C programozási nyelv
Pointerek, tömbök, pointer aritmetika

Dr. Schuster György

2011. június 16.

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem valamilyen objektumnak (változónak, függvénynek) a címét tartalmazza.

Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a `*` karakter jelzi, hogy pointerről van szó.

FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!

Hogyan kap értéket a pointer?

```
int *ip;
int i;
  :
ip=&i;
```

Itt a **&** operátor az úgynevezett címképző operátor. Ez unáris, ne tévesszük össze a bitenkénti és-sel.

Mostmár `azip` az `i` címét tartalmazza.

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket. Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Szó szerint!!

`j` változó értéke legyen egyenlő az `ip` által címzett változó értékével.

Mondom

SZÓ SZERINT!!!

Ezt nevezzük indirekt értékadásnak.

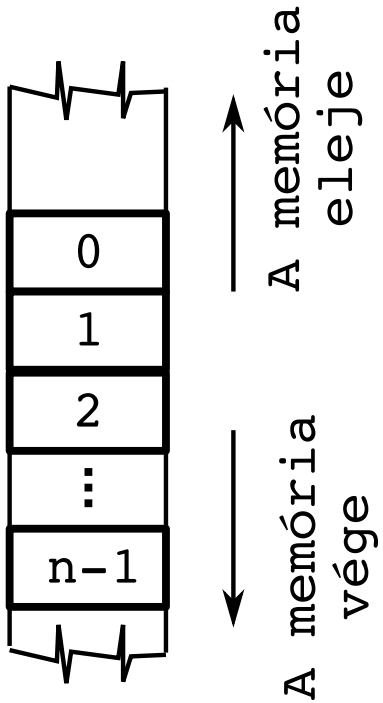
Ha elfelejtjük `ip` inicializálni, a program rendszertől függő hibákat generál. Kultúráltabb rendszereknél futás közben hibaüzenetet kapunk. Sokkal veszélyesebb, ha nem kapunk visszajelzést.

Még visszatérünk a mutatókhoz.

Tömbök

A tömbök a legegyszerűbb összetett adatszerkezetek. Jellemzőik:

- szigorúan azonos típusú adatokat tartalmaznak,
- az adatok egymás után helyezkednek el a tárban.



Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

A tömb egy elemének az elérése, példa:

```
t[5]=23;
```

Tehát az elemet indexeléssel érhetjük el. Az index csak pozitív egész szám lehet. Tehát konstans szám, vagy egész jellegű változó.

```
t[i]=23;
```

ahol `i` egy pl. `int`.

A tömb kérédéses elemét úgy használhatjuk, mint egy azonos típusú közönséges változót.

Probléma a túlindexelés. Változóval történő indexelés esetén futásközben az index kívül van az idexelési tartományon.

Pl. `i` értéke 11.

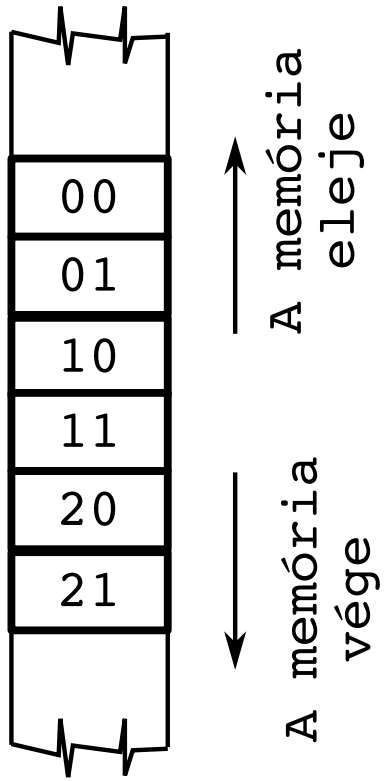
A program rendszertől függő hibákat generál.

Többdimenziós tömbök

Kétdimenziós egész tömb deklarálása, pl.:

```
int t[3][2];
```

A leghátsó index változik a leggyorsabban.
Vagyis.



Tetszőleges számú dimenzió lehetséges. De a dimenziók számának növekedésével exponenciálisan növekszik a memória foglalás.
Az elem elérése itt is indexeléssel történik. Tehát:

```
t[1][1]=5;
```

Az indexekre azonos szabályok vonatkoznak, mint egydimenziós esetben.

Tömbök inicializálása deklarációnál

Megadhatunk kezdőértéket a tömbnek deklarációnál.

```
int t[10]={6,8,-2,6,123,-8,3,4,2,1};
```

Ezt szépen sorban berakja a tömb elemeibe, de csak deklarációnál!
Ha inicializálunk, nem kell megadni a tömb méretét.

```
int t[]={6,8,-2,6,123,-8,3,4,2,1};
```

A fordító kitalálja mekkora legyen a tömb.
Ha kevesebb elemet adunk meg, mint amekkora a méret a fennmaradó helyeket 0-val tölti ki.
Többdimenziós esetben.

Szépen:

```
int t[3][2]={ {1,2},
              {3,4},
              {5,6}};
```

Csúnyán:

```
int t[3][2]={ 1,2,3,4,5,6};
```

Sokkal nagyobb figyelmet kíván, mint a baloldali.

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   ⋮
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.
Vagyis:

`p=t;` azonos **`p=&t[0];`**

Ha tehát ez igaz, akkor a kezdőcím átadása után, használhatjuk a pointert, mint tömböt, pl.:

```
p[5]=13;
```

Ha nem a tömb kezdőcímét akarjuk megadni, akkor csak a címképző operátorral dolgozhatunk.

```
p=&t[5];
```

Akkor most vissza a pointerekhez!

Pointer aritmetika

Deklaráljunk egy `int` pointert és egy 10 elemű `int` tömböt.

```
int *p;
int t[10];
   ⋮
p=t;
```

Ezt eddig már láttuk.

```
p++;
```

Kérdés hová mutat a p ezután.

A következő bájtra, vagy `t[1]` -re?

A válasz: **`t[1]`** -re.

Tehát egy adott pointer inkrementálása annyival (bájttal) növeli az általa tartalmazott címet, amekkora a mérete annak a típusnak, amelyre mutat.

Pointer aritmetika

További műveletek:

- dekrementálás fordítottja az inkrementálásnak,
- egész hozzáadása pointerhez

```
int *p,*q;
p=q+5;
```

p pointer értéke q+5*sizeof(int).

- egész kivonása pointerből

```
int *p,*q;
p=q-5;
```

p pointer értéke q-5*sizeof(int).

- két pointer kivonása megadja a két pointer által mutatott cím távolságát az adott típusban.

Szóval akkor:

`t[5]=13;` ugyanaz, mint `*(t+5)=13;`

Példa a pointer használatára

Írjunk egy olyan függvényt, amely két `int` tipusú változót összead és egyben ki is von egymásból.
Egy függvény egyetlen paramétert adhat vissza a `return` használatával.
Az ötlet az, hogy azoknak a változóknak, amelyekben majd az eredményt kapjuk, a címét adjuk át a függvénynek.
Hogyanis:

```
1  void muv(int a,int b,int *r1,int *r2)
2     {
3     *p1=a+b;
4     *p2=a-b;
5     }
```

Vagyis a 3. sorban `p1` által címzett változóba kerüljön `a, b` összege.
Vagyis a 4. sorban `p2` által címzett változóba kerüljön `a, b` különbsége.

Példa a pointer használatára

A teljes program:

```
void muv(int,int,int *,int *);
int main(void)
   {
   int x,y,i,j;
   x=6,y=7;
   muv(x,y,&i,&j);
   printf("%i %i",i,j);
   return 0;
   }

void muv(int a,int b,int *r1,int *r2)
   {
   *p1=a+b;
   *p2=a-b;
   }
```

`void` pointer

A `void` pointer egy **típus nélküli** pointer.

Akkor szoktuk használni, ha egy memória címre van szükségünk, de nem ismerjük azt a konkrét típust, amelynek a példányára mutat.

A legnevezetesebb `void` pointer a **`NULL`** pointer, amelyenk a definíciója az `stdio.h` header fájlban található meg.

```
#define NULL (void *)0
```

Ez konkrétan a nullás memória címre mutat, amely a programozható eszközök nagy többségében nem használható változó tárolására.
A `NULL` pointer hibajelzésre, szükségtelen paraméterek jelzésére használjuk.
Fájl és dinamikus memória kezelésnél fogjuk látni.

Sztringek

A sztring egy speciális karakter típusú tömb, amelyet egy 0 értékű karakter zár le.

Teljesen mindegy, hogy mekkora a tömb a sztringet a 0 zárja le. Azért ne legyen hosszabb, mint a tömb!

A sztring inicializálása speciális módon történhet deklarációnál és csakis deklarációnál:

```
char nev[]="Schuster";
```

Ez hozzáfüzi a lezáró 0-t is. Tehát a nev tömb 9 elemű.

Ez sokkal egyszerűbb, mintha a hagyományos inicializálást csinálnánk:

```
char nev[]={'S','c','h','u','s','t','e','r',0};
```

A lezáró elem 0 és nem '0'!!

Függvény pointerek

Van egy függvényünk, a deklarációja:

```
int fgv(int);
```

Deklaráljunk egy függvény pointert!

```
int (*fgvp)(int);
```

Adjunk neki értéket!

```
fgvp=fgv;
```

Hívjuk a függvényt a pointeren keresztül!

```
r=fgvp(5);
```