Programiranje (C)
11. predavanje

Saša Singer

singer@math.hr
web.math.hr/~singer
www.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Pokazivači (zadnji dio):
  - Pokazivači i polja.
  - Dinamička alokacija memorije.
  - Pokazivač na funkciju.
  - Argumenti komandne linije.
  - Složene deklaracije.
- Strukture (prvi dio):
  - Deklaracija strukture.
  - Rad sa strukturama. Operator točka.
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (->).
  - Samoreferencirajuće strukture. Vezane liste.

Pokazivači (nastavak)

Pokazivači i jednodimenzionalna polja

- Ime jednodimenzionalnog polja je konstantan pokazivač na prvi element polja.

Primjer:

```
char *px, x[128];

px = &x[0];          /* Isto kao px = x; */
*(px + 3) = 'd';     /* Isto kao x[3] = 'd'; */
px++;                /* Isto kao px = &x[1]; */
*(x + 1) = 'b';      /* Isto kao x[1] = 'b'; */
x++;                 /* GRESKA - konst. pointer */
```

Pokazivači i stringovi

Pokazivač na `char` možemo inicijalizirati stringom.

Primjer:

```
char *pmessage;
...
pmessage = "Sad je vrijeme";
```

Ovo je zaista operacija s pokazivačima. String konstanta je spremljena negdje u memoriji i ima svoju adresu.

Pokazivači i stringovi (nastavak)

Uočiti bitnu razliku između definicija:

```
char amessage[] = "Poruka";  /* polje */
char *pmessage = "Poruka";  /* pointer */
```

- `amessage` je polje od 7 znakova i možemo mu promijeniti sadržaj (`amessage` konstantni pointer).
- `pmessage` je pointer, inicijaliziran tako da pokazuje na string konstantu.
- Tom pointeru kasnije možemo promijeniti vrijednost tako da pokazuje na nešto drugo.
- Međutim, ne smijemo mijenjati sadržaj ovog stringa (kroz `pmessage`) — rezultat je nedefiniran!

Broj riječi u stringu

Primjer. Napisati funkciju koja broji riječi u stringu koji je stigao kao argument. Riječ je niz znakova bez praznina, a riječi su odvojene bar jednom prazninom ili znakom `\t`.

Koristimo logičku varijablu `blank` koja pamti da li smo:

- u prazninama ("između" riječi) i tad je `TRUE` (`1`), ili
- u riječi i tad je `FALSE` (`0`).

Brojač riječi povećavamo na izlasku iz riječi.

```
int broj(char *str)
{
    int brojac = 0;
    int blank = TRUE;  /* Ispred prve rijeci. */
```

Broj riječi u stringu (nastavak)

```
    while (*str != '\0') {
        if ((*str == ' ') || (*str == '\t')) {
            if (!blank) {
                brojac++;
                blank = TRUE;
            }
        }
        else
            blank = FALSE;
        str++;
    }
    if (!blank) brojac++;  /* Zadnja rijec */
    return brojac;
}
```

Broj riječi u stringu (nastavak)

Ovaj primjer možemo napraviti i tako da brojač povećavamo

- na ulasku u riječ.

Umjesto `blank`, zgodnije je koristiti logičku varijablu `rijec` s obratnim značenjem:

- ako smo u riječi, vrijednost je `TRUE` (`1`),
- ako smo u prazninama ("između" riječi), vrijednost je `FALSE` (`0`).

```
int broj(char *str)
{
    int brojac = 0;
    int rijec = FALSE;  /* Ispred prve rijeci. */
```

Broj riječi u stringu (nastavak)

```
    for ( ; *str != '\0'; str++)
        if ((*str == ' ') || (*str == '\t')) {
            if (rijec)
                rijec = FALSE ;  /* Moze BEZ if */
        }
        else
            if (!rijec) {
                brojac++;
                rijec = TRUE;
            }
    return brojac;
}
```

Umjesto `while` petlje iskoristili smo `for` petlju.

Polja pokazivača

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];
```

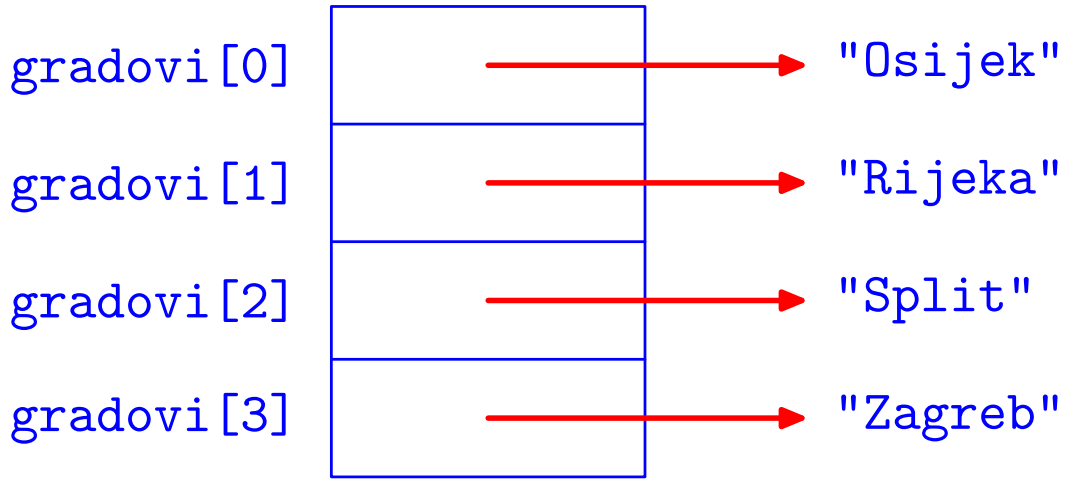
od pokazivača na polje (ovdje od 10 elemenata):

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

Polja pokazivača (nastavak)

Pokazivač na `char` možemo inicijalizirati stringom. Isto vrijedi i za polje takvih pokazivača (često se koristi).

```
static char *gradovi[] = { "Osijek", "Rijeka",
                           "Split", "Zagreb"};
```



Pokazivači i višedimenzionalna polja

Indeksiranje jednodimenzionalnog polja:

```
double x[10];
```

`x[i]` $\Longleftrightarrow$ `*(x + i)`.

Indeksiranje višedimenzionalnog polja:

```
double x[10][20];
```

`x[i][j]` $\Longleftrightarrow$ `*(x[i] + j)` $\Longleftrightarrow$ `*(*(x + i) + j)`.

`x[i]` je pokazivač na prvi element u polju `x[i]`, a to je `x[i][0]`. Dakle, `x[i] = &x[i][0]`, kao za svako polje.

Također, `x + 1` je pokazivač na sljedeći redak (polje) `x[1]`.

Pokazivači i višedimenzionalna polja (nastavak)

Pri deklaraciji višedimenzionalnog polja (ali ne kod definicije) mogu se koristiti ekvivalentne forme:

```
tip_pod ime[izraz_1][izraz_2]...[izraz_n];
```

ili bez prve dimenzije

```
tip_pod ime[][izraz_2]...[izraz_n];
```

ili pomoću pokazivača

```
tip_pod (*ime)[izraz_2]...[izraz_n];
```

U definiciji polja mora biti prvi oblik ili inicijalizacija.

Pokazivači i `const`

- Osim što modifikator `const` možemo koristiti za definiciju konstanti, možemo ga primijeniti i na pokazivače.
- Moguće je definirati konstantni pokazivač i na konstantni tip i na nekonstantni tip.
- Konstantni pokazivač uvijek pokazuje na istu lokaciju.

Primjer (sljedeća stranica):

Pokazivači i `const` *(nastavak)*

```
double x[] = {0.1, 0.2, 0.3};
const double y[] = {0.1, 0.2, 0.3};

const double *p1;  /* ptr na konst. double */
double * const p2 = x;
      /* konst. ptr na double */
const double * const p3 = y;
      /* konst. ptr na konst. double */

p1 = x;  /* OK, x ne mogu mijenjati kroz p1 */
p1[1] = 4.0;  /* GRESKA */
p2 = &x[2];   /* GRESKA */
p3 = &y[2];   /* GRESKA */
*p3 = 4.0;    /* GRESKA */
```

Dinamička alokacija memorije

Dinamička alokacija memorije služi za alociranje memorije

- za polja kod kojih dimenzija nije unaprijed poznata,
- za kreiranje dinamičkih struktura podataka (na pr. vezane liste).

Funkcijom `malloc` deklariranom u `<stdlib.h>` možemo dinamički alocirati memoriju. Deklaracija:

```
void *malloc(size_t n);
```

gdje je `size_t` cjelobrojni tip bez predznaka definiran u `<stddef.h>`, a

- `n` = ukupan broj bajtova koji treba alocirati.

Dinamička alokacija memorije (nastavak)

- `malloc` vraća pokazivač na rezervirani blok memorije ili `NULL` ako zahtjev nije ispunjen.
- Vraćeni pokazivač je generički, tipa `void*`, pa ga prije upotrebe treba konvertirati u potrebni tip pokazivača.

Druga mogućnost za dinamičku alokaciju memorije je funkcija `calloc`, isto iz `<stdlib.h>`. Deklaracija je:

```
void *calloc(size_t n_obj, size_t size);
```

Rezervira prostor za `n_obj` objekata, od kojih svaki pojedini objekt ima veličinu `size`.

Dodatno, inicijalizira cijeli prostor na nul–znakove (`'\0'`).

Dinamička alokacija memorije (nastavak)

Primjer: alokacija memorije za 128 objekata tipa `double`

```
double *p;
    ...
    p = (double *) malloc(128 * sizeof(double));
    if (p == NULL) {
        printf("Greska: neuspjela alokacija!\n");
        exit(-1);
    }
```

Može i ovako (s inicijalizacijom):

```
    p = (double *) calloc(128, sizeof(double));
```

Dealokacija memorije

Memoriju alociranu pomoću funkcije `malloc` ili `calloc` oslobađamo nakon upotrebe funkcijom `free`.

```
void free(void *p);
```

Ova funkcija uzima pokazivač na početak alociranog bloka memorije i oslobađa memoriju.

```
free(p);
```

Ako je `p == NULL`, onda ne radi ništa!

Argumenti komandne linije

Programi vrlo često koriste parametre, koji se učitavaju zajedno s imenom programa. Takvi parametri zovu se argumenti komandne linije.

```
cp ime1 ime2
```

Program koji želi koristiti argumente komandne linije, mora `main` deklarirati s dva argumenta:

- `argc` tipa `int`, i
- `argv` — polje pokazivača na `char`.

```
int main(int argc, char *argv[])
{ ... }
```

Argumenti komandne linije (nastavak)

Značenje `argc` ("argument count"):

- Broj `argc - 1` je broj argumenata komandne linije. Ako ih nema, onda je `argc = 1`.

Značenje `argv` ("argument value"):

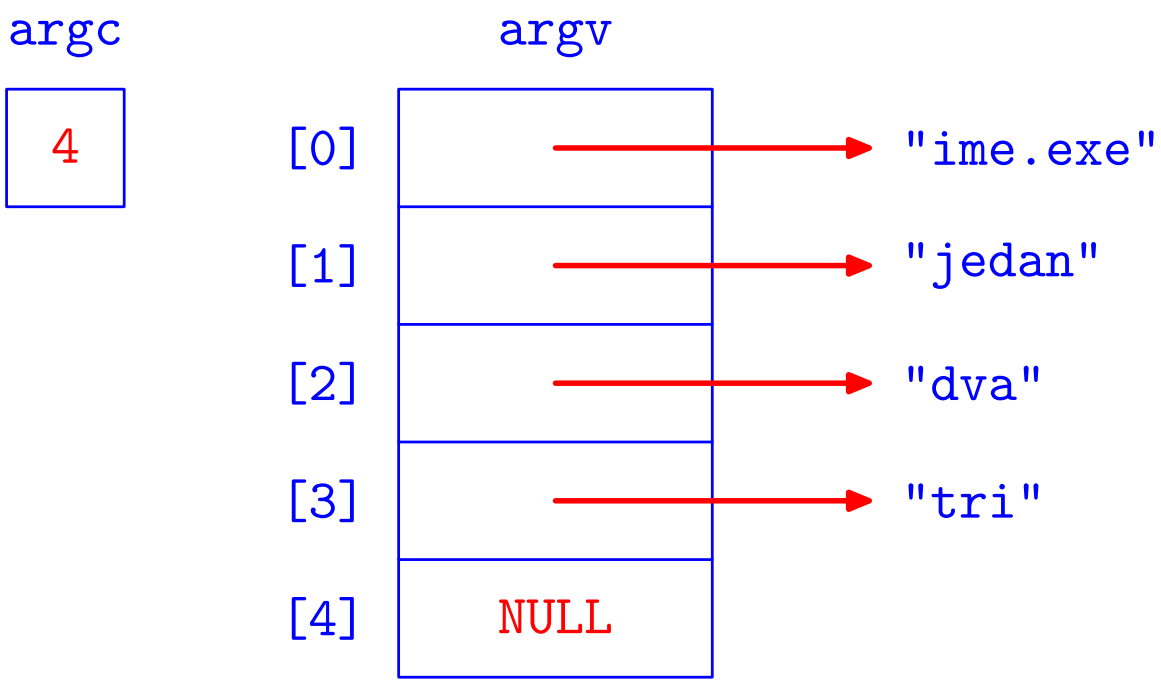
- U `argv` je polje pokazivača na argumente komandne linije.
- `argv[0]` uvijek pokazuje na string koji sadrži ime programa.
- Ostali parametri smješteni su redom kojim su upisani.
- Iza svega je `argv[argc] = NULL`.

Argumenti komandne linije (nastavak)

Primjer: Ako program pozovemo s:

```
ime.exe jedan dva tri
```

onda je:



Argumenti komandne linije (nastavak)

Primjer. Program koji ispisuje argumente komandne linije:

```
#include <stdio.h>  /* program args */

int main(int argc, char *argv[])
{
    int i;
    for(i = 0; i < argc; i++)
        printf("%s%s", argv[i],
               (i < argc-1) ? "," : ".");
    printf("\n");
    return 0;
}
```

Argumenti komandne linije (nastavak)

Poziv programa `args` naredbom:

```
args ovo su neki parametri
```

ispisuje

```
args,ovo,su,neki,parametri.
```

Dinamičko otvaranje polja

Primjer. Program dinamički "otvara" polje cijelih brojeva tipa `int`, s tim da se broj elemenata polja učitava.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *P;
    int i, broj, z;

    printf("Ucitaj broj elemenata polja P:");
    scanf("%d", &broj);
```

Dinamičko otvaranje polja (nastavak)

```
if ((P = (int*) calloc(broj, sizeof(int)))
        == NULL) {
    printf ("Nema dovoljno memorije\n");
    exit(1);
}
for (i = 0; i < broj; ++i) {
    printf ("Upisi element polja: ");
    scanf ("%d", &P[i]); }
z = 0;
for (i = 0, i < broj; ++i)
    z = z + P[i];
printf ("%d\n", z);
free(P);
return 0; }
```

Dinamičko otvaranje polja (nastavak)

Broj elemenata polja možemo učitati i s komandne linije, samo ga treba pretvoriti iz stringa u tip `int` (funkcija `atoi`).

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int *P;
    int i, broj, z;

    broj = atoi(argv[1]);
    ...
}
```

Pokazivač na funkciju

Pokazivač na funkciju deklarira se kao:

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

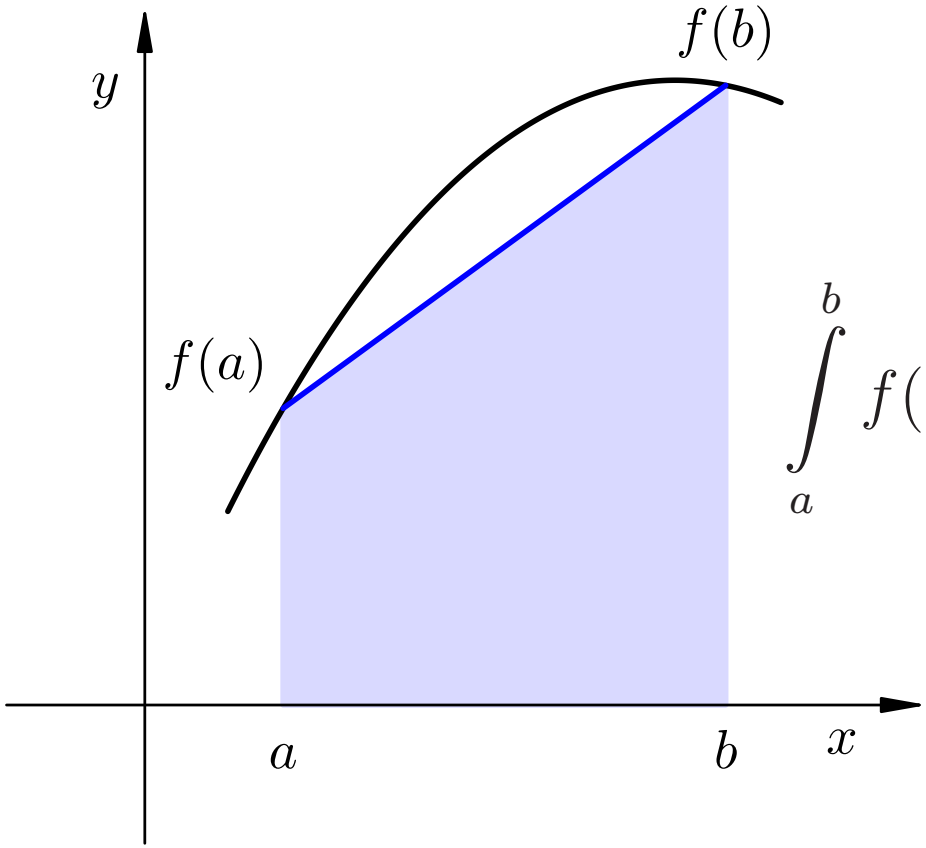
gdje je `ime` pokazivač na funkciju koja uzima `n` argumenata tipa `tip_1` do `tip_n` i vraća vrijednost tipa `tip_pod`.

Primjer:

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

Pokazivač na funkciju (nastavak)

Primjer. Funkcija koja po trapeznoj formuli približno računa integral zadane funkcije. Trapezna formula:



$$\int_a^b f(x)\,dx \approx \frac{f(a)+f(b)}{2}\,(b-a).$$

Pokazivač na funkciju (nastavak)

```
#include <stdio.h>
#include <math.h>
double integracija (double, double,
                    double (*)(double));
int main(void) {
    printf("Sin: %f\n", integracija(0, 1, sin));
    printf("Cos: %f\n", integracija(0, 1, cos));
    return 0; }

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

Složene deklaracije

Pri interpretaciji deklaracije uzimaju se u obzir prioriteti pojedinih operatora. Ti prioriteti mogu se promijeniti upotrebom zagrada.

Primjeri: p je:

```
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 (nastavak)

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

Strukture

Deklaracija strukture

Polja grupiraju veći broj podataka istog tipa, dok strukture služe grupiranju više podataka različitih tipova.

Struktura se deklarira na sljedeći način:

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

`struct` je rezervirana riječ, `ime` je ime strukture, a unutar vitičastih zagrada su članovi strukture.

Definicija varijabli tipa strukture

Primjer: Struktura `tocka` definira točku u ravnini.

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

Želimo li varijablu tipa `tocka`, to možemo učiniti na dva načina.

```
struct tocka {
   int x;
   int y;
};
struct tocka p1, p2;
```

Definicija varijabli tipa strukture (nastavak)

ili

```
struct tocka {
   int x;
   int y;
} p1, p2;
```

Općenito se varijabla tipa strukture definira sa:

```
mem_klasa struct ime var1, var2, ...;
```

Inicijalizacija strukture

Varijabla tipa strukture može se inicijalizirati pri definiciji:

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

pri čemu se `v_i` pridružuje `i`–tom članu strukture.

Primjer: Ako je definirana struktura

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

onda varijablu `kupac` inicijaliziramo ovako:

```
struct racun kupac = {1234, "Goran S.", -234.00};
```

Polje struktura

Slično se može inicijalizirati i polje struktura:

```
struct racun kupci[] = {34, "Ivo R.",    456.00,
                        35, "Josip S.",  234.00,
                        36, "Dalibor M.", 00.00};
```

Struktura definirana pomoću strukture

Strukture mogu sadržavati druge strukture kao članove.

Primjer: Pravokutnik je određen donjim lijevim `pt1` i gornjim desnim vrhom `pt2`.

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

Pritom deklaracija strukture `tocka` mora prethoditi deklaraciji strukture `pravokutnik`.
Ista imena članova mogu se koristiti u različitim strukturama.

Rad sa strukturama

- Članovima strukture može individualno pristupiti korištenjem primarnog operatora točka (`.`).
- Operator točka (.) separira ime varijable i ime člana strukture.
- Ima najvišu prioritetnu grupu i ima asocijativnost L → D.

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

```
var.memb
```

član `memb` u strukturi `var`.

Rad sa strukturama (nastavak)

Zbog najvišeg prioriteta točka operatora vrijedi:

$$\texttt{++varijabla.clan} \Longleftrightarrow \texttt{++(varijabla.clan)}$$
$$\texttt{\&varijabla.clan} \Longleftrightarrow \texttt{\&(varijabla.clan)}.$$

Primjer:

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

onda je prva koordinata (komponenta) varijable `ishodiste` `ishodiste.x`, a druga koordinata (komponenta) `ishodiste.y`.