Vježbe za 5. *blitz*

PiPI 2005. Grupa 01
Zdenko Šimić

Teme za 5. *blitz*

- **Sve od početka**
- **void funkcije**
- **Formalni i stvarni argumenti (izrazi, redoslijed, tipovi pri pozivu funkcije)**
- ***Call by value* (primjeri gdje se argument ne promijeni)**
- ***Call by reference* (bez polja)**
- **Pokazivači (zasebno, u programu bez funkcije i aritmetike)**
- **Aritmetika s pokazivačima**
- **Jednostavne funkcije (vraćaju jedan rezultat)**
- **Prototipovi**
- **Jednodimenzionalno polje kao argument funkcije**
- **Dvodimenzionalno polje kao argument funkcije**
- **Smještajni razredi, postojanost, područje važenja varijabli**
- **Macro s parametrima**
- **Matematičke ugrađene funkcije**
- **Vlastite funkcije za rad s nizovima (deklariranim kao polje)**
- **Vlastite funkcije za rad s nizovima (deklariranim kao pokazivač)**

Funkcije

- Funkcije su korisne
  - bolja preglednost koda
  - mogu smanjiti kod
  - neograničeno pozivanje
  - lakše ispravljanje koda
- Ne ubrzavaju uvijek program
- *Call by value*
  - može mijenjati vrijednosti argumenata, ali oni **nisu** vidljivi nakon izlaska iz funkcije
- *Call by reference*
  - može mijenjati vrijednosti argumenata koji su vidljivi nakon izlaska iz funkcije
- *Default* tip funkcije je `int`
- Prototip funkcije
  - pomaže otkrivanju krivog poziva funkcije
  - smješta se na vrh datoteke ili zajedno s drugim prototipovima u posebnu datoteku koja se referencira s #include <ime.h>
  - nije nužan ako se funkcije poziva nakon što je opisana

void funkcija

- void funkcija
  - ne vraća vrijednost
  - može ali ne mora imati argumente
- sintaktički je ispravno:

```
void nako(){;}
```

  - najjednostavniji ispravni oblik
  - koristiti `return` bez argumenta unutar *void* funkcije

```
void neka(int i){
  printf("2i=%d",2*i);
}
```

- sintaktički je pogrešno:
  - pozivati *void* funkciju očekujući rezultat:

```
neka(neka(10));
```

  - koristiti `return` s argumentom unutar *void* funkcije

void funkcija

Dio programa:

```
void fn2(int n){
   printf("fn2- %d ", 2*n);
}
void fn1(int n){
   printf("fn1- %d ", 2*n);
   fn2(--n);
}
void main(){
  fn1(2);
}
```

Rezultat/ispis:

```
fn1- 4 fn2- 2
```

```
void uradi(int m) {
   int  i, zbroj=0;
   for(i=0; i<m; i++)
       if(i%3) zbroj+=i;
  printf("%d ", zbroj);
 }
```

void funkcija **uradi**:
- sumira sve brojeve manje od **m** koji nisu djeljivi sa **3** i
- na kraju ispisuje rezultat

Formalni i stvarni argumenti funkcija

```
int n, m;
char z;
double r, u;
```

Prototip funkcije:

`m=uf(z,n,3*r,2e-4);`	`int uf(char, int, double, double);`
`uh(m,u,z-32);`	`void uh(int i, double d, char c);` `void uh(int, double, char);`
`oj(10.f, 3., 45);`	`void oj(float f, double d, char c);` `void oj(float, double, int);`
`u = ah(n,&r,&z);`	`double ah(int i, double *d, char *c);`

```
printf("\n%d", eh(u,z,m));
```

```
int eh(double, char, int);
```


Što nam znači prototip funkcije?

```
void fu(char c, int *n);
```

-može vratiti rezultat preko pokazivača **n**

```
int *bu(char *z);
```

-funkcija vraća pokazivač na cijeli broj i ima pokazivač na niz znakova kao ulazni argument

```
mu(int *n);
```

- funkcija vraća cijeli broj (*default* tip) i prima ulazni pokazivač na cijeli broj kao ulazni argument

Kako glasi prototip funkcije koja treba izračunati sumu cijelih brojeva u nekom intervalu?

```
long su(int d, int g);
```
ili
```
void su(long *s, int d, int g);
```


Call by value

Program:

```
char gore(int n){
   n+10;
   return n;
}

void ravno(int n){
   n-=10;
}

void main(){
   int n=10;
   n = gore(n);
   ravno(n);
   printf("n=%d", n);
}
```

Rezultat/ispis:

-funkcija **gore** zapravo ne mijenja **n**, ali ima problem za veliki ulazi argument!

-funkcija **ravno** zapravo ne radi ništa jer je ***void*** i prima vrijednost varijable!

n=10 nakon izvršavanja **gore(n)**
n opet nepromijenjen nakon **ravno(n)**
n=10

Call by value

Program:

```
void ispisi(int n){
  n--;
  printf("ni=%d ", n);
}
void main(){
  int n=10;
  ispisi(n);
  ispisi(++n);
  ispisi(n++);
  n-=2;
  printf("nm=%d ",n);
}
```

Rezultat/ispis:

-funkcija **ispisi** ne mijenja **n**, jer prima vrijednost od **n** te istu umanjenu za 1 ispisuje!

```
ni=9 ni=10 ni=10 nm=10
```


Call by value

Program:

```
int fdp(int n){
  n+=5;
  return n;
}
void main(){
  int n=0, m, k;
  n = fdp(n);
  fdp(m=0);
  k=fdp(fdp(m-n));
  printf("%d%d%d",
   n, m, k);
}
```

Rezultat/ispis:

-funkcija **fdp** vraća vrijednost **n**, uvećanu za 5

n=5 nakon izvršavanja **fdp(0)**

-ovdje se ne mijenja ništa, **fdp(0)** opet vraća 5, ali se to ne sprema nigdje

k=5 nakon izvršavanja **fdp(fdp(-5)) = fdp(0) = 5**

```
505
```


Jednostavne funkcije

Program:

```
int fp(int n, int m){
  return n*m;
}
int fm(int n, int m){
  return n%m;
}
void main(){
  int n=5, m=2, k=6;
  n = fp(fm(n,m),k);
  m = fm(5,fp(m,1));
  k = fp(fp(2,3),fm(3,2));
}
```

Rezultat:

-funkcija **fp** vraća produkt ulaznih vrijednosti

-funkcija **fm** vraća produkt ulaznih vrijednosti

```
n=fp(1,6)=6
m=fm(5,2)=1
k=fp(6,1)=6
```


Jednostavne funkcije

Program:

```
int lim(int m){
  if (m<10) return 10*m;
  return -10*m;
}
int mil(int n){
  return n%12;
}
double rec(int m){
  return 10./m;
}
void main(){
  int n=0, m=10;
  n = lim(n+m);
  m = mil(m+8);
  printf("%f",rec(n));
}
```

Rezultat:

Za početno kao u programu (n=0 i m=10) rezultat je:

```
n = -100
m = 6
```

ispis: `-0.100000`

Za početno n = m = 1 rezultat je:

```
n = 20
m = 9
```

ispis: `0.500000`

Macro s parametrima

Program:

```
#define ta(a,b,c) a*b*c
#define tb(a,b,c) (a)*(b)*(c)
#define tc(a,b,c) ((a)*(b)*(c))
void main(){
  int x, y, z;
  int n=0, m=2, k=1;
  x = !ta(n,m,k);
  y = !tb(n,m,k);
  z = !tc(n,m,k);
  x = ta(n,m,k);
  y = tb(n,m,k);
  z = tc(n,m,k);
}
```

Rezultat:

Zagrade!!!

```
x = 2
y = 2
z = 1
x = 0
y = 0
z = 0
```


Stog i varijable

- Stog sadrži
  - povratnu adresu (4 bajta)
  - argumente funkcije i sve lokalne automatske varijable (prema tipovima)
  - okvir stoga (ne broji se)
- Stog ne sadrži
  - lokalne konstante
  - statičke lokalne varijable
  - globalne varijable

```
void nema(){}
void main(){
  nema();
}
```

- poziv funkcije **nema()** zauzima na stogu memoriju potrebnu za povratnu adresu

Stog – zauzeće memorije

Program:

```
int fdr(long n, char z){
  short k=5*n/z;
  return k;
}

double fpr(int m){
  return 1.*m/fdr(300,'*');
}

void main(){
  double dupla;
  dupla = fpr(10000);
}
```

Rezultat:

- Prvo se poziva funkcija fpr iz main-a
  1. <na stog ide povratna adresa i int vrijednost>,
- potom se iz fpr poziva fdr
  2. <na stog se dodaje još jedna povratna adresa te vrijednost za long i char>,
  3. unutar fdr na stog se dodaje short varijabla k.
- Tu je sve gotovo i maksimalo zauzeće stoga, bez tzv. okvira, je u bajtima:
- **4 + 4 + 4 + 4 + 1 + 2** = 19 bajta

Pokazivači

Pokazivači

- varijable koje sadrže memorijsku adresu
- pored adrese deklaracijom je određen tip podatka na koji pokazuje
- operator * omogućava čitanje i spremanje vrijednosti na pokazivanu adresu preko pokazivača
- operator & omogućava pridruživanje adrese bilo koje (uključujući pokazivače) varijable pokazivaču
- svaki pokazivač zauzima 4 bajta
- aritmetika nad pokazivačima mijenja vrijednost u kvantima koje određuje tip podatka na koji pokazivač pokazuje

```
char *pc, c='Y';
int *pi, *pk, i=1, j;
double *pd, d=0;
pc = &c;
pi = &i;
pk = pi;
pd = &d;
*pc = 'N';
j = *pi;
*pk = 0;
*pd = 2;
```

Stanje:

```
c='N'  i=0  j=1  d=2.0
```

Aritmetika pokazivača

Dozvoljeno i smisleno

- Zbrajati i oduzimati cijeli broj od pokazivača
- Za dva pokazivača na isto polje:
  - oduzimati ih
  - uspoređivati ih (<=>)
- Nema smisla pridruživati pokazivaču rezultat
  - množenja, dijeljenja ili % (mod) operacija dvaju cijelih brojeva

Nije dozvoljeno

- Među pokazivačima:
  - Množenje, dijeljenje i % (mod)
- Zbrajanje dvaju pokazivača

Pokazivači

```
short n=1, m=2, k;
short *pi = &n, *pj = &m, *pp;

pp = &k;
*pp = *pi;
*pi = m;
m = *pp;
*pp = k * m + n;
pp = pi;
printf("%d %d %d\n", n, m, k);
printf("%d %d %d",*pi,*pj,*pp);
printf("\n%d ", *pi + *pj);
printf("\n%d ", sizeof(n));
printf("\n%d \n", sizeof(pi));
n = sizeof(n) + sizeof(pj);
printf("\n%d %d", n, n + *pi);
```

Ispis:

```
2 1 3
2 1 2
3
2
4

6 12
```


Pokazivači

```
double x=3., y=4., z;
double *pa=&x, *pb=&y, *pd;

pd = &z;
*pa += *pb;
*pb = *pa - y;
x = *pa - *pb;
*pd = x * y ;
pd = pb;
printf("%f %f %f\n", x, y, z);
printf("%f %f %f",*pa,*pb,*pd);
printf("\n%f ", *pb + *pd);
printf("\n%d ", sizeof(x));
printf("\n%d \n", sizeof(pa));
z = sizeof(x) + sizeof(pb);
printf("\n%f %f", z, z + *pd);
```

Ispis:

```
4.0 3.0 12.0
4.0 3.0 3.0
6.0
8
4

12.0 15.0
```


Pokazivači – radi, ali nema smisla:

Ispis:

```
int n;
int *pi = &n, *pj;
double x = 5.;
double *pa = &x, *pb;

pj = (int)x;
printf("%lu %p\n", pj, pj);
*pa = 100 * (int)(&x);
printf("%p %p\n", pa, &x);
printf("%lu %lu\n", pa, &x);
printf("%f %f\n", *pa, x);
printf("%lu", 10 * (int)(&x));
```

```
5 00000005

0042CCA8 0042CCA8
4377768 4377768
437776800.0 437776800.0
43777680
```

Adresa varijable `x` je proizvoljna i rezultat se može mijenjati kod ponovnog izvođenja!

Pokazivači – aritmetika

Sve adrese samo su ilustracija!

```
float ar[3]={1.1, 2.2, 3.3};
float *pa, *pb;
char az[40], *pz=&az[0];
pa = pb = ar;
printf("\n%lu %f\n", pa, *pa);
pb += 2;
printf("%lu %f\n", pb, *pb);
printf("%lu %f\n", ar+2, *(ar+2));
printf("%lu %f\n", &ar[2], ar[2]);
printf("%d\n", pb-pa);
printf("%d\n", (int)pb-(int)pa);
printf("%lu %f\n", pb-1, *(pb-1));
pz += 4;
printf("%d\n", pz-az);
printf("%d\n", (int)pz-(int)az);
pz = (char *) ((int *) az +4);
printf("%d\n", pz-az);
printf("%d\n", (int)pz-(int)az);
```

Ispis:

```
4346216 1.100000
4346224 3.300000
4346224 3.300000
4346224 3.300000
2
8
4346220 2.200000
4
4
16
16
```


Call by reference – bez polja

Program:

```
void pomakni(char *c) {
  *c += 2;
}

void main () {
  char c = 'a', z = 'b';
  pomakni(&c);
  pomakni(&z);
  printf ("c='%c'\n", c);
  printf ("z='%c'", z);
}
```

Rezultat/ispis:

```
c='c'
z='d'
```


Call by reference – bez polja

Program:

```
int fopa(char *z, short t) {
  (*z)--;
  t-=2;
  return (*z) * t;
}

main () {
  char c = '3';
  int i;
  short t = 3;

  i = fopa (&c, t);
  printf ("%d %d\n", i, t);
  printf ("%d %c\n", c, c);
}
```

Rezultat/ispis:

```
50 3
50 2
```


Call by reference – bez polja

Program:

```
int puk(int *i, int j) {
   int k;
    k=*i * j;
    (*i)++;
    j--;
    *i *= j;
    printf ("f: %d ", *i);
    printf ("%d %d", j, k);
    return *i;
}

main () {
  int i, j=5, k=1;
  i = puk (&k, j);
  printf ("\nm: %d ", i);
  printf ("%d %d", j, k);
}
```

Rezultat/ispis:

```
f: 8 4 5

m: 8 5 8
```

Call by reference – 1D polje

Program:

```
int puni(int *ari, int n) {
  int i=0, p=5;
   for(;i<n;i++) ari[i]=p;
  return p;
}

main () {
  int ari[20];
  int i=20, j;
  j = puni(ari, i);
  printf ("%d %d", *ari, ari[0]);
  printf ("\n%d", j);
}
```

Rezultat/ispis:

```
5 5
5
```


Call by reference – 2D polje

Program:

```
void fda(int *ari, int n, int m, int stupmax) {
   printf("%d ", *(ari+n*stupmax+m));
   printf("%d \n", ari[n*stupmax+m]);
}

void fdb(int *ari, int n, int m, int stupmax) {
    int i=0, j;
     for (;i<n;i++)
        for (j=0;j<m;j++)
            printf("%d ", ari[i*stupmax+j]);
}


main () {
  int ari[5][10]={{1, 2, 3}, {4, 5, 6}, {7, 8,
     9}};
  int i=3, j=3, k=10;
  fda(&ari[0][0], i-1, j-1, k);
  fdb(ari, i, j, k);
}
```

Rezultat/ispis:

```
9 9

1 2 3 4 5 6 7 8 9
```