Objektno orjentisano programiranje – C++

Nizovi, stringovi i pokazivači

Teme

- Jednodimenzioni nizovi
- Višedimenzioni nizovi
- Stringovi
- Inicijalizacija stringova
- Nizovi stringova
- Pokazivači
- Dinamička alokacija memorije

Nizovi

Nizovi

- Niz je skup promenljivih istog tipa kojima se pristupa preko zajedničkog imena
- Opšti oblik deklaracije niza je **tip ime[veličina]**
- Niz može biti jednodimenzioni ili višedimenzioni
- Pojedinačnim elementima niza se pristupa preko indeksa
- Indeks prvog elementa niza je 0
- Elementi niza zauzimaju susedne lokacije u memoriji

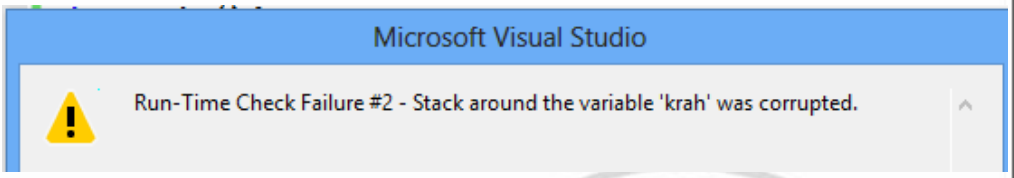
Nizovi (2)

```
#include <iostream>
using namespace std;
int main(){
    int myArray[5], i;
    for (i=0; i<5; i++) {
        cout << "Vrednosti niza myArray[" << i << "]: "<<endl;
        cin >> myArray[i];
    }
    for (i = 0; i<5; i++)
        cout << i << ": " << myArray[i] << "\n";
    return 0;
}
```

Nizovi (3)

- C++ ne proverava granice niza zbog efikasnosti, što je često uzrok krahova u programima.

```
int krah[10];
for (int i=0;i<100;i++)
    krah[i] =i;
```



- Dužnost je programera da obezbedi da u niz može da stane potreban broj elemenata i da proveri granice (engl. bounds checking) kad god je to neophodno.

Nizovi (4)

•Direktno kopiranje nizova nije dozvoljeno

```
int a[10], b[10];
 a=b; //greška
```

- Ispravan način kopiranja niza.

```
int a[5] = {1,2,3,4,5}, b[5];
for (int i=0; i<5;i++) {
    b[i]=a[i];
    cout<< "b["<<i<<"] ="<<b[i]<<endl;
}
```

Višedimenzioni nizovi

tip ime[dimenzija1][dimenzija2]...[dimenzijan]

- Najjednostavniji oblik je 2D niz (matrica), koji je zapravo niz 1D nizova
- Pošto niz rezerviše memoriju za sve elemente, nizovi sa više od 3D se vrlo retko koriste jer bi ubrzo “pojeli” svu raspoloživu memoriju

  kapacitet zauzete memorije =

  dimenzija 1 * dimenzija2 * ... * dimenzija n * veličina tipa
- Nizovi se smeštaju u memoriju tako da se “najdešnja” vrednost najbrže menja

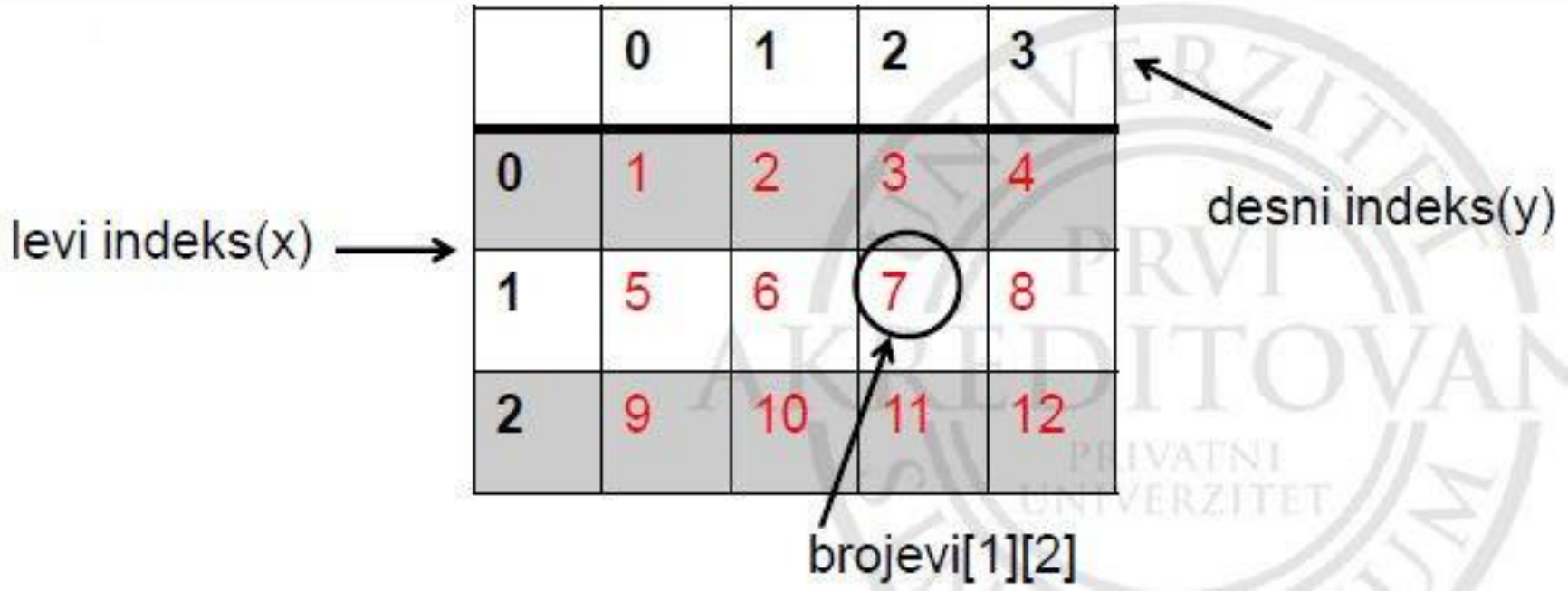
Ilustracija 2D niza

	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]

Primer 2D niza

```
#include <iostream>
using namespace std;
int main(){
  const int BROJ_REDOVA = 3;
  const int BROJ_KOLONA = 4;
  int niz [BROJ_REDOVA][BROJ_KOLONA];
  for (int x=0; x<BROJ_REDOVA; x++) {
     for (int y=0; y<BROJ_KOLONA; y++) {
        niz[x][y] = x * 4 + y + 1;
        cout<< "niz["<<x<<"]["<<y<<"]= "<<niz[x][y]<<endl;
     }
  }
  return 0;
}
```

Primer 2D niza (2)



Inicijalizacija nizova

- C++ omogućava inicijalizaciju nizova

ime_niza[dimenzija] = lista vrednosti;

- Lista vrednosti je spisak vrednosti istog tipa kao tip niza, razdvojenih zarezima

int kvadratni_metri[5] = {30, 50, 70, 80, 100};

Inicijalizacija nizova (2)

- Ne može se navesti više elemenata nego što je dimenzija niza.
- Ako se navede manje elemenata, oni neće biti inicijalizovani i dobijaju vrednost 0
- To nije isto kao kada niz nije inicijalizovan, tada njegovi elementi dobijaju slučajne vrednosti koje su se zatekle u memoriji

Inicijalizacija nizova (3)

- Kada se deklariše inicijalizovan niz, može se izostaviti dimenzija, jer C++ jezik u tom slučaju automatski određuje dimenziju niza.

```
int vrednost [ ] = { 2,3,4 };
```

- Ceo niz se može inicijalizovati na nula pomoću samo jedne vrednosti u listi:

```
int vrednost [10] = {0};
```

Inicijalizacija nizova (4)

- U slučaju inicijalizovanih višedimenzionih nizova, prva dimenzija može biti prazna
- Inače se 2D nizovi inicijalizuju slično kao 1D nizovi

Primer

- Niz koji sadrži brojeve i njihove kvadrate

```
int kvadrati [] [2] = { 2, 4, 3, 9, 4, 16, 5, 25 };
```

Inicijalizacija nizova: Primer

```
#include <iostream>
using namespace std;
int main () {
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   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;
}
```

Stringovi

Stringovi

- Najčešća upotreba 1D nizova je za kreiranje znakovnih nizova (stringova)
- C++ podržava dve vrste znakovnih nizova:
  - niz znakova koji se završava nulom (znakom '\0') (engl. null-terminated string), kao u jeziku C
  - tip string definisan u biblioteci klasa
- C++ kompajler automatski dodaje nulu na kraju znakovnog niza

Stringovi (2)

Primer:

char fizicar [16] = {‘A’, ‘l’ ,’b’,’e’,’r’,’t’,’ ‘,’E’,’i’, ‘n’, ‘s’, ‘t’, ‘e’, ‘i’, ‘n’, ‘\0’};

char fizicar [16] =“Albert Einstein”

char fizicar [] =“Albert Einstein”

Definisanje stringova

- Niz se mora deklarisati tako da mu dimenzija bude za jedan veća od dužine najdužeg stringa koji će sadržati
- C++ automatski dodaje nulu na kraj znakovnih konstanti
- “ “ je prazan string (null string) i sadrži samo nulu
- Generalno, ime znakovnog niza koji sadrži string može se koristiti svuda gde se prihvata string konstanta

Stringovi: Primer

```
#include <iostream>
using namespace std;
int main(){
   char fizicar [16]={'A', 'l' ,'b','e','r','t',' ','E','i', 'n', 's', 't', 'e', 'i', 'n', '\0'};
  cout<<fizicar<<endl;
  char fizicar1 [16] ="Albert Einstein";
  cout<<fizicar1<<endl;
  char fizicar2 [] ="Albert Einstein";
  cout<<fizicar2<<endl;
  return 0;
}
```

```
Albert Einstein
Albert Einstein
Albert Einstein
```

Učitavanje stringa sa tastature

- Najlakši način za unošenje stringa sa tastature jeste korišćenje funkcije cin i znakovnog niza
- Funkcije cin i cout prihvataju i samo ime niza, bez indeksiranja
- Funkcija cin međutim prestaje da učitava string čim naiđe na razmak, tabulator ili nov red

Učitavanje stringa sa tastature (2)

- Problem sa učitavanjem razmaka rešava se korišćenjem funkcije gets(ime_niza) koja je definisana u zaglavlju <cstdio>
- Funkcija gets učitava znakove sa tastature sve dok ne naiđe na CR (engl. Carriage Return – znak za novi red).

Učitavanje stringa sa tastature (4)

```
#include <iostream>
#include<cstdio>
using namespace std;
int main(){
   char str[80];
   cout<<"Unesite string:";
   gets(str);
   cout<<"Uneli ste " <<str<<endl;
   return 0;
}
```

```
Unesite string:Divan dan!
Uneli ste Divan dan!
```

Funkcije string biblioteke

- Definisane u zaglavlju <cstring>

strcpy(s1, s2)

- Kopira string s2 u string s1. Niz s1 mora da bude dovoljno dugačak, jer će se u suprotnom desiti prekoračenje (engl. overrun) i program će pasti.

strcmp(s1, s2)

- Poredi stringove s1 i s2; ako su jednaki, vraća 0, ako je s1 > s2 (po leksikografskom redu) vraća 1, u suprotnom vraća negativan broj

strlen(s)

- Vraća dužinu stringa s

strcpy(s1, s2)

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    char s1[30] = "prvi string";
    char s2[30] = "drugi string";
    strcpy(s1,s2);
    cout<<"s2 ="<<s2<<endl;
    return 0;
}
```

strncmp function: int strncmp

(const char *str1, const char *str2, size_t n)

```
#include <cstring>
#include <iostream>
int main() {
   char s1[20] = "123456789";   char s2[20] = "12345";
   if (strncmp(s1, s2, 8) ==0)   {
        std::cout<<"string 1 i 2 su isti";
    }else {
        std::cout<<("string 1 i string 2 nisu isti");
    }
   return 0;
}
```

strlen function

size_t strlen(const char *str)

```
#include <iostream>
#include <cstring>
int main()
{
    using namespace std;
    char str1[20] = "Kakav divan dan!";
    cout<<"Duzina stringa str1=”
        <<strlen(str1)<<endl;
    return 0;
}
```

Funkcije za rad sa znakovima

- toupper(c): prevodi string u velika slova (uppercase)
- tolower(c): prevodi string u mala slova (lowercase)
- isupper(c): true ako su sva slova u stringu velika
- islower(c): true ako su sva slova u stringu mala
- isalpha(c): true ako je znak slovo (veliko ili malo)
- isdigit(c): true ako je c broj
- isalnum(c): true ako je znak slovo ili broj
- isspace(c): true ako je whitespace (razmak, nov red, carriage return, vertical tab)

Funkcije za rad sa znakovima (2)

```
#include <iostream>
#include <cstring>
#include <cctype>  //funkcija toupper
int main() {
   int i;
   char str[80];
   strcpy (str, "Funkcija za rad sa znakovima");
   for (int i=0; str[i];i++)
      str[i]=toupper (str[i]);
   std::cout<<str<<std::endl;
   std::cin.get();
   return 0;
}
```

FUNKCIJA ZA RAD SA ZNAKOVIMA

Nizovi stringova

- Niz stringova je specijalan oblik 2D niza

```
//inicijalizacija niza od 7 stringova sa po 11 znakova
char daniUNedelji [7][11] = { “Ponedeljak”, “Utorak”, “Sreda“, “ Četvrtak“, “Petak”, “Subota”, “ Nedelja” };
```

- Pojedinačnim stringovima se pristupa navođenjem levog indeksa

```
daniUNedelji [3]  //četvrtak
```

- Pojedinačnim znakovima pristupa se navođenjem oba indeksa

```
daniUNedelji [3][2] //slovo t u stringu čet vrtak
```

Nizovi stringova (2)

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char daniUNedelji [7][11] = { "Ponedeljak","Utorak", "Sreda",
                                 "Cetvrtak", "Petak", "Subota"," Nedelja"};
   cout<< "daniUNedelji [3]= "<<daniUNedelji [3]<<endl;
   cout<< "daniUNedelji [3][2]= "<<daniUNedelji [3][2]<<endl;
   return 0;
}
```

```
daniUNedelji [3]= Cetvrtak
daniUNedelji [3][2]= t
```

Pokazivači

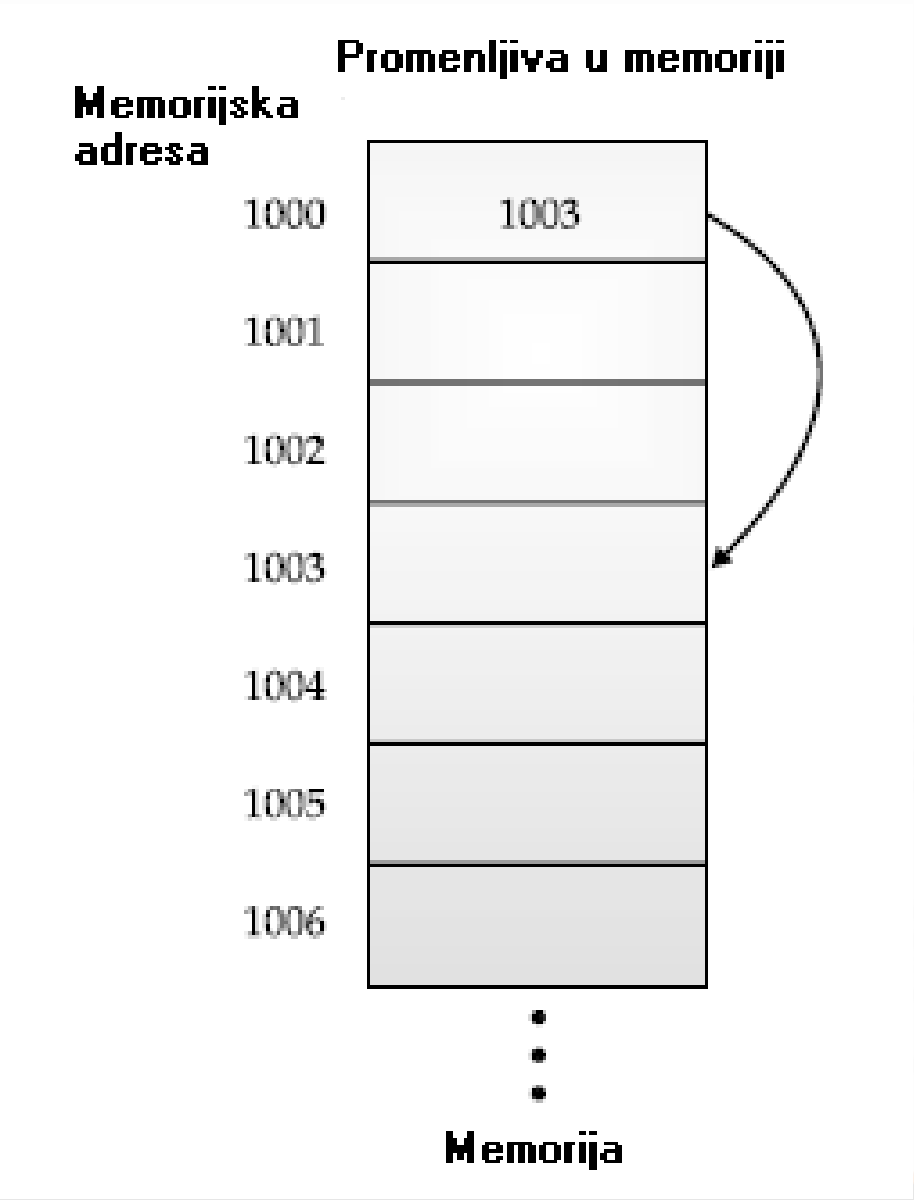
Pokazivači

- Pokazivači (engl. pointer) su jedna od najsnažnijih mogućnosti jezika C/C++, ali njihovo savladavanje je obično problematično
- Pokazivač je promenljiva koja sadrži neku adresu u memoriji
- Pošto memorijska adresa nije običan ceo broj, ni pokazivač ne može biti predstavljen kao obična int promenljiva

Pokazivači (2)

- Svaka promenljiva ima adresu, a to je broj koji određuje njen položaj u radnoj memoriji računara
- Na nekoj adresi u memoriji nalazi se promenljiva:
  - ako promenljiva x (pokazivač) sadrži adresu neke druge promenljive y, kaže se da “x “ pokazuje na “ y”

Pokazivači (3)



Pokazivači (4)

- Promenljiva koja je pokazivač mora se na odgovarajući način definisati:

tip *ime_promenljive

- Generalno, ako se ispred imena promenljive stavi *, to je oznaka da je ta promenljiva zapravo pokazivač; postoji dogovor da nazivi pokazivačkih promenljivih u jeziku C++ počinju slovom *p*
- Tip određuje na koji će tip podataka pokazivač pokazivati

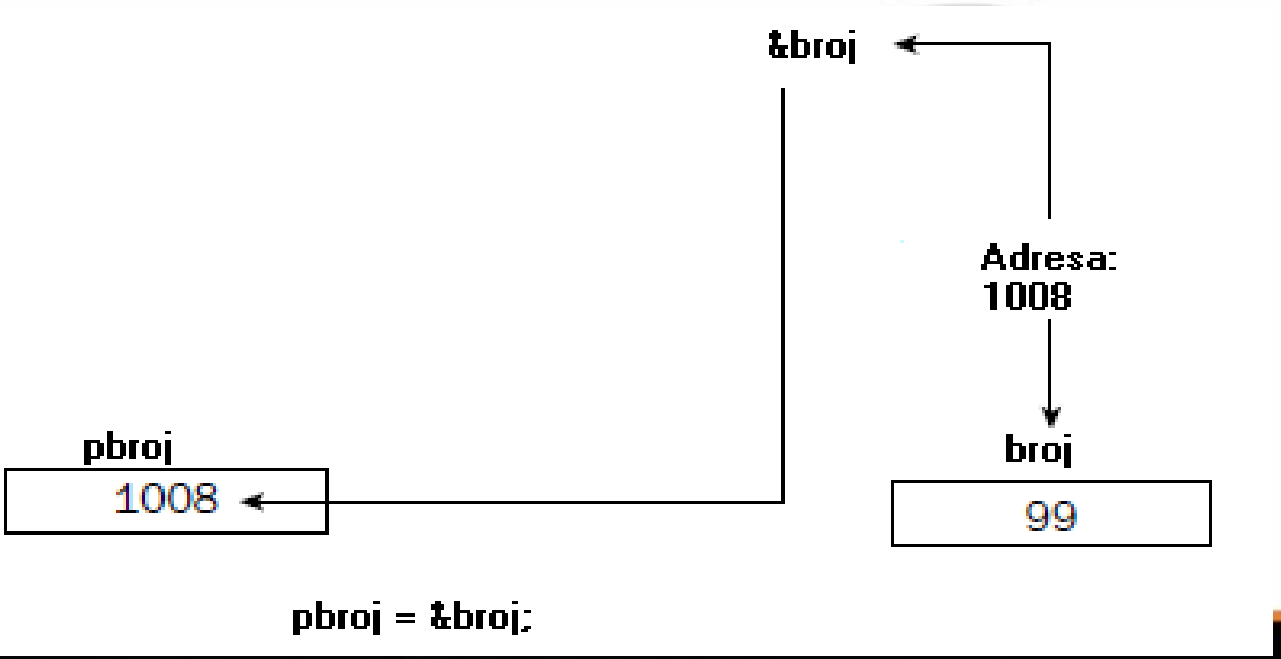
```
int *p;  //pokazivač na int
float *fp; //pokazivač na promenljivu float
```

Operacije sa pokazivačima

- U radu sa pokazivačima koriste se dva unarna operatora: * (operator indirekcije ili operator derefenciranja) i & (operator adresiranja)

•& vraća memorijsku adresu svog operanda

```
int * pbroj;
int broj;
pbroj = &broj;
```



Operator indirekcije (*)

- Operator * vraća vrednost promenljive koja se nalazi na adresi na koju pokazuje njen operand
- Zove se i operator dereferenciranja, jer “vraća” vrednost promenljive

```
int *pbroj;
int broj;
//vrednost promenljive je sadržaj adrese na koju
//pokazuje pokazivač
```

Zašto su uopšte potrebni pokazivači?

- Rad sa nizovima je mnogo brži ako koristimo pokazivače umesto indeksiranja
- Pokazivači se koriste kada je u funkciju potrebno preneti veliku količinu podataka (npr. niz)
- I najvažnije, pokazivači omogućavaju dinamičku alokaciju memorije (tj. rezervisanje tačno onoliko memorije koliko je programu zaista potrebno tokom izvršavanja), nasuprot statičkom rezervisanju koje nije efikasno
- **Zaključak**: Potrebni su zbog efikasnosti koda

Tip pokazivača

- Tip pokazivača određuje tip podataka sa kojima pokazivač radi

```
int *pokazivač;
float promenljiva;
pokazivač = &promenljiva; // greška
```

Error: a value of **type float *** cannot be assigned to an entity of **type int***

- Moguće je primjeniti konverziju tipa na pokazivače, ali to obično nije dobra ideja

Konverzija

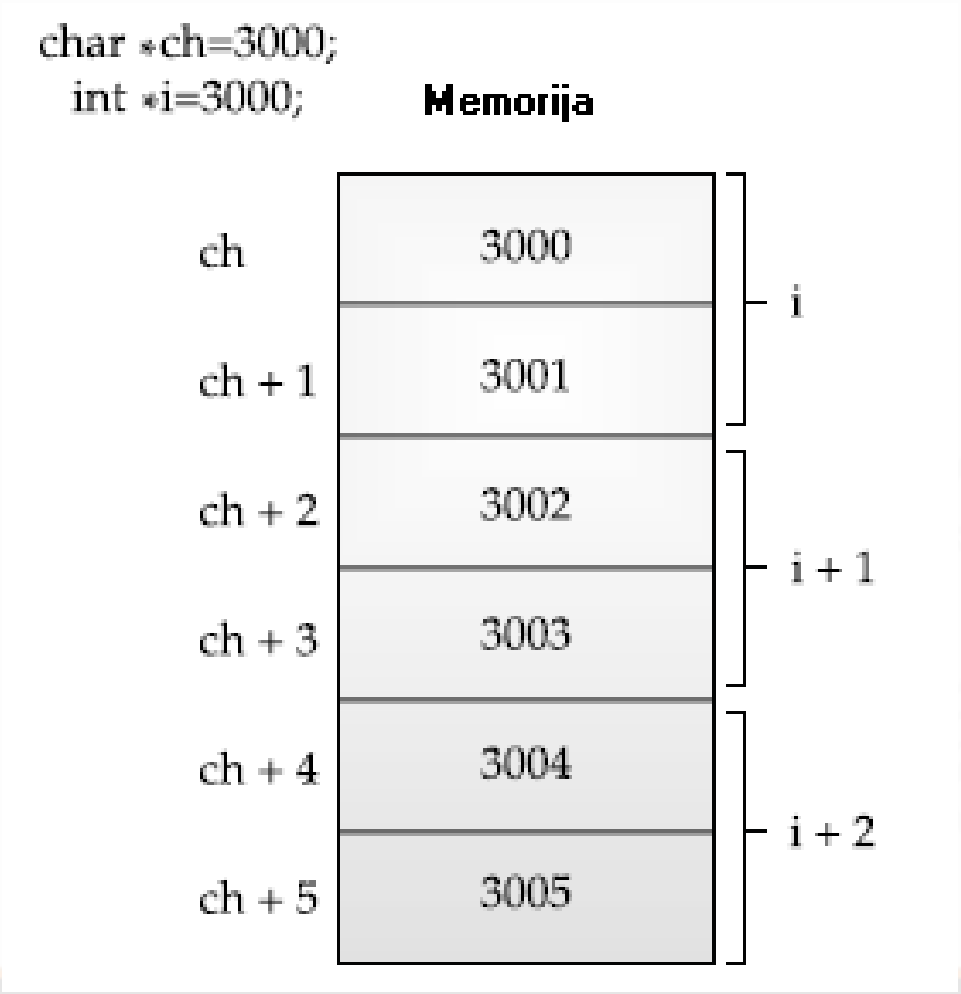
```
#include <stdio.h>
int main(){
   int *integer_ptr;
   float *float_ptr;
   int my_int = 17;
   float my_float = 23.5;
   integer_ptr = &my_int;
   float_ptr = &my_float;
  *integer_ptr = *float_ptr;
   cout<< “integer_ptr”<< *integer_ptr;
  return 0;
}
```

Pokazivačka aritmetika

- Na pokazivačke promenljive se mogu primenjivati samo operatori ++, --, + i –
- Pokazivači se mogu porediti pomoću operatora >, < i ==
- Svaki pokazivač se može porediti sa vrednošću null, odnosno 0

Pokazivačka aritmetika

- Uvek je zavisna od tipa pokazivača



Primer pokazivača

```
 #include <iostream>
using namespace std;
int main () {
  int*p, broj=44;
  p=&broj;
  cout<<"broj = "<<broj<<endl;
  (*p)++;
  cout<<"broj = "<<broj<<endl;
  cout<<"p ="<<p<<endl;
  cout<<"*p ="<<*p<<endl;
  return 0;
}
```

```
broj = 44
broj = 45
p =0036F9BC
*p =45
```

Inicijalizacija pokazivača

- Korišćenje pokazivača koji nisu inicijalizovani je jako loša praksa
  - dešava se nasumično prepisivanje sadržaja memorije
- Pokazivač se inicijalizuje adresom promenljive odgovarajućeg tipa koja je već inicijalizovana, ili na vrednost 0 (NULL) ako ne pokazuje ni na šta

Inicijalizacija pokazivača (2)

```
#include <iostream>
using namespace std; int
main () {
  int*p, *pbroj, *pbroj1;
  pbroj = NULL;
  p=NULL;
  pbroj = 0; //isto što i NULL
  if (!pbroj)
     cout<<"pbroj broj ne pokazuje ni na sta"<<endl;
  pbroj1=p;
  cout<<"pbroj1 =" <<pbroj1<<endl;
  return 0;
}
```

```
pbroj broj ne pokazuje ni na sta
pbroj1 =00000000
```

Pokazivači i nizovi

- Ime niza je pokazivač na prvi element niza
- Ime niza je konstantan pokazivač i nijedna naredba ne može da mu promeni vrednost (npr. ne može se pomeriti inkrementiranjem)

Pokazivači i nizovi (2)

```
#include <iostream>
using namespace std;
int main () {
  int broj[]= {1, 3, 5};
  int *pbroj;
  pbroj = broj;
  //pbroj = broj[2]; //greška
  //broj = pbroj; //greška
  ++pbroj;   //pbroj sada pokazuje na broj[1]
  cout<<"pbroj  ="<<pbroj<<endl;
  cout<<"*pbroj  ="<<*pbroj<<endl;
  cout<<"(*(++pbroj))  ="<<*(++pbroj)<<endl;
  //++broj ; //greška
  return 0;
}
```

Pokazivači i stringovi

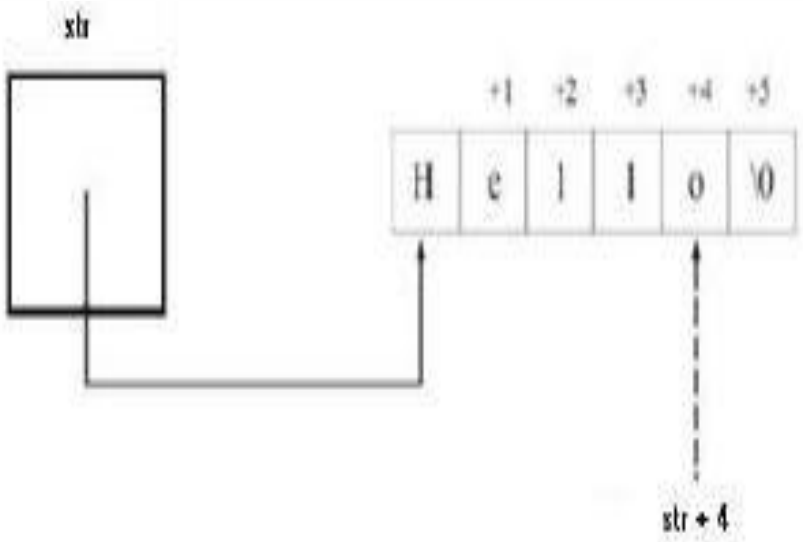
- Pokazivač na char ima zanimljivu osobinu da se može inicijalizovati string literalom (konstantom)
- Kada prevodilac naiđe na string literal, smešta ga u tabelu stringova programa i generiše pokazivač na taj string
- Prevodilac tretira string konstantu kao pokazivač na prvi znak stringa

Pokazivači i stringovi

- Pokazivač na char ima zanimljivu osobinu da se može inicijalizovati string literalom (konstantom)

```
char *str = "Hello";
```

str[2] je isto sto i *(str + 2)



Nizovi pokazivača

- Pokazivači se mogu nizati kao i svaki drugi tip podataka

```
int *px[10]; //niz od deset pokazivaca na int
int var = 5;
px[2]= &var; //inicijalizacija treceg elementa niza pokazivaca
var = *(px[2]);
```

- Uobičajena namena nizova pokazivača jeste da sadrže pokazivače na stringove

Primer #1

```
#include <iostream>
using namespace std;
int main () {
  int broj[]= {1, 3, 5};
  int *px[10]; //niz od deset pokazivača na int
  int *py= & (*px[10]);
  int var = 5;
  cout <<"&var ="<<&var<<endl;
  cout <<"*&var ="<<*&var<<endl;
  cout <<"px[2] = "<<px[2] <<endl;
  px[2] = &var; //inicijalizacija elementa niza broj 3
  var = *(px[2]);
  return 0;
}
```

```
&var =00D1F800
*&var =5
px[2] = CCCCCCCC
```

Nizovi i pokazivači

- Ime niza je i početna adresa niza

Primer

```
int vals[] = {4, 7, 11};
```

- Neka je početna adresa niza vals 0x4a00

```
cout << vals;          // 0x4a00
cout << vals[0];       // 4
```

4	7	11

Nizovi i pokazivači (2)

- Ime niza se može koristiti kao konstanta pokazivača

```
int vals[] = {4, 7, 11};
cout << *vals;     // 4
```

- Pokazivač se može koristiti kao ime niza:

```
int *valptr = vals;
cout << valptr[1]; //  7
```

Nizovi i pokazivači (3)

•Zadato:

```
int vals[]={4,7,11};
int *valptr = vals;
```

- Dati rezultat za valptr + 1?

•To je: adresa za valptr + 1 * size of an int

```
cout << *(valptr+1); // 7
cout << *(valptr+2); // 11
```

- Moraju se koristiti izrazi.

Pristup nizu

- Nizu elemenata se može pristupiti na razne načine.

```
int vals[]={4,7,17};
int *valptr = vals
```

Način pristupa	Primjer
Ime niza i [ ]	vals[2] = 17;
Pokazivač na niz i [ ]	valptr[2] = 17;
Ime niza/ i aritmetika indeksa	*(vals+2) = 17;
Pokazivač na niz i aritmetika indeksa	*(valptr+2) = 17; (vals + 2)= adresa elementa br.2

Aritmetika pokazivača

- Zadato

```
int vals[]={4,7,11};
int *valptr = vals;
```

- Primeri korišćenja ++ i --

```
valptr++; // 7
valptr--; // 4
```

Aritmetika pokazivača (2)

- Zadato:

  int vals[]={4,7,11};

  int *valptr = vals;

- Primer korišćenja +=:

  valptr = vals; // points at 4

  valptr += 2; // points at 11

Poređenje pokazivača

- Relacioni operatori se mogu koristiti za poređenje adresa pokazivača

```
if (ptr1 == ptr2)   // poređenje adresa
```

- Poređenje adresa pokazivača nije isto kao poređenje sadržaja na koje pokazuju pokazivači

```
if (*ptr1 == *ptr2) //poređenje sadržaja
```

Niz pokazivača

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

```
Vrednost var[0] = 10
Vrednost var[1] = 100
Vrednost var[2] = 200
```

Dinamička alokacija memorije

Situacija

- Program tokom izvršavanja traži od OS da odvoji deo neiskorišćene memorije za smeštaj promenljive određenog tipa.
- OS vraća početnu adresu u memoriji gde će promenljiva biti smeštena
- Program može pristupiti varijabli samo preko adrese, a zato je potreban pokazivač.

Dinamička alokacija memorije (2)

- Dinamički alocirana memorija se uzima iz dela memorije koji se zove hrpa (engl. heap)
- Hrpa se razlikuje od drugih područja memorije koje koristi C++ program.
- Ako se promenljiva definiše kao statička ili globalna, prostor za promenljivu se alocira iz područja globalnih podataka pre nego što počne izvršavanje programa.

Dinamička alokacija memorije (3)

Dinamička alokacija memorije (4)

- Kada funkcija main () završi, OS dealocira prostor za statičke i globalne varijable.
- Lokalna promenljiva (promenljiva deklarisana unutar bloka) se kreira u oblasti steka kada program ulazi u blok, a uništava kada program izlazi iz bloka.
- Dinamička alokacija memorije zahteva operator koji se zove new.
- Operator alocate oslobađa memoriju koja je predhodno bila alocirana pomoću new operatora.

Dinamička alokacija memorije (5)

- **Opšta sintaksa**

pokazivač = new tip;

delete pokazivač;

- Pokazivač je promenljiva koja prihvata adresu alocirane memorije
- **Primjer**

int *p =new int;

int *p =new int[100];

Dinamička alokacija memorije (6)

- Ako nema dovoljno prostora za alokaciju memorije:

```
int *p =new int[100];
if (!p) //ili if (p==NULL)
   cout<<“greška u alociranju memorije!“);
```

- Kada program završi sa korišćenjem dinamičke memorije, treba je osloboditi korišćenjem operatora delete.

```
delete p;    //ako p pokazuje na promenljivu
delete [] p; //ako p pokazuje na niz
```

Zaključak

- Niz je skup promenljivih istog tipa kojima se pristupa preko zajedničkog imena: **tip ime[velicina]**
- Jednodimenzioni i višedimenzioni nizovi
- Inicijalizacija nizova: ne može se navesti više elemenata nego što je dimenzija niza
- Stringovi – znakovni nizovi koji se završavaju nulom (\0)
- Funkcije string biblioteke: **strcpy(s1, s2), strcmp(s1, s2), strlen(s)**

Zaključak (2)

- Pokazivač je promenljiva koja sadrži neku adresu u memoriji: **tip *ime_promenljive**
- Operacije sa pokazivačima: * (operator indirekcije) i & (operator adresiranja)
- Inicijalizacija pokazivača: adresom promenljive ili na vrednost 0 (NULL) ako ne pokazuje ni na šta
- Dinamička alokacija memorije znači da program tokom izvršavanja traži od računara da odvoji deo neiskorišćene memorije da bi u njega smestio promenljive određenog tipa

Kraj prezentacije

HVALA NA PAŽNJI!