

P2: Sedma nedelja

Pokazivači i dinamička alokacija memorije

Zadatak 1 – pokazivači (C55a)

- Šta ispisuje sledeći program?

```
#include <stdio.h>
void main() {
    int a[]={1,0}, *p;
    p=a;
    printf("%d, ", *p++);
    printf("%d\n", *p);
}
```

Odgovori:

- A) 1, 2
- B) 1, 0
- C) 0, 0

Pokazivači

- **Pokazivač je promenljiva koja sadrži adresu neke lokacije u operativnoj memoriji**

- svi pokazivači imaju istu veličinu, u zavisnosti od veličine adresnog prostora na odgovarajućoj mašini

- **Definisanje pokazivača**

```
tip *ime;
```

- tip predstavlja tip podatka na koji se pokazuje (int, double)
- tip je potreban da bismo znali o kakvom se podatku radi prilikom pristupa podatku preko pokazivača

- **Adresa nekog podatka se pokazivaču dodeljuje pomoću adrsnog operatora &**

```
pa = &a;
```

- **Lokaciji na koju pokazivač ukazuje se pristupa pomoću operatora za indirektan pristup (dereferenciranje) ***

```
*pa = 1;
```

Adresna aritmetika

- **Dodela vrednosti**

```
pa = pb;
```

- **Sabiranje i oduzimanje celobrojnog podatka**

- jedinica mere pri sabiranju i oduzimanju pokazivača sa celobrojnim podatkom je veličina pokazivanog podatka!

```
pa = pa + x;
```

- nova adresa `pa` će biti formirana kao zbir stare adrese `pa` i `x*sizeof(pokazani_tip)`

- **Oduzimanje i upoređivanje dva pokazivača**

- binarni operator za oduzimanje i svi relacioni operatori
- ima smisla samo za pokazivače koji pokazuju na elemente istog niza

- **Upoređivanje pokazivača sa nulom**

- konstanta 0 ili simbolička konstanta `NULL` iz `<stdlib.h>`

```
#define NULL (0L)
```

- dodela vrednosti 0 ili `NULL` nekom pokazivaču označava da pokazivač ne pokazuje ni na koji podatak

Pokazivači i nizovi

- Ako je definisan neki statički niz:

```
int a[] = {1,2,3,4};
```

- Ime niza **a** predstavlja konstantni pokazivač na početak niza!

```
&a[0] Ó a  
&a[i] Ó a+i  
a[i] Ó *(a+i)
```

a:	1	2	3	4
	a[0]	a[1]	a[2]	a[3]
	a+0	a+1	a+2	a+3

- Zbog ove osobine, pokazivačke promenljive se mogu tretirati kao nizovi
 - moгуće je koristiti operator za indeksiranje [], kao kod nizova

Zadatak 1 - rešenje

- Posmatramo deklaraciju:

```
int a[]={1,0}
```

- a je niz celih brojeva, indeksira se od 0
- dužina niza nije eksplicitno zadata, ali se može doznati iz broja inicijalizatora

a:	1	0
----	---	---

- Dodela vrednosti

```
p=a;
```

a:	1	0
----	---	---

p



Zadatak 1

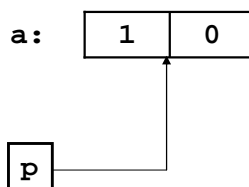
```
printf("%d, ", *p++);
```

- Prioritet operatora ***** i **++** je jednak, ali je asocijativnost ove grupe operatora sdesna ulevo
- Prema tome, prvo će se izvršiti operator **++**, čime se uvećava vrednost pokazivača i on pokazuje na naredni celobrojni podatak u memoriji
- Zbog osobine operatora **++**, operator ***** će biti primenjen na staru adresu pokazivača **p** i biće ispisana vrednost 1

```
printf("%d\n", *p);
```

- Ispis: 1, 0
- Odgovor: B

a:	1	0
----	---	---



Zadatak 2 – pokazivači (C55b)

- Šta ispisuje sledeći program?

```
#include <stdio.h>
void main() {
    int a[]={0, 1, 2, 3, 4};
    int i, *p;

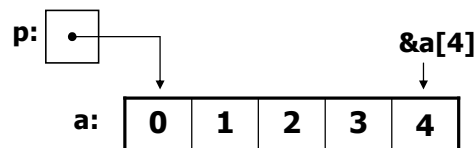
    for (p=a, i=0; p<=&a[4]; p++)
        printf("%d", *p);
    printf("\n");

    for (p=&a[0], i=0; p+i<a+4; p++, i++)
        printf("%d", *(p+i));
    printf("\n");

    for (p=a+4, i=0; i<=4; i++)
        printf("%d", p[-i]);
    printf("\n");
}
```

Zadatak 2 – rešenje

```
for (p=a; p<=&a[4]; p++)  
    printf("%d", *p);
```



Ispis: 0 1 2 3 4

- Operator `&` dohvata adresu operanda u memoriji

- Operatori indirektnog adresiranja (`*`) i dohvatanja adrese imaju suprotno dejstvo

$\&a[i] \leftrightarrow a+i$
 $a[i] \leftrightarrow *(a+i)$

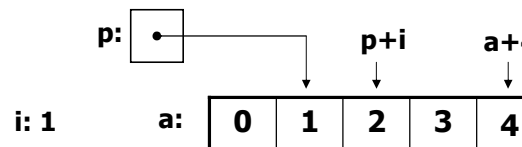
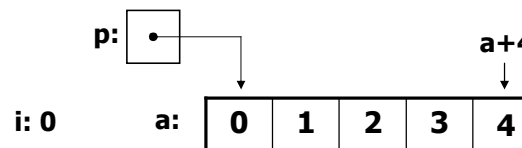
za `i==0`:

$\&a[0] \leftrightarrow a$
 $a[0] \leftrightarrow *a$

Zadatak 2

```
for (p=&a[0], i=0; p+i<a+4; p++, i++)  
    printf("%d", *(p+i));
```

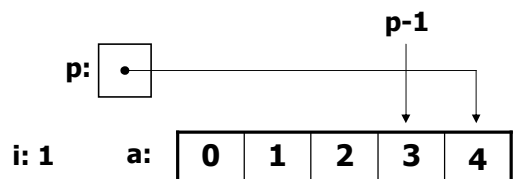
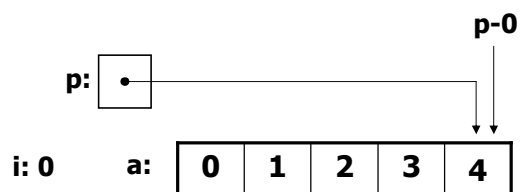
Podsetnik:
 $*(p+i) = p[i]$



Ispis: 0 2

Zadatak 2

```
for (p=a+4, i=0; i<=4; i++)  
    printf("%d", p[-i]);
```



Ispis: 4 3 2 1 0

Oprez!

- C ne proverava opseg indeksa

- Programer je dužan da obezbedi ispravne vrednosti

- Indeks predstavlja pomeraj u odnosu na prvi element niza i može biti negativan

Zadatak 3

- Ako je u datom računaru celobrojni podatak predstavljen na dužini od 4 bajta, i ako je 1000 vrednost pokazivačke promenljive `p`, deklarisan na sledeći način:

```
int *p;
```

- Koja će biti vrednost promenljive `p`, nakon izvršenja operacije `p++`?

- Odgovori:

A) 1001

B) 1002

C) 1004

Zadatak 3 – rešenje

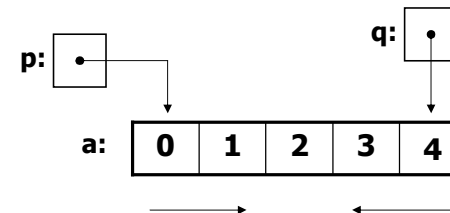
- Programski jezik C automatski uzima u obzir veličinu pokazanog podatka prilikom primene operatora ++, --, +, - nad pokazivačima
- Veličina nekog podatka u memoriji, izražena u bajtovima, programskim putem može da se odredi primenom operatora `sizeof(tip)` ili `sizeof(promenljiva)`

`p++ ↔ p = p + sizeof(*p)`

- Rezultat je : $1000 + 4 = 1004$
- Odgovor: C

Zadatak 4 – obrtanje niza (4.2)

- Napisati program na programskom jeziku C koji sa standardnog ulaza učitava niz celih brojeva, obrće redosled brojeva u nizu i ispisuje niz redom od početka.
- Algoritam:
 - krenuti sa dva pokazivača, od početka i sa kraja
 - zamenjivati mesta pokazanim elementima dok god se pokazivači ne sretnu



Zadatak 4 - rešenje

- Rešenje – ciklus sa izlaskom na sredini

```
#include <stdio.h>
#define NMAX 100
void main () {
    int a[NMAX], *p, *q, n;
    while (1) {
        printf ("n? "); scanf ("%d", &n);
        if (n<=0 || n>NMAX) break;
        printf ("A? ");
        for (p=a; p<a+n; p++) scanf ("%d", p);
        for (p=a, q=a+n-1; p<q; p++, q--) {
            int b = *p; *p = *q; *q = b;
        }
        printf ("A=");
        for (p=a; p<a+n; p++) printf (" %d", *p);
    }
}
```

Zadatak 4

- Modifikacija – ciklus sa izlaskom na vrhu

```
#include <stdio.h>
#define NMAX 100

void main () {
    int a[NMAX], *p, *q, n;
    printf ("n? "); scanf ("%d", &n);

    while (n>0 && n<=NMAX) {
        printf ("A? "); for (p=a; p<a+n; p++) scanf ("%d", p);
        for (p=a, q=a+n-1; p<q; p++, q--) {
            int b = *p; *p = *q; *q = b;
        }
        printf ("A="); for (p=a; p<a+n; p++) printf (" %d", *p);

        printf ("n? "); scanf ("%d", &n);
    }
}
```

Dinamička alokacija memorije

- **Prilikom definisanja skalarnih promenljivih ili nizova, memorija se zauzima statički**
 - sve potrebe za memorijom moraju biti poznate unapred, pre prevođenja izvornog programa
 - alokira se najčešće više prostora nego što je potrebno
 - prostor se zauzima u statičkoj zoni memorije
 - bilo kakva izmena kapaciteta zahteva ponovno prevođenje programa
- **Rešenje – dinamička alokacija memorije**
 - radi se u vreme izvršavanja programa
 - broj i veličinu podataka ne moramo znati unapred
 - memorija se alokira u dinamičkoj zoni memorije (eng. heap)
 - podacima u dinamičkoj zoni memorije se pristupa preko pokazivača
 - ograničenje samo ukupan raspoloživ memorijski prostor u sistemu
- **Odgovornost programera za ispravnu alokaciju i dealokaciju dinamičke memorije!**

Manipulacija dinamičkom memorijom (1)

- **U C-u se za rad sa dinamičkom memorijom koriste funkcije iz standardnog zaglavlja `<stdlib.h>`**
- **Zauzimanje dinamičke memorije**
`void* malloc(vel)`
 - `vel` predstavlja traženu veličinu memorije u bajtovima**`void* calloc(n,vel)`**
 - `n` predstavlja broj elemenata niza
 - `vel` predstavlja veličinu jednog elementa
 - inicijalizuje zauzeti prostor nulama
- **Obe funkcije vraćaju generički (`void`) pokazivač!**
 - ovakav pokazivač sadrži samo adresu, a nije poznat tip podatka na koji ukazuje
 - ne može se dereferencirati, niti koristiti u adresnoj aritmetici, osim poređenja
 - ukoliko želimo da pristupimo memoriji preko njega, moramo izvršiti eksplicitnu konverziju (`cast`) u neki tip
- **Ukoliko je alokacija neuspešna ove funkcije vraćaju `NULL`**

Manipulacija dinamičkom memorijom (2)

- **Zauzeta memorija se obavezno mora osloboditi**
`free(p)`
 - gde je `p` pokazivač koji sadrži početnu adresu bloka u memoriju koji je zauzet sa `malloc`, `calloc` ili `realloc`
- **Promena veličine zauzetog prostora**
`void* realloc(p, vel)`
 - menja veličinu dodeljene memorije na koju ukazuje pokazivač `p` na `vel` bajtova
 - biće detaljno objašnjeno kasnije

Zadatak 4 – dinamički nizovi

- **Modifikovan da radi sa dinamičkim nizovima**

```
#include <stdio.h>
#include <stdlib.h>

void main () {
    int *a, *p, *q, n;
    printf ("n? ");
    scanf ("%d", &n);

    while (n>0) {
        a = calloc(n, sizeof(int));
        if (a == NULL) {
            printf("Neuspesna alokacija memorije!");
            exit(1);
        }
    }
```

Zadatak 4

```
printf ("A? ");
for (p=a; p<a+n; p++) scanf ("%d", p);

for (p=a, q=a+n-1; p<q; p++, q--) {
    int b = *p; *p = *q; *q = b;
}

printf ("A=");
for (p=a; p<a+n; p++) printf (" %d", *p);

free(a);

printf ("n? ");
scanf ("%d", &n);
}
```

Zadatak 5

- Date su sledeće deklaracije:

```
int i, x;
```

```
int *a, *b;
```

Koje od sledećih dodela su ispravne:

- a) $a+=i$; _____
- b) $a-=i$; _____
- c) $a = i$; _____
- d) $a *= i$; _____
- e) $a = b$; _____
- f) $a = b+i$; _____
- g) $a += b$; _____
- h) $a -= b$; _____
- i) $x = a>b$; _____

Zadatak 6 – matrice (C55c)

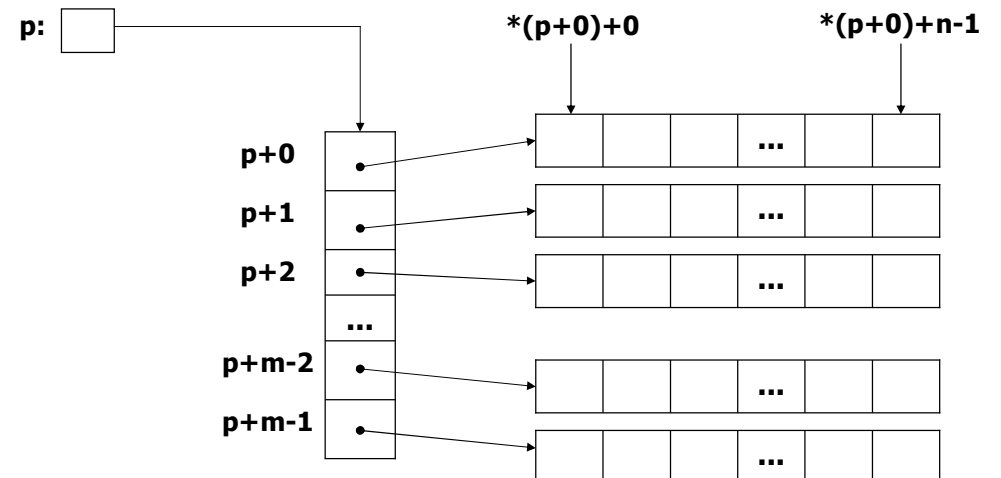
- Šta ispisuje sledeći program?

```
#include <stdio.h>
#include <stdlib.h>
void main() {
    int **a, n, i, j;
    printf("N="); scanf("%d", &n);
    a = calloc(n, sizeof(int*));
    for (i=0; i<n; i++) {
        *(a+i)= calloc(n, sizeof(int));
        for (j=0; j<n; j++)
            *(*a+i)+j) = rand()/((double)RAND_MAX + 1) * 10;
    }
    for (i=0; i<n; printf("\n"), i++)
        for (j=0; j<n; j++) printf("%d ", *(*a+i)+j));
    for (i=n-1; i>=0; i--) printf("%d ", *(*a+i)+n-1-i));
    putchar('\n');
}
```

- A) sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sleva-udesno
- B) sadržaj matrice vrstu po vrstu, a potom glavnu dijagonalu, sdesna-ulevo
- C) sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sdesna-ulevo

Dinamičke matrice (1)

- Osnovna ideja – matricu predstaviti kao dinamički niz pokazivača na dinamičke nizove elemenata



Dinamičke matrice (2)

- **U C-u je moguće definisanje pokazivača na neki drugi pokazivač, do proizvoljne "dubine"**

– pokazivač na pokazivač na celobrojni podatak

```
int **p;
```

- **Prvi korak u stvaranju dinamičke matrice je alokacija memorije za niz pokazivača**

```
p = calloc(m, sizeof(int*));
```

- `p` je statički alocirani dvostruki pokazivač u koji će biti smeštena adresa niza pokazivača
- `m` je broj vrsta (redova) matrice
- svaki pokazivač u alociranom nizu pokazivača će ukazivati na jedan niz elemenata koji će predstavljati vrste (redove) matrice
- `p` mora biti dvostruki pokazivač, jer da bi se pristupilo jednom elementu matrice (`p[i][j]`), mora se prvo odrediti adresa `i`-te vrste i pristupiti `i`-tom pokazivaču u nizu pokazivača, a zatim odrediti adresa `j`-tog elementa u nizu elemenata i pristupiti tom elementu

```
p[i][j] Ó (*(p+i)+j)
```

Dinamičke matrice (3)

- **Drugi korak u stvaranju dinamičke matrice je alokacija vrsta matrice**

```
for (i=0; i<m; i++)
```

```
*(p+i)= calloc(n, sizeof(int));
```

- u svakoj iteraciji petlje se alokira prostor za jednu od `m` vrsta matrice
- alocirani nizovi koji predstavljaju vrste matrice ne moraju biti fizički susedni u memoriji, već mogu biti razbacani bilo gde

- **Uništavanje (deallociranje) matrice se radi u obrnutom redosledu od redosleda kreiranja**

- prvo se deallociraju nizovi koji predstavljaju vrste, a zatim niz pokazivača na vrste

```
for (i=0; i<m; i++)
```

```
free (p[i]);
```

```
free (p);
```

Zadatak 6 - rešenje

```
printf("N="); scanf("%d", &n);
```

```
/*alocira memoriju za n pokazivaca na vrste*/
```

```
a = calloc(n, sizeof(int*));
```

```
for (i=0; i<n; i++) {
```

```
/*alocira memoriju za n elemenata vrste koji su tipa int*/
```

```
*(a+i)= calloc(n, sizeof(int));
```

```
for (j=0; j<n; j++)
```

```
*(*(a+i)+j) = rand()/((double)RAND_MAX + 1) * 10;
```

```
}
```

- **Nakon izvršenja ovog dela koda biće stvorena kvadratna dinamička matrica dimenzija `n x n`**
 - dimenzija `n` se unosi sa glavnog ulaza
- **Matrica će biti popunjena pseudoslučajnim brojevima u rasponu od 0 do 9**

Zadatak 6

- **Deo koda:**

```
for (i=0; i<n; printf("\n"), i++)
```

```
for (j=0; j<n; j++) printf("%d ", (*(a+i)+j));
```

- **ispisuje sadržaj matrice, vrstu po vrstu**

- nakon svake iteracije spoljne petlje će u post uslovu biti odštampan znak za novi red

- **Šta radi sledeći deo koda?**

```
for (i=n-1; i>=0; i--)
```

```
printf("%d ", (*(a+i)+n-1-i));
```

- **Uzmimo primer kvadrata matrice za `n = 4`**

Zadatak 6

- Izraz:** `*(*(a+i)+n-1-i);`

Za $i = 3$

`*(a + 3) \circ a[3]`

`*(*(a + 3) + 4 - 1 - 3) \circ a[3][0]`

a[0]	->	1	2	3	4
a[1]	->	5	6	7	8
a[2]	->	9	10	11	12
a[3]	->	13	14	15	16

Za $i = 1$

`*(a + 1) \circ a[1]`

`*(*(a + 1) + 4 - 1 - 1) \circ a[1][2]`

a[0]	->	1	2	3	4
a[1]	->	5	6	7	8
a[2]	->	9	10	11	12
a[3]	->	13	14	15	16

- Ispisuje sadržaj sporedne dijagonale matrice sleva-udesno (odozdo nagore)**
- Odgovor: A**

Zadatak 7 – transponovanje (4.5)

- Sastaviti na jeziku C program za transponovanje pravougaone matrice celih brojeva. Matricu smestiti u dinamičku zonu memorije. Postupak ponavljati sve dok se za dimenzije matrice ne unesu nekorektne vrednosti.**
- Transponovanje matrice:**
 - zamena mesta vrsta i kolona matrice
 - prva vrsta postaje prva kolona, druga vrsta druga kolona itd...

1	1	1
2	2	2
3	3	3
4	4	4

=>

1	2	3	4
1	2	3	4
1	2	3	4

Zadatak 7 - rešenje

- Rešenje – uz očuvanje originalne matrice**

```
#include <stdio.h>
#include <stdlib.h>

void main () {
    int **a, **b, m, n, i, j;

    printf ("\nBroj vrsta i kolona? ");
    scanf ("%d%d", &m, &n);

    while (m>0 && n>0) {

        /* Stvaranje matrice i citanje elemenata matrice: */
        a = malloc (m*sizeof(int*));
        for (i=0; i<m; i++) {
            a[i] = malloc (n*sizeof(int));
            printf ("%2d. vrsta? ", i);
            for (j=0; j<n; j++) scanf ("%d", &a[i][j]);
        }
    }
}
```

Zadatak 7

```
/* Obrazovanje transponovane matrice: */
b = malloc (n*sizeof(int*));
for (i=0; i<n; i++) {
    b[i] = malloc (m*sizeof(int));
    for (j=0; j<m; j++) b[i][j] = a[j][i];
}

/* Ispisivanje rezultata: */
printf ("\nTransponovana matrica:\n");
for (i=0; i<n; i++){
    for (j=0; j<m; j++) printf ("%5d", b[i][j]);
    printf ("\n");
}

/* Unistavanje matrice: */
for (i=0; i<m; i++) free (a[i]); free (a);
for (i=0; i<n; i++) free (b[i]); free (b);

/* Citanje dimenzija matrice: */
printf ("\nBroj vrsta i kolona? ");
scanf ("%d%d", &m, &n);
}
```


Zadatak 7

- **Rešenje – uz izmenu originalne matrice**

- zadata matrica ne mora biti kvadratna
- mora se alocirati prostor kao za kvadratnu matricu čija je dimenzija jednaka većoj dimenziji zadate matrice

```
#include <stdio.h>
#include <stdlib.h>

void main () {
    int **a, dim, m, n, i, j, t;

    /* Citanje dimenzija matrice: */
    printf ("\nBroj vrsta i kolona? ");
    scanf ("%d%d", &m, &n);

    while (m>0 && n>0) {
        dim = (m > n) ? m : n;
```

Zadatak 7

```
a = malloc (dim*sizeof(int*));
for (i=0; i<dim; i++) {
    a[i] = malloc (dim*sizeof(int));
    if (i<m) {
        printf ("%2d. vrsta? ", i+1);
        for (j=0; j<n; j++) scanf ("%d", &a[i][j]);
    }
}
/* Obrazovanje transponovane matrice: */
for (i=1; i<dim; i++)
    for (j=0; j<i; j++) {
        t = a[i][j]; a[i][j] = a[j][i]; a[j][i] = t;
    }

printf ("\nTransponovana matrica:\n");
for (i=0; i<n; i++)
    { for (j=0; j<m; j++) printf ("%5d", a[i][j]); printf ("\n"); }
for (i=0; i<dim; i++) free (a[i]); free (a);
printf ("\nBroj vrsta i kolona? "); scanf ("%d%d", &m, &n);
}
```