# 6. gyakorlat

## Mostantól 8:15-től kezdünk

Még mindig nem 8:30, de jobb, mint a 8:05.

## Pluszminusz elmaradt

Megszavaztuk. (Én ma írtam volna, de 18 vs. 3 leszavaztak.)

Következő alkalommal lesz, mai anyag nem kerül bele.

## Javítás múlt órai jegyzetbe

char**\*** p**=**"Like"**;**

p**[**0**]=**'w'**;** //Ez mégsem működik teljesen karakterekre.

Futásidőben elszáll. Csak fordításidőben működik.

Nem konstans pointernek oda lehet adni konstans változó címét. (?) T**\*** ← **&(**const T**)**

# Paraméterátadás, objektum orientált kód írása

## Paraméterátadás elméleti típusai

### Érték szerinti

Lemásoljuk a paramétert, a másolatot adjuk át. Ebbe a függvény irkálhat is, nem fog az eredeti módosulni.

### Cím (referencia) szerinti

Ha nagy objektumot adunk át, költséges lenne lemásolni.

Sőt, lehet, hogy épp az eredeti objektumon akarunk dolgozni.

### Cím-érték szerinti, stb.

Kitaláltak sokféle mást is. Ez nekünk most nem fontos.

### Inicializálás szerinti

C++-ban egyféle paraméterátadás van, ez az inicializálás szerinti.

Paraméter átadás / visszaadás mindig kezdeti értékadás.

## Inicializálás szerinti paraméterátadás (C)

int increase**(**int n**){return** **++**n**;}**

int x**=**4**;**

cout**<<**increase**(**x**);**

TFH $n$ nagyon nagy objektum:

LargeObject**\*** fun**(**LargeObject n**){**

 LargeObject tmp**=**n**;**

 //dolgozunk a temporálissal.

 **return** tmp**;**

**}**
(Ez a kód le is fut **typedef** int LargeObject-tel.)

Pointert adunk rá vissza, hogy ne kelljen kétszer másolni.

LargeObject fun**(**LargeObject**\*** n**){**

 ++(\*n);

 **return** n**;**

**}**

Természetesen meghíváskor címlekérdezést kell végezni: increase**(&**x**)**

Vagy eleve pointert adunk/veszünk át. Ez az eredeti objektumot módosítja. Ez általában már kiírás előtt is megtörténik, tehát a cout**<<**x**<<**increase**(&**x**)** 55-öt ír ki, nem 45-öt. Ha közben kiíratjuk a puffert, akkor működik, 45 lesz. / Ezt még lehet tovább optimalizálni, hiszen a visszatérési érték még mindig név nélküli temporális változóba került. Viszont ha pointert veszünk át és pointert is adunk vissza…

LargeObject**\*** increase**(**LargeObject**\*** n**){**

 ++(\*n);

 **return** n**;**

**}**

Ekkor **\*(**increase**(&**x**))**-szel lehet kiírni, vagyis kiíráskor lekérdezzük, hova mutat a visszatérési érték. Ha változóba mentettük: **\***y.

Ha azt szeretnénk, hogy ne másoljunk kétszer, de ne is módosítsunk az eredetin:

LargeObject**\*** increase**(**LargeObject**\*** n**){**

 LargeObject tmp **=** **\***n**+**1**;**

 **return** **&**tmp**;** //Hiba

**}**

Mi a hiba: A lokális változó a függvény futása után megsemmisült. Erre adtunk vissza pointert, ami nem fog működni, mert a stack-en hoztuk létre. Szabály: Sosem adunk vissza függvényből lokális objektumra mutató pointert!

Nem fogjuk megúszni a dupla másolást:

LargeObject increase**(**LargeObject**\*** n**){**

 LargeObject tmp **=** **\***n**+**1**;**

 **return** tmp**;**

**}**

Kétszer másol. Először a lokális változó létrehozásánál, másodszor a visszaadásnál.

Ez nem így szokott előfordulni, hanem: El akarunk végezni egy feladatot, de nem szeretnénk módosítani a paramétert.

void foo**(**const LargeObject**\*** n**){**

 //do something;

**}**

foo**(&**x**);**

Ha csak átadjuk érték szerint, akkor kétszer másol, nem jó.

Ha pointert adunk át, véletlen módosíthatjuk a paramétert.

Ha konstansként adjuk át, minden működik.

Ez mind C volt. Nézzünk C++-t!

## C++-os paraméterátadás

Vegyünk egy egyszerű vektort:

myVector**:**

int capacity //max. elemszám

push //elemet rak be

pop //elemet vesz ki (pozícióról)

size //méret lekérdezése (elemszám)

getItem**(**i**)** //i-edik elem lekérdezése

Lefoglaljuk heap-en:

T**\*** p**=new** T**[**n**]**

Rossz C-s kód:

int**\*** v**;** //ez a tömb. Hiba: globális változó

const int CAPACITY **=** 10**;**

int size**;**

void init**(**int**\*** t**,** int size**){**

 t**=new** int**[**size**];**

 **for(**int i**=**0**;**i**<**size**;++**i**){**

 t**[**i**]=**0**;**

 **}**

**}**

int main**(){**

 init**(**v**,** CAPACITY**);**

 push**(**v**,**4**);**

 push**(**v**,**5**);**

 **for(**int i**=**0**;**i**<**size**;++**i**){**

 std**::**cout**<<**i**<<**". elem: "**<<**getItem**(**v**,**i**)<<**std**::**endl**;**

 **}**

**}**

Ha letesztelnénk (csak $init$ után, kiírással), eredménye: "Segmentation fault" lenne. Maga az operációs rendszer vágta el a programot.

Miért: Stack-en van a $v$ pointer. Ezt a memóriaszeméttel teli pointert pedig átadtuk a függvénynek. (Mivel csak egyféle paraméterátadás létezik, a kezdeti érték szerinti.) A lefoglalt memóriát visszatölti $t$-be, ami ettől kezdve nem lesz egyenlő $v$-vel.

Javítása: void init**(**int**\*&** t**,** int size**){**

Tehát $t$ referencia egy pointerre, ami int-ekre mutathat. Mi az a referencia:

## Referencia

int**\*&**t**=**v**;** //pl.

int x**=**4**;**

int**&** r**=**x**;** //Ez a referencia

Ha a változó deklarációban szerepel a $\&$ jel, akkor mindig referenciát hozunk létre. (Kivéve int **\***p**=&**x**;** mert a $\&$ jel már a kifejezés jobb oldalán van, ami nem a deklaráció része.)

Ha nem deklarációban tesszük ki, akkor címképzés. Az adott objektum címére mutató pointert ad vissza.

### Mi az a referencia?

C++ feature.

Nem hoz létre új változót, csak egy álnév (alias).

Nem kerül memóriába. Bármit teszünk vele, az nem a referenciával történik, hanem az általa vonatkozott változóval. Ha lekérdezzük a címét, az is x címét kérdezi le. Nem olyan bizonytalan, mint a pointer. Nem lehet átállítani másik változóra. Nem lehet kezdőérték nélkül megadni:

int**&** r**;**//Hiba

Gyakorlat vége.