

# Programiranje 2

Predavanje 07 - pokazivač na funkciju, složene deklaracije,  
strukture, unije, polja bitova

Matej Mihelčić

Prirodoslovno-matematički fakultet  
Matematički odsjek

16. travnja 2023.



## Složene deklaracije - primjeri

```
1 int (*p(char *a))[10]; /* funk. uzima ptr na char
2 i vraca ptr na polje od 10 elem. tipa int */
3
4 int p(char (*a)[8]); /* funk. uzima ptr na polje
5 od 8 char i vraca int */
6
7 int (*p)(char (*a)[8]); /* ptr na funk. koja
8 uzima ptr na polje od 8 char i vraca int */
9
10 int *(*p)(char (*a)[8]); /* ptr na funk. koja
11 uzima ptr na polje od 8 char i vraca ptr na int*/
12
13 int *(*p[10])(char *a); /* polje od 10 ptr na
14 funk. koja uzima ptr na char i vraca ptr na int*/
```

## 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:

```
1 typedef tip_podataka novo_ime_za_tip_podataka;
```

`novo_ime_za_tip_podataka` postaje sinonim za `tip_podataka` i smije se tako koristiti u svim kasnijim deklaracijama (ima isto značenje).

Deklaracijom:

```
1 typedef double Masa;
```

identifikator `Masa` postaje sinonim za `double`.

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

```
1 Masa m1, m2, *pm1; //pm1 pokazivac na double
2 Masa elementi[10]; //polje od 10 el. tipa double
```

## Svrha deklaracije tipova

Svrha jednostavnih deklaracija:

- Lakše razumijevanje (čitanje) koda
- Lakše dokumentiranje programa

Korisno kod složenijih tipova podataka kada u programu koristimo **hijerarhiju** tipova. Primjer su tipovi koji se grade jedni iz drugih, npr. samoreferencirajuće strukture (vezane liste, stabla itd.).

```
1 typedef int Metri, Kilometri;
2 Metri duljina, sirina;
3 Kilometri udaljenost;
```

Ideja: ime tipa sugerira jedinice u kojima su izražene određene vrijednosti. Veće prednosti se mogu dobiti definiranjem složenijih tipova.

## Složenije typedef deklaracije

Deklaracija imena za složeniji tip počinje s `typedef`, a dalje ima isti oblik kao i deklaracija varijable tog imena i tog tipa. U tom slučaju ime nije varijabla tog tipa već sinonim za taj tip.

```
1 #define n 10
2 typedef double skalar;
3 typedef skalar vektor[n];
4 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. sinonim za dvodimenzionalno polje **skalara**, sinonim za tip `double [n] [n]` = `double [10] [10]`.

## typedef i polja

Funkciju za računanje umnoška  $y = Ax$ , kvadratne matrice  $A$  i vektora  $x$  možemo pisati:

```
1 void prod_mat_vek(matrica A, vektor x, vektor y){  
2     int i, j;  
3  
4     for (i = 0; i < n; ++i) {  
5         y[i] = 0.0;  
6         for (j = 0; j < n; ++j)  
7             y[i] += A[i][j] * x[j];  
8     }  
9 }
```

Izmijenite funkciju tako da  $n$  bude argumenat funkcije, a ne konstanta 10.

## Deklaracije tipova - string, pokazivači

Kod rada sa **stringovima** možemo uvesti oznaku:

```
1 typedef char *string;
```

Ovdje je **string** sinonim za pokazivač na **char** (tip **char \***).

Funkcija **strcmp** za **uspoređivanje stringova** može se deklarirati:

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

Pokazivač na **double** možemo nazvati **Pdouble**.

```
1 typedef double *Pdouble;
```

Sada možemo pisati:

```
1 Pdouble px; /* = double *px */
2 void f(Pdouble, Pdouble);
3 /* = void f(double *, double *); */
4 px = (Pdouble) malloc(100 * sizeof(double));
```

## typedef i deklaracija funkcija

typedef koristimo za kraće zapisivanje složenih deklaracija.

Pokazivač na funkciju:

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

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

Umjesto deklaracije:

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

možemo pisati:

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

## Operatori u deklaracijama i `typedef`

U deklaracijama varijabli i tipova dozvoljeno je koristiti neke operatore (kada to ima smisla). Npr. `*` (dereferenciranje), `[]` (polje), `()` (funkcija). Navedeni operatori djeluju samo na jedan pripadni argument, a ne na sve navedene.

U sljedećoj deklaraciji varijabli `a` i `b`:

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

operator `*` djeluje samo na identifikator `a`, a ne na tip `int`. Zato je `a` pokazivač na `int`, a `b` je varijabla tipa `int`. Operator u deklaraciji ima viši prioritet od navođenja tipa (djeluje na jednom argumentu, varijabli).

Kod deklaracije argumenata funkcije tip se navodi **za svaki argument posebno**.

## Operatori u deklaracijama i `typedef`

Ukoliko želimo omogućiti deklaracije kojima i varijable *a* i *b* postaju pokazivači na int, treba pisati:

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

Umjesto toga, možemo: deklarirati tip za pokazivač na int.

```
1 typedef int *pok_int; // deklarirati tip za int*
2 pok_int a, b; // deklarirati varijable tipa int*
```

Za polja *a* i *b* duljine 10 trebamo **zasebno** navesti duljinu.

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

Možemo uvesti i novi tip:

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

Identifikator *polje* je ime tipa za `int[10]`.

# 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. Elementi (članovi) strukture mogu, ali ne moraju, biti različitog tipa i svaki element ima svoje posebno ime. U deklaraciji strukture moramo navesti ime i tip svakog člana.

Tip strukture možemo deklarirati na dva načina:

```
1 struct ime {  
2     tip_1 ime_1;  
3     tip_2 ime_2;  
4     ...  
5     tip_n ime_n;  
6 };
```

struct je rezervirana riječ, a ime je ime strukture. Stvarni tip strukture je struct ime. Unutar vitičastih zagrada popisani su članovi strukture.

## Definicija varijabli tipa strukture — bez `typedef`

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:

```
1 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

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

```
1 struct tocka {  
2     int x;  
3     int y;  
4 };
```

Varijable tipa strukture tocka definiramo tako da:

Nakon gornje deklaracije strukture tocka (kao tipa) pišemo:

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

Ili pišemo:

```
1 struct tocka {  
2     int x;  
3     int y;  
4 } t1, t2;
```

## Deklaracija strukture preko typedef

Postoji i bolji način deklaracije tipa strukture, koji olakšava definiciju varijabli tog tipa, koristeći `typedef`. **Prednost:** ne treba stalno navoditi riječ `struct` u deklaracijama varijabli.

```
1 typedef struct ime {  
2     tip_1 ime_1;  
3     tip_2 ime_2;  
4     ...  
5     tip_n ime_n;  
6 } ime_tipa;
```

Na kraju deklaracije, cijelom tipu strukture dajemo ime `ime_tipa`. Stvarni tip strukture je i `ime_tipa`, kao sinonim za `struct ime`. Ostalo je kao i ranije.

## Definicija varijabli tipa strukture - uz typedef

Ime strukture (odmah iza struct) smijemo i ispuštiti ako ga nigdje nećemo koristiti. Ukoliko pišemo ime strukture, ono mora biti različito od svih ostalih imena (identifikatora) pa i od `ime_tipa`. Uobičajeno je kao ime strukture koristiti `_ime_tipa` (npr. `_osoba`). Varijable tog tipa općenito definiramo:

```
1 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 točka

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

```
1 typedef struct {
2     int x;
3     int y;
4 } Tocka;
5 ...
6 Tocka t1, t2, *pt1;
```

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

Nismo navodili ime strukture iza struct, jer ga nećemo koristiti.

## Inicijalizacija strukture

Varijablu tipa strukture možemo inicijalizirati pri definiciji:

```
1 mem_klasa struct ime var = {v_1, ..., v_n};  
2 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.

Ako je definirana struktura:

```
1 struct racun {  
2     int broj_racuna;  
3     char ime[80];  
4     float stanje;  
5 }
```

Varijablu **kupac** možemo inicijalizirati:

```
1 struct racun kupac = { 12, "Ivo Ban", -200.00f};
```

## Inicijalizacija polja struktura

Slično možemo inicijalizirati i čitavo polje struktura:

```
1 struct racun kupci[] = {  
2     15, "Goga\u0107Markic", 45.00f,  
3     18, "Josip\u0107Lovric", -23.00f,  
4     23, "Martina\u0107Knezic", 0.00f };
```

## Operator tocka - pristup clanu 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:

1 var . memb

vrijednost člana memb u strukturi var.

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 prioritetu grupu (primarni operatori) i ima asocijativnost  $L \rightarrow D$ . Zbog najvišeg prioriteta vrijedi:

$$++ \text{varijabla.clan} \Leftrightarrow ++ (\text{varijabla.clan})$$

$$\&\text{varijabla.clan} \Leftrightarrow \&(\text{varijabla.clan})$$

Član strukture (kao i element polja) smije pisati na lijevoj strani naredbe pridruživanja.

Pristup članovima strukture:

```
1 struct tocka {  
2     int x; /* prvi clan strukture */  
3     int y; /* drugi clan strukture */  
4 };  
5 struct tocka ishodiste;
```

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

Za strukturu:

```
1 struct racun {  
2     int broj_racuna;  
3     char ime[80];  
4     float stanje;  
5 } kupac = { 1234, "Pero\u0107Peric", -1234.00f };
```

vrijedi:

```
1 kupac.broj_racuna = 1234,  
2 kupac.ime = "Pero\u0107Peric",  
3 kupac.stanje = -1234.00f.
```

## Struktura kao član druge strukture

Strukture mogu sadržavati druge strukture kao članove.

Pravokutnik paralelan koordinatnim osima možemo zadati parom dijagonalno suprotnih vrhova, npr. donjim lijevim (pt1) i gornjim desnim (pt2). Vrhovi su točke.

```
1 struct pravokutnik {  
2     struct tocka pt1; /* ili Tocka pt1; */  
3     struct tocka pt2; /* ili Tocka pt2; */  
4 };
```

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 (nazovimo ga `clan`) pristupa izrazom:

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

Koristi se asocijativnost  $L \rightarrow D$  za **primarne operatore točka (.)** i **indeksiranje polja ([])**. Operator `.` se primjeni na varijabli `varijabla` i dohvaća se član `clan`. Operator `[]` se primjenjuje na polju `clan` i dohvaća se element s indeksom koji se dobije evaluacijom izraza `izraz` u polju `clan`.

```
1 typedef struct {
2     int broj_racuna;
3     char ime[80];
4     float stanje;
5 } Racun;
6 Racun kupac = { 12, "Ivo Ban", 10.00f };
7 if (kupac.ime[0] == 'I') puts(kupac.ime);
```



## Polje struktura

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

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

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

```
1 struct tocka {  
2     int x;  
3     int y;  
4 } vrhovi[1024]; /* Polje tocka. */  
5 ...  
6 if (vrhovi[17].x == vrhovi[17].y) ...
```

## Strukture - operacije i funkcije

Dozvoljene operacije nad strukturu su: a) pridruživanje, b) uzimanje adrese, primjena operatora sizeof. Nije dozvoljeno **uspoređivanje** cijelih struktura.

Struktura može biti **argument** funkcije. Funkcija dobiva **kopiju** cijele strukture. Funkcija može **vratiti** strukturu. Ponašanje je kao kod varijabli osnovnog tipa, različito od polja.

```
1 typedef struct {
2     int x;
3     int y;
4 } Tocka;
5 Tocka t, ishodiste = {0, 0}, t1 = {1, 7};
6 //argumenti funkcije tipa Tocka
7 Tocka suma(Tocka p1, Tocka p2) {
8     p1.x += p2.x;
9     p1.y += p2.y;
10    return p1; }//povratna vrijednost tipa Tocka
```

## Strukture i funkcije

```
1 int main(void) {
2     /* Dodjeljivanje struktura:
3         t i ishodiste moraju biti istog tipa */
4
5     t = ishodiste;
6     printf("Velicina u %u byteova\n", sizeof(t));
7
8     /* Zbroj tocka, rezultat je tocka. */
9     t1 = suma(t1, t1);
10    printf("t1=(%d,%d)\n", t1.x, t1.y);
11    return 0;
12 }
```

Struktura i funkcija za rad s kompleksnim brojevima.

```
1 typedef struct {
2     double re; /* ili x */
3     double im; /* ili y */
4 } complex;
5
6 /* Napomena: cabs vec postoji u <math.h>! */
7 double zabs(complex a) {
8     return sqrt( a.re * a.re + a.im * a.im );
9 }
```

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.

```
1 struct tocka {
2     int x;
3     int y;
4 } p1, *pp1;
5 ...
6 pp1 = &p1;
7 (*pp1).x = 13; /* Zagrade su NUZNE! */
8 (*pp1).y = 27;
9 *pp1.x = 13; /* GRESKA!
10 *pp1.x je isto sto i *(pp1.x) */
```

## Operator strelica ( $\rightarrow$ )

Primarni operator strelica ( $\rightarrow$ ) omogućava jednostavno dohvaćanje člana strukture preko pokazivača na tu strukturu. Asocijativnost operatorka  $\rightarrow$  je  $L \rightarrow D$ . Ako je *ptvar* pokazivač na strukturu, a *clan* je neki član te strukture, onda je:

$$ptvar \rightarrow clan \Leftrightarrow (*ptvar).clan$$

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

## Složeni izrazi

Pristup koordinatama vrhova pravokutnika r izravno i preko pokazivača pr.

```
1 struct pravokutnik {  
2     struct tocka pt1;  
3     struct tocka pt2;  
4 } r, *pr = &r;
```

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

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

## Unija

Unija je složeni tip podataka sličan strukturi, jer sadrži članove različitog tipa. Za razliku od strukture kod koje su članovi smješteni u memoriji jedan za drugim, kod unije svi članovi počinju **na istom mjestu u memoriji** - na istoj lokaciji. Svi članovi dijele zajednički dio memorije ovisno o veličini članova unije. Ukupna rezervirana memorija za varijablu tipa unije dovoljno je velika da u nju stane najveći član unije.

**Ideja:** zajednički dio memorije možemo **interpretirati na razne načine** - kao **vrijednosti različitih tipova**. Od tuda dolazi i ime **unija tipova**.

Osnovna svrha unije nije ušteda memoriskog prostora, iako se može koristiti i za to.

Osim navedenih razlika u rezervaciji memorije, nema drugih razlika između strukture i unije u C-u. 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.

```
1 union ime {  
2     tip_1 ime_1;  
3     ... ...  
4     tip_n ime_n;  
5 };
```

Kao i kod struktura, bolje je koristiti typedef za deklaraciju tipa unije. Varijable x i y ove unije mogu se deklarirati:

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

## Unija - primjer

```
1 union podatak {  
2     int i;  
3     float x;  
4 } u, *pu;
```

u.i i pu->i su varijable tipa int, a u.x i pu->x tipa float.

Članovi i (tipa int) i x (tipa float) počinju na istoj lokaciji u memoriji. Standardno zauzimaju po 4 bytea, tj. dijele isti prostor.

Uniju možemo iskoristiti za ispis heksadecimalnog oblika prikaza realnog broja tipa float u računalu.

```
1 u.x = 0.234375f;  
2 printf("0.234375\u00b7binarno\u00b7=%x\n", u.i);
```

Zadamo u.x kao float, te ista 4 byte-a čitamo i pišemo kao u.i tipa int. Pravi binarni prikaz možemo dobiti algoritmom koji ispisuje binarni prikaz cijelog broja (Prog1).

## Primjer - binarni prikaz realnog broja

Pišemo program koji učitava realni broj tipa `double` i piše **binarni** prikaz tog broja u računalu.

Broj tipa `double` standardno zauzima 8 byteova = 64 bita. Taj prostor možemo reprezentirati kao polje od 2 cijela broja tipa `int` (2 riječi).

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 (donji bitovi),
- ...
- 8. byte = bitovi 63 – 56 (gornji bitovi).

## Primjer - binarni prikaz realnog broja

Globalno deklariramo tip unije za jedan double i polje od 2 int-a.

```
1 #include <stdio.h>
2 /* Binarni prikaz realnog broja tipa double. */
3 typedef union {
4     double d; /* 8 byteova = 64 bita. */
5     int i[2]; /* 2 rijeci od po 4 bytea. */
6 } Double_bits;
7
8 //funkcija za prikaz cijelog broja
9 void prikaz_int(int broj){
10     int nbits, bit, i;
11     unsigned mask;
12 /* Broj bitova u tipu int. */
13     nbits = 8 * sizeof(int);
14 /* Pocetna maska ima bit 1 na najznacajnijem */
15     mask = 0x1 << nbits - 1; //mjestu.
```

## Primjer - binarni prikaz realnog broja

```
16     for (i = 1; i <= nbits; ++i) {
17         /* Maskiranje odgovarajućeg bita. */
18         bit = broj & mask ? 1 : 0;
19         /* Ispis i blank nakon svaka 4 bita,
20          osim zadnjeg. */
21         printf("%d", bit);
22         if (i % 4 == 0 && i < nbits) printf(" ");
23         /* Pomak maske za 1 bit udesno. */
24         mask >>= 1;
25     printf("\n");
26
27     return;
28 }
```

## Primjer - binarni prikaz realnog broja

```
29 void prikaz_double(double d){
30     Double_bits u;
31
32     u.d = d;
33
34     printf("1.rijec:");
35     prikaz_int( u.i[0] );
36     printf("2.rijec:");
37     prikaz_int( u.i[1] );
38
39     return;
40 }
```

## Primjer - binarni prikaz realnog broja

```
41 int main(void){  
42  
43     double d;  
44  
45     printf("Upisi realni broj:");  
46     scanf("%lf", &d);  
47     printf("Prikaz broja %10.3f u racunalu:\n", d);  
48     prikaz_double(d);  
49  
50     return 0;  
51 }
```

Za ulaz 1.0 dobivamo:

Prikaz broja 1.000 u racunalu:

- rijec: 0000 0000 0000 0000 0000 0000 0000 0000
- rijec: 0011 1111 1111 0000 0000 0000 0000 0000

Za ulaz 0.1 dobivamo:

Prikaz broja 0.100 u racunalu:

- rijec: 1001 1001 1001 1001 1001 1001 1001 1010
- rijec: 0011 1111 1011 1001 1001 1001 1001 1001

Primijetite da je došlo do zaokruživanja mantise (signifikanda) u zadnja 2 bita prve riječi.

## Binarni prikaz realnog broja — zadaci

Napišite varijantu ovog programa za realni broj tipa float.

Preuređite 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) bit mantise, ovisno o eksponentu
- posebnih vrijednosti Inf i NaN

**Polja bitova** (eng. *bit-fields*) omogućavaju rad s pojedinim bitovima unutar jedne riječi u računalu. Jedno polje 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. Npr. u aplikacijama kao što je tablica simbola za prevodioc. Komunikacija s vanjskim uređajima kada 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 znak : i broj bitova:

```
1 struct ime { /* ili: union ime */
2     ...
3     tip_polja ime_polja : broj_bitova;
4     ...
5 }
```

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)
- **broj\_bitova** mora biti **nengativan cijeli broj** (**nula** ima posebno značenje, tada se **ime\_polja** smije ispuštiti).

Gore opisani način deklariranja člana `ime_polja` predstavlja jedno polje uzastopnih bitova u računalu, duljine `broj_bitova`. Stvarna svrha je u deklaraciji uzastopnih članova tog oblika.

Kod običnih članova, svaki član započinje u novoj riječi i zauzima cijeli broj riječi, ovisno o poravnanju riječi (eng. *byte/word/memory alignment*). Susjedno deklarirana polja bitova spremaju se u bloku uzastopnih bitova, bez praznina, tj. nastavljaju se jedan do drugog (potencijalno i unutar iste riječi).

Poredak spremanja  $\leftarrow$  ili *rightarrow* u riječi i eventualni prijelom sljedećih članova između riječi ovise o implementaciji.

## Deklaracija polja bitova - primjeri

```
1 struct primjer {  
2     unsigned int a : 1;  
3     unsigned int b : 3;  
4     unsigned int c : 2;  
5     unsigned int d : 1;  
6 };  
7 struct primjer v;  
8 ...  
9 if (v.a == 1) ...  
10    v.c = 2;
```

Deklariramo strukturu sastavljenu od četiri uzastopna polja bitova: a, b, c i d. Ta polje imaju duljinu 1, 3, 2 i 1 bit (ukupno zauzimaju 7 bitova - spremaju se u bloku). Poredak 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.).

## Deklaracija polja bitova - primjeri

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.

```
1 #include <stdio.h>
2 int main(void){
3     struct primjer {
4         unsigned int a : 5;
5         unsigned int b : 5;
6         unsigned int c : 5;
7         unsigned int d : 5;
8     };
9     struct primjer v = {1, 2, 3, 4};
10    printf("v.a=%d, v.b=%d, v.c=%d, "
11           "v.d=%d\n", v.a, v.b, v.c, v.d);
12    printf("sizeof(v)=%u\n", sizeof(v));
13    return 0;
}
```

## Deklaracija polja bitova - primjeri

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.

```
1 struct primjer {  
2     unsigned int a : 5;  
3     unsigned int b : 5;  
4     unsigned int : 5; /* Razmak 5 bitova. */  
5     unsigned int c : 5;  
6 };  
7 struct primjer v;
```

Neimenovani član duljine 0 bitova označava da sljedeće polje iz bloka treba smjestiti u sljedeću riječ.

## Neimenovano polje bitova u bloku

```
1 #include <stdio.h>
2
3 int main(void) {
4     struct primjer {
5         unsigned int a : 5;
6         unsigned int b : 5;
7         unsigned int : 0; /* Idi u novu rijec. */
8         unsigned int c : 5;
9     };
10    struct primjer v = {1, 2, 3};
11    printf("v.a=%d, v.b=%d, v.c=%d\n",
12           v.a, v.b, v.c);
13    printf("sizeof(v)=%u\n", sizeof(v));
14    return 0;
15 }
```

## Neimenovano polje bitova u bloku

Izlaz;

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

Na IA-32, struktura *v* sad zauzima dvije riječi (8 byteova).