Osnove objektno orjentiranog programiranja u C++-u

Flisar, Stjepan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski stručni studij elektrotehnike

OSNOVE OBJEKTNOG ORIJENTIRANOG POGRAMIRANJA U C++-u

Završni rad

Stjepan Flisar

Osijek, 2016.

SADRŽAJ

1. UVOD

Cilj ovog završnog rada je objasniti ključne koncepte objektno orijentiranog programiranja (OOP), a to su enkapsulacija, polimorfizam i nasljeđivanje. Takav pristup programiranju donosi znatna poboljšanja za razliku od starijih pristupa, kako bi programeri mogli držati korak sa zahtjevnim potrebama tržišta. Osim teorijski, koncepti će biti prikazani slikama primjera programskog kôda s kojima će se pobliže prikazati njihovo definiranje i funkcioniranje. U drugom poglavlju biti će opisani glavni elementi objektno orijentiranog programiranja, te njihovo međusobno povezivanje u smislenu cjelinu što nazivamo enkapsulacija. Treće poglavlje objašnjava koncept polimorfizma, osobina koja dozvoljava korištenje metoda istih naziva, a drugačijih oblika. Četvrto poglavlje opisuje koncept nasljeđivanja, upotrebljivost postojećeg kôda za definiranje novih i proširenih klasa. U petom poglavlju bit će prikazan jednostavna primjena OOP-a na programu vođenja nogometnog kluba.

2. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Objektno orijentirano programiranje (OOP) predstavlja pokušaj da se programi približe ljudskom načinu razmišljanja. Evolucija OOP kroz povijest se dogodila preko starijih pristupa programiranja, a to su proceduralno i modularno programiranje.

Proceduralni pristup programiranju zasniva se na promatranju programa kao niza funkcija, procedura, koje međusobno sudjeluju u rješavanju zadataka. Svaka procedura je konstruirana tako da obavlja jedan manji zadatak. Poziv procedure skače na mjesto u programu gdje je ona definirana te nakon što bude izvršena, program se nastavlja izvršavati od mjesta odakle je procedura pozvana. Svaku proceduru prije pozivanja moguće je zasebno testirati da se odmah vide rezultati te ako postoji, pronađe greška. Glavni program je odgovoran da proslijedi podatke pojedinim pozivima procedure.

Modularan pristup programiranju omogućava grupiranje procedura zajedničkih funkcionalnosti u odvojene module. Program je stoga podijeljen u nekoliko manjih dijelova koji su povezani preko poziva procedura. Svaki modul može sadržavati vlastite varijable što mu omogućava upravljanje unutrašnjim stanjima. Podaci su odvojeni od operacija nad njima, pa je modularno programiranje bazirano na operacijama umjesto nad podacima. Za razliku od modularnog, kod OOP moduli grupiraju podatke i operacije nad njima te ih se predstavlja kao ključne elemente u razvoju programa, a nazivamo ih objektima[1]. Svaki objekt ima svoje ponašanje, zadržava informaciju i može komunicirati s ostalim objektima.

Ključni koncepti objektno orijentiranog programskog jezika:

- baziranost na objektima i klasama
- podržava nasljeđivanje
- podržava polimorfizam

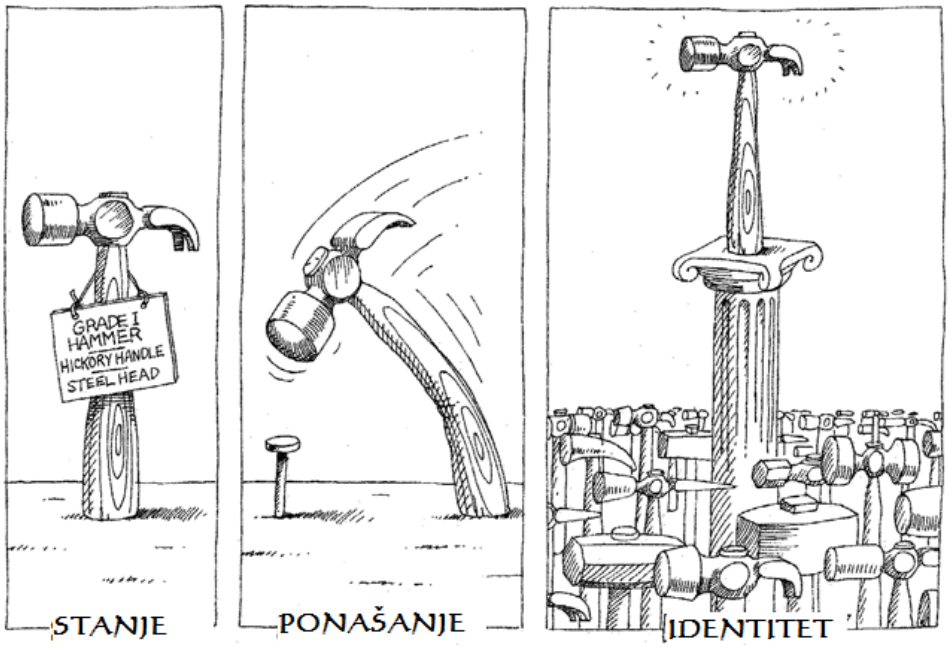
Neke od značajki OOP-a su:

- više se daje naglasak na podatak nego postupak
- programi su podijeljeni u cjeline koje grupiraju podatke i operacije, poznati kao objekti
- struktura podataka je dizajnirana tako da okarakterizira objekt
- funkcije koje djeluju nad podacima usko su povezane u strukturi podataka
- podaci su skriveni i ne može im se pristupiti vanjskim funkcijama
- novi podaci i funkcije se mogu jednostavno dodati kad god postoji potreba za tim

- prati *bottom-up* pristup u dizajnu programa, gdje se glavi program izvodi nakon deklaracije baznih elemenata

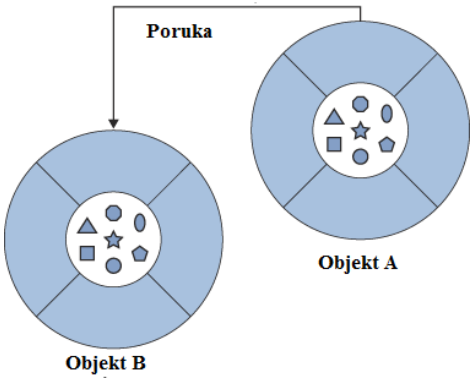
2.1. Objekt i klasa

U OOP-u kreirani objekti se modeliraju prema uzoru stvari iz stvarnog svijeta, imaju svoje stanje i ponašanje (Sl.2.1.). Objekt održava svoje stanje u jednoj ili više varijabli. Varijable su informacije nazvane po jedinstvenom identifikatoru. Varijable provode svoje ponašanje pomoću metoda. Metode su funkcije koje su povezane sa objektom, pa se stoga objekt može definirati kao skup varijabli i srodnih metoda. Objekt je također poznat kao instanca određene klase što varijable objekta čini i varijablama instance. Program koji se izvršava može sadržavati više objekata neke klase, npr. možemo imati više objekata klase *Student*. Svaki od tih objekata sadržavati će vlastite varijable i svaki objekt može imati različite vrijednosti spremljene u vlastitim varijablama. Pri izvršavanju programa, objekti međusobno komuniciraju, slanjem poruka jednim drugima bez poznavanja detalja o međusobnim podacima. Dovoljno im je znati samo vrstu primljene poruke, te vrstu povratnog odgovora objekta.



Slika 2.1. *Ilustrirani prikaz objekta stvarnog svijeta*

Na slici (Sl.2.2.) su prikazani dijagrami dva objekta gdje im pojedinačni centar, ili jezgru, čine varijable objekta. Metode okružuju i skrivaju centar objekta od drugih objekata u programu. Pakiranje varijabli objekta sa metodama nazivamo enkapsulacija ili učahurivanje.



Slika 2.2. *Dijagram objekata*

Enkapsulacija povezanih varijabli i metoda u prikladan paket je jednostavna, ali i moćna ideja koja nudi dvije prednosti softverskim programerima:

- Modularnost: izvorni programski kôd za objekt može biti napisan i održiv nezavisno o izvornom programskom kôdu drugih objekata. Primjer iz stvarnog života: svoj bicikl možete dati nekome drugom, a on će i dalje biti vozno ispravan.
- Skrivanje informacija: objekt ima javno sučelje kako bi drugi objekti mogli komunicirati s njim. Objekt može sadržavati privatne varijable i metode koji se mogu mijenjati bilo kada bez utjecaja na ostale objekte koji ovise o njima.

Uz pomoć klase definiraju se vrste varijabli nekog objekta. Jednom definiranom klasom, kreiramo bilo koji broj objekata koji pripadaju toj klasi. Klasa predstavlja tip podataka koji definira predložak kojem se opisuju zajednička svojstva svih objekata proizašlima iz navedene klase. Sastoji se od varijabli, koje mogu biti ugrađeni ili korisnički definirani tipovi. Sintaksa deklaracije klase prikazana je na slici (Sl.2.3.). Deklaracije se sastoji od dva osnovna dijela, to su zaglavlje i tijelo klase. Zaglavlje klase se sastoji od ključne riječi *class* iza koje slijedi naziv klase. Tijelo klase slijedi nakon njezina naziva i omeđeno je parom vitičastih zagrada.

```
1  class nazivKlase    //zaglavlje klase
2  {
3
4      // tijelo klase
5
6  };
```

Slika 2.3. *Sintaksa deklaracije klase*

Klase su predlošci prema kojima kreiramo objekte. Slika (Sl.2.4.) prikazuje sintaksu kreiranja objekta na primjeru, gdje se nazivom *Vozilo* definira klasa, a nazivom *Auto* definirao objekt koji pripada klasi *Vozilo*.

```
1   int main()
2   {
3
4       Vozilo Auto;
5
6       return 0;
7   }
```

Slika 2.4. *Sintaksa kreiranja objekta*

Vrlo važno je uočiti razliku između klase i objekta; klasa je samo opis, dok je objekt stvarna, konkretna realizacija napravljena na temelju klase. Tijelo klase može sadržavati podatkovne i funkcijske članove, deklaraciju ugniježđenih klasa i specifikacije prava pristupa.

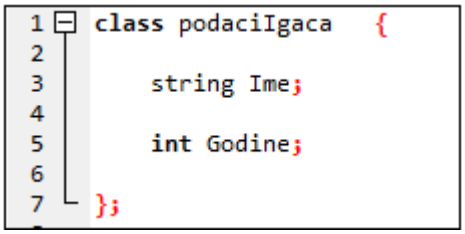
2.2. Podatkovni članovi

Podatkovni i funkcijski članovi klase mogu biti statični ili ne statični, definirani su sa ili bez ključne riječi *static*. Ukoliko se varijabla klase deklarira kao statična, bez obzira koliko se objekata klase kreira, kreirati će se samo jedna kopija statičkog člana. Obrnuta je situacija kod ne statičnih deklariranih varijabli, koji se kreiraju za svaki objekt posebno. Slika (Sl.2.5.) prikazuje jednostavnu klasu sa dvije statične varijable koje se mogu koristiti za pohranu informacija o korisniku programa.

```
1   class podaciKorisnika   {
2
3       static string Ime;
4
5       static int Godine;
6
7   };
```

Slika 2.5. *Klasa pod nazivom podaciKorisnika sa statičkim varijablama*

Pri korištenju ove klase, postoji samo jedan primjerak svake varijable *podaciKorisnika.Ime* i *podaciKorisnika.Godine*. Može sadržavati samo jednog korisnika, jer na korištenje raspolažemo memorijskim prostorom za pohranu podataka samo jednog korisnika. Klasa, *podaciKorisnika*, i njezine varijable postoje onoliko dugo koliko se program izvršava. Slična klasa koja uključuje ne statične varijable prikazana je na slici (Sl.2.6.).

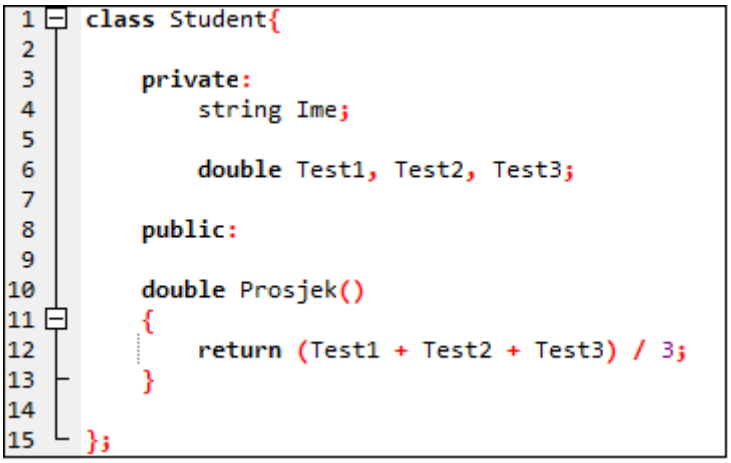


Slika 2.6. *Klasa pod nazivom podaciKorisnika sa nestatičkim varijablama*

U ovom primjeru, ne postoje varijable *podaciIgraca.Ime* ili *podaciIgraca.Godine*, jer varijable *Ime* i *Godine* nisu statični članovi klase *podaciIgraca*. Definiranjem varijabli u klasi kao ne statičnim, klasa se može koristiti za kreiranje više objekata. Svaki objekt će imati vlastite varijable *Ime* i *Godine*. Program koji će koristiti ovakvu klasu moći će pohraniti informacije više igrača u igri. Kada se novi igrač želi priključiti igri, kreira se novi objekt klase *podaciIgraca* koji će ga predstavljati. Ukoliko se neki igrač odluči napustiti igru, objekt klase *podaciIgraca* koji ga predstavlja može se uništiti.

2.3. Funkcijski članovi

Izvorni kôd funkcija ili metoda definiran je u klasi, no bolje ih je promatrati kao metode objekata. Ne statične metode u klasi definiraju samo metode koji će sadržavati svaki kreirani objekt iz klase. Na primjer, metoda *crtaj()* u dva različita objekta radi iste stvari u smislu da će oba objekta nešto nacrtati. Postoji značajna razlika između te dvije metode – stvari koje će nacrtati mogu biti različite. Stoga se može zaključiti da se rezultat metode nad objektom razlikuje od objekta do objekta, ovisno o njihovim vrijednostima varijabli. Statične i nestatične funkcijske članove možemo istodobno koristiti u jednoj klasi. Statične definirane metode u izvornom kôdu određuju ponašanje koje je dio same klase i izvršava se samo jednom, dok nestatične metode određuju ponašanje svakog objekta. Slika (Sl.2.7.) prikazuje jednostavnu klasu *Student*, koja se koristi za pohranu informacija studenata koji pohađaju nekakav tečaj.



Slika2.7. *Klasa s nazivom Student koja sadrži podatkovne članove Ime, Test1, Test2, Test3 i metodu Prosjek*

Nijedan od članova klase nije deklariran kao statični, što omogućava kreiranje objekata klase *Student*. Svaki kreirani objekti sadržavat će varijable naziva *Ime*, *Test1*, *Test2*, *Test3* i metode koja se naziva *Prosjek*. Različiti objekti imat će i različite vrijednosti imena i testova. Prilikom poziva određenog objekta klase *Student*, metoda *Prosjek* će izračunati prosjek pomoću studentovih ocjena dobivenih na testovima. Također različiti studenti mogu imati različite izračunate prosjeke.

2.4. Prava pristupa

Na slici (Sl.2.7.) deklariranim varijablama navedeno je privatno *(eng. private)* pravo pristupa podacima ključnom riječi *private*. Osim privatnog koriste se javni, (*eng. public)* i zaštićeni, (*eng. protected)* pristup podacima i metodama klase. Prava pristupa određuju koji će članovi klase biti dostupni izvan klase ili samo unutar klase, a koji iz naslijeđenih klasa. Sve varijable i funkcije koje slijede iza navedene ključne riječi i dvotočke imaju navedeno pravo pristupa, do kraja deklaracije ili do početka sljedeće ključne riječi za pravo pristupa. Članovi navedeni u deklaraciji klase odmah nakon prve vitičaste zagrade do prve ključne riječi, ili do kraja klase ako prava pristupa uopće nisu navedena, imaju privatan pristup. [2]

```
1  class pravaPristupa{
2
3      public:
4          float e, f;
5          void Funkcija1 (float brojac);
6
7      private:
8          string g;
9
10     protected:
11         int h;
12         int Funkcija2();
13
14 };
```

Slika 2.8. *Klasa s nazivom pravaPristupa i članovima različiti prava pristupa*

Slika (Sl.2.8.) prikazuje klasu *pravaPristupa* koja sadrži varijable i metode s različitim pravima pristupa. Na primjeru klase biti će objašnjena svaka od tri navedena prva pristupa:

- javni pristup dodjeljuje se ključnom riječi *public*. Javni pristup omogućava članovima da im se pristupa izvan klase . Varijable s nazivima *e* i *f*, te *Funkcija1( )* mogu se pozvati tako da se definira neki objekt klase *pravaPristupa* te im se pomoću operatora točka (.). pristupi
- Varijabla *g* ima privatni pristup, koji je dodijeljen s ključnom riječi *private*. Ta varijabla nije dostupna naslijeđenim klasama niti bili kojoj vanjskoj definiranoj metodi– može mu se pristupiti samo preko funkcijskih članova klase *pravaPristupa*
- zaštićeni pristup određen je ključnom riječi *protected*. Tako se varijabli *h* i funkcijskom članu *Funkcija2( )* ograničava pristup. Njima se ne može pristupiti iz vanjskog programa preko objekta. Dostupni su samo funkcijskim članovima klase *pravaPristupa* i njezinim naslijeđenim klasama.

2.5. Konstruktor

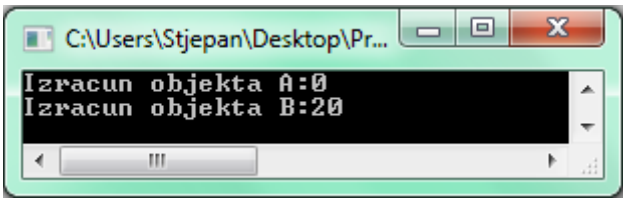
Kreiranje objekta zahtijeva inicijalizaciju njegovih varijabli kako bi mu postali upotrebljivi te nebi vraćali neočekivane vrijednosti. Svaka klasa može sadržavati konstrukcijski član nazvan konstruktor, koji se automatski poziva prilikom kreiranja objekta. Konstruktor je samim time metoda unutar klase, koja mora imati isto ime kao i klasa, i ne smije imati nikakav povratni tip. Jedna klasa može sadržavati jedan ili više konstruktora. Namjena više konstruktora se očituje u vrijednostima njihovih parametara. Ukoliko su nam nepoznate vrijednosti svih svojstava objekta pozivamo konstruktor bez parametara, defaultni konstruktor, koji će svojstva

postaviti na neke početne vrijednosti. Ukoliko znamo vrijednosti svih svojstava klase, pozivamo konstruktor sa parametrima, parametarski konstruktor, koji će vrijednosti svojstava postaviti na vrijednosti koje smo proslijedili preko parametara.

```
1  class Trokut{
2
3      private:
4          int a, b;
5
6      public:
7          Trokut ()               //konstruktor sa zadanim parametrima
8          {
9              a = 0;
10             b = 0;
11         }
12         Trokut( int x, int y)   //konstrukttor sa parametrima
13         {
14             a = x;
15             b = y;
16         }
17         int Izracun()           //metoda koja računa umnožak varijabli
18         {
19             return ( a * b);
20         };
21 };
22
23 int main()
24 {
25
26     Trokut A;               //kreiranje objekta A sa konstruktorom bez parametara
27     Trokut B(5,4);          //kreiranje objekta B sa konstruktorom sa parametara
28     cout<<"Izracun objekta A:"<< A.Izracun() <<endl;
29     cout<<"Izracun objekta B:"<< B.Izracun() <<endl;
30
31     return 0;
32 }
```

Slika 2.9. *Klasa Trokut sa svojim podatkovnim i funkcijskim članovima, te kreiranje objekata sa različitim konstruktorima i izvršavanje metoda nad njima*

Na slici (Sl.2.9.) je prikazan programski kôd u kojem je definirana klasa *Trokut* koja sadrži podatkovne i funkcijske članove. Sadrži dva konstruktora, jedan sa parametrima i jedan gdje su zadane vrijednosti varijablama *a* i *b*. Također sadrži i metodu Iz*racun ( )* koja vraća umnožak varijabli objekta *a* i *b*. Prilikom kreiranja objekta u glavnom programu automatski se poziva konstruktor. Objekt *A* kreiran je defaultnim konstruktorom sa već zadanim vrijednostima varijabli. Objekt *B* kreiran je konstruktorom sa parametrima, koji se navode nakon naziva objekta u zagradama. Prevođenjem izvornog koda programa dobiven je rezultat prikazan na slici 2.10.



Slika 2.10. *Rezultat prevođenja izvornog koda*

2.6. Destruktor

Kada neki objekt želimo uništiti, on se mora ukloniti iz memorije, tada koristimo destruktor koji je zadužen za oslobađanje svih resursa dodijeljenih objektu. Poziva se prije nego što se memorija zauzeta objektom oslobodi te se obavlja deinicijalizacija objekta. Destruktor mora imati isto ime kao i klasa, no ispred sebe ima znak tildu (~). Kao i konstruktor, nema povratni tip. Automatski se poziva u sljedećim situacijama:

- na kraju bloka za automatske objekte u kojem je objekt definiran
- za statičke i globalne objekte nakon izlaska iz funkcije *main ( )*
- za dinamičke objekte prilikom uništenja dinamičkog objekta operatorom *delete*

Svaka klasa može sadržavati samo jedan destruktor. Na primjeru klase Trokut na slici (Sl.2.9.), destruktor bi imao naziv ~Trokut ( ).

3. POLIMORFIZAM

Polimorfizam označava svojstvo promjenjivosti oblika. U OOP omogućava definiranje metoda koje su ovisne o tipu, što znači da se sa istim nazivom može definirati više metoda s različitim tipovima podataka. Korisnik koji koristi program mora znati samo jedinstveno ime metode i tipove podataka koji su definirani s metodom. Polimorfizam se očituje u preopterećenju operatora i funkcija.

3.1. Preopterećenje operatora

Preopterećenje operatora je proces kojim se ugrađeni operatori primjenjuju na svojstven način, što znači da ih možemo koristiti sa korisničkim definiranim tipovima podataka. Kao i bilo koja metoda, preopterećeni operator ima povratni tip i popis argumenata. Ne mogu se svi operatori preopteretiti. Tablica (Tab.3.1.) prikazuje pet operatora koji se ne mogu opteretiti.

Tablica 3.1. *Operatori koji se ne mogu opteretiti*

.	.*	: :	?:	sizeof

Nije moguće uvoditi novi operator, već samo proširiti značenje postojećih. Operator po definiciji je funkcijski član koji za naziv ima ključnu riječ *operator* iza koje se stavlja simbol operatora. Tablica (Tab.3.2.) prikazuje popis svih operatora koji se mogu preopteretiti.[2]

Tablica 3.2. *Operatori koji se mogu opteretiti*

+	-	*	/	%	^	&	|
~	!	,	=	<	>	<=	=>
++	--	<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=	| =	* =
<<=	>>=	[ ]	( )	->	- >*	new	delete

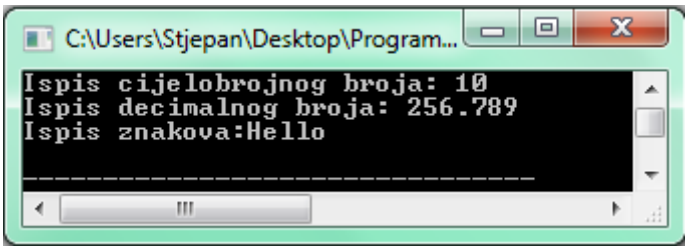
3.2. Preopterećenje funkcija

Preopterećenje funkcija omogućava korištenje istog naziva za više funkcija ili metoda koje se razlikuju po tipovima i/ili broju argumenata. Ne možemo preopteretiti funkcije koje se razlikuju samo u povratnom tipu.

```
1   class ispisPodataka
2   {
3
4      public:
5         void ispis(int i) {
6           cout << "Ispis cijelobrojnog broja: " << i << endl;
7         }
8
9         void ispis(double  f) {
10          cout << "Ispis decimalnog broja: " << f << endl;
11        }
12
13        void ispis(string c) {
14          cout << "Ispis znakova:" << c << endl;
15        }
16  };
17
18  int main(void)
19  {
20     ispisPodataka iP;
21
22     // Poziv funkcije ispis () za ispis cijelobrojnog broja
23     iP.ispis(10);
24     // Poziv funcije ispis () za ispis decimalnog  broja
25    iP.ispis(256.789);
26     // Poziv funkcije ispis () za ispis znakova
27     iP.ispis("Hello");
28
29     return 0;
30  }
```

Slika 3.1. *Primjer preopterećenja funkcije ispis( )*

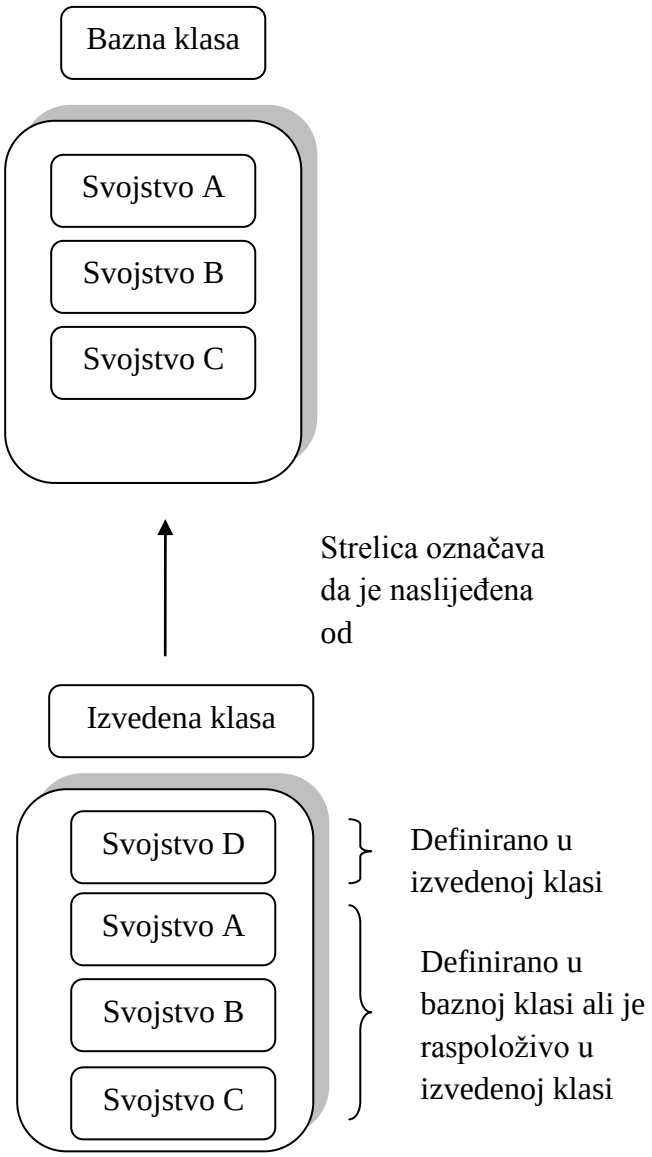
Slika (Sl.3.1.) prikazuje primjer preopterećenih funkcija s nazivom *ispis( )*. Preopterećene funkcije sadrže različite tipove argumenata. Pri prvom pozivu funkcije *ispis( )* u glavnom programu, funkcija će primiti cjelobrojni broj kao argument. Slično vrijedi i za preostala dva poziva funkcije *ispis( )* gdje se funkcija poziva za ispis decimalnog broja i ispis znakova. Rezultat prevođenja programa prikazan je na slici (Sl.3.2.).



Slika 3.2. *Rezultat prevođenja preopterećene funkcije*

4. NASLJEĐIVANJE

Jedan od najvažnijih koncepata u objektno orijentiranom programiranju je nasljeđivanje. Nasljeđivanje predstavlja definiranje jedne klase u pogledu druge klase, što ju čini lakšim za kreiranje i implementaciju u programu. Također pruža mogućnost ponovne uporabe kôda odnosno izbjegavanje višestrukog pisanja istih dijelova. Kod kreiranja klase, umjesto pisanja potpuno novih podatkovnih i funkcijskih članova, programer može definirati novu klasu nasljeđujući članove postojeće klase. Postojeća klasa se naziva baznom klasom, a nova klasa se naziva izvedenom. Ideja nasljeđivanja implementirana je na roditeljskom odnosu. Odnos nasljeđivanja prikazan je na slici (Sl.4.1.).



Slika 4.1. *Odnos nasljeđivanja bazne i izvedene klase*

Programi koji izvode nove klase od postojećih nude nekoliko prednosti:

- uštedu vremena jer dio kôda koji je potreban novoj klasi već napisan i testiran, to jest, znamo da radi ispravno i pouzdano
- proširivanje bazne klase bez ugrožavanja rada objekata i metoda izvedene klase
- ako je baznu klasu naslijedilo puno izvedenih klasa, ona je zasigurno pouzdana – što se više kôda koristilo, to je vjerojatnije da su logičke pogreške pronađene i ispravljene.

Objekti izvedenih klasa sadržavaju sve funkcijske i podatkovne članove bazne klase, a mogu sadržavati i neka svoja specifična svojstva definirana u izvedenoj klasi. Nasljeđivanje se definira na način da se u deklaraciji iza izvedene klase navodi lista baznih klasa, razdvojeni zarezom. Sintaksa nasljeđivanja izvedene klase s nazivom *imeIzvedeneKlase* od bazne klase s nazivom *imeBazneKlase* je prikazana na slici (Sl.4.2.).

```
1  class imeIzvedeneKlase : pravo_pristupa imeBazneKlase{
2      //...
3  }
```

Slika 4.2. *Sintaksa nasljeđivanja klase*

Jedna bazna klasa se ne može navesti više puta u listi navođenja baznih klasa. Ispred svakog naziva bazne klase moguće je staviti jednu od ključnih riječi *public*, *private* ili *protected*, kojima se definiraju prava pristupa.

4.1. Nasljeđivanje i prava pristupa

Prava pristupa nasljeđivanja određuju načine preslikavanja varijabli i metoda bazne klase izvedenoj klasi.

Tablica 4.1. *Pravo pristupa varijablama i metodama bazne klase prema pristupu nasljeđivanja*

Pristup nasljeđivanja	Pravo pristupa atributima i metodama bazne klase		
	Javni pristup	Zaštićeni pristup	Privatni pristup
Javni	Javni	Zaštićeni	Privatni
Zaštićeni	Zaštićeni	Zaštićeni	Privatni
Privatni	Privatni	Privatni	Privatni

Prema tablici (Tab.4.1.) vidljivo je kako različiti pristupi nasljeđivanja imaju različita prava pristupa atributima i metodama definiranim u baznoj klasi:

- javnim pristupom nasljeđivanja, članovi bazne klase koji su definirani javnim pristupom ostaju javni članovi izvedene klase kao i zaštićeni članovi koji ostaju zaštićeni i u izvedenoj klasi. Članovi bazne klase koji su definirani privatnim pristupom nisu dostupni direktno preko izvedene klase
- zaštićenim pristupom nasljeđivanja, članovi bazne klase koji su definirani javnim pristupom postaju članovi sa zaštićenim pristupom, a članovi koji su definirani zaštićenim pristupom u baznoj klasi ostaju zaštićeni članovi izvedene klase. Privatno definirani članovi bazne klase ostaju privatni članovi izvedene klase.
- privatnim pristupom nasljeđivanja, članovi bazne klase koji su definirani javim i zaštićenim pristupom postaju članovi sa privatnim pristupom u izvedenoj klasi. Privatno definirani članovi bazne klase ostaju privatni članovi izvedene klase.

4.2. Konstruktor i destruktor izvedenih klasa

Konstruktor izvedene klase zadužen je za inicijalizaciju isključivo članova definiranih u pripadajućoj klasi; članovi bazne klase se moraju inicijalizirati u konstruktoru izvedene klase. Prilikom kreiranja objekta izvedene klase poštuje se redoslijed pozivanja konstruktora. Program pri pozivu konstruktora izvedene klase prvo izvršava konstruktor bazne klase. Nakon završetka konstruktora bazne klase nastavlja se izvođenje kôda konstruktora izvedene klase, jer je klasa iz koje se kreira objekt kreirana postupnim nadograđivanjem.

Obrnuti je redoslijed kod uništavanja objekta izvedene klase. Pri pozivu destruktora prvo se izvršava destruktor izvedene pa nakon njega destruktor bazne klase.

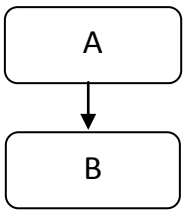
4.3. Tipovi nasljeđivanja

Tipovi nasljeđivanja se razlikuju po načinu kako je izvedena klasa naslijeđena. Tako postoji:

- jednostruko
- hijerarhijsko
- višerazinsko

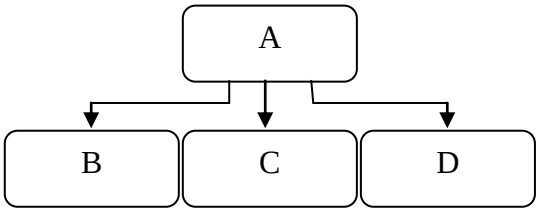
- hibridno
- i višestruko nasljeđivanje.

Jednostruko nasljeđivanje omogućuje izvedenoj klasi nasljeđivanje svojstva i ponašanja iz jedne bazne klase. Slika (Sl.5.1.) prikazuje dijagram gdje je izvedena klasa *B* naslijeđena od samo jedne bazne klase *A*.



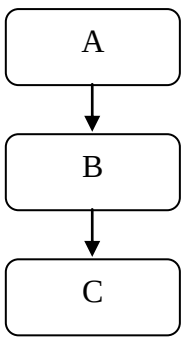
Slika 4.3. *Dijagram jednostrukog nasljeđivanja*

U hijerarhijskom nasljeđivanju više izvedenih klasa nasljeđuju svojstva i ponašanja od jedne bazne klase. Slika (Sl.5.2.) prikazuje dijagram gdje su izvedene klase *B*, *C* i *D* naslijeđene od jedne bazne klase A.



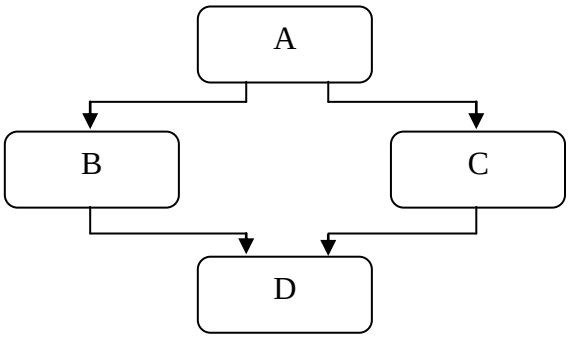
Slika 4.4. *Dijagram hijerarhijskog nasljeđivanja*

Kod višerazinskog nasljeđivanja izvedena klasa je naslijeđena od druge naslijeđene klase. Slika (Sl.5.3.) prikazuje dijagram gdje je izvedena klasa *C* naslijeđena od izvedene klase *B*, naslijeđene od bazne klase *A*.



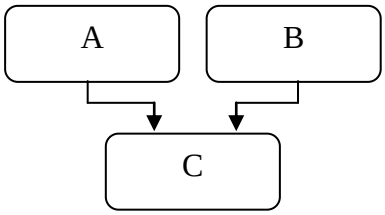
Slika 4.5. *Dijagram višerazinskog nasljeđivanja*

Hibridno nasljeđivanje je kombinacija jednostrukog i višerazinskog nasljeđivanja. Na slici (Sl.5.4.) prikazan je dijagram gdje je izvedena klasa *D* naslijeđena od dvije izvedene klase, *B* i *C*, koje su naslijeđene od iste bazne klase *A*.



Slika 4.6. *Dijagram hibridnog nasljeđivanja*

Kod višestrukog nasljeđivanja izvedena klasa je naslijeđena od više baznih klasa. Slika (Sl.5.5.) prikazuje dijagram u kojem je izvedena klasa *C* naslijeđena od baznih klasa *A* i *B*.



Slika 4.7. *Dijagram višestrukog nasljeđivanja*

5. PRIMJER PRAKTIČNE PRIMJENE OOP U C++-u

Primjer praktične primjene OOP u programskom jeziku C++-u je prikazan na programu vođenja nogometnog kluba koji ima karakteristike:

- kreiranje igrača, članova uprave i sudaca
- kreiranje kluba
- prikaz detaljnih informacija o igračima, klubovima, članovima uprave i sudcima
- definirati metode koje će omogućiti unos rezultata pojedine odigrane utakmice
- spremi/učitaj u/iz tekst datoteke
- izbornik – navigacija

Primjer je napisan u programu DEV C++. Za pohranu i ispis podataka se koristile tekstualne datoteke (ekstenzije .txt).

Program na početku izvršavanja prikazuje izbornik kojim unosom ponuđenih brojeva izvršavamo opisane metode. Tako za svaku osobu, igrač, član uprave, sudac, i klub postoji opcija kreiranja novih te pretraživanja informacija po njihovim varijablama. Prije izvršavanja odabranih metoda i kreiranja objekata, moramo definirati klase po kojima ćemo stvarati objekte i metode.

OOP temelji se na objektima koji imaju svoje stanje i ponašanje, odnosno varijable i metode nad njima. Objekti su kategorizirani po klasama. Nogometni klub se generalno sastoji od osoba, te se daljnjom podjelom dobivaju podgrupe osoba sa različitim djelatnostima u klubu. Osoba će predstavljati baznu klasu koja sadrži definirane varijable koje opisuju stanja svih osoba u nogometnom klubu i izvan njega. Prilog P.5.1. prikazuje programski kod definirane bazne klase s nazivom *osoba* koja sadrži definirane varijable svih zajedničkih stanja objekata, osoba. Također su u klasi definirani konstruktori i destruktor, koji imaju iste nazive kao klasa i nemaju povratni tip, i metode koje vračaju varijable objekta, te imaju povratni tip kao i tip varijable.

Osobe su u klubu podijeljene po članovima uprave kluba, igrače, te vanjske osobe koje sudjeluju na nogometnim utakmicama kluba, sudce. Tako su definirane izvedene klase s nazivima *Uprava, Igraci, Sudci*. Svaka od njih naslijedit će varijable i metode koje sadrži bazna klasa *osoba*. Također svaka od naslijeđenih klasa sadrži vlastite privatne definirane varijable. Kako se u prilozima P.5.2., P.5.3., P.5.4. vidi, definirane su naslijeđene klase sa vlastitim

metodama za upis. Varijable bazne klase ne moraju biti definirane u izvedenoj klasi da bi ih izvedena klasa mogla koristiti. Za kreiranje kluba koji se natječe u prvenstvu definirana je klasa s nazivom *klub*.

Za pohranu i ispis iz datoteke koristimo klase *ifstream* za čitanje i *ofstream* za upisivanje. Kreiranje i pohrana podataka u tekstualnu datoteku prikazana je u prilogu P.5.5., na primjeru igrača. Otvaramo tekstualnu datoteku gdje će se spremati vrijednosti varijabli objekta. Kreira se objekt te se poziva metoda za upisivanje vrijednosti traženih varijabli koje se pridodaju objektu. Nakon toga se varijable objekta pohranjuju pomoću metoda koje vraćaju vrijednost upisanih varijabli. Isti način kreiranja i pohrane je kod članova uprave i sudaca kao i kod kluba, jedina razlika je u traženim varijabla i tekstualnim datotekama u koje spremamo vrijednosti. Pretraživanje se vrši prema odabiru iz izbornika po čemu želimo pretraživati osobe.

Na izboru su nam sve varijable koje sadrži svaki kreirani objekt klase *Igraci*, *Uprava*, *Sudci*, *klub*. Nakon odabira upisujemo traženu vrijednost i pomoću *while* i *if* petlji s kojima se provjerava i uspoređuje svaka linija sa osobom ili osobama sa traženom vrijednošću koje se na kraju ispisuju. Programski kôd za pretraživanje prikazan je u prilogu P.5.6. Svako pretraživanje ima isti način provedbe, razlika je u varijablama po kojima pretražujemo, tip varijable i tekstualna datoteka iz koje pretražujemo. U glavnom izborniku imamo opciju koja omogućava pregled klubova koji nastupaju u prvenstvu. Odabirom na tu opciju ispisuje se tablica sa rednim brojem svakog kluba i nazivom.

Program omogućava unos rezultata odigranog kola prvenstva opcijom *Unos rezultata* u izborniku, gdje klub koji se služi programom naziva *Nk Slavija.* Unos rezultata sastoji se od odabira protivnika iz liste svih klubova, odabira jednog suca iz liste sudaca te unosa rezultata, broj golova pojedinačne ekipe. Pregled svih odigranih kola nudi nam opcija *Prikazi rezultate.*

6. ZAKLJUČAK

Ovim radom su objašnjene osnove objektno orijentiranog pristupa programiranju. OOP nudi nova i bolja rješenja zahtjevnih zadataka u odnosu na starije pristupe programiranja, koja se očituju u većoj produktivnosti programera, boljom kvalitetom softvera i manjim troškovima održavanja. Što se iz samog naziva vidi, objekti su ključ ovoga pristupa programiranju. Kategorizacija objekta ostvaruje se klasama koje sadrže definirana svojstva i ponašanja objekta iz stvarnog svijeta. Imamo mogućnost kreirati više objekata iz jedne klase ili klase koja je naslijedila drugu, što programerima olakšava posao jer moraju upotrebljavati isti programski kôd. Ta prednost je opisana najvažnijim konceptom OOP nasljeđivanjem, gdje možemo eliminirati suvisli kôd te nadograditi postojeće klase. OOP nam pruža bolji mehanizam za ispravljanje grešaka ili bilo kakve izmjene u programu, jer kada primijetimo da neka naša funkcija ne radi ispravno, ne moramo mijenjati kôd u svim dijelovima programa koji je koriste, pošto je ona bila samo pozivana. Mijenjanjem definicije te funkcije na mjestu gdje je definirana, ispravit ćemo pogrešku te će takva funkcija biti ispravna u svim ostalim programima.

POPIS LITERATURE

[1] N. Plazibat, Objektno orijentirano programiranje, ECSAT d.o.o., Split, listopad 2002

[2] B. Motik, J. Šribar, Demistificirani C++, Element, Zagreb, 1997

[3] J. Farrell, Obect-Oriented Programming Using C++. Course Technology, Boston, 2009.

[4] A. Pillay, Object-Oriented Pogramming, School of Computer Science, University of KwaZulu-Natal, Durban, February 2007.

[5] R. Lafore, Object-Oriented Programming in C++, SAMS, 800 East 90th St., Indianapolis, Indiana 46240 USA.

[6] V. Davidović, I. Pogarčić, Objektno orijentirane tehnologije I, Veleučilište u Rijeci, Rijeka, 2011.

[7] K. Žubrinić, Uvod u objektno orijentirano programiranje, Informatički klub FUTURA, Dubrovnik, ožujak 2014.

SAŽETAK

Osnove objektno orijentiranog programiranja u C++-u.

Zbog zahtjevnosti tržišta dolazi do razvoja novih načina programiranja. Ljudsko razmišljanje i njegovo gledište na rješavanje problema ideja su za razvoj objektno orijentiranog programiranja. Stvari koje čovjek vidi pokušavaju se kategorizirati, u objektno orijentiranom programiranju tu kategorizaciju predstavlja klasa. Klasa sadrži stanje nekog objekta i njegovo ponašanje u stvarnom svijetu. U ovom radu opisani su osnovni pojmovi objektno orijentiranog programiranja te glavni koncepti koji ga razlikuju od ostalih načina programiranja, a to su enkapsulacija, polimorfizam i nasljeđivanje. U radu je također prikazan i objašnjen praktični primjer programa za vođenje nogometnog kluba napisanog u jednom od objektno orijentiranih programskih jezika, C++.

Ključne riječi: klasa, objekt

ABSTRACT

Basic of object-oriented programming in C++.

Because of new standards set by the market there is a need of new types of programming. The human perception and it's point of view on solving problems are the cause of finding new ideas in object oriented programming. The things that a human being perceives are being categorised, in object oriented programming that categorisation is represented by classes. A class contains the state of an object and it's behaviour in real world. This bachelor thesis describes the basic concepts of object orientated programming and the main concepts that differentiate it from other types of programming. Those are: encapsulation, polymorphism and inheritance. This bachelor thesis also shows and explains an example of a program used to run a football club written in C++, one of object orientated programming languages.

Key words: class, object.

ŽIVOTOPIS

Stjepan Flisar rođen je 05.12.1993. godine u Požegi, osnovnu školu od prvog do četvrtog razreda je pohađao u Područnoj školi Antuna Kanižlića u Vidovcima, a do osmog razreda je pohađao Osnovnu školu Antuna Kanižlića u Požegi. 2008. godine upisao je Tehničku školu u Požegi, smjer tehničar za računalstvo. Stručni studij elektrotehnike, smjer informatika upisao je 2012. godine na sadašnjem Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

PRILOZI

P.5.1. *Definirana klasa osoba*

```
14  class osoba
15  {
16      protected:
17
18          string ime_o;
19          string  prezime_o;
20          int godina_o;
21          double visina_o;
22          double tezina_o;
23          double placa_o;
24          int oib_o;
25
26      public:
27
28          osoba ()
29          {
30              godina_o = 0;
31              visina_o = 0;
32              tezina_o = 0;
33              oib_o = 0;
34              placa_o=0;
35              string ime_o=" ";
36              string prezime_o=" ";
37          };
38
39          osoba(string a, string b, int c, double d, double e, double g, int f){
40              ime_o = a;
41              prezime_o = b;
42              godina_o = c;
43              visina_o = d;
44              tezina_o = e;
45              placa_o = g;
46              oib_o = f;
47          }
48
49          string get_ime_o(){
50              return ime_o;
51          }
52          string get_prezime_o(){
53              return prezime_o;
54          }
55          int get_godina_o(){
56              return godina_o;
57          }
58          double get_visina_o(){
59              return visina_o;
60          }
61          double get_placa_o(){
62              return placa_o;
63          }
64          double get_tezina_o(){
65              return tezina_o;
66          }
67          int get_oib_o(){
68              return oib_o;
69          }
70  };
```

P.5.2. *Izvedena klasa Uprava*

```
75  class Uprava : public osoba{
76
77      private:
78          string pozicija;
79
80      public:
81
82          string get_pozicija(){
83          return pozicija;
84          }
85
86
87          void upis_uprave(){
88          cout<<"Ime:";
89          cin>>ime_o;
90          cout<<"Prezime:";
91          cin>>prezime_o;
92          cout<<"Oib[13 znamenki]:";
93          cin>>oib_o;
94          cout<<"Koliko osoba ima godina:";
95          cin>>godina_o;
96          cout<<"Placa[kn]:";
97          cin>>placa_o;
98          cout<<"Pozicija u upravi kluba:";
99          cin>>pozicija;
100
101         }
102 };
```

P.5.3. *Izvedena klasa Sudci*

```
147 class Sudci : public osoba
148 {
149     private:
150         string vrsta;
151     public:
152
153         string get_vrsta()
154         {
155             return vrsta;
156
157         }
158
159         void unos_sudac(){
160         cout<<"Ime:";
161         cin>>ime_o;
162         cout<<"Prezime:";
163         cin>>prezime_o;
164         cout<<"Oib[13 znamenki]:";
165         cin>>oib_o;
166         cout<<"Koliko sudac ima godina:";
167         cin>>godina_o;
168         cout<<"Placa[kn]:";
169         cin>>placa_o;
170         cout<<"Linijski/glavni sudac:";
171         cin>>vrsta;
172         }
173 };
```

P.5.4. *Izvedena klasa Igraci*

```
106 class Igraci : public osoba{
107
108     private:
109         string pozicija_igraca;
110         int broj_dresa;
111
112     public:
113
114         string get_pozicija_igraca(){
115         return pozicija_igraca;
116         }
117         int get_broj_dresa(){
118         return broj_dresa;
119         }
120
121         void unos_igrac(){
122
123         cout<<"Ime:";
124         cin>>ime_o;
125         cout<<"Prezime:";
126         cin>>prezime_o;
127         cout<<"Oib[13 znamenki]:";
128         cin>>oib_o;
129         cout<<"Koliko igrac ima godina:";
130         cin>>godina_o;
131         cout<<"Visina[cm]:";
132         cin>>visina_o;
133         cout<<"Tezina[kg]:";
134         cin>>tezina_o;
135         cout<<"Placa[kn]:";
136         cin>>placa_o;
137         cout<<"Koju poziciju igra:";
138         cin>>pozicija_igraca;
139         cout<<"Broj na dresu:";
140         cin>>broj_dresa;
141         }
142 };
```

P.5.5. *Kreiranje i pohrana podataka*

```
624 void unos_igraca()
625 {
626
627     ofstream novi_igrac("igraci.txt", ios::app);
628     int kolicina, i;
629     cout<<"Koliko igraca zelite unijeti:";
630     cin>>kolicina;
631     for(i=0; i<kolicina; i++)
632     {
633     Igraci I;
634     I.unos_igrac();
635     novi_igrac << I.get_ime_o() << ' ' << I.get_prezime_o() << ' ' << I.get_oib_o() << ' ' << I.get_godina_o() << ' '
636     << I.get_visina_o() << ' ' <<I.get_tezina_o() << ' '<<I.get_placa_o()<< ' ' << I.get_pozicija_igraca() << ' ' << I.get_broj_dresa()<<endl;
637     }
638     novi_igrac.close();
639     main();
640 }
```

P.5.6. *Pretraživanje člana uprave po imenu*

```
370 void trazi_ime_uprave(){
371 
372         ifstream trazi_uprava("uprava.txt");
373         string ime_o;
374         string  prezime_o;
375         int godina_o;
376         string pozicija;
377         double placa_o;
378         int oib_o;
379         string tr_ime_uprave;
380         cout<<"Pretrazivanje po imenu: ";
381         cin>>tr_ime_uprave;
382         system("cls");
383         cout<<"Pronadjena/e osoba/e uprave sa imenom: "<<tr_ime_uprave<<endl;
384         cout<<"----------------------------------------"<<endl;
385         while (trazi_uprava >> ime_o >> prezime_o >> oib_o >> godina_o >> placa_o >> pozicija)
386         {
387 
388             if( tr_ime_uprave == ime_o){
389 
390                 cout<<"Ime: "<<ime_o<<endl;
391                 cout<<"Prezime: "<<prezime_o<<endl;
392                 cout<<"Oib: "<<oib_o<<endl;
393                 cout<<"Godine: "<<godina_o<<endl;
394                 cout<<"Placa: "<<placa_o<<endl;
395                 cout<<"Pozicija: "<<pozicija<<endl;
396                 cout<<"----------------------------------"<<endl;
397             }
398 
399         }
400 
401         system ("pause");
402         cin.get();
403         pretrazivanje_uprave();
404 
405 }
```

Kompletni izvorni kod aplikacije kao i svi ostali materijali nalaze se na CD-u priloženom uz ovaj rad.