem primeru stalen indeks s, njegova vrednost bo 3. Potrebovali bomo zanko, ki bo izpisala stolpec s petimi elementi.

```
for (int v = 0; v < 5; v++)
     Console.Write(crke[v, 3] + " ");
```

Deklariraj tabelo velikosti 3 x 6 in jo ob deklaraciji napolni s celimi števili od 1 do 18. Nato vsebino tabele izpiši tako, da bodo vrstice izpisane navpično. Vsebina tabele:

```
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
```



Slika 32: prikaz izpisa programa

Deklaracija tabele

Ustvarimo 2-D tabelo celih števil in jo podobno kot v prejšnji nalogi napolnimo s pomočjo gnezdene *for* zanke. Števila od 1 do 18 dobimo s pomočjo dodatne spremenljivke *x*, ki jo

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.

ustvarimo pred zanko, njena začetna vrednost bo 1, v zanki pa se njena vrednost vedno poveča za 1.

```
int x = 1;
int[,] crke = new int[3,6];
for (int v = 0; v < 3; v++)
     for (int s = 0; s < 6; s++)
          crke[v,s] = x++;
```

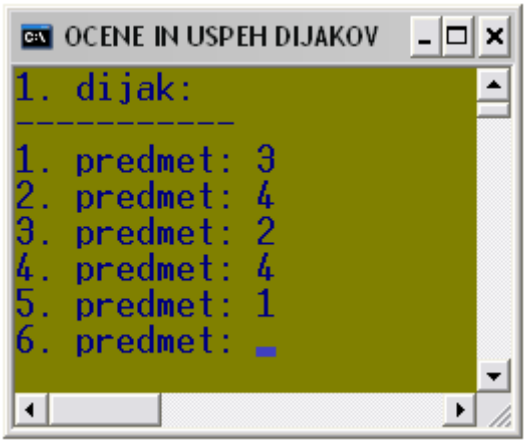
V zadnjem stavku se vrednost spremenljivke x najprej vpiše v tabelo, nato pa se njena vrednost poveča za 1.

Izpis tabele v konzolno okno

V izpisu bomo imeli 6 vrstic s po tremi elementi, zato bo števec zunanje zanke tekel do 6, notranje pa do 3. Potrebna bo še zamenjava imen števcev v zanki ali indeksov v oglatem oklepaju. Po izpisu vrstice vedno skočimo v novo vrsto, za izpis elementa smo v konzolnem oknu rezervirali 4 mesta.

```
for (int s = 0; s < 6; s++)
{
     for (int v = 0; v < 3; v++)
          Console.Write("{0,4}",crke[v,s]);
     Console.WriteLine();
}
```

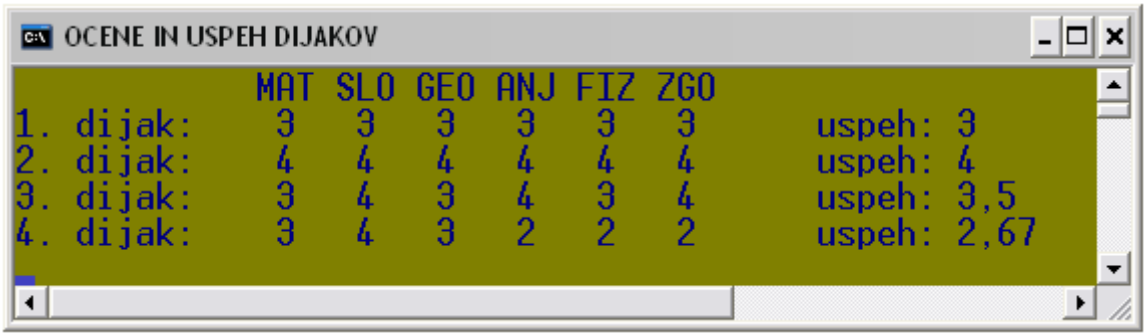
Napiši program, ki izračuna povprečno vrednost ocen pri šestih predmetih (uspeh) za štiri dijake. Podatke uporabnik vnese v tabelo preko tipkovnice, rezultate izračuna pa shrani v tabeli z imenom *uspeh*. Na koncu program izpiše vse ocene in povprečje ocen za vsakega od štirih dijakov.



Slika 33: prikaz vnosa podatkov za prvega dijaka

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.

Ko vnesemo ocene vseh dijakov, se vsebina okna pobriše, pojavijo pa se ocene dijakov in njihov uspeh.



Slika 34: izpis ocen in uspeha dijakov

Oblikovanje tabele

Vzemimo, da stolpci v tabeli predstavljajo predmete, vrstice pa dijake. V tem primeru imamo 4 vrstice in 6 stolpcev, čemur ustreza tabela velikosti 4 x 6 tipa *int*, ker so ocene cela števila. Tabeli damo ime *ocene* in jo napolnimo z metodo *ReadLine()* v gnezdeni zanki *for*.

```
for (int d = 0; d < 4; d++)          //štirke dijaki
{
     Console.Clear();
     Console.WriteLine("{0}. dijak:\n----------", d + 1);
     for (int p = 0; p < 6; p++)     //šest predmetov
     {
          Console.Write("{0}. predmet: ", p + 1);
          ocene[d, p]=Convert.ToInt32(Console.ReadLine());
     }
}
```

Vnos podatkov opremimo z navodili uporabvniku (če bi uporabili še naštevni podatkovni tip, bi lahko namesto številke predmeta in dijaka uporabili imena dijakov in predmetov).

Izračun uspeha

Za izračun napišemo metodo, v kateri najprej izračunamo vsoto ocen za vsakega dijaka in to shranimo v tabeli *uspeh*. Tabelo *uspeh* deklariramo v glavni metodi in jo metodi za izračun pošljemo kot argument. Njeno velikost določa število dijakov. Ker bo njena vsebina povprečje ocen, ki ga dobimo z deljenjem, mora pripadati tipu *double*.

```
for (int d = 0; d < 4; d++)
     for (int p = 0; p < 6; p++)
          uspeh[d] = uspeh[d] + ocene[d, p];
```

Nato izračunamo še povprečno vrednost tako, da vse elemente tabele *uspeh* delimo s 6.

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.

```
for (int d = 0; d < 4; d++)
     uspeh[d] = uspeh[d] / 6;
```

Obe zgornji zanki s pomočjo orodja *Refactor* oblikujemo v metodo z imenom *Uspeh*.

Izpis ocen in uspeha

Tudi tega oblikujemo kot novo metodo, ki jo pokličemo iz glavne metode in ji kot argument pošljemo obe tabeli.

```
Console.WriteLine("\t    MAT SLO GEO ANJ FIZ ZGO");
for (int d = 0; d < 4; d++)
{
     Console.Write("{0}. dijak: ", d + 1);
     for (int p = 0; p < 6; p++)
     {
          Console.Write("{0,4}", ocene[d, p]);
     }
     Console.WriteLine("\tuspeh:"+Math.Round(uspeh[d],2));
}
```

V izpisu naštejemo imena predmetov, poskrbimo za obliko, uspeh pa zaokrožimo na dve decimalki.

Preprečevanje napačnega vnosa

Če želimo preprečiti, da bi uporabnik vnesel oceno, ki ni med 1 in 5, dodamo filter vnosa. Tega izdelamo tako, da vnos neke ocene ponavljamo tako dolgo, da je vnesena ocena ustrezna. Uporabimo neskončno zanko *while*, ki jo prekinemo, ko je vnesena ocena med 1 in 5.

```
while (true)        //neskončna zanka
{
     Console.Write("{0}. predmet: ", p + 1);
     ocene[d, p] = Convert.ToInt32(Console.ReadLine());
     if (ocene[d, p] < 5 && ocene[d, p] > 0)
          break;    //prekinemo zanko
     else Console.Write("  napaka!\n");
}
```

Z zgornjo kodo nadomestimo vrstico za vnos ocene v glavni metodi.

79. Kako pridemo do elementa v dvodimenzionalni tabeli?
80. Zakaj je pri delu z dvodimenzionalnimi tabelami primerna gnezdena zanka?
81. Kaj pomeni, če na mestu, ki je določeno za pogoj, v zanki *while* pišemo besedo *true*?

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.

82. Kateremu tipu mora pripadati spremenljivka, če nastopa kot indeks?
83. Kaj se zgodi, če med izvajanjem programa prekoračimo velikost tabele?

PRILOGA
– pregled podatkovnih tipov v jeziku C#

S podatkovnim tipom ob deklaraciji spremenljivke določimo, kakšne vrste podatek bo predstavljala ta spremenljivka in koliko prostora bo v pomnilniku zasedla. Glede na vrsto podatke delimo v tri osnovne skupine, dve od teh skupin pa se nadalje delita še v dve podskupini:

1. **Števila**
   - cela števila
   - decimalna (realna) števila
2. **Znaki**
   - en sam znak
   - znakovni niz
3. **Logična vrednost.**

CELA ŠTEVILA

Spremenljivke, ki predstavljajo cela števila, ne dovolijo uporabe decimalk. V primeru, ko je rezultat neke računske operacije decimalno število (npr. deljenje), se decimalna mesta enostavno odrežejo in ostane samo celi del števila. Imamo več celoštevilskih podatkovnih tipov, ki se razlikujejo glede na količino pomnilnika, ki ga zasedejo, s tem pa imajo tudi različne zaloge vrednosti, (vrednosti, ki jih lahko spremenljivke določenega tipa zasedejo). Če vrednost spremenljivke pade izven zaloge vrednosti, govorimo o prekoračitvi, ki povzroči napako v delovanju programa. Velikost pomnilnika merimo v byte-ih, 1 byte ima 8 bitov in zasede 1 pomnilniško lokacijo.

- **int** (integer)
  Je osnovni podatkovni tip za cela števila. V pomnilniku zasede 4 lokacije oz. 4 byte (32 bitov). Zaloga vrednosti je v mejah med približno +/-2 000 000 000 (točno vrednost dobimo, če izračunamo 2 na 31 potenco).
- **uint**

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.

Je različica podatkovnega tipa **int** za števila brez predznaka (**u** pomeni unsigned). Zaloga vrednosti se giblje približno v mejah med 0 in 4 000 000 000.

- **byte**
  Je najmanjši celoštevilčni podatkovni tip, zasede 1 pomnilniško lokacijo oz. 8 bitov. Namenjen je samo pozitivnim številom, zaloga vrednosti je v mejah med 0 in 255 (2 na 8 potenco je 256). Uporabimo ga, kadar delamo z zelo majhnimi števili. Ker takšni podatki zasedejo manj pomnilniškega prostora, bo delovanje programa hitrejše.
- **sbyte**
  Je različica tipa byte, ki velja tudi za negativna števila, zaloga vrednosti je v mejah med -128 in +127.
- **short** (short integer)
  Je zmanjšana različica tipa int, ki v pomnilniku zasede 16 bitov, zaloga vrednosti se giblje približno v mejah +/-32 000. S predpono **u** (**ushort**) dobimo podatkovni tip samo za pozitivna števila.
- **long** (long integer)
  Je povečana različica tipa **int**, v pomnilniku zasede 64 bitov in je uporaben za zelo velika cela števila.

DECIMALNA ŠTEVILA

V to skupino spadajo vsa realna števila. Pri decimalnih številih nas ne zanima zaloga vrednosti, temveč natančnost, ki jo določa število decimalnih mest. Ta pa za nek podatkovni tip ni fiksno določena. Število delimo na celi del in decimalni del, če je celi del velik, je zato manj mest na voljo za decimalni del. Govorimo o principu plavajoče vejice (floatig point), ker se decimalna vejica premika po številu glede na velikost celega dela.

- **float**
  Nekoč osnovni decimalni podatkovni tip, z napredkom tehnologije so se zahteve in zmožnosti povečale, zato se danes redkeje uporablja. V pomnilniku zasede 4 lokacije, njegova natančnost je do 7 decimalk.
- **double**
  Je največkrat uporabljeni decimalni podatkovni tip, v pomnilniku zasede 8 lokacij, njegova natančnost je do 15 decimalk.
- **decimal**
  Namenjen je delu z zelo velikimi števili ali pa je zahtevana zelo velika natančnost, v pomnilniku pa zasede 16 lokacij (128 bitov).

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.

ZNAKOVNI PODATKOVNI TIP

Znaki so podatki, ki so kodirani na osnovi ASCII (oz. danes Unicode) standarda. V skupino znakov spadajo črke, številke, simboli, prav tako pa imajo svoje mesto v ASCII tabeli tudi *presledek, enter, backspace, zvonček …*

- **char**
  Spremenljivka tipa **char** omogoča shranjevanje enega samega znaka, njene vrednosti pa vedno navajamo v enojnih narekovajih. Znakovni podatkovni tip ni namenjen podatkom, nad katerimi lahko izvajamo računske operacije. Edini operaciji, ki jih dovoli, sta inkrement in dekrement, ki nas v ASCII tabeli premakneta za mesto naprej ali nazaj. Pri pretvorbi znakovne vrednosti v celo število dobimo ASCII kodo znaka, ki ustreza tej vrednosti.

ZNAKOVNI NIZ

Znakovni niz je sestavljen iz zaporedja znakov. Število znakov v nizu ni omejeno. Večinoma se uporablja za delo s tekstovnimi podatki, nepogrešljiv pa je tudi pri vnosu podatkov v računalnik, saj se ti običajno vnašajo v tekstovni obliki.

- **string**
  Sestavljajo ga podatki, ki pripadajo tipu **char**, vendar ti podatki v pomnilniku nastopajo kot ena sama spremenljivka. Omogoča nam dostop do posameznega znaka v nizu in s tem tudi urejanje niza.

LOGIČNI PODATKOVNI TIP

- **bool** (boolean)
  Namenjen je spremenljivkam, ki imajo lahko le dve vrednosti in ponazarjata dve logični stanji (1 in 0 oz. DA in NE). Ti dve vrednosti sta *true* in *false*, ki predstavljata zalogo vrednosti tega podatkovnega tipa.

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 SKLOPOV IN UČNIH SITUACIJ

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.

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.

MEDPREDMETNE POVEZAVE

Gradivo ponuja kar nekaj možnosti za različne medpredmetne povezave.

Povezave z matematiko ali fiziko
Nekatere učne situacije temeljijo na reševanju določenega matematičnega problema (npr. računanje ploščin, stranic in kotov v geometrijskih likih). Medpredmetno povezavo bi izvedli tako, da profesor, ki poučuje UPN skupaj s profesorjem matematike izbere temo v okviru snovi, ki jo pri matematiki obravnavajo v drugem letniku. Skupaj določita obseg naloge (katere izračune naj opravi računalnik in v kakšni obliki naj poda rezultate) in postavita učno situacijo. Profesor matematike nato dijakom razloži snov in skupaj z dijaki pripravi potrebne izraze.
Nato nastopi še delo pri predmetu UPN, kjer dijaki skupaj s profesorjem postavijo algoritem za rešitev problema, na osnovi algoritma pa nato napišejo še programsko kodo in ustvarijo izvršilno datoteko.
Pri matematiki nato preskusijo delovanje programa in ovrednotijo dobljene rezultate.
Na podoben način bi se lahko povezali tudi s fiziko in namesto matematičnega reševali nek fizikalni problem.

Povezovanje angleščino
Programski jezik vsebuje veliko število besed, ki izhajajo iz angleškega jezika. Medpredmetno povezavo bi izvedli tako, da bi profesor UPN pripravil nove besede, ki jih dijaki srečajo v nekem poglavju, profesor angleškega jezika pa bi natančno razložil njihov pomen in načine uporabe. Na ta način bi dijakom (predvsem tistim s slabšim znanjem angleščine) olajšali delo pri prevajanju algoritma v programski jezik.

Povezovanje s slovenščino
Pri povezovanju s slovenskim jezikom imam v mislih dve možnosti. Prva se ponuja kar sama in predstavlja izdelavo delovnega poročila o izvedbi neke naloge, v našem primeru bi bila to neka učna situacija. Profesor slovenščine

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.

dijakom razloži, kako se napiše poročilo, katere prvine naj vsebuje, kakšna naj bo vsebina posameznih elementov poročila… Pri predmetu UPN nato dijaki izberejo eno od učnih situacij, ki jo nato v šoli izvedemo. Profesor nato dijake pripravi na izdelavo poročila (pove, kaj je pomembno, česa ne smejo izpustiti, česa se v poročilo ne piše…), dijaki pa doma izdelajo poročilo. Izdelana poročila nato oba profesorja pregledata in ovrednotita, da dobijo dijaki povratno informacijo o svojem izdelku.
Druga možnost medpredmetne povezave pa bi bila izvedljiva na področju primerjave dveh jezikov, programskega in slovenskega. Dijaki bi lahko pri pouku slovenščine primerjali lastnosti, prvine in pravila, ki jih uporabljamo pri pisanju v enem ali drugem jeziku. Pri tem bi poudarili razloge za nastanek teh razlik (jezik, ki je namenjen komunikaciji med ljudmi, in jezik, ki je namenjen za komuniciranje z računalnikom). Z znanjem, ki bi ga dijaki pridobili na ta način, postane raba obeh jezikov bolj poglobljena in kvalitetnejša.

Zgoraj navedene medpredmetne povezave seveda niso edine možnosti za tovrstno poučevanje. Z malo domišljije bi se dalo povezati tudi z drugimi splošnimi predmeti. Kar se tiče povezave z ostalimi strokovnimi predmeti, pa je ta v višjih letnikih, ko dijaki pridobijo nekoliko več znanja, tako ali tako neizbežna in se izvaja po učnem načrtu, ker se znanja določenih predmetov dopolnjujejo.

VIRI IN LITERATURA

- Uranič, S. Microsoft C#.NET, (online). Kranj: TŠC, 2008. Dostopno na naslovu: http://uranic,tsckr.si/C%23/C%23.pdf
- Lokar, M. Osnove programiranja: programiranje – zakaj ali vsaj kaj. Ljubljana: Zavod za šolstvo, 2005.
- Lokar, M. et.al. Projekt UP – kako poučevati začetni tečaj programskega jezika, sklop interaktivnih gradiv, (online). 2008. Dostopno na naslovu: http://up.fmf.uni-lj.si
- Petric, D. Spoznavanje osnov programskega jezika C#, diplomska naloga. Ljubljana UL FMF, 2008

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.

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.