



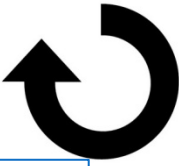
Bevezetés a programozásba

5. Előadás Tömbök, számábrázolás

2016.10.17.1215. papíros ZH

- **Simonyi terem:** Angi Péter .. Metzing Máté
-
- **Neumann terem:** Mezősi Anikó .. Pirnák Péter
-
- **320-as terem:** Pomogyi Flóra .. Szűcs Eszter
-
- **419-es terem:** Takács Valentin .. Zsenyei Roland
+ **digitálisbölcsész hallgatók**

Összegzés tétele



Változók: összeg, a : T

összeg := 0

CIKLUS AMÍG *nincs vége a sorozatnak*

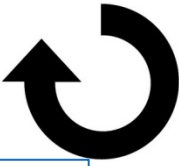
 a := *következő elem*

 összeg := összeg \oplus f(a)

CIKLUS_VÉGE

Ahol T az összegzendő adatok típusa,
 \oplus a T típuson értelmezett összeadási művelet,
f() pedig a művelet elvégzéséhez szükséges függvény.
(Ez utóbbira nem feltétlenül van szükség.)

Számlálás tétele



Változók: számláló: EGÉSZ,
a : T

számláló := 0

CIKLUS AMÍG *nincs vége a sorozatnak*

a := *következő elem*

HA *feltétel(a)* AKKOR

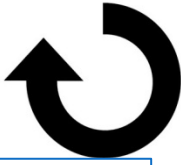
számláló := számláló + 1

HA_VÉGE

CIKLUS_VÉGE

Ahol T a vizsgálandó adatok típusa,
a *feltétel()* azon vizsgálat, amely megállapítja egy adott
elemre, hogy igaz-e a vizsgálandó tulajdonság.

Lineáris keresés tétele



Változók: hol, i: EGÉSZ,
 van: LOGIKAI, a : T

van := HAMIS

hol := 0

i := 0

CIKLUS AMÍG *nincs vége a sorozatnak* ÉS NEM van

 a := *következő elem*

 i := i + 1

 HA *feltétel(a)* AKKOR

 van := IGAZ

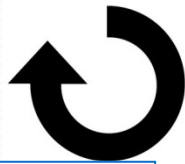
 hol := i

 HA_VÉGE

CIKLUS_VÉGE

**Ahol T a vizsgálandó adatok típusa,
a *feltétel()* azon vizsgálat, amely
megállapítja egy adott elemre,
hogy igaz-e a vizsgálandó
tulajdonság.**

Maximum keresés tétele



Változók: i , hol : EGÉSZ,
 a , max : T

$i := 1$

$a := \text{első elem}$

$max := f(a)$

$hol := 1$

CIKLUS AMÍG *nincs vége a sorozatnak*

$a := \text{következő elem}$

$i := i + 1$

 HA $max < f(a)$ AKKOR

$max := f(a)$

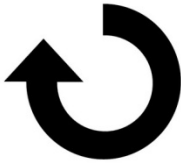
$hol := i$

 HA_VÉGE

CIKLUS_VÉGE

**Ahol T a vizsgálandó adatok típusa,
a $f()$ azon függvény, amely
meghatározza az adott elem
értékét.**

Ismert hosszú sorozat



- A sorozat hossza ismert, vagy beolvasható
- Számoljuk az olvasások számát, és ha elérjük a sorozat hosszát, abbahagyjuk

Változók: $i, n : \text{EGÉSZ}, X : T$

$n := A \text{ sorozat hossza}, (pl: BE: n)$

$i := 0$

CIKLUS AMÍG $i < n$

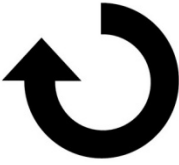
BE: X

X *feldolgozása*

$i := i+1$

CIKLUS_VÉGE

Végjeles sorozat



Változók: $X : T$

BE: X

CIKLUS AMÍG X nem végjel
 X feldolgozása

BE: X

CIKLUS_VÉGE

Adat beolvasása

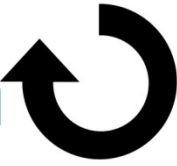
Adat ellenőrzése:

- ha sorozat elem, akkor az elem feldolgozása, majd új adat beolvasása

- ha végjel, akkor végeztünk a sorozattal és folytatódik a program

Ezt a technikát hívjuk:
Előreolvasásnak

Fájl kezelése általában PLaNG-ban



VÁLTOZÓK:

fb: BEFÁJL,

a: T

MEGNYIT fb: "fájlnev"

BE fb: a

CIKLUS AMÍG NEM VÉGE fb

a feldolgozása

BE fb: n

CIKLUS_VÉGE

LEZÁR fb

Mi a közös az eddigiekben?

- Tetszőleges hosszú sorozat feldolgozása
- Kevés (max 5-6) változó elég
- Kizárólag elemenként feldolgozható feladatok
 - Minden elemet pontosan egyszer kezelünk
 - Később már nem tudjuk elérni őket, hogy esetleg újra felhasználjuk egy művelethez
 - A sorrend kötött: csak abban a sorrendben tudunk olvasni, ahogy a sorozatban egymást követik az elemek

A tömb

- Jó lenne, ha **egy** változóban tudnánk tárolni **egymás után** az összes **azonos típusú** adatot. A ciklusban hivatkozni tudnánk az ebben a változóban tárolt értékekre, műveletet végezhetnénk velük, esetleg később módosíthatnánk őket.
- A programozási nyelvek adnak eszközt ilyen változó használatára: ez lesz a **tömb**.
 - A **tömb** elemek sorozata, amelyek ugyanahhoz a változóhoz tartoznak
 - hivatkozni tudunk az egyes elemeire, szabadon címezhető, tetszőleges sorrendben bejárható,
 - futás közben átírható, kezdeti értéket nem tartalmazó változósorozat, ellentétben a bemeneten kapott sorozatokkal

Tömbök

- Tömb típusú változóknál **előre**, fordítási időben tudni kell a **méretét** és hogy **milyen típusú** elemeket szeretnénk eltárolni benne, ugyanis a tömböt előzetesen létre kell hoznia a programnak, mielőtt feltöltené elemekkel.
- *A hallgatók hajlamosak abba a hibába esni, hogy mindent tömbökkel akarnak megoldani. Egy tipikus hibás hozzáállás, hogy „ismeretlen hosszú” sorozatot akarnak beolvasni mondjuk egy 1000 méretű tömbbe...*

A tömb

- A tömb egy típuskonstrukció: egy egyszerű típust megsokszorozunk. Adott hosszú sorozatát, mint „önálló” típust kezelünk.

<változónév>: <típus>[<elemszám>]

a: EGÉSZ[10]

- A tömb típusú változót közvetlenül ritkán, inkább a hordozott sorozat egy-egy tagját kezeljük

PL.: a[2] := 3 (vagy a₂ = 3)

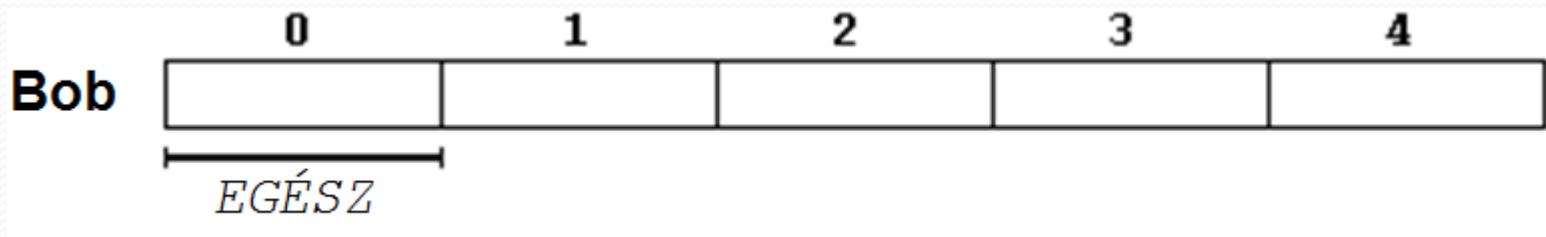
- Lényegében (nullától) indexelt, egyforma típusú változók egységes kezeléséről van szó

A tömb címzése

- A tömb elemei **indexelve** vannak, azaz egy-egy sorszám van hozzájuk társítva. Ezek azt jelzik, hogy egy adott elem hányadik a tömbben, és ezzel az értékkel tudunk hivatkozni rá.
 - tömbelem elérése: $\langle \text{változónév} \rangle [\langle \text{index} \rangle]$
 - pl.:
 $a[1] := 3$
KI: $a[4]$, BE: $a[3]$,
 $a[5] := a[1] + a[2]$
- **Figyelem!** Az indexelést mindig **0**-tól kezdjük, és (**tömb mérete - 1**)-ig tart, azaz
 - A ***t*** tömb első eleme ***t[0]***
 - Ha ***t*** mérete ***n***, akkor ***t*** utolsó eleme ***t[n-1]***
- Ha rossz indexet adunk meg, a fordító azt is elfogadja, de futás közbeni hibát kaphatunk, ha például a ***t[n]***-t akarjuk lekérdezni!

A tömb tulajdonságai

- Azonos típusú elemek
- Összefüggő memóriaterület (egymás után)
- Indexelhető (0-tól kezdve!)
- A méretnek fordítási időben ismertnek kell lennie.
- Bob: EGÉSZ[5]



Példa tömbre PLaNG-ban

PROGRAM tömb

VÁLTOZÓK:

a : EGÉSZ[10],

i : EGÉSZ

a változó egy 10 elemű tömb,
mely EGÉSZ típusú értékeket
tud tárolni

i := 0

CIKLUS AMÍG i < 10

a[i] := RND 5 + 1

i := i + 1

Az *a* tömb *i*. indexű
elemének értéket
adunk

CIKLUS_VÉGE

PROGRAM_VÉGE

Tömb, mint ismert hosszú sorozat

- A tételek tömbökön is alkalmazhatóak, mivel ismert hosszú sorozatokról van szó, ahol az adott elem a sorszámmal hivatkozható

```
...  
sum := 0  
i := 0  
CIKLUS AMÍG i < 10  
    sum := sum + a[i]  
    i := i + 1  
CIKLUS_VÉGE  
...
```

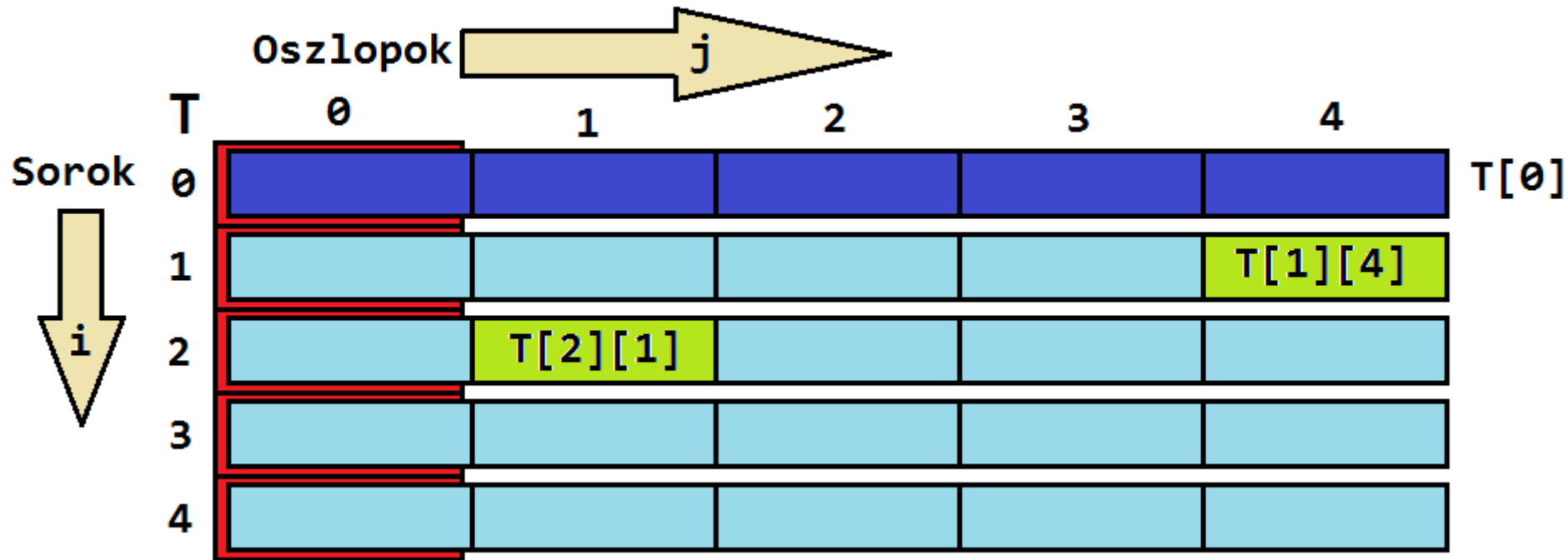
Többsdimenziós tömbök

- A tömb szintaxisa szerint **T[méret]** a tömb, ahol T tetszőleges típus
- A tömb is típus
- Tehát **T[méret1][méret2]** is helyes, és tömbök tömbjét jelenti
 - Ez kétdimenziós tömb, két független indexet lehet használni, mint koordinátákat
- Természetesen tetszőlegesen fokozható a dimenziószám – elméletben. Gyakorlatban kifogyunk a memóriából.

Tömb: vektor, mátrix

- Az egydimenziós tömböt szokás vektornak, a kétdimenziós tömböt mátrixnak nevezni
- Egy lehetséges mátrix reprezentáció:

T: EGÉSZ[5][5]



Példa kétdimenziós tömbre

```
PROGRAM mátrix
  VÁLTOZÓK:
    a : EGÉSZ[10][10],
    i, j: EGÉSZ

    i := 0
    CIKLUS AMÍG i < 10
      j := 0
      CIKLUS AMÍG j < 10
        a[i][j] := RND 5 + 1
        j := j + 1
      CIKLUS_VÉGE
      i := i + 1
    CIKLUS_VÉGE
```

a változó egy 10 elemű tömb,
mely 10 elemű EGÉSZ típusú
értékeket tároló tömböket tud
tárolni (10x10-es mátrix)

Az *a* mátrix *i*. sor, *j*.
oszlopában helyet
foglaló elemének
értéket adunk

```
PROGRAM_VÉGE
```

Tömbök jelentősége

- Olyan feladatoknál, ahol
 - több adatra van szükség, mintsem külön változóknak kényelmesen kezelni lehessen, de fix számú, és még beleférünk a memóriába
 - többször kell kiértékelni ugyanazt az értéket
 - tetszőleges sorrendben kell hozzáférni az elemekhez
 - Az adatokat módosítani is kell, nem csak olvasni
- ... szükségessé válik a tömb használata.

Néhány példa

- Táblázatos adatok
- A szöveg típus néhány művelettől eltekintve felfogható karakter-vektornak
- Hagyományos mátrixműveletek, Gauss elimináció, bázistranszformáció
- Képfeldolgozó szoftverek a képet mátrixként tárolják
 - A pixelek színértékét mátrixba rendezett számhármassal írják le

Tömbök és a PLanG

- A tömbelemek kezdeti érték nélkül jönnek létre (mint minden egyéb változó...)
- A tömb típusú változók kiírhatóak, de nem beolvashatóak, csak elemenként
- A változók értékeit mutató táblázatban az egész tömb nyomon követhető

Összefoglaló

- Tömbök: fix hosszú homogén változósorozat
- Szintaxis: Típus[méret]
- Akkor és csak akkor használandó, ha ismert hosszú, többszöri írást vagy olvasást, vagy tetszőleges sorrendben feldolgozást igényel a feladat
- Lehet több dimenziós is

Rövid kitérő a számábrázoláshoz

- Az „EGÉSZ” illetve „VALÓS” típusok nevei azt a látszatot keltik, hogy a típusértékhalmoz a teljes egész illetve valós szám tartományt fedi
- Ez természetesen lehetetlen, véges hosszú memóriaszeletek állnak rendelkezésre a számok ábrázolásához
- A fenti típusok tehát nem mindenben viselkednek a várakozásnak megfelelően

Rövid kitérő a számábrázoláshoz

- Például az EGÉSZ típus 32 biten ábrázolt szám, tehát legfeljebb 2^{32} féle értéket vehet fel. Ezt praktikusán a $-2^{31} .. +2^{31}-1$ tartományra tolták: $-2147483648 .. 2147483647$
- Ennek az a következménye, hogy:
 $2147483647 + 1 = -2147483648$
- Ezt a jelenséget túlcsordulásnak nevezik

Rövid kitérő a számábrázoláshoz

- A kicsit könnyebb érthetőség miatt használjunk 32 bites „ELŐJELMENTES EGÉSZ” típust. Továbbra is 2^{32} féle számot fogunk tudni ábrázolni, de most csak kizárólag **pozitív** számokkal foglalkozunk.
- Azaz a tartomány $0 \dots 2^{32}-1$ lesz:
 $0 \dots 4294967295$
- A következmény persze változatlan :
 $4294967295 + 1 = 0$

Rövid kitérő a számábrázoláshoz


- A valós számok számítógéppel történő megadása a véges ábrázolás miatt pontatlan
- Számológépről ismerős lehet:
 - $20/3 = 6.66666667$
 - vagy $20/3 = 6.6667$
 - vagy $20/3 = 6.67$
- **Ebből kifolyólag soha nem vizsgáljuk programban egy valós típusba tartozó változó egyenlőségét semmivel!**

Rövid kitérő a számábrázoláshoz

- A valós számokat $X * 2^Y$ alakban tárolják, visszavezetve az egész számokra
- Pl.: 32 bites esetben 23 bit jut X-re, 8 bit Y-ra, plusz 1 bit az előjelre
- Azt a számábrázolást lebegőpontos-nak hívják, mert a tizedesvessző az ábrázolandó „értékes” számjegyeken belül bárhova kerülhet:
 - Pl.: 1,23 vagy 12,3 vagy 123 (mindegyik 3 értékes számjegyet tartalmaz)
- Ennek következménye, hogy nem mindenhol egyformán sűrű az ábrázolás, ha 23 kettesjegynél nagyobb a nagyságrendi különbség, akkor előfordulhat, hogy $A \neq 0$, de $A+B = B$

Összehasonlítás

Primitív Adat Típus	Méret	Tartomány
EGÉSZ	32 bit	-2147483648 .. 2147483647
VALÓS	64 bit	$\pm 4.9\text{E}-324$ vagy $\pm 1.7976\text{E}+308$
KARAKTER	16 bit	0 .. 65535
LOGIKAI	1 bit	Igaz, Hamis



Itt a vége a PLanG
képeségeinek, legközelebb a
C++ alapoktól folytatjuk