Pokazivači

Operativna memorija

- **niz lokacija sa pridruženim adresama**
- **najmanja adresibilna jedinica bajt**
- **pristup na nivou mašinske reči**
- **podatak u uzastopnim bajtovima**

Pokazivači

Pokazivač

- **promenljiva koja sadrži adresu objekta u memoriji (ukazuje na bajt sa najnižom adresom)**

- **obično 4 ili 8 bajtova (zavisi od adresnog prostora, a ne od veličine objekta)**

Pokazivači

definicija pokazivača

tip *ime ;

- **tip pokazivanog objekta**
- **ime pokazivačke promenljive**

```
int *pa, a;
double *px, *py;
```

Pokazivači

& - <u>adresni operator</u>
(samo za promenljive, odnosno lvrednost)

```
pa = &a;
```

- **daje <u>adresu</u> zadatog objekta (ne može konstanta, ni proizvoljan izraz)**

Pokazivači

*** operator <u>indirektnog adresiranja</u>**

```
*pa = 5;
```

- **posredni pristup podatku preko adrese u pokazivačkoj promenljivoj**

Pokazivači

oba operatora (& *)

- **prefiksni unarni**
- **prioritet 14**
- **asocijativnost D->L**

Pokazivači

generički pokazivači

```
void *pv;
```

- **nije definisan tip pokazanog objekta**
- **void umesto oznake tipa objekta**
- **ne može pristup pokazanom podatku preko generičkog pokazivača**

Pokazivači

Primer:

```
int x=1, y=2, z[7];
int *pi;
void *pj;
pi = &x;
y = *pi;
pj = pi;
*pi = 0;
*pj = 3;      /*Greška*/
*(int*) pj = 3;
pi = &z[0];
pj = &z[5];
```

Pokazivači

Kvalifikatori i pokazivači

```
const tip *p; // pokazana prom. je const
tip *const p=&k; // pokazivač je const
```

- **Pokazivaču na promenljive podatke ne mogu se dodeliti adrese nepromenljivih ili nepostojanih podataka.**
- **Pokazivaču na nepromenljive podatke mogu se dodeliti adrese promenljivih podataka, ali se neće moći preko njega promeniti podatak.**
- **Prevazilaženje ograničenja pomoću cast-a**

Pokazivači

konstantni pokazivači i podaci

Primer:

```
int a=2, *pa=&a, *const kpa=&a;
const int b=5, *pkb=&b, *const kpkb=&b;
pkb = pa;
*pkb = 4;  //Greska, pok. na nepromenljiv
kpa = pa;  //Greska, nepromenljiv pok.
pa = kpkb; //Greska, pok. na nepromenljiv
//ne moze se dodeliti pok. na promenjiv
pa = (int*) kpkb; // prevazilazenje
*pa =4 ; //menja vrednost konstante, moze!
```

Pokazivači

Primer:

```
typedef double *P_double;
double a, b;
P_double pa=&a, pb=&b, pc;
scanf (“%lf%lf”, pa, pb);
pc=(*pa<*pb)?pa:pb;
printf (“%g”, *pc);
```

Pokazivači

Može i pokazivač na pokazivač

tip **ime

Primer:

```
int a;
int *pa = &a ;
int **ppa = &pa;
```

Pokazivači

```
int *a[5]
```

a - niz pokazivača na int

```
int (*a)[5]
```

a - pokazivač na niz celih brojeva

```
int *f()
```

f – funkcija koja vraća pokazivač na int

```
int (*f)();
```

f – pokazivač na funkciju koja vraća int

Operacije sa pokazivačima

- **dodela vrednosti**
  jednog pokazivača drugom
- **dodela vrednosti između pokazivača istog tipa**
- **ako ukazuju na različite tipove mora cast**
  (osim ako je jedan generički – void*)

Operacije sa pokazivačima

- **dodavanje i oduzimanje celobrojnog podatka na/od vrednosti pokazivača**
- **+ - ++ -- += -=**
- **kvant promene pokazivača je veličina pokazivanog objekta**
- **generički pokazivač: ne može sabiranje/ oduzimanje celog broja**

Operacije sa pokazivačima

oduzimanje dva pokazivača

upoređivanje dva pokazivača

- **ima smisla za pokazivače na elemente istog niza**
- **rezultati su isti kao da su operatori primenjivani na indekse**
- **nepredvidljivi rezultati ako su pokazivači na nezavisne objekte**

Operacije sa pokazivačima

upoređivanje pokazivača sa nulom

- **provera da li pokazivač uopšte pokazuje na neki objekat**
- **NULL iz <stdio.h> - pokazivač koji ne pokazuje ni na koji objekat**

Pokazivači i nizovi

- **ime niza ⇔konstantni pokazivač na početak niza**

```
&a[0]   ⇔   a
&a[i]   ⇔   a+i
a[i]    ⇔   *(a+i) ⇔ *(i+a) ⇔ i[a]
                    čudno je, izbegavati!
```

Operacije sa pokazivačima

Primer: **(++a i * prio=14, a++ prio=15)**

```
int a[10], *pa, *pb, x, y;
pa = &a[4];
x = *(pa+3);  // x=a[7]
y = *pa+3;    // y=a[4]+3
*pa++;        // povećava se pokazivač
(*pa)++;      // povećava se podatak
*--pa;        // smanjuje se pokazivac
--*pa;        // smanjuje se podatak
pb  = a+2;    // &a[2]
if (pa<pb) …
```

Pokazivači i nizovi

- **Primer za skalarni proizvod:**

```
int i, n;
double a[100], b[100], *pa, *pb, s=0;
for (pa=a, pb=b;
     pa<a+n;
     s+= *pa++ * *pb++);
```

Pokazivači i nizovi

dve definicije znakovnih nizova

- `char tekst[]="Pointeri";`
- `char *p_tekst="Pointeri";`

Pokazivači

prikazivanje pokazivača

- **vrednost pokazivača – adresa**
- **način prikazivanja zavisi od implementacije i računara**
- **konverzija %p**

```
#include <stdio.h>
int main() {
   int x, *px;
   px = &x;
   printf ("%p\n", px);
}
```

Dinamička dodela memorije

- **naredbe za definisanje promenljivih vrše statičku dodelu memorije**
- **za nizove konstantna dužina – često na očekivani maksimum**
- **alokacija u statičkoj zoni memorije**

Dinamička dodela memorije

- **dinamičko alociranje memorije (u vreme izvršavanja)**
- **za podatke čiji se broj i veličina ne znaju pri prevođenju**
- **smeštanje u dinamičkoj zoni**
- **pristup objektima u dinamičkoj zoni preko pokazivača**

Dinamička dodela memorije

- **statički definisani pokazivač može ukazati na dinamički objekat, a on zatim može imati i druge dinamičke pokazivače**

- **jedino ograničenje za broj dinamičkih objekata – veličina slobodnog prostora u operativnoj memoriji**

Dinamička dodela memorije

bibliotečke funkcije
za upravljanje memorijom
(u <stdlib.h>)

- **malloc**
- **calloc**
- **realloc**
- **free**

Dinamička dodela memorije

```
malloc (broj_bajtova)
```

dodeljuje memoriju za
objekat zadate veličine
(sadržaj nedefinisan)

vrednost funkcije je generički
pokazivač na taj prostor
(ili NULL ako alokacija nije uspela)

Dinamička dodela memorije

```
calloc (n, broj_bajtova)
```

dodeljuje memoriju za
niz od n elemenata
od kojih je svaki zadate veličine

vrednost funkcije je generički
pokazivač na taj prostor
(ili NULL ako alokacija nije uspela)

inicijalizacija nulama

Dinamička dodela memorije

```
realloc (p, broj_bajtova)
```

menja veličinu objekta
pokazanog sa p na zadatu vrednost

smanjuje se od kraja
(čuva se prethodni sadržaj)

dodavanje na kraj
(nedefinisanih vrednosti)

Dinamička dodela memorije

```
realloc (p, broj_bajtova)
```

vrednost funkcije je generički pokazivač na taj prostor (ili NULL ako alokacija nije uspela)

ako ne uspe, mesto i sadržaj objekta ostanu očuvani

Dinamička dodela memorije

`free (p)`

oslobađa prostor na koji pokazuje p

p mora biti ranije dobijeno sa
malloc, calloc, realloc

Dinamička dodela memorije

Primer:

```
long double *p;
…
p = malloc (sizeof (long double));
…
*p=…
…
free (p);
```

Višedimenzionalni nizovi i pokazivači

- **n-dimenzionalni niz je niz čije su komponente n-1 dimenzionalni nizovi**
- **matrica je niz vektora, 3D niz je niz matrica, ...**
- **smeštanje po vrstama**

```
enum {m=3, n=5};
int a[m][n] ;
```

Dinamički nizovi i pokazivači

zbog toga je <u>indeksiranje</u>

ekvivalentno <u>adresnoj aritmetici</u>

```
a[i][j]     ⇔ *((int*)a + n*i + j)
```

- **nema dužine po prvoj dimenziji niza!**
- **a – pokazivač na nizove od 5 int-ova (a+1 pokazivač na narednu vrstu)**

Višedimenzionalni nizovi i pokazivači

- **dinamički niz**

```
int *a = malloc (m*n*sizeof (int));
a[i*n+j] = …
```

- **uz pomoć pokazivača može se smestiti matrica kao <u>nezavisni komponentni nizovi</u>**
- **primer celobrojne matrice od 5 vrsta i 10 kolona**

Višedimenzionalni nizovi i pokazivači

```
#include <stdio.h>
#include <stdlib.h>
main () {
   int *a[5], i, j;
   for (i=0; i<5; i++) {
      a[i]=calloc (10, sizeof(int));
      for (j=0; j<10; j++)
        printf ("%2.2d", *(a[i]+j)=10*i+j);
      putchar ('\n');
   }
}
```

Višedimenzionalni nizovi i pokazivači

pristup elementu može dvojako:

- **ako je objekat definisan kao niz, obično se koristi indeksiranje**
- **ako je indirektan pristup, obično adresna aritmetika**

Višedimenzionalni nizovi i pokazivači

prednosti ovakvog predstavljanja:

- **vrste mogu biti promenljive dužine (čak zadate i u vreme izvršavanja)**
- **pogodno za predstavljanje teksta**

nedostatak:

- **dodatni prostor za pokazivače**

Višedimenzionalni nizovi i pokazivači

ograničenje prethodnog primera:
fiksan broj vrsta

prevazilazi se
jednim skalarnim pokazivačem
i dinamičkom alokacijom
pokazivača na vrste

Višedimenzionalni nizovi i pokazivači

```
#include <stdio.h>
#include <stdlib.h>
main () {
   int **a, i, j, m, n;
   printf("m,n? "); scanf("%d%d", &m, &n);
   a=calloc(m, sizeof(int*));
   for (i=0; i<m; i++) {
      *(a+i)=calloc (n, sizeof(int));
      for (j=0; j<n; j++)
         printf ("%2.2d", *(*(a+i)+j)=10*i+j);
      putchar ('\n');
   }
}
```

Pokazivači na nizove

```
int (*p)[5];
int v[5], w[10], m[3][5];
p=v;     // GREŠKA: Neslaganje tipova
p=&v;    // OK
(*p)[3]=55;  // v[3]=55
p=&w;    // GREŠKA: Neslaganje dužina
p=m;     // OK; m je niz nizova
p[1][3]=55;  // m[1][3]=55
p=&m[1]; // Pokazivač na drugu vrstu
(*p)[3]=55; // m[1][3]=55
p=&m; // GREŠKA: pok.
      // dvodimenzionalnog niza
```

Pokazivači na nizove

```
int (*p)[3][5];
int v[5], m[3][5], t[6][3][5];
p=v;      // GREŠKA
p=m;      // GREŠKA
p=&m;     // OK
p=t;      // OK
p=&t[2]; // OK
p=t[2];  // GREŠKA
p=&t;     // GREŠKA
```

Pokazivači na nizove

```
int n=5;
int (*p)[5], (*q)[n];
int a[5], b[10], c[n], d[2*n];
p=&a;
p=&b;    // GREŠKA: neslaganje dužina
p=&c;
p=&d;    // PROBLEM: prevodilac pušta
q=&a;
q=&b;    // PROBLEM: prevodilac pušta
q=&c;
q=&d;    // PROBLEM: prevodilac pušta
```

Ograničeni pokazivači

```
*pa += *pc;
*pb += *pc;
```

- **Kad bi prevodilac znao da sva tri pokazivača pokazuju na različite lokacije, mogao bi samo jednom dohvatiti `*pc`**
- **Uvodi se modifikator `restrict`**
- **Tako nikad dva ili više ne pokazuju na isti podatak**

Ograničeni pokazivači

```
int a = 5; b = 3; c = 2;
int *restrict pa=&a, *restrict pb=&b,
    *restrict pc=&c;
*pa += *pc; //a=5+2=7
*pb += *pc; //b=3+2=5
pc=pa;
*pa += *pc; //a=7+7=14
*pb += *pc; //b=? (5+7=12 ili 5+14=19)
```

Ograničeni pokazivači

- **Nismo poštovali restrikciju pa je rezultat neizvestan kod druge dodele!**
- **Jednostavnije pisanje ako uvedemo tip pomoću** `typedef`

```
typedef int *P;
restrict P pa=&a, pb=&b, pc=&c;
```

Složeni literali

- **Neimenovani podatak datog tipa i sadržaja**

(*tip*) {*vrednost, vrednost, … vrednost*}

- **Primeri:**

```
int i=1; ekvivalent int i = (int) {1};
int k=2;
typedef int Niz[];
int *p1 = (int[5]) {1, 2, 3, 4, 5}
p1 = (int[]) {[9]=1 [4]=2}
p1 = (Niz) {k, k*k, k*k*k}
```

Složeni literali

- **Najpre je p1 pokazivač na niz od 5 elemenata: 1 2 3 4 5**
- **Zatim niz od 10 elemenata (jer je 9 najveći indeks): 0 0 0 0 2 0 0 0 0 1**
- **Zatim niz od 3 elementa: 2 4 8**

```
typedef int Matr[][3];
int (*p2)[3] = (int[3][3]){{1, 2, 3},
                           {4, 5, 6}, {7, 8, 9}};
p2 = (int[][3]) {{1},{[1]=2},{[2]=3}};
p3 = (Matr) {k, k+1, k+2, k+3, k+4};
```

Složeni literali

- **Najpre je p2 pokazivač na matricu:**
  1 2 3
  4 5 6
  7 8 9
- **Zatim je p2 pokazivač na matricu:**
  1 0 0
  0 2 0
  0 0 3
- **Zatim matrica od dve vrste:**
  2 3 4
  5 6 0

Bibliotečke funkcije

- **Nekoliko funkcija za rad sa blokovima bajtova; potrebno zaglavlje `<string.h>`**
- **`t, u, s` su tipa `void *` i treba da pokazuju na blok dužine najmanje `n`**
- **prve tri funkcije vraćaju t**

`memmove(t,s,n)` – prepisuje prvih n bajtova iz s u t

`memcpy(t,s,n)` – isto, ali bez preklapanja s i t

`memset(t,k,n)` – stavlja prvi bajt iz k u prvih n bajtova niza t

Bibliotečke funkcije

`memcmp(u,s,n)` – upoređuje prvih n bajtova `u` i `s`. <0 za u<s, >0 za u>s, =0 ako su svi parovi bajtova jednaki

`memchr (u,b,n)` – vraća pokazivač tipa `void *`, pokazuje na prvi bajt u `u` koji je jednak bajtu najmanje težine u `b` tipa `int`. `NULL` ako nije nađen.

Bezbedne varijante sa dodatnim argumentom m koji predstavlja dužinu t:

`memmove_s(t,m,s,n)`

`memcpy_s(t,m,s,n)`

`memset_s(t,m,k,n)`