



Bevezetés a programozásba

4. Előadás Sorozatok, fájlok

Ismétlés

PROGRAM sorozatösszeadó

VÁLTOZÓK:

n, a, összeg, i: EGÉSZ

BE: n

i := 0

összeg := 0

CIKLUS AMÍG i < n

BE: a

összeg := összeg + a

i := i + 1

CIKLUS_VÉGE

KI: összeg

PROGRAM_VÉGE

Sorozatok

- Sorozatok: több (hasonló) adat érkezik a program számára
- A jellemző műveletek:
 - Inicializálás
 - Kezdeti értékek beállítása
 - Olvasás lehetővé tétele
(Pl.: A sorozat hosszának beolvasása (n))
 - Következő elem olvasása
 - Sorozat végének felismerése/lekérdezése

Sorozatok fajtái

- **Ismert hosszú sorozat**
 - Programba „beégetett”
 - Felhasználó által megadott
 - Érkező adatokból meghatározható
- **Végjeles sorozat**
 - Speciális érték/karakter jelöli a sorozat végét

Ismert hosszú sorozat

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

Változók: i, n : EGÉSZ, X : T

$n := A$ sorozat hossza

******(pl : BE: n)

$i := 0$

CIKLUS AMÍG $i < n$

 BE: X

X feldolgozása

$i := i+1$

CIKLUS_VÉGE

Ismert hosszú sorozatra példa

- Egész számok egy adott intervalluma $[-5 .. 10]$
- Egy meghatározott számú sor (n db elem)
- Táblázatoknál szokás először jelezni a méreteket
 - Kép, videó és hang formátumok sokszor ilyenek (.wav, .bmp, stb.)

Végjeles sorozat

- A sorozat értékkészletét megszorítva lehetővé válik, hogy speciális jelentésű értékeket használjunk. Pl.:
 - Csupa nem negatív elem van a sorozatban és az első negatív jelzi a sorozat végét
 - A sorozat elemeit summázzuk addig, amíg 0 nem érkezik
 - A mondatvégi írásjelig olvassuk a karaktereket.
- **Előnye:** könnyen bővíthető a sorozat
- **Hátránya:** nem használhatjuk az adott típus teljes értékkészletét

Végjeles sorozat feldolgozása

- **Jellegzetesség:** a **beolvasás után** még el kell dönteni, hogy **sorozatelemről** van-e szó, vagy a **végjelről**, ami **nem** része a sorozatnak!
- Tehát a **beolvasás** és a **feldolgozás között** kell lennie az **ellenőrzésnek**
- Az ellenőrzésnek a ciklusfeltételben kell lennie
- Következésképpen:
 - A **beolvasásnak** a **ciklus utolsó lépésének** kell lennie
 - A **ciklus előtt** is kell olvasni

Végjeles sorozat

Változók: $X : T$

BE: X

CIKLUS AMÍG *X nem végjel*
 X feldolgozása

BE: X

CIKLUS_VÉGE

Végjeles sorozat

Változók: $X : T$

BE: X ***olvasás ciklus előtt*

CIKLUS AMÍG X *nem végjel*
 X *feldolgozása*

BE: X ***az olvasás az utolsó lépés*

CIKLUS_VÉGE

„Előreolvasás”

Előreolvasás

- Általános technika: a ciklusfeltételhez szükséges adatokat a ciklus előtt elő kell állítani, különben „még nem kapott kezdeti értéket” hiba van
 - Ez akár az első néhány elem előreolvasását is jelentheti, ha a végjel úgy van megfogalmazva
 - Pl.: Addig olvassunk be számokat, amíg növekvő sorrendben érkeznek
- Hátránya, hogy a beolvasás többször szerepel a kódban

Végjeles sorozatra példa

- Bizonyos kódolásokban létezik „üzenet vége” karakter
 - Kiterjesztett Morse kódban is van befejezést jelző kód
- Kisebb programoknál, saját formátumoknál kedvelt forma az egyszerűség miatt
- Ősi DOS-ban a szöveges állományok végét is így jelezték
- Szöveges állományokban, részsorozatoknál bevett módszer üres sorral jelezni, hogy vége a sorozatnak, pl.: .srt mozifelirat formátumban

Fájlok

- A fájl névvel azonosított adattároló
- A programok meghatározó része használ fájlokat az adatok kezelésére, hiszen nem mindig közvetlenül adjuk meg az adatokat, illetve megeshet, hogy több adatforrást is kell használnunk egyszerre.
- **De:** a fájlok kezelése rendszertől, nyelvtől, kontextustól függ!

Fájlkezelés

- A fájlknál hasonló a működés, mint a szokásos bemenet és kimenet esetében:
- Több típusú adat is előfordulhat, tetszőleges sorrendben, csak figyelniünk kell arra, hogy a megfelelőt olvassuk ki.
- Ha az adatokat sorban olvassuk be, akkor **szekvenciális fájl**ról beszélünk, BevProgból csak ilyen fájlokkal foglalkozunk.
- Általában nem tudjuk, pontosan mennyi adatunk van a fájlban, de azt tudjuk, mikor van vagy volt vége az állománynak, ez lekérdezhető.

Fájlkezelési esetek

- Valamikor a bemenő adatokat olvassuk be a fájlból, valamikor a kimenő adatokat írjuk ki, ennek megfelelően megkülönböztetünk:
 - **bemeneti fájl:** csak beolvasásra szolgál, a tartalma nem változik meg használat közben, léteznie kell
 - **kimeneti fájl:** csak adatkírásra szolgál
 - *be- és kimeneti fájl:* felváltva írhatunk bele és olvashatunk belőle, ugrálhatunk benne, tipikusan bináris adatra (PLanG-ban nincs ilyen!)
- A **fájlok ugyanúgy változó**k a programban, amelyek típussal rendelkeznek

Logikai és fizikai fájl

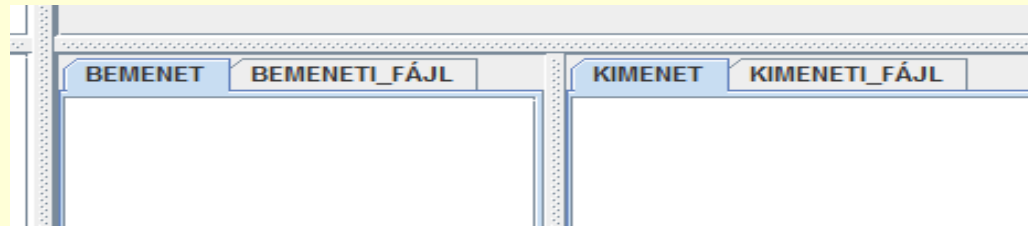
- Amikor deklarálunk egy fájl típusú változót, akkor létrehozunk egy logikai fájl objektumot, ami a változónk, amellyel a fájlra a programban hivatkozhatunk.
- A fájl változónk neve független attól hogy, melyik az a tényleges fájl, amit kezelni szeretnénk, ezért a változóhoz egy **fizikai fájl**nevet rendelünk. Ez a tényleges fájl neve, ami a lemezünkön van. (Pl.: /home/gelan/adatok.txt)
- A logikai fájlnevet a program során összekapcsoljuk a fizikaival, ez tipikusan a **megnyitás** művelet. Csak ettől a ponttól érhetjük a fizikai fájlt.

Logikai és fizikai fájl

- A logikai és fizikai fájlok szabadon kapcsolhatóak össze
 - Egy logikai fájlnevhez több fizikai név is tartozhat a program futása során, de egyszerre csak egy!
 - Egy fizikai név többször is előfordulhat a program során, más típusú vagy más nevű logikai fájlként.
 - Egyszerre több fájlból olvashatunk, ezek egymástól függetlenül lépnek előre
- A „kifájl” és „befájl” típusú változók szerepe számon tartani, hogy hol tartunk a hozzájuk tartozó fájlban
 - Ezért nem elég csak a fizikai fájl nevét írni olvasáskor
 - Ipari nyelveknél fájl írásakor a lezárás elmulasztása veszteséget okozhat: mindig minden fájlt zárjunk le, még ha nem is fordítási vagy futási hiba ennek elhagyása

Fájlok és a PLaNG

- PLaNG-ban a fájlok nem jelennek meg az operációs rendszer fájlrendszerében, **virtuális** fájlokról van szó
- De ettől eltekintve a használatuk ugyan olyan, mint az igazi fájloknak
- Használat előtt meg kell nyitni a fájlt, megadva a nevét, utána pedig illik lezárni (hogymás is hozzá férjen)
- Ha a PLaNG kódban megjelenik egy megnyitás, a bemenet és kimenet fülek bővülnek
- A „sima” **BE:** és **KI:** mintájára használható a fájl olvasás és írás, ugyanolyan működésűek



Fájlok és a PLaNG

- PLaNG-ban két fájl típust különböztetünk meg: **BEFÁJL**, **KIFÁJL**
 - `f` : **BEFÁJL** `**logikai fájl név`
- Egy fájlban bármilyen típusú adatot tárolhatunk, egyben többfélét is, bármilyen sorrendben, csak mindig a megfelelő típusra kell hivatkoznunk (ahogy megszokhattuk).
- A fizikai fájlnevet a **MEGNYIT** paranccsal kapcsoljuk a logikai névhez .
 - A fizikai fájlnek **most** nem kell megadnunk az elérését, hiszen nem tényleges fájlt fog hozzárendelni a környezet.
 - `MEGNYIT f : "adatok.txt"`
- A megnyitott fizikai fájlokat be is kell zárni, mert addig más program nem férhet hozzá (az operációs rendszer nem engedélyezi a párhuzamos hozzáférést), a **LEZÁR** paranccsal zárjuk be.
 - `LEZÁR f`

Fájlok és a PLanG

PROGRAM fájl

VÁLTOZÓK:

fb: BEFÁJL,

n: EGÉSZ

MEGNYIT fb: "olvasnivalo"

BE fb: n

KI: n

LEZÁR fb

PROGRAM_VÉGE

Fájlok és a PLanG

PROGRAM fájl
VÁLTOZÓK:

fb: BEFÁJL,
n: EGÉSZ

MEGNYIT fb: "olvasnivalo"
BE fb: n
KI: n
LEZÁR fb

PROGRAM_VÉGE

Specifikáció

előfeltétel: egy "olvasnivalo" nevű fájl elején legalább egy egész szám van

utófeltétel: a fájlban szereplő első számot írja ki

Fájlok és a PLaNG

- Lehetséges műveletek az olvasás és kiírás (a típustól függően csak egyik alkalmazható)
- Beolvasást változóba végezhetjük
 - BE f : a
 - ekkor a fájl aktuális következő eleme az **a** változóba kerül, a változó deklarációjától függő módon (pl. 123 egész, „123” szöveg, `1` karakter)
- Kiírásnál változót, értéket, vagy tetszőleges kifejezést is megadhatunk
 - KI f : a, ” : ”, b
 - bármit kiírhatunk, amit a kimeneti ablakra szoktunk
 - Minden szöveges tartalommal alakul kiírásnál

Fájlok és a PLanG

- A fájlok kezelése sokban hasonlít a végjeles sorozathoz a fájl végének kezelésében
- Addig olvasunk egy fájlból, amíg nem értük el a fájl végét jelző lezáró jelet, azaz sikertelen volt az érvényes adat beolvasás
 - Nyilvánvalóan ez esetben a sikertelen olvasás mellékhatásaként a beolvasott változó tartalma nem a sorozat része, általában invalid tartalom, tehát nem szabad feldolgozni, ahogy a végjelet sem szabad!
- → Itt is az **előreolvasás technikát** alkalmazzuk.

Fájlok és a PLanG

PROGRAM fájl-sorozatos
VÁLTOZÓK:

fb: BEFÁJL,
n: EGÉSZ

MEGNYIT fb: "olvasnivalo"

BE fb: n

CIKLUS AMÍG NEM VÉGE fb

KI: n, SV

BE fb: n

CIKLUS_VÉGE

LEZÁR fb

PROGRAM_VÉGE

Specifikáció

előfeltétel: egy "olvasnivalo" nevű fájlban csak egész számok vannak

utófeltétel: minden a fájlban szereplő összes számot külön sorba ír ki

Fájlok, műveletek összefoglalás

- **KIFÁJL, BEFÁJL**

- Típusok fájlok kezeléséhez, ezeken keresztül mozgatjuk az adatokat

- **Megnyit** f : "*fájlnev*"

- Fájlnev rendelése a KIFÁJL/BEFÁJL változóhoz

- **KI** f : X / **BE** f : X

- Írás/olvasás a megadott fájlba/fájlból

- **VÉGE** f

- Logikai kifejezés, mely jelzi, hogy vége van-e a fájlnek

- **LEZÁR** f

Homogén 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: a

CIKLUS_VÉGE

LEZÁR fb

Példa: Maximumkeresés fájlra

```
PROGRAM fajlosmaxker
VÁLTOZÓK:  fb: BEFÁJL,
           i, hol: EGÉSZ,
           a, max : VALÓS

MEGNYIT fb: "olvasnivalo"
BE fb: a
max := a
hol := 0
BE fb: a
i := 1
CIKLUS AMÍG NEM VÉGE fb
    HA max < a AKKOR
        max := a
        hol := i
    HA_VÉGE
    i := i + 1
    BE fb: a
CIKLUS_VÉGE
KI: hol, ".: " , max
LEZÁR fb
PROGRAM_VÉGE
```

Specifikáció

előfeltétel: egy "olvasnivalo" nevű fájlban csak egész számok vannak, legalább egy darab

utófeltétel: a fájlban levő számok közül kiírja a legnagyobbat és az indexét, 0-tól indexelve, "index .: érték" formátumban

Példa: Maximumkeresés fájlra

Változók:

i, hol: EGÉSZ,
a, max : T

i := 1
a := első elem
max := f(a)
hol := 1

CIKLUS AMÍG *nincs vége a sorozatnak*

a := következő elem
i := i + 1
HA max < f(a) AKKOR
max := f(a)
hol := i

HA_VÉGE

CIKLUS_VÉGE

PROGRAM fajlosmaxker

VÁLTOZÓK: fb: BEFÁJL,
i, hol: EGÉSZ,
a, max : VALÓS

MEGNYIT fb: "olvasnivalo"

BE fb: a ** első elem

max := a

hol := 0

BE fb: a ** felkészülés a "következő
i := 1 ** elem"-re a ciklusban

CIKLUS AMÍG NEM VÉGE fb

HA max < a AKKOR

max := a

hol := i

HA_VÉGE

i := i + 1

BE fb: a ** a "következő elem"

CIKLUS_VÉGE

KI: hol, ".: " , max

LEZÁR fb

PROGRAM_VÉGE

Formátumok

- Beolvasás legyen egyértelmű, meg kell adni mi után mi jön
- Technikai megkötések
 - A PLaNG szám beolvasásakor nem tud különbséget tenni szóközzel elválasztás és sorvégével elválasztás között, tehát az a forma, hogy az összetartozó számok egy sorban vannak, és a következő sor már megkülönböztetendő, nem jó formátum a PLaNG-ban
 - A szöveg beolvasása soronként történik, tehát két megkülönböztetendő szöveget ne írjunk egy sorba ha PLaNG-ban akarjuk feldolgozni

Példa formátumra

A feladat kezelni egy középiskolai osztályt egyik tanár szemszögéből:
van sok diák, mindegyiknek valahány jegye

- A fájlban két sor ír le egy diákot
 - az első sor a neve
- Mivel ha egy sorban lennének a jegyek is, a név beolvasásakor az is belekerülne a szövegbe, és az lenne a név, hogy „Gipsz Jakab 5 3 4 5 ...”
 - a következő sorban vannak a jegyek, és a végén -1
- Ez megtehető, mert a jegy eleve szűk értékkészletű, így az általánosság megszorítása nélkül vehetünk fel végjelet az egész számok közül
- Az is jó lenne, ha a jegyek számával kezdődne a sor

Példa formátumra

A feladat kezelni egy középiskolai osztályt és van sok diák, mindegyiknek valahány jegye

- A fájlban két sor ír le egy diákot
 - az első sor a neve
- Mivel ha egy sorban lennének a jegyek is, a név belekerülne a szövegbe, és az lenne a név, hogy "Gipsz Jakab 5 3 4 5 "
- a következő sorban vannak a jegyek, és a végén
- Ez megtehető, mert a jegy eleve szűk értékkel megszorítása nélkül vehetünk fel végjelet az egész sor végére
- Az is jó lenne, ha a jegyek számával kezdődne a

```
Gipsz Jakab
3 2 4 -1
Minta Panna
5 5 4 5 5 5 -1
Gémer Géza
1 1 2 1 -1
```

```
Gipsz Jakab
3 3 2 4
Minta Panna
6 5 5 4 5 5 5
Gémer Géza
4 1 1 2 1
```

Néhány elterjedt egyszerű formátum

- .sub mozifelirat:

```
{start-frame}{stop-frame}Text    {0}{25}Hello!
```

- .srt mozifelirat:

```
#subtitle 1
```

```
start-time --> end-time 00:02:45,561 --> 00:04:31,937
```

```
Text    Egyszer volt, hol nem volt...
```







- .ini konfigurációs fájl:

```
[fejléc] [section]
```

```
name = value    a = 220
```

- .csv: táblázatformátum, melyben vesszővel elválasztva vannak a mezők, a szöveges mezők „” jelek között szerepelnek

Összehasonlítás

	Ismert hosszú	Végjeles	Fájl
Teljes értékkészlet			
Beszúrással bővíthető			
Előre olvasást igényel			
Nyelvtől független	