Programski jezik C#

zanke

Matija Lokar in Srečo Uranič

V 0.91

oktober 2008

Predgovor

Omenjeno gradivo predstavlja naslednji (tretji) del gradiv, namenjenih predmetu Programiranje 1 na višješolskem študiju Informatika.

Pokriva zanke v jeziku C#. Glede samega besedila velja tisto, kar je bilo napisano v predgovorih prejšnjih delov gradiv, torej – ne poveva vsega, določene stvari poenostaviva, veliko je zgledov ... .

Gradivo vsekakor ni dokončano in predstavlja delovno različico. V njem so zagotovo napake (upava, da čimmanj), za katere se vnaprej opravičujeva. Da bo lažje spremljati spremembe, obstaja razdelek Zgodovina sprememb, kamor bova vpisovala spremembe med eno in drugo različico. Tako bo nekomu, ki si je prenesel starejšo različico, lažje ugotoviti, kaj je bilo v novi različici spremenjeno.

Matija Lokar in Srečo Uranič

Kranj, oktober 2008

Zgodovina sprememb

6. 10. 2008: Različica V0.8 – prva, ki je na voljo javno
25. 10. 2008: Različica V0.9 - oblikovne spremembe
31. 10. 2008: Različica V0.91 - zanka do while

KAZALO

Zanke

Denimo, da bi želeli izpisati vsa naravna števila med 1 in 20. Z znanjem, ki ga imamo, to ni nobena težava. Hitro je tu program:

```
static void Main(string[] args)
{
        Console.WriteLine("1 2 3 4 5 6 7 8 9 10 "
         + "11 12 13 14 15 16 17 18 19 20");
}
```

Kaj pa, če bi želeli izpisati števila med 1 in 1000?

Janezek je bil v šoli poreden. Zelo poreden. In zato mu je učiteljica naročila, naj sestavi program, ki 5x izpiše "V šoli se moram lepo obnašati!" Janezek hitro vzame "okostje“ programa in ga dopolni s

```
Console.WriteLine("V šoli se moram lepo obnašati!");
Console.WriteLine("V šoli se moram lepo obnašati!");
Console.WriteLine("V šoli se moram lepo obnašati!");
Console.WriteLine("V šoli se moram lepo obnašati!");
Console.WriteLine("V šoli se moram lepo obnašati!");
```

Ker pa se Janezek še vedno ne obnaša lepo, sledi nova kazen. Ker učiteljica ve, da je podravnatelj Robert, ki na šoli med drugim opravlja tudi nalogo vzdrževalca programske opreme, uspel "sesuti" operacijski sistem tako, da se enostavno ne da več kopirati besedila, Janezku pa ni nič bolj zoprnega kot veliko tipkanja, je kazen vzgojna. Sedaj mora 100x izpisati omenjeni stavek.
Janezek je sprva obupan. Ampak bistra glavica, v Google natipka “C# ponavljanje istih ukazov" in kaj hitro se mu razvedri čelo …

Izpiši 100x isti stavek, seštej 10 števil, izpiši 20 zvezdic, nariši n krogov, seštej dosežene točke vsakega dijaka, izračunaj plačo zaposlenih, …

Kaj je skupno vsem tem problemom. Gre za ponavljanje. Izvajamo isti postopek le s spremenjenimi podatki. Ponavljanju stavkov v programiranju pogosto rečemo **zanke**.

Zanke v programiranju uporabljamo takrat, kadar želimo enega ali več stavkov ponoviti večkrat zaporedoma. V C# poznamo naslednje zanke: `while`, `for`, `do while` in `foreach`. V tem poglavju si bomo ogledali zanki `while in for`. Zanki `do while` in `foreach` uporabljamo redkeje, saj za programiranje povsem zadoščala že zanka `while` (ali zanka `for`). Vsako zanko lahko izvedemo z zanko `while`. Kljub temu bomo spoznali še zanko `for`, saj jo srečamo zelo pogosto. Zanko `while` uporabljamo predvsem takrat, kadar število ponavljanj zanke ni vnaprej znano. Če pa je število zanka odvisno od nekega števca, običajno uporabimo zanko `for`. Zanka `foreach` bo razložena v poglavju o tabelah.

While zanka

Zanka `while` je zanka, ki jo verjetno uporabljamo najbolj pogosto. Uporabimo jo, kadar želimo, da se izvajanje določenih stavkov (oz. stavek) ponavlja, dokler je določen pogoj izpolnjen. Predvsem pa jo uporabljamo takrat, kadar število ponavljanj zanke ni vnaprej znano. Če pa je število prehodov zanke odvisno od nekega števca, običajno uporabimo zanko **for**.

Najpomembnejše značilnosti zanke **while** so

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

Struktura zanke **while**:

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

Tako kot pri pogojnem stavku za besedo `if`, tukaj za besedo `while` v oklepajih zapišemo pogoj. Sledi mu blok stavkov (oz. stavek), ki jih želimo izvajati večkrat – toliko časa, dokler je pogoj izpolnjen.

Zanka `while` se izvaja, dokler je pogoj resničen (ima vrednost *true*). Njegovo resničnost se preveri tudi takoj na začetku, zato ni nujno, da se stavki (oz. stavek), ki mu sledijo, sploh izvedejo.

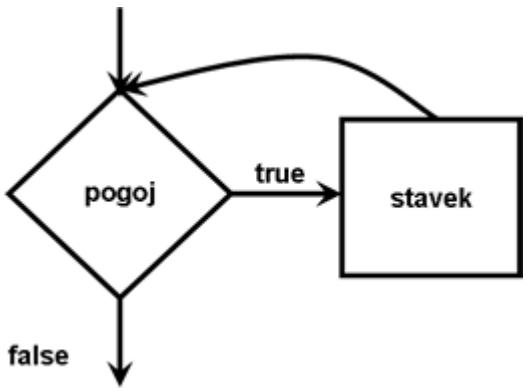
Če je blok sestavljen samo iz enega stavka, lahko oklepaje izpustimo.

```
  while (pogoj) stavek;
```

Običajno pa, tako kot pri pogojnem stavku, tudi tu zaradi boljše preglednosti oklepaje ohranimo.

Na dolgo bi delovanje `while` zanke opisali takole:

- najprej se preveri pogoj,
- če ima pogoj vrednost `true` (je resničen), se izvede stavek,
- nato se ponovno preveri pogoj,
- če je še vedno izpolnjen, se znova izvede stavek,
- ponovno se preveri pogoj,
- ...
- če pogoj ob preverjanju nima več vrednosti `true` (ni več resničen), se konča izvajanje zanke `while`.



Ponavljaj:

če je **pogoj true,**
    izvedi **stavek**,
sicer prekini izvajanje,

Na kratko pa zanko while lahko opišemo z: *Dokler je logični pogoj izpolnjen, izvajaj stavke v zavitih oklepajih.*

Kaznovani Janezek

Poglejmo, kako bi rešili Janezkov problem.
kaj ponavljamo:
izpis stavka
pogoj:
dokler ne izpišemo 100 stavkov
torej moramo šteti izpisane stavke
(izpisanihStavkov < 100)
pred začetkom ponavljanja izpisanih stavkov še ni
izpisanihStavkov = 0;
v zanki
izpišemo stavek
števec izpisanih stavkov povečamo za 1
izpisanihStavkov = izpisanihStavkov + 1;

In ustrezni del programa

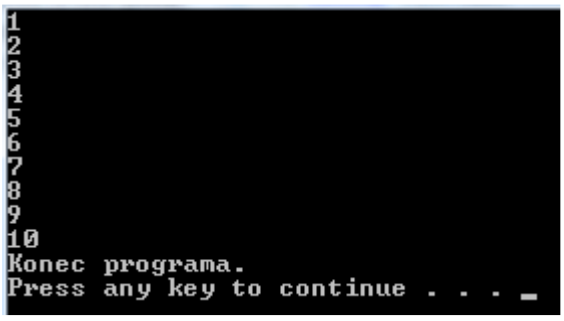
```
int izpisanihStavkov = 0;
while (izpisanihStavkov < 100)
{
    Console.WriteLine("V šoli se moram lepo obnašati!");
    // nov izpisani stavek
    izpisanihStavkov = izpisanihStavkov + 1;
}
```

Izpis števil

Izpišimo števila od *1* do *10*. *V*sako naj bo v svoji vrstici.

```
1:  static void Main(string[] args)
2:  {
3:        int stevilo = 1;
4:        while(stevilo <= 10)
5:        {
6:          Console.WriteLine(stevilo);
7:          stevilo = stevilo + 1;
8:        } // while
9:         Console.WriteLine("Konec programa.");
10: } // main
```

Program prevedemo in poženemo:



Na začetku ob deklaraciji spremenljivki *stevilo* priredimo začetno vrednost *1* (vrstica 3). Nato se preveri pogoj (vrstica 4). Ker je vrednost v spremenljivki manjša ali enaka od *10* (*1 <= 10*), se izpiše vrednost

spremenljivke *stevilo* (vrstica 6), torej *1*. Nato se izračuna desni del, ki se priredi spremenljivki na levi strani (*stevilo*). Vrednost v spremenljivki je sedaj *2*. Nato se zopet preveri pogoj v vrstici 5 in ker je resničen (*2 <= 10*), se izvede blok zanke. Izpiše se trenutna vrednost v spremenljivki *stevilo*, tokrat *2*. Povečamo vrednost spremenljivke *stevilo* na 3. Ponovno preverimo pogoj v vrstici 5. Ker je izpolnjen, ...
Zanka se torej ponavlja, dokler je vrednost v spremenljivki *stevilo* manjša ali enaka *10*. Ko je vrednost v spremenljivki enaka *10*, pogoj še vedno velja. Zato se izvede ukaz v vrstici 6 in na zaslon se izpiše *10*. Nato se vrednost spremenljivke *stevilo* poveča za *1*. Sedaj je vrednost v spremenljivki *stevilo* enaka *11*. Zopet se preveri pogoj v peti vrstici in ker ni več resničen (*11 <= 10*), se zanka zaključi. Program se nadaljuje v vrstici 9. Izpiše se besedilo *Konec programa.*.

Stavek v sedmi vrstici *stevilo = stevilo + 1;* lahko krajše zapišemo kot *stevilo++;*.

Program bi lahko napisali tudi malce drugače

```
static void Main(string[] args)
{
   int stevilo = 0;
   while(stevilo < 10)
   {
      stevilo = stevilo + 1;
      Console.WriteLine (stevilo);
   } // while
   Console.WriteLine ("Konec programa.");
} // main
```

Če program izvedemo, vidimo, da nam da enak izpis kot prejšnji program. Spremembo smo naredili v vrednostih, ki jih zavzame števec. Pri prvi obliki programa je spremeljivka `stevilo` imela zaporedoma vrednosti 1, 2, 3, ..., 9, 10 in 11, pri drugi pa 0, 1, 2, 3, ..., 9 in 10.

Izpis lihih števil

Izpišimo liha števila od *1* do *18*. Tokrat izpišimo vsa v isti vrstici in števila ločimo z presledkom.

```
static void Main(string[] args)
{
    int lihoStevilo = 1;
    while(lihoStevilo <= 18)
    {
       Console.Write(lihoStevilo + " ");
       lihoStevilo = lihoStevilo + 2;
    } // while
} // main
```

Program prevedemo in poženemo:

```
1 3 5 7 9 11 13 15 17 Press any key to continue . . .
```

Opis programa.
Program deluje podobno kot prejšnji, le da tu vrednost, ki je shranjena v spremenljivki *lihoStevilo*, povečujemo za 2.

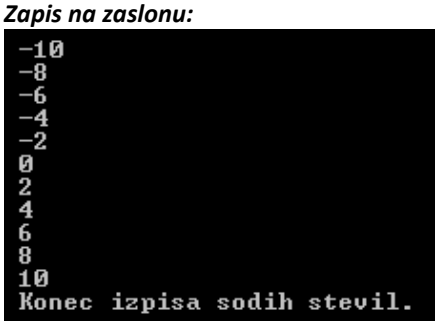
Stavek v vrstici *lihoStevilo = lihoStevilo + 2;* lahko krajše zapišemo *lihoStevilo += 2;*. Podobno okrajšano pisanje lahko uporabimo tudi za ostale operacije (*-, *, /, %*).

Izpis sodih števil

Izpišimo vsa soda števila od -10 do 10. Vsako število izpišimo v svojo vrstico.

```
static void Main(string[] args)
{
      // Deklaracija spremenljivke
      int  a = -10;
      // Izpisujemo vsa soda števila
      while (a <= 10)
      {
         Console.WriteLine(a);
         a = a + 2;
      }
       Console.WriteLine("Konec izpisa sodih stevil.");
}
```

Zapis na zaslonu:



Razlaga. Spremenljivko `a` na začetku nastavimo na `-10`. Nato preverimo pogoj. Ker je vrednost v spremenljivki manjša ali enaka `10`, se izpiše vrednost spremenljivke `a`, torej `-10`. Vrednost spremenljivke `a` se poveča za `2` in je sedaj `-8`. Nato se ponovno preveri veljavnost pogoja. Ker je resničen (ima vrednost `true`), se izvede telo zanke. Izpiše se trenutna vrednost spremenljivke `a`. V našem primeru je to `-8`. Vrednost spremenljivke se poveča na `-6`. Ponovno preverimo pogoj. Ker je resničen …
Zanka se torej ponavlja, dokler je vrednost spremenljivke `a` manjša ali enaka `10`. Ko je vrednost v spremenljivki enaka `10`, se telo zanke izvede zadnjič. Vrednost spremenljivke `a` je po izvedbi `12`. Pogoj se ponovno preveri in ker ni več resničen, se zanka konča. Program se nadaljuje z naslednjo vrstico. Izpiše se stavek `Konec izpisa sodih stevil.`.

Buuum

"Buuum" je igra z naslednjim pravilom: Sodelujoči zaporedoma govorijo števila, vendar morajo namesto večkratnikov števila pet in večkratnikov števila sedem reči *buuum*. Sestavimo program, ki bo po tem pravilu izpisal cela števila od `a` do `b`.

Pomagali si bomo s pogojnim stavkom, ki preveri, ali je število večkratnik števila 5 ali števila 7. Ko bo pogoj pogojnega stavka resničen, bomo izpisali besedo *buuum*.

```
static void Main(string[] args)
{
      // Preberemo interval
      Console.Write("Vnesite a: ");
      int a = int.Parse(Console.ReadLine());
      Console.Write("Vnesite b: ");
      int b = int.Parse(Console.ReadLine());
      // Drugi vnos je manjši od prvega
      If (b < a)
```

```
        {
            int t = a;
            a = b;
            b = t;
        }
        // Izpis besede buuum ali števil
        while(a <= b)
        {
            if((a % 5 == 0) || (a % 7 == 0))
            {
                Console.Write("buuum");
            }
            else
            {
                Console.Write(a);
            }
            Console.Write(" ");
            a = a + 1;
        }
}
```

Zapis na zaslonu:

```
Vnesite a: 1
Vnesite b: 20
1 2 3 4 buuum 6 buuum 8 9 buuum 11 12 13 buuum buuum 16 17 18 19 buuum
```

Razlaga. Z branjem napolnimo spremenljivki `a` in `b`. Nato s pomočjo pogojnega stavka preverimo, če je vrednost spremenljivke `b` manjše od vrednosti spremenljivke `a`. Če je pogoj resničen, zamenjamo vrednost spremenljivke `a` z vrednostjo spremenljivke `b` in vrednost spremenljivke `b` z vrednostjo spremenljivke `a`. Pri zamenjavi vrednosti si pomagamo s pomožno spremenljivko `t`.
S pomočjo zanke `while` nato izpisujemo števila od `a` do `b`. Če je število večkratnik števila `5` ali števila `7`, izpišemo besedo `buuum`, sicer izpišemo preverjeno število. Na koncu vsakega izpisa izpišemo presledek in se pomaknemo na naslednje število.

Neskončna zanka

Pri zanki *`while`* moramo paziti, da se zanka konča. V primeru nepravilne uporabe lahko zanka teče v neskončnost (se zacikla). Pravilno zapisana zanka bo torej taka, kjer pogoj nekoč le dobi vrednost *`false`*.

```
static void Main(string[] args)
{
     int stevilo = 1;
     while(stevilo > 0)
     {
          Console.WriteLine (stevilo);
          stevilo++;
     } // while
} // main
```

Program prevedemo in poženemo:

Program izpisuje števila v neskončnost in se sploh ne ustavi. Pogoj je vedno resničen, saj je vrednost spremenljivke *`stevilo`* vedno večja od *`0`*. No, če bi bili potrpežljivi, bi videli, da se bo program vseeno ustavil. Namreč ko bi prišlo do prekoračitve obsega celih števil, bi v spremenljivki *`stevilo`* dobili negativno število, ki bi ustavila zanko.

No, pri naslednjem zgledu pa se zanka zagotovo nikoli ne ustavi.

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

```
{
     int stevilo = 1;
     while(stevilo < 10)
        Console.WriteLine(stevilo);
        stevilo++;
} // main
```

Program prevedemo in poženemo:

```
1
1
1
1
1
1
.
.
.
```

Tudi v tem primeru je pogoj vedno resničen, saj je vrednost spremenljivke *stevilo* vedno *1* in je vedno večja od *0*. Ker ni zavitih oklepajev, spada v *while* zanko samo stavek *Console.WriteLine(stevilo);* in ker je pogoj vedno resničen, se vrstica `stevilo++;` sploh ne izvede. Vrednost spremenljivke *stevilo* se torej nikoli ne spremeni in je pogoj vedno izpolnjen, iz česar sledi, da se program zacikla.

Povzetek

Kako torej sestavljamo programe, ki potrebujejo zanke:

Premislimo, kaj se dogaja v splošnem

- Tekoča ponovitev zanke
  - Rišemo i-ti krog
  - Pregledujemo tekočo vrstico
  - ...
- Dogajanje na začetku (pred vstopom v zanko)
  - Posebni pogoji ...
  - Nastavitev števcev
  - vrednost kontrolne spremenljivke mora biti taka, da se zanka sploh začne, ...
- Dogajanje na koncu
  - Ali je potrebno z zadnjim elementom kaj posebnega narediti
  - Smo števec po "nepotrebnem" preveč povečali
  - ...

Pogoste napake

Premislimo, kaj naredijo naslednji delčki programa. Njihov učinek bomo le na kratko opisali, bralec pa naj dobro premisli, zakaj in kako!

```
int i = 100;
string odgovor = "";
while (i > 100)
{
     odgovor = odgovor + "riba raca rak ";
     i = i + 1;
}
Console.WriteLine(odgovor);
```

Ta del programa izpiše le prazen niz (torej dejansko ne izpiše nič, le v novo vrstico skoči).

```
string odgovor = "";
while (true)
{
    odgovor = odgovor + "klop pod klopjo, ";
    Console.WriteLine(odgovor);
}
Console.WriteLine("Končali smo...");
```

Te vrstice povzročijo neskončno zanko v kateri se izpisuje

```
klop pod klopjo,
klop pod klopjo, klop pod klopjo,
klop pod klopjo, klop pod klopjo, klop pod klopjo,
klop pod klopjo, klop pod klopjo, klop pod klopjo, klop pod klopjo,
...
```

```
string odgovor = "";
i = 1;
while (i <= 10)
    odgovor = odgovor + "klop pod klopjo, ";
    Console.WriteLine(odgovor);
    i = i + 1;
Console.WriteLine("Končali smo ...");
```

Tudi tu gre za neskončno zanko. A ta ne izpiše nič, saj do `Console.WriteLine(odgovor);` sploh ne pride.

```
string odgovor = "";
int i = 1;
while (i <= 10) ;
{
     odgovor = odgovor + "klop pod klopjo, ";
     Console.WriteLine(odgovor);
     i = i + 1;
}
Console.WriteLine("Končali smo ...");
```

Verjeli ali ne, tudi tu gre za neskončno zanko.

Če povzamemo najbolj pogoste napake:

- Pogoj napačen
  - Zanka se nikoli ne konča!
- Pozabljeni { }
  - Kot pri pogojnem stavku
  - V telesu le en stavek
  - Ker običajno želimo narediti več stvari: sestavljeni stavek
  - Zamikanje ne pomaga (prevajalniku je vseeno)
- Napačno ; takoj za pogojem
  - Zanka, ki ima v telesu "prazen" stavek

Zgledi

Izpis vsote števil

Preberimo število in izračunajmo vsoto celih števil od 1 do prebranega števila. Sicer bi lahko uporabili kar znamenito Gaussovo formulo, s katero bi takoj izračunali to vsoto. A ker se je morda ne spomnimo, bomo s pomočjo zanke res sešteli vsa števila.

```
1:      static void Main(string[] args)
2:      {
3:          int stevilo1 = 1;
4:          int vsota = 0;
5:          Console.Write("Do kam seštevamo: ");
6:          beri = Console.ReadLine(");
7:          int doKam = int.Parse(beri);
8:          while(stevilo1 <= doKam)
9:          {
10:          vsota = vsota + stevilo1;
11:          stevilo1++;
12:         } // while
13:         Console.WriteLine(vsota);
14:     } // main
```

Opis programa.
Zanimivi del programa se začne v vrstici 8. Tam se začne zanka, ki ponavlja stavka v vrsticah 10 in 11 toliko časa, dokler je spremenljivka *stevilo1* manjša ali enaka vnesenemu številu. V 10. vrstici seštevamo števila in njihovo vsoto hranimo v spremenljivki *vsota*.
Denimo, da smo vnesli število *2*. Preveri se pogoj (vrstica 8) in ker je *1* manjše ali enako *2*, se izračuna *0 + 1* (*vsota + stevilo1*) in ta vsota se shrani v spremenljivko *vsota*. Sedaj je v spremenljivki *vsota* vrednost *1*. Vrednost v spremenljivki *stevilo1* se poveča za *1* (vrstica 11) in spremeni se vrednost v spremenljivki *stevilo1* na *2*. Zopet se preveri pogoj (vrstica 8) in ker je *2* manjše ali enako *2*, se izračuna *1 + 2* (*vsota + stevilo1*) in ta vsota se shrani v spremenljivko *vsota*. V spremenljivki *vsota* je sedaj vrednost *3*. Vrednost v spremenljivki *stevilo1* se poveča za *1* (vrstica 11) na *3*. Zopet se preveri pogoj (vrstica 8) in ker *3* ni manjše ali enako *2*, se zanka konča.

Rešitev enačbe

Poiščimo celoštevilske rešitve enačbe *x * x - 4 = 0* na intervalu *[-5, 5]*. V primeru, da na izbranem intervalu ni take rešitve, to izpišimo. Rešitev bomo poiskali enostavno tako, da bomo za vsa cela števila na izbranem intervalu preverili, če ustrezajo enačbi.

```
2:      static void Main(string[] args) {
3:        int x = -5;
4:        bool jeResitev = false;
5:
6:        while(x <= 5) {
7:          if(x * x - 4 == 0) {
8:            Console.WriteLine ("Resitev: x = " + x);
9:            jeResitev = true;
10:         } // if
11:         x = x + 1;
12:       } // while
13:
```

```
14:         if(!jeResitev) {
15:           Console.WriteLine ("Ni resitve!");
16:         } // if
17:       } // main
```

Program prevedemo in poženemo:

```
Resitev: x = -2
Resitev: x = 2
```

Opis programa.
V 3. vrstici deklariramo spremenljivko *x* in ji določimo začetno vrednost *−5*. Ob deklaraciji spremenljivko *jeResitev* nastavimo na false. S pomočjo te spremenljivke bomo ugotovili, ali enačba ima rešitve.
Zanka *while* se bo končala, ko bo *x* večji od *5*, torej *x* teče od *−5* do *5*. Če *x* ustreza enačbi, se izpiše vrednost v spremenljivki *x* (vrstica 8). Pri tem si v logično spremenljivko zapomnimo, da smo našli vsaj eno rešitev. V vsakem primeru se *x* poveča za *1* (vrstica 11).
V vrstici 14 preverimo, če smo našli rešitev. Če je spremenljivka *jeResitev* ostala false, enačba nima celoštevilske rešitve in to dejstvo izpišemo.

Vsota členov zaporedja

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

Zapišimo le "zanimivi del" programa.

```
int n = 1;
int vsota = 0;
Console.WriteLine("Zaporedje ima splosni clen a[n]=(n+1)*(n-1); \n");
Console.Write("Vpisi stevilo clenov zaporedja: ");
int steviloClenov = int.Parse(Console.ReadLine());
while (n <= steviloClenov) // nismo še pregledali dovolj členov
{
    int clen;
    clen = (n + 1) * (n - 1); // izračun n-tega člena
    Console.WriteLine(n +". člen zaporedja je:" + clen); //izpis člena
    vsota = vsota + clen; //vsoto povečamo za n-ti člen
    n = n + 1;  //krajši zapis tega stavka je:    n++;
}
Console.WriteLine("\nVsota prvih "+ steviloClenov +
                  " clenov tega zaporedja je "+ vsota);
```

Izlušči sode števke

Napiši program, ki iz prebranega pozitivnega celega števila naredi novo število, v katerem so le sode števke danega števila. Iz 122436 torej naredimo število 2246. Če so vse števke danega števila lihe, je novo število enako 0.

Ideja programa je, da iz števila "luščimo" zadnje števke. Če so sode (ostanek pri deljenju z 2 je 0), jih dodamo na začetek novega števila. Da jih bomo lahko dodali na začetek, jih bomo morali pomnožiti z ustrezno potenco števila 10. Poglejmo, kako se bodo spreminjali glavni "igralci", če je vnešeno število 14556. Zapisali bomo stanje na začetku vsake ponovitve zanke (torej tik preden se preveri pogoj).

stevilo	novo stevilo	cifra	faktor
14556	0	-	1
1455	6	6	10
145	6	5	10
14	6	5	10
1	46	4	100
0	46	1	100

Tudi tokrat bomo zapisali le "zanimivi" del.

```
1:    int novoStevilo = 0, cifra, faktor = 1;
2:    Console.Write("Vnesi poljubno pozitivno celo število: ");
3:    int stevilo = int.Parse(Console.ReadLine());
4:    while (stevilo > 0){
5:        cifra = stevilo % 10;
6:        if (cifra % 2 == 0)
7:        {
8:            novoStevilo = novoStevilo + cifra * faktor;
9:            faktor = faktor * 10;
10:       }
11:       stevilo = stevilo / 10;
12:   }
13:   Console.WriteLine("\nNovo stevilo je " + novoStevilo);
```

Opis:
V vrstici 1 smo sočasno deklarirali tri spremenljivke: `novoStevilo`, `cifra` in `faktor`. Prvi in zadnji smo priredili tudi vrednost. Pogosto napačno mislimo, da smo v primeru kot je ta, tudi spremenljivki `cifra` priredili vrednost 1. Vendar prirejanje v takem primeru vedno velja le za zadnjo navedeno spremenljivko. Zato je morda bolj smiselno, da bi ta del zapisali kar kot

```
int novoStevilo = 0;
int cifra;
int faktor = 1;
```

in se s tem izognili morebitnim nesporazumom.
Razložimo še vrstico 3. V tej vrstici opravimo cel kup dela. Deklariramo spremenljivko `stevilo` (tipa `int`) in ji priredimo vrednost. To dobimo tako, da z metodo Parse v število pretvorimo niz, ki ga vnesemo preko tipkovnice (in ga posreduje metoda ReadLine). Ta vrstica se torej izvede takole:

- Najprej se pripravi prostor za spremnljivko `stevilo` in sicer tako, da se vanjo lahko shrani celo število (nekaj tipa int)
- Računalnik čaka, da vnesemo določeno zaporedje znakov.
- Ko pritisnemo na tipko Enter, se vnešeni niz posreduje metodi Parse.
- Ta metoda pretvori niz v celo število in ga vrne kot rezultat.
- Dobljeno število se shrani v spremenljivko `stevilo`.

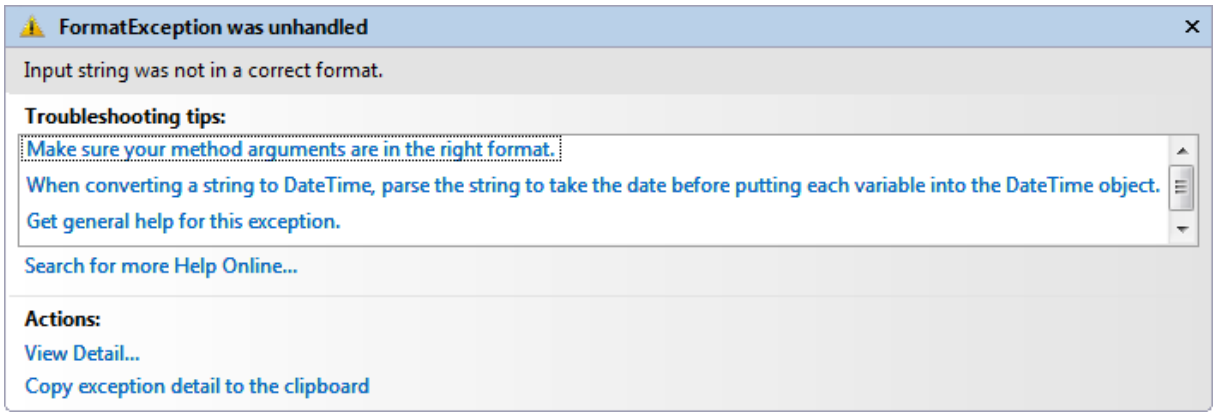
Glavno delo opravimo v zanki `while (4 – 12)`. Zanko izvajamo toliko časa, dokler nam v spremenljivki `stevilo` "ne zmanjka" znakov. Ta spremenljivka bo za vnešeno število 122436 to zaporedoma 122436, 12243, 1224, 122, 12, 1 in na koncu 0. Takrat se bo izvajanje zanke končalo. Če pa bi bilo število enako 65, bi zanko izvedli le 2x, ko bi število postalo najprej 6 in nato 0. V V5 izluščimo tekočo števko. Če je soda (pogoj v V6), jo pomnožimo z ustrezno potenco števila 10. Če jo prištejemo obstoječemu novemu številu, smo jo s tem

dodali ravno z leve. Sedaj še faktor pomnožimo z 10, da bomo naslednjo sodo števko spet dodali levo. V V11 pa "skrajšamo" število, torej odrežemo to števko, ki smo jo ravno obdelali.

Oglejmo si še eno obliko tega programa. Tu se bomo izognili potencam števila 10. Da pa bomo števke lahko "lepili" z leve, bomo uporabili kar nize. Novo število bomo torej sestavljali kot niz in ga na koncu z metodo Parse pretvorili v število.

```
1:    int novoStevilo, cifra;
2:    string noSt = "";
3:    Console.Write("Vnesi poljubno pozitivno celo število: ");
4:    int stevilo = int.Parse(Console.ReadLine());
5:    while (stevilo > 0){
6:        cifra = stevilo % 10;
7:        if (cifra % 2 == 0)
8:        {
9:            noSt = cifra + noSt; // prilepimo z leve
10:       }
11:       stevilo = stevilo / 10;
12:   }
13:   novoStevilo = int.Parse(noSt);
13:   Console.WriteLine("\nNovo stevilo je " + novoStevilo);
```

Pozorni bralec se bo verjetno vprašal, kaj se bo zgodilo, če bo prebrano število npr. 1335, torej tako, samo z lihimi števkami. V prvi različici ni dileme, novo število bo (kot zahteva naloga) enako 0. Kaj pa pri drugi obliki? Vprašanje je torej, kaj stori metoda Parse, če ji "podtaknemo" prazen niz. Žal ji to ni všeč in se pritoži z



Kako se rešiti iz tega položaja, presega naše dosedanje znanje. Namreč poznati bi morali način, kako primerjati nize. To pa si bomo ogledali šele v naslednjem razdelku.

Vsota vnesenih števil

Zanima nas, koliko je vseh štirimestnih števil, pri katerih je vsota zadnjih dveh cifer enaka vsoti prvih dveh. Vsa ta števila bi radi tudi izpisali in sicer tako, da je izpis po 10 v vrsti.

Navedimo le ustrezni del programa brez razlage.

```
int stevilo = 1000, e, d, s, t, vVrsti = 0, vseh = 0;
while (stevilo <= 9999) //preko vseh 4mestnih števil
{
    e = stevilo % 10;     //enice
    d = (stevilo % 100) / 10;  //desetice
    s = (stevilo % 1000) / 100;  //stotice
    t = (stevilo % 10000) / 1000;  //tisočice
```

```
    if (e + d == s + t) //ali število ustreza pogoju
    {
        Console.Write(stevilo + ", ");
        vVrsti = vVrsti + 1;
        vseh = vseh + 1;
        if (vVrsti % 10 == 0) Console.WriteLine();
           // vsakih 10 izpisanih gremo v novo vrsto
    }
    stevilo = stevilo + 1;
}
Console.WriteLine("\n\nVseh takih števil je " + vseh);
```

Povprečje ocen

Denimo da imamo n predmetov. Iz ocen predmetov izračunajmo povprečje ocen, najmanjšo in največjo oceno. Ocene so cela števila med 1 in 10. Pomagali si bomo z metodama *Min()* in *Max()* iz razreda *Math*. Prva vrne manjše, druga pa večje število izmed dveh.

```
4:   static void Main(string[] args) {
5:        int vsota = 0; // sestevek ocen
6:        int stevec = 1; // steje predmete
7:        int najnizja = 11; // najnizja ocena
8:        int najvisja = 0; // najvisja ocena
9:        Console.WriteLine("Stevilo predmetov:" );
10:       string beri = Console.ReadLine();
11:       int steviloPredmetov = int.Parse(beri);
12:
13:       // sestevamo vnesene predmete ter
14:       // iscemo najnizjo in najvisjo oceno
15:       while(stevec <= steviloPredmetov) {
16:          Console.WriteLine("Vnesi " + stevec + ". oceno:");
17:          beri = Console.ReadLine();
18:          int ocena = int.Parse(beri);
19:          vsota = vsota + ocena;
20:          najnizja = Math.Min(najnizja, ocena);
21:          najvisja = Math.Max(najvisja, ocena);
22:          stevec++;
23:       } // while
24:
25:       // povprecje ocen
26:       double povprecje = (double)vsota / steviloPredmetov;
27:
28:       Console.WriteLine ("Tvoje povprecje je " +
29:                          povprecje + ".");
30:       Console.WriteLine ("Zaokrozeno povprecje je " +
31:                           (int)(povprecje + 0.5) + ".");
32:       Console.WriteLine ("Najnizja ocena je " + najnizja);
33:       Console.WriteLine ("Najvisja ocena je " + najvisja);
34:   } // main
```

Opis programa.
Najprej deklariramo ustrezne spremenljivke in jim priredimo ustrezne začetne vrednosti. Preko tipkovnice smo prebrali število predmetov in jih shranili v spremenljivko `steviloPredmetov`.
S pomočjo zanke *while* beremo ocene predmetov. Če je novo število (nova ocena) večje od dosedaj največjega, je med vsemi do sedanjimi števili največje prav to *(Math.Max(najvisja, ocena))*, če pa je manjše ali enako, pa doslej največje število ostane enako. Podobno velja tudi za najmanjše število.

Na vsakem koraku seštevamo ocene. Njihovo vsoto bomo potrebovali za izračun povprečja. Na koncu se izpišemo želene vrednosti. Pri računanju povprečja ne pozabimo, da imamo ves čas opraviti s celimi števili, zato moramo pred delejenjem opraviti ustrezno pretvorbo izraza.

Število e

Določimo vrednost števila `e` (osnove naravnih logaritmov) tako, da seštevamo vrsto `1 + 1/1 + 1/(1·2) + 1/(1·2·3) +` … dokler ne bo prišteti člen manjši od `1/100000`.

Vrednost števila `e` bomo določili tako, da bomo ponavljali postopek, kjer bomo najprej izračunali posamezen člen vrste, nato pa ta člen prišteli preostalim že izračunanim členom.

```
static void Main(string[] args)
{
        // Deklaracija spremenljivk
        int i = 0;
        double clen = 1.0;
        double e = 0;
        const double NATANCNOST = 0.00001;
        // Izračun števila e
        while (NATANCNOST <= clen)
        {
            e = e + clen;
            i = i + 1;
            clen = clen / i;
        };
        // Izpis
        Console.WriteLine("Stevilu e smo izracunali vrednost " + e + ".");
}
```

Zapis na zaslonu:

```
Stevilu e smo izracunali vrednost 2,71827876984127.
```

Razlaga. Najprej deklariramo spremenljivko `i` in ji priredimo začetno vrednost `0`. S to spremenljivko povemo, kateri člen po vrsti računamo. Zatem deklariramo spremenljivko `clen` in ji priredimo začetno vrednost, ki je `1.0`. To spremenljivko potrebujemo za vrednost posameznega člena. Zatem deklariramo spremenljivko `e` in ji priredimo začetno rednost `0`. V tej spremenljivki bomo hranili trenutno vrednost števila `e`.
Vrednost prvega člena že poznamo. Zato to vrednost prištejemo vrednosti spremenljivke `e`. Nato povečamo vrednost spremenljivke `i`. S tem povemo, da bomo računali drugi člen. Potem izračunamo vrednost drugega člena. Ko vrednost izračunamo, jo preverimo. Če je pogoj zanke resničen (če torej člen še ni dovolj majhen), ponovno ponovimo telo zanke in prištejemo vrednost drugega člena vrednosti spremenljivke `e`. Nato ponovno povečamo vrednost spremenljivke `i`. Izračunamo vrednost tretjega člena. Vrednost ponovno preverimo. Če je pogoj zanke resničen, se telo zanke ponovi. Če pa je zaračunani člen že dovolj majhen, se izvajanje zanke konča. Izvrši se stavek za zanko, s katerim izpišemo dobljeno vrednost števila `e`.

Limone

Se še spomnite te naloge med tistimi iz pogojnega stavka?
Na trgu sodelavec prodaja limone. Vsakih nekaj minut se oglasi s klicem
KUPITE! KUPITE! ŠE 3 LIMONE!
oziroma
KUPITE! KUPITE! ŠE 75 LIMON!
glede na to, koliko limon ima.

Seveda je po nekaj dneh že čisto hripav! Zato mu boste pripravili robota s sintetizatorjem govora, ki bo vpil namesto njega.
Kot prvi korak napišimo program, ki bo prebral število limon in izpisal stavek *KUPITE! KUPITE! ŠE x LIMON!*
Seveda v pravilni slovenščini, torej 1 limona, 2 limoni, 3 limone, 4 limone, 5 limon, 6 limon, 7 limon, ...

Takrat smo pri rešitvi vsakič prebrali število limon. Sedaj pa bi našega robota radi naučili, da bo štel limone (seveda slovnično pravilno):

- 98 limon, 99 limon, 100 limon, 101 limona, 102 limoni, 103 limone, 104 limone, 105 limon, …
- povedali mu bomo od kje do kam naj šteje
- zacetnoSteviloLimon, koncnoSteviloLimon, trenutnoSteviloLimon

- zanka
  - trenutnoSteviloLimon se spreminja za 1
  - od zacetnoSteviloLimon do koncnoSteviloLimon
  - trenutnoSteviloLimon = zacetnoSteviloLimon;
  - while (trenutnoSteviloLimon <= koncnoSteviloLimon)
- v zanki
  - tisti slovnično pravilni del iz primera s pogojnim stavkom…
- Ali vemo število ponovitev?
  - da: koncnoSteviloLimon – zacetnoSteviloLimon + 1
- Če sta začetno in kočno število limon narazen za več kot 20, je štetje predolgo. Zato:
- šteje naj po 10, le zadnjih 10 - 20 limon naj prešteje po 1
  - od 46 do 108
  - 46 limon, 56 limon, 66 limon, 76 limon, 86 limon, 96 limon, 97 limon, 98 limon, 99 limon, 100 limon, 101 limona, 102 limoni, 103 limone, 104 limone, 105 limon, 106 limon, 107 limon, 108 limon.
  - zacetnoSteviloLimon, koncnoSteviloLimon, trenutnoSteviloLimon
- zanka po 10
  - trenutnoSteviloLimon se spreminja po 10
  - do kam?
- zanka po 1
  - kot prej!
- Ali vemo število ponovitev obeh zank?
  - da, a če nočemo "mučiti" z matematiko, le približno
    - prva zanka: (koncnoSteviloLimon – zacetnoSteviloLimon) / 10
    - druga zanka: med 10 in 20

```
static void Main(string[] args)
{
    string limone;
    string odg = "";
    int zacetnoSteviloLimon, koncnoSteviloLimon;
    Console.Write("Začetno število limon: ");
    limone = Console.ReadLine();
    zacetnoSteviloLimon = int.Parse(limone);
    Console.Write("Končno število limon: ");
    koncnoSteviloLimon = int.Parse(Console.ReadLine());

    int trenutnoSteviloLimon = zacetnoSteviloLimon;
    int kjeNehamoStetiPo10 = koncnoSteviloLimon - 10;
    // ustavimo se tukaj ali nekoliko prej

    while (trenutnoSteviloLimon <= kjeNehamoStetiPo10)
    {
       int st_limon = trenutnoSteviloLimon;
       st_limon = st_limon % 100; // vsakih 100 se slovnična oblika ponovi!
```

```
        if (st_limon == 0)
        {
            odg = "limon";
        }
        if (st_limon == 1)
        {
            odg = "limono";
        }
        if (st_limon == 2)
        {
            odg = "limoni";
        }
        if ((st_limon == 3) || (st_limon == 4))
        {
            odg = "limone";
        }
        if (st_limon > 4)
        {
            odg = "limon";
        }
        odg = "Imamo " + trenutnoSteviloLimon + " " + odg + "!";
        Console.WriteLine(odg);
        trenutnoSteviloLimon = trenutnoSteviloLimon + 10; // štejemo po 10
    }
    // sedaj moramo prešteti še do konca
    trenutnoSteviloLimon = trenutnoSteviloLimon - 10 + 1;
    // zadnji prirastek za 10 "ni šel skozi"!
    while (trenutnoSteviloLimon <= koncnoSteviloLimon)
    {
        int st_limon = trenutnoSteviloLimon;
        st_limon = st_limon % 100; //vsakih 100 se slovnična oblika ponovi!
        if (st_limon == 0)
        {
            odg = "limon";
        }
        if (st_limon == 1)
        {
            odg = "limono";
        }
        if (st_limon == 2)
        {
            odg = "limoni";
        }
        if ((st_limon == 3) || (st_limon == 4))
        {
            odg = "limone";
        }
        if (st_limon > 4)
        {
            odg = "limon";
        }
        odg = "Imamo " + trenutnoSteviloLimon + " " + odg + "!";
        Console.WriteLine(odg);
        trenutnoSteviloLimon = trenutnoSteviloLimon + 1; // štejemo po 1
    }
    Console.WriteLine("\n\n\nBi kdo limonado?");
    Console.ReadKey();
}
```

V Butalah bodo dobili novo valuto

V Butalah je prava panika. 1. januarja bodo butalske cekine zamenjali za novo valuto, še vedno pa ni znano, kakšen bo menjalni tečaj. No, panika je odveč, saj so iz Banke Butale sporočili, da so vladni modreci odločili, da bo menjalni tečaj odvisen od obnašanja Šprince Marogaste, glavne butalske uši, opoldne na Slivestrovo. Tako bodo imeli celo popoldne in celo noč, da bodo lahko pripravili nove cenike.
No, na srečo pa so v Butalah v veljavi samo cene 50, 100, 150, 200, …, 1000 cekinov. Zato bo glavni butalski informatik pripravil program, ki bo takoj, ko bo znan menjalni tečaj, za vse te cene izpisal njihovo vrednost v novi valuti.
Ampak razbojnik Cefizelj je ravno tiste dni ukradel edino miško v Butalah. Zato pomagaj Butalcem in ti sestavi ustrezni program. Pri tem upoštevaj, da bo menjalni tečaj dan na decimalke, a nova valuta pozna le cele vrednosti. A pozor, Butalci imajo malo drugačno zaokrožanje – če so desetinke sode, se zaokroži navzdol, če pa lihe, pa navzgor.

Ideja programa je sledeča:

- za vsako ceno je potrebno narediti isti postopek
  - izračunati vrednost v novi valuti
  - pravilno zaokrožiti
  - Izpisati vrednost v cekinih in novi valuti
- cene se lepo "urejeno" spreminjajo
  - naraščajo po 50
  - while (cena <= 1000)
    - cena = cena + 50;

Skica programa:

```
int cena = 50; // najnižja cena
double novaVrednost; // pretvorjena cena v novi valuti
int pravaNovaVrednost; // cena v novi valuti
... Beremo ...
double menjalniTecaj = double.Parse (...
while (cena <= 1000) { // cene so manjše od 1000
   novaVrednost = cena / menjalniTecaj;
   // zaokrožanje
   int zacVred = (int)(novaVrednost * 10);
   int enice = zacVred % 10;
   pravaNovaVrednost = zacVred / 10;
   if (enice % 2 == 1) { // lihe desetinke, popravek navzgor
     pravaNovaVrednost = pravaNovaVrednost + 1;
   }
   Console.WriteLine …
   cena = cena + 50; // nova cena
}
```

Stavka break in continue

Stavek **break** povzroči izstop iz (najbolj notranje) zanke tipa **for**, **while** ali **do while**. Stavek **continue** pa ima nasprotno vlogo. Pri zanki **while** skoči na pogoj zanke ter sproži ponovno preverjanje pogoja. Če je ta še izpolnjen, se izvajanje zanke nadaljuje, sicer pa se zanka zaključi.

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

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

Vsota vnesenih števil

Berimo števila in izračunajmo njihovo vsoto. Števila beremo, dokler ne vpišemo 0 za konec.

Pomagali si bomo z ukazom *break*, ki poskrbi, da lahko zanko predčasno zaključimo. Ko se izvede stavek *break;*, zapustimo zanko, v kateri smo. Uporabili bomo neskončno zanko, tako da bo pogoj vedno resničen. Znotraj zanke bomo spraševali po številu toliko časa, dokler ne vnesemo *0*. Takrat bomo zanko prekinili s stavkom *break*.

```
4:      static void Main(string[] args) {
5:        int vsota = 0;
6:        int stevec = 1;
7:
8:        Console.WriteLine ("Za konec vnesi k.\n");
9:
10:        // sestevamo vnesena stevila dokler ne vnesemo k
11:        while (true) {
12:          Console.Write("Vnesi " + stevec + ". stevilo: ");
13:          string beri = Console.ReadLine();
14:          int stevilo = int.Parse(beri);
15:          if (stevilo == 0) {
16:            break;
17:          } // if
18:          else {
19:            vsota = vsota + stevilo;
20:            stevec++;
21:            Console.Write(beri + " ");
22:          } // else
23:        } // while
24:        Console.WriteLine ("\n");
25:        Console.WriteLine ("Vsota teh stevil je " + vsota + ".");
26:      } // main
```

Opis programa.
Najprej deklariramo spremenljivko *vsota* in ji priredimo začetno vrednost *0*. V tej spremenljivki bomo hranili trenutno vsoto števil. Zatem deklariramo spremenljivko *stevec* in ji priredimo začetno vrednost, ki je *1*. To spremenljivko potrebujemo za štetje števil.
V 11. vrstici smo uporabili neskončno zanko, saj je pogoj vedno *true* (resničen). Znotraj zanke definiramo pogojni stavek *if* in če je ta resničen s stavkom, prekinemo izvajanje zanke `while`. Če pogoj ni resničen, se izvedejo stavki od 19. do 21. vrstice. Zatem se ponovno vrnemo na začetek zanke *while* in ponovimo postopek, natanko toliko časa, da je pogoj *(stevilo == 0)* resničen in se zanka zaradi tega, ker se izvede stavek *break;* konča.

Trgovec

Z novim letom se je povečala stopnja davka na dodano vrednost (DDV) z 19% na 20%. Podjetje Trgovec d.o.o. mora v kratkem času spremeniti cene izdelkov tako, da se cena brez davka ne spremeni. To bi bilo sicer nepotrebno, a kaj, ko imajo v svojih podatkih le končne cene (torej cene z že upoštevanim davkom). Zato

napišimo program, s katerim jim bomo pomagali. V program preko tipkovnice vnašamo cene, program pa izpisuje nove vrednosti. To počnemo, dokler ne vnesemo 0. Če je vnesena cena negativna, naj program izpiše "*Cena izdelka je narobe vnesena. Vnesi znova.*". Novo ceno zaokrožimo na dve mesti natančno.

Pomagali si bomo s stavkom `break`, ki poskrbi, da se zanka predčasno zaključi. Ko se namreč izvede stavek `break`, zapustimo stavek (zanko), v katerem se nahajamo.
Uporabili bomo neskončno zanko, v kateri je pogoj vedno `true` (resničen). Znotraj zanke bomo spraševali po ceni toliko časa, dokler ne vnesemo števila 0. Takrat bomo zanko prekinili s stavkom `break`.

```
static void Main(string[] args)
{
     // Deklaracija spremenljivk
     const double ddvPrvi = 1.19; // DDV 19%
     const double ddvDrugi = 1.20; // DDV 20%
     double cena;

     // Preračunavanje cene iz 19% DDV v 20% DDV
     while (true)
     {
         Console.Write("Vnesi ceno izdelka z 19% DDV: ");
         cena = double.Parse(Console.ReadLine());

         if (cena == 0)
         { // Konec vnosov
             break;
         }
         if (cena < 0)
         {
            Console.WriteLine("Cena izdelka je vnesena narobe. " +
                              "Vnesi znova.\n");
         }
         else
         {
            cena = ddvDrugi * cena / ddvPrvi; // Nova cena
            cena = Math.Round(cena * 100) / 100.0;
            Console.WriteLine("Cena izdelka z 20% DDV je " + cena + ".\n");
         }
     }
}
```

Zapis na zaslonu:

```
Vnesi ceno izdelka z 19% DDV: 200
Cena izdelka z 20% DDV je 201.68.

Vnesi ceno izdelka z 19% DDV: -65.5
Cena izdelka je vnesena narobe. Vnesi znova.

Vnesi ceno izdelka z 19% DDV: 0
```

Razlaga. Najprej deklariramo konstantni spremenljivki `ddvPrvi` in `ddvDrugi` in jima priredimo začetni vrednosti. V spremenljivki `ddvPrvi` hranimo 19% DDV, medtem, ko v spremenljivki `ddvDrugi` hranimo 20% DDV. Zatem najavimo spremenljivko `cena`. To spremenljivko potrebujemo za ceno posameznega izdelka. Nato vstopimo v zanko, katere pogoj je vedno `true`. Torej se načeloma zanka izvaja neskončno dolgo, saj bo pogoj ves čas izpolnjen. Znotraj zanke sprašujemo po ceni izdelka. Če je vnesena cena enaka `0`, prekinemo izvajanje zanke `while`. Če je vnesena cena negativna, izpišemo opozorilno besedilo `Cena izdelka je vnesena narobe. Vnesi znova.`. Če pa je vnesena cena izdelka pozitivna, izračunamo novo ceno. Zaokrožimo jo na dve decimalni mesti ter jo izpišemo. Po izpisu opozorila ali nove cene se vrnemo na začetek zanke `while` in ponovimo postopek. Postopek ponavljamo tako dolgo, dokler ni vnesena cena enaka `0`. Takrat se izvede stavek `break` in izvajanje zanke se konča.

Števila deljiva s tri

Izpišimo števila od 1 do 30, ki so deljiva s številom tri. Števila naj bodo izpisana v isti vrstici in ločena s presledkom.

Pomagali si bomo z ukazom `continue`, ki poskrbi, da se posamezna ponovitev zanke predčasno prekine in se zanka nadaljuje z naslednjo ponovitvijo (če je pogoj za ponovitev zanke še izpolnjen).
Uporabili bomo pogojni stavek, s katerim bomo preverjali, če število ni deljivo s 3. Če bo pogoj resničen, bomo s pomočjo stavka `continue` takoj nadaljevali z naslednjo ponovitvijo stavkov telesa zanke.

```
static void Main(string[] args)
{
      // Deklaracija spremenljivke
      int stevilo = 0;
      const int zgMeja = 30; // Zgornja meja števil

      // Izpis števil deljivih s tri
      while (stevilo <= zgMeja)
      {
          stevilo++;
          if (stevilo % 3 != 0) continue;
          Console.Write(stevilo + " ");
          // Izpis le, če je število deljivo s 3
      }
      // Premik v novo vrstico
      Console.WriteLine();
}
```

Razlaga.
Spremenljivko `stevilo` nastavimo na vrednost `0` in konstantno spremenljivko `zgMeja` na vrednost `30`. Potem povečamo vrednost spremenljivke `stevilo` za `1`. Če je vrednost spremenljivke deljivo s številom `3`, se izpiše vrednost spremenljivke in presledek. Če vrednost spremenljivke ni deljivo s `3`, pa se samo poveča vrednost spremenljivke `stevilo` za `1`. V tem primeru s stavkom `continue` skočimo na začetek zanke.

Zapis na zaslonu:

```
3 6 9 12 15 18 21 24 27 30
```

Naloge za utrjevanje znanja iz zanke while

- Tabelirajte funkcijo *y = tan(x)* na intervalu od 0 do *π* s korakom *π/10*. Če vrednost *y* vmes preseže 10, prekinite izvajanje zanke.
  Namig: Pomagajte si z metodo *Math.Tan()*.

- Pitagora je imenoval dve celi števili prijateljski, če je vsota deliteljev prvega števila enaka drugemu številu in obratno. Napišite program, ki bo prebral dve celi števili in preveril, ali sta števili prijateljski.

  Primer:
  Števili 220 in 284 sta si prijateljski.
  220 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284
  284 = 1 + 2 + 4 + 71 + 142 = 220

- Podjetje Parkiranje d.o.o. upravlja parkirno hišo in hoče voditi evidenco njene zasedenosti. Zato vas pokliče, da bi jim napisali program, ki bi jim pri tem pomagal. Parkirna hiša ima 100 parkirnih mest in je ob

zagonu programa prazna. Za vsak avto, ki v njej parkira, operater vtipka **1**, za vsak avto, ki odpelje iz nje pa 2. Za izhod iz programa pritisne 0. Program vsakič tudi izpiše, koliko parkirnih mest je zasedenih. Program naj ustrezno reagira, če hoče avto parkirati v polni parkirni hiši ali če avto pripelje iz "prazne" parkirne hiše.

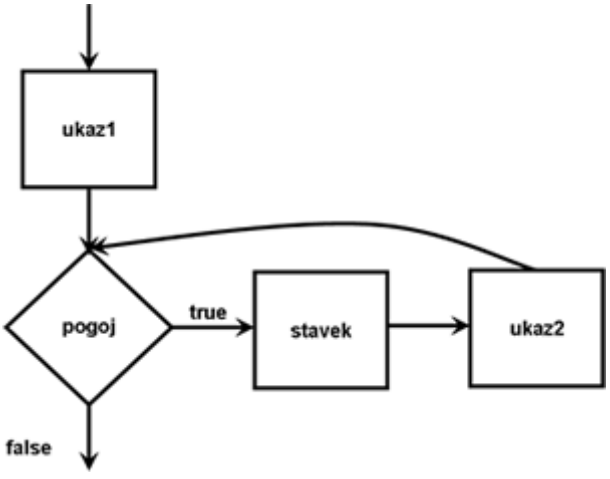
Gosenulje so posebne živali. Ko se izvalijo iz jajčeca, so videti natanko tako kot gosenice. Za razliko od gosenic, ki se zabubijo in iz katerih nato nastane metulj, odraščajo gosenulje malo drugače. Prvo zimo svojega življenja se zabubijo in zaspijo za toliko let, kolikor je vsota števk števila njihovih nog. V času, ko spijo, jim vsako leto zrase dodaten par nog. Primer: gosenulja, ki bi šla spat s 173 pari nog, bi se zbudila čez 1+7+3=11 let, hkrati pa bi imela tudi 11 parov nog več, torej 184. Naslednjič zaspijo v istem letu, kot se zbudijo. Napiši program, ki najprej vpraša, koliko parov nog je imela gosenulja, ko je šla prvič spat, nato pa izpiše, ali se bo kdaj zgodilo, da se bo gosenulja zbudila s toliko pari nog, da bo njihovo število deljivo s 199. Gosenulje umrejo, ko dopolnijo 1234 let.

Zanka For

Zanka `For` je verjetno najpogosteje uporabljena zanka. Je zelo podobna zanki `while`. Za besedo `for` v oklepajih zapišemo tri dele: `ukaz1`, `pogoj` in `ukaz2`, ločene z podpičjem. Sledi mu blok stavkov (oz. stavek), ki jih želimo izvajati večkrat – toliko časa, dokler je `pogoj` izpolnjen.
Preden se zanka sploh začne izvajati, se izvede `ukaz1`. Nato se preveri `pogoj`. Če je ta izpolnjen, se izvedejo stavki (oz. stavek), nato še `ukaz2`. Ponovno se preveri `pogoj`. Če je še vedno izpolnjen, se spet izvedejo stavki v telesu zanke in za njimi še `ukaz2`. Če pogoj ni izpolnjen, se zanka konča.

```
for(predZanko/ukaz1/; pogoj; poKoraku/ukaz2/) {
      stavek1;
      stavek2;
      …
      stavekn;
}
```

Oblika zanke pomeni: "Izvedi stavek `predZanko`. Nato preveri `pogoj`. Če je izpolnjen, izvedi stavke $stavek_1$, $stavek_2$ … $stavek_n$ in nato še stavek `poKoraku`. Ponovno preveri `pogoj`. Če je še vedno izpolnjen, ponovno izvedi stavke $stavek_1$, $stavek_2$ … $stavek_n$ in še stavek `poKoraku`. Ponovno preveri `pogoj`. Ko `pogoj` ni izpolnjen, se zanka konča."



Najprej izvedi **ukaz1**

Dokler **pogoj resničen,** ponavljaj:

izvedi **stavek** in nato še **ukaz2**

Čeprav je v zanki `for` `ukaz2` napisan pred `stavki`, se najprej izvedejo `stavki` in šele nato `ukaz2`.

Katero zanko bomo uporabili, je odvisno od naše odločitve. Zanko `for` običajno uporabljamo, ko moramo šteti ponovitve. Stavek `predZanko` takrat običajno uporabimo za nastavitev števca, v pogoju preverjamo, če je števec že dosegel določeno mejo, v stavku `poKoraku` pa povečujemo (ali zmanjšujemo) števec.

Vsako zanko *for* lahko zapišemo z enakovredno zanko *while*:

```
ukaz1
while(pogoj) {
  stavki;
       ukaz2;
}
```

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

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

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

Izpis števil

Popravimo program *IzpisStevil* iz prejšnjega razdelka tako, da bomo uporabili zanko *for*. Izpisati želimo števila od *1* do *10* vsakega v svoji vrstici.

```
1:    class IzpisStevil2 {
2:      static void Main(string[] args) {
3:        for(int stevilo = 1; stevilo <= 10; stevilo++) {
4:          Console.WriteLine (stevilo);
5:        } // for
6:        Console.WriteLine ("Konec zanke");
7:      } // main
8:    } // IzpisStevil2
```

Spremenljivki *stevilo* se najprej priredi začetna vrednost *1 (int stevilo = 1)*. Nato se preveri pogoj (*stevilo <= 10*). Če je resničen, se izvedejo stavki v telesu zanke, sicer ne. V našem primeru je pogoj izpolnjen (*1 <= 10*), zato se izvede stavek v 4. vrstici, ki izpiše vrednost spremenljivke (*1*). Nato se izvede *ukaz2*, ki vrednost spremenljivke *stevilo* poveča za ena. Preveri se pogoj in ker še vedno velja (*2 <= 10*), se s stavkom v 4. vrstici zopet izpiše vrednost spremenljivke (*2*). Na enak način se *for* zanka izvaja, dokler je vrednost v spremenljivki *stevilo* manjša ali enaka *10*. Ko je v spremenljivki vrednost 10, je pogoj še vedno izpolnjen. Ponovno se izvede stavek *Console.WriteLine (stevilo);*, ki izpiše 10. Nato se z *ukazom2* (*stevilo++*) vrednost v spremenljivki *stevilo* zopet poveča za ena in postane *11*. Preveri se pogoj. Ker 11 ni manjše od 10, pogoj ni več izpolnjen, zato se zanka zaključi in program se nadaljuje v vrstici 6. Izpiše se besedilo *Konec zanke*.

Izpis lihih števil

Popravimo program *LihaStevila* iz prejšnjega poglavja tako, da bomo uporabili zanko *for*. Izpisati želimo liha števila od *1* do *10*, v isti vrstici in s presledkom kot ločilnim elementom.

```
1:    class LihaStevila1 {
2:      static void Main(string[] args) {
3:        for(int lihoStevilo = 1; lihoStevilo <= 10;
4:            lihoStevilo = lihoStevilo + 2) {
5:          Console.Write (lihoStevilo + " ");
6:        } // for
7:      } // main
8:    } // LihaStevila1
```

Spremenljivki *lihoStevilo* se najprej priredi začetna vrednost *1*. Nato se preveri pogoj. Ker je resničen *(1 <= 10)*, vstopimo v zanko in izpiše se vrednost spremenljivke (*1*). Nato se izvede ukaz2 (*lihoStevilo = lihoStevilo + 2*), ki vrednost spremenljivke *lihoStevilo* poveča za dva. Preveri se pogoj in ker še vedno velja *(3 <= 10)*, se zopet izpiše vrednost spremenljivke (*3*). Na ta način se zanka *for* izvaja, dokler je vrednost v spremenljivki *lihoStevilo* manjša ali enaka *10*. Ko je v spremenljivki vrednost *9*, je pogoj izpolnjen in 5. vrstica izpiše to število. Sedaj spremenljivka *lihoStevilo* dobi vrednost *11*. Preveri se pogoj. Ker *11* ni manjše od *10*, pogoj ni več izpolnjen, zato se zanka konča.

Neskončna zanka for

Tudi pri zanki for moramo paziti, da se zanka konča. V primeru nepravilne uporabe tudi zanka *for* teče v neskončnost (se zacikla).

```
1:    class Ciklanje {
2:      static void Main(string[] args) {
3:        for(int stevilo = 1; stevilo > 0; stevilo++) {
4:          Console.WriteLine (stevilo);
5:        } // for
6:      } // main
7:    } // Ciklanje
```

Program prevedemo in poženemo:

```
1
2
3
4
5
6
.
.
.
```

Velja enaka opazka, kot smo jo naredili pri zanki *while*. Tudi tu bi zaradi prekoračitve obsega prišli do negativnih števil in dejansko bi se zanka ustavila. A programirati na način, ko izrabljamo, da za *1* povečano največje možno število tipa *int* postane negativno, je zelo nevarno in ga ne smemo uporabljati.

Zgledi

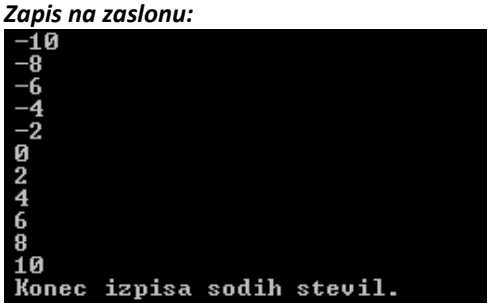
Izpis sodih števil

Popravimo še program `SodaStevila.cs` iz prejšnjega poglavja tako, da bomo uporabili zanko `for`. Izpisati želimo soda števila od -10 do 10, vsakega v svojo vrstico.

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

    // Izpisujemo vsa soda števila
    for(int a = -10; a <= 10; a = a + 2)
    {
        Console.WriteLine(a);
    }
    Console.WriteLine("Konec izpisa sodih stevil.");
}
```

Zapis na zaslonu:



Razlaga.
Na začetku for zanke se spremenljivki `a` najprej priredi začetna vrednost `-10`. Nato se preveri pogoj. Ker je resničen, vstopimo v zanko in izpiše se vrednost spremenljivke (`-10`). Nato se vrednost spremenljivke `a` poveča za `2`. Preveri se pogoj in ker še vedno velja (ima vrednost `true`), se ponovno izpiše vrednost spremenljivke (`-8`). Na ta način se zanka `for` izvaja, dokler je vrednost spremenljivke `a` manjša ali enaka `10`. Ko vrednost spremenljivke doseže `12`, se zanka zaključi. Program se nadaljuje z vrstico za zanko `for` . Izpiše se besedilo `Konec izpisa sodih stevil.`.

Pravokotnik

Napišimo program, ki vpraša po širini in višini pravokotnika, nato pa izriše pravokotnik sestavljen s samih zvezdic.

Tu bomo uporabili gnezdeno zanko `for`. Tako kot pri pogojnem stavku, so tudi stavki v telesu zanke `for` poljubni stavki. In če uporabimo spet zanko `for` (ali tudi katero drugo zanko), je ta znotraj druge zanke `for` ("v gnezdu"). Zato rečemo, da gre za gnezdeno zanko `for`.

```
static void Main(string[] args)
{
      // Vnos dimenzij pravokotnika
      Console.Write("Vnesi visino: ");
      int visina = int.Parse(Console.ReadLine());
      Console.Write("Vnesi sirino: ");
      int sirina = int.Parse(Console.ReadLine());

      // Preverimo, obstoj pravokotnika
      if (0 < visina && 0 < sirina)
      {
          Console.WriteLine();
          // Izpisovanje vrstic
          for (int i = 1; i <= visina; i++)
          {
              // Izpis stolpcev
              for (int j = 1; j <= sirina; j++)
              {
```

```
                Console.Write("*");
            }
            // Po izpisu stolpcev, se pomaknemo v novo vrstico
            Console.WriteLine();
        }
    }
  }
}
```

Zapis na zaslon:

```
Vnesi visino: 5
Vnesi sirino: 8

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

Zanimiva števila

Sestavi program ki poišče vsa naravna števila med 0 in 10000, ki so enaka vsoti kubov svojih števk. Eno od števil, ki ima zahtevano lastnost, je npr. 153: $153 = 1^3 + 5^3 + 3^3$.

Ideja programa je v tem, da s pomočjo štirojne zanke ustvarimo vsa možna števila med 0 in 9999.

```
static void Main(string[] args)
{
   int i, j ,k, l;  //števke
   double st, vk;    //število in vsota kubov števk
   for (i = 0; i < 10; i++)
     for (j = 0; j < 10; j++)
       for (k = 0; k < 10; k++)
         for (l = 0; l < 10; l++)
         {
           st = 1000 * i + 100 * j + 10 * k + l; // sestavimo število
           vk = Math.Pow(i, 3) + Math.Pow(j, 3) + Math.Pow(k, 3)
                + Math.Pow(l, 3); // vsota kubov
           if (st == vk)
              Console.Write(st+",   ");
         }
}
```

Ali znaš napisati program tako, da uporabiš le enojno zanko?

Vzorec

Napiši program ki izpiše naslednji vzorec. Primer: za n = 5 je izpis takle:

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```

```
static void Main(string[] args)
{
   int i,j;
```

```
   Console.Write("Velikost vzorca (1 - 9): ");
   int n = int.Parse(Console.ReadLine());
   for(i = n; i > 0; i--)
   {  // z zunanjo zanko izpisujemo vrstice
      for(j = n; j >= i; j--)// izpis posameznih števil v vrstici
        Console.Write(i);
      Console.WriteLine(); // zaključimo tekočo vrstico
   }
}
```

Tu smo uporabili oprator --. Ta zmanjša vrednost spremenljivke za 1. Zapis

```
i--
```

torej pomeni

```
i = i - 1
```

Vsota deljivih števil

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

```
static void Main(string[] args)
{
   int vs = 0; //začetna vsota je enaka 0
   Console.Write("Vnesi naravno število n: ");
   int n = int.Parse(Console.ReadLine());
   Console.Write("Vnesi delitelj: ");
   int delitelj = int.Parse(Console.ReadLine());
   //v naslednji for zanki je prvi stavek (pogoj) izpuščen!
   //začetna vrednost spremenljivke n je določena že izven zanke
   for (; n >= delitelj; n--)
   {
      if ((n % delitelj) == 0)
           vs = vs + n;
   }
   Console.WriteLine("\nVsota je {0}", vs);
}
```

Vsota vrste

Napiši program, ki izračuna naslednjo vsoto za poljubno število sumandov!

$$\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \ldots$$

Izpisuj tudi vmesne vsote!

```
static void Main(string[] args)
{
   int clenov = 0;
   //dimenzijo trikotnika vnese uporabnik. Rišemo le, če dimenzija >=1
   while (clenov < 1)  //vneseni podatek mora biti več ali enako 1
   {
     Console.Clear();
```

```
      Console.Write("Vnesi število sumandov (1 ali več) : ");
      clenov = int.Parse(Console.ReadLine());
    }
    double suma = 0;
    int n = 0;
    while (n < clenov)
    {
      suma = suma + (double)2 / (2 + n);
      Console.WriteLine(n + 1 + " : " + suma); //izpisujem vmesne rezultate
      n++;
    }
    Console.WriteLine("\nVsota " + clenov + ". členov tega zaporedja je " +
                      suma);
}
```

Naloge

- Kolikšna je vsota vseh naravnih števil med 1 in 1000?
- Tabeliraj linearno funkcijo f(x) = 4x + 3 na intervalu [-5 , 5] s korakom 1!
- Seštej 100 členov vrste: **vrsta = 1 + 1/2+ 1/3 +1/4 + ..**
- Preberi poljubno celo število in izpiši njegov poštevanko (do 10)!
- Preberi poljubno celo število in ga izpiši navpično!
- Preberi poljubno celo število in ga izpiši z besedami. Celo število **235** tako izpiši kot **dva tri pet**.
- Napiši program, ki bo izpisal vse kvadrate naravnih števil do n (n preberemo), ki se naprej in nazaj berejo enako. (npr 141, 232, 181, …).
- Napiši program, ki bo prebral dve celi števili, ugotovil, katero je večje in izpisal:
  - vsa števila med najmanjšim in največjim številom.
  - vsa števila med najmanjšim in največjim številom, ki delijo največje število.
  - vsa števila med najmanjšim in največjim številom, ki so soda in delijo največje število.
  - vsa števila med najmanjšim in največjim številom, ki so soda ali delijo največje število.
- Sestavi program *Kopije*, ki prebere niz *beseda* in pozitivno celo število ***k*** ter izpiše niz, ki je sestavljen iz ***k*** kopij niza *stavek*.
- Sestavi program, ki bo prebral naravno število in izračunal produkt njegovih neničelnih števk. Primer: za število 2304701 program vrne 168.
- Preberi poljubno celo število manjše od 100, računalnik pa naj ga ugane. Seveda predpostaviš, da računalnik števila ne pozna in da ga ugiba. Ugotovi in izpiši koliko poskusov bo računalnik potreboval za to, da ugotovil pravo število. Kakšno taktiko pa bo računalnik ubral, je odvisno od tebe. Nekaj primerov:
  - naključno ugiba števila, dokler ne ugane pravega. Pri tem nič ne upošteva podatka o tem, ali je izžrebano število od iskanega manjše ali večje.
  - gre po vrsti od 1 do 100. Pri tem nič ne upošteva podatka o tem, ali je izžrebano število od iskanega manjše ali večje.
  - naključno ugiba števila, dokler ne ugane pravega. Pri tem upošteva podatek ali je naključno število preveliko ali premajhno število tako, da zoži interval, iz katerega žreba števila.

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

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

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

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

- Preberi poljubno celo število in ga pretvori v dvojiški sestav. Pretvorjeno število nato izpiši!

- Poišči največji skupni delitelj dveh celih števil

- Napiši program, ki bo izpisal vse kvadrate naravnih števil do n (n preberemo), ki se naprej in nazaj berejo enako (npr 121, 676, 484, …). Uporabi **while** zanko.

- Sestavi program, ki prebere n decimalnih števil (tudi n je podatek). Izpiši, koliko od teh števil je manjših od 10, med 10 (vključno) in 100 (vključno) in koliko večjih od 100.

- Sestavi program, ki prebere velikost paralelograma in ga izpiše na zaslon. Izpis paralelograma za velikost 5 naj bo takle

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

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

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

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

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

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

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

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

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

Zanka do while

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

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

Struktura **do while** zanke:

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

Trikotnik iz zvezdic

Napišimo program, ki nariše trikotnik iz samih zvezdic. Velikost trikotnika določi uporabnik.

```
static void Main(string[] args)
{
  int velikost;
  do
  {
    Console.Clear();
    Console.Write("Vnesi velikost trikotnika ( 1 - 24 ) : ");
    velikost = Convert.ToInt32(Console.ReadLine());
  }
  while (velikost < 1 || velikost > 24); //Podatek mora biti med 1 in 24

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

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

Zaporedje

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

```
static void Main(string[] args)
{
   int n = 3;
   long f, f1 = 1, f2 = 1;
   Console.WriteLine("Vrednosti funkcije f(n) = f(n-1) + f(n-2) na
      intervalu od 1 do 40\n\n");
   Console.Write(f1+"  "+ f2);
   do
   {
      f = f1 + f2;
      f2 = f1;
      f1 = f;
      Console.Write(f);
      n++;
      if (n % 5 == 1) Console.WriteLine(); //zato, ker se zanka začne z n=3
   } while (n <= 40);
}
```

Naloge (do while zanka)

- S pomočjo **do while** zanke beri znake toliko časa, da je vneseni znak enak pika (.). Prebrane znake nato izpiši v obliki stavka.
- Napiši program, ki bo izpisal vse kvadrate naravnih števil do n (n preberemo), ki se naprej in nazaj berejo enako (npr 121, 676, 484, …). Uporabi **do while** zanko.
- Sestavi preprost program, ki bo kodiral in dekodiral kratka telefonska sporočila (sms-e). Program mora v obrniti vrstni red besed v sporočilu in vrstni red črk v besedi.
- Preberi naravno število n. Če je liho, izračunaj 3n+1, sicer pa n/2.Isto naredi z novim številom in postopek ponavljaj, dokler ne dobiš 1. Izpiši število korakov postopka!
- Nek izdelek naj bi se vsak teden podražil za 2%. Da bi državljani razumeli, kaj to pomeni, napiši program, ki prebere začetno in končno ceno tega izdelka in izpiše, v kolikšnem času (število tednov) bo cena dosegla ali presegla to končno vrednost.

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

Naključna števila

V programiranju večkrat potrebujemo naključna števila.
V jeziku C# je za to na voljo razred (kar pač je že to) `Random`. Metode razreda `Random` uporabljamo tako, da najprej ustvarimo objekt tipa `Random`. To storimo s pomočjo operatorja `new`. Kaj dejansko to pomeni, bomo pojasnili kasneje, ko bomo govorili o razredih in objektih.

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

Obekt **`nakljucno`** sedaj lahko uporabimo za generiranje naključnih števil. Ta objekt si lahko predstavljamo kot nekakšen boben, iz katerega potem z različnimi metodami "žrebamo" naključna števila. Najpogostejše metode razreda `Random` so prikazane v spodnji tabeli.

C#	Razlaga
Next(n)	Naključno celo število tipa `int`.
Next(int n)	Naključno število tipa `int` iz intervala [`0`, `n`).
Next(int n, int m)	Naključno število tipa `int` iz intervala [`n`, `m`).
NextDoube()	Naključno število tipa `double` iz intervala [`0`, `1`).

Primer uporabe:

```
Random nakljucno = new Random();

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

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

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

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

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

Kocka

Marko bi rad zvedel, kolikokrat mora vreči kocko, da bo skupaj vrgel 100 pik ali več. A ve, da je metanje kocke naporno opravilo. Zato mu napišimo program, ki bo simuliral metanje kocke tako, da bo "metal" kocko toliko časa, dokler ne bo vsota vseh pik presegla 100. Program naj po vsakem metu izpiše vržemo število pik in skupno vsoto. Na koncu naj izpiše število metov kocke, potrebnih, da je skupna vsota presegla 100.

V zanki bomo z ustrezno metodo razreda `Random` določili število pik pri posameznem metu. Dobljene pike bomo nato prišteli k skupni vsoti vseh pik.

```
static void Main(string[] args)
{
        // Ustvarimo generator naključnih števil
        Random nakljucno = new Random();

        // Zgornja meja vseh pik
        const int zgMeja = 100;
        // Šteje število pik
        int vsota = 0;
        // Šteje število metov
        int stevec = 0;

        // Simulacija meta kocke
        while(vsota <= zgMeja){
            // Število pik ob posameznem metu kocke
            int met = nakljucno.Next(6) + 1;
            vsota = vsota + met;
            Console.WriteLine("Zadnji met: " + met +
                  "\tVsota: " + vsota);
            stevec = stevec + 1;
        }
        // Izpis števila metov kocke
        Console.WriteLine("\nStevilo metov kocke je " + stevec + ".");
}
```

Razlaga.
Najprej ustvarimo generator naključnih števil. Nato deklariramo ustrezne spremenljivke in jim priredimo ustrezne začetne vrednosti.
S pomočjo zanke `while` simuliramo metanje kocke. Pri vsakem "metu" kocke določimo število dobljenih pik. Te pike prištejemo skupni vsoti pik. Dobljene pike in skupno vsoto pik na vsakem koraku izpišemo. Na vsakem koraku tudi povečujemo števec, s katerim štejemo število metov kocke oz. korake zanke. Na koncu še izpišemo število vseh metov kocke.

MonteCarlo

Izračunajmo število π po metodi MonteCarlo. Ta metoda na karatko pomeni, da določen poskus ponovimo velikokrat in štejemo ugodne izide. Nato po statistični metodi izračunamo verjetnost (število ugodnih poskusov / število vseh poskusov). Ker poznamo teoretično verjetnost (ki se izraža s π), iz tega lahko določimo pi. V našem primeru je postopek sledeč. Izberemo koordinate točke v kvadratu s stranico 1. Če to naredimo velikokrat, je razmerje med številom tistih, ki ležijo v krogu z radijem 1 in vsemi, je pi/4.

Denimo, da bomo izbrali 10000000 točk. Izberimo naključno točko v enotskem kvadratu. Nato pogledamo ali se nahaja v enotskem krogu, če se povečamo število zadetkov za ena, sicer pa ne. Število π izračunamo po naslednji enačbi π = 4.0 * zadetki / poskusi.

```
static void Main(string[] args)
{
     int n = 10000000;
     int k = 0;
     Random nakljucno = new Random();
     for (int i = 0; i < n; i = i + 1)
     {
          double x = nakljucno.NextDouble();
          double y = nakljucno.NextDouble ();
```

```
            if ( x*x + y*y <= 1) // zadeli smo enotski krog
               k = k + 1;
      }
      Console.WriteLine ("V "+ n +" metih dobimo, da je pi = "
                     + (4.0 * k / n));
}
```

```
V 10000000 metih dobimo, da je pi = 3,1412956
```

Ugibanje števila

Napišimo program, ki si bo "izbral" naključno število med 1 in 100, mi pa ga moramo v čim manj poskusih uganiti. Program naj glede na naše ugibanje pove, ali je povedano (vneseno) število premajhno ali preveliko. Ko število uganemo, naj izpiše število poskusov.

V zanki bomo uporabniku ponudili vnos števila in nato glede na vneseno število izpisali ustrezno obvestilo.

```
static void Main(string[] args)
{
      int stevec = 0; // steje stevilo poskusov
      Random boben = new Random();// boben naključnih števil
      int stevilo = -1; // stevilo, ki ga vnesemo
      // stevilo, ki ga ugibamo
      int nakljucnoStevilo = boben.Next(1, 101);

      while (nakljucnoStevilo != stevilo)
      {
            Console.Write("Vnesi stevilo:");
            string beri = Console.ReadLine();
            stevilo = int.Parse(beri);
            stevec++; // nov poskus

            if (stevilo < nakljucnoStevilo)
            {
                  Console.WriteLine("Stevilo " + stevilo +" je premajhno!");
            } // if
            else if (stevilo > nakljucnoStevilo)
            {
                Console.WriteLine("Stevilo " + stevilo + " je preveliko!");
            } // else
      } // while

      Console.WriteLine("\n");
      Console.WriteLine("Bravo! Uganil si!");
      Console.WriteLine("Stevilo je " + nakljucnoStevilo + ".");
      Console.WriteLine("Stevilo poskusov: " + stevec);
} // main
```

Opis programa:
Najprej definiramo *stevec* s katerim bomo šteli število poskusov in *stevilo*, v katerem bomo hranili vneseno število. Slednjemu dodelimo vrednost -1, zakaj? Odgovor dobimo v 10. vrstici, kjer si podrobneje oglejmo pogoj zanke *while*. V pogoju primerjamo *nakljucnoStevilo*, v katerem hranimo naključno število med 1 in 100 in *stevilo*. Če sta vrednosti spremenljivk različni, vstopimo v zanko. Da je delovanje programa pravilno, moramo v zanko vstopiti vsaj enkrat . To bomo zagotovili le če, pogoj na začetku ne bo izpolnjen. Ker v spremenljivki *nakljucnoStevlo* hranimo cela števila med 1 in 100, moramo začetno vrednost spremenljivke *stevilo* nastaviti na katerokoli število, ki ni med 1 in 100, na primer na -1.

Znotraj zanke `while` sprašujemo po tej naključni številki. Če jo uganemo, se ob naslednjem poskusu izvajanja zanke, le-ta konča. Če število ni enako izbranemu, program izpiše, ali je naše število premajhno ali preveliko in nam tako pomaga čim hitreje uganiti število. Ko uganemo, se izpiše koliko poskusov smo potrebovali, da smo prišli do želenega števila.

Naloge za utrjevanje znanja iz naključnih števil

- Napišite program, ki bo prebral število *n* ter izpisal 5 naključnih realnih števil med 0 in *n-1*. Realna števila naj imajo največ dve decimalki.

- V poglavju Odločitve smo med nalogami za utrjevanje znanja naredili nalogo, ki za vnesene stranice preveri, če lahko določajo trikotnik. V tej nalogi pa naj bodo stranice naključna izbrana števila iz intervala [-6, 20). Sestavimo program, ki bo za tri naključno izbrane stranice preveril, ali lahko tvorijo trikotnik. V primeru, da je trikotnik mogoče sestaviti, naj program izračuna še njegovo ploščino in obseg.
  Namig: Kot smo povedali pri omenjeni nalogi, je pogoj, da lahko tri števila določajo stranice trikotnika ta, da so stranice pozitivna števila, posamezna stranica pa mora biti manjša kot vsota drugih dveh.

- Generirajte naključno štirimestno pozitivno število in izračunajte vsoto njegovih števk.

- Računalnik si je "izmislil" število med 1 in 100. Koliko korakov boste potrebovali, da število uganete? Sestavite ustrezen program. Računalnik vam odgovarja:
  - s "premajhno", če bo vaše število manjše od računalnikovega in
  - s "preveliko", če bo vaše število večje.
  - Seveda vam bo povedal tudi, ko boste število uganili.

- Napišite program, ki bo pomagal generirati števila za igro Loto. Računalnik naj naključno generira samo prvo in drugo število, medtem ko ostalih 5 vnese uporabnik. Pri tem mora uporabnik pri vnosu sam poskrbeti, da kombinacija ne vsebuje dveh enakih števil. Števila izpišite.
  Namig: Pri generiranju prvega in drugega števila z ustrezno kodo poskrbite, da se prvi števili ne ujemata.

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

- Napišite program, ki generira naključno celo število med 1 in 100000 in nato izpiše, koliko je lukenj v tem številu. Z luknjami so mišljene "luknje" v grafični podobi posameznih števk:
  - števke 0, 4, 6, 9 imajo po eno luknjo,
  - števka 8 ima dve luknji in
  - preostale števke nimajo nobene luknje.

  Primer: `Stevilo 6854 ima 4 luknje.`

Naloge iz zank

- Spodaj imaš delček programa, ki "izpiše" števila med 0 in 9. Najprej ga dopolni tako, da jih bo tudi v resnici izpisal (vstavi košček kode v program, dodaj izpis).

```
int i = 0;
while(i<10) {
 //povecaj i
 ...
 //izpisi i
```

```
    ...
  }
```

Dopolni ga tako, da:

- 10x izpiše na zaslon tvoje ime in priimek
- izpiše vsa števila med 1 in 10 (vključno z ena in 10)
- izračuna vsoto števil med 1 in 10
- izračuna vsoto števil med 1 in številom, ki ga vnese uporabnik
- izračuna vsoto vseh števil med 1 in številom, ki ga vnese uporabnik, ki dajo pri deljenju s tri ostanek dva. Taka števila tudi izpiši.
- izračuna vsoto vseh števil med 1 in številom, ki ga vnese uporabnik, ki dajo pri deljenju s tri ostanek dva in so hkrati soda. Taka števila tudi izpiši.
- izračunaj povprečno vrednost števil iz zgornjih dveh primerov.

Sestavi program VelikiZ, ki prebere pozitivno celo število n > 1 in nariše veliko črko Z velikosti n vrstic, kot je to prikazano v primeru (n = 7).

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

Dan je naslednji program:

```
static void Main(string[] args)
{
      int n = 4;
      int i = 0, j;
      while (i < n)
      {
            j = 0;
            while (j < i)
            {
                Console.Write(" ");
                j++;
            }
            Console.Write("/");
            j = 0;
            while (j < n - i)
            {
                Console.Write("*");
                j++;
            }
            i++;
            Console.WriteLine("\\");
      }
}
```

Program naj bi izpisal na ekran naslednji vzorec:

```
\**********/
 \********/
  \******/
```

```
\****/
 \**/
```

- Kaj dejansko program izpiše na ekran? Odgovor nariši čitljivo in jasno označi presledke.
- Popravi program tako, da bo pravilno delal.

Napiši program `Deljitelj.cs`, ki prebere pozitivno celo število `n` in izpiše na zaslon vse pozitivne delitelje `n`. Primer uporabe:

```
Vnesi n: 28
1
2
4
7
14
28
```

Janezek je v šoli napisal program, ki izračuna vsoto vseh sodih celih števil med 1 in 100, ki so deljiva s sedem, a ne z devet. Program je začel pisati, a se mu je zaradi pomanjkljivega znanja C# hitro ustavilo. Pomagaj mu in dopolni spodnji program:

```
int stevec = 1;
int vsota = 0;
while ( ____________________ ) {
        if(stevec%2 == 0 ____________________________ ) {
                vsota = vsota + _____________ ;
        }
        ________________________
}
```

Dopolni del programa, ki z uporabo dvojne zanke izpiše na zaslon:

```
1+1+1+1+1
2+2+2+2
3+3+3
4+4
5
```

Pri tem si pomagaj z ogrodjem:

```
int i = 0;
int j = 0;
while(i < ______) {
   ____________________
   while (j < ______) {
         ____________________
         j = j + 1;
   }
```

```
   i = i + 1;

   _____________________________
}
```

- Dan je del programa, ki naj bi izpisal prvo število večje kot 1, ki deli tako število 121 kot število 55

```
int i = 2;
while ((121 % 2 == 0) && (55 / 2 != 0))
{
   i = i + 1;
}
Console.WriteLine("121 in 55 prvo deli število " + i);
```

  - *Kaj izpiše program, ko ga poženemo? _____*
  - *Popravi ta del programa tako, da bo pravilno deloval.*

- Za dano število rečemo da je srečno, če je vsota njegovih števk deljiva s sedem. Napiši program ki ugotovi, ali je število, ki ga vnese uporabnik, srečno.

- Napiši program, ki izbera naključno celo število med –5 in 3 in izpiše, ali je pozitivno, 0 ali negativno. To počne toliko časa, dokler dvakrat zaporedoma ne izžreba negativno število.

- Kaj izpiše naslednji del programa?

```
int stevec = 15;
string niz1 = "";
string niz2 = "";
while (stevec >= 9) {
   niz1 = niz1 + "*";
   niz2 = niz2 + ((17 - stevec) / 2) + ":" + niz1 + "\n";
   stevec = stevec - 2;
}
Console.WriteLine(niz2);
```

  Kakšno vrednost ima spremenljivka števec po izvajanju zgornjega dela programa?
  Kakšno vrednost ima spremenljivka niz1 po izvajanju zgornjega dela programa?

- Sestavi program, ki prešteje, koliko števk ima dano pozitivno število. Tako ima število 23764 5 števk, število 12 2 števki, 7 eno. Program naj deluje prav tudi za negativna števila (-17 ima dve števki) in 0 (ki ima 1 števko).

- Janezek je v šoli napisal program, ki naj bi 5x izpisal niz `tralala`. Program je začel pisati, a se mu je zaradi pomanjkljivega znanja C# hitro ustavilo. Pomagaj mu in dopolni spodnji program. Manjkajoči deli so kvečjemu na označenih delih:

```
 void Main(string[] ___)
 {
        int n = 5;
        while (______);
        {
            _________ ("tralala");
            _________;
        }
  }
```

- Kaj izpiše naslednji del programa:

```
int vsota = 0;
int i = 0;
while(i < 10) {
  vsota = vsota + i;
  i = i + 1;
}
Console.WriteLine(vsota);
```

_____

- Kaj izpiše naslednji del programa:

```
int kajDelam = 0;
int bla = 0;
while(10 >= bla) {
  bla = bla + 2;
  kajDelam = kajDelam + bla;
  bla = bla + 1;
}
Console.WriteLine (bla + kajDelam);
```

_____

- Sestavi program `IzpisNiza`, ki n-krat na zaslon izpiše dani niz, vsakič v svojo vrsto.

- Numerologi računajo "osebno število" iz datuma rojstva osebe tako, da seštevajo števke rojstnega datuma, dokler ne dobijo enomestnega števila. Na primer, za rojstni datum 19.8.1985 bi dobili število 5, saj je 1+9+8+1+9+8+5 = 41, 4+1 = 5. Sestavi metodo, ki za prebrano naravno število n vrne osebno število. Tako za podatek 1981985 metoda vrne 5.

- Kolikokrat spodnja koda izpiše "`Pridno bom študiral`" ?

```
int stevec = 1;
while (stevec < 9) {
   stevec = stevec * 2;
   Console.WriteLine("Pridno bom študiral!");
}
```

- Sestavi program, ki za dano naravno število stevilo in števko stevka, ki ju vnese uporabnik, prešteje, kolikokrat se stevka pojavi v stevilo. Primer za stevilo = 12342 in stevka = 2:

```
Vpisi                                                    stevilo:231242
Vpisi                                                          stevko:2
Stevka 2 se v stevilu 231242 pojavi 3 krat!
```

- Sestavi program NarisiPiramido, ki prebere višino piramide in izpiše tako visoko piramido, sestavljeno iz *. Primer, ko je višina piramide enaka 6:

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

- Manca in Jure morata vsak dan hoditi v oddaljeno vas h kmetu po mleko. Ker jima opravilo nič kaj preveč ne diši, sta se odločila, da bosta vsak dan žrebala. Žreb bo določil, kdo izmed njiju bo nesrečnik, ki se bo opdravil na sprehod. V ta namem sta si pripravila 100 listkov s števili med 1 in 100 in jih vrgla v škatlo. Nato sta vsako jutro izvlekla en listek, pogledala njegovo številko in ga vrnila nazaj. Če je bilo število sodo, se je po mleko odpravila Manca, sicer pa Jure.

- Sestavi program, ki simulira enodnevno žrebanja in izpiše, ali bo šla danes po mleko Manca ali Jure.
- Program dopolni tako, da opravi žrebanje za cel teden vnaprej in povej, kolikokrat bo šla po mleko Manca in kolikokrat Jure.
- Denimo, da Manco zelo zanima, kolikokrat bo šla v naslednjih n dneh po mleko. Preberi n in izpiši rezultat!

Vrtnar Polde ima že od mladih nog rad številko 9. Pravzaprav jo ljubi tako zelo, da doživi napad evforije vsakič, ko naleti na Njen večkratnik.

V imenu lastnika posestva, na katerem dela Polde, morate poskrbeti, da zaposleni svoje delo opravijo čim hitreje. Prišla je jesen in po travniku je nastlana debela odeja odpadlega listja. Polde bo svoje delo opravil z največjim užitkom, če bo le število listov na travniku deljivo z devet.

- Napiši program, v katerega vnesete število listov, on pa vam izpiše "Polde je vesel!", če je to število deljivo z devet in "Polde ga gre raje pit", če število ni deljivo z 9.
- Ker sumimo, da se računalnik moti, bi radi preverili deljivost števila z 9 še na malo drugačen način. V osnovni šoli so nas naučili, da je vsota cifer večkratnikov števila 9 prav tako deljiva z 9. Napišite program, ki za poljubno (prebrano) število izračuna vsoto njegovih cifer. Nato naj izpiše "Juhu ! Polde dela zastonj!", če je vsota deljiva z 9, in "Jejhata, Polde zahteva denar !", če vsota ni deljiva z 9.
- Ker je listje z ulice poceni, boste Poldeta prelisičili. Na travnik boste dodali toliko listov, da bo Polde delal zastonj. Napišite program, v katerega boste vnesli število listov na travniku, on pa bo izračunal, koliko listov je treba dodati na travnik, da bo število listov na travniku deljivo z 9.
- Ker smo preleni, da bi liste šteli, jih bomo na travnik metali kar z lopato. Napišite program, v katerega najprej vnesete začetno število listov na travniku. Če je število deljivo z 9, naj program izpiše "Pazi, Polde z grabljami divja sem !". Če število listov na travniku ni deljivo z 9, boste morali na travnik vreči lopato listja. Program naj vas vpraša, koliko listov ste dodali na travnik. Če je število listov na travniku sedaj deljivo z 9, izpišite "Pazi, Polde z grabljami divja sem !", sicer pa naj vas program spet vpraša, koliko listov ste dodali na travnik. Program naj vas spraševuje, koliko listov ste dodali na travnik, dokler ne bo število listov na travniku deljivo z 9. Namig: uporabite zanko while.
- Skopi lastnik vas bo nadomestil z robotom, ki bo namesto vas na travnik metal listje. Predelajte prejšnji program, ki bo namesto vnosa začetno število listov (za katerega vemo, da je večje od 10000 in manjše ali enako 1000000) določil kar z uporabo naključnega števila. Tudi število listov na vsaki vrženi lopati bo določeno naključno in sicer bo med (vključno) 100 in 1000. Na koncu naj program izpiše število listov na travniku. To število mora biti seveda deljivo z 9.
- Čas je denar. Zanima nas, koliko lopat bo moral robot iz prejšnjega programa zmetati na travnik, da ga bo Polde potem z veseljem pospravil. Predelajte prejšnji program tako, da na koncu namesto števila listov izpiše, koliko lopat listja je moral robot zmetati na travnik.

Od uporabnika preberi poljubno število in izpiši vse njegove števke. Namig: uporabi celoštevilsko deljenje z deset, ostanek pri deljenju z deset in while zanko. Ko bo to narejeno za vnešeno število preveri, če je "srečno". Srečno število število je tisto, katerega vsota števk je deljiva s 7.

Sestavimo preprosto igro! Program bomo gradili po korakih, nazadnje pa bomo imeli še nekaj zabave.

- Najprej od uporabnika preberi število. Glej, da ga nikamor ne izpišeš, ampak si ga samo zapomniš!
- S pomočjo zanke while poskrbi, da bo število v mejah med 1 in 100 (vključno). Namig: če uporabnik vnese napačno število, potem mora pogoj v zanki while poskrbeti, da ga bo program ponovno vprašal za vpis števila.
- Dopolni program tako, da za pravilnim vnosom prvega števila zahteva še vnos drugega. Shrani ga v spremenljivko.

- Dopolni program tako, da mora uporabnik drugo število vnašati toliko časa, dokler ni enako prvemu. Namig: zanka while.
- Programu dodaj tudi spremenljivko stevec, v katero shrani število vnosov drugega števila. Namig: vsakič, ko prebereš drugo število, moraš spremenljivko stevec povečati za ena.
- Po branju drugega števila izpiši, ali je:
  - premajhno (torej manjše od prvega)
  - preveliko (torej večje od prvega)
  - enako prvemu - v tem primeru izpiši tudi, v katerem poizkusu je bilo število uganjeno
- Sedaj pa se igra laho začne! Najprej vnesi v program prvo število (pazi, da ga boš videl le ti!), nato pa poprosi svojega soigralca, naj ga poskusi uganiti. Ko ga ugane, se zamenjajta. Tisti, ki uspe soigralčevo število uganiti najprej, je zmagovalec.
- Včasih pa se zgodi, da soigralca ne moreš dobiti. Zato dopolni program tako, da bo prvo število izžrebal računalnik.

Dan je program

```
 public static void Main(string[] m)
 {
     int k = 20;
     while (k > 0)
     {
       Console.WriteLine(k);
       while (k > 1)
       {
          k = k / 2;
       }
       k = k - 1;
     }
     if (k < 0) {
     Console.WriteLine(k);
}
```

Kaj izpiše ZADNJI klic metode WriteLine, ki se izvede?
Dopolni program tako, da izpiše niz aha-aha-aha-aha-aha-aha. Ni nujno, da moraš popisati vse črte.

```
string izpis = "_________________";
int nivo = 11;
while (nivo <= 19)
{
   ____________________________________;
   izpis = izpis + "_________________";
   ____________________________________;
}
Console.WriteLine(izpis);
```

Kaj izpiše naslednji del programa (napovej, ne da poženeš program!)

```
int n = 30;
int f = 2, i = 0;
string odg = "";
while (f <= n)
{
   if (n % f == 0)
   {
      i = i + 1;
   }
```

```
   odg = odg + i + ". = " + f + "\n";
   f = f + 1;
}
Console.WriteLine(odg);
```

Za vsako vrstico spodnjega dela programa izpiši, kakšne vrednosti imajo spremenljivke, ki nastopajo v njem! Vpisuj le v tiste vrstice, ki se dejansko izvedejo! Če se določena vrstica izvede večkrat, vrednosti vpisuj zaporedoma in jih loči z vejico. Če se vrstica ne izvede, to označi v stolpcu Se ne izvede.

	Se ne izvede	n	b	m	d
`int n = 10, b = 3, m = 2, d = 4;`					
`n = n + b - (int)(3.5*d);`					
`n = (n + m) % b;`					
`if( n % 2 == 0) {`					
`m = n;`					
`n = m;`					
`} else {`					
`m = n + d;`					
`n = b * 100;`					
`}`					
`d = 0;`					
`while (n % m == 0) {`					
`d = d + 1;`					
`n = n / m;`					
`}`					
`n = n + 1;`					
`d = d / 3;`					