Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

2. gyakorlat

PLanG:
Vezérlési szerkezetek és tömbök használata

© 2011.09.20. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Vezérlési szerkezetek

Elágazás

- A programunkat nem mindig fogalmazhatjuk meg egyértelműen egymást követő utasítások sorozataként
- Előfordulhat, hogy egyes utasításokat csak bizonyos esetben kell elvégeznie a programnak, valamilyen feltételtől függenek
- Ezek lekezelésére teszünk *elágazás*okat a programba, ennek egyik ágát fogja végrehajtani a program a feltétel teljesülése esetén, míg egy másik ágát a feltétel nem teljesülése estén
- Az elágazás szerkezete:

```
HA <feltétel> ** az elágazás kezdete és feltétele
AKKOR <utasítások> ** ha teljesül a feltétel
KÜLÖNBEN <utasítások> ** ha nem teljesül
HA_VÉGE ** vége az elágazásnak
```

Vezérlési szerkezetek

Elágazás

- A feltétel egy logikai kifejezés (pl.: **`a < 5`**, **`SZAM x`**, …), amiben lehet kötőszavakat használni (**`VAGY`**, **`ÉS`**), így összetettebb kifejezéseket is megadhatunk feltételként
- Ha a feltétel teljesül, az **`AKKOR`** ág utasításai hajtódnak végre, itt annyi utasítást teszünk egymást követően, amennyit csak szeretnénk, ha a feltétel nem teljesül, a **`KÜLÖNBEN`** ág utasításai hajtódnak végre
- A **`KÜLÖNBEN`** ág kihagyható, ha nincs szükségünk rá:

```
HA <feltétel>
AKKOR <utasítások>
HA_VÉGE
```

- Mindig figyeljünk arra, hogy minden **`HA`**-t megfelelő helyen lezárjunk egy **`HA_VÉGE`**-vel

Vezérlési szerkezetek

Példa

Feladat: Írjuk ki, ha a bemeneten 10 karakternél hosszabb szöveget írtunk, és ne írjunk semmit, ha nem hosszabb.

- ha csak kiíratnánk a kifejezés értékét, akkor hamisat is kiírna, de ekkor nem szeretnénk semmilyen üzenetet
- ezért elágazással oldjuk meg, amelynek csak az igaz ágában hajtjuk végre a kiírást

Specifikáció:

- bemenet: egy szöveg (*szo*)
- kimenet: igaz, ha a szöveg hosszabb 10-nél

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM szo_hossza
  VÁLTOZÓK:
    szo: SZÖVEG
  BE: szo
  HA (|szo| > 10) ** feltétel
  AKKOR
    KI: "a szó hosszabb 10-nél" ** ha igaz
  HA_VÉGE ** elágazás vége
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Példa

Feladat: Döntsük el egy valós számról, hogy pozitív-e.

- ne csak a kifejezés eredményét írjuk ki, hanem szövegesen, hogy „pozitív”
- elágazást használunk, és a feltételben megvizsgáljuk, hogy az adott szám nagyobb-e, mint nulla
- egy valós változót használunk a programban

Specifikáció:

- bemenet: egy valós szám (*szam1*)
- kimenet: ha a szám pozitív, akkor ”pozitív” szöveg, ha nem, akkor ”nem pozitív”

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM pozitiv_e
  VÁLTOZÓK:
    szam2: VALÓS
  BE: szam1
  HA (szam1 > 0) ** feltétel
  AKKOR
    KI: "pozitív" ** ha igaz
  KÜLÖNBEN
    KI: "nem pozitív" ** ha hamis
  HA_VÉGE ** elágazás vége
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Többszörös elágazás

- Lehetőségünk van, hogy elágazásunk egyik ágában ismét egy elágazást adjunk meg, így egymásba ágyaztathatjuk az elágazásokat
  - pl.:

```
HA <feltétel> ** külső elágazás
AKKOR HA <másik feltétel> ** beágyazott elágazás
    AKKOR <utasítások>
…
```

  - a belső elágazás is ugyanolyan, mint a külső, oda is feltételt kell adnunk, nem kötelező a különben ága, illetve ugyanúgy be kell zárnunk
  - a belső elágazást tehetjük az igaz, és a hamis ágba is
  - a beágyazás tetszőleges szintig folytatható

Feladat: Az előző programot kiegészíthetjük, hogy eldöntse, a szám pozitív, negatív, vagy nulla-e.

- be kell ágyaznunk egy újabb elágazást a különben ágba, amely mentén szétválasztjuk a nulla és negatív értékeket

Specifikáció:

- bemenet: egy valós szám (*szam1*)
- kimenet: ha a szám pozitív, akkor ”pozitív” szöveg, ha negatív, akkor ”negatív”, különben „nulla”

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM beagyazott_elagazas
  VÁLTOZÓK:
    szam2: VALÓS
  BE: szam1
  HA (szam1 > 0) AKKOR ** külső elágazás
    KI: "pozitív" ** külső igaz
  KÜLÖNBEN
    HA (szam1 < 0) AKKOR ** belső elágazás
      KI: "negatív" ** külső hamis, belső igaz
    KÜLÖNBEN
      KI: "nulla" ** külső hamis, belső hamis
    HA_VÉGE ** belső elágazás vége
  HA_VÉGE ** külső elágazás vége
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Ciklusok

- Gyakran előfordul, hogy valamilyen utasítást többször (akár nagyon sokszor) is végre akarunk hajtani, teljesen ugyanúgy, vagy nagyon hasonlóan
- Gyakran előfordul, hogy az utasítások végrehajtásának száma valamilyen feltételtől függ, pl.: amíg van bemeneti adat, amíg véget nem ér a fájl, amíg van hálózati kapcsolat, …
- A programszerkezetben, többször, feltétel függvényében lefutó utasításokat *ciklus*ba tehetjük
  - a ciklusban lévő utasításokat a program a feltétel függvényében valahányszor fogja lefuttatni
  - a feltétel lehet valamilyen logikai kifejezés, vagy megadhatjuk, hogy pontosan hányszor fusson a ciklus

Vezérlési szerkezetek

Ciklusok

- A ciklusban végrehajtandó utasításokat nevezzük *ciklusmag*nak, a végrehajtási feltételt nevezzük *ciklusfeltétel*nek
- A feltétel fajtájának függvényében a ciklus 3 típusát különböztetjük meg:
  - *előtesztelő*: a ciklusfeltétel a ciklusmag előtt van, már elsőre is csak akkor futnak le az utasítások, ha a feltétel teljesül
  - *utántesztelő* (*hátultesztelő*): a ciklusfeltétel a ciklusmag után van, ezért az egyszer mindenképpen lefut, de többször csak akkor, ha teljesül a feltétel
  - *számláló*: mi akarjuk pontosan megadni, hányszor forduljon le a ciklusmag

Vezérlési szerkezetek

Ciklusok

- Előtesztelő ciklus szerkezete:

```
CIKLUS AMÍG <feltétel>
   ** csak akkor lép be, ha a feltétel már
   ** kezdetben teljesül
   <utasítások>
CIKLUS_VÉGE
```

- Hátultesztelő ciklus szerkezete:

```
CIKLUS
   <utasítások>
   ** az utasítások egyszer mindenképpen
   ** lefutnak
AMÍG <feltétel>
```

Vezérlési szerkezetek

Ciklusok

- A ciklusmagban bármennyi, és bármilyen utasítás szerepelhet, akár másik ciklusokat is betehetünk oda
- Csak az előtesztelőnek kell jelezni a végét a **`CIKLUS_VÉGE`** kulcsszóval
- A ciklusba csak addig lép be, amíg a ciklusfeltétel teljesül, ha ez már kezdetben nem teljesül, akkor az előtesztelő egyszer sem, az utántesztelő pedig egyszer futtatja le az utasításokat
- A feltétel egy (egyszerű, vagy összetett) logikai értékű kifejezés lehet (ahogy az elágazásnál is), pl.: **`i < 10`**
- Figyeljünk arra, hogy mindig olyan ciklusfeltételt adjunk meg, amely garantálja, hogy a ciklusból valamennyi lépés után kilép a program a ciklusból, különben a ciklus a végtelenségig fog futni (a program *végtelen ciklusba* kerül)

Vezérlési szerkezetek

Példa

Feladat: Olvassunk be egy ponttal végződő szót a bemenetről, és írjuk ki csak nagybetűkkel.

- van egy kulcsszavunk, amely nagybetűt készít (**`NAGY`**), de ez csak karakterekre alkalmazható
- ezért karakterenként kell beolvasnunk a szöveget, és kiíratnunk, használjunk utántesztelő ciklust
- addig kell olvasnunk, amíg a ponthoz nem érünk, tehát ez fog szerepelni a ciklusfeltételben

Specifikáció:

- bemenet: egy szöveg, karakterek (*x*) sorozata
- kimenet: a szöveg nagybetűsen

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM karakter_konverzio
  VÁLTOZÓK:
    x: KARAKTER
  CIKLUS
    BE: x
    KI: NAGY x
  AMÍG (x /= '.')
  ** amikor .-t olvasunk be, azt még kiírja
  ** akkor is, ha csak egy pontot írunk, de
  ** utána már kilép a ciklusból
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Példa

Feladat: adjuk meg a bemenetről kapott számok összegét, amíg 0-t nem olvasunk be.

- bármennyi számot írhatunk a bemenetre, de valahol kell lennie egy 0-nak a sorozatban
- a beolvasást és az összeadást a ciklusmagba helyezzük, így annyi számot tudunk feldolgozni, amennyit szeretnénk
- legyen a beolvasott változó az `a`
- a feltétel az lesz, hogy a beolvasott szám ne a 0 legyen, tehát: `a /= 0`
- ha a 0-t hozzáadjuk, az összeg nem változik, ezért mindegy, hogy a 0 beolvasása után, vagy előtte terminálják a ciklust
- használjunk utántesztelő ciklust

Vezérlési szerkezetek

Példa

- kell egy változó, amiben eltároljuk az eddig összeadott számokat, legyen ez **`s`**
- **`s`**-t kezdetben 0-re kell állítanunk, majd minden ciklusléptésben hozzáadni a most beolvasott számot
- miután lefutott a ciklus, kiírhatjuk az eredményt (**`s`**-t)

Specifikáció:

- bemenet: egész számok (*a*) sorozata
- kimenet: a számok összege (*s*)

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM osszegzes
  VÁLTOZÓK:
    a, s: EGÉSZ
  s := 0 ** az összeget kezdetben nullázzuk
  CIKLUS
    BE: a
    s := s + a
    ** s-be összegezzük az értékeket
  AMÍG (a /= 0)
  ** utántesztelő, egyszer mindenképpen lefut
  KI: s ** az összegzett értéket kiírjuk
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Számláló ciklus

- A számláló ciklus szerkezete megegyezik az előtesztelő ciklusével, de valamivel nyomon kell követnünk, hogy hányszor futott le eddig a ciklus
  - deklarálunk egy változót, amelyet adott kezdőértékről indítunk (általában 0, vagy 1), és minden lépésben megnöveljük az értékét
  - ha pl. 10-szer szeretnénk lefuttatni a ciklusmagot, akkor 0-val indulva ez a változó az utolsó lépésben 10-re fogja növelni az értékét, ezért ha olyan feltételt adunk neki, hogy kisebb legyen, mint 10, akkor pontosan 10 lépés után ki fog lépni a ciklusból
  - az ilyen változókat nevezzük *ciklusváltozók*nak, vagy *ciklusszámláló*nak

Vezérlési szerkezetek

Számláló ciklus

- A számláló ciklus szerkezete:

```
VÁLTOZÓK:
  i : EGÉSZ ** ez lesz a ciklusszámláló

i := 0 ** a ciklusváltozót a ciklus kezdete
       ** előtt lenullázzuk
CIKLUS AMÍG i <10
  ** ciklusmag
  i := i + 1
  ** ciklusváltozó növelése a cikluson belül az
  ** utolsó lépés (máshol nem szabad
  ** változtatni az értékét a cikluson belül)
CIKLUS_VÉGE
** innentől másra is használhatjuk az i-t
```

Vezérlési szerkezetek

Számláló ciklus

- A ciklusváltozót másképpen is változtathatjuk a ciklusmagban
  - pl.: 10-es értékkel indítjuk, minden lépésben eggyel csökkenjük, 0-ig, ekkor ugyanaz lesz, mint az előbb
- Mindig adjunk meg ciklusváltozó módosítást a ciklusmagban, méghozzá olyat, ami a ciklus befejezéséhez vezet (azaz idővel teljesülni fog a feltétel), különben végtelen ciklusba kerülünk
- Nem kötelező az utolsó lépésként növelni a ciklusváltozót, de ha használjuk a ciklusváltozó értékét használjuk a cikluson belül, akkor ügyeljünk arra, hogy a megfelelő értéke legyen mindig

Vezérlési szerkezetek

Példa

Feladat: Adjuk meg a bemenetre adott 5 szám átlagát.

- az általhoz kell összeadás és osztás, 5-ször kell beolvasnunk és összeadnunk, majd osztanunk
- amit ismétlünk, azt ciklusba tesszük, amit pontosan ötször kell lefuttatni, tehát számláló ciklust használunk
- használjuk az összegzés tételét, beolvasunk egy változóba (`a`), egybe összegzünk (`s`), és lesz egy változó ciklusszámlálónak (`i`), `s`-t osztani fogjuk, tehát legyen valós

Specifikáció:

- bemenet: 5 egész szám
- kimenet: a számok átlaga

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM atlag
  VÁLTOZÓK:
    a, i: EGÉSZ, s: VALÓS
  s := 0 ** beállítjuk az összeget
  i := 0 ** beállítjuk a ciklusváltozót
  CIKLUS AMÍG (i < 5)
    ** összesen 5 lépést kell megtennie
    BE: a
    s := s + a
    i := i + 1 ** i-t minden lépésben növeljük
  CIKLUS_VÉGE
  KI: "A számok átlaga: ", s / 5
PROGRAM_VÉGE
```

Vezérlési szerkezetek

Példa

Feladat: Szorozzuk össze páronként a bemenetre adott 8 számot.

- tudjuk, hogy pontosan 4 szorzást kell végeznünk, mindegyiket két számmal, ezért használhatunk számláló ciklust, ami 4 lépést végez
- minden lépésben beolvasunk két számot, majd kiíratjuk a szorzatukat

Specifikáció:

- bemenet: 8 egész szám
- kimenet: a számok szorzata páronként

Vezérlési szerkezetek

Példa

Megoldás:

```
PROGRAM paronkenti_szorzat
  VÁLTOZÓK:
    a, b, i: EGÉSZ
  i := 0 ** beállítjuk a ciklusváltozót
  CIKLUS AMÍG (i < 4)
    ** összesen 4 lépést kell megtennie
    BE: a, b
    KI: a * b
    i := i + 1
    ** i-t minden lépésben eggyel növeljük
  CIKLUS_VÉGE
PROGRAM_VÉGE
```

Tömbök

Azonos típusú adatok feldolgozása

- A programozás során legtöbb esetben nem csak egy adattal dolgozunk, hanem adatok sorozatával, amiket fel szeretnénk dolgozni, ezek általában azonos típusúak
  - ciklusban fel tudunk dolgozni sok azonos típusú adatot, ekkor a számokat közvetlenül alávetjük valamilyen műveletnek, és az eredeti értékek elvesznek
  - ha szeretnénk, hogy az eredeti értékek is megmaradjanak, akkor azokat el kell tárolnunk változókba , így a program során később bármikor felhasználhatóak lesznek
- Ha sok adatunk van a bemeneten, akkor sok változót kéne létrehozni, ez elbonyolítaná a programot, és nem tudnánk ciklusba tenni az adatok kezelését

Tömbök

Azonos típusú adatok feldolgozása

- Jó lenne, ha egy változóban el tudnánk tárolni az összes adatot, egymás után, és a ciklusban hivatkozni tudnánk ebben a változóban tárolt értékekre, műveletet végezni velük
- A programozási nyelvek erre megoldásként a *tömb*öt adják
- A tömb elemek sorozata, amelyek ugyanahhoz a változóhoz tartoznak, és tudunk hivatkozni az egyes elemekre
- Tömb típusú változóknál meg kell adnunk, hogy hány elemet szeretnénk eltárolni, és milyen típusú elemből, ugyanis a tömböt előzetesen létre kell hoznia a programnak, mielőtt feltöltené elemekkel:

  <*változónév*> : <*típus*>[<*elemszám*>]

  - pl.: **`t: EGÉSZ[10]`**

Tömbök

Használata

- A tömb elemei meg vannak *indexelve*, azaz egy sorszám van hozzájuk társítva, hogy hányadikak a tömbben, ezzel az értékkel tudunk hivatkozni rájuk
  - tömbelem elérése: **`<változónév>[<index>]`**
  - pl.:
```
t[1] := 3
KI: t[4]
t[5] := t[1] + t[2]
```
- *Figyelem*: az indexelést mindig 0-tól kezdjük, és

  (tömb mérete - 1)-ig tart, azaz, pl. a **`t`** első eleme **`t[0]`**, és ha a **`t`** mérete **`n`**, akkor az utolsó eleme **`t[n-1]`**
- Ha rossz indexet adunk meg, azt a fordító nem veszi észre, ezért futás közbeni hibát fogunk kapni

Tömbök

Használata

- Az index egész típusú érték, és megadható változó segítségével is, pl.: **`s:= 5, a[s]`**
- Ez passzol a számláló ciklussal is: a ciklusváltozó amúgy is változik állandóan, az értékét minden lépésben növeljük, ezért használható a tömb elemeinek elérésére is
- A tömb mérete mondja meg, mennyi helyet foglalunk le a memóriában, a szükségesnél többet is adhatunk neki
  - elemszám lekérdezés: **`|<változónév>|`**

Tömbök

Példa

Feladat: Adjuk meg a bemenetre kapott 5 valós szám átlagát úgy, hogy egy tömbbe olvassuk be az értéket.

- legyen a tömb neve **`t`**, mérete 5

Specifikáció:

- bemenet: 5 egész szám
- kimenet: a számok átlaga

Megoldás:

```
PROGRAM ossszead_tomb
  VÁLTOZÓK:
    t: VALÓS[5], ** valós számok 5 hosszú tömbje
```

Tömbök

Példa

```
    s: VALÓS, i: EGÉSZ

  i := 0 ** ciklusszámláló nullázása
  s := 0
  CIKLUS AMÍG (i < |t|) ** tömb méretéig
    BE: t[i] ** a tömb aktuális eleme
    s := s + t[i]
    i := i + 1 ** ciklusszámláló növelése
  CIKLUS_VÉGE
  KI: "A számok átlaga: ", s / 5
PROGRAM_VÉGE
```

Tömbök

Példa

Feladat: Generáljunk véletlen számokat 0 és 100 között, adjuk őket össze, majd írjuk ki szövegszerűen az összeadást a kimenetre.

- először írjuk ki az összeget, majd aztán az értékeket, pl.: 123 = 16 + 81 + 26
- most már van értelme a tömb használatának
- 3 ciklus lesz a programban: egyik feldolgozza a sorozatot, egyik összegez, egy pedig kiírja az eredményt és az értékeket (igazából nem szükséges a három ciklus, megoldható kettővel is, de eggyel már nem)

Tömbök

Példa

Specifikáció:

- bemenet: egy egész tömb (*a*)
- kimenet: a tömb elemeinek összege (*s*), az összeg kiíratása

Megoldás:

```
PROGRAM veletlen_osszegzes
  VÁLTOZÓK:
    a: EGÉSZ[10],
    s, i: EGÉSZ
  s := 0
```

Tömbök

Példa

```
i := 0 ** feldolgozás
CIKLUS AMÍG (i < 10)
  a[i] := RND 100 ** véletlenszámok
  i := i + 1
CIKLUS_VÉGE
** összeadó ciklus, ismét i-t használjuk, újra
** nullázni kell
i := 0
CIKLUS AMÍG i < 10
  s := s + a[i]
  i := i + 1
CIKLUS_VÉGE
```

Tömbök

Példa

```
    ** a tömb első elemét rögtön kiíratjuk
    KI: s, " = ", a[0]
    ** a többit ciklusban, ezért 1-ről indul a
    ** számláló, használhatjuk ismét az i-t
    i := 1
    CIKLUS AMÍG i < 10
      KI: '+', a[i]
      i := i + 1
    CIKLUS_VÉGE
PROGRAM_VÉGE
```

Tömbök

Egy és több dimenziós tömbök

- Felmerül a kérdés: ha a tömb bármilyen adattípusból tud tárolni egy sorozatot, akkor tömbből is tud?
- Igen: az eddig bemutatott tömböket *egy dimenziós tömbök*nek, vagy vektoroknak nevezzük, lehetnek *több dimenziós tömb*ök is, amelyek tömbökben tömböket tárolnak
- Pl. készítsünk 10 elemű tömböt, amelynek minden eleme egy 10 elemű, egész számokat tartalmazó tömb:

```
a : EGÉSZ[10][10] ** 10*10 = 100 elem
```

- Tetszőleges mélységig haladhatunk a konstrukcióban: n dimenziós tömb
  - az 1 dimenziós tömböket *vektor*nak, a 2 dimenziós tömböket *mátrix*nak, a 3 dimenziós tömböket *térbeli mátrix*nak nevezzük

Tömbök

Példa

Feladat: Generáljunk véletlenszerűen 5 db 3 dimenziós koordinátát, és írjuk ki őket sorban.

- összesen 5 darab koordináta kell, mindegyik koordináta 3 elemből áll, ezért el kell tárolni 5 elemet egy tömbben, ahol minden elem egy 3 elemű, egészeket tartalmazó tömb lesz
- tehát 3*5 = 15 számot tárolunk el egy mátrix segítségével
- a mátrix bejáráshoz egymásba ágyazott számláló ciklusok szükségesek, a külső ciklus 5-ig, a belső ciklus 3-ig halad, ehhez két ciklusváltozó kell (*i*, *j*)

Tömbök

Példa

Specifikáció:

- bemenet: 15 egész szám
- kimenet: 5 db 3 dimenziós koordináta (*koord*)

Megoldás:

```
PROGRAM harom_dim_koordinatak
  VÁLTOZÓK:
    koord: EGÉSZ[5][3],
    ** egészeket tartalmazó mátrix
    i, j: EGÉSZ
```

Tömbök

Példa

```
** generáló rész
i := 0 ** külső ciklus: i index
CIKLUS AMÍG (i < 5) ** külső ciklus
  j := 0 ** belső ciklus: j index
  CIKLUS AMÍG (j < 3) ** belső ciklus
    koord[i][j] := RND 100
    j := j + 1 ** belsőben növeljük j-t
  CIKLUS_VÉGE ** belső ciklus vége
  i := i + 1 ** külsőben növeljük i-t
CIKLUS_VÉGE ** külső ciklus vége
```

Tömbök

Példa

```
    ** kiíratő rész:
    i := 0
    CIKLUS AMÍG (i < 5)
      j := 0
      ki : "[" ** kiíratás formázottan
      CIKLUS AMÍG (j < 3)
        ki : " ", koord[i][j]
        j := j + 1
      CIKLUS_VÉGE
      ki : "]", sv ** minden koordináta új sorba
      i := i + 1
    CIKLUS_VÉGE
  PROGRAM_VÉGE
```

II. Vegyes feladatok:

1. b) "Rajzolj" ki egy NxN-es négyzetet *-okból.

   c) Rajzolj ki egy N hosszú befogójú, egyenlő szárú derékszögű háromszöget *-okból.

3. Sorold fel két pozitív egész szám közös osztóit.

5. Sorold fel a K-nál kisebb négyzetszámokat.

7. (*) Add meg az N. Fibonacci-számot. A Fibonacci sorozat egész számokból áll, az első két tagja 0 és 1, és minden további tagja az előző két tag összege.

13. a) Egy két tagú névnek add meg a monogramját.

IV. Tömbök:

3. Vektor szórása (átlagtól való eltérések átlaga).
4. Van-e két egyforma elem a vektorban?
5. (*) Vektor permutálása, hogy végül monoton növekedő sorrendben legyenek az elemek a vektorban.