Osnove programiranja

Funkcije - Metode

Sadržaj

- Funkcije
- Metode
  - Definisanje
  - Pozivanje
- Povratne vrednosti
- Parametri

Funkcije - prednosti

- Blok koda koji može biti izvršen na bilo kom mestu u aplikaciji i može se iznova upotrebljavati.
- Kod čine čitljivijim i mogu se koristiti za grupisanje međusobno povezanog koda.
- Smanjuje glavni deo koda u aplikaciji - sporedni delovi se izvršavaju izvan glavnog.
- Višenamenski kod - mogu se izvršiti iste operacije nad različitim podacima.

Funkcije

- Može se snabdeti potrebnim podacima preko **parametara**
- Rezultati funkcija se mogu dobiti u obliku **povratnih vrednosti**.
  - Primer određivanje maksimalne vrednosti niza brojeva - parametar bi bio niz koji se pretražuje, dok bi povratna vrednost bila maksimalna vrednost unutar niza.
- Funkcije koje su deo klase poznate su kao **metode**.

C# metode

- Metoda - grupa iskaza koji zajedno obavljaju neki zadatak.
- C # program ima barem jednu klasu (npr. **class Program**) sa metodom po imenu **Main**:
  - **static** (nije potrebno kreirati instancu unutar klase u kojoj je metoda definisana)
  - **void** – nema povratnu vrednost
- Korišćenje metode podrazumeva:
  - definisanje i
  - pozivanje metode.

Definisanje metoda u C#-u

- Definisanjem metoda u osnovi se deklariše njegova struktura.

```
<Specifikator pristupa> <Povratni tip> <ImeMetode>(Lista parametara)
{
 Telo metode
}
```

Vidljivost metode ili promeljive iz druge klase

To je vrednost tipa podatka koju vraća metoda. Ako metoda ne vraća nikakvu vrednost onda je ona tip **void**.

Jedinstveno ime koje je 'case sensitive' PascalCase

Parametri se stavljaju unutar zagrada i koriste se za prosleđivanje ili prihvatanje podataka iz metode

Deklarisanje i pozivanje metode

```
static void Ispisi()
{
Console. WriteLine("Pozdrav");
}
static void Main (string[ ] args)
{
   Ispisi();
}
```

Definisanje metode

Pozivanje metode

Redosled metoda nije bitan

```
static void Main (string[ ] args)
{
   Ispisi();
Console.WriteLine("Pozdrav u Main metodi");
}
static void Ispisi()
{
Console. WriteLine(“Pozdrav”);
}
```

Pozivanje metode

Definisanje metode

Lokalne promenljive metode

- Promenljive deklarisane unutar jedne metode su **lokalne promenljive** i *ne mogu* se videti i koristiti u telu neke *druge metode* čak iako imaju ista imena.
- Svaka metoda ima svoj skup lokalnih promenljivih.
- Lokalnoj promenljivoj se može dodeliti početna vrednost prilikom deklaracije.

Lokalne promenljive

```
int Metoda1(int x, int y)
{
      int x;
                        Deklaracija lokalnih promenljivih
      int y;
      …
}
static void Metoda2(int x, int y,int z)
{
      x=1;
                        Inicijalizacija lokalnih promenljivih
      y=2;

      z=3;
      …
}
```

Opseg važenja promenljive

- Memorija u kojoj se čuvaju lokalne promenljive se alocira svaki put kada se pozove metoda i oslobađa se nakon izvršavanja metode.
- To znači da se bilo koje vrednosti, koje su čuvaju u ovim promenljivima, **neće** zadržati nakon što se jednom pozvana metoda ponovo pozove.
- Zagrade { } koje definišu telo metode istovremeno označavaju i opseg važenja svih promenljivih deklarisanih u toj metodi.

Primer

```
static void Main(string[] args)
        {
            int x = 1;
            Write();
            Console.WriteLine(x);
        }
         static void Write()
            {
            int x = 2;
            Console.WriteLine(x);
            }
```

SM1

Opseg važenja promenljive

```
static void Main(string[] args)
{
    string mojString="Definisan je u Main funkciji";
    Write()
}
1 reference
static void Write()
{
  Console.WriteLine("String je...!", mojString);
}
```

The variable 'mojString' is assigned but its value is never used

The name 'mojString' does not exist in the current context

```
static void Main(string[] args)
{
    string mojString="String koji je definisan u Main funkciji";
    Write();
    Console.WriteLine(mojString);
    Console.ReadKey();

}
1 reference
static void Write()
{
   Console.WriteLine(mojString);
}
```

The name 'mojString' does not exist in the current context

SM1

```
static void Main(string[] args)
        {
            string mojString="String koji je definisan u Main funkciji";
            Write();
         //Console.WriteLine(mojString);

        }
        static void Write()
        {
Console.WriteLine(mojString);
        }
```

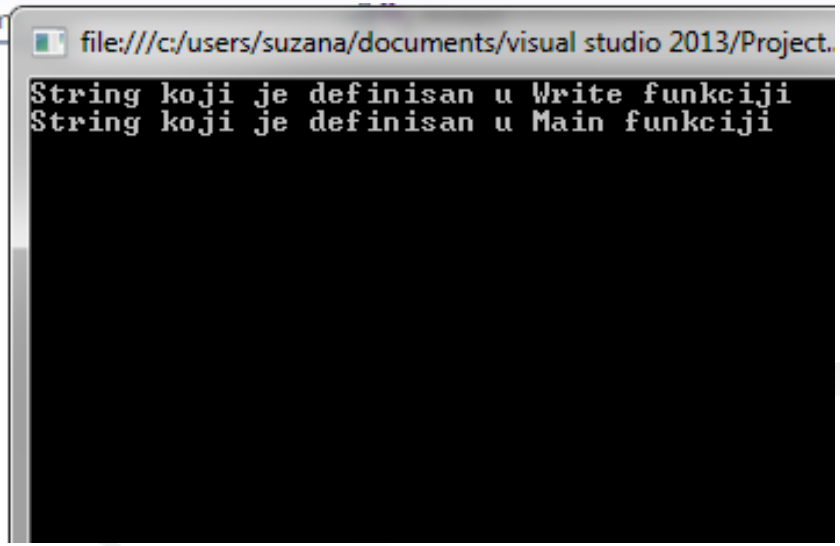
Suzana Marković; 29.11.2016.

Opseg važenja promenljive

eApplication3 ▾ ConsoleApplication3.Program

```
0 references
static void Main(string[] args)
{
    string mojString="String koji je definisan u Main funkciji";
    Write();
    Console.WriteLine(mojString);
    Console.ReadKey();

}
1 reference
static void Write()
{
    string mojString = "String koji je definisan u Write funkciji";
    Console.WriteLine(mojString);
}
```



Promenljive mojString u metodama Main i Write su lokalne pa su i različite!

S4

```
static void Main(string[] args)
        {
            string mojString="String koji je definisan u Main funkciji";
            Write();
Console.WriteLine(mojString);
Console.ReadKey();

        }
        static void Write()
        {
string mojString = "String koji je definisan u Write funkciji";
            Console.WriteLine(mojString);
        }
```

Suzana; 7.12.2014.

Povratne vrednosti (1)

- Najjednostavniji način za razmenu podataka sa metodom jeste korišćenje povratne vrednosti.
- Metoda koja ima povratnu vrednost izračunava je na isti način kao što se unutar izraza izračunava vrednost promenljive.
- Povratne vrednosti, kao i promenljive, imaju svoj tip.

Povratne vrednosti (2)

- Kada metoda vraća vrednost onda je potrebno:
  - navesti tip povratne vrednosti u njenoj deklaraciji, **umesto** ključne reči **void.**
  - koristiti ključnu reč **return** da bi se metoda završila i prenela povratna vrednost u pozivajući kod.
  - Jedino ograničenje je u tome da <PovratniTip> mora biti vrednost koja je tipa <PovratnaVrednost>, ili može biti implicitno konvertovana u taj tip.

```
Static <PovratniTip> <ImeMetode>()
{
.....
.....
return < PovratnaVrednost >;
}
```

Parametri (1)

- Lista parametara u deklaraciji metode jesu njeni argumenti - formalni parametri metode.
- Oni su po svojoj prirodi lokalni i nisu vidljivi van granica metode.
- Oni nemaju konkretnu vrednost (promenljive), već samo ukazuju na tip vrednosti i broj argumenata metode.
- Lista parametara može biti prazna, pri čemu su male zagrade obavezne.

Parametri (2)

- **Stvarni parametri** su konkretne vrednosti koje se zadaju kod poziva funkcije.
- To su promenljive koje prihvataju vrednosti argumenata prosleđenih metodi u trenutku njenog pozivanja.
- Podrazumeva se sledeći kod:

```
Static <povratniTip> <ImeMetode>(<paramTip> <paramIme>, ...)
{
.....
return <povratnaVrednost>;
}
```

```
…double a=5.3, b=1.0;
static double product (double param1, double
param2)
{
return param1 * param2;
}
```

SM3

Primer 1

```
static <povratniTip><ImeMetode>(<paramTip> <paramIme>, ...)
{
.....
return <returnValue>;
}
```

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

        int a = 25;
        int rezultat;
            rezultat = kvadriraj(a);
        Console.WriteLine("Rezultat je: {0}", rezultat);
        Console.ReadKey();
    }
     static int kvadriraj(int a )
    {
        int rezultat;
        rezultat = a *a ;
        return rezultat;
    }
```

Poziv metode
Stvarni parametar (argument metode)

Formalni parametar

Telo metode

SM3

```
while(true)
        {
        Console.WriteLine("unesite broj a:");
        int a = int.Parse(Console.ReadLine());
        int rez;
            rez = kvadriraj(a);
        Console.WriteLine("Rezultat je: {0}", rez);
        Console.ReadKey();
        }
    }
     static int kvadriraj( int a )
    {
       return  a *a ;
    }
```

Suzana Marković; 27.11.2017.

Primer 1 - modifikovan

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

    {
        Console.WriteLine("Unesi a:");
        int a = int.Parse(Console.ReadLine());
        int rezultat;
        rezultat = kvadriraj(a);
        Console.WriteLine("Rezultat je: {0}", rezultat);
    }
    static int kvadriraj(int x)
    {

        return x * x;
    }
```

Poziv metode

Stvarni parametar

Formalni parametar

Telo metode

Formalni i stvarni parametri

```
int Metoda1(int x, int y)
    {
      y=f(x)
      ...
    }
    static void Main()
    {
      a=1;
      b=2;
     Metoda1 (a,b)
          ...
    }
```

⇒ Formalni parametri

⇒ Stvarni parametri

Stvarni parametri (argumenti funkcije) moraju odgovarati formalnim parametrima po **broju, redosledu i tipu** parametara, ali NE moraju po nazivu.

Formalni i stvarni parametri

```
DEKLARACIJA: int max(int a, int b, int c)
{...}
LOŠ POZIV: max(15, 6);
```

```
DEKLARACIJA: string student(string ime, string prezime)
{...}
LOŠ POZIV: student("Jovanović", "Ana");
```

```
DEKLARACIJA: int proizvod(int a, int b)
{ return a*b; }
LOŠ POZIV: proizvod("Joca", „Mika");
```

Primer 2

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

          int a = 7;
          int b = 9;
          int rezultat;
              rezultat = zbir(a, b);
          Console.WriteLine("Rezultat je: {0}", rezultat);
          Console.ReadKey();
       }
        static int zbir( int x, int y )
      {
          int s;
          s= x + y;
          return s;
      }
```

return x+y;

S5

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

          int c = 7;
          int d = 9;
          int rezultat;
              rezultat = zbir(c, d);
          Console.WriteLine("Rezultat je: {0}", rezultat);
          Console.ReadKey();
       }
        static int zbir( int a, int b )
      {
          int s;
          s= a + b;
          return s;
      }
```

Suzana; 7.12.2014.

Primer 3 – Suma n prvih brojeva

```
static void Main(string[] args)
        {
            int n;
            int zbir;
            Console.WriteLine("Unesi n:");
            n = int.Parse(Console.ReadLine());
            zbir = suma(n);
            Console.WriteLine("Rezultat je: {0}", zbir);
        }
static int suma(int n)
        {
            int zbir = 0;
            for (int i = 0; i <= n; i++)
                zbir = zbir + i;
            return zbir;
        }
```

Poziv metode

Inicijalizacija lok.promenljive

Deklarisanje metode

Primer 4

```
static void Main(string[] args)
        {
            Console.Write("a=");
            int a = int.Parse(Console.ReadLine());
            Console.Write("\nb=");
            int b = int.Parse(Console.ReadLine());
            int veci;
            veci = NadjiMax(a, b);
            Console.WriteLine("\nVeći broj je: {0}", veci);
            Console.ReadKey();
        }
static int NadjiMax(int broj1, int broj2)
        {
            if (broj1 > broj2)
                return broj1;
            else
                return broj2;
        }
```

SM4

Primer 5

```
static void Main(string[] args)
        {
            int[] mojNiz = { 1, 8, 3, 6, 2, 5, 9, 3, 0, 2 };
            int maxProm = MaxVrednost(mojNiz);
            Console.Write("Maksimalna vrednost u nizu mojNiz je
{0} ", maxProm);
        }
static int MaxVrednost(int[] niz)
        {
            int MaxProm = niz[0];
            for (int i = 1; i < niz.Length; i++)
            {
                if (niz[i] > MaxProm)
                    MaxProm = niz[i];
            }
            return MaxProm;
        }
```

SM4 http://www.edusoft.matf.bg.ac.rs/csharp/Verzija2012/klasa2.html

Suzana Marković; 1.5.2019.

Primer 6

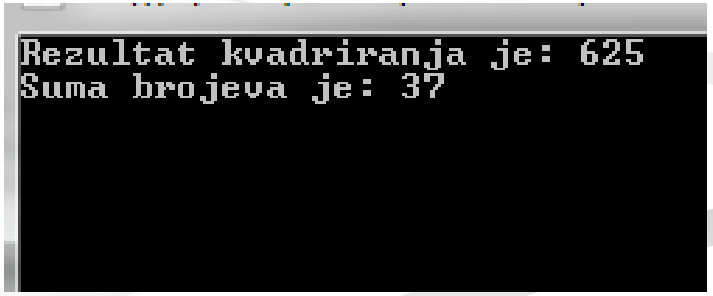
```
static void Main(string[] args)
        {
            Console.Write("Unesite neki broj:");
            int a = int.Parse(Console.ReadLine());
            int fakt = Faktorijel(a);
            Console.WriteLine( a + "! = " + fakt);
        }

static int Faktorijel(int broj)
        {
            int faktor = 1;
            for (int i = 2; i <= broj; i++)
            {
                faktor *= i;
            }
            return faktor;
        }
```

Pozivanje više metoda

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

    int a = 25, b=12;
    int rezultat;
    rezultat = kvadriraj(a);
    Console.WriteLine("Rezultat kvadriranja je: {0}", rezultat);
    int suma = zbir(a, b);
    Console.WriteLine("Suma brojeva je: {0}", suma);
    Console.ReadKey();
}
1 reference
static int kvadriraj(int a)
{
    int rez;
    rez = a * a;
    return rez;
}
1 reference
static int zbir(int a, int b)
{
    int s;
    s = a + b;
    return s;
}
```



Pozivanje više metoda i parametri različitog tipa

```
static void Main(string[] args)
        {
            int a = 7;
            int b = 9;
            int rezultat1;
            double rezultat2;
                rezultat1 = zbir(a, b);
                rezultat2 = kolic(a, b);
            Console.WriteLine("Rezultatati su: {0}, {1}", rezultat1,rezultat2 );
            Console.ReadKey();
        }
static int zbir(int x, int y)
        {
            return x+y;
        }
static double kolic(int x, double y)
         {
            return x/y;
         }
```

Pozivanje više metoda

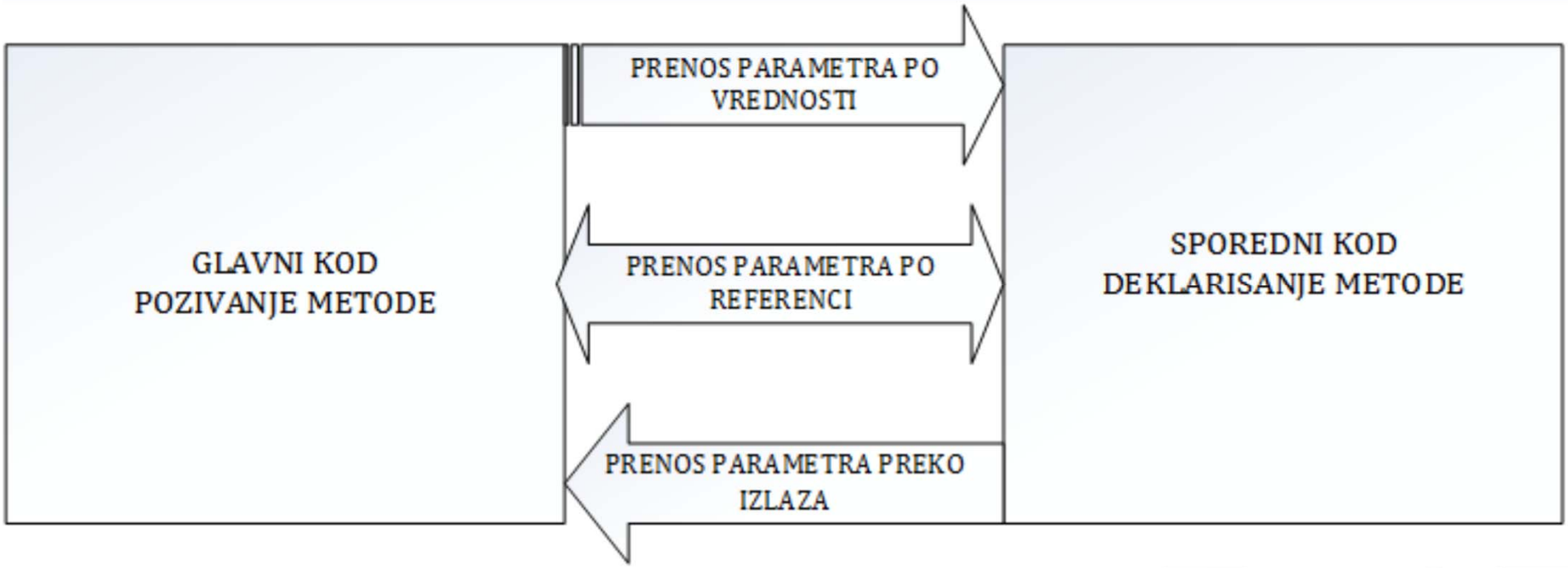
```
static void Main(string[] args)
        {
            int[] niz = new int[] { 1, 1, 1, 2, 3 };
            Console.Write("Pre funkcije ModifikujNiz() niz je: ");
            StampajNiz(niz);
            ModifikujNiz(niz);
            for (int i = 0; i < niz.Length; i++)
                {
                    if (niz[i] == niz[niz.Length - i - 1])
                    continue;
                }
                    Console.Write("Nakon ModifikujNiz() f-je niz je simetrican: ");
            StampajNiz(niz);
        }
        static void ModifikujNiz(int[] a)
        {
            for (int i = 0; i < a.Length / 2; i++)
               {
                 if (a[i] != a[a.Length - i - 1])

                     a[i] = a[a.Length - i - 1];
               }
            Console.Write("U funkciji ModifikujNiz() niz je: ");
            StampajNiz(a);
         }
        static void StampajNiz(int[] a)
            {
               Console.Write("[");
                for (int i = 0; i < a.Length; i++)
                     Console.Write("{0}", a[i]);
                 Console.WriteLine("]");
            }
```

Prenos parametara

- Metoda može *vratiti* isključivo *jednu vrednost* uz pomoć *return* operatora.
- Za više od jedne vrednosti koriste se mehanizmi za prenos parametara metode:
  - **preko vrednosti** – vrši *se inicijalizacija ulaznih parametara* u glavnom kodu. **Podaci** se prenose **U metodu**, ali se **NE mogu preneti iz nje**. Svi prethodi primeri koristili su ovaj način prenosa parametara;
  - **preko reference** – *inicijalizacija ulaznih i izlaznih parametara*. **Podaci** se prenose **U metodu i mogu se preneti i IZ nje**;
  - **preko izlaza** – *inicijalizacija izlaznih parametara*. **Podaci se** prenose **IZ metode**, ali se **NE mogu preneti u nju**;

Prenos parametara



Prenos parametra po vrednosti

- Vrednosni tipovi (int, char, long, float,…) prenose **KOPIJU** svoje **VREDNOSTI**.
- Referentni tipovi (niz, string) prenose **KOPIJU** svoje **REFERENCE**.

Prenos parametra po vrednosti i referenci

```
static void Main()
    {
        int i = 10;
        int iKopija = i;
        i++;
        Console.WriteLine("i={0}, iKopija={1}",
i,iKopija);
    }
```

Uvećanje promenljive **i** ne utiče na njenu kopiju.

```
static void Inkrementiraj(int a)
        {
                a++;
                Console.WriteLine("a="+a);
        }
    static void Main()
    {
        int i = 10; //argument f-je
        Console.WriteLine("i=" + i);
        Inkrementiraj(i);
        Console.WriteLine("i="+i);
    }
```

```
i=10
a=11
i=10
```

SM1

```
static void Inkrementiraj(ref int a)
        {
            a++;
              Console.WriteLine("a="+a);
        }
    static void Main()
    {
        int i = 10;
        Console.WriteLine("i=" + i);
        Inkrementiraj(ref i);
        Console.WriteLine("i="+i);
```

```
i=10
a=11
i=11
```

SM1

Kada se prosledi argument metodi, odgovarajući parametar se inicijalizuje kopijom argumenta. Ovo važi bez obzira na to da li je parametar vrednosnog tipa (npr. int) ili je referentnog tipa (npr. string). To znači da bilo kakva promena vrednosti parametra ne može da utiče na promenu vrednosti originalnog argumenta.
Ako se parametru doda prefiks ref, on postaje referenca na originalni argument (umesto njegova kopija). Zbog ovoga svaka izmena nad ref parametrom je automatski izmena originalnog argumenta. Ovaj prefiks se mora dodati i ispred originalnog argumenta i ispred parametra u metodi.

Suzana Marković; 9.12.2015.

Upoređenje – parametar vrednosnog tipa

- Kada se promenljiva vrednosnog tipa predaje metodi, u stvari se predaje kopija promenljive metodi.
- Svaka promena izvršena nad parametrom unutar metode neće imati nikakvog uticaja na originalne podatke koji su smešteni u promenljivoj izvan metode.
- Metoda koja će promeniti vrednost parametra, mora da se prenese po referenci koristeći **ref** ili **out** ključne reči (referenca na originalni argument).

S6

```
static void Kvadrat(int n)
{
    n *= n;
    System.Console.WriteLine("Vrednost unutar je: {0}", n);
}
static void Main()
{
    int n = 5;
    System.Console.WriteLine("Vrednost pre je: {0}", n);

    Kvadrat(n);  // Passing the variable by value.
    System.Console.WriteLine("Vrednost posle je: {0}", n);
```

Upoređenje – parametar vrednosnog tipa

```
Vrednost pre je: 5
Vrednost unutar je: 25
Vrednost posle je: 5
```

Svaka promena unutar metode Kvadrat neće imati uticaja na promenljivu n izvan metode.

```
static void Kvadrat(ref int n)
  {
      n *= n;
      Console.WriteLine("Vrednost unutar je: {0}", n);
  }
  static void Main()
  {
      int n = 5;
      Console.WriteLine("Vrednost pre je: {0}", n);
//5
      Kvadrat(ref n);
Console.WriteLine("Vrednost posle je: {0}", n);
  }
```

```
Vrednost pre je: 5
Vrednost unutar je: 25
Vrednost posle je: 25
```

S6

```
static void Kvadrat(int n)
        {
            n *= n;
            Console.WriteLine("Vrednost unutar je: {0}", n);
        }
        static void Main()
        {
            int n = 5;
            Console.WriteLine("Vrednost pre je: {0}", n);
            //5
            Kvadrat(n);
            Console.WriteLine("Vrednost posle je: {0}", n);
        }
```

Suzana; 7.5.2018.

Upoređenje – parametar referentnog tipa

- Promenljiva referentnog tipa ne sadrži podatke direktno, već sadrži referencu na podatke.
- Kada se preda parametar referentnog tipa po vrednosti, moguće je promeniti podatke na koje pokazuje referenca, ali se NE menja vrednost same reference.
- Unutar metode se može stvoriti novi objekat i referenca preusmeriti na njega, ali to će trajati koliko i metoda.
- Promene nisu trajne zato što se zapravo metodi predaje kopija reference kada se NE koristi ključna reč ref. Kopija se menja unutar metode, ali original ostaje isti.
- Izvan metode, referenca će opet pokazivati na stari objekt.
- Za promene se moraju koristiti **ref** ili **out** ključne reči.
- Kada se koristi ref, realokacija niza postaje trajna.

Prenos parametra referentnog tipa po vrednosti

```
1
-3
1
```

```
static void Change(int[] niz)
        {
            niz = new int[5] { -3, -1, -2, -3, -4 };
            System.Console.WriteLine("Unutar metode, prvi element je: {0}", niz[0]);
        }
        static void Main()
        {
            int[] vektor = { 1, 4, 5 };
            System.Console.WriteLine("Unutar metode Main, pre pozivanja metode,
prvi element je: {0}", vektor[0]);
            Change(vektor);
            System.Console.WriteLine("Unutar metode Main, posle pozivanja metode,
prvi element je: {0}", vektor[0]);
        }
```

S5

Isključiti niz[0] = 0; Kako se sada ponaša funkcija? 1, -3, 1
Ako zamenimo mesta biće 1, -3, 0 pošto se prenosi kopija reference, ali se objekat promenio
Suzana; 7.5.2018.

Prenos parametra referentnog tipa po referenci

```
static void Change(ref int[] niz)
    {
        niz = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Unutar metode, prvi element je: {0}", niz[0]);
    }

    static void Main()
    {
        int[] vektor = {1, 4, 5};
        System.Console.WriteLine("Unutar metode Main, pre pozivanja metode,
prvi element je: {0}", vektor[0]);

        Change(ref vektor);
        System.Console.WriteLine("Unutar metode Main, posle pozivanja metode,
prvi element je: {0}", vektor[0]);
```

```
1
-3
-3
```

S4 Isključiti niz[0] = 1; Kako se sada ponaša funkcija?

Suzana; 7.5.2018.

Parametar out

- Za prenos po referenci se koristi ključna reč **ref**, a za izlazni parametar - **out**
- Ove ključne reči se koriste i u **definiciji metoda** i na **mestu poziva**.
- Izlazni parametri **ne moraju** da budu inicijalizovani u glavnom delu programa pre prosleđivanja metodi.
- Izlaznom parametru se **mora** pridružiti vrednost u metodi.

Parametar out

```
static void Metoda(out int x)
{
    //x = 5;
    Console.WriteLine(x);
}
static void Main()
{
    int x=5;
    //Console.WriteLine(x);
Metoda(out x);
}
```

The out parameter 'x' must be assigned to before control leaves the current method

```
static void Metoda(out int x)
    {
        x = 5;
        Console.WriteLine(x);
    }
    static void Main()
    {
        int x; //može i int x=1;
     Metoda(out x);
    }
```

x = 5

```
static void Metoda(out int x)
{
    int x;
    Cons
}
static void Main()
{
    int x=5;
    //Console.WriteLine(x);
Metoda(out x);
```

A local variable named 'x' cannot be declared in this scope ...

Izlazni parametar

```
static void Change(out int[] niz)
    {
        niz = new int[5] ; //neinicijalizovani niz
        System.Console.WriteLine("Unutar metode, prvi element je: {0}",
niz[0]);
    }

    static void Main()
    {
        int[] vektor = {1, 4, 5};
        System.Console.WriteLine("Unutar metode Main, pre pozivanja
metode, prvi element je: {0}", vektor[0]);

        Change(out vektor);
        System.Console.WriteLine("Unutar metode Main, posle pozivanja
metode, prvi element je: {0}", vektor[0]);

    }
```

Primeri 1.

```
static void Main(string[] args)
        {
            int x = 1, y = 2;
            x = f(ref x) + y;
            Console.WriteLine(x);
        }
        static int f(ref int x)
        {
            x += 2;
            return x;
        }
```

Primeri 2.

```
static void Main(string[] args)
        {
            int x = 2;
            x = f(ref x);
            Console.WriteLine(x);
        }
        static int f(ref int x)
        {
            x += (++x) + x;
            return x;
        }
```

Primeri 3.

```
static void Main(string[] args)
{
    int n = 4, k = 5;
    f(n, k);
    Console.WriteLine(n + " " + k);
}
    static void f(int x, int y)
{
     x = 3 * y;
}
```

```
static void Main(string[] args)
    {
        int n = 4, k = 5;
        f(ref n, ref k);
        Console.WriteLine(n + " " + k);
    }
    //static void f(int x, int y)
    static int f(ref int x, ref int y)
{
    x = 3 * y;
    return x;
}
```

Primeri 4.

```
 static void Funkcija(ref int[] nizF)
{
    nizF = new int[4] { -1, 0, -3, -2 };
      Console.Write(nizF[3] + " ");
}
static void Main()
{
    int[] nizM = { 1, 2, 4 };
    for (int i = nizM.Length - 1; i >= 0; i--)
      Console.Write(nizM[i] + " ");
    Console.WriteLine();
    Funkcija(ref nizM);
    System.Console.Write(nizM[0] + " " +nizM[1]);
}
```

```
static void Funkcija(int[] nizF)
        {
            nizF = new int[4] { -1, 0, -3, -2 };
              Console.Write(nizF[3] + " ");
        }
        static void Main()
        {
            int[] nizM = { 1, 2, 4 };
            for (int i = nizM.Length - 1; i >= 0; i--)
               Console.Write(nizM[i] + " ");
            Console.WriteLine();
            Funkcija(nizM);
            System.Console.Write(nizM[0] + " " +nizM[1]);
        }
```

S2

Opseg važenja promenljive u drugim strukturama

```
static void Main(string[] args)
 {
     int i;
     for (i = 0; i < 5; i++)
     {
         string text = "i=" + i;
         Console.WriteLine("For petlja za {0}", text);

     }
      Console.WriteLine("Van petlje je:{0}", text);
      Console.ReadKey();
      }
      }
```

The name 'text' does not exist in the current context

```
static void Main(string[] args)
 {
     int i;
     string text = " ";
     for (i = 0; i < 5; i++)
     {
         text = "i=" + i;
         Console.WriteLine("For petlja za {0}", text);

     }
      Console.WriteLine("Van petlje je:{0}", text);
      Console.ReadKey();
      }
```

S2

```
int i;
        for (i=0;i<10;i++)
        {
            string text = "za i="+i;
          Console.WriteLine("For petlja {0}",text);

        }
        Console.WriteLine("Van petlje {0}", text);
        Console.ReadKey();
```

Suzana; 29.11.2016.

Opseg važenja promenljive u drugim strukturama

- Zašto promenljiva **text** posle petlje ne zadržava vrednost “ ” koja joj je dodeljena pre petlje?
- Kada se promenljivama dodele vrednosti, one se raspoređuju u memoriji.
- Kada se ova raspodela dogodi unutar petlje, vrednost je deklarisana kao lokalna i izlazi izvan opsega važenja kada je van petlje.

Opseg važenja promenljive u drugim strukturama

- Iako sama promenljiva nije lokalnog karaktera za petlju, vrednost koju sadrži jeste.
- Dodeljivanje vrednosti izvan petlje omogućava (u primeru string text=“ “) da ta vrednost postane lokalna za glavni deo koda, a da i dalje bude u opsegu unutar petlje.
- To znači da promenljiva ne izlazi iz svog opsega važenja, dokle god ne izađe iz bloka glavnog dela koda, tako da imamo pristup njenim vrednostima izvan petlje.