Programiranje 2
7. predavanje

Saša Singer

singer@math.hr
web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja

- Tipovi i složene deklaracije:
  - Pokazivači i polja (ponavljanje).
  - Pokazivač na funkciju (ponavljanje).
  - Složene deklaracije — primjeri.
  - Deklaracija tipova — `typedef`.
- Strukture (prvi dio):
  - Deklaracija strukture. Strukture i `typedef`.
  - Rad sa strukturama. Operator točka.
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (`->`).
  - Unije.
  - Dodatak: Polja bitova.

Informacije — predavanja

Trenutno nema konkretnih informacija.

- Pratite web stranice fakulteta!
- Za "nastavu na daljinu" pratite web stranice kolegija.

Tipovi i složene deklaracije

Sadržaj

- Tipovi i složene deklaracije:
  - Pokazivači i polja (ponavljanje).
  - Pokazivač na funkciju (ponavljanje).
  - Složene deklaracije — primjeri.
  - Deklaracije tipova — `typedef`.

Polje pokazivača i pokazivač na polje

Polje pokazivača ima deklaraciju:

```
tip_pod *ime[izraz];
```

Napomena: Primarni operator `[ ]` ima viši prioritet od unarnog operatora `*`.

Primjer. Razlikujte polje pokazivača (ovdje, 10 pokazivača):

```
int *ppi[10];
```

od pokazivača na polje (ovdje, od 10 elemenata):

```
int (*ppi)[10];
```

Pokazivač na funkciju

Pokazivač na funkciju deklarira se kao:

```
tip_pod (*ime)(tip_1 arg_1, ..., tip_n arg_n);
```

Ovdje je `ime` varijabla tipa — pokazivač na funkciju, koja

- uzima `n` argumenata, tipa `tip_1` do `tip_n`,
- i vraća vrijednost tipa `tip_pod`.

Slično kao i u prototipu funkcije, u deklaraciji ne treba pisati imena argumenata `arg_1` do `arg_n`.

Primjer:

```
int (*pf)(char c, double a);
int (*pf)(char, double);
```

Pokazivač na funkciju (nastavak)

U deklaraciji pokazivača na funkciju — zagrade su nužne.

- Primarni operator `( )` — "poziva" ili argumenata funkcije, ima viši prioritet od unarnog operatora `*`.

Primjer. Razlikujte funkciju koja vraća pokazivač na neki tip (ovdje, na `double`):

```
double *pf(double, double);
double *(pf(double, double));    /* Isto */
```

od pokazivača na funkciju koja vraća vrijednost nekog tipa (ovdje, tipa `double`):

```
double (*pf)(double, double);
```

Pokazivač na funkciju — primjeri

Primjeri pokazivača na funkciju f iz integracije (prošli put):

```
double integracija(double, double,
                   double (*)(double));

double integracija(double a, double b,
                   double (*f)(double)) {
    return 0.5 * (b - a) * ( (*f)(a) + (*f)(b) );
}
```

ili:

```
double integracija(double, double, int,
                   double (*)(double));
    ...
```

Složene deklaracije — primjeri

Kod interpretacije deklaracije uzimaju se u obzir prioriteti i asocijativnost pojedinih operatora. Ti prioriteti mogu se promijeniti upotrebom zagrada.

Primjeri. Što je `p` u sljedećim deklaracijama?

```
int *p[10];           /* polje od 10 ptr na int */
int *p(void);         /* funkcija koja nema arg i
                         vraca pokazivac na int */
int p(char *a);       /* funkcija koja uzima ptr na
                         char i vraca int */
int *p(char *a);      /* funkcija koja uzima ptr na
                         char i vraca ptr na int */
int (*p)(char *a);    /* ptr na funkciju koja uzima
                         ptr na char i vraca int */
```

Složene deklaracije — primjeri (nastavak)

```
int (*p(char *a))[10];    /* funk. uzima ptr na char
                             i vraca ptr na polje
                             od 10 elem. tipa int */
int p(char (*a)[8]);      /* funk. uzima ptr na polje
                             od 8 char i vraca int */
int (*p)(char (*a)[8]);   /* ptr na funk. koja uzima
                             ptr na polje od 8 char i
                             vraca int */
int *(*p)(char (*a)[8]); /* ptr na funk. koja uzima
                             ptr na polje od 8 char i
                             vraca ptr na int */
int *(*p[10])(char *a);   /* polje od 10 ptr na funk.
                             koja uzima ptr na char
                             i vraca ptr na int */
```

Deklaracije tipova — `typedef`

Ključna riječ `typedef`

Korištenjem ključne riječi `typedef`

- postojećim ili složenim tipovima podataka dajemo nova imena (ne kreiramo nove objekte ili varijable tog imena).

Jednostavni oblik `typedef` deklaracije je:

```
typedef  tip_podatka  novo_ime_za_tip_podatka;
```

To znači da:

- `novo_ime_za_tip_podatka` postaje sinonim za `tip_podatka`

i smije se tako koristiti u svim kasnijim deklaracijama — tamo gdje smijemo napisati jedno, smijemo napisati i drugo, i to s istim značenjem.

Jednostavne `typedef` *deklaracije*

Primjer. Deklaracijom

```
typedef double Masa;
```

identifikator `Masa` postaje sinonim za `double`.

Nakon toga, varijable tipa `double` možemo deklarirati i kao:

```
Masa m1, m2, *pm1;
Masa elementi[10];
```

Uočite da je

- `pm1` — pokazivač na `double`,
- `elementi` — polje od 10 elemenata tipa `double`.

Međutim, nije baš jasno što smo s tim "dobili"!

Svrha deklaracije tipova

Zaista, kod ovako jednostavnih deklaracija — svrha se ne vidi odmah.

Stvarna svrha deklaracije ili imenovanja tipova je:

- lakše razumijevanje (čitanje) kôda i
- dokumentiranje programa.

To postaje vrlo korisno kod složenijih tipova podataka — kad u programo koristimo

- čitavu hijerarhiju tipova — koji se grade jedni iz drugih.

Korist će se vidjeti vrlo skoro, kad dođemo na

- strukture i samoreferencirajuće strukture (vezane liste, binarna stabla i sl.).

Primjer jednostavne deklaracije tipova

Korist od deklaracije tipova može se vidjeti i na jednostavnim primjerima — ako dobro izaberemo ime za tip.

Primjer.

```
typedef int Metri, Kilometri;
Metri duljina, sirina;
Kilometri udaljenost;
```

Ideja (ili svrha): ovdje ime tipa sugerira jedinice u kojima su izražene određene vrijednosti!

No, stvarna korist od `typedef` je tek kod složenijih tipova.

- Kako se pišu takve deklaracije?

Složenije `typedef` *deklaracije*

Sasvim općenito, deklaracija imena za složeniji tip:

- počinje s `typedef`, a
- dalje ima isti oblik kao i deklaracija varijable tog imena i tog tipa.

Sve je isto, osim što tada

- ime nije varijabla tog tipa (ne dobiva memorijski prostor i adresu), već
- ime postaje sinonim za taj tip, kojeg "bi imala" takva varijabla.

```
typedef  deklaracija_za_tip_podatka;
```

Primjer složenije deklaracije tipa — za polja

Primjer. Uvedimo imena tipova za vektore i matrice odgovarajućih dimenzija (recimo, $n = 10$).

```
#define n 10
typedef double skalar;
typedef skalar vektor[n];
typedef vektor matrica[n];
```

Zadnje dvije deklaracije daju imena poljima:

- `vektor` je ime tipa za polje od `n` (10) skalara (`double`),
- `matrica` je ime tipa za polje od `n` (10) vektora, tj.
- `matrica` je dvodimenzionalno polje skalara, ili sinonim za tip `double[n][n]` = `double[10][10]`.

`typedef` *i polja (nastavak)*

Funkciju za računanje produkta $y = Ax$, kvadratne matrice A i vektora x, možemo i ovako napisati:

```
void prod_mat_vek(matrica A, vektor x, vektor y)
{
    int i, j;
    for (i = 0; i < n; ++i) {
        y[i] = 0.0;
        for (j = 0; j < n; ++j)
            y[i] += A[i][j] * x[j];
        }
}
```

Napomena. Ovdje je `n` fiksan — `n = 10`. Popravite funkciju tako da stvarni red matrice i vektora bude argument funkcije.

Primjer deklaracije tipa — stringovi

Primjer. Kod obrade stringova možemo uvesti deklaraciju

```
typedef char *string;
```

Ovdje je `string` sinonim za pokazivač na `char` (tip `char *`), s očitom svrhom:

- taj pokazivač interpretiramo kao pokazivač na prvi element u polju znakova,
- a to polje znakova obrađujemo kao string (do nul-znaka)!

Funkcija `strcmp` za uspoređivanje stringova smije se ovako deklarirati:

```
int strcmp(string, string);
```

`typedef` *i pokazivači (nastavak)*

Primjer. Pokazivač na `double` nazvat ćemo `Pdouble`.

```
typedef double *Pdouble;
```

`Pdouble` postaje pokazivač na `double`, pa smijemo pisati:

```
Pdouble px;     /* = double *px */

void f(Pdouble, Pdouble);
    /* = void f(double *, double *); */

px = (Pdouble) malloc(100 * sizeof(double));
```

`typedef` *i deklaracije funkcija*

Općenito, `typedef` koristimo za kraće zapisivanje složenih deklaracija.

Primjer. Pokazivač na funkciju.

```
typedef int (*PF)(char *, char *);
```

`PF` postaje ime za tip — pokazivač na funkciju koja uzima dva pokazivača na `char` i vraća `int`. Umjesto deklaracije:

```
void f(double x, int (*g)(char *, char *)) { ... }
```

možemo pisati:

```
void f(double x, PF g) { ... }
```

Operatori u deklaracijama i `typedef`

U deklaracijama varijabli i tipova dozvoljeno je koristiti operatore — one koji imaju smisla u deklaraciji, poput

- `*` = dereferenciranje, `[ ]` = polje, `( )` = funkcija.

Ti operatori djeluju samo na jedan pripadni argument, a ne na sve navedene.

Primjer. U sljedećoj deklaraciji varijabli `a` i `b`

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

operator `*` djeluje samo na identifikator `a`, ne na tip `int`. Dakle, `a` je pokazivač na `int`, a `b` je baš `int`.

Operator u deklaraciji ima "viši" prioritet od navođenja tipa. Zato djeluje na jedan operand i taj je ime varijable (a ne tip).

Operatori u deklaracijama i `typedef`

Ono što zbunjuje je navođenje tipova (recimo, kod funkcija).

- Naime, tip varijable `a` je "`int *`".

U deklaraciji argumenata funkcije (s imenima ili bez njih)

- tipovi se navode za svaki argument posebno,
- i tamo treba navesti korektan tip, zajedno s operatorima.

Za razliku od toga, u deklaraciji varijabli,

- možemo navesti više varijabli u istoj deklaraciji,
- pa se operatori odnose na svaku varijablu posebno, a ne na "zajednički" dio tipa s početka deklaracije.

Ako želimo "zajednički" tip s operatorima, onda treba uvesti ime tipa, preko `typedef`.

Operatori u deklaracijama i `typedef`

Primjer. Ako želimo da obje varijable `a` i `b` budu pokazivači na `int`, onda treba napisati

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

Umjesto toga, možemo deklarirati tip za pokazivač na `int`

```
typedef int *pok_int;
```

a zatim napisati

```
pok_int a, b;
```

Operatori u deklaracijama i `typedef`

Primjer. Za polja `a` i `b`, svakom treba zasebno navesti duljinu

```
int a[10], b[10];
```

Čak i kad su polja iste duljine, ne smijemo napisati

```
int[10] a, b;
```

Ako želimo nešto slično, onda moramo uvesti novi tip

```
typedef int polje[10];
polje a, b;
```

Identifikator `polje` je ime tipa za `int[10]`, tj. sinonim za ono što ne smije pisati na početku deklaracije. A ime smije pisati!

Strukture

Sadržaj

- Strukture (prvi dio):
  - Deklaracija strukture. Strukture i `typedef`.
  - Rad sa strukturama. Operator točka.
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (`->`).
  - Unije.
  - Dodatak: Polja bitova.

Što je struktura?

Struktura je složeni tip podataka, kao i polje. Za razliku od polja, koje služi

- grupiranju podataka istog tipa,

struktura služi

- grupiranju podataka različitih tipova.

Može i ovako — malo detaljnije.

- Svi elementi polja imaju isti tip i zajedničko ime, a razlikuju se po indeksu. To se vidi i u deklaraciji polja.
- Elementi (ili članovi) strukture mogu, ali ne moraju, biti različitog tipa i svaki element ima svoje posebno ime.

Zato u deklaraciji strukture moramo navesti ime i tip svakog člana. Tip strukture možemo deklarirati na dva načina.

Deklaracija strukture — bez `typedef`

Prvi način — bez `typedef`. Tip strukture deklarira se ovako:

```
struct ime {
    tip_1 ime_1;
    tip_2 ime_2;
    ...
    tip_n ime_n;
};
```

Ovdje je `struct` rezervirana riječ, a `ime` je ime strukture.

Stvarni tip strukture je

- `struct ime` — dvije riječi (i to je, ponekad, nezgodno!).

Unutar vitičastih zagrada popisani su članovi strukture.

Definicija varijabli tipa strukture — bez `typedef`

Napomena. Kao i kod polja, članovi strukture

- smješteni su u memoriji jedan za drugim, onim redom kojim su navedeni.

Kod ovakve deklaracije tipa strukture, varijable tog tipa, općenito, definiramo ovako:

```
mem_klasa struct ime var_1, var_2, ..., var_n;
```

- `var_1`, `var_2`, ..., `var_n` su varijable tipa `struct ime`.

Primjer — struktura za točke

Primjer. Struktura `tocka` definira točku u ravnini. Uzmimo da točka ima cjelobrojne koordinate, poput pixela na ekranu.

```
struct tocka {
    int x;
    int y;
};
```

Varijable tipa strukture `tocka` možemo definirati na (barem) dva načina.

Nakon gornje deklaracije strukture `tocka` (kao tipa), napišemo "običnu" definiciju varijabli:

```
struct tocka t1, t2;
```

Primjer — struktura za točke (nastavak)

Deklaraciju tipa strukture ne moramo napisati posebno.

Možemo ju napisati i u sklopu definicije varijabli tog tipa:

```
struct tocka {
    int x;
    int y;
} t1, t2;
```

Prvi način je pregledniji!

Međutim, postoji i bolji način deklaracije tipa strukture, koji olakšava i definiciju varijabli tog tipa — preko `typedef`.

Prednost: tako možemo izbjeći stalno navođenje riječi `struct` u deklaracijama varijabli.

Deklaracija strukture — preko `typedef`

Drugi način — preko `typedef`.

Tip strukture deklarira se ovako:

```
typedef struct ime {
    tip_1 ime_1;
    tip_2 ime_2;
    ...
    tip_n ime_n;
} ime_tipa;      /* <-- Ime tipa za strukturu. */
```

Ovdje smo još, na kraju deklaracije, cijelom tipu strukture dali ime `ime_tipa`. Stvarni tip strukture je onda i

- `ime_tipa` — kao sinonim za `struct ime`.

Sve ostalo je isto kao i prije.

Definicija varijabli tipa strukture — uz `typedef`

Napomena. "Prvo" `ime` strukture (odmah iza `struct`) smijemo i ispustiti, ako ga nigdje nećemo koristiti!
(Uvijek bi trebalo pisati `struct` ispred tog imena.)

Ako pišemo to "prvo" `ime`, ono mora biti različito od svih ostalih imena (identifikatora), pa i od `ime_tipa`.

- Običaj: prvo `ime` = `_ime_tipa` (na primjer, `_osoba`).

Kod ovakve deklaracije tipa strukture, varijable tog tipa, općenito, definiramo ovako:

```
mem_klasa ime_tipa var_1, var_2, ..., var_n;
```

- `var_1`, `var_2`, ..., `var_n` su varijable tipa `ime_tipa`, što je sinonim za `struct ime`.

Primjer — struktura za točke

Primjer. Umjesto ranije definicije strukture za točku u ravnini, možemo uvesti tip `Tocka` za cijelu strukturu.

```
typedef struct {
    int x;
    int y;
} Tocka;
...
Tocka t1, t2, *pt1;
```

Identifikator `Tocka` je ime tipa za cijelu strukturu, a `t1` i `t2` su varijable tipa `Tocka`. Što je `pt1`?

Uočite da ovdje nismo napisali ime strukture iza `struct`, jer ga nećemo koristiti.

Inicijalizacija strukture

Varijablu tipa strukture možemo inicijalizirati pri definiciji (kao i svaku drugu varijablu):

```
mem_klasa struct ime var = {v_1, ..., v_n};
mem_klasa ime_tipa   var = {v_1, ..., v_n};
```

Konstante `v_1`, `v_2`, ..., `v_n` pridružuju se navedenim redom

- odgovarajućim članovima strukture `var` — član, po član.

Inicijalizacija strukture — primjer

Primjer. Ako je definirana struktura

```
struct racun {
    int broj_racuna;
    char ime[80];
    float stanje;
};
```

onda varijablu `kupac` možemo inicijalizirati ovako:

```
struct racun kupac = { 1234, "Pero Bacilova",
                       -12345.00f };
```

Inicijalizacija polja struktura

Primjer. Slično se može inicijalizirati i čitavo polje struktura:

```
struct racun kupci[] = {
                          2234, "Goga",    456.00f,
                          1235, "Josip",  -234.00f,
                           436, "Martina",   0.00f };
```

Operator točka — pristup članu strukture

Članovima strukture može se pojedinačno pristupiti korištenjem primarnog operatora točka (`.`).

- Operator točka (`.`) separira ime varijable i ime člana te strukture.

Ako je `var` varijabla tipa strukture koja sadrži član `memb`, onda je

```
var.memb
```

član `memb` u strukturi `var` (preciznije, vrijednost tog člana).

Napomena. Ime člana je lokalno za svaku strukturu. Zato smijemo koristiti

- isto ime člana u raznim strukturama.

Prioritet i asocijativnost operatora točka

Operator točka (.)

- spada u najvišu prioritetnu grupu (primarni operatori) i ima asocijativnost L $\rightarrow$ D.

Zbog najvišeg prioriteta vrijedi:

- `++varijabla.clan` $\Longleftrightarrow$ `++(varijabla.clan)`
- `&varijabla.clan` $\Longleftrightarrow$ `&(varijabla.clan)`

Član strukture (kao i element polja), naravno,

- smije pisati na lijevoj strani naredbe pridruživanja.

Rad sa strukturama — pristup članovima

Primjer. Pristup članovima strukture.

```
struct tocka {
    int x;    /* prvi clan strukture */
    int y;    /* drugi clan strukture */
};
struct tocka ishodiste;
```

Imena objekata i značenje:

- `ishodiste` je varijabla tipa `struct tocka`,
- `ishodiste.x` je prvi član (ili prva komponenta) varijable `ishodiste`,
- `ishodiste.y` je drugi član (ili druga komponenta) varijable `ishodiste`.

Pristup članovima strukture (nastavak)

Primjer. Ako je

```
struct racun {
    int broj_racuna;
    char ime[80];
    float stanje;
} kupac = { 1234, "Pero Bacilova", -12345.00f };
```

tada je, redom:

```
kupac.broj_racuna = 1234,
kupac.ime = "Pero Bacilova",
kupac.stanje = -12345.00f.
```

Struktura kao član druge strukture

Strukture mogu sadržavati druge strukture kao članove.

Primjer. Pravokutnik paralelan koordinatnim osima možemo zadati parom dijagonalno suprotnih vrhova — na pr. donjim lijevim (`pt1`) i gornjim desnim (`pt2`). Vrhovi su točke.

```
struct pravokutnik {
    struct tocka pt1;    /* ili  Tocka pt1; */
    struct tocka pt2;    /* ili  Tocka pt2; */
};
```

Struktura `struct tocka` (ili `Tocka`) mora biti deklarirana prije deklaracije strukture `pravokutnik`.

U različitim strukturama mogu se koristiti ista imena članova.

Polje kao član strukture

Kad struktura sadrži polje kao član strukture, onda se pojedinim elementima tog polja (zovimo ga `clan`) pristupa izrazom:

```
varijabla.clan[izraz]
```

Koristi se asocijativnost L → D za primarne operatore

- točka (`.`) i
- indeksiranje polja (`[ ]`).

Dakle, značenja objekata se interpretiraju ovim redom:

- prvo za `.` — element `clan` u strukturi `varijabla`,
- zatim za `[ ]` — element s indeksom `indeks` u polju `clan`.

Polje kao član strukture — primjer

Primjer.

```
typedef struct {
    int broj_racuna;
    char ime[80];
    float stanje;
} Racun;
Racun kupac = { 1234, "Pero Bacilova",
                -12345.00f };
...
if (kupac.ime[0] == 'P') puts(kupac.ime);
```

Polje struktura

Ako imamo polje struktura, svaki element polja je struktura. Nekom članu pripadne strukture pristupamo izrazom

```
polje[izraz].clan
```

Opet je bitna asocijativnost: `polje[izraz]` je cijela struktura.

Primjer.

```
struct tocka {
   int x;
   int y;
} vrhovi[1024];     /* Polje tocaka. */
...
if (vrhovi[17].x == vrhovi[17].y) ...
```

Strukture — operacije i funkcije

Dozvoljene operacije nad strukturom, kao cjelinom, su:

- pridruživanje,
- uzimanje adrese, primjena `sizeof` operatora.

Napomena. Nije dozvoljeno uspoređivanje cijelih struktura (čak ni jednako/različito).

Strukture i funkcije:

- Struktura može biti argument funkcije. Funkcija tada dobiva kopiju cijele strukture kao argument.
- Funkcija može vratiti strukturu.

Ovo je isto kao za "obične" varijable, a ne kao kod polja.

Strukture i funkcije — primjer

Primjer. Argumenti funkcije `suma` su dvije strukture tipa `Tocka`, a funkcija vraća sumu argumenata (tipa `Tocka`).
Suma točaka = zbroj odgovarajućih koordinata (kao vektori).

```
typedef struct {
    int x;
    int y;
} Tocka;
Tocka t, ishodiste = {0, 0}, t1 = {1, 7};

Tocka suma(Tocka p1, Tocka p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

Strukture i funkcije — primjer (nastavak)

```
int main(void) {

        /* Dodjeljivanje struktura:
           t i ishodiste moraju biti istog tipa */
    t = ishodiste;

    printf("Velicina = %u byteova\n", sizeof(t));

        /* Zbroj tocaka, rezultat je tocka. */
    t1 = suma(t1, t1);
    printf("t1 = (%d, %d)\n", t1.x, t1.y);

    return 0;
}
```

Strukture i funkcije — kompleksni brojevi

Primjer. Biblioteka funkcija za osnovne operacije s kompleksnim brojevima (v. `complex.c`).

```
typedef struct {
    double re;    /* ili x */
    double im;    /* ili y */
} complex;

    /* Napomena: cabs vec postoji u <math.h>! */
double zabs(complex a) {
    return sqrt( a.re * a.re + a.im * a.im );
}
```

U C99 standardu postoje tipovi i odgovarajuće funkcije za kompleksne brojeve (zaglavlje `<complex.h>`).

Strukture i pokazivači

Pokazivač na strukturu definira se isto kao i pokazivač na druge tipove objekata.

```
struct tocka {
    int x;
    int y;
} p1, *pp1;
...
pp1 = &p1;
(*pp1).x = 13;     /* Zagrade su NUZNE! */
(*pp1).y = 27;

*pp1.x = 13;     /* GRESKA!
                    *pp1.x je isto sto i *(pp1.x) */
```

Operator strelica (->)

Primarni operator strelica (`->`) omogućava jednostavano dohvaćanje člana strukture, preko pokazivača na tu strukturu.

- Asocijativnost operatora `->` je L $\rightarrow$ D.

Ako je `ptvar` pokazivač na strukturu, a `clan` je neki član te strukture, onda je:

`ptvar->clan` $\Longleftrightarrow$ `(*ptvar).clan`

Primjer.

```
struct tocka p1, *pp1 = &p1;
pp1->x = 13;
pp1->y = 27;
```

Složeni izrazi

Pristup koordinatama vrhova pravokutnika `r` — izravno i preko pokazivača `pr`.

```
struct pravokutnik {
    struct tocka pt1;
    struct tocka pt2;
} r, *pr = &r;
```

Sljedeći su izrazi ekvivalentni (`x`-koordinata prvog vrha `pt1`):

```
r.pt1.x          // Operatori . i ->
pr->pt1.x        // imaju isti prioritet.
(r.pt1).x        // Asocijativnost im je
(pr->pt1).x      // L -> D.
```

Unije

Unija

Unija je složeni tip podataka sličan strukturi, jer sadrži

- članove različitog tipa.

Gdje je razlika?

Članovi strukture su

- smješteni u memoriji jedan za drugim.

Za razliku od toga, svi članovi unije

- počinju na istom mjestu u memoriji — na istoj lokaciji,

tj. dijele jedan zajednički dio memorije (na početku), ovisno o veličini članova unije.

Ukupna rezervirana memorija za varijablu tipa unije

- dovoljno je velika da u nju stane "najveći" član unije.

Svrha unije i rad s unijama

Ideja: taj zajednički dio memorije možemo interpretirati

- na razne načine — kao vrijednost(i) različitih tipova.

Zato i ime — unija tipova!

Napomena. Osnovna svrha unije nije

- ušteda memorijskog prostora,

iako se može koristiti i za to.

Osim navedene razlike između unija i struktura u rezervaciji memorije, sve ostalo u C-u je potpuno isto, samo

- umjesto ključne riječi `struct` za strukture,
- pišemo ključnu riječ `union` za unije.

Deklaracija unije

Deklaracija tipa unije ima isti oblik kao i za tip strukture — umjesto `struct`, pišemo `union`.

```
union ime {
    tip_1 ime_1;
    ...   ...
    tip_n ime_n;
};
```

Kao i kod struktura, bolje je koristiti `typedef` za deklaraciju tipa unije.

Varijable `x` i `y` tipa ove unije mogu se deklarirati ovako:

```
union ime x, y;
```

Unija — primjer

Primjer.

```
union podatak {
    int i;
    float x;
} u, *pu;
```

Ovdje su:

- `u.i` i `pu->i` — varijable tipa `int`.
- `u.x` i `pu->x` — varijable tipa `float`.

Član `i` (tipa `int`) i član `x` (tipa `float`)

- počinju na istoj lokaciji u memoriji.

Standardno zauzimaju po 4 bytea, tj. "dijele" isti prostor!

Unija — primjer (nastavak)

Primjer. Uniju možemo iskoristiti za ispis

- "binarnog" (preciznije, heksadecimalnog) oblika prikaza realnog broja tipa `float` u računalu.

```
u.x = 0.234375f;
printf("0.234375 binarno = %x\n", u.i);
```

Zadali (= spremili) smo `u.x` kao `float`, a zatim ista 4 bytea čitamo i pišemo kao `u.i` tipa `int`.

Za pravi "binarni" prikaz možemo iskoristiti algoritam s Prog1

- koji ispisuje binarni prikaz cijelog broja.

Ovdje je taj algoritam realiziran funkcijom `prikaz_int`.

Primjer — binarni prikaz realnog broja

Primjer. Napišite program koji učitava realni broj tipa `double` i piše binarni prikaz tog broja u računalu (v. `p_double.c`).

Broj tipa `double` standardno zauzima 8 byteova = 64 bita. Taj prostor "gledamo" kao

- polje od 2 cijela broja tipa `int` (= 2 "riječi").

Još jedna "sitnica" — bitovi u IEEE prikazu za `double` imaju sljedeći raspored po byteovima (na IA–32):

- 1. byte = bitovi 7 – 0 (donji bitovi),
- 2. byte = bitovi 15 – 8,
- ...
- 8. byte = bitovi 63 – 56 (gornji bitovi).

Binarni prikaz realnog broja — program

Početak programa s globalnom deklaracijom tipa unije za

- jedan `double` i
- polje od 2 `int`-a.

```
#include <stdio.h>

  /* Binarni prikaz realnog broja tipa double. */

typedef union {
    double d;     /* 8 byteova = 64 bita. */
    int i[2];     /* 2 rijeci od po 4 bytea. */
} Double_bits;
```

Binarni prikaz realnog broja — program (nast.)

```
void prikaz_int(int broj)
{
    int nbits, bit, i;
    unsigned mask;

        /* Broj bitova u tipu int. */
    nbits = 8 * sizeof(int);

        /* Pocetna maska ima bit 1
           na najznacajnijem mjestu. */
    mask = 0x1 << nbits - 1;
```

Binarni prikaz realnog broja — program (nast.)

```
    for (i = 1; i <= nbits; ++i) {
            /* Maskiranje odgovarajuceg bita. */
        bit = broj & mask ? 1 : 0;
            /* Ispis i blank nakon svaka 4 bita,
               osim zadnjeg. */
        printf("%d", bit);
        if (i % 4 == 0 && i < nbits) printf(" ");
            /* Pomak maske za 1 bit udesno. */
        mask >>= 1;
    }
    printf("\n");

    return;
}
```

Binarni prikaz realnog broja — program (nast.)

```
void prikaz_double(double d)
{
    Double_bits u;

    u.d = d;

    printf("   1. rijec: ");
    prikaz_int( u.i[0] );
    printf("   2. rijec: ");
    prikaz_int( u.i[1] );

    return;
}
```

Binarni prikaz realnog broja — program (kraj)

```
int main(void)
{
    double d;

    printf(" Upisi realni broj: ");
    scanf("%lf", &d);
    printf(" Prikaz broja %10.3f u racunalu:\n", d);

    prikaz_double(d);

    return 0;
}
```

Binarni prikaz realnog broja — rezultati

Za ulaz `1.0`, dobivamo (v. `p_d_3.out`):

```
Prikaz broja       1.000 u racunalu:
  1. rijec: 0000 0000 0000 0000 0000 0000 0000 0000
  2. rijec: 0011 1111 1111 0000 0000 0000 0000 0000
```

Za ulaz `0.1`, dobivamo (v. `p_d_6.out`):

```
Prikaz broja       0.100 u racunalu:
  1. rijec: 1001 1001 1001 1001 1001 1001 1001 1010
  2. rijec: 0011 1111 1011 1001 1001 1001 1001 1001
```

Obratite pažnju na zadnja 2 bita u prvoj riječi — to je rezultat zaokruživanja mantise (signifikanda)!

Binarni prikaz realnog broja — zadaci

Zadatak. Napišite varijantu ovog programa za realni broj tipa `float` (v. `p_float.c`).

Zadatak. Preuredite oba programa tako da pregledno ispisuju sve bitne dijelove u prikazu realnog broja:

- bit predznaka,
- bitove karakteristike (eksponenta),
- bitove značajnog dijela (signifikanda/mantise).

Dodajte ovom programu i ispis

- vodećeg (skrivenog) bita mantise, ovisno o eksponentu,
- tzv. posebnih vrijednosti `Inf` i `NaN`.

Dodatak: Polja bitova

Polja bitova

Polja bitova (engl. "bit–fields") omogućuju rad s pojedinim bitovima unutar jedne riječi u računalu.

- Jedno polja bitova je član (ili element) strukture ili unije.
- Sprema se u "bloku" susjednih bitova u memoriji računala, a zadaje se brojem bitova koje zauzima.
- Susjedna polja spremaju se u "bloku" susjednih bitova!

Svrha:

- Spremanje 1-bitnih zastavica (engl. flag) u jednu riječ. Na primjer, koriste se u aplikacijama kao što je tablica simbola za kompajler.
- Komunikacija s vanjskim uređajima — treba postaviti ili očitati samo dijelove riječi.

Deklaracija polja bitova

Deklaracija jednog polja bitova, kao člana strukture ili unije, ima sljedeći oblik — iza člana dolazi još znak `:` i broj bitova:

```
struct ime {   /* ili: union ime */
    ...
    tip_polja ime_polja : broj_bitova;
    ...
};
```

Ograničenja (svi detalji ovise o implementaciji):

- `tip_polja` mora biti: `int`, `unsigned int` ili `signed int`.
- `ime_polja` je identifikator (kao i za ostale članove), a
- `broj_bitova` mora biti nenegativan cijeli broj (nula ima posebno značenje i onda se `ime_polja` smije ispustiti).

Svrha = uzastopna polja bitova

Ovako deklarirani član `ime_polja` predstavlja jedno

- polje uzastopnih bitova u računalu, duljine `broj_bitova`.

Stvarna svrha je u deklaraciji uzastopnih članova tog oblika!

Razlika između "običnih" članova i polja bitova:

- "Obični" član započinje u novoj riječi i zauzima cijeli broj riječi, ovisno o tzv. "poravnanju" riječi (engl. "byte/word/memory alignment").
- Susjedno deklarirana polja bitova spremaju se u bloku uzastopnih bitova, bez "rupa", tj. nastavljaju se jedan do drugog — može i unutar iste riječi, i baš to je svrha!

Poredak spremanja ($\leftarrow$ ili $\rightarrow$ u riječi) i eventualni "prijelom" sljedećih članova između riječi — ovisi o implementaciji!

Deklaracija polja bitova — primjer

Primjer.

```
struct primjer {
    unsigned int a : 1;
    unsigned int b : 3;
    unsigned int c : 2;
    unsigned int d : 1;
};
struct primjer v;
...
if (v.a == 1) ...
v.c = STATIC;
```

Deklaracija polja bitova — primjer (nastavak)

- Prva deklaracija definira strukturu sastavljenu iz četiri uzastopna polja bitova: `a`, `b`, `c` i `d`.
- Ta polja, redom, imaju duljinu `1`, `3`, `2` i `1` bit. Prema tome, ukupno zauzimaju `7` bitova i spremaju se u bloku.
- Poredak tih bitova unutar jedne riječi u računalu ovisi o implementaciji.
- Pojedine članove, koji su polje bitova, dohvaćamo na isti način kao i "obične" članove strukture — `v.a`, `v.b`, itd.
- Ako broj bitova, deklariran u polju bitova, nadmašuje duljinu jedne riječi u računalu, za pamćenje tog polja bit će upotrebljeno više riječi.
- Isto vrijedi i za "blok" uzastopnih polja bitova.

Polja bitova — primjer

Primjer. Program koji upotrebljava uzastopna polja bitova (v. `bf_3.c`):

```
#include <stdio.h>

int main(void)
{
    struct primjer {
        unsigned int a : 5;
        unsigned int b : 5;
        unsigned int c : 5;
        unsigned int d : 5;
    };
    struct primjer v = {1, 2, 3, 4};
```

Polja bitova — primjer (nastavak)

```
    printf(" v.a = %d, v.b = %d, v.c = %d,"
           " v.d = %d\n", v.a, v.b, v.c, v.d);
    printf(" sizeof(v) = %u\n", sizeof(v));
    return 0;
}
```

Izlaz:

```
v.a = 1, v.b = 2, v.c = 3, v.d = 4
sizeof(v) = 4
```

Na IA–32, cijela struktura `v` zauzima jednu riječ = 4 bytea.

Kod ispisa, vrijednosti članova (polja bit.) se pretvaraju u `int` (standardna konverzija kratkih cjelobrojnih tipova za `printf`).

Neimenovano polje bitova u bloku

Raspored bitova unutar riječi može se kontrolirati korištenjem

- neimenovanih članova pozitivne duljine, unutar bloka uzastopnih polja bitova.

Primjer.

```
struct primjer {
    unsigned int a : 5;
    unsigned int b : 5;
    unsigned int   : 5;  /* Razmak 5 bitova. */
    unsigned int c : 5;
};
struct primjer v;
```

Neimenovano polje bitova u bloku (nastavak)

Neimenovani član duljine 0 bitova označava da

- sljedeće polje iz bloka treba smjestiti u sljedeću riječ.

Primjer.

```
#include <stdio.h>

int main(void) {
    struct primjer {
        unsigned int a : 5;
        unsigned int b : 5;
        unsigned int   : 0;  /* Idi u novu rijec. */
        unsigned int c : 5;
    };
    struct primjer v = {1, 2, 3};
```

Neimenovano polje bitova u bloku (nastavak)

```
    printf(" v.a = %d, v.b = %d, v.c = %d\n",
           v.a, v.b, v.c);
    printf(" sizeof(v) = %u\n", sizeof(v));
    return 0;
}
```

Izlaz:

```
v.a = 1, v.b = 2, v.c = 3
sizeof(v) = 8
```

Na IA–32, struktura `v` sad zauzima dvije riječi = 8 byteova.

Primjeri za poredak spremanja uzastopnih polja bitova unutar pojedinih riječi: `bf_pos_1.c`, `bf_pos_2.c`, `bf_pos_3.c`.