Datum: 18.03.2015.

POKAZIVAČI

Pokazivač je podatkovni objekt koja čuva **memorijsku adresu**. Pokazivač deklariramo 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++.

Na primjer:

```
int *p;
```

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;
```

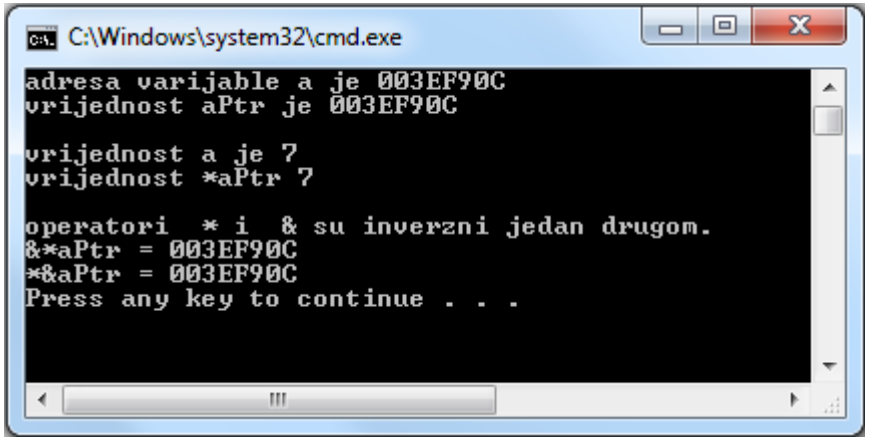
Ukoliko je prethodno deklariran pokazivač p vrijednost pokazivača se inicijalizira na sljedeći način:

```
p = &objekt;
```

gdje je:
p – identifikator (ime) pokazivača
& - operator reference (ampersand) koji vraća adresu objekta
objekt – objekt na koji pokazivač upućuje.

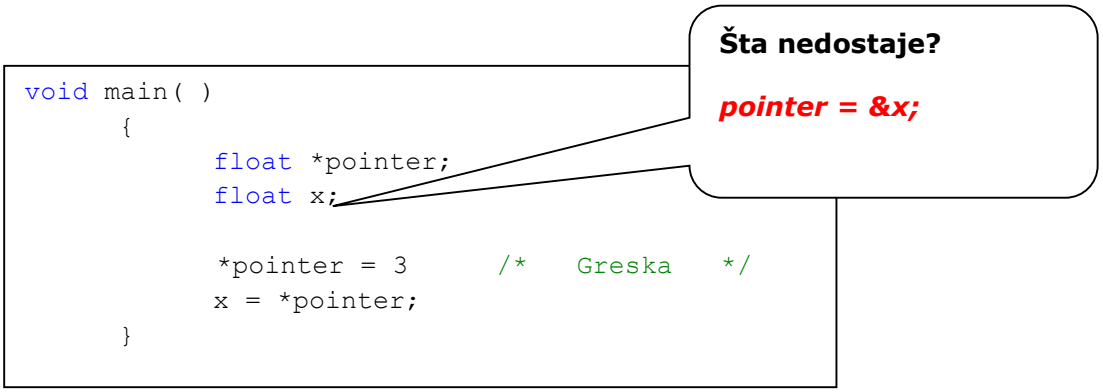
Operator adrese **&** i operator derefenciranja ***** su **inverzni operatori** što pokazuje sljedeći primjer:

```
// 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;
}
```



Nije dozvoljeno dereferncirati pokazivač koji nigdje ne pokazuje!

Na primjer:



Pokušajte uraditi sljedeći zadatak:

Zadatak 1
Napišite program u kojem ćete:

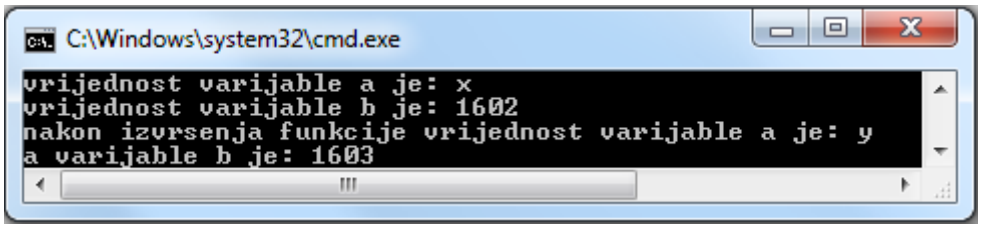
a) Deklarirati dvije varijable tipa float (broj1 i broj2).
b) Inicijalizirati varijablu broj1 vrijednošću 7.3.
c) Deklarirati pokazivač na tip podatka float.
d) Inicijalizirati pokazivač na adresu varijable broj1.
e) Ispisati vrijednost varijable na koju upućuje pokazivač koristeći dereferenciranje pokazivača.
f) Inicijalizirati vrijednost varijable broj2 na istu vrijednost na koju upućuje pokazivač, te ju ispisati.
g) Ispisati adresu varijable broj1.

(**Rješenje zadatka se nalazi na kraju dokumenta**.)

POKAZIVAČI TIPA VOID

Pokazivači tipa void su posebna vrsta pokazivača. U C++ rezervirana riječ void ukazuje na odsustvo tipa, tako da se može reći:
Void pokazivači, su pokazivači koji pokazuju na vrijednost koja nema tipa (odnosno na vrijednost koja ima neodređena svojstva dereferenciranja i čija dužina nije određena).
Ovo omogućava void pokazivačima da pokazuju na bilo koji tip podatka (cjelobrojni, realni, karakter …). No, void pokazivači imaju i ograničenje: vrijednosti na koje pokazuju void pokazivači ne mogu se direktno dereferencirati. Budući da nema vrijednosti koja se dereferencira, nužno je promijeniti void pokazivač u neki drugi tip pokazivača prije nego ga dereferenciramo. To se postiže upotrebom type-casting operatora kao u sljedećem primjeru:

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

void povecaj(void* podatak, int velicina)
{
  switch (velicina)
  {
    case sizeof(char) : (*((char*)podatak))++; break;
    case sizeof(int) : (*((int*)podatak))++; break;
  }
}

int main ()
{
  char a = 'x';
  int b = 1602;
  cout<<"vrijednost varijable a je: "<<a<<endl;
  cout<<"vrijednost varijable b je: "<<b<<endl;
  povecaj(&a,sizeof(a));
  povecaj(&b,sizeof(b));
  cout<<"nakon izvrsenja funkcije vrijednost varijable a je: "<<a<<
          "\na varijable b je: "<<b<<endl;

  return 0;
}
```



POKAZIVAČI I KONSTANTE

Kada je u pitanju odnos pokazivača i konstanti postoje tri situacije:

- **promjenljivi pokazivač na konstantnu vrijednost**

Primjer:

```
const int x=5;
const int* pX;   // pomjenjivi pokazivac na konstantni int
int nestoDrugo;
pX=&x;           //pokazivacu je dodijeljanavrijednost - adresa varijable x
*pX = 3;        // nije dozvoljeno -  ne mozete koristiti pX za mofifikaciju int vrijednosti
pX =&nestoDrugo;// dozvoljeno - pokazivac pX moze pokazivati na nesto drugo
```

- **konstantan pokazivač na pomjenjljivu vrijednost**

Primjer:

```
int Y;
int* const pY=&Y;             // konstantan pokazivac na promjenjivi int
*pY = 4;                    // dozvoljeno - moguce je koristiti pY za modifikaciju int
vrijednosti
pY = &someOtherIntVar;      // nije dozvoljeno - ne mozete usmjeriti pokazivac pY na nesto drugo
```

- **konstantan pokazivač na konstantu vrijednost**

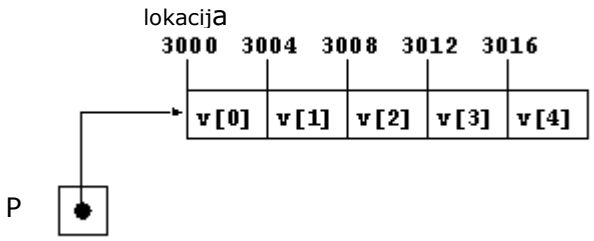
Primjer:

```
const int Z=9;
const int* const pZ=&Z;        // konstantan pokazivac na konstantan int
*pZ = 5;                    // nije dozvoljeno - ne mozete koristiti pZ za modifikaciju int
vrijednosti
pZ = &someOtherIntVar;      // nije dozovoljeno - ne mozete usmjeriti pokazivac pZ na nesto drugo
```

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, i oduzimati od drugih pokazivača.

Pretpostavimo da smo deklarirali 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:

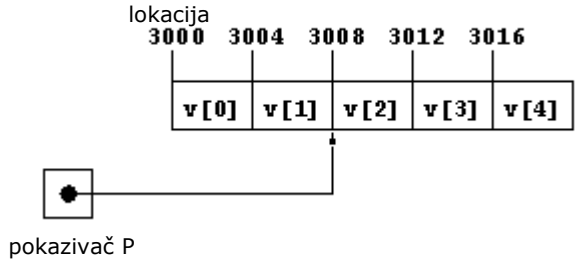
```
P=v; // ime niza upućuje na adresu prvog elementa niza
P=&v[0];
```

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 2
Napišite program u kojem ćete:

a) deklarirati 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) deklarirati pokazivač koji pokazuje na objekt tipa double
d) inicijalizirate pokazivač na adresu prvog elementa niza
(pokušajte ovu inicijalizaciju napraviti na dva načina)
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.

(Rješenje zadatka se nalazi na kraju dokumenta.)

UPOTREBA POKAZIVAČA

U dosadašnjim primjerima dodjeljivali smo adrese varijabli pokazivaču i koristili vrijednosti na koje pokazivač upućuje korištenjem operatora dereferenciranja. U praksi ćete ovako nešto raditi rijetko. Razlog za ovu vrstu manipulacije pokazivačima je demonstracija načina na koji pokazivači funkcioniraju.

Pokazivače ćete najčešće upotrebljavati za:
- za upravljanje podacima na slobodnom skladištu (free store) tj. dinamičkoj memoriji,
- za pristupanje podacima članovima i funkcijama članicama struktura i
- za prosljeđivanje parametara funkcijama.

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.

Zadatak 3
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 cube ( int * );
```

(**Rješenje zadatka se nalazi na kraju dokumenta**.)

Zadatak 4
Napišite program u kojem ćete deklarirari niz od 5 cijelobrojnih elemenata te uz pomoć:

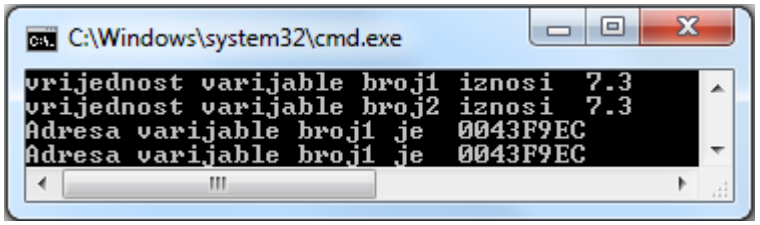
- funkcije: `void unos (int *, int);` omogućiti unos elemenata niza
- funkcije: `void ispis (int *, int); omogućiti ispis elemenata niza`
- funkcije: `int * najveci (int *, int);` vratiti adresu najvećeg elementa u nizu; obavezno provjerite je li funkcija vratila korektnu adresu.

(**Rješenje zadatka se nalazi na kraju dokumenta**.)

Rješenja:

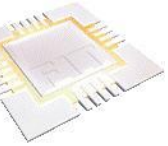
Zadatak 1

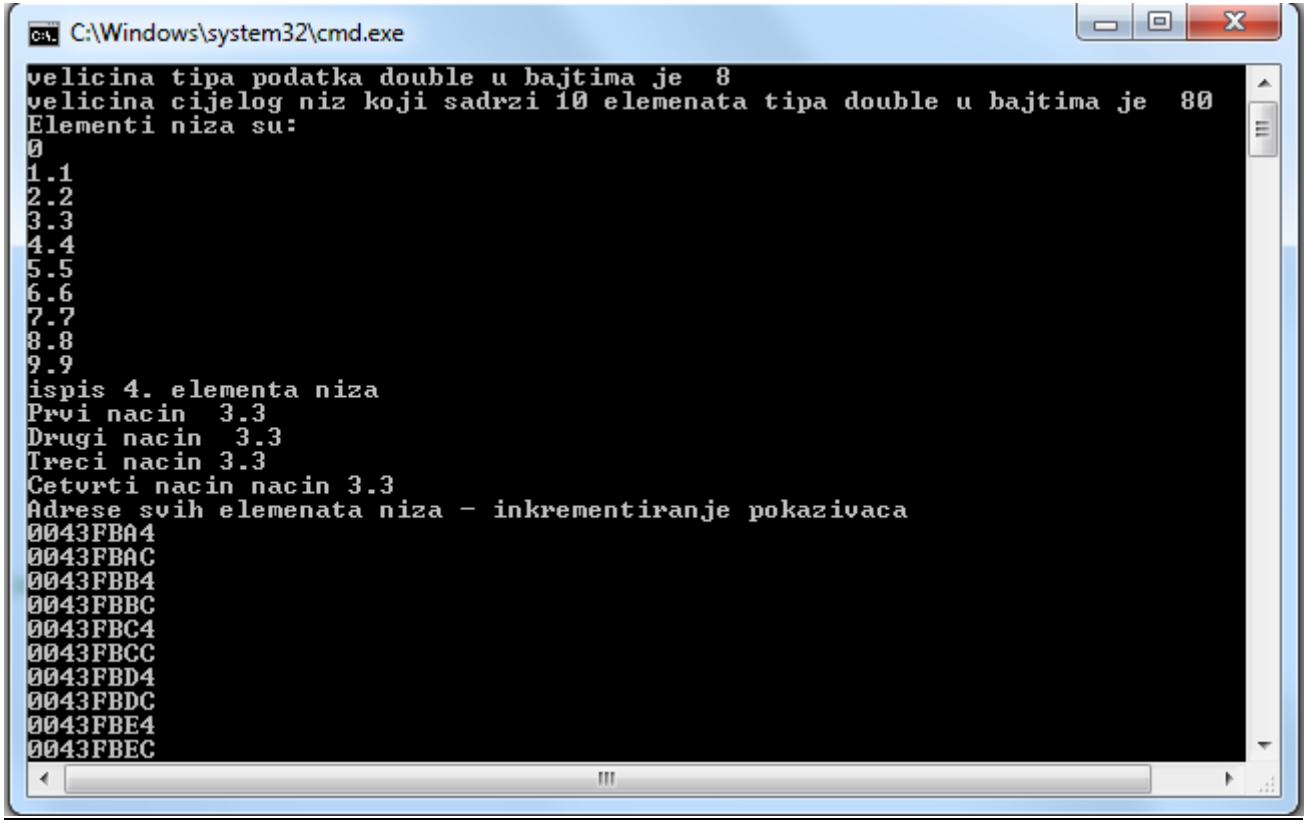
```
#include <iostream>
using namespace std;

int main()
{
	float broj1, broj2;
	broj1=7.3;//inicijalizacija vrijednosti varaijble 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;
}
```

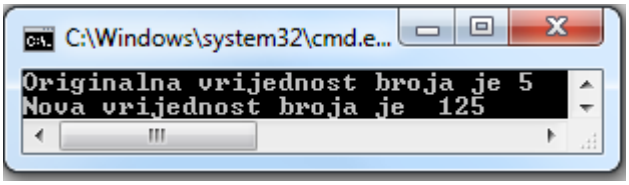


Zadatak 2

```
#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;
}
```




C:\Windows\system32\cmd.exe
velicina tipa podatka double u bajtima je 8
velicina cijelog niz koji sadrzi 10 elemenata tipa double u bajtima je 80
Elementi niza su:
0
1.1
2.2
3.3
4.4
5.5
6.6
7.7
8.8
9.9
ispis 4. elementa niza
Prvi nacin 3.3
Drugi nacin 3.3
Treci nacin 3.3
Cetvrti nacin nacin 3.3
Adrese svih elemenata niza - inkrementiranje pokazivaca
0043FBA4
0043FBAC
0043FBB4
0043FBBC
0043FBC4
0043FBCC
0043FBD4
0043FBDC
0043FBE4
0043FBEC



Zadatak 3

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

void cube ( int * );    // prototip funkcije

int main()
{
int broj = 5;
cout << "Originalna vrijednost broja je " << broj;
cube(&broj); //parametar funkcije je adresa varijable broj
cout << "\nNova vrijednost broja je  " << broj << endl;
system ("pause>null");
return 0;
}

void cube ( int * pokazivac )
{
     *pokazivac = *pokazivac * *pokazivac * * pokazivac;
//razlikovati operator dereferenciranja od operatora mnozenja
}
```

Zadatak 4

```
#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;//dereferncirамo 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;
}
```