ADATSZERKEZETEK ÉS ALGORITMUSOK

LIFO, FIFO

„LIFO, FIFO, Szekvenciális adatszerkezetek”

Verem, Stack, LIFO

Verem fogalma

- LIFO: last-in, first-out
- Köznapi fogalma

Mókus	Borz	Nyuszi	Sün

Sorrend az elemek betétele előtt:

Mókus	Borz	Nyuszi	Sün

Sorrend miután az összes elem kikerült:

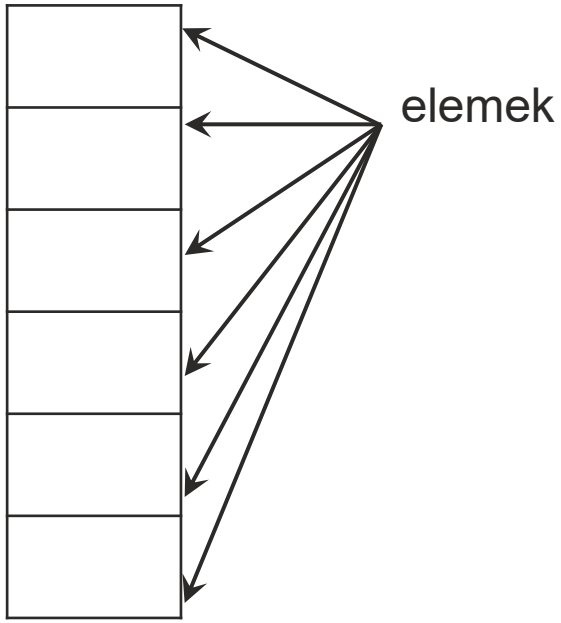
A verem műveletei

- Műveletek
  - empty
  - isempty
  - push
  - pop
  - top
- Fontos:
  - pop és top művelet nem hajtható végre üres vermen

- Műveletek jelentése
  - Üres verem létrehozása
  - Üres a verem?
  - Elem betétele a verembe
  - Elem kivétele a veremből
  - Felső elem lekérdezése
- Figyelem!
  - A műveletek között nem szerepel „isfull” művelet!

Verem szerkezete

- Lineáris adatszerkezet

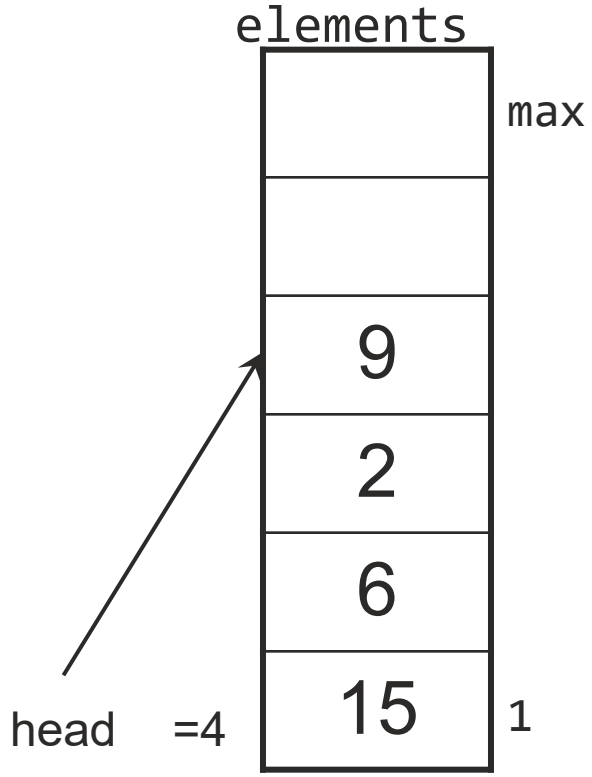


Elemek száma

- Felépített adatszerkezet
  - Az adatszerkezet elemeinek száma a feldolgozás során rögzített vagy változtatható
  - A rögzített nem jelenti, hogy a tárolt adatok nem megváltoztathatók
- Adatelemek száma
  - Fix
    - A tárolható adatelemek számának felső korlátja a létrehozáskor (esetleg fordítási időben) rögzített.
  - Változó
    - A memória mérete (illetve kapcsolódó technikai korlátok) szab határt az adatelemek számának

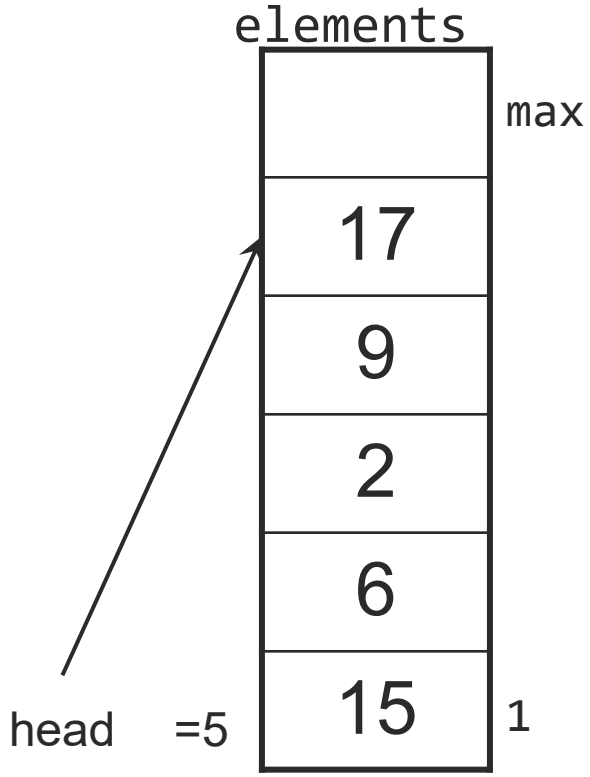
Reprezentáció

- Aritmetikai ábrázolás:
  - egy max hosszú vektor (ez az elemek tömbje) elements[1..max]
  - a verem tetejének mutatója head $\in$ [0, max] head=0 $\Leftrightarrow$ üres a verem
  - Választási lehetőség, hogy hova mutat a head
    - Az első szabad helyre
    - Az utolsó elfoglalt helyre



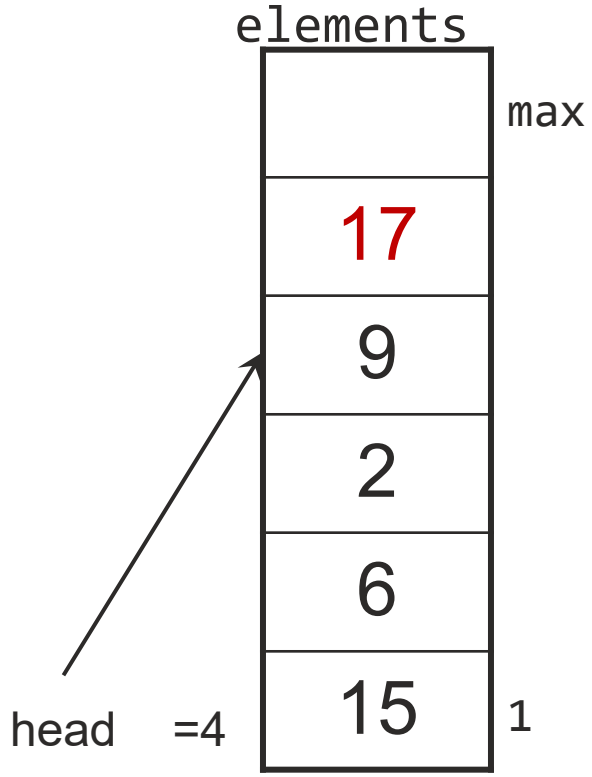
Reprezentáció

v.push(17) után



Egyenlő a két verem?

v.pop() után



Implementáció

- Műveletek pszeudokódja/struktogramja:
  - `v.empty`

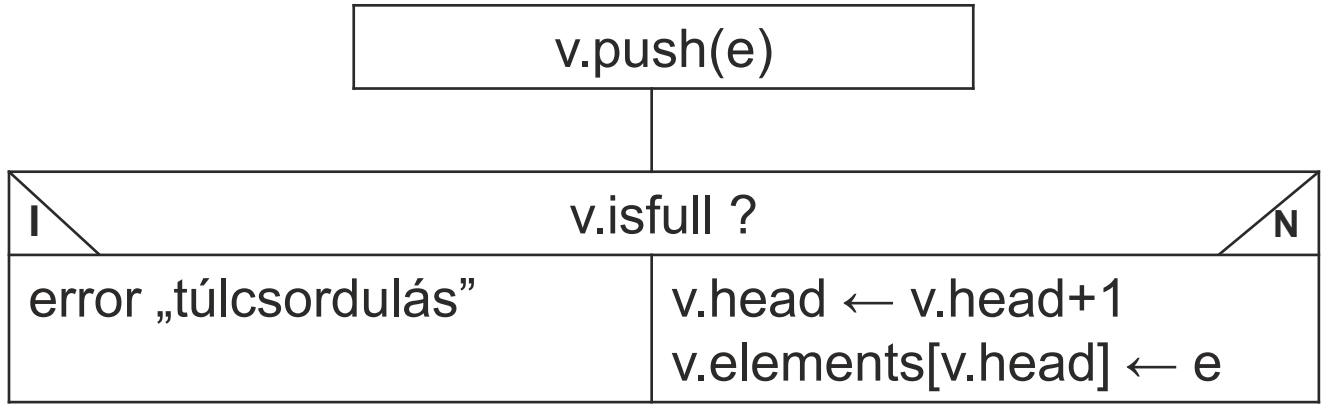
    ```
    -- üresre állítja a vermet
    v.head ← 0
    ```
  - `v.isempty`

    ```
    -- Üres a verem? – Logikai értéket ad vissza
    return (v.head=0)
    ```
  - `v.isfull`

    ```
    -- Tele van a verem? – Logikai értéket ad vissza
    return (v.head=max)
    ```

Implementáció

- v.push(e)
```
-- e-t beteszi a v verem tetejére
if v.isfull
  then error "túlcsordulás"
  else v.head ← v.head +1
  v.elements[v.head] ← e
end if
```



Implementáció

- v.pop
  -- kiveszi a legfelső elemet és visszaadja
  if v.isempty
    then error "alulcsordulás"
    else v.head ←v.head -1
      return v.elements[v.head+1]
  end if

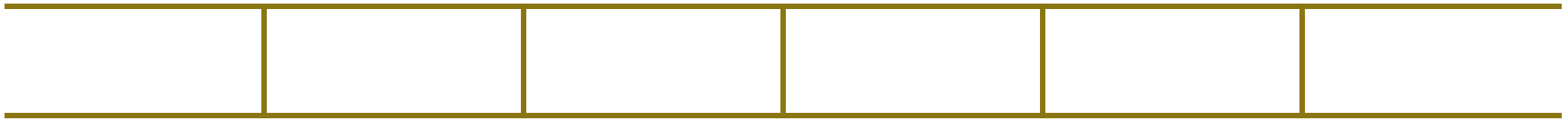
Implementáció

- ```
  v.top
  -- lekérdezi a legfelső elemet
  if v.isempty
    then error "alulcsordulás"
    else return v.elements[v.head]
  end if
  ```
  ```

Sor, Queue, FIFO

Sor (Queue)

- FIFO: First-in, First-out
- Köznapi fogalma



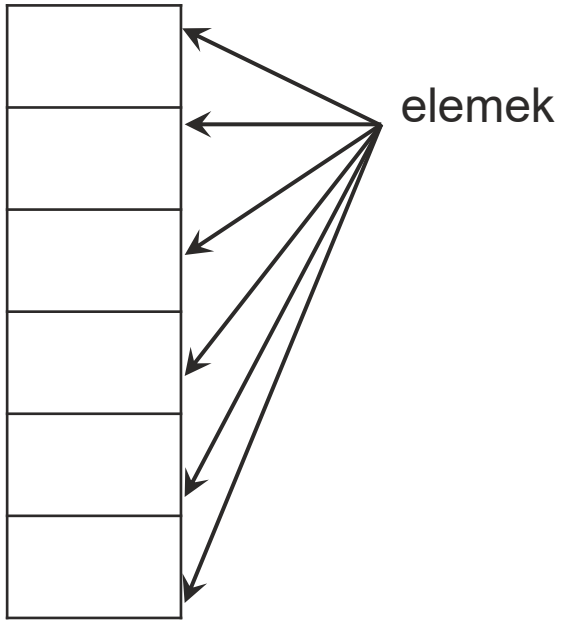
A sor műveletei

- Műveletek
  - empty
  - isempty
  - in
  - out
  - first
- Fontos:
  - out és first nem működnek üres sor esetén

- Műveletek jelentése
  - Üres sor létrehozása
  - Üres a sor?
  - Elem betétele a sorba
  - Elem kivétele a sorból
  - Első elem lekérdezése
- Figyelem!
  - A műveletek között nem szerepel „isfull” művelet!

Sor szerkezete

- Lineáris adatszerkezet

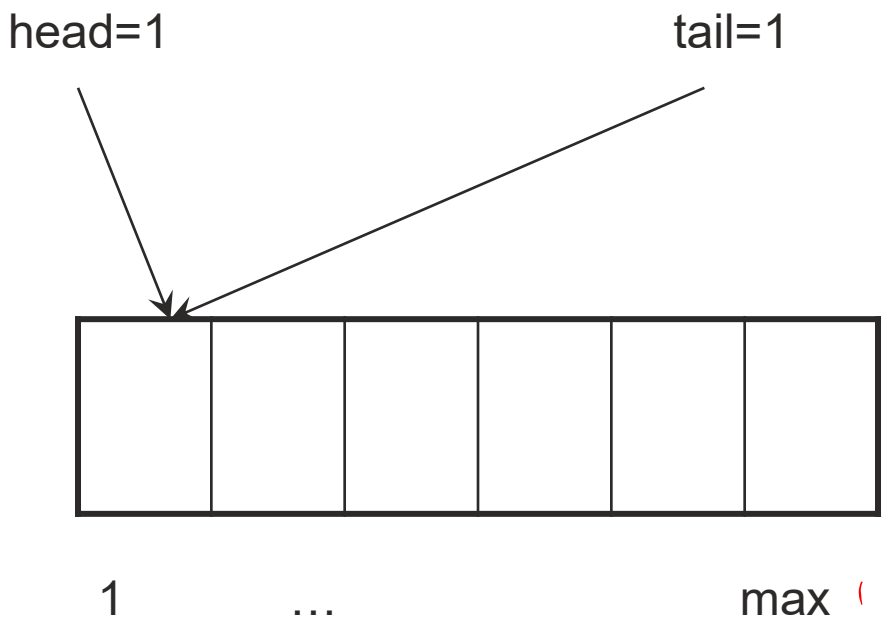


Reprezentáció

- Aritmetikai ábrázolás:
  - egy $\max$ hosszú vektor
    - ez az elemek tömbje
    - $\text{elements}[1..\max]$
  - és a sor első elemének mutatója
    - $\text{head} \in [1, \max]$
  - és a sor első üres (utolsó) helyének mutatója
    - $\text{tail} \in [1, \max]$
- Vegyük észre, hogy az aritmetikai ábrázolás három részből áll!

Reprezentáció

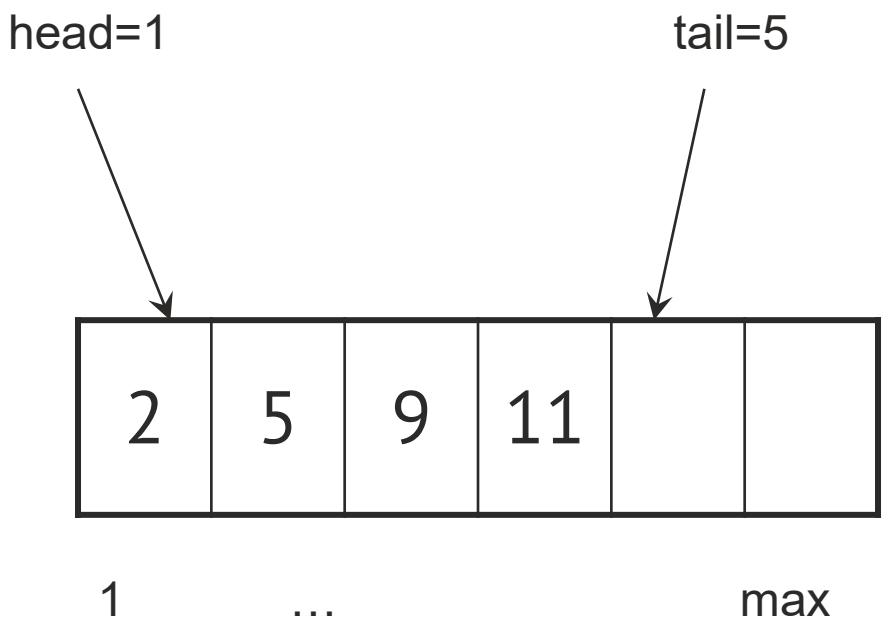
- Ciklikus ábrázolással
  - Kezdetben



Üres a sor

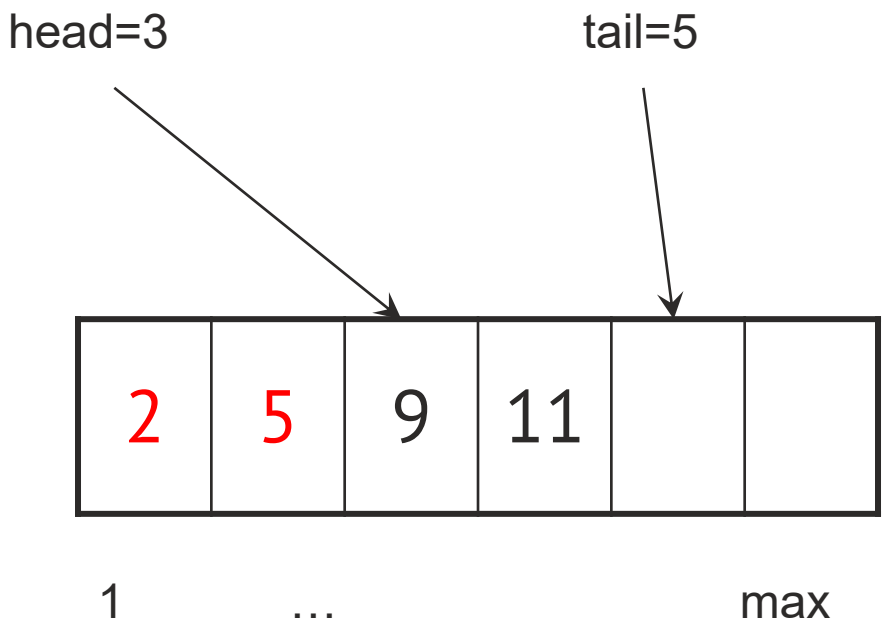
Reprezentáció

- 2, 5, 9, 11 betétele után:



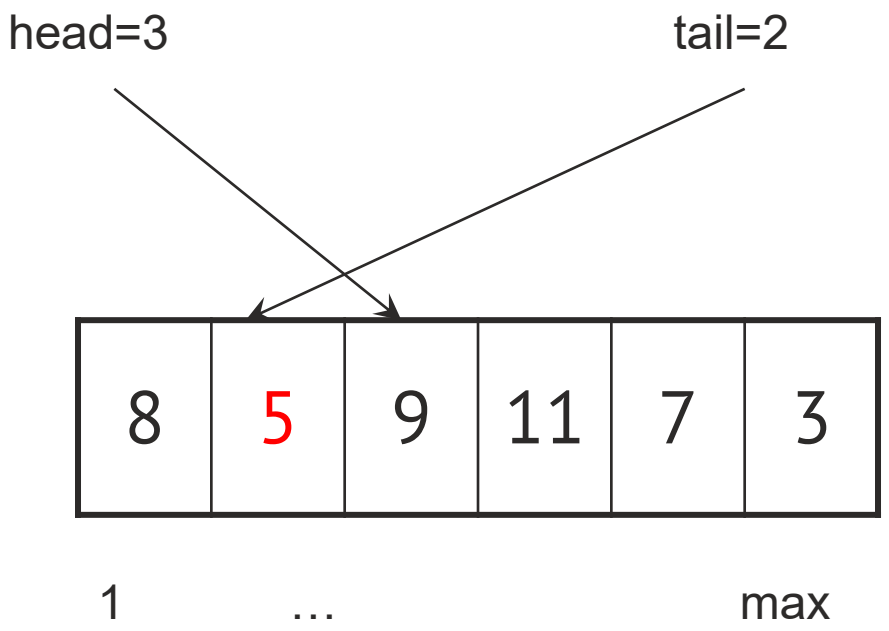
Reprezentáció

- 2, 5 kivétele után:



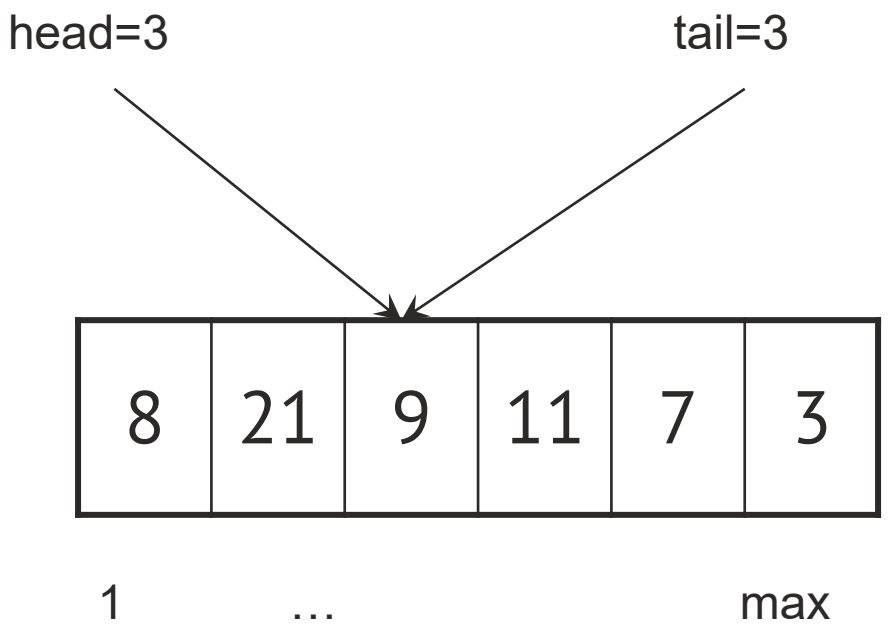
Reprezentáció

- 7, 3, 8 betétele után:



Reprezentáció

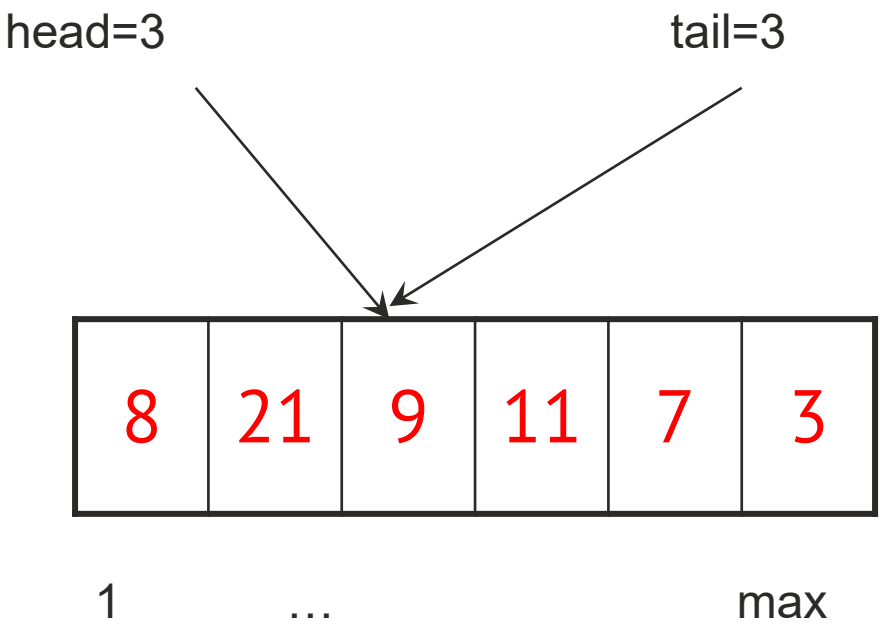
- 21 betétele után:



Tele a sor

Reprezentáció

- 9, 11, 7, 3, 8, 21 kivétele után:



Üres a sor

Reprezentáció

- Mit tegyünk? – Milyen lehetőségek vannak:
  - Vezessünk be még egy jelzőt a reprezentációba, ami mutatja, hogy a sor üres-e
    - `empt`
    - kezdetben igaz, később vizsgáljuk, és megfelelően állítjuk
  - Vezessünk be még egy attribútumot a reprezentációba, ami mutatja, hogy hány elem van a sorban
    - `count`

Implementáció

- Műveletek pszeudokódja:
  - s.empty
    -- üresre állítja a sort
    s.head ← 1; s.tail ← 1; s.empt ←true
  - s.isempty
    -- üres a sor? - logikai értéket ad vissza
    return s.empt
  - s.isfull
    -- tele van a sor?
    return ((not s.empt) and (s.head = s.tail))

Implementáció

- s.In(e)

```
-- e-t beteszi az s sor végére
-- s.tail-t ciklikusan növeli
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
```

Implementáció

- s.Out

```
-- kiveszi és visszaadja az s sor első elemét
-- s.head-et ciklikusan növeli
-- figyeli, hogy nem üres-e a sor
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
```

Implementáció

- ```
  s.First
  -- visszaadja az s sor első elemét,
  -- figyeli, hogy nem üres-e a sor
  if s.empt
    then error "alulcsordulás"
    else return s. elements[s.head]
  end if
  ```

- Lehetne az is, hogy a darabszámot tároljuk
  - Házi feladat: átgondolni
  ```

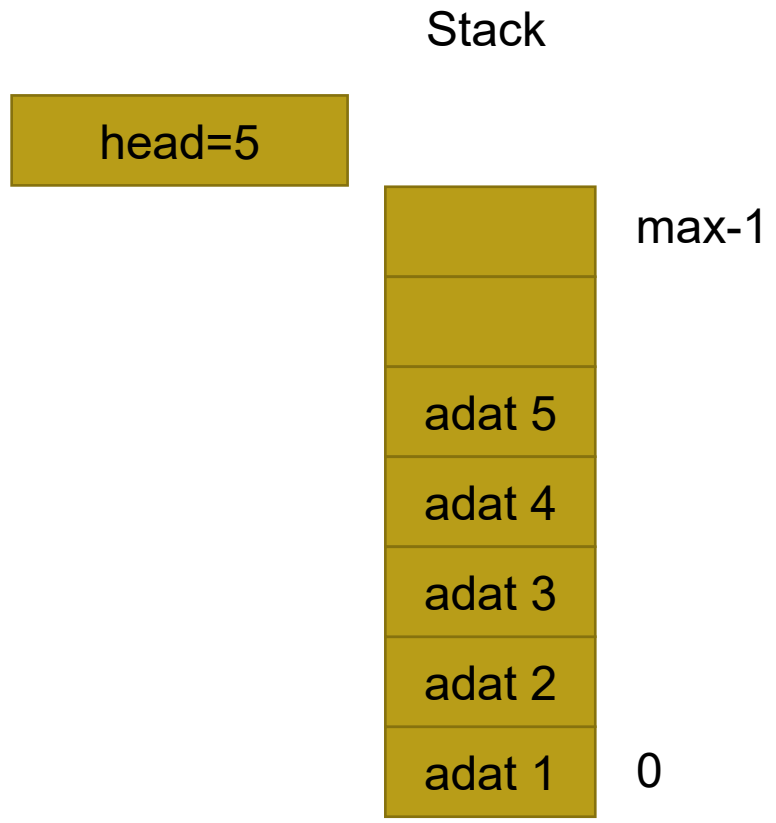
Implementáció

- Vigyázni kell, amikor a programokat a választott programnyelven megvalósítjuk!
  - Például
    - C++ nyelv esetén a vektorok indexelése nullával kezdődik!
  - Értékadás jele
  - Egyenlőség vizsgálat jele

Verem – fix méretű megvalósítás

- A verem egy max hosszú tömb és a head (int) direkt szorzata
  - A tömb elemei [0…max-1] között indexeltek)
- A head az első szabad pozíciót jelzi a tömbben, ahova beszúrhatunk értéket

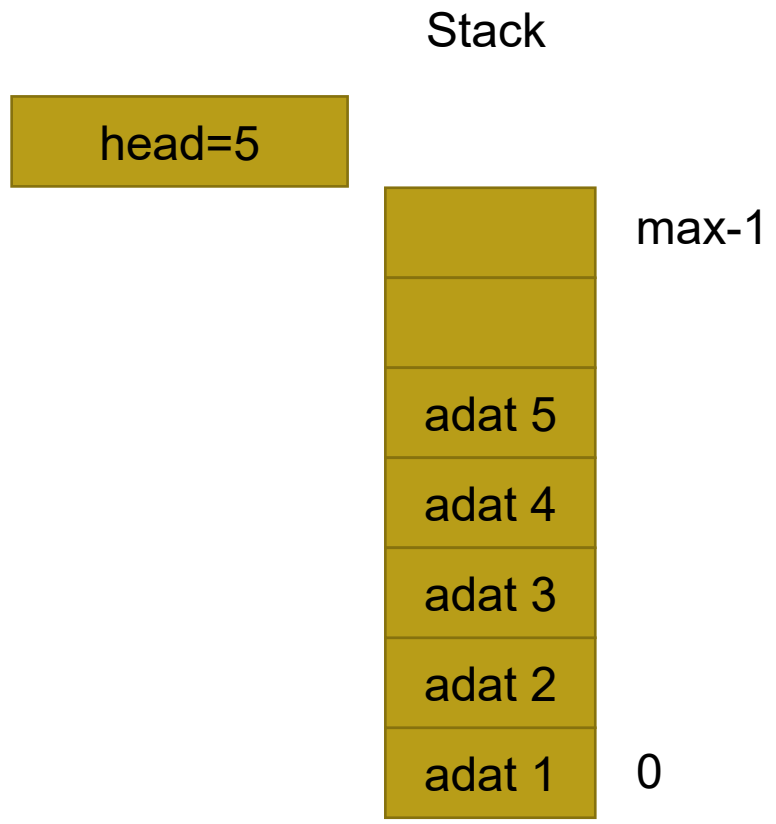
$$0 \leq head \leq max$$



Verem – fix méretű megvalósítás

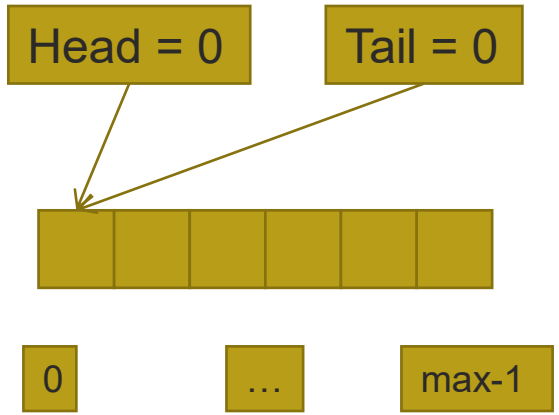
- Megvalósítás osztály segítségével:

```
class Stack{
 static const int max = 7;
 private:
   int tomb[max];
   int head;
 public:
   Stack();
   ~Stack();
   void push(int new_item);
   int pop();
   int top() const;
   bool isEmpty() const;
};
```



Sor – fix méretű megvalósítás

- A sor elemeit egy statikusan létrehozott `max` méretű tömbbel, a `head` és `tail` mutatókkal, `empty` paraméterrel reprezentáljuk.
  - elemei: `array[0...max-1]`
- A veremmel ellentétben a sornál a tömbnek mind a két végére szükségünk van, ezért azok helyét két változó, a `head`, és a `tail` fogják megadni.
- A megvalósításhoz ciklikus ábrázolást használunk



Sor – fix méretű megvalósítás

- Kezdetben a `head` és a `tail` a tömb ugyanazon elemének indexei.
  - `head`: a tömb első elemének indexe
  - `tail`: a tömb első szabad helyének indexe
- Mikor üres a sor?
  - Ha a `head` és a `tail` ugyanoda mutat, akkor a sor vagy üres, vagy tele van.
  - Ha üres, akkor az utolsó művelet szükségszerűen kivétel volt.
    - Ha az utolsó művelet betétel volt, akkor a sor most tele van.
  - Tartsunk karban egy változót, amellyel ezt követni tudjuk!

Sor – fix méretű megvalósítás

```
class FixedQueue {
 public:
   FixedQueue();
   ~FixedQueue();
   void in(int new_item);
   int out();
   int first() const;
   bool isEmpty() const;
   bool isFull() const;
 private:
   static const int CAPACITY = 10;
   int array[CAPACITY];
   int head, tail;
   bool empty;
};
```