Programozási tételek

Ez a file önmagában nem biztosít elégséges tudást a tárgy teljesítéséhez! **Nem helyettesíti az előadást, csak az ott elhangzottakból egy kivonatot ad!** Mindenképpen az órai anyagokkal együtt értelmezendő, de ha azok mennek, akkor segít összefoglalni kicsit.

Egyre kevesebbet írok pluszba az egyes tételekről, nyilván sok elméleti lépés átvihető egyikről a másikra.

Figyelj rá végig, hogy itt 1-től N-ig indexelünk, nem 0-tól (N-1)-ig! Sok helyen emiatt i <= N helyett 0-tól indexelő nyelvben i < N kell, és i := 1 helyett i := 0.

Nem deklarálgatok N-eket meg i-ket, ha magától értetődő.

Egyszerű tételek (4. előadás)

Összegzés

```
1   Összeg: ℕ
2   Értékek: Tömb[1..N | ℕ]
    ------------------------
3   Összeg := 0
4   Ciklus i=1..N:
5       Összeg := Összeg + Értékek[i]
6   Ciklus vége
```

Az `Összeg` egyébként **bármilyen** típus lehet (1. sor), amire értelmezett valamilyen művelet, amit összegzés jelleggel akarunk elvégezni. Ekkor arra kell figyelni, hogy a 2. sorban az `Értékek` tömb elemeinek típusát is erre javítsuk, a 3. sorban az összeg értéke a művelet nulleleme legyen és az 5. sorban a műveletet írjuk át a megfelelőre. Például, ha szövegeket akarunk összefűzni, akkor az `Összeg` típusa string, az `Értékek` típusa Tömb[1..N | string], a 3 sorban a kezdőérték az üres string ('') az 5. sorban a művelet pedig string összefűzés (ami jelölhető plusz jellel is). Összegzéssel lehet pl. valós számokat összeszorozni is, ekkor a típus a valós számok, a tömbelemek is valósak, az összeg 1-ről indul, a művelet pedig a szorzás. Fontos, hogy a művelet zárt legyen a típusra! Kivonás művelettel nem tudunk összegezni természetes számokon, csak egészeken!

Megszámolás

```
Db: ℕ
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
------------------------
Db := 0
Ciklus i=1..N:
    Ha Feltétel(Értékek[i]):
        Db := Db + 1
    Elágazás vége
Ciklus vége
```

Megint csak, a tömbértékek típusa változhat, de ez esetben a `Feltétel` függvény értelmezési tartományát is megfelelően változtatni kell. A függvény értékkészlete mindig logikai (igaz vagy hamis), hiszen egy elágazás feltételébe írjuk bele.

Maximum-kiválasztás

Indexes megoldás

```
MaxIndex : ℕ
Értékek: Tömb[1..N | ℝ]
------------------------
MaxIndex := 1
Ciklus i=2..N:
    Ha Értékek[i] > Értékek[MaxIndex]:
        MaxIndex := i
    Elágazás vége
Ciklus vége
```

Ha más típussal akarunk dolgozni, csak az `Értékek` tömb elemeinek típusát kell változtatni, illetve arra figyelni, hogy a feltételben adott reláció értelmezve legyen. Ez a reláció bármilyen rendezés lehet, tehát kereshetünk többek közt minimumot is vele.

Maxértékes megoldás

```
MaxÉrték : ℝ
Értékek: Tömb[1..N | ℝ]
------------------------
MaxÉrték := Értékek[1]
Ciklus i=2..N:
    Ha Értékek[i] > MaxÉrték:
        MaxÉrték := Értékek[i]
    Elágazás vége
Ciklus vége
```

Itt változtatni kell a `MaxÉrték` típusát is, ha más típussal dolgozunk.

Eldöntés

Alap eset

Van olyan elem, amire teljesül a feltétel?

```
Van: L
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
```

```
------------------------
Van := hamis
i := 1
Ciklus amíg i <= N és !Van:
    Van := Feltétel(Értékek[i])
    i := i + 1
Ciklus vége
```

A `Feltétel` az `Értékek` elemeinek típusából rendel logikaiba. Ez bármilyen típus lehet, nem feltétlen kell valós számnak lennie, de egyezzen meg a kettő!

Fordított eset

Minden elemre teljesül a feltétel?

```
Mind: L
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
------------------------
Mind := igaz
i := 1
Ciklus amíg i <= N és Mind:
    Mind := Feltétel(Értékek[i])
    i := i + 1
Ciklus vége
```

Kiválasztás

Adjunk meg egy olyan elemet, amire teljesül a feltétel. **BIZTOSAN VAN ILYEN ELEM** -> ezért is hívjuk a korábbi tételt maximum kiválasztásnak, mert biztosan van maximum.

```
Elem: ℝ
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
------------------------
i := 0
Ciklus amíg nem Feltétel(Értékek[i])
    i := i + 1
Ciklus vége
Elem := Értékek[i]
```

`Elem`, `Értékek` elemei és `Feltétel` értelmezési tartománya azonos típus.

Keresés

Van olyan elem, amire teljesül a feltétel? Ha igen, melyik az? Nem követelmény, hogy legyen ilyen elem -> ezért is hívjuk a majdani összetett algoritmust feltételes maximumkeresésnek, mert nem biztos, hogy van olyan elem, amire teljesül a feltétel, így nem is biztos, hogy van maximum.

Van-os megoldás

```
Elem: ℝ
Van: L
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
------------------------
i := 1
Ciklus amíg i <= N és nem Van
    Van := Feltétel(Értékek[i])
    i := i + 1
Ciklus vége
Ha Van:
    Elem := Értékek[i-1]
Elágazás vége
```

`Elem`, `Értékek` elemei és `Feltétel` értelmezési tartománya azonos típus.

Indexes megoldás

```
Elem: ℝ
Értékek: Tömb[1..N | ℝ]
Feltétel: ℝ → L
------------------------
i := 1
Ciklus amíg i <= N és nem Feltétel(Értékek[i])
    i := i + 1
Ciklus vége
Ha i <= N:
    Elem := Értékek[i]
Elágazás vége
```

Összetett tételek (5. előadás)

Innentől *X* és *Y* típusokat fognak jelölni, bármi behelyettesíthető a helyükre (ℕ, ℝ, L, Ember, Képviselő, Tömb stb.), csak felesleges lenne mindenhol kifejteni, hogy mi cserélhető, mi azonos stb.

Természetes mindegyik megoldható dinamikus tömbökkel (vektor és push_back), ekkor nem kell előre megadni az eredmény tömböknek méretet.

Másolás (függvényszámítás)

```
Értékek: Tömb[1..N | X]
Eredmények: Tömb[1..N | Y]
Függvény: X → Y
```

```
------------------------
Ciklus i=1..N:
    Eredmények[i] := Függvény(Értékek[i])
Ciklus vége
```

A `Függvény` lehet az identitás is, ekkor sima másolás.

Kiválogatás

```
1   Értékek: Tömb[1..N | X]
2   Eredmények: Tömb[1..N | X]
3   Db: ℕ
4   Feltétel: X → L
    ------------------------
5   Db := 0
6   Ciklus i=1..N:
7       Ha Feltétel(Értékek[i]):
8           Db := Db + 1
9           Eredmények[Db] := Értékek[i]
10      Elágazás vége
11  Ciklus vége
```

Ha nullától indexelnénk, `Db` továbbra is 0-ról indulna, csak megcserélnénk a 8. és 9. sorokat.

Szétválogatás

Itt a legáltalánosabbat írom le, vannak specifikusabb megoldások az előadásban.

```
Értékek: Tömb[1..N | X]
Eredmények1: Tömb[1..N | X]
Db1: ℕ
Feltétel1: X → L
Eredmények2: Tömb[1..N | X]
Db2: ℕ
Feltétel2: X → L
...
EredményekM: Tömb[1..N | X]
DbM : ℕ
FeltételM: X → L
------------------------
Db1, Db2 ... DbM := 0
Ciklus i=1..N:
    Ha Feltétel1(Értékek[i]):
        Db1 := Db1 + 1
        Eredmények1[Db1] := Értékek[i]
    Egyébként ha Feltétel2(Értékek[i]):
        Db2 := Db2 + 1
        Eredmények2[Db2] := Értékek[i]
    ...
```

```
    Egyébként ha FeltételM(Értékek[i]):
        DbM := DbM + 1
        Eredmények[DbM] := Értékek[i]
    Elágazás vége
Ciklus vége
```

`Feltétel1`, `Feltétel2`, ..., `FeltételM` kölcsönösen kizárják egymást minden kombinációban.

Metszet

Inline

```
Értékek1: Tömb[1..N | X]
Értékek2: Tömb[1..M | X]
Metszet: Tömb[1..N+M | X]
Db: ℕ
BenneVan: L
------------------------
Db := 0
Ciklus i=1..N:
   BenneVan := hamis
   j := 1
   Ciklus amíg j <= M és nem BenneVan:
       BenneVan := Értékek1[i] = Értékek2[j]
       j := j + 1
   Ciklus vége
   Ha BenneVan:
       Db := Db + 1
       Metszet[Db] := Értékek1[i]
   Elágazás vége
Ciklus vége
```

Az i-s ciklus megy végig az `Értékek1` tömbön, a j-s ciklus pedig `Értékek2`-n ellenőriz.

Függvénnyel

```
BenneVan(Elem: Y, Tömb: [1..K | Y]):
    Van := hamis
    i := 1
    Amíg i <= K és nem Van:
        Van := Elem = Tömb[i]
        i := i + 1
    Ciklus vége
    Return Van
BenneVan vége
```

```
 Értékek1: Tömb[1..N | X]
 Értékek2: Tömb[1..M | X]
 Metszet: Tömb[1..N+M | X]
 Db: ℕ
 BenneVan: X → L
 ------------------------
 Db := 0
 Ciklus i=1..N:
    Ha BenneVan(Értékek1[i], Értékek2):
        Db := Db + 1
        Metszet[Db] := Értékek1[i]
    Elágazás vége
Ciklus vége
```

A BenneVan függvény bármilyen típusra meg lehet írva, a lényeg, hogy legyen olyan is, ahol X-re értelmezett.

Unió

Inline

```
 Értékek1: Tömb[1..N | X]
 Értékek2: Tömb[1..M | X]
 Unio: Tömb[1..N+M | X]
 Db: ℕ
 BenneVan: L
 ------------------------
 Db := M
 Unio[1..M] := Értékek2
 Ciklus i=1..N:
    BenneVan := hamis
    j := 1
    Ciklus amíg j <= M és nem BenneVan:
        BenneVan := Értékek1[i] = Értékek2[j]
        j := j + 1
    Ciklus vége
    Ha nem BenneVan:
        Db := Db + 1
        Unio[Db] := Értékek1[i]
    Elágazás vége
Ciklus vége
```

Függvénnyel

```
BenneVan(Elem: Y, Tömb: [1..K | Y]):
    Van := hamis
    i := 1
    Amíg i <= K és nem Van:
        Van := Elem = Tömb[i]
```

```
        i := i + 1
    Ciklus vége
    Return Van
BenneVan vége
```

```
 Értékek1: Tömb[1..N | X]
 Értékek2: Tömb[1..M | X]
 Unio: Tömb[1..N+M | X]
 Db: ℕ
 BenneVan: X → L
 ------------------------
 Db := M
 Unio[1..M] := Értékek2
 Ciklus i=1..N:
    Ha nem BenneVan(Értékek1[i], Értékek2):
        Db := Db + 1
        Unio[Db] := Érték[i]
    Elágazás vége
Ciklus vége
```