# 3. előadás

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

Kezdés: 12:05

→ \ELTE\OAF\EA\dia\3.dinamikus\_osztaly.pdf

## Statikus terv

3 db pont osztály **kompozíciós kapcsolat**ban van a sokszög osztállyal. (Háromszög esetén. Ha 4 pont; négyszög)

## Sokszög osztály

**Invariáns** tulajdonság (mindig igaz), hogy legalább 3 csúcsa van.

Súlypont kiszámítása:

## Sokszög osztály C++ kódja (header)

Első beadandóban még az a cél, hogy nehéz dolguk legyen 🡺 nem szabad vektort használni.

Vektor helyett tömb és külön nyilvántartott mérete:

private**:**

int \_meret**;**

Pont**\*** \_pontok**;**

## Sokszög osztály C++ kódja ()

Konstruktor: Először ellenőrzi az invariáns tulajdonságot. Ha túl kevés csúcs, kivételt dob.

Van konstruktor, ezért ez a tömblétrehozó kód jó (és szép!):

\_pontok **=** **new** Pont**[**\_meret**];**

Indextúlcsordulás esetén is kivételt dobunk. (Csúcslekérdező függvényben.)

## Indexelő operátorok

Lekérdezés és megváltoztatás szignatúrái különbözőek. (Egyik , másik referenciát ad vissza a kért pontra. Így tudjuk megváltoztatni)

# Kell még egy másoló konstruktor (fontos!), hogy ne ugyan az a tömb legyen mindkét sokszög mögött

(Ez a legfontosabb dia a mai előadáson.)

Példa: Létrehozunk egy -t, beállítjuk pontjait, majd lemásoljuk -be. Megváltoztatjuk egyik pontját. Mi történik -vel? A háromszög is megváltozik! Ugyan az a reprezentáció van mindkét háromszög mögött, mert a default copy constructor nem elég jó. Az alapértelmezett **sekély másolás** helyett most mély másolás kell.

## Írjuk meg kézzel

//copy ctor

Sokszög**::**Sokszög**(**const Sokszög **&**s**){**

\_méret **=** s**.**\_méret**;**

\_pontok **=** **new** Pont**[**\_méret**];**

**for(**int i**=**0**;**i**<**\_méret**;**i**++)** \_pontok**[**i**]=**s**.**\_pontok**[**i**];**

**}**

Sokszög**&** Sokszög**:operator=(**const Sokszög **&**s**){**

**if(this==&**s**)** **return** **\*this;**

**delete[]** \_pontok**;**

\_méret **=** s**.**\_méret**;**

\_pontok **=** **new** Pont**[**\_méret**];**

**for(**int i**=**0**;**i**<**\_méret**;**i**++)** \_pontok**[**i**]=**s**.**\_pontok**[**i**];**

**return** **\*this;**

**}**

Most nem muszáj a gettereket és settereket használni, mert a sokszög osztályon belül vagyunk.

(A dián keverve van a \_méret és az \_oldalszám változó. Én javítottam mindet \_méretre. A kivetített dia frissebb, mint a letölthető, már javítva van.)

Miért kell destruktor az értékadásba? Memóriaszivárgás megelőzése végett. Az eddig nyilvántartott memóriafoglalást törölni kell, mielőtt újat hozunk létre helyette.

## Osztályok csoportosítása tárfoglalásuk helye szerint

### Egyszerű

Nincs benne . parancs. (Legfeljebb közvetett módon, pl. vektor használata esetén.)

### Dinamikus

~ foglalást végző osztály.

Kell saját copy konstruktort létrehozni, stb.

## Sokszög osztály főprogramja (main)

… Miért kell mátrix? Mert nincs paraméter nélküli (üres) konstruktor. Nem is lehet. Invariáns tulajdonság miatt kötelező méretet megadni. Ezért lesz a "Sokszögekre mutató pointerek tömbje".

Felszabadítás: Program végén végig kell menni a sokszögeken és egyenként fel kell szabadítani mindet, majd a tömböt lehet törölni. (Hibás a dia, fel lesz töltve javított változat Neptunba.)

A legszebb sor:

**(\***s**)[**i**].**ujX**(**x**);** **(\***s**)[**i**].**ujY**(**y**);**

Miért ? Mert memóriacím, és az az által mutatott tömb -edik elemének az és pontját kell beállítani.

## Konzolos részt vegyük külön (hiányzó dia)

Ha valaki például grafikusan szeretné kirajzolni a sokszögeket, nem kell a kiírás. (Szerintem nem zavar senkit, debuggolni pont jól jön, ha megmarad. Ne vegyük külön.)

## Típusorientált, OOP, menü vezérelt, osztályszintű elemek

Ez alapján már az első beadandó megoldható.

Előadás vége.