OBJEKT-ORIJENTISANO PROGRAMIRANJE

Oznaka predmeta: OOP

Predavanje broj: 4

Nastavna jedinica: Oblast važenja i tipovi.

Nastavne teme: Oblast važenja i životni vek. Objekti i lvrednosti. Automatski objekti. Dinamički i privremeni objekti. Memorijska oblast. Konverzije tipova. Ugrađeni izvedeni tipovi.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Dragan Milićev, “Objektno orijentisano programiranje na jeziku C++”, Mikro knjiga, Beograd, 2005.

Oblast važenja i životni vek

- Oblast važenja (opseg važenja, engl. *scope*) nekog imena u programu je onaj deo teksta u kome se to ime može koristiti.
    - Kaže se da se dato ime "vidi" u svojoj oblasti važenja.
    - Van definisane oblasti važenja svako korišćenje datog imena predstavlja grešku u prevođenju(ime se "ne vidi").
- U jeziku C++ postoje četiri vrste oblasti važenja imena:
  - oblast fajla,
  - oblast klase,
  - oblast funkcije,
  - lokalna oblast.
- Lokalna imena su deklarisana unutar bloka (složene naredbe), uključujući i krajnje spoljašnji blok tela funkcije, ili kao formalni argumenti funkcije.
  - oblast važenja lokalnih imena počinje od mesta svoje deklaracije sve do kraja bloka u kome su deklarisana.
  - Formalni argumenti funkcija imaju oblast važenja kao da su deklarisana u krajnjem spoljašnjem bloku tela funkcije.
- Lokalna imena se mogu koristiti samo u bloku u kome su deklarisana, kao i u blokovima koji su ugnežđeni unutar tog bloka, počevši od mesta deklaracije.

Oblast važenja i životni vek

- Primer:

```
void f (int i) {    // i je lokalno ime
  int j,k;          // j,k su lokalna imena
  //...
  {                 // ugnežđeni blok
    char c;         // c je lokalno ime u ugnežđenom bloku
    //...           // ovde su vidljivi i,j,k,c
  }
            // ovde c više ne postoji; vidljivi su i,j,k
}           // ovde i,j,k više ne postoje
```

- Lokalno ime u nekom bloku sakriva (čini da se ono samo vidi, a ne ime koje je sakriveno) isto ime koje je bilo vidljivo u okružujućem bloku. Na primer:

```
void f (int i) {
  int j=0;
  {
    int j=1;       // ovo je novi j, lokalan za ugnežđeni blok
    j++;           // odnosi se na j iz ugnežđenog bloka
    j+=i;          // koristi se argument i
  }
  j++;         // odnosi se na j iz spoljašnjeg bloka funkcije
}
```

Oblast važenja i životni vek

- Formalni argumenti funkcije imaju lokalnu oblast za krajnje spoljašnji blok tela funkcije. Primer:

```
void f  (int i) {
int i; // greška: dve definicije istog imena u istoj oblasti!
}
```

- Funkcijsku oblast važenja imaju samo labele (naredba *goto*) jer se definišu unutar funkcije i koriste u njoj bilo gde bez obzira na ugnežđene blokove.
- Oblast važenja fajla imaju *globalna* imena koja su deklarisana van svih blokova i van svih klasa (vide se od deklaracije pa sve do kraja prevođenja fajla). Globalna imena: imena klasa, funkcija nečlanica i imena promenljivih proizvoljnog tipa.

```
int x;              // globalni x;
void f () {
  int x;            // lokalni x, sakriva globalni x;
  x=1;              // pristup lokalnom x;
  {
    int x;          // lokalni x, sakriva prethodnog;
    x=2;            // pristup drugom lokalnom x;
  }
  x=3;              // pristup prvom lokalnom x;
}
int *pi=&x; //pi je globalna promenljiva, uzima adresu globalnog x
```

Oblast važenja i životni vek

- Globalnom imenu koje je sakriveno lokalnim imenom se pristupa navođenjem operatora **::** ispred imena:

```
int x;              // globalni x;
void f () {
  int x=0;          // lokalni x;
  x=1;              // pristup lokalom x;
  ::x=2;            // pristup globalnom x;
}
```

- Oblast važenja klase imaju članovi klase. Njima se pristupa samo unutar funkcije članice te klase, ili preko operatora za pristup članovima klase: operator **.** (preko objekta) i operator **->** (preko pokazivača na objekat).
- Ako je ime člana klase sakriveno od strane nekog drugog imena, njemu se ipak može pristupiti preko operatora **::** , ispred koga se nalazi ime klase.
- Mesto početka oblasti važenja nekog imena je mesto njegove deklaracije. To je tačka iza koje je poznat u potpunosti tip tog imena, a pre same inicijalizacije, ako je ima. Besmisleno je ime upotrebiti i u izrazu u kome se inicijalizuje:

```
void f () {  int x=0;
    {int x=x;} // promenljiva x se inicijalizuje na sopstvenu,
}              // nedefinisanu vrednost
```

Objekti i lvrednosti

- U jeziku C++ pojam *objekta* ima dva značenja.
  - Objekat je primerak klase.
  - Objekat je područje u memoriji podataka za vreme izvršavanja programa.
    - Objekat je entitet koji ima svoj tip i za koji je odvojen prostor u memoriji, sinonim za promenljivu nekog tipa.
- Lvrednost je izraz koji može da stoji sa leve strane znaka jednakost.
- Pojam lvrednosti koristi se tako što je za sve operatore precizno definisano da li zahtevaju kao operande lvrednosti i da li vraćaju lvrednosti kao rezultat (operator = za ugrađene tipove zahteva lvrednost).
- Levi operand operatora = može biti ime promenljive, jer je ono lvrednost.
- Specifikacija operatora = kaže da on vraća rezultat koji je lvrednost. To znači da se ovaj rezultat može opet upotrebiti kao levi operand istog operatora, pa se može napisati:

```
int a=1,b=2,c=3;
(a=b)=c;    // posle ovoga a i c imaju vrednost 3,
            // b ima vrednost 2
```

- Iz navedenog pravila termin lvrednost i vodi poreklo (*lvalue* za *left-value*): on ima smisao "onoga što može da stoji sa leve strane znaka dodele“

Objekti i lvrednosti

- Za operator sabiranja + se ne zahtevaju lvrednosti kao operandi, niti on vraća lvrednost kao rezultat. Kako operator dodele = zahteva lvrednost za levi operand, ali ne i za desni, to se može pisati:

  ```
  c=a+b;
  (a+b)=c;    // greška: levi operand operatora = nije lvrednost!
  ```

- Životni vek objekta je vreme u toku izvršavanja programa za koje neki objekat postoji u memoriji računara.
  - Na početku svog životnog veka objekat se kreira: ako je objekat pripadnik neke klase, poziva se konstruktor te klase koji inicijalizuje taj objekat.
  - Na kraju svog životnog veka objekat se ukida: ako je objekat pripadnik neke klase koja ima destruktor, poziva se destruktor te klase koji ukida taj objekat.
  - Objektu se može pristupati samo u toku njegovog životnog veka.
- U odnosu na životni vek, u jeziku C++ postoje sledeće kategorije objekata:
  - automatski,
  - statički,
  - dinamički,
  - privremeni.

Automatski objekti

- *Automatski* (*automatic*) objekti imaju svoj životni vek koji:
  - počinje od trenutka nailaska toka programa na njihovu deklaraciju
  - se završava u trenutku napuštanja oblasti važenja tog objekta.
  - Ovo su uvek lokalni objekti za neki blok i životni vek im se automatski završava kada se u toku izvršavanja programa napušta blok u kome su deklarisani.
  - Ovi objekti imaju poseban život za svaku aktivaciju (poziv) bloka u kome su deklarisani kao lokalni
    - (isti blok može imati više istovremenih, ugnežđenih aktivacija, od kojih svaka ima svoj poseban skup automatskih objekata).

- Automatski objekat se kreira i inicijalizuje iznova pri svakom pozivu bloka u kom je deklarisan, uvek su lokalni (po oblasti važenja).
    - Formalni argumenti funkcija su lokalni objekti (automatski).

Statički objekti

- Životni vek *statičkih* (*static*) objekata je od početka do kraja izvršavanja programa.
  - Kreiraju se i inicijalizuju samo jednom, na početku izvršavanja programa, pre poziva funkcije main(), a prestaju da žive po završetku ove funkcije.
- Globalni objekti su uvek statički po životnom veku.
- Lokalni objekti mogu biti statički ako se deklarišu navođenjem ključne reči static ispred tipa. Ovakvi objekti se kreiraju i inicijalizuju samo jednom, pri prvom pozivu bloka u kome su deklarisani. Na primer:

```
int glob=1;    // globalni statički objekat,
                  // životni vek je ceo program
   void f () {
     int lok=2;  // lokalni automatski objekat, životni vek
                 // je trajanje spoljnjeg bloka funkcije
     static int s=3; // lokalni statički objekat, oblast
                   // važenja je funkcija, a životni vek
                   // ceo program;
                      // inicijalizuje se samo jednom,
                      // pri prvom pozivu funkcije
     //...
}
```

- Moguće je deklarisati i članove klasa koji imaju statički životni vek.

Dinamički i privremeni objekti

- *Dinamički* (*dynamic*) objekti imaju životni vek koga kontroliše programer eksplicitnim navođenjem trenutka njihovog kreiranja i ukidanja.
    - Ovakvi objekti su bezimeni objekti (pristupa im se posredno).
    - Objekat ove kategorije kreira se operatorom new, a ukida operatorom delete, tako da je životni vek potpuno proizvoljan.
- *Privremeni* (*temporary*) objekti imaju kratak životni vek koji traje samo u toku izračunavanja nekog izraza.
    - Ovi objekti služe za smeštanje povratne vrednosti neke funkcije koja je pozvana u izrazu.
    - Oni se uništavaju najčešće odmah čim se njihova vrednost upotrebi za dalje izračunavanje izraza u kome se nalazi poziv funkcije.
    - Ovi objekti su bezimeni, ne može im se pristupiti, a često su i fiktivni: prevodilac ne odvaja poseban prostor za njih, već njihovo smeštanje na neki način optimizuje.
        - Smisao ovakvih objekata je da obezbede semantiku kreiranja i ukidanja objekta koji čuva povratnu vrednost neke funkcije.
- Objekti koji su članovi neke klase imaju životni vek isti kao i objekat kome pripadaju.

Memorijska oblast

```
int a=1;
void f() {
  int b=1;         // inicijalizuje se pri svakom pozivu
  static int c=1; // inicijalizuje se samo pri prvom pozivu
  a++;  b++;  c++;
}
void main () {   for(;a<4;) f(); }
// Neposredno pre povratka iz funkcije f() biće:
// pri 1. pozivu f():  a=2, b=2, c=2
// pri 2. pozivu f():  a=3, b=2, c=3
// pri 3. pozivu f():  a=4, b=2, c=4
```

- Pojam memorijske oblasti objekta (*storage class*):
- Prevodilac objekte smešta u različite strukture podataka u memoriji računara, posebno namenjene za njihovu kategoriju prema životnom veku.
- Da bi obezbedio da svaki objekat ima svoj odgovarajući životni vek, prevodilac organizuje posebne oblasti u memoriji u koje po određenom pravilu smešta objekte.
- Memorijske oblasti: za automatske objekte, za statičke objekte, za dinamičke objekte, za registarske objekte i za spoljne (eksterne) objekte

Memorijska oblast

- Automatski objekti se najčešće smeštaju na stek, čime se obezbeđuje da svaka nova aktivacija nekog bloka ima svoj skup automatskih promenljivih.
  - Stek je LIFO struktura podataka.
  - Pri svakom pozivu bloka, na steku se formira struktura podataka koja sadrži sve njegove automatske objekte.
  - Izvršni kôd bloka operiše objektima na vrhu steka, tako da ne utiče na objekte eventualnih prethodnih aktivacija istog ili drugih blokova. Kada se blok završi, njegovi automatski objekti se skidaju sa steka.
- Statički objekti se na početku izvršavanja programa smeštaju u posebnu oblast u memoriji (nema nikakva posebna svojstva) i tu ostaju do njegovog kraja.
- Dinamički objekti se smeštaju u oblast u memoriji koja se naziva *dinamička memorija* (*freestore* ili *heap*). Pri kreiranju dinamičkog objekta izdvaja se prostor potreban za njegovo smeštanje, a kada se taj objekat ukida, prostor se oslobađa za ponovnu upotrebu.
- Dubina ugnežđavanja poziva zna se tek u vreme izvršavanja programa, tako da je moguće je da se pri izvršavanja programa dogodi prekoračenje memorijske oblasti za smeštanje automatskih objekata (*stack overflow*) i/ili dinamičkih objekata (*heap overflow*).
  - Program reaguje na ovakve greške prema implementaciji prevodioca.

Memorijska oblast

- Posebno mesto za smeštanje objekata je interna memorija procesora, tj. registri procesora.
- Jezik C++ omogućuje da programer zahteva od prevodioca da neke objekte smesti u registre procesora, kako bi pristup njima bio brži.
  - Ovaj zahtev prevodilac ne mora da ispuni, što zavisi od broja raspoloživih registara i ukupne veličine registarskih objekata.
  - Ovakvu memorijsku oblast mogu imati samo lokalne promenljive. Zahtev prevodiocu se postavlja stavljanjem ključne reči register ispred tipa objekta:

```
void f(register int i) {  // formalni argument automatski,
                          // pa može da se pise register
  register int j=0;
  //...
}
```

- Ako neki objekat treba koristiti u jednom programskom fajlu, a on je definisan u nekom drugom fajlu, onda prevodilac ne treba da alocira prostor za taj objekat u fajlu koji trenutno prevodi.
  - Deklarisani objekat je alociran "negde drugde", on je "eksterni". Ovakvi objekti deklarišu se navođenjem ključne reči extern ispred tipa:

```
extern int i;//nije definicija, i je definisan u drugom fajlu
```

O konverzijama tipova

- C++ je strogo tipizirani jezik (*svaki* objekat i funkcija imaju svoje strogo definisane tipove).
- Svaka funkcija ima svoj tip određen brojem i tipovima argumenata koje ta funkcija prima kao i tipom rezultata koji ta funkcija vraća.
- Prema tipu prevodilac zna šta se može raditi sa objektom ili funkcijom.
- Za svaki operator jezika je definisano koje tipove operanada zahteva.
- Ako postoji potreba da se na nekom mestu upotrebi objekat nekog drugog tipa od onog koji se zahteva potrebno je izvršiti *konverziju* (*conversion*) tipa:
  1. Ako naredba zahteva izraz koji daje rezultat određenog tipa, a koristi se izraz koji daje rezultat nekog drugog tipa. (switch((int)d1/d2))
  2. Ako operator jezika zahteva operand određenog tipa, a koristi se operand nekog drugog tipa. (d1=d2+i)
  3. Ako funkcija zahteva formalni argument nekog tipa, a kao stvarni argument dostavlja se objekat nekog drugog tipa. (c=f((int)d))
  4. Vraćanje vrednosti iz funkcije koja je deklarisana da vraća rezultat jednog tipa, a u izrazu iza return nalazi se vrednost drugog tipa. (return (int)d;)
  5. Ako se neki objekat inicijalizuje (u deklaraciji) objektom nekog drugog tipa. (int i1 = (int) d1;)

O konverzijama tipova

- Neke konverzije definisane su jezikom C++ (ugrađene su u jezik) i njih obezbeđuje sam prevodilac kada je to potrebno.
- Ove konverzije su *standardne konverzije* (*standard conversions*) i definisane su za ugrađene tipove podataka, kao i generalno za osnovne i izvedene klase.
- Korisnik može definisati svoje takozvane *korisničke konverzije* (*user-defined conversions*) za svoje tipove (klase).
  - Korisnik može definisati koji korisnički tip se može konvertovati u neki drugi korisnički ili ugrađeni tip, ili koji se ugrađeni tip može konvertovati u neki korisnički tip, kao i način na koji se to radi.
- Ako se na nekom od gore nabrojanih mesta očekuje objekat nekog tipa, a koristi se objekat drugog tipa, i ako je definisana standardna ili korisnička konverzija koja se na tom mestu može (po pravilima jezika) upotrebiti, onda će prevodilac sam, bez ikakvog eksplicitnog zahteva u programu, obezbediti da se ta konverzija izvrši.
  - ovakva konverzija naziva se *implicitnom*.
- Korisnik može eksplicitno zahtevati na nekom mestu da se izvrši konverzija koju prevodilac sam ne bi izvršio.
  - Ovakve konverzije su *eksplicitne* i zahtevaju se operatorom konverzije

Ugrađeni tipovi podataka

- Ugrađeni tipovi podataka mogu da služe kao elementi za izgradnju složenijih struktura podataka unutar korisničkih tipova.
- Ugrađeni tipovi podataka (*built-in types*) obuhvataju:
  - osnovne tipove (*fundamental types*) su model po kome se elementarni podaci predstavljaju na datoj mašini
  - jedan broj izvedenih tipova (*derived types*) podataka:
    - nizovi,
    - funkcije,
    - reference,
    - pokazivači,
    - pokazivači na članove klasa,
    - konstante.
- Tipovi podataka:
  - strukture,
  - unije,
  - klase.

  se nazivaju *korisničkim* (*user-defined types*) tipovima.

Tipovi char

- Objekti tipa char služe za prihvatanje znakova iz skupa znakova date mašine.
- Vrednost objekta tipa char ekvivalentna je celobrojnoj vrednosti znaka koji je smešten u taj objekat, prema načinu kodovanja znakova na datoj mašini.
  Na primer, ako mašina podržava ASCII kodovanje, onda je:

```
char c1='0', // vrednost c1 je '0' ili, ekvivalentno, 48
     c2='A', // vrednost c2 je 'A' ili, ekvivalentno, 65
     c3=' '; // vrednost c3 je ' ' ili, ekvivalentno, 32
```

- Varijante tipa char::
  - `unsigned char` (neoznačeni znakovni tip, u smislu aritmetičkog znaka)
  - `signed char` (označeni znakovni tip)
- Ovi objekti zauzimaju isti prostor u memoriji, ali se razlikuju po tretiranju bita najviše težine.
  - Za signed char, najviši bit tretira se kao aritmetički znak tako da je opseg vrednosti -128,...,0,...,127.
  - Za unsigned char, celobrojna vrednost je uvek pozitivna u opsegu 0,...,255.
  - Za tip char, način tretiranja bita najviše težine zavisi od implementacije prevodioca, pa programe ne treba pisati tako da zavise od načina tretiranja aritmetičkog znaka tipa char, ukoliko se želi prenosivost programa.

Tipovi char

- Ako je veličina objekta tipa char na datoj mašini 1 bajt i ako se tip char tretira kao označen, onda je:

```
char c='\xff';              // celobrojna vrednost je -1
unsigned char uc='\xff';    // celobrojna vrednost je 255
signed char sc='\xff';      // celobrojna vrednost je -1
```

- Objekti sva ova tri tipa mogu se koristiti svugde gde se može koristiti tip int, jer postoji standardna konverzija tipa char u tip int.
  - To znači da se tip char može potpuno ravnopravno pojavljivati u izrazima zajedno sa drugim celobrojnim tipovima.
- Vrednost znakovnog objekta jednaka je celom broju kojim se koduje znak na datoj mašini. Na primer, ako mašina podržava ASCII kodovanje (ili bilo koje drugo kodovanje u kome iza znaka '0' slede redom ostale cifre):

```
char c1='0',c2;
for (int i=0; i<10; i++) {
   c2=c1+i; cout<<c2<<endl;
}
```

- U izrazu unutar for petlje, na vrednost promenljive c1 (a to je kôd znaka '0') dodaje se vrednost promenljive i (od 0 do 9). Promenljiva c2 dobija vrednosti znakova čiji su kodovi za i veći od koda za znak '0', a to su znakovi '0', '1', ... , '9'.

Tipovi int

- Pored tipa int kojim se predstavljaju celi brojevi, postoje još tipova za smeštanje celih brojeva:
  - `short int` (kratki ceo broj)
  - `long int` (dugi ceo broj)
  - `long long` (64-bitni ceo broj)
- U zavisnosti od prevodioca long int ako zauzima veći prostor u memoriji od tipa int (što ne mora biti slučaj) onda predstavlja i veći opseg celih brojeva.
- U deklaracijama se reč int iza reči short i long može izostaviti:

```
int i=0;
short int si1=127;     // tip short int
short si2=-1;          // isto tip short int
long int li1=12345L;   // tip long int
long li2=0xff;         // isto tip long int
```

- Za svaki od tipova short int, int, long , long long , postoji odgovarajući neoznačeni tip unsigned.
  - Ovaj neoznačeni tip zauzima isti prostor u memoriji kao i odgovarajući (označeni) tip, samo što se najviši bit binarnog broja kojim je predstavljena vrednost objekta ne tretira kao znak (vrednost unsigned objekta je uvek pozitivna)

Tipovi int, float i double

```
int i=-1;
unsigned short ui=0; // tip 2-bajtni neoznaceni celobrojni broj
ui+=i;               // ui dobija vrednost 65535
```

promenljiva ui će dobiti vrednost 65535, jer se taj broj dobija (na širini 16 bita) kada se od 0 oduzme 1.

- Za predstavljanje realnih (racionalnih) brojeva u pokretnom zarezu (format IEEE754 $(-1)^s\ 2^{E-127}\ (1.m)$, (po bitovima 1-8-23 float, 1-11-52 double) postoje tri tipa:
  - `float`
  - `double`
  - `long double`
- Tip double predstavlja brojeve u većoj tačnosti od tipa float, u zavisnosti od implementacije prevodioca.
  Karakteristične vrednosti za opseg i tačnost pojedinih tipova date su u standardnom zaglavlju <cfloat>. Na primer:

```
float       f=3.1f;      // promenljiva tipa float
double      d=0.0;       // promenljiva tipa double
long double ld=1.2L;     // promenljiva tipa long double
```

Nabrajanja

- Nabrajanje (*enumeration*) predstavlja skup diskretnih, unapred definisanih vrednosti.
- Nabrajanje omogućuje da korisnik definiše apstraktni, diskretni skup vrednosti koje imenuje simboličkim konstantama.
- Često je u programima potrebno da neke funkcije vraćaju kôd koji predstavlja status izvršenja odgovarajuće operacije. Operacija se može završiti uspešno, ali i neuspešno, kada funkcija treba da vrati kôd greške.
  - Ako se definiše poseban tip za statusni kôd funkcije (ne koristimo neki od ugrađenih tipova) kao skup tačno definisanih diskretnih vrednosti povećaće se čitljivost koda.

```
enum FunStat {OK,NOTFND,EMPTY,ERR};    //zavrsava sa ;
```

- Ključnom reči `enum` počinje definicija nabrajanja, sledi identifikator tipa nabrajanja (može se upotrebiti kao ime tipa pri deklarisanju objekata tipa tog nabrajanja) i sledi skup diskretnih vrednosti koje predstavljaju dato nabrajanje.
- Objekti tipa nabrajanja deklarišu se na uobičajen način:

```
FunStat retValue = OK;    // promenljiva retValue tipa FunStat
retValue = f();           // f je funkcija koja vraća tip FunStat
```

Nabrajanja

- Diskretne vrednosti iz skupa koji je naveden u deklaraciji nabrajanja predstavljaju konstante koje imaju vrednosti celih brojeva počevši od 0.
- Ova dodela vrednosti može se promeniti eksplicitnim definisanjem celobrojne vrednosti neke konstanti iz skupa. Tada je naredna konstanta za jedan veća. Na primer, skup boja ekrana računara može da se predstavi nabrajanjem:

```
enum Colors { black,        // vrednost 0
              red=10,       // vrednost 10
              green,        // vrednost 11
              white=7};     // vrednost 7
```

- Dve konstante iz istog skupa nabrajanja mogu imati i iste celobrojne vrednosti.
- Postoji standardna konverzija iz tipa nabrajanja u tip int. Pri tome ove konstante imaju odgovarajuću celobrojnu vrednost:

```
int i=white+1;              // i ima vrednost 8
```

- Međutim, obratna konverzija ne postoji. `Colors c=1;  // greška!`
- Konstantama je moguće dodeliti izraz koji sadrži pobrojane konstante:

```
enum Voce {jabuka, kruska, sljiva=kruska+5};
```

- Tipovi char, int i nabrajanja imaju naziv *celobrojni tipovi* (*integral types*).
- Celobrojni tipovi i racionalni tipovi su *aritmetički tipovi* (*arithmetic types*).

Tip void i izvedeni tipovi

- Tip void je poseban tip koji predstavlja prazan skup vrednosti. Nema objekata ovog tipa, već se on koristi za formiranje izvedenih tipova.
- Tip void upotrebljava se kao tip rezultata funkcija koje ne vraćaju nikakvu vrednost, kao i za pokazivače koji mogu da ukazuju na bilo koji objekat u memoriji (izvedeni tip `void*`).
- Izvedeni tip - Nizovi

```
Tip T[]; // predstavlja tip "niz elemenata tipa T".
```

  - Objekat ovakvog tipa deklariše se u opštem slučaju deklaracijom oblika:

```
T ime[konstantni_izraz];
```

  - Konstantni izraz mora biti celobrojnog tipa i imati vrednost veću od 0 (računa se za vreme prevođenja programa i predstavlja broj elemenata niza).
  - Ako je vrednost ovog izraza *n*, onda su elementi niza indeksirani počev od 0 zaključno sa *n*-1.
- Elementi niza mogu da budu:
  - objekti nekog od osnovnih tipova (osim tipa void),
  - pokazivači, pokazivači na članove klase,
  - objekti korisnički definisanih tipova (struktura, unija, klasa),
  - elementi tipa nabrajanja,
  - drugi nizovi.

Nizovi

- Kada su elementi niza tipa drugog niza, radi se o višedimenzionim nizovima.
- Ovakvi nizovi smeštaju se u memoriju tako da se najbrže menja krajnje desni indeks.
- Dozvoljeno je da se (samo) prva dimenzija višedimenzionog niza izostavi kada se navodi deklaracija niza koja nije definicija. To je čest slučaj pri prenosu nizova kao argumenata funkcija.

```
void f(int n[][2][7]) { // prva dimenzija se može izostaviti
  //...
  n[1][1][3]++;
}
void main() {
  int a[4][2][7];       // trodimenzionalni niz veličine 4x2x7
  //...
  f(a);// a je tipa "niz od 4 elementa tipa niz od 2 elementa
       // tipa niz od 7 elemenata tipa int"
}
```

- Kada se ***a*** prenosi kao stvarni argument pri pozivu funkcije ***f***, zapravo se prenosi samo adresa početka niza ***a***.
- Kada prevodilac prevodi funkciju ***f***, potrebno je da zna kako da izračuna gde se nalazi svaki element niza ***n*** (formalnog argumenta), npr. element ***n***[1][1][3].

Nizovi

- Adresa n[1][1][3] elementa u nizu [][2][7] se dobija na sledeći način:
  - n[1] počinje na adresi koja se dobija kada se na adresu početka niza n doda veličina 1-og elementa tipa int[2][7], a svaki ovakav element zauzima prostor veličine 2x7 veličina int.
  - izračunava se adresa početka n[1][1] tako što se na prethodno dobijenu vrednost doda veličina 1-og elementa tipa int[7] , a svaki ovakav element zauzima prostor veličine 7 veličina int.
  - na prethodno dobijenu vrednost dodaje se vrednost 3 veličine int, čime se dobija adresa traženog elementa.
- Za pristup elementu niza potrebno je znati sve njegove koordinate, kao i sve dimenzije niza osim prve (može se i razlikovati od prve dimenzije stvarnog argumenta).
  - Ako se ove dimenzije ne poklapaju sa dimenzijama stvarnog argumenta (osim prve dimenzije koja nije bitna), pristupiće se pogrešnom elementu niza-stvarnog argumenta.
- Nizovi imaju fiksnu dužinu koja se određuje u vreme prevođenja i svi se realizuju kao jednodimenzioni (višedimenzioni nizovi su nizovi nizova).
- Informacije o dimenzijama niza koriste se u vreme prevođenja.

Zadatak – kompleksan broj 1/6

```
//_____________________fajl KompleksanBroj.h _____________________
//klasa KompleksanBroj bi trebalo da omoguci sabiranje, oduzimanje
//mnozenje i deljenje kompleksnih brojeva kao i ispis kompleksnog broja
//u formi npr. 0, 10, 10+2i, 10-2i, +2i, -2i
#pragma once
class KompleksanBroj
{
   private:
      double realni_deo;
      double imaginarni_deo;
   public:
      KompleksanBroj(double formalni_argument_za_realni_deo,
                     double formalni_argument_za_imaginarni_deo);
      KompleksanBroj SaberiSa (KompleksanBroj drugi_operand);
      KompleksanBroj OduzmiSa (KompleksanBroj drugi_operand);
      KompleksanBroj PomnoziSa(KompleksanBroj drugi_operand);
      KompleksanBroj PodeliSa (KompleksanBroj drugi_operand);
      void IspisiKompleksanBroj();
};
```

Zadatak – kompleksan broj 2/6

```
//_____________________fajl KompleksanBroj.cpp _____________________
#include <iostream>
#include <iomanip>
using namespace std;
#include "KompleksanBroj.h"
#define USLOVNO_PREVEDI_EXIT

KompleksanBroj::KompleksanBroj(double formalni_argument_za_realni_deo,
                               double formalni_argument_za_imaginarni_deo)
{
    realni_deo     = formalni_argument_za_realni_deo;
    imaginarni_deo = formalni_argument_za_imaginarni_deo;
}

KompleksanBroj KompleksanBroj::SaberiSa(KompleksanBroj drugi_operand){
   return KompleksanBroj(realni_deo     + drugi_operand.realni_deo,
                         imaginarni_deo + drugi_operand.imaginarni_deo) ;
}

KompleksanBroj KompleksanBroj::OduzmiSa(KompleksanBroj drugi_operand){
   return KompleksanBroj(realni_deo     - drugi_operand.realni_deo,
                         imaginarni_deo - drugi_operand.imaginarni_deo) ;
}
```

Zadatak – kompleksan broj 3/6

```
KompleksanBroj KompleksanBroj::PomnoziSa(KompleksanBroj drugi_operand){
   return KompleksanBroj(
              (realni_deo     * drugi_operand.realni_deo) -
              (imaginarni_deo * drugi_operand.imaginarni_deo) ,
              (realni_deo     * drugi_operand.imaginarni_deo) +
              (imaginarni_deo * drugi_operand.realni_deo) ) ;
}

KompleksanBroj KompleksanBroj::PodeliSa(KompleksanBroj drugi_operand){
   double delilac =
            (drugi_operand.realni_deo     * drugi_operand.realni_deo) +
            (drugi_operand.imaginarni_deo * drugi_operand.imaginarni_deo);
   if(0==delilac) {
#ifdef USLOVNO_PREVEDI_EXIT
      exit(EXIT_FAILURE);//pozvala bi se funkcija PreKraja()
                         //errorlevel = EXIT_FAILURE = 1
#else
      cout<<"Pritisnite taster i sledi poziv funkcije abort()"<<endl;
      system("pause");
      abort();  //nasilni zavrsetak programa, errorlevel=3
#endif
   }
```

Zadatak – kompleksan broj 4/6

```
        return KompleksanBroj(
          ( (realni_deo     * drugi_operand.realni_deo) +
            (imaginarni_deo * drugi_operand.imaginarni_deo) ) / delilac,
          ( (imaginarni_deo * drugi_operand.realni_deo) -
            (realni_deo     * drugi_operand.imaginarni_deo) ) / delilac );
}

void KompleksanBroj::IspisiKompleksanBroj(){
   if( (0 == realni_deo) && (0 == imaginarni_deo) ){
      cout<<noshowpos<<0<<endl;
      return; //malo ubrzanje
   }
   if(realni_deo != 0){
      cout<<noshowpos<<realni_deo;
   }
   if(imaginarni_deo != 0){
     cout<<showpos<<imaginarni_deo<<"i";
   }
   cout<<endl;
}
```

Zadatak – kompleksan broj 5/6

```
//_____________________fajl gde Vam je main _____________________
#include <iostream>
using namespace std;
#include "KompleksanBroj.h"

void PreKraja(){
   cout<<"Pozvana je funkcija PreKraja()"<<endl;   system("pause");
}

int main(){
   atexit(PreKraja);
   KompleksanBroj z1(10.0,-2.);
   cout<<"z1=";
   z1.IspisiKompleksanBroj();

   KompleksanBroj z2(0.0,-2.0);
   cout<<"z2=";
   z2.IspisiKompleksanBroj();

   KompleksanBroj z3(0,0);
   cout<<"z3=";
   z3.IspisiKompleksanBroj();
```

Zadatak – kompleksan broj 6/6

```
    z3 = z1.SaberiSa (z2);
    cout<<"z3=z1+z2= ";     z3.IspisiKompleksanBroj();
    z3 = z1.OduzmiSa (z2);
    cout<<"z3=z1-z2= ";     z3.IspisiKompleksanBroj();
    z3 = z1.PomnoziSa(z2);
    cout<<"z3=z1*z2= ";     z3.IspisiKompleksanBroj();
    z3 = z1.PodeliSa (z2);
    cout<<"z3=z1/z2= ";     z3.IspisiKompleksanBroj();
    return 0;
}

//_____________________________IZLAZ:
z1=10-2i
z2=-2i
z3=0
z3=z1+z2= 10-4i
z3=z1-z2= 10
z3=z1*z2= -4-20i
z3=z1/z2= 1+5i
Pozvana je funkcija PreKraja()
Press any key to continue . . .
```

Nabrajanja – zadatak 1/7

```
//_____________________fajl Nabrajanja.h _____________________________
#pragma once
enum POL { MUSKI, ZENSKI, NIJE_POSTAVLJEN, BROJ_ELEMENATA_SKUPA_POL};
enum PROFESIJA { PROFESOR, INZENJER, RADNIK,
                 NIJE_POSTAVLJENA, BROJ_ELEMENATA_SKUPA_PROFESIJA };

//_____________________fajl Radnik.h _____________________________
#pragma once
#include <iostream>
#include <string>
using namespace std;
#include "Nabrajanja.h"
class Radnik{
  private:
    string ime;
    POL pol;
    PROFESIJA profesija;
  public:
    Radnik();
    void PostaviPodatke(string,POL,PROFESIJA);
    void IspisiPodatke();
};
```

Nabrajanja – zadatak 2/7

```
//_____________________fajl Radnik.cpp _____________________________
#include "Radnik.h"
Radnik::Radnik(){
  ime = "nema ime";
  pol = POL::NIJE_POSTAVLJEN;
  profesija = PROFESIJA::NIJE_POSTAVLJENA;
}
void Radnik::PostaviPodatke(string imeradnika, POL polradnika,
                                                PROFESIJA profesijaradnika){
  ime = imeradnika;
  pol = polradnika;
  profesija = profesijaradnika;
}
void Radnik::IspisiPodatke(){
  cout<<"Ime       :"<<ime<<endl;
  cout<<"Pol       :";
  switch(pol){
    case POL::MUSKI  : cout<<"muski";break;
    case POL::ZENSKI : cout<<"zenski";break;
    case POL::NIJE_POSTAVLJEN : cout<<"nije postavljen";break;
    default: exit(EXIT_FAILURE);//grubo resenje
  }
```

Nabrajanja – zadatak 3/7

```
  cout<<endl;
  cout<<"Profesija:";
  switch(profesija)  {
    case PROFESIJA::PROFESOR         : cout<<"profesor";break;
    case PROFESIJA::INZENJER         : cout<<"inzenjer";break;
    case PROFESIJA::RADNIK           : cout<<"radnik";break;
    case PROFESIJA::NIJE_POSTAVLJENA : cout<<"nije postavljena";break;
    default: exit(EXIT_FAILURE);//grubo resenje
  }
  cout<<endl;
}
//_____________________fajl gde Vam je main _____________________________
#include <iostream>      #include <string>
#include "Radnik.h"
using namespace std;
const int BROJ_RADNIKA = 3;//ili zastarelo #define BROJ_RADNIKA 3
int main(){
  Radnik radnici[BROJ_RADNIKA]; //fiksno, nedinamicki
  string ime;
  int intpol;
  int intprofesija;
  POL pol;
  PROFESIJA profesija;
```

Nabrajanja – zadatak 4/7

```
cout<<"INICIJALNO:"<<endl;
for (int i=0; i < BROJ_RADNIKA; i ++){
  cout<<"Radnik "<<i<<endl;
  radnici[i].IspisiPodatke();
  cout<<endl;
}
for (int i=0; i < BROJ_RADNIKA; i ++){
  cout << "Radnik["<< i+1 <<"] Unesite ime: ";
  cin  >> ime;
  cout << "Radnik["<< i+1 <<"] ";
  cout << "Unesite pol (0=muski, 1=zenski, 2=nije postavljen): ";
  cin  >> intpol;
  switch(intpol){
    case POL::MUSKI           : pol = POL::MUSKI; break;
    case POL::ZENSKI          : pol = POL::ZENSKI; break;
    case POL::NIJE_POSTAVLJEN : pol = POL::NIJE_POSTAVLJEN; break;
    default: exit(EXIT_FAILURE);//grubo resenje
  }
  cout <<"Radnik["<< i+1 <<"] ";
  cout <<"Unesite profesiju"<<endl;
  cout <<"(0=profesor, 1=inzenjer, 2=radnik, 3=nije postavljena): ";
  cin  >> intprofesija;
```

```
    switch(intprofesija)
    {
      case PROFESIJA::PROFESOR : profesija = PROFESIJA::PROFESOR;break;
      case PROFESIJA::INZENJER : profesija = PROFESIJA::INZENJER;break;
      case PROFESIJA::RADNIK   : profesija = PROFESIJA::RADNIK  ;break;
      case PROFESIJA::NIJE_POSTAVLJENA :
                           profesija = PROFESIJA::NIJE_POSTAVLJENA;break;
      default: exit(EXIT_FAILURE);//grubo resenje
    }
    radnici[i].PostaviPodatke(ime,pol,profesija);
  }

  cout<<endl<<"R A D N I C I"<<endl;
  for (int i=0; i < BROJ_RADNIKA; i ++)
  {
    radnici[i].IspisiPodatke();
  }

  system("pause"); return 0;
}
```

Nabrajanja – zadatak 6/7

```
//_____________________________IZLAZ:
INICIJALNO:
Radnik 0
Ime       :nema ime
Pol       :nije postavljen
Profesija:nije postavljena

Radnik 1
Ime       :nema ime
Pol       :nije postavljen
Profesija:nije postavljena

Radnik 2
Ime       :nema ime
Pol       :nije postavljen
Profesija:nije postavljena

Radnik[1] Unesite ime: Pera
Radnik[1] Unesite pol (0=muski, 1=zenski, 2=nije postavljen): 0
Radnik[1] Unesite profesiju
(0=profesor, 1=inzenjer, 2=radnik, 3=nije postavljena): 0
```

Nabrajanja – zadatak 7/7

```
Radnik[2] Unesite ime: Tina
Radnik[2] Unesite pol (0=muski, 1=zenski, 2=nije postavljen): 1
Radnik[2] Unesite profesiju
(0=profesor, 1=inzenjer, 2=radnik, 3=nije postavljena): 1
Radnik[3] Unesite ime: Elvis
Radnik[3] Unesite pol (0=muski, 1=zenski, 2=nije postavljen): 0
Radnik[3] Unesite profesiju
(0=profesor, 1=inzenjer, 2=radnik, 3=nije postavljena): 2

R A D N I C I
Ime      :Pera
Pol      :muski
Profesija:profesor

Ime      :Tina
Pol      :zenski
Profesija:inzenjer

Ime      :Elvis
Pol      :muski
Profesija:radnik

Press any key to continue . . .
```