Programiranje – III razred

Funkcije – 2. deo

Metode u C#

Prilikom dizajniranja većine aplikacija vrši se njihova podela na funkcionalne jedinice

- prednost:
  - jedinice se mogu koristiti više puta u aplikaciji
  - doprinosi boljoj struktuiranosti programa.
- U C#-u aplikacija se struktuira pomoću klasa.
- Sve metode moraju pripadati nekoj klasi ili zapisu.
- Metoda je član klase koji izvršava odereñenu akciju.
  - skup C# naredbi koje su objedinjene i imenovane.
- Metode mogu biti:
  - metode instance
    - koje rade nad određenom instancom klase
  - statičke metode
    - koje obezbeđuju opštiju funkcionalnost, tj. ne zahtevaju da postoji instanca klase.

Kreiranje metoda u C#

Neophodno je navesti sledeće:

- **ime**
  - metoda ne može imati isto ime kao i bilo koja promenljiva, konstanta ili neki drugi član koji je deklarisan unutar klase
  - za imenovanje metoda važe ista pravila kao i za imenovanje promenljivih.
- **lista parametara**
  - navodi se unutar zagrada koje moraju biti navedene i ukoliko nema parametara
- **telo metode**
  - navodi se unutar vitičastih zagrada, čak i ukoliko se telo sastoji samo od jedne naredbe.
- **tip povratne vrednosti**
  - navodi se ispred imena metode
  - ukoliko metoda ne vraća vrednost navodi se void.

Kreiranje metoda u C#

```
TipPovratneVrednosti ImeMetode (ListaParametara)
{
TeloMetode
...
...
...
Return povratnaVrednost;
}
```

Metoda koja ima samo zaglavlje naziva se operacija.

Pozivanje metoda u C#

Nakon definisanja metode, ona se može pozvati iz same klase ili iz neke druge klase.

- Pozivanje metoda iz same klase
  - koristi se ime metode iza koje sledi lista parametara unutar zagrada

```
ImeMetode();
```

- Pozivanje metoda iz drugih klasa:
  - prvo se mora navesti ime klase čija se metoda poziva

```
ImeKlase.ImeMetode();
```

  - kada ne bi bilo navedeno ime klase kompajler bi tražio datu metodu unutar klase iz koje se poziva, i ukoliko ne bi postojala javio bi gresku.
  - metoda se mora deklarisati kao javna
    - ukoliko metoda nije deklarisana kao javna, ona je po default-u privatna za tu klasu, te se ona ne može pozvati iz drugih klasa.
- Metode se takođe mogu pozivati i iz drugih metoda.

Lokalne promenljive

Svaka metoda ima svoj skup lokalnih promenljivih.

- Ove promenljive se mogu koristiti samo unutar metode u kojoj su deklarisani.
- Može im se dodeliti početna vrednost prilikom deklaracije.

```
void Primer {
int x;
int y = 0; // inicijalizacija
... }
```

- Promenljive deklarisane unutar jedne metode su potpuno nezavisne od promenljivih koje su deklarisane unutar drugih metoda, čak iako imaju ista imena.
- Memorija u kojoj se čuvaju lokalne promenljive se alocira svaki put kada se pozove metoda, i oslobađa nakon izvršavanja metode.
  - to znači da se bilo koje vrednosti, koje su čuvaju u ovim promenljivima, neće zadržati od jednog poziva metode do drugog.

Zajedničke promenljive

```
class LosPrimer {
void Init()
{ int brojac = 0;}
void Prebroj()
{ int brojac;
++brojac; }
...
{ Init();
Prebroj();
}
}
```

- Ovaj program ne bi mogao da se iskompajlira zbog toga što:
  - promenljiva brojač u metodi **Init** nije ista kao i promenljiva brojac u **Prebroj**.
  - bez obzira koliko puta se pozove metoda Prebroj, vrednost brojača se gubi nakon što se metoda izvrši.

Zajedničke promenljive

Ovaj problem se može rešiti tako što će se promenljiva brojač deklarisati na nivou klase, a ne metode.

```
class DobarPrimer {
int brojac;
void Init()
{ brojac = 0;}
void Prebroj()
{ ++brojac; }
...
{ Init();
Prebroj();
}
}
```

- Promenljivu **brojač** dele sve metode date klase.

Naredba **return**

- Može se koristiti da bi se izvršavanje momentalno vratilo iz metode na mesto poziva.
  - ukoliko se izostavi, izvršavanje se vraća na mesto poziva nakon izvršenja poslednje naredbe u metodi.

```
void Primer()
{ int broj = 5;
if (broj < 10)
return;
Console.WriteLine("Dvocifren broj"); }
```

- Može se navesti i više return naredbi unutar jedne metode.

```
void Primer()
{ int broj = 5;
if (broj < 10)
return;
return; }
```

- Ukoliko je u definiciji metode naveden tip podatka koji se vraća mora se dodati bar jedna return naredba kojom se vraća vrednost iz metode.

Povratne vrednosti

- Da bi se vratila vrednost neophodno je:
  - prilikom deklaracije metode definisati tip povratne vrednosti
    - umesto ključne reči void navodi se tip povratne vrednosti.
  - dodati return naredbu u metodu uz vrednost (ili izraz) koji treba vratiti
    - ovim će se postaviti povratna vrednost, momentalno prekinuti izvršavanje tekuće metode i vratiti izraz kao povratna vrednost metode.
  - prilikom poziva metode prihvatiti vrednost

```
class Primer {
int Brojac()
{ int a = 0;
return ++a; }
...
{ int x;
x = Brojac();}
}
```

- Return naredbom metoda može vratiti isključivo jednu vrednost.

Prenos parametara

- Return naredbom metoda može vratiti isključivo jednu vrednost.
- Ukoliko je neophodno vratiti više od jedne vrednosti:
  - može se vratiti referenca na niz, klasu ili zapis (koji mogu da sadrže više vrednosti)
  - mogu se koristiti mehanizmi za prenos parametara.
- Mehanizmi za prenos parametara:

Prenos preko **vrednosti**	Ulazni parametri	Podaci se mogu preneti u metodu, ali se ne mogu preneti iz nje
Prenos preko **referenci**	Ulazni i izlazni parametri	Podaci se mogu preneti u metodu i iz metode
Prenos preko **izlaza**	Izlazni parametri	Podaci se mogu preneti iz metode, ali se ne mogu preneti u nju

Mehanizmi za prenos parametara

U principu, argumenti se u metodu mogu preneti:

- **preko referenci**
  - metoda referencira originalne promenljive
  - na originalne promenljive utiče bilo kakva izmena unutar pozvane metode
- **preko vrednosti**
  - metoda referencira kopije originalnih promenljivih
  - originalne promenljive se ne menjaju bez obira na to kakve izmene nad njom vrši pozvana metoda (menjaju se samo kopije).

- U C#-u se svi parametri prenose po vrednosti, ukoliko se ne navede drugačije.

- **NAPOMENA:**
  - kod stringova, iako su referentnog tipa, bilo kakva izmena nad stringom u metodi neće uticati na originalni string.
- Sve promenljive moraju biti inicijalizovane pre nego što se mogu preneti u metodu, bez obzira na to da li se prenose preko vrednosti ili reference.

Prenos preko vrednosti

- Vrednosni parametar se navodi tako što se navede ime tipa, iza koga sledi ime promenljive.
- Kada se pozove metoda zauzima se nova memorijska lokacija za svaki vrednosni parametar.
  - vrednosti odgovarajućih izraza se kopiraju u njih.
- Unutar metode se može promeniti vrednost parametra, a da to ne utiče ni na jednu promenljivu izvan metode.

```
void Brojac(int x)
{ x++; }
...
{ int k = 6;
Brojac(k);
Console.WriteLine(k); } // 6, a ne 7
```

k=6

x=7

- Prosleđenja vrednost mora biti istog ili kompatibilnog tipa sa tipom u deklaraciji parametra.
  - izrazi koji su navedeni za svaki vrednosni parametar moraju biti istog tipa kao i tip koji je naveden u njegovoj deklaraciji, ili tipa koji se implicitno može konvertovati u taj tip.

Prenos preko reference

- Referentni parametar predstavlja referencu na memorijsku lokaciju.
- Za razliku od vrednosnih parametara, oni ne zauzimaju novu memorijsku lokaciju, već pokazuju na istu onu lokaciju na koju pokazuje promenljiva prosleđena pozivom metode.
- Referentni parametri se deklarišu navođenjem ključne reči ref ispred tipa.

```
void Primer(ref int x, long y)
{ ... }
```

- Ključna reč ref se odnosi samo na parametar ispred koga je navedena, a ne na celu listu parametara.

```
void Primer(ref int x, ref long y)
{ ... }
```

- Prilikom poziva metode takođe se mora navesti ključna reč ref ispred promenljive koja se prosleđuje.

```
int p =10;
long q = 15;
Primer(ref p, ref q);                    Primer(p,q) // Greška
```

Prenos preko reference

- Vrednost koja se prosleđuje u pozivu metode mora biti istog tipa, kao i tip naveden u definiciji metode, i mora biti promenljiva (ne može biti konstanta ili vrednost izraza).

```
void Primer(ref long x)
{ x++; }
...
int y = 10;
Primer(ref y); // Greška
```

- Referentnom parametru se mora dodeliti vrednost pre poziva metode.

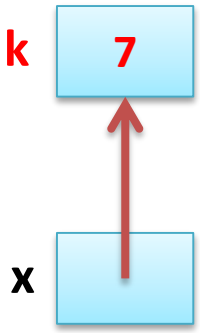
```
int y = 10;
Primer(ref y);

int y;
Primer(ref y); // Greška
```

Prenos preko reference

- Ukoliko se promeni vrednost referentnog prarametra promeniće se i promenljiva koja je prosleđena pozivom metode.
  - obe predstavljaju referencu na istu lokaciju u memoriji.

```
void Primer(ref int x)
{ x++; }
...
{ int k = 6;
Primer(ref k);
Console.WriteLine(k);      // 7 }
```



Izlazni parametri

- Koriste se kada je potrebno vrati više od jedne vrednosti iz metode
  - mogu samo da prosleđuju podatke iz metode (ne mogu ih proslediti u metodu).
- Kao i referentni parametri, predstavljaju referencu na memorijsku lokaciju, koja je prosleđena u pozivu metode
  - izlazne vrednosti se dodeljuju promenljivima koje su u metodu prenesene preko reference.
- Izlazni parametri se deklarišu navođenjem ključne reči out ispred tipa

```
void Primer(out int x, long y)
{ ... }
```

- Ključna reč out se odnosi samo na parametar ispred koga je navedena, a ne na celu listu parametara.

```
void Primer(out int x, out long y)
{ ... }
```

- Prilikom poziva metode mora se navesti ključna reč out ispred promenljive koja se prosleđuje.

```
int p;
Primer(out p);                    Primer(p) // Greška
```

Izlazni parametri

- C# zahteva da sve promenljive budu inicijalizovane nekom početnom vrednošću pre nego što se mogu referencirati.
- Međutim, početne vrednosti promenljivih koje se u ovom slučaju prenose su nebitne
    - metoda ih možda nikad neće ni koristiti, a u metodi će one biti pregažene.
- Način da se zaobiđe ovaj zahtev C# kompajlera je da se ispred ulaznih parametara metode navede ključna reč out
    - tada promenljivoj koja je obezbeđena za izlazni parametar ne mora biti dodeljena vrednost pre poziva metode
    - tretira se kao neinicijalizovana lokalna promenljiva.
- Izlazne parametre je pogodno koristiti kada je potrebno vratiti vrednost iz metode pomoću parametra, bez dodeljivanja početne vrednosti tom parametru.
- Izlaznom parametru je neophodno dodeliti vrednost unutar metode.
    - u suprotnom metoda se neće iskompajlirati.
- Pošto se promenljiva prenosi preko reference, sve promene koje metoda izvrši nad tom promenljivom ostaju i nakon povratka kontrole na mesto poziva.

Preporuke za prenos parametara

- Prilikom izbora načina prenosa parametara treba voditi računa o mehanizmu prenosa i njegovoj efikasnosti.
- Mehanizmi prenosa
  - prenos preko vrednosti je najčešći
  - vrednosne parametre ne treba koristiti ukoliko je potrebno proslediti informacije iz metode
  - ukoliko je potrebno proslediti podatke izvan metoda mogu se koristiti return naredba, referentni parametri ili izlazni parametri
  - ukoliko je potrebno vratiti samo jednu vrednost koristiti naredbu return sa povratnom vrednošću
  - ukoliko je potrebno vratiti više vrednosti koristiti ref i/ili out parametre
  - varijantu ref treba koristiti samo kada podatke treba proslediti u oba pravca
- Efikasnost
  - prenos preko vrednosti je generalno najefikasniji
    - jednostavni tipovi (kao što su int i long) se najefikasnije prenose preko vrednosti.
  - Za kompleksne tipove podataka, prenos preko referenci je efikasniji zbog velike količine podataka koju bi trebalo kopirati ukoliko bi se prenosile preko vrednosti.

Promenljivi broj parametara

- Ponekad je korisno da metoda može da primi promenljiv broj parametara.
- Koristi se ključna reč **params** da bi se navela lista prarmetara promenljive dužine.
- OGRANIČENJA:
  - može se navesti samo jedan params parametar po metodi
  - on se mora navesti na kraju liste parametara
  - deklariše se kao jednodimenizionalni niz.

```
int Primer(params int[] v)
{ int ukupno, i;
for(i=0, ukupno=0; i<v.Length; i++)
ukupno += v[i];
return ukupno; }
```

- Na osnovu svojstva niza Length može se odrediti broj prosleđenih parametara.
- Sve vrednosti koje se navode moraju biti istog tipa pošto je tip parametra params uvek niz.

Promenljivi broj parametara

- Pri pozivu metode vrednosti se mogu proslediti params parametru na dva načina (u oba slučaja parametar se tretira kao niz):
  - preko liste elemenata, odvojenih zarezima (lista može biti prazna)

```
int x;
x = Primer(63, 21, 84);
```

  - preko niza

```
int x;
x = Primer(new x[] {63, 21, 84});
```

- Kod params parametra, pravi se kopija podataka, pa promene vrednosti unutar metode neće uticati na vrednosti izvan metode.
- Uvek ih treba prenositi preko vrednosti.

Programiranje – III razred

Rekurzivne metode

Rekurzija

- **Rekurzija** (лат. recursio, recursion od recurrere: vraćanje) u matematici i informatici označava postupak ili funkciju koji u svojoj definiciji koriste sami sebe. Drugim rečima, ukoliko neki postupak zahteva da delovi problema koje je razdvojio od drugih bivaju nezavisno podvrgnuti istom tom postupku, taj postupak je rekurzivan[1].

- Metoda može samu sebe da poziva
  - direktno
  - indirektno
- Ova mogućnost se naziva rekurzija.
- Korisne su za manipulisanje složenim strukturama podataka kao što su liste ili stabla.
- Metode u C#-u mogu da budu međusobno rekurzivne
  - dozvoljena je situacija u kojoj metoda A može da zove metodu B, a i metoda B može da zove metodu A.
- Rekurzivna metoda mora imati uslov izlaza koji obezbeđuje da se iz metode može vratiti bez daljih poziva.

1: http://sr.wikipedia.org/sr-el/рекурзија

Vrednost faktorijela

Faktorijel je matematička funkcija koja se u edukativne svrhe često spominje u kontekstu rekurzije. Faktorijel prirodnog broja n je proizvod njega samog i svih prirodnih brojeva koji su manji od njega:

$$n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot \ldots \cdot (n-2)(n-1)n$$

```
int fakt(int n)
{
if(n < 2)    // Ukoliko je broj manji od 2
{
return 1; // vratiti 1
}
else    // u suprotnom
{
return n*fakt(n-1); // vratiti trenutni broj pomnožen sa faktorijelom broja za
//jedan manjeg od njega
}
```

Vrednost faktorijela

```
fakt(5) = 5 · fakt(4)
        = 5 · (4 · fakt(3))
        = 5 · (4 · (3 · fakt(2)))
        = 5 · (4 · (3 · (2 · fakt(1))))
        = 5 · (4 · (3 · (2 · 1)))

        = 5 · (4 · (3 · (2 · 1)))
        = 5 · (4 · (3 · 2))
        = 5 · (4 · 6)
        = 5 · 24
        = 120
```

$$n! = \begin{cases} 1, & n = 0 \\ n \times (n-1)!, & n > 0 \end{cases}$$

Faktorijel – primer koda

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace RekurzijaFaktorijel
{
    class Program
    {
        int unetiBroj = 0 ;
        int faktorijel = 1;
        int lokalnaVarijabla = 0;
        Program()
        {
            unetiBroj = unosBroja(); // Unos broja preko konzole
            // Poziv rekrzivne metode
            lokalnaVarijabla = unetiBroj;
            faktorijel = izracunajFaktorijel(lokalnaVarijabla);
           // Ispis
            ispisiRezultat();
         }
        private void ispisiRezultat()
        {
            Console.Clear();
            Console.WriteLine("*****************************");
            Console.WriteLine("* IZRAČUNAVANJE FAKTORIJELA *");
            Console.WriteLine("*****************************\n\n");
            Console.WriteLine("Faktorijel broja {0} je: {1} ",unetiBroj,faktorijel);
            Console.WriteLine("\n\n\n\nPritisnite bilo koji taster za kraj...");
            Console.ReadLine();
        }
```

Faktorijel – primer koda – 2. deo

```
private int izracunajFaktorijel(int lokalnaVarijabla)
        {
            int rezultat;
            if(lokalnaVarijabla == 1)
            {
                return 1;
            }
            rezultat = izracunajFaktorijel(lokalnaVarijabla - 1) * lokalnaVarijabla;
            return rezultat;
        }
        private int unosBroja()
        {
            Console.Write("Unesite broj čiji faktorijel tražite: ");
            string x = Console.ReadLine();
            return Convert.ToInt32(x);
        }
        static void Main(string[] args)
        {
            Program f = new Program();
        }
    }
}
```

Fibonačijev niz

Fibonačijev niz je matematički niz primećen u mnogim fizičkim, hemijskim i biološkim pojavama. Ime je dobio po italijanskom matematičaru Fibonačiju.

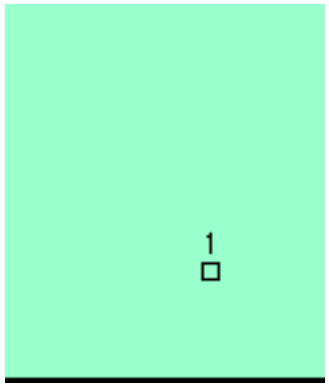
Predstavlja niz brojeva u kome zbir prethodna dva broja u nizu daju vrednost narednog člana niza. Indeksiranje članova ovog niza počinje od nule a prva dva člana su mu 0 i 1.

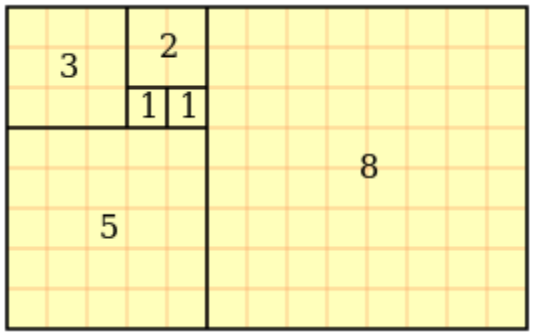
$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2}, \quad n \geq 2
\end{aligned}
$$

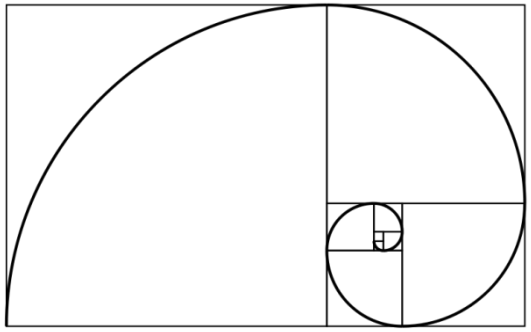
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

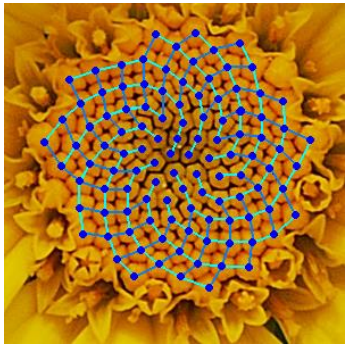
Fibonačijev niz

1. Napisati rekurzivni algoritam za pronalaženje n elemenata Fibonačijevog niza, gde se n unosi preko tastature. Obezbediti ispis svih n elemenata niza preko konzole.









Fibonačijev niz

1. Napisati rekurzivni algoritam za pronalaženje n elemenata Fibonačijevog niza, gde se n unosi preko tastature. Obezbediti ispis svih n elemenata niza preko konzole.

Uputstvo:

```
...
long Fibonacci(long n)
{ if(n <= 2) // uslov izlaza
return 1;
else
return Fibonacci(n-1) + Fibonacci(n-2);
}
...
```

Fibonačijev niz - rešenje

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FibonačijevNiz
{
    class Program
    {
        int elemenata = 0; // Željeni broj elemenata za prikaz
        Program()
        {
            Console.Write("Unesite broj elemenata Fibonačijevog niza za prikaz: ");
            elemenata = Convert.ToInt32(Console.ReadLine());
            Console.Clear();
            for (int i = 0; i < elemenata; i++ )
            {
                Console.WriteLine(Fibonaci(i));
            }
            Console.ReadLine();
        }
        static int Fibonaci(int x)
        {
            if (x <= 1) { return 1; }
            return Fibonaci(x - 1) + Fibonaci(x-2);
        }
        static void Main(string[] args)
        {
            Program fn = new Program();
      }
   }
}
```