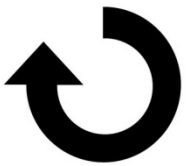
Bevezetés a programozásba

4. Előadás
Sorozatok, fájlok

https://wiki.itk.ppke.hu/twiki/bin/view/PPKE/BevezetSAProgramozSba201516

Elágazás

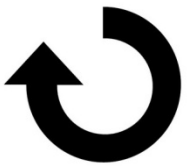


```
PROGRAM elágazás
      VÁLTOZÓK:
            a: EGÉSZ

      BE: a
      HA a > 0 AKKOR
            KI: „pozitív”
      KÜLÖNBEN
            KI: „nem pozitív”
      HA_VÉGE

PROGRAM_VÉGE
```

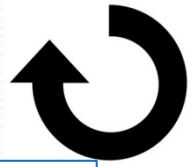
Ciklus



```
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
```

Összegzés tétele

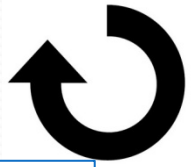


```
Változók: összeg, a : T

    összeg := 0
    CIKLUS AMÍG nincs vége a sorozatnak
        a := következő elem
        összeg := összeg ⊕ f(a)
    CIKLUS_VÉGE
```

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

Számlálás tétele

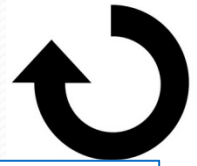


```
Változók: számláló: EGÉSZ,
          a : T

    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

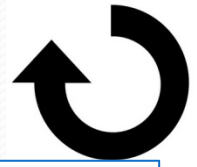


```
Változók:    hol, i: EGÉSZ,
             van: LOGIKAI,   a : T

      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 := 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
```

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

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, feldolgozása
  - 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 **ellenőrzés**nek
- Az ellenőrzésnek a ciklusfeltételben kell lennie
- **Következésképpen:**
  - A **beolvasás**nak a **ciklus utolsó lépés**ének kell lennie
  - A **ciklus előtt** is kell **olvas**ni

Végjeles sorozat

```
Változók: X : T

BE: X
CIKLUS AMÍG X nem végjel
        X feldolgozása
    BE: X
CIKLUS_VÉGE
```

Ciklus előtt olvasás!

Adat beolvasása

Következő adat a ciklus végén!

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

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
- Szöveges állományok végét is így jelzik
- 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 fájloknál hasonló a működés, mint a szokásos bemenet és kimenet esetében:
  - Több típusú adat is előfordulhat, tetszőleges sorrendben, csak figyelnünk kell arra, hogy a megfelelőt olvassuk ki.
  - Ha az adatokat sorban olvassuk be, akkor **szekvenciális fájl**ról beszélünk, mi csak ilyen fájlokkal foglalkozunk.
  - Általában nem tudjuk, pontosan mennyi adatunk van a fájlban, de azt tudjuk, mikor van vége az állománynak, ugyanis minden fájl végén egy **fájl vége jel** van (end of file: **EOF**), és le tudjuk kérdezni, hogy ezt sikerült-e beolvasni.

Fájlkezelési esetek

- Valamikor a bemenő adatokat olvassuk be a fájlból, valamikor a kimenő adatokat írjuk ki, ennek megfelelően megkülönböztetünk:
  - **bemeneti fájlt:** csak beolvasásra szolgál, a tartalma nem változik meg használat közben, léteznie kell
  - **kimeneti fájlt:** csak adatkiírásra szolgál
  - *be- és kimeneti fájl:* egyszerre írhatunk benne és olvashatunk belőle (PLanG-ban nincs ilyen!)
- A **fájlok ugyanúgy változók** a programban, amelyek típussal rendelkeznek.
  - Vannak nyelvek, ahol egy fájl csak egy típusú adatot tartalmazhat, és vannak olyanok, ahol a fájlban bármi lehet.
- **De:** A fájlműveletek külön utasításokat igényelnek, nem teljesen úgy dolgozunk velük, mint a többi változóval.

Logikai és fizikai fájl

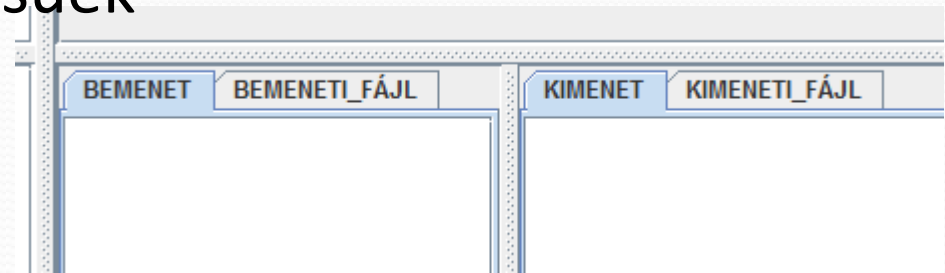
- Amikor deklarálunk egy fájl típusú változót, akkor létrehozunk egy **logikai fájl**nevet, ami a változó neve lesz, amellyel a fájlra a programban hivatkozhatunk.
- A logikai fájlnévből önmagában nem derül ki, melyik az a tényleges fájl, amit kezelni szeretnénk, ezért a változóhoz egy **fizikai fájl**nevet rendelünk. Ez a tényleges fájl neve, ami a lemezünkön van. (Pl.: /home/gelan/adatok.txt)
- **A logikai fájlnevet a program során összekapcsoljuk a fizikaival.** Csak ettől a ponttól használhatjuk a fájlt!

Fájlok száma

- Egy logikai fájlnévhez 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, de egyszerre csak egy változóhoz tartozhat!
- 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 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 (hogy más 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 fájltípust 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 fájlnak **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

- Lehetséges műveletek az olvasás és kiírás (a típustól függően csak egyik alkalmazható)
- Beolvasást változóba végezhetjük
  - `BE f : a`
  - ekkor a fájl aktuális eleme az ***a*** változóba kerül, a típusa lehet bármi, amit be szeretnénk olvasni
- Kiírásnál változót, értéket, vagy tetszőleges szöveg eredményű kifejezést is megadhatunk
  - `KI f : a, ” : ”, b`
  - ahol ***a*** és ***b*** szöveg típusú változók
  - bármit kiírhatunk, amit a kimeneti ablakra szoktunk, a sorvége jelet is

Fájlok és a PLanG

```
PROGRAM fájlos
VÁLTOZÓK:
        fb: BEFÁJL,
        n: EGÉSZ

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

PROGRAM_VÉGE
```

A logikai fájl (bemeneti típus)

Összekapcsolás

A fizikai fájl neve

Egy (EGY!) adat beolvasása a fájlból. *NEM az egész fájl!*

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, tehát nem szabad feldolgozni, ahogy a végjelet sem szabad!
- Itt is az **előreolvasás technikát** alkalmazzuk!

Fájlok és a PLanG

```
PROGRAM fájlos-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
```

A logikai fájl

A fizikai fájl

Ciklus előtt olvasás!

Következő adat ciklus végén!

A fájl teljes tartalma beolvasára kerül elemenként.

Fájlok, műveletek összefoglalás

- **KIFÁJL, BEFÁJL**
  - Típusok fájlok kezeléséhez, ezeken keresztül mozgatjuk az adatokat
- **Megnyit** *f: ”fájlnév”*
  - Fájlnév 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 fájlnak
- **LEZÁR** *f*

Fájl kezelése általában PLanG-ban

```
VÁLTOZÓK:
      fb: BEFÁJL,
      a: T

      MEGNYIT fb: "fájlnév"
      BE fb: a
      CIKLUS AMÍG NEM VÉGE fb
            a feldolgozása
            BE fb: a
      CIKLUS_VÉGE
      LEZÁR fb
```

Példa: Maximumkeresés fájlra

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

        MEGNYIT fb: "olvasnivalo"
        i := 1
        BE fb: a
        max := a
        hol := 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
```

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

        MEGNYIT fb: "olvasnivalo"
        i := 1
        BE fb: a
        max := a
        hol := 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
```

Formátumok

- Beolvasás egyértelműségét segíti
  - Különösen különböző típusú adatok esetén fontos
- 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

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

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	✓	✓	