
Funkcije

Primjer: Izračunati "m povrh n"

$$\binom{m}{n} = \frac{m!}{n! \cdot (m-n)!}$$

Rješenje: m povrhn - (1.dio)

MPovrhNBezFunkcije

```
#include <stdio.h>
int main () {
    int m, n, i;
    double brojnik, nazivl, nazivd, mpovrh;

    /* unos vrijednosti za m i n */
    printf ("Unesite m i n:");
    scanf ("%d %d", &m, &n);
```

3

Rješenje: m povrhn - (2.dio)

```
    brojnik = 1;
    for (i = 1; i <= m; i++)
        brojnik *= i;
    nazivl = 1;
    for (i = 1; i <= n; i++)
        nazivl *= i;
    nazivd = 1;
    for (i = 1; i <= m-n; i++)
        nazivd *= i;
    mpovrh = brojnik/(nazivl*nazivd);
    printf("%d povrhn %d iznosi = %g\n", m, n,
           mpovrh);
    return 0;
}
```

4

Komentar prethodnog rješenja

- Sličan programski odsječak ponavlja se 3 puta.
- *Nedostaci:*
 - broj linija programskog koda raste
 - povećava se mogućnost pogreške
- *Preporuka:* program razdvojiti u logičke cjeline koje obavljaju određene, jasno definirane poslove.

5

Rješenje s korištenjem funkcije

```
MPovrhNSFunkcijom
/* Funkcija za računanje faktoriijela */
#include <stdio.h>
double fakt (int n) {
    int i;
    double f;
    for (f = 1, i = 1; i <= n; i++)
        f *= i;
    return f;
}
```

6

Rješenje s korištenjem funkcije - nastavak

```
/* Računanje m povrh n - glavni program */
int main () {
    int m, n;
    double mpovrh;
    printf ("Unesite m i n:");
    scanf ("%d %d", &m, &n);
    mpovrh = fakt (m)/(fakt(n)*fakt(m-n));
    printf("%d povrh %d iznosi = %g\n", m, n,
           mpovrh);

    return 0;
}
```

7

Definicija funkcije:

```
tip_fun ime_fun(tip1 arg1, tip2 arg2, ...) {
    tijelo funkcije: def. varij. i naredbe
}
```

Primjer:

```
int veci (int a, int b) {
    int c;
    c = a > b ? a : b;
    return c;
}
```

rezultat funkcije je int
(tj. tip funkcije je int)

formalni argumenti

definicija varijable

naredba za povratak
(programski slijed i rezultat)

8

Poziv funkcije:

```
int x, y, a, b, c, d;  
x = 7;  
y = 4;
```

```
a = veci (x, y*2);
```

```
b = 5 * veci (3, 4);
```

```
c = veci(3, 4) - veci(7, 8);
```

```
d = veci(3, veci(4, 5));
```

```
veci(7, 8); /* ispravno, iako beskorisno */
```

```
/* primjer funkcije koja vraća rezultat, ali ga najčešće zanemarujemo */
```

```
printf("Poruka\n");
```

```
int veci (int a, int b) {  
    int c;  
    c = a > b ? a : b;  
    return c;  
}
```

formalni
argumenti

stvarni
argumenti

- stvarni argumenti mogu biti izrazi
- rezultat funkcije se može koristiti u izrazima

9

Pretpostavljeni (*default*) tip funkcije

- ako se pri definiciji funkcije ne navede tip funkcije, podrazumijeva se da je funkcija tipa `int`

```
int kvadrat (int n) {  
    return n*n;  
}
```



isto

```
kvadrat (int n) {  
    return n*n;  
}
```

10

void funkcija

- Funkcija koja ne vraća rezultat. Na primjer:

```
void pisi_koordinate (int x, int y) {  
    printf ("x=%d y=%d\n", x, y);  
}
```

poziv funkcije:

```
pisi_koordinate (2, 3);
```

11

Funkcija koja nema argumenata

Nema argumenata, ali vraća rezultat. Npr.:

```
double vratiPi(void) {  
    return 3.1415926;  
}
```

poziv funkcije:

```
double povrsina, r = 3.0;  
povrsina = r * r * vratiPi();
```

Nema argumenata i ne vraća rezultat. Npr.:

```
void pisiPoruku(void) {  
    printf("C je super programski jezik\n");  
}
```

poziv funkcije:

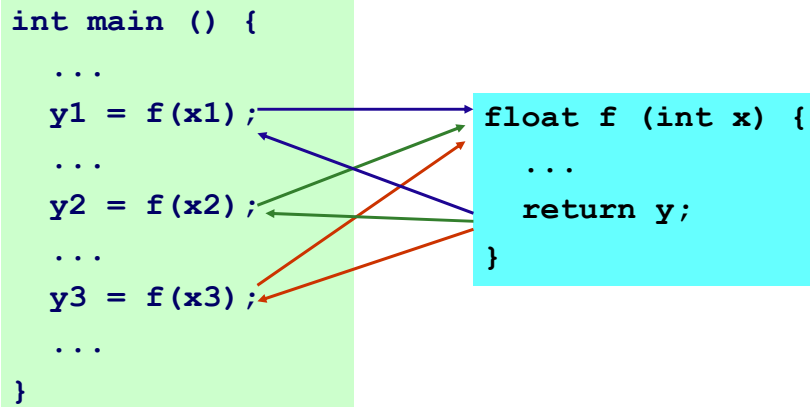
```
pisiPoruku();
```

12

Programski slijed pri pozivu funkcije

```
int main () {  
    ...  
    y1 = f(x1);  
    ...  
    y2 = f(x2);  
    ...  
    y3 = f(x3);  
    ...  
}
```

```
float f (int x) {  
    ...  
    return y;  
}
```



13

Naredba return

- naredba **return** služi za povrat rezultata i nastavljanje programa na mjestu s kojeg je funkcija pozvana
`return;` ili `return izraz;`
- program će se nastaviti obavljati na mjestu s kojeg je funkcija pozvana i u slučaju nailaska na kraj tijela funkcije

```
void ispis (int x) {  
    printf("x=%d\n", x);  
    return;  
}
```



```
void ispis (int x) {  
    printf("x=%d\n", x);  
}
```

14

Naredba `return`

- ako funkcija treba vratiti rezultat (tj. nije `void`), a ne obavi se odgovarajuća `return` naredba, rezultat funkcije je nedefiniran

```
double vratiPi(void) {  
    double pi = 3.1415926;  
    return;  
}
```

prevodilac će upozoriti, ali
prevođenje će ipak uspjeti.

```
double vratiPi(void) {  
    double pi = 3.1415926;  
}
```

- u oba slučaja, rezultat funkcije nakon poziva bio bi nedefiniran:

```
double rez;  
rez = vratiPi();
```

15

Naredba `return`


- u tijelu funkcije smije biti više `return` naredbi

```
double vratiAbs(double a) {  
    if (a >= 0.0)  
        return a;  
    else  
        return -a;  
}
```

16

Tip podatka u naredbi `return`

- Ako tip podatka u naredbi `return` ne odgovara tipu funkcije, **automatski** se obavlja pretvorba tipa. Pretvorba tipa slična je pretvorbi koja se obavlja kod pridruživanja.

```
double kvadrat (short a) {  
    int p;  
    p = a*a;  
    return p;  return (double)p;  
}
```

isto

- primjer:

```
long kvadrat (int n) {  
    return (long)n*n;  
}
```

uočite zašto je ovdje cast. Nije radi toga što je funkcija tipa long!

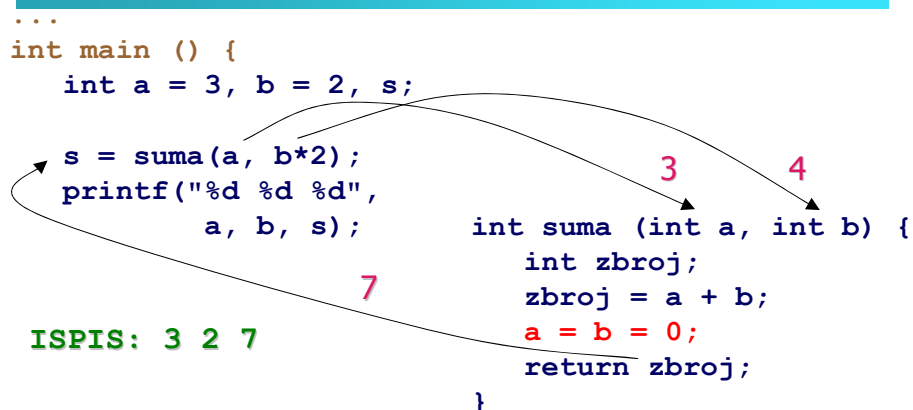
17

Formalni i stvarni argumenti

```
...  
int main () {  
    int a = 3, b = 2, s;  
    s = suma(a, b*2);  
    printf("%d %d %d",  
          a, b, s);  
}
```

```
int suma (int a, int b) {  
    int zbroj;  
    zbroj = a + b;  
    a = b = 0;  
    return zbroj;  
}
```

ISPIS: 3 2 7



- prenose se **vrijednosti** (tj. "kopije") stvarnih argumenata
- izmjena vrijednosti formalnih argumenata **ne utječe** na stvarne argumente (`a = b = 0` nije promijenila varijablu `a` u `main`)
- unutar funkcije se formalni argumenti mogu koristiti jednako kao bilo koje druge "normalne" varijable

18

Tipovi podataka formalnih i stvarnih argumenata

- Ako tip stvarnog argumenta ne odgovara tipu formalnog argumenta, stvarni argument se pri pozivu funkcije **automatski** pretvara u tip koji odgovara tipu formalnog argumenta. Pretvorba tipa slična je pretvorbi koja se obavlja kod pridruživanja.
- Stvarni argumenti moraju odgovarati formalnima po broju i po formi (pretvorba tipa mora biti izvediva), npr.
 - ako je formalni arg. tipa int, stvarni arg. može biti tipa float
 - ako je formalni arg. tipa int, stvarni arg. ne smije biti polje
 - itd.

19

Primjer: konverzija tipova argumenata pri pozivu funkcije

```
#include <stdio.h>
int veci(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}

int main() {
    int a[10] = {10};
    printf("%d", veci(1, 2));           → 2
    printf("%d", veci(4.5, 5.5f));     → 5
    printf("%d", veci('A', 63));       → 65
    printf("%d", veci(9, a));          ← ispisat će se adresa člana a[0]
    return 0;
}
```

20

Funkcije koje trebaju vratiti više vrijednosti

Primjer: napisati funkciju za izračunavanje sume i produkta dva cijela broja

- Ideja (loša!): predati funkciji dvije "varijable" u koje će funkcija "pohraniti" izračunatu sumu i produkt

...

```
void suma_prod(int x, int y, int suma, int prod) {
    suma = x + y;
    prod = x * y;
    return;
}
```

```
int main() {
    int x = 3, y = 4, suma, prod;
    suma_prod(x, y, suma, prod);
    printf("x=%d y=%d suma=%d prod=%d\n", x, y, suma, prod);
    return 0;
}
```

x=3 y=4 suma=-82736442 prod=7623423

"smeće" - ispisuju se neinicijalizirane varijable

21

Koje vrijednosti poprimaju varijable, stvarni i formalni argumenti

nakon obavljanja naredbe broj:

	main				suma_prod			
	x	y	suma	prod	x	y	suma	prod
1	3	4	?	?				
2	3	4	?	?	3	4	?	?
3	3	4	?	?	3	4	7	?
4	3	4	?	?	3	4	7	12
5	3	4	?	?				

- NIJE USPJELO! ZAŠTO? Kako riješiti taj problem?

22

Programski slijed pri pozivu funkcije

```
int main () {  
  ...  
  y1 = f(x1);  
  ...  
  y2 = f(x2);  
  ...  
  y3 = f(x3);  
  ...  
}
```

```
float f (float x) {  
  ...  
  return y;  
}
```

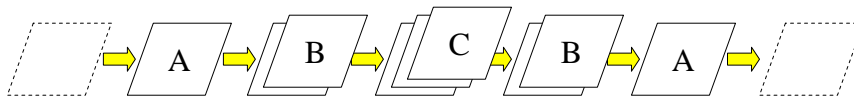
Kako se prenosi x?

Kojim se putem vratiti?

23

Stog (*stack*)

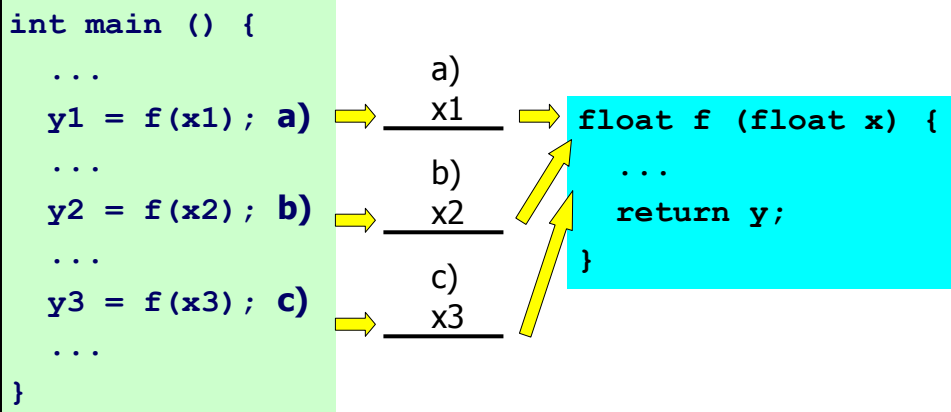
- Dio memorije koji služi za privremeni smještaj varijabli i povratnih adresa
- Struktura podataka tipa LIFO (Last In First Out)



- Pozivajući program na stog postavlja vrijednosti argumenata i povratnu adresu

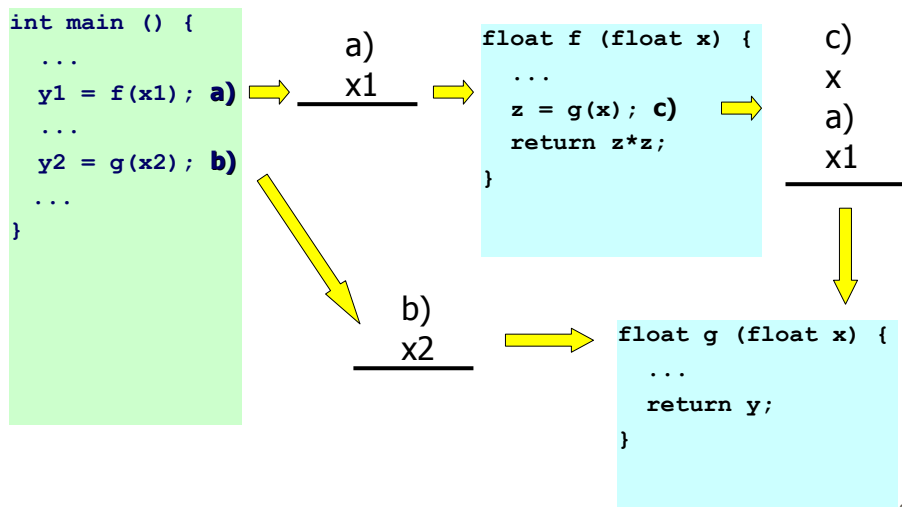
24

Stog



25

Programski slijed i stog pri pozivu funkcije – složeniji primjer



26

Načini prijenosa argumenata u funkciju

- *call by value* – poziv predavanjem **vrijednosti** argumenata (funkcija dobiva svoje vlastite kopije argumenata)
- *call by reference* - poziv predavanjem adresa argumenata
 - takav način predaje argumenata postoji npr. u Pascalu. Strogo promatrano, u C-u **ne postoji** *call by reference*. U C-u se taj mehanizam može **nadomjestiti** predajom pokazivača kao argumenata funkcije

27

Primjer: napisati funkciju za izračunavanje sume i produkta dva cijela broja

- Ideja (**dobra!**): predati funkciji dvije vrijednosti pokazivača (pokazivače na varijable `suma` i `prod` iz glavnog programa)

...

```
void suma_prod(int x, int y, int *psuma, int *pprod) {  
    *psuma = x + y; 4  
    *pprod = x * y; 5  
    return; 6  
}
```

```
int main() {  
    int x = 3, y = 4; 1  
    int suma, prod; 2  
    suma_prod(x, y, &suma, &prod); 3  
    printf("x=%d y=%d suma=%d prod=%d\n", x, y, suma, prod);  
    return 0; 4  
}
```

x=3 y=4 suma=7 prod=12

28

Koje vrijednosti poprimaju varijable, stvarni i formalni argumenti

- pretpostavka o adresama varijabli: &suma=85610, &prod=85614

nakon obavljanja naredbe broj:

	main				suma_prod			
	x	y	suma	prod	x	y	psuma	pprod
1	3	4						
2	3	4	?	?				
3	3	4	?	?	3	4	85610	85614
4	3	4	7	?	3	4	85610	85614
5	3	4	7	12	3	4	85610	85614
6	3	4	7	12				

29

Kada kao argumente treba koristiti pokazivače

- Kada pozvana funkcija treba direktno izmijeniti vrijednost jedne ili više varijabli iz pozivajuće funkcije (primjeri su funkcije `suma_prod` i `uduplaj2`, te funkcija **zamijeni** na sljedećem slajdu)

```
int uduplaj1(int x) {
    x *= 2;
    return x;
}

int main() {
    int broj = 10;
    broj = uduplaj1(broj);
    printf("%d\n", broj);
    return 0;
}
```

```
void uduplaj2(int *x) {
    *x *= 2;
    return;
}

int main() {
    int broj = 10;
    uduplaj2(&broj);
    printf("%d\n", broj);
    return 0;
}
```

30

Primjer: napisati funkciju koja zamjenjuje sadržaj dviju varijabli tipa `short`

CallByReference

```
#include <stdio.h>
void zamijeni (short *x, short *y) {
    short pom;
    pom = *x;
    *x = *y;
    *y = pom;
    return;
}

int main () {
    short a = 3, b = 5;
    zamijeni (&a, &b);
    printf ("Poslije zamjene: %d %d\n", a, b);
    return 0;
}
```

31

Koje vrijednosti poprimaju varijable, stvarni i formalni argumenti

- pretpostavka o adresama varijabli: `&a=64720`, `&b=64722`

nakon obavljanja naredbe broj:

	main		zamijeni		
	a	b	x	y	pom
1	3	5			
2	3	5	64720	64722	
3	3	5	64720	64722	?
4	3	5	64720	64722	3
5	5	5	64720	64722	?
6	5	3	64720	64722	?
7	5	3			

32

Primjer funkcije koja računa aritmetičku sredinu

- Napisati funkciju koja izračunava aritmetičku sredinu za tri zadana realna broja i glavni program koji učitava tri broja i korištenjem napisane funkcije izračunava njihovu aritmetičku sredinu.

```
#include <stdio.h>
float arit_sred( float a, float b, float c ) {
    float ar;
    ar = ( a + b + c ) / 3;
    return ar; /* Koliko se vrijednosti može vratiti s return */
}
int main() {
    float x, y, z, sred;
    printf("\nUcitaj tri realna broja : ");
    scanf("%f %f %f", &x, &y, &z );
    sred = arit_sred(x,y,z);
    printf("\nAritmeticka sredina unesenih brojeva je : %f",
        sred);
    return 0;
}
```

33

Primjer: izračunavanje funkcije sinus pomoću sume n članova reda

- Napisati funkciju koja će za zadani argument u radianima izračunati vrijednost funkcije sinus kao sumu n članova reda. Funkcija sinus je definirana redom:

$$\sin(x) = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^{2i-1}}{(2i-1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

34

Realizacija bez korištenja pomoćnih funkcija

```
#include <stdio.h>
/* realizacija bez korištenja pomoćnih funkcija */
float sinus(float x, int n) {
    int    i, predznak;
    float  sum, clan, fakt, xpot;
    sum = 0.0;
    xpot = x;
    fakt = 1.0;
    predznak = 1;
    for( i=1; i<=n; i++ ) {
        clan = predznak * xpot / fakt;
        predznak *= -1;
        xpot *= x*x;
        fakt *= (2*i) * (2*i+1);
        sum += clan;
    }
    return sum;
}
```

35

Realizacija uz korištenje pomoćnih funkcija

```
#include <stdio.h>
#include <math.h>
/* realizacija s korištenjem pomoćnih funkcija */
long fakt( int n ) {
    int    i;
    long  f = 1;
    for( i = 1; i <= n; i++ )
        f *= i;
    return f;
}

float sinus(float x, int n) {
    int i, predznak;
    float sum, clan;
    sum = 0.0;
    predznak = 1;
    for( i = 1; i <= n; i++ ) {
        clan = predznak * pow(x, 2*i-1) / fakt(2*i-1);
        predznak *= -1;
        sum += clan;
    }
    return sum;
}
```

36

Primjer s *call by value*

CallByValue

```
#include <stdio.h>
void f (int y) {
    printf ("y u funkciji: %d\n", y);
    y = 2;
    printf ("y u funkciji nakon pridruzivanja: "
           "%d\n", y);
}
int main () {
    int x;
    x = 1;
    f (x);
    printf ("x u programu nakon povratka: %d\n", x);
    return 0;
}
```

37

Rezultat izvođenja

```
y u funkciji: 1
y u funkciji nakon pridruzivanja: 2
x u programu nakon povratka: 1
```

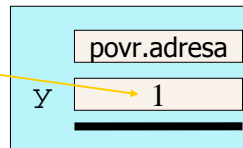
38

Izvođenje - poziv funkcije

Glavni program

x 1

Stog



Napomena: ovisno o prevodiocu i njegovim opcijama, na stog se nakon povratne adrese stavljaju i neki registri procesora, što čini tzv. okvir stoga (*stack frame*). Radi općenitosti, okvir stoga se neće razmatrati.

39

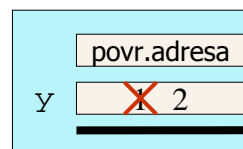
Izvođenje – obavljanje naredbe $y = 2$

Glavni program

x 1

Funkcija

Stog



40

Što će se ispisati nakon obavljanja programa koji poziva funkciju ?

- Što će se ispisati nakon obavljanja programa koji poziva funkciju ?

```
void uduplaj1(int x) {
    printf ("\nF:Ulazni argument prije izmjene je %d ", x);
    x *= 2;
    printf ("\nF:Ulazni argument nakon izmjene je %d ", x);
}

int main() {
    int broj=10;
    printf ("\nG:Broj prije poziva funkcije je %d ", broj);
    uduplaj1 (broj);
    printf("\nG:Broj nakon poziva funkcije je %d ", broj);
    return 0;
}
```

41

Obrazloženje rezultata izvođenja

- *Rezultat ispisa je:*

```
G:Broj prije poziva funkcije je 10
F:Ulazni argument prije izmjene je 10
F:Ulazni argument nakon izmjene je 20
G:Broj nakon poziva funkcije je 10
```

- Promjena unutar funkcije nije zapamćena nakon povratka u glavni program ! Zašto ?

42

Način vraćanja funkcijske vrijednosti

```
int uduplaj2(int x) {
    printf ("\nF:Ulazni argument prije izmjene je %d ", x);
    x *= 2;
    printf ("\nF:Ulazni argument nakon izmjene je %d ", x);
    return x;
}

int main() {
    int broj=10;
    printf ("\nG:Broj prije poziva funkcije je %d ", broj);
    broj = uduplaj2 (broj);
    printf("\nG:Broj nakon poziva funkcije je %d ", broj);
    return 0;
}
```

G:Broj prije poziva funkcije je 10
F:Ulazni argument prije izmjene je 10
F:Ulazni argument nakon izmjene je 20
G:Broj nakon poziva funkcije je 20

43

Little Endian ili Big Endian?

```
>Endian
#include <stdio.h>
int main () {
    int a=511;
    unsigned char *p;
    p = (char *) &a; // ili p=(unsigned char *) &a;
    printf ("%d %d %d %d\n", *p, *(p+1), *(p+2),
            *(p+3));

    return;
}
```

Ispis: 255 1 0 0

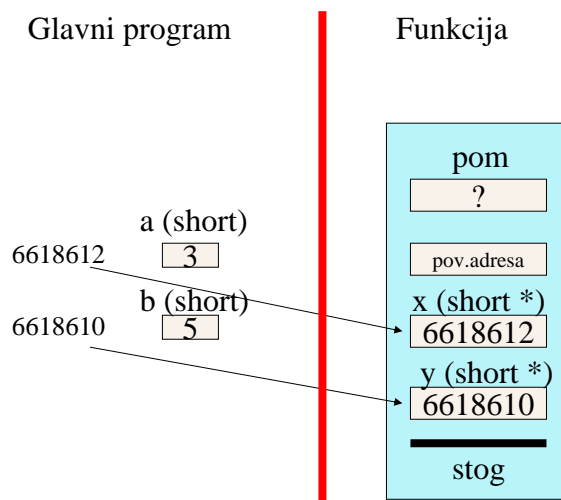
44

Primjer poziva funkcije predavanjem adresa argumenata (*call by reference*)

```
CallByReference
#include <stdio.h>
void zamijeni (short *x, short *y) {
    short pom;
    pom = *x;
    *x = *y;
    *y = pom;
}
int main () {
    short a, b;
    a = 3;
    b = 5;
    printf ("Prije zamjene: %d %d\n", a, b);
    zamijeni (&a, &b);
    printf ("Poslije zamjene: %d %d\n", a, b);
    return 0;
}
```

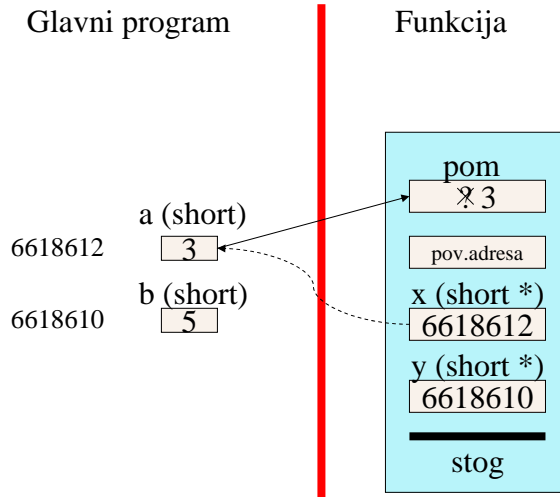
45

Izvođenje - poziv funkcije



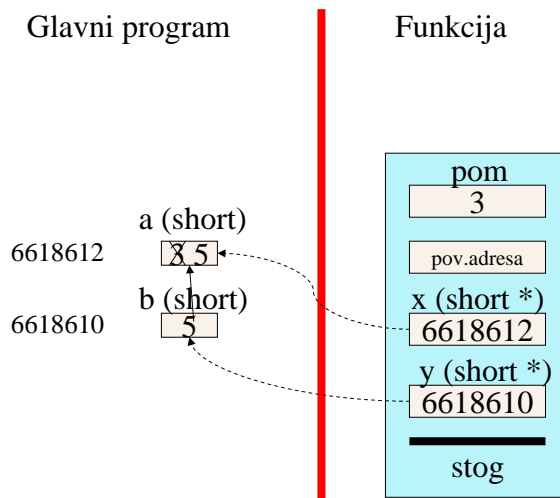
46

Izvođenje - naredba `pom = *x:`



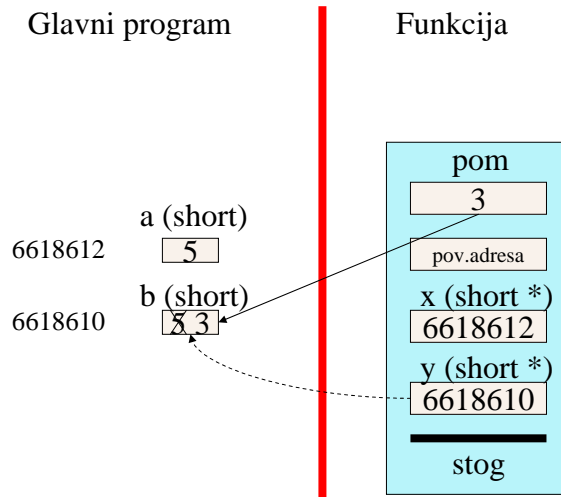
47

Izvođenje - naredba `*x = *y`



48

Izvođenje - naredba `*y = pom;`



49

Organizacija složenijih programa

a) Funkcije smještene u jednoj datoteci (modulu)

```
glavni.c
#include <stdio.h>
int f1 (double a, double b) {
    ...
}

double f2 (int c) {
    ...
}

int main () {
    double x;
    int y;
    x = f2 (15);
    y = f1 (2, 5);
}
```

Zašto smo do sada funkcije uvijek smještali **ispred** main funkcije?

Kada prevodilac naiđe na poziv funkcija f1 i f2, već mu je poznato koje argumente funkcije primaju i kakve tipove podataka vraćaju.

50

Organizacija složenijih programa

a) Funkcije smještene u jednoj datoteci (modulu)

```
glavni.c
#include <stdio.h>

int main () {
    double x;
    int y;
    x = f2 (15);
    y = f1 (2, 5);
}

int f1 (double a, double b) {
    ...
}

double f2 (int c) {
    ...
}
```

Kada prevodilac naiđe na poziv funkcija f1 i f2, može samo pretpostaviti da su f1 i f2 tipa int, ta da im je pri pozivu poslan ispravan broj i tip argumenata. Program se neće uspjeti prevesti ili neće raditi ispravno. Prevodiocu bi na neki način trebalo opisati tip i argumente funkcije prije nego naiđe na poziv dotične funkcije.

51

Prototip (deklaracija) funkcije

- Omogućuje prevodiocu kontrolu tipa funkcije, te broja i tipa argumenata.

```
tip_fun ime_fun (tip1 arg1, tip2 arg2, ...);
tip_fun ime_fun (tip1, tip2, ...);
```

Primjer:

```
double fakt (int n);
double fakt (int);
int veci (int a, int b);
int veci (int, int);
void pisi_koordinate (int x, int y);
double vratiPi (void);
void pisiPoruku (void);
void suma_prod (int x, int y, int *psuma, int *pprod);
```

bolje!!!

52

Organizacija složenijih programa

a) Funkcije smještene jednoj datoteci (uz prototipove)

```
glavni.c
#include <stdio.h>
int f1 (double a, double b);
double f2 (int c);

int main () {
    double x;
    int y;
    x = f2 (15);
    y = f1 (1, 5);
}

int f1 (double a, double b) {
    ...
}

double f2 (int c) {
    ...
}
```

Kada prevodilac naiđe na poziv funkcija f1 i f2, prema prototipu može provjeriti jesu li pozvane na ispravan način. Kada prevodilac naiđe na definiciju funkcija f1 i f2, prema prototipu može provjeriti jesu li ispravno definirane.

53

Organizacija složenijih programa

b) Funkcije smještene u više datoteka (modula)

```
funkcije.c
int f1 (long a, long b) {
    ...
}
double f2 (int c) {
    ...
}
```

```
glavni.c
void f3 (int d) {
    ...
}
int main () {
    y = f1 (a1, b1);
}
```

Kako sada provjeriti je li poziv funkcije korektno napisan?

54

Organizacija složenijih programa

c) Funkcije smještene u više datoteka (modula) s prototipovima

```
prototip.h  
int f1 (long a, long b);  
double f2 (int c);  
void f3 (int d);
```

```
funkcije.c  
#include "prototip.h"  
int f1 (long a, long b) {  
    ...  
}  
double f2 (int c) {  
    ...  
}
```

```
glavni.c  
#include "prototip.h"  
void f3 (int d) {  
    ...  
}  
int main () {  
    ...  
    y = f1 (a1, b1);  
}
```

55

Definicija/deklaracija

- Do sada se govorilo samo o DEFINICIJI varijable: definicijom varijable određuje se ime varijable, tip varijable, **te rezervira područje u memoriji** u kojem će varijabla biti pohranjena.
- Do sada su se varijable definirale isključivo UNUTAR funkcije ili bloka unutar funkcije. Tako definirane varijable se koriste isključivo UNUTAR funkcije ili bloka u kojem su definirane, a njihova vrijednost se GUBI u trenutku završetka funkcije ili bloka. Vrijednost takvih varijabli je uvijek nepoznata ("smeće") do trenutka kada se varijabli pridruži vrijednost.

56

Definicija/deklaracija

Primjer:

```
int main () {
    int a = 10, b, *p1 = &a;
    float pp[3] = {1.0f, 2.0f, 3.0f}, *p2;
    ...
    *{
        int i;
        ...
    *}
}
```

- za varijable **b**, **p2**, **i** rezervirano je područje u memoriji, ali je trenutna vrijednost varijabli nepoznata. Vrijednost ostalih varijabli je poznata
- varijable **a**, **b**, **p1**, **pp**, **p2** mogu se koristiti isključivo unutar funkcije u kojoj su definirane (u ovom slučaju, unutar funkcije **main**). Varijabla **i** se može koristiti samo unutar bloka označenog sa *
- sadržaj varijabli **a**, **b**, **p1**, **pp**, **p2** se gubi završetkom funkcije
- sadržaj varijable **i** se gubi završetkom bloka označenog sa *

57

Definicija/deklaracija

- **Deklaracija varijable:** uputa (objava) prevodiocu - postoji (tj. negdje je definirana) varijabla s navedenim imenom i tipom
 - Deklaracija funkcije: uputa (objava) prevodiocu: postoji (negdje je definirana) funkcija s navedenim imenom, tipom i argumentima. Deklaracija funkcije → prototip funkcije
- Deklaracija iste varijable (funkcije) može se pojaviti više puta u istom programu, dok se definicija varijable (funkcije) smije pojaviti samo jednom
- Definicijom varijable (funkcije) ujedno se ta varijabla (funkcija) i deklarira (na mjestu na kojem se nalazi definicija)
- Nasuprot tome, deklaracijom se varijabla (funkcija) ne definira

58

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

Općenito:

```
smještajni_razred tip_podatka varijabla ...;
```

- `smještajni_razred` i mjesto definicije varijable određuju postojanost i područje važenja varijable u memoriji.
- **postojanost varijable (trajnost, *duration*)**
 - određuje područje programskog kôda tijekom čijeg izvršavanja sadržaj varijable ostaje sačuvan
- **područje važenja varijable (doseg, *scope*)**
 - određuje područje programskog kôda unutar kojeg se varijabla može referencirati ("unutar kojeg se varijabla može koristiti")

Smještajni razredi: `auto`, `register`, `static`, `extern`

59

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

auto - automatski smještajni razred ("lokalne" varijable)

- područje važenja varijable i njena trajnost: od mjesta definicije do kraja funkcije ili bloka unutar kojeg je varijabla definirana
- automatske varijable uobičajeno se nazivaju lokalne varijable (lokalne u funkciji, lokalne u bloku).
- podrazumijeva se, ako eksplicitno nije drugačije navedeno, da je svaka varijabla definirana UNUTAR funkcije, razreda `auto`. Varijablu razreda `auto` moguće je definirati jedino unutar funkcije (bloka)

```
int main () {  
    auto int i;  
    i = 7;  
    {  
        auto int j = 3;  
    }  
}
```



isto

```
int main () {  
    int i;  
    i = 7;  
    {  
        int j = 3;  
    }  
}
```

60

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

- formalni argumenti funkcije imaju ista svojstva kao varijable smještajnog razreda `auto`. Jedina razlika je u tome što se formalnim argumentima prilikom poziva funkcije pridružuje vrijednost stvarnih argumenata.

61

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

register - registarski smještajni razred

- predstavlja preporuku prevodiocu da, ukoliko je moguće, vrijednost varijable pohrani u CPU registar.
- područje važenja varijable i njena trajnost određeni su na isti način kao za varijablu razreda `auto`
- **register** varijablu moguće je definirati jedino unutar funkcije (bloka)

```
int main () {  
    register int i;  
    double fact = 1.0;  
    for (i = 1; i < 15; i++) {  
        fact *= i;  
    }  
}
```

62

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

static - statički smještajni razred

- područje važenja varijable
 - ako je varijabla definirana unutar funkcije (bloka): od mjesta na kojem je definirana do kraja funkcije (bloka)
 - ako je varijabla definirana izvan funkcije: od mjesta na kojem je definirana do kraja modula
- trajnost varijable
 - od početka izvršavanja programa do završetka programa
- ako varijabla nije eksplicitno inicijalizirana tijekom definicije, njena se vrijednost automatski postavlja na 0

63

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

- statičku varijablu treba definirati unutar funkcije ako varijabla treba biti vidljiva samo unutar te funkcije, a istovremeno je potrebno sačuvati vrijednost varijable tijekom više poziva funkcije
- statičku varijablu treba definirati izvan tijela funkcije ako istu varijablu koristi nekoliko funkcija unutar istog modula

64

Primjer: napisati funkciju `zbroj` za zbrajanje dva cijela broja. Funkcija vraća zbroj tijekom prvih tri poziva, a za svaki sljedeći poziv vraća 0.

```
#include <stdio.h>
int zbroj (int a, int b);
int main () {
    printf("%d\n", zbroj(1, 2));
    printf("%d\n", zbroj(3, 4));
    printf("%d\n", zbroj(5, 6));
    printf("%d\n", zbroj(7, 8));
    printf("%d\n", zbroj(9, 10));
}
int zbroj (int a, int b) {
    static int brojPoziva;
    brojPoziva++;
    if (brojPoziva <= 3)
        return a + b;
    else
        return 0;
}
```

ISPIS:
3
7
11
0
0

kolika je vrijednost varijable na početku?

ZADATAK: što bi se ispisalo da je ispuštena riječ `static` u funkciji `zbroj`?

65

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

extern - vanjski smještajni razred ("globalne" varijable)

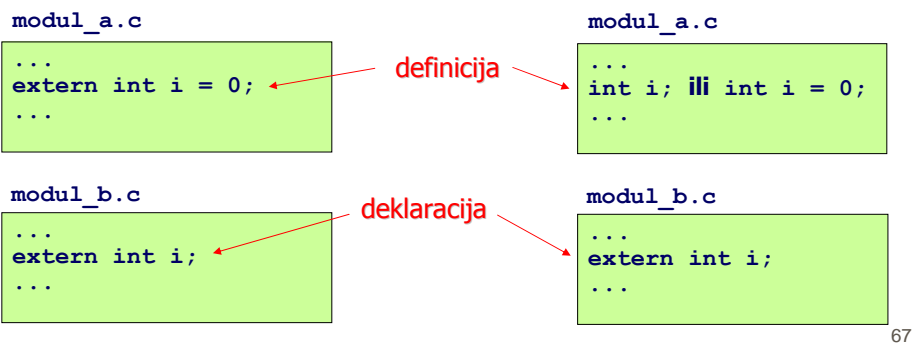
- **extern** se koristi za DEKLARACIJU varijable (osim izuzetno). Predstavlja uputu prevodiocu: objavljujem da je varijabla čije ime i tip opisujem, definirana "negdje drugdje" (podrazumijeva se da varijabla zaista jest definirana negdje drugdje)
- područje važenja varijable
 - ako je varijabla deklarirana unutar funkcije (bloka): od mjesta na kojem je deklarirana do kraja funkcije (bloka)
 - ako je varijabla deklarirana izvan funkcije: od mjesta na kojem je deklarirana do kraja modula
- trajnost varijable (jednako kao varijable razreda `static`)
 - od početka izvršavanja programa do završetka programa
- ako varijabla nije eksplicitno inicijalizirana tijekom definicije, njena se vrijednost automatski postavlja na 0

66

Definicija/deklaracija varijabli - smještajni razredi (*storage classes*) -

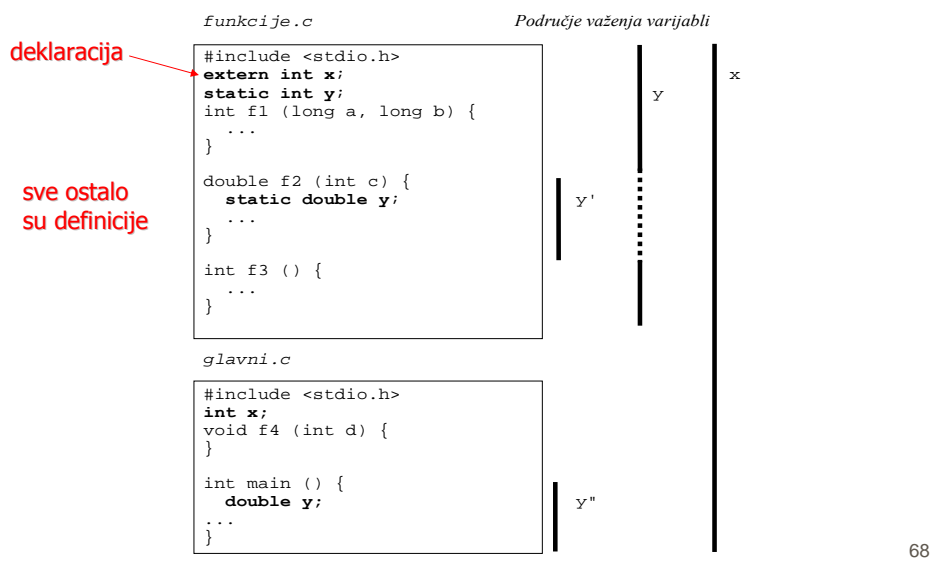
Gdje se nalazi i kako izgleda definicija varijable na koju se poziva **extern** deklaracija?

- izvan funkcije, a unutar istog ili nekog drugog modula
- dva načina **definicije**:
 - koristiti ključnu riječ **extern** uz obaveznu inicijalizaciju
 - ispustiti ključnu riječ **extern**



67

Smještajni razredi i komunikacija funkcija pomoću globalnih varijabli



68

Primjer: u modulu m1.c napisati funkciju `zbroj`, a u modulu m2.c napisati funkciju `prod`. Funkcije vraćaju zbroj, odnosno produkt dvaju cijelih brojeva tijekom prva tri poziva, a za svaki sljedeći poziv vraćaju 0.

```
m1.c
#include "proto.h"
int brojPoziva;
int zbroj (int a, int b) {
    brojPoziva++;
    if (brojPoziva <= 3)
        return a + b;
    else
        return 0;
}
```

```
m2.c
#include "proto.h"
int prod (int a, int b) {
    extern int brojPoziva;
    brojPoziva++;
    if (brojPoziva <= 3)
        return a * b;
    else
        return 0;
}
```

```
proto.h
int zbroj (int a, int b);
int prod (int a, int b);
```

```
glavni.c
#include <stdio.h>
#include "proto.h"
int main () {
    printf("%d\n", zbroj(1, 2));
    printf("%d\n", prod(3, 4));
    printf("%d\n", zbroj(5, 6));
    printf("%d\n", prod(7, 8));
    printf("%d\n", zbroj(9, 10));
}
```

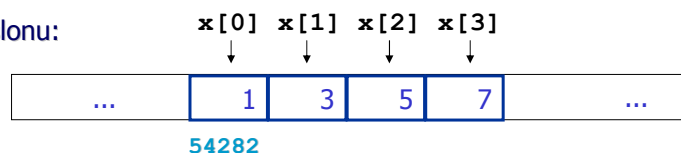
ISPIS:
3
12
11
0
0

69

Ponavljanje: polja i pokazivači

```
#include <stdio.h>
int main () {
    int x[4] = {1, 3, 5, 7};
    int *p = &x[0]; /* 54282 */
    printf("%d %d %d %d", *p, *(p+1), *(p+2), *(p+3));
    return 0;
}
```

Ispis na zaslonu:
1 3 5 7



- Umjesto `&x[0]` može se koristiti `x` (ime 1-dimenzionalnog polja je isto što i adresa prvog člana polja)

70

Polja i pokazivači

PoljaIPokazivaci

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

```
int main () {
```

```
    int x[4] = {1,2,3,4};
```

```
    printf("%d %d\n", *x, *(x+1));
```

```
    f(x);
```

```
    return 0;
```

```
}
```

```
void f (int *x) { ili void f (int x[]) {
```

```
    printf("%d %d\n", *x, x[0]);
```

```
    ++x;
```

```
    printf("%d %d %d\n", *x, x[0], *(x-1));
```

```
}
```

Ispis na zaslonu:

```
1 2
```

```
1 1
```

```
2 2 1
```

	x[0]	x[1]	x[2]	x[3]	
...	1	2	3	4	...
	54282				

71

Jednodimenzionalna polja kao argumenti funkcije

- polje se u funkciju **ne može** prenijeti na isti način kao što se u funkciju prenose ostali tipovi podataka
- umjesto kopije svih elemenata polja, u funkciju se prenosi samo **kopija adrese prvog elementa polja**
- osim u posebnim slučajevima (u primjerima s nizovima znakova), u funkciju je potrebno prenijeti dodatni argument: broj članova polja

72

Primjer: napisati funkciju kojom se zbrajaju članovi jednodimenzionalnog cijelobrojnog polja.

```
#include <stdio.h>
int zbroji (int *p, int n);
int main () {
    int polje[4] = {1, 3, 5, 7};
    int suma;
    suma = zbroji(&polje[0], 4);
    printf("Zbroj je %d\n", suma);
}

int zbroji (int *p, int n) {
    int i, s = 0;
    for (i = 0; i < n; i++)
        s = s + *(p+i);
    return s;
}
```

Izračunat će se:

```
s = *p
+ *(p+1)
+ *(p+2)
+ *(p+3)
```

73

Jednodimenzionalna polja kao argumenti funkcije

Dopušteno je koristiti drugačije oznake (koje međutim imaju isto značenje):

```
#include <stdio.h>
int zbroji (int *p, int n);
int main () {
    int polje[4] = {1, 3, 5, 7};
    int suma;
    suma = zbroji(polje, 4);
    printf("Zbroj je %d\n", suma);
}

int zbroji (int p[], int n) {
    int i, s = 0;
    for (i = 0; i < n; i++)
        s = s + p[i];
    return s;
}
```

Umjesto `&polje[0]`

Umjesto `int *p`

Umjesto `*(p+i)`

74

Primjer: u main funkciji ("glavnom programu") definirati jednodimenzionalno polje od 30 članova. S tipkovnice učitati cijeli broj n , $2 \leq n \leq 30$, te pozvati funkciju koja polje puni s n Fibonaccijevih brojeva

```
#include <stdio.h>
#define MAX 30
void puniFib (int *niz, int n);
void ispisPolja (int *niz, int n);
int main () {
    int polje[MAX], n;
    do
        scanf("%d", &n);
    while (n < 2 || n > MAX);
    puniFib(polje, n);
    ispisPolja(polje, n);
}

void ispisPolja (int niz[],
                int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", niz[i]);
}

void puniFib (int niz[], int n) {
    int i;
    niz[0] = niz[1] = 1;
    for (i = 2; i < n; i++)
        niz[i] = niz[i-2] + niz[i-1];
}
```

sizeof(polje) → 120

sizeof(niz) → 4

75

Primjer: brojanje članova polja s vrijednostima unutar zadanog intervala

Napisati funkciju koja će u jednodimenzionalnom realnom polju pronaći koliko ima brojeva koji su veći od zadane donje granice i istovremeno manji od zadane gornje granice. U slučaju da je funkciji zadan neispravan raspon (tj. ako je donja granica veća ili jednaka gornjoj granici), funkcija u pozivajući program treba vratiti vrijednost -1.

U glavnom programu treba učitati stvarni broj članova n ($1 \leq n \leq 100$) i vrijednosti članova polja. Učitavati vrijednosti donje i gornje granice, pozivati potprogram i ispisivati rezultat, sve dok se granice ispravno zadaju.

76

Jednodimenzionalna polja kao argumenti funkcije - rješenje, I dio

📁BrojanjeClanovaPolja

```
#include <stdio.h>
int main () {
    int n, i, ibr;
    float x[100], dgr, ggr;

    do {
        printf ("Upisite broj clanova polja>");
        scanf ("%d", &n);
    } while (n < 1 || n > 100);
    printf ("Upisite vrijednosti clanova polja >");
    for (i = 0; i < n; i++) {
        scanf ("%f", &x[i]);
    }
}
```

77

Jednodimenzionalna polja kao argumenti funkcije – rješenje, II dio

```
do {
    printf ("Upisite donju i gornju granicu >");
    scanf ("%f %f", &dgr, &ggr);
    ibr = broji(n, x, dgr, ggr);
    if(ibr == -1) {
        printf ("Neispravno zadane granice\n");
        break;
    } else {
        printf ("U polju je pronadjeno %d clanova"
            " vecih od %f i manjih od %f\n", ibr,
            dgr, ggr);
    }
} while (1);

return 0;
}
```

78

Jednodimenzionalna polja kao argumenti funkcije – rješenje, III dio

```
int broji(int n, float polje[], float dg, float gg) {
    int i, ibroj;
    if (dg < gg) {
        for (ibroj = 0, i = 0; i < n; i++) {
            printf ("%f\n", polje[i]);
            if (polje[i] > dg && polje[i] < gg) {
                ++ibroj;
            }
        }
        return ibroj;
    } else {
        return -1;
    }
}
```

79

Ponavljanje: polje znakova kao niz znakova (*string*)

Konstanta "Ovo je niz"

...	O	v	o		j	e		n	i	z	\0	...
-----	---	---	---	--	---	---	--	---	---	---	----	-----

Varijabla: ne postoji tip podatka *string*. Za pohranu niza znakova koristi se jednodimenzionalno polje znakova:

```
char ime[4+1] = {'I', 'v', 'a', 'n', '\0'};
char ime[4+1] = "Ivan";
char ime[] = "Ivan";
```

...	I	v	a	n	\0	...
-----	---	---	---	---	----	-----

80

Polje znakova kao niz znakova (*string*)

```
char ime[] = "Ivan";  
printf("%s", ime);
```



Ivan

Zašto funkciji printf ne moramo predati broj članova polja?
Zato jer printf pomoću '\0' može zaključiti gdje je kraj niza znakova.

```
char ime[4] = {'I', 'v', 'a', 'n'};
```



```
printf("%s", ime)
```

Ivan*)%&/!)=()Z)(B#DW=)(@(\$/"#*!@!/["&/\$/...)

... i nastaviti će se ispisivati dok se ne nađe na oktet u kojem je upisana vrijednost 0x00 (tj. '\0')

81

Primjer: napisati funkciju koja prima niz znakova, ispisuje znak po znak, ali tako da umjesto malih slova ispisuje velika

```
#include <stdio.h>  
void ispisNizaZnakova (char niz[]);  
int main () {  
    char ime[] = "Ivana 123";  
    ispisNizaZnakova (ime);  
}  
  
void ispisNizaZnakova (char niz[]) {  
    int i = 0;  
    while (niz[i] != '\0') {  
        if (niz[i] >= 'a' && niz[i] <= 'z')  
            printf("%c", niz[i] - ('a' - 'A'));  
        else  
            printf("%c", niz[i]);  
        i++;  
    }  
}
```

Može: char *niz
Može: char *niz
Može: *(niz+i)

IVANA 123

82

Primjer: napisati funkciju koja prima niz znakova, ispisuje znak po znak, ali tako da umjesto malih slova ispisuje velika

```
void ispisNizaZnakova (char *niz) {
    while (*niz != '\0') {
        if (*niz >= 'a' && *niz <= 'z')
            printf("%c", *niz - ('a' - 'A'));
        else
            printf("%c", *niz);
        niz++;
    }
}

void ispisNizaZnakova (char *niz) {
    for (; *niz != '\0'; niz++) → for (; *niz; niz++)
        if (*niz >= 'a' && *niz <= 'z')
            printf("%c", *niz - ('a' - 'A'));
        else
            printf("%c", *niz);
}
```

Funkcija se može pozvati i ovako:

```
ispisNizaZnakova("Ivana 123");
```

83

Primjer: napisati funkciju koja prima niz znakova, te sva mala slova unutar niza pretvara u velika

```
#include <stdio.h>
void velikaSlova (char *niz);
int main () {
    char ime[] = "Ivana 123";
    velikaSlova(ime);
    printf("%s", ime);
}

void velikaSlova (char *niz) {
    for (; *niz; niz++)
        if (*niz >= 'a' && *niz <= 'z')
            *niz = *niz - ('a' - 'A');
}
```

84

Primjer: napisati funkciju koja znak po znak uspoređuje dva niza znakova s1 i s2. Za prvi par znakova u kojima se dva niza razlikuju, vraća razliku ASCII vrijednosti ta dva znaka. Ako su nizovi jednaki, funkcija vraća 0

```
strcmp("ABCDEF", "ABCEF") → 'D' - 'E'  
strcmp("ABC", "AB") → 'C' - '\0'  
strcmp("AB", "ABC") → '\0' - 'C'  
strcmp("AB", "AB") → 0
```

```
int strcmp (char *s1, char *s2) {  
    while (*s1 == *s2 && *s1) {  
        s1++;  
        s2++;  
    }  
    return *s1 - *s2;  
}  
  
int strcmp (char *s1, char *s2) {  
    for (; *s1 == *s2 && *s1; s1++, s2++);  
    return *s1 - *s2;  
}
```

85

Dvodimenzionalna i višedimenzionalna polja kao argumenti funkcije

```
#include <stdio.h>  
int zbroji (int p[][] ...) {  
    ...  
    return s;  
}  
  
int main () {  
    int polje[2][3] = {{1, 3, 5},  
                      {7, 8, 9}};  
  
    int suma;  
    suma = zbroji(polje ...);  
    printf("Zbroj je %d\n", suma);  
}
```

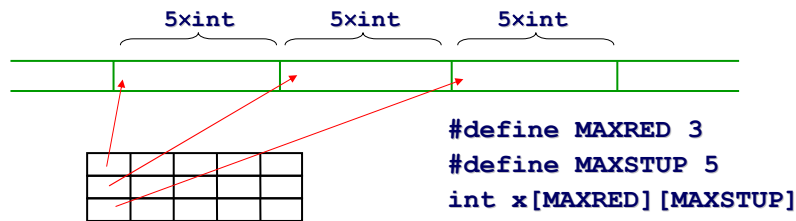
POGREŠKA

Polje se u funkciji može "dočekati" jedino kao pokazivač.

86

Ponavljanje: dvodimenzionalna polja i pokazivači

- Dvodimenzionalno polje: redak za retkom



```
int *p = &x[0][0];
*(p + 0*MAXSTUP + 0) → vrijednost člana x[0][0]
*(p + 0*MAXSTUP + 1) → vrijednost člana x[0][1]
*(p + 2*MAXSTUP + 3) → vrijednost člana x[2][3]
```

Općenito vrijedi (ako je p adresa prvog člana polja x):

vrijednost člana `x[i][j]` → `*(p + i*MAXSTUP + j)`

87

Pristup elementu dvodimenzionalnog polja

Za pristup elementu iz n -tog retka treba prvo preskočiti $n-1$ punih redaka. Ako je polje **definirano** na sljedeći način:

```
polje[MAXRED][MAXSTUP] → broj članova u jednom retku
```

tada se unutar funkcije, koja je za vrijednost argumenta `p` dobila kopiju pokazivača na prvi element polja `polje`, elementu `polje[i][j]` može pristupiti na sljedeći način:

```
p[i*MAXSTUP + j]
```

ili

```
*(p + i*MAXSTUP + j)
```

88

Primjer: napisati funkciju za zbrajanje članova dvodimenzionalnog cjelobrojnog polja

```
#include <stdio.h>
int zbroji (int p[], int brRed, int maxStup) {
    int i, j, s = 0;
    for (i = 0; i < brRed; i++)
        for (j = 0; j < maxStup; j++)
            s = s + p[i*maxStup + j];
    return s;
}

int main () {
    int polje[2][3] = {{1, 3, 5},
                      {7, 8, 9}};

    int suma;
    suma = zbroji(&polje[0][0], 2, 3);
    printf("Zbroj je %d\n", suma);
}
```

Izračunat će se:

```
s = *(p+0*3+0)
    + *(p+0*3+1)
    + *(p+0*3+2)
    + *(p+1*3+0)
    + *(p+1*3+1)
    + *(p+1*3+2)
      ↑   ↑
      i   j
```

dopušteno je napisati `polje[0]`

89

Primjer: funkcija za zbrajanje članova dvodimenzionalnog cjelobrojnog polja - korištenje drugačijih oznaka

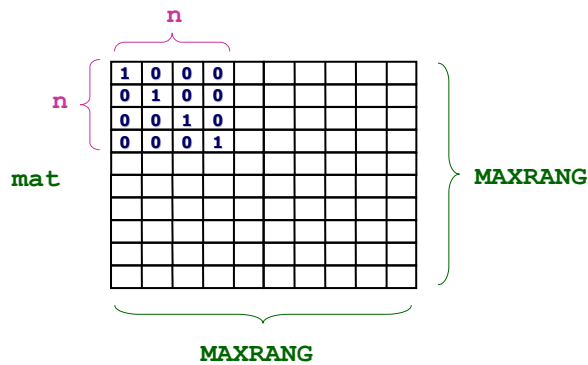
```
int zbroji (int *p, int brRed, int maxStup) {
    int i, j, s = 0;
    for (i = 0; i < brRed; i++)
        for (j = 0; j < maxStup; j++)
            s = s + *(p + i*maxStup + j);
    return s;
}
```

90

Primjer za dvodimenzionalno polje kao argument funkcije: formiranje jedinične matrice

Napisati funkciju za formiranje jedinične matrice ranga N, gdje je N proizvoljan prirodni broj. U glavnom programu definirati matricu, učitati rang matrice ≤ 10 , pozvati funkciju i ispisati generiranu matricu.

Npr: ako se zada da treba generirati matricu ranga $n=4$, treba se dobiti:



91

Rješenje - parametrizacija i zadavanje ranga

GeneriranjeJedinicneMatrice

```
#include <stdio.h>
#define MAXRANG 100
int main () {
    int m[MAXRANG][MAXRANG], n, i, j;

    do {
        printf ("Zadajte rang matrice iz "
            "intervala [1,%d] !\n", MAXRANG);
        scanf("%d", &n);
    } while (n < 1 || n > MAXRANG);
```

92

Rješenje - poziv funkcije i ispis nenultih članova

```
genmat (m, n, MAXRANG);
/* kontrolni ispis */
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        if (m[i][j] > 0) {
            printf ("M(%d,%d)=%d\n", i, j, m[i][j]);
        }
    }
}
return 0;
}
```

93

Rješenje - funkcija za generiranje matrice

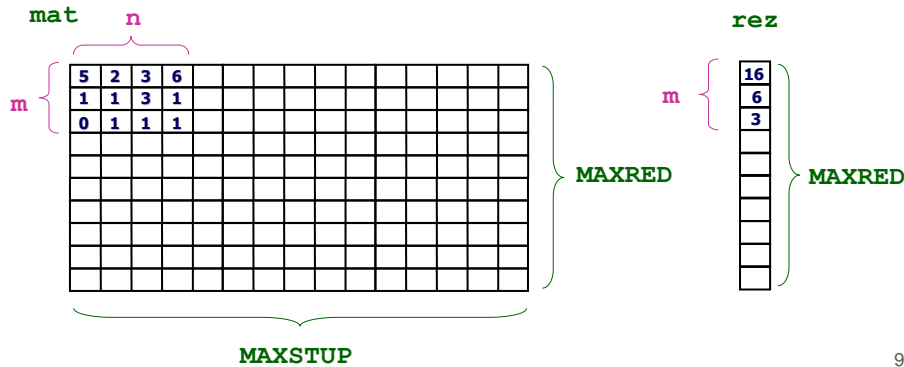
```
void genmat(int m[], int n, int maxstu) {
    int i, j;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            m[i * maxstu + j] = 0; /* m[i][j] */
        }
        m[i * maxstu + i] = 1; /* m[i][i] */
    }
}
```

94

Primjer za dvodimenzionalno polje kao argument funkcije: zbrajanje članova polja po retcima

Napisati funkciju koja u jednodimenzionalno realno polje upisuje sume elemenata redaka realne matrice dimenzija $m \times n$. U glavnom programu definirati matricu od najviše 10×15 elemenata, definirati polje u koje funkcija treba upisati rezultate, učitati dimenzije m i n , učitati elemente polja, pozvati funkciju, te ispisati učitanu matricu i dobiveni rezultat.



95

Rješenje - definiranje i učitavanje polja

```
#include <stdio.h>
#define MAXRED 10
#define MAXSTUP 15
void sumaRed(float mat[],
             int maksStup, int m, int n,
             float rez[]);

int main () {
    float mat[MAXRED][MAXSTUP], rez[MAXRED];
    int m, n, i, j;
    printf ("\nUpisite dimenzije m i n:");
    scanf("%d %d", &m, &n);
    printf ("\nUpisite elemente matrice po retcima:");
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
            scanf("%f", &mat[i][j]);
```

96

Rješenje - poziv funkcije i ispis

```
sumaRed (&mat[0][0], MAXSTUP, m, n, rez);
/* ispis ucitane matrice */
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++)
        printf ("%f ", mat[i][j]);
    printf("\n");
}
/* ispis rezultata */
printf ("\nSume po retcima:\n");
for (i = 0; i < m; i++)
    printf ("%f\n", rez[i]);
return 0;
}
```

97

Rješenje - funkcija za zbrajanje elemenata po retcima

```
void sumaRed(float mat[],
             int maksStup, int m, int n,
             float rez[]) {
    int i, j;
    for (i = 0; i < m; i++) {
        rez[i] = 0.0f;
        for (j = 0; j < n; j++)
            rez[i] += mat[i*maksStup + j];
    }
}
```

Treba li funkciju ili njezin poziv promijeniti ukoliko se promijene najveće dopuštene dimenzije matrice? Treba li što promijeniti u glavnom programu?

98

Macro s parametrima

Korištenjem funkcija se:

- povećava preglednost napisanog kôda (primjer: izračunavanje m povrh n)
- smanjuje broj linija programskog kôda (primjer: izračunavanje m povrh n)
- ali također i usporava izvršavanje programa: za prijenos argumenata i povratak rezultata troši se dodatno vrijeme

Primjer: koji se program brže izvršava?

```
glavni.c
#include <stdio.h>
int zbroj (int a, int b) {
    return a + b;
}

int main () {
    int i=3, j=4, k=5;
    printf("%d\n", zbroj(i, j));
    printf("%d\n", zbroj(j, k));
}
```

```
glavni.c
#include <stdio.h>
int main () {
    int i=3, j=4, k=5;
    printf("%d\n", i + j);
    printf("%d\n", j + k);
}
```

99

Macro s parametrima

```
#define VECI(a, b) ((a) > (b) ? (a) : (b))
```

U programskom kôdu pretprocesor zamjenjuje macro **prije** prevođenja:

```
#include <stdio.h>
#define VECI(a, b) ((a) > (b) ? (a) : (b))
int main () {
    int i = 3, j = 4;
    printf("%d\n", VECI(i, j));
    printf("%d\n", VECI(j, i));
}
```



```
...
int main () {
    int i = 3, j = 4;
    printf("%d\n", ((i) > (j) ? (i) : (j)));
    printf("%d\n", ((j) > (i) ? (j) : (i)));
}
```

100

Macro s parametrima: važna pravila

1. Macro definicija se **mora** nalaziti u jednom retku

~~#define BROJ_PI
3.14159~~ ispravno → #define BROJ_PI \
3.14159

2. Ako macro koristi operator, staviti cijeli izraz unutar zagrada

~~#define PI2 3.14*3.14~~ ispravno → #define PI2 (3.14*3.14)

3. Macro parametar unutar izraza uvijek staviti unutar zagrada

~~#define PROD(a, b) (a*b)~~
ispravno → #define PROD(a, b) ((a)*(b))

4. Macro s parametrima ne smije imati prazninu između imena i zagrade kojom započinje "lista argumenata"

~~#define NEG (a) (-a)~~ ispravno → #define NEG(a) (-a)

101

Macro s parametrima: važnost zagrada

```
/* ispravna macro definicija */
```

```
#define PI2 (3.14*3.14)
```

```
/* neispravna macro definicija */
```

```
#define PI2 3.14*3.14
```

U kôdu:

```
x = 1/PI2;
```

prije prevođenja obaviti će se zamjena:

```
x = 1/3.14*3.14;
```

Izračunat će se

```
(1/3.14)*3.14
```

a trebalo se izračunati

```
1/(3.14*3.14)
```

102

Macro s parametrima: važnost zagrada

```
/* ispravna macro definicija */  
#define VECI(a, b) ((a) > (b) ? (a) : (b))
```

```
/* neispravna macro definicija */  
#define VECI(a, b) (a) > (b) ? (a) : (b)
```

U kôdu:

```
x = 2; y = 3;  
z = 2 * VECI(x, y);
```

drugi redak zamijenit će se prije prevođenja u:

```
z = 2 * (x) > (y) ? (x) : (y);
```

Dobit će se $\Rightarrow 2 * 2 > 3 ? 2 : 3 \Rightarrow 2$

A pravi rezultat trebao je biti: 6

103

Macro s parametrima: važnost zagrada

```
/* ispravna macro definicija */  
#define PROD(a, b) ((a)*(b))
```

```
/* neispravna macro definicija */  
#define PROD(a, b) (a*b)
```

U kôdu:

```
i = 1; j = 3;  
k = PROD(i+1, j);
```

drugi redak zamijenit će se prije prevođenja u:

```
k = (i+1*j);
```

Dobit će se $\Rightarrow 1 + 1 * 3 \Rightarrow 4$

A pravi rezultat trebao je biti: 6

104

Macro u funkciji genmat:

GeneriranjeJedinicneMatriceMacro

```
#define polje(i, j) polje[(i)*maxstu+(j)]

void genmat(int polje[], int n, int maxstu) {
    int i, j;

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            polje(i, j) = 0;
        }
        polje(i, i) = 1;
    }
}
```

105

Važnost zagrada (još jednom...)

```
#define polje(i,j) polje[(i)*maxstu+(j)]
```

```
#define polje(i,j) polje[i*maxstu+j]
```

...

```
k = 4; maxstu = 10;
```

```
x = polje(k+3,2); ⇨
```

```
x = polje[k+3*maxstu + 2]; ⇨
```

```
x = polje[4+3*10+2]; ⇨ x = polje[36]; /*!!!*/
```

a trebalo bi

```
x = polje[72];
```

106

typedef deklaracija

```
typedef postojeći_tip novi_tip;
```

- deklarira sinonim: novo ime tipa s istim značenjem
- npr. ako se stanje računa izražava u lirama (bez decimala)

```
typedef int novac_t;  
novac_t stanjeRacuna;  
novac_t *pstanje = &stanjeRacuna;
```
- ako se stanje računa treba početi izražavati u kunama (decimale predstavljaju lipe), dovoljno je promijeniti deklaraciju tipa:

```
typedef float novac_t;  
novac_t stanjeRacuna;  
novac_t *pstanje = &stanjeRacuna;
```

107

typedef deklaracija u biblioteci potprograma

- C biblioteka potprograma koristi `typedef` za deklariranje tipova koji se razlikuju u različitim implementacijama prevodioca. Npr. funkcija `strlen` iz `<string.h>` vraća duljinu zadanog znakovnog niza. U nekim prevodiocima duljina se izračunava kao `unsigned int`, u nekim kao `unsigned long int`, itd. Da bi prototip funkcije bio jednak kod svih prevodioca, u biblioteci se koristi `typedef`:

```
typedef unsigned int size_t;  
ili  
typedef unsigned long int size_t;  
ili ...
```

Prototip funkcije `strlen` sada može biti jednak za sve prevodioce:

```
size_t strlen(const char *s);
```

108

NULL pokazivač

- Primjer: napisati funkciju koja vraća pokazivač na prvi član jednodimenzionalnog cjelobrojnog polja koji je manji od nule

```
int *nadj (int niz[], int n) {
    int i;
    for (i = 0; i < n; i++)
        if (niz[i] < 0) return &niz[i];
    return; /* a ako nema niti jedan < 0 ? */
}
```

- Umjesto nepoznate vrijednosti, funkcija bi trebala vratiti neku vrijednost koju će pozivajući program moći prepoznati.
- u takvim se situacijama koristi **null pokazivač**. Null pokazivač je "pokazivač na ništa".
- u standardnoj biblioteci potprograma definiran je macro NULL

109

NULL pokazivač

- Primjer: napisati funkciju koja vraća pokazivač na prvi član jednodimenzionalnog cjelobrojnog polja koji je manji od nule. U slučaju da takav član polja ne postoji, funkcija vraća null pokazivač.

```
#include <stdlib.h>
int *nadj (int niz[], int n) {
    int i;
    for (i = 0; i < n; i++)
        if (niz[i] < 0) return &niz[i];
    return NULL;
}

int main () {
    int polje[3] = {1, 3, 5}, *p;
    p = nadj(polje, 3);
    if (p == NULL) printf ("Nema takvog\n");
    else printf("Manji od nule je: %d\n", *p);
    return 0;
}
```

110