



PROGRAMIRANJE I

P-06: Složeni tipovi podataka

prof. dr **Dražen Brđanin**
2023/24



P-06: Složeni tipovi podataka

■ Sadržaj predavanja

- prosti i složeni tipovi podataka
- jednodimenzionalni niz
- dvodimenzionalni niz
- višedimenzionalni nizovi
- strukture
- unije
- nabranjanja (enumeracije)
- korisnički definisani tipovi



Prosti i složeni tipovi podataka

Tipovi podataka

- Svaki podatak koji se koristi u programu je nekog tipa
- **Svaki tip podataka karakterišu:**
 - vrsta podataka koje opisuje
 - način reprezentacije i format
 - dozvoljene operacije
- S obzirom na to da li dati tip već postoji u jeziku ili ga mora definisati programer, razlikujemo:
 - **standardni (ugrađen u jezik, built-in)**
 - **korisnički definisan**
- S obzirom na to da li dati tip omogućava reprezentaciju jednostavnijih podataka (npr. jedna cjelobrojna vrijednost) ili složenih podataka (npr. niz vrijednosti), tip podataka može biti:
 - **prosti**
 - **složeni**

Složeni tipovi podataka u jeziku C

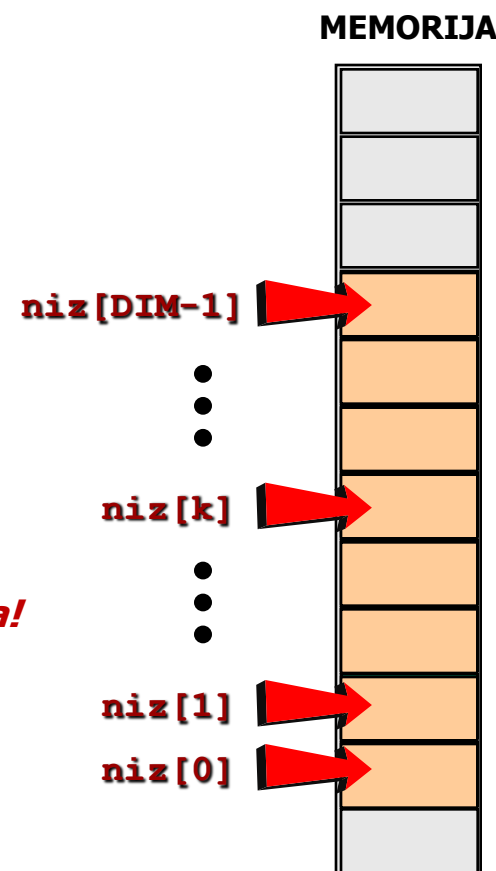
- C raspolaže sljedećim složenim tipovima podataka:
 - **niz (indeksirana promjenljiva)**
 - **jednodimenzionalni = NIZ/VEKTOR,**
 - **dvodimenzionalni = MATRICA,**
 - **višedimenzionalni**
 - **struktura (slog/zapis)**
 - **unija**
- S obzirom na to kakve podatke sadrži, složeni tip može biti:
 - **homogeni** (podaci istog tipa)
 - **heterogeni** (podaci različitih tipova)

Jednodimenzionalni niz

Jednodimenzionalni niz

- Niz je **homogena kolekcija podataka** (podaci istog tipa), koji su **u memoriji smješteni sekvencijalno** (na uzastopnim memorijskim lokacijama).
- Svaki element u nizu jedinstveno je određen:
 - **imenom niza**, i
 - **indeksom** (pomjeraj u odnosu na početak niza)
- **IME NIZA** predstavlja početnu adresu niza u memoriji
- Alternativni nazivi u literaturi:
NIZ, VEKTOR, POLJE, eng. *ARRAY*
- Kada se koriste nizovi?
Kada je neophodno MEMORISANJE većeg broja podataka istog tipa!
Čak i kada se u nekom programu koristi veća količina podataka, to nužno ne znači da ih sve treba pamtit (npr. odrediti najveći broj u nekom skupu, ili odrediti aritmetičku sredinu nekog skupa)!

Sekvencijalna reprezentacija niza



Jednodimenzionalni niz

Deklaracija jednodimenzionalnog niza

- Niz je promjenljiva i mora da se deklarirše (definiše), kao i svaka druga promjenljiva

Opšti oblik deklaracije:

`tip ime_niza [broj_elemenata] = {lista_vrijednosti};`

tip niza
(tip podataka u nizu)

ime niza
(identifikator u skladu sa sintaksom jezika, važe sva pravila kao i za skalarne promjenljive)

broj elemenata u nizu
(cjelobrojna konstanta)
(rezervirše se potreban broj bajtova za memorisanje deklarisanog broja elemenata)

Inicijalne vrijednosti elemenata niza
(inicijalizacija niza)
(nije obavezno inicijalizovanje)
(vrijednosti se razdvajaju zapetama)

Jednodimenzionalni niz

Deklaracija jednodimenzionalnog niza

Primjer deklaracije:

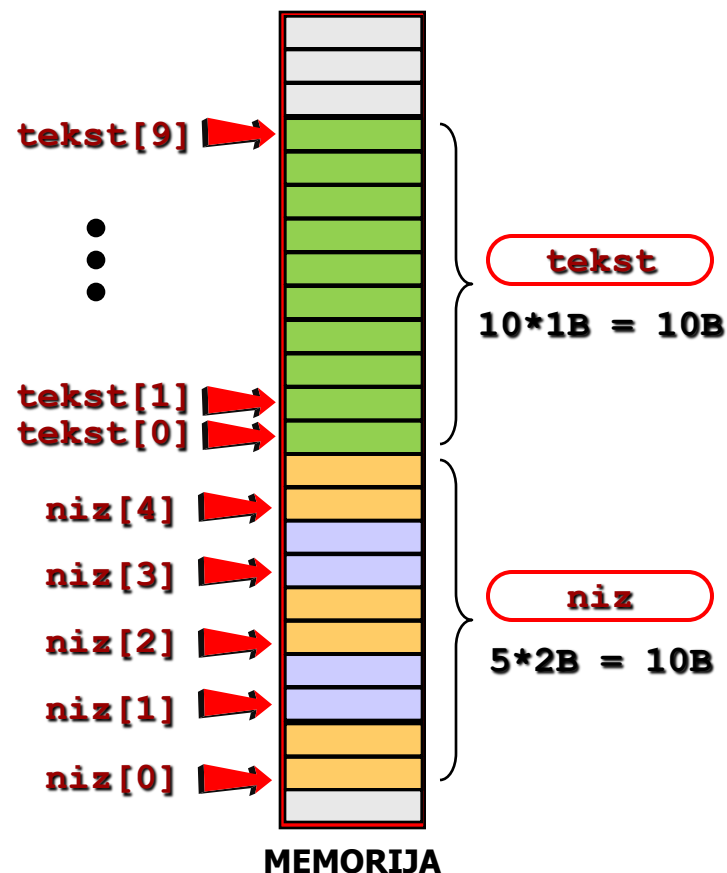
```
char tekst[10];  
short niz[5];  
float f1[10], f2[10];
```

Primjer deklaracije sa korištenjem simboličkih konstanti:

```
#define MAX 5  
#define LEN 10  
int main()  
{  
    char tekst[LEN];  
    short niz[MAX];  
    float f1[LEN], f2[LEN];  
    ...  
}
```

Primjer neispravne deklaracije (nije specifikovana dimenzija niza):

```
double d[];
```



Jednodimenzionalni niz

Deklaracija jednodimenzionalnog niza

Primjer deklaracije sa inicijalizacijom:

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

a[0]	a[1]	a[2]
1	2	3

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

b[0]	b[1]	b[2]	b[3]
1	2	3	4

`int c[5]={10};`

c[0]	c[1]	c[2]	c[3]	c[4]
10	0	0	0	0

`int d[5]={0};`

d[0]	d[1]	d[2]	d[3]	d[4]
0	0	0	0	0

`int e[5]={};`

e[0]	e[1]	e[2]	e[3]	e[4]
0	0	0	0	0

`int f[5]={1,[3]=5};`

f[0]	f[1]	f[2]	f[3]	f[4]
1	0	0	5	0

potpuna inicijalizacija
(navedene inicijalne vrijednosti svih elemenata)

dimenzija niza može da se izostavi
samo ako se niz inicijalizuje –
dimenzija je jednaka broju
vrijednosti u inicijalizatoru

ako je lista vrijednosti nepotpuna
– elementi se inicijalizuju redom,
a preostali elementi imaju
početnu vrijednost 0

ako je lista prazna lista –
elementi se inicijalizuju 0

moгуće je eksplicitno navesti
indeks elementa koji se
inicijalizuje



Jednodimenzijski niz

Operacije nad jednodimenzijskim nizom

- Jezik C nema operacije koje se izvode na kompletnim nizom, nego se operacije izvode nad pojedinačnim elementima niza
- **Pristup elementu niza** izvodi se primjenom operatora indeksiranja []
- Indeks može biti specifikovan kao konstantna vrijednost ili kao izraz čijim se sračunavanjem dobija cjelobrojna vrijednost

Npr.

```
tekst[0]='A';           // dodjela vrijednosti 'A' elementu tekst[0]
niz[i]=100;             // dodjela vrijednosti 100 elementu niz[i]
fib[i]=fib[i-1]+fib[i-2]; // izracunavanje i-tog Fibonacijevog broja
                        // i-ti F. broj jednak je sumi prethodna dva F. broja
printf("x[%d]=%5.2f", i, x[i]); // ispis elementa niza, npr. x[1]=25.75
scanf("%d", &niz[i]);      // učitavanje elementa niz[i]
                        // u memoriju na adresu &niz[i]

niz[2.75]=100.75;        // pogresno: indeks mora biti cjelobrojan
niz[(int)2.75]=100.75;   // ispravno: indeks je cjelobrojan
```


Jednodimenzionalni niz

Operacije nad jednodimenzionalnim nizom

- **Element niza jeste L-value**, tj. element niza je ekvivalentan skalarnoj promjenljivoj istog tipa

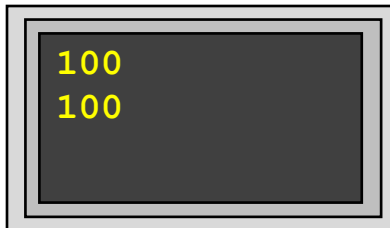
Npr.

```
tekst[0]='A'; // dodjela vrijednosti
               // 'A' elementu tekst[0]

niz[i]++;     // inkrementovanje
               // elementa niz[i]
```

Izrazi `niz[indeks]` i `indeks[niz]` mogu ravnopravno da se koriste

```
niz[0]=100;
printf("%d\n", niz[0]);
printf("%d\n", 0[niz]);
```



- **Niz nije L-value**, tj. ime niza sadrži adresu početka niza u memoriji, a to ne može da se mijenja

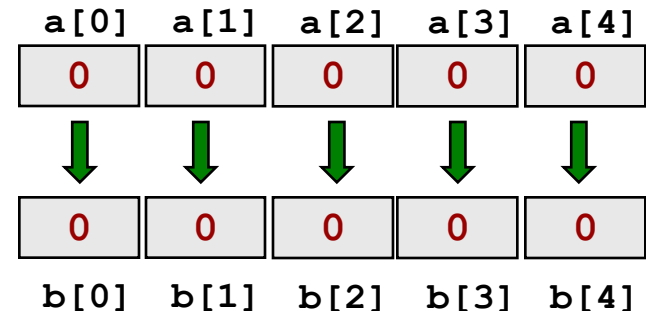
Npr.

```
int a[5]={}, b[5];
```

```
b=a; // greska: ovako ne moze da se
      // kopira niz a u niz b
```

```
// ispravno kopiranje niza:
// element po element
```

```
for (int i=0; i<5; i++)
    b[i]=a[i];
```



Jednodimenzionalni niz

Operacije nad jednodimenzionalnim nizom

➤ Učitavanje i ispisivanje niza izvodi se element po element

```
#include <stdio.h>
#define MAX 100
int main()
{
```

```
    int niz[MAX]; }
```

Deklaracija niza sa MAX elemenata

```
    int i,n;
    do
    {
        printf("n="); scanf("%d", &n);
    }
    while (n<1 || n>MAX);
```

U nizu ima mjesta za MAX elemenata, ali to ne znači da će u konkretnom slučaju baš biti toliko podataka. Zato n predstavlja stvaran broj podataka, koji mora biti u granicama [1..MAX]

```
    for ( i=1 ; i<=n ; i++ )
    {
        printf("Unesite %d. broj:", i); scanf("%d", &niz[i-1]);
    }
```

Učitavanje niza element po element

```
    printf("Unijeli ste: ");
    for ( i=0 ; i<n ; i++ ) printf(" %d", niz[i]);
    return 0;
```

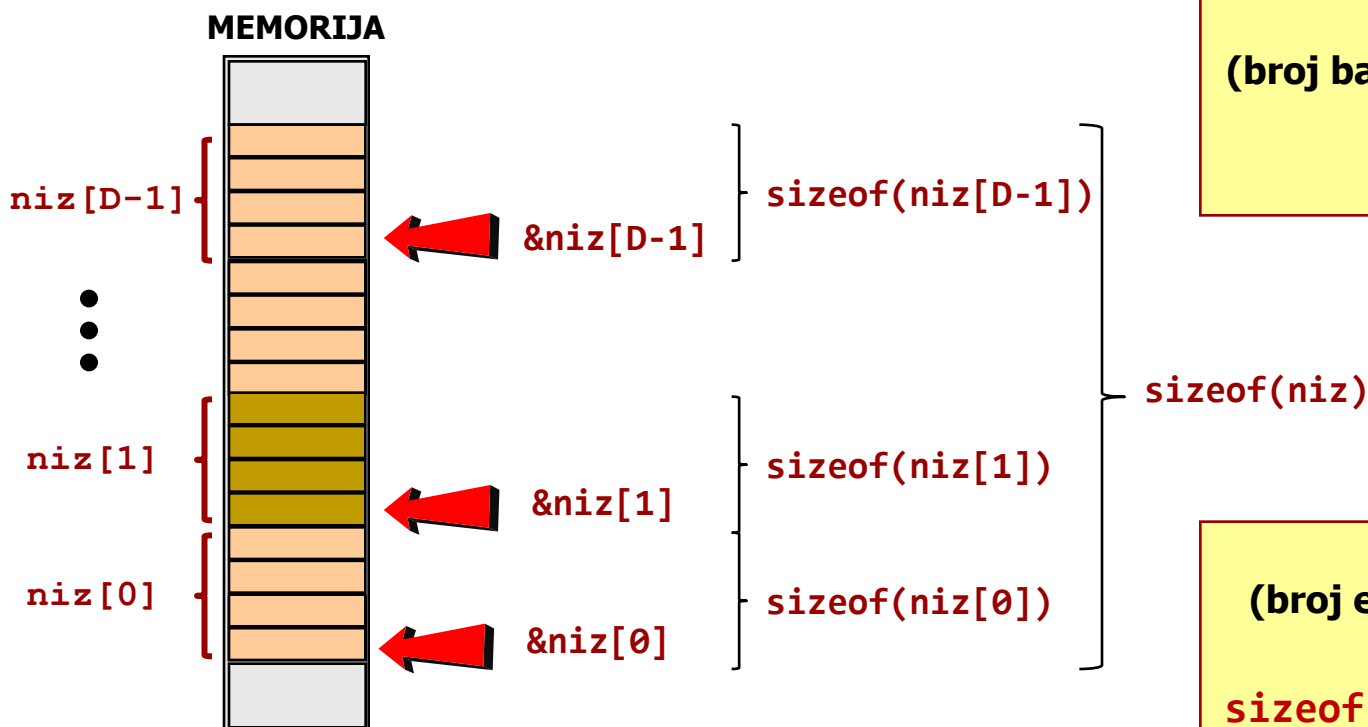
Ispisivanje niza element po element

```
}
```

Jednodimenzionalni niz

Razmještaj niza u memoriji

- Niz je **homogena kolekcija podataka** (podaci istog tipa), koji su **u memoriji smješteni sekvencijalno** (na uzastopnim memorijskim lokacijama).



Veličina elementa `niz[indeks]`
(broj bajtova koje element zauzima u memoriji)
`sizeof(niz[indeks])`

Veličina niza
(broj bajtova koje niz zauzima u memoriji)
`sizeof(niz)`

Kapacitet niza
(broj elemenata koji mogu da stanu u niz)
`sizeof(niz)/sizeof(niz[0])`



Jednodimenzionalni niz

Razmještaj niza u memoriji

Primjer:

```
#include <stdio.h>
int main()
{
    int niz[5]={};

    printf("niz: %p\n", niz);
    printf("&niz[0]: %p\n", &niz[0]);
    printf("&niz[1]: %p\n", &niz[1]);
    printf("&niz[2]: %p\n", &niz[2]);
    printf("&niz[1]-&niz[0]: %d\n", &niz[1]-&niz[0]);
    printf("(int)&niz[1]-(int)&niz[0]: %d\n", (int)&niz[1]-(int)&niz[0]);
    printf("sizeof(niz[0]): %d\n", sizeof(niz[0]));
    printf("sizeof(niz): %d\n", sizeof(niz));

    return 0;
}
```

```
niz: 0028ff0c
&niz[0]: 0028ff0c
&niz[1]: 0028ff10
&niz[2]: 0028ff14
&niz[1]-&niz[0]: 1
(int)&niz[1]-(int)&niz[0]: 4
sizeof(niz[0]): 4
sizeof(niz): 20
```

Jednodimenzijski niz

Kontrola pristupa van granica niza

➤ Jezik C nema kontrolu pristupa elementima van granica (neki programski jezici imaju)

- u fazi prevođenja programa ne vrši se kontrola pristupa van granica ($\text{indeks} < 0 \mid \mid \text{indeks} > \text{DIM}-1$), ali u fazi izvršavanja programa može da dođe do grešaka
- čitanje tipično neće rezultovati *run-time* greškom, ali upis može da rezultuje *run-time* greškom (*segmentation fault*)

Primjer:

```
#include <stdio.h>
int main()
{
```

```
    int niz[5]={};
```

```
    printf("niz[-1]:  %x\n", niz[-1]);
    printf("niz[5]:  %x\n", niz[5]);
```

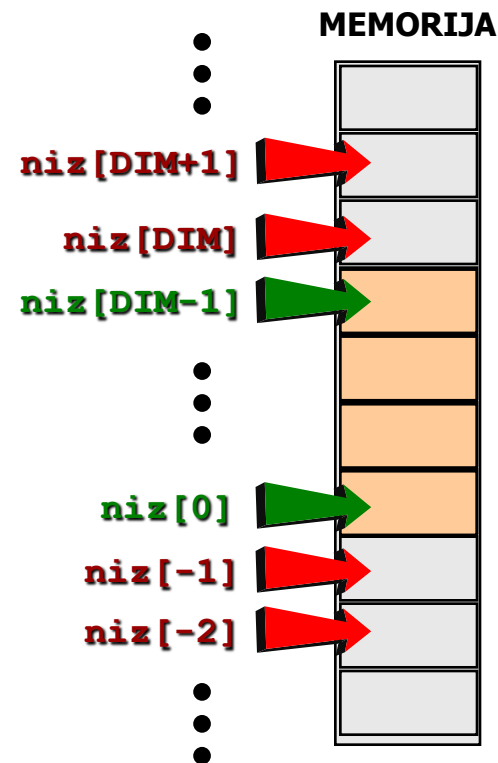
Čitanje izvan granica niza
neće izazvati grešku

```
    niz[100] = 10; }
```

Upis izvan granica niza prouzrokuje grešku

```
    return 0;
```

```
}
```



Jednodimenzionalni niz

Nizovi promjenljive dužine

- **Prethodno smo koristili nizove fiksne dimenzije** (dimenzija je specifikovana prilikom definicije niza, ili je indirektno bila specifikovana veličinom inicijalizatora)
- Novije verzije standarda (nakon C99) omogućavaju i rad sa **nizovima promjenljive dužine** (engl. *Variable Length Array* – VLA)
 - **Dimenzija nije specifikovana kao konstantna vrijednost koja je poznata u fazi prevođenja, nego se određuje u toku izvršavanja** (zato se često zovu *run time-sized array*)
 - Prednost se ogleda u tome što **ne moramo da unaprijed procjenjujemo najgori slučaj i ne moramo da definišemo niz maksimalne dimenzije**, nego **definišemo niz potrebne dimenzije**
 - **Ne možemo da očekujemo da je na svakoj platformi omogućeno korištenje nizova promjenljive dužine!**
 - **Nizovi promjenljive dužine ne mogu da se inicijalizuju prilikom definicije!**

Primjer:

```
#include <stdio.h>
int main()
{
    int n;
    do
    {
        printf("n? "); scanf("%d", &n);
    } while (n<1);
    int niz[n];
    printf("sizeof: %d\n", sizeof(niz));
    for (int i=0; i<n; i++)
        printf("%d ", niz[i]);
    return 0;
}
```

```
n? 3
sizeof: 12
3 4199375 4227112
```

Jednodimenzionalni niz

Primjeri primjene jednodimenzionalnih nizova

Primjer (Reprezentacija polinoma)

```
#include <stdio.h>
#define MAX 10
int main()
{
    int a[MAX], b[MAX];
    int c[2*MAX]={0};
    int i,j,m,n;

    // učitavanje polinoma A
    printf("Polinom A:\n");
    do
    {
        printf("Stepen (m)? ");
        scanf("%d", &m);
    }
    while ((m<0) || (m>=MAX));
    for ( i=m; i>=0; i-- )
    {
        printf("Koef. a%d? ",i);
        scanf("%d", &a[i]);
    }
}
```

```
// učitavanje polinoma B
printf("Polinom B:\n");
do
{
    printf("Stepen (n)? ");
    scanf("%d", &n);
}
while ((n<0) || (n>=MAX));
for ( i=n; i>=0; i-- )
{
    printf("Koef. b%d? ",i);
    scanf("%d", &b[i]);
}
}
```

```
// izračunavanje proizvoda
for ( i=0; i<=m; i++ )
    for ( j=0; j<=n; j++ )
        c[i+j] += a[i] * b[j];

// ispis proizvoda
printf("Proizvod:\n");
for ( i=m+n; i>=0; i-- )
    printf(" %d%d ", c[i], i);

return 0;
}
```

Polinom A:

$$A_m(x) = a_m x^m + a_{m-1} x^{m-1} + \dots + a_1 x + a_0$$

Polinom B:

$$B_n(x) = b_n x^n + b_{n-1} x^{n-1} + \dots + b_1 x + b_0$$

Sabiranje dva monoma:

$$a_i x^i + b_i x^i = (a_i + b_i) x^i$$

Množenje dva monoma:

$$a_i x^i * b_j x^j = a_i * b_j x^{i+j}$$

Jednodimenzionalni niz

Primjeri primjene jednodimenzionalnih nizova

Primjer (Reprezentacija velikih brojeva)

Standardni prosti tipovi (int, long, long long)
ne omogućavaju izračunavanje više od 20!
 $19! = 121645100408832000$

Rješenje problema?

Problemi "velikih brojeva" rješavaju se
tako što se broj posmatra kao niz cifara!!!

	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
1 !=	0	0	0	0	0	0	0	0	0	1
	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
2 !=	0	0	0	0	0	0	0	0	0	2
	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
3 !=	0	0	0	0	0	0	0	0	0	6
	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
4 !=	0	0	0	0	0	0	0	0	2	4
	f[9]	f[8]	f[7]	f[6]	f[5]	f[4]	f[3]	f[2]	f[1]	f[0]
5 !=	0	0	0	0	0	0	0	1	2	0

```
#include <stdio.h>
#define MAXCIF 1000
int main()
{
    int f[MAXCIF]={1};
    int i,j,n, pom, prenos;

    do
    { printf("n="); scanf("%d", &n); }
    while ((n<1) || (n>100));

    for ( i=2; i<=n; i++ )
        for ( j=0, prenos=0; j<MAXCIF; j++ )
        {
            pom = f[j]*i + prenos;
            f[j] = pom % 10;
            prenos = pom / 10;
        }

    printf("%d!=",n);
    for ( j=MAXCIF-1; f[j]==0; j-- );

    for (; j>=0; j-- )
        printf("%d",f[j]);

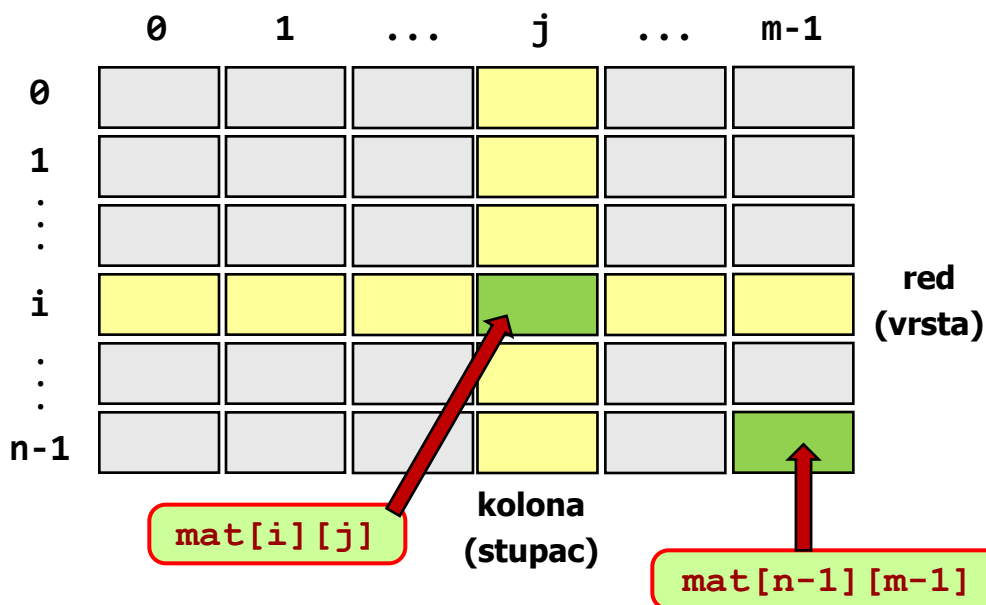
    return 0;
}
```


Dvodimenzionalni niz (matrica)

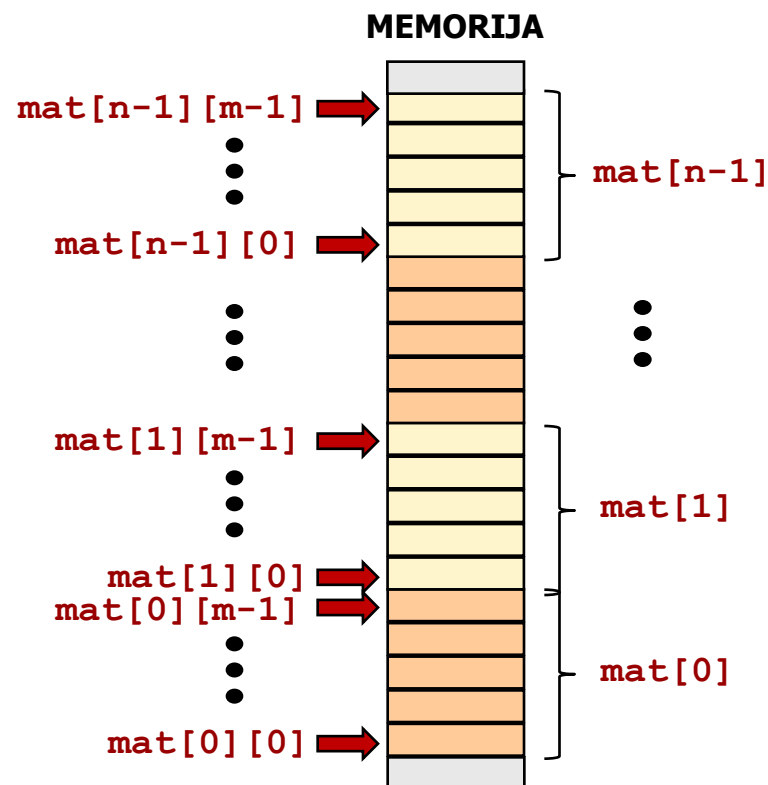
Dvodimenzionalni niz

- Dvodimenzionalni niz je niz nizova
(niz čiji su elementi nizovi)

Logička reprezentacija



Fizička reprezentacija (razmještaj u memoriji)

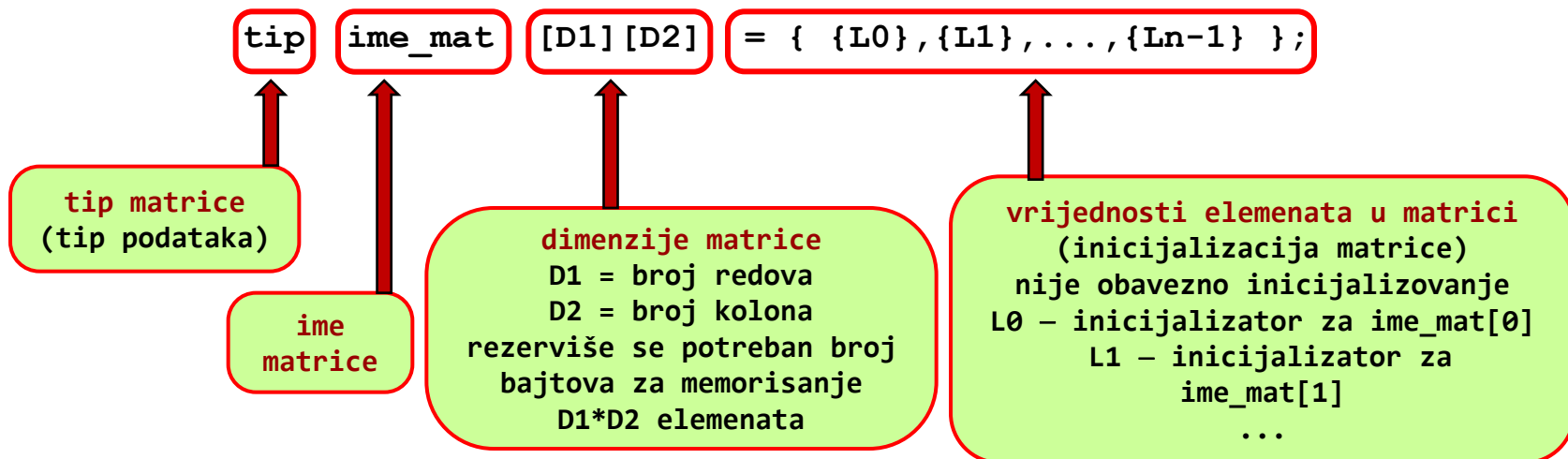


Dvodimenzionalni niz (matrica)

Deklaracija dvodimenzionalnog niza

- Niz je promjenljiva i mora da se deklarise (definiše), kao i svaka druga promjenljiva

Opšti oblik deklaracije:



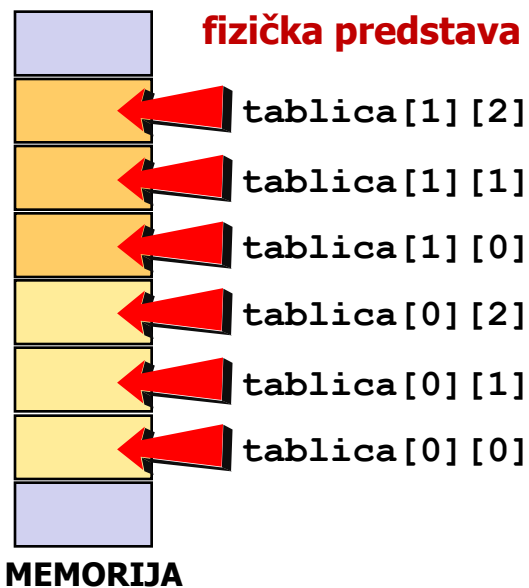
Dvodimenzionalni niz (matrica)

Deklaracija dvodimenzionalnog niza

Primjer deklaracije:

```
char tablica[2][3];
```

	0	1	2	
0				logička reprezentacija
1				



Primjer deklaracije sa inicijalizacijom:

```
int mat[2][3] = { {3,1,8}, {2,5,6} };
```

	0	1	2
0	3	1	8
1	2	5	6

Primjer deklaracije sa inicijalizacijom:

```
int mat[2][3] = { 3,1,8,2,5,6 };
```

	0	1	2
0	3	1	8
1	2	5	6

Dvodimenzionalni niz (matrica)

Deklaracija dvodimenzionalnog niza

Primjer deklaracije sa inicijalizacijom:

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

	0	1	2
0	3	1	0
1	0	0	0

Primjer deklaracije sa inicijalizacijom:

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

	0	1	2
0	3	1	0
1	2	0	0

Primjer deklaracije sa inicijalizacijom:

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

	0	1	2
0	0	0	0
1	3	1	0

Primjer deklaracije sa inicijalizacijom:

```
int mat[][3] = { [1]={3,1} };
```

	0	1	2
0	0	0	0
1	3	1	0

Prva dimenzija može da se izostavi, ako je naveden inicijalizator na osnovu kojeg se može zaključiti dimenzija



Dvodimenzionalni niz (matrica)

Operacije nad dvodimenzionalnim nizom

- Kao i kod jednodimenzionalnih nizova, i kod dvodimenzionalnih nizova operacije se izvode nad pojedinačnim elementima
- **Pristup elementu dvodimenzionalnog niza** izvodi se primjenom dva operatora indeksiranja []
`mat[red][kolona]`
- Indeksi mogu biti specifikovani kao konstantne vrijednosti ili kao izrazi čijim se sračunavanjem dobija cjelobrojna vrijednost

Npr.

```
stranica[0][0]='A';  
zbir[i][j]=a[i][j]+b[i][j];  
  
printf("x[%d][%d]=%.2f", i,j,x[i][j]);  
scanf("%d", &mat[i][j]);  
  
mat[2.75][2]=100.75;  
mat[1,2]=10;
```

```
// dodjela vrijednosti elementu matrice  
// izracunavanje elementa [i][j] u matrici  
// koja predstavlja zbir matrica a i b  
// ispis elementa matrice, npr. x[1][2]=2.50  
// učitavanje elementa mat[i][j]  
// u memoriju na adresu &mat[i][j]  
// pogresno: indeks mora biti cjelobrojan  
// pogresno: mat[1,2] je ekvivalentno mat[2]  
// a mat[2] je niz, a nizu ne može da se  
// dodijeli skalarna vrijednost
```

Dvodimenzionalni niz (matrica)

Operacije nad dvodimenzionalnim nizom

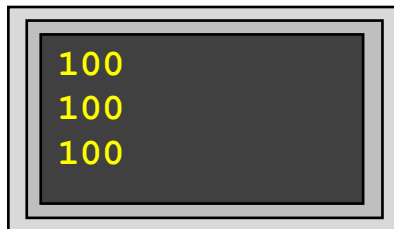
- **Element matrice jeste L-value**, tj. element matrice je ekvivalentan skalarnoj promjenljivoj istog tipa

Npr.

```
stranica[0][0]='A'; // dodjela vrijednosti  
mat[i][j]++;      // inkrementovanje elementa  
                // mat[i][j]
```

`mat[i][j] ⇔ j[mat[i]] ⇔ j[i[mat]]`
mogu ravnopravno da se koriste

```
m[1][2]=100;  
printf("%d\n", m[1][2]);  
printf("%d\n", 2[m[1]]);  
printf("%d\n", 2[1[m]]);
```



- **Matrica nije L-value**, tj. ime matrice sadrži adresu početka matrice u memoriji, a to ne može da se mijenja

Npr.

```
int a[2][2], b[2][2]={};  
a=b;    // greska: ovako ne moze  
        // da se kopira matrica  
  
// ispravno kopiranje matrice:  
// element po element  
for (int i=0; i<2; i++)  
    for (int j=0; j<2; j++)  
        a[i][j]=b[i][j];  
  
a[i]=b[i];    // greska: niz ne moze  
              // da se kopira  
  
// ispravno kopiranje i-tog reda  
// matrice: element po element  
for (int j=0; j<2; j++)  
    a[i][j]=b[i][j];
```



Dvodimenzionalni niz (matrica)

Operacije nad jednodimenzionalnim nizom

➤ Učitavanje i ispisivanje matrice izvodi se element po element

```
#include <stdio.h>
#define MAX 10
int main()
{
    int i,j,n,m, mat[MAX][MAX];
    // učitavanje stvarnih dimenzija matrice
    do
    {
        printf("n="); scanf("%d", &n);
    }
    while (n<1 || n>MAX);
    do
    {
        printf("m="); scanf("%d", &m);
    }
    while (m<1 || m>MAX);
```

```
// učitavanje elemenata matrice
for ( i=0; i<n; i++ )
    for ( j=0; j<m; j++ )
    {
        printf("mat[%d][%d]:", i,j);
        scanf("%d", &mat[i][j]);
    }

// ispisivanje matrice
for ( i=0; i<n ; i++ )
{
    for ( j=0; j<m; j++ )
        printf(" %d", mat[i][j]);
    printf("\n");
}
return 0;
}
```

U matrici ima mjesta za $MAX * MAX$ elemenata, ali to ne znači da će u konkretnom slučaju baš biti toliko podataka. Zato n predstavlja stvarni broj redova, a m predstavlja stvarni broj kolona, koji moraju biti u granicama $[1..MAX]$

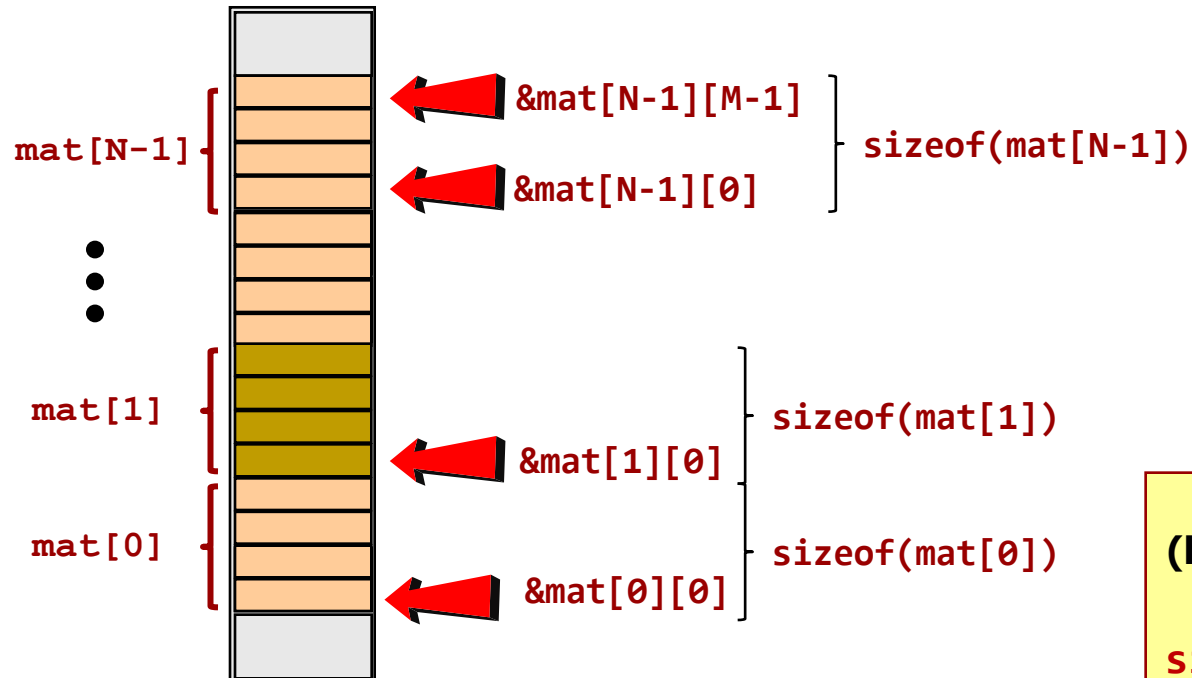
Dvodimenzionalni niz (matrica)

Razmještaj matrice u memoriji

- Matrica je niz nizova, koji su **u memoriji smješteni sukcesivno** (niz po niz).

```
int mat[N][M];
```

MEMORIJA



Veličina elementa `mat[i][j]`
(broj bajtova koje element
zauzima u memoriji)
`sizeof(mat[i][j])`

Veličina jednog reda
(broj bajtova koje red zauzima u
memoriji)
`sizeof(mat[red])`

Veličina matrice
(broj bajtova koje matrica
zauzima u memoriji)
`sizeof(mat)`

Kapacitet matrice
(broj elemenata koji mogu da stanu
u matricu)
`sizeof(mat)/sizeof(mat[0][0])`

Dvodimenzionalni niz (matrica)

Razmještaj matrice u memoriji

- Treba voditi računa o rasporedu elemenata, ako su stvarne dimenzije matrice manje od maksimalnih.

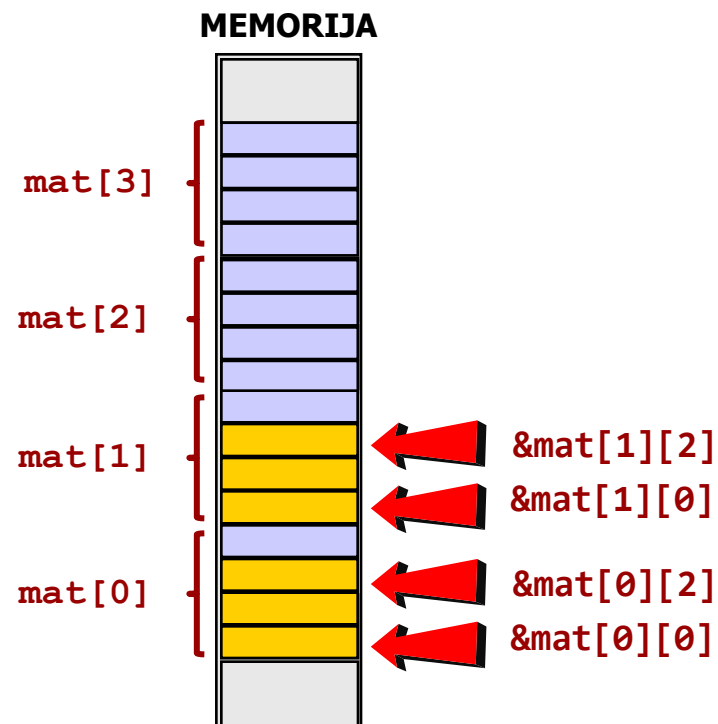
Primjer:

```
#define MAX 4  
int mat[MAX][MAX];  
int n=2,m=3; // stvarni broj redova i kolona
```

Logička reprezentacija

	0	1	2	3
0				
1				
2				
3				

Fizička reprezentacija





Dvodimenzionalni niz (matrica)

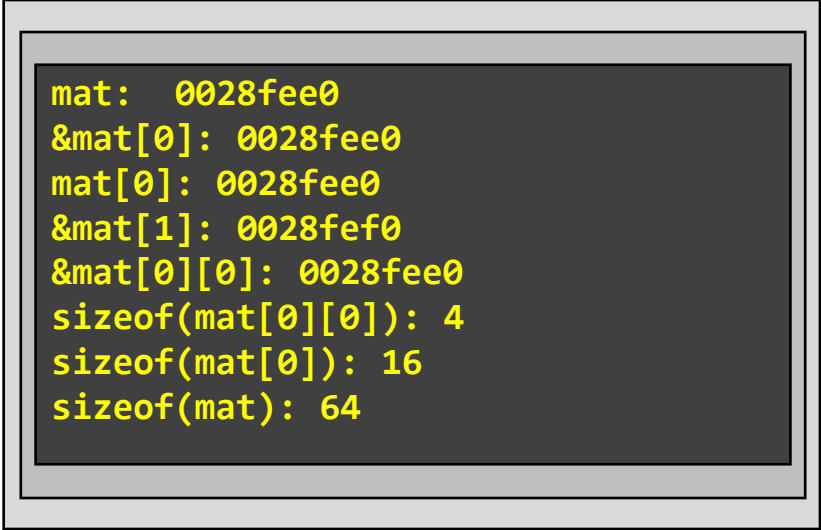
Razmještaj matrice u memoriji

Primjer:

```
#include <stdio.h>
#define MAX 4
int main()
{
    int mat[MAX][MAX];

    printf("mat: %p\n", mat);
    printf("&mat[0]: %p\n", &mat[0]);
    printf("mat[0]: %p\n", mat[0]);
    printf("&mat[1]: %p\n", &mat[1]);
    printf("&mat[0][0]: %p\n", &mat[0][0]);
    printf("sizeof(mat[0][0]): %d\n", sizeof(mat[0][0]));
    printf("sizeof(mat[0]): %d\n", sizeof(mat[0]));
    printf("sizeof(mat): %d\n", sizeof(mat));

    return 0;
}
```



```
mat: 0028fee0
&mat[0]: 0028fee0
mat[0]: 0028fee0
&mat[1]: 0028fef0
&mat[0][0]: 0028fee0
sizeof(mat[0][0]): 4
sizeof(mat[0]): 16
sizeof(mat): 64
```



Dvodimenzionalni niz (matrica)

Matrice promjenljive dužine

- Kao što je novijim verzijama standarda omogućeno korištenje jednodimenzionalnih nizova promjenljive dužine, moguće je koristiti i matrice promjenljivih dimenzija

Primjer:

```
#include <stdio.h>
int main()
{
    int n,m;

    do
    {
        printf("n? "); scanf("%d", &n);
        printf("m? "); scanf("%d", &m);
    } while (n<1 || m<1);

    int mat[n][m];

    printf("sizeof: %d\n", sizeof(mat));

    return 0;
}
```



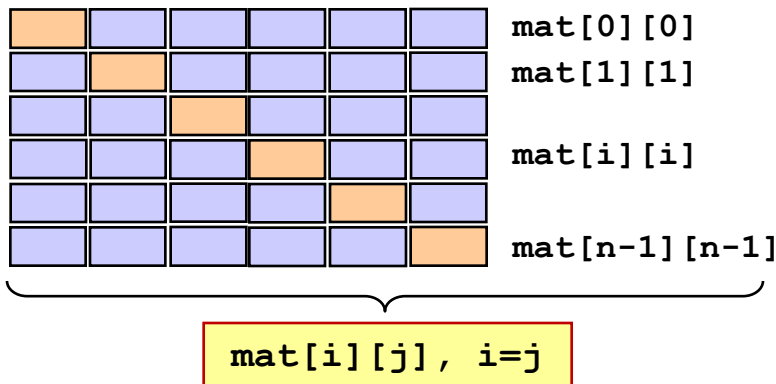
```
n? 5
m? 6
sizeof: 120
```

Dvodimenzionalni niz (matrica)

Primjeri primjene dvodimenzionalnih nizova

Primjer (Neke operacije nad matricama)

manipulacija glavnom dijagonalom:



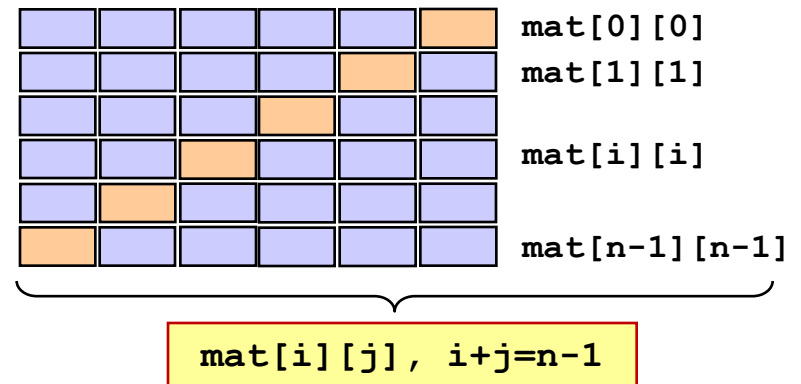
// ispis elemenata na GD

```
for ( i=0; i<n; i++ )  
    printf(" %d", mat[i][i]);
```

// suma elemenata na GD

```
for ( s=i=0; i<n; i++ )  
    s+=mat[i][i];  
printf("Suma elemenata na GD: %d",s);
```

manipulacija sporednom dijagonalom:



// ispis elemenata na SD

```
for ( i=n-1; i>=0; i-- )  
    printf(" %d", mat[i][n-1-i]);
```

// suma elemenata na SD

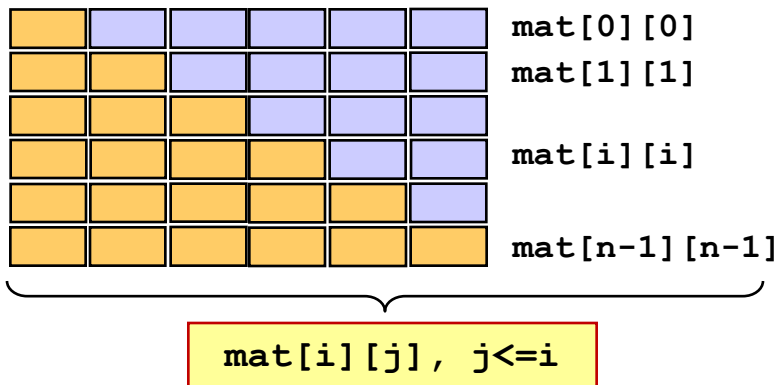
```
for ( s=i=0; i<n; i++ )  
    s+=mat[i][n-1-i];  
printf("Suma elemenata na SD: %d",s);
```

Dvodimenzionalni niz (matrica)

Primjeri primjene dvodimenzionalnih nizova

Primjer (Neke operacije nad matricama)

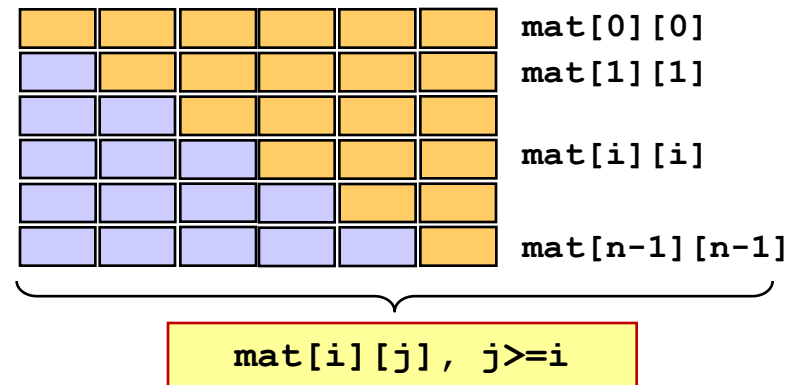
manipulacija donjom trougaonom matricom:



// ispis donje trougaone matrice

```
for ( i=0; i<n; i++ )
{
    for ( j=0; j<=i; j++ )
        printf(" %4d", mat[i][j]);
    printf("\n");
}
```

manipulacija gornjom trougaonom matricom:



// ispis gornje trougaone matrice

```
for ( i=0; i<n; i++ )
{
    for ( j=1; j<=i; j++ )
        printf(" %5c", ' ');
    for ( j=i; j<n; j++ )
        printf(" %4d", mat[i][j]);
    printf("\n");
}
```

Dvodimenzionalni niz (matrica)

Primjeri primjene dvodimenzionalnih nizova

Primjer (Neke operacije nad matricama)

Transponovanje matrice

polazna matrica

1	2	3
4	5	6
7	8	9



$\text{mat}[i][j] \leftrightarrow \text{mat}[j][i]$

transponovana matrica

1	4	7
2	5	8
3	6	9

Ispis transponovane matrice na osnovu originalne matrice:

```
printf("Transponovana:\n ");
for ( i=0; i<n; i++ )
{
    for ( j=0; j<n; j++ )
        printf(" %4d", mat[j][i]);
    printf("\n");
}
```

Transponovanje matrice:

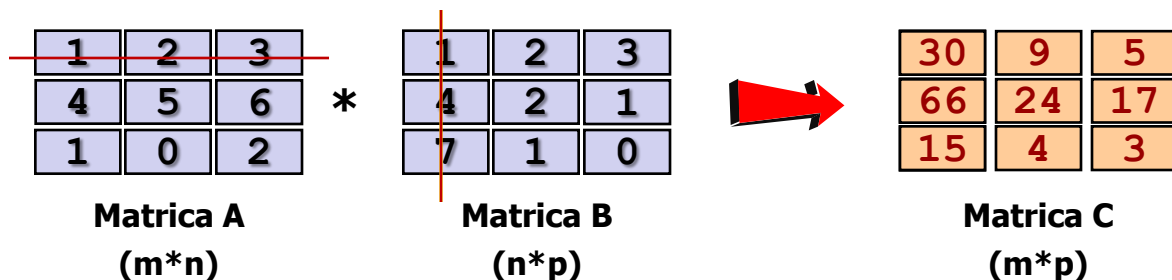
```
for ( i=0; i<n; i++ )
    for ( j=i+1; j<n; j++ )
    {
        pom=mat[i][j]);
        mat[i][j]=mat[j][i];
        mat[j][i]=pom;
    }
```

Dvodimenzionalni niz (matrica)

Primjeri primjene dvodimenzionalnih nizova

Primjer (Neke operacije nad matricama)

Množenje dvije matrice



Množenje matrica:

```
for ( i=0; i<m; i++ )      /* red matrice C */
  for ( j=0; j<p; j++ )    /* kolona matrice C */
    for ( k=0; k<n; k++ )
      c[i][j]+=a[i][k]*b[k][j];
```

Višedimenzionalni nizovi

Trodimenzionalni niz

- Trodimenzionalni niz je niz matrica (niz čiji su elementi matrice)

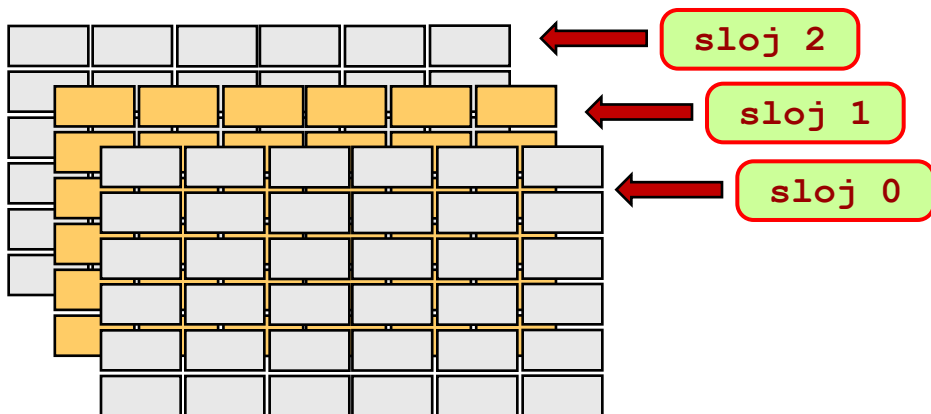
Deklaracija

```
tip ime[d1][d2][d3];
```

Primjer

```
int kocka[3][6][6];
```

Logička reprezentacija



Pristup elementu

```
kocka[sloj][red][kolona];
```

Višedimenzionalni niz

- C ne nameće ograničenja u pogledu broja dimenzija niza
- Kod viših dimenzija nemamo geometrijsku reprezentaciju, nego programer uvodi dimenzije u cilju lakše organizacije veće količine podataka

Neki primjeri i interpretacija dimenzija:

```
// ocjene jednog studenta
```

```
int ocjeneStudenta[40];
```

```
// ocjene svih studenata (max. 100)
```

```
// na jednoj godini studija
```

```
int OcjeneNaGodini[100][40];
```

```
// ocjene svih studenata (max. 100)
```

```
// na 4-godisnjem studijskom programu
```

```
int OcjeneNaSP[4][100][40];
```

```
// ocjene svih studenata na jednom
```

```
// fakultetu sa max. pet 4-godisnjih
```

```
// studijskih programa
```

```
int OcjeneNaFaksu[5][4][100][40];
```


Strukture

Struktura (engl. *structure*)

- Struktura je skup heterogenih podataka logički međusobno povezanih
- Pogodna za grupisanje atributa nekog entiteta (stvarnog ili nestvarnog)
- Alternativni nazivi: **slog, zapis, rekord** (engl. *record*)

Atributi?

ime

prezime

visina

Definicija strukture

ključna riječ

ime strukture

struct

ime {

tip1 element1;

tip2 element2;

...

tipN elementN;

};

Deklaracija
elementa (polja)

Primjer definicije strukture:

```
struct osoba {  
    char prezime[15];  
    char ime[15];  
    int visina;  
};
```



Strukture

Definicija promjenljivih tipa struktura

```
struct ime {  
    tip1 element1;  
    tip2 element2;  
    ...  
    tipN elementN;  
} lista_promjenljivih;
```

ili

```
struct ime {  
    tip1 element1;  
    tip2 element2;  
    ...  
    tipN elementN;  
};  
struct ime lista_promjenljivih;
```

Npr.

```
struct osoba {  
    char prezime[15];  
    char ime[15];  
    int visina;  
} o1, ekipa[10];  
  
struct razlomak {  
    int brojilac;  
    int imenilac;  
} r;  
  
struct datum { int dd, mm, gg; } d;
```

Npr.

```
struct osoba {  
    char prezime[15];  
    char ime[15];  
    int visina;  
};  
struct osoba o1, ekipa[10];  
  
struct razlomak { int brojilac, imenilac; };  
struct razlomak r;  
  
struct datum { int dd, mm, gg; };  
struct datum d;
```



Strukture

Definicija i inicijalizacija promjenljivih tipa struktura

Npr.

```
struct razlomak { int brojilac, imenilac; };  
struct razlomak r={1,2};           // definicija i inicijalizacija razlomka 1/2  
struct razlomak niz[]={1,2},{3,4}}; // definicija i inicijalizacija niza razlomaka 1/2, 3/4  
  
struct datum { int dd, mm, gg; };  
struct datum d={10,12,2021}; // definicija i inicijalizacija datuma 10.12.2021.  
  
struct vrijeme{ unsigned char hh, mm, ss; };  
struct vrijeme t1={12};           // definicija i inicijalizacija vremena 12:00:00  
struct vrijeme t2={};             // definicija i inicijalizacija vremena 00:00:00  
struct vrijeme t3={.ss=10};       // definicija i inicijalizacija vremena 00:00:10  
  
struct mjesto { char naziv[15]; int posta; };  
struct mjesto bl={"BL", 78000};  
struct mjesto bg={.naziv="BG", .posta=11000};  
struct mjesto gradovi[]={{"BL", 78000}, {.naziv="BG", .posta=11000}};
```



Strukture

Pristup elementima (poljima) strukture

- Pristup pomoću operatora

`"."` = ELEMENT STRUKTURE

- Izraz oblika

`struktura.polje`

omogućava pristup elementu "polje"

Npr.

```
struct razlomak { int brojilac, imenilac; };  
struct razlomak r={1,2};  
printf("%d/%d\n", r.brojilac, r.imenilac);
```

- Element strukture je */-value*

Npr.

```
struct razlomak { int brojilac, imenilac; };  
struct razlomak r1={1,2}, r2={3,4};  
struct razlomak reciprocno, zbir;  
reciprocno.brojilac = r.imenilac;  
reciprocno.imenilac = r.brojilac;  
zbir.imenilac = r1.imenilac * r2.imenilac;  
zbir.brojilac = r1.brojilac*r2.imenilac +  
               r2.brojilac*r1.imenilac;
```



Strukture

Pristup elementima (poljima) strukture

Primjer:

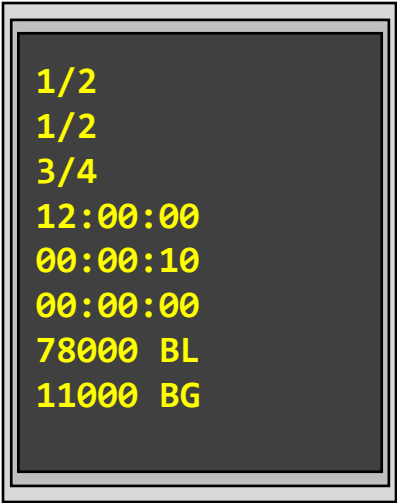
```
#include <stdio.h>
int main()
{
    struct razlomak { int brojilac, imenilac; };
    struct razlomak r={1,2};
    printf("%d/%d\n", r.brojilac, r.imenilac);

    struct razlomak niz[]={ {1,2}, {3,4} };
    for (int i=0; i<2; i++)
        printf("%d/%d\n", niz[i].brojilac, niz[i].imenilac);

    struct vrijeme{ unsigned char hh, mm, ss; };
    struct vrijeme t[3]={ {12}, { .ss=10 } };
    for (int i=0; i<3; i++)
        printf("%02d:%02d:%02d\n", t[i].hh, t[i].mm, t[i].ss);

    struct mjesto { char naziv[15]; int posta; };
    struct mjesto gradovi[]={ {"BL", 78000}, { .naziv="BG", .posta=11000 } };
    for (int i=0; i<2; i++)
        printf("%d %s\n", gradovi[i].posta, gradovi[i].naziv);

    return 0;
}
```



```
1/2
1/2
3/4
12:00:00
00:00:10
00:00:00
78000 BL
11000 BG
```

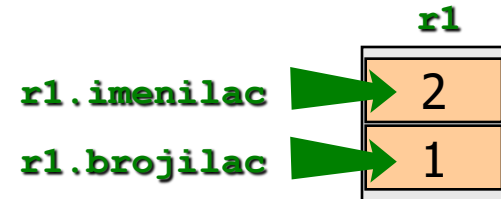
Strukture

Pristup elementima (poljima) strukture

- Struktura je *l-value*

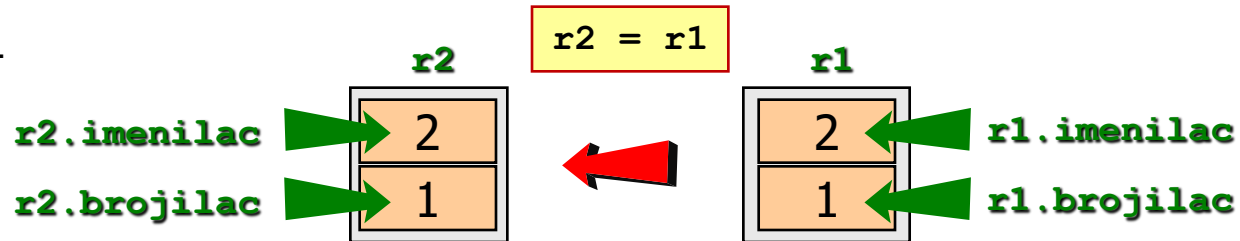
Npr.

```
struct razlomak { int brojilac, imenilac; };  
struct razlomak r1={1,2}, r2;
```



```
// dodjela strukture strukturi
```

```
r2 = r1;
```



```
// dodjela strukture strukturi
```

```
// je ekvivalentna dodjeli clan po clan
```

```
r2.brojilac = r1.brojilac;
```

```
r2.imenilac = r1.imenilac;
```

```
r2.brojilac = r1.brojilac  
r2.imenilac = r1.imenilac
```



Strukture

Struktura kao element strukture

- Struktura može da sadrži drugu strukturu

Npr.

```
struct datum { int dd, mm, gg; };
```

```
struct osoba {  
    char ime[10];  
    struct datum datRod;  
};
```

```
struct osoba o1={ "Marko", {1,1,2000} };
```

```
printf("%s %d.%d.%d\n", o1.ime, o1.datRod.dd, o1.datRod.mm, o1.datRod.gg);
```

```
struct osoba o2;
```

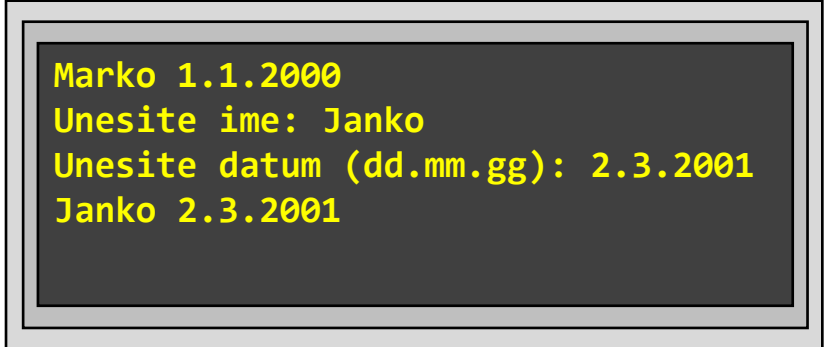
```
printf("Unesite ime: ");
```

```
scanf("%s", o2.ime);
```

```
printf("Unesite datum (dd.mm.gg): ");
```

```
scanf("%d.%d.%d", &o2.datRod.dd, &o2.datRod.mm, &o2.datRod.gg);
```

```
printf("%s %d.%d.%d", o2.ime, o2.datRod.dd, o2.datRod.mm, o2.datRod.gg);
```



```
Marko 1.1.2000  
Unesite ime: Janko  
Unesite datum (dd.mm.gg): 2.3.2001  
Janko 2.3.2001
```

Strukture

Primjer korištenja strukture

Program koji učitava koordinate vrhova poligona, pa ispisuje njegov obim.

```
#include <stdio.h>
#include <math.h>
#define KV(x) (x)*(x)

int main()
{
    struct tacka { float x,y; } pol[100];
    float ob=0;
    int i,n;

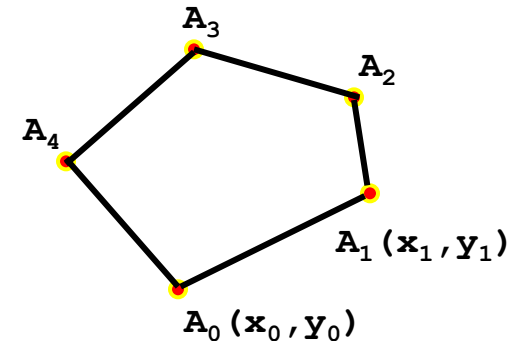
    do
    { printf("Broj vrhova: "); scanf("%d", &n); }
    while ((n<3) || (n>100));

    for (i=1; i<=n; i++)
    { printf("%d. vrh:\n", i);
      printf("  x="); scanf("%f", &pol[i-1].x);
      printf("  y="); scanf("%f", &pol[i-1].y); }

    for (i=1; i<n; i++)
        ob += sqrt( KV(pol[i].x-pol[i-1].x)+ KV(pol[i].y-pol[i-1].y));
    ob += sqrt( KV(pol[0].x-pol[n-1].x)+ KV(pol[0].y-pol[n-1].y));

    printf("Obim: %5.2f", ob);

    return 0;
}
```



Broj vrhova: 3

1. vrh:

x=1

y=1

2. vrh:

x=4

y=1

3. vrh:

x=1

y=5

Obim: 12.00

Strukture

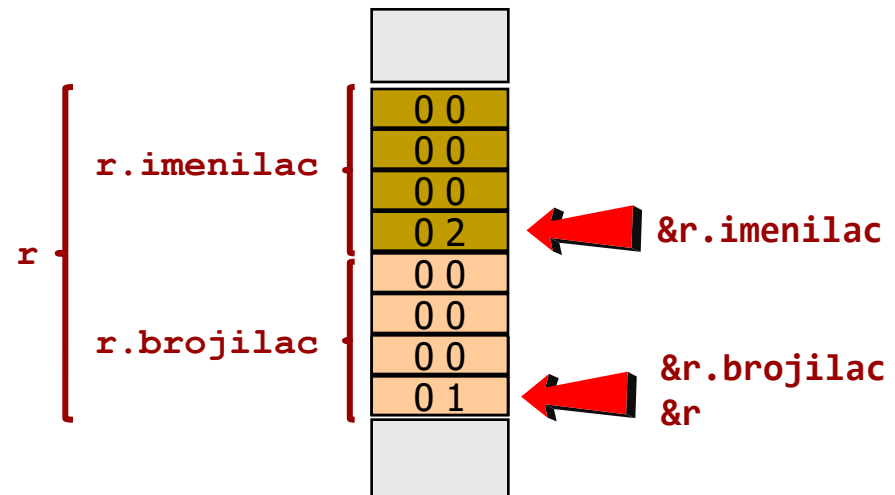
Fizička reprezentacija strukture (razmještaj u memoriji)

- Elementi strukture smještaju se redom u memoriju

Npr.

```
struct razlomak { int brojilac, imenilac; };  
struct razlomak r={1,2};  
printf("sizeof(struct razlomak): %d\n", sizeof(struct razlomak));  
printf("sizeof(r): %d\n", sizeof(r));  
printf("&r: %p\n", &r);  
printf("&r.brojilac: %p\n", &r.brojilac);  
printf("&r.imenilac: %p\n", &r.imenilac);
```

```
sizeof(struct razlomak): 8  
sizeof(r): 8  
&r: 0028ff18  
&r.brojilac: 0028ff18  
&r.imenilac: 0028ff1c
```



Strukture

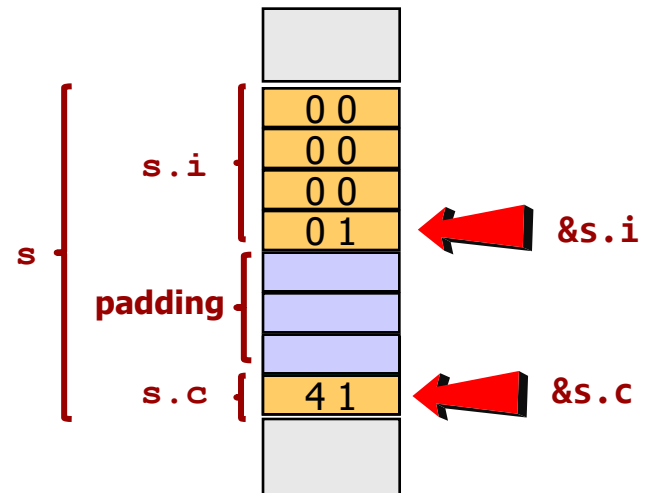
Fizička reprezentacija strukture (razmještaj u memoriji)

- Razmještaj elemenata strukture u memoriji ne mora biti na sukcesivnim lokacijama
- Čitanje podataka iz memorije ne vrši se bajt po bajt, nego se odjednom čita onoliko bajtova koliko je široka magistrala podataka.
- Budući da je pristup memoriji skupa operacija, često se razmještaj elemenata strukture vrši tako da se minimizuje broj pristupa memoriji (npr. podatak tipa int se smješta u memoriji tako da može odjednom da upiše/pročita) – u tom slučaju imamo prazne bajtove između pojedinih elemenata strukture (engl. *padding*)

Npr.

```
struct str { char c; int i; };  
struct str s={ 'A', 1 };  
printf("sizeof(s): %d\n", sizeof(s));  
printf("&s: %p\n", &s);  
printf("&s.c: %p\n", &s.c);  
printf("&s.i: %p\n", &s.i);
```

```
sizeof(s): 8  
&s: 0028ff18  
&s.c: 0028ff18  
&s.i: 0028ff1c
```

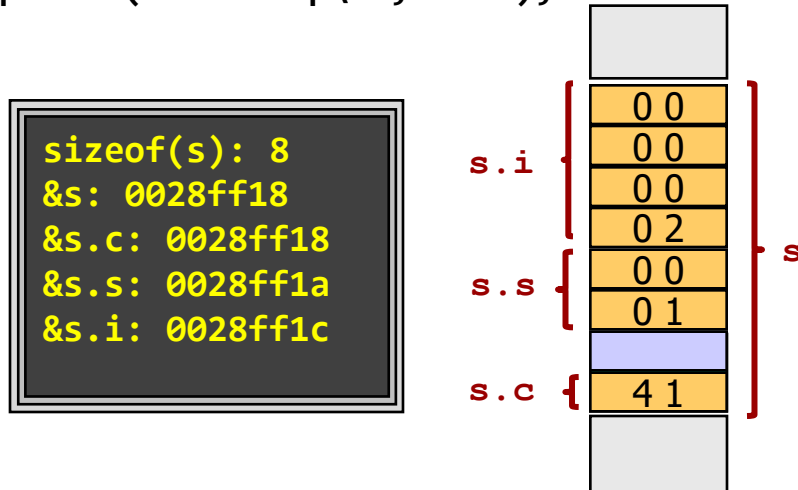


Strukture

Fizička reprezentacija strukture (razmještaj u memoriji)

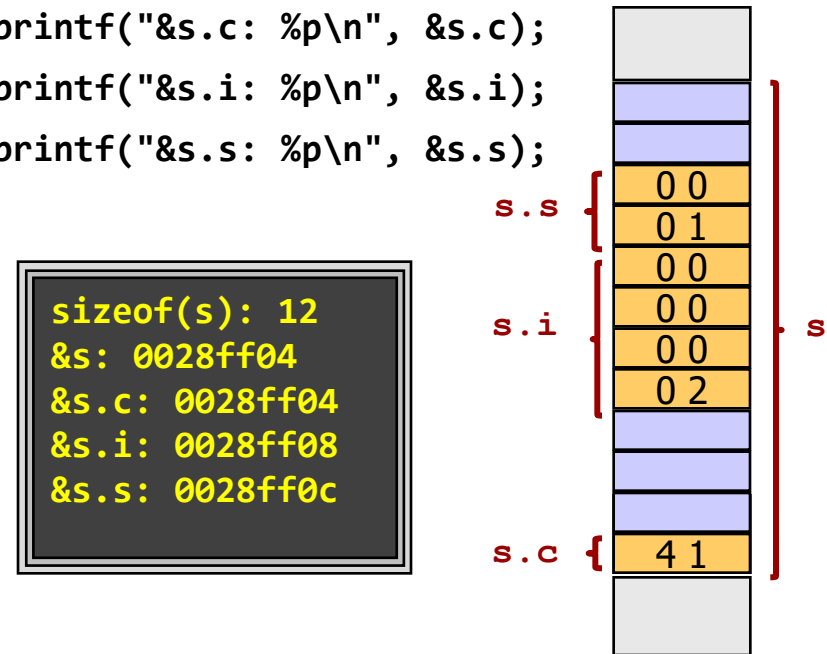
Npr.

```
struct str { char c; short s; int i};  
struct str s={ 'A', 1, 2 };  
printf("sizeof(s): %d\n", sizeof(s));  
printf("&s: %p\n", &s);  
printf("&s.c: %p\n", &s.c);  
printf("&s.s: %p\n", &s.s);  
printf("&s.i: %p\n", &s.i);
```



Npr.

```
struct str { char c; int i; short s; };  
struct str s={ 'A', 2, 1 };  
printf("sizeof(s): %d\n", sizeof(s));  
printf("&s: %p\n", &s);  
printf("&s.c: %p\n", &s.c);  
printf("&s.i: %p\n", &s.i);  
printf("&s.s: %p\n", &s.s);
```



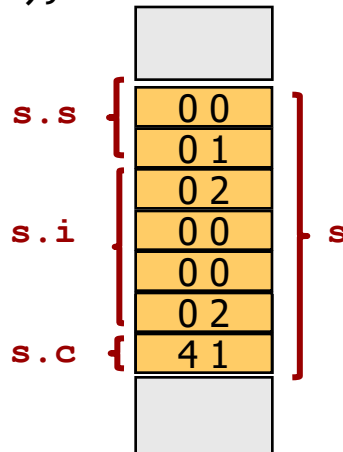
Strukture

Fizička reprezentacija strukture (razmještaj u memoriji)

Npr.

```
#include <stdio.h>
#pragma pack(1)
int main()
{
    struct str { char c; int i; short s; };
    struct str s={ 'A', 2, 1 };
    printf("sizeof(s): %d\n", sizeof(s));
    printf("&s: %p\n", &s);
    printf("&s.c: %p\n", &s.c);
    printf("&s.i: %p\n", &s.i);
    printf("&s.s: %p\n", &s.s);
    return 0;
}
```

```
sizeof(s): 7
&s: 0028ff09
&s.c: 0028ff09
&s.i: 0028ff0a
&s.s: 0028ff0e
```



Pretprocesorska direktiva

#pragma pack(1)

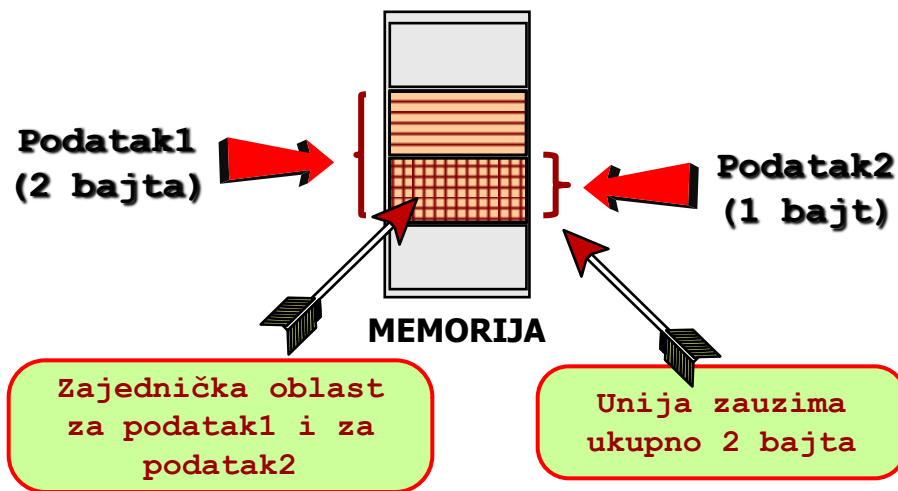
nalaže pakovanje podataka (podaci se smještaju bez *paddinga*)

Pakovanjem podataka smanjuje se potreban memorijski prostor, ali se smanjuje efikasnost pristupa (povećava se broj pristupa pa se produžava vrijeme pristupa)

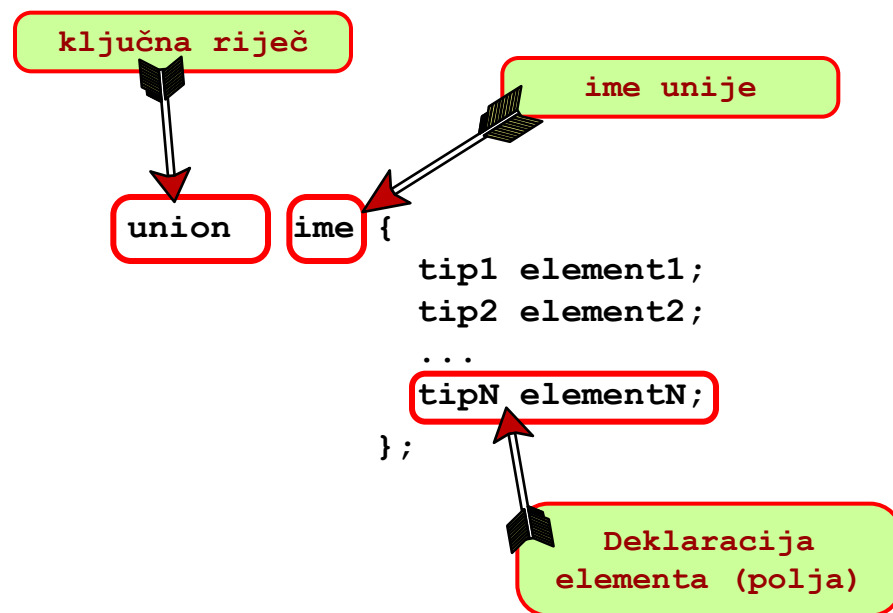
Unije

Union (engl. *union*)

- Unija je skup podataka različitih tipova smještenih u istom memorijskom prostoru.
- Podaci u uniji nisu međusobno nezavisni, jer koriste isti memorijski prostor (u memoriju se smještaju od iste početne lokacije).
- Promjena jednog podatka može dovesti i do promjene drugog podatka u uniji.
- Unija zauzima u memoriji onoliko bajtova koliko je potrebno za memorisanje najvećeg podatka u uniji.



Definicija unije



Primjer definicije unije:

```
union unija {  
    float prosjek;  
    int kabinet;  
};
```



Unije

Definicija promjenljivih tipa unija

```
union ime {  
    tip1 element1;  
    tip2 element2;  
    ...  
    tipN elementN;  
} lista_promjenljivih;
```

ili

```
union ime {  
    tip1 element1;  
    tip2 element2;  
    ...  
    tipN elementN;  
};  
union ime lista_promjenljivih;
```

Npr.

```
union unija {  
    unsigned char c[8];  
    double d;  
} u, niz[10];
```

Npr.

```
union unija {  
    unsigned char c[8];  
    double d;  
};  
union unija u, niz[10];
```

Pristup elementima unije

- **Pristup pomoću operatora**

`"."` = ELEMENT UNIJE

- **Izraz oblika**

`unija.polje`

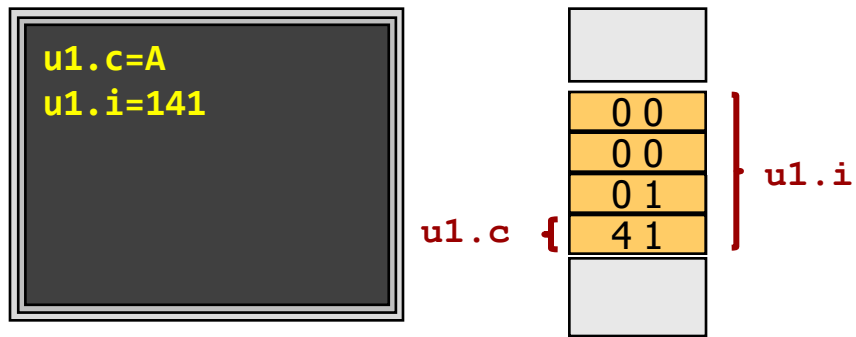
omogućava pristup elementu "polje"

Unije

Definicija i inicijalizacija promjenljivih tipa unija

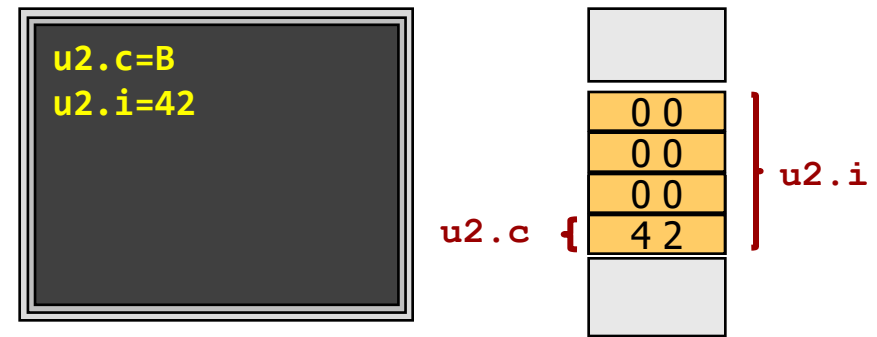
Npr.

```
union unija { char c; int i; };  
  
// inicijalizacija konkretnog polja  
union unija u1={.i=0x141};  
  
printf("u1.c=%c\n", u1.c);  
printf("u1.i=%x\n", u1.i);
```



Npr.

```
union unija { char c; int i; };  
  
// ako se ne specifikuje konkretno polje,  
// inicijalizuje se prvo polje, ovdje polje c  
union unija u2={0x342};  
  
printf("u2.c=%c\n", u2.c);  
printf("u2.i=%x\n", u2.i);
```



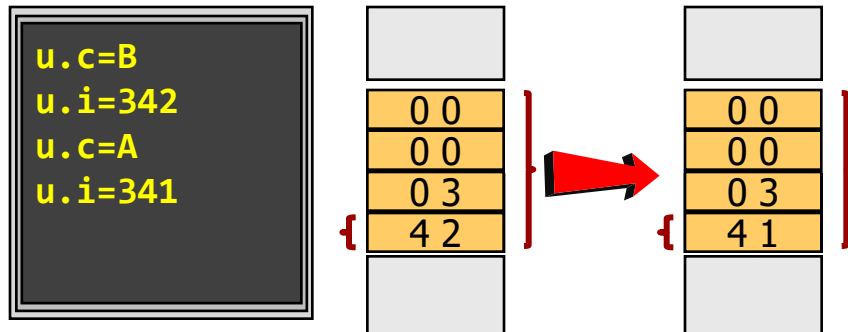
Unije

Dodjela vrijednosti

- Element unije je *l-value*

Npr.

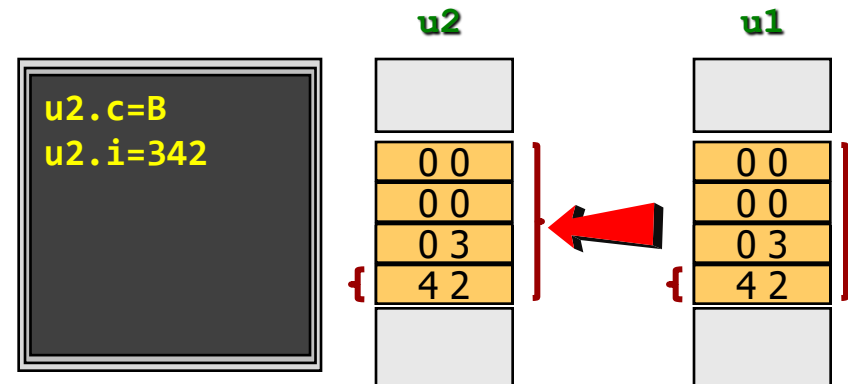
```
union unija { char c; int i; };  
union unija u = { .i=0x342};  
printf("u.c=%c\n", u.c);  
printf("u.i=%x\n", u.i);  
u.c = 'A';  
printf("u.c=%c\n", u.c);  
printf("u.i=%x\n", u.i);
```



- Unija je *l-value*

Npr.

```
union unija { char c; int i; };  
union unija u1={.i=0x342}, u2;  
u2=u1;  
printf("u2.c=%c\n", u2.c);  
printf("u2.i=%x\n", u2.i);
```



Neke primjene unije

- Unija omogućava da jednu memorijsku oblast interpretiramo na različite načine, zavisno od potrebe.
- Pretpostavimo da u programu treba da predstavimo podatke o različitim kategorijama članova, npr. članovi biblioteke mogu biti profesori i studenti. To bismo mogli predstaviti sa dvije različite strukture – jedna za profesore, a druga za studente.
- Isto možemo uraditi i pomoću jedne strukture, koja sadrži uniju u kojoj se nalaze atributi koji su specifični za svaku kategoriju. To nam dalje omogućava da različite kategorije objekata reprezentujemo jedinstveno, npr. i profesore i studente možemo da smjestimo u isti niz.
- Za razlikovanje podataka u strukturi, dovoljno je uvesti dodatni atribut koji omogućava razlikovanje kategorija, npr. kategorija='p' za profesore, a kategorija='s' za studente

```
struct profesor { char ime[15]; int bk; };
struct student { char ime[15]; float pr; };

struct clan {
    char ime[15];
    char kategorija;
    union unija {
        int bk;
        float pr; } u;
};

struct clan p={"Marko", 'p', {.bk=1307}};
struct clan s={"Ana", 's', {.pr=9.75}};
struct clan clanovi[100];

printf("%s %d\n", p.ime, p.u.bk);
printf("%s %5.2f\n", s.ime, s.u.pr);

for (int i=0; i<n; i++)
    if (clanovi[i].kategorija)=='p'
        // profesor
    else
        // student
```

Enumeracije (nabrajanja)

enum

- Ponekad u programu trebamo veći broj konstanti (npr. za reprezentaciju različitih stanja nekog uređaja – OFF, ON, BUSY, READY, ...)
- **Enumeracija (nabrajanje / nabrojivi tip)** predstavlja tip sa manjim brojem cjelobrojnih vrijednosti kojima su pridružena specifična imena – programer ne mora da pamti konkretne vrijednosti, nego koristi simbolička imena

```
enum Rim { I=1, II, III, IV };  
printf("%d %d %d %d", I, II, III, IV);
```



1 2 3 4

```
enum ime { V0, V1, V2 };
```

V0 ima vrijednost 0, V1 ima vrijednost 1, V2 ima vrijednost 2

```
enum uredjaj { OFF, ON };
```

OFF ima vrijednost 0, ON ima vrijednost 1

```
enum uredjaj { ON=1, OFF=0 };
```

OFF ima vrijednost 0, ON ima vrijednost 1

```
enum Rim { I=1, II, III, IV };
```

I ima vrijednost 1, II ima vrijednost 2 (za jedan više od I), itd.

```
enum K { A=1, B=5, C, D=B+C };
```

A:1, B:5, C:6, D:11



Definisanje korisničkih tipova podataka

typedef

- Jezik C omogućava definisanje korisničkih tipova
- **Naredba `typedef` omogućava preimenovanje nekog tipa i uvođenje novog identifikatora tipa, koji može da se koristi ravnopravno sa ugrađenim tipovima podataka**

Opšti oblik:

```
typedef staritip novitip;
```

Primjeri:

```
typedef int INTEGER;  
INTEGER i,j;  
-----
```

```
typedef float NIZ[100];  
NIZ niz1, niz2;  
-----
```

```
typedef struct s {  
    int i;  
    char c;  
} TIP;
```

```
TIP niz[10];
```

Tip `int` preimenovan je u tip `INTEGER`.

Ovo je pogodno za olakšavanje prenosivosti koda na različite platforme, jer na različitim platformama `int` može biti različite dužine. Ako je negdje 16b, a treba nam 32b, dovoljno je samo u `typedef` naredbi ubaciti odgovarajući 32-bitni tip.

Preimenovanje se često koristi kod struktura.

Umjesto da se svuda koristi `struct s` kao identifikator tipa, dovoljno je `struct s` preimenovati u `TIP` i dalje koristiti `TIP`.