Uvod u računarstvo
13. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Polja

Sadržaj predavanja

- Složene strukture podataka: nizovi (polja):
  - Definicija jednodimenzionalnog polja.
  - Inicijalizacija jednodimenzionalnog polja.
  - Polje kao argument funkcije.
  - Pokazivači i jednodimenzionalna polja.

Polje

Polje je niz varijabli istog tipa (sa zajedničkim imenom) numeriranih cjelobrojnim indeksom.

- Indeks uvijek počinje od nule.
- Radi efikasnosti pristupa, elementi polja smještaju se u uzastopne memorijske lokacije (redom po indeksu).

Primjer:

```
double x[3];  /* polje x tipa double */
              /* s 3 clana ili elementa */
x[0] = 0.2;
x[1] = 0.7;
x[2] = 5.5;
/* x[3] = 4.4; - greska, nije definirano */
```

Definicija polja

Jednodimenzionalno polje definira se na sljedeći način:

```
mem_klasa tip ime[izraz];
```

gdje je:

- `mem_klasa` memorijska klasa cijelog polja,
- `tip` tip podatka svakog elementa polja,
- `ime` ime polja (zajednički dio imena svih elemenata),
- a `izraz` konstantan, cjelobrojni, pozitivan izraz koji zadaje broj elemenata.

Ovaj `izraz` je najčešće pozitivna konstanta ili simbolička konstanta.

Definicija polja (nastavak)

Elementi jednodimenzionalnog polja su:

`ime[0]`, . . . , `ime[izraz - 1]`.

Svaki element je varijabla tipa `tip`.

Deklaracija memorijske klase nije obavezna.

Polje deklarirano bez memorijske klase:

- unutar funkcije je automatska varijabla (rezervacija memorije na "run–time stacku", ulaskom u funkciju),
- a izvan svih funkcija je statička varijabla.

Unutar funkcije polje se može učiniti statičkim pomoću identifikatora memorijske klase `static`.

Inicijalizacija polja

Polja se mogu inicijalizirati (element po element),

- navođenjem popisa vrijednosti elemenata unutar vitičastih zagrada.
- U tom popisu, pojedine vrijednosti odvojene su zarezom (koji nije operator).

Sintaksa:

```
mem_klasa tip ime[izraz] = {v_1, ..., v_n};
```

što daje

`ime[0] = v_1`, ... , `ime[n - 1] = v_n`.

Inicijalizacija polja (nastavak)

Primjer:

```
float v[3] = {1.17, 2.43, 6.11};
```

je ekvivalentno s

```
float v[3];
v[0] = 1.17;
v[1] = 2.43;
v[2] = 6.11;
```

Inicijalizacija polja (nastavak)

Ako je broj inicijalizacijskih vrijednosti `n`

- veći od dimenzije polja — javlja se greška,
- manji od dimenzije polja, onda će preostale vrijednosti biti inicijalizirane nulom.

Prilikom inicijalizacije dimenzija polja ne mora biti zadana.

- Tada se dimenzija polja računa automatski, iz broja inicijalizacijskih vrijednosti.

Primjer: možemo pisati

```
float v[] = {1.17, 2.43, 6.11};
```

što kreira polje `v` dimenzije 3 i inicijalizira ga.

Inicijalizacija polja (nastavak)

Polja znakova mogu se inicijalizirati znakovnim nizovima.

Primjer: naredbom

```
char c[] = "tri";
```

definirano je polje od 4 znaka:

`c[0] = 't'`, `c[1] = 'r'`, `c[2] = 'i'`, `c[3] = '\0'`.

Takav način pridruživanja dozvoljen je samo u definiciji varijable (kao inicijalizacija). Nije dozvoljeno pisati:

```
c = "tri";  /* Pogresno! Koristiti strcpy! */
```

jer lijeva strana pridruživanja ne smije biti polje (ime polja je konstantni pointer — adresa prvog elementa).

Primjer

Računanje aritmetičke sredine.

```
int main(){
int i, n;
    double a_sredina = 0.0;
    double v[] = {2.0, 3.11, 4.05, -1.07};
    n = sizeof(v) / 8;

    for(i = 0; i < n; ++i)
        a_sredina += v[i];
    a_sredina /= n;
    printf("Sredina je %20.12f\n", a_sredina);
    return 0;
}
```

Polje kao argument funkcije

Zapamtiti: Ime polja je sinonim za

- konstantni pokazivač koji sadrži adresu prvog elementa polja (više u sljedećem poglavlju).

Polje može biti formalni (i stvarni) argument funkcije. U tom slučaju:

- ne prenosi se cijelo polje po vrijednosti (kopija polja!),
- već funkcija dobiva (po vrijednosti) pokazivač na prvi element polja.

Unutar funkcije elementi polja mogu se

- dohvatiti i promijeniti, korištenjem indeksa polja.

Razlog: aritmetika pokazivača (v. sljedeće poglavlje).

Polje kao argument funkcije (nastavak)

Funkciju `f` koja uzima polje `v` tipa `tip` kao argument, možemo deklarirati na dva načina:

```
f(tip v[])    ili    f(tip *v)
```

U prvom načinu ne treba navesti dimenziju. Drugi način direktno kaže da je ime polja `v` pokazivač na objekt tipa `tip` i podrazumijeva se da je to adresa prvog elementa polja.

Ako ne želimo da funkcija mijenja elemente polja unutar funkcije, onda dodajemo ključnu riječ `const` na početku deklaracije argumenta:

```
f(const tip v[])    ili    f(const tip *v)
```

Polje kao argument funkcije (nastavak)

Primjer. Funkciju koja uzima polje realnih brojeva (tipa `double`) i računa srednju vrijednost svih elemenata polja možemo napisati ovako:

```
double srednja_vrijednost(int n, double v[]) {
    int i;
    double suma = 0.0;

    for (i = 0; i < n; ++i) suma += v[i];
    return suma/n;
}
```

Uočite da je broj elemenata `n`, također, argument funkcije. Inače funkcija ne zna broj elemenata (osim iz neke globalne varijable).

Polje kao argument funkcije (nastavak)

Pri pozivu funkcije koja ima polje kao formalni argument, stvarni argument je

- ime polja ili pokazivač na "prvi" element u polju.

```
int main(void) {
    int n;
    double v[] = {1.0, 2.0, 3.0}, sv;

    n = 3;
    sv = srednja_vrijednost(n, v);
    return 0;
}
```

Poziv `srednja_vrijednost(2, &v[1])` je korektan!

Pokazivači i jednodimenzionalna polja

Ime jednodimenzionalnog polja je konstantni pokazivač na prvi element polja!

Primjer:

```
int a[10], b[10];
...
a = a + 1; /* Greska, a konst. pokazivac. */
b = a;     /* Greska! */
```

Pokazivači i jednodim. polja (nastavak)

Primjer:

```
int a[10], *pa;
...
pa = a;          /* ekviv. s pa = &a[0]; */
pa = pa + 2;     /* Nije greska - &a[2] */
pa++;            /* &a[3] */
```

Primjer:

```
int a[10], *pa;
...
pa = &a[0];
*(pa + 3) = 20;   /* ekviv. s a[3] = 20; */
*(a + 1) = 10;    /* ekviv. s a[1] = 10; */
```

Prioriteti

Primjer: važnost prioriteta – ako je

```
int a[4] = {0, 10, 20, 30};
int *ptr, x;
ptr = a;
```

onda je

izraz	x	ptr
`x = *ptr;`	0	1245040
`x = *ptr++;`	0	1245044
`x = (*ptr)++;`	10	1245044
`x = *++ptr;`	20	1245048
`x = ++(*ptr);`	21	1245048