



Bevezetés a programozásba

5. Előadás Tömbök, számábrázolás

Elemenkénti feldolgozás

- 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 PlanGban 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**)*

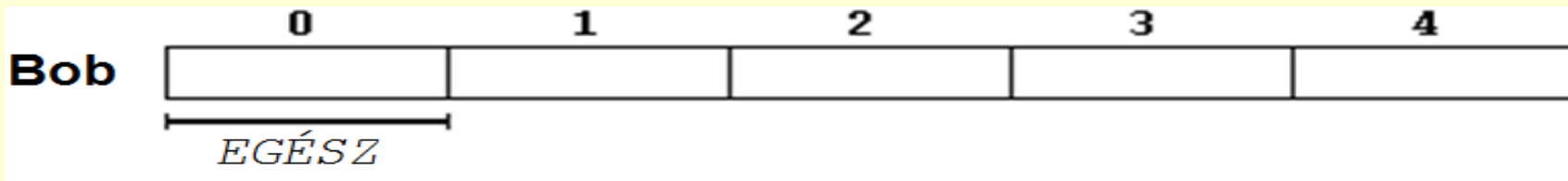
- Lényegében (nullától) indexelt, egyforma típusú változók egységes kezeléséről van szó

A tömb címezése

- 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: $a[4]$ illetve BE: $a[3]$,
 $a[5] := a[i] + a[j] \quad ** \quad i, j : \text{EGÉSZ}$
- Az indexelést mindig 0-tól kezdjük, 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 futásidejű hibát kapunk, 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
```

```
  i := 0
```

```
  CIKLUS AMÍG i < 10
```

```
    a[i] := RND 5 + 1
```

```
    i := i + 1
```

```
  CIKLUS_VÉGE
```

```
PROGRAM_VÉGE
```

Specifikáció
előfeltétel: (nincs)
utófeltétel: jön létre a 10 elemű a tömb amikben egyenletes eloszlású véletlen számok vannak 1..5 tartományban

Tömb, mint ismert hosszú sorozat

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

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

Specifikáció

előfeltétel: létezzen az a nevű egészekből álló tömb

utófeltétel: a sum változóban legyen az a tömb elemeinek összege

Tömb mint típuskonstrukció

- 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.

Példa kétdimenziós tömbre

```
PROGRAM mátrix
  VÁLTOZÓK:
    a : EGÉSZ[8][10],
    i, j: EGÉSZ

  i := 0
  CIKLUS AMÍG i < 8
    j := 0
    CIKLUS AMÍG j < 10
      a[i][j] := RND 5 + 1
      j := j + 1
    CIKLUS_VÉGE
    i := i + 1
  CIKLUS_VÉGE

PROGRAM_VÉGE
```

Specifikáció

előfeltétel: (nincs)

utófeltétel: jön létre az a 8x10 méretű mátrix, benne 1..5 egyenletes eloszlású véletlen számokkal

Tömbök jelentősége

- Olyan feladatoknál, ahol
 - több adatra van szükség, mintsem külön változókbán kényelmesen kezelni lehessen, de fix számú, és még beférünk 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 kényelmi okokból 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 használandó, ha ismert hosszú, többszöri írást vagy olvasást, vagy tetszőleges sorrendben feldolgozást igényel a feladat
- Lehet többdimenziós is

Rövid kitérő a számábrázoláshoz

- Az „EGÉSZ” illetve „VALÓS” típusok nevei azt a látszatot keltik, hogy a típusértékhalmaaz a teljes egész illetve valós szám tartományt fedí
- 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 $-2^{31} \dots +2^{31}-1$ tartományra tolták: $-2147483648 \dots 2147483647$
- 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 ekvivalens :
 $4294967295 + 1 = 0$

Kettes komplement

- Praktikus ha a csupa nulla memóriatartalom a nulla értéket jelenti.
- $00000000 = 11111111 + 1$ (bináris, 8 bites)
- \rightarrow az 11111111 minta jó választás a -1 értéknek, hisz $-1+1 = 0$
- Így már csak azt kell eldönteni, hogy melyik érték legyen tekintve pozitívnak
- Vegyük az első bitet előjelbitnek, ha 1, akkor negatív a szám
- $\rightarrow 01111111$ (127) a legnagyobb pozitív, és 10000000 (-128) a legkisebb negatív érték

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 nem vizsgáljuk programban egy valós típusú művelet eredményének egyenlőségét semmivel!

Rövid kitérő a számábrázoláshoz

- A valós számokat $X * 2^Y$ alakban tárolják, visszavezetve az egész számokra
- Ezt a számábrázolást lebegőpontos-nak hívják, mert a tizedesvessző az ábrázolandó értékes számjegyeken belül (vagy kívül) bárhova kerülhet:
 - Pl.: 1,23 vagy 12,3 vagy 123 (mindegyik ugyanazt a 3 értékes számjegyet tartalmaz)
 - Lásd $1.23 * 10^4$ forma, 123 a hasznos tizedesjegyek, és 4 pozícióval kell eltolni a tizedespontot

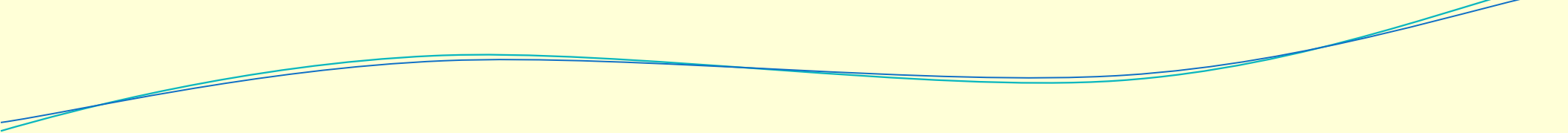
Rövid kitérő a számábrázoláshoz

Ennek következménye, hogy nem mindenhol egyformán sűrű az ábrázolás, ha 23 kettedesjegynél nagyobb a nagyságrendi különbség, akkor előfordulhat, hogy $A \neq 0$, de $A+B = B$

- pl. $1e20 + 1e-20 = 1e20$ ha nincs legalább 40 hasznos tizedesjegyre kapacitás
- Sok szám összegénél a kicsikkel érdemes kezdeni tehát, ami azt jelenti, hogy az összeadás nem asszociatív a lebegőpontos számoknál

Lebegőpontos bináris számok

- Mantissza: a hasznos kettedesjegyek
- Exponens: kettedespont eltolása
- 32 bites esetben 23 bit mantissza, 8 bit exponens, plusz 1 bit az előjelre
 - $2^{23} = 8388608 \rightarrow$ kb 6 tizedesjegy pontosság
- Speciális értékek: inf, -inf, nan
 - $1/0 = \text{inf}$, $1+\text{inf} = \text{inf}$, $-\text{inf}+\text{inf}=\text{nan}$, $0/0=\text{nan}$



Itt a vége a PLaNG képességeinek,
legközelebb a C++ alapoktól
folytatjuk