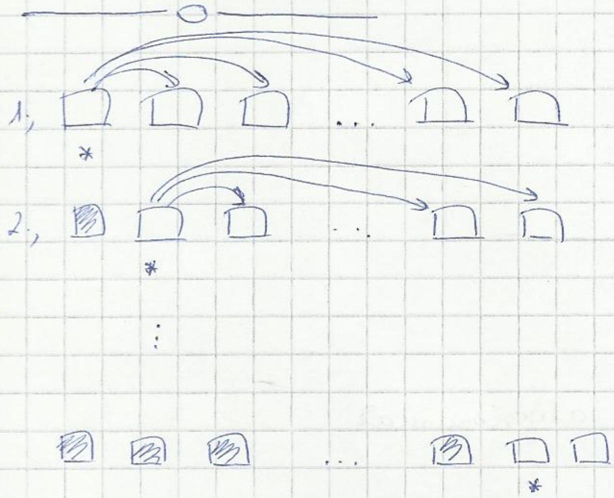
- ha eleve rendezett a sorozat, nincs szükség értékadásra
- ha fordítottan rendezett, max $n-1$ cserére lehet szükség,



Addig folytatjuk, amíg kételemű sorozatot kapunk.

Tárigénye:

Ennek a rendezésnek is $n-1$ a tárigénye

Összehasonlítások száma: $\frac{n(n-1)}{2}$

Minden összehasonlítást követ csere. Az értékadások száma max 3x akkora

Az előzőnél n csere volt, ennél $\frac{n(n-1)}{2}$.

Algoritmus:

```
F := 1
Ciklus J := F+1 .. N
  Ha A[J] < A[F] akkor
    C := A[J]
    A[J] := A[F]
    A[F] := C
  H vége
C vége
```

Ciklus I := 1 .. N-1

Ciklus J := I +1 .. N

Ha A[J] < A[I] akkor

C := A[J]

A[J] := A[I]

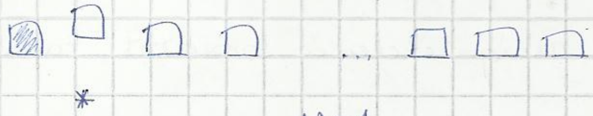
A[I] := C

H vége

C vége

C vége

Alulmarad az elöző algoritmusnak

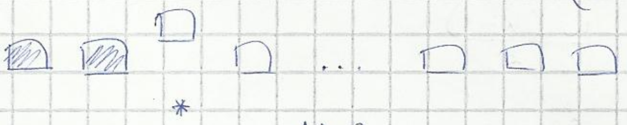
1, 

N-1

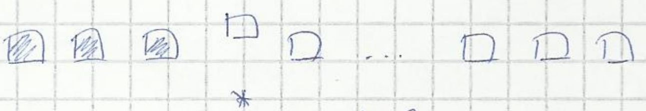
Ez rendezett.

El tudjuk dönteni, hogy hová illik a kártya. → KÁRTYÁS REND.

(Beszúró rendezés)

2, 

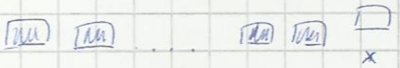
N-2

3, 

N-3

- Az eleve növekvő sorrendbe rendezett a sorozat, nem kell a többiekkel összehasonlítani.
- Itt az összehasonlítások száma függ a sorozat elő-rendezettségétől.
- A tárigény itt is n+1.

– Ha eleve rendezett a sorozat:



Az összes rendezések száma $n-1$.

Ha rendezett a sorozat, beteszem, meg kiveszem (az elemet.) Oda illik.

Ha nem rendezett a sorozat, akkor $n-1$ elemet kell előre csúsztatni, és $n-1$ értékadás.

El kell végezni $\frac{n(n-1)}{2} + 2(n-1)$

ha rendezett: $2(n-1)$ → 1-el feljebb csúsztatjuk

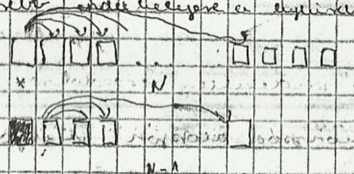
Algoritmus:

```
J := 1            J: rendezett részsorozat utolsó elemének indexe

I := J; X := A[I+1].  → az esetleges elemet ide lehessen csúsztatni
Ciklus míg (I ≥ 1) és (A[I] > X)
    A[I+1] := A[I];
    I := I-1;
C vége
A[I+1] := X
```

```
Ciklus J := 1 .. N-1;
    I := J; X := A[I+1]
    Ciklus amíg (I ≥ 1) és (A[I] > X)
        A[I+1] := A[I];
        I := I-1;
    C vége
    A[I+1] := X
C vége
```

KÖZVETLEN KIVÁLASZTÁSOKKAL TÖRTÉNŐ RENDEZÉS

2, kijelöljük minden helyre a legkisebb elemet



Hatékonysága: - műveletigény N elemű vektorra
- [illegible] is kell elhelyezni
- Tárigénye így itt is N+1
- itt is szükséges helycserék
- itt nem függ az összehasonlítás sorrendtől

Itt is az összehasonlítások száma $\frac{N(N-1)}{2}$

Értékadások száma $0 - 3 \cdot \frac{N(N-1)}{2}$ így változtat

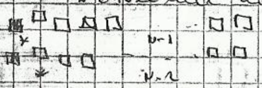
```
CIKLUS J:=1..N-1;
   J:=1;
  CIKLUS I:=J+1..N
     HA A[I] < A[J] AKKOR
           C:=A[I];
           A[I]:=A[J];
           A[J]:=C
     HVÉGE;
  CVÉGE;
CVÉGE;
```

ezért kell hogy [illegible] és [illegible]

és nem volna [illegible]

BESZÚRÓ RENDEZÉS (KÁRTYA RENDEZÉS)

3) az az elem [illegible] van és [illegible] [illegible] [illegible] felvennénk egyet, [illegible] az első rendezett majd a [illegible] is [illegible] [illegible] elé a [illegible]. Ha ez megtörtént akkor [illegible] [illegible] elem, [illegible] és az elem helyét egy [illegible] sorozatban kell megtalálni. [illegible] hogy rendezett [illegible] [illegible] [illegible] még kell mozgatni akkor [illegible] el hogy a elemek nem a végleges helyére kerülnek,
• össze kell hasonlítani a rendezettekkel is.



csak akkor kell változtatni, ha rendezett elemeknél [illegible].

- Az összehasonlítások száma függ a rendezettségtől
- összehasonlítások száma legalább $N+1$.
  ha [illegible] az sorozat össz száma min. $N-1$, maximum $\frac{N(N-1)}{2}$ [illegible]

- Értékadás: [illegible] [illegible] elemnél, [illegible] [illegible] [illegible] van és ez $n-1$ sorban míg min $2(n-1)$
  Ha valamennyi elemnél [illegible] akkor csak [illegible] [illegible] nem kell [illegible]

és [illegible] is el [illegible] [illegible] a [illegible] és a [illegible] [illegible].

Min: $2(N-1)$ ... Max: $\underbrace{\frac{N(N-1)}{2} + 2(N-1)}_{Max}$

Algoritmus

```
CIKLUS J:=1..N-1
    J:=1;
    I:=J; x:=A[I+1]
    CIKLUS MÍG (I≥1) ÉS (A[I]>x).
          A[I+1]:=A[I];
           I:=I+1;
    CVÉGE;
    A[I+1]:=x;
CVÉGE
```

[illegible] [illegible], [illegible] [illegible]

[illegible] pozíció: [illegible]
a első elemet is [illegible] [illegible].

[illegible] hogy [illegible] [illegible] az x-et

és a [illegible] [illegible] [illegible] [illegible] [illegible] nem [illegible] [illegible]

2003. márc. 19.

7. óra

SHELL RENDEZÉS

Legyen [illegible] [illegible] az n értéke $D=5$ [illegible] [illegible] jel
D régiségtől egymástól azonos távolságra:

#	*	@	+	#	A	@	@+	=
1.	2.	3	4.	5	6.	7	8.	

1 # # #
2 * * *

Először próbáljuk rendezni a nem sorozatokat [illegible] külön - külön [illegible] egymástól [illegible] [illegible] [illegible] összehasonlítunk így az elemek gyorsan közelednek a végleges helyükhöz. Az D értékét mindig csökkenteni kell [illegible] [illegible] [illegible] elemek [illegible] [illegible] értéke 1 kell, hogy legyen.

[illegible] sorozat [illegible]
1. rendezés [illegible] kell [illegible] D-ször de [illegible] még [illegible]
[illegible] [illegible]
minden [illegible] sorozatot [illegible] [illegible]
[illegible]
D=k..1

Ha ez a [illegible] még van, akkor már nem kell megmondanunk, hogy milyen rendezési [illegible] van [illegible].
Nem 1 [illegible] [illegible] valamely rendezési [illegible] van [illegible] hogy a sorozatokat még [illegible] sorozatokra is bontjuk!!!

ELJÁRÁS SHELL-Min

```
CIKLUS míg j:=1..n-1
    mini:=j;
    CIKLUS míg i:=j+1..n
        HA a[i] < a[mini] AKKOR mini:=i;
    CVÉGE;
    HA j<>mini AKKOR
        c:=a[j]; a[j]:=a[mini]; a[mini]:=c;
    HVÉGE
CVÉGE
```

Eljárás shell_Min

```
konstans   N=sorozat elemszáma;
változó d, k, i, j : Egész;
            c : elemtípus;
d:=n;
CIKLUS
    d:=d div 3+1
    CIKLUS k:=1..d
        j:=k
        CIKLUS míg j<n-d
            mini:=j
            i:=j+d
            CIKLUS míg i<=n
                Ha a[i] < a[mini] akkor mini:=i
                i:=i+d
            CVÉGE
            Ha j<>mini akkor
                c:=a[j]
                a[j]:=a[mini];
                a[mini]:=c;
            HVÉGE;
            j:=j+d;                -> rendez
        CVÉGE
    CVÉGE
    ~~CVÉGE~~;
CVÉGE d=1 ESETÉN
EVÉGE.
```

Ez már adva kell

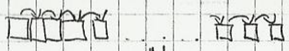
$\forall\, i,j \in N$ $\quad i<j$ $\quad a_i \le a_j$ $\quad j:=i+1$

akkor a sorozat elemei növekvő sorrendben vannak

BUBORÉK RENDEZÉS

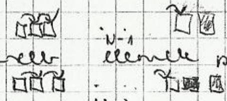
KÜLSŐ ciklus, hogy hányadik sorozatnál járunk

- lényege: szomszédos elemeket hasonlítunk össze.
- N elem esetén N-1 összehasonlítás van



N

- a sorozat felbontható rendezett és rendezetlen részre. A sorozat utolsó eleme a későbbiekben már nem lesz dolgunk.



N-1

- a kisebb elemek pedig már a sorozat eleje felé vándorolnak

N-2

A kisebb lassabban, a nagyobbak gyorsabban mozdulnak a helyükre, a rendezett rész már így is gyorsabban nő. Addig kell folytatni míg két elem marad, amelyeket csak meg kell cserélni

S.SZ	E.SZ	Ö.SZ
1	N	N-1
2	N-1	N-2
3	N-2	N-3
⋮	⋮	
N-2	3	2
N-1	2	1

$\frac{N(N-1)}{2}$

Minden egyes összehasonlítást cserélni kell, n. esetben.
Öt szóma $3 \cdot \frac{N(N-1)}{2}$, amennyiben a sorozat rendezett egyetlen egy

összehasonlítás mindig
Tárigény: N+1 tárigénye.
Közvetlen kiválasztásos rendezéssel, egyszerű meg a bonyolultsága
Így az sem valami jó, de bonyolultabbá teszi.

```
CIKLUS J:=1..N-1;      -> csak meghatározni minden rendezett sorozatnál.
  J:=1;
  CIKLUS I:=I..N-J
    HA A[I]>A[I+1] AKKOR      nagyobbat kell, akkor csere
        C:=A[I]
        A[I]:=A[I+1]
        A[I+1]:=C
    HVÉGE;
  CVÉGE;
CVÉGE;
```

rendezetlen az elemek mozgása, mint más keresésnél

Meg kell nézni, hogy a rendezetlen sorozat is rendezett volt, mellett n-1 sorozat elindulás
így sok összehasonlítást meg lehet spórolni

$i<j$

$a_i < a_j$

$a_i \le a_{i+1}$

Rendezetlen rész, egyszerre ... rendezett

- Bizonyos szinte sorozatok kijelölése dirigálhatóak.
- Az kell megnézni, hogy rendezett-e, xxxx, hol történt utoljára csere
- Egy bizonyos szinte.

2003. márc. 20. 8. óra

Buborék rendezés, ne csökkentsük, hogy a sáv, hol végig a cseréknek ha nem második is.

```
J:= 1;              KÉSZ:=HAMIS;
CIKLUS MIG (J ≤ N-1) ∧ (NEM KÉSZ)
   KÉSZ:=IGAZ;
   CIKLUS I:=1.. N-J
      HA A[I] > A[I+1] AKKOR
         C:= A[I]; A[I]:= A[I+1];
         A[I+1]:=C;
         KÉSZ:=HAMIS;
      HVÉGE;
   CVÉGE;
   J:=J+1;
CVÉGE;
```

HO

és az igaz; ha feltétel nem, újra végig kell haladni.

→ Hamis, akkor a kis igaz marad és nem fogja a sorozat következő elemeit kezelni.

Van-e már olyan pozíció amely már rendezett, és már csak a maradék rendezetlen kell már, csak kijelölni, ugyanis hogy a hátsó elemek nem nagyobbak, ha meg jegyeznénk hol történt utoljára a csere, akkor már ide kellene tolni a rendezést.

Utolsó csere helye:

holvan az utolsó eleme a részsorozatnak

```
J:= N;
CIKLUS MIG J ≥ 2
   CSH:= Ø;
   CIKLUS I:=1 ... J-1
      HA A[I] > A[I+1] AKKOR
         CSERE
         CH:= I;
      HVÉGE;
   CVÉGE;
   J:= CSH;
CVÉGE;
```

→ mert a J már csere

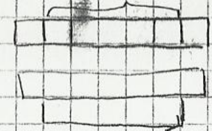
ha ez nem volt igaz, akkor az elemek rendezetlen vannak

hol történt a csere ott jegyezzük

Ha volt egy nagyobb akkor azt is átírja.

Ez a megoldás nagyobb jogalap az előzőnél is, nem jobb mert nem csak azt ismeri őrre a rendezett vagy nem, a hanem megjegyzi menteni a rendezett sorozat helyét is és ab hogy ugyan.

nagyobb elemet megkeresni a legnagyobb indexet, egyidejűleg a legkisebb is a maradékon ugyan ezt kell tenni.



Ekkor kell alkalmazni, a max, min, ~~amit~~ ugyanis

- a külső átírás fele annyiszor kell, mert kétszer az elemek fogynak.

Eljárások és függvények

Így meghívni, ami csinál az adatokat amellyel dolgoznia kell

nevével megvívhatók, adatokat paraméter átadással adunk át.

~~FÜ~~

FÜGGVÉNY függvénynév (paraméterlista): típus, és új ugyanis kimenő paraméter.
   deklarációs rész
   utasítások
FVÉGE

paraméter átadás - címszerinti ⇒
- értékszerinti átadás

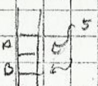
utolsóval felfogadó címszerint

ELJÁRÁS ADD (A,B: EGÉSZ) ⇒ formális paraméterek memória, akkor már lefoglalja a
   KI: A+B
ÉVÉGE

Belső

ADD (6,3); ⇒ aktuális paraméterek ez már ide is írja.
KI: X)

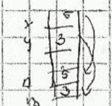
címszerinti par. átadás



Kellő okos memória értékek és elfogadhatók.

Csak változók lehetnek a aktuális paraméterek x:=5 y:=3
Add (x, y);

Érték szerint:
csakugyan a nem kell memória első helyet, első változót második második?



szintén minden egyéb A, B memória területen felmásolva,

x, változót nem kell megváltoztatni.

Jó arra hogy az algoritmus bemenetét megmutassa

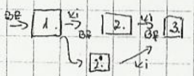
Fölöslegesen lehet szaporítani a könyvet, ha érték szerinti átadás van mert akkor két helyen foglalja le a memóriát.
Vagy magam a vektor adjuk át cím szerint és így nem foglalunk le több memóriát
• Rendezem a sorozatot Rend(S)

Később a fv lokális változóinak foglal helyet!
az adott alprogramban deklarált változók
A változók érvényessége az hatóköre! egy adott eljárásban deklarált változó, csak ebben az eljárásban használható és ebben az eljárásban amit ezen belül írunk.

A lokális, számára nem kerül lefoglalásra, akkor törlődik meg amikor az eljárást aktiválták.

2003. április 2.

9. óra



Tudnunk kell körül határolni ezeket a részeket míg határozottan is kell keresni.
Bemenete rendezetlen kimenete rendezett.

Szekvenciális eljárás: - len feldolgozó
- feldolgozó
- kijelző rész.

fv v. eljárás jelentkezik meg nómenklatúra, hogyan tudom megvalósítani a bemenetet és kimenetet?

- eljárás, fv, deklaráláskor formális paramétereket kell megadni, jelezni kell a paraméter átadás formáját is.
Alprogramnál amikor közben pedig kell, hogy a címszerinti vagy értékszerinti.

- Bemenete, eljárásnak v. fv. paraméter listába adható meg.

- a fv csak egy egyszerű kifejezésnek megfelelő fel.
úgy mint ahogy kell értelmezni ($\sqrt{4}$) SQRT(4).
a fv visszatérési értékének típusa, csak egy egyszerű típus lehet
- ha rendezetten se akarjuk írni a fv egy ~~[illegible]~~ összetett struktúrát ~~[illegible]~~ kell végrehajtani, ezért fv visszatérési értéke nem lehet ezért van az érték szerinti átadás

eljárás használatával egyben akar rendezni)
Buborék (A, B)

formális paraméter listában megjelenő
ELJÁRÁS BUBORÉK (x, y : SOROZAT); változók, deklarálását is jelentenek

tartalma át másolódik.

Ha érték paramétert alkalmazok, akkor lefoglal két memória területet, és adat átmásolás is történik.
• nekem vissza kapni rendezett formában.

RENDEZŐ algoritmus:

ELJÁRÁS BUBORÉK (x, y : sorozat);

címszerinti paraméter átadás szükséges, lokális, a [illegible] miatt

y := x ;

Ez az oda-vissza párhuzamos esetén kell, ha nem akarom a változásokat elveszteni érték szerinti átadás kell.

címszerinti, ha nincs szükségem rá.

FÜGGVÉNY ADD (A, B: EGÉSZ) : EGÉSZ;

fv kellett két dologban külön helyen értékadás egy lokális változóra,

ADD := A + B;
F. VÉGE;

mert ADD az azonos típusa.

BE : A, B
KI : ADD(~~x~~, ~~y~~)

→ Az jelenik itt meg ami az értékadás jobb oldalán szerepel

csak értéket adhatok neki, ha összegzés tételét szeretném megvalósítani

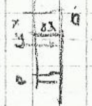
FÜGGVÉNY ÖSSZ (A : sorozat) : ELEMTÍP;
VÁLTOZÓ : I : EGÉSZ;
S : ELEMTÍP;
KEZD
ÖSSZ := 0;
CIKLUS I := 1 .. N
~~ÖSSZ~~ S := ~~ÖSSZ~~ S + A[I];
C.VÉGE;
ÖSSZ := S;

↳ ez itt nem lehet mert fv hívásnak veszi ezt, ezért Be kell vezetni egy újabb lokális változót;

jobb oldalon első típusa, meg kell egyezni a bal oldalival, ez jó mert az S is elemtípusú,

ELJÁRÁS ADDE (A, B: EGÉSZ)
A := A + B;
x = 3, y = 5

ADDE (x, y);
KI (x, y);



Adva van egy sorozat mennyi az értéke
Soros keresés, megmutatja van T tulajdonságú elem és hogy hol van.
Hiba helyén amennyiben van írja ki sorszámát, ha nincs írja ki nincs ilyen elem
Ebben a programban a logikai kimenetet kell feldolgozni.
A vektor tartalmazza az elemeit.

```
                          kimeneti paraméter
ELJÁRÁS   KERE(A: SOROZAT; HELY: EGÉSZ, F: LOGIKAI);
     VÁLTOZÓ: I: EGÉSZ

  KEZD
       I := 1;
       CIKLUS MIG (I ≤ N) ∧ (A[I] nem T tul.)
              I := I+1;
       CVÉGE;
       F := (I ≤ N);
       HELY := I;
  EVÉGE;
```

meghívás :

```
          helye
  : KERE (B, 4, X);  van vagy nincs

      HA X AKKOR
           KI: B[H]
      KÜLÖNBEN   KI: NINCS  T-tul. elem
      HVÉGE;
```

Ez a jobb megoldás!

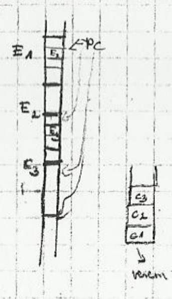
```
                    címszerinti       ki
FÜGGVÉNY  KERF(A: SOROZAT; HELY: EGÉSZ); LOGIKAI;

  KEZD
      HELY := 1
      CIKLUS MIG  (HELY ≤ N) ∧ (A[HELY] nem T tul.)
              HELY := HELY+1;
      CVÉGE;
      KERF := (HELY ≤ N);
  FVÉGE
```

meghívás

```
 :
 :
 : HA KERF(B,H) AKKOR
           KI: B[H];
   KÜLÖNBEN
           KI: 'NINCS T-tul. ..'
   HVÉGE;
```

VEREM

PC - regiszter utasítás számláló.



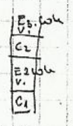
Következő már utasítás címével, ha a utolsót is végrehajtotta, és vissza kellene térni az E1-be, de ezt nem tárolta

Ehhez van szükség olyan adatszerkezetre, hogy az adatokat tárolni hogy majd hová kell vissza térni. Mindenféle adatot tárolni kell olyan tár-nak kell lenni, hogy fordított sorrendben tudjuk használni mondjuk ezt tudjuk kiolvasni belőle amit legutoljára raktunk bele.
Ezt az adatszerkezetet nevezzük LIFO-nak → verem

Lokális változók, amelyek csak addig élnek amíg az alprogram aktív. Akkor amikor megtörténik a visszatérés, és elmenti a visszatérési címet lefoglalja E2 lokális változói számára ezt a memóriaterületet amelyek nem címszerintiek.

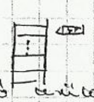
Függv (paraméter lista)
saját célra

Címszerintiekben kívül mindegyiknek lefoglal egy memóriaterületet.
Lefoglalódik a lokális változók számára nem címszerinti, és címszerinti ez azok, és a verembe tárolni



Ha megtörténik az utolsó utasítás, akkor visszatér C1 re fejlődik E2 utasításaival majd végrehajtja az elemet sorra és törlődik mindenki.

— o —



Lefoglalni nem üres területet tárolandó adatok számára.
~~Adatot~~ feltölteni kellene azt is amelyet a legutolsóra mutat ki, ennek tárolására használjuk a verem mutatót SP stack-pointert.

Tudnunk kell a verembe írunk, ha SP utolsó pozícióra mutat akkor nem tudok írni mert ott van adat tehát növelni kell

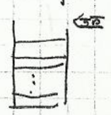
VEREBE
SP ← SP+ 1
ÍR

Tudjuk kiolvasni, mert a csak SP nem utolsó elért is

VEREMBŐL
OLVAS
SP ← SP - 1

ÍR
SP ← SP+ 1

Ez a jó -



SP ← SP - 1
olvas

MIRE JÓ a verem:

Olyan adatsorozat amelyből a legutóbb tett adatokat tudom kiolvasni

((()()) ())

Legutóbbi nyitott zárójelet kell csukni;

Ha nyitó zárójelet olvasunk rakjuk be a verembe, ha végzárójel ezzel olvasunk ki a veremből
És a végén üres veremnek kell lenni.

<u>Algoritmus</u> végig olvasni, tenni nyitót, kiolvasni a zárót.
(] nem tudja megmondani, hogy melyiket zárják be
Ha vég következik akkor azt mondja végül helyesek voltak.
Ha nyitó fordulna elő hogy nincs akkor üres veremből akarunk olvasni.

A hibajelzés helyét nem tudja megmondani.

A visszatérés fordított esetben történik a veremben, mint egyébként.

verembe	veremből	veremből	verembe
		ír	sp ← sp − 1
sp ← sp + 1	olvas	sp ← sp + 1	olvas
ír	sp ← sp − 1		☹
✓			

– Tömböt lehet indexelni ⇒ az indexet tároljuk. Ezek szorosan összetartozó adatok (tömbelem és a mutató) ⇒ REKORDBA foglaljuk.

```
– KONSTANS   N = tárolandó elemek száma;
  TÍPUS      ELEMT = tárolandó elemek típusa;
             ELEMEK = TÖMB[1..N] ELEMT;
             VEREMT = REKORD
                        ELEM: ELEMEK ;
                        SP : EGÉSZ  ;
                        <↳ veremmutató>
             REKORD VÉGE;

  VÁLTOZÓ    V1, V2 : VEREMTIP;

             V1.ELEM
             V1.ELEM[J]  } ⇒ V1.ELEM[V1.SP]
             V1.SP
```

az a pozíció, ahonnan kiolvasunk

és bármilyen esetben a verem tetején lévő elem lesz.

A verem inizalizációja

```
eljárás veremkezd (V:VEREM);
kezd
    V.SP=0;
elj. vége;
```

A veremmel kapcsolatban tudnunk kell, h. lehet-e új adatot beletenni (tele van), vagy kiolvasni (üres). 2 kérdezést kell végrehajtani.