

Funkcije i pokazivači

Funkcije

Primjer: Izračunati "m povrh n"

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

2

Rješenje: m povrh n - (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 povrh n - (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 povrh %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

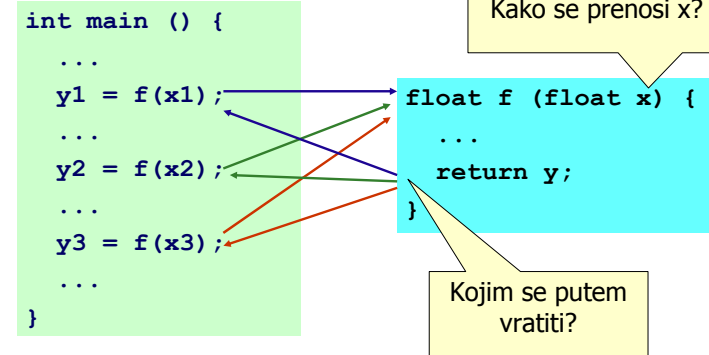
Rješenje s korištenjem funkcije - nastavak

```
/* 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

Programski slijed pri pozivu funkcije



8

Stog (*stack*)

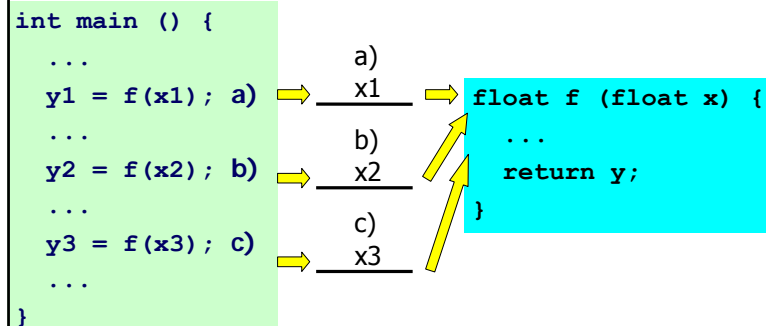
- Dio memorije koji služi za privremeni smještaj varijabli i povratnih adresa
- Struktura podataka tipa LIFO (Last In First Out)



- Pozivajući program na stog postavlja vrijednosti argumenata i povratnu adresu

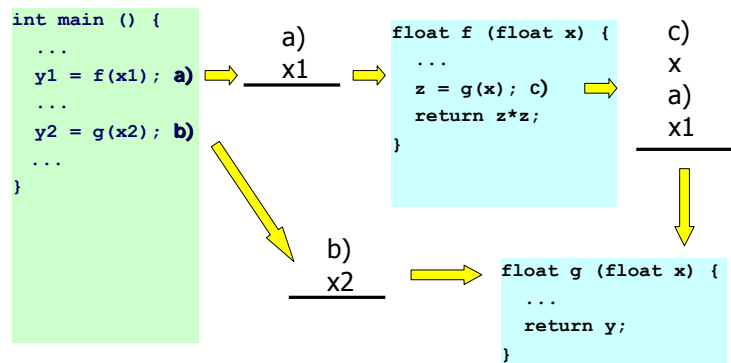
9

Stog



10

Programski slijed i stog pri pozivu funkcije – složeniji primjer



11

Funkcije u općenitom obliku

- Općeniti oblik funkcije
tip ime(tip1 arg1, tip2 arg2, ...)
npr.

```
int veci (int a, int b) {  
    return a > b ? a : b;  
}
```
- Primjer poziva funkcije:
`rez = veci(5, int(x)%3);`
- Funkcija koja ne vraća rezultat. Na primjer:

```
void pisi_koordinate(int x, int y) {  
    printf ("x=%d y=%d\n", x, y);  
}
```
- Primjer poziva void funkcije:
`pisi_koordinate (2,3);`

12

Komunikacija glavnog programa s funkcijom preko formalnih argumenata

- Načini komunikacije preko formalnih argumenata:
 - *call by value* – poziv predavanjem vrijednosti argumenata (funkcija ima interne kopije argumenata)
 - *call by reference* - poziv predavanjem adresa argumenata

13

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("\nUčitaj tri realna broja : ");
    scanf("%f %f %f", &x, &y, &z );
    sred = arit_sred(x,y,z);
    printf("\nAritmetička sredina unesenih brojeva je : %f",
        sred);
    return 0;
}
```

14

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;
}
```

15

Rezultat izvođenja

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

16

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;
}
```

17

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 ?

18

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
```

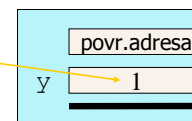
19

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.

20

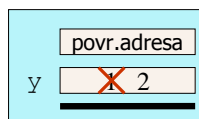
Izvođenje – obavljanje naredbe $y = 2$

Glavni program

x 1

Funkcija

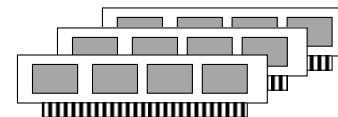
Stog



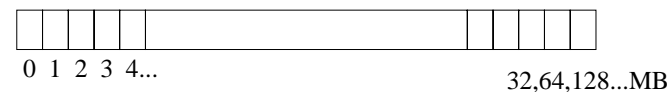
21

Pokazivač (*pointer*)

Memorija računala:



zapravo je kontinuirani niz bajtova:



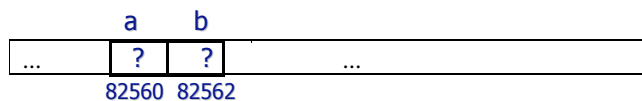
Svaki bajt ima svoj redni broj: *adresa*

22

Definicija "običnih" varijabli

Definicijom varijabli rezervira se prostor u memoriji na nekim adresama, npr.:

```
short a, b;
```

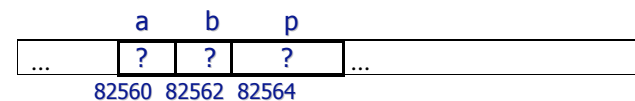


23

Definicija pokazivača

Definicijom pokazivača rezervira se prostor u memoriji u duljini 4 bytea kako bi u njega stala bilo koja adresa, npr.:

```
short *p;
```

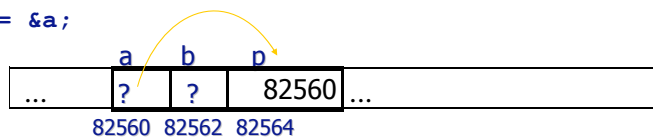


24

Dodjeljivanje početne vrijednosti pokazivaču

Nakon pridruživanja pokazivač `p` će sadržavati *adresu* neke druge varijable.

```
p = &a;
```

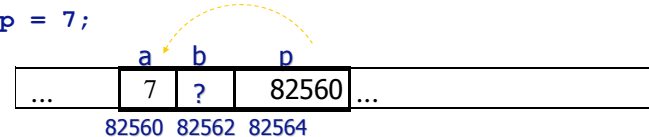


25

Indirektna izmjena vrijednosti varijable preko pokazivača

Nakon pridruživanja konstante 7 izmijenit će se podatak na adresi koja je sadržana u pokazivaču `p`, npr.:

```
*p = 7;
```

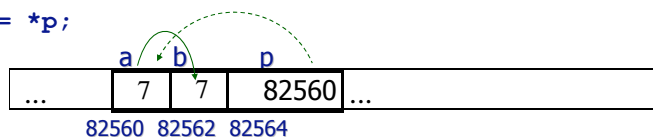


26

Indirektno uzimanje vrijednosti varijable preko pokazivača

Preko pokazivača može se indirektno uzeti vrijednost varijable, npr.:

```
b = *p;
```



27

Little Endian ili Big Endian?

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

    return;
}
```

Ispis: 255 1 0 0

28

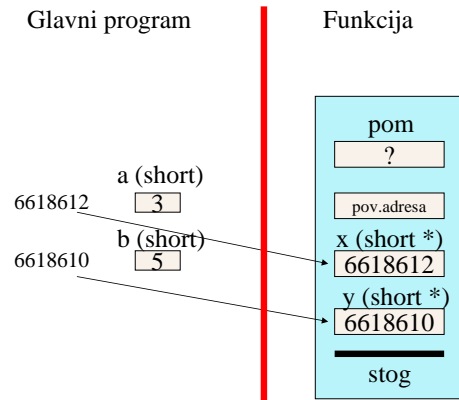
Primjer poziva funkcije predavanjem adresa argumenata (*call by reference*)

```

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;
}
    
```

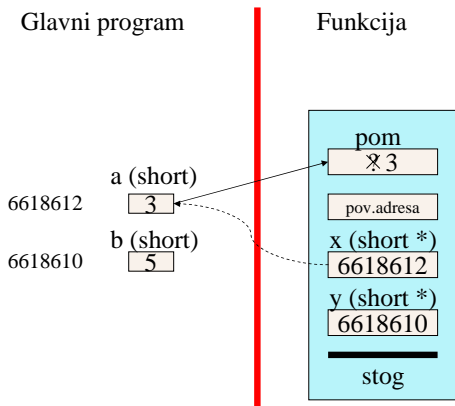
29

Izvođenje - poziv funkcije



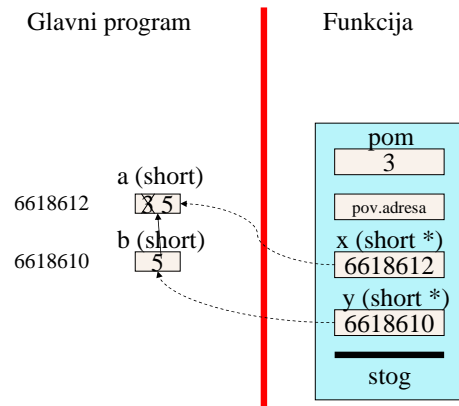
30

Izvođenje - naredba `pom = *x;`



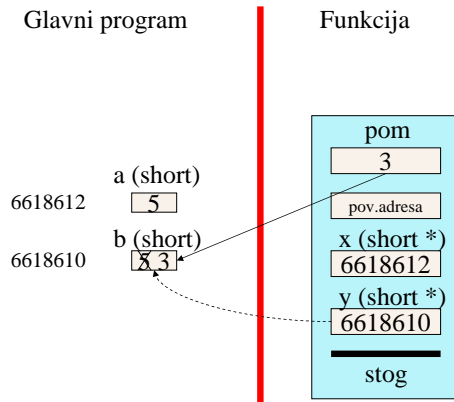
31

Izvođenje - naredba `*x = *y;`



32

Izvođenje - naredba `*y = pom;`



33

Izračunavanje finkcije sinus pomoću sume n članova reda

- Napisati funkciju koja će za zadani argument u radijanima 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>
#include <math.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

```
// realizacija sa 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

Organizacija složenijih programa

a) Funkcije smještene u istu datoteku s glavnim programom

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

double f2 (int c) {
    ...
}

int main () {
    ...
}
```

37

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 provjeriti je li
poziv funkcije korektno
napisan?

38

Prototip

- Omogućuje prevodiocu kontrolu broja i tipa argumenata.

```
tip ime (tip1 arg1, tip2 arg2, ...);
```

npr.

```
long kvadrat (int broj);
```

```
int veci (int a, int b);
```

39

Organizacija složenijih programa

c) Funkcije smještene u više datoteka 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);
}
```

40

Organizacija složenijih programa

d) Funkcije smještene u istu datoteku s glavnim programom s prototipom

```

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

Opći oblik naredbe za definiciju/deklaraciju varijabli ili polja - smještajni razredi (*storage classes*)

```

smještajni_razred tip_podatka varijabla ... ;
smještajni_razred tip_podatka polje[izraz1][izraz2]... ;
    
```

- **smještajni_razred** i mjesto definicije/deklaracije utvrđuje postojanost (trajnost) i područje važenja varijable ili polja u memoriji.
- **Smještajni razredi:**
 - auto** – automatski smještajni razred. Obično se ključna riječ **auto** ne navodi. Podrazumijeva se da su sve varijable i polja definirani unutar neke funkcije razreda **auto** ako drugačije nije eksplicitno navedeno. Takve su varijable vidljive samo funkciji, a smještaju se na stog, te zato bivaju prepisane sljedećim pozivom bilo koje funkcije.

42

Smještajni razredi - nastavak

extern – vanjski smještajni razred ukazuje na varijable i polja koji su globalni (zajednički) za sve funkcije unutar programa. Varijabla razreda u jednom od modula mora biti definirana izvan tijela funkcija bez ključne riječi **extern**

static –

- definirano unutar funkcije: statički razred koji se koristi kada vrijednost varijable ili članova polja treba zadržati nakon izlaska i ponovnog povratka u neku funkciju.
- definirano izvan funkcije: ukazuje na varijable koje su zajedničke za sve funkcije unutar modula, ali nisu vidljive drugim modulima.

register – registarski razred; preporuka prevodiocu da, ako je moguće, za smještaj varijabli koristi registre procesora, jer će se varijable intenzivno koristiti.

43

Smještajni razredi i komunikacija funkcija pomoću globalnih varijabli

funkcije.c	Područje važenja varijabli
<pre> #include <stdio.h> extern int x; static int y; int f1 (long a, long b) { ... } double f2 (int c) { static double y; ... } int f3 () { ... } </pre>	
<pre> glavni.c #include <stdio.h> int x; void f4 (int d) { } int main () { double y; ... } </pre>	

Aritmetika s pokazivačima

```
long l; double d;
long *pl; double *pd;
pl = &l;
pd = &d;
++pl;
pd = pd + 2;
```

Vrijednost pl	Vrijednost pd
?	?
128560	?
128560	128564
128564	128564
128564	128580

45

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

46

Jednodimenzionalna polja kao argumenti funkcije

Primjer:

Napisati funkcijski potprogram koji 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 raspon neispravno zadan, u glavni program treba vratiti vrijednost -1.

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

47

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]);
    }
}
```

48

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;
}
```

49

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;
    }
}
```

50

Dvodimenzionalna i višedimenzionalna polja kao argumenti funkcije

U funkciji npr.

`void f(int polje[][])` ⇒ **pogreška!**

Polje u funkciji treba dočekati kao jednodimenzionalno polje ili pokazivač.

51

Dvodimenzionalna i višedimenzionalna polja kao argumenti funkcije

Primjer:

```
int polje[3][4] = { { 1, 2, 3, 4},
                   { 5, 6, 7, 8},
                   { 9, 10, 11, 12}
                 };
```

u memoriji računala spremljeno je kao:

1	2	3	4	5	6	7	8	9	10	11	12
---	---	---	---	---	---	---	---	---	----	----	----

tj. jednako kao jednodimenzionalno polje od 12 elemenata.

52

Dohvat elementa dvodimenzionalnog polja

Za dohvat elementa iz n -tog retka treba preskočiti $n-1$ punih redaka. Kako prvi redak ima indeks 0, drugi 1, treći 2 itd., za dohvat elemenata iz retka s indeksom i treba preskočiti $i \cdot \text{broj_stupaca}$ članova polja. Općenito vrijedi:

```
dvodim_polje[i][j]    =  
jednodim_polje[i*broj_stupaca + j]
```

U našem primjeru:

```
polje[1][2]  
p[1*4+2] = p[6] ⇒ 7
```

53

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 učitati rang matrice ≤ 100 , pozvati funkciju i provjeriti ispravnost generirane matrice tako da se ispisuju indeksi i i vrijednosti svih onih članova koji su različiti od 0 u obliku $M(i,j) = x$.

54

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);
```

55

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;  
}
```

56

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] */
    }
}
```

57

Macro s parametrima

```
#define maxof(a,b) (a > b ? a : b)
Zamjenjuje se u programskom kodu prije prevođenja.
```

Važnost zagrada:

```
/* neispravno! */
```

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

U kodu:

```
x = 2; y = 3;
```

```
z = 2 * veciod (x,y);
```

poziv veciod se zamjenjuje s

```
z = 2 * x > y ? x : y; ⇒
```

```
2 * 2 > 3 ? 2 : 3 ⇒ 2
```

Pravi rezultat: 6

58

Poboljšanje funkcije genmat:

```
GeneriranjeJedinicneMatriceMacro
#define m(i,j) m[(i)*maxstu+(j)]
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,j) = 0;
        }
        m(i,i) = 1;
    }
}
```

59

Važnost zagrada (još jednom...)

```
#define m(i,j) m[i*maxstu+j]
...
k = 4; maxstu = 10;
x = m(k+3,2); ⇒
x = m[k+3*maxstu + 2]; ⇒
x = m[4+3*10+2]; ⇒ x = m[36]; /*!!!*/
a trebalo bi
x = m[72];
```

60

Znakovni niz u funkciji

```
char niz1[] = "IME:ETF";  
char niz2[] = "IME:FER";  
...  
strcmp (niz1, niz2);  
..  
int strcmp (char *s1, char *s2) {  
    for (;*s1 == *s2 && *s1; s1++, s2++);  
    return *s1 - *s2;  
}
```

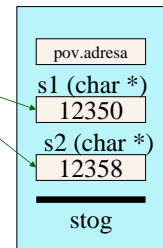
61

Poziv funkcije

Glavni program

12350 12358
I M E : E T F 0 I M E : F E R 0

Funkcija



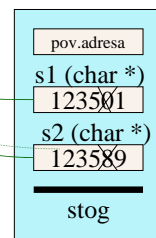
62

Izvršenje naredbi `s1++`, `s2++`

Glavni program

12350 12358
I M E : E T F 0 I M E : F E R 0

Funkcija



63