

Programiranje (C)

3. predavanje

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

`www.math.hr/~singer`

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

Nastavak pregleda programskog jezika C na (sad već) nešto složenijim primjerima programa.

- Osnovni elementi jezika kroz primjere:
 - Funkcije.
 - Polja.
 - Pokazivači.

Sve što ovdje ilustriramo bit će detaljnije obrađeno kasnije.

Primjer 5 — funkcije

Primjer 5. Treba napisati funkciju koja računa binomni koeficijent

$$\binom{n}{k} = \frac{n \cdot (n - 1) \cdots (n - k + 1)}{1 \cdot 2 \cdots k}.$$

Funkcija treba imati dva cjelobrojna ulazna parametra:

- n (za n) i k (za k).

Funkcija treba vratiti binomni koeficijent $\binom{n}{k}$.

Glavni program treba ispisati Pascalov trokut za $n < 10$.

- U liniji s indeksom n ispisani su brojevi $\binom{n}{k}$, za sve vrijednosti k od 0 do n .

Primjer 5 (nastavak)

Primjer Pascalovog trokuta za $n < 10$:

$n = 0$

1

$n = 1$

1 1

$n = 2$

1 2 1

$n = 3$

1 3 3 1

$n = 4$

1 4 6 4 1

$n = 5$

1 5 10 10 5 1

$n = 6$

1 6 15 20 15 6 1

$n = 7$

1 7 21 35 35 21 7 1

$n = 8$

1 8 28 56 70 56 28 8 1

$n = 9$

1 9 36 84 126 126 84 36 9 1

Primjer 5 (nastavak)

Radi jednostavnosti, Pascalov trokut ispisujemo poravnani po lijevoj strani:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1

Primjer 5 (nastavak)

```
#include <stdio.h>

/* Funkcija binom() racuna binomni koeficijent. */

long binom(int n, int k)
{
    long rezult = 1; // inicializacija rezultata
    int i;

    if (n == k) return 1;
    if (k == 0) return 1;
```

Primjer 5 (nastavak)

```
for (i = n; i > n - k; i--)
    rezult = rezult * i;

for (i = 1; i <= k; ++i)
    rezult = rezult / i;

return rezult;
}
```

Uočite: Funkcija **binom** definirana je **ispred** funkcije **main**.
Ima dva argumenta tipa **int** i vraća rezultat tipa **long**
(binomni koeficijenti mogu biti “veliki”).

Primjer 5 (nastavak)

```
int main(void)
{
    long bnm;
    int n, k;

    for (n = 0; n < 10; n++) {
        for (k = 0; k <= n; k++) {
            bnm = binom(n,k);
            printf("%ld ", bnm);
        }
        printf("\n");
    }
    return 0;
}
```

Definicija funkcije

Definicija funkcije ima oblik:

```
tip ime_funkcije(tip_1 arg_1, tip_2 arg_2, ...)  
{  
    deklaracije varijabli  
    naredbe  
}
```

Idemo redom.

- **tip** je tip podatka koji funkcija **vraća** kao rezultat.
- Možemo pisati i **void** kao tip, ako funkcija **ne vraća** nikakvu vrijednost (već samo radi neki posao).

Zatim ide **ime_funkcije**.

Definicija funkcije (nastavak)

Unutar zagrada nalaze se **deklaracije** tzv. **formalnih argumenata** funkcije:

- **tip₁** **arg₁**, **tip₂** **arg₂**,

Za svaki argument navodi se **tip** i **ime**.

Napomena: Svaki **formalni argument** ujedno je i **lokalna varijabla** u toj funkciji.

- **Vrijednost** te varijable **inicijalizira** se pri pozivu funkcije iz pripadnog “**stvarnog**” argumenta.

(Tzv. prijenos argumenata **po vrijednosti**).

Primjer: **long binom(int n, int k)**.

Definicija funkcije (nastavak)

Ako funkcija **nema** argumenata,

- pišemo **void** kao tip, **bez imena argumenta**.

Primjer: **int main(void)**.

Tijelo funkcije nalazi se unutar vitičastih zagrada. U tijelu:

- prvo se pišu deklaracije varijabli (i to su **lokalne variable**),
- a zatim **izvršne naredbe**.

Poziv funkcije

Funkciju obično pozivamo u naredbi pridruživanja (kao izraz ili dio izraza):

```
varijabla1 = ime_funkcije(a_1, a_2, ...);
```

U pozivu navodimo **ime_funkcije**, a u zagradama pišemo **stvarne argumente** funkcije

- **a_1, a_2,**

Primjeri poziva funkcije **binom**:

<code>bnm = binom(n,k);</code>	<code>bnm = binom(i,j);</code>
<code>bnm = binom(5,3);</code>	<code>bnm = binom(5 + 4, 7 - 2);</code>
<code>bnm = binom(5 * i, j - 3) + binom(k + 4, 4);</code>	

Poziv funkcije (nastavak)

Što se događa kod poziva funkcije?

- Vrijednosti stvarnih argumenata se kopiraju u pripadne formalne (tj. u lokalne variable u funkciji);
- Funkcija radi svoj posao;
- Vrijednost vraćena naredbom return zamjenjuje cijeli poziv funkcije.

U našem primjeru

```
varijabla1 = ime_funkcije(a_1, a_2, ...);
```

nakon poziva, vraćena vrijednost se kopira u varijablu varijabla1 na lijevoj strani naredbe pridruživanja.

Poziv funkcije (nastavak)

Ako funkcija **ne vraća** vrijednost, ili nam vraćena vrijednost **ne treba**, poziv može imati oblik **naredbe**

```
ime_funkcije(a_1, a_2, ...);
```

Primjer: pozivi funkcija **printf** i **scanf**.

Operacijski sustav **poziva** funkciju **main**. Vraćena vrijednost je izlazni status:

- **return 0;** kod normalnog završetka programa,
- **return n;** ($n \neq 0$) ako je došlo do greške.

Prototip funkcije

Pravilo: Funkcija mora biti deklarirana **prije** mesta poziva, kako bi prevodilac u trenutku poziva funkcije znao

- tip rezultata,
- broj i tip argumenata.

Svrha: kontrola grešaka!

Mogućnosti u C-u:

- funkciju potpuno definirati **prije** mesta poziva (kod nas: **binom** je definiran **ispred main**, gdje je poziv),
- napisati tzv. **prototip** funkcije **ispred** poziva, a funkciju onda smijemo definirati i **iza** mesta poziva.

Prototip funkcije (nastavak)

Prototip funkcije je samo **deklaracija** funkcije. Dobiva se tako da:

- u definiciji funkcije **ispustimo tijelo** funkcije (ostaje samo zaglavlje ili deklaracija).

Prototip završava točka–zarezom iza), kao svaka naredba.

Prototip ima oblik:

```
tip ime_funkcije(tip_1 arg_1, tip_2 arg_2, ...);
```

Prototip za funkciju **binom** je:

```
long binom(int n, int k);
```

Prototip funkcije (nastavak)

U prototipu smijemo **izostaviti imena** formalnih argumenata (dovoljno je navesti **tip**). Takav prototip ima oblik:

```
tip ime_funkcije(tip_1, tip_2, ...);
```

Odgovarajući prototip za funkciju **binom** je:

```
long binom(int, int);
```

Stilska preporuka: Dobro izabrana **imena** bitno pomažu čitljivosti programa.

Ako **ime funkcije** očito asocira na **svrhu** funkcije i **značenje argumenata**, može i **skraćeni prototip** (lakše se čita).

Primjer 5 (uz pomoć prototipa)

```
#include <stdio.h>

long binom(int n, int k) ; /* Prototip */

int main(void) /* Glavni program */
{
    /* Sve isto kao prije */
}

long binom(int n, int k) /* Definicija */
{
    /* Sve isto kao prije */
}
```

Primjer 5 — Zadaci

Zadatak. Isprobajte (testiranjem) za koje ulazne brojeve n i k funkcija **binom** radi **dobro**, tj. korektno računa binomni koeficijent $\binom{n}{k}$. Uočite da

- prvo množimo sve brojeve u brojniku (taj rezultat brzo raste),
- zatim dijelimo sa svim brojevima u nazivniku (rezultat stalno pada).

Razmotrite da li je bolje računati ovako:

```
for (i = 1; i <= k; ++i)
    rezult = rezult * (n + 1 - i) / i;
```

(brojnik i nazivnik “unaprijed”).

Primjer 5 — Zadaci (nastavak)

Ili čak ovako (brojnik unazad, nazivnik unaprijed):

```
for (i = 1; i <= k; ++i)
    rezult = rezult * (n - k + i) / i;
```

Da li u nazviku moramo ići “unaprijed”. Zašto?

Zadatak. Preuređite **glavni** program tako da ispisuje Pascalov trokut **centrirano**, kao u primjeru!

Zadatak. Kad napravimo strukturu **polja** (sljedeći primjer), napravite program koji računa **red po red** Pascalovog trokuta, koristeći **polje** za jedan red trokuta i formula

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

Primjer 6 — polja

Primjer 6. Napisati program koji učitava koordinate dva vektora u \mathbb{R}^3 i ispituje okomitost tih vektorâ.

Matematički pojmovi vektora i matrica (LA) implementiraju se u jeziku C pomoću polja.

Polje je složeni tip podataka. Sastoji se od

- niza indeksiranih varijabli istog tipa,
- smještenih u memoriji na uzastopnim lokacijama.
- Indeks polja uvijek kreće od nula.

U ovom primjeru polja koristimo za spremanje vektora.

Primjer 6 (nastavak)

Za ispitivanje okomitosti vektora \vec{a} i \vec{b} koristimo formule:

$$\cos \varphi = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}.$$

Koordinatno zapisano:

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3, \quad \|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}.$$

Uvjet okomitosti je $\cos \varphi = 0$ (tj. $\varphi = \pi/2$). Zbog **grešaka zaokruživanja** u realnoj aritmetici koristimo test:

$$|\cos \varphi| < \varepsilon, \quad \varepsilon = 10^{-10}.$$

Primjer 6 (nastavak)

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

double epsilon = 1.0E-10;      // globalna varijabla

/* Prototip funkcije */
double kosinus_kuta(double x[], double y[]);

int main(void)
{
    double a[3], b[3];
    double cos_phi;
    int i;
```

Primjer 6 (nastavak)

```
printf("Unesite vektor a.\n");
for (i = 0; i < 3; ++i) {
    printf("a[%d]= ", i+1);
    scanf(" %lf", &a[i]);
}

printf("Unesite vektor b.\n");
for (i = 0; i < 3; ++i) {
    printf("b[%d]= ", i+1);
    scanf(" %lf", &b[i]);
}

cos_phi = kosinus_kuta(a, b);

printf("Kosinus kuta = %f\n", cos_phi);
```

Primjer 6 (nastavak)

```
if (fabs(cos_phi) < epsilon)
    printf("Vektori su okomiti.\n");
else
    printf("Vektori nisu okomiti.\n");

return 0;
}
```

Definicije funkcija **norma**, **produkt** i **kosinus_kuta** pišu malo kasnije, nakon opisa deklaracije polja.

Deklaracija polja

Deklaracija polja ima oblik:

tip ime[MAX] ;

pri čemu je:

- **ime** je **ime** polja — zajedničko ime svih članova niza (elemenata polja);
- **tip** je **tip** polja, tj. tip svakog pojedinog člana, na pr. **double**, **int**, ... ;
- **MAX** je **broj** elemenata polja (**konstanta**).

Elementi ovako deklariranog polja su:

ime[0], **ime[1]**, ... , **ime[MAX-1]**.

Svaki od tih elemenata je **varijabla** tipa **tip**.

Deklaracija polja (nastavak)

Primjer: deklaracija

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

rezervira prostor za polje od 10 elemenata tipa **double**, i to u **bloku susjednih lokacija** u memoriji (**80 byteova**).

Elementi tog polja su:

```
x[0], x[1], x[2], x[3], x[4],  
x[5], x[6], x[7], x[8], x[9].
```

Svaki od tih elemenata je “normalna” **varijabla** tipa **double**.

Napomena. Objekt s imenom **x** je

- **pokazivač** na **prvi** element polja, tj. **adresa** varijable **x[0]**.

Deklaracija polja (nastavak)

U deklaraciji polja

- **mora** se zadati **broj elemenata** tog polja,
zbog rezervacije memorije.
- Dakle, **double x[]** je **greška**.

Izuzetak je ako odmah iza toga inicijaliziramo cijelo polje,
pa se broj elemenata “vidi” iz inicijalizacije (v. kasnije).

Kad želimo da **polje** bude **argument** neke funkcije, u
pripadnoj deklaraciji **formalnog argumenta**

- **ne moramo** navesti **broj elemenata** između **[i]**.
- Same zagrade **moramo** napisati, da bude jasno da je
riječ o **polju** (a ne o običnoj varijabli).

Polja kao formalni argumenti

Zato prototip funkcije `kosinus_kuta` možemo napisati kao:

```
double kosinus_kuta(double x[], double y[]);
```

ali i u obliku:

```
double kosinus_kuta(double x[3], double y[3]);
```

Uočiti: kod ovih deklaracija **polja** kao **formalnih argumenata** **nema** rezervacije memorije, tj. **nema lokalnog polja** u funkciji (izbjegava se kopiranje polja kod poziva)!

Pravi formalni argument je:

- samo **ime** **polja**, s pripadnim **tipom**, tj.
- **pokazivač** na **prvi element** **polja**.

Dakle, formalni argument je **adresa** **variabile** **x[0]**!

Primjer 6 (nastavak)

Definicije funkcija **norma**, **produkt** i **kosinus_kuta**.

```
double norma(double x[]) {  
    int i;  
    double suma;  
  
    suma = 0.0;  
    for (i = 0; i < 3; ++i)  
        suma = suma + x[i] * x[i];  
  
    return sqrt(suma);  
}
```

Primjer 6 (nastavak)

```
double produkt(double x[], double y[]) {  
    int i;  
    double suma;  
  
    suma = 0.0;  
    for (i = 0; i < 3; ++i)  
        suma = suma + x[i] * y[i];  
  
    return suma;  
}
```

Primjer 6 (nastavak)

```
double kosinus_kuta(double x[], double y[]) {  
    double cos_phi;  
  
    cos_phi = produkt(x,y);  
    cos_phi = cos_phi / (norma(x) * norma(y));  
  
    return cos_phi;  
}
```

Napomena. Za funkcije **norma** i **produkt** ne moramo pisati prototip, jer su definirane **ispred** funkcije **kosinus_kuta**, a samo u njoj ih pozivamo.

Funkcija **main** ne zove **norma** i **produkt**.

Globalne i lokalne varijable

Svaka varijabla deklarirana **unutar** neke funkcije (pa i **main**) je **lokalna** varijabla za tu funkciju.

- Ona “**živi**” toliko dugo **dok se funkcija izvršava** (na tzv. “run–time stacku”).

Posljedica: razne funkcije mogu imati **lokalne** varijable **istog** imena.

Varijable deklarirane (definirane) **izvan** svih funkcija su tzv. **globalne** varijable.

- One “**traju**” (tj. može im se pristupiti) od **mjesta deklaracije** pa do **kraja datoteke** u kojoj su deklarirane.

Primjer: Varijabla **epsilon** deklarirana je na početku datoteke. Može se koristiti u **svim** funkcijama definiranim **iza** nje.

Matematičke funkcije

Matematičke funkcije iz standardne biblioteke

- imaju argument(e) tipa **double**, i
- također, vraćaju rezultat tipa **double**.

Neke standardne funkcije iz matematičke biblioteke:

funkcija	značenje	funkcija	značenje
sin	sin	exp	e^x
cos	cos	log	ln
tan	tg	log10	\log_{10}
asin	arcsin	sqrt	$\sqrt{}$
acos	arccos	fabs	$ $
atan	arctg		

Potenciranje

Sve navedene funkcije imaju jedan argument tipa **double**.

C **nema** operator **potenciranja**. Zato postoji funkcija **pow**:

double pow(double, double)

koja računa

$$\text{pow}(x, y) = x^y.$$

Ona daje **grešku** ako je $x = 0$ i $y \leq 0$, ili $x < 0$ i y **nije** cijeli broj.

Ako koristimo funkcije iz standardne biblioteke treba uključiti zaglavlje **<math.h>** (i compileru dati opciju **-lm**).

Primjer 6 — Zadaci

Zadatak. Napišite funkciju za učitavanje koordinata vektora $\vec{x} \in \mathbb{R}^3$ i iskoristite ju u glavnom programu za učitavanje koordinata vektora \vec{a} i \vec{b} .

Kad napravimo **pokazivače** (sljedeći primjeri), preuređite formalne parametre u pokazivače na prvi element odgovarajućeg polja.

Zadatak. Napišite funkciju za računanje **vektorskog produkta** dva vektora. Ta funkcija mora vratiti polje koordinata vektora $\vec{z} = \vec{x} \times \vec{y}$.

Primjer 7 — pokazivači

Primjer 7. Svaka **varijabla** deklarirana u C programu ima svoju **adresu** u memoriji. Tu adresu moguće je “dohvatiti” putem **adresnog operatora &**. Na primjer:

```
#include <stdio.h>

int main(void) {
    double x = 5;

    printf("x = %f, adresa od x = %p\n", x, &x);
    return 0;
}
```

Ovaj program ispisuje **sadržaj** i **adresu (&x)** varijable **x**.
Napomena: **%p** je znak konverzije za ispis adresa (**pointer**).

Primjer 7 (nastavak)

Stvarna **adresa** varijable **x**, naravno, ovisi o računalu, tj. ne mora biti ista na svakom računalu.

Uzmimo da je (u trenutku izvršavanja programa) varijabla **x** smještena na **adresi** **0065FDF0** (heksadecimalno). Onda će naš program ispisati:

x = 5.000000, adresa od x = 0065FDF0

Adresa je, također, ispisana **heksadecimalno** (po **%p**).

Adresu varijable nekog **tipa** možemo zapamtiti u varijabli koja je **tipa**

- pokazivač ili pointer na dani tip.

Takva varijabla sadrži **adresu** objekta određenog **tipa**.

Primjer 7 (nastavak)

Uočiti: adresa je, općenito, vezana za tip podatka koji je smješten na toj adresi \iff kako se “čita sadržaj” te adrese!

Na primjer, ovaj program radi isto kao i prethodni.

```
#include <stdio.h>

int main(void) {
    double x = 5;
    double *px;

    px = &x;
    printf("x = %f, adresa od x = %p\n", x, px);
    return 0;
}
```

Deklaracija pokazivača (pointera)

Deklaracija varijable pokazivača na neki tip, tj. varijable koja sadrži adresu objekta nekog tipa, ima oblik:

```
tip *ime;
```

- **ime** je varijabla tipa pokazivač na **tip** i služi pamćenju adresa varijabli tipa **tip**.

Primjeri:

```
double y, x = 5; /* varijable tipa double */  
double *px;        /* pokazivac na tip double */  
int i, *m;         /* i je varijabla tipa int */  
                    /* m je pokazivac na int */
```

Deklaracija pokazivača (nastavak)

Pokazivač se može **inicijalizirati** adresom varijable:

```
px = &x; /* &x je adresa varijable x */
```

- Operator dereferenciranja ***** daje **sadržaj** lokacije na koju pokazivač pokazuje (**sadržaj** na pripadnoj **adresi**).

Primjer:

```
double x = 5, y; /* varijable tipa double */  
double *px; /* pokazivac na tip double */  
  
px = &x; /* px = adresa var. x */  
y = *px; /* y = vrijednost var. x (=5) */
```

*Simbol * ima različita značenja*

Uočiti: simbol (znak) ***** ima različita značenja, ovisno o tome da li se nalazi u **deklaraciji** ili **izvršnoj naredbi**.

- U **deklaraciji**, znak ***** ukazuje da je varijabla **pokazivač** na neki tip.
- U **izvršnoj naredbi**, znak ***** je **operator dereferenciranja pokazivača**, tj. daje **sadržaj** pripadne **adrese**.

Na kraju, primijetite da naredbu iz prethodnog programa

```
printf("x = %f, adresa od x = %p\n", x, px);
```

možemo napisati i u obliku

```
printf("x = %f, adresa od x = %p\n", *px, px);
```
