Pokazivači

Pointeri

Definicija pokazivača

- Pokazivač na `tip` je varijabla koja sadrži adresu varijable tipa `tip`.
- Definicija pokazivača:

`mem_klasa tip` * p_var;

Primjer:

```
static int * pi;
double *px;
char* pc;
int a, *b;
float* pf, f;
```

Operatori & i *

- Adresni operator (&):
  - &x je adresa varijable x.
- Operator dereferenciranja (*):
  - *p_x je vrijednost spremljena u memorijsku lokaciju na koju p_x pokazuje.

Primjer:

```
int i = 5;
int *pi = &i;
i = 2 * (*pi + 6);
printf("i = %d, adresa od i = %p\n", i, pi);
/* i = 22, adresa od i =  000000004DA0FE04 */
```

Pokazivači i funkcije

- Pokazivači mogu biti argumenti funkcije. U tom slučaju funkcije može promijeniti vrijednost varijable na koju pokazivač pokazuje:

```
void zamjena(int *x, int *y){...}
```

- Aritmetičke operacije nad pokazivačima trebaju biti konzistentne (imati smisla).
- Svakom pokazivaču moguće je dodati i oduzeti cijeli broj. Ako je px pokazivač i n varijabla tipa int, onda su dozvoljene operacije:

```
++px    --px    px + n    px - n
```

Operacije nad pokazivačima

nova_adresa = stara_adresa + (`sizeof`(tip_podatka)*broj)

Primjer:

```
char *pc;  /* 1000 */
int  *pi;  /* 1000 */
double *pd;  /* 1000 */

pc++;
pi++;
pd++;
```

Operacije nad pokazivačima (2)

Primjer:

```
int a[ ] = {10, 2, 3, 4};
int *pi, *pj;

pi = &a[0];
pj = &a[2];

printf("%d\n", pi); /* 6422016 */
printf("%d\n", pj);

printf("%d\n", pj - pi);
printf("%d\n", *pj - *pi);
```

Operacije nad pokazivačima (3)

```
int main(void)
{
   int x[3] = {10, 20, 30};
   int *px;
   px = x; /* 00…061FE0C */

   printf("%d, %p\n", *px+1, px);   /* 11, 00…061FE0C */
   printf("%d, %p\n", ++*px, px);    /* 11, 00…061FE0C */
   printf("%d, %p\n", *px++, px);    /* 11, 00…061FE10 */

   return 0;
}
```

Uspoređivanje pokazivača

- Pokazivače istog tipa možemo uspoređivati pomoću relacijskih operatora. Takva operacija ima smisla ako pokazivači pokazuju na isto polje:

  px < py px > py px == py px != py

- Sljedeće dvije petlje su ekvivalentne:

```
int i, *pi, x[10];
.....
for(i = 0; i < 10; i++) x[i] = i;

for(i = 0, pi = &x[i]; pi <= &x[9]; pi++, i++) *pi = i;
```

Pokazivači i cijeli brojevi

- Pokazivaču nije moguće pridružiti vrijednost cjelobrojnog tipa, osim nule.
- Nula nije legalna adresa; ona označava da pokazivač nije inicijaliziran:

  `double` *p = 0;

  ili

  `#define` NULL 0

  ……

  `double` *p = NULL;
- Simbolička konstanta `NULL` definirana je u `<stdio.h>`.

Primjer: Važnost prioriteta

Primjer:

```
int main(void)
{
   double x[ ]={10.0, 20.0, 30.0, 40.0}, *px;
   px = &x[0];
   printf("%d\n", px);          /* 6422000 */
   printf("%g, %g \n", *px+1, *px+2);        /* 11, 12 */
   printf("%d\n", px);
   printf("%g, %g \n", *px++, *++px+2);    /* 20, 22 */
   printf("%d\n", px);
   return 0;
}
```

Razlika pokazivača

- Pokazivači se mogu oduzimati ukoliko pokazuju na isto polje.
- Ako su px i py dva pokazivača, onda je py – px + 1 broj elemenata između px i py, uključujući i krajeve:

```
int d(int a[ ]){
   int *pa = a;
   while(*pa != 0) pa++;
   return pa - a;
}

   int x[5] = {3, 2, 0, 0, 4};   printf("%d\n", d(x));
```

Pridruživanje

- Pokazivači na različite tipove podataka ne mogu se pridruživati bez eksplicitne konverzije:

```
char *pc;
int *pi;
.....
pi = pc;      /* Greška */
pi = (int *) pc;     /* Ispravno */
```

Generički pokazivač

- Pokazivač može biti generiran i kao pokazivač na `void` – tada govorimo o generičkom pokazivaču:

```
void *p;
```

- Pokazivač na bilo koji tip može se konvertirati u pokazivač na `void` i obratno:

```
double *pd0, *pd1;
void *p;
……
p = pd0;
pd1 = p;
```

Generički pokazivač (2)

- Osnovna uloga generičkog pokazivača je da omogući funkciji uzimanje pokazivača na bilo koji tip podatka.

- Primjer:

```
double *pd0;
int *pd1;
void f(void *);
……
f(pd0);
f(pd1);
```

Pokazivači i jednodimenzionalna polja

- Ime jednodimenzionalnog polja je konstantni pokazivač na prvi element polja!

Primjer:

```
int a[10], b[10];

……
a = a+1; /* Greška, a je konstantni pokazivač. */
b = a; /* Greška! */
```

Pokazivači i jednodimenzionalna polja (2)

Primjer:

```
int a[10], *pa;
……
pa = a; /* ekvivalentno je s pa = &a[0]; */
pa = pa + 2;  /* nije pogreška - &a[2] */
pa++;  /* &a[3] */
```

Primjer:

```
int a[10], *pa;
……
pa = &a[0];
*(pa + 3) = 20; /* ekvivalentno je s a[3] = 20; */
*(a + 1) = 10; /* ekvivalentno je s a[1] = 10; */
```

Polje kao argument funkcije

- Funkciju f koja uzima polje v tipa `tip` možemo deklarirati kao

  f(`tip` v[ ]); ili f(`tip` *v);

- Ispravni su sljedeći pozivi funkcije f:

```
char z[100];
void f(char *);
……
f(z);
f(&z[0]);
```

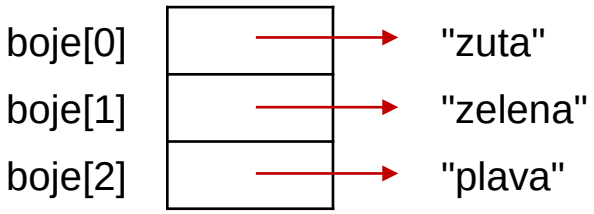
Polje pokazivača

- pp je polje od 10 pokazivača na int:

```
int *pp[10];
```

Primjer:

```
char *boje[ ] = {"zuta", "zelena", "plava"};
printf("%s\n", boje[1]);
printf("%c\n", boje[1][3]);
printf("%c\n", *(boje[1] + 3));
```



- pp je pokazivač na cjelobrojno polje od 10 elemenata:

```
int (*pp)[10];
```

Pokazivači i dvodimenzionalna polja

Primjer:

```
int a[2][3] = {{1,2,3},{4,5,6}}, *pa;
……
pa = a; /* ekvivalentno je s pa = &a[0][0]; */
pa = pa + 3;
printf("%d\n", *pa);   /* 4 */
printf("%d\n", *(a[1]));   /* 4 */
```

- a[i][j] ⇔ *(a[i] + j) ⇔ *(*(a + i) + j)

- Dvodimenzionalno polje kao argument funkcije:

```
void funkcija(int (*a)[MAX_n], int n, int m);
```

Dinamičko rezerviranje memorije

- Memoriju možemo dinamički alocirati pomoću funkcija:
  - `void *malloc(size_t n);`
  - `void *calloc(size_t n, size_t b);`
- `malloc` alocira blok memorije od `n` bajtova.
- `calloc` alocira memoriju za `n` elemenata od kojih je svaki veličine `b` bajtova. Alocirano područje inicijalizirano je nulom.
- Obje funkcije vraćaju pokazivač na blok memorije ili `NULL` ukoliko zahtjev za memorijom nije mogao biti ispunjen.

Primjer: Funkcija `malloc`

```
double *p;
.....
p = (double *) malloc(128*sizeof(double));
if(p == NULL){
    printf("Greška! Alokacija nije uspjela.\n");
    exit(-1);
}
.....
free(p); /* Memorija više nije potrebna. */
```

- Funkcija `realloc`:
  - `void *realloc(void *ptr, size_t n);`

Funkcija `free`

- Alociranu memoriju, kada nam više ne bude potrebna, možemo osloboditi pomoću funkcije `free`:
  - `void free(void *blok);`
- Funkcija `free` uzima pokazivač na početak alocirane memorije.

Primjer:

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

int main ( )
{
   int *a; int i, n, zbroj;

   printf ("Učitaj broj elemenata polja a:");
   scanf ("%d", &n);

   if((a = (int*) calloc (n, sizeof(int))) == 0){
         printf ("Alokacija nije uspjela. \n");
         exit(-1); }
```

Primjer: (2)

```
    for (i = 0;  i< n; ++i){
            printf ("Upiši elemenat polja:");
            scanf("%d", &a[i]); }

    zbroj = 0;
    for (i = 0; i < n; ++i)
        zbroj = zbroj + a[i];
    printf("%d\n", zbroj);

    free(a);

    return 0;
}
```