Sadržaj

1.Funkcije....3
1.1 Definicija funkcije....3
1.2 Deklaracija (prototip) funkcije....4
1.3 Poziv funkcije....5
1.4 Argumenti funkcije....6
1.4.1 Poziv po vrijednosti....7
1.4.2 Poziv po pokazivaču....8
1.4.3 Poziv po referenci....10
1.4.4 Zadane vrijednosti za parametre....11
1.5 Zadaci za vježbu....12
2. Rekurzivne funkcije....17
2.1 Implementacija rekurzije....19
2.2 Zadaci za vježbu....20
3. Brojevi....24
3.1 Definisanje brojeva u C++....24
3.2 Matematičke operacije u C++....25
3.3 Nasumični brojevi u C++....27
3.4 Zadaci za vježbu....29
4. Dvodimenzionalni statički nizovi....31
4.1 Inicijalizacija dvodimenzionalnih nizova....31
4.2 Pristup elementima dvodimenzionalnog niza....33
4.3. Zadaci za vježbu....34
5. Reference....38
5.2 Reference vs pokazivači....38
5.3 Reference u C++....38
5.3.1 Reference kao parametri....39
5.3.2 Reference kao povratne vrijednosti....40
6. Pokazivači....42
6.1 Šta su pokazivači....42
6.2 Upotreba pokazivača u C++....45
6.3 NULL pokazivači....46
6.4. Operacije nad pokazivačima....47
6.4.1 Povećanje pokazivača (inkrement)....50
6.4.2 Smanjenje pokazivača (dekrement)....51

6.4.3 Poređenje pokazivača....52
6.5 Pokazivači vs nizovi....53
6.6 Niz pokazivača....55
6.7 Pokazivači i funkcije....57
6.7.1 Poziv funkcije po referenci korištenjem pokazivačkih parametara....60
6.8 Zadaci za vježbu:....60
7 Strukture....65
7.1 Definicija strukture....65
7.2 Pristupanje članovima strukture....65
7.3 Strukture kao argumenti funkcije....69
7.4 Zadaci za vježbu....71
8 Dinamički nizovi....75
8.1 Alokacija niza s operacijom New....77
8.1.1 Zadaci za vježbu....79
9 Unije....85
9.1 Definicija unije....85
9.2 Deklaracija unije....86
9.2 Pristup članovima unije....86
9.3 Razlika između unije i strukture....86
10 Enumeracija....91
11 Rad s datotekama....94
11.1 Standardna biblioteka-fstream....94
11.2 Pristup datotekama....95
11.3 Otvaranje datoteke za pisanje....95
11.3.1 Otvaranje pomoću funkcije open....95
11.4 Otvaranje datoteka za čitanje....97
11.5 Zatvaranje datoteka....98
11.6 Upis u datoteku....98
11.7 Čitanje iz datoteka....100

1.Funkcije

U ovom dijelu će biti objašnjeno:

- Pisanje funkcija
- Prosljeđivanje podataka funkcijama
- Imenovanje funkcija sa različitim argumentima
- Izrada prototipova funkcije
- Rad sa uključenim datotekama

Programi pisani na kursu Uvod u programiranje bili su dovoljno mali da se lako mogu čitati kao jedna cjelina. Veći, realni programi mogu ima hiljadu (ili milion!) linija koda. Programeri „razbiju“ te čudovišne programe u manje dijelove kako bi ih lakše pregledavali, ravijali i održavali. C ++ dozvoljava programerima da dijele svoj kôd u upravo takve komade (poznate kao funkcije).

Funkcija je skup izjava koja zajedno obavljaju zadatak. Svaki C++ program ima barem jednu funkciju, tj. Main(), i sve najneobičniji programi mogu definisati dodatne funkcije.

Kôd se može podijeliti u zasebne funkcije. Kako podijeliti kôd između različitih funkcija zavisi od programera, ali obično je podjela tako da svaka funkcija obavlja određeni zadatak.

Deklaracija funkcije govori kompajleru o nazivu funkcije, vrsti povratne vrijednosti i i tipovima parametara koje će funkcija primiti.

Definicija funkcije daje stvarno tijelo funkcije.

Standardna biblioteka C ++ pruža brojne ugrađene funkcije koje vaš program može pozvati. Na primjer, funkcija **strcat()** služi za povezivanje dva stringa, funkcija **memcpy()** za kopiranje jednog memorijskog mjesta na drugo mjesto i još mnogo više funkcija.

Funkcija je poznata i pod imenima kao što su metoda ili podrutina ili procedura itd.

1.1 Definicija funkcije

Opšti oblik definicije funkcije C ++ je sljedeći:

```
Tip povratne vrijednosti ime_funkcije (lista parametara)
{
     Tijelo funkcije
}
```

Definicija funkcije C ++ sastoji se od zaglavlja funkcija i funkcijskog tijela. Dijelovi funkcije su:

- **Tip povratne vrijednosti** – Funkcija može vratiti vrijednost. Tip povratne vrijednosti je tip podatka koji će funkcija vratiti. Neke funkcije izvršavaju željene operacije bez vraćanja vrijednosti. U ovom slučaju, tip povratne vrijednosti je void.
- **Ime funkcije** - Ime funkcije može biti bilo koji validan identifikator u C++.
- **Parametri** - Parametar je kao *placeholder*. Kada se poziva neka funkcija, vrijednost se prenosi parametru. Ova vrijednost se naziva stvarni parametar ili argument. Lista parametara odnosi se na vrstu, red i broj parametara funkcije. Parametri su opcioni, to jest, funkcija može da ne sadrži parametre.
- **Tijelo funkcije** – Tijelo funkcije sadrži kod koji definiše šta funkcija radi.

Primjer:

Slijedeći izvorni kod je za funkciju koja se zove **max().** Ova funkcija prima dva parametra broj1 i broj2, a zatim vraća najveći od ta dva:

```
int max(int broj1, int broj2)
{
 // deklaracija lokalne varijable
 int rezultat;
 if (broj1 > broj2)
 rezultat = broj1;
 else
 rezultat = broj2;
 return rezultat;
}
```

1.2 Deklaracija (prototip) funkcije

Deklaracija funkcije govori kompajleru o nazivu funkcije i načinu poziva funkcije. Stvarno tijelo funkcije može se definisati odvojeno.

Deklaracija funkcije ima sljedeće dijelove:

```
Tip povratne vrijednosti ime_funkcije( tip parametara );
```

Za gornju definisanu funkciju **max()**, sljedeća je deklaracija funkcije:

```
int max(int, int);
```

Imena parametara nisu važna u deklaraciji funkcije samo njihov tip.

Deklaracija funkcije potrebna je kada korisnik definiše funkciju u jednoj izvornoj datoteci i tu funkciju poziva u drugoj datoteci. U tom slučaju, potrebno je deklaristi funkciju na vrhu datoteke koja zove funkciju.

1.3 Poziv funkcije

Prilikom stvaranja funkcije C + +, dajete definiciju funkcije sta će funkcija raditi. Da bi se funkcija koristila, mora se pozvati. Kada program zove funkciju, kontrola programa prenosi se na pozvanu funkciju. Pozvana funkcija obavlja definisani zadatak i kada se radi o povratnoj funkciji koja vraća vrijednost ili kada funkcija ispisuje vrijednost.

Da biste pozvali funkciju, jednostavno trebate prenijeti potrebne parametre i naziv funkcije, a ako funkcija vraća vrijednost, morate ispisati vraćenu vrijednost.

Primjer:

```
#include <iostream>
using namespace std;
// deklaracija funkcije
int max(int, int );
int main()
{
      // definisanje varijabli:
      int a = 100;
      int b = 200;
      int rez;
      // poziv funkcije da vrati najveći broj.
      rez = max(a, b);
      cout << "Najveći broj je: " << rez << endl;
      return 0;
}
// funkcija vraća najveći od dva broja
int max(int broj1, int broj2)
{
      // definisanje varijable
      int rezultat;
```

```
    if (broj1 > broj2)
        rezultat = broj1;
    else
        rezultat = broj2;
    return rezultat;
}
```

1.4 Argumenti funkcije

Ako funkcija upotrebljava argumente, onda se moraju deklarisati varijable koje prihvaćaju vrijednosti argumenata. Ove se varijable nazivaju formalnim parametrima funkcije. Formalni parametri se ponašaju kao i druge lokalne varijable unutar funkcije i stvorene su nakon ulaska u funkciju i uništene na izlazu. Prilikom poziva funkcije, postoji više načina na koje se argumenti mogu proslijediti funkciji:

Način poziva	Opis
Poziv po vrijednosti	Ova metoda kopira stvarnu vrijednost argumenta u formalni parametar funkcije. U tom slučaju, promjene na parametru unutar funkcije nemaju utjecaja na argument.
Poziv po pokazivaču	Ova metoda kopira adresu argumenta u formalni parametar. Unutar funkcije, adresa se koristi za pristup stvarnom argumentu koji se koristi u pozivu. To znači da promjene na parametaru utječu na argument.
Poziv po referenci	Ova metoda kopira referencu argumenata u formalni parametar. Unutar funkcije, referenca se koristi za pristup stvarnom argumentu koji se koristi u pozivu. To znači da promjene na parametaru utječu na argument.

1.4.1 Poziv po vrijednosti

Poziv po vrijednosti je metoda koja kopira stvarnu vrijednost argumenta u formalni parametar funkcije. U ovom slučaju, promjene na parametru unutar funkcije nemaju utjecaja na argument. Po defaultu,, C ++ koristi poziv po vrijednosti da prenese argumente. Generalno, ovo znači da kod unutar funkcije ne može da izmjeni argumente koji se koriste za pozivanje funkcije. Pogledati definiciju funkcije **swap()** koja slijedi ispod:

```
// definicija funkcije za zamjenu vrijednosti.

void swap(int x, int y)

{

 int temp;

 temp = x; /* cuva vrijednost x */

 x = y; /* stavlja vrijednost y u x */

 y = temp; /* stavlja vrijednost x u y */

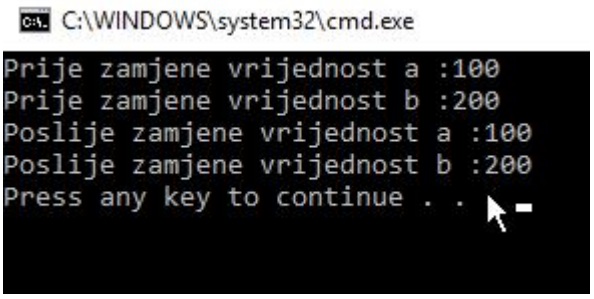
}
```

Sada, pozovimo funkciju **swap()** tako što ćemo prenijeti stvarne vrijednosti kao u nastavku:

```
#include <iostream>
using namespace std;
// deklaracija funkcije
void swap(int, int );
int main()
{
      //definisanje varijabli:
      int a = 100;
      int b = 200;
      cout << "Prije zamjene vrijednost a :" << a << endl;
      cout << "Prije zamjene vrijednost b :" << b << endl;
      // poziv funkcije da zamjeni vrijednosti
      swap(a, b);
      cout << "Poslije zamjene vrijednost a :" << a << endl;
      cout << "Poslije zamjene vrijednost b :" << b << endl;
      return 0;
}
```

```
// definicija funkcije za zamjenu vrijednosti.
void swap(int x, int y)
{
      int temp;
      temp = x; /* cuva vrijednost x */
      x = y; /* stavlja vrijednost y u x */
      y = temp; /* stavlja vrijednost x u y */
}
```

Kada se gore navedeni kod kompajlira i pokrene, daje sljedeći rezultat:



Što pokazuje da vrijednosti uopće nisu promijenjene iako su bile promijenjene unutar funkcije.

1.4.2 Poziv po pokazivaču

Poziv po pokazivaču je metoda kopira adresu argumenta u formalni parametar. Unutar funkcije, adresa se koristi za pristup stvarnom argumentu koji se koristi u pozivu. To znači da promjene na parametaru utječu na argument.

Da biste proslijedili vrijednost pomoću pokazivača, pokazivači argumenta prosljeđuju se funkcijama kao i svaka druga vrijednost. Prema tome, morate deklarisati parametre funkcije kao pokazivače, kao što je prikazano u sljedećem primjeru za funkciju **swap()** koja mijenja vrijednosti dvije cjelobrojne varijable pokazujući na njene argumente.

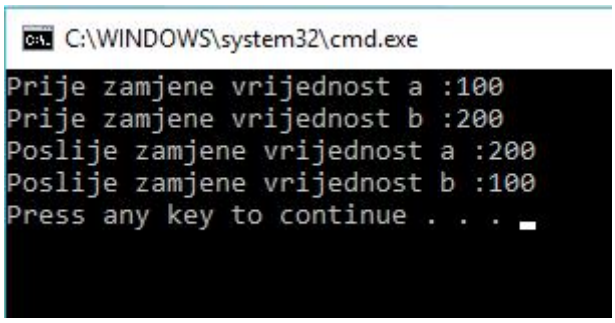
```
//definicija funkcije za zamjenu vrijednosti
      void swap(int *x, int *y)
      {
            int temp;
            temp = *x; //spašava vrijednosti na adresu x
            *x = *y; // stavlja vrijednost y u x
            *y = temp; //stavlja vrijednost x u y
      }
```

Detaljno o C ++ pokazivačima, biće objašnjeno u poglavlju C ++ Pokazivači. Za sada, pozovimo funkciju **swap()** tako što ćemo proslijediti vrijednosti po pokazivaču kao u sljedećem primjeru:

```
#include <iostream>
using namespace std;
// deklaracija funkcije
void swap(int *, int *);
int main()
{
      //definisanje loklanih varijabli
      int a = 100;
      int b = 200;
      cout << "Prije zamjene vrijednost a :" << a << endl;
      cout << "Prije zamjene vrijednost b :" << b << endl;
      /* poziv funkcije da zamjeni vrijednosti.
      * &a pokazuje pokazivač na adresu varijable a i
      * &b pokazuje pokazivač na adresu varijable b.
      */
      swap(&a, &b);
      cout << "Poslije zamjene vrijednost a :" << a << endl;
      cout << "Poslije zamjene vrijednost b :" << b << endl;
}
//definicija funkcije za zamjenu vrijednosti
      void swap(int *x, int *y)
      {
            int temp;
            temp = *x; //spašava vrijednosti na adresu x
            *x = *y; // stavlja vrijednost y u x
            *y = temp; //stavlja vrijednost x u y

      }
```

Rezultat izvršenja gore navedenog koda je:



1.4.3 Poziv po referenci

Ova metoda kopira referencu argumenata u formalni parametar. Unutar funkcije, referenca se koristi za pristup stvarnom argumentu koji se koristi u pozivu. To znači da promjene na parametaru utječu na argument.

Da biste vrijednost proslijedili po referenci, referencu argumenta prosljeđujete funkcijama kao i svaka druga vrijednost. Prema tome, morate deklarisati parametre funkcije kao referencu, kao što je prikazano u sljedećem primjeru za funkciju **swap()** koja mijenja vrijednosti dvije cjelobrojne varijable pokazujući na njene argumente.

```
//definicija funkcije za zamjenu vrijednosti
    void swap(int &x, int &y)
    {
        int temp;
        temp = x; //spašava vrijednosti na adresu x
        x = y; // stavlja vrijednost y u x
        y = temp; //stavlja vrijednost x u y
    }
```

Za sada, pozovimo funkciju **swap()** tako što ćemo proslijediti vrijednosti po referenci kao u sljedećem primjeru:

```
#include <iostream>
using namespace std;
// deklaracija funkcije
void swap(int &, int &);
int main()
{
```

```
      //definisanje loklanih varijabli
      int a = 100;
      int b = 200;
      cout << "Prije zamjene vrijednost a :" << a << endl;
      cout << "Prije zamjene vrijednost b :" << b << endl;
      //poziv funkcije da zamjeni vrijednosti upotrebom referenci.
            swap(a, b);
      cout << "Poslije zamjene vrijednost a :" << a << endl;
      cout << "Poslije zamjene vrijednost b :" << b << endl;
}
//definicija funkcije za zamjenu vrijednosti
      void swap(int &x, int &y)
      {
            int temp;
            temp = x; //spašava vrijednosti na adresu x
            x = y; // stavlja vrijednost y u x
            y = temp; //stavlja vrijednost x u y

      }
```

Prema zadanim postavkama, C++ koristi poziv prema vrijednosti za prosljeđivanje argumenata funkciji. Općenito, to znači da kod unutar funkcije ne može mijenjati argumente koji se koriste za pozivanje funkcije i iznad navedenog primjera.

1.4.4 Zadane vrijednosti za parametre

Kada definišete funkciju, možete odrediti zadanu vrijednost za svaki od posljednjih parametara. Ova vrijednost će se koristiti ako odgovarajući argument ostane prazan kada pozivate funkciju. Ovo se radi pomoću operatora dodjele i određivanja vrijednosti argumenata u definiciji funkcije. Ako se vrijednost za taj parametar ne proslijedi kada se funkcija pozove, koristi se zadana vrijednost, ali ako je vrijednost određena, ova zadana vrijednost se zanemaruje i umjesto toga se koristi proslijeđena vrijednost.

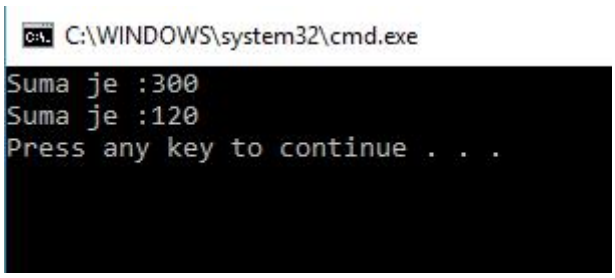
Primjer:

```
#include <iostream>
using namespace std;
int sum(int a, int b = 20)
{
```

```
	int rezultat;
	rezultat = a + b;

	return rezultat;
}
int main()
{
	// deklaracija varijabli
	int a = 100;
	int b = 200;
	int rez;
	// poziv funkcije.
	rez= sum(a, b);
	cout << "Suma je :" << rez << endl;
	// ponovno pozivanje funkcije na sljedeci nacin
	rez = sum(a);
	cout << "Suma je :" << rez << endl;
	return 0;
}
```

Kada se kod iznad kompajlira i pokrene rezultat izvršenja je sljedeći:



1.5 Zadaci za vježbu

1. Koristeći funkciju napraviti program koji izračunava N!?

Rješenje

```
#include <iostream>
using namespace std;
int factorial(int n)
{
```

```
    int res = 1;
    for (int i = 2; i <= n; i++)
        res *= i;
    return res;
}
int main()
{
    int n = 4;
    int f = factorial(n); //f = 24
    cout << f << endl; //ispis '24' (1*2*3*4)
    return 0;
}
```

2. Napisati program upotrebom funkcije za traženje najmanjeg od 3 učitana broja. Zatim napisatii glavni program koji će pozvati napisanu funkciju i ispisati njezino rješenje.

Rješenje

```
#include <iostream>
#include <cmath>
using namespace std;
int najmanji(int x, int y, int z)   /*funkcija za određivanje
najmanjeg*/
{
    int min;
    min = x;
    if (y<min)
        min = y;
    if (z<min)
        min = z;
    return min;
}

int main()
{
    int p, a, b, c;
    cout << " Unesi brojeve (npr. 5 6 8)==>"; /*upis brojeva*/
    cin >> a >> b >> c;
```

```
    p = najmanji(a, b, c);                                  /*poziv
funkcije*/
    cout << " Najmanji je : " << p << "\n";
    system("PAUSE");
    return 0;
}
```

3. Napravite program koji će od korisnika zahtijevati unos dva broja, m za početak niza i n za kraj niza. Funkcija main treba pozvati funkciju f1 za svaki cijeli broj iz tog niza [m, n]. Funkcija f1 treba provjeriti da li je broj koji ona prima kvadrat nekog broja, tj. da li korijen tog broja cijeli broj. Samo ako jeste, funkcija treba broj ispisati na ekran.

Pomoć:

- u funkciji f1 treba provjeriti da li je u1 pozitivan broj
- da bi provjerili da li je korijen broja u1 cijeli broj moramo učiniti sljedeće:
    - vrijednost korijena od u1 ćemo smjestiti u varijablu korijen_f koja je tipa float
    - vrijednost korijena od u1 ćemo smjestiti u varijablu korijen_i koja je tipa int
    - pošto je varijabla korijen_i tipa int, decimalni dio (ako postoji) će se zanemariti
    - ako je korijen_f jednako korijen_i, što znači je vrijednost korijen_f cijeli broj, ispisat ćemo na ekran broj u1

```
#include <iostream>

#include <cmath>

using namespace std;

void f1(int); // prototip (deklaracija) funkcije

void main()

{

    int m, n;

    cout << "Unesi pocetak i kraj niza: \n";

    cin >> m >> n;

    for (int i = m; i <= n; i++)

    {

        f1(i);
```

```
	}
}
void f1(int u1)
{
	if (u1 >= 0)
	{
		float korijen_f;
		korijen_f = sqrt(float(u1));
		int korijen_i;
		korijen_i = korijen_f; // korijen_i će zanemariti decimalni dio od korijen_f
		if (korijen_i == korijen_f)
			cout << u1 << " = " << korijen_i << " * " << korijen_i << endl;
	}
}
```

4. Napraviti program koji će od korisnika zahtjevati unos pet brojeva (a,b,c,d,e). Potrebno je izračunati:

$Y_1=\sqrt{a}$	uslov a≥0
$Y_2=\log_{10}b$	uslov b>0
$Y_3=\log_e b=\ln B$	uslov c>0
$Y_4=\cos(c)$	vrijednsot c je u radijanima
$Y_5=\text{sinc}(c)$	
$Y_6=d^3$	ako je d=0, onda e mora biti različito od 0, jer je 0^0 nedefinisano.

Rješenje

```
#include <iostream>
#include <cmath>
using namespace std;
void main()
{
	float a, b, c, d, e, y1, y2, y3, y4, y5, y6;
	cout << "Unesi broj a koji nije negativan: ";
	cin >> a;
	cout << "Unesi broj b koji je pozitivan: ";
	cin >> b;
	cout << "Unesi ugao c u radijanima (1 rad = 90 stepeni): ";
	cin >> c;
	cout << "Unesi bazu d: ";
	cin >> d;
	cout << "Unesi eksponent e: ";
	cin >> e;
	if (a >= 0)
	{
		y1 = sqrt(a);
		cout << "y1 = " << y1 << endl;
	}
	if (b > 0)
	{
		y2 = log10(b);
		y3 = log(b);
		cout << "y2 = " << y2 << endl;
		cout << "y3 = " << y3 << endl;
	}
	y4 = cos(c);
	y5 = sin(c);
	cout << "y4 = " << y4 << endl;
	cout << "y5 = " << y5 << endl;
	if (d != 0 || e != 0)
	{
		y6 = pow(d, e);
		cout << "y6 = " << y6 << endl;
```

```
}}
```

2. Rekurzivne funkcije

U programiranju rekurzija nastaje kada funkcija poziva samu sebe direktno ili indirektno. Indirektna rekurzija nastaje kada jedna funkcija poziva drugu funkciju, iz koje se ponovo poziva pozivna funkcija.

Znači, rekurzija predstavlja sposobnost funkcije da poziva samu sebe. Tom prilikom se vodi računa da se svaki naredni poziv izvršava za jednostavniji problem od polaznog, a da se najjednostavniji problemi rješavaju direktno, tj. bez dalje upotrebe rekurzije. Značaj i primjena rekurzivnih funkcija posebno je naglašena u određenim područjima programiranja kao što je Umjetna inteligencija. Pored toga, primjena rekurzivnih funkcija nije baš rijetka ni u svakodnevnom radu. Pred programere se ponekad postavljaju zadaci koji zbog svoje veličine zahtijevaju kompleksna rješenja. Kompleksni zadaci se u većini slučajeva mogu podijeliti na više manjih cjelina. Ideja rekurzije leži upravo u tome da se, koristeći rekurzivne algoritme, problem podijeli na manje dijelove koje je jednostavnije riješiti.

Kada govorimo o rekurziji mi zapravo govorimo o stvaranju petlje.

```
for(int i=0;  i<10; i++) {

      cout << "Broj je: " << i << endl;

}
```

Ova jednostavna `for` petlja ispisuje rečenicu „Broj je“ i vrijednost varijable `i` kao što je prikazano u sljedećem primjeru:

```
Broj je : 0

Broj je: 1

Broj je: 2

Broj je: 3

Broj je: 4

Broj je: 5

Broj je: 6

Broj je: 7
```

```
Broj je: 8

Broj je: 9
```

Unutar `for` petlje inicijalizirali smo varijablu `i` (brojač) na početnu vrijednost 0. Prvi prikaz je „Broj je : 0“, zatim se naš brojač uvećava za jedan naredbom `i++;`. `For` petlja će ispisivati vrijednost brojača dok brojač `i` ne dosegne broj 10. `For` petlja se sastoji od tri dijela: inicijalizacija brojača, uslov, uvećanje/smanjenje brojača. Sve što je sadržano unutar zagrada {} je ono što program izvodi. Vjerovatno se pitate, kakve veze to ima sa rekurzijom? Zapamtite, rekurzija je petlja.

Sljedeći kod je identičan prethodnom samo što sad koristimo funkciju.

```
#include <iostream>
using namespace std;
void BrojFunkcija(int i)
{
	cout << "Broj je: " << i << endl;
}
int main()
{
	for (int i = 0; i<10; i++)
	{
		BrojFunkcija(i);
	}
	return 0;
}
```

Deklarirali smo `void` funkciju, što znači da ne vraća nikakvu vrijednost, a prima parametar tipa `int`. Funkcija koju smo imenovali BrojFunkcija nam ispisuje vrijednost brojača `i`. Funkcija se poziva unutar `for` petlje, a pozivati će se sve dok brojač ne dosegne vrijednost 10.

U nastavku slijedi zadatak koji ima istu funkciju, samo sada ne koristimo `for` petlju.

```
#include <iostream>
using namespace std;
void BrojFunkcija(int i) {
	cout << "Broj je: " << i << endl;
	i++;
	if (i<10) {
```

```
            BrojFunkcija(i);
      }}
int main()
{
      int i = 0;
      BrojFunkcija(i);
      return 0;
}
```

U prethodnom kodu koristili smo rekurziju. Kao što možemo vidjeti poziv funkcije `BrojFunkcija`, se vrši u `main` funkcije, ali i u samoj funkciju `BrojFunkcija`. Poziv funkcije u samo funkciji će biti svaki put dok vrijednost brojača `i` ne dosegne 10.

2.1 Implementacija rekurzije

```
void imeFunkcije (argument)//definicija funkcije koja ne
vraća vrijednost

{ Iskaz1; //iskaz koji će se izvršiti pri svakom pozivu
funkcije – uzlazni tok

if (nekiUslov){ //provjera nekog uslova

argument--; //dekrementiranje vrijednosti

imeFunkcije(argument); //ponovni poziv funkcije --
REKURZIJA

} //kraj if-a

Iskaz2; //iskaz koji će se izvršiti kada uslov ne bude
ispunjen – silazni tok

} //kraj funkcije
```

Bitno je spomenuti da rekurzivne funkcije posjeduju nekoliko osobina o kojima treba voditi računa. Jedna od osobina rekurzivnih funkcija se odnosi na mogućnost da se nepažnjom dobije nekontrolisana ili beskonačna rekurzija. Najčešći uzrok beskonačne rekurzije je nepostojanje ili nepravilno definisanje tzv. baznog slučaja. Koristeći bazni slučaj programer je u stanju da, u zavisnosti od toga šta želi postići rekurzijom, u određenom momentu onemogući ponovno

pozivanje funkcije, odnosno uzrokuje prekid rekurzije. Upravo zbog toga, za efikasno funkcionisanje, svaka rekurzivna funkcija zahtijeva postojanje najmanje jednog baznog slučaja.

U nastavku je prikazan primjer izračunavanja sume kvadrata cijelih brojeva u rasponu od n do m – iteracijom i rekurzijom.

Iteracija

```
int SumaKvadrata(int m, int n)
{
int suma = 0;
for(int i = m; i <= n; i++)
suma += i*i;
return suma;
}
```

Rekurzija

```
int SumaKvadrata(int m, int n)
{
if(m> n)
return 0;
return m*m + SumaKvadrata(m+1, n);
}
```

2.2 Zadaci za vježbu

1. Koristeći rekurziju, napisati program koji od korisnika traži da unese broj čija vrijednost mora biti u opsegu od 1 do 10. Unesenu vrijednost predati funkciji, koja treba da se izvršava onoliko puta kolika je vrijednost predanog argumenta (ako korisnik unese broj 5, funkcija treba da 5 puta pozove sama sebe). Pri svakom pozivu, funkcija treba da ispiše trenutnu vrijednost predanog argumenta.

Rješenje

```
#include <iostream>
using namespace std;
void rekFunkcija(int broj) //zaglavlje rekurzivne funkcije
{
        if (broj>0){             //provjera uslova da li je broj veci od 0
            cout<<"Trenutna vrijednost varijable broj je : "<<broj<<endl;
            broj--;          //dekrementiranje vrijednosti argumenta broj
            rekFunkcija(broj); //ponovni poziv funkcije - rekurzija
}              //kraj if-a
}//kraj definicije funkcije
void main()
{
    int broj = 0;
    do{
        cout << "Unesite broj od 1 do 10: ";
        cin >> broj;
        //onemogucavamo unos brojeva vecih od 10 ili manjih od 1
    } while (broj>10 || broj<1);
    rekFunkcija(broj); //poziv funkcije

}
```

Kao što se može vidjeti, unutar main funkcije je deklarisana varijabla cjelobrojnog tipa (varijabla broj) koja služi za čuvanje vrijednosti koju je korisnik unio. Odmah nakon validnog unosa (1-10), vrijednost varijable broj se kao argument predaje funkciji rekFunkcija. Na početku definicije funkcije rekFunkcija se provjera da li je vrijednost primljenog argumenta veća od nula (0). Ukoliko je uslov ispunjen, trenutna vrijednost argumenta se ispisuje, a nakon toga dekrementira (broj--). Neposredno nakon dekrementiranja vrši se ponovni poziv funkcije, te se na taj način implementira rekurzija. Ono što je bitno primijetiti je da se pri ponovnom pozivu funkcije kao argument predaje umanjena vrijednost argumenta broj, a ne vrijednost koju je korisnik proslijedio iz `main` funkcije. Dekrementiranjem vrijednosti primljenog argumenta se izbjegava mogućnost pojave beskonačne ili nekontrolisane rekurzije. U slučaju da prilikom sljedećeg poziva funkcije uslov (broj>0) ne bude ispunjen, tj. ukoliko vrijednost argumenta broj nije veća od nula (0), neće doći do ponovnog poziva funkcije čime je onemogućena dalja rekurzija.

2. Koristeći rekurziju, napisati program koji korisniku omogućava da unese željeni tekst. Uslov za završetak programa je da uneseni tekst sadrži najmanje jedan uzvičnik (!). Nakon unosa, program treba obrnutim redoslijedom ispisati sve znakove koji su uneseni do prvog znaka tačke.

Rješenje

```
#include <iostream>
using namespace std;
void prikaziUnazad()
{ char znak;
cin >> znak;
if (znak != '!')
{
    prikaziUnazad();
    cout << znak;
}
else
cout << "Obrnutim redosljedom ste unijeli: ";
}
void main()
{
    cout << "Unesite vas tekst: ";
    prikaziUnazad();
}
```

Prethodni program počinje ispisivanjem poruke kojom se od korisnika traži da unese željeni tekst, nakon čega se poziva funkcija `prikaziUnazad`. Pomenuta poruka je ispisana u `main` funkciji sa osnovnim ciljem da se prilikom rekurzije onemogući višestruko ponavljanje poruke.

Unutar funkcije `prikaziUnazad` deklarisana je lokalna varijabla znak koja će prilikom svake rekurzije čuvati jedan od karaktera koje je unio korisnik. Ovo možda izgleda malo zbunjujuće, pa je zbog toga bitno da razumijete način funkcionisanja input buffera. Naime, input buffer predstavlja dio memorije koji služi za čuvanje ulaznih vrijednosti sa tastature.

Uz pretpostavku da je korisnik unio karaktere „PROGRAMIRANJE!“ i pritisnuo tipku Enter, sadržaj buffera će biti sljedeći: **P R O G R A M I R A N J E !**

Kada korisnik pritisne tipku `Enter`, komanda `cin` (govorimo o prvom izvršenju funkcije `prikaziUnazad`) će iz buffera preuzeti prvi karakter (u našem slučaju slovo **P** jer se u varijablu tipa `char` može pohraniti samo jedan znak) i pohraniti ga u varijablu `znak`. Odmah nakon toga se vrši provjera uslova tj. da li je `znak` (ne)jednak znaku UZVIČNIK (**!**). Ako je uslov ispunjen (ako `znak` nije jednak znaku (UZVIČNIK) funkcija se poziva ponovo. Prilikom sljedećeg poziva funkcije sadržaj input buffera je sljedeći (pošto je prethodna funkcija iskoristila slovo P):

R O G R A M I R A N J E !

Na početku izvršenja sljedeće funkcije, komanda `cin` će preuzeti naredni karakter (slovo **O**) i pohraniti ga u varijablu `znak`. Ovdje se nameće pitanje: zašto komanda `cin`, prilikom sljedećeg izvršenja funkcije, nije tražila ponovni unos sa tastature? Odgovor na prethodno pitanje leži u činjenici da komanda `cin` neće zahtijevati interakciju sa korisnikom (misli se na unos) sve dok u input bufferu postoji bilo kakav sadržaj. Pošto je u našem slučaju sadržaj input buffera „ROGRAMIRANJE!“, komanda `cin` preuzima sljedeći znak tj. slovo **R**.

Izvršenje programa se na taj način nastavlja sve do momenta dok varijabla `znak` ne bude inicijalizirana vrijednošću karaktera UZVIČNIK (**!**). U tom momentu, ponovni poziv funkcije je onemogućen i rekurzija se završava.

3. Brojevi

Obično, kada radimo sa brojevima, koristimo primitivne tipove podataka kao što su **int, short, long, float i double**, itd. Tipovi podataka, njihove moguće vrijednosti i opseg brojeva detaljno su obrađeni na kursu Uvod u programiranje.

3.1 Definisanje brojeva u C++

U nastavku je dat primjer za definiranje različitih vrsta brojeva u C++:

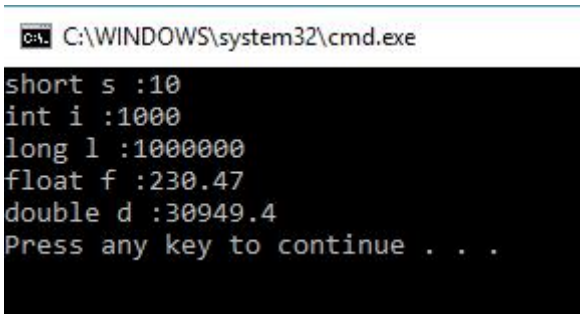
```
#include <iostream>
using namespace std;
int main()
{
	// definisanje varijabli za brojeve:
	short s;
	int i;
	long l;
	float f;
	double d;

	// dodjela brojeva definisanim varijablama;
	s = 10;
	i = 1000;
	l = 1000000;
	f = 230.47;
	d = 30949.374;

	// ispis brojeva;
	cout << "short s :" << s << endl;
	cout << "int i :" << i << endl;
	cout << "long l :" << l << endl;
	cout << "float f :" << f << endl;
	cout << "double d :" << d << endl;
	return 0;

}
```

Kada se kod iznad kompajlira i pokrene rezultat izvršenja je sljedeći:



3.2 Matematičke operacije u C++

Pored različitih funkcija koje možete kreirati, C++ takođe posjeduje i neke korisne funkcije koje možete koristiti. Ove funkcije su dostupne u standardnim C i C++ bibliotekama i nazivaju se ugrađenim funkcijama. To su funkcije koje mogu biti uključene u vaš program, a zatim a nakon toga korištene u istom.

C++ ima bogat skup matematičkih operacija, koje se mogu izvoditi na različitim brojevima. Sljedeća tabela navodi niz korisnih ugrađenih matematičkih funkcija koje su dostupne u C++. Za korištenje ovih funkcija potrebno je uključiti datoteku zaglavlja **<cmath>**.

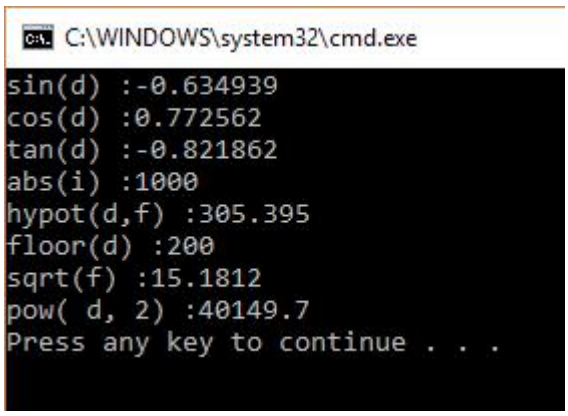
Redni broj	Funkcija i svrha
1	**double cos (double);** Ova funkcija prima ugao (kao double vrijednost) i vraća kosinus.
2	**double sin (double);** Ova funkcija prima ugao (kao double vrijednost) i vraća sinus.
3	**double tan (double);** Ova funkcija prima ugao (kao double vrijednost) i vraća tangent.
4	**double log (double);** Ova funkcija prima broj i vraća prirodni zapis tog broja..
5	**double pow (double, double);** Ova funkcija prima prvi broj koji želite stepenovati, i drugi broj na koji ćete stepenovati prvi broj.
6	**double hypot (double, double);** Ako ovoj funkciji predate dužine dvije stranice pravouglog trogula, rezultat je dužina hipotenuze.
7	**double fabs(double);** Ova funkcija vraća absolutnu vrijednost bilo kojeg decimalnog broja.
8	**double floor(double);** Pronalazi cijeli broj koji je manji ili jednak broju koji mu se dodjeljuje.

Slijedi jednostavan primjer za prikazivanje nekoliko matematičkih operacija:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
      // deklaracija brojeva:
      short s = 10;
      int i = -1000;
      long l = 100000;
      float f = 230.47;
      double d = 200.374;
      // mmatematicke operacije;
      cout << "sin(d) :" << sin(d) << endl;
      cout << "cos(d) :" << cos(d) << endl;
```

```
    cout << "tan(d) :" << tan(d) << endl;
    cout << "abs(i) :" << abs(i) << endl;
    cout << "hypot(d,f) :" << hypot(d,f) << endl;
    cout << "floor(d) :" << floor(d) << endl;
    cout << "sqrt(f) :" << sqrt(f) << endl;
    cout << "pow( d, 2) :" << pow(d, 2) << endl;
    return 0;
}
```

Kada se kod iznad kompajlira i pokrene rezultat izvršenja je sljedeći:



3.3 Nasumični brojevi u C++

Postoje mnogi slučajevi u kojima želite generisati slučajni broj. Postoje zapravo dvije funkcije koje trebate znati o stvaranju slučajnih brojeva. Prva je **rand()** funkcija, koja će vratiti samo pseudo nasumičan broj. Način da se to riješi je da najprije zovete funkciju **srand()**.

Da bi ste koristili biranje slučajnih brojeva neophodno je da uključite i **<cstdlib> i <ctime>** biblioteke zaglavlja u vaš projekat. Slučajno izabrani cijeli broj se vraća preko funkcije **rand().** Međutim prvo mora da se pozove funkcija **srand()** koji vraća raspon od 0 do RAND_MAX broja, tako što uzima broj tipa unsigned int koji se poziva u funkciji **rand().** Pozivanjem funkcije **time(NULL)** proslijeđuje se sistemsko vrijeme kao početna vrijednost. Sve ovo vam vjerovatno zvuči komplikovano i nejasno, ali ako pogledate i analizirate kod sljedećeg programa, možete uočiti da biranje slučajnih brojeva i nije toliko komplikovano kao što zvuči:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
```

```
{
	int menu, random;
		do
	{
		cout << endl << "\tLOTTO" << endl << endl;
		cout << endl << "\t\t1. Random numbers" << endl;
		cout << "\t\t0. Exit" << endl << endl;
		cout << "Enter 0 for exit or 1 for random lotto numbers: ";
		cin >> menu;
		cout << endl;
		srand((unsigned)time(0));

		for (int i = 0; i < 7; i++)
		{
			random = (rand() % 39) + 1;
			cout << random << "\t";

		}

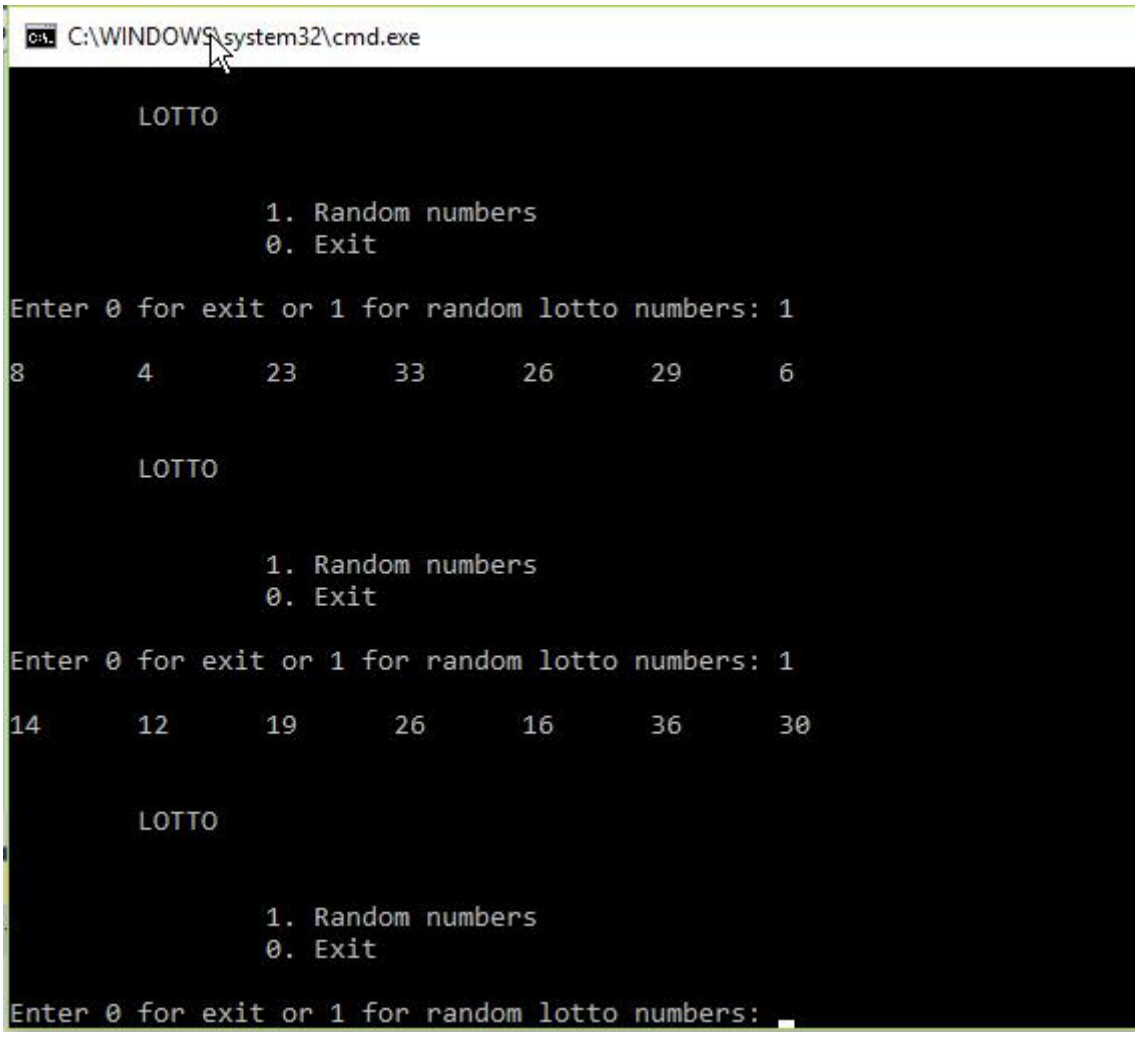
		cout << endl << endl;

	} while (menu != 0);

	return 0;

}
```

Kad pokrenete program, rezultat će izgledati slično:



3.4 Zadaci za vježbu

Napisati program koji će omogućiti korisniku unijeti koordinate tačaka A i B, pa izračunati njihovu udaljenost u koordinatnom sistemu. Ispis neka bude oblika:

Koordinate tacke A :
x1=...
y1= ...
Koordinate tacke B :
x2=...
y2=...
Udaljenost tacaka A(...,...) i B(...,...) je ...

Udaljenost tačaka A(x1,y1) i B(x2,Y2) u koordinatnom sistavu računa se formulom:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
#include<iostream>
#include<cmath>
using namespace std;
int main()

{

	float x1, y1, x2, y2, pom, d;
	cout << "Koordinate tacke A:" << endl;
	cout << "x1= ";
	cin >> x1;
	cout << "y1= ";
	cin >> y1;
	cout << "Koordinate tacke B:" << endl;
	cout << "x2= ";
	cin >> x2;
	cout << "y2= ";
	cin >> y2;
	pom = pow((x2 - x1), 2) + pow((y2 - y1), 2);
	d = sqrt(pom);
	cout << "Udaljenost tacaka A(" << x1 << "," << y1 << ") i B(" <<
x2 << "," << y2 << ") je " << d;
	return 0;
}
```

4. Dvodimenzionalni statički nizovi

Dvodimenzionalni niz je, u suštini, lista jednodimenzionalnih nizova. Dvodimenzionalni niz se može opisati kao niz čiji su elementi nizovi. Za deklaraciju niza cijelih brojeva x,y potrebno je napisati

```
Tip podataka ime_niza [x][y];
```

Gdje tip podatka može biti bilo koji validan tip podatka u C++, a ime niza bilo koji validan identifikator u C++.

Dvodimenzionalni niz može se smatrati kao tabela, koja će imati x broj redova i y broj kolona. Dvodimenzionalni niz **a**, koji sadrži ima reda i četiri kolone, može se prikazati:

	Kolona 0	Kolona 1	Kolona 2	Kolona 3
Red 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Red 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Red 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Članovima (elementima) dvodimenzionalnog niza pristupa se preko dva indeksa: prvi je određen redm, a drugi kolonom u kojem se podatak nalazi. Početni indeks i za redak i za stupac je nula (0). Dakle, svaki element u nizu **a** je identifikovan elementom oblika **a [i] [j],** gdje **a** je naziv niza, a **i** i **j** su indeksi koji jedinstveno identifikuju svaki element u nizu **a**.

Dvodimenzionalni niz se naziva još i tablicom, matricom ili dvostruko indeksirani (double-subscripted) niz.

4.1 Inicijalizacija dvodimenzionalnih nizova

Višedimenzionalni nizovi mogu biti inicijalizirani navođenjem vrijednosti u vitičaste zagrade za svaki red.

U nastavku je prikazan niz sa 3 reda, a svaki red ima 4 kolone.

```
int a[3][4] = {
          { 0, 1, 2, 3 },/* clanovi niza za red sa indeksom 0 */
          { 4, 5, 6, 7 },/* clanovi niza za red sa indeksom  1 */
          { 8, 9, 10, 11}/* clanovi niza za red sa indeksom  2 */
               }
```

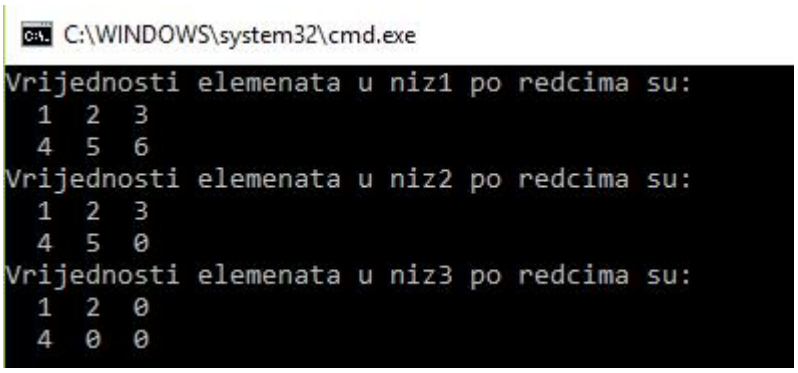
Vitičaste zagrade, koje označavaju željeni red, nisu obavezne. Sljedeća inicijalizacija je ekvivalentna prethodnom primjeru:

```
int a[3][4] = { 0,1,2,3,4,5,6,7,8,9,10,11};
```

Primjer: Naredni program deklarira i inicijalizira tri niza od kojih svaki ima dva reda i tri kolone.

```
// Inicijalizacija elemenata dvodimenzionalnog niza
#include <iostream>
#include <iomanip>
using namespace std;
const int red = 2, kolona = 3;
void ispisi_niz(int[][kolona]); //prototip funkcije koja ispisuje
elemente niza, obavezno navodjenje velicine druge dimenzije
int main()
{
     int niz1[red][kolona] = { { 1,2,3 },{ 4,5,6 } },
//inicijalizacija niza pri deklaraciji
          niz2[red][kolona] = { 1,2,3,4,5 },
          niz3[red][kolona] = { { 1,2 },{ 4 } };
     cout << "Vrijednosti elemenata u niz1 po redcima su:" << endl;
     ispisi_niz(niz1); //poziv funkcije, argument je ime niza (ime
niza je adresa prvog elementa niza)
     cout << "Vrijednosti elemenata u niz2 po redcima su:" << endl;
     ispisi_niz(niz2);
     cout << "Vrijednosti elemenata u niz3 po redcima su:" << endl;
     ispisi_niz(niz3);
     system("pause>null");
     return 0;
}
void ispisi_niz(int a[][kolona]) //definicija funkcije
{
     for (int i = 0; i<red; i++)
     {
          for (int j = 0; j<kolona; j++)
               cout << setw(3) << a[i][j] << setw(3);
          cout << endl;
     }
```

}



U prethodnom primjeru, pri deklaracija niza **niz1** članovima niza su dodijeljene i vrijednosti (elementima u prvom redu dodijeljene su vrijednosti 1,2 i 3; a elementima u drugom redu vrijednosti 4,5 i 6).

Pri deklaraciji niza **niz2** inicijalizirano je 5 elemenata. Najprije su dodijeljene vrijednosti u prvom redu, a zatim u drugom. Svi elementi kojima nije eksplicitno dodijeljena vrijednost automatski se inicijaliziraju vrijednošću 0. U skladu s tim element niz2[2][3] ima vrijednost 0.

Pri deklaraciji niza **niz3** inicijalizirana je vrijednost 3 elementa, od kojih su prva dva elementa u prvom redu, a treći element je u drugom redu. Ostalim elementima niza automatski se dodjeljuje vrijednost 0.

Program poziva funkciju **ispisi_niz** kako bi se ispisale vrijednosti elemenata svakog od nizova. Funkcija ima jedan parametar: ime niza (a[][3]). Dimenzije niza su u ovom primjeru deklarirane kao globalne konstante, stoga nije bilo potrebe za njihovim prosljeđivanjem u funkciju preko parametara.

Kad smo prosljeđivali jednodimenzionalni niz u funkciju uglaste zagrade nakon imena niza su bile prazne. Kod prosljeđivanja višedimenzionalnih nizova u funkciju, također nije potrebno navoditi veličinu prvog indeksa, ali drugi indeks (indeks koji određuje drugu dimenziju) je neophodno navesti.

```
void ispisi_niz( int [][kolona] );
```

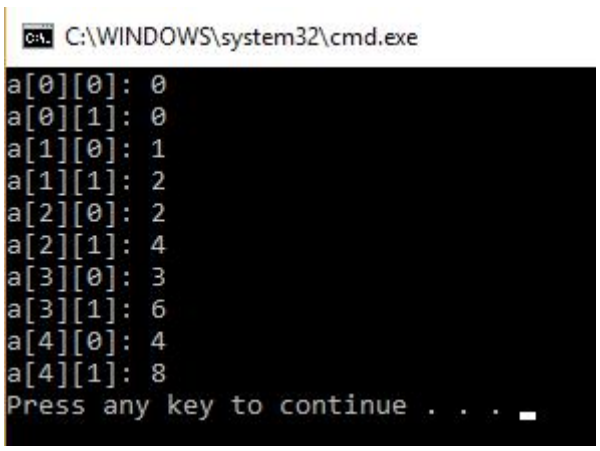
4.2 Pristup elementima dvodimenzionalnog niza

Elementu u dvodimenzionalnom nizu se pristupa korištenjem indeksa, tj. indeksa redova i indeksa kolona u nizu. Na primjer:

```
#include <iostream>
using namespace std;
```

```
int main()
{
	// inicijalizacija niza sa 5 redova i dvije kolone.
	int a[5][2] = { { 0,0 },{ 1,2 },{ 2,4 },{ 3,6 },{ 4,8 } };
	// ispis vrijednosti svakog elementa polja
	for (int i = 0; i < 5; i++)
		for (int j = 0; j < 2; j++)
		{
			cout << "a[" << i << "][" << j << "]: ";
			cout << a[i][j] << endl;
		}
	return 0;
}
```

Kada se gore navedeni kod kompajlira i pokrene, on proizvodi sljedeći rezultat:



Kao što je već objašnjeno, možete imati nizove sa bilo kojim brojem dimenzija, iako je vjerovatno da će većina nizova koje kreirate biti jednodimenzionalni ili dvodimenzionalni.

4.3. Zadaci za vježbu

1. Napisati program koji će omogučiti učitavanje i ispis elemenetata u dvodimenzionalni niz - matricu 4x3.

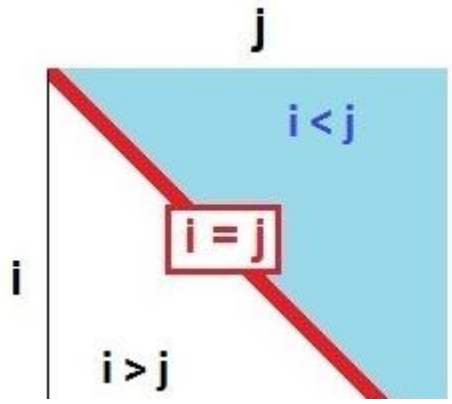
```
#include <iostream>
using namespace std;
int main()
```

```
{
	int iMatrica[4][3];
		for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 3; j++)
		{
		cout << "iMatrica [" << i << "," << j << "] ---> ";
			cin >> iMatrica[i][j];
		}
	}
	cout << endl;
	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 3; j++)
		{

			cout << iMatrica[i][j] << "   ";
		}
		cout << endl << endl;
	}
	system("PAUSE");
	return 0;}
```

2. Učitati i ispisati dvodimenzionalni niz - matricu nxn. Zatim ispisati članove niza koji su na glavnoj dijagonali.

   Opis programa: Glavna dijagonala - Elementi čiji su indeksi isti (iste indekse reda i kolone odnosno i=j) su elementi glavne dijagonale. Elementi iznad glavne dijagonale (indeks reda manji od indeksa kolone i<j), a elementi ispod glavne dijagonale (indeks reda veći od indeksa kolone i>j). Elementi a[0],[0], a[1],[1], a[2],[2], ..., a[n-1],[n-1] čine glavnu dijagonalu.



$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$

```
#include <iostream>
using namespace std;
int main()
{
	int iMatrica[4][3];
	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 3; j++)
		{
		cout << "iMatrica [" << i << "," << j << "] ---> ";
			cin >> iMatrica[i][j];
		}
	}
	cout << endl;
	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (i==j)
			cout << iMatrica[i][j] << "  ";
		}
		cout << endl << endl;
	}
	system("PAUSE");
	return 0;
}
```

3. Napisati program koji će omogućiti učitavanje elemenata u dvodimenzionalni niz 3X3. Ispisati parne članove trećeg reda.

```
#include <iostream>
using namespace std;
int main()
{

	int iMatrica[3][3];
```

```
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
	cout << "iMatrica [" << i << "," << j << "] ---> ";
			cin >> iMatrica[i][j];
		}
	}
	cout << endl;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if ((i == 2) && (iMatrica[i][j] % 2 == 0))
			{
				cout << iMatrica[i][j] << "  ";
			}
		}
		cout << endl << endl;
	}

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

5. Reference

Referenca ili referentna varijabla je pseudonim, odnosno drugo ime za već postojeću varijablu. Kada se referenca incijalizira pomoću varijable, u nastavku se može koristiti ili ime varijable ili referenca koja se odnosi na varijablu.

5.2 Reference vs pokazivači

Reference se često mijenjaju pokazivačima, ali tri glavne razlike između referenci i pokazivača su:

- Referenca ne može imati NULL vrijednost. Referenca uvijek mora uvijek biti spojena sa određenom memorijom.
- Kada se referenca inicijalizuje na objekat, ne može se mijenjati da bi se odnosila na drugi objekat. Pokazivači mogu biti usmjereni na drugi objekt u bilo kojem trenutku.
- Referenca mora biti inicijalizirana kada se stvori. Pokazivači se mogu inicijalizirati u bilo kojem trenutku.

5.3 Reference u C++

Zamislite naziv varijable kao oznaku koja je povezana sa lokacijom varijable u memoriji. Onda možete zamisliti referencu kao drugu oznaku koja se nalazi na toj memorijskoj lokaciji. Prema tome, sadržaju varijable moguće je pristupiti upotrebom imena varijable ili referencom. Na primjer,

```
int i=17;
```

ili deklarisati referencu na sljedeći način:

```
int& r=i;
```

Znak & (ampersand) se čita kao referenca. Dakle, prethodna linija koda se čita kao r je referenca cijelog broja inicijaliziranog na i, a sljedeća linija koda, s je referenca realnog broja inicijalizirana na d. "

```
double& s = d;
```

Slijedeći primjer koristi reference na int i double:

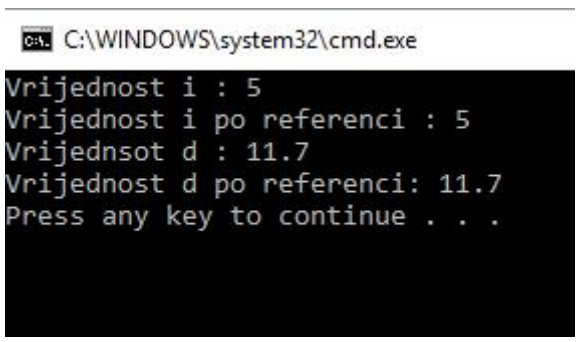
```
#include <iostream>
using namespace std;
int main()
{ // deklaracija varijabli
```

```
    int i;
    double d;
    // deklaracija referenci
    int& r = i;
    double& s = d;

    i = 5;
    cout << "Vrijednost i : " << i << endl;
    cout << "Vrijednost i po referenci : " << r << endl;
    d = 11.7;
    cout << "Vrijednsot d : " << d << endl;
    cout << "Vrijednost d po referenci: " << s << endl;

    return 0;
}
```

Kada se gore navedeni kod kompajlira i pokrene, on proizvodi sljedeći rezultat:



Reference se obično koriste za prosljeđivanje argumenata funkciji i povratne vrijednosti funkcija.

5.3.1 Reference kao parametri

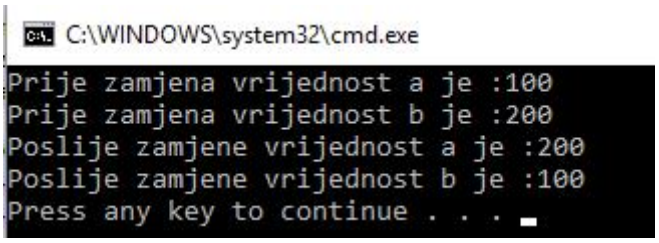
C++ podržava prosljeđivanje referenci kao parametre funkciji više nego parametri po vrijednosti. U nastavku dat primjer poziva funkcije po referenci:

```
#include <iostream>
using namespace std;
// function declaration
void zamjena(int& , int& );
int main()
{
    // deklaracija lokalni varijabli:
    int a = 100;
```

```
    int b = 200;
    cout << "Prije zamjena vrijednost a je :" << a << endl;
    cout << "Prije zamjena vrijednost b je :" << b << endl;
    /* poziv funkcije da zamjeni vrijednosti.*/
    swap(a, b);
    cout << "Poslije zamjene vrijednost a je :" << a << endl;
    cout << "Poslije zamjene vrijednost b je :" << b << endl;
    return 0;
}
// definicija funkcije
void zamjena(int& x, int& y)
{
    int temp;
    temp = x; /* privremena varijabla u koju se sprema x */
    x = y;
    y = temp;

}
```

Kada se gore navedeni kod kompajlira i pokrene, on proizvodi sljedeći rezultat:



5.3.2 Reference kao povratne vrijednosti

Program C ++ može biti lakši za čitanje i održavanje korištenjem referenci umjesto pokazivača. C ++ funkcija može vratiti referencu na sličan način kao i ona koja vraća pokazivač.

Kada funkcija vrati referencu, ona vraća implicitni pokazivač na svoju povratnu vrijednost. Na ovaj način, funkcija se može koristiti na lijevoj strani izjave o dodjeljivanju. Na primjer, razmotrite ovaj jednostavan program:

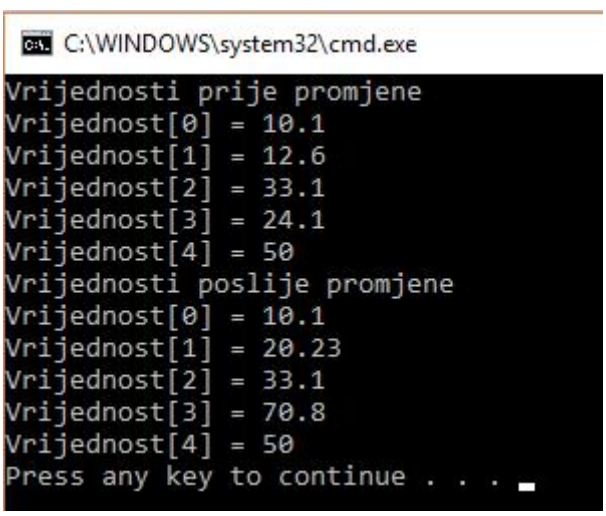
```
#include <iostream>
using namespace std;
double niz[] = { 10.1, 12.6, 33.1, 24.1, 50.0 };
double& setValues(int i)
```

```
{
      return niz[i]; // vraca referencu na zeljeni element
}

int main() {
      cout << "Vrijednosti prije promjene" << endl;
      for (int i = 0; i < 5; i++)
      {
            cout << "Vrijednost[" << i << "] = ";
            cout << niz[i] << endl;
      }
      setValues(1) = 20.23; // mijenja drugi element niza
      setValues(3) = 70.8; // mijenja cetvrti element niza
      cout << "Vrijednosti poslije promjene" << endl;
      for (int i = 0; i < 5; i++)
      {
            cout << "Vrijednost[" << i << "] = ";
            cout << niz[i] << endl;
      }
      return 0;

}
```

Kada se gore navedeni kod kompajlira i pokrene, on proizvodi sljedeći rezultat:



6. Pokazivači

C++ pokazivači su jednostavni i zabavni za učenje. Neki C++ zadaci se lakše izvršavaju pomoću pokazivača, dok drugi zadaci, poput dinamičke alokacije memorije, ne mogu se izvoditi bez njih.

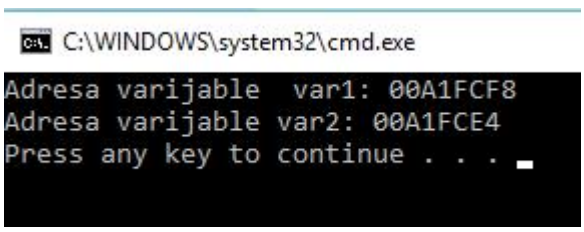
Kao što znate svaka varijabla je memorijska lokacija, a svaka memorijska lokacija ima definisanu adresu kojoj se može pristupiti pomoću ampersand (&) operatora koji označava adresu u memoriji.

Sljedeći primjer ispisuje adrese definisanih varijabli:

```
#include <iostream>
using namespace std;
int main()
{
      int var1;
      char var2[10];
      cout << "Adresa varijable  var1: ";
      cout << &var1 << endl;
      cout << "Adresa varijable var2: ";
      cout << &var2 << endl;
      return 0;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



6.1 Šta su pokazivači

Pokazivač je varijabla čija je vrijednost adresa druge varijable. Pokazivač je podatkovni objekt koja čuva memorijsku adresu. Kao i svaka varijabla ili konstanta, morate deklarisati pokazivač prije nego što ćete ga koristiti. Pokazivači se deklarišu na sljedeći način:

```
tip * identifikator;
```

gdje je:

tip - tip podatka na kojeg upućuje pokazivač;

***** - operator **dereferenciranja**. Kada se pokazivač dereferencira, dobije se vrijednost pohranjena na adresi koja je zapisana u pokazivaču.

identifikator - bilo koji validan identifikator u C++.

Poželjno je odmah pri deklaraciji inicijalizirati pokazivač ili s NULL ili na adresu nekog objekta.

```
int *p = NULL;
```

```
int*p = 0;
```

ili

```
int *p = &objekt;
```

Ukoliko je prethodno deklariran pokazivač p vrijednost pokazivača se inicijalizira na sljedeći način:

```
p = &objekt;
```

gdje je:

p – identifikator (ime) pokazivača

& - operator reference (ampersand) koji vraća adresu objekta

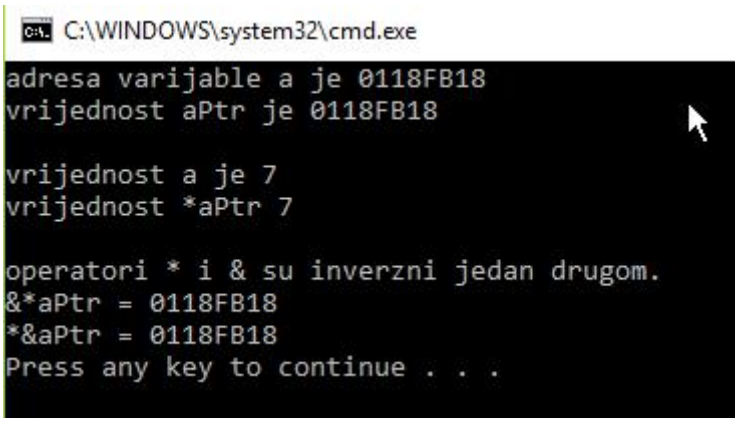
objekt – objekt na koji pokazivač upućuje.

Operator adrese & i operator derefenciranja * su inverzni operatori što pokazuje sljedeći primjer:

```
// korištenje & i * operatora
#include <iostream>
using namespace std;
int main()
{
      int a;
      int *aPtr; // aPtr je pointer na integer-cjelobrojnu vrijednost
      a = 7;
      aPtr = &a; // aPtr postavljen na adresu varijable a
      cout << "adresa varijable a je " << &a << "\nvrijednost aPtr je "
<< aPtr;
      cout << "\n\nvrijednost a je " << a << "\nvrijednost *aPtr " <<
*aPtr;
```

```
	cout << "\n\noperatori * i & su inverzni " << "jedan drugom.
\n&*aPtr = "
		<< &*aPtr << "\n*&aPtr = " << *&aPtr << endl;
	return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Primjer:

Napišite program u kojem ćete:

a) Deklarirati dvije varijable tipa float (broj1 i broj2).

b) Inicijalizirati varijablu broj1 vrijednošću 7.3.

c) Deklarirati pokazivač na tip podatka float.

d) Inicijalizirati pokazivač na adresu varijable broj1.

e) Ispisati vrijednost varijable na koju upućuje pokazivač koristeći dereferenciranje pokazivača.

f) Inicijalizirati vrijednost varijable broj2 na istu vrijednost na koju upućuje pokazivač, te ju ispisati.

g) Ispisati adresu varijable broj1.

```
#include <iostream>
using namespace std;
int main()
{
	float broj1, broj2;
	broj1 = 7.3;//inicijalizacija vrijednosti varaijable broj1
```

```
    float *pokazivac; //deklaracija pokazivaca
    pokazivac = &broj1; //inicijalizacija pokazivaca adresom
varijable broj1
    cout << "vrijednost varijable broj1 iznosi " << *pokazivac <<
endl;
    broj2 = *pokazivac; //varijabli broj2 smo dodijelili vrijednost
koristeci dereferenciranje pokazivaca
    cout << "vrijednost varijable broj2 iznosi " << broj2 << endl;
    cout << "Adresa varijable broj1 je " << pokazivac <<
endl;//adresu varijable broj1 ispisuemo pomocu pokazivaca
    cout << "Adresa varijable broj1 je " << &broj1 << endl; //adresu
varijable broj1 ispisujemo upotrebom operatora &
    return 0;
}
```

Sljedeće su važeće deklaracije pokazivača:

```
int *ip; // pokazivac na integer

double *dp; // pokazivac na double

float *fp; // pokazivac na float

char *ch // pokazivac na character
```

6.2 Upotreba pokazivača u C++

Postoji nekoliko važnih operacija, koje ćemo vrlo često raditi sa pokazivačima.

1. Definišemo varijablu pokazivača.
2. Dodijeliti adresu varijable pokazivaču.
3. Pristupiti vrijednosti na adresi na koju pokazuje pokazivač.

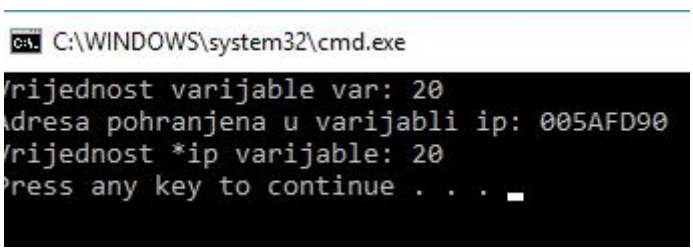
Ovo se vrši pomoću operatora * koji vraća vrijednost varijable koja se nalazi na adresi koju je odredio njegov operand.

Primjer:

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int var = 20; // deklaracija varijable
    int *ip; // deklaracija pokazivaca
    ip = &var; // pohranjuje adresu var u varijabli pokazivača
    cout << "Vrijednost varijable var: ";
    cout << var << endl;
    // ispisuje adresu pohranjenu u ip pokazivacu
    cout << "Adresa pohranjena u varijabli ip: ";
    cout << ip << endl;
    // pristup vrijednosti na adresi dostupnoj u pokazivaču
    cout << "Vrijednost *ip varijable: ";
    cout << *ip << endl;
    return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



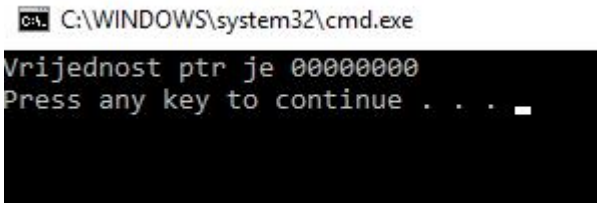
6.3 NULL pokazivači

Uvijek je dobra praksa dodijeliti pokazivaču vrijednost NULL u slučaju da nemate tačnu adresu koju želite dodijeliti. NULL pokazivač se deklariše prilikom deklaracije varijable. Pokazivač kojem je dodijeljena NULL vrijednost naziva se **nulti pokazivač**.

NULL pokazivač je konstanta sa vrijednošću nula definisana u nekoliko standardnih c++ biblioteka, uključujući i biblioteku iostream. Razmotrimo sledeći program:

```
#include <iostream>
using namespace std;
int main()
{
    int *ptr = NULL;
    cout << "Vrijednost ptr je " << ptr;
    return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



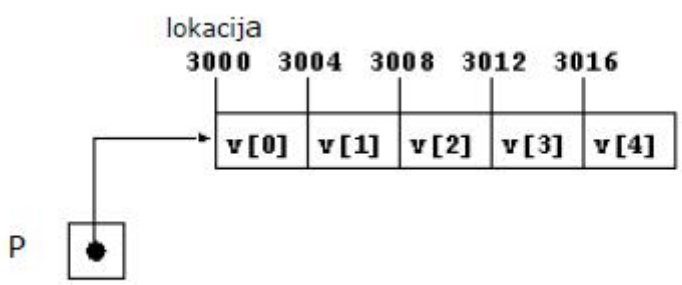
Na većini operativnih sistema programima nije dozvoljeno pristupiti memoriji na adresi 0 jer je ta memorija rezervisana od strane operativnog sistema. Međutim, memorijska adresa 0 ima posebno značenje; signalizira da pokazivač ne treba usmjeriti na dostupnu memorijsku lokaciju. Međutim, konvencijom, ako pokazivač sadrži nultu (nulu) vrijednost, pretpostavlja se da ukazuje na ništa.

6.4. Operacije nad pokazivačima

Pokazivači su validni operandi u aritmetičkim izrazima, izrazima dodjeljivanja i izrazima usporedbe. No, na pokazivače je moguće primijeniti ograničen skup aritmetičkih operacija. Pokazivači se mogu:

- dekrementirati (--),
- inkremnetirati (++),
- zbrajati (+ ili +=) i
- oduzimati (- ili -=) s cjelobrojnim vrijednostima,
- oduzimati od drugih pokazivača.

Pretpostavimo da smo deklarisali niz int v[5] i da se prvi element niza nalazi na memorijskoj lokaciji 3000. Inicijalizirajmo pokazivač P na adresu 3000.



Pokazivač P je moguće inicijalizirati na adresu prvog elementa niza na dva načina:

```
P=v; // ime niza upućuje na adresu prvog elementa niza
```

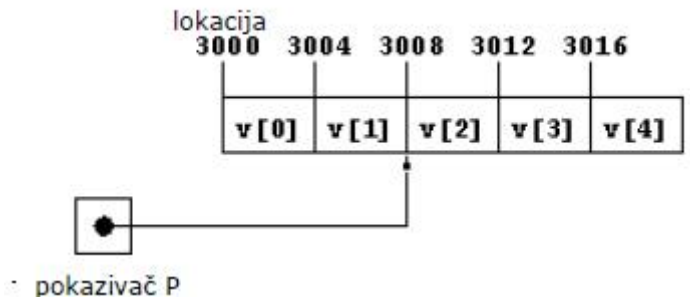
```
P=&v[0];
```

U konvencionalnoj aritmetici zbrajanje vrijednosti 3000 + 2 za rezultat ima vrijednost 3002. No, kad su u pitanju pokazivači sljedeći izraz:

```
P+=2;
```

će imati ovakav rezultat: 3008 (3000 + 2*4), pod pretpostavkom da je za pohranu cjelobrojnog tipa podatka potrebno 4 bajta (upotrebom operatora sizeof provjerite koliko je potrebno bajta za pohranu određenog tipa podatka).

U našem primjeru pokazivač P će upućivati na treći element niza, tj v[2].



Ukoliko pokazivač P dekrementiramo za 2:

```
P-=2;
```

Pokazivač će opet upućivati na memorijsku lokaciju 3000, tj na prvi element niza.

Izrazi poput P++ i ++P, kao i izrazi --P i P— inkrementiraju, tj. dekrementiraju vrijednost pokazivača na sljedeći, odnosno prethodni element niza.

Moguće je vršiti i oduzimanje dva pokazivača. Na primjer, ako P1 sadrži lokaciju 3000, a P2 lokaciju 3008, izraz:

```
X= P2-P1;
```

će dodijeliti varijabli X broj elemenata niza koji se nalaze između ove dvije lokacije, tj. 2.

Primjena aritmetičkih operacija na pokazivačima ima smisla ukoliko je riječ o nizovima.

Ukoliko nije riječ o nizu, ne možemo tvrditi da će dvije varijable istog tipa biti pohranjene u memoriji jedna iza druge.

Kad primjenjujete navedene aritmetičke operacije na nizove karaktera rezultat će biti u skladu s konvencionalnom aritmetikom, jer je za pohranjivanje karaktera u memoriju potreban jedan bajt.

Zadatak za vježbu

1. Napišite program u kojem ćete:

a) deklarisati niz tipa double koji ima 10 elemenata:

```
double niz [10]={0.0, 1.1, 2.2, 3.3., 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};
```

b) provjeriti koliko prostora u memoriji zauzima tip podatka double i koliko prostora u memoriji zauzima cijeli niz, `(upotrijebite operator sizeof)`

c) deklarisati pokazivač koji pokazuje na objekt tipa double,

d) inicijalizirate pokazivač na adresu prvog elementa niza,

e) ispišite 4 element niza na barem 3 različita načina .

(koristite indeksaciju elemenata niza, dereferenciranje pokazivača i operacije nad pokazivačem)

f) ispišite adrese svih elemenata niza koristeći operator inkrementa na pokazivač.

(vodite računa da ukoliko dekrementirate pokazivač – rezultat je adresa pomaknuta za onoliko bajta koliko zauzima tip podatka na koji upućuje pokazivač)

g) ispišite elemente niza koristeći dereferenciranje pokazivača.

```
#include <iostream>
using namespace std;
int main()
{
	double niz[10] = { 0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8,
9.9 };
	cout << "velicina tipa podatka double u bajtima je " <<
sizeof(double) << endl;
	cout << "velicina cijelog niz koji sadrzi 10 elemenata tipa
double u bajtima je " << sizeof(niz) << endl;
	double *pokazivac;
	pokazivac = &niz[0]; //inicijalizacija pokazivaca na adresu prvog
elementa niza - 1 nacin
                              // pokazivac=niz; inicijalizacija
pokazivaca na adresu prvog elementa niza - 2 nacin
	cout << "Elementi niza su: " << endl;
	for (int i = 0; i<10; i++)
```

```
        cout << *(pokazivac + i) << endl;//ispis elemenata niza
dereferenciranjem pokazivaca
                                                /*unutar for petlje
smo mogli staviti i cout<<*(pokazivac++),no u tom slucaju bi po
zavrsetku petlje pokazivac sadrzavao adresu memorijske lokacije koja
se ne nalazi unutar niza*/
    cout << "ispis 4. elementa niza " << endl;
    cout << "Prvi nacin " << niz[3] << endl;
    cout << "Drugi nacin " << *(niz + 3) << endl;
    cout << "Treci nacin " << pokazivac[3] << endl;
    cout << "Cetvrti nacin nacin " << *(pokazivac + 3) << endl;
    cout << "Adrese svih elemenata niza - inkrementiranje pokazivaca"
<< endl;
    for (int i = 1; i <= 10; i++)
    {
        cout << pokazivac << endl;
        pokazivac++;
    }
    system("pause>null");
    return 0;
}
```

6.4.1 Povećanje pokazivača (inkrement)

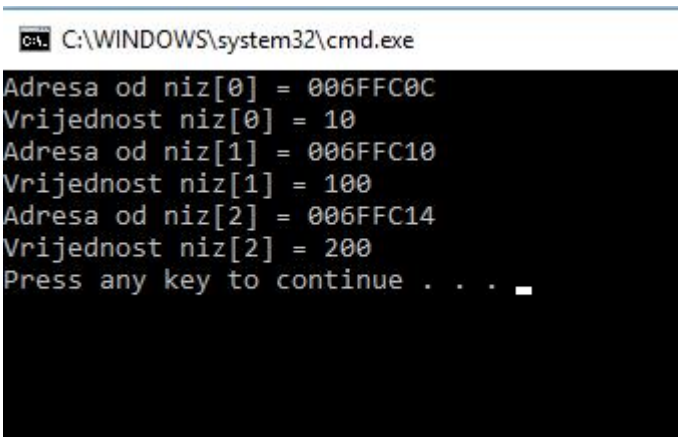
Bolje je koristiti pokazivač u programu umjesto niza, jer se varijabla pokazivača može povećati, za razliku od niza koji se ne može povećati jer ima konstantnu veličinu.

Sljedeći program povećava varijablu pokazivača da bi pristupio svakom sljedećem elementu polja:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int niz[MAX] = { 10, 100, 200 };
    int *pok;
    // imamo adresu polja u pokazivaču.
    pok = niz;
```

```
    for (int i = 0; i < MAX; i++)
    {
        cout << "Adresa od niz[" << i << "] = ";
        cout << pok << endl;
        cout << "Vrijednost niz[" << i << "] = ";
        cout << *pok << endl;
        //pokazuje na sljedecu lokaciju
        pok++;
    }
    return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



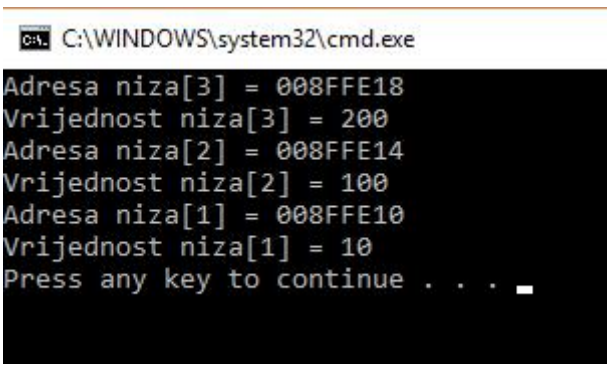
6.4.2 Smanjenje pokazivača (dekrement)

Isto razmatranje se odnosi i na smanjenje pokazivača, koji smanjuje svoju vrijednost po broju bajtova njegovog tipa podatka kao što je prikazano u nastavku:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int niz[MAX] = { 10, 100, 200 };
    int *pok;
    // adresa posljednjeg elementa na koji pokazuje pokazivač.
    pok = &niz[MAX - 1];
    for (int i = MAX; i > 0; i--)
```

```
    {
        cout << "Adresa niza[" << i << "] = ";
        cout << pok << endl;
        cout << "Vrijednost niza[" << i << "] = ";
        cout << *pok << endl;
        // pokazuje na prethodnu lokaciju
        pok--;
    }
    return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



6.4.3 Poređenje pokazivača

Pokazivači se mogu porediti pomoću relacijskih operatora, kao što su ==, <i>. Ako p1 i p2 ukazuju na međusobno povezane varijable, kao što su elementi istog polja, tada se p1 i p2 mogu porediti.

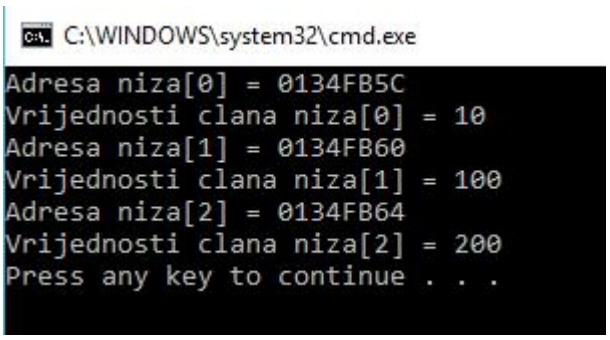
Sljedeći program mijenja prethodni primjer povećavanjem varijable pokazivača sve dok adresa na koju se upućuje nije manja ili jednaka adresi posljednjeg elementa polja, &niz [MAX - 1]:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int niz[MAX] = { 10, 100, 200 };
    int *pok;
    // adresa prvog elementa na koji pokazuje pokazivac.
    pok = niz;
    int i = 0;
    while (pok <= &niz[MAX - 1])
```

```
    {
        cout << "Adresa niza[" << i << "] = ";
        cout << pok << endl;
        cout << "Vrijednosti clana niza[" << i << "] = ";
        cout << *pok << endl;
        // pokazuje na sljedecu lokaciju
        pok++;
        i++;
    }
    return 0;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



6.5 Pokazivači vs nizovi

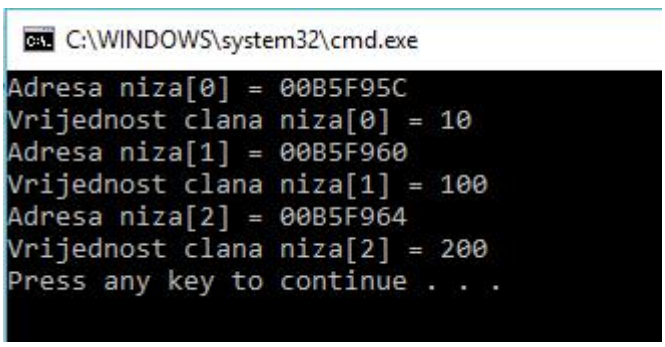
Pokazivači i nizovi su snažno povezani. U stvari, pokazivači i nizovi su međusobno zamjenjivi u mnogim slučajevima. Na primjer, pokazivač koji ukazuje na početak niza može pristupiti tom nizu pomoću aritmetičkog pokazivača ili indeksa u nizu. Razmotrimo sljedeći program:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int niz[MAX] = { 10, 100, 200 };
    int *pok;
    pok = niz;
    for (int i = 0; i < MAX; i++)
    {
        cout << "Adresa niza[" << i << "] = ";
```

```
            cout << pok << endl;
            cout << "Vrijednost clana niza[" << i << "] = ";
            cout << *pok << endl;
        pok++;
      }
      return 0;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Međutim, pokazivači i polja nisu potpuno zamjenjivi. Na primjer, razmotrite sljedeći program:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
      int niz[MAX] = { 10, 100, 200 };
      for (int i = 0; i < MAX; i++)
      {
            *niz = i; // Ovo je ispravno po sintaksi
            niz++; // Ovo je neispravno
      }
      return 0;

}
```

Sasvim je prihvatljivo da primjenite operator pokazivača * na niz, ali je nezakonito modifikovati niz vrijednost. Razlog za to je što je niz konstantne veličine koja ukazuje na početak polja i ne može se koristiti kao inkrement vrijednost.

Zbog toga što ime niza generiše konstantu pokazivača, i dalje se može koristiti u izrazima u stilu pokazivača, sve dok se ne mjenja. Na primjer, sljedeći kod je važeća izjava koja dodjeljuje niz [2] vrijednost 500:

```
*(niz + 2) = 500;
```

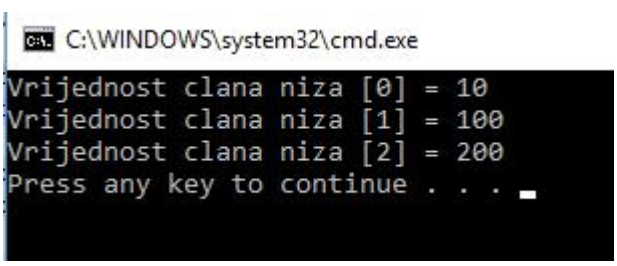
Izjava iznad je važeća i kompajlirat će se uspešno jer niz nije promijenjen.

6.6 Niz pokazivača

Pre nego što shvatimo koncept niza pokazivača, razmotrimo sljedeći primjer, koji koristi niz od 3 cijela broja:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
      int niz[MAX] = { 10, 100, 200 };
      for (int i = 0; i < MAX; i++)
      {
            cout << "Vrijednost clana niza [" << i << "] = ";
            cout << niz[i] << endl;
      }
      return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



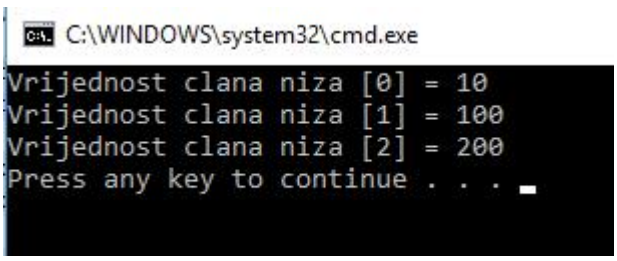
Možda postoji situacija kada želimo zadržati niz, koji može pohraniti podatke tipa int ili char ili bilo koju drugu dostupnu vrstu podataka. U nastavku prikazana deklaracija niza pokazivača na cijeli broj:

```
int *pok[MAX]; // deklaracija pokazivaca pok kao niz MAX integer pokazivača.
```

Stoga, svaki element u pok, sada drži pokazivač na int vrijednost. Sljedeći primjer koristi tri cijela broja koji će biti sačuvani u nizu pokazivača na sljedeći način:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
      int niz[MAX] = { 10, 100, 200 };
      int *pok[MAX];
      for (int i = 0; i < MAX; i++)
      {
            pok[i] = &niz[i]; //dodjeljuje adresu cijelog broja
      }
      for (int i = 0; i < MAX; i++)
      {
            cout << "Vrijednost clana niza[" << i << "] = ";
            cout << *pok[i] << endl;
      }
      return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:

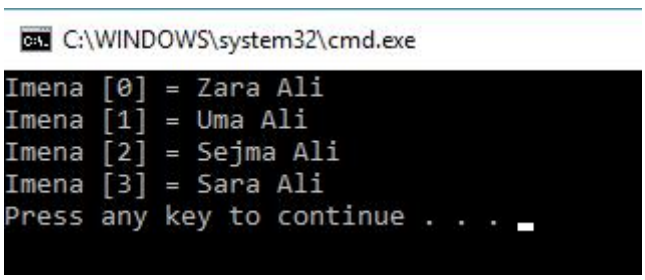


Također možete upotrijebiti niz pokazivača za tim podatka char kako biste pohranili niz stringova:

```
#include <iostream>
using namespace std;
const int MAX = 4;
int main()
{
      char *imena[MAX] = {
            "Zara Ali",
```

```
            "Uma Ali",
            "Sejma Ali",
            "Sara Ali",
      };
      for (int i = 0; i < MAX; i++)
      {
            cout << "Imena [" << i << "] = ";
            cout << imena
[i] << endl;
      }
      return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



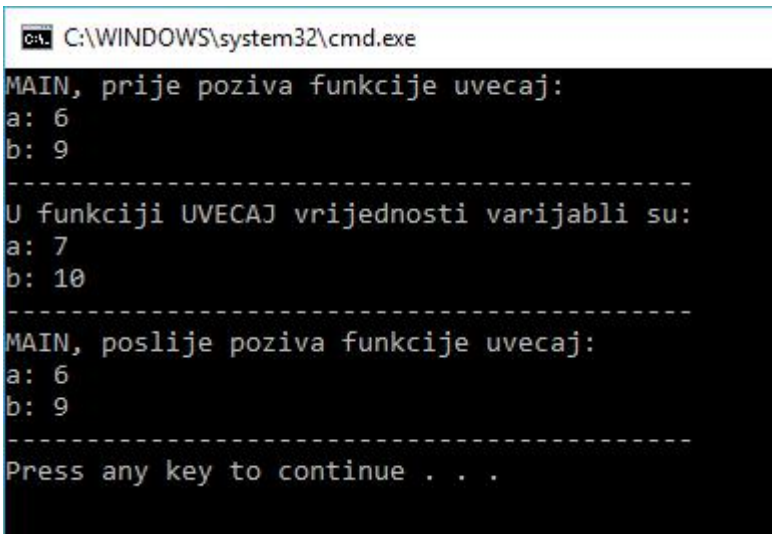
6.7 Pokazivači i funkcije

U nastavku će biti objašnjen način korištenja pokazivača u radu sa funkcijama. Za one koji su zaboravili, na samom početku ćemo se podsjetiti načina na koji se parametri predaju funkciji. U narednom primjeru funkcija ima zadatak da uveća (inkrementira), te ispiše vrijednosti primljenih parametara.

```
#include <iostream>
using namespace std;
void Uvecaj(int a, int b) {
      //pošto se varijable a i b predaju po vrijednosti
      a++; //ovaj inkrement nece utjecati na varijablu 'a' u main-u
      b++; //ovaj inkrement nece utjecati na varijablu 'b' u main-u
      cout << "U funkciji UVECAJ vrijednosti varijabli su: " << endl;
      cout << "a: " << a << endl; //ispisuje 7
      cout << "b: " << b << endl; //ispisuje 10
      cout << "----------------------------------------\n";
}
```

```
void main() {
	cout << "MAIN, prije poziva funkcije uvecaj: " << endl;
	cout << "a: " << a << endl; //ispisuje 6
	cout << "b: " << b << endl; //ispisuje 9
	cout << "-----------------------------------------\n";
	Uvecaj(a, b);
	cout << "MAIN, poslije poziva funkcije uvecaj: " << endl;
	cout << "a: " << a << endl; //ispisuje 6
	cout << "b: " << b << endl; //ispisuje 9
	cout << "-----------------------------------------\n";
	system("pause");
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Predavanjem parametara po vrijednosti je onemogućena izmjena originalnih vrijednosti. U prethodnom primjeru, unutar main funkcije je izvršena deklaracija i inicijalizacija dvije varijable: a i b. Nakon ispisa vrijednosti deklarisanih varijabli, uslijedio je poziv funkcije Uvecaj kojoj se kao parametri predaju varijable a i b. Bitno je uočiti da se funkciji Uvecaj predaju kopije varijabli a i b (ne originali), te se promjene na kopijama ne reflektuju na originalne vrijednosti. Kao dokaz tome, prethodni program ispisuje vrijednosti varijabli prije i poslije poziva funkcije, kao i u samoj funkciji.

Ukoliko je potrebno omogućiti izmjenu originalnih vrijednosti parametara, funkciji je potrebno predati memorijske adrese varijabli čije vrijednosti funkcija treba mijenjati. Već vam je poznat način na koji je moguće saznati adresu varijable (koristeći operator '&' ispred imena varijable), pa još ostaje zadatak da se u zaglavlju naznači da ulaze u funkciju predstavljaju adrese varijabli, a ne

njihove vrijednosti. Primljene adrese se smještaju u pokazivač, čime se dobija mogućnost manipulisanja originalnim vrijednostima varijable. U nastavku je prikazan primjer koji je identičan prethodnom, samo uz korištenje pokazivača:

```
#include <iostream>
using namespace std;
void Uvecaj(int * a, int * b) {
	(*a)++; //inkrementira vrijednost varijable 'a' u main-u
	(*b)++; //inkrementira vrijednost varijable 'b' u main-u
	cout << "U funkciji UVECAJ vrijednosti varijabli su: " << endl;
	//pošto se radi o pokazivaču, za ispisivanje vrijednosti
	//moramo koristiti dereferenciranje tj. znak (*)
	cout << "a: " << *a << endl; //ispisuje 7
	cout << "b: " << *b << endl; //ispisuje 10
	cout << "----------------------------------------\n";
}
void main() {
	int a = 6;
	int b = 9;
	cout << "MAIN, prije poziva funkcije UVECAJ: " << endl;
	cout << "a: " << a << endl; //ispisuje 6
	cout << "b: " << b << endl; //ispisuje 9
	cout << "----------------------------------------\n";
	//pošto funkcija prima pokazivače,
	Uvecaj(&a, &b); //moramo proslijediti adrese varijabli
	cout << "MAIN, poslije poziva funkcije UVECAJ: " << endl;
	cout << "a: " << a << endl; //ispisuje 7
	cout << "b: " << b << endl; //ispisuje 10
	cout << "----------------------------------------\n";
	system("pause");
}
```

Na osnovu prethodnog primjera može se zaključiti da funkcija, koja kao argument prima adresu varijable, mora posjedovati odgovarajući pokazivač pomoću kojeg će manipulisati vrijednostima tj. koji će prihvatiti adresu proslijeđenog argumenta. Na osnovu ispisa je vidljivo da je funkcija Uvecaj, koristeći pokazivače, izmijenila originalne vrijednosti varijabli a i b.

Mada je pojašnjeno u prethodnom materijalu, bitno je shvatiti razlog zbog kojeg se prilikom inkrementiranja vrijednosti varijabli koriste zagrade (*a)++. Glavni razlog tome je činjenica da operator inkrementiranja (++) posjeduje veći prioritet od operatora indirekcije (*). Na osnovu toga, iskaz (*a)++ se može protumačiti kao: uvećaj vrijednost na koju pokazuje pokazivač 'a'. Da je kojim slučajem napisano *a++ tada bi se uvećala vrijednost samog pokazivača tj. pokazivač bi pokazivao na narednu memorijsku lokaciju. Ukoliko se zanemari stvarni način adresiranja memorije, te pretpostavimo da pokazivač 'a' pokazuje na varijablu koja se nalazi na adresi 10036, nakon iskaza *a++ pokazivač će pokazivati na adresu 10037 odnosno 10040 (ako pretpostavimo da je veličina tipa int 4 bajta). Da biste se u to uvjerili, u prethodnom primjeru uklonite zagrade prilikom inkrementiranja vrijednosti pokazivača a i b, te ponovo pokrenite program.

6.7.1 Poziv funkcije po referenci korištenjem pokazivačkih parametara

U C++ funkciji može se proslijediti parametre (argumente) na tri načina: pozivom funkcije po vrijednosti (call bay value), pozivom funkcije korištenjem referentnih parametara (call bay reference with refrence arguments) i pozivom funkcije po referenci korištenjem pokazivačkih parametara (call bay reference with pointer arguments). Prva dva načina već su vam poznata. Pozabavimo se s pozivom funkcije po referenci korištenjem pokazivačkih parametara. Pri pozivu funkcije s parametrima koje je potrebno modificirati, ne prosljeđujete vrijednosti parametara, već njihove adrese. Prosljeđivanje adrese kao parametra funkcije se postiže upotrebom adresnog operatora (&) prije imena varijable čiju vrijednost želimo modificirati u funkciji, a u zaglavlje funkcije kao parametar se stavlja pokazivač.

6.8 Zadaci za vježbu:

1. Napišite program koji će vrijednost unesenog cijelog broja mijenjati u trostruko veću – kub unesenog broja. Neka funkcija za računanje kuba ima jedan argument i neka njeno zaglavlje izgleda ovako:

```
void kub ( int * );
```

```
#include <iostream>
using namespace std;
void kub(int *); // prototip ili deklaracija funkcije
int main()
{
      int broj;
      cout << "Unesite broj ";
      cin >> broj;
```

```
	cout << "Originalna vrijednost broja je " << broj;
	kub(&broj); //parametar funkcije je adresa varijable broj
	cout << "\nNova vrijednost broja je " << broj << endl;
	system("pause>null");
	return 0;
}
void kub(int * pokazivac)
{
	*pokazivac = *pokazivac * *pokazivac * * pokazivac;
	//razlikovati operator dereferenciranja od operatora mnozenja
}
```

2. Napišite program u kojem ćete deklarisati niz od 5 cjelobrojnih elemenata te uz pomoć:

- funkcije: `void unos (int *, int);` omogućiti unos elemenata niza
- funkcije: `void ispis (int *, int);` omogućiti ispis elemenata niza
- funkcije: `int * najveci (int *, int);` vratiti adresu najvećeg elementa u nizu;

```
#include <iostream>
using namespace std;
void unos(int *, int);
void ispis(int *, int);
int * najveci(int *, int); //funkcija vraca adresu, odnosno pokazivac
void provjera(int *, int);
int main()
{
	const int velicina = 5;
	int niz[velicina];
	int * pokazivac = 0;
	cout << "Unesite elemente niza" << endl;
	unos(niz, velicina);//ime niza je adresa prvog elementa niza
	cout << "Niz sacinjavaju sljedeci elementi" << endl;
	ispis(niz, velicina);
	pokazivac = najveci(niz, velicina);
	cout << "Najveci element se nalazi na sljedecoj adresi" <<
pokazivac << endl;
```

```
	cout << "Prva provjera - ispis svih adresa elemenata niza" << endl << endl;
	provjera(niz, velicina);
	cout << "Druga provjera - na adresi koju je vratila funkcija najveci se nalazi vrijednost" << *pokazivac << endl;//dereferniciramo vrijednost na adresi koju cuva pokazivac
	system("pause>null");
	return 0;
}

void unos(int * pok, int vel)// buduci da je ime niza adresa provog elementa - koristimo pokazivac da pohranimo navedenu adresu
{
	for (int i = 0; i<vel; i++)
		cin >> *(pok + i); //dereferenciramo pokazivac da bi smo mogli unijeti vrijednost na zeljenu adresu
}
void ispis(int * pok, int vel)
{
	for (int i = 0; i<vel; i++)
		cout << *(pok + i) << "\t";
	cout << endl;
}
int * najveci(int * pok, int vel)
{
	int *lokalni = 0;
	int max = pok[0];
	for (int i = 0; i < vel; i++)
	{
		if (*(pok + i) > max)
		{
			max = *(pok + i);
			lokalni = pok + i;//pokazivac lokalni cuva adresu trenutno najveceg elementa
		}
	}
```

```
	return lokalni;//funkcija vraca pokazivac
}
	void provjera(int * pok, int vel)//funkcija ispisuje adrese svih
elemenata niza
{
			for (int i = 0; i<vel; i++)
				cout << pok + i << endl;
}
```

3. Napravite program u kome ćete:

- unijeti dva broja (a i b tipa float).
- deklarisati pokazivač p1 koji će pokazivati na varijablu a
- deklarisati pokazivač p2 koji će pokazivati na varijablu b
- izračunati sljedeće matematičke izraze koristeći pokazivače p1 i p2 (tj. bez korištenja varijable a i b).

$$c_1 = a - b,\ c_2 = \sin(a) - \cos(b),\ c_3 = b * a,\ c_4 = \frac{a}{b} \quad za \quad b \neq 0,\ c_5 = a^b,\ c_6 = (\sqrt{a})^b$$

```
#include<iostream>
#include<cmath>
using namespace std;
void main()
{

	float a, b;
	cin >> a >> b;
	float* p1 = &a;
	float* p2 = &b;
	//float c1 = a - b;
	//float c1 = p1 - p2; <-- ovo je neispravno: ovdje se računa
razlika adresa, a ne razlika vrijednosti
	float c1 = *p1 - *p2;
	cout << "c1 = " << c1 << endl;
	float c2 = sin(*p1) - cos(*p2);
	cout << "c2 = " << c2 << endl;
```

```
    float c3 = *p2 * (*p1);
    cout << "c3 = " << c3 << endl;
        if (*p2 != 0) //ako je b!=0
    {
        float c4 = *p1 / *p2;
        cout << "c4 = " << c4 << endl;
    }
    float c5 = pow(*p1, *p2);
    cout << "c5 = " << c5<<endl;
    float c6 = pow(sqrt(*p1), *p2);
    cout << "c6= " << c6<<endl;
}
```

7 Strukture

C / C ++ nizovi vam omogućavaju da definišete više varijabli istog tipa podataka, dok struktura je još jedan korisnički definirani tip podataka koji vam omogućava kombinovanje podataka različitih vrsta podataka.

Pod konstatacijom da strukture predstavljaju kolekciju varijabli različitog tipa se podrazumijeva da su strukture pogodne za čuvanje više različitih informacija o nekom entitetu npr. Informacije o knjigama u biblioteci: naslov, autor, tema, knjigaID, itd. Na prvi pogled je jasno da se pobrojane informacije ne mogu smjestiti u jedan niz, a razlog tome je činjenica da nizovi mogu sadržavati više elemenata koji moraju biti istog tipa. Idealan način za rješavanje problema koji zahtijevaju čuvanje veće količine podataka različitog tipa predstavlja upravo korištenje struktura.

7.1 Definicija strukture

Pored ugrađenih tipova podataka (int, float, double itd.) strukture predstavljaju korisnički-definisane tipove. Pri deklaraciji varijable nekog od ugrađenih tipova prvo se navodi tip podatka, pa tek onda ime varijable.

Da biste definisali strukturu, morate koristiti ključnu riječ **`struct`**, nakon čega slijedi naziv strukture. Naziv strukture se može posmatrati kao novi tip podatka. Također, bitno je primijetiti da se tijelo strukture nalazi unutar vitičastih zagrada koje završavaju znakom tačka–zarez (;). Unutar tijela strukture se nalaze njeni elementi koji se nazivaju obilježja ili atributi.

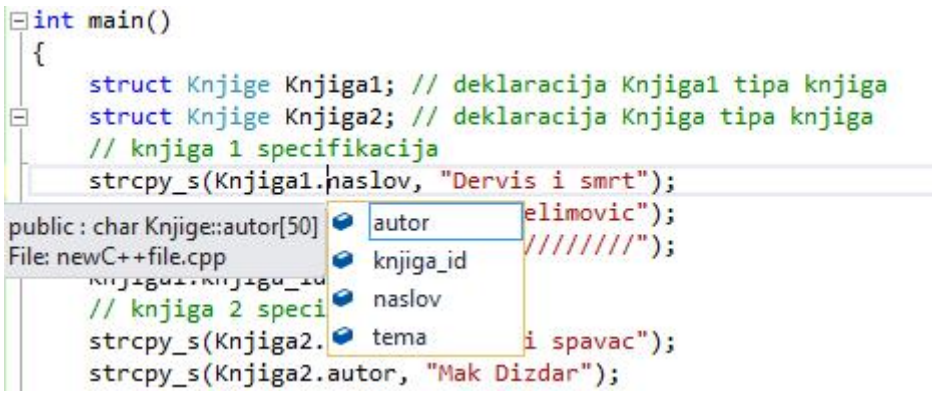
```
struct Knjige
{
      char naslov[50]; //niz u koji se može pohraniti 50 karaktera
      char autor[50];
      char tema[100];
      int knjiga_id; // varijabla u koju se pohranjuje id broj knjige

};
```

7.2 Pristupanje članovima strukture

Za pristup bilo kojem članu strukture, tj. Inicijalizaciju obilježlja strukture koristimo operater pristupa članovima (.). Operator pristupa članu kodiran je kao razdoblje između naziva strukturne varijable i člana strukture kojoj želimo pristupiti. Pomoću **struct** ključne riječi definišete varijable tipa strukture.

Prilikom inicijalizacije obilježja, većina razvojnih okruženja, kao što je Visual Studio, programerima pruža malu pomoć. Naime, odmah nakon navođenja imena objekta i znaka tačke, dobija se lista svih obilježja koji se nalaze u strukturi (što se vidi na sljedećoj slici).



Bitno je napomenuti da se obilježja tipa char (niz karaktera) inicijalizuju koristeći funkciju strcpy_s(). Prvi dio naziva funkcije str se odnosi na string, a drugi dio cpy predstavlja skraćenicu riječi copy. Dodatak _s označava da se radi o sigurnijoj verziji funkcije strcpy(). Funkcija strcpy_s() prihvata dva argumenta; prvi kojim se definiše gdje se nešto kopira (destinaciju) i drugi koji definiše šta se kopira (izvor).

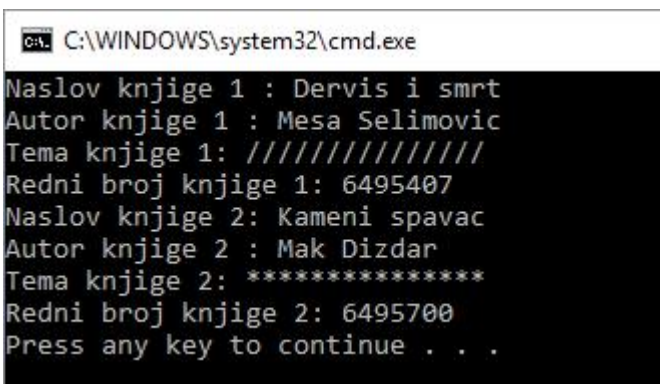
Slijedi primjer objašnjenja upotrebe strukture:

```
#include <iostream>
using namespace std;
struct Knjige
{
	char naslov[50];
	char autor[50];
	char tema[100];
	int knjiga_id;
};
int main()
{
	struct Knjige Knjiga1; // deklaracija Knjiga1 tipa knjiga
	struct Knjige Knjiga2; // deklaracija Knjiga tipa knjiga
	// knjiga 1 specifikacija
	strcpy_s(Knjiga1.naslov, "Dervis i smrt");
	strcpy_s(Knjiga1.autor, "Mesa Selimovic");
	strcpy_s(Knjiga1.tema, "///////////////");
	Knjiga1.knjiga_id = 6495407;
```

```
    // knjiga 2 specifikacija
    strcpy_s(Knjiga2.naslov, "Kameni spavac");
    strcpy_s(Knjiga2.autor, "Mak Dizdar");
    strcpy_s(Knjiga2.tema, "***************");
    Knjiga2.knjiga_id = 6495700;
    // ispis info o knjizi 1
    cout << "Naslov knjige 1 : " << Knjiga1.naslov << endl;
    cout << "Autor knjige 1 : " << Knjiga1.autor<< endl;
    cout << "Tema knjige 1: " << Knjiga1.tema << endl;
    cout << "Redni broj knjige 1: " << Knjiga1.knjiga_id << endl;
    // ispis info o knjizi 2
    cout << "Naslov knjige 2: " << Knjiga2.naslov << endl;
    cout << "Autor knjige 2 : " << Knjiga2.autor << endl;
    cout << "Tema knjige 2: " << Knjiga2.tema << endl;
    cout << "Redni broj knjige 2: " << Knjiga2.knjiga_id << endl;
    return 0;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Najvažnije je primijetiti da su vrijednosti obilježja objekta Knjiga1 i Knjiga2 apsolutno nezavisne. Dakle, objekti su međusobno u potpunosti odvojeni, a struktura Knjige služi samo kao kalup kojim se definiše koja će obilježja posjedovati svaki od njenih objekata.

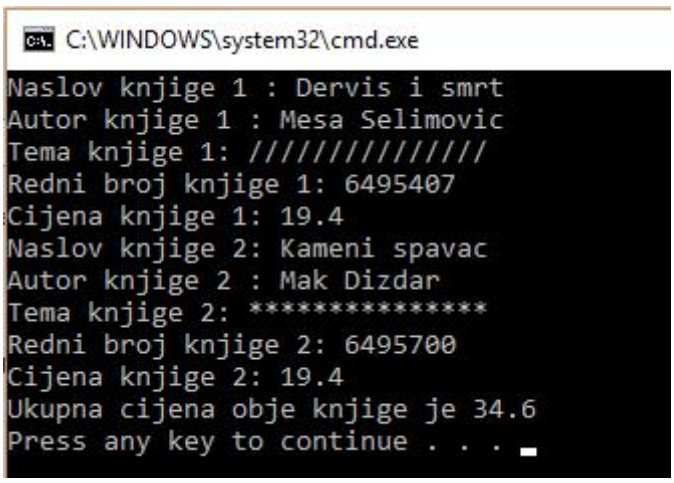
Sa objektima, odnosno obilježjima objekata, se mogu vršiti i aritmetičke operacije. Kao ilustraciju pomenutog i u narednom primjeru dodat ćemo obilježlje cijena knjige. Kreirat ćemo program u kome je deklarisana struktura Knjige, a koja je malo izmijenjena u odnosu na verziju iz prethodnog primjera.

```
#include <iostream>
```

```
using namespace std;
struct Knjige
{
	char naslov[50];
	char autor[50];
	char tema[100];
	int knjiga_id;
	double cijena;
};
int main()
{
	struct Knjige Knjiga1; // deklaracija Knjiga1 tipa knjiga
	struct Knjige Knjiga2; // deklaracija Knjiga tipa knjiga
	// knjiga 1 specifikacija
	strcpy_s(Knjiga1.naslov, "Dervis i smrt");
	strcpy_s(Knjiga1.autor, "Mesa Selimovic");
	strcpy_s(Knjiga1.tema, "///////////////");
	Knjiga1.knjiga_id = 6495407;
	Knjiga1.cijena = 19.40;
	// knjiga 2 specifikacija
	strcpy_s(Knjiga2.naslov, "Kameni spavac");
	strcpy_s(Knjiga2.autor, "Mak Dizdar");
	strcpy_s(Knjiga2.tema, "***************");
	Knjiga2.knjiga_id = 6495700;
	Knjiga2.cijena = 15.20;
	double ukupnaCijena = Knjiga1.cijena +
Knjiga2.cijena;//kalkulacije nad obilježljima objekata
	// ispis info o knjizi 1
	cout << "Naslov knjige 1 : " << Knjiga1.naslov << endl;
	cout << "Autor knjige 1 : " << Knjiga1.autor<< endl;
	cout << "Tema knjige 1: " << Knjiga1.tema << endl;
	cout << "Redni broj knjige 1: " << Knjiga1.knjiga_id << endl;
	cout << "Cijena knjige 1: " << Knjiga1.cijena << endl;
	// ispis info o knjizi 2
	cout << "Naslov knjige 2: " << Knjiga2.naslov << endl;
	cout << "Autor knjige 2 : " << Knjiga2.autor << endl;
```

```
    cout << "Tema knjige 2: " << Knjiga2.tema << endl;
    cout << "Redni broj knjige 2: " << Knjiga2.knjiga_id << endl;
    cout << "Cijena knjige 2: " << Knjiga1.cijena << endl;
    cout << "Ukupna cijena obje knjige je " << ukupnaCijena<<endl;
    return 0;
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



7.3 Strukture kao argumenti funkcije

Strukturu možete proslijediti kao argument funkcije na vrlo sličan način kao što se prosljeđuje bilo koja druga varijabla ili pokazivač, a funkcije mogu imati povratnu vrijednost tipa neke strukture. Primjer:

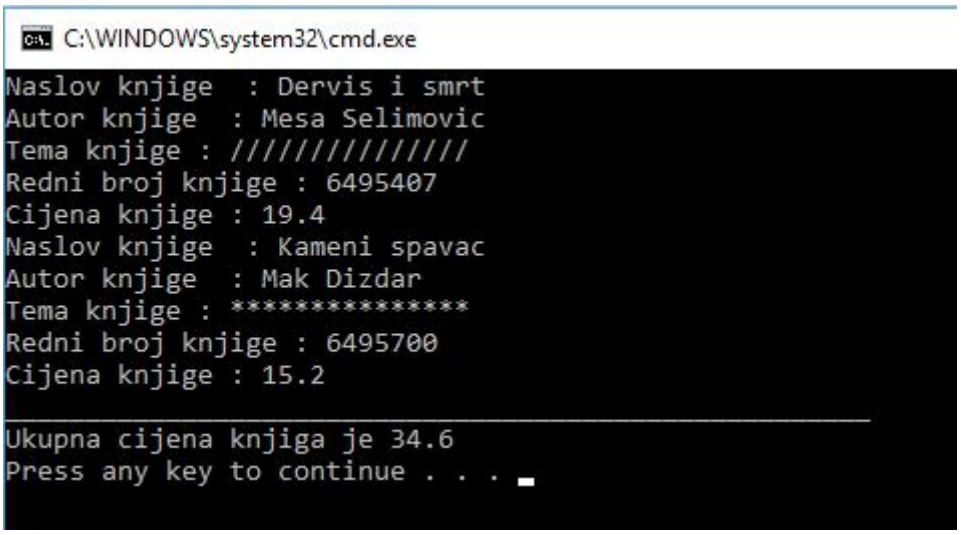
```
#include <iostream>
using namespace std;
void ispisInfo(struct Knjige );
struct Knjige
{
    char naslov[50];
    char autor[50];
    char tema[100];
    int knjiga_id;
    double cijena;
};
int main()
{
    struct Knjige Knjiga1; // deklaracija Knjiga1 tipa knjiga
```

```
    struct Knjige Knjiga2; // deklaracija Knjiga tipa knjiga
    // knjiga 1 specifikacija
    strcpy_s(Knjiga1.naslov, "Dervis i smrt");
    strcpy_s(Knjiga1.autor, "Mesa Selimovic");
    strcpy_s(Knjiga1.tema, "///////////////");
    Knjiga1.knjiga_id = 6495407;
    Knjiga1.cijena = 19.40;
    // knjiga 2 specifikacija
    strcpy_s(Knjiga2.naslov, "Kameni spavac");
    strcpy_s(Knjiga2.autor, "Mak Dizdar");
    strcpy_s(Knjiga2.tema, "***************");
    Knjiga2.knjiga_id = 6495700;
    Knjiga2.cijena = 15.20;
    double ukupnaCijena = Knjiga1.cijena +
Knjiga2.cijena;//kalkulacije nad obilježljima objekata
    //poziv funkcije za ispis informacija o knjigama
    ispisInfo(Knjiga1);
    ispisInfo(Knjiga2);
    cout << "_____________________________________________________"
<< endl;
    cout << "Ukupna cijena knjiga je " << ukupnaCijena << endl;
    return 0;
}
void ispisInfo(struct Knjige knjiga)
{

    cout << "Naslov knjige  : " << knjiga.naslov << endl;
    cout << "Autor knjige  : " << knjiga.autor << endl;
    cout << "Tema knjige : " << knjiga.tema << endl;
    cout << "Redni broj knjige : " << knjiga.knjiga_id << endl;
    cout << "Cijena knjige : " << knjiga.cijena << endl;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



7.4 Zadaci za vježbu

1. Napraviti programa u kojem se unose podaci za: ime, prezime, matični broj, prosjek i datum rođenja za sve učenike jednog razreda (najviše 40). Nakon unosa svih podataka traži se učenik s najboljim prosjekom te se na kraju programa na zaslonu računala ispisuju svi podaci za učenika koji ima najbolji prosjek.

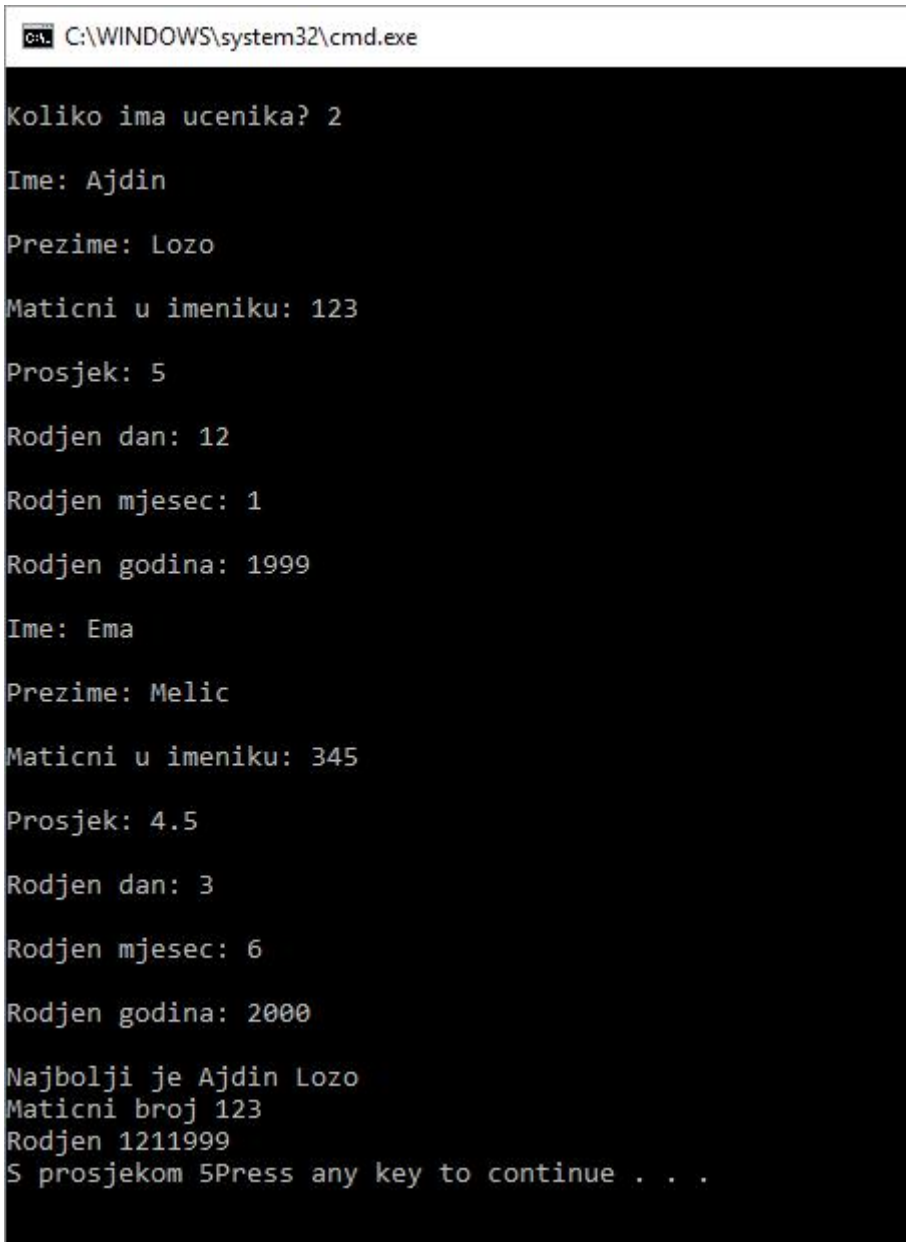
   Pomoć: U programu su definirane dvije strukture: datum i ucenik. Struktura datum služi za unos datuma rođenja učenika i sastoji se od dana, mjeseca i godine rođenja. Druga struktura se zove ucenik i sadrži 2 niza znakova (za ime i prezime), matični broj i prosjek ocjena. Niz (polje) razred se sastoji od struktura tipa učenik.

   Na primjer poziv za ispis varijable `razred[0].rodjendan.godina` će dati za ispis godine rođenja za 1. učenika, a `razred[1].prosjek` će dati ispis prosjeka za 2. učenika.

```
#include <iostream>
#include <string>
using namespace std;
struct datum
{
	int dan;
	int mjesec;
	int godina;
};
struct ucenik
{
	char ime[15];
	char prezime[15];
```

```
	int maticni;
	float prosjek;
	struct datum rodjendan;
}razred[40];		/*polje razred čiji su članovi strukture tipa
ucenik*/
int main()
{
	int n, i, k;
	float max;
	cout << "\nKoliko ima ucenika? ";
	cin >> n;
	if (n>40)
		cout << "Broj ucenika ne moze biti >40";
	else
	{
		for (i = 0; i<n; i++)		/*unos podataka za pojedinog
učenika u polje razred*/
		{
			cout << "\nIme: ";
			cin >> razred[i].ime;
			cout << "\nPrezime: ";
			cin >> razred[i].prezime;
			cout << "\nMaticni u imeniku: ";
			cin >> razred[i].maticni;
			cout << "\nProsjek: ";
			cin >> razred[i].prosjek;
			cout << "\nRodjen dan: ";
			cin >> razred[i].rodjendan.dan;
			cout << "\nRodjen mjesec: ";
			cin >> razred[i].rodjendan.mjesec;
			cout << "\nRodjen godina: ";
			cin >> razred[i].rodjendan.godina;
		}
		max = razred[0].prosjek;	/*određivanje najboljeg
prosjeka*/
		k = 0;
```

```
            for (i = 0; i<n; i++)
                if (max<razred[i].prosjek)
                {
                    max = razred[i].prosjek;
                    k = i;
                }
            /*ispis najboljeg*/
            cout << "\nNajbolji je " << razred[k].ime << " " <<
razred[k].prezime;
            cout << "\nMaticni broj " << razred[k].maticni;
            cout << "\nRodjen " << razred[k].rodjendan.dan <<
razred[k].rodjendan.mjesec << razred[k].rodjendan.godina;
            cout << "\nS prosjekom " << razred[k].prosjek;
      }
      system("PAUSE");
      return 0;
}
```


C:\WINDOWS\system32\cmd.exe
Koliko ima ucenika? 2
Ime: Ajdin
Prezime: Lozo
Maticni u imeniku: 123
Prosjek: 5
Rodjen dan: 12
Rodjen mjesec: 1
Rodjen godina: 1999
Ime: Ema
Prezime: Melic
Maticni u imeniku: 345
Prosjek: 4.5
Rodjen dan: 3
Rodjen mjesec: 6
Rodjen godina: 2000
Najbolji je Ajdin Lozo
Maticni broj 123
Rodjen 1211999
S prosjekom 5Press any key to continue . . .

8 Dinamički nizovi

Deklaracija varijable u obliku:

```
Tip ImeVarijable;
```

"govori" kompajleru da alocira blok memorije dovoljno velik da pohrani vrijednost specificiranu tipom podatka i povezuje identifikator ImeVarijable s adresom tog bloka u memoriji.

Kao što znate, rezultat izraza:

```
&ImeVarijable
```

je adresa spomenutog bloka u memoriji. Memorijski blok povezan s imenom varijable se alocira u trenutku kompajliranja programa i ne može se mijenjati bez editiranja i rekompajliranja programa.

Slično je i s deklaracijom statičkih nizova.

Deklaracija:

```
const int velicina=10;

double Niz[velicina];
```

rezultira alokacijom memorijskog bloka dovoljno velikog da pohrani deset vrijednosti tipa double i poveže ih s imenom niza (Niz). Veličina memorijskog bloka se ne može promijeniti bez editiranja i rekompajliranja programa. Može se zaključiti, statički definirani nizovi imaju sljedeće slabosti:

- ukoliko ima manje elemenata od deklarirane veličine niza – nepotrebno se rasipa memorija;
- ukoliko je veličina niza premala da pohrani sve vrijednosti pojavit će se greška – array overflow.

Oba navedena problema bi bilo moguće riješiti alokacijom memorije za vrijeme izvršavanja programa (tzv. run-time alokacija). Ovakav mehanizam bi morao uključivati sljedeće operacije:

- dodjeljivanje dodatne memorije nekom objektu ako se za tim pokaže potreba,
- "oslobađanje" memorije u trenutku kad više nije potrebna.

C++ nudi dvije predefinirane operacije new i delete za izvođenje ovih dviju operacija.

Operacija **`new`** upućuje zahtjev operativnom sistemu za još memorijskog prostora za vrijeme izvođenja programa. Opća sintaksa izgleda ovako:

```
new tip;
```

ovakav izraz rezultira zahtjevom za memorijskim blokom (tokom run-time) dovoljno velikim da pohrani podatkovni objekt specificiran tipom. Ukoliko je operativni sistem na zahtjev odgovori pozitivno, `new` vraća adresu memorijskog bloka, u suprotnom vraća NULL.

Budući da `new` vraća adresu, a adresa se može pohraniti u pokazivač, operacija `new` se uvijek izvodi u kombinaciji s pokazivačem.

Primjer:

```
int *pok;
pok = new int;
```

Izvršavanje izraza `new int` će rezultirati zahtjevom operativnom sistemu za memorijskim blokom dovoljno velikim da pohrani cjelobrojnu vrijednost. Ukoliko je operativni sistem u mogućnosti da odgovori na ovaj zahtjev varijabli pok će biti dodijeljena adresa memorijskog bloka. U protivnom, ako nema slobodnih memorijskih blokova pokazivaču će biti dodijeljena vrijednost 0. Stoga je poželjno provjeriti vrijednost koju vraća operacija new:

```
assert(Pokazivac!=0);
```

tj. upotrijebite assert mehanizam koji nudi C++. Ovaj mehanizam omogućava nastavak izvršavanja programa samo u slučaju ukoliko je rezultat navedenog izraza true (istina). Da bi ste mogli koristiti navedeni mehanizam potrebno je uključiti datoteku zaglavlja `<cassert>` (ili `<assert.h>` prema staroj notaciji).

Kad smo utvrdili da vrijednost pokazivača nije 0 (dodijeljena mu je adresa nekog memorijskog bloka) zapravo smo dobili tzv. anonimnu varijablu, odnosno s novoalociranom memorijom nije povezano nikakvo ime.

Budući da novoalociranoj memorijskoj lokaciji nije dodijeljeno nikakvo ime nije joj moguće pristupiti direktno. Međutim, adresa tog memorijskog bloka je pohranjena u pokazivač, te se vrijednostima pohranjenim u taj blok može pristupati indirektno – dereferenciranjem pokazivača.

Za manipulaciju anonimnim varijablama mogu se koristiti sljedeći iskaz:

```
cin >> *pok; //unosi se vrijednost s tastature u novoalocirani
memorijski blok
if (*pok<100) //primjena relacijskih operatora
(*pok)++; // primjena aritmetičkih operatora
else
```

```
*pok = 100; // dodjela vrijednosti
```

U kratko, sve što je moguće s "uobičajenim" cjelobrojnim varijablama moguće je i s anonimnim cjelobrojnim varijablama. Ista tvrdnja važi i za ostale tipove podataka.

8.1 Alokacija niza s operacijom New

U praksi, operacija **new** se rijetko koristi za alociranje prostora za skalarne vrijednosti kao što su cjelobrojne varijable. Daleko češće se koriste za alociranje niza vrijednosti.

Pogledati sljedeću deklaraciju:

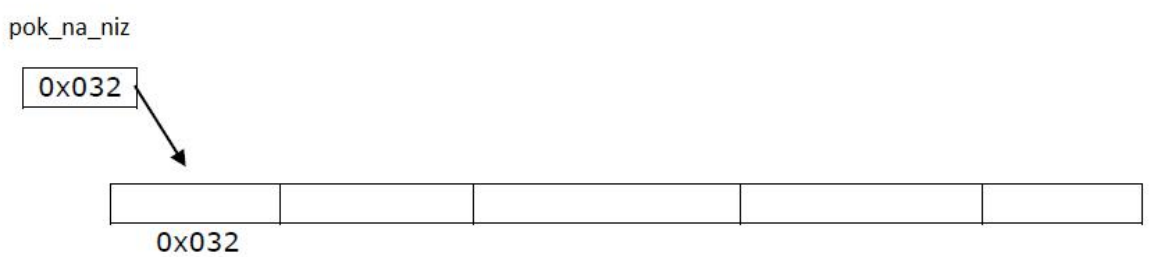
```
int Niz[10];
```

Vrijednost povezana s imenom Niz je adresa prvog elementa niza. Tip objekta Niz je niz od 10 cjelobrojnih vrijednosti. Takav tip podatka se može koristiti za alokaciju memorije potrebne za niz pomoću operacije `new`. Na primjer, iskazi:

```
int *pok_na_niz;

pok_na_niz = new int[10];
```

alociraju prostor za niz 10 cjelobrojnih vrijednosti. U prvom iskazu deklarirali smo varijablu pok_na_niz čija je vrijednost nedefinirana. Nakon izvršavanja drugog iskaza (pod pretpostavkom da ima dovoljno raspoložive memorije). Pok_na_niz sadrži adresu prvog elementa novoalociranog niza. Ako je adresa niza 0x032, grafički bi se moglo predstaviti ovako:



Vrijednost dinamički alociranog niza je adresa prvog elementa niza.

Znači, ako je adresa dinamički alociranog niza pohranjena u pokazivač, tada se prvom elementu niza može pristupiti pomoću pokazivača, na isti način na koji pristupamo prvom elementu statički alociranog niza koristeći njegovo ime i operator [].

Prvom elementu niza možemo pristupiti koristeći notaciju:

```
pok_na_niz[0];
```

drugom elementu:

```
pok_na_niz[1];
```

Vrijednost pokazivača je adresa prvog elementa niza, a za dani indeks i

```
pok_na_niz[i];
```

jednostavno se pristupa memorijskoj lokaciji `pok_na_niz + i`.

Prednost dinamičkog alociranja niza leži u činjenici da ne morate unaprijed znati veličinu niza.

Primjer

```
cout << "Koliko clanova niza zelite?";
cin >> broj_elemenata;
double *Pokazivac = new double[broj_elemenata];
for (int i = 0; i < broj_elemenata; i++)
     cin >> Pokazivac[i];
......
```

Memorijski blokovi koji više nisu potrebni mogu se "osloboditi" upotrebom operacije **`delete`**. "Oslobođena" memorija se zatim može ponovno dodijeliti pri sljedećem **`new`** zahtjevu. Operacije **`new`** i **`delete`** su komplementarne operacije.

Opći oblik operacije delete izgleda ovako:

```
delete pok;
```

ili

```
delete [] pok_na_niz;
```

Na primjer, ako pok ima sljedeću vrijednost:

```
int *pok=new int;
```

iskazom

```
delete pok;
```

će te "osloboditi" memorijski blok i omogućiti da bude kasnije ponovno upotrijebljen. Nakon toga vrijednost pokazivača će biti nedefinirana, te će dereferenciranje pokazivača

```
*pok
```

rezultirati nepredvidljivim rezultatima. Da bi se izbjegli ovakvi problemi poželjno je postaviti vrijednost pokazivača na nulu.

```
delete pok;

pok=0; //ili pok = NULL;
```

Slično tome, ako je vrijednost varijable pok_na_niz adresa prvog elementa dinamički kreiranog niza

```
double *pok_na_niz= new double[n];
```

memoriju možete "osloboditi" na sljedeći način:

```
delete [] pok_na_niz;

pok_na_niz=0;
```

8.1.1 Zadaci za vježbu

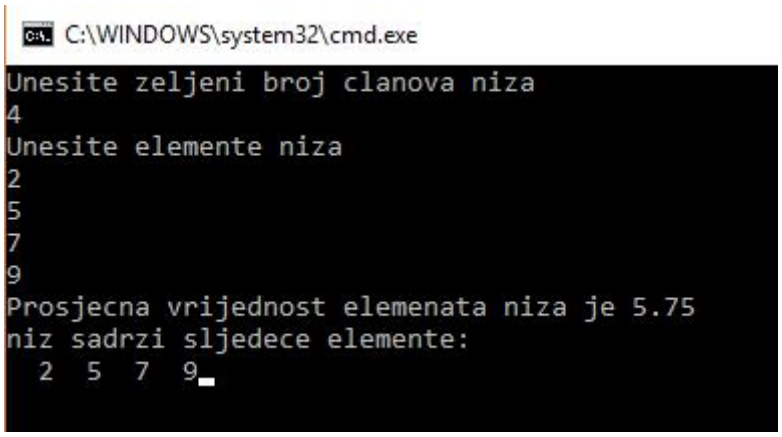
1. Napišite program u kojem ćete:

- Omogućiti korisniku da unese vrijednost N (broj članova niza);
- Alocirati niz operacijom new (članovi niza su tipa double);
- Omogućiti inicijalizaciju članova niza unosom vrijednosti s tastature (koristiti funkciju);
- Izračunati i ispisati prosječnu vrijednost elemenata niza (koristiti funkciju);
- Dealocirati memorijski blok upotrijebljen za kreiranje niza.

```
#include<iostream>
#include<iomanip>
using namespace std;
void unos(int *, int n);
float prosjek(int *, int);
void ispis(int *, int);
int main()
{
     int N, suma = 0;
     int *Pokazivac = NULL;
```

```
	cout << "Unesite zeljeni broj clanova niza" << endl;
	cin >> N;
	Pokazivac = new int[N]; //dinamicka alokacija jednodimenzionalnog
niza
	cout << "Unesite elemente niza" << endl;
	unos(Pokazivac, N);//poziv funkcije za unos elemenata niza
	cout << "Prosjecna vrijednost elemenata niza je " <<
prosjek(Pokazivac, N) << endl;
	cout << "niz sadrzi sljedece elemente: " << endl;
	ispis(Pokazivac, N);
	delete[]Pokazivac; //obavezno dealocirajte memoriju
	Pokazivac = NULL;
	system("pause>0");
	return 0;
}
void unos(int *pok, int n)// unos elemenata niza
{
	for (int i = 0; i<n; i++)
		cin >> pok[i];
}
float prosjek(int *pok, int n)//racunanje prosjeka
{
	int suma = 0;
	for (int i = 0; i<n; i++)
		suma += pok[i];
	return float(suma) / n;
}
void ispis(int * pok, int n)
{
	for (int i = 0; i<n; i++)
		cout << setw(3) << pok[i];
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



2. Napišite program u kojem će:
   - Korisnik odrediti veličinu x.
   - Alocirati dinamičke nizove A veličine x.
   - Alocirati dinamičke nizove B veličine x.
   - Korisnik treba unijeti elemente nizova A i B.
     - Kreirati funkciju ucitaj za upis članova niza.
   - Alocirati dinamičke nizove C veličine x
   - Sabrati elemente nizova A i B, te rezultat smjestiti u niz C
     - koristite funkciji saberi.
   - Ispisati elemente niza C
     - Kreirati funkciju ispis za ispis članova niza.

```
#include <iostream>
using namespace std;
void ispis(int* p, int x)
{
   for (int i = 0; i<x; i++)
        cout << i + 1 << ": " << p[i] << endl;
}
void ucitaj(int* p, int x)
{
   cout << "\nUnesite elemente niza: \n";
   for (int i = 0; i<x; i++)
   {
        cout << i + 1 << ": ";
        cin >> p[i];
   }
}
```

```
void saberi(int* a, int* b, int* c, int x)
{
   for (int i = 0; i<x; i++)
        c[i] = a[i] + b[i];
}
void main()
{
   int x;
   cout << "Koliko elemenata zelis unijeti: ";
   cin >> x;
   int* A = new int[x];
   int* B = new int[x];
   int* C = new int[x];
   ucitaj(A, x);
   ucitaj(B, x);
   saberi(A, B, C, x);
   cout << "\n===suma===\n";
   ispis(C, x);
}
```

3. Napišite program u kojem ćete:
   - Kreirati dinamički niz od onoliko cjelobrojnih elemenata koliko želi korisnik;
   - Omogućiti inicijalizaciju članova niza unosom s tastature;
   - Utvrditi koliko elemenata niza je manje od 0;
   - Kreirati novi niz čiji će elementi biti elementi prvog niza koji zadovoljavaju prethodni uvjet (manji su od 0);
   - Koristite funkcije.

```
#include <iostream>
#include <iomanip>
using namespace std;

int unos(int *, int);//funkcija vraca vrijednost jer ce osim unosa
elemenata i prebrojati koliko elemenata je manje od 0
void inicijalizacija_niza(int *, int*, int);//funkcija smjesta
negativne elemente iz prvog niza u drugi
```

```
void ispis(int *, int);

void main()
{
   int velicina1, velicina2;
   cout << "Unesite zeljenu velicinu niza" << endl;
   cin >> velicina1;
   int *Pokazivac1 = new int[velicina1];//alokacija prvog niza

   cout << "Unesite elemente niza" << endl;
   velicina2 = unos(Pokazivac1, velicina1);

   int *Pokazivac2 = new int[velicina2]; //alokacija drugog niza

   inicijalizacija_niza(Pokazivac1, Pokazivac2, velicina1);
   cout << "Prvi niz ima " << velicina2 << " elemenata manjih od
nula, a to su " << endl;
   ispis(Pokazivac2, velicina2);
   cout << endl;
   delete[] Pokazivac1; //obavezno dealocirajte memoriju
   delete[] Pokazivac2;
   Pokazivac1 = NULL;
   Pokazivac2 = NULL;
   system("pause>0");
}
int unos(int *pok, int vel)//funkciju za unos elemenata prvog niza
smo iskoristili da prebrojimo koliko elemenata zadovoljava trazeni
uvjet (manji od 0)
{
   int brojac = 0;
   for (int i = 0; i<vel; i++)
   {
        cin >> pok[i];
        if (pok[i]<0)
             brojac++;
   }
```

```
   return brojac;
}
void inicijalizacija_niza(int *pok1, int* pok2, int vel)
{
   int j = 0;
   for (int i = 0; i<vel; i++)
   {
        if (pok1[i]<0)
        {
             pok2[j] = pok1[i];
             j++;
        }
   }
}
void ispis(int * pok, int vel)
{
   for (int i = 0; i<vel; i++)
        cout << setw(4) << pok[i];
}
```

9 Unije

Unije su složeni tipovi podataka koji omogućavaju da se u isti memorijski prostor, u različitim trenucima, smještaju podaci različitih tipova. Unije se difinišu naredbom `union`, sve ostalo je kao i kod struktura, navode se polja koja su različitog tipa.

Za razliku od struktura, kod kojih sva obilježlja imaju definisanu vrijednost, kod unije samo jedno obilježlje u datom trenutku ima definisanu vrijednost, obilježlje kojem je posljednje dodijeljena vrijednost. Ako se inicijalizira unija, u zagradama se stavlja samo jedna vrijednost i ako se ne navede čija je vrijednost, dodeljuje se prvom obilježlju.

Unije možemo koristiti na sljedećim lokacijama:

- Podijeliti jednu memorijsku lokaciju za varijablu i koristiti istu lokaciju za drugu varijablu različitog tipa podataka.
- Ne znamo koju vrsta podataka treba proslijediti nekoj funkciji, tad prosljeđujemo uniju koja sadrži sve moguće vrste podataka.

9.1 Definicija unije

Unija se definiše ključnom riječju `union`, praćenom listom varijabli sadržanih u vitičastom zagradama.

```
union ime_unije {
      int element1;
      float element2;
      char element3;

};
```

Unije se koriste na gotovo identičan način kao i strukture, ali se ne smije zaboraviti da svi atributi unije dijele isti memorijski prostor

Primjer:

```
union Tezina{
      long gram;
      int kilogram;
      float tona;
```

```
};
void main()
{
	Tezina tezina; //deklaracija varijable (objekta) unije
	tezina.tona = 5.10;
	//tezina u tonama vise ne potrebna, ali i dalje koristimo isti
memorijski prostor
	tezina.kilogram = 50;
}
```

9.2 Deklaracija unije

Uniju možemo deklarisati na različite načine. Uzimajući gornji primjer, možemo deklarirati gore definisanu uniju kao:

```
union Tezina{
	long gram;
	int kilogram;
	float tona;
}tezina;
```

Ili

```
Tezina tezina;
```

Što znači da je varijabla `tezina` tipa `Tezina`.

9.2 Pristup članovima unije

Članu ili objektu unije se može pristupiti na sličan način kao i objektu strukture. Pretpostavimo, da želimo pristupati varijabli `kilogram` za varijablu `tezina` u gornjem primjeru:

```
tezina.kilogram
```

9.3 Razlika između unije i strukture

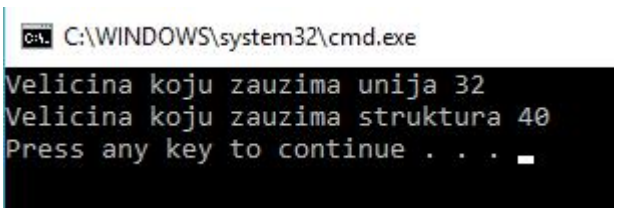
Iako su unije na sličan način slične strukturama, razlika između njih je ključna za razumijevanje. To se može pokazati ovim primjerom:

```
#include <iostream>
using namespace  std;
union posao { //definicija unije
```

```
        char ime[32];
        float plata;
        int id_radnika;
}u;
struct posao1 {
        char ime[32];
        float plata;
        int id_radnika;
}s;
int main() {
        cout<<"Velicina koju zauzima unija "<<sizeof(u)<<endl;
        cout<<"Velicina koju zauzima struktura " << sizeof(s)<<endl;
        return 0;

}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Postoji razlika u dodjeli memorije između unije i strukture, kao što je navedeno u gornjem primjeru. Količina memorije koja je potrebna za pohranu objekata iz strukture je zbir veličine memorije svih članova.



Slika 1.Memorijska alokacija u slučaju struktura

Ali, memorije potrebne za čuvanje objekata iz unije je memorija potrebna za najveći objekat unije.

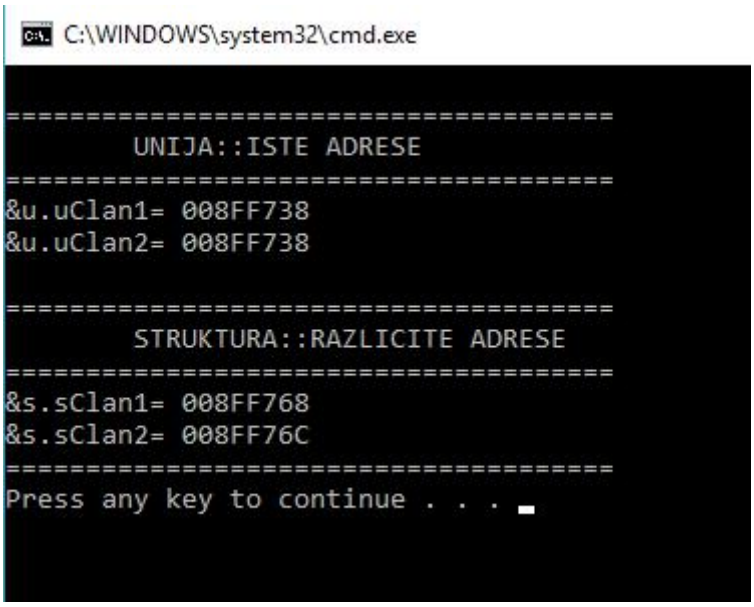


Slika 2. Memorijska alokacija u slučaju unije

U nastavku je prikazan primjer koji demonstrira jednu od glavnih razlika između struktura i unija.

```
#include <iostream>
using namespace std;
char crt[] = "\n=======================================\n";
struct Struktura {
	int sClan1;
	float sClan2;
};
union Unija {
	int uClan1;
	float uClan2;
	float niz[10];
};
void main() {
	Struktura s;
	Unija u;
	cout << crt << "\tUNIJA::ISTE ADRESE" << crt;
	cout << "&u.uClan1= " << &u.uClan1 << endl;
	cout << "&u.uClan2= " << &u.uClan2<<endl;
	cout << crt << "\tSTRUKTURA::RAZLICITE ADRESE" << crt;
	cout << "&s.sClan1= " << &s.sClan1 << endl;
	cout << "&s.sClan2= " << &s.sClan2 << crt;}
```

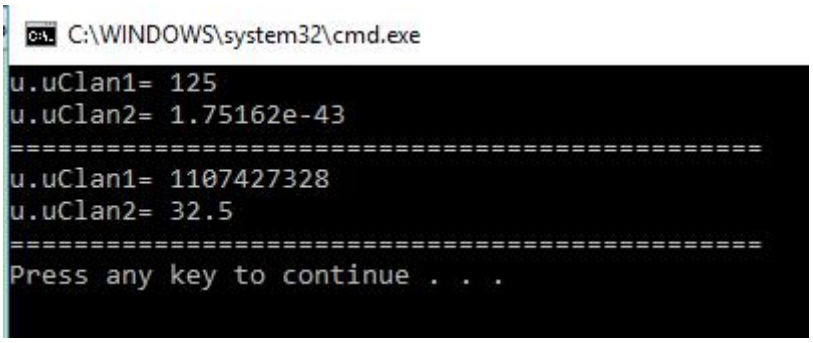
Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



Pošto atributi unije koriste isti memorijski prostor, mogli bismo očekivati da određenoj vrijednosti možemo pristupiti preko bilo kojeg atributa unije. Ipak, ovakvo očekivanje nije ispravno, što možete pogledati u sljedećem primjeru.

```
#include <iostream>
using namespace std;
char crt[] = "\n=============================================\n";
union Unija {
	int uClan1;
	float uClan2;
};
void main() {
	Unija u;
	u.uClan1 = 125;//inicijalizujemo clan1
	cout << "u.uClan1= " << u.uClan1 << endl;//OK
	cout << "u.uClan2= " << u.uClan2 << crt;//neocekivano
	u.uClan2 = 32.5;//inicijalizujemo clan2
	cout << "u.uClan1= " << u.uClan1 << endl;//neocekivano
	cout << "u.uClan2= " << u.uClan2 << crt;//OK
}
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:


C:\WINDOWS\system32\cmd.exe
u.uClan1= 125
u.uClan2= 1.75162e-43
==========
u.uClan1= 1107427328
u.uClan2= 32.5
==========
Press any key to continue . . .

10 Enumeracija

Enumeracija ili kako je programeri skraćeno zovu enums su u stvari konstante nabrajanja koje programer definiše. Enumeracija je korisnički definiran tip podataka koji se sastoji od integralnih konstanti. Uglavnom se koristi zato što programerima olakšava čitanje, pisanje, identifikovanje i održavanje koda, nego što umanjuje kod. Enumeracija se deklariše ključnom riječju `enum` sve više kao i struktura, van funkcija, iako u starim projektima ćete često nailaziti na enumeraciju unutat `main()` funkcije. Sintaksa za enumeracija je:

```
enum ime_enumeracije
{
      vrijednost1,
      vrijednost2,
      . . .
}
```

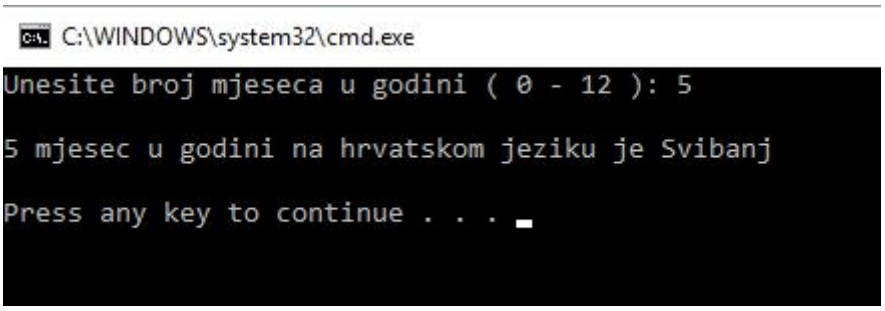
Najbolje da pogledate primjer kako se koristi enumeracija. Npr. u hrvatskom jeziku se nazivi mjeseci u godini pišu i govore drugačije od cijelog svijeta, pa često ljudi nisu sigurni kad neko kaže neki mjesec na hrvatskom jeziku bez prevoda; na koji mjesec misli. Ali ako uradite nešto ovako poput ovog programa koji koristi enumeraciju mjeseci, stvari postaju jasnije. Pogledajte kod programa.

```
#include <iostream>
#include <string>
using namespace std;
enum mjeseci
{
      Januar = 1,
      Februar,
      Mart,
      April,
      Maj,
      Juni,
      Juli,
      August,
      Septembar,
      Oktobar,
```

```
      Novembar,
      Decembar
};

int main()
{
      int broj;
      string mjesec;
      cout << "Unesite broj mjeseca u godini ( 0 - 12 ): ";
      cin >> broj;
      switch (broj)
      {
      case Januar: mjesec = "Sijecanj";break;
      case Februar: mjesec = "Veljaca";break;
      case Mart:     mjesec = "Ozujak";  break;
      case April:    mjesec = "Travanj"; break;
      case Maj: mjesec = "Svibanj"; break;
      case Juni:     mjesec = "Lipanj"; break;
      case Juli:     mjesec = "Srpanj";  break;
      case August: mjesec = "Kolovoz";break;
      case Septembar:     mjesec = "Rujan"; break;
      case Oktobar: mjesec = "Listopad";break;
      case Novembar: mjesec = "Studeni"; break;
      case Decembar: mjesec = "Prosinac";break;
      default: cout << "Unijeli ste broj za mjesec koji ne postoji";
break;
      }
      cout << endl << broj << " mjesec u godini na hrvatskom jeziku je
" << mjesec << endl << endl;
           system("PAUSE");
      return 0;
      }
```

Kada se gore navedeni kod kompajlira i izvrši, on proizvodi sljedeći rezultat:



U ovom program ste mogli enumeraciju i da definišete ovako:

```
enum { Januar = 1,Februar,Mart,April,Maj,Juni,Juli,August,
Septembar,Oktobar,Novembar,Decembar }mjeseci;
```

Verovatno ste primijetili da član enumeracije Januar ima dodijeljenu vrijednost 1. To znači da enumeracija `mjeseci` broji članove od 1, inače enumeracije se broje od 0. Vi možete dodijeliti brojne vrijednosti enumeracije svakom članu kako god hoćete.

11 Rad s datotekama

Datoteka je skup podataka koji su snimljeni na neku formu trajne memorije(hard disk, CD-ROM, floppy disk, ...). Datoteci se pristupa preko njenog imena (filename), u čijem sastavu se obično nalazi ekstenzija (dio imena iza tačke), koja označava tip podataka u datoteci. Postoje dvije vrste datoteka: tekstualne i binarne. Tekstualne datoteke sadrže tekst, dok binarne mogu sadržati i kompleksne vrste podataka, kao što su slike, izvršni programi, baze podataka, itd. Tekstualnim datotekama je nešto jednostavnije pristupiti, pisati podatke u njih, te čitati sa njih.

Najvažnija stvar svake aplikacije je da negdje čuva informacije i da koristi iste po potrebi. Inače rezultati vaše aplikacije bi trajali samo dok radi program; čim bi ste ga pokrenuli ponovo, ne bi ste mogli vidjeti informacije koje ste prije unosili. Čak i najbanalnija igrica mora negdje da pamti rezultate igrača, međutim RAM memorija nije postojana i ona traje dok se računar ne ugasi. Zato se podaci čuvaju u nekom fajlu ili u bazi podataka. Kad C++ programeri pričaju o tokovima ili drugačije rečeno; streams – strimovima; to znači da pričaju o nečemu gdje mogu da se učitaju ili upisuju podaci. Strimovi u suštini obezbjeđuju jednostavan rad i sa unosom i sa čitanjem podataka u tekstualni ili binarni fajl. Prvo je potrebno da omogućite podršku operacijama fajl tokova direktivom #include <fstream> koja vam omogućava korišćenje objekata ifstrem i ofstream koji regulišu otvaranje i zatvaranje datoteka.

Rad s fajlovima zahtijeva malo kompleksnije operacije nego što je to slučaj sa standardnim ulazima i izlazima (cin>> i cout<<). Prije svega, u zavisnosti od toga šta želimo uraditi, moramo kreirati odgovarajući stream objekat. Kada određeni sadržaj želimo upisati u fajl tada koristimo objekat klase `ofstream`, dok za ispis sadržaja nekog fajla koristimo objekat klase `ifstream`.

11.1 Standardna biblioteka-fstream

Da bismo mogli manipulisati fajlovima u naš program je potrebno uključiti `fstream` biblioteku unutar koje se nalaze `ifstream` i `ofstream` klase. Standardna biblioteka fstream (f se odnosi na datoteku, tj. file), koja omogućava pisanje na i čitanje sa datoteka. Ovo se postiže pozivanjem sadržaja fstream sa:

```
#include<fstream>
```

Datoteka `fstream` definiše tri nova tipa podataka:

- `ofstream` – Ovaj tip podataka predstavlja stream za izlazne datoteke (of odnosi se na output). Pravac izlaza je sa programa na datoteku. Ovaj tip podataka se koristi za kreiranje datoteka i pisanje informacija na njih. Ne može se koristiti za čitanje datoteka.
- `ifstream` - Ovaj tip podataka predstavlja stream za ulazne datoteke (i odnosi se na input). Pravac ulaza je sa datoteke prema programu. Ovaj tip podataka se koristi za čitanje informacija sa datoteka. Ne može se koristiti za kreiranje datoteka i pisanje u njih.
- `Fstream` – Ovaj tip podataka predstavlja stream za datoteke, ima karateristike i ofstream i ifstream . On može kreirati datoteke, pisati na njih i čitati sa njih.

11.2 Pristup datotekama

Kada program pristupa datotekama, bez obzira da li ih čita, ili u njih upisuje, ili oboje, prolazi kroz sljedeće korake:

- Datoteka prvo mora da se otvori. Ovo otvara put u komunikaciji između datoteke i stream objekta u programu (fstream ,ifstream, ofstream), koji se koristi u pristupu datoteci.
- Nakon otvaranja program čita sa datoteke, piše na nju, ili čini oboje.
- Na kraju, program zatvara datoteku. Ovo je bitan korak, pošto održavanje komunikacije između datoteke i stream objekta zahtjeva resurse, tako da zatvaranje datoteke oslobađa ove resurse kada više nisu potrebni. Uz to, postoji mogučnost da se kasnije u programu ne može pristupiti datoteci ako nije zatvorena prije predhodnog pristupa.

Bez obzira da li se sadržaj datoteke treba pročitati ili se u datoteci treba upisati neki podaci, datoteka prvo treba da se otvori.

11.3 Otvaranje datoteke za pisanje

Datoteke za pisanje se mogu otvoriti pomoću `fstream` i `ofstream` objekata na dva načina:

- pomoću metode (member function) `open`, ili
- pomoću konstruktora (constructor).

11.3.1 Otvaranje pomoću funkcije open

Prvi argument funkcije `open` je ime i lokacija datoteke koja se treba otvoriti. Međutim, moguće je dodati i drugi argument zavisno od toga da li se `fstream` i ofstream poziva funkcijom `open` , ili se želi neki drugi modul od onog koji se daje. Datoteka u koju želimo upisivati podatke ne mora postojati. U slučaju da ne postoji, ona će se automatski kreirati pod imenom i na lokaciji koju smo upisali. Lokacija se može dati kao relativna(relative path) i apsolutna (absolute path). Pri tome,

relativni put predstavlja lokaciju u odnosu na naš program, tj. datoteka se stvara u direktorijumu u kojem se nalazi i naš program. Ovo postižemo na sljedeći način:

```
ofstream izlaz;

izlaz.open(„polaznici.txt“);
```

Za razliku od relativnog puta, apsolutni put predstavlja lokaciju koja započinje slovom drajva,sadrže i sve direktorijume i poddirektorijume dok se ne dođe do datoteke. Na primjer, ako je datoteka polaznici.txt u folderu Kursevi , a ovaj je podirektorium direktoriuma KCKF , a sve se nalazi na tvrdom disku sa slovom C, onda bi se datoteka otvorila na sljedeći način:

```
ofstream izlaz;

izlaz.open(„C:\\KCKF\\Kursevi\\polaznici.txt“);
```

Koriste se po dva znaka \\, jer samo jedan između navodnika predstavlja escape-sekvencu. Bez obzira da li koristimo relativni ili apsolutni put, argument za funkciju open ne mora da bude neko ime (riječ), nego i (string) varijabla.

Primjer:

```
ofstream izlaz;

char imeDatoteke[80];

cout << „Unesite ime datoteke: „;

cin >> imeDatoteke;

izlaz.open(imeDatoteke);
```

Važno je zapamtiti da je korištenje relativnog puta bolja opcija, jer se može desiti da neki folder koji se navodi u apsolutnoj putanji ne postoji, naročito ako se program koristi na nekom drugom računaru (sa drugačijim folderima).

Korištenje drugog argumenta u funkciji open definiše modul u kojem se datoteka treba otvoriti.

Ako za otvaranje datoteke koristimo ofstream objekat, onda ne moramo koristiti dodatne argumente, ali treba zapamtiti da u tom slučaju možemo samo upisivati informaciju u datoteku, ali ne i čitati sa nje. Ako želimo, na primjer, da pratimo greške izazvane pokretanjem nekog programa i sačuvamo sve zapise o tome, koristimo opciju ios::app , tj.

```
ofstream izlaz;

izlaz.open(„polaznici.txt", ios::app);
```

Međutim, ako se za otvaranje datoteke za pisanje koristi fstream objekat, treba se dodati još jedan argument, tj. opcija za pisanje (ios::in). U ovom slučaju bi bilo:

```
fstream izaz;

izlaz.open(„polaznici.txt", ios::out)
```

11.4 Otvaranje datoteka za čitanje

Sve što je rečeno može se primijeniti i na otvaranje datoteka za čitanje. Jedina razlika je što se, uz korištenje objekta fstream, umjesto objekta ofstream koristi iostream objekat. Uz to, datoteka sa koje se čita mora postojati, jer se pokretanjem jednog od prethodnih objekata ne kreira datoteka.

Otvaranje datoteke za čitanje:

```
ifstream ulaz;

ulaz.open(„polaznici.txt");

fstream ulaz;

ulaz.open(„polaznici.txt", ios::in);//obavezno dodati argument ios::in

ifstream ulaz(„polaznici.txt");

fstream ulaz(„polaznici.txt", ios::in)
```

Otvaranje datoteka za čitanje i pisanje:

Kao što je ranije rečeno, objekat fstream se može koristiti za otvaranje datoteka i za pisanje i za čitanje. U tu svrhu koristi se sljedeća sintaksa:

```
fstream izlazUlaz;

izlazUlaz.open(„polaznici.txt", ios::in | ios::out);
```

U oba primjera korišten je tzv.bitwiseoperator (|), koji ima isto značenje kao logički operator ||(ili).

11.5 Zatvaranje datoteka

Svaka otvorena datoteka se treba zatvoriti prije nego se napusti program. To je zbog toga što svaka otvorena datoteka zahtjeva sistemske resurse. Osim toga, neki operativni sistemi imaju ograničen broj otvorenih datoteka kojima se ne 'manipuliše'. Zatvaranje datoteka se vrši pomoću funkcije `close`. Primjeri pokazuju njenu upotrebu pri zatvaranju datoteka za pisanje i čitanje:

```
ofstream izlaz;

izlaz.open(„polaznici.txt“);

// skup naredbi

outfile.close();

 ifstream ulaz;

ulaz.open(„polaznici.txt“);

// skup naredbi

ulaz.close();
```

11.6 Upis u datoteku

Pisanje podataka na datoteku se izvodi pomoću operatora za ispisivanje (<<), kao što je to bio slučaj sa ispisivanjem na ekran (cout <<). Jedina razlika je u tome što se ovde koristi `fstream` ili `iostrem` objekat, a ne cout objekat. Sljedeći program pokazuje primjer upisivanja podataka u datoteku polaznici.txt:

```
#include <fstream>
#include <iostream>
using namespace std;
int main()

{
      char podaci[80];
      ofstream izlaz;
      izlaz.open("polaznici.txt");
      cout << "Zapisivanje u datoteku" << endl;
      cout << " == == == == == == == == == == == = " << endl;
```

```
	cout << "Upisite grupu "<<endl;
	cin.getline(podaci, 80);
	izlaz << podaci << endl;
	cout << "Unesite ID polaznika: " <<endl;
	cin >> podaci;
	cin.ignore();
	izlaz << podaci << endl;
	izlaz.close();
	return 0;
}
```

Upotrebom drugog argumenta pri otvaranju datoteke za pisanje (ios::app) možemo dodavati sadržaj na postojeću datoteku kao što to pokazuje sljedeći primjer. U ovom primjeru zapisivanje se prekida nakon unosa znakova ***.

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
int main()

{
	string x;
	ofstream izlaz;
	izlaz.open("podaci.txt", ios::app);
	while (x!= "***")

	{
		cout << "Unesite neki tekst (za kraj unesite ***) :" << endl;
		cin >> x;
		izlaz << x << endl;
	}
		izlaz.close();

}
```

11.7 Čitanje iz datoteka

Čitanje podataka iz datoteka obavlja se pomoću operatora za čitanje (>>) kao što je to slučaj sa ispisivanjem sa tastature (cin>>). Sljedeći primjer nadopunjava onaj iz prethodnog poglavlja, tj. nakon što korisnik upisuje informacije na datoteku, program čita iste podatke i ispisuje ih na ekran.

```
#include <fstream>
#include <iostream>
#include<string>
using namespace std;
int main()

{
	string podaci;
	ofstream izlaz;

	izlaz.open("polaznici.txt");
	cout << "Upisivanje u datoteku"<< endl;
	cout << " == == == == == == == == == = " << endl;
	cout << "Unesite grupu "<<endl;
	getline(cin, podaci);
	izlaz << podaci << endl;
	cout << "Unesite ID polaznika "<<endl;
	cin >> podaci;
	cin.ignore();
	izlaz << podaci << endl;
	izlaz.close();
	ifstream ulaz;
	cout << "Citanje iz datoteke" << endl;
	cout << "== == == == == == == == == = " << endl;
	ulaz.open("polaznici.txt");
	getline(ulaz, podaci);
	cout << podaci << endl;
	getline(ulaz, podaci);
	cout << podaci << endl;
	ulaz.close();
	return 0;
```

```
}
```

Treba voditi računa, da fajl iz kojeg čitamo mora postojati. Drugim riječima, treba voditi računa o slučajevima u kojima program nije u mogućnosti otvoriti zahtijevani fajl (bilo da fajl ne postoji, da je oštećen, da nemamo privilegije za pristup i sl.). Ako fajl iz bilo kojih razloga nije moguće otvoriti na raspolaganju je nekoliko mehanizama koji se mogu iskoristiti za provjeru.

Jedan od pomenutih mehanizama svakako je korištenje funkcije `fail()`. Pomenuta funkcija vraća vrijednost true ukoliko postoji neki problem prilikom otvaranja fajla tj.vrijednost false ukoliko je fajl uspješno otvoren. Dakle, provjeru možemo vršiti i na sljedeći način:

```
#include <fstream>
#include <iostream>
#include<string>
using namespace std;
int main()

{
	string podaci;
	ofstream izlaz;

	izlaz.open("polaznici.txt");
	cout << "Upisivanje u datoteku" << endl;
	cout << " == == == == == == == == == == = " << endl;
	cout << "Unesite grupu " << endl;
	getline(cin, podaci);
	izlaz << podaci << endl;
	cout << "Unesite ID polaznika " << endl;
	cin >> podaci;
	cin.ignore();
	izlaz << podaci << endl;
	izlaz.close();
	ifstream ulaz;
	if (ulaz.fail())
		cout << "Greska, nemoguce otvoriti zahtjevani fajl" << endl;
	else
	cout << "Citanje iz datoteke" << endl;
	cout << "== == == == == == == == == = " << endl;
```

```
    ulaz.open("polaznici.txt");
    getline(ulaz, podaci);
    cout << podaci << endl;
    getline(ulaz, podaci);
    cout << podaci << endl;
    ulaz.close();
    return 0;
}
```