Osnovi programiranja – C

Nizovi i matrice. Pokazivači

Teme

- Jednodimenzioni nizovi
- Višedimenzioni nizovi
- Pokazivači
- Nizovi i pokazivači kao argumenti funkcija

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)

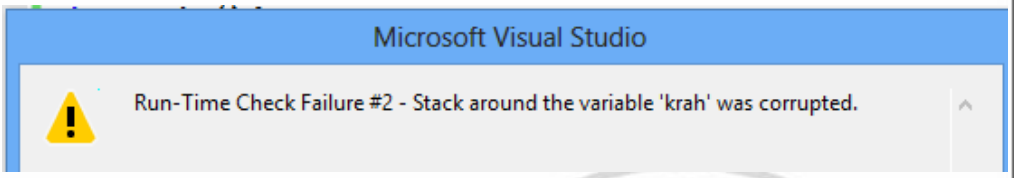
```
#include <stdio.h>

int main(){
    int mojNiz[5], i;
    for (i=0; i<5; i++) {
        printf("Vrednosti niza mojNiz[%d]: ", i);
        scanf("%d", &mojNiz[i]);
    }
    for (i = 0; i<5; i++)
        printf("%d: %d\n", i, mojNiz[i]);
    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];
    printf("b[%d]=%d\n", i, b[i]);
}
```

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

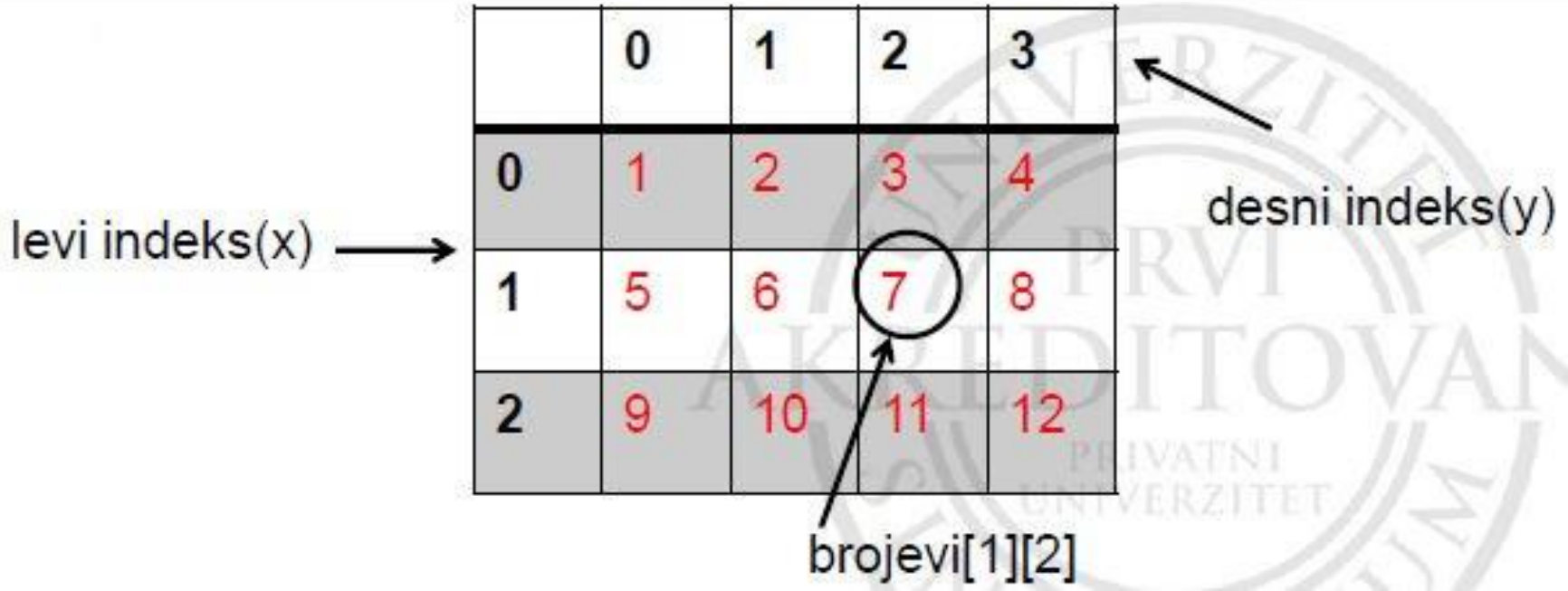
Ilustracija 2D niza

	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 <stdio.h>
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;
        printf("niz[%d][%d]=%d\n", x, y, niz[x][y]);
     }
  }
  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 <stdio.h>
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++ )  {
         printf("a[%d][%d]: %d", i, j, a[i][j]);
         printf("       ");
      }
      printf("\n");
   }
   return 0;
}
```

Pokazivači

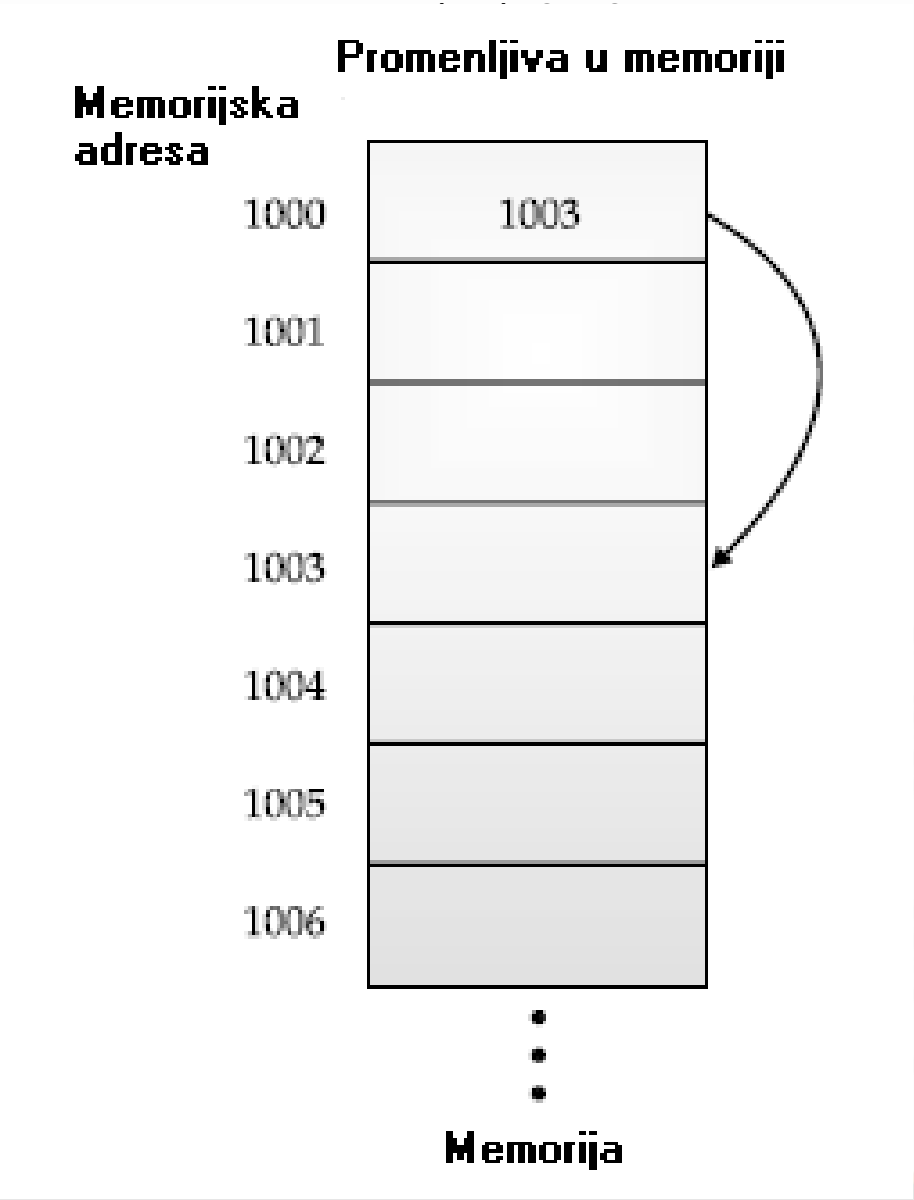
Pokazivači

- Pokazivači (engl. pointer) su jedna od najsnažnijih mogućnosti jezika 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

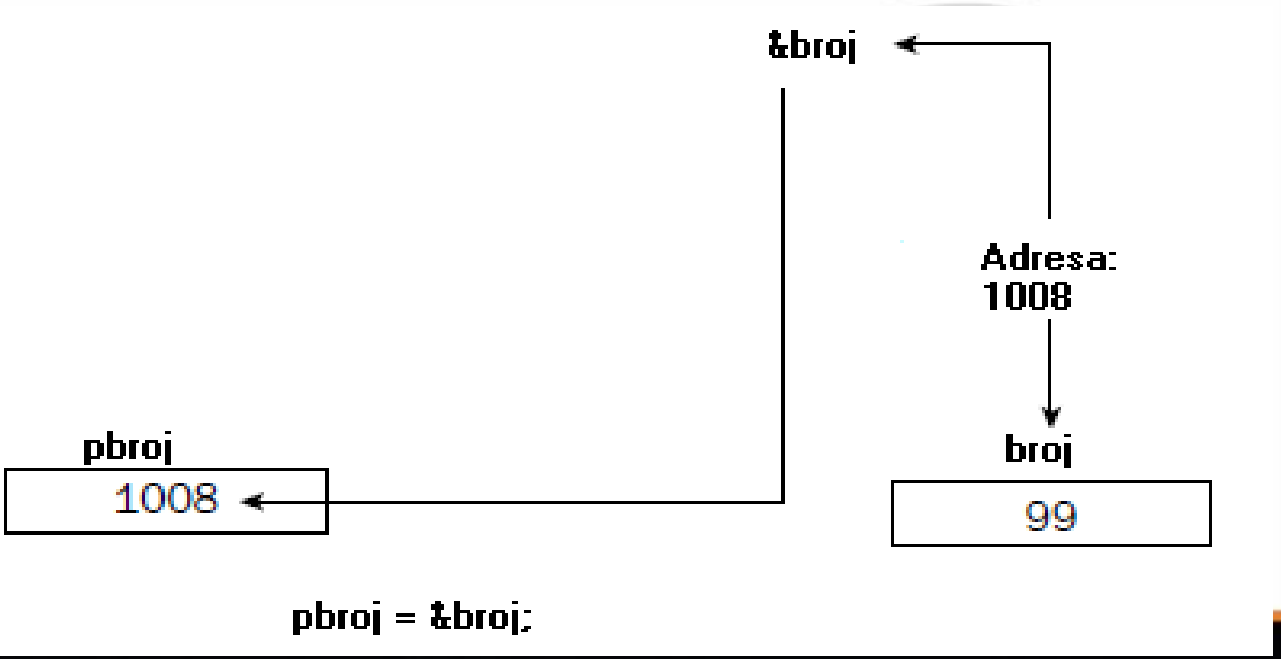
- Generalno, ako se ispred imena promenljive stavi *, to je oznaka da je ta promenljiva zapravo pokazivač; postoji dogovor da nazivi pokazivačkih promenljivih u jeziku C++ počinju slovom *p*
- Tip određuje na koji će tip podataka pokazivač pokazivati

```
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 primeniti konverziju tipa na pokazivače, ali to obično nije dobra ideja

Konverzija

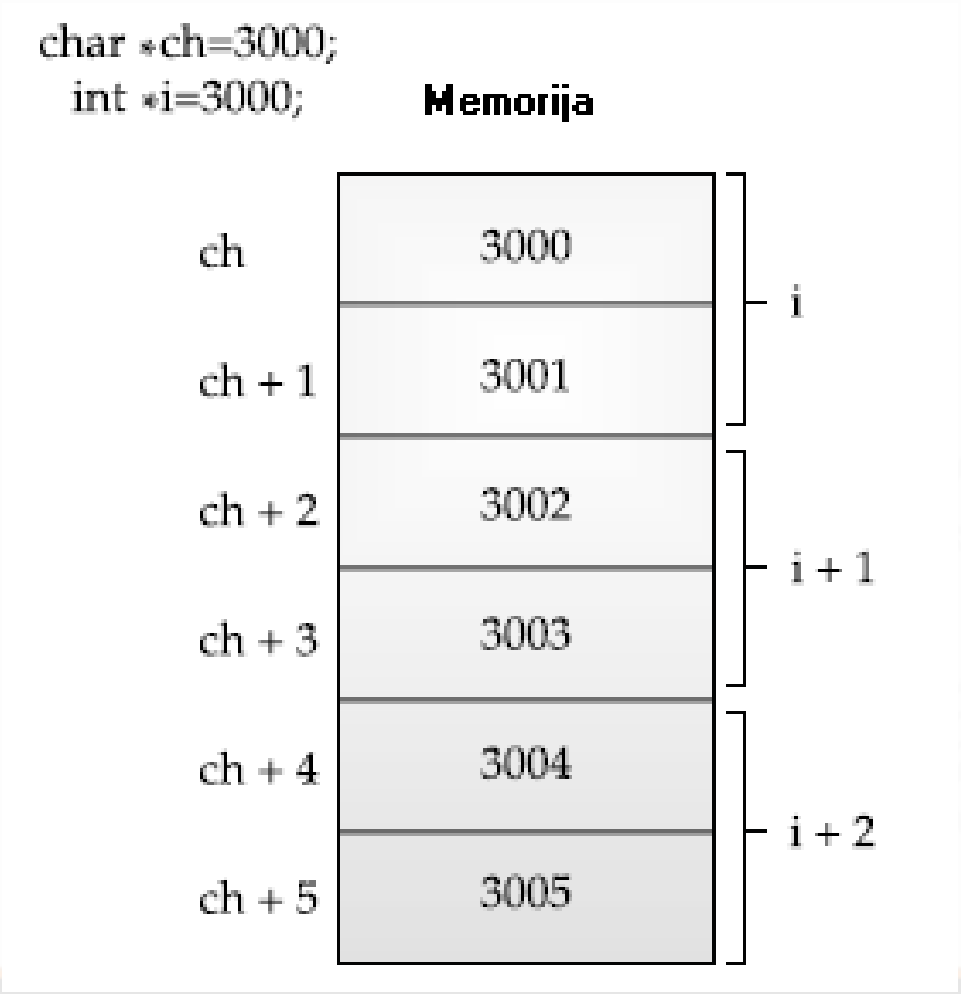
```
#include <stdio.h>
int main(){
   int *integer_ptr;
   float *float_ptr;
   int my_int = 17;
   float my_float = 23.5;
   integer_ptr = &my_int;
   float_ptr = &my_float;
  *integer_ptr = *float_ptr;
   printf("integer_ptr: %d", *integer_ptr);
  return 0;
}
```

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 <stdio.h>
int main () {
    int*p, broj=44;
    p=&broj;
    printf("broj = %d\n", broj);
    (*p)++;
    printf("broj = %d\n", broj);
    printf("p= %p\n", p);
    printf("*p= %d\n", *p);
    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 <stdio.h>
int main () {
    int*p, *pbroj, *pbroj1;
    pbroj = NULL; p=NULL;
    pbroj = 0; //isto što i NULL
    if (!pbroj)
        printf("pbroj broj ne pokazuje ni na sta\n");
    pbroj1=p;
    printf("pbroj1 = %p\n", pbroj1);
    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 <stdio.h>
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]
    printf("pbroj= %p\n", pbroj);
    printf("*pbroj= %d\n", *pbroj);
    printf("(*(++pbroj))= %d\n", *(++pbroj));
    //++broj ; //greška
    return 0;
}
```

```
pbroj= 0061FE10
*pbroj= 3
(*(++pbroj))= 5
```

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 <stdio.h>
int main () {
    int broj[]= {1, 3, 5};
    int *px[10]; //niz od deset pokazivača na int
    int *py= & (*px[10]);
    int var = 5;
    printf("&var = %p\n", &var);
    printf("*&var = %d\n", *&var);
    printf("px[2] = %p\n", px[2]);
    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

```
printf("%p", vals);        // 0x4a00
printf("%d", vals[0]);     // 4
```

4	7	11

Nizovi i pokazivači (2)

- Ime niza se može koristiti kao konstanta pokazivača

```
int vals[] = {4, 7, 11};
printf("%d", *vals);   // 4
```

- Pokazivač se može koristiti kao ime niza:

```
int *valptr = vals;
printf("%d", 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

```
printf("%d", *(valptr+1)); // 7
printf("%d", *(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 <stdio.h>
#define 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++) {
        printf("Vrednost var[%d] = %d\n", i, *ptr[i]);
   }
   return 0;
}
```

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

Kako C prosleđuje argumente funkciji

- Prenos po vrednosti (engl. pass-by-value)
  - vrednost argumenta kopira se u parametar funkcije
  - promene parametra nemaju nikakvog efekta na argument koji se koristi za poziv funkcije
  - poziv po vrednosti je default način prenosa parametara
- Prenos po adresi (engl. pass-by-reference)
  - u parametar se kopira adresa, a ne vrednost argumenta
  - promene parametra menjaju i argument
  - **postiže se tako što je pokazivač argument funkcije**

Prosleđivanje pokazivača funkciji

- Kada parametri funkcije nisu pokazivači, prilikom poziva funkcije pravi se kopija argumenata sa kojima se radi u funkciji; to se zove prosleđivanje po vrednosti (pass-by-value)
  - funkcija ne može nikako da promeni vrednosti argumenata koji joj se prosledjuju po vrednosti
- Kada je parametar funkcije pokazivač, funkcija može da promeni vrednost promenljive na koju pokazuje pokazivač; to je prosleđivanje po adresi (pass-by-address)

#include <stdio.h>

Pokazivači kao argument funkcije

```
void funkcija (int *p) {
    *p = 100;
}

int main () {
  int i=0;
  int *p;
  p = &i;
  funkcija (p);
  printf("i = %d\n", i);
return 0;
}
```

i = 100

Prenos argumenata funkciji po referenci pomoću pokazivača

```
#include <stdio.h>
void calc (int *p);

int main() {
   int x = 10;
   calc(&x);
   printf("x = %d\n", x);
   return 0;
}

void calc (int *p) {
   *p = *p + 10;
}
```

x = 20

Prenos argumenata funkciji po referenci pomoću pokazivača (2)

```
#include <stdio.h>
void square(int *);

int main(){
    int n = 8;
    printf("&n= %p\n", &n);
    printf("%d\n", n);
    square(&n);
    printf("%d\n", n);
    return 0;
}

void square(int * pN){
    printf("pN= %p\n", pN);
    *pN *= *pN;
}
```

```
&n= 001CFEC4
8
pN = 001CFEC4
64
```

Vraćanje niza iz funkcije

```
#include <stdio.h>
int * f1(int arr[], int length){
for (int i = 0;i < length;++i){
      arr[i] = arr[i] * 4;
   }
   return arr;
}

int main(int argc, char* argv[]){
   int arr[] = { 1,2,3,4,5 };
   int *arr2;
   arr2 = f1(arr, 5);
   printf("*arr2 = %d\n", *arr2);
   printf("*(arr2+1) = %d\n", *(arr2+1));
   return 0;
}
```

*arr2 =4
*(arr2+1) =8

Vraćanje niza iz funkcije pomoću pokazivača

```
#include <stdio.h>
int * f1 (int a1[], int a2[]){
      for (int i = 0;i < 5;++i){
           a1[i] = a1[i] + a2[i];
       }
      return a1;
}
int main(){
   int arr1[] = { 1,2,3,4,5 };
   int arr2[] = { 6,7,8,9,10 };
   int *arr3;
   arr3 = f1(arr1,arr2);
   for (int i = 0;i < 5;++i)
        printf("*arr3[%d] = %d\n", i, *(arr3+i));
   return 0;
}
```

```
*arr3[0] =7
*arr3[1] =9
*arr3[2] =11
*arr3[3] =13
*arr3[4] =15
```

Prenos argumenata po referenci (nizovi)

```
#include <stdio.h>
void test (int* a, int* b, int* c, int len) {
  for (int i = 0; i < len; ++i)
       c[i] = a[i] + b[i];
}
int main() {
   int a[5] = {1,2,3,4,5}, b[5] = {6,7,8,9,10}, c[5] = {};
   test(a, b, c, 5);
   for (int i = 0; i < 5; ++i)
        printf("c[%d] = %d\n", i, c[i]);
   return 0;
}
```

```
c[0] = 7
c[1] = 9
c[2] = 11
c[3] = 13
c[4] = 15
```

Prenos argumenata po referenci (nizovi)

```
#include <stdio.h>
int * test (int* a,int* b,int*c,int len){
      for (int i = 0; i < len; ++i)
            c[i] = a[i] + b[i];
       return c;
}

int main() {
      int a[5] = {1,2,3,4,5}, b[5] = {6,7,8,9,10}, c[5] = {};
      int * p= test(a, b, c, 5);
     for (int i = 0; i < 5; ++i)
           printf("p[%d] = %d\n", i, p[i]);
      return 0;
}
```

Niz kao argument funkcije

- Kada je argument funkcije niz, prosleđuje se adresa prvog elementa niza, a ne ceo niz
- Pošto je ime niza pokazivač na prvi element niza, to znači da se zapravo prosleđuje pokazivač na niz, pa funkcija može da promeni sadržaj niza koji se koristi za pozivanje funkcije

Prosleđivanje nizova funkciji

```
#include <stdio.h>
void prikaziNiz (int brojevi[10]);

int main () {
   int t[10],i;
   for (int i=0;i<10;i++)
         t[i] =i;
   prikaziNiz (t);
   return 0;
}

void prikaziNiz (int brojevi[10]) {
     for (int i=0;i<10;i++)
     printf("%d", brojevi[i]);
     printf("\n");
}
```

```
0 1 2 3 4 5 6 7 8 9
```

Zaključak (1)

- 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

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

Kraj prezentacije

HVALA NA PAŽNJI!