# 2. előadás

## Berezvai Dániel jegyzete <http://elte.3ice.hu/>

Memóriakezelés

→ \ELTE\OAF\EA\dia\2.pointer.pdf

Diák sorrendje nem jó. Dinamikusan lefoglalt (new) változók és struktúrák előbbre jöttek.

p->t === \*p.t

## Dinamikusan lefoglalt objektum

(hiányzó dia)

class Pont**{**

 private**:**

 int \_x**,** \_y**;**

 public**:**

 int x**();**

 void ujX**(**int x**);**

 //...

**}**

Pont **\***p **=** **new** Pont**;**

int i **=** p**->**x**();** //\*p.x()

p**->**ujX**(**5**);**

p**->**\_x**;** // Nem!

**delete** p**;**

A this arra az objektumra mutató pointer változó, amelyre az ujX() metódust meghívtuk:

void Pont**::**ujX**(**int x**){**

 **this->**\_x **=** x**;** // === \_x=x;

**}**

## Referencia változó

$$int i=3;$$

$$int \&r=i;$$

$r$ az $i$ "álneve"

Nem elég deklarálni, azonnal értéket kell adni a referencia változóknak. Függvényeknél használjuk például:

### Referencia szerinti paraméterátadás

$$void csere\left(int \&a, int \&b\right);$$

Ha elhagynánk a "&" jeleket, nem történne csere. Átmásolódnak az új $a$ és $b$ változókba, de a függvény futása végén törlődnek is.

Bármit csinálunk referencia szerint átadott változóval, az látszik a függvényen kívül is.

## Konstans referencia szerinti paraméter átadás

Akkor hasznos, ha az átadott adat összetett szerkezetű. Nem akarjuk terhelni a programot felesleges másolgatással.

$$void f(const valami \&x)$$

## Automatikusa helyfoglalású mátrix

Automatikusa helyfoglalású mátrix paraméter átadásban rossz. Helyette:

## Dinamikus helyfoglalású mátrix

int**\*\*** w**;**

Nem kell egy hatalmas $n\*m$-es memóriablokk, soronként más-más helyre tehetőek az adatok.

Paraméter átadásban HF.

Előadás vége. 13:00