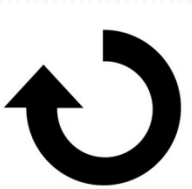
Bevezetés a programozásba

3. Előadás
Algoritmusok, programozási tételek

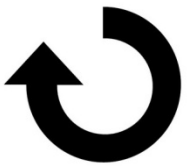
https://wiki.itk.ppke.hu/twiki/bin/view/PPKE/BevezetSAProgramozSba201516

Specifikáció



- A specifikáció lényege, hogy a feladatot a lehető legprecízebben megfogalmazzuk.
- **Előfeltétel:** Azaz milyen bemeneteket kell minden esetben lekezelnie a programnak, milyen körülmények között követelünk helyes működést.
- **Utófeltétel:** A meghatározott bemenetekre milyen válaszokat/eredményeket kell adnia a programnak, mit várunk a kimenettől, mi a kapcsolat a bemenet és kimenet között.
- Ha a megadott **feltételek** teljesülnek akkor a program megoldja a feladatot.
- A specifikáció nem utasításokból áll, hanem feltételekből, mert a feladatot írja le, nem a programot.
  (A specifikáció a MIT, a program a HOGYAN)

Elágazás

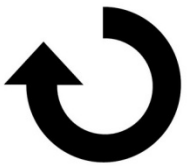


```
PROGRAM elágazás
      VÁLTOZÓK:
            a: EGÉSZ

      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
```

Programkonstrukciók összefoglalás

- Elágazas kell, ha más kódra van szükség egyes esetekben
- Ciklus kell, ha ismételni kell lépéseket
- Okos megbízható matematikusok bebizonyították: Minden algoritmikusan megoldható probléma megoldható szekvencia, elágazás és ciklus segítségével.

- Tehát vége is a féléves anyagnak…

- Érdemes lesz azért bejárni …

Algoritmus

- Sokféle definíció van: *„Az algoritmus egy megengedett lépésekből álló módszer, utasítássorozat, részletes útmutatás, recept, amely alkalmas arra, hogy valamilyen felmerülő problémára megoldást adjon.”*
- Praktikusan elágazásból, ciklusból, értékadásból és I/O műveletekből álló helyes program elve, amely egy adott feladatot megold
  - A fogalom ennél absztraktabb
- Néhány algoritmus általános iskolai anyag, mint a „papíron szorzás”
- Néhány száz algoritmusnak neve is van
- **Miért jó ismerni algoritmusokat?** Ha valaki már talált megoldást egy problémára, nekünk ne kelljen újra kitalálni!

Program - Algoritmus

- A program egy **konkrét** feladatot old meg, **konkrét** formátumban megadott bemenettel és kimenettel.
  - Valamint a programnak figyelembe kell vennie a rendelkezésre álló erőforrásokat.
- Az algoritmus a megoldás elve, formátumtól, típusoktól amennyire lehet, függetlenül.
  - Azaz az algoritmus egy általános megfogalmazás.

A továbbiakban megnézzünk konkrét programokat, majd azokból absztrakcióval kapott általános algoritmusokat

Sorozat

- A mai előadás minden sorozatának formátuma a következő:
  - Az első bemenet egy szám, amely az érkező sorozat hosszát jelenti (azaz a bemeneten érkező elemek számát)
  - Ezután következnek a sorozatelemek
  - Például:

BEMENT:

5 ** *azaz 5 elemű sorozatot fogok megadni*

1 3 5 7 9 ** *maga az 5 elemű sorozat*

Sorozat minden elemét „lemásoló”

```
PROGRAM sorozat_elemek_masolasa
      VÁLTOZÓK:
            n, i: EGÉSZ,
            a: VALÓS

      BE: n
      i := 0
      CIKLUS AMÍG i < n
            BE: a
            KI: a, SV
            i := i + 1
      CIKLUS_VÉGE

PROGRAM_VÉGE
```

Összegzés PLanG-ban

```
PROGRAM sorozatösszeadó
        VÁLTOZÓK:
                n, i: EGÉSZ,
                a, összeg: VALÓS

        BE: n
        i := 0
        összeg := 0
        CIKLUS AMÍG i < n
                BE: a
                összeg := összeg + a
                i := i + 1
        CIKLUS_VÉGE
        KI: összeg
PROGRAM_VÉGE
```

n: ennyi elem van a sorozatban

a: az aktuális elem, amivel éppen dolgozunk

i: az ennyiedik elemnél tartunk

összeg: az eddig feldolgozott elemek összege

Minden sorozatelem pontosan egy alkalommal szerepel *a*-ban!

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.)

Összegzés tétele

- Mire jó?
  - Kumulatív (halmozó) feladatok sorozatokon
- Az algoritmus melyik része cserélhető?
  - Kezdőérték
  - Művelet, függvény
- Példák
  - Átlag
  - Faktoriális
  - Négyzetösszeg

Számlálás PLanG-ban

```
PROGRAM számlálás
        VÁLTOZÓK:
                n, i, számláló, a: EGÉSZ

        BE: n
        i := 0
        számláló := 0
        CIKLUS AMÍG i < n
                BE: a
                HA a MOD 2 = 0 AKKOR
                        számláló := számláló + 1
                HA_VÉGE
                i := i + 1
        CIKLUS_VÉGE
        KI: számláló
PROGRAM_VÉGE
```

A *számláló* az eddig látott adott feltételnek megfelelő elemek száma.

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.

Számlálás tétele

- Mire jó?
  - „mennyi?”, „hány?” feladatok sorozatokon
- Az algoritmus melyik része cserélhető?
  - Feltétel
  - Növelő függvény
- Példák
  - Osztók száma
  - Szavak száma egy szövegben

Lineáris keresés PLanG-ban

```
PROGRAM lineáris_keresés
        VÁLTOZÓK:
                n, i, a, hol: EGÉSZ,
                van: LOGIKAI

        BE: n
        i := 0
        van := HAMIS
        hol := 0
        CIKLUS AMÍG i < n ÉS NEM van
                BE: a
                i := i + 1
                HA a MOD 2 = 0 AKKOR
                        van := IGAZ
                        hol := i
                HA_VÉGE
        CIKLUS_VÉGE
        KI: van
        HA van AKKOR
                KI: hol
        HA_VÉGE
PROGRAM_VÉGE
```

Két eredmény van: van-e megfelelő, és (ha igen) hol

Az *i* futóindex-et előbb inkrementáljuk, hogy az elem sorozatban betöltött „valódi” sorszámát kapjuk.

A *van* az eddig látott elemek között találtunk-e adott feltételnek megfelelőt?

A *hol* az elsőnek megtalált elem indexe. (ha nincs megfelelő, akkor nem érdekes, mi az éréke)

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.

Lineáris keresés tétele

- Mire jó?
  - „van-e?”, „és hányadik, ha van?” feladatok sorozatokon
- Az algoritmus melyik része cserélhető?
  - Feltétel
- Példák
  - Szerepel-e egy bizonyos érték egy sorozatban és hol
  - Prím-e egy szám (van-e osztója?)

Eldöntés és kiválasztás tétele

- A **Lineáris keresés tétele** igazából két egyszerűbb tételből áll össze:
  - **Eldöntés tétele:**
    - Van-e a sorozatnak olyan eleme amelyre igaz a *feltétel()*?
    - Változat: Igaz-e a *feltétel()* a sorozat összes elemére?
  - **Kiválasztás tétele:**
    - Hányadik az a sorozat elem amelyre igaz a *feltétel()*?
    - Itt egy index-re kérdezünk rá. Szokás, hogy ilyenkor a kezdeti *hol* változót **-1**-re állítjuk, amely azt jelenti, hogy még nem volt találat, hisz érvénytelen pozíciót jelöl.

Maximum keresés PLanG-ban

```
PROGRAM maximumkeresés
        VÁLTOZÓK:
                n, i, hol: EGÉSZ,
                a, max: VALÓS

        BE: n
        i := 1
        BE: a
        max := ???
        hol := ???
        CIKLUS AMÍG i < n
                BE: a
                i := i + 1
                HA max < a AKKOR
                        max := a
                        hol := i
                HA_VÉGE
        CIKLUS_VÉGE
        KI: max, SV, hol
PROGRAM_VÉGE
```

A *max* és *hol* az eddig látott maximális elemre vonatkozik. Ez az elején mi legyen?

Két eredmény van: a maximális értékű elem és indexe. (Mindig létezik, ha legalább egy elemű a sorozat

Maximum keresés PLanG-ban

```
PROGRAM maximumkeresés
        VÁLTOZÓK:
                n, i, hol: EGÉSZ,
                a, max: VALÓS

        BE: n
        i := 1
        BE: a
        max := a
        hol := 1
        CIKLUS AMÍG i < n
                BE: a
                i := i + 1
                HA max < a AKKOR
                        max := a
                        hol := i
                HA_VÉGE
        CIKLUS_VÉGE
        KI: max, SV, hol
PROGRAM_VÉGE
```

Két eredmény van: a maximális értékű elem és indexe. (Mindig létezik, ha legalább egy elemű a sorozat

A *max* és *hol* az eddig látott maximális elemre vonatkozik. Ez az elején az első elem -> i:=1

Az *i* futóindex-et előbb inkrementáljuk, hogy az elem sorozatban betöltött „valódi” sorszámát kapjuk.

Ha találtunk az eddigi maximumnál egy nagyobb elemet, akkor lecseréljük az újra

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.

„Minimum” keresés tétele

```
Változók:     i, hol: EGÉSZ,
              a, min : T

      i := 1
      a := első elem
      min := f(a)
      hol := 1
      CIKLUS AMÍG nincs vége a sorozatnak
            a := következő elem
            i := i + 1
            HA min > f(a) AKKOR
                  min := f(a)
                  hol := i
            HA_VÉGE
      CIKLUS_VÉGE
```

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

Maximum keresés tétele

- Mire jó?
  - „Mi / Mennyi a leg… ?” feladatok sorozatokon
- Az algoritmus melyik része cserélhető?
  - Feltétel, reláció
  - Függvény
- Példák
  - Maximum, minimum keresés
  - Zárójelek mélysége szövegben

Elemenként feldolgozás

- Sok adat, kevés változó
- Egy feladat elemenként feldolgozható, ha
  - Meg lehet úgy oldani, hogy a hosszú adatsorból egyszerre csak kevésre van szükség
  - Csak egyszer kell végignézni mindegyiket
- Sok hasznos algoritmus tartozik ide
  - Az előbb látottak
  - Válogatás (pl. sorozatból a párosakat írjuk ki)
  - Összefésülés (két sorozat egyesítése)
  - Metszetképzés, unióképzés

Az eredmény ezeknél egy kicsit más: egy sorozathoz egy sorozatot rendelünk hozzá.

Elemenként feldolgozás

- Példa egyéb elemenként feldolgozható feladatra:
  - Sorozat értékkészletét befoglaló intervallum (min – max)
  - Sorozat második legnagyobb értéke
  - Van-e még egy olyan elem, amely megegyezik az első elemmel?
- Példa feladatra, amely **NEM** elemenként feldolgozható:
  - Sorozat rendezése (növekvő/csökkenő sorrendbe)
  - Sorozat mediánja (középértéke)
  - Sorozat elemeinek a szórása (átlagtól való eltérés négyzetes átlaga)
  - Van-e két egyforma elem a sorozatban

Tételek kombinációi

- Feladat: egy intervallumban számoljuk meg a prímszámokat!
- Számlálás tétel kell hozzá
- De ez még nem elég: nincsen „prím-e?” műveletünk
- Ezért a prímdöntést lineáris kereséssel (van-e valódi osztója?) dönthetjük el a számlálás belsejében
- -> „állapottér transzformáció”, úgy teszünk a számlálásban, mintha logikai értékeket olvasnánk, amelyeket egy másik tétellel állítunk elő

Prímek száma n elemű sorozatban

```
Változók:    n, i, a, sz: EGÉSZ

  BE: n
  i := 0
  sz := 0
  CIKLUS AMÍG i < n
    BE: a
    HA prím(a) AKKOR
      sz:= sz + 1
    HA_VÉGE
    i := i + 1
  CIKLUS_VÉGE
  KI: sz
```

Ahol *a* a vizsgálandó aktuális szám,
a *prím( )* azon függvény, amely meghatározza, hogy az adott szám prím-e.

Prímek száma n elemű sorozatban

```
Változók:

  BE: n
  i := 0
  sz := 0
  CIKLUS AMÍG
    BE: a
    HA prím(a
      sz:= sz
    HA_VÉGE
    i := i +
  CIKLUS_VÉGE
  KI: sz
```

```
PROGRAM van_e_valódi_osztó
    VÁLTOZÓK:      osztók, a: EGÉSZ,
                   van: LOGIKAI

    BE: a
    van := HAMIS
    osztók := 2
    CIKLUS AMÍG osztók < a ÉS NEM van
         HA a MOD osztók = 0 AKKOR
            van := igaz
         HA_VÉGE
         osztók := osztók + 1
    CIKLUS_VÉGE
    KI: van
PROGRAM_VÉGE
```

Ahol az *osztók* segítségével vizsgálom végig az *a* számot, hogy 2-től n-1-ig osztható-e maradék nélkül valamelyikkel.

```
PROGRAM tétel_kombináció
  Változók: n, i, a, sz, osztók: EGÉSZ,
            van: LOGIKAI

  BE: n
  i := 0
  sz := 0
  CIKLUS AMÍG i < n
    BE: a
    van := HAMIS
    osztók := 2
    CIKLUS AMÍG osztók < a ÉS NEM van
           HA a MOD osztók = 0 AKKOR
                van := igaz
           HA_VÉGE
           osztók := osztók + 1
    CIKLUS_VÉGE
    HA NEM VAN AKKOR
        sz:= sz + 1
    HA_VÉGE
    i := i + 1
  CIKLUS_VÉGE
  KI: sz

PROGRAM_VÉGE
```

```
Változók:   n, i, a, sz: EGÉSZ

  BE: n
  i := 0
  sz := 0
  CIKLUS AMÍG i < n
    BE: a
    HA prím(a) AKKOR
           sz:= sz + 1
    HA_VÉGE
    i := i + 1
  CIKLUS_VÉGE
  KI: sz
```

```
PROGRAM van_e_valódi_osztó
    VÁLTOZÓK:   osztók, a: EGÉSZ,
                van: LOGIKAI

    BE := a
    van := HAMIS
    osztók := 2
    CIKLUS AMÍG osztók < a ÉS NEM van
              HA a MOD osztók = 0 AKKOR
                    van := igaz
              HA_VÉGE
              osztók := osztók + 1
    CIKLUS_VÉGE
    KI: van
PROGRAM_VÉGE
```