Esencijalni deo C#

UVOD

Uvod

Ovo predavanje posvećeno je nekim esencijalnim elementima kodiranja u C#: anatomija C# fajla, tipovi podataka u C#.Takodje, funkcijama tj. metodama u C#. Takodje, razgovaraćemo o :

load-event, metode Parse() i ToString(), messagebox metodama, listbox metodama, messagebox dijalogu,

a isto tako o konzolnim aplikacijama.

U ovoj lekciji bavimo se raznim specijalnim tipovima podataka, npr. „strukture“, enum-varijable, stringovima. Ovo spada u oblast data-structures tj. „struktura-podataka“. Takodje, pomenućemo konverziju varijabli i upotrebu modifikatora „ref“ i „out“.

Ključna pitanja:

Kako izgleda anatomija C# fajla?

Kako se prave konzolne aplikacije u C#?

Kako se koriste funkcije tj. metode u C#?

Kako se vrši konverzija tipa varijabli?

Kako se manipuliše stringovima?

Kako se koriste modifikatori *„ref“ i „out“?*

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Elementi kodiranja u C#

01

ELEMENTI KODIRANJA

Postoji potreba da se piše programski kod, i ovim se kodom kontrolišu vizuelni elementi jezika.

Visual C# omogućuje da se puno toga uradi vizuelno, i koristeći „miš“, ali i dalje postoji potreba da se piše programski kod, i ovim se kodom kontrolišu vizuelni elementi jezika. Programi su kolekcije instrukcija koje komanduju kompjuterom i definišu šta kompjuter treba da uradi. Svaka instrukcija ima preciznu sintaksu, i svaka instrukcija ponaosob je relativno jednostavna. Ali, ako se u program stavi stotine ili hiljade instrukcija program postaje veoma složen. Jezik C# obuhvata sledeće elemente:

Iskazi , i Logičke naredbe, *statements/control-statements*, to su instrukcije koje definišu kompjuteru kako da izvršava program tj. neke operacije, npr**. if-iskaz** kaže da se uradi nešto, neka operacija, ako je neki uslov ispunjen tj. tačan.

Promenljive, *variables*, to su reči koje se koriste da bi se memorisale neke vrednosti tj. **podaci**, npr. instrukcija var1 = “ beograd”; memoriše seriju znakova kao varijablu koja se zove Var1.

Operatori, *operators*, to su simboli koji označavaju operacije, npr aritmetički operatori, + ili – , ili operator zadavanja nekog podatka nekoj varijabli =, ili operator množenja *, ili deljenja /, a npr. logički operator ILI (tj. engleski *or)* je predstavljen kao ||, operator jednakosti je ==, a operator nije-jednakost je !=, itd.

Objekti i klase (*objects, classes)*: objekti su npr. Label i Button, itd., i to su predefinisani objekti, ali mogu se kreirati i objekti po želji programera koji označavaju neki fizički ili abstraktni objekat iz sveta koji nas okružuje, a klasa je deo programa kojim se opisuju objekti tj. serija sličnih objekata odnosno serija objekata iz iste “fanilije”. Klasa se može definisati i kao deo programskog koda koji definiše objekte.

Karakteristike, *properties*, su osobine objekta. **Metode,** methods, to su akcije koje neki objekat može da uradi, npr. Metoda Hide za objekat Label; medjutim, neke metode rezultiraju u neki podatak, tj. vraćaju kao rezultat neki podatak (*return a value)*.

Parametri, parameters, odnosno argumenti, *arguments*, to su podaci koji se daju tj. isporučuju metodama kao ulazni podaci.

Najbolji način da se nauči programiranje je kroz samo programiranje tj. Kroz vežbu. Visual C# pruža ugodno okruženje za vežbanje tj. učenje programiranja. Niz pomoćnih alata je na raspolaganju.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ANATOMIJA C# FAJLA

Dole su nabrojani i kratko opisani ovi pojedini delovi C# fajla.

Neki C# fajl ima definisanu strukturu, tj sastoji se od nekoliko definisanih različitih delova. Dole su nabrojani i kratko opisani ovi pojedini delovi C# fajla. A kasnije će ovi elementi C# fajlova biti detaljnije opisani.

1. ***Using statements*:** *Using*-iskazi: na početku mnogih C# - fajlova se nalaze jedan ili nekoliko iskaza koji počinju sa rečju:

using,

i ovi iskazi daju instrukcije kompajleru gde da potraži klase koje se kasnije pominju u programu. Ovi *using*-iskazi se ne izvršavaju, i oni su neobavezni tj. oni su opcioni, dakle štede kucanje nekih instrukcija u programu.

2. ***Namespace declaration*:** *Namespace*-deklaracija je prisutna u većini C# - fajlova, a iza toga je blok u velikim zagradama, i ove deklaracije predstavljaju jedan način organizacije programskog koda, i omogućuju kompajleru da pronadje klase koje su definisane u programu

3. ***Comments:*** Komentari, *comments,* u programu se označavaju sa dve ili više kosih crta, naime bilo koja linija programskog koda koja počinje sa dve ili više kosih crta je samo komentar a ne i prava instrukcija.

4. ***Class statements:*** Neki C# fajl može da sadrži jednu ili nekoliko klasa, iako jedna klasa u fajlu je tipičan slučaj. Klasni iskazi, *class statements*, to su iskazi koji uvode nove klase u program, a markirani su velikim zagradama. I mada većina fajlova definišu klase, ima fajlova koji ne definišu klasu već npr interfejs ili *custom type*, tj. uslužni tip podataka.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ANATOMIJA C# FAJLA

Dole su nabrojani i kratko opisani ovi pojedini delovi C# fajla.

5. ***Class-level variables*****:** Kao što je na prethodnoj strani ilustrovano, varijable klasnog nivoa, class-level variables , su varijable deklarisane unutar klase, koje su vidljive tj. pristupne u okviru klase, a opciono su pristupne i van klase.

6. ***Method declarations*****:** Deklaracije metoda, *method declarations*, to je dodatni programski kod za neki fajl tj. neku klasu, i on opisuje funkcionalnost klase, i tu u ovom dodatnom kodu, je obično data i lista karakteristika, *properties*. Ovo su tzv. *member functions* koje definišu funkcionalnost klase.

7. **# *directives*:** # - uputstva tj direktive**,** # *directives*, to su iskazi koji počinju sa #, i to su direktive kompajleru, tzv. preprocesorske direktive, tj. uputstva.

8. ***Generated code:*** Automatski generisan programski kod, to je kod koji se automatski pojavi u form-dizajneru, i on se po pravilu ne dira, ali to je isto programski kod kao i ostali programski kod koji se svojeručno piše. Ovaj programski kod je neophodan kod izvršavanja programa, i ako se on modifikuje može biti vrlo problematično da se ispravi takva greška, pa treba biti vrlo oprezan da se ne menja ili ne briše nešto što vi niste svojeručno napisali u programu.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

TIPOVI PODATAKA

*Takodje postoje i tipovi podataka **array** i **collection** koji su u stvari skupovi varijabli istog tipa.*

Svaka varijabla pripada nekom od nekoliko tipova, u zavisnosti za koji tip podataka se koristi. Npr. varijabla tipa **string** je varijabla koja sadrži niz znakova. Ovo je korisno za memorisanje, tekstualnih varijabli, npr imena, adrese. Šta god da je izmedju duplih navoda, npr. "abc123" je string a ne neki drugi tip, npr. broj. Medjutim, ako želite da memorišete brojeve, i vršite operacije sa brojevima, onda se koriste numeričke varijable, npr tipa **int** tj. skraćeno od integer i ovaj tip označava varijable koje memorišu cele brojeve. Takodje postoji opšti tip varijable , tip nazvan **Object** i ovo su podaci bilo kog tipa, medjutim ovde se mora voditi računa da kompajler ovakve varijable tretira kao opšte varijable, pa se mora dodatno dati instrukcija kompajleru ako se one žele tretirati kao neki poseban tip, npr. kao brojne varijable. Pri tome se koristi instrukcija **cast,** i ovo će biti kasnije detaljnije pomenuto.

Osnovni tj. ugradjeni, *built-in* type, tipovi podataka su **int**, celi broj, **bool**, logička varijabla, **char**, znakovi, **float**, decimalni broj, npr. sledeća deklaracija i instrukcija:

```
int broj;

broj = 14;
```

A složeni tipovi podataka, se definišu kao uslužne klase, *custom classes*, *custom type*, kao deo klasne biblioteke u okviru **.NET Framework.** Takodje postoje i tipovi podataka **array** i **collection** koji su u stvari skupovi varijabli istog tipa.

Ako imamo "u" na početku nekog tipa brojne varijable, to označava unsigned tj. Bez znaka + ili – već samo +. Takodje, tip *char* je u suštini isto broj, jer kompjuteri memorišu znakove kao brojeve. Ima tri decimalna tipa, float sa *preciznošću od 7 cifara, double sa reciznošću od 15 cifara, i decimal sa preciznošću od 29 cifara (za rad sa novcem).*

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UGRAĐENI TIPOVI

Ako imamo "u" na početku nekog tipa brojne varijable, to označava unsigned tj. Bez znaka + ili – već samo +.

Od ugradjenih tipova , ima 9 tipova celobrojnih varijabli, 13 tipova jednostavnih tipova varijabli, dva složena tipa varijabli, string i object, i oni su dole navedeni, kao i njihovi opsezi:

byte, ceo broj od 0 do 255

sbyte, ceo broj od -127 do 128

short, ceo broj od -32768 do 32767

ushort, ceo broj od 0 do 65535

int, ceo broj od -2147483648 do 2147483647

uint, ceo broj od 0 do 4294967295

long, ceo broj od -9223372036854775808 do 9223372036854775807

ulong, ceo broj od 0 do 18446744073709661615

char, znak

float, od 1.5×10^{-45} do $3.4 \times 10^{+38}$

double, od 5.0×10^{-324} do $1.7 \times 10^{+308}$

bool, True ili false tj. DA ili NE

decimal, od 10^{-28} do $7.9 \times 10^{+28}$

string, niz znakova

object, bilo koji tip

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KLJUČNA REČ *NEW*

*Umesto operatora zadavanja =, može se koristiti ključna reč **new.***

Zadavanje vrednosti varijabli može se uraditi pomoću operatora zadavanja =. Npr.

string sss ;

sss = “petar petrovic” ;

ali, deklarisanje varijable i zadavanje vrednosti može se obaviti istovremeno, npr.

int nnn = 5 ;.

Medjutim, umesto operatora zadavanja =, može se koristiti ključna reč ***new,*** npr.

int nnn = new int () ; //create an integer

int[] arr1 = new int[]; //create an array

Example example = new Example(); // create an object

Što znači u prevodu obezbedi mi novi primerak ovog tipa varijable, a posle tipa varijable se nalaze ili male zagrade ili srednje (kvadratne) zagrade, i ove male zagrade podsećaju da se vrši pozivanje tzv. „konstruktora“, što će biti objašnjeno kasnije. A srednje zagrade označavaju niz (tj. vektor) , naime tzv. ***array,*** dakle koriste se ako je varijabla tipa ***array.*** Pri tome, ako je iskorišćena ključna reč tj operator ***new***, onda istoivremeno joj se kao početna vrednost zadaje predefinisana vrednost, *default value*, to nula ili prazno, u zavisnosti od tipa varijable.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

VIDLJIVOST VARIJABLI, *VARIABLE VISIBILITY (SCOPE)*

Ova karakteristika .vidljivost varijable-definiše koji delovi programa mogu da pristupe ili promene vrednost varijable.

Jedna karakteristika varijable je njen tip, *type*, koji smo ranije objasnili. Medjutim, druga važna karakteristika je *scope* tj ***visibility***, vidljivost varijable. Ova karakteristika definiše koji delovi programa mogu da pristupe ili promene vrednost varijable. Ako je npr neka varijabla deklarisana u okviru neke funkcije tj. metode, onda takva varijabla je vidljiva samo u okviru te metode. Ako je, medjutim, neka varijabla deklarisana na nivou klase, pri čemu ovakve varijable se nalaze na početku klase, naime posle klasne deklaracije a pre deklaracija metoda klase, onda, ako je ova varijabla deklarisana kao privatna, ***private***, onda ova varijabla je vidljiva na nivou klase, tj. bilo koja metoda tj. funkcija u okviru te klase može da koristi tu varijablu, ali ako je deklarisana neka varijabla kao javna, ***public***, onda ta varijabla je vidljiva svuda. A ako nije definisana vidljivost, onda po predefiniciji (*default*) takve varijable su privatne. U principu, poželjno je minimizirati vidljivost varijabli, jer to smanjuje mogućnost greški u programu. Postoje takodje varijable specificirane kao ***protected***, tj. zaštićene, i ***internal***, interne varijable, ali ovo će biti objašnjeno kasnije. Posmatrajmo sledeći primer, gde je vidljivost pojedinih varijabli definisana na razne načine. Vidljivost se reguliše pomoću **private, public**, i velikih zagrada **{…}.**

Primer:

....

```
public string string1; //vidljivo svuda u okviru projekta
private string string2; //vidljivo samo u okviru ovog modula tj forme
private void button1_Click(object sender, System.EventArgs e)
{           bool bool1;//vidljivo samo u ovoj funkciji
            if (bool1)     {           int int1;//lokalna varijabla vidljiva samo u ovom if-bloku //etc}
            //kompajler nece ovo da kompajlira jer je int1 nevidljiva izvan if-bloka
            int1 = 1;}
```

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

VIDLJIVOST VARIJABLI-PRIMER

Evo primera koji ilustruje vidljivost varijable.

Evo još jednog primera, gde se pogrešno koristi opseg važenja varijable:

```
class Primer
{void PrvaMetoda(){ int var1; ...........}
void DrugaMetoda(){ var1=50; //greska  }
}
```

I još jedan primer,

```
class Primer{void Metoda1(){polje1=50; //ok}
void Metoda2(){polje1=60;//ok}
int polje1=0;}
```

U jeziku C#, promenljiva definisana klasom (a ne funkcijom) se zove polje, tj. ***field***. Polje, za razliku od lokalnih promenljivih, može se koristiti na nivou klase, tj. za razmenjivanje informacija izmedju metoda u okviru klase. Promenjiva polje1 je definisana u klasi, a izvan meoda Metoda1 i Metoda2, pa je promenjiva polje1 u području važenja klase, i može se koristiti u svim metodama u klasi. Treba reći da kod metoda, prvo se deklariše promenljiva pa se tek onda koristi ta promenjiva, ali, kad je u pitanju klasno polje tj. *class field*, onda to nije slučaj, može se klasno polje deklarisati kasnije, posle njegove upotrebe, jer. kompajler je tako napravljen da pretraži deklaraciju.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STRINGOVI

Za „sabiranje“ tj. nadovezivanje stringova može se koristiti operator plus: +. Takdje mogu se koristiti operatori =, >,<, itd.

Već smo videli kako se deklarišu i inicijalizuju string varijable, npr.

string strIme1 = “Pera”;

string strPrezime1 = “Petrovic“;

string strIme2 = strIme1;

Sa stringovima se može manipulisati. Npr. za „sabiranje“ tj. nadovezivanje stringova može se koristiti operator plus: +. Takdje mogu se koristiti operatori =, >,<, itd. Takodje, postoji čitav niz metoda za rad sa podacima tipa string. Npr.

Compare(string s1, string s2), poredi dva stringa, i vraća 1 ili -1 ili 0, u zavisnosti da li je levi string veći ili manji ili jednake dužine kao desni string,

Copy(), pravi novi primerak stringa,

Insert(), ubacuje string na indeksiranu poyiciju stringa,

Remove(), briše specificirani broj znakova iz stringa,

Replace(), zamenjuje neki znak (na svim mestima u stringu) sa nekim drugim znakom,

Itd.

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MANIPULISANJE STRINGOVIMA

String varijabla i String object je istog tipa, tj. oni su kompatibilni, npr. string varijable mogu koristiti metode klase String.

U mnogim aplikacijama string varijable mogu se tretirati kao *built-in value-type tip* varijable, npr. int ili float, itd. Npr.

int i =1; //deklaracija i inicijalizacija int varijable

string s = “abcd mnlpq“; // deklaracija i inicijalizacija string varijable

Medjutim, string varijable su u stvari klasne varijable, jer u stvari string varijable pripadaju klasi podataka String. Npr.

string s1 = new String();

string s2 = “abcd abcd“;

int intLength = s2.Length;

MessageBox.Show(“you typed: “ + textBox1.Text.Length. ToString( ) + “characters.“);

Gde je Length svojstvo tj. *property* od string varijable, i to je dužina string varijable.

Postavlja se pitanje, da li su string varijable vrednosni tip ili klasa podataka. U stvari, String je klasa,ali pošto se string varijable puno koriste u programima, a i istorijski gledano string varijable se koriste dok još klase nisu postojale, onda ključna reč string (počinje sa malim s) je u stvari sinonim sa klasom String (veliko S), npr.

String s1= “abcd abcd”; //zadaje se vrednost String objektu

string s2 = s1; //string varijabli zadaje se vrednost String objekta

Dakle mogu se definisati objekti u klasi String, a isto tako mogu se definisati jednostavne string varijable. Medjutim, string varijabla i String object je istog tipa, tj. oni su kompatibilni, npr. string varijable mogu koristiti metode klase String, itd.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONSTANTE

U jeziku C# mogu se koristiti varijable koje imaju konstantnu vrednost tj. varijable čije se vrednost ne mogu menjati.

Evo primera definisanja i upotrebe konstante:

```
const string Temp273 = “ – 273 stepena“;
MessageBox.Show(Temp273);
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

NULL-VREDNOST

Da biste inicijalizovali referentnu promenjivu, npr. objekat, može se koristiti specijalna vrednost null.

Kad se deklariše neka promenljiva, uvek je dobro da se ona inicijalizuje. Npr. kod operisanja sa vrednosnim tipovima,

int int1 = 0;

double double1 =0.0;

Ali,da biste inicijalizovali referentnu promenjivu, npr. objekat, može se koristiti specijalna vrednost *null,* npr.

Circle circle1 = null;

Vrednost null se ne može zadavati vrednosnom tipu, npr. pogrešno je napisati

int int1 = null; //greska, nije ovo dozvoljeno

ali je dozvoljeno ovo

int? int1 = null; //ovo je dozvoljeno, nije greska

takodje i ovo je dozvoljeno

int? int1 = null;

int1 = 50;

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

NIZOVI, ARRAYS

Niz tj. Array, to je* red *ili uredjen* skup *koji se sastoji od jedne kolone ili nekoliko kolona jednih pored drugih poredjanih elemenata, predstavlja poseban tj. specijalan tip varijabli.

vrlo često je korisno, da se pod jednim imenom smesti čitav niz podataka, istog tipa. Npr.

Npr., dole su dve instrukcije zadavanja vrednosti *Array* varijabli:

Temperature[0] = 18.5 ;

Temperature[1] = 18.9 ;

Pri tome, prethodno je potrebno deklarisati *array*-varijablu, npr.

float[] Temperature = new float [24] ;

gde se zadaje i dimenzija ove *Array* varijable.

Pri tome, prva pozicija se obeležava sa indeksom 0, pa ako hoćemo *Array* dimenzije 24, onda u stvari to je *Array* čiji indeks ide od 0 do 23. Jedna od prednosti uvodjenja *Array* varijable, pored toga što se koristi jedno ime za više varijabli, je i to da se mogu koristiti petljaste konstrukcije za procesiranje *Array* varijabli, Npr.

i=0;

Do { DTemp[i] = Temp[i] -20 ; i=i++;} while(i<24) ;

Videli smo da se koristi indeksiranje niza, da bi se označili pojedini elementi niza. Nizovi mogu biti bilo kog tipa, npr. niz objekata, *array of objects*. Dalje, u C# postoji jedna specijalna *Array* varijabla, specijalni tip niza, naime specijalan tip podataka („kolekcija“):

ArrayList, koji ima prednost da automatski menja dimenziju po potrebi. Elementi nizova ne moraju biti samo primitivni tipovi podataka, već mogu biti i elementi strukture ili enumeracije ili klase (objekti), npr. struktura Student,

Student[] studenti;

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

DUŽINA NIZA

Ali dimenzija niza ne mora da bude konstantna.

Pomoću ključne reči new mogu se zadati vrednosti nizu, npr.

```
int[]  arr1;
arr1 =new int[5];
```

na taj način se inicijalizuje niz, tj. njegovi članovi, i to ili 0 (nula) ili *null* (specijalnia vrednost „null“ se koristi za referentne tipove) ili *false*, u zavisnosti da li je u pitanju numerički tip ili referentni tip ili bulov tip, *bool type*. Takodje na ovaj način se definiše i dimenzija niza.

Ali dimenzija niza ne mora da bude konstantna, evo primera gde se ona učitava sa konzole,

```
int sizeArray1 =  int.Parse(Console.ReadLine());
int[] arr1 = new int [sizeArray1];
```

Niz se može inicijalizovati i na sledeći način,

```
int[] arr1 = new int[5]{1,5,7,8,9};
```

ili

```
int[] arr1 = {1,5,7,8,9};
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

NIZOVI-SVOJSTVO LENGTH

Nizovi su referentni tipovi. Neki niz je u stvari instanca klase System.Array.

Nizovi su referentni tipovi. Neki niz je u stvari instanca klase System.Array. Dakle, neka nizovna promenjiva sadrži i referencu (adresu) na instancu niza. Nizovi imaju ugradjena svojstva i metode, *properties/methods*. Svaki niz nasledjuje metode i svojstva od klase System.Array u radnom okviru .NET. Npr. svojstvo Length omogućuje da se dobije dužina niza. Evo primera kretanja kroz niz, gde se na konzoli ispisuju vrednosti elementa niza,

```
int[] arr1 = {5, 7, 4, 3};

for (int index =0;  index<arr1.Length; index++)

{

int arr1element = arr1[index];

Console.WriteLine(arr1element);

}
```

Length je svojstvo a ne metoda, i zato ne trebaju zagrade kad pozivate svojstvo.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Primer sa message-box-om

02

PRIMER PROGRAMA

Ovde posmatramo jedan program koji koristi if-else instrukciju i message-box instrukcije.

Evo jednog jednostavnog programa, kao ilustracije, koji se sastoji od 3 koraka.

1. Postaviti LABEL, TEXTBOX i BUTTON na FORMU. Zadati tekstualne karakteristike, *text properties*, kao što je dole prikazano.
2. Duplo kliknuti BUTTON da bi se otvorio kodirajući-editor za dogadjaj Click.
3. Ukucati programski kod kao što je dole prikazano, izmedju dve velike zagrade { ....}, a ostatak programskog koda se generiše automatski.

U jeziku C#, potrebno je svaku varijablu deklarisati, tj. označiti njen tip, kao i njeno ime, jer ima više tipova varijabli. Npr. instrukcija

string var1;

kaže kompjuteru tj. programu kompjutera da promenljiva pod imenom var1 je tipa ”string”, niz, niz znakova, tj. da je to tekstualna varijabla koja se sastoji od niza tekstualnih znakova. Drugim rečima, neki podatak koji se sastoji od niza znakova će biti memorisan pod imenom var1.

Instrukcija:

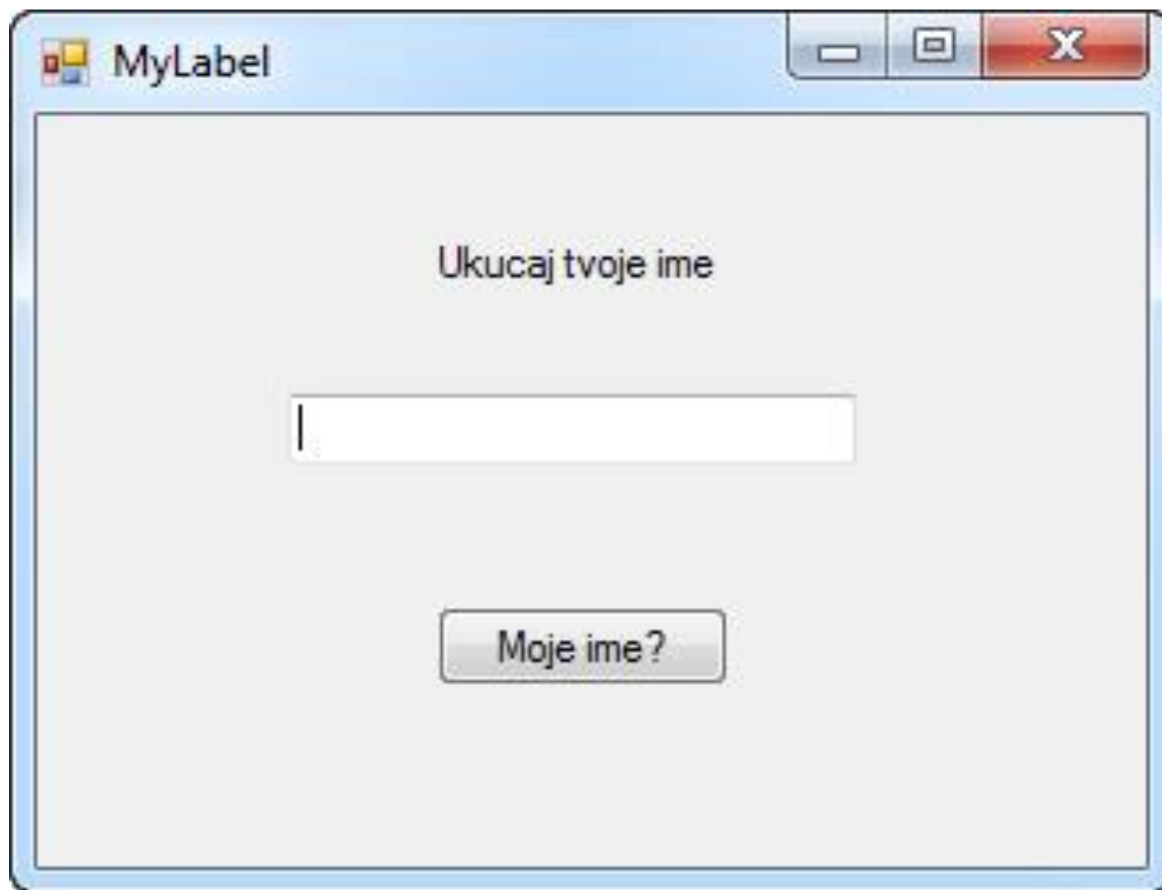
var1 = textBox1.Text;

nalaže da **tekst-karakteristika od textBox1** se zadaje i memoriše kao vrednost varijable var1.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

FORMA SA DUGMETOM, NALEPNICOM I TEKSTBOKSOM

Na slici je FORMA sa dugmetom, nalepnicom i tekstboksom.

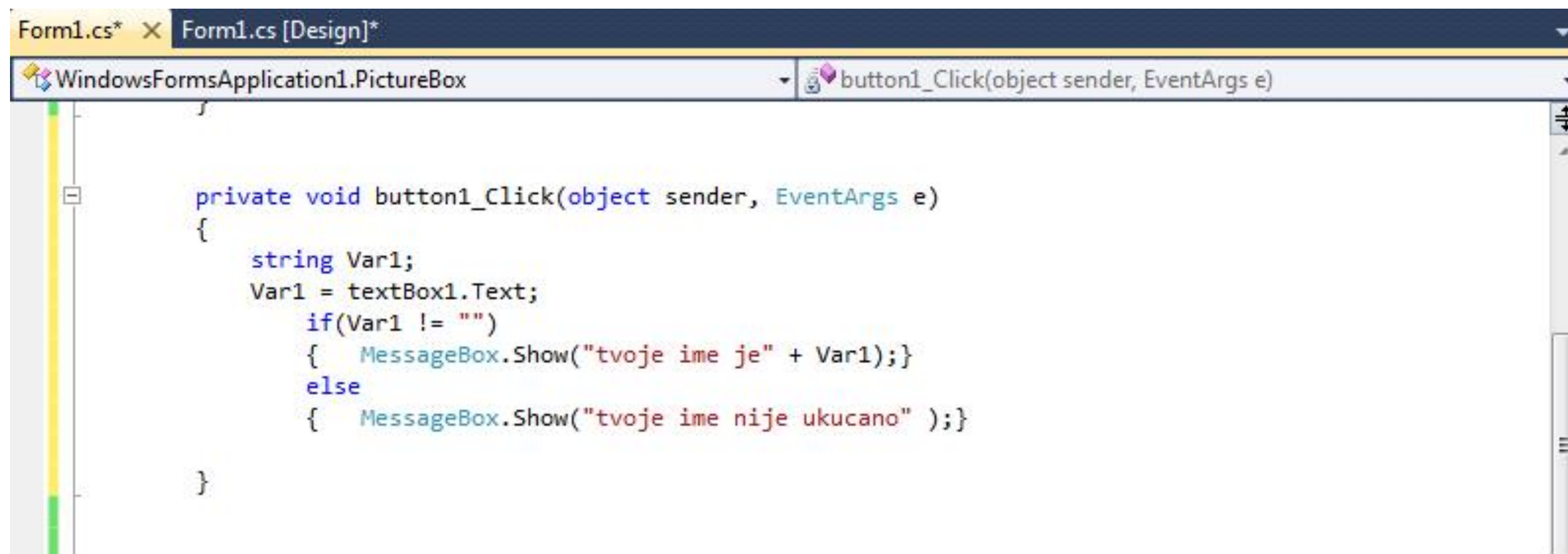


Sl.1: forma

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PROGRAM U KODIRAJUĆEM EDITORU

Slika prikazuje program u kodirajućem editoru.



Sl.2: program u kodirajućem editoru

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MESSAGEBOX-PRIMER

Ovde ilustrujemo upotrebu message-box-a.

A instrukcija:

if(var1 != “”) ....

poredi var1 i prazan niz ””, i ako var1 nije prazan niz, npr. ako je ukucano MARKO, onda se pojavljuje poruka u **message-box** prikazujući poruku koja je opisana izmedju zagrada od:

Message.Show( )

A to je:

“tvoje ime je” + var1,

Npr.:

tvoje ime je MARKO

A ako var1 je prazan niz onda odgovarajuća poruka se pojavljuje u **message-box**:

tvoje ime nije ukucano

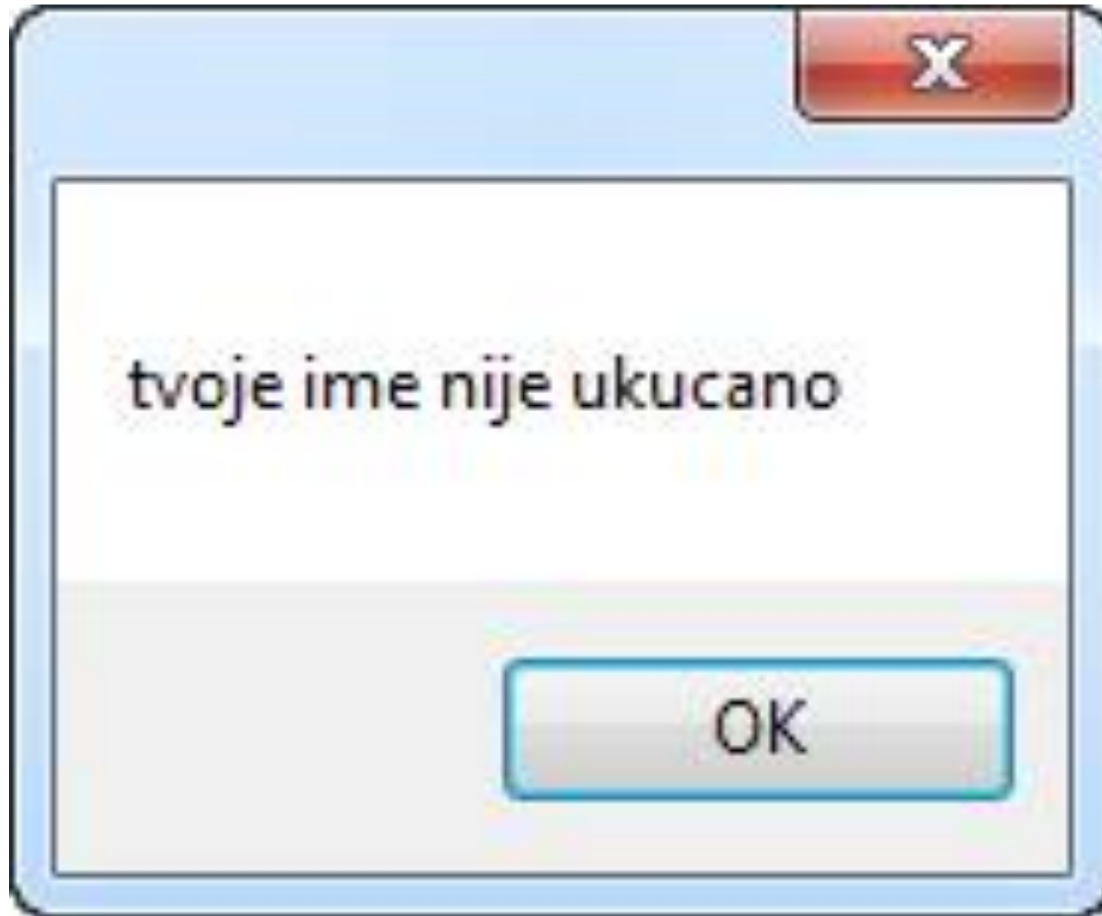
Pri tome, prazan niz se označava sa ““.

Inače, *private* označava da je to podprogram kome se može pristupiti samo iz te dotične FORME, i nijedne druge. A void da se radi o metodi bez *return* instrukcije.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MESSAGEBOX

Na slici je MessageBox.



Sl.3: messagebox

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Primer sa load-event

03

LOAD-EVENT

Ovde imamo primer programa sa load-event.

Posmatrajmo primer „forme“, gde imamo jednu „nalepnicu“ tj. *label* na formi.

Primer:

1. Postaviti LABEL na FORMU.

2. Duplo kliknuti FORMU da bi se otvorio CODE-EDITOR na LOAD-EVENT. Pri tome duplo kliknuti na pozadini FORME, a ne na NALEPNICI, LABEL,

3. Ukucati sledeći kod izmedju dve velike zagrade koje se pojave automatski:

```
private void Form1_Load(object sender, System.EventArgs e)
{                    DayOfWeek day0;
                     day0 = DateTime.Now.DayOfWeek;
if ( day0 == DayOfWeek.Saturday )
{   label1.Text = “Subota“;}
else
{  label1.Text = “nije Subota“;}  }
}
```

4. Egzekutovati program

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

POGODNE .NET FRAMEWORK FUNKCIJE

.NET Framework obuhvata niz korisnih predefinisanih funkcija i promenljivih.

Gornji program koristi neke pogodne .NET Framework funkcije i promenljive. Gde promenljiva

„Now“

sadrži trenutno vreme i datum.

DayOfWeek predstavlja dan. Kada se startuje aplikacija, gornji program će se izvršiti, i pojaviće se „forma“ . Na sličan način može se napraviti program koji će štampati na C#-formi „nalepnicu“ (label1) na kojoj piše „Today is working day“ ili „Today it is weekend“.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STRUKTURA „IF ....ELSE“

Struktura „if ....else“ je logička struktura koja omogućuje da se u program ubaci odlučivanje tj. neka vrsta veštačke inteligencije.

Struktura „if ....else“ je logička struktura koja omogućuje da se u program ubaci odlučivanje tj. neka vrsta veštačke inteligencije. Naime, testira se odredjeni uslov, i ako je ispunjen uslov onda se izvršava jedan blok, a ako nije ispunjen taj uslov onda se izvršava drugi blok. Primetimo sledeće: uslov je dat izmedju dve male zagrade, a deo strukture else{…………..} je opcioni, tj. može se primeniti ili ne po potrebi.

Npr.

```
if ( day0 == DayOfWeek.Saturday )
{   label1.Text = “Subota“;}
else
{  label1.Text = “nije Subota“;}  }
```

Varijacija ove strukture je struktura gde se može testirati nekoliko različituih uslova: if .....else if........else, tj. testirati seriju blokova sa različitim uslovima.

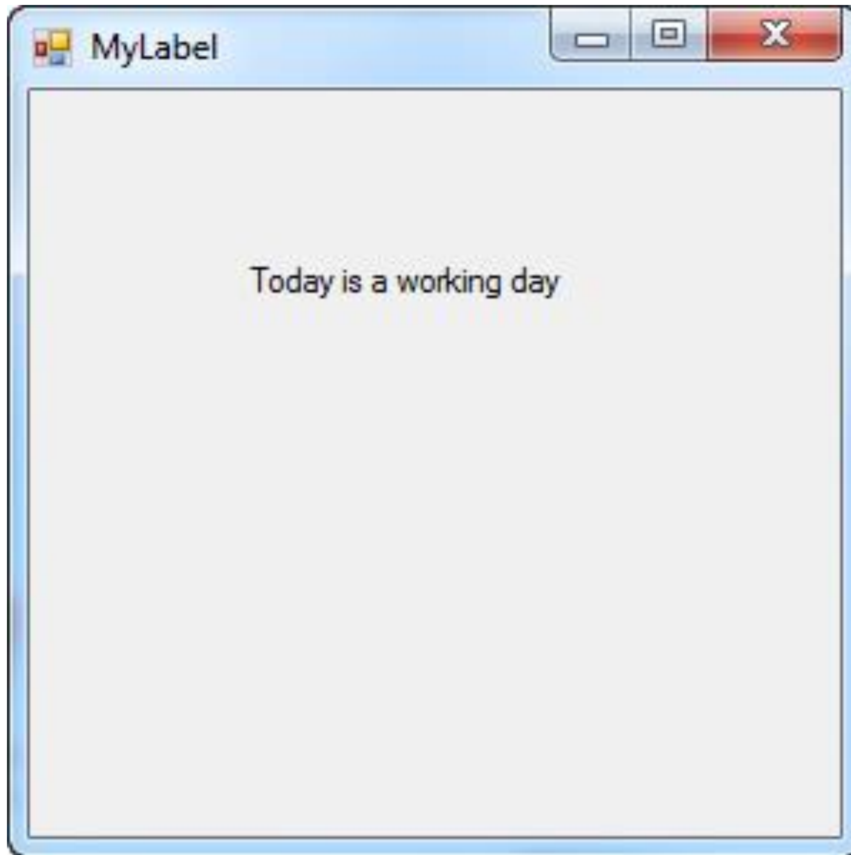
A if-instrukcija (bez „else“) ima sintaksu:

```
if(.....) {.....}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

„FORMA“ SA NATPISOM „TODAY IS A WORKING DAY”

Na slici je „Forma“ sa natpisom „Today is a working day”.



Sl.1: forma

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Metode Parse() i ToString()

04

PRIMER-*NAME PROPERTY*.

Ovde imamo primer gdese koristi Name property.

Ovde posmatramo primer koji opisuje kako da se pomoću Visual C# napravi tj. izprogramira „računska mašina“ tj. kalkulator za sabiranje dva broja, i prikazivanje rezultata tog sabiranja, kao što je dole prikazano. Jednostavno se može, ovaj primer proširiti tako, što će se dodati jedno dugme za oduzimanje dva broja, još jedno dugme za množenje dva broja, i najzad jedno dugme za deljenje dva broja, ukupno četiri dugmeta. Dakle imamo dva *textbox* objekta, 3 *label* objekta, i 4 *button* objekta (Add, Subtract, Divide, Multiply). Pri tome smo koristili alate Toolbox i Property window, i edituje se:

Name property.

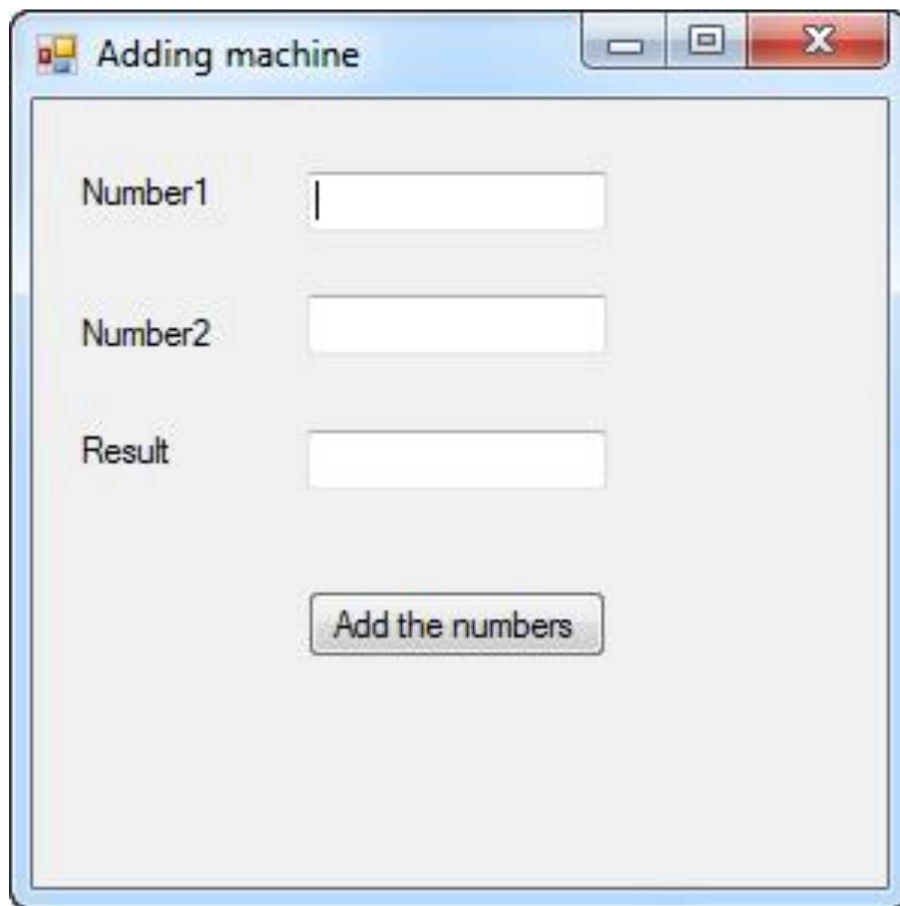
Imena za 7 aktivnih C# kontrola zadajemo kao: txtFirstNumber, txtSecondNumber, btnAdd, btnSubtract, btnDivide, btnMultiply, lbResult, gde txt podseća da je to *textbox* a btn da se radi o *button* objektu.

Imamo onda „kalkulator“ za 4 aritmetičke operacije. Korisnik ukucava dva broja, a rezultat se prikazuje na ekranu. Ovde, ulazni podaci (ukucani od strane korisnika aplikacije), kao i izlazni podaci (podaci prikazani na ekranu) su *string* varijable, dok se proračun obavlja sa decimalnim brojevima tipa *float* (*floating point numbers*).

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

„*ADDING MACHINE*“ NAPRAVLJENA POMOĆU VISUAL C# FORME

Na slici je „Adding machine“ napravljena pomoću Visual C# forme.



Sl.1: forma-kalkulator

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

METODA BTNADD_CLICK()

Dole je prikazan programski kod za metodu btnAdd_Click().

Dole je prikazan programski kod za metodu btnAdd_Click() koja računa i prikazuje rezultat na *label* objektu lbResult (posle duplog kliktanja dugmeta Add, ukucavamo instrukcije u *Click event handler*):

```
private void btnAdd_Click(object sender, System.EventArgs e)

{

float fNumber =float.Parse(txtFirstNumber.Text);

float sNumber =float.Parse(txtSecondNumber.Text);

float res =  fNumber  + sNumber;

lbResult.text = "Answer:" + res.ToString();

}

.........
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

INICIJALNA VREDNOST VARIJABLI

Ako ne zadate inicijalnu vrednost varijabli, kompajler će javiti kompilacionu grešku, i neće doći do kompilacije takvog programskog koda.

Ako posmatramo primer: int nnn1; nnn1 = 5;, ovde se varijabli pod imenom nnn1 zadaje vrednost 5, što znači da će vrednost tj podatak 5 biti storniran tj. memorisan pod imenom nn1, i može se po potrebi pročitati vrednost tog podatka. Ime varijable označava da se vrednost tog podatka može menjati u toku izvršavanja programa. Zadavanje vrednosti tj. konkretnih podataka varijablama je veoma važan deo programiranja. Potrebno je znati da ako se deklariše neka varijabla, ona nema startnu tj. inicijalnu vrednost tj. inicijalni podatak, znači, nije dovoljno deklarisati varijablu, već je potrebno i zadati inicijalnu vrednost. Medjutim, ako ne zadate inicijalnu vrednost varijabli, kompajler će javiti kompilacionu grešku, i neće doći do kompilacije takvog programskog koda. Npr. ako imate sledeći programski kod, gde se deklariše celobrojna varijabla „nnn“, i zatim se pokušava njeno korišćenje a da prethodno nije inicijalizovana ta varijabla, takav programski kod neće biti kompajliran od strane kompajlera:

```
int nnn, int nnn1;

nnn1 = nnn;
```

ili

```
int yy;

if (yy == 0){......}; //nece doci do kompilacije
```

Ako želimo da radimo sa velikim brojevima, onda umesto *float type* možemo da koristimo *double type*. Na potpuno sličan način mogu se napraviti metode za oduzimanje, množenje i deljenje, naime:

operator „ –„ se koristi za oduzimanje,

operator „ *„ se koristi za množenje,

operator „ /„ se koristi za deljenje.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

METODE PARSE() I TOSTRING()

Primetimo da smo koristili dve zgodne C# metode, naime: metoda type.Parse(), koja učitava string type i konvertuje u željeni tip podataka. I metoda ToString().

A metode su btnMultiply_Click(),btnDivide_Click(), btnSubtract_Click(). A kontrole se zovu, txtFirstNumber, txtSecondNumber, btnAdd, btnSubtract, btnDivide, btnMultiply i lbResult.

Npr.,

```
private void btnMultiply_Click(object sender, System.EventArgs e)
{
        float fNumber =float.Parse(txtFirstNumber.Text);
float sNumber =float.Parse(txtSecondNumber.Text);
float res = fNumber * sNumber;
        lbResult.text = “Answer:“ + res.ToString();
}
```

Možemo tartovati aplikaciju, ali pre kliktanja dugmeta treba ukucati brojeve u textbox objekte, inače program neće raditi.

Primetimo da smo koristili dve zgodne C# metode, naime:

metoda type.Parse(),

koja učitava *string type* i konvertuje u željeni tip podataka, npr *float.Parse(),*

i metodu variable.*ToString(),*

koja konvertuje neki tip podataka u *String type*.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

„BELI PROSTOR“

Zgodno je koristiti takodje tzv „beli prostor“, white space, jer omogućuje lakše razumevanje programa. Velike zagrade se koriste da označe blokove linija koje se moraju pročitati zajedno.

Zagrade i tačka-zarez se koriste da označe pojedine blokove programa. Naime, programi se izvršavaju liniju po liniju, pa je potrebno označiti kraj linije, i za to se koristi tačka-zarez. Program u stvari može da tretira i nekoliko linija kao jednu liniju, sve dok se ne pojavi tačka-zarez.

Zgodno je koristiti takodje tzv „beli prostor“, *white space*, jer se on ignoriše od strane programa, a omogućuje lakše razumevanje programa. S druge strane, velike zagrade se koriste da označe blokove linija koje se moraju pročitati zajedno. Npr. početak i kraj klase se označavaju pomoću velikih zagrada, početak i kraj funkcije, početak i kraj bloka posle if-instrukcije. Npr. ako se ne koriste velike zagrade onda if-instrukcija dozvoljava samo jednu liniju:

if (var = =2) var = 1;

Dok:

if (var = =2) {var = 1; var2 = 5;}

obuhvata dve instrukcije.

Da napomnemo, operatori modulus %, i inkrementalni operator ++ (+1)i dekrementalni operator - - (-1) važe u jeziku C#. Npr. 54%13 = 2 (ostatak deljenja dva broja).

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Operatori u C#

05

LOGIČKI OPERATORI

Jezik C# pruža pored aritmetičkih operatora akodje i logičke operatore, i to dve vrste, logički operatori poredjenja numeričkih varijabli, i logički operatori za bulove varijable.

Jezik C# pruža pored aritmetičkih operatora akodje i logičke operatore, i to dve vrste, logički operatori poredjenja numeričkih varijabli, i logički operatori za bulove varijable. U donjoj tabeli su prikazani logički operatori poredjenja, i oni vraćaju kao rezultat bulovu varijablu, dakle ili „*true*“ ili „*false*“.

Table The Logical Comparison Operators

Operator	***Operator Is True If . . .***
a == b	a i b imaju iste vrednosti
a > b	a je veće od b
a >= b	a je veće od ili isto sa b
a < b	a je manje od b
a <= b	a je manje od ili isto sa b
a != b	a nije isto sa b

Evo ednog primera logičkog poredjenja,

```
int m = 50;
int n = 60;
bool b1 = m > n;
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

BULOVE VARIJABLE

Na bulove varijable mogu da se primene logički operatori OR i AND i NOT.

Na bulove varijable mogu da se primene logički operatori OR i AND i NOT. U donjoj tabeli su dati ovi operatori.

Table The Compound Logical Operators

Operator	***Operator Is True If . . .***
!a	a je netačno.
a && b	a i b su tačni
a || b	a ili b je tačno

Evo jednog primera:

```
float f1;
float f2;
f1 = 10;
f2 = f1 / 3;
bool b1 = (3 * f2) == f1;
f1 = 9;
f2 = f1 / 3;
bool b2 = (3 * f2) == f1;
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRIORITET OPERATORA

Bolje je koristiti zagrade da bi program bio jasniji, nego oslanjati se na prioritet tipa operatora

I jednog primera sa logičkim operatorima:

v1=v2 >=0&& v2<=100;

v1=(v2 >=0)&& (v2<=100);

Obe prethodne instrukcije daju isti rezultat, jer prioritet operatora && je manji od prioriteta operatora >= I <=. Ipak, druga instrukcija je mnogo jasnija jer ima zagrade, pa je bolje koristiti zagrade da bi program bio jasniji, nego oslanjati se na prioritet tipa operatora.

Prioritet operatora je:

(), postfiks ++, postfiks --, !, + sign,- sign, ++ prefix, -- prefix, *, /, %, <,<=,>,>=,==,!=,&&,||,=.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Konzola

06

PRVA KONZOLNA APLIKACIJA:

Obično se u C# razmatraju Windows-aplikacije. Medjutim potrebno je da znamo da pravimo i konzolne aplikacije.

Prva konzolna aplikacija:

Obično se u C# razmatraju Windows-aplikacije. Medjutim potrebno je da znamo da pravimo i konzolne aplikacije. Kod Windows-aplikacije, korisnik komunicira sa programom koristeći „miš“ i GUI, dok kod konzolne aplikacije, korisnik komunicira sa programom preko tastature. Dakle, konzolna aplikacija je aplikacija koja koristi komandni prozor za svoje pokretanje, i nema grafički korisnički interfejs. Ako koristite npr. Visual Studio 2008, Standard Edition ili Professional Edition, treba uraditi sledeće:

1. posle pokretanja Visual Studio-a, na meniju File, uperite „miš“ na New, zatim Project, zatim izaberete u Project types izaberete Visual C#, a u Templates izaberete ikonu:Console Application
2. U polju Location, otkuca se lokacija tj. direktorijum gde se želi smestiti program. U polju Name otkuca se ime fajla tj. programa TextHello. Zatim, OK.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRVA KONZOLNA APLIKACIJA:

Visual Studio otvara projekat koristeći šablon, template, za konzolnu aplikaciju.

3. Visual Studio otvara projekat koristeći šablon, *template*, za konzolnu aplikaciju, i pokazuje sledeći početni programski kod koji se automatski dakle generiše:

```
using System;
using System.Collections.Generic;
    using System.Ling;
    using System.Text;

    namespace TextHello
{
class Program
{
            static void Main(string[] args)
            {
            }
}
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRVA KONZOLNA APLIKACIJA

Treba ubaciti nove instrukcije. Npr. Console.WriteLine(“Hello Serbia“);

4. Sada treba napisati program, pri tome voditi računa da mala i velika slova treba razlikovati. Treba ubaciti nove instrukcije tako da sada program izgleda ovako:

```
using System;
using System.Collections.Generic;
using System.Ling;
using System.Text;

namespace TextHello
{
class Program
{
        static void Main(string[] args)
        {
                Console.WriteLine(“Hello Serbia“);
        }
}
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRVA KONZOLNA APLIKACIJA:

Umeniju Debug izaberite Start Without Debugging.

5. Umeniju Debug izaberite Start Without Debugging. Ovo rezultira u otvaranje komandnog prozora, tj. „konzole“, i program se pokreće. Poruka:

hello world, hi i Press any key to continue

se pojavljuje na ekranu, program čeka da pritisnete bilo koji taster na tastaturi da bi se završio.

Da smo izabrali Start Debugging a ne Start Withiut Debugging, onda se ne b pojavila poruka Press any key to continue, već bi se program odmah završio tj. komansdni prozor bi se odmah zatvorio.

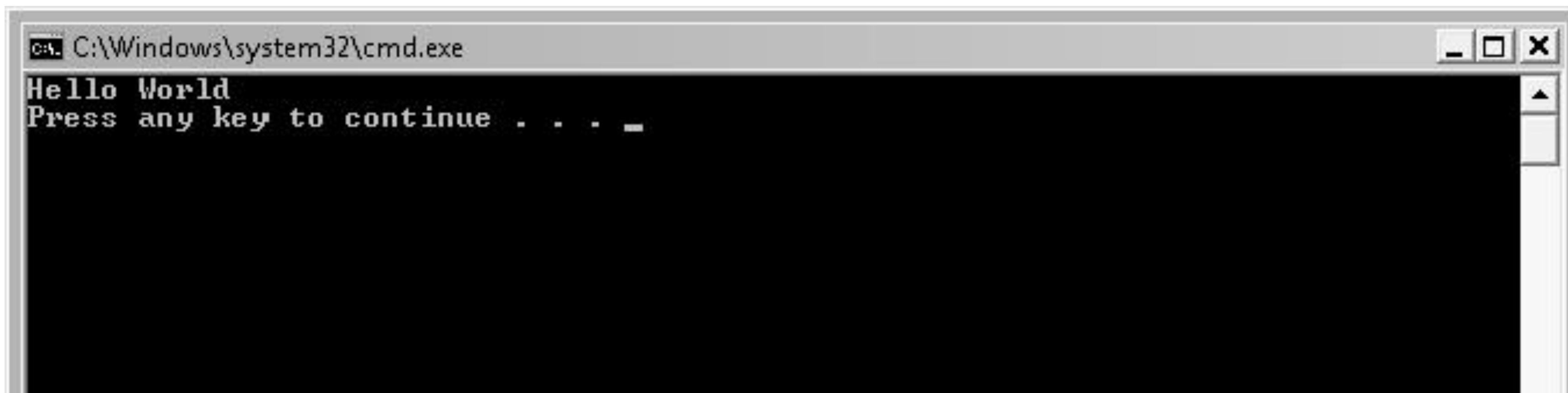
Ovde vidimo komandni prozor, sa porukom “Hello World“, dakle malo drukčiju od one u gornjem programu, dakle ako želimo tu poruku, onda umesto “hello world, hi“ treba otkucati “Hello World“. U gornjem primeru koristili smo metodu:

Console.WriteLine()

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KOMANDNI PROZOR GDE SE KOMANDUJE PREKO TASTATURE („KONZOLA“)

Na slici vidimo komandni prozor gde se komanduje preko tastature (tzv. „konzola“).



sl1: konzola

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

UNOS PODATAKA PREKO TASTATURE

Ako želimo da unesemo neki podatak preko tastature, onda se može koristiti neka metoda za unos podataka.

Najednostavniji konzolni program je vrlo jednostavan, i samo šalje poruku na ekran"Hello Serbia". Medjutim, ako želimo da unesemo neki podatak preko tastature, onda se može koristiti primer sledećeg programa:

Gde se koristi metoda:

```
Console.ReadLine();
```

Ovde se koristi instrukcija :

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

za učitavanje preko „konzole". Tj. metoda:

Console.ReadLine()

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

CONSOLE.READLINE()

*Metoda **Console.ReadLine()** služi za učitavanje tj. unos podataka preko tastature.*

```
using System;
namespace ConsoleAppTemplate
{ // velike zagrade
// class Program
public class Program{// ovde pocinje program
// Main() method
static void Main(string[] args)
{// poruka korisniku
Console.WriteLine(“ Name, please:”);
// ucitaj ime
string sName = Console.ReadLine();
// poruka korisniku
Console.WriteLine(“Hello “ + sName);
Console.WriteLine(“Press Enter to terminate the program”);
Console.Read();// kraj programa Main() }
} // kraj class Program
} // kraj namespace ConsoleAppTemplate
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MessageBox dijalog

07

„DO“ I „WHILE“

*Postoje npr. dve logičke konstrukcije tj strukture, **„do“ i „while“** za ponavljanje instrukcija.*

U mnogim programima potrebno je neki blok instrukcija ponoviti čitav niz puta. Postoje npr. dve logičke konstrukcije tj strukture, **„do“ i „while“** za ponavljanje instrukcija. Naime u strukturi „while“ imamo uslov „while“:

while(……………..) da prethodi bloku koji se ponavlja: {.............................}.

Dok u strukturi „do“, imamo blok koji se ponavlja:

do {.............................}, i imamo test while(); na kraju petlje,

pa takva petlja se mora izvršiti bar jednom. Ako napravite petlju koja se beskonačno vrti, onda mora da se pribegne Task Manager-u, ili, alternativno, treba otići u Visual Studio i izabrati Debug i Stop debugging.

Postoji instrukcija if(...) **break** pomoću koje se može izaći iz petlje, i koju treba koristiti za izlazak iz petlje, da bi se izbegle greške u kompilaciji. A naredba if(....) **continue** omogućuje da se preskoči jedna iteracija tj. krug u kruženju. Takodje postoji instrukcija if(...) goto, koja omogućuje skok u programu, npr. skok izvan ugneždene petlje.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRIMER: MESSAGEBOX DIJALOG

Evo primera sa messagebox dijalogom.

postavimo „dugme“ na „formu“

duplo kliknuti „dugme“, i upisati sledeće programske instrukcije za Click event handler:

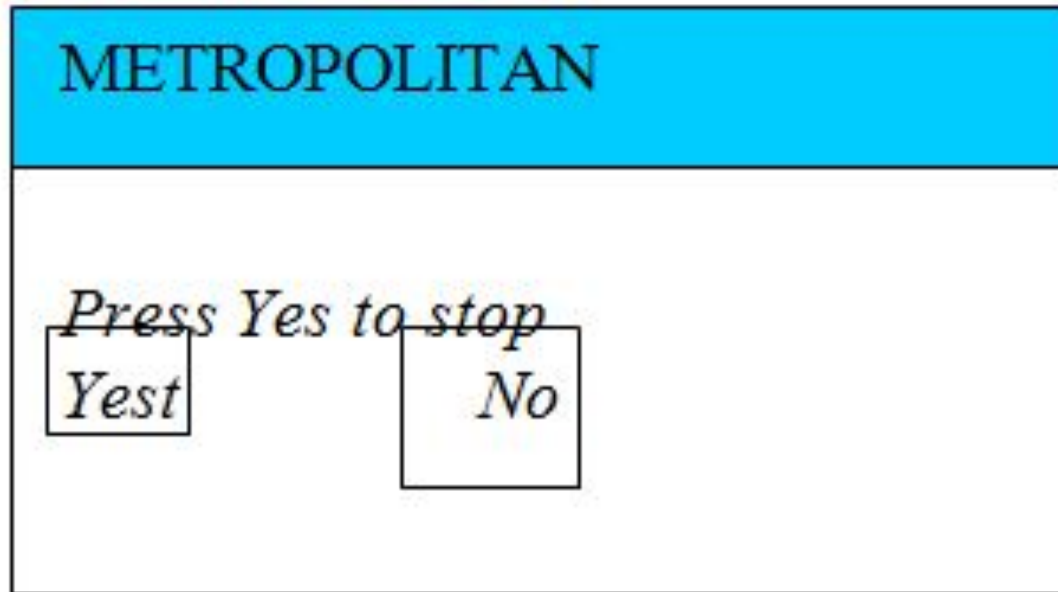
```
DialogResult index;
do
{
index = MessageBox.Show(“Press Yes to stop“, „METROPOLITAN“, MessageBoxButtons.YesNo;
}
while (index != DialogResult.Yes);
```

3. Izvršiti aplikaciju, i kliknuti „dugme“. Pojavljuje se MessageBox dijalog. Ako kliknete Yes, dijalog nestaje, a ako izaberete No, dijalog nestane ali se odmah pojavi ponovo.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MESSAGEBOX DIJALOG

Slika ilustruje MessageBox dijalog.



Sl1: messageboxdialog

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ListBox metode

08

LISTBOX METODE

Posmatrajmo primer gde koristimo ListBox metode.

Posmatrajmo primer gde računamo godišnju kamatu na neku sumu novca.

Primer:

Postaviti „nalepnicu“, „tekstboks“, „dugme“ i „listboks“ na „formu.

Duplo kliknuti „dugme“, i ukucati sledeći programski kod za Click event

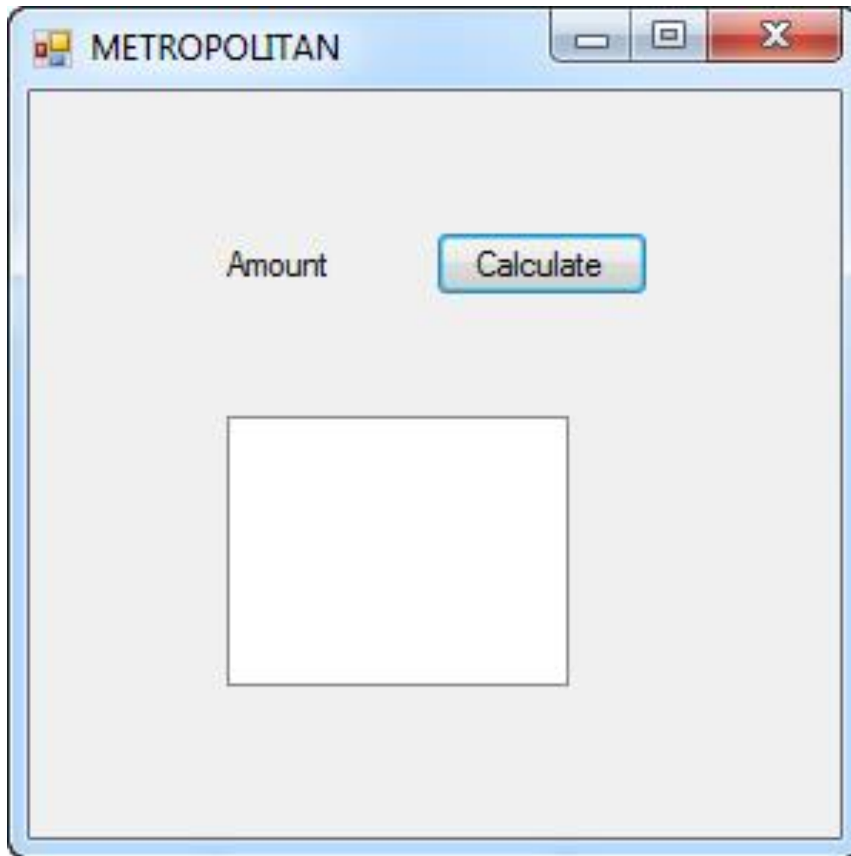
```
decimal value;
listBox1.Items.Clear();
value = decimal.Parse(textBox1.Text);
for (int index=1; index<5; index++)
{
decimal money = value*index/100;
listBox1.Items.Add(index.ToString() + money.ToString());
}
```

3. Da bi se koristila aplikacija, potrebno je da se ukuca broj u „tekstboks“, i onda kliknuti dugme „calculate“.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

„FORMA“ SA „NALEPNICOM“, „TEKSTBOKSOM“, „DUGMETOM“ I „LISTBOKSOM“

Na slici je „forma“ sa „nalepnicom“, „tekstboksom“, „dugmetom“ i „listboksom“ .



Sl.1: formasa ListBox-om

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJA *FOR*

*Petlja **for** je slična sa petljom „do“ i „while“, ali se razlikuje po tome što koristi jednu varijablu tj. brojač.*

Petlja ***for*** je slična sa petljom „do“ i „while“, ali se razlikuje po tome što koristi jednu varijablu tj. brojač koji pamti koliko je iteracija tj krugova uradjeno. Inače, petlje se mogu postavljati jedna unutar druge, tj. da se „ugnezdi“ neka petlja unutar druge petlje, *nest*, ali treba biti oprezan, naročito ako se kombinuje sa if-else blokovima. U okviru for-iskaza, npr for (int i=1; i<10; i++) prvi deo označava varijablu, i to celobrojnu varijablu, koja se koristi kao brojač, npr varijabla i, kao i njenu početnu vrednost, a drugi deo opisuje uslov, npr i<10, i ako je uslov ispunjen onda se petlja izvršava, a treći deo opisuje kako da se brojač menja, npr ++ znači da se brojač povećava u koracima po 1 tj. i = i+1; ali može se koristiti i alternativni izrazi, npr i = i + 5. Petlje mogu biti ugnežđene.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Funkcije u C#

09

FUNKCIJE (METODE):

Nije potrebno staviti tačka-zarez posle definicije funkcije.

U programskom jeziku C#, postoje dve vrste funkcija tj. podprograma, funkcija koja vraća tj. isporučuje (pomoću ***return***) neki podatak u glavni program tj. u program iz koga je pozvana, i funkcija koja ne vraća neki podatak, i ovakva funkcija se zove ”void”.
Ako želite da napišete neku funkciju,

- *prvo je potrebno da deklarišete tip varijable koja se vraća od strane te funkcije,*
- *i potrebno je definisati ime funkcije,*
- *sintaksa je: tipRezultata ImeMetode(lista parametara){lista instrukcija}*

npr.

```
string Helloworld()
{                    return "helloworld";}
```

Pri tome, nije potrebno staviti tačka-zarez posle definicije funkcije, kao što vidimo gore da nema tačka-zarez, što znači, da ako se stavi tačka-zarez, doći će do greške u kompilaciji programskog koda. A velike zagrade označavaju početak ii kraj programskog koda funkcije, dok, male zagrade posle imena funkcije, obuhvataju listu ulaznih podataka za tu funkciju, tj. podataka koji se isporučuju tj. predaju funkciji, ***passed to the function***, i ovi ulazni podaci funkcije se zovu parametri funkcije tj. argumenti funkcije. A u slučaju da nema ulaznih podataka tj. parametara funkcije, ipak se stavljaju male zagrade, ali izmedju tih zagrada nema onda parametara.

Predavanje vrednosti parametara tj. vrednosti argumenata kao ulazni podatak funkciji je od suštinske važnosti za neku funkciju, npr. pogledajmo funkciju koja broji znakove u nekoj reči;

```
int CountLetters(string word)
{                    return word.Length;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

POZIVANJE METODE

Bitno je znati sintaksu pozivanja metode.

Npr. funkcija je tipa void:

```
public static void Output()
{Console.WriteLine(“function-example”);
}
```

Dakle iskaz *return* se nalazi na kraju metode, i posle toga se ne stavlja više ništa, osim velike zagrade. Ako metoda ne vraća nikakvu vrednost, onda nema *return*, a metoda se označava kao metoda tipa *void*.

Bitno je znati sintaksu pozivanja metode. Naime, u okviru neke funkcije može se pozvati neka druga funkcija. Ova sintaksa glasi,

rezultat = NazivMetode (argument1, argument2, argument3, ...);

ili jednostavno

NazivMetode (argument1, argument2, argument3, ...);

ako je to „void“ metoda. Ovo je klasično pozivanje funkcija, a pozivanje funkcija tj.metoda u okviru o.o. programiranja se obavlja pomoću objekata i tačka-operatora. Pri tome je bitno, da C# razlikuje mala i velika slova, i da se koristi isto ime metode u definiciji metode i kod pozivanja metode.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

FUNKCIJA MAIN()

Ovde objašnjavamo strukturu složenog programa kod proceduralnog programiranja. C# je o.o. Jezik, ali može da se koristi i za razvoj ne-o.o. programa tj. za proceduralno programiranje.

Veliki programi se razbijaju na skup manjih programa koji se lakše prave, nego da se pravi jedan veliki program. Složeni programi se razbijaju na niz manjih pomoću pravljenja funkcija. Tako se dramatično pojednostavljuje posao pravljenja složenih programa, pa je pravljenje funkcija vrlo važan deo programiranja. Postoji glavna funkcija *main()*, odakle počinje program, a iz te glavne funkcije mogu se pozivati ostale funkcije. Takodje, iz neke funkcije koja nije glavna *main*, može se pozivati neka funkcija. Na taj način mogu se napraviti vrlo složeni programi. U nekom programu tj. aplikaciji, može postojati čitav niz funkcija tj. metoda, ali postoji jedna specijalna funkcija, tzv. „glavna“ funkcija, označena sa rečju *Main()*, koju kompajler prepoznaje kao mesto gde počinje da se izvršava program, U jeziku C#, kod se može organizovati koristeći „funkcije“. Ako se funkcije dizajniraju i implementiraju kako treba, ovo radikalno olakšava posao pisanja složenih i velikih programa. Imena javnih funkcija započinjemo velikim slovom, npr. Kvadrat(), MemberFunction(), ClassFunction(), itd. Evo strukture složenog programa sa više funkcija:

```
public static void Main(.....)
{………..//user to enter input data
……………………………
//call functions
// output the result
………}
//other functions definitions
………………………
………………..
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRIMER FUNKCIJE

Evo primera proceduralnog programa sa funkcijom main().

```
using System;
namespace Example
{public class Program
{public static void Main(string[] args)
{
Average(“value1”, 3.5, “value2”, 4.0);
Console.WriteLine(“Press Enter ”);
Console.Read();}
// Average
public static void Average(string s1, double d1,string s2, double d2)
{double dAverage = (d1 + d2) / 2;
Console.WriteLine(“average : “ + dAverage);}}}
```

Ovaj program se izvršava tako što počne od prve instrukcije u glavnoj funkciji Main(), tj. to je prva instrukcija posle Main(). A funkcija Average() izračunava prosek od dve *double* vrednosti.

Egzekucija programa ima za izlaz: average : 3.75

Press Enter

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STATIČKE FUNKCIJE

*Ako je neka metoda deklarisana tj proglašena kao statička, **static**, onda ta metoda tj. funkcija se ponaša kao neka funkcija u ne-objektno-orijentisanom jeziku.*

C# je objektno-orijentisani jezik, tako da sve funkcije u stvari pripadaju „klasama”. Ovo znači da se često funkcije zovu „metodama“ umesto funkcijama. I ako sve funkcije (metode) pripadaju klasama, ako se metoda deklariše „statičkom“, *static method*, ona se onda ponaša kao ne-objektno-orijentisana funkcija. Programski jezik C# je objektno-orijentisan, i u okviru objektne orijentacije se definišu klase, a funkcije se definišu tako da pripadaju klasama tj. svaka funkcija pripada nekoj klasi, i funkcije se zovu klasnim metodama ili metodama objekata. Medjutim, ako je neka metoda deklarisana tj proglašena kao statička, ***static***, onda ta metoda tj. funkcija se ponaša kao neka funkcija u ne-objektno-orijentisanom jeziku, i u ovom slučaju nije potrebno deklarisati objekte.

Npr.

```
class Example
{Example e1 = new Example(); // kreiranje objekta
e1.MemberFunction(); // poziva clansku funkciju pomocu objekta
Example.ClassFunction(); // poziva klasnu funkciju pomocu klase
public void MemberFunction() // non-static
{Console.WriteLine(“clanska funkcija”);}
public static void ClassFunction() // static
{Console.WriteLine(“klasna funkcija”);}
}
```

Pored statičkih funkcija postoje i statičke varijable, *static variable*.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

“PREOPTEREĆIVANJE“ METODE

Metoda WriteLine u klasi Console, koja se koristi za ispisivanje string-a na ekranu, postoji u 19 verzija.

Ponekad je korisno da neku metodu napravimo u nekoliko verzija, a sa istim imenom. Npr. metoda WriteLine u klasi Console, koja se koristi za ispisivanje *string*-a na ekranu, postoji u 19 verzija. Svaka od ovih verzija koristi različit skup ulaznih parametara (argumenata). A jedna verzija ne koristi parametre uopšte već ispisuje prazan red na ekranu. Jedna druga verzija koristi bulovu varijablu kao parametar, itd. u vreme kompilacije, kompajler analizira skup argumenata funkcije, i poziva odgovarajuću verziju funkcije. Npr.

```
static void Main() {            Console.WriteLine (“ hello“);
        Console.WriteLine(42);}
```

“Preopterećivanje“ (*overloading*) tj. „preklapanje“ metode je korisno kada treba da uradite istu operaciju nad različitim tipovima podataka, tj. možete preopteretiti metodu kada različite verzije metode imaju različite skupove parametara a isti naziv metode. Npr. broj parametara različit, ili tip parametara različit. Mogu se koristiti dve funkcije sa istim imenom u okviru iste klase, pod uslovom da im se lista argumenata razlikuje. Ovo je tzv. preopterećenje imena funkcije, *function name overloading*.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

AUTOMATSKO KREIRANJE FUNKCIJA

Visual Studio je automatski kreirao ostatak programa, a to je glavna funkcija koja mora da postoji kod svake aplikacije.

Složeni programi se sastoje od jedne glavne funkcije i niza „sporednih“ funkcija (sporedna funkcija je funkcija koja nije glavna funkcija, a postoji samo jedna glavna funkcija). Sporedne funkcije se pišu ispod glavne funkcije jedna za drugom koristeći sintaksu,

...Main(....){...}

...Funkcija1(....){…..}

....Funkcija2(….){….}

Itd.

Ovo pisanje složenog programa koji se sastoji od niza funkcija, se može raditi manuelno, dakle jednostavno ukucavanjem instrukcija koristeći gore opisanu sintaksu, ili poluautomatski, gde VisualStudio pomaže u tom procesu. Inače Visual Studio pomaže kod kreiranja funkcija. Da napomenemo, mi smo već pravili nekoliko grafičkih aplikacija, gde smo pisali *event handler* funkcije, i Visual Studio je pomagao kod kreiranja programa, naime, Visual Studio je nudio kostur *event handler* funkcije, koju je onda lakše napisati nego da nema tog kostura, i osim toga, Visual Studio je automatski kreirao ostatak programa, a to je glavna funkcija koja mora da postoji kod svake aplikacije. Mi nismo ni videli glavnu funkciju,ali se ona može pronaći u odgovarajučem fajlu u okviru projekta sa ekstenzijom **.cs**,

npr. Form1.cs ili AddingMachineForm.cs, ili Progrm .cs,

gde je smešten sam program tj. programski kod, kom se može pristupiti pomoću *Code editor*-a, i onda vršiti inspekciju i modifikacije.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MODIFIKATORI „*REF*“ I „*OUT*“

*Ako se vrši predavanje parametra preko reference parametra, onda se ispred parametra stavi modifikator **ref** a može se koristiti i modifikator **out,** i onda nije potrebno inicijalizovati varijablu.*

U kompjuterskoj memoriji svaka varijabla ili objekt imaju svoju adresu koja definiše njenu lokaciju. Često se varijable koriste kao parametri (argumenti), tj. kao ulazni podaci za neku funkciju, npr.

FunkcijaHelloworld(parameter1, parameter2, parameter3)

Znači, predaje se tj unosi vrednost neke varijable dotičnoj funkciji na procesiranje. Pri tome, kod predavanja tj unosa vrednosti neke varijable kao parametra nekoj funkciji, postoje dve mogućnosti,

-da se funkciji preda samo kopija te varijable,

-ili da se funkciji preda adresa te varijable

U prvom slučaju, pomoću funkcije se ne može menjati izvorna varijabla tj. ulazni parametar, već samo kopija tog parametra se može menjati, a u drugom slučaju se može trajno promeniti izvorna varijabla, i ovo će biti kasnije ilustrovano primerom. Prvi slučaj predavanja tj unosa parametra se zove predavanje tj. slanje preko vrednosti parametra, ***passed by value***, a drugi slučaj predavanja tj. slanja parametra se zove predavanje tj. unošenje preko reference parametra, ***passed by reference***.

Ako se vrši predavanje parametra preko reference parametra, onda se ispred parametra stavi ključna reč tj tzv. modifikator **ref** a može se alternativno koristiti i modifikator **out,** pri čemu onda nije potrebno inicijalizovati varijablu. A ako se ispred parametra ne stavi nikakav **modifikator**, onda se automatski vrši predavanje tj. unošenje parametra ne preko reference već preko vrednosti parametra. Dakle,ako se želi promeniti vrednost unesenog parametra, vrlo je važno znati razliku izmedju unosa parametra preko reference parametra i preko vrednosti parametra. Vrlo je važno ovde uočiti da pomoću modifikatora „ref“ i „out“ se argumenti funkcije tj. parametri funkcije onda mogu koristiti kao IZLAZNI podaci tj. kao IZLAZ funkcije, dok bez ovih modifikatora argumenti funkcije mogu biti samo ulazni parametri tj. ULAZ funkcije.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

REFERENCIRANJE PARAMETARA

Objekti su takodje varijable koje imaju adresu, dakle može se primenjivati referenciranje.

Objekti su takodje varijable koje imaju adresu, dakle može se primenjivati referenciranje. Znači, umesto samog objekta može se nekoj funkciji tj. metodi predati adresa objekta. Adresa objekta po pravilu zahteva manje memorije od samog objekta. Ako se objekti unesu u funkciju preko vrednosti objekta a ne preko reference objekta, i onda promeni vrednost objekta, onda se izvan te funkcije ne „vidi“ ta promena vrednosti objekta. Dakle u pozivajućoj funkciji, iz koje je pozvana dotična funkcija, neće se videti promena vrednosti objekta, već samo u funkciji koja se poziva dakle funkcija gde se menja vrednost objekta.

U C# postoje *value types* (vrednosni tipovi) i *reference types* (*referentni tipovi*). U prvu grupu spadaju jednostavni tipovi (int, char, float, double, decimal, bool, itd.) kao i *structs*, a u drugu grupu svi ostali tipovi (array, string, object, itd.). Ako se *reference types* šalju funkciji kao parametri, šalje se samo njena adresa. Ako se šalje *value type* , normalno se šalje kopija, ali može se dati posebna instrukcija da se vrednosni tip pošalje kao adresa.. Vrednosni tipovi se memorišu u „*stack* “ memoriji (brza, lokalna memorija), a referentni tipovi se memorišu u „*heap*“ memoriji (velikoj, generalnoj memoriji).

Znači postoje u suštini dva načina unosa parametara u neku funkciju, ili unos u funkciju preko vrednosti parametra, i tada se ne koristi modifikator, i unos preko adrese varijable, i tada se koristi modifilator **ref** ili **out**.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PRIMER REFERENCIRANJA

Evo primera referenciranja parametara.

1.Započeti novu Windows-aplikaciju, i dodati jedno „dugme“, *button*, na „formu“.

2.Duplo kliknuti „dugme“, da bi se otvorio *Click event handler*. Pre nego se unese programski kod za *button click*, ukucati definiciju funkcije FunkcijaHello, i onda ukucati programski kod u *Click event handler.*

3.Izvršiti aplikaciju u cilju njenog testiranja. Primetimo da se vrednost prvog parametra unosi preko vrednosti parametra, pa se njegova vrednost ne menja i ako se u funkciji FunkcijaHello pokušava zadavanje druge vrednosti. Ostali parametri menjaju svoje vrednosti, jer se koristi referenciranje.

Evo dole je dat *Click event handler, i takodje ispod je funkcija FunkcijaHello:*

```
private void button1_Click(object sender, System.EventArgs e)
{           string parameter1 = ”oooooooo” ;string parameter2 = ”oooooooo” ;
string parameter3; //bez inicijalizacije
FunkcijaHello(parameter1, ref parameter2, out parameter3) ;
            MessageBox.Show(parameter1); MessageBox.Show(parameter2); MessageBox.Show(parameter3);}

private void FunkcijaHello(string parameter1, ref string parameter2, out string parameter3) ;
{           parameter1 = ”helloworld”;parameter2 = ”helloworld”; parameter3 = ”helloworld”;}
```

Rezultat rada ovog programa bi bio, štampanje u MessageBox-u, sledećih vrednosti: 00000000,

Helloworld, helloworld.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Strukture, *Structs*

10

„STRUKTURE“

Strukture imaju specifične osobine, npr. mogu se koristiti bez upotrebe ključne reči „new“.

„Struktura“, „*struct*“, tj. „strukturne varijable“, je složeni tip varijable (*multi-field variable*) koja se sastoji od nekoliko običnih varijabli tj. od nekoliko polja, i to različitih tipova. Naime. jednom složenom varijablom tipa „struktura“, *struct* , se mogu obuhvatiti nekoliko varijable različitog tipa, npr. ime, adresa i telefon neke osobe, tako da umesto da koristimo tri varijable, IMEKLIJENTA, ADRESAKLIJENTA, TELEFONKLIJENTA, možemo samo jednu *multi-field* varijablu koja se zove „struktura“, a koja ima tri polja. I dole je dat primer definicija strukturne varijable:

```
struct Klijent
{public string ime;
public string adresa;
public string telefon;}
```

Strukture imaju specifične osobine, npr. mogu se koristiti bez upotrebe ključne reči „new“. Strukture liče na klase, jer podržavaju konstruktore i seriju osobina „*properties*“. One su nekad bolje od klasa, npr. za neke manje važne objekte. I ako ne morate da upotrebite ključnu reč „new“, ipak dozvoljeno je koristiti „new“, što olakšava razumevanje programskog koda.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STRUCT-PRIMER

A zatim pogledajmo primer korišćenja te strukturne varijable u Visual C#, na formi sa samo jednim dugmetom:

1.Desnim klikom kliknuti „formu“, i izabrati *code editor*. Na dnu fajla, pre krajnje velike zagrade uneti definiciju strukture navedene gore:

```
...................
struct Klijent
{public string ime;
public string adresa;
public string telefon;
} }
```

2. Izabrati onda form-dizajner, i dodati „dugme“ na „formu“, i onda duplo kliknuti da se otvori *click event hendler,* i tada dodati programski kod gde se koristi prethodno definisana struktura Klijent:

```
Klijent klijent1;
klijent1.ime = ”petar petrovic”;
klijent1.adresa = ”mutapova 1”;
klijent1.telefon = ”065123456”;
MessageBox.Show(klijent1.ime + ”,” + klijent1.telefon);
```

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Enum-varijable

11

ENUM-VARIJABLE

Često se dešava da neka varijabla može imati samo nekoliko mogućih vrednosti. Zato se koriste enum-varijable, koje imaju prednost da štede memoriju i povećavaju brzinu aplikacije.

Često se dešava da neka varijabla može imati samo nekoliko mogućih vrednosti. Npr. boja novog Opel-automobila može biti crvena, siva, crna, i teget, dok ostale boje ne postoje. Zato se koriste *enum*-varijable, koje imaju prednost da štede memoriju i povećavaju brzinu aplikacije, i isto tako pomažu da se izbegnu greške u vrednostima tih varijabli sa odredjenim opsegom vrednosti. Takodje, ove varijable omogućuju da se koristi Visual C# *auto-complete* osobina.

Evo jednog primera *enum* varijabli („nabrojivih“ promenjivih):

enum Meseci

{Januar, Februar, Mart, April, ..., Novembar, Decembar};

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ENUM-PRIMER

Dole je dat primer u Visual C#, na formi sa samo jednim dugmetom, gde se koristi tip podataka enum, „enum-varijabla“, gde neka varijabla ima nekoliko mogučih vrednosti.

1.Otvoriti novu Windows-aplikaciju. Dodati „dugme“ na formu, i duplo kliknuti „dugme“ na formi da bi se otvorio code editor. Na dnu fajla , pre poslednje velike zagrade, uneti definiciju za strukturu „auto“ i za enum-varijablu „autoBoja“.

2. Zatim, dodati programski kod za *click event*.

3. Na kraju, izvršiti aplikaciju da se testira programski kod.

A programski kod izgleda ovako:

```
private void button1_Click(.....)
{Auto auto1 = new auto () ;
auto1.boja = autoBoja.Crna;
auto1.cena = 15000.00M;
MessageBox.Show(auto1.boja.ToString());
}}
enum autoBoja
{           Crvena, Crna, Teget }
struct Auto
{           public autoBoja boja;
            public decimal cena;
} }
```

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Konvezija varijabli

12

KONVEZIJA VARIJABLI

Ponekad je potrebno tretirati neku varijablu kao da je drugog tipa, npr. neku celobrojnu varijablu int ponekad je korisno tretirati kao da je decimalna varijabla decimal, itd.

Svaka varijabla mora da ima definisan svoj tip, Npr. int, float, decimal, bool, itd.,medjutim, ponekad je potrebno tretirati neku varijablu kao da je drugog tipa, npr. neku celobrojnu varijablu **int** ponekad je korisno tretirati kao da je decimalna varijabla ***decimal,*** itd. Ovo se zove „***casting***“**.** To je dakle neka vrsta konvertovanja tj pretvaranja neke varijable u drugi tip, ali pri tome se originalna varijabla ne menja već se ona sačuva. Ova operacija se obavlja tako da se željeni tip varijable stavi u male zagrade ispred varijable koju želimo da predstavimo kao varijablu željenog tipa, pri čemu je ta varijabla u stvari različitog tipa od ovog željenog tipa. Npr.

```
float float1;
int int1;
float1 = 5.1234;
int1 = (int)float1; //ovo je konvertovanje tj. casting
MessageBox.Show(int1.toString());
```

Kod ove tehnike pretvaranja jednog tipa varijable u drugi, može se desiti da

-ovo konvertovanje je uspešno, i to bez ikakve promene samog podatka, npr. kod pretvaranja iz integer-varijable u float-varijablu,

-ovo konvertovanje je uspešno, ali se podatak menja, npr. kod pretvaranja float-varijable u int-varijablu,

-neuspešno konvertovanje, npr. ne može se pomoću *casting*-a pretvoriti string-varijable u broj (“400“ u 400)

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

CASTING-PRIMERI

Casting tehnika tj. tehnika konverzije može se primeniti i na objekte.

Dakle, generalno, ne može se pomoću *casting*-a konvertovati string u neki broj, ali može se zato koristiti metoda toString() da se predje iz broja u string, a Parse metoda da bi se prešlo iz string-a u broj. Npr.,

```
string string1 = (int.Parse(“400“)).ToString(); //ok
```

Casting tehnika tj. tehnika konverzije može se primeniti i na objekte. Medjutim, da bi ovo konvertovanje uspelo, potrebno je voditi računa o nekim pravilima. Npr. konverzija objekata će uspeti, ako objekat koji se konvertuje je istog tipa kao i konvertovani objekt.

Koverzija iz „int“ u „float“ obavlja se automatski, ali obrnuto ne važi. Evo primera,

```
double dValue = 10;
long lValue = (long)dValue;
```

Sve koverzije iz i u “decimal”, zahtevaju primenu „casting” tehnike. U stvari, svi tipovi brojeva mogu se konvertovati u sve ostale tipove brojeva pomoću ”casting” tehnike. Medjutim, niti “bool” niti “string” se ne mogu konvertovati direktno u neki drugi tip.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Vežbe 4

13

VRSTE VARIJABLI

U programskom jeziku C# postoje dve vrste promenljivih.

U programskom jeziku C# postoje dve vrste promenljivih. Promenljive mogu biti

- vrednosnog tipa,
- referentnog tipa.

Promenljive vrednosnog tipa su specifične za svaki programski jezik i one se moraju usvojiti takve kakve jesu. Karakteristično je za vrednosni tip promenljivih da one čuvaju vrednost, a konkretnije infomracije o ovom tipu su u nastavku.

Promenljive referentnog tipa ne čuvaju vrednosti, nego čuvaju reference na memorijske adrese.

Vrednosni tip promenljivih

Vrednosni tip promenljivih sadrži podatak. S obziro da se na fakultetu mnogo proučavao programski jezik Java, najbolje je dati sledeće poređenje: vrednosni tip promenljivih je vrlo sličan primitivnim tipovima u Javi.

Karakteristično je da su svi tipovi izvedeni iz klase Object. Vrednosne promenljive ne mogu da sadrže null vrednost za razliku od referentnih tipova. Vrednosne promenljive se čuvaju na steku.

Slika 1:

C# Type	.NET Framework Type	Označen	Dužina [bajt]	Opseg vrednosti	Default
sbyte	System.Sbyte	Da	1	-128 do 127	0
short	System.Int16	Da	2	-32768 do 32767	0
int	System.Int32	Da	4	-2147483648 do 2147483647	0
long	System.Int64	Da	8	-9223372036854775808 do 9223372036854775807	0L
byte	System.Byte	Ne	1	0 do 255	0
ushort	System.UInt16	Ne	2	0 do 65535	0
uint	System.UInt32	Ne	4	0 do 4294967295	0
ulong	System.UInt64	Ne	8	0 do 18446744073709551615	0
float	System.Single	Da	4	7 cifara preciznost	0.0F
double	System.Double	Da	8	15-16 cifara	peciznost 0.0D
decimal	System.Decimal	Da	12	28-29 cifara peciznost	0.0M
char	System.Char	N/A	2	6-bitni Unicode karakter	'\0'
bool	System.Boolean	N/A	1	true ili false	false

Slika-1: Tipovi podataka

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

REFERENTNI TIP PROMENLJIVIH

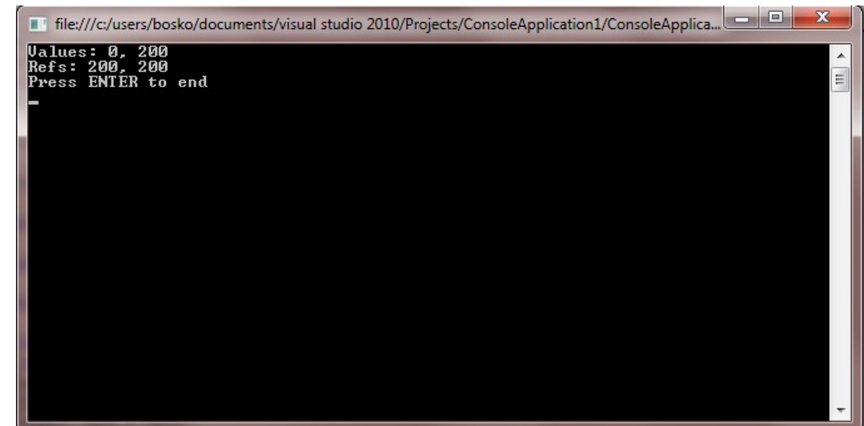
Ovaj tip promenljivih cuva reference na memorijske adrese sa vrednostima i one se alociraju na tzv. heap memorijskom delu.

Ovaj tip promenljivih cuva reference na memorijske adrese sa vrednostima i one se alociraju na tzv. heap memorijskom delu. Heap je deo memorije u kome se dinamicki alociraju promenljive. Ovaj tip promenljivih poznat je još i kao objekti. Ovo je razlika u odnosu na C++. Ove promenljive cuvaju reference na stvarne podatke tj. Na stvarni objekat u memoriji.

Primer: Kod koji prikazuje razlike vrednosnog i referencijalnog tipa promenljive.

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

namespace ConsoleApplication1
{
class Program
{
class Class1
{
public int Value = 0;
}
static void Main(string[] args)
{
int val1 = 0;
int val2 = val1;
```

Rezultat ovog programa je sledeći:



Slika-2: Konzolna aplikacija

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

REZIME VARIJABLI

Promenljiva referencnog tipa pravi se svaki put kada deklarišete promenljivu kao

Promenljiva referencnog tipa pravi se svaki put kada deklarišete promenljivu kao:

- klasu
- interfejs
- niz
- string
- objekat
- delegat

Promenljiva vrednosnog tipa se pravi kada deklarišete promenljivu tipa:

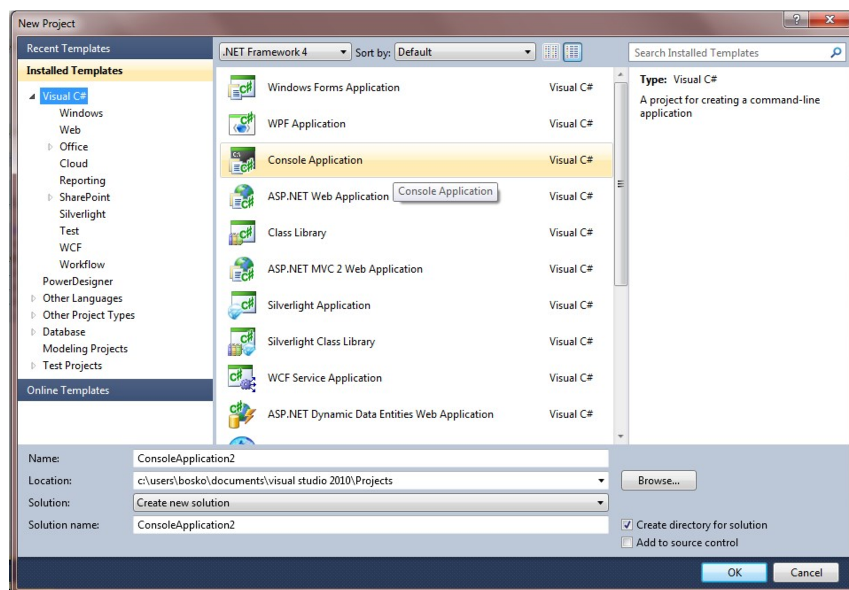
- Integer
- Floating
- Boolean
- Enumeration
- Structure

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONZOLNE APLIKACIJE - POSTUPAK KREIRANJA KONZOLNIH APLIKACIJA

Postupak kreiranja konzolnih aplikacija

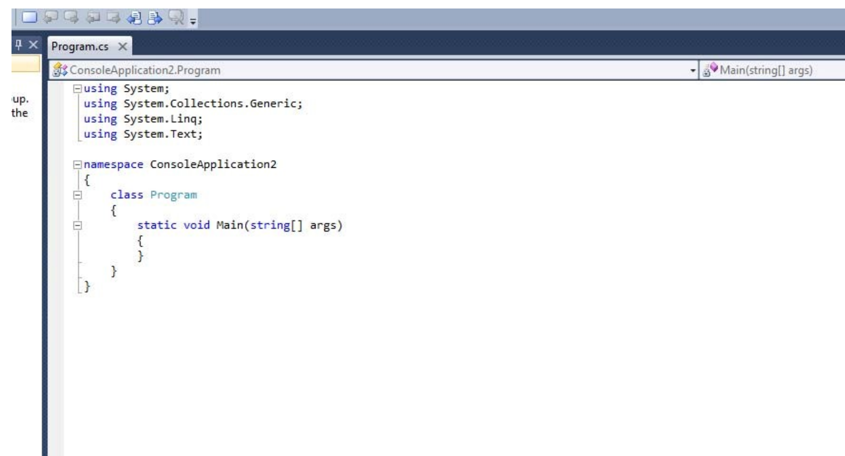
Da bismo kreirali novu konzolnu aplikaciju, potrebno je da nakon kreiranja novog projekta u opcijama odaberemo konzolnu aplikaciju kao na slici ispod.



Slika-3: Kreiranje konzolne aplikacije

Sva podešavanja oko imena i lokacije važe sa prethodnih vežbi.

Nakon potvrde na dugme ok, biće Vam kreiran osnovni kod konzolne aplikacije.



Slika-4: Kreiranje konzolne aplikacije

Ovaj kod je polazni osnov za kreiranje svih ostalih akcija u kodu.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONZOLNA APLIKACIJA – HELLO WORLD

Najjednostavnija konzolna aplikacija je HelloWorld I po nekom nepisanom pravilu, prilikom svakog početka rada sa konzolnim aplikacijama, upravo se ovaj primer obrađuje

Najjednostavnija konzolna aplikacija je HelloWorld I po nekom nepisanom pravilu, prilikom svakog početka rada sa konzolnim aplikacijama, upravo se ovaj primer obrađuje.

Konkretno, ovaj kod obezbeđuje ispis na konzoli rečenice Hello World! Kod je dat ispod.

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

namespace HelloWorld
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello World!"); Console.ReadLine();
}
}
}
```

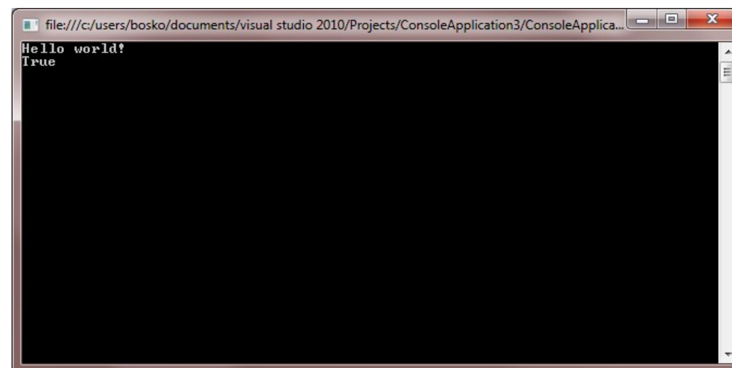
Konzolna aplikacija – sabiranje stringova

Malo uanpređenija verzija programa Hello World je rad sa sabiranjem promenljivih stringovnog tipa I ispis Boolean vrednosti za zbir tih promenljivih. U ovom slučaju izvršićemo deklaraciju tri stringovne promenljive I dodeliti im vrednosti, a zatim ih sabrati. Na kraju ćemo izvršiti ispis rezultata pitanja da li je zbir tih promenljivih isti kao I tekst kojim ga upoređujemo.

Kod:

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

namespace ConsoleApplication3
{
class Program
{
static void Main(string[] args)
{
string a = "Hello "; // Deklarisanje i iniciranje promenljive a
string b = "world"; // Deklarisanje i iniciranje promenljive b string c = "!"; // Deklarisanje i
iniciranje promenljive vrednosti
Console.WriteLine(a + b + c); // Konzolni ispis zbira stringova
Console.WriteLine(a + b + c == "Hello world!"); // Konzolni ispis boolean
Console.ReadLine();
}
}
}
```

Rezultat:



Slika-5: Rezultat

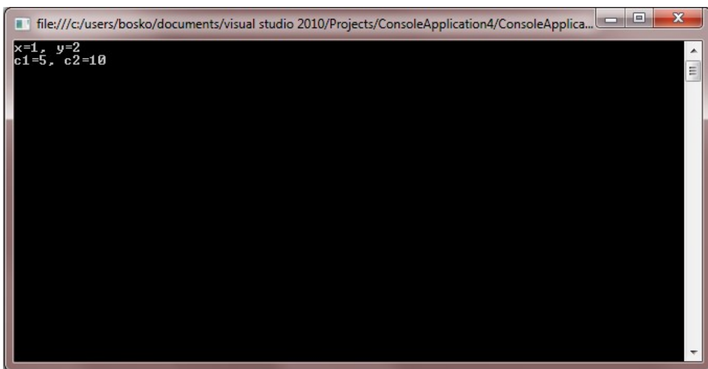
 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONZOLNA APLIKACIJA – OPERACIJE SA CELIM BROJEVIMA

Analogno radu sa stringovima, možemo se pozabaviti i drugim tipovima. Konkretno, u ovom primeru je operacija sa celim brojevima.

Analogno radu sa stringovima, možemo se pozabaviti i drugim tipovima. Konkretno, u ovom primeru je operacija sa celim brojevima.

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

namespace ConsoleApplication4
{
class Program
{
static void Main(string[] args)
{
int x = 1;
int y = 2;
const int c1 = 5;
const int c2 = c1 + 5;
Console.WriteLine("x={0}, y={1}", x, y); Console.WriteLine("c1={0}, c2={1}", c1, c2);
Console.ReadLine();
}
}
}
```



Slika-6: Rezultat

Konzolna aplikacija – unos celog broja

U slučaju da želimo da unesemo neki ceo broj sa tastature, to možemo da uradimo na sledeći način:

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

namespace ConsoleApplication5
{
class Program
{
static void Main(string[] args)
{

int a;
Console.Write("Unesite ceo broj ");
string str = Console.ReadLine();//Svaki unos iz konzole je u stringovnom
a = int.Parse(str); // Parsiranje - string u int

Console.WriteLine();
Console.WriteLine("Broj koji ste uneli je {0}", a);


Console.ReadLine();
}
}
```

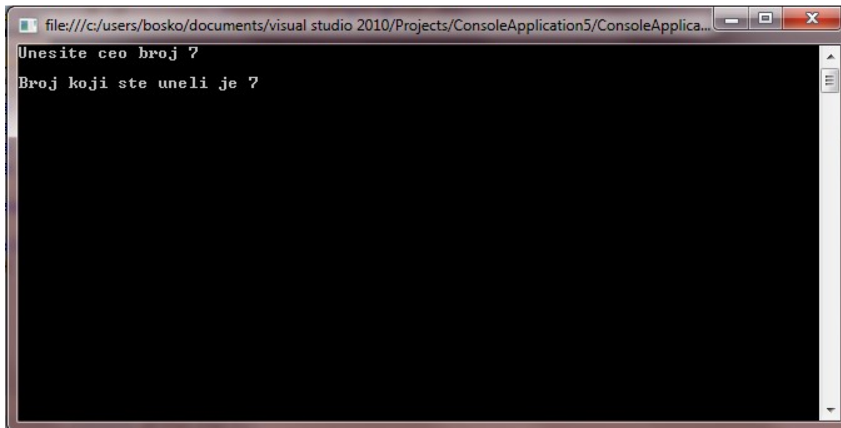
Napomena: Da bi se u potpunosti obezbedila funkcionalnost ovog programa, potrebno je obezbediti elemente koji će proveravati unos, odnosno neće dozvoliti nepravilne vrednosti. To se može uraditi dodatnim kodom koji će vršiti validaciju, ili try-catch mehanizmom. O svemu ovome će biti reči na narednim vežbama.

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONZOLNA APLIKACIJA – UNOS CELOG BROJA

U slučaju da želimo da unesemo neki ceo broj sa tastature, to možemo da uradimo na sledeći način:

U slučaju da želimo da unesemo neki ceo broj sa tastature, to možemo da uradimo na sledeći način. Slika rezultat:



Slika-7: Rezultat

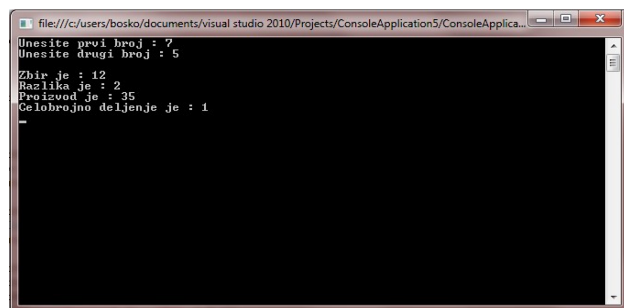
© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONZOLNA APLIKACIJA – PRORAČUN UNETIH BROJEVA

U slučaju da želimo da izvršimo nekakvu kalkulaciju od unetih brojeva sa konzolnog ulaza, to možemo uraditi na sledeći način:

U slučaju da želimo da izvršimo nekakvu kalkulaciju od unetih brojeva sa konzolnog ulaza, to možemo uraditi na sledeći način:

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

namespace ConsoleApplication5
{
class Program
{
static void Main(string[] args)
{
int x, y;

Console.Write("Unesite prvi broj : ");
string s1 = Console.ReadLine();
x = int.Parse(s1);

Console.Write("Unesite drugi broj : ");
s1 = Console.ReadLine();
y = int.Parse(s1);

Console.WriteLine(); Console.WriteLine("Zbir je : {0}", x + y);
Console.WriteLine("Razlika je : {0}", x - y);
Console.WriteLine("Proizvod je : {0}", x * y);
Console.WriteLine("Celobrojno deljenje je : {0}", x / y);

Console.ReadLine();
}
```



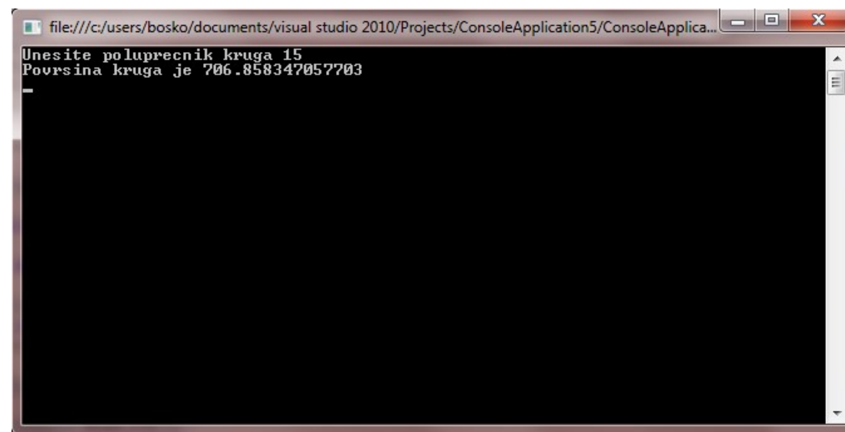
Slika-8: Rezultat

Zgodan primer za rad sa decimalnim brojevima je recimo proračun površine kruga. U tom cilju, možemo napraviti malu konzolnu aplikaciju koja će na osnovu unetog poluprečnika izračunati površinu kruga.

Konzolna aplikacija – proračun unetih brojeva

U ovom delu možemo iskoristiti i metode klase Math kako bismo prikazali i njene mogućnosti, a za Vaš dalji rad, otvorite klasu Math i proučite njene mogućnosti.

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

namespace ConsoleApplication5
{
class Program
{
static void Main(string[] args)
{
double r;
Console.Write("Unesite poluprecnik kruga ");
string s = Console.ReadLine();
r = double.Parse(s);
Console.WriteLine("Povrsina kruga je {0}", Math.Pow(r, 2) * Math.PI);
Console.ReadLine();
}
```



Slika-9: Rezultat

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MODIFIKATORI PRISTUPA

Drugo nastavno pitanje - Modifikatori pristupa

Vrlo je bitno da u ovoj vežbi razlučimo šta su modifikatorui pristupa, koja su njihova ograničenja i shvatimo njihovu važnost.

Modifikatori pristupa u jeziku C# služe da odrede koje metode i promenljive članice drugih klasa određena klasa može da vidi i da koristi.

U daljem objašnjenju će ova rečenica dobiti plastični prikaz.

Public

Modifikatorom public svi elementi klase koji su njim modifikovani su vidljivi svim metodama svih klasa.

```
public class ModifikatoriPristupa
{

public int celobrojnaVrednost = 1;
private void pristupiPoljuIzKlase()
{
celobrojnaVrednost++;
}
}
```

Private

Modifikatorom private je članovima klase moguće pristupiti samo metodama iz klase

Primer upotrebe je:

```
public class ModifikatoriPristupa
{

private int celobrojnaVrednost = 1;
private void pristupiPoljuIzKlase()
{
celobrojnaVrednost++;
}
}
```

Nemoguće je pristupiti polju izvan klase, odnosno nije moguće uraditi sledeće:

```
public class ModifikatoriPristupa
{

public int celobrojnaVrednost = 1;
private void pristupiPoljuIzKlase()
{
celobrojnaVrednost++;
}
}

public class TestKlasa
{
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

MODIFIKATORI PRISTUPA DRUGI DEO

Drugo nastavno pitanje - Modifikatori pristupa

Protected

Modifikatorom protected svi članovi klase dostupni su samo metodama klase i metodama klasa koje su izvedene iz klase.

Internal

Članovi klase su dostupni svim metodama svih klasa iz programskog sklopa u kome je klasa.

Protected Internal

Članovi klase su dostupni svim metodama klasa, metodama izvedenim iz klase I klasama programskog sklopa u kome je klasa.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

LOGIČKE STRUKTURE

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja.

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja. Takođe, postoje i različite petlje koje će omogućiti višestruko prolaženje kroz kod kada je to potrebno. U narednom odeljku dat je prikaz tih struktura.

U programskom jeziku C# postoji nekoliko naredbi koje pod određenim uslovima mogu da realizuju grananje i to:

If

Jednostavna if naradba vrši proveru nekog uslova i omogućava realizaciju bloka naredbi ako je uslov tačan (true).

Primer:

Ako je broj veći od 0 inkrementiraj brojač.

```
if(x > 0)
{
brojac++;
}
```

if – else

If – else naredba se koristi ako imamo slučaj u kome je potrebno uraditi jednu stvar za vrednost uslova, u suprotnom je potrebno uraditi drugu stvar.

Primer:

Ako je broj veći od 0 inkrementiraj brojač, u suprotnom dekrementiraj brojač.

```
if (x > 0)
brojac++;
else
brojac--;
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

LOGIČKE STRUKTURE DRUGI DEO

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja.

Kombinacijom višestrukih uslova može se dobiti nekoliko varijacija I to:

if – else – if

Primer:

Ispiši poruku da li je broj manji od 6, u opsegu od 6 do 13 ili je veći od 13.

```
if (x < 6)
{
System.Console.WriteLine("Manji  od sest.");
} else  if (x >= 6 && x <= 13) {
System.Console.WriteLine("U  opsegu  je od 6 do 13");
} else
{
System.Console.WriteLine("Veci   je od 13.");
}
```

ugnježdeni if – else

Primer:

Ako je broj veći od 0, a ako je brojač manji od 2, brojač postavi na 6, a ako je broj manji od 2, brojač postavi na nulu.

Kod:

```
if (x > 0)
{
if (brojac < 2)
{
brojac = 6;
}
else
{
brojac = 0;
}
}
```

switch

Ukoliko imamo jedan uslov po kome imamo samo jednu opciju koja je ispunjena, u tom slučaju je idealan izbor korišćenja naredbe switch.

Primer:

Na osnovu prosledjenog broja od 1 do sedam ispisati naziv dana u nedelji.

```
public void primerSwitch(int dan)
{

System.Console.WriteLine("Danas je " + dan + " dan u nedelji");

switch (dan)
{
case 1: Console.WriteLine("Ponedeljak"); break;
case 2: Console.WriteLine("Utorak"); break;
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJE

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja.

Petlje su kontrolne strukture pomoću kojih možemo da izvršimo višestruko ponavljanje jednog dela koda. Petlje se koriste u nekoliko karakterističnih situacija: prilikom kontrola toka, rada sa nizovima, rada sa kolekcijama, iteracijama, čekanje na ispunjenje uslova, itd.

U programskom jeziku C# postoji nekoliko petlji koje se mogu realizovati i to:

for

For petlja se sastoji od naredbe for, tri dela for naredbe u običnoj zagradi, a u vitičastoj zagradi se nalaze naredbe koje se vrte u okviru petlje. Tri dela koja for petlja ima su odvojena pomoću oznake ;. Prvi deo for petlje od otvorene obične zagrade do prve; služi za inicijalizaciju brojaèa. Drugi deo, koji se nalazi između dve oznake ; , je uslov. Treći deo, od ; do zatvorene obične zagrade, je inkrement odnosno decrement brojača. U prvom delu petlje, koji služi za inicijalizaciju brojača, se nalazi deklaracija brojača (u ovom slučaju int i) i njegova inicijalizacija na početnu vrednost (u ovom slučaju na 0). U drugom delu je uslov. Dokle god je uslov

ispunjen, odnosno dokle god je izraz između dve oznake ; true, petlja će da se izvršava (vrti).

Kod:

```
for (int i = 0; i < x; i++)
{
x = i * i;
```

Specijalni slučaj for petlje je kada se u bloku naredbi nalazi samo jedna naredba koja se ponavlja. U tom slučaju mogu da se izostave oznake za blok (otvorena i zatvorena vitičasta zagrada), ali to nije dobra programerska praksa jer otežava održavanje koda.

```
for (int i = 0; i < x; i++)
x = i * i;
```

U slučaju da je brojač neka promenljiva koja je prethodno definisana, moguće je da se u okviru for petlje, izostavi deo za definiciju brojača.

```
int i = 0
for (; i < niz.length; i++)
{
x = i * i;
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJE DRUGI DEO

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja.

Brojač može da se uvećava u delu bloka naredbi (ili da se umanjuje), i u toj situaciji nije neophodno imati inkrement (decrement) u delu for naredbe.

```
for (int i = 0; i < x; )
{
x = i * i++;
```

Specijalna verzija for petlje je izostavljen uslov. Uslov postoji, ali nije u delu koji je predviđen u for petlji, već se nalazi u samom telu petlje.

```
for (int i = 0; ; i++)
{
if (i < x)
break;
x = i * i;
}
```

Specijalni slučaj for petlje je i kada se petlja vrti u beskonačno. Moguće je kreirati beskonačnu for petlju tako što se izostave sva tri dela for petlje.

Kod:

```
for (; ; )
{
//beskonacna petlja
}
```

foreach

Naredba foreach je nova naredba u porodici c jezika I služi da obezbedi iterativni pristup svim elementima niza ili kolekcije.

Sintaksa je sledeća:

```
foreach (tip identifikator in izraz )
{
naredba
}
```

Primer:

Popunjavanje niza I štampanje elemenata niza.

```
int[] intniz = new int[5];
int[] intniz2 = new int[3];

for (int i=0; i<intniz2.Length; i++ )
{
intniz2[i] = i + 5;
```

 © UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PETLJE DRUGI DEO

U programskom jeziku postoji više naredbi koje mogu da uslove različite tokove i izvrše grananja.

while

While petlja je petlja koja je specijalno optimizovana za prolazak kroz petlju dokle god je neki uslov ispunjen.

Primer:

```
int i = 0;
while (i < niz.Length)
{
niz[i] = i * i;
i++;
```

do-while

Razlika između do – while petlje i svih ostalih petlji je u tome što se do – while petlja uvek izvršava bar jednom. Ispitivanje uslova ove petlje je na kraju.

Primer:

```
int i = 0;
int[] niz = new int[5];
do
{
niz[i] = i * i;
i++;
}
```

Pored naredbi grananja i mehanizama grananja, postoje i naredbe koje utiču na tok programa.

Naredba break spada u naredbe grananja zbog toga što prekida tok izvršavanja programa u određenom trenutku i prosleđuje na neko drugo mesto nastavak izvršavanja programa

Naredba continue prekida započeti tok i vraća petlju na početak, pre njenog logičkog kraja. Naredba return prekida tok neke metode i vraća rezultat pre njenog kraja.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

STRUKTURE PODATAKA

Struktura (engl. struct) je jednostavan korisnički definisan tip, lakša alternativa klasi.

Struktura (engl. struct) je jednostavan korisnički definisan tip, lakša alternativa klasi. Strukture su slične klasama po tome što mogu imati konstruktore, svojstva, metode, polja, operatore, ugnježdene tipove i indeksere.

Međutim, postoje i značajne razlike između klasa i struktura.

Na primjer, strukture ne podržavaju naasleđivanje, desturktore, a najvažnija razlika je u tome što su klase referentni tipovi, dok su strukture vrednosni tipovi.

Samim tim, primena struktura je odlična za predstavljanje objekata kojima nije potrebna semantika referenci.

Definisanje struktura

Sintaksa za deklarisanje structure vrlo je slična sintaksi za kreiranje klase:

[atributi] [modifikator pristupa] struct identifikator [:lista interfejsa]

{ članovi }

Kreiranje struktura

U primeru ispod kreirana je struktura koja ilustruje njenu upotrebu

```
using System;
using System.Collecti-ons.Generic;
using System.Text;


namespace CreatingAStruct
{
public struct Location
{
private int xVal;
private int yVal;

public Location(int xCoordinate, int yCoordinate)
{
xVal = xCoordinate;
yVal = yCoordinate;
}

public int x
{
get { return xVal; }
set { xVal = value; }
}

public int y
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ENUMERATORI

Nabrajanja (engl. enumerations) su moćna alternative konstantama. Predstavljaju vrednosni tip podataka koji predstavlja skup imenovanih konstanti

Nabrajanja (engl. enumerations) su moćna alternative konstantama. Predstavljaju vrednosni tip podataka koji predstavlja skup imenovanih konstanti.

Deklaracija i upotreba enum-a

```
enum Temperature
{
ApsolutnaNula = -273, TackaSmrzavanja = 0,
TackaKlucanja = 100,
};
```

Ovom enumeracijom postižemo bolju logičku vezu u kodu, nego da smo recimo kompletnu funkcionalnost uradili bez enumeratora. Ista stvar bi mogla da se realizuje na sledeći način:

```
const int ApsolutnaNula = -273;
const int tackaSmryavanja = 0;
const int tackaKljucanja = 100;
```

ali bismo ovakvom upotrebom u kasnijem delu povećali složenost održavanja, a smanjili logičku povezanost.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

REFERENCIRANJE I KONVERZIJA

Da bismo razumeli referenciranje, potrebno je prethodno da se podsetimo klasifikacije tipova i razlika među njima.

Da bismo razumeli referenciranje, potrebno je prethodno da se podsetimo klasifikacije tipova i razlika među njima.

Klasifikacija tipova

Osnovna podela tipova je na:

–vrednosne (value) tipove

–referentne (reference) tipove

Vrednosni tipovi su:

–jednostavni tipovi (kao što su npr. byte, int, long, float, double)

–nabrojivi tipovi

–strukture

Referentni tipovi su:

–klase

–interfejsi

–nizovi

–delegati

Svi tipovi (uključujući jednostavne kao što je **int**) su podtipovi **System.Object**

Razlike između vrednosnih i referentnih tipova:

Nekoliko bitnih razlika u osnovnim karakteristikama je između vrednosnih I referentnih tipova i to:

Razlike u alokaciji memorije:

- vrednosni tipovi se čuvaju na steku, odnosno u okviru objekta, a ako su članovi referentnih tipovana čuvaju se na hipu
- referentni tipovi se čuvaju na hipu

Razlike u sadržaju:

- vrednosni tipovi sadrže podatak
- referentni tipovi sadrže pokazivač (referencu) na podatak

Razlike u uništenju:

- vrednosni tipovi se uništavaju odmah pošto se napusti oblast definisanosti
- referentni tipovi se uništavaju pomoću sakupljača đubreta

Veza između referentnih I vrednosnih tipova je mehanizam pakovanja (boxing) I

raspakivanja (unboxing).

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

BOXING AND UNBOXING

Pakovanje je mehanizam koji od vrednosnog podatka pravi referentni objekat.

Pakovanje (boxing)

Pakovanje je mehanizam koji od vrednosnog podatka pravi referentni objekat, odnosno pravi se primerak objekta na hipu u koji se kopira vrednosni podatak.

Primer –pakovanje int promenljive:

int i=10; object o=i; System.Console.WriteLine("i="+i+", o="+o);

Primer –pakovanje long literala:

object longObj = 1000L;

Strukture se mogu konvertovati u tipove interfejsa koje implementiraju Primer –pakovanje struct podatka S koji implementira interfejs I:

S s=new S(); I i=s;

Implicitno pakovanje:

Ø prilikom dodele vrednosti (kao u gornjim primerima)

Ø prilikom pozivanja metoda strukture

Raspakivanje

Obrnut proces od pakovanja - od objekta koji sadrži prethodno spakovanu vrednost se pravi podatak vrednosnog tipa.

Nije moguce raspakivanje bilo kog objekta (onog koji ne sadrži spakovanu vrednost) Primer –pakovanje i raspakivanje int promenljive:

int i=10; object o=i; int ii=(int)o; System.Console.WriteLine("i="+i+", o="+o+", ii="+ii); Neophodna je eksplicitna konverzija (cast)

Izvršni sistem proverava tip konverzije koji mora da odgovara tipu spakovane

vrednosti, a ako se ne koristi odgovarajuća konverzija ispaljuje se izuzetak

System.InvalidCastException.

Napomena: Ako su performanse bitne treba izbegavati pakovanje/raspakivanje, jer troši vreme.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

PARAMETRI REF I OUT

Za prenos po referenci se koristi ključna reč ref, a za izlazni out.

Za prenos po referenci se koristi ključna reč ref, a za izlazni out. Ove ključne reči se koriste i u definiciji metoda i na mestu poziva

Prenos parametara po referenci omogućava bočne efekte metoda nad njima:

- ne pravi se kopija stvarnog argumenta već se izmene vrše nad njim
- Izlazni parametri ne moraju da budu inicijalizovani pre prosleđivanja metodu
- Ako se izlaznom parametru ne pridruži vrednost u metodu –greška

Primer:

```
public class C{
public static void M(out int x){x=5;}
public static void Main(){int x; C.M(out x);}
}
```


© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

KONVERZIJA

Da bismo realizovali konverziju bilo koje promenljive potrebno je da realizujemo proveru tipa. Ukoliko je tip kompatibilan, konverzija je moguća.

Da bismo realizovali konverziju bilo koje promenljive potrebno je da realizujemo proveru tipa. Ukoliko je tip kompatibilan, konverzija je moguća.

Šta je to provera tipa? Provera tipa je aktivnost koja obezbenuje primenu operatora na operande kompatibilnih tipova. (Operandi su promenljive, izrazi, parametri, a operatori: dodeljivanje, relacioni operatori, funkcije…).

Postoji dva tipa konverzije i to:

Eksplicitna konverzija

Eksplicitna konverzija iizmenu razllličitih tipova mora biti specificirana!

Primer: Eksplicitna konverzija long u int:

int intValue = (int) longValue;

Implicitna konverzija

Implicitna konverzija je izmenu različitih tipova odrenena je definicijom jezika.

Primer: Implicitna (automatska) konverzija int u long:

int intValue = 123;

long longValue = intValue;

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ZADACI ZA SAMOSTALNI RAD

Uradite sabiranje, oduzimanje, množenje, deljenje i ispis svih rezultata na ekranu.

Uradite sabiranje, oduzimanje, množenje, deljenje i ispis svih rezultata na ekranu. Pokušajte da upotrebom klase Math izvršite sve osnovne matematičke operacije sa dva broja.

Analogno primeru sa celobrojnim vrednostima, proradite varijantu sa vrednostima tipa float, double, decimal.

Proradite nekoliko zadataka iz petlji u cilju familijarizacije sa sintaksom.

Razmislite i provežbajte upotrebu struktura i utvrdite razlike u odnosu na klase. Takođe provežbajte upotrebu nabrajanja – enumeratora.

Proučite iz literature sa interneta statičke, stack-dinamičke i heap-dinamičke promenljive.

Izvršite sve tipove eksplicitne konverzije i utvrdite razloge kada eksplicitna konverzija ne može da se vrši.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

Zaključak

ZAKLJUČAK

Zaključak

- Neki C# fajl ima definisanu strukturu, tj sastoji se od nekoliko definisanih različitih delova. Dole su nabrojani i ratko opisani ovi pojedini delovi C# fajla. A kasnije će ovi elementi Cž fajlova biti detaljnije opisani. 1. ***Using statements*****:** Using-iskazi: na početku mnogih C# - fajlova se nalaze jedan ili nekoliko iskaza koji počinju sa rečju: **using,** i ovi iskazi daju instrukcije kompajleru gde da potraži klase koje se kasnije pominju u programu. Ovi using-iskazi se ne izvršavaju, i oni su neobavezni tj. oni su opcioni, dakle štede kucanje nekih instrukcija u programu. 2. ***Namespace declaration*****:** *Namespace*-deklaracija je prisutna u većini C# - fajlova, a iza toga je blok u velikim zagradama, i ove deklaracije predstavljaju jedan način organizacije programskog koda, i omogućuju kompajleru da pronadje klase koje su definisane u programu. 3. ***Comments:*** Komentari, *comments,* u programu se označavaju sa dve ili više kosih crta, naime bilo koja linija programskog koda koja počinje sa dve ili više kosih crta je samo komentar a ne i prava instrukcija. 4. ***Class statements:*** Neki C# fajl može da sadrži jednu ili nekoliko klasa, iako jedna klasa u fajlu je tipičan slučaj. Klasni iskazi, *class statements*, to su iskazi koji uvode nove klase u program, a markirani su velikim zagradama. I mada većina fajlova definišu klase, ima fajlova koji ne definišu klasu već npr interfejs ili *custom type*, tj. uslužni tip podataka. 5. ***Class-level variables*****:** Kao što je na prethodnoj strani ilustrovano, varijable klasnog nivoa, class-level variables , su varijable deklarisane unutar klase, koje su vidljive tj. pristupne u okviru klase, a opciono su pristupne i van klase. 6. ***Method declarations*****:** Deklaracije metoda, *method declarations*, to je dodatni programski kod za neki fajl tj. neku klasu, i on opisuje funkcionalnost klase, i tu u ovom dodatnom kodu, je obično data i lista karakteristika, *properties*. 7. ***# directives*****:** # - uputstva tj direktive**,** # *directives*, to su iskazi koji počinju sa #, i to su direktive kompajleru, tzv. preprocesorske direktive, tj. uputstva. 8. ***Generated code:*** Automatski generisan programski kod, to je kod koji se automatski pojavi u form-dizajneru, i on se po pravilu ne dira, ali to je isto programski kod kao i ostali programski kod koji se svojeručno piše. Ovaj programski kod je neophodan kod izvršavanja programa, i ako se on modifikuje može biti vrlo problematično da se ispravi takva greška, pa treba biti vrlo oprezan da se ne menja ili e briše nešto što vi niste svojeručno napisali u programu.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.

ZAKLJUČAK

Zaključak

- nije potrebno staviti tačka-zarez posle definicije funkcije, što znači, da ako se stavi tačka-zarez, doći će do greške u kompilaciji programskog koda. a velike zagrade označavaju početak ii kraj programskog koda funkcije, dok, male zagrade posle imena funkcije, obuhvataju listu ulaznih podataka za tu funkciju, tj. podataka koji se isporučuju tj. predaju funkciji, ***passed to the function***, i ovi ulazni podaci funkcije se zovu parametri funkcije, a u slučaju da nema ulaznih podataka tj. parametara funkcije, ipak se stavljaju male zagrade, ali izmedju tih zagrada nema onda parametara.

-Programski jezik C# je objektno-orijentisan, i u okviru objektne orijentacije se definišu klase a funkcije se definišu tako da pripadaju klasama tj. svaka funkcija pripada nekoj klasi, i funkcije se zovu klasnim metodama ili metodama objekata.

-Medjutim, ako je neka metoda deklarisana tj proglašena kao statička, ***static***, onda ta metoda tj. funkcija se ponaša kao neka funkcija u ne-objektno-orijentisanom jeziku, i u ovom slučaju nije potrebno deklarisati objekte.

© UNIVERZITET METROPOLITAN, Beograd / Kopiranje i umnožavanje nije dozvoljeno / Sva prava su zadržana.