Definicija vlastitih tipova podataka

`typedef`

typedef

- Korištenjem ključne riječi `typedef` postojećim tipovima podataka dajemo nova imena (ne kreiramo nove tipove!).
- Općenito, `typedef` ima oblik:

  `typedef` tip_podatka novo_ime_za_tip_podatka

Primjer:

```
typedef double Masa;
```

Masa postaje sinonim za `double`. Nakon toga, varijable tipa `double` možemo deklarirati kao:

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

Primjeri:

```
typedef int metri, kilometri;
metri duljina, sirina;

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

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

`typedef` i pokazivači

Primjer: Pokazivač na `double` nazvat ćemo Pdouble.

```
typedef double *Pdouble;
```

Pdouble postaje pokazivač na `double`:

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

`typedef` i deklaracija funkcije

- Korištenjem ključne riječi `typedef` možemo kraće zapisati kompleksne deklaracije.

Primjer:

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

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

```
void f(double x, int (*g)(char *, char *)){ ...
    /* = void f(double x, PF g){ ... */
```

Strukture

Deklaracija strukture

- Polja grupiraju veći broj podataka istog tipa.
- Strukture služe grupiranju više podataka različitih tipova.

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

Primjer:

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

tocka t1, t2;
```

t1 i t2 su varijable tipa tocka.

Definicija varijabli tipa strukture

- Općenito, var_1, var_2, …, var_n su varijable tipa *ime*:

  `mem_klasa` *ime* var_1, var_2, …, var_n;

- Strukture možemo deklarirati i bez ključne riječi `typedef`:

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

Definicija varijabli tipa strukture (2)

- U tom slučaju, varijable var_1, var_2, …, var_n tipa *ime* definiramo na sljedeći način:

  `mem_klasa struct` *ime* var_1, var_2, …, var_n;

Primjer:

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

ili

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

Inicijalizacija strukture

`mem_klasa [struct]` *ime* varijabla={v_1, v_2, …, v_n};

- Konstante v_i pridružuju se odgovarajućim članovima strukture.

Primjer:

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

struct racun kupac = {1234, "Ivica Rastrosni", -23456.00};
```

Inicijalizacija polja struktura

- Slično se može inicijalizirati i polje struktura:

```
struct racun kupci[]={2211, "Ana", 4567.00,
                      1326, "Marko", -567.00,
                      133, "Maja", 0.00};
```

Strukura definirana pomoću strukture

- Strukture mogu sadržavati druge strukture kao članove.

Primjer: Pravokutnik čije su stranice paralelne koordinatnim osima je određen donjim lijevim (pt1) i gornjim desnim (pt2) vrhom.

```
struct pravokutnik{
    struct tocka pt1;
    struct tocka pt2;
};
```

Deklaracija strukture tocka mora prethoditi deklaraciji strukture pravokutnik.

Rad sa strukturama

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

- *ishodiste* je varijabla tipa `struct` tocka.
- *ishodiste.x* je prva komponenta varijable *ishodiste*.
- *ishodiste.y* je druga komponenta varijable *ishodiste*.

Primjer:

```
struct racun{
    int broj_racuna;
    char ime[80];
    double stanje;
} kupac = {1234, "Ivica Rastrosni", -23456.00};
```

Tada je:

kupac.broj_racuna = 1234,

kupac.ime = "Ivica Rastrosni",

kupac.stanje = -23456.00.

Općenito:

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

  var.*memb*

  član *memb* u strukturi *var*.
- Operator točka (.) separira ime varijable i ime člana strukture. Spada u najvišu prioritetnu grupu i ima asocijativnost slijeva nadesno.

- ++varijabla.clan; ⇔ ++(varijabla.clan);
- &varijabla.clan; ⇔ &(varijabla.clan);

Polje kao član strukture

- Kada struktura sadrži polje kao član strukture, onda se elementima polja pristupa izrazom
  varijabla.clan[izraz]

Primjer:

```
typedef struct {
    int broj_racuna;
    char ime[80];
    double stanje;
} racun;
racun kupac = {1234, "Ivica Rastrosni", -23456.00};
……
if (kupac.ime[0] == 'I') puts(kupac.ime);
```

Polje struktura

- Ako imamo polje struktura, onda za pojedini element polja članu strukture pristupamo izrazom

  polje[izraz].clan

Primjer:

```
struct tocka{
    int x;
    int y;
} vrhovi[1000];
……
if (vrhovi[13].x == vrhovi[13].y) …
```

Polje struktura (2)

Primjer:

```
struct Prva_godina{
    char ime[15];
    char prezime[15];
    int ocjene[6];
} student[100];
.....
printf ("Ocjena: %d\n", student[5].ocjene[3]);
```

Operacije nad strukturom kao cjelinom

- Pridruživanje
- Uzimanje adrese, primjena operatora `sizeof`
- Struktura može biti argument funkcije
- Funkcija može vratiti strukturu.

- Napomena. Nije dozvoljeno uspoređivanje struktura.

Pridruživanje i operator `sizeof`

```
struct tocka{
    int x;
    int y;
} t, ishodiste = {0,0};
……
t = ishodiste;        /* t i ishodište moraju biti
                         istog tipa */
printf("%d\n", sizeof(t));
```

Primjer: Funkcija `zabs`

```
typedef struct {
    double re;
    double im;
} compl;

double zabs(compl a) {
    return sqrt(a.re * a.re + a.im * a.im);
}
```

- Zadatak: Napraviti biblioteku funkcija za osnovne operacije s kompleksnim brojevima.

Primjer: Funkcija `cabs`

- `double cabs (double complex` *z);*

```
#include <math.h>

int main(){
   struct complex a;

   a.x = 1.0;
   a.y = 12.0;
   printf("%f\n", cabs(a));

   return 0;
}
```

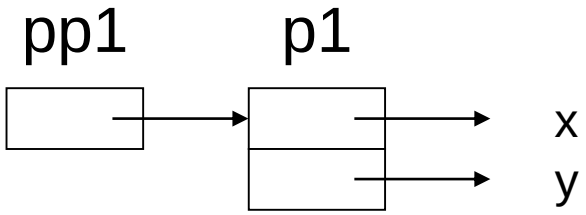
Strukture i funkcije

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

- Pokazivač na strukturu definira se kao i pokazivač na osnovne tipove varijabli.

Strukture i pokazivači

```
struct tocka{
    int x;
    int y;
} p1, *pp1;
……
pp1 = &p1;
(*pp1).x = 16;
(*pp1).y = 27;
printf("%d\n", *pp1);    /* Što će se ispisati? */
*pp1 = 13;  /* Greška! */
```



Operator ->

- Pomoću primarnog operatora -> možemo jednostavno dohvatiti član strukture korištenjem pokazivača.
- Asocijativnost operatora -> je L→D.
- Ako je *ptvar* pokazivač na strukturu, a *clan* jedan član strukture, onda je:

$$ptvar \texttt{->} clan \Leftrightarrow (*ptvar).clan$$

Primjer:

```
struct tocka p1, *pp1 = &p1;
pp1->x = 16;
pp1->y = 27;
```

Ekvivalentni izrazi

- Neka je

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

- Tada su sljedeći izrazi ekvivalentni:

```
r.pt1.x
pr -> pt1.x          /* Operatori . i -> imaju isti prioritet */
(r.pt1).x
(pr -> pt1).x                /* i asocijativnost L→D. */
```

Unije

Unija

- Unija, kao i struktura, može sadržati članove različitog tipa.
- Kod unije svi članovi dijele istu memorijsku lokaciju (počinju na istom mjestu u memoriji).
- Memorijska lokacija bit će dovoljno velika da u nju stane najveći član unije.
- Sintaksa:

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

Definicija varijabli

- Varijable x i y tipa *ime* mogu se deklarirati na sljedeći način:

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

Primjer:

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

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

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

Binarni prikaz realnog broja

- Ispis binarnog prikaza (u računalu) realnog broja tipa `double`.

```
#include <stdio.h>

typedef union {
   double d;
   int i[2];
} Double_bits;
```

Binarni prikaz realnog broja (2)

```
void prikaz_int(int broj)
{
   int bit, i;
   unsigned mask;
   mask = 0x1 << 31;

   for (i = 1; i <= 32; ++i) {
      bit = broj & mask ? 1 : 0;
      printf("%d", bit);
      if (i % 4 == 0) printf(" ");
      mask >>= 1;
   }
```

Binarni prikaz realnog broja (3)

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

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 (4)

- Ispis binarnog prikaza (u računalu) realnog broja tipa `double`.

```
int main(void)
{
   double d;
   printf(" Upiši realni broj: ");
   scanf("%lf", &d);
   printf(" Prikaz broja %10.3f u računalu:\n", d);
   prikaz_double(d);
   return 0;
}
```

Polja bitova

Polja bitova

- Polja bitova (*bit–fields*) omogućavaju rad s pojedinim bitovima unutar jedne riječi u računalu.
- Polje bitova je skup susjednih bitova u sklopu jedne memorijske jedinice (riječi).
- Može se proširiti na više susjednih riječi – spremanje „u bloku”.
- Upotreba:
  - spremanje 1-bitnih zastavica (*flag*) u jednu riječ
  - komunikacija s vanjskim uređajima – treba postaviti ili očitati samo dijelove riječi.

Deklaracija polja bitova

- Opći oblik deklaracije polja bitova sličan je deklaraciji strukture:

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

- Ograničenja (svi detalji ovise o implementaciji):
  - tip_polja mora biti: `int` (može i `unsigned`)
  - ime_polja je identifikator
  - broj_bitova mora biti nenegativan cijeli broj
  - nula ima posebno značenje.

Polja bitova - primjer

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

Neimenovani članovi polja bitova

- Raspored polja unutar riječi može se kontrolirati korištenjem neimenovanih članova pozitivne duljine unutar polja.

- Primjer:

```
struct primjer {
    unsigned int prva_cetiri    : 4;
    unsigned int                : 24 ;
    unsigned int zadnja_cetiri : 4;
};
```

Neimenovani članovi polja … (2)

- Primjer. Neimenovani član duljine 0 bitova „tjera” prevoditelj da sljedeće polje smjesti u sljedeću računalnu riječ.

- Primjer:

```
struct primjer {
    unsigned int a : 5;
    unsigned int b : 5;
    unsigned int   : 0;
    unsigned int d : 5;
}; struct primjer v = {1, 2, 3};
printf(" sizeof(v) = %u\n", sizeof(v));
```