Univerzitet u Beogradu

Matematički fakultet

MASTER RAD

Elektronske lekcije o osnovnim pojmovima programskog jezika C# za učenike srednjih škola

Mentor:

Prof. dr Miroslav Marić

Kandidat:

Ana Milovanović

Beograd 2016.

Sadržaj

1. Uvod

Kroz ovaj master rad prikazane su elektronske lekcije o osnovnim pojmovima programskog jezika C#, namenjene učenicima srednjih škola i drugim početnicima u programiranju. Izbor najprihvatljivijeg programskog jezika za početnike nije nimalo jednostavan, a trebalo bi pronaći programski jezik kroz koji nije preterano teško usvojiti osnovne koncepte modernog programiranja. Programski jezik C# je odabran kao tema ovog master rada iz razloga što je to moderan, objektno orijentisan programski jezik, široko rasprostranjen i može biti pogodan za početnike [1]. Način učenja koji podrazumeva korišćenje edukativnih sadržaja na internetu je u sve većoj ekspanziji, pa je samim tim glavna ideja ovog rada da početnicima pruži materijale, u vidu elektronskih lekcija na internetu, koji mogu biti od koristi za savladavanje osnovnih pojmova programiranja u programskom jeziku C#.

Uvodni deo ovog rada odnosi se na opis samih elektronskih lekcija. Elektronske lekcije prikazane u ovom radu su vrsta interaktivnog udžbenika i sastoje se od neophodnih tekstova, slika, urađenih zadataka, programskih kodova, korisnih linkova i video materijala postavljenih na internet servis YouTube. Pored opisa lekcija, u uvodnom delu opisan je i koncept objektno orijentisanog programiranja i dat je uvod u razvojno okruženje. U glavnom delu rada opisane su elektronske lekcije koje se odnose na programiranje u programskom jeziku C#, gde je pre svega opisan način na koji su one prezentovane učenicima. Teme koje su obrađene u okviru elektronskih lekcija vezane su za osnovne pojmove kao što su promenljive, tipovi podataka, operatori, uslovne naredbe, petlje, nizovi, metode i niske. Završni deo predstavlja zaključak u kome se navode razlozi zašto bi programski jezik C# mogao biti dobar za početnike, razmatraju se prednosti i mane elektronskog načina prezentovanja sadržaja i daju se predlozi za poboljšanje istog.

1.1. Istorijat i verzije programskog jezika C#

Programski jezik C# je razvila američka kompanija Microsoft u okviru .NET platforme. Januara 1999. godine oformljen je tim za kreiranje novog programskog jezika koji bi pomogao pri pisanju biblioteka klasa za .NET. Novi programski jezik je dobio naziv Cool (C-Like Object Oriented Language). Projekat .NET je javno objavljen jula 2000. godine, a programski jezik Cool je preimenovan u C#. Programski jezik C# je kasnije odobren po standardu ISO/IEC 23270:2006. Ime C# je inspirisano muzičkom notacijom – C# je oznaka za notu cis koja predstavlja povisilicu note C. U tabeli 1., preuzete iz literature [2], prikazane su verzije programskog jezika C#.

Tabela 1: Verzije programskog jezika C#

Verzija	Datum	.NET Framework	Visual Studio
C# 1.0	Januar 2002.	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.2	April 2003.	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	Novembar 2005.	.NET Framework 2.0	Visual Studio 2005
C# 3.0	Novembar 2007.	.NET Framework 2.0 .NET Framework 3.0 .NET Framework 3.5	 Visual Studio 2008 Visual Studio 2010
C# 4.0	April 2010.	.NET Framework 4	Visual Studio 2010
C# 5.0	Avgust 2012.	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0	Jul 2015.	.NET Framework 4.6	Visual Studio 2015

2. Elektronske lekcije o osnovnim pojmovima programskog jezika C#

Elektronske lekcije o osnovnim pojmovima programskog jezika C# za učenike srednjih kreirane za potrebe ovog master rada se nalaze na sledećoj adresi:

http://alas.matf.bg.ac.rs/~ml09119/Master/index.php

Nakon pokretanja ove adrese prikazuje se početna strana elektronskih lekcija na kojoj se nalazi naslov i dugme sa natpisom „Pokreni“. Klikom na dugme „Pokreni“ prikazuje se sadržaj poglavlja „Uvod“, što se može videti na slici 1.

Slika 1: Elektronska lekcija - Uvod

Elektronske lekcije se nalaze na veb sajtu koji se sastoji iz horizontalne linije menija u kojoj se nalazi 8 glavnih menija i ikonica za povratak na početnu stranu. Kada se jedan od menija odabere otvara se pomoćni meni sa leve strane. Pomoćni meniji sa leve strane imaju po nekoliko karakterističnih stavki. Korisnik bira najpre glavni, pa pomoćni meni i na taj način se prikazuje odgovarajući sadržaj, odnosno elektronska lekcija.

2.1. Teme

Teme koje su obrađene u okviru elektronskih lekcija mogu se videti u glavnoj liniji menija, a to su sledeće teme:

1. ***Uvod***: Svrha teme je upoznavanje učenika sa programskim jezikom C#, karakteristikama objektno orijentisanog programiranja i upoznavanje sa razvojnim okruženjem.

2. ***Osnovni pojmovi***: Kroz ovu temu učenici se upoznaju sa pojmovima kao što su promenljive, tipovi podataka i operatori.

3. ***Zadaci / Programi sa linijskim tokom izvršavanja***: Tema ima za cilj da učenike kroz video materijale uvede u programiranje aplikacija koje imaju linijski tok (bez grananja i petlji).

4. ***Uslovne naredbe***: Cilj teme je upoznavanje učenika sa uslovnim naredbama `if`, `else` i `switch`.

5. ***Petlje***: Kroz ovu temu su prikazane petlje `for`, `while`, `do while` i `foreach`.

6. ***Metode***: Tema ima za cilj da upozna učenike sa definisanjem i pozivanjem sopstvenih metoda.

7. ***Nizovi***: Kroz ovu temu prikazani su jednodimenzioni nizovi i rad sa njima, kao i osnovni algoritmi za pretragu i sortiranje.

8. ***Niske***: Cilj teme je da se učenici upoznaju sa objektima tipa niske i metodama za rad sa niskama.

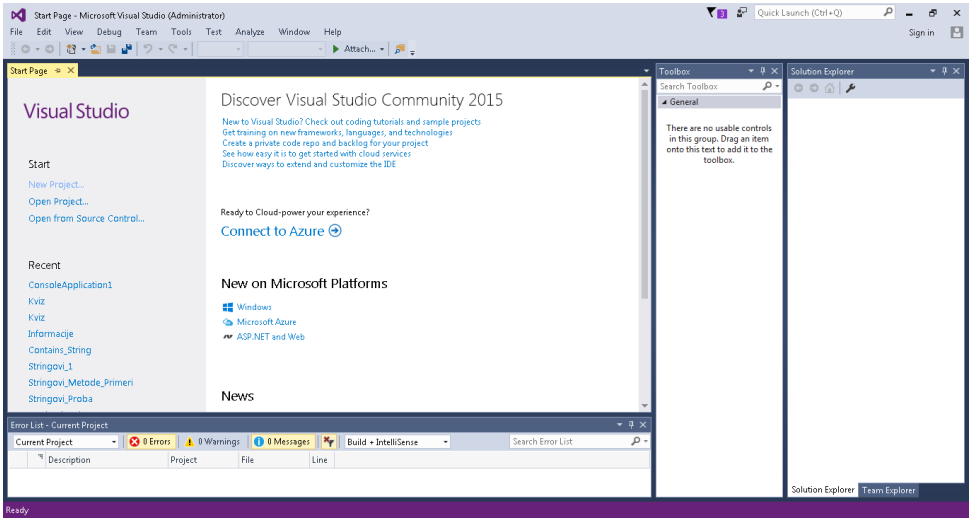
2.2. Objektno orijentisano programiranje

Učenicima je predstavljen uvodni deo koji govori o objektno orijentisanom programiranju kako bi se kasnije u okviru elektronskih lekcija mogli koristiti pojmovi kao što su objekti, klase i metode. Objektno orijentisana paradigma u programiranju je, pored proceduralne, jedna od najznačajnijih programskih paradigmi među jezicima višeg nivoa. Objektno orijentisano programiranje se može shvatiti kao poseban način razmišljanja u programiranju gde se sve bazira na objektima i komunikaciji objekata.

Objekti su uvek predstavljeni svojom klasom. Klasa predstavlja opis po kome će se objekat napraviti. Klase sadrže različite elemente, među kojima su najvažniji atributi i metode. Atributi predstavljaju svojstva koje će imati objekat, a metode su funkcije koje služe za rad sa objektima. Na osnovu opisa koji se nalazi u klasi, kreira se objekat koji predstavlja instancu klase. Može se napraviti neograničen broj objekata klase. Objekat se pravi preko konstruktora, koji je definisan u klasi. U okviru klase može biti i definisano i više konstruktora koji zavise od različitih parametara.

2.3. Razvojno okruženje i komponente

U okviru elektronskih lekcija obrađeno je razvojno okruženje Microsoft Visual Studio 2015. Na slici 2. može se videti izgled razvojnog okruženja nakon pokretanja.



Slika 2: Pokrenuto razvojno okruženje Microsoft Visual Studio 2015

U okviru elektronske lekcije „Uvod“ je preko video materijala prikazano kako se u razvojnom okruženju Microsoft Visual Studio 2015 kreira novi projekat, snima projekat, otvara postojeći projekat i kako se pokreće aplikacija.

Celina „Razvojno okruženje“ sadrži i listu komponenti koje se kasnije koriste u urađenim zadacima u okviru elektronskih lekcija. To su komponente:

1. ***Button*** – komponenta koja se koristi za pokretanje neke akcije na klik mišem.

2. ***Label*** – komponenta preko koje se prikazuje tekst.

3. ***TextBox*** – komponenta preko koje se tekst prikazuje ili unosi od strane korisnika.

4. ***RichTextBox*** – slična komponenta kao i *TextBox*, samo što se preko nje može prikazati tekst u više linija, za razliku od komponente *TextBox* kod koje je tekst u jednoj liniji.

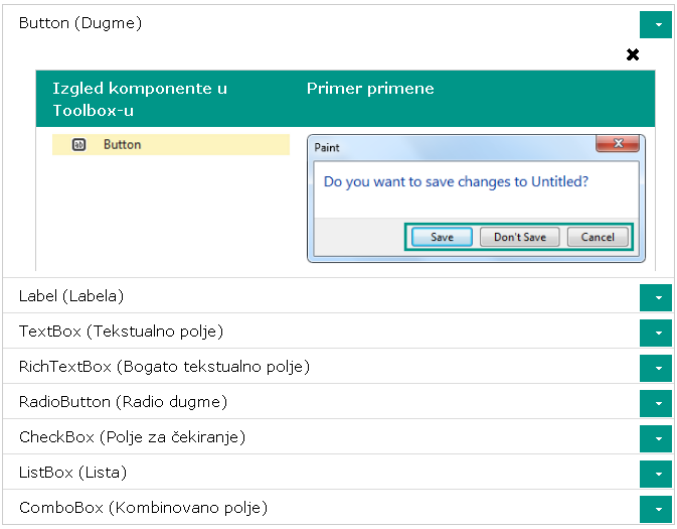
5. ***RadioButton*** – komponenta izbora koja se koristi kada je potrebno od više opcija odabrati samo jednu.

6. ***CheckBox*** – komponenta koja se koristi za potvrdu određene opcije (čekiranje).

7. ***ListBox*** – komponenta koja prikazuje listu određenih opcija.

8. ***ComboBox*** – komponenta koja predstavlja kombinaciju komponenti *ListBox* i *TextBox*.

Učenicima je za sve ove komponente prikazano kako one izgledaju u paleti sa alatima (toolbox) i dat je prikaz primene u konkretnim aplikacijama koje učenici najverovatnije već dobro poznaju. Na slici 3. može se videti prikaz komponente *Button* u okviru ove elektronske lekcije zajedno sa ostalim komponentama. Prikaz ostalih komponenti dobija se klikom na strelicu pored naziva komponente.



Slika 3: Komponenta Button – prikaz u okviru celine „Razvojno okruženje“

2.4. Prvi program napisan na programskom jeziku C#

U okviru elektronskih lekcija je učenicima izložena razlika između aplikacija zasnovanih na prozorima i konzolnih aplikacija. Kod aplikacija zasnovanih na prozorima jednostavna je komunikacija sa korisnikom preko grafičkog korisničkog interfejsa, dok kod konzolnih aplikacija to nije slučaj. Kod konzolnih aplikacija komunikacija se vrši preko konzole. Svodi se na unošenje podataka za obradu preko tastature. Učenicima je jasno izloženo da će se u elektronskim lekcijama obrađivati aplikacije koje su zasnovane na prozorima, dok je prikaz konzole uveden informativno.

Prvi program napisan na programskom jeziku C# jeste čuveno ispisivanje poruke „Zdravo svete!“. Ispisivanje ove poruke izvršeno je u konzoli kao ilustracija konzolnog programiranja. Programski kod aplikacije je objašnjen učenicima kroz komentare, a može se pregledati sa i bez komentara.

3. Osnovni pojmovi

U okviru poglavlja „Osnovni pojmovi“ prikazani su osnovni pojmovi, kako u programskom jeziku C#, tako i u programiranju uopšte. Cilj je da se učenici pre izrade aplikacija teorijski upoznaju sa onim što će koristiti. To su promenljive, tipovi podataka, operatori i upravljačke strukture. Početna strana ove elektronske lekcije može se videti na slici 4.

Slika 4: Elektronska lekcija - Osnovni pojmovi

3.1. Promenljive

Promenljive u programskom jeziku C# služe za čuvanje podataka. Mogu čuvati numeričke vrednosti, stringove ili objekte neke klase [3]. C# je programski jezik koji je strogo tipiziran, pa promenljiva ima svoj tip, ime i vrednost koju čuva. Prilikom deklarisanja promenljive navodi se tip i ime kao u primeru 1.

Primer 1: Deklarisanje promenljive

```
int ceo_broj;
```

Promenljiva se inicijalizuje, odnosno dodeljuje joj se vrednost na način prikazan u primeru 2.

Primer 2: Inicijalizovanje promenljive

```
ceo_broj = 5
```

Promenljiva se može inicijalizovati u istom trenutku u kom se deklariše kao u primeru 3.

Primer 3: Deklarisanje i inicijalizacija promenljive

```
int ceo_broj = 5;
```

Važno je napomenuti da je programski jezik C# koji je *case sensitive*. Ovo znači da razlikuje mala i velika slova. To bi značilo da promenljive `ceo_broj` i `Ceo_broj` nisu iste promenljive.

3.2. Tipovi podataka

C# je strogo tipiziran, što znači da svaki podatak ima svoj **tip**. Tip podatka može biti **vrednosni** (value type), **referentni** (reference type) i **pokazivački** (pointer type). Treba napomenuti da u elektronskim lekcijama nisu predstavljeni svi tipovi podataka, već samo oni koje bi učenici u okviru njih mogli da koriste. Tipovi podataka su predstavljeni tabelama u kojima se može videti definicija tipa, vrste tipa i primeri.

3.3.1. Celobrojni tip

Ako promenljiva čuva celobrojnu vrednost, koristi se celobrojni tip. U zavisnosti od karakteristike podatka, može se birati između više vrsta celobrojnih tipova. Ove vrste su u okviru elektronskih lekcija predstavljene tabelom koja se može videti na slici 5.

Tip podatka	Skup vrednosti	Zauzeće memorije u bajtovima
sbyte	od -128 do 127	1
byte	od 0 do 255	1
short	od -32768 do 32767	2
ushort	od 0 do 65535	2
int	od -2 147 483 648 do 2 147 483 647	4
uint	od 0 do 4 294 967 295	4
long	od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807	8
ulong	od 0 do 18 446 744 073 709 551 615	8

Slika 5: Vrste celobrojnog tipa

Primer 4. predstavlja inicijalizaciju promenljive koja je celobrojnog tipa `short` , što je takođe dato u elektronskim lekcijama klikom na dugme „Primeri“.

Primer 4: Inicijalizovanje promenljive tipa short

```
short ceo_broj1 = 1;
```

3.3.2. Realni tip

Realni tipovi podataka koji se koriste jesu `float` i `double`. Tip `float` je tip koji čuva 32-bitne, dok je `double` tip koji čuva 64-bitne vrednosti realnih brojeva sa pokretnim zarezom [3]. U okviru elektronskih lekcija ovo nije objašnjeno, već je samo navedena tabela prikazana na slici 6., koja prikazuje tip podatka i skup vrednosti.

Tip podatka	Skup vrednosti
float	od $\pm 1.5 \times 10^{-45}$ do $\pm 3.4 \times 10^{38}$
double	od $\pm 5.0 \times 10^{-324}$ do $\pm 1.7 \times 10^{308}$

Slika 6: Vrste realnog tipa

Potom je prikazan primer deklaracije i inicijalizacije promenljive realnog tipa (primer 5.).

Primer 5: Deklaracija i inicijalizacija realnog tipa

```
float realan_broj;
double realan_broj1 = 2.5;
```

Inicijalizovanje promenljive tipa `float` vrši se na način prikazan u primeru 6.

Primer 6: Inicijalizovanje promenljive tipa float

```
float realan_broj = 5.3F;
```

Ukoliko se ne bi navela oznaka F nakon realnog broja 5.3, pojavila bi se greška, s obzirom na to da se 5.3 tretira kao tip `double` i mora se eksplicitno naglasiti da je promenljiva tipa `float`.

3.3.3 Logički tip

Logički tip `bool` može imati dve vrednosti – tačno (`true`) i netačno (`false`). Promenljive tipa `bool` se inicijalizuju i deklarišu na način prikazan u primeru 7.

Primer 7: Inicijalizovanje i deklarisanje promenljive tipa bool

```
bool indikator = true;
bool indikator1;
```

Promenljiva `indikator` ima vrednost `true`, dok promenljiva `indikator1` ima vrednost `false` što je podrazumevana vrednost za logičke promenljive.

3.3.4. Znakovni tip

Ključna reč koja se koristi za deklarisanje promenljive znakovnog tipa je `char`. Vrednost promenljive znakovnog tipa je jedan karakter, odnosno znak. Taj znak je u stvari predstavljen kodom koji se nalazi u Unicode 16-bitnoj tabeli karaktera. U ovoj elektronskoj lekciji, pored definicije tipa `char` dat je link za Unicode tabelu karaktera, radi ilustracije o kakvim se karakterima radi.
Inicijalizacija promenljive tipa `char` izgledala bi kao u primeru 8.

Primer 8: Inicijalizacija promenljive tipa char

```
char znak = 'a';
```

Trebalo bi primetiti da se karakter ***a*** čija se vrednost čuva u promenljivoj `znak` piše između jednostrukih zagrada. Karakter `'a'` treba razlikovati od stringa `"a"`.

3.3.5. Tip niza

Nizovi će biti obrađeni kao posebna celina ovog rada, iako su ipak navedeni među tipovima podataka. Nizovni tip služi za čuvanje više podataka istog tipa. Više o nizovima biće izloženo kasnije. U okviru elektronskih lekcija je dat primer kreiranja niza celih brojeva, što je prikazano u primeru 9.

Primer 9: Inicijalizacija niza od 100 celobrojnih vrednosti

```
int[] brojevi;
brojevi = new int[100];
```

3.3.6. Tip string

Stringovi, odnosno niske predstavljaju nizove karaktera. Stringovi će, kao i nizovi, biti obrađeni kao posebno poglavlje ovog rada. Kao ilustracija objekta tipa string, u okviru elektronskih lekcija je dat primer 10.

Primer 10: Inicijalizovanje objekta tipa string

```
string ime_i_prezime = "Pera Perić";
```

3.3. Operatori

Operatori su simboli koji označavaju koja se operacija vrši. Postoji više podela operatora, a neke od podela su podela prema vrsti izraza u kojima se koriste i podela prema broju operanada.
U okviru elektronskih lekcija nisu navedeni svi operatori već samo oni koje bi učenici eventualno mogli da koriste. Izostavljeni su bitski operatori, čije izučavanje nije predviđeno za učenike srednjih škola.

3.3.1. Podela operatora prema vrsti izraza u kojima se koriste

Izrazi se grade od promenljivih i operatora. Mogu biti matematički i logički. U zavisnosti od vrste izraza operatori mogu biti: **aritmetički, operatori poređenja, operatori dodele, logički operatori, operatori uvećanja (inkrementiranja) i umanjenja (dekrementiranja).** Ove vrste operatora su prikazane u okviru elektronskih lekcija preko tabela koje sadrže simbol i opis operatora, gde je u okviru opisa operatora navedeno on čemu služi.

Aritmetički operatori služe za izvršavanje aritmetičkih operacija sa promenljivama brojevnog tipa. Mogu se primeniti nad celobrojnim i realnim tipovima. Važno je naglasiti da ukoliko se operator za deljenje primeni nad celobrojnim tipom, vrši se celobrojno deljenje. To je kasnije u okviru elektronskih lekcija obrađeno na konkretnim aplikacijama i naglašeno. Tabela sa aritmetičkim operatorima prikazana u ovoj elektronskoj lekciji može se videti na slici 7.

Simbol operatora	Opis operatora
+	sabiranje
-	oduzimanje
*	množenje
/	deljenje
%	ostatak pri deljenju (moduo)

Slika 7: Aritmetički operatori

Operatori poređenja služe za upoređivanje dva izraza. Rezultat primene operatora poređenja je logička vrednost koja može biti tačna ili netačna. Tabela koja prikazuje operatore poređenja prikazana u elektronskim lekcijama može se videti na slici 8.

Simbol operatora	Opis operatora
==	jednakost (jednako)
!=	nejednakost (različito)
<	manje od
>	veće od
<=	manje ili jednako
>=	veće ili jednako

Slika 8: Operatori poređenja

Operatori dodele su operatori koji vrednost desne strane izraza dodeljuju levoj strani izraza. U tabeli koja je prikazana u okviru elektronske lekcije o osnovnim pojmovima, a može se videti na slici 9., izložen je malo detaljniji opis ovih operatora.

Simbol operatora	Opis operatora
=	Dodeljuje vrednost desne strane levoj
+=	Umesto zapisa a=a+b, može da se koristi a+=b
-=	Umesto zapisa a=a-b, može da se koristi a-=b
*=	Umesto zapisa a=a*b, može da se koristi a*=b
/=	Umesto zapisa a=a/b, može da se koristi a/=b
%=	Umesto zapisa a=a%b, može da se koristi a%=b

Slika 9: Operatori dodele

Logički operatori se primenjuju nad logičkim vrednostima i takođe vraćaju logičku vrednost. Pri tome logičko I i logičko ILI se primenjuju nad dva operanda, a operator negacije nad jednim operandom. U okviru elektronskih lekcija je navedena tabela koja prikazuje logičke operatore i može se videti na slici 10. Ono što nije navedeno je tablica u kojoj je prikazano kako se nad dva iskaza u zavisnosti od njihove

logičke vrednosti primenjuju operatori && i ||, kao ni tablica logičke negacije. Ovo nije navedeno iz razloga što su elektronske lekcije namenjene za učenike srednjih škola, a već u prvom razredu srednje škole učenici se susreću sa ovim tablicama u okviru predmeta matematika.

Simbol operatora	Opis operatora
!	negacija
||	logičko ILI (or)
&&	logičko I (and)

Slika 10: Logički operatori

Operatori uvećanja i umanjenja služe da vrednost promenljive umanje ili uvećaju za 1, a detaljniji opis postfiksne i prefiksne notacije operatora dat je u tabeli u okviru ove elektronske lekcije, što je prikazano na slici 11.

Simbol operatora	Opis operatora
++a	Prvo se uveća vrednost promenljive a za 1, pa se dalje koristi u izrazu
a++	Iskoristi se trenutna vrednost promenljive a u izrazu, pa se krajnji rezultat poveća za 1
--a	Prvo se smanji vrednost promenljive a za 1, pa se dalje koristi u izrazu
a--	Iskoristi se trenutna vrednost promenljive a u izrazu, pa se krajnji rezultat smanji za 1

Slika 11: Operatori inkrementiranja i dekrementiranja

3.3.2. Podela operatora prema broju operanada

Prema broju operanada operatori mogu biti:
1. ***Unarni*** – operatori koji se primenjuju nad jednim operandom, kakav je npr. operator ! (logička negacija).
2. ***Binarni*** – operatori koji se primenjuju nad dva operanda, npr. operatori +, * itd.
3. ***Ternarni*** – operator koji se primenjuje nad tri operanda. Jedan od primera ovakvog operatora je operator ?:, o kome će biti reč u poglavlju o uslovnim naredbama ovog rada.

3.3.3. Prioritet operatora

Prioritet operatora predstavlja redosled izvršavanja operatora u izrazu. Najviši prioritet imaju zagrade, a prioritet ostalih operatora prikazan je na slici 12. koja se može videti u okviru elektronskih lekcija.

Primarni	x++ x--
Unarni	! ++x --x - (predznak broja)
Aritmetički: množenje, deljenje, moduo	* / %
Aritmetički: sabiranje, oduzimanje	+ -
Operatori poređenja	< > <= >=
Jednako, različito	== !=
Ločigko I	&&
Ločigko ILI	||
Uslovni	?:
Opratori dodele	= *= /= %= += -=

Slika 12: Redosled izvršavanja nekih operatora

3.4. Upravljačke strukture

Upravljačke strukture programa definišu redosled izvršavanja linija koda. Linije koda mogu imati linijski tok izvršavanja, redosled izvršavanja može zavisiti od ispunjenja nekog uslova ili se njihovo izvršavanje može ponavljati više puta. Na osnovu ovoga javljaju se tri osnovna tipa programskih struktura:

1. Sekvenca.

2. Selekcija (grananje).

3. Iteracija (petlja).

3.4.1. Sekvenca

Glavna karakteristika sekvence je da se naredbe pišu u okviru bloka između vitičastih zagrada i da se izvršavaju linija po linija. U okviru elektronskih lekcija je prikazano izvršavanje sekvence kao u primeru 11.

Primer 11: Sekvenca

```
{
    prva_linija_koda;
    druga_linija_koda;
    treca_linija_koda;
          ...
    poslednja_linija_koda;
}
```

Treba napomenuti i da se svaka naredba u programskom jeziku C# završava znakom **;** (tačka zarez).

3.4.2. Selekcija

Za razliku od sekvence koja ima linijski tok izvršavanja, selekcija odnosno grananje pruža mogućnost izbora. Kod se izvršava u zavisnosti od ispunjenja uslova. Uslovi mogu biti različiti, što je detaljnije opisano u celini „Uslovne naredbe“. Ovakav tip strukture zahteva ključne reči `if`, `else`, `switch` i `case` . Opšti oblik selekcije prikazan je pseudokodom koji se može videti u primeru 12.

Primer 12: Selekcija

```
if (uslov)
{
    kod_ako_je_uslov_ispunjen;
}
else
{
    kod_ako_uslov_nije_ispunjen;
}
```

Navedeno je jednostavno grananje sa dva toka izvršavanja. Prvi tok je dešavanje u slučaju da je neki uslov ispunjen, a drugi tok dešavanje ako uslov ne važi. Grananje može biti i složenije i program može imati više tokova izvršavanja.

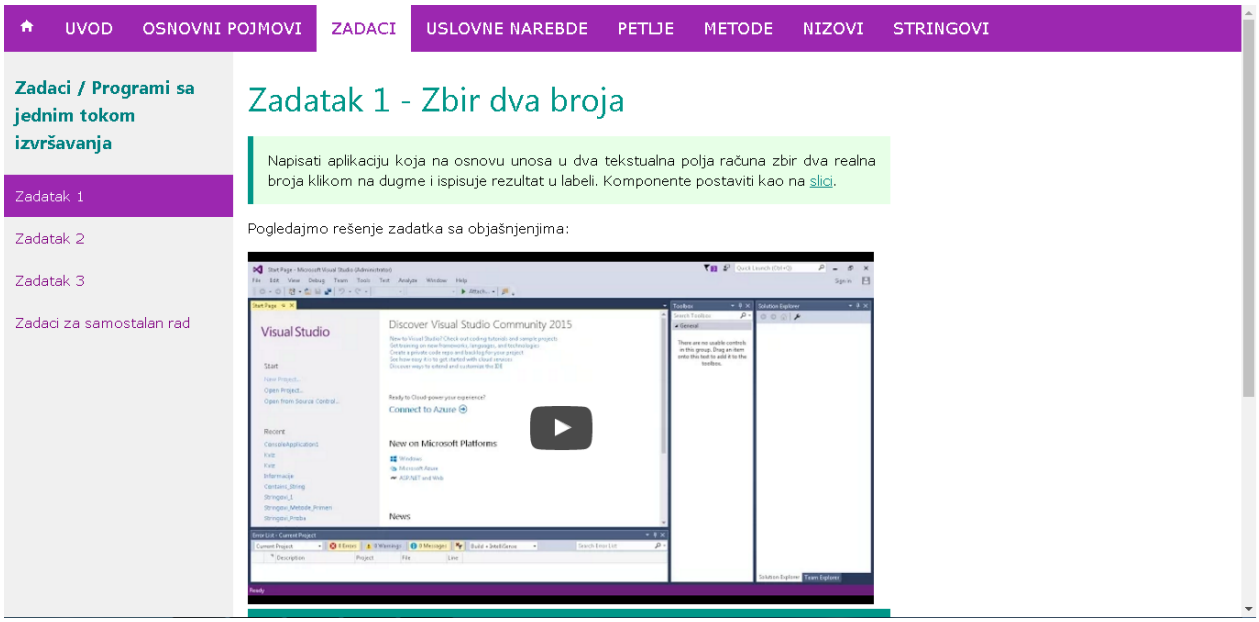
3.4.3. Iteracija (petlja)

Kada je potrebno neku naredbu ili blok naredbi izvršavati više puta, koristi se petlja. Naredbe se ponavljaju u više iteracija čiji broj može biti unapred poznat, a i ne mora. U zavisnosti od toga petlje se dele na petlje sa konstantnim brojem prolaza i sa promenljivim brojem prolaza.
U programskom jeziku C# ključne reči za petlje sa konstantnim brojem prolaza, koje se nazivaju i brojačkim petljama, su `for` i `foreach`. Za petlje sa promenljivim brojem prolaza koriste se ključne reči `while` i `do while`. Više govora o petljama biće u njima posvećenom odeljku.

4. Programi sa linijskim tokom izvršavanja

Cilj ovog poglavlja elektronskih lekcija, predstavljenog jednostavno naslovom „Zadaci“, jeste da se učenici upoznaju sa najjednostavnijim aplikacijama čiji je tok izvršavanja linijski. Data su tri „početna“ zadatka čija su rešenja detaljno objašnjena preko video materijala koji prati svaki od njih. Takođe, data su dva zadatka za samostalan rad. Ova dva zadatka učenici mogu najpre samostalno da rešavaju, pa potom da pogledaju rešenje, kao i da ga preuzmu.

Početna strana ove elektronske lekcije prikazana je na slici 13.



Slika 13: Elektronska lekcija – Programi sa jednim tokom izvršavanja

4.1. Aplikacija za sabiranje dva broja

Kao prvi zadatak učenicima je dato kreiranje aplikacije koja sabira dva realna broja. Korisnik u tekstualna polja (kontrola *TextBox*) unosi dva realna broja i klikom na dugme vrši se njihovo sabiranje i ispis rezultata u labeli.

Kod ovog zadatka učenici se prvo upućuju preko video materijala na to šta je forma i kako se dodaju komponente na formu. Kada se komponente dodaju, programira se dugme. Koriste se promenljive `a` i `b`, tipa `double` i u njih se smeštaju vrednosti koje je korisnik uneo preko tekstualnih polja. Ovde je neophodno primeniti metodu *Parse()* iz klase Double, jer su podaci u kontroli *TextBox* tipa `string`. Nakon učitavanja, brojevi se sabiraju aritmetičkim operatorom +, pa se smeštaju u promenljivu `zbir`, takođe tipa `double`. Ispis u labelu se vrši tako što se na labelu primeni svojstvo *Text* i dodeli vrednost promenljive `zbir` koja se prebaci u tip `string` metodom *ToString()*. Deo programskog koda koji se izvršava klikom na dugme prikazan je u primeru 13.

Primer 13: Deo programskog koda preko koga se vrši sabiranje dva realna broja

```
double a, b;
a = double.Parse(textBox1.Text);
b = double.Parse(textBox2.Text);
double zbir = a + b;
label1.Text = zbir.ToString();
```

4.2. Aplikacija za izračunavanje obima i površine pravougaonika

Drugi zadatak koji je predstavljen u okviru ove celine jeste zadatak da se napravi aplikacija koja na osnovu unetih stranica pravougaonika preko kontrola tipa *TextBox* računa obim i površinu pravougaonika i ispisuje njihove vrednosti u odgovarajućim labelama. Aplikacija ima dva dugmeta – preko jednog se

računa obim, a preko drugog površina. Za rešavanje se koriste dobro poznate matematičke formule, koje poznaju kako učenici srednjih, tako i osnovnih škola. Takođe, zahteva se da stranice pravougaonika budu celi brojevi.

Ovaj zadatak se rešava tako što se iz *TextBox* kontrola učitaju dva cela broja i smeste se u dve promenljive `a` i `b` tipa `int`. Sada se koristi metoda *Parse()* iz klase Int, koja omogućava da se stringovi iz kontrola tipa *TextBox* prebace u tip `int`, ako je moguće. Klikom na dugme „Obim“ izvršava se izračunavanje vrednosti promenljive `obim` po formuli *2a + 2b* i upisuje u odgovarajuću labelu (ponovo primena svojstva *Text*, dok se klikom na dugme „Površina“ računa vrednost promenljive `povrsina` po formuli $a \cdot b$ i takođe upisuje u odgovarajuću labelu.

4.3. Aplikacija za izračunavanje zbira cifara trocifrenog broja

Još jedan od zadataka koji je detaljno objašnjen u okviru elektronske lekcije „Zadaci“ je zadatak koji zahteva kreiranje aplikacije koja računa zbir cifara trocifrenog broja. Korisnik unosi trocifren broj u *TextBox* i klikom na dugme računa se zbir njegovih cifara i ispisuje se u odgovarajući *TextBox*.

Za rešavanje ovog zadatka potrebno je na neki način izdvojiti cifre unetog broja, pa ih potom sabrati. Najvažnije je da učenici shvate način „izdvajanja cifara“, koji se svodi na primenu operatora za celobrojno deljenje i moduo. Cifre se izdvajaju na sledeći način:

- Cifra jedinica se izdvaja tako što se odredi ostatak pri deljenju trocifrenog broja sa 10 (npr. za broj 456 to je broj 6):

```
jedinica = broj % 10;
```

-Cifra stotina se izdvaja tako što se trocifren broj celobrojno podeli sa 100 (npr.za broj 456 to je broj 4):

```
stotina = broj / 100;
```

-Za izdvajanje cifre desetica neophodno je izračunati ostatak pri deljenju trocifrenog broja sa 100 (npr. za broj 456 to je broj 56), pa potom taj rezultat celobrojno podeliti sa 10 (npr. 56/10 = 5):

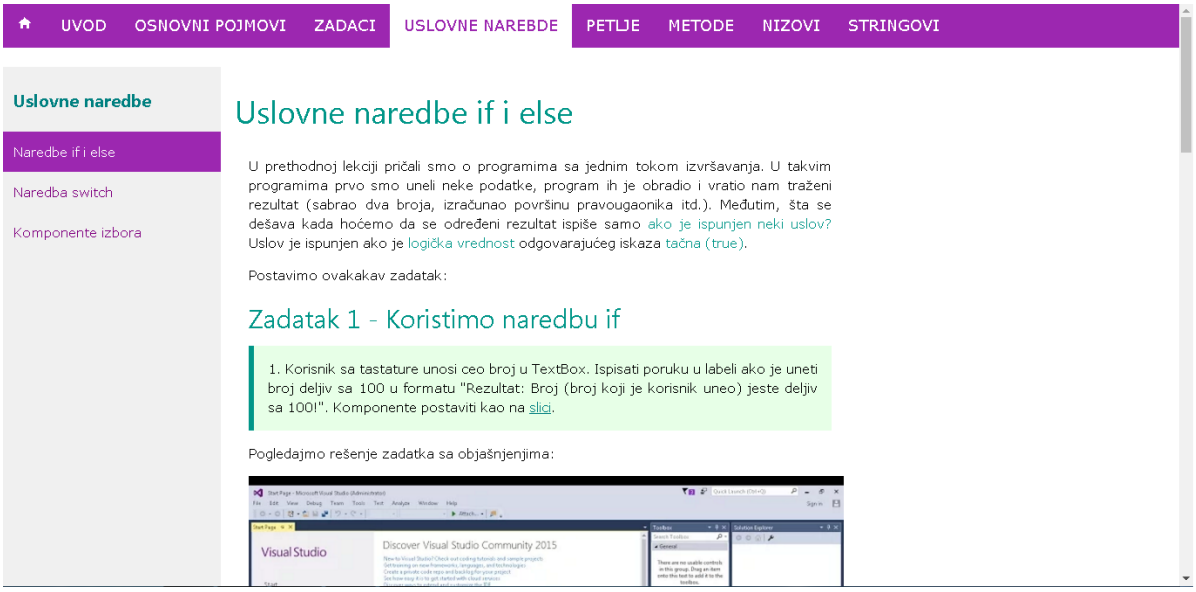
```
desetica = (broj % 100) / 10;
```

4.4. Zadaci za samostalan rad

U okviru ove elektronske lekcije nalazi se i deo „Zadaci za samostalan rad“ gde su dati tekstovi za dva zadatka i rešenja. Njihov cilj je da učenici probaju samostalno da ih reše pre nego što pogledaju rešenje pri tom koristeći prethodno urađene zadatke i lekcije o operatorima i tipovima podataka.
Prvi zadatak zahteva kreiranje jednostavne aplikacije, mini kalkulatora sa 4 osnovne računske operacije, dok se kod drugog zadatka traži izračunavanje obima pravouglog trougla kome su date jedna kateta i hipotenuza.

5. Uslovne naredbe

Programi razmatrani u prethodnom poglavlju su bili programi sa jednim tokom izvršavanja. Kao takvi, uglavnom su se svodili na neko izračunavanje. Za realizovanje programa koji ima više tokova izvršavanja, koristi se upravljačka struktura koja se zove selekcija. Na taj način program može zavisiti od korisnikovog unosa, što pruža velike mogućnosti prilikom programiranja.
Elektronska lekcija u kojoj su prikazane uslovne naredbe prikazana je na slici 14.



Slika 14: Elektronska lekcija - Uslovne naredbe

5.1. Naredbe if i else

Naredba `if` određuje da li će se blok naredbi izvršiti u zavisnosti od logičke vrednosti iskaza. Naredba `else` se koristi za izvršavanje dela koda ukoliko se blok naredbi u okviru if naredbe nije izvršio. Moguće je i nakon naredbe `else` upotrebiti naredbu `if` što je prikazano u primeru 14.

Primer 14: Pseudokod koji ilustruje primenu naredbi if i else

```
if (uslov1)
{
    kod_ako_je_ispunjen_uslov1;
}
else if (uslov2)
{
    kod_ako_je_ispunjen_uslov2;
}
else
{
    inače;
}
```

5.1.1. Prikaz naredbi if i else

Korišćenje naredbi `if` i `else` je u okviru elektronske lekcije o uslovnim naredbama prikazano kroz dva zadatka. U prvom zadatku pravi se aplikacija preko koje se unosi ceo broj u *TextBox*, a u labeli se ispisuje poruka ukoliko je uneti broj deljiv sa 100. Drugi zadatak predstavlja unapređenje prvog zadatka, tako što se u labeli ispisuje da broj jeste deljiv sa 100 ili da broj nije deljiv sa 100. Takođe, u okviru drugog zadatka učenici se upoznaju sa naredbama `try` i `catch`. Oba zadatka su detaljno rešena u okviru pratećih video materijala.

Za rešavanje prvog zadatka, dovoljna je naredba `if`. Programski kod koji se izvršava klikom na dugme je prikazan u primeru 15.

Primer 15: Primena naredbe if u okviru zadatka 1 celine „Uslovne naredbe“

```
int broj;
broj = int.Parse(textBox1.Text);
if (broj % 100 == 0)
{
    label2.Text = label2.Text + "Broj " + broj + " je deljiv sa 100!";
}
```

U zadatku se vrši se provera uslova: `broj % 100 == 0`. Ovaj uslov ima logičku vrednost `true` ako je broj koji je korisnik uneo sa tastature deljiv sa 100. Samo ako je logička vrednost uslova `true` izvršava se kod u okviru vitičastih zagrada, odnosno prikazuje se poruka u labeli.

Drugi zadatak je dopuna prvog. Programski kod prikazan u primeru 16. to opisuje.

Primer 16: Primena naredbi if i else u okviru drugog zadatka celine „Uslovne naredbe“

```
try
{
    int broj;
    broj = int.Parse(textBox1.Text);
    if (broj % 100 == 0)
    {
        label2.Text = label2.Text + "Broj " + broj + " je deljiv sa 100!";
    }
    else
    {
        label2.Text = label2.Text + "Broj " + broj + " nije deljiv sa 100!";
    }
}
catch
{
    MessageBox.Show("Nije ispravan unos!", "Greška");
}
```

Kroz ovaj zadatak je prikazano šta se dešava u zavisnosti od uslova, odnosno da program ima dva toka izvršavanja.

5.1.2. Naredbe try i catch

Naredbe `try` i `catch` služe za „hvatanje“ grešaka. Konkretno, u drugom zadatku može se desiti da korisnik aplikacije unese u *TextBox* bilo koji karakter sa tastature, slovo, reč, rečenicu itd. Tada ne može da se primeni metoda *int.Parse()*, koja čita podatak tipa `string` iz kontrole tipa *TextBox* i konvertuje ga u tip `int`, pa program prijavljuje grešku.
Da se ovo ne bi dešavalo, ceo programski kod koji se eventualno može izvršiti ako je korisnikov unos pravilan (zaista je uneo ceo broj) smesti se u okviru bloka `try` (eng. pokušati). Program pokušava da izvrši naredbe u okviru bloka `try` i ukoliko ne dođe do greške, blok `catch` se ne izvršava. Ukoliko se greška pojavi, izvrši se blok `catch` preko koga se obaveštava korisnik da je pronađena greška. Korisnik se obaveštava o greški preko objekta klase MessageBox.

MessageBox je klasa koja omogućava prikaz određenih poruka korisniku u posebnom prozoru koji blokira rad ostalih prozora aplikacije dok ga korisnik ne zatvori.

5.1.3. Uslovni operator

Uslovni operator može da služi kao zamena za naredbe `if` i `else` kada se koriste zajedno. Primer 17. predstavlja korišćenje naredbi `if` i `else` u kombinaciji.

Primer 17: Kombinacija naredbi if i else

```
if (uslov)
{
    kod_ako_je_ispunjen_uslov;
}
else
{
    kod_ako_nije_ispunjen_uslov;
}
```

Pseudokod u primeru 17. može se zameniti pseudokodom prikazanim u primeru 18., koji ilustruje korišćenje uslovnog operatora.

Primer 18: Primena uslovnog operatora

```
Uslov ? kod_ako_je_ispunjen_uslov : kod_ako_nije_ispunjen_uslov;
```

5.2. Naredba switch

Naredba `switch` se koristi kod višestrukog grananja. Nekada je jednostavnije da se umesto primene višestrukih `if` naredbi koristi naredba `switch` kod koje se vrši grananje u zavisnosti od neke promenljive ili izraza. Opšti oblik naredbe dat je pseudokodom prikazanim primerom 19.

Primer 19: Opšti oblik naredbe switch

```
switch (izraz)
{
    case slučaj1:
        naredbe1;
        break;
    case slučaj2:
        naredbe2;
        break;
         ...
    case slučajn:
        naredben;
        break;
    default:
        naredbe;
        break;
}
```

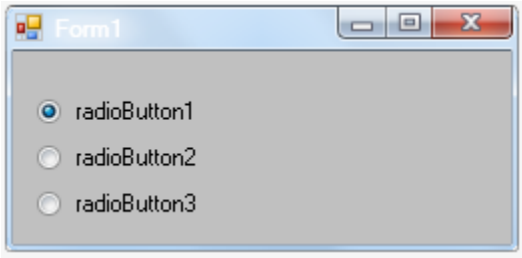
Izraz u zagradi može biti promenljiva ili izraz i njegova vrednost određuje koje će se naredbe u okviru `case` izvršiti. Slučaj u okviru naredbe `case` se izvršava ako se njegova vrednost poklopi sa izrazom, pa se naredbom `break` izlazi iz `switch` bloka naredbi. Ukoliko se nijedan od slučajeva ne poklopi sa izrazom, vrše se naredbe u okviru `default`.

5.3. Komponente izbora

Komponente izbora obrađene u okviru elektronskih lekcija su komponenta *RadioButton* i komponenta *CheckBox*. Kako ove komponente omogućavaju grananje u programu i najčešće se koriste sa uslovnim naredbama, upravo je njihova primena i prikazana u toj celini.

5.3.1. Komponenta RadioButton

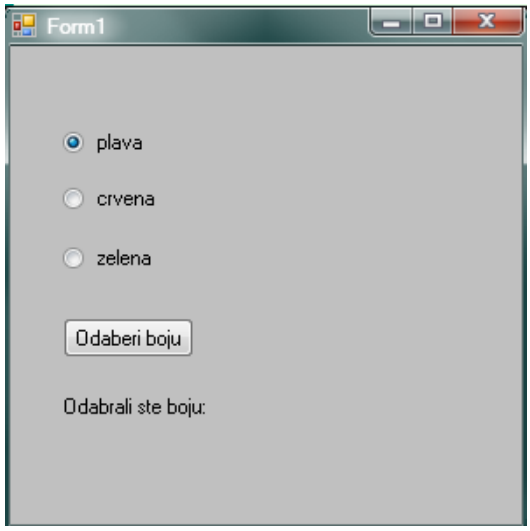
Komponenta ***RadioButton*** (radio dugme) je kontrola koja omogućava korisniku selektovanje samo jednog od više izbora i koristi se uparena sa drugim *RadioButton* kontrolama [3]. Izgled ove kontrole može se videti na slici 15.



Slika 15: Forma sa tri kontrole tipa RadioButton

Najvažnije svojstvo kontrole *RadioButton* je svojstvo *Checked*. Podrazumevana vrednost ovog svojstva je `false`, što bi značilo da *RadioButton* nije selektovan. Kada je svojstvo *Checked* postavljeno na `true`, to znači da je opcija selektovana i pored imena selektovanog radio dugmeta se markira kružić.

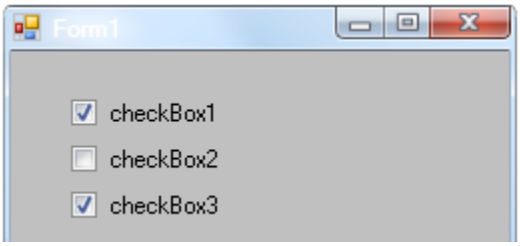
U okviru ove elektronske lekcije je primena kontrole *RadioButton* objašnjena kroz primer (slika 16.), gde su na formi postavljena tri radio dugmeta sa natpisima boja: plava, crvena i zelena. U zavisnosti od izbora radio dugmeta, korisniku se u objektu *MessageBox* prikazuje koju boju je izabrao. Ovakav jednostavan primer je odabran kako bi učenici razgraničili kada se koristi kontrola *RadioButton*, a kada kontrola *CheckBox*, koja pruža mogućnost izbora više opcija.



Slika 16: Primer aplikacije koja koristi kontrole tipa RadioButton

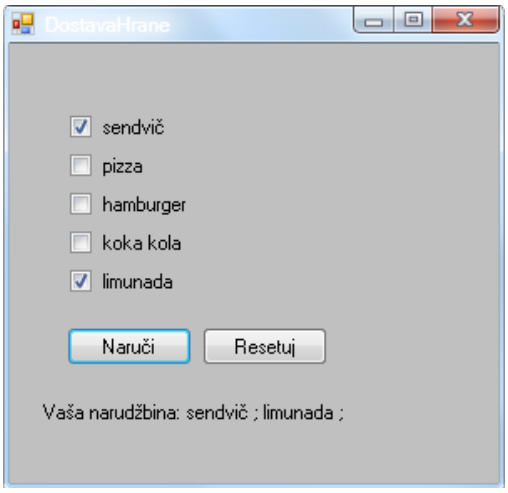
5.3.2. Komponenta CheckBox

Komponenta ***CheckBox*** (ček boks) je kontrola koja omogućava biranje jedne ili više ponuđenih, međusobno nezavisnih opcija. Ključno svojstvo, kao i kod kontrole *RadioButton*, je svojstvo *Checked* koje uzima vrednosti `true` ili `false`. Izgled ove komponente prikazan je na slici 17.



Slika 17: Forma sa tri kontrole tipa CheckBox

Učenicima je na primeru objašnjena upotreba ove kontrole. Primer je aplikacija za dostavu hrane preko koje korisnik može da čekira koju hranu ili piće želi da naruči, a potom se u zavisnosti od čekiranih opcija ispisuje šta je sve to korisnik „naručio“, kao što je prikazano na slici 18.



Slika 18: Primer aplikacije koja koristi kontrole tipa CheckBox

6. Petlje

Petlja je programska upravljačka struktura koje se koristi kada naredbu ili blok naredbi treba ponoviti više puta. Izgled elektronske lekcije o petljama može se videti na slici 19.

Slika 19: Elektronska lekcija – Petlje

Učenicima je kao motivacija zašto je neophodno koristiti petlje u programiranju dat zadatak da se u nekoj od tekstualnih kontrola ispišu brojevi od 1 do 1000. Naravno, moguće je 1000 puta dati naredbu koja ispisuje broj 1, broj 2 itd., ali takvo ispisivanje je mukotrpan posao. U ovakvim slučajevima, kada je očigledno da bi neku „radnju“ trebalo ponavljati, koristi se petlja. Prva petlja koja je obrađena jeste `for` petlja, preko koje je primer za ispisivanje brojeva od 1 do 1000 i urađen.

6.1. For petlja

Petlja `for` se koristi kada je poznat tačan broj prolazaka kroz petlju. Iz tog razloga `for` petlja spada u brojačke petlje. Tok petlje bi bio sledeći: inicijalizuje se brojač, postavi se uslov petlje i postavi se korak petlje. U opštem obliku `for` petlja izgleda kao u primeru 20., što je prikazano u okviru ove elektronske lekcije.

Primer 20: Opšti oblik for petlje

```
for (inicijalizacija; uslov; korak)
{
    telo petlje;
}
```

U okviru ove elektronske lekcije je obezbeđeno da se klikom na reči „inicijalizacija“, „uslov“, „korak“ i „telo petlje“ dobiju i dodatna objašnjenja. To su sledeća objašnjenja:

1. *Inicijalizacija* – predstavlja inicijalizovanje promenljivih u petlji,npr. inicijalizovanje brojača petlje.
2. *Uslov* – izraz koji ograničava petlju. Dok god je ispunjen uslov, telo petlje se izvršava.
3. *Korak* – izraz kojima se definiše promena vrednosti promenljivih u petlji.
4. *Telo petlje* – blok naredbi koji se ponavlja dok god važi uslov.

Treba napomenuti i da se bilo koji od navedenih izraza može izostaviti. Ako se izostave svi izrazi, ulazi se u beskonačnu petlju čija je sintaksa: `for ( ; ; ).`

6.1.1. Zadaci u kojima se primenjuje for petlja

Nakon teorijskog uvoda, urađena su dva zadatka koja ilustruju primenu `for` petlje.
Prvi od tih zadataka je napisati aplikaciju koja u kontroli *RichTextBox* ispisuje brojeve od 1 do 1000, što je i bila početna motivacija za uvođenje petlji. Kako treba ispisati brojeve od 1 do 1000, znači da telo petlje treba izvršiti 1000 puta. Uvodi se promenljiva `i` koja predstavlja brojač i inicijalizuje na 1. Petlju se izvršava 1000 puta, odnosno dok god je brojač manji ili jednak 1000 što se postavlja kao uslov petlje. Korak je ovde izraz koji uvećava brojač za 1 nakon svakog prolaska kroz petlju. Najzad u telu petlje daje se naredba za ispis. Ispisuje se baš trenutni brojač `i`, odnosno u prvom prolazu 1, u drugom 2 itd. Ovo je prikazano programskim kodom u primeru 21.

Primer 21: For petlja koja ispisuje brojeve od 1 do 1000 u RichTextBox

```
for (int i = 1; 1 <= 1000; i++)
{
    richTextBox1.Text += i + "\n";
}
```

U drugom zadatku zahteva se pisanje aplikacije koja sumira brojeve od 1 do unetog broja n. Uvodi se promenljiva `suma` koja se inicijalizuje na 0. Ovde se petlja vrti n puta, a u okviru tela petlje se uvećava suma za brojač `i`. U programskom kodu to izgleda kao u primeru 22.

Primer 22: For petlja preko koje se sumiraju brojevi od 1 do n

```
suma = 0;
for (int i = 1; 1 <= n; i++)
{
    suma = suma + i; //na sumu čija je početna vrednost 0 u svakom prolazu dodaje se
    //vrednost brojača i
}
```

Ovaj primer može biti od koristi učenicima kao ideja za rešavanja zadataka gde je potrebno izračunati sumu ili proizvod nekog niza koji zavisi od n.

6.2. While petlja

Petlja `while` nije brojačka petlja, već petlja u kojoj je broj prolaza ograničen uslovom. Kada određeni uslov prestane da važi, izlazi se iz petlje. U okviru elektronske lekcije o petljama je dat opšti oblik `while` petlje prikazan u primeru 23 .

Primer 23: While petlja

```
while (uslov)
{
    telo petlje;
}
```

Ovo može da se shvati kao: dok god se ne ispuni uslov, petlja se vrti. Kad se uslov ispuni, prekida se petlja. Napomenuto je da se `for` petlja može zameniti `while` petljom, pa je zadatak sa ispisom brojeva do 1000 „preveden“ na `while` petlju, što je prikazano u primeru 24.

Primer 24: Petlje for i while koje daju isti rezultat

```
for (int i = 1; 1 <= 1000; i++)
//inicijalizuje se brojač koji se kreće
//od 1 do 1000 sa korakom 1
{
    richTextBox1.Text += i + "\n";
    //u RichTextBox se upisuje svaki broj
    //i iz petlje
}
```

```
int i=1;
//promenljiva i se koristi kao zamena
//brojača
while(i <= 1000)
{
    richTextBox1.Text += i + "\n";
    i++;
    //promenljiva i se uvećava za 1 pri
    //svakom prolazu kroz petlju dok se
    //ne zadovolji uslov
}
```

6.2.1. Zadaci u kojima se primenjuje while petlja

Primena `while` petlje je objašnjena kroz dva zadatka. Prvi zadatak predstavlja aplikacija koja računa faktorijel unetog broja n, odnosno proizvod brojeva od 1 do n. Postavlja se početna vrednost proizvoda na 1. Inicijalizuje se promenljiva `i`, koja se uvećava za 1 prilikom svakog prolaska kroz petlju. Petlja se vrti

sve dok `i` ne postane veće od n, a u telu petlje se vrši množenje proizvoda sa `i`. Naravno, ovo je moglo biti urađeno i preko `for` petlje.

Kod drugog zadatka vrši se ispis kvadrata brojeva do unetog broja n, ali od najvećeg do najmanjeg (obrnuto) i izračunava se njihova suma. Suma se na početku inicijalizuje na 0. Ovde ne mora da se koristi imitacija brojača, odnosno promenljiva `i` čija se početna vrednost inicijalizuje. Jednostavno, petlja se vrti sve dok je uneti broj n veći od 0, a prilikom svakog prolaska kroz petlju, n se smanjuje za 1. U telu petlje vrši se ispis trenutnog kvadrata broja n, kao i sumiranje. Deo programskog koda koji ovo radi prikazan je u primeru 25.

Primer 25: Ispisivanje kvadrata brojeva od 1 do n i sumiranje preko while petlje

```
while(n > 0)
{
    richTextBox1.Text += n*n +"\n";
    suma += n * n;
    n--;
}
```

6.3. Do while petlja

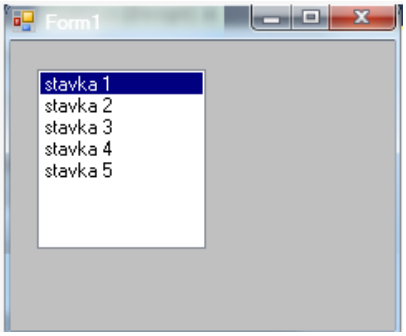
Petlja `do while` nije brojačka petlja. Razlika između petlji `while` i `do while` je sledeća: kod petlje `do while` se prvo izvrši telo petlje, pa se proverava uslov petlje. To znači da se kod ove petlje makar jednom prođe kroz petlju, za razliku od petlje `for` i petlje `while`, gde se u petlju i ne ulazi ako uslov ne važi. Opšti oblike petlje `do while` je prikazan u primeru 26.

Primer 26: Petlja do while

```
do
{
    telo petlje;
}
while (uslov);
```

6.3.1. Primena petlje na popunjavanje stavki kontrole ListBox

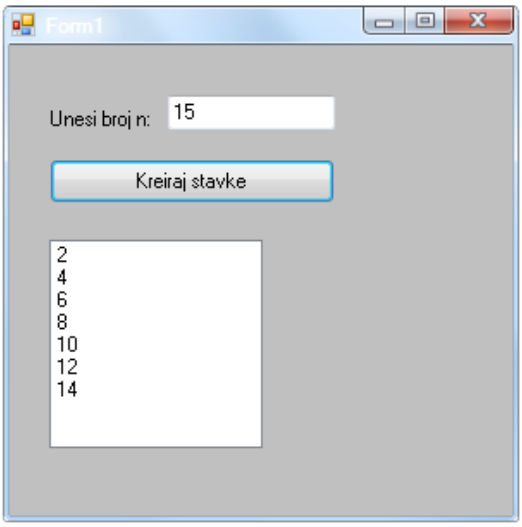
U okviru elektronske lekcije o petljama dat je zadatak u kome se učenici, osim sa petljom `do while`, prvi put susreću i sa kontrolom ***ListBox***. Mesto gde se obrađuju petlje našlo se pogodnim za upoznavanje sa ovom kontrolom. Naravno, kontrolu je moguće ispuniti „ručno“ stavkama, što je prikazano preko video materijala. Pored takvog načina popunjavanja, jedan od načina je i preko petlje.
Kontrola *ListBox* predstavlja listu sa opcijama, odnosno stavkama. Izgled ove kontrole može se videti na slici 20.



Slika 20: Komponenta ListBox sa 5 stavki

Jedno od najvažnijih svojstava kontrole *ListBox* jeste svojstvo *Items*, preko koga se pristupa stavkama liste. Za dodavanje stavke u listu koristi se metoda *Add()* koja kao argument prima objekat koji se dodaje kao stavka.

Učenicima je postavljen zadatak da se napiše aplikacija koja popunjava *ListBox* stavkama, parnim brojevima od 1 do unetog broja n (slika 21).



Slika 21: Aplikacija koja prikazuje kontrolu ListBox

Koristi se petlja `do while`. Najpre se celobrojna promenljiva `i` postavi na 1. Nakon ulaska u petlju ispita se da li je `i` paran broj, pa ako jeste, primenom metode *Add ()* broj `i` se dodaje kao stavka liste. Nakon toga se `i` uvećava za 1. Ovaj postupak se ponavlja sve dok `i` ne veće od n. Preko primera 27. prikazan je programski kod koji ovo radi.

Primer 27: Dodavanje stavki u kontrolu ListBox preko do while petlje

```
int i = 1;
do
{
    if (i % 2 == 0)
    {
        listBox1.Items.Add(i);
    }
    i++;
}
while (i<=n);
```

6.4. Foreach petlja

Programski jezik C#, pored tri osnovne petlje, omogućava i korišćenje petlje `foreach` radi jednostavnije iteracije kroz uređene skupove objekata. Postoje dva načina za grupisanje objekata, a to su nizovi objekata i kolekcije objekata. Nizovi se koriste za rad sa fiksnim brojem strogo tipiziranih objekata, dok su kolekcije pogodnije za fleksibilniji rad sa objektima u smislu da broj objekata u kolekciji može da se menja u zavisnosti od potrebe aplikacije i može da sadrži objekte različitog tipa [3]. Iteracija kroz uređene skupove objekata je jednostavnija ako se koristi petlja `foreach` jer njena primena ne zahteva poznavanje veličine skupa objekata, već iterator prolazi kroz skup objekata od početka do kraja. U okviru elektronske lekcije o petljama dat opšti oblik petlje `foreach`, što se može videti u primeru 28.

Primer 28: Petlja foreach

```
foreach (tip_promenljive ime_promenljive in izraz)
{
    telo petlje;
}
```

6.4.1. Aplikacija koja ilustruje primenu petlje foreach

U okviru elektronskih lekcija u celini o petljama je kao prezentacija petlje `foreach` učenicima postavljen zadatak da se napiše aplikacija preko koje se za uneti broj n popunjavaju stavke kontrole tipa *ListBox* brojevima od 1 do n. To bi trebalo uraditi preko `for` petlje, a potom preko petlje `foreach` u kontroli *RichTextBox* ispisati kubove svih brojeva koji se nalaze kao stavke liste.
Ovaj zadatak kombinuje primenu dve vrste petlji. Kada se lista popuni preko `for` petlje, ostaje da se prođe kroz nju petljom `foreach` kako bi se pristupilo njenim stavkama. U primeru 29. prikazan je deo koda koji ilustruje primenu petlje `foreach` u ovom zadatku.

Primer 29: Ispis kubova brojeva koji su stavke liste preko petlje foreach

```
foreach (int element in listBox1.Items)
{
    richTextBox1.Text += "Kub broja " + element + " je " + Math.Pow(element, 3) + "\n";
}
```

Petlja `foreach` koristi element tipa `int` koji se nalazi u kolekciji *listBox1.Items* i za svaki takav element, dok se ne stigne do kraja kolekcije vrši određenu naredbu. U ovom zadatku naredba je upisivanje u *RichTextBox* odgovarajućeg elementa i računanje i ispisivanje kuba tog elementa. Ovakav prolazak kroz kolekciju je jednostavan i efikasan, iz razloga što nema potrebe za poznavanjem dimenzije kolekcije. Petlja `foreach` jednostavno prolazi od prvog do poslednjeg člana kolekcije, pa se izlazi iz petlje.
Kod ovog zadatka, učenicima je predstavljena i metoda *Pow()* koja se nalazi u klasi Math. Ova funkcija služi za stepenovanje dva broja. Iako je umesto ove funkcije moglo da se upotrebi množenje elementa tri puta samim sobom, ipak je korisnije da učenici znaju da postoje različite metode koje omogućavaju da se neke stvari jednostavnije i u mnogim slučajevima brže urade.

7. Metode

Metode su objedinjena grupa naredbi koje se primenjuju nad objektom [4]. U objektno orijentisanom programiranju, metode se definišu u okviru klasa. U prethodnim elektronskim lekcijama učenici su se već susreli sa nekim definisanim metodama, npr. metoda *Pow()* iz klase Math. Cilj elektronske lekcije o metodama je upoznavanje učenika sa definisanjem sopstvenih metoda. Početna strana elektronske lekcije o metodama data je na slici 22.

Slika 22: Elektronska lekcija - Metode

7.1. Definisanje metode

Za definisanje metode koristi se opšti oblik definicije [4]:

```
<Vidljivost> <Tip povratne vrednosti> <Ime Metode>(Lista parametara)
{
   telo metode;
}
```

Vidljivost metode može biti *public*, *private* i *protected*. Tipovi vidljivosti su ukratko opisani. Ako je metoda dostupna u svim klasama i podklasama, onda je njena vidljivost public . Vidljivost metode je protected , ako je metoda dostupna u svojoj klasi i u izvedenim klasama. Vidljivost private znači da je metoda dostupna samo u okviru klase u kojoj je deklarisana.

Tip povratne vrednosti metode može biti bilo koji od tipova podataka. Ovo znači da nakon primene metode ona vraća vrednost koja je tog definisanog tipa. Kasnije se u pozivanju metode vrednost koju metoda vrati može čuvati u promenljivoj koja je istog tipa kao i povratna vrednost. Napomenuto je i da neke metode nemaju povratnu vrednost, pa se tada kao tip povratne vrednosti koristi ključnu reč `void`.

Ime metode je jedinstveni identifikator koji se kasnije koristi kod pozivanja metode. Kod davanja imena metodi, C# razlikuje mala i velika slova, odnosno metode *metoda()* i *Metoda()* nisu iste.

U okviru liste parametara pišu se svi parametri metode: metoda ne mora imati parametre, može imati jedan ili više parametara. Kod pozivanja metode, metoda se primenjuje na realne parametre, koji se tada nazivaju argumentima. Argumente se moraju pisati istim redom kao i parametri u listi parametara kod definisanja metode. Broj argumenata i broj parametara takođe mora da se poklapa.

Najzad, u okviru tela metode pišu se naredbe koje treba izvršiti. Ovo znači da telo metode određuje šta će se to konkretno uraditi sa objektom nad kojim se metoda primenjuje.

7.1.1. Prenošenje parametara metodi

Prenošenje parametara metodi, kao što je opisano u [4], može biti:

1. *Prenošenje po vrednosti* – Kada se parametri prosleđuju po vrednosti, kopiraju se vrednosti argumenata na mesta parametara metode. Ključna stvar je da metoda ne može da promeni vrednost prosleđenih argumenta, već samo da nešto sa njima uradi, pa vrati vrednost.

2. *Prenošenje po referenci* – Kada se parametri prosleđuju po referenci, ne kopira se vrednost argumenta, već referenca na memorijsku lokaciju argumenta. Na ovaj način se obezbeđuje da metoda može da promeni vrednost argumenta.

3. *Izlazni parametri* – Ovaj način prenošenja koristi se kada je potrebno da metoda može da vrati više od jedne vrednosti.

U elektronskoj lekciji o metodama su prikazana prva dva načina za prenošenje parametara. Treći način za sada nije prikazan. On bi mogao da se prikaže ukoliko se elektronske lekcije prošire naprednijim sadržajima. Za početak učenicima je najvažnije da shvate prenošenje po vrednosti, dok je prenošenje po referenci dato iz razloga što je za definisanje metode koja razmenjuje vrednosti dva broja neophodno prenošenje po referenci.

7.1.2. Primeri metoda

Učenicima su predstavljena dva primera metoda. Prvi je primer metode koja ispituje da li je broj prost, dok je drugi primer metode koja vraća maksimum dva broja.

Metoda koja ispituje da li je broj prost definisana je na način koji se može videti u primeru 30.

Primer 30: Metoda Prost()

```
public bool Prost(int n)
{
    if (n == 1) return false; //metoda vraća false jer 1 nije prost broj
    if (n == 2) return true; //metoda vraća true jer je 2 prost broj
    for (int i = 2; i<n; i++) //nije potrebno proveravati deljivost sa 1 i n
    {
        if (n % i == 0) //provera da li je argument n deljiv tekućim brojačem i
        {
            return false;
        }
    }
    return true; //podrazumevana povratna vrednost
}
```

Petlja u okviru koje se ispituje da li je broj n deljiv nekim od brojeva koji su manji od njega može i da se prilagodi, pa da se deljivost ispituje do n/2. Međutim, za učenike je verovatno intuitivnije da ispitaju svaki broj do n, pa je na taj način i urađena metoda.
Druga metoda, koja vraća maksimum, urađena je i prikazana kao što se može videti u primeru 31.

Primer 31: Metoda Maks()

```
public double Maks(double a, double b)
{
    if (a > b)
    {
        return a;
    }
    else return b;
}
```

7.2. Pozivanje metode

Metoda koja ima povratnu vrednost se poziva kao u primeru 32.

Primer 32: Pozivanje metode koja vraća vrednost

```
tip vrednost = MojaMetoda(parametar_1, ... , parametar_n);
```

Metoda čija je povratna vrednost `void` poziva se na način prikazan u primeru 33.

Primer 33: Pozivanje metode čija je povratna vrednost void

```
MojaMetoda(parametar_1, ... , parametar_n);
```

Metode se mogu pozivati primenom operatora tačka (.), što se može videti u primeru 34.

Primer 34: Primena operatora tačka kod pozivanja metode

```
MojaKlasa.MojaMetoda();
```

U okviru elektronske lekcije o metodama učenicima se ilustruje pozivanje metoda koje su definisane u delu lekcije pod nazivom „Definisanje metode“ . Konkretna aplikacija u kojoj se koristi metoda *Prost()* je aplikacija preko koje korisnik unosi prirodan broj n preko kontrole *TextBox*, pa se klikom na dugme popunjavaju stavke kontrole *ComboBox* prostim brojevima do unetog broja n. Na ovaj način se učenici upoznaju i sa novom kontrolom i sa načinom pozivanja metode. Metoda *Prost()* je primenjena na način prikazan primerom 35.

Primer 35: Pozivanje metode Prost() u konkretnoj aplikaciji datoj u elektronskoj lekciji o metodama

```
if (Prost(i) == true)
{
    comboBox1.Items.Add(i);
}
```

Metoda je ovde pozvana direktno u izrazu bez inicijalizacije nove promenljive u koju bi se smestila povratna vrednost metode. Argument metode je promenljiva `i`, tipa `int`, koja je brojač u petlji od 1 do n.

Druga aplikacija koja koristi metodu *Maks()* je aplikacija preko koje korisnik unosi dva realna broja preko kontrola tipa *TextBox*, a klikom na dugme poziva se metoda i ispisuje maksimum dva uneta broja u labeli. Poziv metode *Maks()* dat je linijom koda prikazanom u primeru 36.

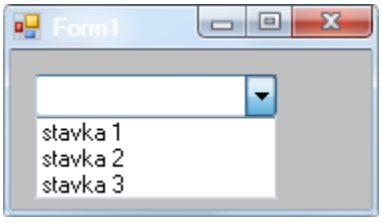
Primer 36: Pozivanje metode Maks() u konkretnoj aplikaciji datoj u elektronskoj lekciji o metodama

```
maksimum = Maks(a, b);
```

Ovde je `maksimum` promenljiva tipa `double`, a u nju se smešta povratna vrednost metode *Maks()* primenjena nad argumentima `a` i `b`, takođe tipa `double`.

7.2.1. ComboBox kontrola

U okviru ove lekcije prvi put se u zadacima pojavila i kontrola ***ComboBox*** (kombo boks), pa bi trebalo opisati tu kontrolu. *ComboBox* je kontrola preko koje se iz padajuće liste može izabrati neka od ponuđenih opcija, a takođe se u nju može uneti tekst. Izgled kontrole *ComboBox* može se videti na slici 23.



Slika 23: Kontrola ComboBox sa tri stavke

Za rešavanje zadatka u kome se pojavila ova kontrola se koristi svojstvo *Items* za pristup stavkama, pa se primenjuje metoda *Add()* za dodavanje stavke u padajuću listu. Bitna svojstva su i *SelectedItem*, svojstvo koje se koristi za dobijanje imena selektovane stavke kontrole *ComboBox*, kao i svojstvo *SelectedIndex* preko koga se dobija indeks stavke u kolekciji stavki.

7.2.3. Primer metode koja ne vraća vrednost

Metoda koja ne vraća vrednost ima povratni tip `void`. U okviru ove elektronske lekcije je dat primer metode koja vrši razmenu vrednosti dve promenljive. Ako su promenljive `a = 5` i `b = 4`, nakon primene metode će vrednost promenljive `a` biti 4, a vrednost promenljive `b` biti 5. Kako takva metoda menja vrednosti argumenata, ovaj primer ilustruje i prenošenje parametara metodi po referenci. Metoda koja vrši razmenu mesta promenljivih data je programskim kodom prikazanim u primeru 37.

Primer 37: Metoda Razmeni() koja vrši razmenu mesta promenljivih

```
public void Razmeni(ref double a, ref double b)
{
    double privremena;
    privremena = a; //u promenljivu privremena smešta se vrednost promenljive a
    a = b; //u promenljivu a smešta se vrednost promenljive b
    b = privremena; //b preuzima vrednost koja je smeštena u promenljivoj privremena,
    //što je prvobitna vrednost promenljive a
}
```

Kod ovog primera može se primetiti da je kod liste parametara iskorišćena ključna reč `ref` ispred deklaracije parametra. Ključna reč `ref` se koristi za prosleđivanje parametra po referenci. Zato se metoda poziva u glavnom delu programa na način prikazan u primeru 38.

Primer 38: Pozivanje metode Razmeni()

```
Razmeni(ref a, ref b);
```

Za testiranje metode, kreira se aplikacija preko koje korisnik unosi realne brojeve a i b i pritiskom na dugme vrši se primena metode i upisivanje redosleda brojeva a i b u dve kontrole tipa *TextBox*: u prvu se upisuje redosledom kojim je korisnik uneo brojeve, a u drugu redosledom nakon izvršene razmene.

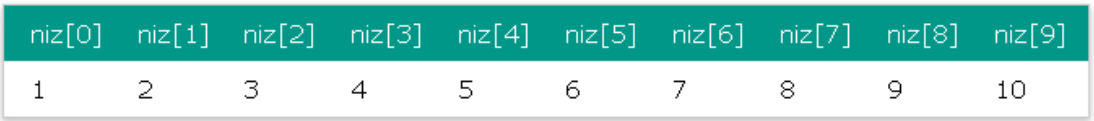
U ovom primeru se učenici upoznaju sa svojstvom *Enabled* koje se primenjuje na kontrolu *Button*. Ovo svojstvo služi da dugme bude omogućeno ili onemogućeno za korišćenje. Kada je svojstvo *Enabled* postavljeno na `true`, dugme je omogućeno (može se kliknuti na njega). Ovo je i podrazumevana vrednost svojstva. Da bi se dugme onemogućilo, svojstvo *Enabled* se postavi na `false`. U aplikaciji u kojoj je prvi put prikazano svojstvo *Enabled*, ovo je urađeno da nakon izvršenih naredbi korisnik ne bi mogao ponovo da klikne na dugme. Primenjuje se i događaj *TextChanged* na kontrole tipa *TextBox*, kako bi se svojstvo *Enabled* ponovo postavilo na `true` kada korisnik izbriše tekst iz njih.

8. Nizovi

Često se javlja potreba da je za rešavanje nekog problema podatke istog tipa potrebno objediniti, kako bi se sa njima jednostavnije radilo. Podaci istog tipa se mogu smestiti u strukturu koja se naziva niz. Nizovi dakle predstavljaju uređeni skup objekata istog tipa. U programskom jeziku C#, nizovi su izvedeni iz klase Arrays, pa iz nje nasleđuju svojstva i metode.
Ideja elektronske lekcije o nizovima jeste da se učenici upoznaju sa konceptom nizova kroz najjednostavnije primere nizova, pre svega kroz rad sa jednodimenzionim nizovima. U okviru ove elektronske lekcije su prikazana i dva osnovna algoritma za rad sa nizovima, algoritam linearne pretrage i algoritam za sortiranje Selection Sort. Početna strana elektronske lekcije o nizovima prikazana je na slici 24.

Slika 24: Elektronska lekcija - Nizovi

Na početku lekcije data je definicija niza, a nakon definicije niza, sledi primer niza prvih 10 prirodnih brojeva, kako bi učenici intuitivno razumeli šta je uopšte niz. Svaki član niza ima svoj *indeks*, odnosno mesto na kome se nalazi u nizu. Najvažnija stvar je da indeksiranje kreće od nule, a ne od 1. Ovo znači da je prvi član niza u stvari nulti član, drugi član je u stvari prvi član itd. Prikaz niza brojeva od 1 do 10 u okviru elektronske lekcije o nizovima može se videti na slici 25.



Slika 25: Niz prirodnih brojeva od 1 do 10 sa indeksima

8.1. Deklarisanje i inicijalizovanje niza

Niz se deklariše na način prikazan u primeru 39.

Primer 39: Deklarisanje niza u opštem slučaju

```
tip_podatka[] Niz;
```

Dakle, prvo se navodi kog tipa su podaci u nizu. Dalje slede uglaste zagrade, kojima se označava dimenzija niza, a u navedenom opštem obliku deklarisanja radi se od jednodimenzionom nizu. Najzad daje se ime niza (u ovom slučaju `Niz`).

Za dodavanje elemenata u niz, mora se inicijalizovati niz. Nizovi su referentni tip podataka, pa se za kreiranje nove instance klase Arrays (odnosno objekta nizovnog tipa) koristi ključna reč `new`. U elektronskoj lekciji o nizovima je predstavljen prikaz kreiranja različitih nizova,što se može videti u primeru 40.

Primer 40: Kreiranje nizova korišćenjem ključne reči new

```
int[] niz_celih = new int[10];
double[] niz_realnih = new double[100];
char[] niz_karaktera = new char[1000];
```

Nakon inicijalizacije prvog niza koji se zove `niz_celih`, kreira se niz u memoriji koji ima 10 elemenata tipa `int` i svih 10 elemenata uzimaju podrazumevanu vrednost za tip `int`, odnosno vrednost 0. Kod drugog niza, nakon inicijalizacije u memoriji se kreira niz od 100 elemenata tipa `double` i svi uzimaju vrednost 0 tipa `double`, tj. podrazumevanu vrednost. Isto važi i za niz karaktera – kreira se niz od 1000 karaktera i svaki element jednak je podrazumevanoj vrednosti tipa `char` ('\0'). Dakle, inicijalizacijom se kreira niz u memoriji kod koga je vrednost svakog elementa podrazumevana vrednost za određeni tip podatka.

8.2. Dodeljivanje vrednosti elementima niza

Elementima niza mogu se dodeliti vrednosti na različite načine. U ovoj elektronskoj lekciji je prikazano nekoliko načina, što se može videti u primerima, od primera 41. do primera 46.

Primer 41: Dodeljivanje vrednosti pojedinačnom elementu niza korišćenjem indeksa odgovarajućeg elementa:

```
double[] niz = new double [5];
niz[0] = 4.0;
niz[1] = 6.0;
niz[2] = 7.5;
niz[3] = 2.3;
niz[4] = 1.3;
```

Primer 42: Dodeljivanje vrednosti u trenutku deklaracije

```
double[] niz = { 4.0, 6.0, 7.5, 2.3, 1.3 };
```

Primer 43: Kreiranje i inicijalizovanje

```
double [] niz = new double[5] { 4.0, 6.0, 7.5, 2.3, 1.3};
```

Primer 44: Kreiranje i inicijalizovanje bez naglašavanja koliki je broj elemenata niza

```
double [] niz = new double[] { 4.0, 6.0, 7.5, 2.3, 1.3};
```

Primer 45: Kopiranje jednog niza u drugi

```
double [] niz = new double[] { 4.0, 6.0, 7.5, 2.3, 1.3};
double [] novi_niz = niz;
```

Primer 46: Preko petlje

```
int[] niz = new int[10];
for (int i = 0; i<10; i++)
{
    niz[i] = i+1;
}
```

8.3. Pristupanje elementima niza

Pristupanje konkretnom elementu nekog niza vrši se pomoću indeksa. Primera radi, ako se napiše niz[0] pristupa se prvom članu niza, ako se napiše niz[9], pristupa se desetom članu itd.
Učenicima je kroz nekoliko zadataka prikazana upotreba nizova, počev od toga kako se inicijalizuje niz, do toga kako se pristupa njegovim članovima sa kojima treba nešto uraditi.

8.3.1. Popunjavanje i ispis elemenata niza

Prvi zadatak koji je urađen u okviru elektronske lekcije o nizovima prikazuje kako se niz popunjava vrednostima preko petlje, a potom kako se u kontroli *RichTextBox* ispisuju članovi ispunjenog niza. Postupna izrada zadatka je prikazana i preko video materijala. Niz koji se kreira u zadatku je niz kvadrata brojeva od 1 do 100. Niz se najpre inicijalizuje na način prikazan primerom 47.

Primer 47: Kreiranje niza od 100 celobrojnih vrednosti

```
int [] niz = new int[100];
```

Potom se niz popunjava elementima preko `for` petlje, pa se elementi niza ispisuju u *RichTextBox*. Deo programskog koda koji ovo radi prikazan je u primeru 48.

Primer 48: Popunjavanje niza preko for petlje i ispis u niza u kontrolu RichTextBox

```
for (int i = 0; i<100; i++)
{
    niz[i] = (i+1)*(i+1); //na nultom mestu je 1*1, na prvom 2*2, itd.
}

for (int i=0; i < 100; i++)
{
    richTextBox1.Text += niz[i] + "\n";
}
```

U ovom zadatku prikazano je i pristupanje konkretnom članu niza. U *TextBox* treba upisati 50.član ovog niza (odnosno kvadrat broja 50). Kako se 50.član nalazi na 49.mestu u nizu, jer indeksiranje kreće od 0, ispis izgleda kao u primeru 49.

Primer 49: Ispis pedesetog člana niza u kontrolu TextBox

```
textBox1.Text = niz[49].ToString();
```

8.3.2. Minimalni i maksimalni element niza

Pronalaženje minimalnog ili maksimalnog elementa niza jedan je od algoritama za rad sa nizovima koje bi učenici trebalo da savladaju. Prikaz nalaženja minimuma i maksimuma niza predstavljen je zadatkom. Zadatak se svodi na kreiranje aplikacije preko koje se unose brojevi bodova za 8 takmičara, a zatim se ispisuje koji takmičar ima najviše bodova, a koji najmanje. Podaci o bodovima smeštaju se u celobrojni niz od 8 elemenata.

Iako je ovaj zadatak bilo moguće raditi preko `for` petlje, nalaženje minimuma i maksimum urađeno je preko petlje `foreach`, koja je pogodna za rad sa nizovima. Programski kod koji predstavlja algoritam nalaženja minimuma i maksimuma niza je dat u primeru 50.

Primer 50: Nalaženje minimalnog i maksimalnog elementa niza

```
min = niz[0]; //minimum i maksimum se postave na prvi član niza
maks = niz[0];
foreach (int element in niz) //prolaz kroz elemente niza
{
    if (element < min) //ispituje da li je tekući element manji od minimuma
    {
        min = element; //novi minimum
    }
    if (element > maks) //ispituje da li je tekući element veći od maksimuma
    {
         maks = element; //novi maksimum
    }
}
```

Dakle, promenljive `min` i `maks` se inicijalizuju na nulti član niza i prolazi se petljom kroz niz. Za nalaženje minimuma traži se član koji je manji od postavljenog minimuma, a za nalaženje maksimum član koji je veći od postavljenog maksimuma.

8.3.3. Suma elemenata niza i prosečna vrednost

Kada su članovi niza numeričkog tipa, može da se vrši njihovo sumiranje. Ovo se radi preko petlje koja prolazi kroz niz i na vrednost sume inicijalno postavljene na 0, dodaje vrednost tekućeg člana niza u petlji. Prosečna vrednost se dobija kada se izračunata suma podeli brojem elemenata niza.

Ovo je prikazano preko aplikacije koja simulira đačku knjižicu: unose se ocene po predmetima i računa se prosek ocena, pa se na osnovu proseka ispisuje opšti uspeh učenika. Kroz taj zadatak je prikazano i kreiranje niza čiji su članovi tipa *TextBox* kako bi se lakše radilo sa njima i kako bi učenici videli još neke vrste nizova osim nizova čiji su članovi brojevi.

8.4. Pretraga niza

Pretraga niza, zajedno sa sortiranjem je jedan od najvažnijih algoritama za rad sa nizovima uopšte. Pod pretragom se podrazumeva ispitivanje da li u nizu postoji određeni element. Učenicima je objašnjena najjednostavnija pretraga niza, a to je linearna pretraga.

Najpre je objašnjena metoda koja implementira linearnu pretragu niza. Metoda kao argumente prima niz i element koji se traži u nizu. Petljom se prolazi kroz niz i upoređuje se trenutni član niza sa traženim elementom. Ako se u nizu pronađe traženi element, metoda vraća indeks tog elementa u nizu. U suprotnom, ako se ne pronađe element, metoda vraća -1. U okviru ove elektronske lekcije, metoda koja implementira linearnu pretragu može da se pogleda klikom na dugme preko koga se otvara kod sa komentarima, što je prikazano na slici 26.

Pogledaj kod za metodu Linearna_Pretraga

```
//definišemo metodu Linearna_Pretraga
public int Linearna_Pretraga(int[] niz_celih, int x)
        {
            int i; //lokalna promenljiva i
            for (i=0; i < niz_celih.Length; i++) //prolazimo kroz n
iz for petljom koja ide od 0 do dužine niza
            {
                                //pitamo  da  li  je  element  na  i-
tom mestu u nizu jednak traženom elementu x
                if (niz_celih[i] == x)
                {
                    return i; //ako je uslov zadovoljen vraćamo ind
eks traženog elementa
                }

            }
            return -1; //inače vraćamo -1 ukoliko prolaskom kroz pe
tlju nismo našli traženi element
        }
```

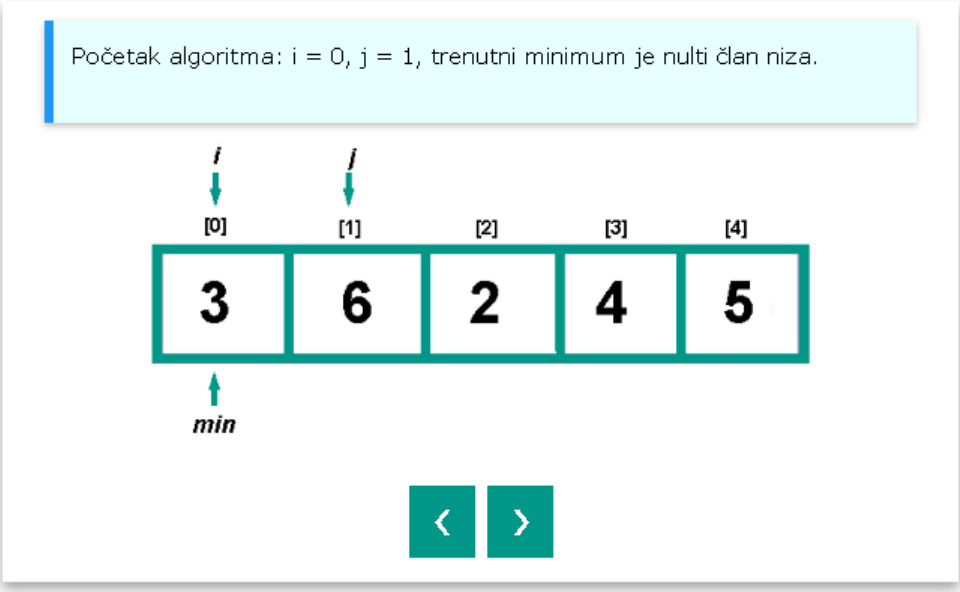
Slika 26: Metoda za linearnu pretragu niza

Nakon prikazane metode za linearnu pretragu, učenicima se postavlja zadatak koji testira metodu. U zadatku se zahteva kreiranje aplikacije koja prikazuje unapred definisan niz preko kontrole *RichTextBox* i omogućava korisniku da preko kontrole *TextBox* unese element za koji ispituje da li se nalazi ili ne nalazi u nizu. Klikom na dugme za obradu podataka primenjuje se metoda i korisniku se ispisuje da li određeni element postoji u nizu ili ne postoji.

8.5. Sortiranje niza

Pod sortiranjem niza podrazumeva se uređivanje niza u rastućem ili opadajućem poretku. Poredak je rastući ako su elementi niza poređani od najmanjeg do najvećeg, a opadajući ako je obrnuto. Iako postoje brojni algoritmi za sortiranje, u okviru elektronskih lekcija je prikazan samo algoritam ***Selection sort*** (sortiranje selekcijom). Ovaj algoritam izabran je zbog svoje jednostavnosti za razumevanje od strane učenika koji tek počinju da uče programiranje.

Učenicima je najpre prikazana animacija (slide show) koja prikazuje korake sortiranja niza {3,6,2,4,5}. Izabran je niz od 5 članova kako bi se pojednostavila animacija i imala manje koraka, a ipak pokazala suštinu algoritma. Na slici 27. prikazan je početak animacije. Dugmad za navigaciju omogućavaju kretanje kroz animaciju, tako da učenik u svakom trenutku može da pogleda sledeći korak ili da se vrati na prethodni korak.



Slika 27: Početak animacije koja prikazuje sortiranje niza

Nakon animacije, učenici mogu da pogledaju programski kod za metodu *Selection_Sort()*. Prikazano je sortiranje u rastućem poretku. Metoda kao argument prima niz koji treba sortirati, a vraća sortirani niz. Ključne promenljive koje se koriste u okviru metode su:

```
int i, j, pozicija_minimuma, pomocna;
```

Promenljive `i`, `j` su brojači koji se koriste u dvostrukoj `for` petlji, `pozicija_minimuma` je promenljiva u kojoj se čuva indeks tekućeg minimalnog elementa, dok promenljiva `pomocna` služi za zamenu mesta dva elementa kada se pronađe manji element od tekućeg minimuma. Potom se inicijalizuju vrednosti promenljivih:

```
i = 0;
pozicija_minimuma = 0;
```

Koristi se dvostruka `for` petlja. Brojač spoljašnje `for` petlje `i` kreće od nule, a brojač unutrašnje `for` petlje `j` kreće od `i + 1`. Dakle, kreće se od nultog člana niza, pa se unutrašnjom `for` petljom prolazi kroz ostatak niza i upoređuje se element na tekućoj poziciji `j` sa trenutnim minimumom. Ako je taj `j` -ti element manji od minimuma, novi minimum se onda nalazi na poziciji `j`, pa se u promenljivu `pozicija_minimuma` sačuva `j` -ta pozicija. To je urađeno na način prikazan u primeru 51.

Primer 51: Unutrašnja for petlja algoritma selection sort

```
for (j = i+1; j < niz.Length; j++)
{
    if (niz[j] < niz[pozicija_minimuma])
    {
        pozicija_minimuma = j;
    }
 }
```

Kada se sačuva pozicija minimuma, postavlja se pitanje da li je ona različita od pozicije `i`. Ako su iste pozicije, ne vrši se nikakva razmena, međutim ako su različite, vrši se razmena elemenata niza sa pozicije `i` i pozicije minimuma. Razmena elemenata se vrši algoritmom za razmenu (već učenicima poznatog iz elektronske lekcije o metodama), uz pomoć promenljive `pomocna`. Nakon završene razmene (ili bez razmene, ako nije bila potreba za njom) nastavlja se spoljašnja `for` petlja i ponavlja se ovaj postupak za svako `i` dok se ne dođe do kraja niza. Nakon završetka niz je sortiran.

U okviru ove lekcije je urađen i zadatak koji ilustruje testiranje metode upravo na nizu {3,6,2,4,5} koji je prikazan animacijom. Takođe, napomenuto je da je sortiranje u opadajućem poretku slično kao i sortiranje u rastućem, jedino što se u metodi *Selection_Sort()* razlikuje uslov za upoređivanje: traži se element koji je veći od tekućeg maksimuma i premešta se na početak niza, itd.

9. Niske

Niske ili stringovi su objekti tipa String i predstavljaju tekstualni tip podataka. Ovaj tip podatka je jedan od najčešće korišćenih tipova i zato mu se posvećuje posebna pažnja. U programskom jeziku C# svi tipovi podataka su objekti odgovarajućih klasa. String je u suštini objekat koji se kreira na osnovu konstruktora iz klase String. U toj klasi se nalaze metode za rad sa stringovima, konstruktori, svojstva. String se može shvatiti i kao kolekcija karaktera, među kojima osim slovnih karaktera, mogu da se nađu i numerički, specijalni znakovi itd.

Elektronska lekcija o niskama prikazana je na slici 28.

Slika 28: Elektronska lekcija – Stringovi (Niske)

9.1. Objekat tipa niske

Ključna reč koja se koristi za deklarisanje objekta tipa niske je `string` (ili `System.String`). Podrazumevana vrednost je vrednost `null`. Ako se deklariše objekat tipa niske, a ne inicijalizuje se, onda on ima vrednost `null` i tada se ništa sa njim ne može raditi, a prilikom pokretanja programa pojavljuje se greška. Treba razlikovati nisku koja je `null` i praznu nisku. Sa praznom niskom se može raditi i ona je u suštini niska koja ne sadrži nijedan karakter. Prazna niska se inicijalizuje ukoliko se promenljivoj dodeli vrednost String.Empty ili ako se vrednost promenljive postavi na "" (između navodnika se ne napiše nijedan karakter). Dužina prazne niske je 0.

9.1.1. Deklarisanje i inicijalizovanje niske

Deklarisanje niske izvršava se tako što se nakon ključne reči `string` navede ime promenljive što se može videti u primeru 52.

Primer 52: Deklarisanje niske

```
string niska;
```

Nakon deklaracije, da bi objekat tipa niske mogao da se koristi, mora mu se dodeliti neka vrednost. Postoji više načina da se objektu tipa niska dodeli vrednost. Ovo je prikazano primerima, od primera 53. do primera 56., koji se mogu videti u okviru elektronske lekcije o niskama.

Primer 53: Dodeljivanje teksta između znaka navoda

```
String niska;
niska = "Ovo je promenljiva tipa string!";
//Umesto ključne reči string, kao što je već rečeno može se koristiti i
//System.String
System.String niska = "Ovo je promenljiva tipa string!";
```

Primer 54: Korišćenje konstruktora klase String koji kreira nisku od niza karaktera:

```
char[] karakteri = {'Z', 'D', 'R', 'A', 'V','O'};
string rec = new string(karakteri); //Niska rec ima vrednost "ZDRAVO ".
```

Primer 55: Dobijanje niske spajanjem niski primenom operatora + za konkatenaciju (nadovezivanje) niski

```
string rec1 = "Zdravo ";
string rec2 = "svete!";
string recenica = rec1 + rec2; //Niska recenica ima vrednost "Zdravo svete! ".
```

Primer 56: Dobijanje nove niske primenom metode ili svojstva koje vraća string

```
string rec1 = "Zdravo ";
string rec2 = "svete!";
string recenica = String.Concat(rec1,rec2);
```

U primeru 56. je iskorišćena metoda *Concat()* iz klase String da nadoveže dva stringa. Nakon primene metode, niska `recenica` ima vrednost "Zdravo svete! ".

9.1.2. Svojstvo Length

Jedno od najvažnijih svojstava objekata tipa niske je svojstvo *Length*. Njegovom primenom na objekat tipa niske dobija se dužina niske, odnosno broj njenih karaktera. Primena svojstva *Length* prikazana je primerom 57.

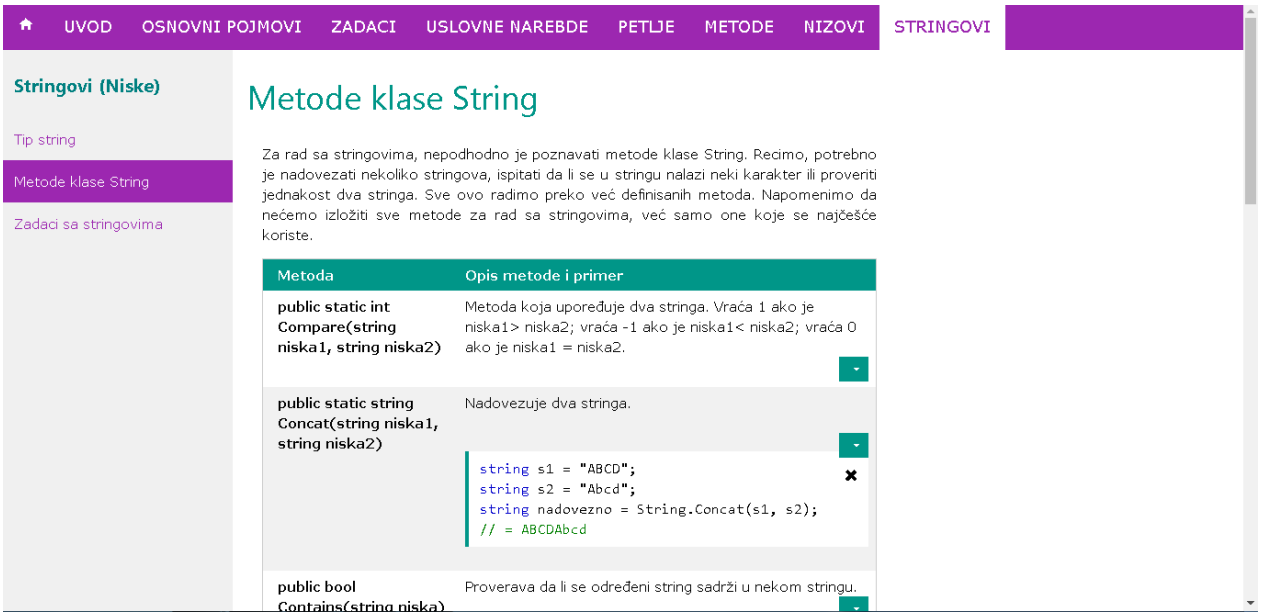
Primer 57: Svojstvo Length koje vraća dužinu niske

```
string recenica = "Zdravo svete!"
int n = recenica.Length;
```

Vrednost promenljive `n` u ovom slučaju je 13, odnosno broj karaktera u rečenici "Zdravo svete!".

9.2. Metode klase String

Metode koje su već definisane u klasi String omogućavaju jednostavniji rad sa niskama. U okviru elektronske lekcije o niskama je napomenuto da nisu prikazane sve metode, već samo one koje bi učenici koristili u zadacima. Metode su prikazane u tabeli, čiji deo se može videti na slici 29. Klikom na strelicu pored opisa metode, dobija se primer kako se metoda koristi.



Slika 29: Deo tabele iz elektronske lekcije o niskama koja prikazuje metode klase String

9.2.1. Spisak metoda klase String

Za izradu zadataka sa niskama učenicima je, kao što je već napomenuto, dat pregled metoda klase String u tabeli. Ovakav pregled metoda može biti koristan za učenika koji želi da samostalno reši zadatke postavljene u elektronskoj lekciji o niskama. Takođe, učenik uvek može da se vrati na metodu koju ne razume najbolje i da ponovo pročita sve o načinu korišćenja te metode, kakva je njena povratna vrednost, koje argumente prima metoda itd. Sve metode koje prikazane u ovoj elektronskoj lekciji mogu se videti u tabeli 2.

Tabela 2: Metode klase String

Metoda	**Potpis metode**	**Opis metode**	**Primer prikazan u okviru elektronske lekcije o niskama**

Metoda *Compare()*	**public static int Compare(string niska1, string niska2)**	Upoređuje dve niske	`String s1 = "ABCD";`
			`string s2 = "Abcd";`
			`string s3 = "ABCD";`
			`int rezultat_s1s2 = String.Compare(s1, s2);`
			`// = 1, jer ABCD > Abcd`
			`int rezultat_s1s3 = String.Compare(s1, s3);`
			`// = 0, jer ABCD = ABCD`
			`int rezultat_s2s3 = String.Compare(s2, s3);`
			`// = -1, jer Abcd < ABCD`

Metoda *Concat()*	**public static string Concat(string niska1, string niska2)**	Nadovezuje dve niske	`string s1 = "ABCD";`
			`string s2 = "Abcd";`
			`string nadovezno = String.Concat(s1, s2);`
			`// = ABCDAbcd`

Metoda *Contains()*	**public bool Contains(string niska)**	Ispituje da li se jedna niska nalazi u okviru druge niske	`string s1 = "ABCD";`
			`bool da_li_sadrzi_BC = s1.Contains("BC");//true`
			`bool da_li_sadrzi_MN = s1.Contains("MN");//false`

Metoda *Copy()*	**public static string Copy(string niska)**	Kopira nisku. Kada se primeni, kreira se novi string objekat koji ima istu vrednost kao niska nad kojom je primenjena metoda.	`string s1 = "ABCD";` `string novi = String.Copy(s1); //=ABCD`

Metoda *EndsWith()*	**public bool EndsWith(string niska)**	Proverava da li se određena niska završava string objektom „niska“, koji je ovde parametar metode.	`string s1 = "ABCD";`
			`bool zavrsetak_CD = s1.EndsWith("CD");//true`
			`bool zavrsetak_MN = s1.EndsWith("MN");//false`

Metoda *Equals()*	**public bool Equals(string niska)**	Upoređuje dve niske, odnosno ispituje njihovu jednakost.	`string s1 = "ABCD";`
			`bool jednak_1 = s1.Equals("ABCD"); //true`
			`bool jednak_2 = s1.Equals("abcd"); //false`

Metoda *Equals()*	**public bool Equals(string niska1,string niska2)**	Druga „verzija“ metode *Equals()* koja u ovom slučaju prima dva argumenta, dve niske i ispituje da li ove niske imaju jednaku vrednost.	`string s1 = "ABCD";`
			`string s2 = "Abcd";`
			`string s3 = "ABCD";`
			`bool jednaki_s1s2 = String.Equals(s1, s2);`
			`//false`
			`bool jednaki_s1s3 = String.Equals(s1, s3);`
			`//true`

Metoda *IndexOf()*	**public int IndexOf(char karakter)**	Vraća indeks prvog pojavljivanja karaktera u određenoj niski. Indeksiranje ide od 0.	`string s1 = "ABCD";`
			`int indeks_A= s1.IndexOf('A'); //=0`
			`int indeks_B = s1.IndexOf('B'); //=1`
			`int indeks_C = s1.IndexOf('C'); //=2`
			`int indeks_D = s1.IndexOf('D'); //=3`

Metoda *Replace()*	**public string Replace(string stari, string novi)**	Služi da se u niski nad kojom se primeni svako pojavljivanje niske „stari“ zameni niskom „novi“. Vraća string.	`string s1 = "ABCD";` `string s2 = s1.Replace("AB","ab");//="abCD"`

Metoda *StartsWith()*	**public bool StartsWith**	Ispituje da li određena niska počinje string	`string s1 = "ABCD";`
			`bool pocetak_A = s1.StartsWith("A"); //true`
			`bool pocetak_D = s1.StartsWith("D"); //false`

	(string niska)	objektom „ niska“.	

Metoda *Substring ()*	**public string Substring(int početniIndeks,int dužina)**	Vraća podstring string objekta počevši od određenog indeksa do zadate dužine.	`string s = "Zdravo svete!";`
			`string podstring = s.Substring(7, 6);`
			`//="svete!"`
			`//počinje od 7.karaktera i ima dužinu 6`

Metoda *ToCharArray()*	**public char[] ToCharArray(string niska)**	Vraća niz svih karaktera jedne niske.	`string s1 = "ABCD";` `char[] niz_karaktera = s1.ToCharArray();//={'A','B','C','D'}`
Metoda *ToLower()*	**public string ToLower()**	Vraća kopiju niske čija su sva slova prebačena u mala slova.	`string s = "Zdravo svete!";` `string mala = s.ToLower();//="zdravo svete!"`
Metoda *ToUpper()*	**public string ToUpper()**	Vraća kopiju niske čija su sva slova prebačena u velika slova.	`string s = "Zdravo svete!";` `string velika = s.ToUpper();//="ZDRAVO SVETE!"`
Metoda *Trim()*	**public string Trim()**	Uklanja beline u određenoj niski.	`string niska = "   Zdravo";` `string bez_belina = niska.Trim();//="Zdravo"`

Pored ovih metoda prikazanih u tabeli 2., u klasi String su definisane i druge metode, na šta su učenici upućeni u napomeni u kojoj je dat link ka definiciji ostalih metoda.

9.2.2. Primena metoda nad niskama u zadacima

U okviru celine „Zadaci sa stringovima“, učenicima je dato 5 zadataka u kojima mogu primeniti izloženu teoriju o niskama, kao i da uokvire znanje stečeno kroz prethodne lekcije. Zadaci su osmišljeni tako da predstavljaju mini aplikacije tematski bliske učenicima (provera da li tekst sadrži neki string, aplikacija koja obrađuje unete podatke o korisniku, aplikacija koja daje ideju za kreiranje kviza), ili su pak standardni zadaci kod obrade niski (palindromi, anagrami).

Svi zadaci su rešeni. Učenici mogu pogledati programske kodove i mogu ih preuzeti. Zadaci ovde nisu predstavljeni preko video materijala, kako bi učenici najpre krenuli samostalno da ih rešavaju pre nego što pogledaju njihovo rešenje.

10. Zaključak

Programski jezik C# može biti dobra osnova za učenike koji su početnici u programiranju, a žele da savladaju osnovne koncepte programiranja i da razviju način mišljenja pri programiranju, s obzirom na to da programski jezik C# sadrži sve što je potrebno da tu osnovu pruži. Ovaj programski jezik može biti pogodan za učenike i iz razloga što, uz pomoć tehnologije .NET, predstavlja jako dobar programski jezik za izradu aplikacija za Windows, a učenici su u današnje vreme odrasli koristeći takve aplikacije. Pretpostavka da je konačan proizvod koji učenici naprave korisna aplikacija za Windows pruža mogućnost da se učenici više zainteresuju za programiranje uopšte, kao i za programski jezik C#.

Edukativni sadržaji predstavljeni učenicima su birani tako da pokriju osnove programiranja i pruže mogućnost za dalje usavršavanje. Ipak, najvažniji je način predstavljanja tih sadržaja, a to je način koji se zasniva na samostalnom učenju uz pomoć elektronskih sadržaja. Samostalno učenje je jedan od najboljih načina učenja i ostavlja učenicima prostor za napredovanje po sopstvenim nahođenjima. Kako je sadržaj interaktivan, veća je verovatnoća da će privući učenike i zainteresovati ih. Mana samostalnog učenja uz korišćenje elektronskih sadržaja je ta što ne postoji mogućnost postavljanja dodatnih pitanja u vezi eventualnih nejasnoća, drugim rečima, nema interakcije sa profesorom. Ova mana se može prevazići ukoliko se elektronske lekcije prikazane u ovom master radu koriste za vreme časova predmeta računarstvo i informatika.

Trebalo bi istaći da se sadržaj elektronskih lekcija opisanih u master radu može dopuniti naprednijim konceptima programskog jezika C#, ukoliko bude bilo potrebe, odnosno ako se postigne cilj rada. Sadržaj se uvek može dopuniti i proširiti, što je prednost prikazivanja sadržaja u elektronskom obliku. Pored proširivanja sadržaja, bilo bi korisno poboljšati video materijale dodavanjem zvuka na video snimak, a to može biti od koristi učenicima koji lakše uče auditivnim, nego vizuelnim putem.

Literatura

[1] Nakov, S., Kolev, V., Dimitrov, D.,. (2013.). FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#. Sofia, BASD

[2] C Sharp (programming language): https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Pristupljeno avgusta 2016.

[3] Microsoft API and reference catalog: https://msdn.microsoft.com/en-us/library/

Pristupljeno jula 2016.

[4] Tutorials point - C# Tutorial: http://www.tutorialspoint.com/csharp/

Pristupljeno jula 2016.