

# *Programiranje (C)*

## *12. predavanje*

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

`www.math.hr/~singer`

PMF – Matematički odjel, Zagreb

# Sadržaj predavanja

- **Strukture** (zadnji dio):
  - Strukture i funkcije.
  - Strukture i pokazivači. Operator strelica (->).
  - Samoreferencirajuće strukture. Vezane liste.
  - typedef.
  - Unije.
- **Datoteke** (prvi dio):
  - Vrste datoteka — tekstualne i binarne.
  - Otvaranje i zatvaranje datoteke.
  - Standardne datoteke — `stdin`, `stdout`, `stderr`.
  - Funkcije za čitanje i pisanje — znak po znak, liniju po liniju, formatirani ulaz i izlaz.

# Strukture (nastavak)

# 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  $L \rightarrow D$ .

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:

`++varijabla.clan`  $\iff$  `++(varijabla.clan)`

`&varijabla.clan`  $\iff$  `&(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`.

## Rad sa strukturama (nastavak)

Primjer: Ako je

---

```
struct racun {  
    int broj_racuna;  
    char ime[80];  
    float stanje;  
} kupac = {1234, "Dalibor M.", 23456.00};
```

---

tada je

---

```
kupac.broj_racuna = 1234,  
kupac.ime = "Dalibor M.",  
kupac.stanje = 23456.00.
```

---

## Struktura i polje

Kada struktura sadrži **polje kao član strukture**, onda se elementi polja dosežu izrazom:

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

Primjer:

```
struct racun {  
    int broj_racuna;  
    char ime[80];  
    float stanje;  
} kupac = {1234, "Dalibor M.", -23456.00};  
  
...  
if (kupac.ime[0] == 'D') ...
```

## Struktura i polje (nastavak)

Ako imamo **polje struktura**, onda strukturu sadržanu u pojedinom elementu polja dosežemo izrazom

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

Asocijativnost je bitna, jer su svi operatori istog prioriteta.

Primjer:

```
struct tocka {
    int x;
    int y;
} vrhovi[1024] ;

...
if vrhovi[17].x == vrhovi[17].y ...
```

# Strukture i funkcije

Operacije nad strukturom kao cjelinom su:

- Pridruživanje.
- Uzimanje adrese, primjena `sizeof` operatora.
- Struktura **može** biti argument funkcije. Funkcija tada dobiva kopiju strukture kao argument.
- Funkcija **može** vratiti strukturu.

**Primjer.** Funkciji `suma` argumenti su 2 strukture tipa `tocka`, a vraća sumu argumenata (tipa `tocka`).

---

```
struct tocka {  
    int x;  
    int y;  
} t, ishodiste = {0, 0};
```

## Strukture i funkcije (nastavak)

```
struct tocka suma(struct tocka p1,
                  struct tocka p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
...
t = ishodiste /* t i ishodiste moraju
              biti istog tipa */
printf("%d\n", sizeof(t));
```

---

Napomena: **Nije** dozvoljeno **uspoređivanje** struktura.

# Strukture i pokazivači

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

---

```
struct tocka {
    int x;
    int y;
} p1, *pp1;
...
pp1 = &p1;
(*pp1).x = 13;
(*pp1).y = 27;
*pp1.x = 13; /* GRESKA */
/* *pp1.x je isto sto i *(pp1.x) */
```

---

## Operator strelica (->)

- Primarni operator strelica (->) nudi jednostavan način dohvaćanja člana strukture korištenjem pokazivača.
- Asocijativnost operatora -> je  $L \rightarrow D$ .

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

$$\text{ptvar} \rightarrow \text{clan} \iff (*\text{ptvar}).\text{clan}$$

Primjer:

---

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

---

## Složeni izrazi

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

Sljedeći su izrazi ekvivalentni:

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

## Složeni izrazi (nastavak)

Primjer. Što će ispisati sljedeći program?

---

```
struct {
    int n;
    char *ch;
} t[10], *pt = &t[0]; /* globalne var. */

int main(void) {
    int i;
    char tmp;

    for (i = 0; i < 10; i++) { /* inic. polja */
        t[i].n = i;
        t[i].ch = (char *)malloc(10);
    }
}
```

## Složeni izrazi (nastavak)

```
        sprintf(tmp, "%c", 'a' + i);
        strcat(tmp, "++");
        strcpy(t[i].ch, tmp);
    }
    printf("%s \n", (++pt) -> ch);
    printf("%s \n", pt++ -> ch);
        /* pt++ -> ch <=> (pt++) -> ch */
        /* zbog razlicite asoc. operatora */
    printf("%c \n", *pt -> ch++);
    printf("%c \n", *pt++ -> ch);
    printf("%s \n", pt -> ch);
    return 0;
}
```

---

# *Složeni izrazi (nastavak)*

Ispis programa:

---

b++

b++

c

+

d++

---

# Samoreferentne strukture

- Za implementaciju tipova podataka kao što su **vezane liste** i **stabla** koristimo **samoreferentne strukture**.
- **Samoreferentna** struktura ima **jedan** ili **više** članova koji su **pokazivači** na strukturu **istog tipa**.

Primjer:

---

```
struct element {  
    char ime[64];  
    struct element *next;  
};
```

---

## Samoreferentne strukture — vezana lista

**Primjer.** Ilustrirajmo upotrebu samoreferentne strukture na primjeru **jednostruko** povezane **liste** čiji su elementi znakovni nizovi. Početak programa izgleda ovako:

---

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

/* oznaka kraja liste */
#define KRAJ (struct ime *) NULL
```

## Vezana lista (nastavak)

```
/* deklaracija samoreferentne strukture */  
/* Vezana lista */  
  
struct ime {  
    char *p_ime;  
    struct ime *next;  
};  
  
/* pokazivac na prvi element liste  
   (globalna varijabla) */  
/* Na pocetku lista je PRAZNA! */  
  
struct ime *start=KRAJ;
```

---

# Operacije nad vezanom listom

Uobičajene operacije nad vezanom listom su:



- dodavanje novog elementa na kraj liste: `void unos(void)`,
- ispis liste: `void ispis(void)`,
- pretraživanje liste: `void trazi(void)`,
- brisanje elementa iz liste: `void brisi(void)`,

## *Dodavanje na kraj liste*

**Primjer.** Funkcija `unos` dodaje novi element na **kraj** liste.

---

```
void unos(void) {
    struct ime *zadnji, *novi;
    char line[128];

    novi = (struct ime *) malloc(sizeof(struct ime));
    novi -> next = KRAJ;
    printf("Unesite ime > ");
    scanf("%[^\n]", line);
    novi -> p_ime = (char *) malloc(strlen(line) + 1);
```

## *Dodavanje na kraj liste (nastavak)*

```
strcpy(novi -> p_ime, line);

if (start == KRAJ) /* prazna lista */
    start=novi;
else { /* pronadji kraj liste */
    for (zadnji = start; zadnji -> next != KRAJ;
        zadnji = zadnji -> next) ;
    /* zadnji pokazuje na novi element */
    zadnji -> next = novi;
}
}
```

---

## Ispis liste

**Primjer.** Funkcija `ispis` ispisuje elemente vezane liste.

---

```
void ispis(void) {
    struct ime *element;
    int i = 1;
    if (start == KRAJ) {
        printf("Lista je prazna.\n");
        return; }
    printf("Ispis liste \n");
    for (element = start; element != KRAJ;
        element = element -> next) {
        printf("%d. %s\n", i, element -> p_ime); i++;
    }
}
```

---

# Pretraga po listi

**Primjer.** Funkcija `trazi` ispituje nalazi li se učitani podatak u listi.

---

```
void trazi(void) {  
    struct ime *test;  
    char line[128];  
    int i;  
  
    printf("Nalazenje imena u listi.\n");  
    printf("Unesite ime ");  
    scanf(" %[^\n]", line);
```

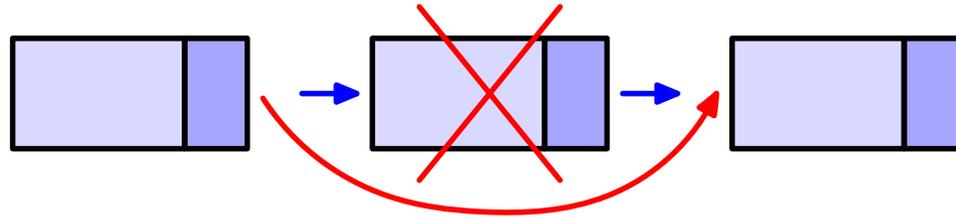
## Pretraga po listi (nastavak)

```
i = 1;
for (test = start; test != KRAJ;
     test = test -> next) {
    if ( !strcmp(test -> p_ime, line) ) {
        printf("Podatak je %d. u listi\n",i);
        return;
    }
    i++;
}
printf("Podatak nije u listi.\n");
return;
}
```

---

## Brisanje u listi

Primjer. Funkcija `brisi` (ako postoji) briše podatak iz liste.



```
void brisi(void) {
    struct ime *test, *tmp;
    char line[128];
    int i;

    printf("Brisanje imena iz liste.\n");
    if (start == KRAJ){
        printf("Lista je prazna.\n");
        return; }
}
```

## Brisanje u listi (nastavak)

```
printf("Unesite ime ");
scanf(" %[^\n]", line);

/* ako je prvi element onaj koji trazimo */
if ( !strcmp(start -> p_ime, line) ) {
    printf("Podatak je 1. u listi\n");
    tmp = start;
    start = start -> next;
    free(tmp -> p_ime);
    free(tmp);
    return;
}
i = 1;
```

## Brisanje u listi (nastavak)

```
for (test = start; (tmp = test -> next) != KRAJ;
     test = test -> next) {
    i++;
    if ( !strcmp(tmp -> p_ime, line) ) {
        printf("Brisemo podatak");
        printf("    br. %d u listi\n", i);
        test -> next = test -> next -> next;
        free(tmp -> p_ime);
        free(tmp);
        return; }
    }
printf("Podatak nije u listi.\n");
return; }
```

---

# Glavni program s izbornikom operacija

**Primjer.** Izbornik `menu` i glavni program koji objedinjuje prethodne 4 operacije.

---

```
int menu(void) {
int izbor;
printf("\n\n\n");
printf("\n      OPERACIJE : ");
printf("\n      =====");
printf("\n 1. Unos novog elementa ");
printf("\n 2. Ispis liste ");
printf("\n 3. Pretrazivanje liste ");
printf("\n 4. Brisanje iz liste ");
printf("\n 5. Izlaz ");
printf("\n      izbor = ");
```

## *Glavni program s izbornikom operacija (nast.)*

```
scanf("%d",&izbor);  
return izbor;  
}
```

---

Glavni program:

---

```
int main(void) {  
    int izbor;  
    do {  
        switch(izbor = menu()) {  
            case 1: unos(); break;  
            case 2: ispis(); break;  
            case 3: trazi(); break;  
            case 4: brisi(); break;  
            case 5: break;
```

## *Glavni program s izbornikom operacija (nast.)*

```
    default: printf("\n Pogresan izbor."); }  
    } while(izbor != 5);  
    return 0;  
}
```

---

# typedef

Korištenjem ključne riječi **typedef** postojećim tipovima podataka dajemo nova imena (**ne kreiramo nove tipove!**).

Primjer:

---

```
typedef double Masa;
```

---

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

---

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

---

Općenito **typedef** ima oblik:

---

```
typedef tip_podatka novo_ime_za_tip_podatka;
```

---

## typedef *i pokazivači*

Primjer: Pokazivač na `double` možemo nazvati `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 strukture*

`typedef` se često koristi sa strukturama kako bi se **eliminirala potreba** da se pri deklaraciji varijabli navodi `struct`.

Primjer: Umjesto

---

```
struct tocka {
    int x;
    int y;
};
...
struct tocka p1, *pp1;
```

---

## typedef *i* strukture (nastavak)

možemo pisati:

---

```
typedef struct {  
    int x;  
    int y;  
} tocka;  
  
...  
tocka p1, *pp1;
```

---

## typedef *i samoreferentne strukture*

Kada se `typedef` primjenjuje na samoreferentnu strukture potrebno je uvesti jednu prethodnu deklaraciju tipa, kako bi se izbjegla cirkularna deklaracija.

Primjer:

---

```
typedef struct element *Pelement;
```

```
typedef struct element {  
    char ime[64];  
    Pelement next;  
} Element;
```

```
...  
Element root;
```

---

## typedef i deklaracije funkcija

Česta upotreba naredbe `typedef` je **skraćivanje kompleksnih deklaracija** kao što su pokazivači na funkcije.

Primjer:

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

`PF` postaje ime za 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) { ...
```

# Unija

Osnovna svrha unije je **ušteta memorijskog prostora**.

- **Unija** kao i struktura sadrži **članove različitog tipa**.
- Kod unije **svi** članovi dijele **istu** memorijsku lokaciju.
- Memorijska lokacija bit će dovoljno velika da u nju stane najširi član unije.

Sintaksa:

---

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

---

## Unija (nastavak)

Varijable `x` i `y` tipa `ime` mogu se deklarirati ovako:

```
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` varijable tipa `float`.

## Unija (nastavak)

**Primjer.** Uniju možemo iskoristiti kako bismo ispisali binarni zapis broja s pomičnim zarezom.

---

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

---

# Datoteke

# Osnovno o datotekama

**Datoteka** je imenovano područje u vanjskoj, sporoj memoriji, najčešće na disku (disketi, CD-u i sl.), a služi za smještaj podataka.

- ANSI C standard poznaje dvije vrste datoteka: **tekstualne** i **binarne**.
- **Binarna datoteka** je niz podataka tipa **char**. Svaki tip podatka u C-u može se preslikati u niz vrijednosti tipa **char**, pa je u binarnu datoteku moguće upisati podatke kao što su reprezentirani u računalu.
- **Teksualna datoteka** je niz znakova podijeljenih u **linije**. Svaka linija sadrži nula ili više znakova iza kojih slijedi znak za prijelaz u novi red **\n**. Namijenjene su pohranjivanju tekstualnih podataka.

# Razina ulazno–izlaznih operacija

Postoje **dvije** razine ulazno-izlaznih operacija.

- Funkcije za ulaz/izlaz koje nudi **operacijski sustav**, su tzv. **niska razina** ulaza/izlaza.
- Standardne ulazno/izlazne funkcije čiji su **prototipovi** dani u **<stdio.h>** datoteci su tzv. **visoka razina** ulaza/izlaza.

Funkcije iz standardne biblioteke jezika C skrivaju niz detalja vezanih uz operacijski sustav, pa su **jednostavnije za korištenje** od funkcija koje nudi operacijski sustav. Također **lako su prenosive** na druge operacijske sustave.

# Otvaranje i zatvaranje datoteke

- **Komunikacija** s datotekama vrši se preko **spremnika** (engl. **buffer**) u koji se privremeno pohranjuju informacije koje se šalju u datoteku.
- Svrha je spremnika  **smanjiti komunikaciju** s vanjskom memorijom (diskom) i povećati efikasnost ulazno–izlaznih funkcija.
- Spremnik i ostali podaci potrebni za otvaranje **datoteke** deklarirani su u **strukturi FILE**.
- **Prvi** korak u **otvaranju** datoteke je deklaracija **pokazivača** na **FILE**.

Primjer:

---

```
FILE *ptvar;
```

---

# Otvaranje i zatvaranje datoteke (nastavak)

Prije prve operacije pisanja ili čitanja datoteka mora biti otvorena korištenjem funkcije `fopen`.

```
FILE *fopen(const char *ime, const char *tip);
```

gdje je:

- `ime` pravo ime datoteke koja se otvara,
- a `tip` string koji kaže kako treba otvoriti tu datoteku (v. malo kasnije).

Funkcija `fopen` vraća:

- pokazivač na strukturu `FILE` povezanu s datotekom, ako je datoteka uspješno otvorena,
- `NULL`, ako datoteka nije mogla biti otvorena (greška).

# Otvaranje i zatvaranje datoteke (nastavak)

Primjer: Tipična upotreba `fopen`:

```
ptvar = fopen(ime, tip);
if (ptvar == NULL)
{
    printf("Poruka o gresci"):
    ...
}
```

Ovdje je `ime` pravo **ime datoteke**, onako kako se datoteka “zove” u operacijskom sustavu.

👉 Na primjer: `"podaci.dat"`.

Drugi string `tip` je jedan od sljedećih stringova.

## Otvaranje i zatvaranje datoteke (nastavak)

Za operacije u **tekstualnom modu** koriste se sljedeći tipovi:

tip	značenje
"r"	otvaranje postojeće datoteke samo za čitanje
"w"	kreiranje nove datoteke samo za pisanje
"a"	otvaranje postojeće datoteke za dodavanje teksta
"r+"	otvaranje postojeće datoteke za čitanje i pisanje
"w+"	kreiranje nove datoteke za čitanje i pisanje
"a+"	otvaranje postojeće datoteke za čitanje i dodavanje teksta

**r** = read, **w** = write, **a** = append.

# Otvaranje i zatvaranje datoteke (nastavak)

Pritom vrijede sljedeća pravila:

- Ako se **postojeća** datoteka otvori s "w" ili "w+" njen će sadržaj **biti izbrisan** i pisanje će početi od **početka**.
- Ako datoteka koju otvaramo s tipom "a" ili "a+" **ne postoji** bit će **kreirana**.
- Ako se datoteka otvara s tipom "a" ili "a+" **novi tekst** bit će dodan **na kraju** datoteke.

## Otvaranje i zatvaranje datoteke (nastavak)

Za operacije u **binarnom modu** treba svakom tipu dodati **b**.

tip	značenje
"rb"	binarno čitanje
"wb"	binarno pisanje
"ab"	binarno dodavanje
"rb+" ili "r+b"	binarno čitanje i pisanje (otvaranje postojeće)
"wb+" ili "w+b"	binarno čitanje i pisanje (kreiranje nove)
"ab+" ili "a+b"	binarno čitanje i dodavanje

Unix ima samo jedan tip datoteka (binarno = tekstualno).

# Otvaranje i zatvaranje datoteke (nastavak)

Na kraju programa datoteka treba biti **zatvorena** funkcijom **fclose** koja uzima kao argument pokazivač na **FILE**.

```
int fclose(FILE *fp);
```

Funkcija **fclose** vraća:

- **nulu**, ako je datoteka **uspješno** zatvorena,
- **EOF** u slučaju **greške**.

Primjer:

```
fclose(ptvar);
```

# Otvaranje i zatvaranje datoteke (nastavak)

**Primjer.** Otvaranje i zatvaranje datoteke `primjer.dat` izgledalo bi ovako:

---

```
#include <stdio.h>
...
FILE *fpt;
if ((fpt = fopen("primjer.dat", "w")) == NULL) {
    printf("\nGRESKA pri otvaranju datoteke.\n");
    exit(-1);
}
/* Rad s datotekom */
...
fclose(fpt);
```

---