

# Uvod u programiranje

- predavanja -

listopad 2019.

---

## 4. Agregatni tipovi podataka

# Polja - motivacija

- Programski zadatak

- s tipkovnice učitati 10 cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine
- primjer izvršavanja programa

```
5 15 1 2 3 -4 25 6 8 7↵  
sredina = 6.800000↵  
15↵  
25↵  
8↵  
7↵
```

- očito, svih 10 vrijednosti će trebati pohraniti u varijable, izračunati prosjek, a zatim ispisati vrijednosti varijabli koje zadovoljavaju uvjet

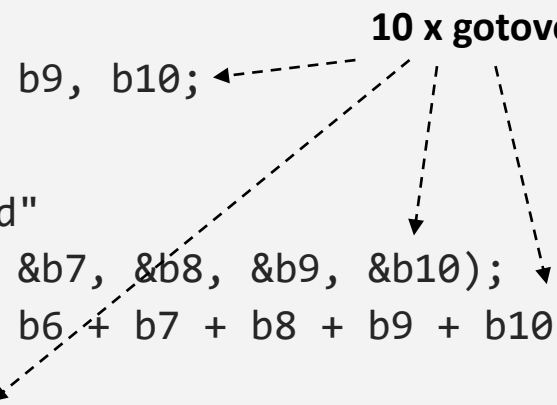
# Rješenje (loše)

```
#include <stdio.h>

int main(void) {
    int b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
    float sredina;

    scanf("%d %d %d %d %d %d %d %d %d %d"
        , &b1, &b2, &b3, &b4, &b5, &b6, &b7, &b8, &b9, &b10);
    sredina = (b1 + b2 + b3 + b4 + b5 + b6 + b7 + b8 + b9 + b10) / 10.f;
    printf("sredina = %f\n", sredina);
    if (b1 > sredina) printf("%d\n", b1);
    if (b2 > sredina) printf("%d\n", b2);
    ...
    if (b9 > sredina) printf("%d\n", b9);
    if (b10 > sredina) printf("%d\n", b10);
    return 0;
}
```

**10 x gotovo isti posao**



# Matematički niz

- Niz brojeva koji imaju zajedničko ime i čiji se članovi identificiraju indeksom:  $\text{broj}_0, \text{broj}_1, \text{broj}_2, \dots$
- Kada bi se niz mogao koristiti u programu, prethodni zadatak bi se mogao riješiti na sljedeći način:

```
suma := 0
za i = 1 do 10
    | učitaj( $b_i$ )
    | suma := suma +  $b_i$ 
sredina := suma / 10
za i = 1 do 10
    | ako je  $b_i > \text{sredina}$ 
    | | ispiši( $b_i$ )
```

# Matematički niz

- Bez teškoća bi se mogli riješiti i mnogo veći problemi, koje bi bilo iznimno teško riješiti bez korištenja niza: npr. učitavanje  $n$  brojeva i ispis onih brojeva koji su veći od njihove aritmetičke sredine

```
učitaj(n)
suma := 0
za i = 1 do n
    | učitaj( $b_i$ )
    | suma := suma +  $b_i$ 
sredina := suma / n
za i = 1 do n
    | ako je  $b_i >$  sredina
    | | ispiši( $b_i$ )
```

# Agregatni tipovi podataka

Polja  
Strukture

# Agregatni tipovi podataka

- do sada su korišteni isključivo jednostavni tipovi podataka: `int` i `float`
  - u varijablu jednostavnog tipa moguće je pohraniti samo jedan elementarni podatak. Jedna varijabla  $\leftrightarrow$  jedna vrijednost
    - **skalarni** tip podatka, **skalarna** varijabla, **skalarni** podatak
- agregatni podatak (*data aggregate*) ili složeni podatak obuhvaća više skalarnih podataka i/ili više agregatnih podataka, objedinjenih pod istim imenom
  - polje (*array*)
  - struktura ili zapis (*structure, record*)
- prednosti korištenja agregatnih podataka
  - jednostavniji postupci nad skupinama podataka
  - naglašava se logička povezanost podataka

# Polja

Jednodimenzijska polja



# Polje

- **Jednostavni tipovi podataka**

- jedna varijabla  $\leftrightarrow$  jedna vrijednost
- do vrijednosti varijable pristupa se navođenjem imena varijable
- varijabla jednostavnog tipa jest *modifiable lvalue*

```
float x;  
x = 3.14f;  
printf("%f", x);
```

x    3.14

- **Polje** je složeni tip podatka koji obuhvaća više članova istog tipa

- jedna varijabla  $\leftrightarrow$  više vrijednosti
- pojedinim vrijednostima (članovima) pristupa se pomoću indeksa
- ime varijable je *non-modifiable lvalue*; ime varijable uz navedeni indeks elementa jest *modifiable lvalue*

```
float y[4];  
y[1] = 2.71f;  
printf("%f", y[1]);
```

y    

?	2.71	?	?
---	------	---	---

  
      y[0]    y[1]    y[2]    y[3]

# Definicija varijable tipa polje

- Pri definiciji varijable potrebno je odrediti
  - ime varijable: određuje se na isti način kao ime varijable za jednostavne tipove podataka
  - tip podatka za članove polja (int, float, ...)
  - veličinu polja izraženu u broju članova polja
    - polje poznatih konstantnih dimenzija
      - dimenzija polja poznata je u trenutku prevođenja
      - u uglatim zagradama navodi se **konstantni cjelobrojni izraz**
    - polje varijabilne veličine (*variable-length array, VLA*)
      - veličina polja se utvrđuje u trenutku izvršavanja naredbe za definiciju polja (ali nakon toga se ne mijenja)
      - u uglatim zagradama navodi se **cjelobrojni izraz** (može sadržavati varijable) čiji rezultat **mora** biti veći od nule

# Primjer

```
#define VELICINA_VEKTORA 50
#define BROJ_CLANOVA (5 * 10)
...
// primjeri definicije polja poznatih konstantnih dimenzija
//float vektor[50];
float vektor[VELICINA_VEKTORA]; // bolje nego 50

//int velicine[5 * 10];
int velicine[BROJ_CLANOVA]; // bolje nego 5 * 10

// primjeri definicije polja varijabilnih dimenzija (VLA)
int n;
scanf("%d", &n); // osigurati da n bude > 0 !
float temperature[n];
int tlakovi[2 * n];
```

# Pristupanje članovima polja

- Članovima (elementima) polja pristupa se korištenjem indeksa
  - indeks može biti cjelobrojni izraz (konstante, varijable, operatori, funkcije) čiji rezultat mora biti nenegativni cijeli broj iz intervala  
 $[0, \text{brojElemenataPolja} - 1]$  (znači: indeks prvog člana je 0)
  - C prevodilac može utvrditi jedino pogrešnu upotrebu tipa podatka indeksa (npr. float umjesto int)
  - poštovanje pravila o dopuštenim granicama indeksa odgovornost je isključivo programera
    - rezultat korištenja neispravnih indeksa za vrijeme izvršavanja programa je nedefiniran: dobije se pogrešna vrijednost (logička pogreška), a pridruživanje vrijednosti uz korištenje neispravnog indeksa može, osim logičke pogreške, dovesti i do pogreške izvršavanja i prekida programa

# Primjer

```
int i = 2, m, brojevi[10];
```

```
float x = 5.0f;
```

```
brojevi[0] = 0;
```

```
brojevi[1] = 10;
```

```
...
```

```
brojevi[i * 4] = 80;
```

```
brojevi[i * 4 + 1] = 90;
```

```
brojevi[x] = 1;    prevodilac dojavljuje pogrešku
```

```
...
```

```
m = brojevi[10];   rezultat je nedefiniran (garbage value). Logička pogreška.
```

```
brojevi[-1] = 15;  15 se upisuje na pogrešno mjesto u memoriji, što može  
izazvati logičku pogrešku ili pogrešku tijekom izvršavanja
```

# Primjer

- s tipkovnice učitati 10 cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine

```
#include <stdio.h>
#define DIMENZIJA 10

int main(void) {
    int brojevi[DIMENZIJA], suma = 0, i;
    float sredina;

    for (i = 0; i < DIMENZIJA; i = i + 1) {
        scanf("%d", &brojevi[i]);
        suma = suma + brojevi[i];
    }
    sredina = 1.f * suma / DIMENZIJA;
    printf("sredina = %f\n", sredina);

    for (i = 0; i < DIMENZIJA; i = i + 1) {
        if (brojevi[i] > sredina) {
            printf("%d\n", brojevi[i]);
        }
    }
    return 0;
}
```

# Primjer (kada koristiti VLA)

- s tipkovnice učitati n, zatim n cijelih brojeva, ispisati njihovu aritmetičku sredinu i brojeve koji su veći od aritmetičke sredine

```
...
int n, suma = 0, i;
float sredina;

scanf("%d", &n);
int brojevi[n];
for (i = 0; i < n; i = i + 1) {
    scanf("%d", &brojevi[i]);
    suma = suma + brojevi[i];
}
sredina = 1.f * suma / n;
printf("sredina = %f\n", sredina);

for (i = 0; i < n; i = i + 1) {
    if (brojevi[i] > sredina) {
        printf("%d\n", brojevi[i]);
    }
}
...
```

# Definicija polja uz inicijalizaciju

- Članovi polja mogu se inicijalizirati u trenutku definicije polja
  - **nije primjenjivo za VLA polja!**
  - bez inicijalizacije, članovi polja sadrže nedefinirane vrijednosti

```
int polje[5];
```

polje

?	?	?	?	?
---	---	---	---	---

- početne vrijednosti se mogu redom navesti u tzv. inicijalizatoru

```
int polje[5] = {1, 3, 5, 7, 9};
```

polje

1	3	5	7	9
---	---	---	---	---

- ako se navede premalo vrijednosti, ostali članovi se postavljaju na 0

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

polje

1	3	5	0	0
---	---	---	---	---

- što se može iskoristiti za inicijalizaciju svih članova na nulu

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

polje

0	0	0	0	0
---	---	---	---	---



# Definicija polja uz inicijalizaciju

- designiranim inicijalizatorom se članovi polja mogu ciljano postaviti

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

**polje**

1	0	0	0	2
---	---	---	---	---

- automatsko određivanje veličine polja na temelju inicijalizatora

```
int polje[] = {1, 3, 5, 7, 9};
```

**polje**

1	3	5	7	9
---	---	---	---	---

- u inicijalizatoru se mora nalaziti barem jedna vrijednost

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

Prevodilac dojavljuje pogrešku

- u inicijalizatoru se ne smije napisati previše vrijednosti

```
int polje[5] = {1, 3, 5, 7, 9, 11};
```

Prevodilac dojavljuje pogrešku

# Inicijalizacija nije pridruživanje!

- varijabla tipa polje ne smije se nalaziti na lijevoj strani izraza pridruživanja jer polje nije *modifiable lvalue*
  - neispravan način kopiranja sadržaja polja izvor u polje cilj:

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

inicijalizacija polja izvor. O.K.

prevodilac dojavljuje pogrešku!

- ispravan način kopiranja sadržaja polja izvor u polje cilj:

```
int izvor[4] = {1, 2, 3, 4};  
int cilj[4];  
int i;  
for (i = 0; i < 4; i = i + 1) {  
    cilj[i] = izvor[i];  
}
```

inicijalizacija polja izvor. O.K.

# Primjer

- Programski zadatak
  - s tipkovnice učitati cijeli broj n koji predstavlja broj članova polja koji će biti učitani. Ponavljati učitavanje vrijednosti za n sve dok broj članova polja ne bude ispravan. Zatim učitati n realnih članova polja i ispisati ih u obrnutom poretaku od onog u kojem su učitani
  - primjer izvršavanja programa

```
Upisite broj clanova polja > 0↵
Upisite broj clanova polja > -2↵
Upisite broj clanova polja > 4↵
Upisite 1. clan > 9.1↵
Upisite 2. clan > 101.55↵
Upisite 3. clan > -476.3333↵
Upisite 4. clan > 5↵
5.0, -476.3, 101.6, 9.1↵
```

# Rješenje (1. dio)

```
#include <stdio.h>

int main(void) {
    int n;    // velicina polja
    int i;    // kontrolna varijabla petlje za učitavanje i ispis
    /* ponavljati upisivanje velicine polja dok ne bude ispravna */
    do {
        printf("Upisite broj clanova polja > ");
        scanf("%d", &n);
    } while (n < 1);

    /* definicija VLA polja */
    float polje[n];
}
```

## Rješenje (2. dio)

```
/* učitavanje članova polja */
for (i = 0; i < n; i = i + 1) {
    printf("Upisite %d. clan > ", i + 1);
    scanf("%f", &polje[i]);
}

/* ispisivanje članova polja u obrnutom poretaku */
for (i = n - 1; i >= 0; i = i - 1) {
    if (i < n - 1) {        // ispisati zarez prije svakog osim prvog
        printf(", ");
    }
    printf("%.1f", polje[i]);
}
printf("\n");
return 0;
}
```

# Primjer

- Programski zadatak
  - Učitavati cijele brojeve iz intervala  $[0, 9]$ . Učitavanje prekinuti kad se upiše broj izvan zadanog intervala. Zatim ispisati koliko je puta učitani svaki broj iz zadanog intervala, pri čemu treba ispisati samo one brojeve koji su učitani barem jednom.

```
Upisite broj iz intervala [0, 9] > 1↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 7↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 0↵
Upisite broj iz intervala [0, 9] > 5↵
Upisite broj iz intervala [0, 9] > 7↵
Upisite broj iz intervala [0, 9] > 10↵
```

```
↵
```

```
Broj 0 se pojavio 1 puta↵
Broj 1 se pojavio 1 puta↵
Broj 5 se pojavio 3 puta↵
Broj 7 se pojavio 2 puta↵
```

# Rješenje

- Utvrđivanje frekvencije pojavljivanja brojeva
  - za svaki broj potreban je jedan brojač
  - na početku se svi brojači postavljaju na nulu
  - kad god se učitava neki broj, odgovarajući brojač se poveća za 1

brojac	0	0	0	0	0	0	0	0	0
	brojac[0]	brojac[1]	brojac[2]	brojac[3]	brojac[4]	brojac[5]	brojac[6]	brojac[7]	brojac[8]

```
Upisite broj iz intervala [0, 9] > 1↵ → brojac[1] = brojac[1] + 1
Upisite broj iz intervala [0, 9] > 5↵ → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 7↵ → brojac[7] = brojac[7] + 1
Upisite broj iz intervala [0, 9] > 5↵ → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 0↵ → brojac[0] = brojac[0] + 1
Upisite broj iz intervala [0, 9] > 5↵ → brojac[5] = brojac[5] + 1
Upisite broj iz intervala [0, 9] > 7↵ → brojac[7] = brojac[7] + 1
```

<b>brojac</b>	1	1	0	0	0	3	0	2	0	0
	brojac[0]	brojac[1]	brojac[2]	brojac[3]	brojac[4]	brojac[5]	brojac[6]	brojac[7]	brojac[8]	brojac[9]

# Rješenje (1. dio)

```
#include <stdio.h>
#define D_GR 0          // donja granica intervala
#define G_GR 9          // gornja granica intervala

int main(void) {
    int broj;
    int brojac[G_GR - D_GR + 1] = { 0 };    // inicijalizacija na nulu
    /* učitavanje brojeva i inkrementiranje odgovarajućih brojaca */
    do {
        printf("Upisite broj u intervalu [%d, %d] > ", D_GR, G_GR);
        scanf("%d", &broj);
        if (broj >= D_GR && broj <= G_GR) {
            brojac[broj] = brojac[broj] + 1;
        }
    } while (broj >= D_GR && broj <= G_GR);
}
```



## Rješenje (2. dio)

```
printf("\n");  
/* ispis sadržaja onih brojaca koji su veci od nule */  
int i;  
for (i = D_GR; i <= G_GR; i = i + 1) {  
    if (brojac[i] > 0) {  
        printf("Broj %d se pojavio %d puta\n", i, brojac[i]);  
    }  
}  
return 0;  
}
```

# Primjer

- Programski zadatak (varijanta prethodnog zadatka - promijenjene su samo granice intervala)
  - Učitavati cijele brojeve iz intervala [1000005, 1000014]. Učitavanje prekinuti kad se upiše broj izvan zadanog intervala. Zatim ispisati koliko je puta učitani svaki broj iz zadanog intervala, pri čemu treba ispisati samo one brojeve koji su učitani barem jednom.

```
Upisite broj iz intervala [1000005, 1000014] > 1000008↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000008↵
Upisite broj iz intervala [1000005, 1000014] > 1000012↵
Upisite broj iz intervala [1000005, 1000014] > 1000014↵
Upisite broj iz intervala [1000005, 1000014] > 1000000↵
↵
Broj 1000008 se pojavio 2 puta↵
Broj 1000012 se pojavio 3 puta↵
Broj 1000014 se pojavio 1 puta↵
```

# Rješenje (1. dio)

```
#include <stdio.h>
#define D_GR 1000005           // donja granica intervala
#define G_GR 1000014           // gornja granica intervala

int main(void) {
    int broj;
    int brojac[G_GR - D_GR + 1] = { 0 };    // inicijalizacija na nulu
    /* učitavanje brojeva i inkrementiranje odgovarajućih brojaca */
    do {
        printf("Upisite broj u intervalu [%d, %d] > ", D_GR, G_GR);
        scanf("%d", &broj);
        if (broj >= D_GR && broj <= G_GR) {
            brojac[broj - D_GR] = brojac[broj - D_GR] + 1;
        }
    } while (broj >= D_GR && broj <= G_GR);
}
```

## Rješenje (2. dio)

```
printf("\n");  
/* ispis sadržaja onih brojaca koji su veci od nule */  
int i;  
for (i = D_GR; i <= G_GR; i = i + 1) {  
    if (brojac[i - D_GR] > 0) {  
        printf("Broj %d se pojavio %d puta\n", i, brojac[i - D_GR]);  
    }  
}  
return 0;  
}
```

# Primjer

- Programski zadatak
  - Učitati veličinu polja n (ne treba kontrolirati ispravnost) i učitati n članova cjelobrojnog polja. Ispisati poziciju (indeks) i vrijednost najmanjeg člana polja. Ako više članova polja ima istu najmanju vrijednost, ispisati poziciju prvog člana s najmanjom vrijednošću.

```
Upisite velicinu polja > 5↵
Upisite clanove polja > 7 5 4 6 4↵
↵
Vrijednost 4 na poziciji 2↵
```

# Rješenje

```
#include <stdio.h>

int main(void) {
    int n, i;    // velicina polja, kontrolna varijabla petlje
    printf("Upisite velicinu polja > ");
    scanf("%d", &n);

    int polje[n];

    printf("Upisite clanove polja > ");
    for (i = 0; i < n; i = i + 1) {
        scanf("%d", &polje[i]);
    }

    int min_ind = 0;    // pretpostavka: indeks minimalnog clana
    for (i = 1; i < n; i = i + 1) {
        if (polje[i] < polje[min_ind]) min_ind = i;    // promijeni pretpostavku
    }
    printf("\nVrijednost %d na poziciji %d\n", polje[min_ind], min_ind);
    return 0;
}
```

# Primjer

- Programski zadatak
  - Učitati veličinu polja n (ne treba kontrolirati ispravnost) i učitati n članova cjelobrojnog polja. Poredati (sortirati) članove polja od manjih prema većim. Ispisati sadržaj sortiranog polja.

```
Upisite velicinu polja > 5↵
Upisite clanove polja > 9 4 7 2 5↵
↵
2 4 5 7 9↵
```

# Rješenje

- Sortiranje članova polja
  - Ovdje će se koristiti jedan od najjednostavnijih i najmanje efikasnih algoritama za sortiranje: *selection sort*
    - za svaki  $i$ ,  $0 \leq i \leq n-2$ 
      - pronadi indeks  $\text{ind\_min}$  najmanjeg člana  $\text{polje}[j]$ ,  $i < j \leq n-1$
      - ako je  $\text{polje}[\text{ind\_min}] < \text{polje}[i]$ , zamijeni  $\text{polje}[i]$  i  $\text{polje}[\text{ind\_min}]$

i = 0	polje	<table><tr><td>9</td><td>4</td><td>7</td><td>2</td><td>6</td><td>5</td></tr></table>	9	4	7	2	6	5	ind_min = 3, polje[ind_min] < polje[i] → zamijeni
9	4	7	2	6	5				
i = 1	polje	<table><tr><td>2</td><td>4</td><td>7</td><td>9</td><td>6</td><td>5</td></tr></table>	2	4	7	9	6	5	ind_min = 5, polje[ind_min] ≥ polje[i]
2	4	7	9	6	5				
i = 2	polje	<table><tr><td>2</td><td>4</td><td>7</td><td>9</td><td>6</td><td>5</td></tr></table>	2	4	7	9	6	5	ind_min = 5, polje[ind_min] < polje[i] → zamijeni
2	4	7	9	6	5				
i = 3	polje	<table><tr><td>2</td><td>4</td><td>5</td><td>9</td><td>6</td><td>7</td></tr></table>	2	4	5	9	6	7	ind_min = 4, polje[ind_min] < polje[i] → zamijeni
2	4	5	9	6	7				
i = 4	polje	<table><tr><td>2</td><td>4</td><td>5</td><td>6</td><td>9</td><td>7</td></tr></table>	2	4	5	6	9	7	ind_min = 5, polje[ind_min] < polje[i] → zamijeni
2	4	5	6	9	7				
kraj	polje	<table><tr><td>2</td><td>4</td><td>5</td><td>6</td><td>7</td><td>9</td></tr></table>	2	4	5	6	7	9	
2	4	5	6	7	9				



# Rješenje

```
/* izostavljen uobicajeni kod za ucitavanje n i n clanova polja */
for (i = 0; i < n - 1; i = i + 1) {
    /* trazi indeks najmanjeg medju polje[i+1] ... polje[n-1] */
    ind_min = i + 1;
    for (j = i + 2; j < n; j = j + 1) {
        if (polje[j] < polje[ind_min]) ind_min = j;
    }
    /* u ind_min se sada nalazi indeks najmanjeg clana */
    if (polje[ind_min] < polje[i]) {
        /* obavi zamjenu clanova polje[i] i polje[ind_min] */
        pomocna = polje[i];
        polje[i] = polje[ind_min];
        polje[ind_min] = pomocna;
    }
}

/* izostavljen uobicajeni kod za ispis */
```

# Polja

Višedimenzijska polja

# Višedimenzijsko polje

- Jednodimenzijsko polje (vektor)

```
int a[4];
```

a

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

- Polje može imati više dimenzija
  - dvije dimenzije (matrica, tablica)
    - npr. matrica od 3 retka i 5 stupaca

```
int b[3][5];
```

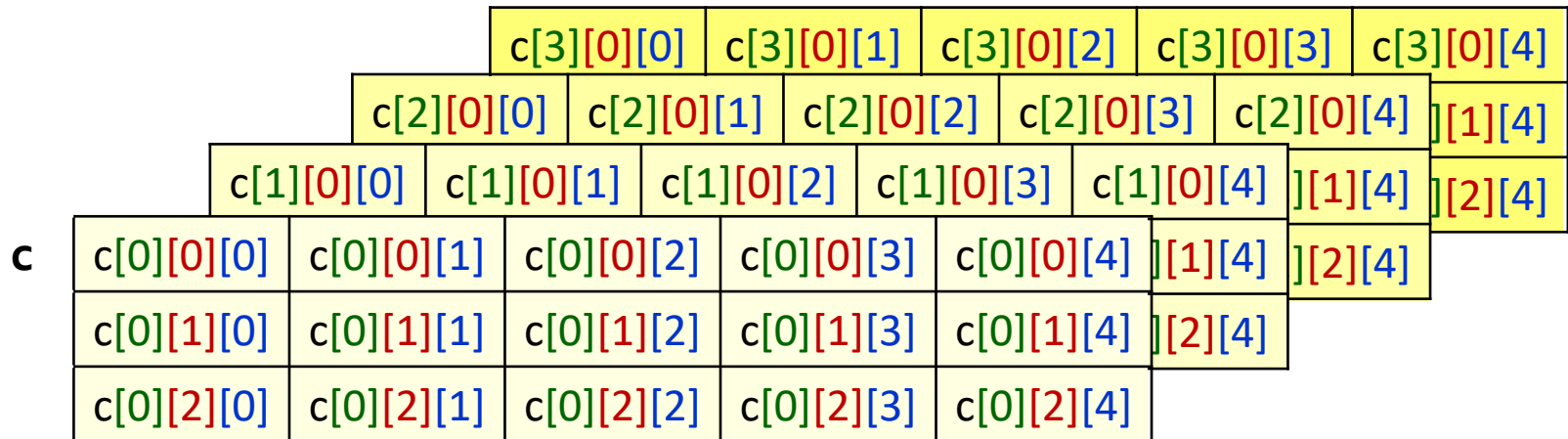
b

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

# Višedimenzijsko polje

- Polje može imati više dimenzija

- tri dimenzije `int c[4][3][5];`



- broj dimenzija nije ograničen, npr.

```
int d[4][4][4][4][4][4][4][4][4][4][4][4][4][4];
```

- oprez: ovo polje ima  $4^{14}$  članova. Memorija?

# Definicija polja uz inicijalizaciju

- Članovi polja mogu se inicijalizirati u trenutku definicije polja
  - **nije primjenjivo za VLA polja!**
  - bez inicijalizacije, članovi polja sadrže nedefinirane vrijednosti

```
int polje[3][4];
```

**polje**

?	?	?	?
?	?	?	?
?	?	?	?

- početne vrijednosti se mogu redom navesti u tzv. inicijalizatoru

```
int polje[3][4] = {1, 2, 3, 4, 5, 6,  
                  7, 8, 9, 10, 11, 12};
```

**polje**

1	2	3	4
5	6	7	8
9	10	11	12

# Definicija polja uz inicijalizaciju

- Raspored vrijednosti po redcima bolje će se vidjeti ako se koristi sljedeći oblik inicijalizatora

```
int polje[3][4] = {{1, 2, 3, 4},  
                  {5, 6, 7, 8},  
                  {9, 10, 11, 12}};
```

**polje**

1	2	3	4
5	6	7	8
9	10	11	12

- navede li se premalo vrijednosti, ostali članovi se postavljaju na nulu

```
int polje[3][4] = {{1, 2},  
                  {5, 6, 7}};
```

**polje**

1	2	0	0
5	6	7	0
0	0	0	0

- designiranim inicijalizatorom se članovi polja mogu ciljano postaviti

```
int polje[3][4] = {{1, 2},  
                  {[2] = 7},  
                  [2][1] = 11};
```

**polje**

1	2	0	0
0	0	7	0
0	11	0	0

# Definicija polja uz inicijalizaciju

- automatsko određivanje veličine polja na temelju inicijalizatora moguće je samo za prvu dimenziju (sve ostale moraju biti navedene)

```
int polje[][4] = {{1, 2, 3, 4},  
                 {5, 6, 7},  
                 {9, 10}};
```

**polje**

1	2	3	4
5	6	7	0
9	10	0	0

- inače, prevodilac dojavljuje pogrešku

```
int polje[][] = {{1, 2, 3, 4},  
                {5, 6, 7, 8},  
                {9, 10, 11, 12}};
```

Prevodilac dojavljuje pogrešku

- u inicijalizatoru ne smije biti navedeno previše vrijednosti

```
int polje[2][3] = {{1, 2, 3, 4},  
                  {5, 6, 7, 8},  
                  {9, 10, 11, 12}};
```

Prevodilac dojavljuje pogrešku

# Primjer

- Programski zadatak
  - Po retcima učitati vrijednosti članova dvodimenzijskog realnog polja od 4 retka i 5 stupaca (kolokvijalno: dimenzija 4 x 5). Sadržaj polja ispisati u obliku tablice

Upisite članove polja >↵

-43.1 15 122.21 0.15 11↵

19.7 0.9761 54 33.7888 1↵

0 0 4.45 4.4 -45↵

28.1 28 6.721 -1 2↵

↵

-43.10	15.00	122.21	0.15	11.00↵
--------	-------	--------	------	--------

19.70	0.98	54.00	33.79	1.00↵
-------	------	-------	-------	-------

0.00	0.00	4.45	4.40	-45.00↵
------	------	------	------	---------

28.10	28.00	6.72	-1.00	2.00↵
-------	-------	------	-------	-------



# Rješenje (1. dio)

```
#include <stdio.h>
#define BR_RED 4           // broj redaka
#define BR_STUP 5         // broj stupaca

int main(void) {
    int redak, stupac;
    float polje[BR_RED][BR_STUP];

    /* učitavanje vrijednosti članova polja */
    printf("Upisite članove polja >\n");
    for (redak = 0; redak < BR_RED; redak = redak + 1) {
        for (stupac = 0; stupac < BR_STUP; stupac = stupac + 1) {
            scanf("%f", &polje[redak][stupac]);
        }
    }
    /* ispis praznog retka nakon učitavanja */
    printf("\n");
}
```

## Rješenje (2. dio)

```
/* ispis polja u obliku tablice */  
for (redak = 0; redak < BR_RED; redak = redak + 1) {  
    for (stupac = 0; stupac < BR_STUP; stupac = stupac + 1) {  
        printf("%8.2f", polje[redak][stupac]);  
    }  
    /* skok u novi red nakon ispisa jednog retka tablice */  
    printf("\n");  
}  
return 0;  
}
```

# Primjer

- Programski zadatak
  - Učitati vrijednosti za broj redaka  $m$  (ne smije biti veći od 10) i broj stupaca  $n$  (ne smije biti veći od 20). Ponavljati učitavanje broja redaka i ponavljati učitavanje broja stupaca dok ne budu ispravni
  - Učitati vrijednosti članova dvodimenzijskog cjelobrojnog polja od  $m$  redaka i  $n$  stupaca.
  - U svakom retku polja pronaći najveći član i ispisati njegovu poziciju i vrijednost

# Primjer

- primjer izvršavanja programa

```
Upisite broj redaka > -2↵
Upisite broj redaka > 11↵
Upisite broj redaka > 3↵
Upisite broj stupaca > 40↵
Upisite broj stupaca > 4↵
Upisite 3 x 4 cijelih brojeva >↵
1 2 4 -8↵
-8 -7 -5 -2↵
9 4 9 1↵
Najveci clanovi po retcima:↵
polje(0, 2) = 4↵
polje(1, 3) = -2↵
polje(2, 0) = 9↵
```

# Rješenje (1. dio)

```
#include <stdio.h>
#define MAKS_RED 10           // najveći dopusteni broj redaka
#define MAKS_STUP 20         // najveći dopusteni broj stupaca

int main(void) {
    int i, j;                 // kontrolne varijable za petlje
    int brred, brstup;        // stvarni broj redaka i stupaca

    /* Ucitavanje brred dok ne bude ispravan */
    do {
        printf("Upisite broj redaka > ");
        scanf("%d", &brred);
    } while (brred < 1 || brred > MAKS_RED);

    /* Ucitavanje brstup dok ne bude ispravan */
    do {
        printf("Upisite broj stupaca > ");
        scanf("%d", &brstup);
    } while (brstup < 1 || brstup > MAKS_STUP);
```

## Rješenje (2. dio)

```
/* definicija VLA polja dimenzija brred x brstup */  
int polje[brred][brstup];  
  
/* Ucitavanje clanova polja */  
printf("Upisite %d x %d cijelih brojeva >\n", brred, brstup);  
for (i = 0; i < brred; i = i + 1) {  
    for (j = 0; j < brstup; j = j + 1) {  
        scanf("%d", &polje[i][j]);  
    }  
}
```

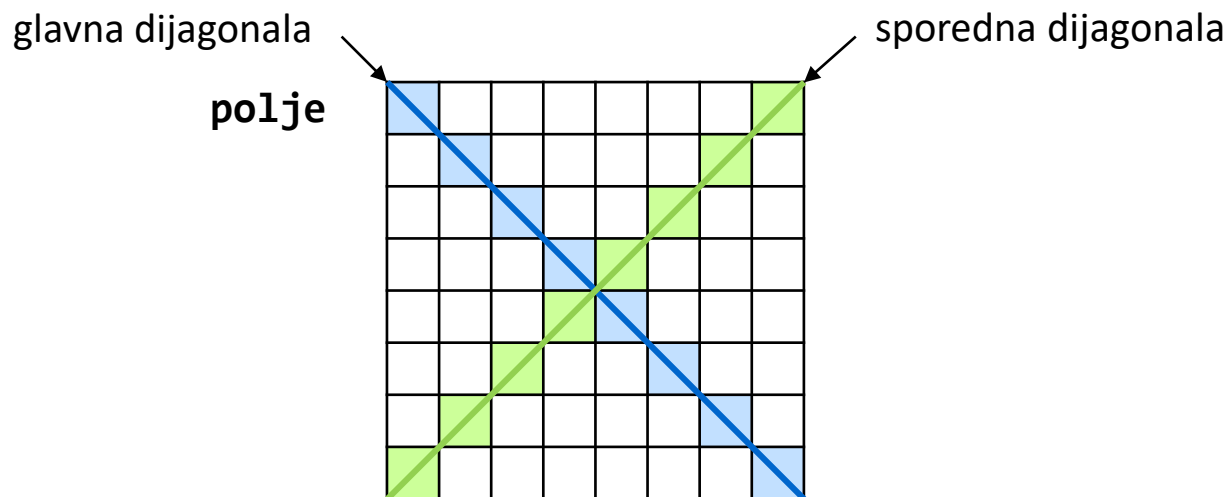
## Rješenje (3. dio)

```
/* pronadji i ispisi najveći član u svakom retku */
int stupacNajveceg;

printf("Najveći članovi po retcima:\n");
for (i = 0; i < brred; i = i + 1) {
    /* pronadji indeks najvećeg u retku i */
    stupacNajveceg = 0;          // pretpostavka: prvi je najveći
    for (j = 1; j < brstup; j = j + 1) {
        if (polje[i][j] > polje[i][stupacNajveceg]) {
            stupacNajveceg = j; // promijeni pretpostavku
        }
    }
    printf("polje(%d, %d) = %d\n",
           i, stupacNajveceg, polje[i][stupacNajveceg]);
}
return 0;
}
```

# Primjer

- Programski zadatak
  - Učitati vrijednosti (realni tip podataka) kvadratne matrice reda 8
    - kvadratna matrica je tablica koja ima jednak broj redaka i stupaca (red matrice)
  - Ispisati najmanju vrijednost na glavnoj dijagonali i najmanju vrijednost na sporednoj dijagonali
    - glavna i sporedna dijagonala kvadratne matrice





# Rješenje

- Glavna dijagonala
  - pretpostavka: `polje[0][0]` je najmanji
  - provjeriti ostale na glavnoj dijagonali

```
min_gl = polje[0][0];  
for (i = 1; i < 8; i = i + 1)  
    if (polje[i][i] < min_gl)  
        promijeni pretpostavku
```

- Sporedna dijagonala
  - pretpostavka: `polje[0][7]` je najmanji
  - provjeriti ostale na sporednoj dijagonali

```
min_sp = polje[0][7];  
for (i = 1; i < 8; i = i + 1)  
    if (polje[i][8 - 1 - i] < min_sp)  
        promijeni pretpostavku
```

0, 0							0, 7
	1, 1					1, 6	
		2, 2			2, 5		
			3, 3	3, 4			
			4, 3	4, 4			
		5, 2			5, 5		
	6, 1					6, 6	
7, 0							7, 7

# Rješenje (1. dio)

```
#include <stdio.h>
#define RED_MATRICE 8

int main(void) {
    int i, j;
    float polje[RED_MATRICE][RED_MATRICE], min_gl, min_sp;
    /* izostavljen je uobicajeni kod za učitavanje članova polja */
    /* pretpostavka: prvi član glavne dijagonale je najmanji */
    min_gl = polje[0][0];
    /* provjeri ostale članove glavne dijagonale */
    for (i = 1; i < RED_MATRICE; i = i + 1) {
        if (polje[i][i] < min_gl) {
            /* promijeni pretpostavku */
            min_gl = polje[i][i];
        }
    }
}
```

## Rješenje (2. dio)

```
/* pretpostavka: prvi clan sporedne dijagonale je najmanji */
min_sp = polje[0][RED_MATRICE - 1];

/* provjeri ostale clanove sporedne dijagonale */
for (i = 1; i < RED_MATRICE; i = i + 1) {
    if (polje[i][RED_MATRICE - 1 - i] < min_sp) {
        /* promijeni pretpostavku */
        min_sp = polje[i][RED_MATRICE - 1 - i];
    }
}

/* izostavljen je uobicajeni kod za ispis rezultata */

return 0;
}
```

# Primjer

- Programski zadatak
  - Učitati vrijednosti za broj redaka  $m$  i broj stupaca  $n$  matrice  $mat$  (dvodimenzijskog cjelobrojnog polja). Nije potrebno provjeravati jesu li upisane ispravne vrijednosti.
  - Učitati vrijednosti matrice  $mat$ . Ispisati ih u obliku tablice
  - Definirati novo polje  $matT$  u koje treba pohraniti matricu dobivenu transponiranjem matrice  $mat$ 
    - transponiranje se obavlja zamjenom redaka sa stupcima: element na poziciji  $[i][j]$  matrice  $mat$  postaje element na poziciji  $[j][i]$  matrice  $matT$
    - transponirana matrica će imati  $n$  redaka i  $m$  stupaca
  - dobivenu matricu  $matT$  ispisati u obliku tablice

# Primjer

- primjer izvršavanja programa

```
Upisite broj redaka i broj stupaca > 3 5↵
```

```
Upisite 3 x 5 cijelih brojeva >↵
```

```
1 2 3 4 5↵
```

```
6 7 8 9 10↵
```

```
11 12 13 14 15↵
```

```
↵
```

```
1 2 3 4 5↵
```

```
6 7 8 9 10↵
```

```
11 12 13 14 15↵
```

```
↵
```

```
1 6 11↵
```

```
2 7 12↵
```

```
3 8 13↵
```

```
4 9 14↵
```

```
5 10 15↵
```

# Rješenje

**mat**

**n**

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

```
for (i = 0; i < m; i = i + 1)
    for (j = 0; j < n; j = j + 1)
        matT[j][i] = mat[i][j];
```

**matT**

**m**

**n**

?	?	?
?	?	?
?	?	?
?	?	?
?	?	?

mat[0][0] → matT[0][0]  
mat[0][1] → matT[1][0]  
mat[0][2] → matT[2][0]  
...  
mat[2][2] → matT[2][2]  
mat[2][3] → matT[3][2]  
mat[2][4] → matT[4][2]

**matT**

**m**

**n**

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

# Rješenje (1. dio)

```
#include <stdio.h>

int main(void) {
    int m, n, i, j;
    printf("Upisite broj redaka i broj stupaca > ");
    scanf("%d %d", &m, &n);
    int mat[m][n], matT[n][m];

    /* ucitaj matricu mat m x n */
    printf("Upisite %d x %d cijelih brojeva >\n", m, n);
    for (i = 0; i < m; i = i + 1) {
        for (j = 0; j < n; j = j + 1) {
            scanf("%d", &mat[i][j]);
        }
    }

    printf("\n");
}
```

## Rješenje (2. dio)

```
/* ispiši matricu mat m x n */
for (i = 0; i < m; i = i + 1) {
    for (j = 0; j < n; j = j + 1) {
        printf("%4d", mat[i][j]);
    }
    printf("\n");
}
printf("\n");

/* vrijednosti iz mat kopiraj na odgovarajuće pozicije u matT */
for (i = 0; i < m; i = i + 1) {
    for (j = 0; j < n; j = j + 1) {
        matT[j][i] = mat[i][j];
    }
}
```



## Rješenje (3. dio)

```
/* ispiši matricu matT n x m */  
for (i = 0; i < n; i = i + 1) {  
    for (j = 0; j < m; j = j + 1) {  
        printf("%4d", matT[i][j]);  
    }  
    printf("\n");  
}  
return 0;  
}
```

# Primjer

- Programski zadatak
  - Učitati red kvadratne matrice  $n$ . Nije potrebno provjeravati ispravnost unosa
  - Učitati vrijednosti kvadratne matrice mat reda  $n$ . U obliku tablice ispisati učitane vrijednosti
  - Transponirati matricu mat zamjenom članova unutar matrice (bez definiranja nove matrice i bez korištenja pomoćnih polja)
  - U obliku tablice ispisati novi sadržaj matrice

# Rješenje

mat

n

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

mat

n

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

```
for (i = 0; i < n - 1; i = i + 1)
    for (j = i + 1; j < n; j = j + 1)
        mat[i][j] ⇔ mat[j][i];
```

```
mat[0][1] ⇔ mat[1][0]
mat[0][2] ⇔ mat[2][0]
mat[0][3] ⇔ mat[3][0]
mat[0][4] ⇔ mat[4][0]
mat[1][2] ⇔ mat[2][1]
...
mat[3][4] ⇔ mat[4][3]
```

# Rješenje

```
#include <stdio.h>

int main(void) {
    int n, i, j, pomocna;
    printf("Upisite red matrice > ");
    scanf("%d", &n);
    int mat[n][n];

    /* izostavljen je uobicajeni kod za ucitavanje polja */
    /* izostavljen je uobicajeni kod za ispis polja */

    for (i = 0; i < n - 1; i = i + 1) {
        for (j = i + 1; j < n; j = j + 1) {
            pomocna = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = pomocna;
        }
    }
    /* izostavljen je uobicajeni kod za ispis novog sadrzaja polja */
    return 0;
}
```

# Agregatni tipovi podataka

## Strukture

# Struktura

- **Struktura ili zapis (structure, record)** je složeni tip podatka koji obuhvaća više članova koji ne moraju biti istog tipa
  - jedna varijabla  $\leftrightarrow$  više vrijednosti
  - jednim imenom objedinjuje se više logički povezanih podataka
    - npr. umjesto zasebnih varijabli

```
int mbr;           // maticni broj studenta
float mi;          // broj bodova medjuispit
float zi;          // broj bodova završni ispit
float lab[8];      // broj bodova lab. vježbe
```

- definira se varijabla bodovi koja će imati članove mbr, mi, zi i lab
- pojedinim vrijednostima se pristupa pomoću imena člana

# Deklaracija strukture

- Struktura se mora prvo opisati
  - deklaracija strukture, nacrt strukture (*structure tag*)
    - ime strukture
    - imena i tipovi članova strukture

```
struct naziv_strukture {  
    tip_elementa_1  ime_elementa_1;  
    tip_elementa_2  ime_elementa_2;  
    ...  
    tip_elementa_n  ime_elementa_n;  
};
```

- Deklaracija strukture se nakon toga može koristiti
  - za definiranje varijabli (može i polja) tipa strukture
  - za deklaraciju drugih struktura koji će sadržavati članove tipa strukture

# Primjer

deklaracija strukture **bodovi\_s**

```
struct bodovi_s {  
    int mbr;  
    float mi;  
    float zi;  
    float lab[8];  
};
```

prema konvenciji imenu strukture dodaje se nastavak **\_s**

definicija varijable **bodovi**

```
struct bodovi_s bodovi;  
...  
bodovi.mbr = 1234;  
bodovi.mi = 23.5f;  
bodovi.lab[0] = 1.5f;  
bodovi.lab[1] = 1.2f;
```

varijabla **bodovi** je tipa strukture **bodovi\_s**

operator pristupa članu strukture: **.** (točka)

pristup članovima strukture: **ime\_varijable.ime\_clana**

**bodovi**

mbr	1234							
mi	23.5							
zi	?							
lab	1.5	1.2	?	?	?	?	?	?



# Deklaracija strukture - varijante

- Varijable je moguće definirati istovremeno uz deklaraciju strukture, a kasnije istu deklaraciju koristiti za definiranje dodatnih varijabli

```
struct datum_s {  
    int dan;  
    int mjesec;  
    int godina;  
} francRev, amerRev;  
...  
francRev.dan = 5; francRev.mjesec = 5; francRev.godina = 1789;  
amerRev.dan = 19; amerRev.mjesec = 4; amerRev.godina = 1775;  
...  
struct datum_s seljBuna;  
seljBuna.dan = 28; seljBuna.mjesec = 1; seljBuna.godina = 1573;
```

# Deklaracija strukture - varijante

- Ime strukture se može ispustiti iz deklaracije ako deklaracija neće biti potrebna za definiranje dodatnih varijabli

```
struct {  
    float latituda;  
    float longituda;  
} geogrPozEiffel, geogrPozFER;  
...  
geogrPozEiffel.latituda = 48.85822f;  
geogrPozEiffel.longituda = 2.2945f;  
geogrPozFER.latituda = 45.80107f;  
geogrPozFER.longituda = 15.97083f;  
...
```

# Definicija uz inicijalizaciju

- Slično poljima, varijabla tipa strukture se može inicijalizirati u trenutku definicije

```
struct bodovi_s bodovi1 = { 4321, 10.5f, 21.5f,  
                           {0.5f, 1.5f, 1.8f}  
                           };
```

mbr	4321							
mi	10.5							
zi	21.5							
lab	0.5	1.5	1.8	0.0	0.0	0.0	0.0	0.0

```
struct bodovi_s bodovi2 = { 1243, .zi = 12.5f,  
                           {0.5f, [5] = 2.0f}  
                           };
```

mbr	1243							
mi	0.0							
zi	12.5							
lab	0.5	0.0	0.0	0.0	0.0	2.0	0.0	0.0

# Složene strukture

- Moguće je definiranje podatkovne strukture proizvoljne složenosti jer član strukture može biti struktura ili polje:

```
struct datum_s {
    int dan;
    int mj;
    int god;
};

struct interval_s {
    struct datum_s dat_od;
    struct datum_s dat_do;
};

struct interval_s zim_rok = {{11, 2, 2019}, {22, 2, 2019}};

printf("Zimski rok: %d.%d.%d. - %d.%d.%d.",
       zim_rok.dat_od.dan, zim_rok.dat_od.mj, zim_rok.dat_od.god,
       zim_rok.dat_do.dan, zim_rok.dat_do.mj, zim_rok.dat_do.god);
```

# Struktura jest *modifiable lvalue*

- Za razliku od polja, varijabla tipa strukture jest *modifiable lvalue*
  - čak i onda kada se kao član strukture koristi polje!

```
struct bodovi_s bodovi1 = { 4321, 10.5f, 21.5f,  
                           {0.5f, 1.5f, 1.8f}  
                           };  
  
struct bodovi_s bodovi2;  
bodovi2 = bodovi1;
```

ispravno

- Operator pridruživanja je jedini operator koji se može koristiti za operacije s dva operanda tipa strukture. Npr. nije moguće koristiti relacijske operatore

```
if (bodovi1 == bodovi2) {  
    ...  
}
```

prevodilac dojavljuje pogrešku

# Strukture kompatibilnog tipa

- Varijabli tipa strukture može se pridružiti sadržaj druge varijable tipa strukture samo u slučaju kada su njihovi tipovi **kompatibilni**, a to znači da su varijable definirane na temelju iste deklaracije

```
struct koordinata_s {  
    float latituda;  
    float longituda;  
};  
struct pozicija_s {  
    float latituda;  
    float longituda;  
};
```

Varijable definirane na temelju deklaracije `koordinata_s` nisu kompatibilne s varijablama definiranim na temelju deklaracije `pozicija_s`. To što su im članovi jednakih imena i tipova za kompatibilnost nije dovoljno.

```
struct koordinata_s tocka1 = {45.80107f, 15.97083f};  
struct koordinata_s tocka2;  
tocka2 = tocka1;  
struct pozicija_s tocka3;  
tocka3 = tocka1;
```

ispravno

prevodilac dojavljuje pogrešku

# Strukture mogu biti članovi polja

- Iako pojedinačne varijable tipa strukture mogu biti korisne, strukture se najčešće koriste kao elementi složenijih podatkovnih struktura, npr. polja

```
struct datum_s {  
    int dan;  
    int mj;  
    int god;  
};  
struct datum_s praznici_2018[] = {  
    { 1,  1, 2018}, { 6,  1, 2018}  
    , { 1,  4, 2018}, { 2,  4, 2018}  
    , { 1,  5, 2018}, {31,  5, 2018}  
    , {22,  6, 2018}, {25,  6, 2018}  
    , { 5,  8, 2018}, {15,  8, 2018}  
    , { 8, 10, 2018}  
    , { 1, 11, 2018}  
    , {25, 12, 2018}, {26, 12, 2018}  
};
```

# Primjer

- Programski zadatak
  - s tipkovnice učitati cijeli broj  $n$  koji predstavlja broj studenata na predmetu Uvod u programiranje. Za svakog studenta učitati matični broj (int), broj bodova na međuispitu (float), broj bodova na završnom ispitu (float) i broj bodova za svaku od osam laboratorijskih vježbi (float). Evidentirati sumu bodova na svim provjerama znanja za svakog studenta (ukupni broj bodova).
  - Sortirati studente prema ukupnom broju bodova, u poretku od većih prema manjim. Poredak studenata koji imaju međusobno jednak broj bodova nije važan.
  - Sortirane podatke ispisati u obliku tablice



# Primjer

## ■ Primjer izvršavanja programa

```
Upisite broj studenata > 740↵
Upisite podatke > 360149290 11.6 5.8 1.9 0.6 1.7 1.4 1.0 1.9 1.5 0.2↵
Upisite podatke > 553721121 15.9 10.2 0.9 0.8 0.8 1.3 1.9 0.4 0.4 1.8↵
Upisite podatke > 277253502 33.3 44.2 1.4 1.8 1.6 0.8 0.9 1.6 0.0 1.4↵
...
Upisite podatke > 380893153 3.4 0.4 1.1 0.2 1.5 0.1 0.4 0.6 1.3 0.7↵
Upisite podatke > 711650074 28.5 9.4 1.8 0.8 1.1 1.2 1.8 0.0 0.0 0.5↵
↵
Rang lista↵
Rbr.  Mat. broj      MI      ZI      LAB  Ukupno↵
=====↵
   1. 277253502   33.3   44.2    9.5    87.0↵
   2. 378063837   33.5   44.6    8.6    86.7↵
   3. 419558299   28.9   44.4   11.9    85.2↵
...
 739. 389674171    0.2    1.2    8.6    10.0↵
 740. 380893153    3.4    0.4    5.9     9.7↵
```

# Rješenje (1. dio)

```
#include <stdio.h>
#define BR_LAB_VJ 8

int main(void) {
    struct bodovi_s {
        int mbr;
        float mi;
        float zi;
        float lab[BR_LAB_VJ];
        float ukupno;
    };

    int n, i, j;
    float ukupno;
    float ukupno_lab;

    printf("Upisite broj studenata > ");
    scanf("%d", &n);

    struct bodovi_s bodovi[n];
```

VLA polje

## Rješenje (2. dio)

```
/* učitavanje bodova za n studenata */
for (i = 0; i < n; i = i + 1) {
    printf("Upisite podatke > ");
    scanf("%d %f %f", &bodovi[i].mbr, &bodovi[i].mi, &bodovi[i].zi);
    ukupno = bodovi[i].mi + bodovi[i].zi;

    // učitavanje bodova za lab. vježbe
    for (j = 0; j < BR_LAB_VJ; j = j + 1) {
        scanf("%f", &bodovi[i].lab[j]);
        ukupno = ukupno + bodovi[i].lab[j];
    }
    bodovi[i].ukupno = ukupno;
}
```

## Rješenje (3. dio)

```
/* sortiranje prema vrijednosti ukupno */
int ind_max;
struct bodovi_s pomocna;
for (i = 0; i < n - 1; i = i + 1) {
    ind_max = i + 1;
    for (j = i + 2; j < n; j = j + 1) {
        if (bodovi[j].ukupno > bodovi[ind_max].ukupno) ind_max = j;
    }
    if (bodovi[ind_max].ukupno > bodovi[i].ukupno) {
        // zamijeni bodovi[i] i bodovi[ind_max]
        pomocna = bodovi[i];
        bodovi[i] = bodovi[ind_max];
        bodovi[ind_max] = pomocna;
    }
}
```

## Rješenje (4. dio)

```
/* ispis rang liste */
printf("\nRang lista\n");
printf("Rbr.  Mat. broj      MI      ZI      LAB  Ukupno\n");
printf("=====\n");

for (i = 0; i < n; i = i + 1) {
    ukupno_lab = 0.f;
    for (j = 0; j < BR_LAB_VJ; j = j + 1) {
        ukupno_lab = ukupno_lab + bodovi[i].lab[j];
    }
    printf("%4d. %9d %5.1f %5.1f %5.1f %7.1f\n",
        i + 1, bodovi[i].mbr,
        bodovi[i].mi, bodovi[i].zi,
        ukupno_lab, bodovi[i].ukupno);
}

return 0;
}
```

# Komentar rješenja

- Zašto rješenje u kojem bi se koristilo pet polja, umjesto polja čiji su članovi strukture, nije ispravno?

```
...  
int mbr[n];  
float mi[n];  
float zi[n];  
float ukupno[n];  
float lab[n][BR_LAB_VJ];  
...
```

- nije prepoznatljiva povezanost podataka u tim poljima
  - npr. nije odmah jasno da se vrijednosti mbr[0], mi[0], zi[0], ukupno[0] i redak polja lab s indeksom 0 odnose na istog studenta
- teže je baratati skupinama podataka
  - npr. napraviti kopiju vrijednosti svih podataka o jednom studentu