PRIRUČNIK ZA PROGRAMIRANJE (C# PROGRAMSKI JEZIK)

Osnove sintakse

Naredba predstavlja iskaz kojim se zadaje komanda računaru da izvrši neku operaciju. Kraj svake naredbe završava se simbolom tačka-zarez (“**;**”).

Naredbe se mogu pisati jedna za drugom u istom redu, ali najčešće se pišu u posebnim redovima.

Naredbe se pišu u blokovima oivičenim vitičastim zagradama **{ }**. Blokovi određuju oblast vidljivosti promenljivih, odnosno oblast dejstva nekog koda. Moguće je da postoji više ugnježdenih blokova.

Programski kod C# jezika razlikuje velika i mala slova (“case sensitive” – ime i Ime nisu iste promenljive).

Komentari se koriste da bi se neki deo programskog koda objasnio. Koristimo **//** za svaki red ili **/* */** za blok komentara.

Naredbe

jedna naredba - naredba;

blok naredbi – { blok naredbi }

Tipovi podataka

int – brojčana vrednost

double – decimalna brojčana vrednost

char – jedan karakter

string – niz karaktera, odnosno tekst

bool – logički tip, vrednost tačno ili netačno (true/false)

Konverzija tipova podataka

int.Parse(nazivPromenljive);

Menja vrednost tipa podataka string u integer, ako je ta vrednost broj.

double.Parse(nazivPromenljive);

Menja vrednost tipa podataka string u double, ako je ta vrednost broj.

char.Parse(nazivPromenljive);

Menja vrednost tipa podataka string u char, ako je ta vrednost jedan karakter.

(double)nazivPromenljive;

Menja vrednost tipa podataka integer u double.

(int)nazivPromenljive;

Menja vrednost tipa podataka double u integer.

Convert.ToString(nazivPromenljive);

Menja vrednost promenljive u string.

Promenljive

Promenljive predstavljaju nazive koji su dati memorijskim lokacijama u kojima se smeštaju podaci. Podaci u tim promenljivima se u toku izvršavanja programa mogu menjati. Promenljive su određene imenom, memorijskom lokacijom, tipom podatka i vrednošću.

Pravila prilikom kreiranja promenljive

- Prvo se navodi tip podatka (int, double, string …), zatim ime promenljive i opciono (može ali ne mora) početna vrednost.
- Naziv promenljive mora početi ili slovom ili donjom crtom, a nakon toga u sebi može sadržati slova, brojeve i donje crte.
    - Ispravni nazivi: ime, _prezime, broj_izostanaka, player2…
    - Neispravni nazivi: 1znak, druga ocena, ime-prezime…
- Nije dozvoljen razmak unutar naziva promenljive.
- Rezervisane reči poput int, float, using se ne mogu koristiti kao nazivi promenljivih.
- Kada je promenljiva sa određenim nazivom kreirana taj naziv se više ne može koristiti za kreiranje novih promenljivih.
- Moguće je kreirati više promenljivih u jednoj liniji koda (sa ili bez dodele vrednosti) i tada se prvo navodi tip podatka, a zatim nazivi promenljivih koji se od drugih promenljivih razdvajaju zarezom.

Primeri kreiranja promenljive – određivanje imena i tipa podatka promenljive, određivanje imena, tipa i dodeljivanje vrednosti, i kreiranje više promenljivih istog tipa.

```
int broj;

double decimalanBroj;

char karakter;

string tekst;

bool proveraTacnosti;


int broj = 10;

double decimalanBroj = 0.7;

char karakter = 'A'; (piše se između po jednog navodnika)

string tekst = "Piše se između dva navodnika";

bool proveraTacnosti = true;


int x, y, z;

int x=1, y, z;

int x=1, y=2, z=3;
```

Operatori

+ sabiranje, spajanje stringova, spajanje stringova i promenljivih

- oduzimanje

***** množenje

/ deljenje

% ostatak od deljenja

+= x+=2 je isto što i x=x+2

-= x-=2 je isto što i x=x-2

***=** x*=2 je isto što i x=x*2

/= x/=2 je isto što i x=x/2

%= x%=2 je isto što i x=x%2

++ x++ je isto što i x=x+1

-- x-- je isto što i x=x-1

== jednakost – vraća vrednost *tačno* ako su vrednosti promenljivih identične

!= nije jednako – vraća vrednost *tačno* ako su vrednosti promenljivih različite

< manje – vrednost izraza je *tačna* ako je promenljiva sa leve strane znaka manja od one sa desne strane

> veće - vrednost izraza je *tačna* ako je promenljiva sa leve strane znaka veća od one sa desne strane

<= manje ili jednako – vrednost izraza je *tačna* ako je promenljiva sa leve strane znaka manja ili jednaka od one sa desne strane

>= veće ili jednako - vrednost izraza je *tačna* ako je promenljiva sa leve strane znaka veća ili jednaka od one sa desne strane

&& uslovno logičko AND - ako je prvi operand *netačno*, drugi se ni ne proverava

|| uslovno logičko OR - ako je prvi operand *tačno*, drugi se ni ne proverava

Konzolne aplikacije

Ispisivanje na ekran

Console.Write("Tekst"); - Ispisuje se sve između navodnika.

Console.WriteLine("Tekst"); - Ispisuje se sve između navodnika dok se naredni ispis prebacuje u sledeći red.

Console.WriteLine("{0} je veće od {1} dok je {2} jednako {3}", a, b, c, d); - Ispisuje vrednost promenljivih a, b, c i d redosledom kojim su nabrojane nakon zatvorenih navodnika i zareza, a na mestima brojeva oivičenih vitičastom zagradom unutar navodnika.

Console.WriteLine($"{a} je veće od {b} dok je {c} jednako {d}"); - Ispisuje vrednost promenljivih a, b, c i d redosledom kojim su navedene unutar vitičastih zagrada.

Console.WriteLine("Rezultat je: " + zbir); - Ispisuje vrednost promenljive zbir nakon teksta između navodnika.

Učitavanje iz konzole

Console.ReadLine(); - Učitava sve što korisnik ispiše u konzolu pre nego što pritisne *Enter* (u kom god da je obliku čita ga kao promenljivu tipa string).

Console.ReadKey(); - Čeka da korisnik pritisne neko dugme pre nego što nastavi sa izvršavanjem naredbi ili zatvaranjem programa. Postavlja se na kraj programa kako se nakon izvršavanja ne bi zatvorila konzola pre nego što korisnik pritisne neko dugme.

Uslovi

U svakodnevnom životu često donosimo odluke u zavisnosti od uslova. Na primer:

- Ako pada kiša, ponesem kišobran.
- Ako je ocena 5, učenik je odličan, u suprotnom ocena je manja.

U programiranju ovaj princip se prenosi pomoću **naredbi uslovnog grananja**.

if naredba

if naredba omogućava izvršavanje jedne ili više naredbi **samo ako je uslov ispunjen (true)**. Ako uslov nije ispunjen (false), program preskače taj deo i nastavlja dalje.

- **if** → izvršava se kod ako je uslov tačan.
- **if ... else** → biramo između dve mogućnosti (tačno – netačno).
- **if ... else if ... else** → koristi se kada postoji više mogućih uslova koji se redom proveravaju.
- **Ugnježdeni if** → jedan if može da stoji unutar drugog.

Ključna stvar: **od gore nadole se proveravaju uslovi, a čim jedan bude tačan, ostali se preskaču**.

switch naredba

switch se koristi kada treba proveriti više mogućih vrednosti **iste promenljive ili izraza**. Svaka mogućnost označava se rečju **case** i vrednošću koju proveravamo. Na kraju svakog slučaja koristi se reč **break** da bi se sprečilo „proklizavanje" u sledeći slučaj. Ako nijedan slučaj nije ispunjen, može se koristiti deo **default**.

switch je pregledniji od velikog broja *if/else if* uslova, ali radi samo kada poredimo **jednu vrednost sa nizom mogućnosti** (ne koristi se za logičke izraze tipa x > 10).

Kada koristiti if, a kada switch?

- **if/else** → kada uslov može biti složeniji (npr. poređenja, kombinovanje logičkih izraza).
- **switch** → kada se proverava **jedna promenljiva** koja može imati više različitih vrednosti.

Kratak pregled

- **if** – izvršava kod samo ako je uslov tačan.
- **if ... else** – bira između dve opcije.
- **if ... else if ... else** – bira između više mogućnosti redom.
- **switch** – proverava vrednost promenljive kroz više „slučajeva" (case).
- **default** – deo koji se izvršava ako nijedan slučaj nije ispunjen.

Petlje

U svakodnevnom životu često ponavljamo iste radnje više puta:

- Dok ima slobodnih mesta u autobusu, ljudi ulaze.
- Dok ima goriva, automobil može da vozi.
- Dok ne dobijemo tačan odgovor, ponavljamo pitanje.

U programiranju za takve situacije koristimo **petlje** – konstrukcije koje omogućavaju da se neki deo koda izvršava više puta, dokle god je ispunjen određeni **uslov**.

while petlja

while znači „dok".

- Petlja proverava **uslov pre svakog ponavljanja**.
- Ako je uslov tačan (true), naredbe u petlji se izvršavaju.
- Kada uslov postane netačan (false), petlja se prekida i program ide dalje.

Najčešće se koristi kada **ne znamo unapred koliko puta** će se nešto ponavljati. Na primer: *dok korisnik ne unese nulu, ponavljaj unos broja.*

do-while petlja

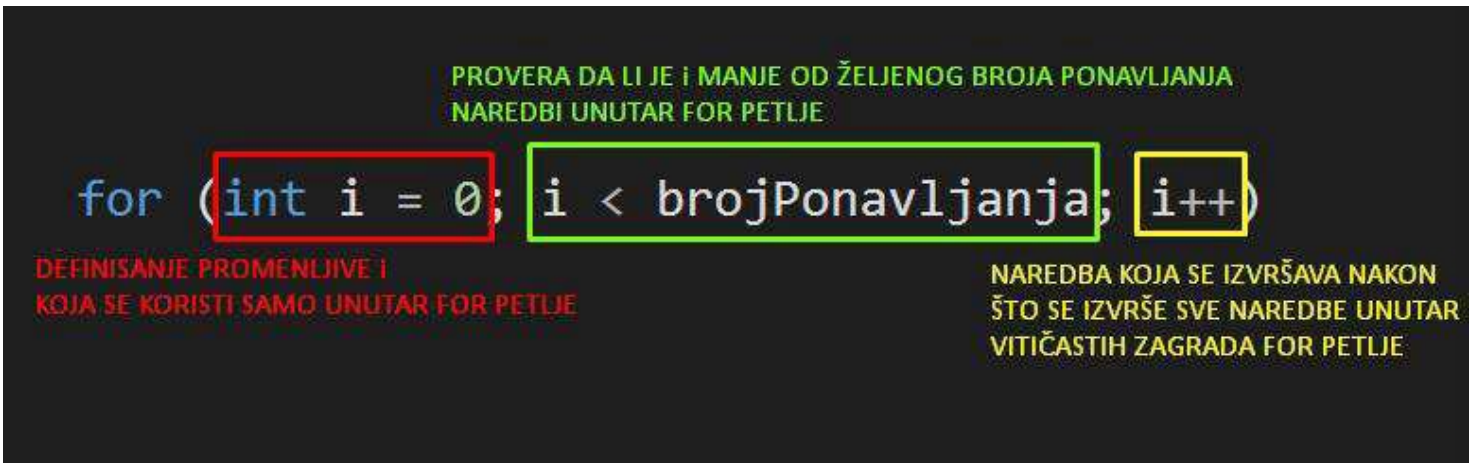
do-while znači „uradi pa proveri".

- Naredbe unutar petlje izvršavaju se **najmanje jednom**, jer se uslov proverava tek nakon izvršenja.
- Nakon toga, ako je uslov i dalje tačan, naredbe se ponavljaju.

Koristi se kada želimo da se neki postupak obavezno uradi barem jednom. Na primer: *traži od korisnika lozinku, pa proveri da li je tačna. Ako nije, traži ponovo.*

for petlja

for (početna_vrednost; uslov; promena)
{ naredbe; } Izvršava naredbe, zatim izvršava proveru, i ponavlja taj proces dok god je uslov ispunjen.



Primer – ukoliko želimo da tri puta ispišemo vrednost tekstualne promenljive:

```
string smer = "Elektrotehnicar racunara";
for (int i = 0; i < 3; i++)
{
Console.WriteLine(smer);
}
```

Isto se moglo postići i sa while petljom:

```
int i = 0;
while (i < 3)
{
Console.WriteLine(smer);
i++;
}
```

Međutim kada znamo broj ponavljanja preporučljivo je koristiti for petlju.
Primer for petlje u radu sa nizovima:

```
/* nizA.Length predstavlja broj članova niza A što znači da će se petlja ponavljati za svaki od
članova */
for (int i = 0; i < nizA.Length; i++)
{
/* nizA[i] predstavlja člana niza sa indeksom i (indeks kreće od nule) - u ovom slučaju se
učitava vrednost iz konzole i smešta unutar nizovne promenljive */
nizA[i] = int.Parse(Console.ReadLine());
}
```

Poređenje while, do-while i for

Karakteristika	**while**	**do-while**	**for**
Provera uslova	na početku	na kraju	na početku
Moguć broj izvršavanja	0 ili više puta	najmanje 1 put	0 ili više puta
Kada se koristi?	kad možda nema ponavljanja	kad mora biti bar jedno ponavljanje	kad unapred znamo broj ponavljanja
Brojač u petlji	ručno (pre i u petlji)	ručno (u petlji)	ugrađen u zaglavlju

Kratak pregled

- **while** – ponavlja dokle god je uslov tačan; može se desiti da se ne izvrši nijednom.
- **do-while** – prvo izvrši naredbe, pa proveri uslov; obavezno se izvrši bar jednom.
- **for** – koristi se kada je poznat broj ponavljanja; pregledna i najčešće korišćena petlja.

Nizovi

- Deklarisanje niza:

tipPodatka[] nazivNiza;
Ovom naredbom je samo deklarisan niz - C#-u još uvek nije saopšteno da treba da kreira niz sa specificiranim brojem elemenata.

- Inicijalizovanje niza:

tipPodatka[] nazivNiza = new tipPodatka[brojČlanovaNiza];
Operatorom new saopštava se C#-u da treba da kreira određeni broj promenljivih izabranog tipa podataka i da treba da omogući nizu pristup do njih.

- Načini dodeljivanja vrednosti inicijalizovanom nizu:

nazivNiza[indeks] = vrednost;
Indeks označava poziciju člana unutar niza. Računanje indeksa kreće od nule (prvi član niza ima indeks 0, a poslednji član niza ima indeks za jedan manje od ukupne dužine (broja članova) niza).

- Načini dodeljivanja vrednosti prilikom inicijalizacije niza:

tipPodatka[] nazivNiza = {vrednosti};
tipPodatka[] nazivNiza = new tipPodatka[] {vrednosti};

nazivNiza.Length - Daje broj članova niza.

Definisanje nizovne promenljive kada znamo koji su članovi niza:

```
// Prvi način
string[] predmeti = { "Srpski", "Matematika", "Programiranje" };
int[] ocene = { 4, 3, 5 };
// Drugi način
string[] predmeti = new string[3];
predmeti[0] = "Srpski";
predmeti[1] = "Matematika";
predmeti[2] = "Programiranje";
int[] ocene = new int[3];
ocene[0] = 4;
ocene[1] = 3;
ocene[2] = 5;
```

Definisanje nizovne promenljive kada znamo samo broj članova niza:

```
string[] predmeti = new string[3];
int[] ocene = new int[3];
```

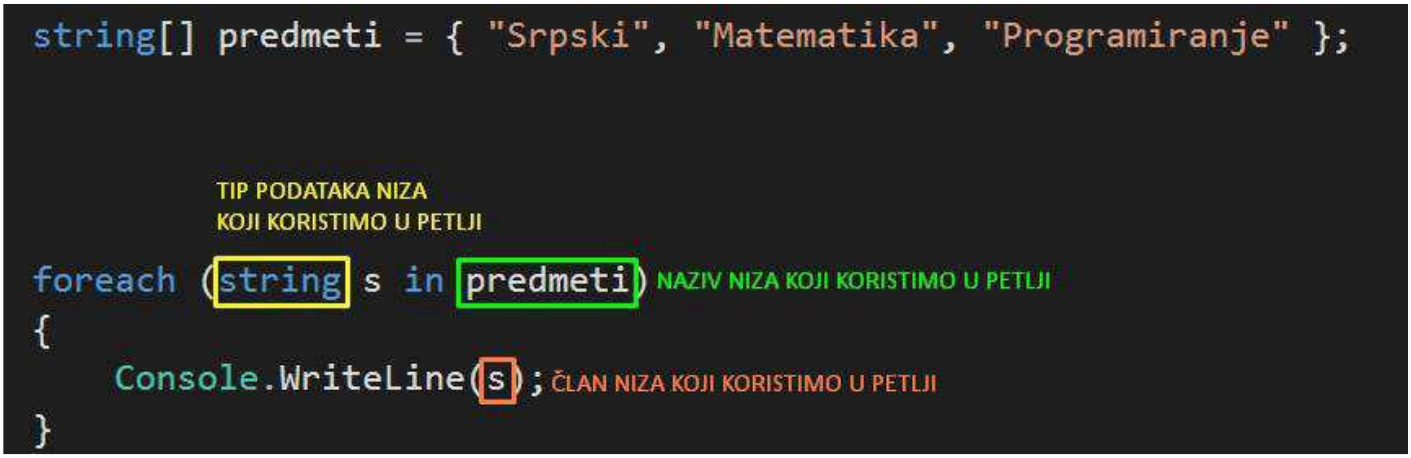
Definisanje nizovne promenljive kada korisnik unosi broj članova niza:

```
// Niz uvek definisati tek nakon što imamo vrednost za promenljivu koju smeštamo u uglaste zagrade
Console.WriteLine("Unesite broj članova niza:");
brojClanova = int.Parse(Console.ReadLine());
string[] predmeti = new string[brojClanova];
// Učitavanje članova niza od korisnika (bez korišćenja for i foreach petlje)
string[] predmeti = new string[3];
Console.WriteLine("Unesite članove za niz predmeti:");
predmeti[0] = Console.ReadLine();
predmeti[1] = Console.ReadLine();
predmeti[2] = Console.ReadLine();
// Ispis članova niza (bez korišćenja for i foreach petlje)
Console.WriteLine("Članovi niza predmeti su:");
Console.WriteLine(predmeti[0]);
Console.WriteLine(predmeti[1]);
Console.WriteLine(predmeti[2]);
```

foreach

foreach (tipPodataka nazivPromenljive in nazivNiza)
{ naredbe; }

Izvršava naredbe za sve članove niza. Koristi se kada želimo da prođemo kroz sve elemente niza, ali nam indeks nije potreban.



Stringovi

String predstavlja niz karaktera, odnosno tekst.

Koristi se za rad sa imenima, porukama, rečenicama i drugim tekstualnim podacima.

String se piše između dva dvostruka navodnika:

```
string ime = "Tijana";
string poruka = "Srećno učenje programiranja!";
```

U C# jeziku string je poseban tip podatka koji ima svoju dužinu, indekse (pozicije karaktera), i veliki broj ugrađenih metoda za obradu teksta.

Indeksi u stringu počinju od nule (0), isto kao kod nizova.

```
string grad = "Sombor";
```

Karakter	S	o	m	b	o	r
Indeks	0	1	2	3	4	5

Ispisivanje stringova na ekran

Stringovi se najčešće koriste za ispis poruka u konzoli.
U nastavku su prikazani najčešći načini ispisa tekstualnih podataka.

Osnovni ispis teksta

Console.Write("Tekst"); - Ispisuje tekst u istom redu.

Console.WriteLine("Tekst"); - Ispisuje tekst i prelazi u novi red.

Ispis promenljivih zajedno sa tekstom

Promenljive se mogu ispisivati zajedno sa stringovima na više načina.

- **Spajanje stringa i promenljive (+)**

```
Console.WriteLine("Rezultat je: " + zbir);
```

Promenljiva (u ovom slučaju ime joj je zbir) se automatski pretvara u string i spaja sa tekstom.

Operator + se izvršava sleva nadesno. Ako se prvo sretnu brojevi – oni se sabiraju. Ako se prvi pojavi string – sve nakon toga se spaja kao tekst.

```
int a = 5;
int b = 3;

Console.WriteLine(a + b + " je rezultat"); // Ispisuje se: 8 je rezultat
Console.WriteLine("Rezultat je " + a + b); // Ispisuje se: Rezultat je 53
```

- **Formatirani ispis pomoću indeksa**

```
Console.WriteLine("{0} je veće od {1} dok je {2} jednako {3}", a, b, c, d);
```

Vrednosti promenljivih (čija su imena u ovom slučaju a, b, c i d) se ubacuju na mesta označena brojevima u vitičastim zagradama.

- **Interpolacija stringova**

```
Console.WriteLine($"{a} je veće od {b} dok je {c} jednako {d}");
```

Promenljive (čija su imena u ovom slučaju a, b, c i d) se direktno ubacuju u string pomoću znaka $ i vitičastih zagrada.

Escape sekvence (specijalni karakteri)

U stringovima postoje escape sekvence – posebne kombinacije znakova koje počinju znakom \ i koriste se za ispis specijalnih karaktera ili kontrolu prikaza teksta.

- **\n – novi red**

```
Console.WriteLine("Ime: Duško\nPrezime: Petrović");
/* Ispis:
Ime: Duško
Prezime: Petrović
*/
```

- **\t – tabulator (razmak)**

```
Console.WriteLine("Ime:\tFilip");
// Ispis: Ime:  Filip
```

Verbatim string (@)

Kod verbatim stringa, tekst se ispisuje tačno onako kako je napisan (prelazak u novi red se ostvaruje pritiskom *Enter*-a).

```
Console.WriteLine(@"Vi ste učenik Srednje tehničke škole u Somboru.
Vaš smer je Tehničar IT.
Trenutno ste druga godina.");
/* Ispis:
Vi ste učenik Srednje tehničke škole u Somboru.
Vaš smer je Tehničar IT.
Trenutno ste druga godina.;
*/
```

Metode za rad sa stringovima

String u C# jeziku poseduje veliki broj ugrađenih metoda koje omogućavaju rad sa stringovima (tekstom).

Length

Svojstvo *Length* vraća broj karaktera u stringu (razmak se takođe računa kao karakter).

```
string predmet = "Programiranje";
Console.WriteLine(predmet.Length); // Ispis: 13
```

ToUpper() i ToLower()

Metode ToUpper() i ToLower() služe za promenu veličine slova u stringu.

- **ToUpper()** – pretvara sva slova u velika
- **ToLower()** – pretvara sva slova u mala

Originalni string se ne menja, već se dobija novi string.

```
string predmet = "Programiranje";
string malaSlova = predmet.ToLower();
string velikaSlova = predmet.ToUpper();

Console.WriteLine(malaSlova); // Ispis: programiranje
Console.WriteLine(velikaSlova); // Ispis: PROGRAMIRANJE
```

Contains()

Metoda Contains() proverava da li string sadrži određeni tekst.

Vraća logičku vrednost: *true* – ako string sadrži dati tekst ili *false* - ako ga ne sadrži.

```
string tekst = "Trenutno je u toku čas predmeta Programiranje.";

Console.WriteLine(tekst.Contains("čas")); // Ispis: True
```

Metoda *Contains()* razlikuje velika i mala slova.

Može se koristiti i unutar uslova kao na primeru ispod:

```
string poruka = "Dobrodošli u Sombor";

if (poruka.Contains("Sombor"))
{
    Console.WriteLine("Poruka sadrži reč Sombor.");
}
else
{
    Console.WriteLine("Poruka ne sadrži reč Sombor.");
}
```

IndexOf()

Metoda *IndexOf()* pronalazi poziciju (indeks) prvog pojavljivanja zadatog teksta u stringu.

Ako se tekst pronađe vraća indeks (broj), a ako se tekst ne pronađe vraća -1.

```
string predmet = "Programiranje";

Console.WriteLine(predmet.IndexOf("ram")); // Ispis: 4
Console.WriteLine(predmet.IndexOf("škola")); // Ispis: -1
```

Metoda IndexOf() razlikuje velika i mala slova.

Takođe se može koristiti unutar uslova, a sama pozicija se može čuvati u celobrojnoj promenljivoj:

```
string poruka = "Dobrodošli u Sombor";

int pozicija = poruka.IndexOf("Sombor");

if (pozicija != -1)
{
    Console.WriteLine($"Reč Sombor je pronađena na poziciji: {pozicija}");
}
else
{
    Console.WriteLine("Reč Sombor nije pronađena.");
}
```

Substring()

Metoda Substring() služi za izdvajanje dela teksta iz stringa. Postoje dva najčešća oblika korišćenja:

- **Substring(početniIndeks) -** Izdvaja tekst počev od izabranog indeksa do kraja stringa.
- **Substring(početniIndeks, brojKaraktera) -** Izdvaja izabrani broj karaktera od izabranog početnog indeksa.

```
string predmet = "Programiranje";

Console.WriteLine(predmet.Substring(6)); // Ispis: miranje
Console.WriteLine(predmet.Substring(0,7)); // Ispis: Program
```

Replace()

Metoda Replace() služi za zamenu jednog dela teksta drugim tekstom.

Originalni string se ne menja, već se dobija novi string.

```
string tekst = "Učimo programiranje";

string noviTekst = tekst.Replace("programiranje", "C#");

Console.WriteLine(noviTekst); // Ispis: Učimo C#
```

Na isti način se može menjati i samo jedan izabrani karakter.

Metoda Replace() razlikuje velika i mala slova.

Split()

Metoda *Split()* služi za razdvajanje teksta na više delova.

Rezultat metode *Split()* je niz stringova.

```
string tekst1 = "Dejan Tijana Filip";
string tekst2 = "Beograd,Sombor,Subotica";

string[] imena = tekst1.Split(' ');
string[] gradovi = tekst2.Split(',');

Console.WriteLine(imena[0]); // Ispis: Dejan
Console.WriteLine(imena[1]); // Ispis: Tijana
Console.WriteLine(imena[2]); // Ispis: Filip
Console.WriteLine(gradovi[0]); // Ispis: Beograd
Console.WriteLine(gradovi[1]); // Ispis: Sombor
Console.WriteLine(gradovi[2]); // Ispis: Subotica
```

Metode

Metoda predstavlja deo programa koji izvršava određeni zadatak i može se pozivati više puta tokom rada programa.

U programiranju se često dešava da isti skup naredbi treba da izvršimo više puta u različitim delovima programa (npr. više puta želimo da izračunamo zbir dva broja, želimo da ispišemo istu poruku korisniku na više mesta u programu ili želimo da ponovimo isti postupak učitavanja podataka). Umesto da svaki put ponovo pišemo isti kod, možemo taj deo programa izdvojiti u posebnu celinu i pozivati ga kada god je potrebno.

Na ovaj način program postaje pregledniji, lakši za razumevanje i lakši za održavanje i ispravljanje grešaka.

U C# jeziku program se često sastoji od više metoda koje međusobno sarađuju kako bi rešile određeni problem.

Svaka metoda u C# programskom jeziku ima određenu strukturu. Struktura metode određuje kako se metoda definiše i kako se njen kod organizuje.

Opšti oblik metode izgleda ovako:

```
static tipPovratneVrednosti NazivMetode(parametri)
{
    naredbe;
}
```

Delovi metode imaju sledeće značenje:

static - označava da metoda pripada klasi i da se može pozvati bez pravljenja posebnog objekta. U konzolnim programima koje trenutno radimo metode se najčešće pišu kao static.

tipPovratneVrednosti - određuje koju vrstu podatka metoda vraća nakon izvršavanja (npr. int znači da metoda vraća ceo broj a void da metoda ne vraća nikakvu vrednost).

NazivMetode - naziv koji sami biramo za metodu, treba da opisuje šta metoda radi (npr. IspisiPoruku, IzracunajProsek).

parametri - predstavljaju podatke koje metoda prima prilikom poziva. Navode se unutar okruglih zagrada (tipPodatka nazivParametra). Ako metoda nema parametre, zagrade ostaju prazne.

{ } - predstavljaju blok naredbi koje metoda izvršava. Sve naredbe koje se nalaze unutar ovog bloka izvršiće se kada se metoda pozove.

Metode bez povratne vrednosti

Kod metoda bez povratne vrednosti koristi se ključna reč void, što znači da metoda izvršava određene naredbe, ali nakon završetka ne vraća nikakav rezultat programu.

Primer metode bez povratne vrednosti i bez parametara:

```
static void Pozdravi()
{
    Console.WriteLine("Dobrodošli na čas programiranja!");
}
```

U ovom primeru static označava da metoda pripada klasi, void znači da metoda ne vraća vrednost, Pozdravi je naziv metode, metoda nema parametre (okrugle zagrade su prazne), unutar vitičastih zagrada nalazi se naredba koja se izvršava (ispis teksta).

Metoda se poziva na sledeći način:

```
Pozdravi();
```

Kada program dođe do ove naredbe, izvršiće se sve naredbe koje se nalaze unutar metode.

Primer metode bez povratne vrednosti i sa parametrima:

```
static void Pozdravi(string ime)
{
    Console.WriteLine($"{ime} dobrodošli na čas programiranja!");
}
```

U ovom primeru static označava da metoda pripada klasi, void znači da metoda ne vraća vrednost, Pozdravi je naziv metode, metoda ima jedan parametar tekstualnog tipa (navodi se u okruglim zagradama), unutar vitičastih zagrada nalazi se naredba koja se izvršava (ispis teksta zajedno sa vrednošću parametra).

Metoda se poziva na sledeći način:

```
Pozdravi("Dejan");
```

Kada program dođe do ove naredbe, izvršiće se sve naredbe koje se nalaze unutar metode.

Metode sa povratnom vrednošću

U ovim metodama umesto reči void navodi se tip podatka koji metoda vraća. Rezultat metode vraća se pomoću ključne reči return.

Primer metode sa povratnom vrednošću i bez parametara:

```
static int VratiBroj()
{
    return 10;
}
```

U ovom primeru: int označava da metoda vraća ceo broj, metoda nema parametre (zagrade su prazne), metoda vraća vrednost 10.

Jedan od načina poziva metode:

```
int broj = VratiBroj();
```

Program će dodeliti vrednost koju metoda vraća promenljivoj broj (promenljiva mora biti istog tipa kog je i povratna vrednost metode).

Primer metode sa povratnom vrednošću i sa parametrima:

```
static int IzracunajZbir(int a, int b)
{
    int rezultat = a + b;
    return rezultat;
}
```

U ovom primeru: int označava da metoda vraća ceo broj, IzracunajZbir je naziv metode, a i b su parametri metode, unutar metode računa se zbir brojeva, naredba return rezultat vraća izračunatu vrednost.

Pozivanje metode:

Metoda sa povratnom vrednošću najčešće se koristi tako što se rezultat smesti u promenljivu.

```
int suma = IzracunajZbir(3, 4);
Console.WriteLine(suma);
```

U ovom primeru: metoda IzracunajZbir se poziva sa vrednostima 3 i 4, metoda izračunava zbir, vrednost se vraća pomoću return, rezultat se smešta u promenljivu suma. Ispis u konzoli biće: 7

Rezultat metode može se odmah ispisati:

```
Console.WriteLine(IzracunajZbir(3, 4));
```

U ovom slučaju metoda se izvršava unutar naredbe WriteLine i njen rezultat se odmah ispisuje.

Preklapanje imena metode (preopterećenje metode, overloading)

Isto ime metode možemo koristiti ukoliko imamo različiti tip parametara ili različiti broj parametara.

```
static void Saberi(int x, int y)
{
Console.WriteLine("Zbir je {0}", x + y);
}
static void Saberi(int x, int y, int z)
{
Console.WriteLine("Zbir je {0}", x + y + z);
}
```

Ukoliko koristimo različiti broj parametara prilikom kreiranja metode možemo koristiti isti naziv metode jer će program automatski pozvati ispravnu metodu u zavisnosti od toga koliko promenljivih stavimo u zagradu prilikom pozivanja:

```
Saberi(2, 8); // Poziva metodu sa dva parametra (x i y)
Saberi(1, 3, 5); // Poziva metodu sa tri parametra (x, y i z)
```

Drugi način preklapanja imena metode:

```
static int Saberi(int x, int y)
{
return x + y;
}
static double Saberi(double x, double y)
{
return x + y;
}
```

Ukoliko koristimo različit tip podataka za povratnu vrednost možemo koristiti isti naziv metode, a program će na osnovu tipa podataka promenljivih unutar zagrada pozvati ispravnu metodu:

```
Saberi(6, 4); // Poziva metodu koja sabira cele brojeve
Saberi(2.5, 1.3); // Poziva metodu koja sabira decimalne brojeve
```