# Objektum elvű alkalmazások fejlesztése 4. előadás

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

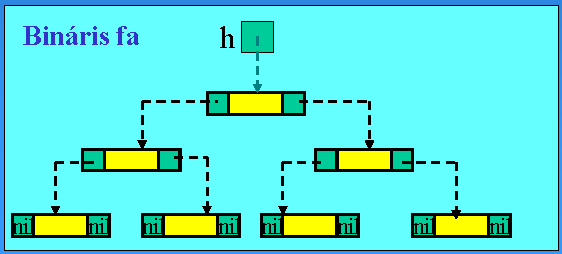
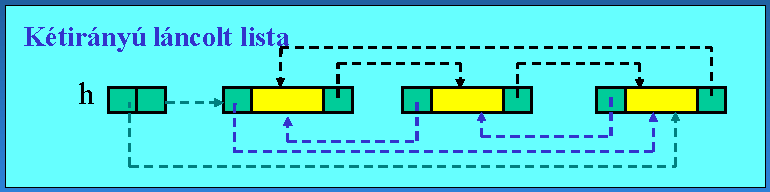
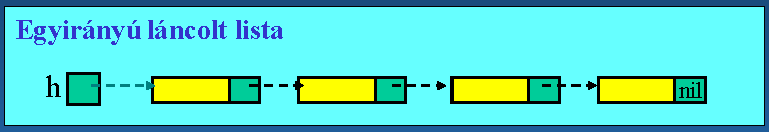
Tanár megint Gregorics Tibor.

Kezdés: 14:10

## Egyirányú láncolt lista

Tömb helyett az adatelemeket szétszórjuk a memóriában. Majd összeláncoljuk valamilyen módon.

### Három példa



## Listaelem

Adat rész + mutató rész.

struct Node **{**

int value**;**

Node **\***next**;**

**};**

## Létrehozás és lebontás

Node**(**int i**=**0**,** Node **\***q**=NULL):**value**(**i**),** next**(**q**){}**

A kettőspont után a default paraméter értékek vannak.

### Listaelem létrehozása

Node **\***p **=** **new** Node**(**3**);**

Node **\***p **=** **new** Node**();**

### Listaelem lebontása

**delete** p**;**

## Fejelemes vagy fejelem nélküli egyirányú láncolt lista

Első listaelemben csak meta​adatot tárolunk (pl. hossz)

Fejelem előnyei számosak. Célszerű mindig fejelemes láncolt listát használni. Csak (pl. hatékonysági) indokkal hagyhatjuk el.

## Felépítés

Feltételezzük, hogy valameddig már felépítettük a listát, van egy kezdeményünk.

### Egy újabb elem

Node **\***u**;**

//...

u**->**next **=** **new** Node**(**35**);**

u **=** u**->**next **;**

### Teljes lista

Node **\***h**,\***u**;** //Nem Node\* h, u

**for(**int i**=**1**;**i**<=**n**;**i**++){**

u**->**next **=** **new** Node**(**i**);**

u **=** u**->**next **;**

**}**

Ciklus elé inicializálás kell, ez kétféleképpen lehetséges:

Fejelemmel

Node **\***h **=** **new** Node**();**

Node **\***u **=** h**;**

Fejelem nélkül

Node **\***h **=** **new** Node**(**1**);**

Node **\***u **=** h**;**

Továbbá fejelem nélkül az index -től megy egy helyett.

## Lebontás

Jó programozó nem engedi a memóriát elszivárogni.

Használat végén végigmegyünk egyesével a lista elemein és mindegyiket megszüntetjük.

A rákövetkező elem címét törlés előtt megjegyezzük, hogy ne veszítsük el a listát törlés közben.

**while(**h**!=NULL){**

Node **\***p **=** h**;**

h **=** p**->**next**;**

**delete** p**;**

**}**

## Beszúrás

### Adott elem után

Node **\***u**;**

//...

u**->**next **=** **new** Node**(**23**,**u**->**next**);**

### Elejére, ha nincs fejelem

Ha van fejelem, akkor az első elemet is a fejelem mögé szúrjuk be, vagyis a fenti kód kell hozzá. Ha viszont nincs fejelem, akkor másféleképpen kell:

h **=** **new** Node**(**23**,**h**);**

Fejelemes listák előnyben, mivel egyszerűbb beszúrni, nem kell speciális esetet kezelni.

## Kifűzés

### Törlés adott elem mögül

Node **\***p **=** u**->**next**;**

**if(**p**!=NULL){**

u**->**next **=** p**->**next**;**

**delete** p**;**

**}**

### Fejelem nélkül megint speciális eset, ha az első elemet akarjuk törölni

**if(**h**!=NULL){**

Node **\***p **=** h**;**

h **=** h**->**next**;**

**delete** p**;**

**}**

## Felsoroló objektum láncolt listára

Programozási tételeket ennek segítségével meg is tudjuk oldani.

t**:**enor**(**E**)** **~** p

t**.**Current**()** **~** p**->**value

t**.**First**()** **~** p **=** h**->**next

t**.**End**()** **~** p **==** **NULL**

t**.**Next**()** **~** p **=** p**->**next

### Fejelem nélkül más a First

t**.**First**()** **~** p **=** h

## Ha az előző listaelemet is szeretnénk nyilván tartani

Főleg láncolt lista adatszerkezeteknél van erre szükségünk.

Újra megkeresni az aktuális elem előtti elemet nagyon drága lenne. Lineárisan végig kellene menni az egész listán az adott elemig.

t**:**enor**(**E**)** **~** Node **\***p**,** **\***u

t**.**Current**()** **~** p**->**value

t**.**First**()** **~** u **=** h

p **=** h**->**next

t**.**End**()** **~** p**==NULL**

t**.**Next**()** **~** u **=** p

p **=**p**->**next

Invariáns tulajdonság: és mindig egymást követő elemekre mutatnak.

## Maximumkiválasztás láncolt listára

t**.**First**();**

int max **=** f **(** t**.**Current**()** **);**

int elem **=** t**.**Current**();**

**for** **(** t**.**Next**();** **!**t**.**End**();** t**.**Next**()** **)** **{**

**if** **(** max **<** f **(** t**.**Current**()** **)** **{**

max **=** f **(** t**.**Current**()** **);**

elem **=** t**.**Current**();**

**}**

**}**

Konkrétan:

Node**\*** u **=** h**;**

Node**\*** p **=** h**->**next**;**

int max **=** f **(** p**->**value **);**

Node**\*** ep **=** p**;**

**for** **(** u **=** p**,** p **=** p**->**next**;** p **!=** **NULL;** u **=** p**,** p **=** p**->**next **)** **{**

**if** **(** max **<** f **(** p**->**value **)** **{**

max **=** f **(** p**->**value **);**

ep **=** p**;**

**}**

**}**

Nem az értéket, hanem az azt tartalmazó listaelem címét tároljuk

UPDATE: Frissebb jegyzetben már ep helyett e és p pointerek vannak. Ezt még a régi jegyzetből másoltam. (Nincs frissítve.)

## Lineáris keresés

### Alap:

bool l **=** **false;**

int elem**;**

**for** **(** t**.**First**();** **!**l **&&** **!**t**.**End**();** t**.**Next**()** **)** **{**

elem **=** t**.**Current**();**

l **=** felt **(** elem **);**

**}**

### Listásítva:

bool l **=** **false;**

Node**\*** prev**,** **\***cur**;**

Node**\*** u**,** **\***p**;**

**for** **(** u **=** h**,** p **=** h**->**next**;** **!**l **&&** p **!=** **NULL;** u **=** p**,** p **=** p**->**next **)** **{**

prev **=** u**;** cur **=** p**;**

l **=** felt **(** p**->**value **);**

**}**

# Példa feladat: rendezés

Rendezzük növekvő sorrendbe a szabványos bemenetről érkező egész számokat!

Használjunk beszúrásos rendezést!

## Programterv

* Bemenő adat: számsorozat (szabványos bemenet)
* Kimenő adat: rendezett számsorozat (szabványos kimenet)
* Segéd adat: számsorozatot tartalmazó láncolt lista

1. Minden lépésben vesszük a következő számot, elhelyezzük egy listaelemben és beszúrjuk a rendezettségnek megfelelő helyre egy fokozatosan épülő láncolt listába.
2. A beszúrás előtt listaelem vagy annak a listaelemnek a címe, amely után már olyan listaelem következik, aminek az értéke nagyobb a beszúrandó elemnél, vagy a legutolsó listaelem.
3. Végül végigolvasva a listát kiírjuk az értékeit (a rendezett sorozatot) miközben a listát lebontjuk.

UPDATE: Megint más a kivetített, mint a feltöltött dia.

Kód:

→ 3. Lancolt\_lista.pdf 19. oldal

Előadás vége.