Programiranje – III razred

Struktura programskog jezika C#

Elementi C# jezika

Sintaksa jezika predstavlja skup pravila koja definišu format i konstrukciju naredbi – komandi u programskom jeziku

Semantika predstavlja specifikaciju funkcije – onoga što naredba radi, izvršava

Statement je iskaz, naredba, komanda koja izvršava neku akciju

Više naredbi u određenom redosledu sačinjava **metod** – imenovani skup naredbi

Sintaksa jezika C# je slična C jeziku, tako da C# zajedno sa C++, JAVA-om, JavaScript-om, PHP-om i drugim jezicima spada u C-olike jezike

Svaka naredba se mora završiti sa ;

Razmak, tab-ovi, novi redovi – white space služe kao separatori

C# - identifikatori

Identifikatori predstavljaju imena imenskih prostora, klasa, metoda, promenljivih

Bilo koji identifikator se sastoji samo od slova (letters) – malih i velikih, cifara (digits) i znaka za podvučeno (underscore)

Identifikator mora početi sa slovom ili sa underscore-om koji se tretira kao slovo

C# je “case sensitive” jezik što znači da razlikuje mala i velika slova

Validni identifikatori:

result, _score, footballTeam, maj9

Nisu identifikatori:

footballTeam$, 9maj

C# - ključne reči

Ključne reči predstavljaju rezervisane identifikatore koji imaju odgovarajuće značenje i koje koristi C#

Ključne reči se ne mogu koristiti kao korisnički identifikatori

Osim ključnih reči postoje i identifikatori u C# - u koji se mogu koristiti kao korisnički identifikatori, mada se to ne preporučuje

from	join	select	yield
get	let	set	
group	orderby	value	
into	partial	where	

C# - ključne reči

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Variables - promenljive

Postoje dva osnovna pravila za imenovanje promenljivih:

- Mađarska notacija (Hungarian notation) i
- Camel notation

Camel notation imenuje promenljive malim slovima a velika se koriste kod imena od više reči da označe početak sledeće reči sa velikim slovom – kamilja notacija
Primeri za Camel notation: prviPodatak, novaGodina, prviMaj, noviSad, noviSadSpens, itd

Hungarian notation počinje sa oznakom tipa promenljive posle čega sledi naziv promenljive

Savetuje se upotreba Camel notation a ne i Hungarian notation

Imena identifikatora su takođe važna za saradnju sa drugim .NET jezicima koja postavlja veća ograničenja nego sam C#

Saradnja sa drugim jezicima

Da bi C# sarađivao – koristio assembly-je iz drugih **.NET** jezika – bilo kog, neophodno je da se:

- ne koristi underscore i
- da se ne koristi razlikovanje promenljivih samo prema veličini slova

Ispravna, preporučena imena promenljivih:

score, footballTeam

Ispravno, ali se ne preporučuje:

_score, FootballTeam

Prvi program u C#

Zadatak broj 1:

1. Otvoriti MS Visual C# 2010
2. Kreirati novu konzolnu aplikaciju pod nazivom **PrvaAplikacija**
3. Ukucati kod sa slajda:

```
static void Main(string[] args)
        {
            Console.WriteLine("Moja prva aplikacija u C#.");
            Console.ReadKey();
        }
```

4. Pokrenuti aplikaciju i pogledati rezultat.
5. Izbrisati: `Console.ReadKey();` i ponovo pokrenuti aplikaciju.

Prvi program u C#

Pojašnjenje koda:

```
static void Main(string[] args)
        {
            Console.WriteLine("Moja prva aplikacija u C#.");
            Console.ReadKey();
        }
```

Console – konzolni prikaz,

WriteLine – metoda konzolnog prikaza namenjena ispisu po konzoli

ReadKey – metoda konzolnog prikaza namenjena očitavanju pritisnutog tastera. Ovde ima za ulogu da ne dozvoli gašenje konzolnog prikaza dok se ne pritisne neki taster (kako bi se mogao videti rezultat rada koda).

Deklaracija promenljivih

C# je strogo tipiziran jezik

Svaki identifikator je neophodno deklarisati pre upotrebe

Deklaracijom se navodi tip i ime promenljive

Primer deklaracije:

```
int godine;
```

Dodeljivanje vrednosti deklarisanoj promenljivoj:

```
age = 42;
```

Korišćenje promenljive:

```
Console.WriteLine(age);
```

Deklaracija promenljivih

Zadatak broj 2: Realizovati kod sa slajda. Pokrenuti aplikaciju. Koja je uloga zelenog dela koda?

```
namespace Promenljive
{
    class Program
    {
        static void Main(string[] args)
        {
            //deklaracija promenljive tipa int koja se zove broj
            int broj;
            //deklaracija promenljive tipa int koja se zove broj2
            //pri deklaraciji promenljive je uradjena i inicijalizacija - dodeljena
inicijalna vrednost promenljivoj
            int broj2 = 7;
            //Console.WriteLine("Vrednost promenljive broj je:" + broj); greska-
kompajler ce javiti gresku
            //u c#-u promenljivoj morate da dodate vrednost pre nego sto pokusate da
se referencirate na vrednost te promenljive
            //ovo je poznato kao izricito dodeljivanje
            //promenljivoj broj je dodeljena vrednost 10 tj inicijalizovana je na
vrednost 10
            broj = 10;
            Console.WriteLine("Vrednost promenljive broj je:" + broj);
            Console.ReadLine();
        }
    }
}
```

Osnovi – primitivni tipovi podataka

Data type	Description	Size (bits)	Range [1]	Sample usage
int	Whole numbers	32	-2^{31} through $2^{31}-1$	`int count;` `count = 42;`
long	Whole numbers (bigger range)	64	-2^{63} through $2^{63}-1$	`long wait;` `wait = 42L;`
float	Floating-point numbers	32	$\pm 1.5 \times 10^{45}$ through $\pm 3.4 \times 10^{38}$	`float away;` `away = 0.42F;`
double	Double-precision (more accurate) floating-point numbers	64	$\pm 5.0 \times 10^{-324}$ through $\pm 1.7 \times 10^{308}$	`double trouble;` `trouble = 0.42;`
decimal	Monetary values	128	28 significant figures	`decimal coin;` `coin = 0.42M;`

string	Sequence of characters	16 bits per character	Not applicable	`string vest;`
				`vest =`
				`"fortytwo";`

char	Single character	16	0 through $2^{16}-1$	`char grill;` `grill = 'x';`
bool	Boolean	8	True or false	`bool teeth;` `teeth = false;`

Dodela vrednosti promenljivoj

Pre korišćenja, promenljivoj se mora dodeliti vrednost

Taj zahtev se naziva **Definite Assignment Rule**

Pokušaj korišćenja nedodeljene promenljive izaziva grešku compiler-a pri prevođenju

```
int age;
Console.WriteLine(age); // compile-time error

float variable;
variable=0.42F;
```

Sa F se obeležava tip float, po default-u je double

Aritmetički operatori

Uobičajeni aritmetički operatori

- **+**,
- **–**,
- *****,
- **/**

- **%** - ostatak deljenja, bilo kog a ne samo celobrojnog (mod)

```
long novac; //deklaracija
novac = 750 * 20;//operacija i dodela
```

Aritmetički operatori se ne mogu primeniti na sve tipove podataka

Svi se mogu primeniti na numeričke tipove

Samo operator + se primenjuje na string tip podataka

Operatori i tipovi podataka

Console.WriteLine(“43” + “1”);

Rezultat je “431”

// compile-time error:

Console.WriteLine(“Tehnička škola” – “škola”);

Int32.Parse – metod za prevođenje string-a u int

Tip podataka rezultata operacije zavisi od tipa podataka operanada

5.0/2.0 = 2.5	double / double = double
5/2 = 2	int / int = int
5/2.0 = 2.5	int / double = double

Operatori i tipovi podataka

Iako compiler rešava slučaj operacija sa mešovitim tipovima podataka, to se ne preporučuje.

Double i float imaju specijalne vrednosti:

- **infinity** (beskonačnost) i
- **NaN** (Not a Number – nije broj)

Infinity se javlja kod deljenja sa nulom, dok se NaN javlja kod izraza tipa 0 / 0

Ove vrednosti dalje učestvuju - propagiraju u izrazima na sledeći način:

10 + NaN=NaN, 10 + Infinity = Infinity
Infinity * 0 = 0, NaN * 0 = NaN.

Svaka klasa u .NET Framework-u ima ToString metod koji prevodi objekat date klase u string oblik prikaza

Prioritet i asocijativnost operatora

Multiplikativni operatori

- *****,
- **/** i
- **%**

imaju viši prioritet (precedence) od aditivnih

- **+** i
- **–** operatora

Zato je 2 + 3 * 4 = 14 a ne 20

Ako priroitet operatora ne obezbeđuje željeni redosled operacija, koriste se zagrade
(2 + 3) * 4 = 20 – ipak

Asocijativnost dolazi do izražaja kod određivanja redosleda operacija istog prioriteta
Operatori * i / su **levo asocijativni**, što znači da se operacije vrše sa leva na desno, pa je
4 / 2 * 6 = 12 a ne 4 / 12

Operator dodele =

```
int myInt;

int myInt2;

myInt2 = myInt = 10;

myInt5 = myInt4 = myInt3 = myInt2 = myInt = 10;
```

Operator dodele je asocijativan sa desna u levo

Increment / decrement, prefix / postfix

```
count = count + 1;

count++;

count--;

count++; // postfix increment

++count; // prefix increment

count--; // postfix decrement

--count; // prefix decrement

int x;
x = 42;
Console.WriteLine(x++); // x is now 43, 42 written out
x = 42;
Console.WriteLine(++x); // x is now 43, 43 written out
```

Implicitna deklaracija promenljivih

```
int myInt;    //deklaracija

int myInt = 99;    //inicijalizacija

int myInt = myOtherInt * 99;    //inicijalizacija preko druge vrednosti

var myVariable = 99;    //implicitna deklaracija u int

var myOtherVariable = “Hello”;    //implicitna deklaracija u string

var yetAnotherVariable;    // Error - compiler cannot infer type
```