Adatszerkezetek I. ZH kalauz

1. dia

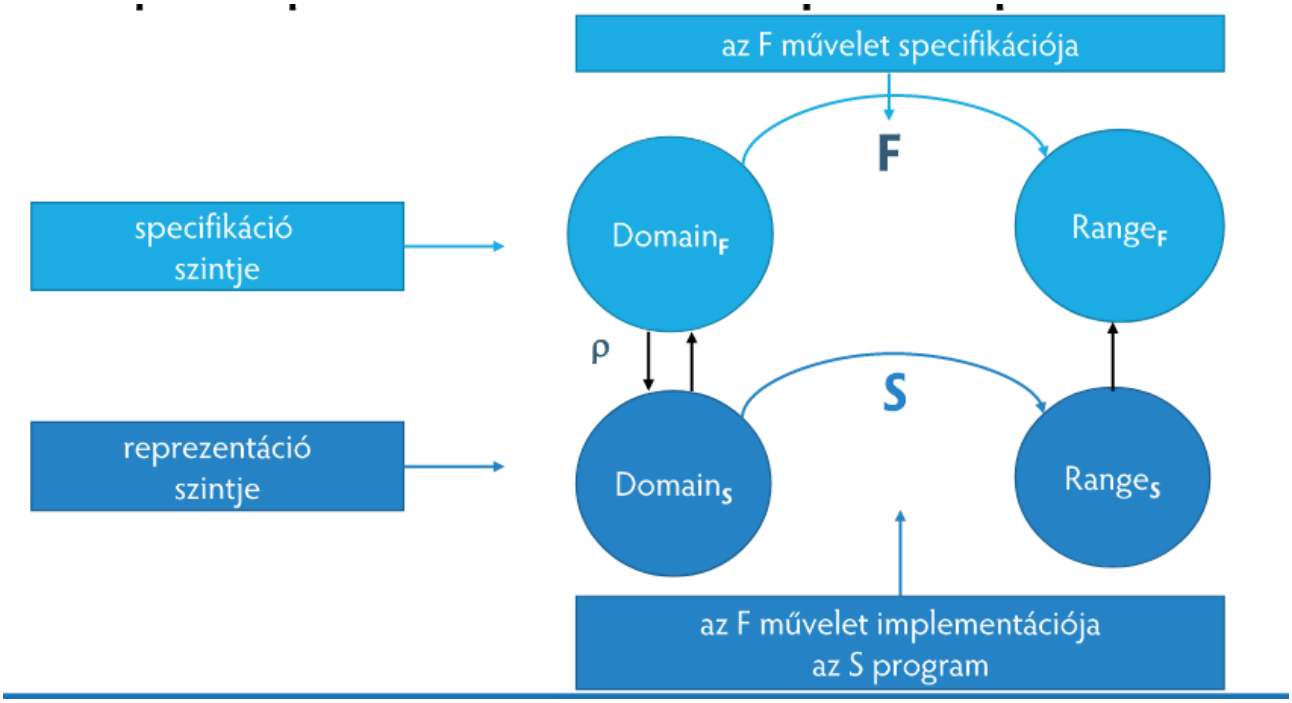
A **típus** egy adat által felvehető lehetséges értékek halmaza, és a rajta értelmezett műveletek együttvéve.

- **Absztrakt adattípus (ADT)**
  - belső szerkezetről nem tudunk semmit
  - típus szemlélet legmagasabb szintje
  - Matematikus leírás
- **Absztrakt adatszerkezet (ADS)**
  - rákövetkeztetések az elemek között
  - irányított gráfok vannak az elemek közt (id-k / pointerek)

A **típus-specifikáció** egy adat külső jellemzésére szolgál. Meg kell adni a típusérték-halmazt, és a típus műveleteket (értelemszerű).

A **típus-reprezentáció** típusértékek ábrázolását jelenti. Egy olyan leképezés, mely az ábrázoló elemek és a típusértékek között leír egy leképezést.

A típus specifikáció és reprezentáció kapcsolata



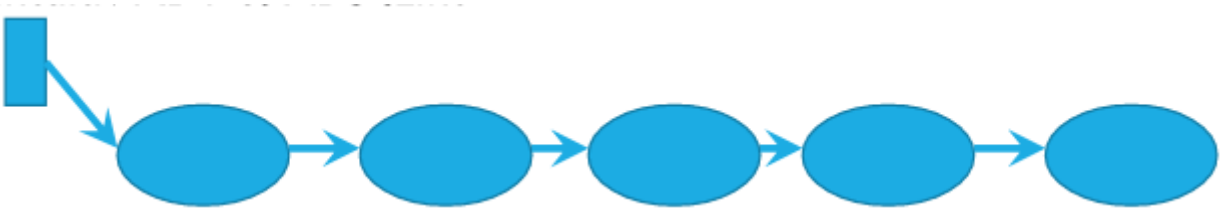
A verem ADT axiomatikus leírása Műveletek: empty, isempty, push, pop, top	**A verem ADT funkcionális leírása** A erem rendezett párok halmaza, mely tartalmazza az adott elem értékét, és a verembe

Megszorítások: pop, top nem érvényes üres halmazra Axiómák: • isempty(empty) • isempty(v) → v = empty • \nem isempty(push(v,e)) • pop(push(v,e)) = (v,e) • push(pop(v)) = v • top(push(v,e)) = e	helyezés időpontját Megszorítás: idő értékek különbözők Valóságban nem implementálunk így, mivel a maradék kura bonyolult fölösleges megtanulni

Az **adatszerkezet** egy <AR> rendezett pár, ahol A adatelemek éges halmaza, az R pedig az A halmazon értelmezett valmilyen reláció R része (AxA)-nak.

Az adatelemek típus szerint lehet egy adatszerkezet **homogén** (azonos típusú), **heterogén** (különböző típusú).

A **reláció** lehet struktúra nélküli, **asszociatív** címzés (nincs kapcsolat, de tartalmuk alapján címezhetők), **szekvenciális** (ez a klasszik láncolt lista, minden elem valakihez kötődik)



Szekvenciális adatszerkezet, olyan adatszerkezet, ahol az R tranzitív lezártja (olyan R-t tartalmazó reláció, maly tranzitív és min. elemszámú) teljes rendezési reláció. Az egyes elemek egymás után helyezkednek el, az adatok között egy-egy jellegű kapcsolat. Két kitüntett elem: az első és az utolsó.

Hierarchikus az az adatszerkezet, ahol van egy kitüntetett r elem, mely nem végpont, minden más elem elérhető belőle, és minden max 1x végpont. Egy elemnek lehet több rákövetkezője is, de csak egy megelőző eleme van

Az adatelemek száma szerint lehet statikus és dinamikus adatszerkezet.

Az adatelemek osztályzása

- **Elemszám**: statikus, dinaumikus
- **Reprezentáció szerint**: folytonos (egymás után vannak), szétszórt (egymásra hivatkoznak)

2. előadás

Objektum: belső állapota van, melyben információt tárol, feladatokat tud végrehajtani, kommunikálhat más objektumokkal, egyértelműen azonosítható.

Osztály: objektum minta, melyből példányt hozhatunk létre.

Példány: egy osztály alapján létrejött konkrét példány

Objektum inicializás: konstruktor végzi, kezdőérték megadás, szükséges előtevékenységek

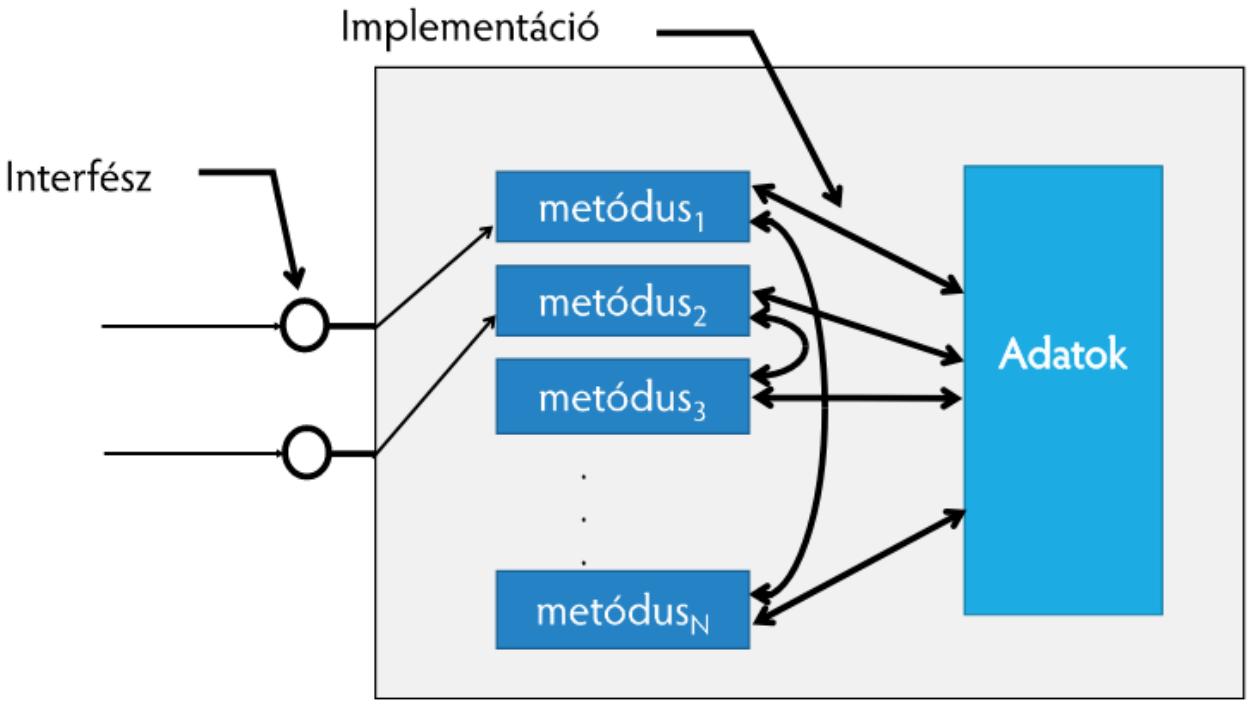
Objektum műveletek: export (más objektum hívja (pl push)), import (neki kell, hogy tudja cisnálni az exportokat)

Export művelet lehet létrehozó (konstruktor), vagy állapot változtató (pop, push), szelektor (kiemeli az objektum egy részét), kiértékelő (jellemzőket lekérdező művelet, pl size), iterátor (bejárás)

OOP elvárások

- Bezárás (enkapszuláció): adatok és metódusok összezárása (class)
- Információ elrejtése: láthatóságok
- Kód újrahasználása

Információ elrejtése



Láthatóságok:

- public: mindenkié

- protected: osztályé és leszármazottaké
- private: csak az osztálynak és a barátoknak elérhető

Az **objektumorientált program** egymással kommunikáló objektumok összessége, melyben minden objektumnak megvan a feladatköre.

Öröklődés

Az alapgondolat: a gyerekek öröklik a szülők metódusait és változóit.

Az öröklött tehet új műveleteket, adattagokat, átdefiniálhatja az örökölt dolgokat is.

Polimorfizmus az a jelenség, hogy egy változó nem csak egyfajta típusú objektumra hivatkozhat. Pl Péter Manager osztály tagja is, ami a leszármazottja az Employeenak, tehát ő is Employee

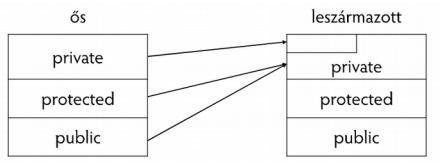
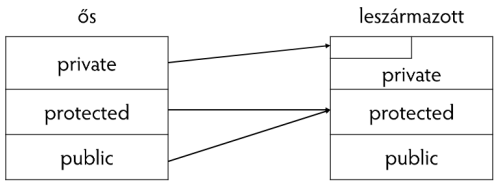
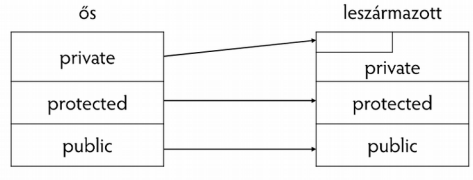
Dinamikus összekapcsolás: Az a jeélenség, hogy a változó éppen aktuális dinamikus típusának megfelelő metódus implementáció hajtódik végre. (run-time)

A C++ virtuallal jelzi, hogy mi az amit enged felüldefiniálni.

Többszörös öröklődés: lehetséges, de az adattagok hányszor, melyiknek a metódusát hívjam meg? → nincs megoldás ilyet nem lehet lefordítani kétértelmáség miatt crashel

Az **absztrakt osztály** egy felső szinten összefogja a közös tualjdonságokat. A metódusok között an olyan ,aminek csak specifikációja van és törzse nincs, majd a leszármazottak konkréttá teszik (pl widget osztály)

Láthatóság öröklődésnél

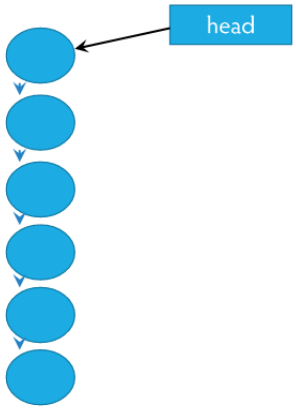
Privát	
Protected	
Public	

Ami öröklődik: adattagok, metódusok

Ami nem: konstruktor, destruktor, értékadó operátorok, barátok

Amit bevezethet: adattag, metódus, felüldefinálhatja őket, konstruktor, destruktor, barátok

Előadás 3



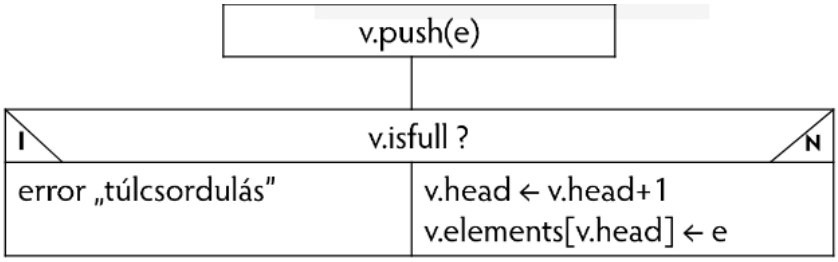
Verem: **LIFO**: Last in first out : kimászás sorrendje

Artimetikai ábrázolása

- egy max hosszú tömb
- a verem tetejének mutatója : head
- ha head = 0 üres a verem
- választhatunk, hogy az első szabad helyre, vagy az azelőttire mutat a head.

Műveletek pszeudo kódja

- push



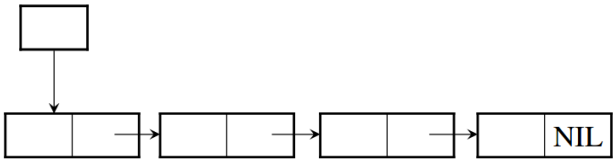
- pop

```
if v.isempty
  then error „alulcsordulás”
  else v.head ←v.head -1
    return v.elements[v.head+1]
end if
```

- top

```
if v.isempty
  then error „alulcsordulás”
  else return v.elements[v.head]
end if
```

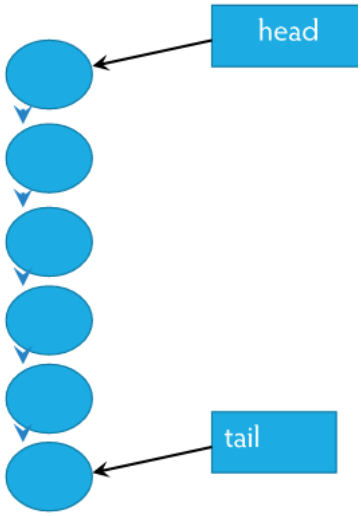
Reprezentáció



Sor: **FIFO**: first in first out, hétköznapi fogalma

Műveletek:

- empty
- isempty
- in (elem betétele)
- out (elem kivétele)
- first (első elem)



Az out és a first nincs értelmezve az üres soron.

Az axiómák:

- isempty(s) <=> s = empty
- \nem isempty(in(s,e))

Artimetikai ábrázolás:

- egy max hosszú vektor
  - az elemek tömbje
- az első elem mutatója
- az utolsó elem mutatója
- üres e atrribútum

Pszeudo kódok:

- in

```
if s.IsFull
  then error „túlcsordulás”
  else s.empt ← false
    s. elements[s.tail] ← e
    if s.tail=max
      then s.tail ← 1
      else s.tail ← s.tail+1
    end if
end if
```

- out

```
if s.empt
  then error „alulcsordulás”;
  else e ← s.elements[s.head]
       if s.head=max
         then s.head ← 1
         else s.head ← s.head+1
       end if
       if s.head=s.tail then s.empt ← true end if
       return e
end if
```

- first

```
if s.empt
  then error „alulcsordulás”
  else return s. elements[s.head]
end if
```

Elsőbbségi sor

ADT axiomatikus leírás

Műveletek:

- empty
- isempty
- insert
- delmax
- max

Megszorítások: delmax és max értelmezési tartományában nincs üres sor

Axiómák:

- isempty(empty) <=> p=empty
- \nem isempty(insert(p.n))

Reprezentáció: tömbbel

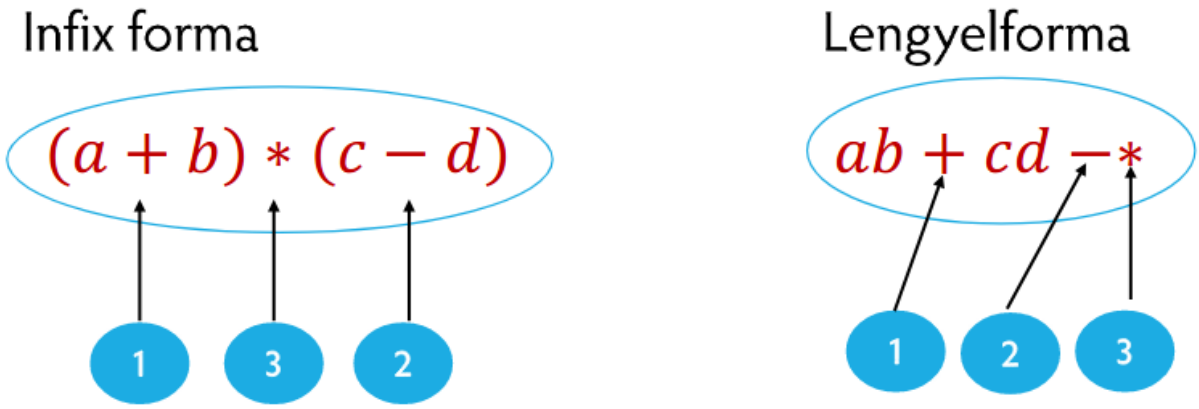
Lengyel forma

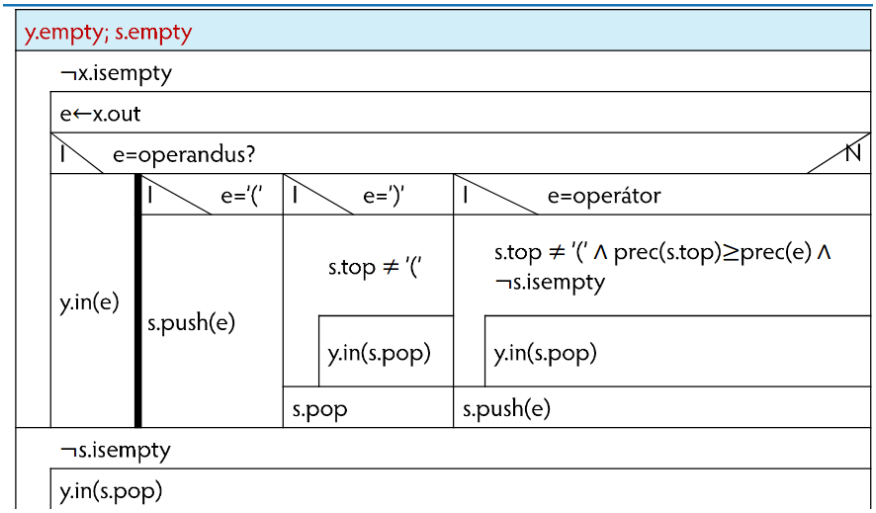
Kifejezések lehetnek

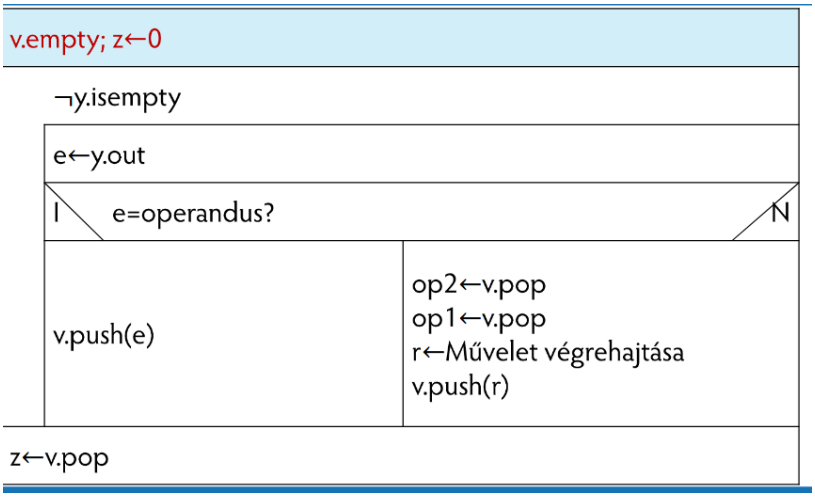
- infix: operátor az operandusok között : a + b
- postfix: operátor az operandusok mögött: a b +
- prefix: operátor az operandusok előtt: + a b

A lengyel forma a predix alak, a fordított lengyel forma a postfix

előny: máveletek olyan sorrendben, ahogy csinálod őket







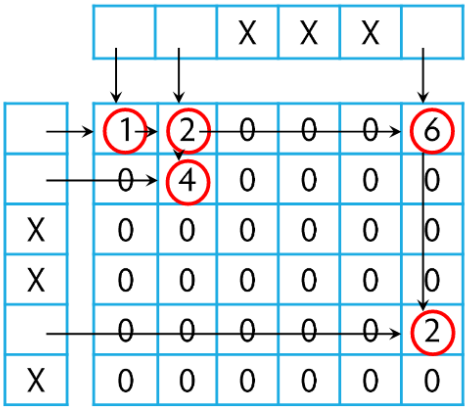
Előadás 4

A k>0 dimenziós **tömbtípus** áll egy I indexhalmazból, és megfelelő számú elemből, melyek megfeleltethetők az indexeknek.

Tehát van egy függvény is, ami összeköti a kettőt.

Nem kötelező a reláció az elemek között.

A legalább 2 dimenziós tömböt **ortogonális adatszerkezetnek** nevezik.

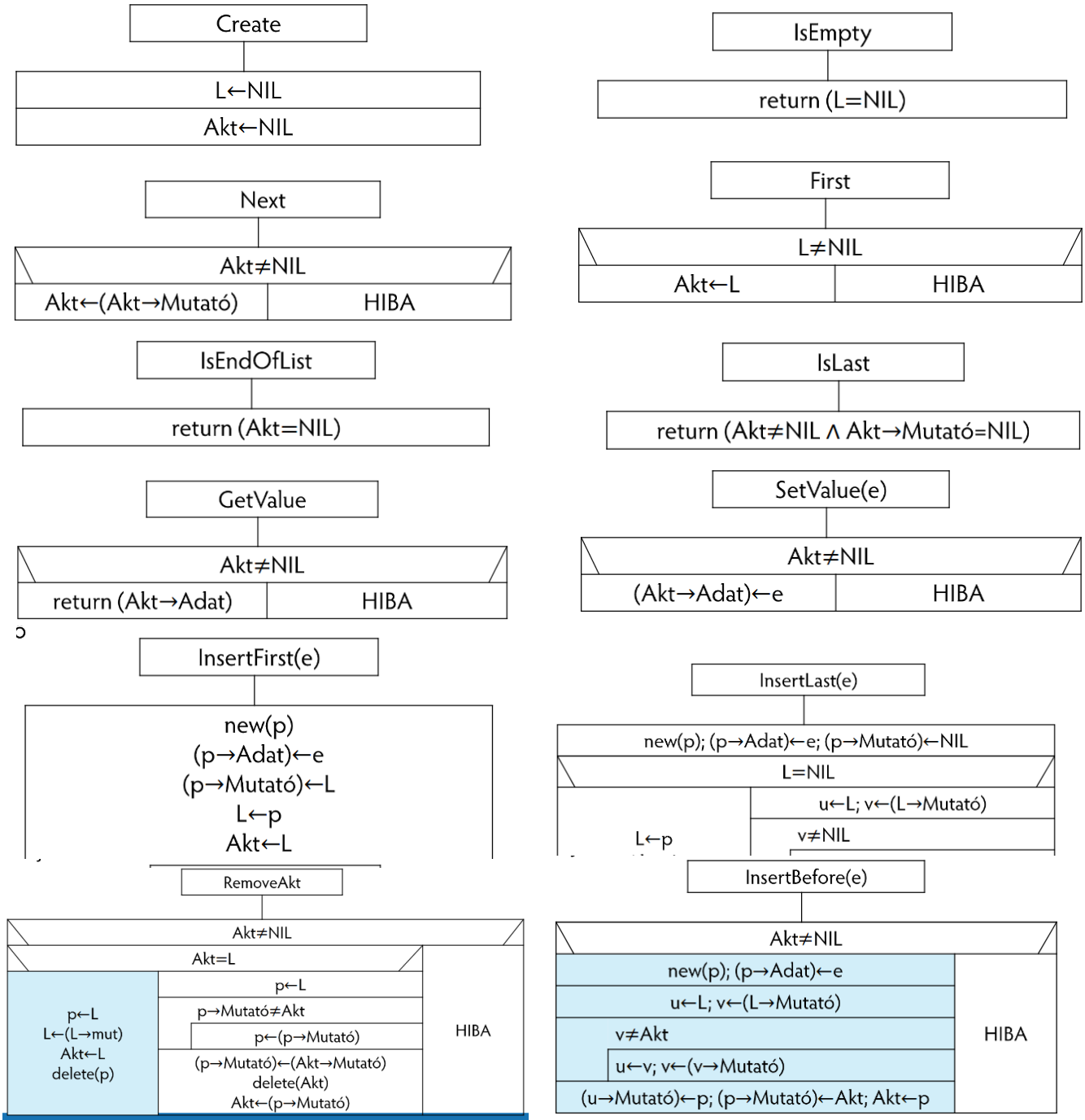


Láncolt ábrázolás

A **láncolt lista** olyan adatszerkezet, amelynek minden eleme tartalmat egy vagy több mutató egy másik ugyanolyan típusú adatelemre ez a "következő elem" logikailag.

A lista feje tartalmazza az első node címét, értéke nincs. A lánc végén nullpointer van.

Lisra típus komponensei: első elem mutatója, aktuális elem mutatója



A **fa** egy hierarchikus adatszerkezet, mely véges számú csomópontból áll és igaz, hogy

- bármely két csúcs között a kapcsolat egyirányú
- van egy kitüntetett csomópont, ami nem végpont: fa gyökere
- az összes többi csomópont pontosan egyszer végpont

A fa rekurzív definíciója:

- a fa üres vagy
- van egy kitüntett csúcs, ami a gyökér
- melyhez 0 vagy több diszjukt fa kapcsolódik

Maximum hány csomópont helyezhető el egy f fokú, m szintet tartalmazó fában?

$$1 + f + f^2 + f^3 + \cdots$$

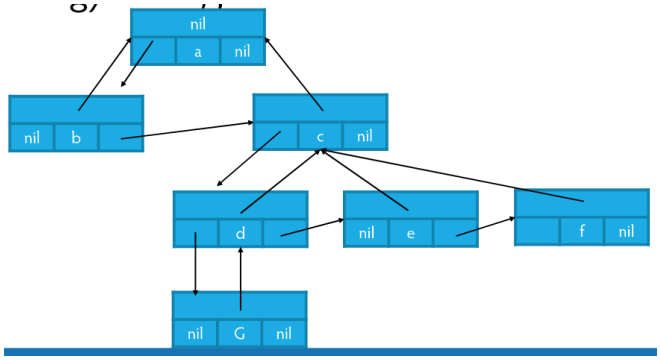
$$\frac{(f^{m+1} - 1)}{(f - 1)}$$

Műveletek: üres-e, gyökérelem, keres(e), üres, beszúr(e), módosítgyökér(e), töröl(e), törölfa

Preorder (gyökér, bal, jobb), postorder(bal, jobb, gyökér), inorder (bal, gyökér, jobb)

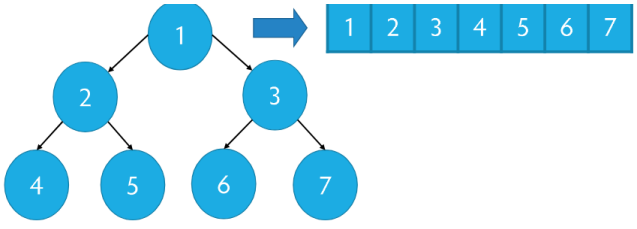
Bináris fa az olyan fa, melynek csúcsaiból max 2 részfa nyílik

Reprezentáció: gyerekek szülő: pointerek

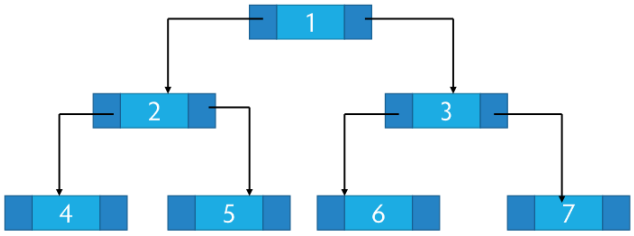


használható multilista: minden csomópontban egy lista, melyben első elem az adat, majd a gyermekek (egyéb kapcsolatok)

Szintfolytonosan tömbbe raknir



Klasszikus: gyerekrkre pointerek



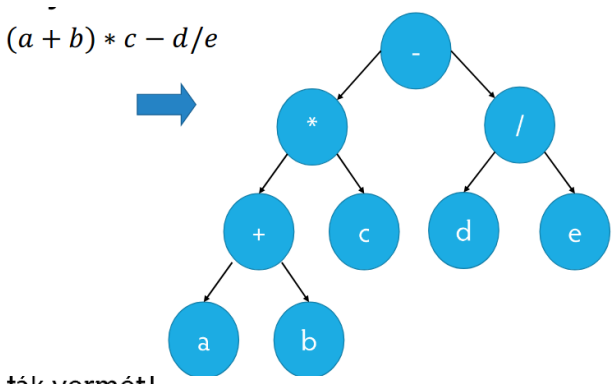
Egy bináris fa t**ökéletesen kiegyensúlyozott**, ha minden elem bal illetve jobb oldali részfájában az elemek száma legfeljebb eggyel tér el

Teljes, ha minden elemenke pontosan két elágazása van (kivéve levél lol)

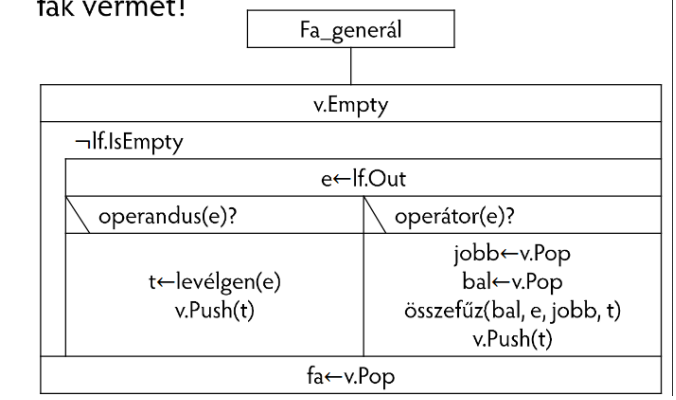
Majdnem teljes: ha csak a level szintjén necó

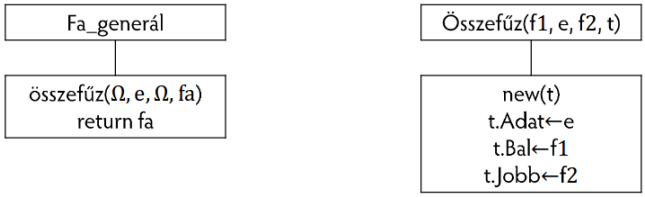
Kiszámítási és kifejezésfának nevezzük azt a struktúra, amely egy nyyelv szimbolúmai és műveletei közötti precedenciát jeleníti meg.

pl



ták vermét!





Előadás 5

Keresőfának nevezzük az olyan bináris fa adatszerkezetet, amelynek kialakítása a különböző adatelemek között meglévő rendezési relációt követi

A fa felépítése olyan, hogy minden csúcsra igaz az, hogy

- a csúcs értéke nagyobb, mint tetszőleges csúcsé a tőle balra lévő leszálló ágon
- a csúcs értéke kisebb minden a tőle jobbra lévő leszálló ágon található csúcs értékénél.

Az ilyen fa az őt tartalamazó elemek veciteli sorrendjét is visszatükrözi

Bináris kerső fa bejárása O(n) ideig tart.

Kereső algoritmus:

```
Fában-keres(x, k)
   if x = NIL or k = kulcs[x]
     then return x
   if k < kulcs[x]
     then return Fában-keres(bal[x], k)
     else return Fában-keres(jobb[x], k)
```

```
Fában-iteratívan-keres(x, k)
   while x ≠ NIL and k  ≠ kulcs[x] do
     if k < kulcs[x]
       then x ← bal[x]
       else x ← jobb[x]
   return x
```

```
Fában-minimum (T)
   x ← gyökér[T]
   while bal[x] ≠ NIL
     do  x ← bal[x]
   return x
```

```
Fában-maximum (T)
   x ← gyökér[T]
   while jobb[x] ≠ NIL
     do  x ← jobb[x]
   return x
```

Futási idő h magasságú fában o(h)

```
Fában-következő(T, x)
if jobb[x] ≠ NIL
   then return Fában-minimum (jobb[x])
y ← szülő[x]
while y ≠ NIL és x = jobb[y] do
  x ← y
  y ← szülő[x]
return y
```

A dinamikus halmazokra vonatkozóKeres, minimum, maximum, következő és előző mmáveletek h magasságú bináris keresőfában o(h) idő alatt végezhetők el.

```
Fába-beszúr (T,p)
y ←NIL; x ← gyökér[T]
while x  ≠ NIL
  do y ← x
    if kulcs[p] < kulcs[x]
      then x ← bal[x]
      else x ← jobb[x]
szülő[p] ← y
if y = NIL
  then gyökér[T] ← p
  else if kulcs[p] < kulcs[y]
    then bal[y] ←p
    else jobb[y] ←p
```

```
if bal[p] = NIL vagy jobb[p] = NIL
  then y ←p
  else y ← Fában-következő(T, p)
if bal[y]  ≠ NIL
  then x ← bal[y]
  else x ← jobb[y]
if x  ≠ NIL
  then szülő[x] ← szülő[y]

if szülő[y] =NIL
  then gyökér[T ] ←x
  else  if  y = bal[szülő[y]]
    then bal[szülő[y]] ← x
    else  jobb[szülő[y]] ← x
if  y  ≠ p
  then kulcs[p] ← kulcs[y]
return y
```

Keresések

Léteznek ún. **szekvenciális keresések**, melyben a keresési idő n-nel arányos O(n), ezek rendezetlen tömbök vagy láncolt listák

Illetve vannak **bináris keresések**, pl rendezett tömbök, melykben a keresési idő log2n-nel arányos.

A **szótár** olyan adatszerkezet, melyen értelmezve van a: keres, beszúr, töröl mávelet.

A **prioritásos sor** olyan szótár, melyen értelmezve van: minimum, maximum, előző, rá következő. Ha n elemből építünk egy bináris keresőfát és minden sorrend ugyanolyan valószíná, akkor ő egy **véletlen építésű bináris keresőfa** lesz. Ennek az átlagos magassága O(log2n)

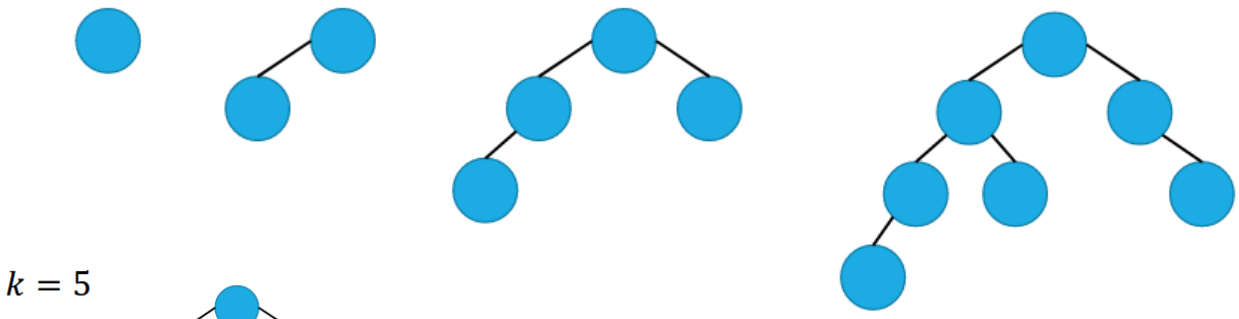
Előadás 6

Egy bináris kersőfa **AVL fa**, ha minden csúcsára teljesül, hogy |m(bal[x])-m(jobb[x])|<=1.

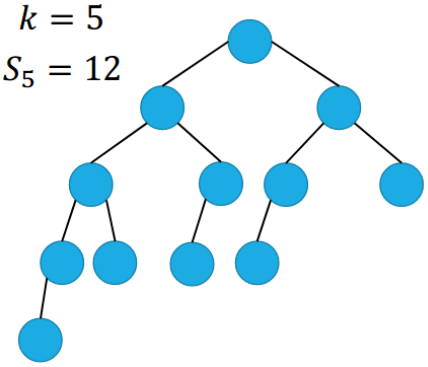
m(x) az x göykerű részfa magassága

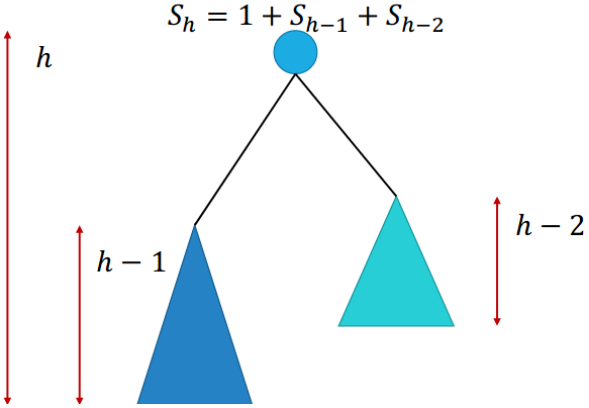
- Mekkora a k-szintű AVL-fa minimális csúcsszáma?

$k = 1$ $S_1 = 1$ $k = 2$ $S_2 = 2$ $k = 3$ $S_3 = 4$ $k = 4$ $S_4 = 7$



$k = 5$
$S_5 = 12$





Egy h magaságú AVL fának F_h+3 -1 csúcsa van.

Ha F avl fa, akkor h(F) <= 1.44 * log_2(n+1), ahol n az F fa pontjainak számát jelöli.

- -1 : bal részfa magasabb 1-gyel
- 0 : egyforma magasak a részfák
- +1: jobb részfa magasabb 1-gyel

EAB