ELTE | IK

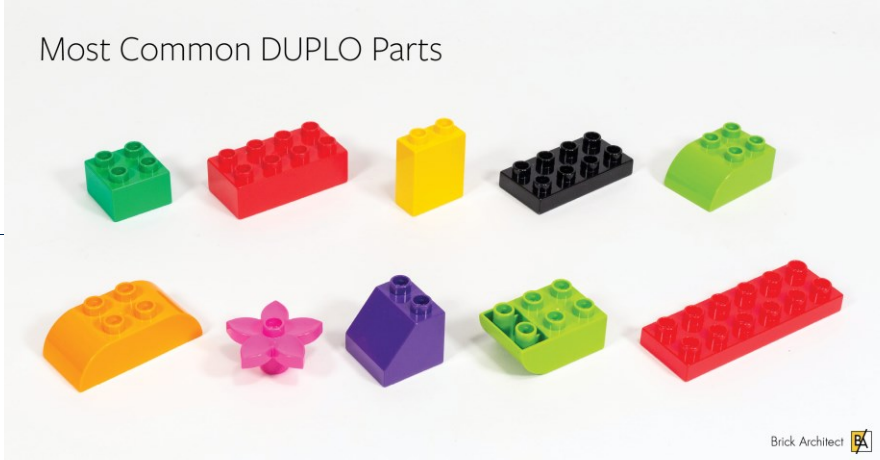
PROGRAMOZÁS

Programozási minták általánosítása

Horváth Győző

Ismétlés

Programozási minták



1. Összegzés
2. Megszámolás
3. Maximumkiválasztás
   a. Minimumkiválasztás
4. Feltételes maximumkeresés
5. Keresés
6. Eldöntés
   a. Mind eldöntés
7. Kiválasztás
8. Másolás
9. Kiválogatás

Összegzés általánosítása

„játék” a programozási mintákkal

Összegzés

Feladatok:

1. Ismerjük egy ember havi bevételeit és kiadásait. Adjuk meg, hogy év végére **mennyivel** nőtt a vagyona!
2. Ismerjük egy autóversenyző körönkénti idejét. Adjuk meg az **átlag**körének idejét!
3. Adjuk meg az n számhoz az n **faktoriális** értékét!
4. Ismerjük egy iskola szakkö szakkörönként. Adjuk meg
5. Ismerünk N szót. Adjuk meg a belőlük összeállított mondatot!

Mi bennük a közös?

n szám összegét kell kiszámolni!

Összegzés

Feladatok:

1. Ismerjük egy ember havi bevételeit és kiadásait. Adjuk meg, hogy év végére **mennyivel** nőtt a vagyona!
2. Ismerjük egy autóversenyző körönkénti idejét. Adjuk meg az **átlag**körének idejét!
3. Adjuk meg az n számhoz az n **faktoriális** értékét!
4. Ismerjük egy iskola szakköreire járó tanulóit, szakkörönként. Adjuk meg, kik járnak szakkörre!
5. Ismerünk N szót. Adjuk meg a belőlük összeállított mondatot!

Összegzés

Mi bennük a közös?

n „valami” szorzatát, unióját, összefűzését kell kiszámolni! Ugyanúgy, mint összegzés esetén, de ebben az esetben összeadás helyett más műveletet kell alkalmazni az egyes „valamik” között.

Feladatok:

1. Adjuk meg az n számhoz az n **faktoriális** értékét!

   ```
   f = 1*2*3*…*n = ∏(i=1..n, i)
   ```

2. Ismerjük egy iskola szakköreire járó tanulóit, szakkörönként. Adjuk meg, kik járnak szakkörre!

   ```
   szakkörösök = Ø ∪ nevek1 ∪ nevek2 … ∪ nevekn =
   ⋃(i=1..n, nevek[i])
   ```

3. Ismerünk N szót. Adjuk meg a belőlük összeállított mondatot!

   ```
   mondat = "" + szó1 + szó2 + szó3 + ... =
   +(i=1..n, szó[i])
   ```

Összegzés

Közös tulajdonságok:

- zárt intervallum (e..u)
- intervallum elemeihez rendelt érték (f(i))
- **számítási művelet**, amivel az értékeket összesíteni lehetett (+, *, ∪, összefűzés)
  - Ehhez a művelethez mindig tartozik egy olyan érték, amelyikkel bármilyen másik értékkel elvégezve a műveletet, a másik értéket kapjuk. = **nullelem**
  - Pl. 0+s=s, 1*s=s, ∅∪s=s, stb.
  - asszociatív művelet

A számítási művelet operandusai ugyanabból a H halmazból kerülnek ki.

Példa

Adjuk meg az n számhoz az n faktoriális értékét

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u,f(i))
Uf: s=SZUMMA(i=e..u,f(i),0,+)
```

n faktoriális

```
Be: n∈N
Ki: f∈N
Ef: -
Uf: f=n!=1*2*3*...*n=∏(i=1..n, i)
Uf: f=SZUMMA(i=1..n,i,1,*)
```

```
s      ~ f
e..u   ~ 1..n
f(i)   ~ i
0,+    ~ 1,*
```

```
s:=0
i=e..u
    s:=s+f(i)
```

```
Változó
  i:Egész
```

```
f:=1
i=1..n
    f:=f*i
```

```
Változó
  i:Egész
```

Általános összegzés sablon

Művelet neve	Operátor	Nul
összeadás	+	0
szorzás	*	1
szövegösszefűzés	+	""

Művelet neve	Operátor	Nullelem
unió	∪	∅
logikai és	és	igaz
logikai vagy	vagy	hamis
tömbösszefűzés	hozzáfűz	üres tömb

Feladat

Adott az egész számok egy [e..u] intervalluma és egy f:[e..u]→H függvény. A H halmaz elemein értelmezett egy asszociatív, baloldali nulla elemmel rendelkező művelet, amit most összeadásnak nevezünk és +-szal jelöljük. Határozzuk meg az f függvény [e..u] intervallumon felvett értékeinek az összegét, azaz a $\sum_{i=e}^{u} f(i)$ kifejezés értékét! (e>u esetén ennek az értéke definíció szerint a nulla elem)

Specifikáció

```
Be: e∈Z, u∈Z
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u,f(i),0,+)
```

Algoritmus

```
s:=0
i=e..u
    s:=s+f(i)
```

```
Változó
  i:Egész
```

Általános összegzés
megszámolás

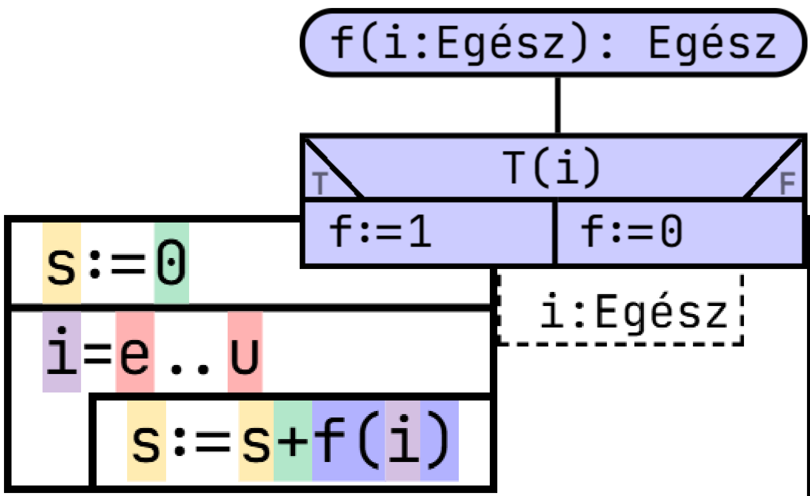
Feladatsablon

Be: e∈Z, u∈Z

Ki: s∈H

Ef: -

Uf: s=**SZUMMA**(i=e..u,f(i),0,+)



Megszámolás

Be: e∈Z, u∈Z

Ki: db∈N

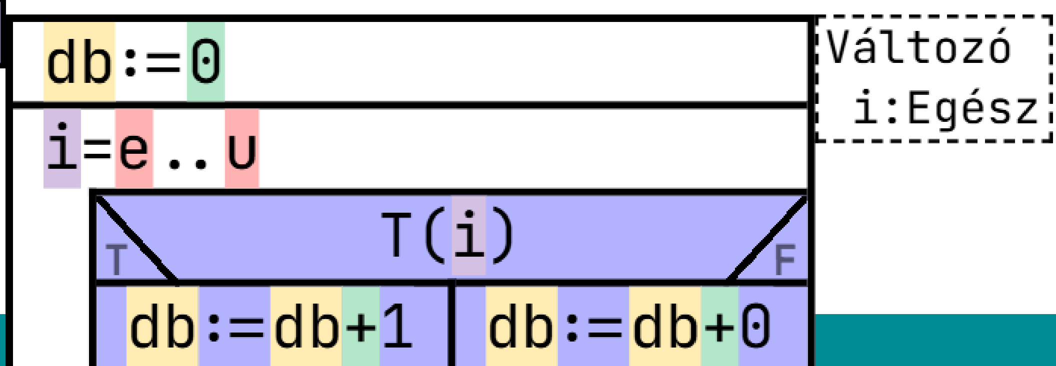
Fv: f:N->N, f(i)={1, ha T(i);
                  0 egyébként}

Ef: -

Uf: db=**SZUMMA**(i=e..u,f(i),0,+)

s ~ db

f(i) ~ {1, ha T(i);
        0 egyébként}



Általános összegzés
feltételes összegzés

Feladatsablon

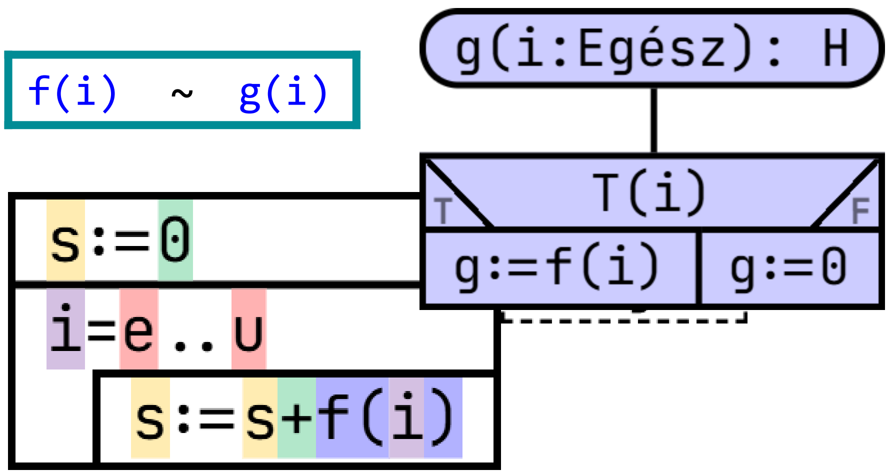
Be: e∈Z, u∈Z

Ki: s∈H

Ef: -

Uf: s=**SZUMMA**(i=e..u,f(i),0,+)

f(i) ~ g(i)



Feltételes összegzés

Be: e∈Z, u∈Z

Ki: s∈H

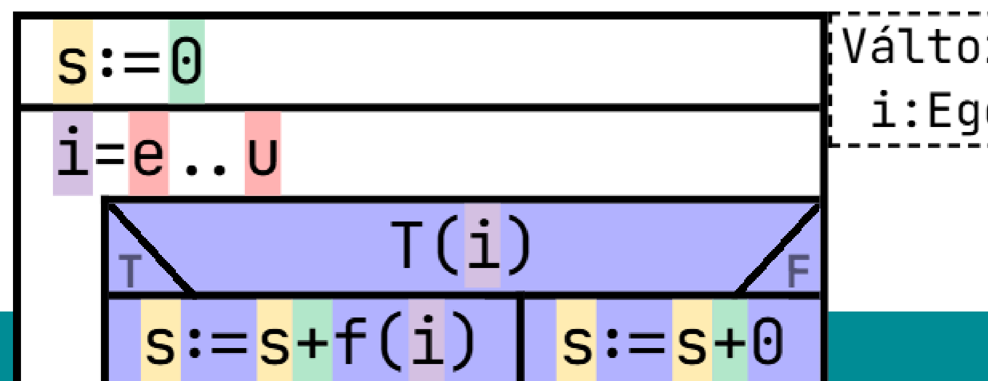
Fv: g:N->H, g(i)={f(i), ha T(i);
0 egyébként}

Ef: -

Uf: s=**SZUMMA**(i=e..u,g(i),0,+)

Uf: s=**SZUMMA**(i=e..u,g(i),T(i),0,+)

f(i) ~ {f(i), ha T(i);
0 egyébként}



Általános összegzés
másolás

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: s∈H

Ef: -
Uf: s=SZUMMA(i=e..u,f(i),0,+)
```

s	~	y
f(i)	~	[f(i)]
0,+	~	üres tömb, hozzáfűz

```
s:=0
i=e..u
  s:=s+f(i)
```

Változó
i:Egész

Másolás

```
Be: e∈Z, u∈Z
Ki: y∈H[1..u-e+1]
Fv: g:Z->H[], g(i)=[f(i)]
Ef: -
Uf: y=SZUMMA(i=e..u,g(i),
            üres tömb,hozzáfűz)
```

Pontosabban: üres sorozat

Dinamikus tömbnél jól használható!

```
y:=üres tömb
i=e..u
  y:=y hozzáfűz [f(i)]
```

Vál
i:

Általános összegzés
kiválogatás

Két speciális feltételes összegzés: megszámolás és tömbösszefűzés

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u,f(i),0,+)
```

```
s      ~  db
f(i)   ~  {1, ha T(i);
           0 egyébként}
0,+    ~  0, +
```

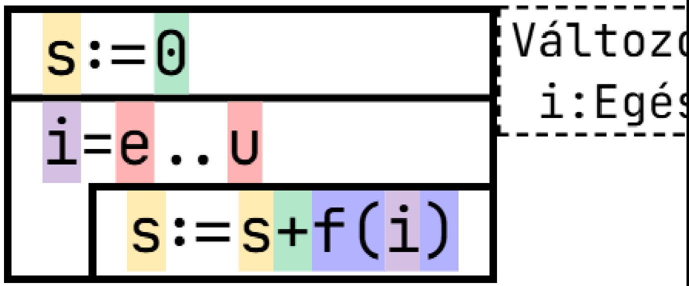
```
s      ~  y
f(i)   ~  {[f(i)], ha T(i);
           [] egyébként}
0,+    ~  üres tömb, hozzáfűz
```

Kiválogatás

```
Be: e∈Z, u∈Z
Ki: db∈N, y∈H[1..db]
Fv: g:N->N, g(i)={1, ha T(i);
                  0 egyébként}
Fv: h:N->H[], h(i)={[f(i)],ha T(i);
                    [] egyébként}
Ef: -
Uf: db=SZUMMA(i=e..u,g(i),0,+) és
    y =SZUMMA(i=e..u,h(i),
              üres tömb,hozzáfűz)
```

Általános összegzés
kiválogatás

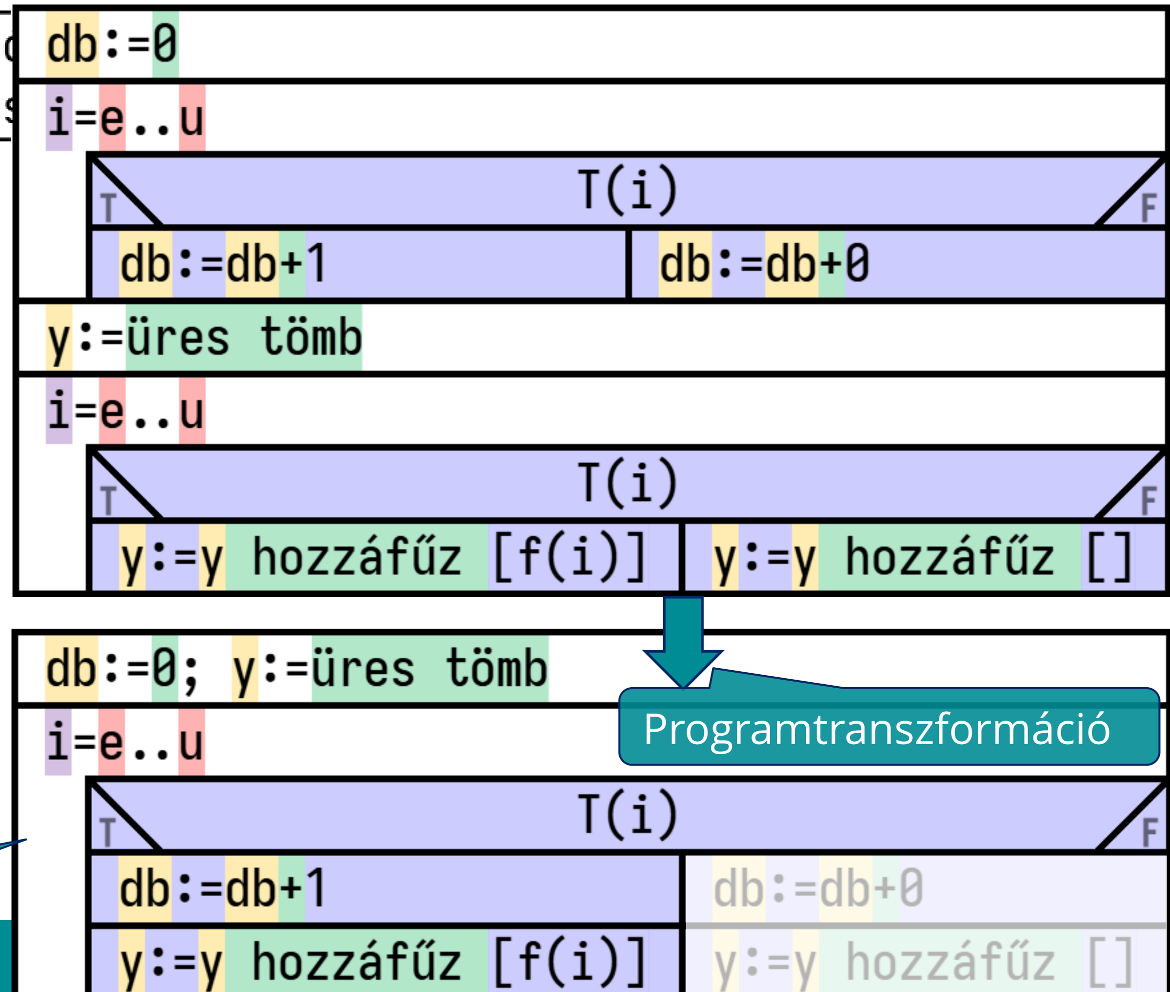
Feladatsablon




```
s       ~  y
f(i)    ~  {[f(i)], ha T(i);
             [] egyébként}
0,+     ~  üres tömb, hozzáfűz
```

```
s       ~  db
f(i)    ~  {1, ha T(i);
             0 egyébként}
0,+     ~  0, +
```

Kiválogatás



Dinamikus tömbnél jól használható!

Általános összegzés
kiválogatás

Elhagyva a darabszámot

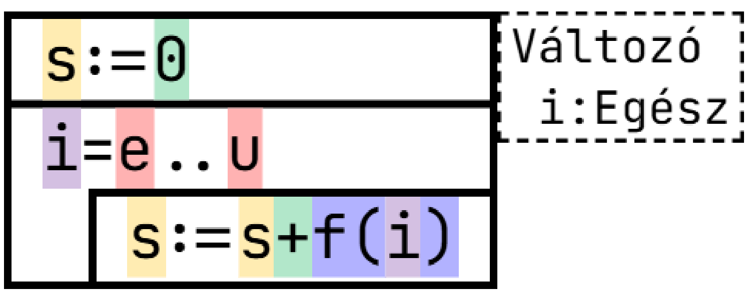
Feladatsablon

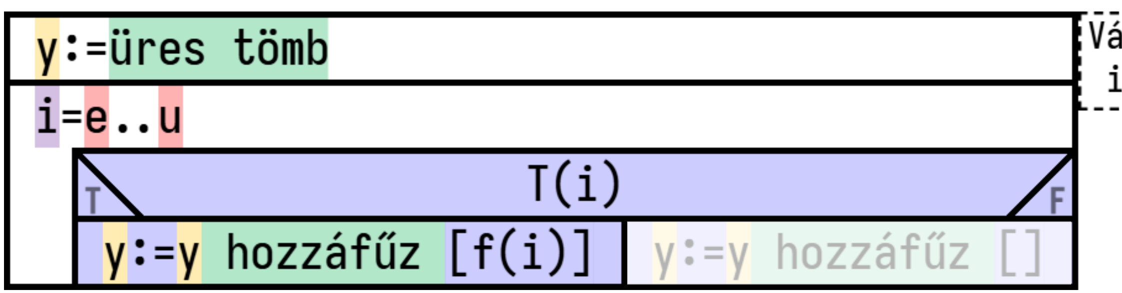
```
Be: e∈Z, u∈Z
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u,f(i),0,+)
```

```
s      ~  y
f(i)   ~  {[f(i)], ha T(i);
           [] egyébként}
0,+    ~  üres tömb, hozzáfűz
```

Kiválogatás

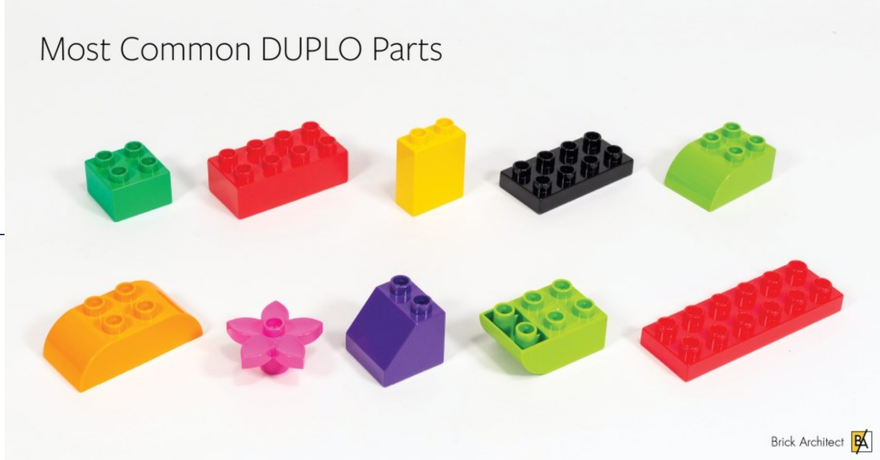
```
Be: e∈Z, u∈Z
Ki: y∈H[1..]
Fv: h:N->H[], h(i)={[f(i)],ha T(i);
                    [] egyébként}
Ef: -
Uf: y=SZUMMA(i=e..u,h(i),
              üres tömb,hozzáfűz)
```





Programozási minták

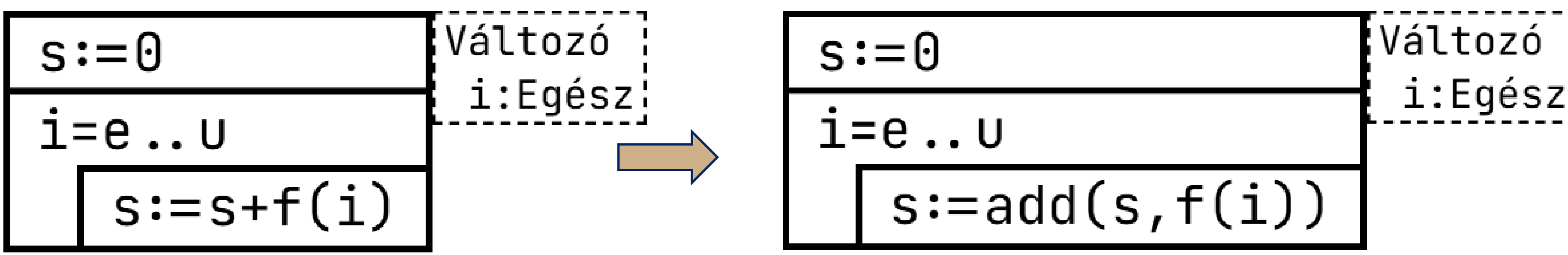
1. Összegzés
    a. Megszámolás
    b. (Feltételes összegzés)
    c. Másolás
    d. Kiválogatás
2. Maximumkiválasztás
    a. Minimumkiválasztás
3. Feltételes maximumkeresés
4. Keresés
    a. Eldöntés
    b. Mind eldöntés
5. Kiválasztás



Még általánosabb összegzés felé

A + műveletet tekintsük egy kétoperandusú függvénynek!

- a + b → +(a,b) → add(a,b)
- előnye, hogy a és b típusa eltérő lehet!



Még általánosabb összegzés felé

Ötlet

Próbáljuk meg valahogyan másképpen kifejezni azt, hogy a részösszeg aktuális értéke az előző részösszeg és az aktuális érték „összege” → rekurzív (önhivatkozó) felírás

Specifikáció

```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: f:Z->H
Ef: -
Uf: s=kezd+f(e)+f(e+1)+f(e+2)+...+f(u)
Uf: s=add(add(add(add(add(kezd,f(e)),f(e+1)),f(e+2)),...),f(u))
Uf: s=add(                                            ,f(u))
         add(                                    ,...)
             add(                          ,f(e+2))
                 add(              ,f(e+1))
                     add(kezd,f(e))
```

```
SZUMMA(e,u)={add(SZUMMA(e,u-1),f(u)), ha e<=u;
             kezd egyébként}
```

Az add művelet asszociativitása teszi lehetővé ezt az átírást..

Még általánosabb összegzés
sablon

Feladat

Adott az egész számok egy [e..u] intervalluma, egy f:[e..u]→H függvény, egy add:GxH→G függvény és egy G-beli kezdőérték. A kezdőértékből kiindulva szeretnénk egy kumulált értéket meghatározni az f függvény [e..u] intervallumon felvett értékein sorban alkalmazva az add függvényt.

Specifikáció

```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: f:Z->H
Fv: add:G x H->G
Fv: SUMMA:Z x Z->G,
    SZUMMA(e, u)={add(SZUMMA(e, u-1), f(u)), ha e<=u;
                  kezd egyébként}
Ef: -
Uf: s=SZUMMA(e, u)
```

$$s = \sum_{i=e}^{u} f(i) = \sum_{i=e}^{u-1} f(i) + f(u)$$

Rövidítve:

Uf: s=**SZUMMA**(i=e..u, f(i), kezd, add)

A specifikációs eszköz egyelőre nem támogatja

Algoritmus

```
s:=kezd
i=e..u
    s:=add(s,f(i))
```

Változó
i:Egész

Még általánosabb összegzés
sablon

Specifikáció

```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: f:Z->H
Fv: add:G x H->G
Ef: -
Uf: s=SZUMMA(i=e..u, f(i), kezd, add)
```

Algoritmus

akár: add(s, f(i), i)

```
s:=kezd
i=e..u
    s:=add(s,f(i))
```

```
Változó
  i:Egész
```

Feladat	kezd	add(s,p)
Összegzés	0	add(s,p)=s+p
Produktum	1	add(s,p)=s*p
Maximumkiválasztás	-∞ vagy f(e)	add(s,p)=max(s,p)
Másolás	[] (üres tömb)	add(s,p)=Végére(s,p)
Megszámolás	0	add(s,p)={s+1, ha p; s egyébként}
Kiválogatás	[] (üres tömb)	add(s,p)={Végére(s,p), ha T(p); s egyébként}
Eldöntés	hamis	add(s,p)=s vagy p

Még általánosabb összegzés
maximumkiválasztás

Feladatsablon

```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: add:G x H->G
Ef: -
Uf: s=
      SZUMMA(i=e..u,f(i),kezd,add)
```

Maximumkiválasztás

```
Be: e∈Z, u∈Z
Ki: maxért∈H
Fv: add:H x H->H, add(s,p)=max(s,p)
Ef: -
Uf: maxért=
      SZUMMA(i=e..u,f(i),f(e),add)
```

s	~	maxért
kezd	~	f(e)
add(s,p)	~	max(s,p)

```
s:=kezd
i=e..u
    s:=add(s,f(i))
```

Változó
i:Egész

```
maxért:=f(e)
i=e..u
    maxért:=max(maxért,f(i))
```

Változó
i:Egész

Még általánosabb összegzés
megszámolás

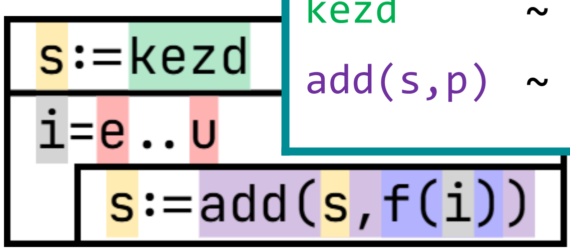
Feladatsablon

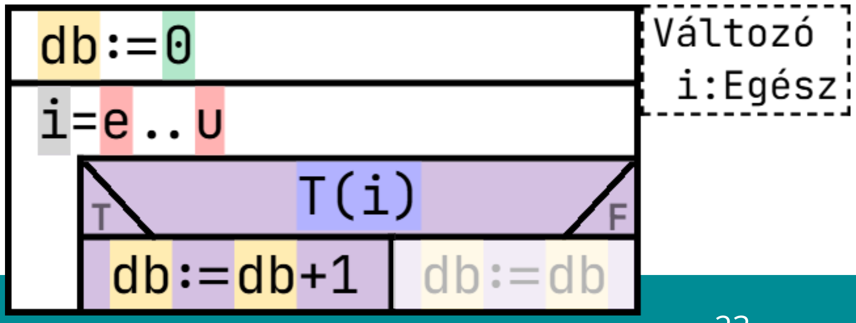
```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: add:G x H->G
Ef: -
Uf: s=SZUMMA(i=e..u,f(i),kezd,add)
```

Megszámolás

```
Be: e∈Z, u∈Z
Ki: db∈N
Fv: add:N x L->N,
    add(s,p)={s+1, ha p;
                   s egyébként}
Ef: -
Uf: db=SZUMMA(i=e..u,T(i),0,add)
```

```
s          ~ db
f(i)       ~ T(i)
kezd       ~ 0
add(s,p)   ~ {s+1, ha p;
               s egyébként}
```





Még általánosabb összegzés
eldöntés

Feladatsablon

```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: add:G x H->G

Ef: -
Uf: s=SZUMMA(i=e..u,f(i),kezd,add)
```

Eldöntés

```
Be: e∈Z, u∈Z
Ki: van∈L
Fv: add:L x L->L,
    add(s,p)=s vagy p
Ef: -
Uf: db=SZUMMA(i=e..u,T(i),hamis,add)
```

s	~	van
f(i)	~	T(i)
kezd	~	hamis
add(s,p)	~	s vagy p

```
s:=kezd
i=e..u
    s:=add(s,f(i))
Változó
  i:Egész
```

```
van:=hamis
i=e..u
    van:= van vagy T(i)
Változó
  i:Egész
```

Még általánosabb összegzés
kiválogatás

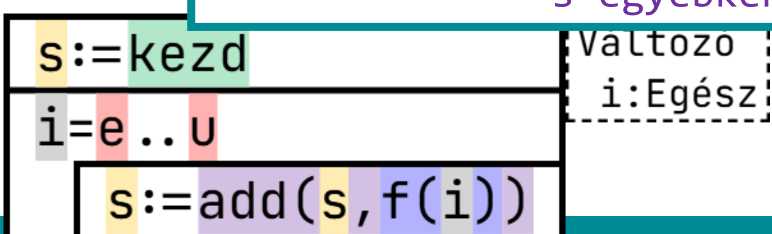
Feladatsablon

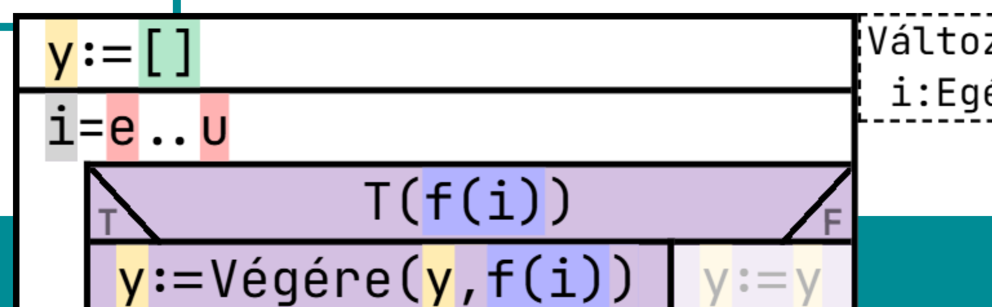
```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: add:G x H->G
Ef: -
Uf: s=SZUMMA(i=e..u,f(i),kezd,add)
```

Kiválogatás

```
Be: e∈Z, u∈Z
Ki: y∈H[1..],
Fv: add:H[] x H->H[],
    add(s,p)={Végére(s,p), ha T(p);
                        s egyébként}
Ef: -
Uf: y=SZUMMA(i=e..u,f(i),[],add)
```

s	~	y
kezd	~	[]
add(s,p)	~	{Végére(s,p), ha T(p); s egyébként}





Még általánosabb összegzés
kiválogatás

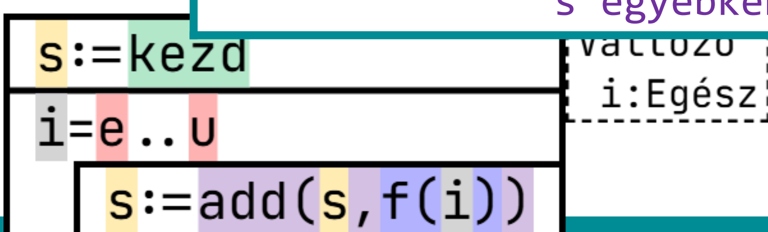
Feladatsablon

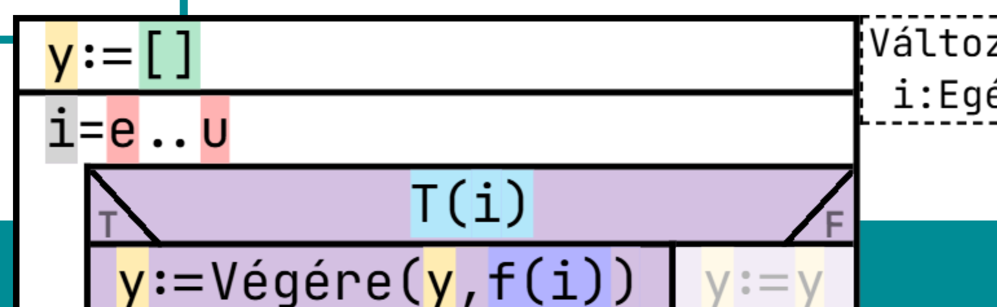
```
Be: e∈Z, u∈Z, kezd∈G
Ki: s∈G
Fv: add:G x H->G

Ef: -
Uf: s=SZUMMA(i=e..u,f(i),kezd,add)
```

Kiválogatás

```
Be: e∈Z, u∈Z
Ki: y∈H[1..],
Fv: add:H[] x H x Z->H[],
  add(s,p,i)={Végére(s,p), ha T(i);
                    s egyébként}
Ef: -
Uf: y=SZUMMA(i=e..u,f(i),[],add)
```

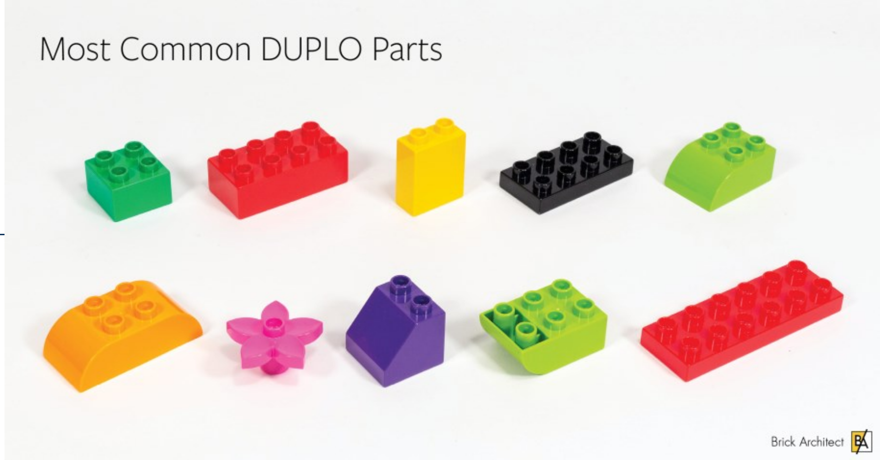
```
s          ~  y
kezd       ~  []
add(s,p)   ~  {Végére(s,p), ha T(i);
                s egyébként}
```





Programozási minták

1. Általános összegzés
    a. Megszámolás
    b. Feltételes összegzés
    c. Másolás
    d. Kiválogatás
    e. Maximumkiválasztás
    f. Minimumkiválasztás
    g. Feltételes maximumkeresés
    h. Keresés
    i. Eldöntés
    j. Mind eldöntés
    k. Kiválasztás



„Programozási tételek”

Programozási tételek

- A programozási mintákat sokszor hívják **programozási tétel**nek is
- Tágabb értelemben szinonimaként
  - „A visszavezetés során mintaként használt *specifikáció-algoritmus párt* **programozási tétel**nek hívjuk. Az elnevezés onnan származik, hogy egy minta specifikáció-algoritmus párt egy matematikai tételhez hasonlóan alkalmazunk: ha egy kitűzött feladat specifikációja hasonlít a mintafeladat specifikációjára, akkor a mintafeladat algoritmusa lényegében megoldja a kitűzött feladatot is.”
- Szűkebb értelemben
  - A tömbökre kimondott mintafeladatok a programozási tételek

Példa
összegzés

Intervallum

Be: e∈Z, u∈Z

Ki: s∈H

Ef: -

Uf: s=**SZUMMA**(i=e..u, f(i))

↔

Tömb

Be: n∈N, x∈H[1..n]

Ki: s∈Z

Ef: -

Uf: s=**SZUMMA**(i=1..n, x[i])

Visszavezetés:

e..u	~	1..n
f(i)	~	x[i]

Algoritmus:

```
s:=0
i=e..u
    s:=s+f(i)
```

Változó
 i:Egész

```
s:=0
i=1..n
    s:=s+x[i]
```

Változó
 i:Egész

Példa
megszámolás

Intervallum

Be: e∈Z, u∈Z

Ki: db∈N

Ef: -

Uf: db=**DARAB**(i=e..u, T(i))

Tömb

Be: n∈N, x∈H[1..n]

Ki: db∈N

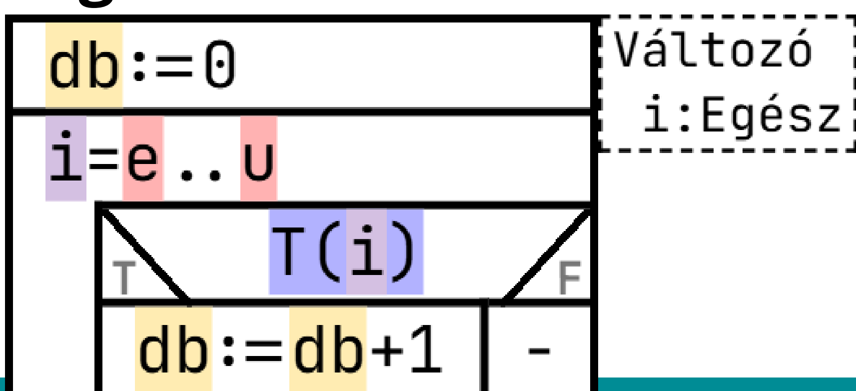
Ef: -

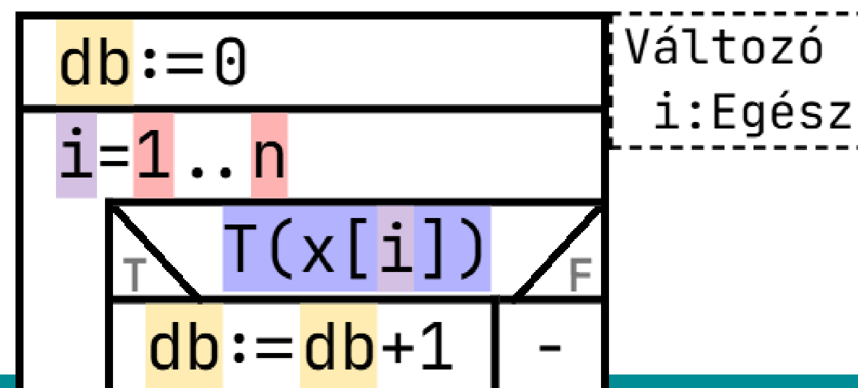
Uf: db=**DARAB**(i=1..n, T(x[i]))

Visszavezetés:

e..u	~	1..n
T(i)	~	T(x[i])

Algoritmus:





Minta → Tétel

Sablon		Tétel
f(i)	→	x[i]
T(i)	→	T(x[i])

Összegzés programozási tétel

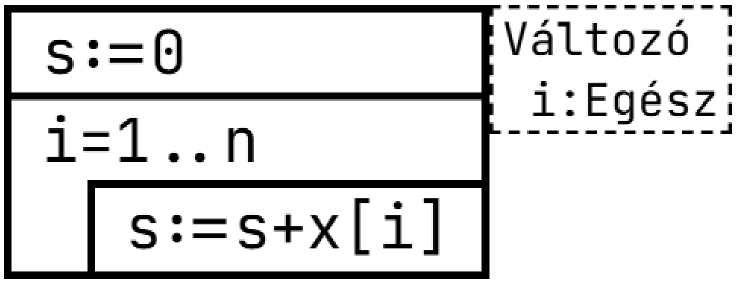
Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb. A H halmaz elemein értelmezett az összeadás művelet. Határozzuk meg a tömb elemeinek az összegét, azaz a $\sum_{i=1}^{n} x[i]$ kifejezés értékét!

Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=1..n, x[i])
```

Algoritmus



Megszámolás programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. Határozzuk meg, hogy a tömb elemeire a T feltétel hányszor veszi fel az igaz értéket!

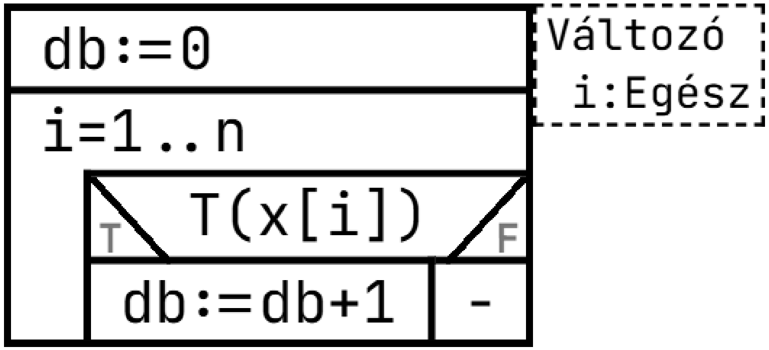
Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: db∈N
Ef: -
Uf: db=SZUMMA(i=e..u, 1, T(x[i]))
```

Rövidítve:

```
Uf: db=DARAB(i=e..u, T(x[i]))
```

Algoritmus



Maximumkiválasztás programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó, nem üres x tömb. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg a tömb legnagyobb elemének értékét és indexét!

Specifikáció

Be: n∈N, x∈H[1..n]

Ki: maxind∈Z, maxért∈H

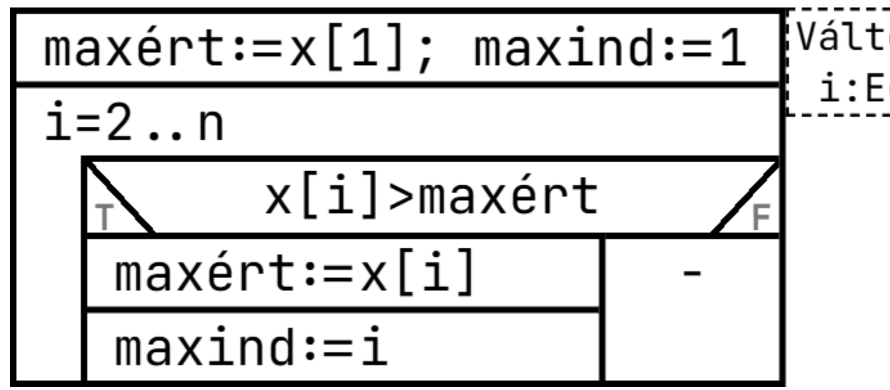
Ef: n>0

Uf: maxind∈[1..n] és
    ∀i∈[1..n]:(x[maxind]>=x[i]) és
    maxért=x[maxind]

Rövidítve:

Uf: (maxind,maxért)=**MAX**(i=1..n,x[i])

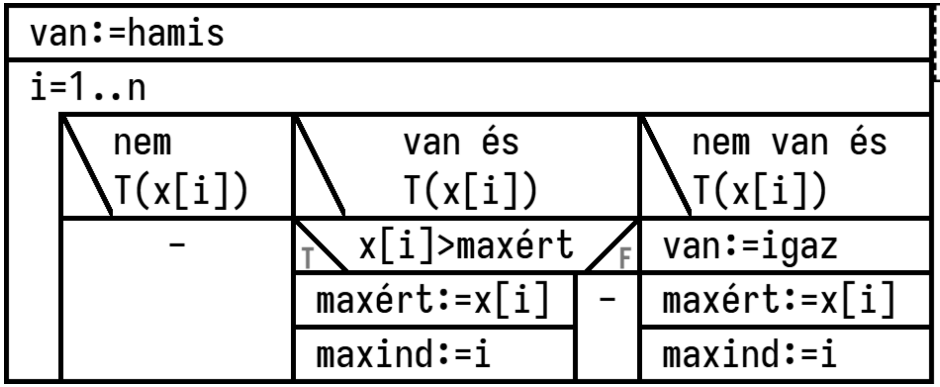
Algoritmus



Feltételes maximumkeresés programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. A H halmaz elemein értelmezett egy teljes rendezési reláció. Határozzuk meg az x tömb T feltételt kielégítő elemei közül a legnagyobb elem értékét és indexét, ha egyáltalán van ilyen!



Specifikáció és algoritmus:

```
Be: n∈N, x∈H[1..n]
Ki: van∈L, maxind∈Z, maxért∈H
Ef: -
Uf: van = ∃i∈[1..n]:(T(x[i])) és
    van -> (maxind∈[1..n] és
            maxért=x[maxind] és T(x[maxind]) és
            ∀i∈[1..n]:(T(x[i]) -> maxért>=x[i]))
```

Rövidítve:

```
Uf: (van,maxind,maxért)=MAX(i=1..n,x[i],T(x[i]))
```

Keresés programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. Határozzuk meg, hol van az x tömb első olyan eleme, ha egyáltalán van, amely kielégíti a T feltételt!

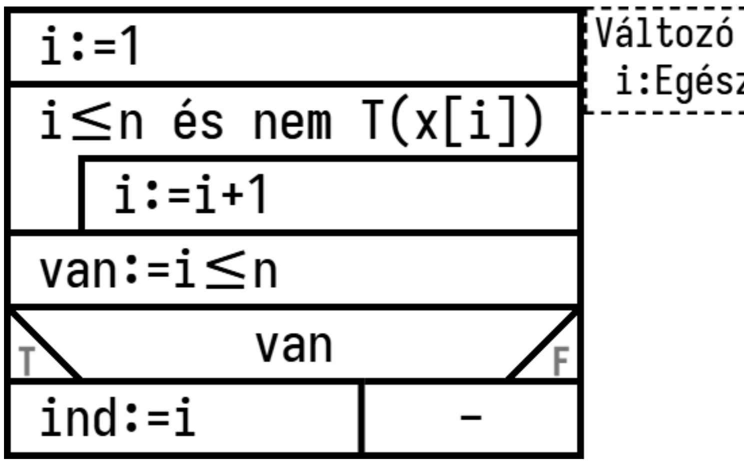
Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: van∈L, ind∈Z
Ef: -
Uf: van=∃i∈[1..n]:(T(x[i])) és
 van->(ind∈[1..n] és T(x[ind]) és
          ∀i∈[1..ind-1]:(nem T(x[i])))
```

Rövidítve:

```
Uf: (van,ind)=KERES(i=1..n,T(x[i]))
```

Algoritmus



Eldöntés programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. Határozzuk meg, hogy van-e a tömbnek olyan eleme, amely kielégíti a T feltételt!

Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: van∈L
Ef: -
Uf: van=∃i∈[1..n]:(T(x[i]))
```

Rövidítve:

```
Uf: van=VAN(i=1..n,T(x[i]))
```

Algoritmus

```
i:=1
i≤n és nem T(x[i])
    i:=i+1
van:=i≤n
```

```
Változó
  i:Egész
```

Kiválasztás programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. Határozzuk meg a tömb első olyan elemének az indexét, amely kielégíti a T feltételt, ha tudjuk, hogy ilyen elem biztosan van!

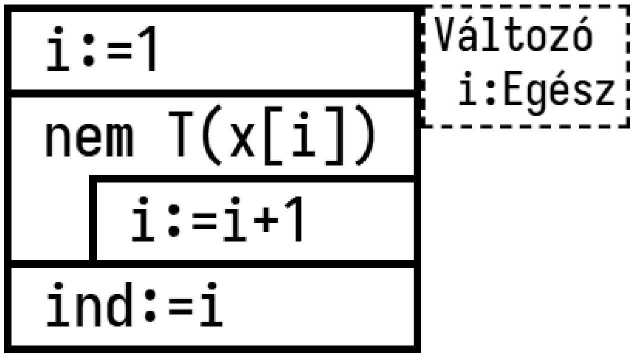
Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: ind∈Z
Ef: ∃i∈[1..n]:(T(x[i]))
Uf: ind>=1 és T(x[ind]) és
    ∀i∈[1..ind-1]:(nem T(x[i]))
```

Rövidítve:

```
Uf: ind=KIVÁLASZT(i>=e,T(x[i]))
```

Algoritmus



Másolás programozási tétel

Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy f:H→G függvény. Rendeljük a tömb minden eleméhez az f függvény hozzá tartozó értékét!

Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: y∈G[1..n]
Ef: -
Uf: ∀i∈[1..n]:(y[i]=f(x[i]))
```

Rövidítve:

```
Uf: y=MÁSOL(i=1..n, f(x[i]))
```

Algoritmus

```
i=1..n
    y[i]:=f(x[i])
```

```
Változó
  i:Egész
```

Kiválogatás programozási tétel

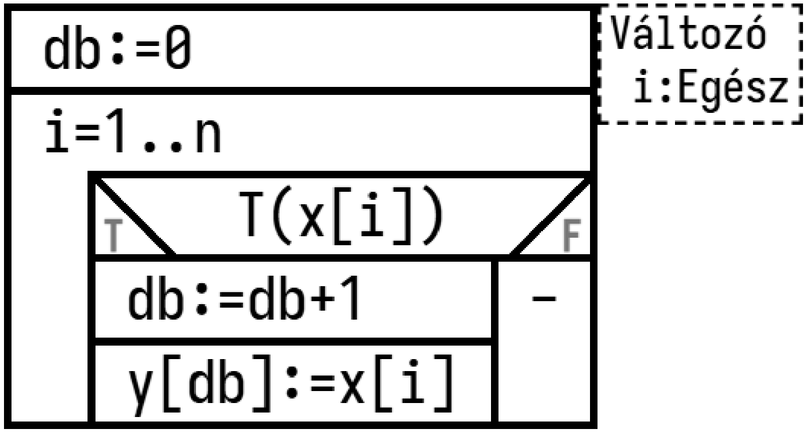
Feladat

Adott egy n elemű H halmazbeli elemeket tartalmazó x tömb és egy T:H→Logikai feltétel. Határozzuk meg a tömb azon elemeit, amelyekre teljesül a T feltétel!

Specifikáció

```
Be: n∈N, x∈H[1..n]
Ki: db∈N, y∈H[1..db]
Ef: -
Uf: db=DARAB(i=1..n,T(x[i])) és
    ∀i∈[1..db]:(T(y[j])) és
    y⊆x
```

Algoritmus



Rövidítve:

```
Uf: (db,y)=KIVÁLOGAT(i=1..n,T(x[i]),x[i])
```

Programozási minták megvalósítása általánosított függvényekkel

Cél

- Programozási minták megvalósítása paraméterekkel általánosított függvényekkel
- Példa: keresés
  - Specifikáció: `(van,ind)=`**`KERES`**`(i=e..u, T(i))`
  - C# függvény: `(van,ind)=Keres(e, u, T)`
    - ahol T egy megfelelően definiált függvény
- Egy konkrét feladatnál (pl. valódi osztó keresése)
  - Specifikáció: `(van,ind)=`**`KERES`**`(i=2..n-1, i|n)`
  - C# függvény: `(van,ind)=Keres(2, n-1, T)`
    - ahol: `bool T(int i) { return n % i==0; }`

Példa
negatív szám

Adjuk meg, hogy hol van negatív szám egy számokat tartalmazó tömbben!

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Fv: T:Z->L
Ef: -
Uf: (van,ind)=KERES(i=e..u,T(i))
```

Negatív szám

```
Be: n∈N, x∈Z[1..n]
Ki: van∈L, negind∈N

Ef: -
Uf: (van,negind)=KERES(i=1..n,
                       x[i]<0)
```

Visszavezetés:

ind	~	negind
e..u	~	1..n
T(i)	~	x[i]<0

Algoritmus:

```
ind:=e
ind≤u és nem T(ind)
    ind:=ind+1
van:=ind≤u
```

```
negind:=1
negind≤n és nem x[negind]<0
    negind:=negind+1
van:=negind≤n
```

Példa
negatív szám

```
static (bool van, int negind) Keres
                            (int n, int[] x) {
  bool van; int negind;

  negind = 1;
  while (negind <= n &&
         !(x[negind - 1] < 0)) {
    negind = negind + 1;
  }
  van = negind <= n;

  return (van, negind);
}
```

Negatív szám

Be: n∈N, x∈Z[1..n]

Ki: van∈L, negind∈N

Ef: -

Uf: (van,negind)=**KERES**(i=1..n,
x[i]<0)

negind:=1
negind ⩽ n és nem x[negind]<0
negind:=negind+1
van:=negind ⩽ n

Példa
negatív szám

Feladatsablon

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Fv: T:Z->L
Ef: -
Uf: (van,ind)=KERES(i=e..u,T(i))
```

Negatív szám

```
Be: n∈N, x∈Z[1..n]
Ki: van∈L, negind∈N
Fv: T:Z->L, T(i)=x[i]<0
Ef: -
Uf: (van,negind)=KERES(i=1..n,
                       T(i))
```

Visszavezetés:

ind	~	negind
e..u	~	1..n

Algoritmus:

```
ind:=e
ind≤u és nem T(ind)
    ind:=ind+1
van:=ind≤u
```

→

```
negind:=1
negind≤n és nem T(negind)
    negind:=negind+1
van:=negind≤n
```

```
T(i:Egész): Logikai
T:=x[i]<0
```

Példa
negatív szám

```
static (bool van, int negind) Keres
                         (int n, int[] x) {
  bool van; int negind;

  negind = 1;
  while (negind <= n && !T(negind, x)) {
    negind = negind + 1;
  }
  van = negind <= n;

  return (van, negind);
}
static bool T(int i, int[] x) {
  return x[i - 1] < 0;
}
```

Külső függvény nem lát rá x-re, ezért át kell azt is adni neki.

Negatív szám

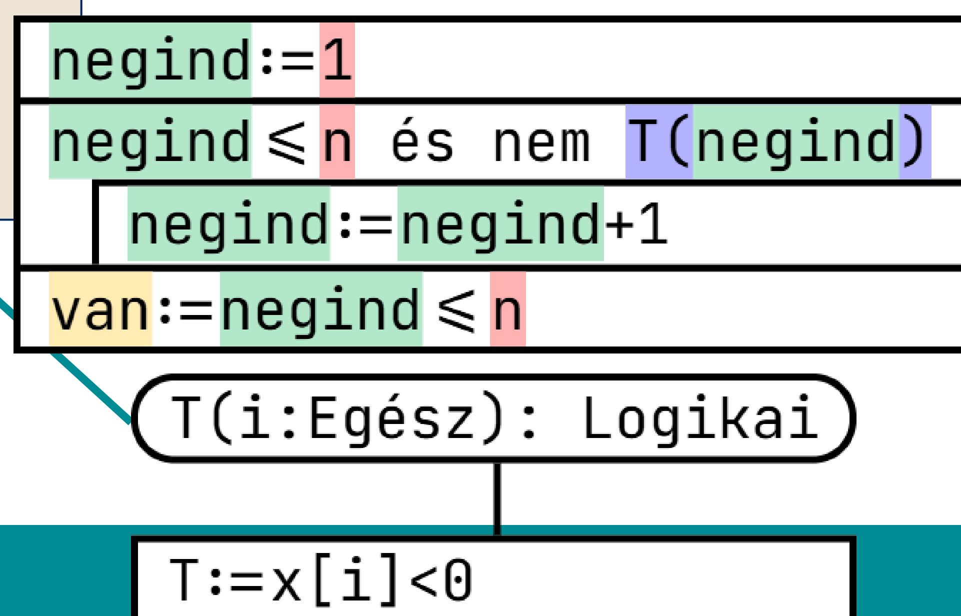
Be: n∈N, x∈Z[1..n]

Ki: van∈L, negind∈N

Fv: T:Z->L, T(i)=x[i]<0

Ef: -

Uf: (van,negind)=**KERES**(i=1..n,
T(i))



Példa
negatív szám

1. megoldás: osztályszintű változók létrehozása
→ osztálybeli függvényekre globális

```
static int[] x;
static (bool van, int negind) Keres
                          (int n, int[] x) {
  bool van; int negind;

  negind = 1;
  while (negind <= n && !T(negind)) {
    negind = negind + 1;
  }
  van = negind <= n;

  return (van, negind);
}
static bool T(int i) {
  return x[i - 1] < 0;
}
```

Negatív szám

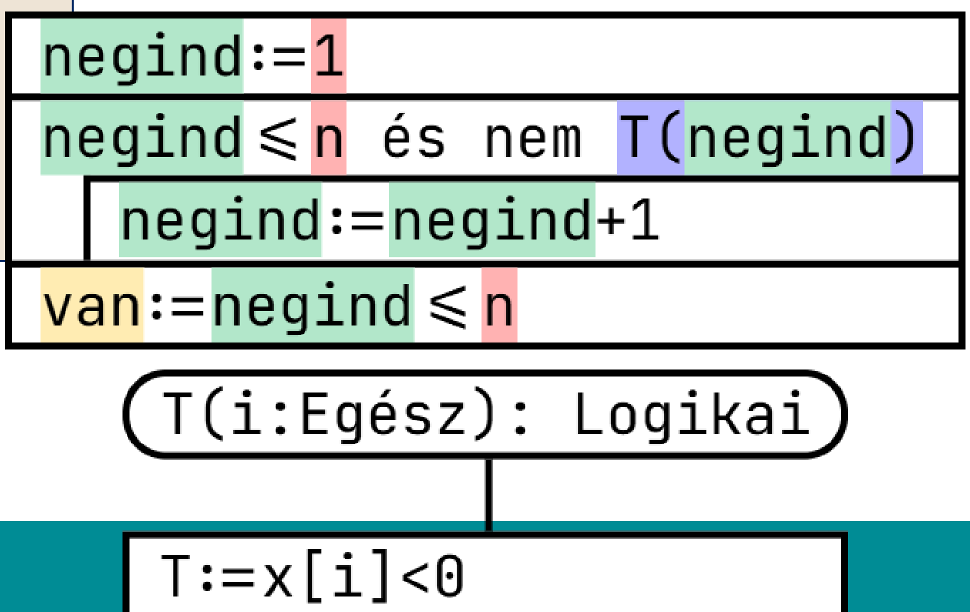
Be: n∈N, x∈Z[1..n]

Ki: van∈L, negind∈N

Fv: T:Z->L, T(i)=x[i]<0

Ef: -

Uf: (van,negind)=**KERES**(i=1..n,
T(i))



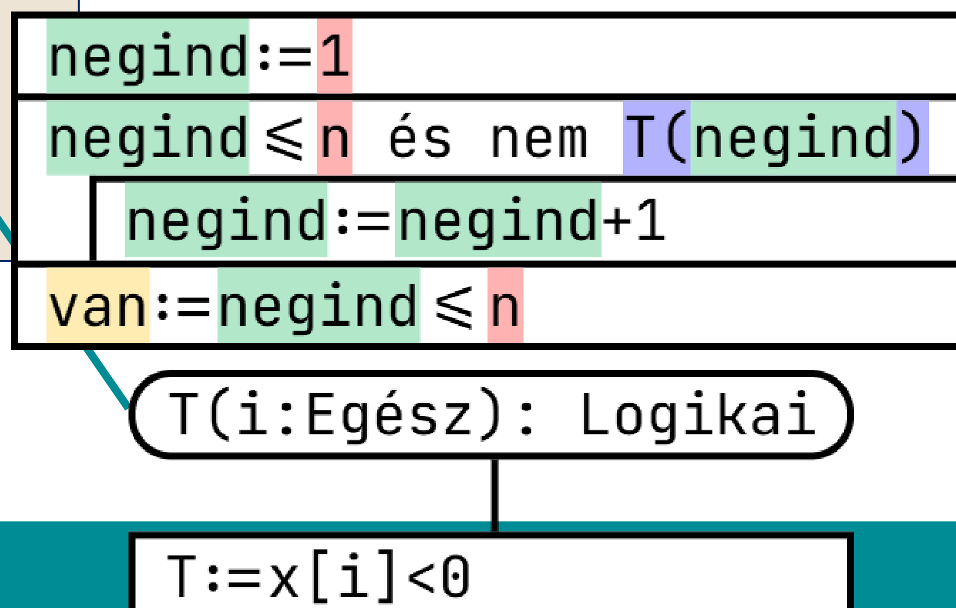
Példa
negatív szám

2. megoldás:
Lokális függvény: rálát a tartalmazó függvény lokális adataira, így x-et nem kell paraméterként átadni, így szignatúrája megegyezik az elvárttal.

```
static (bool van, int negind) Keres
                         (int n, int[] x) {
  bool T(int i) {
    return x[i - 1] < 0;
  }

  bool van; int negind;

  negind = 1;
  while (negind <= n && !T(negind)) {
    negind = negind + 1;
  }
  van = negind <= n;

  return (van, negind);
}
```

Negatív szám

```
Be: n∈N, x∈Z[1..n]
Ki: van∈L, negind∈N
Fv: T:Z->L, T(i)=x[i]<0
Ef: -
Uf: (van,negind)=KERES(i=1..n,
                       T(i))
```



Példa
negatív szám

3. megoldás: Kapja meg a T függvényt kívülről!

Függvényparaméter!

```
static void Main(string[] args) {
  Keres(n, x, T);
}
static (bool van, int negind) Keres
 (int n, int[] x, Func<int, int[], bool> T) {
  bool van; int negind;

  negind = 1;
  while (negind <= n && !T(negind, x)) {
    negind = negind + 1;
  }
  van = negind <= n;

  return (van, negind);
}
static bool T(int i, int[] x) {
  return x[i - 1] < 0;
}
```

Negatív szám

Be: n∈N, x∈Z[1..n]

Ki: van∈L, negind∈N

Fv: T:Z->L, T(i)=x[i]<0

Ef: -

Uf: (van,negind)=**KERES**(i=1..n, T(i))

negind:=1
negind≤n és nem T(negind)
negind:=negind+1
van:=negind≤n

T(i:Egész): Logikai

T:=x[i]<0

A sablon megvalósítása függvényként

Feladatsablon

Be: e∈Z, u∈Z

Ki: van∈L, ind∈Z

Fv: T:Z->L

Ef: -

Uf: (van,ind)=**KERES**(i=e..u,T(i))

Algoritmus:

ind:=e
ind ≤ u és nem T(ind)
ind:=ind+1
van:=ind ≤ u

```
static (bool van, int ind) Keres
            (int e, int u, Func<int, bool> T) {
    bool van; int ind;

    ind = e;
    while (ind <= u && !T(ind)) {
      ind = ind + 1;
    }
    van = ind <= u;

    return (van, ind);
}
```

Az általánosított függvény használata

```
static void Main(string[] args) {
  int n = 6; int[] x = { 1, 3, 4, -5, -2, 0 };
  bool van; int negind;
  bool T(int i) {
    return x[i - 1] < 0;
  }

  (van, negind) = Keres(1, n, T);
}

static (bool van, int ind) Keres(int e, int u, Func<int, bool> T) {
  bool van; int ind;

  ind = e;
  while (ind <= u && !T(ind)) {
    ind = ind + 1;
  }
  van = ind <= u;

  return (van, ind);
}
```

Függvények lambda kifejezéssel

- [illegible]
- [illegible]

```
(formális par.lista) => visszatérési érték
```

- [illegible]

```
bool T(int i) {
  return i % 2 == 0;
}
Console.WriteLine(T(3));
```

- [illegible]

```
Func<int, bool> T = i => i % 2 == 0;

Console.WriteLine(T(3));
```

Az általánosított függvény használata

```
static void Main(string[] args) {
  int n = 6; int[] x = { 1, 3, 4, -5, -2, 0 };
  bool van; int negind;

  (van, negind) = Keres(1, n, i => x[i - 1] < 0);
}

static (bool van, int ind) Keres(int e, int u, Func<int, bool> T) {
  bool van; int ind;

  ind = e;
  while (ind <= u && !T(ind)) {
    ind = ind + 1;
  }
  van = ind <= u;

  return (van, ind);
}
```

Programozási minták általánosított függvényei

- Ez a fajta általánosítás mindegyik programozási minta esetén elvégezhető
- Az így kapott függvényeket egy segédosztályba helyeztük (Mintak.cs)
  - A segédosztályt vagy a feladat osztálya mellé másoljuk a Program.cs fájlban, vagy
  - fájlszinten hozzáadjuk a Mintak.cs fájlt a projekthez.
- Az általánosított függvények használata, pl:
  - `Mintak.Keres(1, n, i => x[i-1] < 0)`

Minták általánosított függvényei
összegzés

- Uf: `s=SZUMMA(i=e..u, f(i))`
- Példa:
  - Uf: `s=SZUMMA(i=1..n, szamok[i])`
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;

int s = Mintak.Szumma(1, n, i => szamok[i - 1]);
int s = Mintak.Szumma(0, n - 1, i => szamok[i]);
```

- Példa:
  - Uf: `s=SZUMMA(i=1..10, i)`
  - Kód:

```
int s = Mintak.Szumma(1, 10, i => i);
```

Minták általánosított függvényei
általános összegzés

- Uf: s=**SZUMMA**(i=e..u, f(i), kezd, add)
- Példa:
  - Uf: p=**SZUMMA**(i=1..n, szamok[i], 1, add)
    add(s,p)=s*p
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;

int s = Mintak.Szumma(1, n, i => szamok[i - 1], 1, (s,p)=>s*p);
```

- Példa:
  - Uf: y=**SZUMMA**(i=1..n, szamok[i], [], add)
    add(s,p)={Végére(s,p), ha p<0;
    s egyébként}
  - Kód:

```
List<int> y = Mintak.Szumma(1, n, i => szamok[i - 1],
                            new List<int>(),
                            (s, p) => {
                             if (p<0) { s.Add(p); }
                            });
```

Minták általánosított függvényei
megszámolás

- Uf: `db=`**`DARAB`**`(i=e..u, T(i))`
- Példa:
  - Uf: `db=`**`DARAB`**`(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { 2, 4, -7, 5, 3, -2, 1 };
int n = szamok.Length;

int db = Mintak.Darab(1, n, i => szamok[i - 1] < 0);
```

Minták általánosított függvényei
maximumkiválasztás

- Uf: (maxind,maxért)=**MAX**(i=e..u,f(i))
- Példa:
  - Uf: (maxi,maxé)=**MAX**(i=2..n,
    szamok[i]-szamok[i-1])
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;

(int maxi, int maxe) =
         Mintak.Max(2, n, i => szamok[i - 1] – szamok[i - 2]);
```

- Példa:
  - Uf: (ind,diák)=**MAX**(i=1..n, diákok[i])
  - Kód:

```
struct Diak { public string nev; public int jegy; }

(int ind,int diak) = Mintak.Max(1, n, i => diakok[i - 1],
                                (d1,d2) => d1.jegy > d2.jegy);
```

Összehasonlító függvény

Minták általánosított függvényei
feltételes maximumkeresés

- Uf: `(van,maxind,maxért)= FELTMAX(i=e..u,f(i),T(i))`
- Példa:
  - Uf: `(van,maxi,maxé)= FELTMAX(i=1..n,szamok[i],szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

(bool van, int maxi, int maxe) =
 Mintak.Max(1, n, i => szamok[i - 1], i => szamok[i – 1] < 0);
```

Minták általánosított függvényei
keresés

- Uf: `(van,ind)=KERES(i=e..u,T(i))`
- Példa:
  - Uf: `(van,ind)=KERES(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

(bool van,int ind) = Mintak.Keres(1, n, i => szamok[i - 1]<0);
```

Minták általánosított függvényei
eldöntés

- Uf: `van=VAN(i=e..u,T(i))`
- Példa:
  - Uf: `van=VAN(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

bool van = Mintak.Van(1, n, i => szamok[i - 1] < 0);
```

- Példa:
  - Uf: `mind=MIND(i=1..n, szamok[i]>0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

bool mind = Mintak.Mind(1, n, i => szamok[i - 1] > 0);
```

Minták általánosított függvényei
kiválasztás

- Uf: `ind=`**`KIVÁLASZT`**`(i>=e,T(i))`
- Példa:
  - Uf: `ind=`**`KIVÁLASZT`**`(i>=1, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int ind = Mintak.Kivalaszt(1, i => szamok[i – 1] < 0);
```

Minták általánosított függvényei
másolás

- Uf: `y=MÁSOL(i=e..u, f(i))`
- Példa:
  - Uf: `y=MÁSOL(i=1..n, abs(szamok[i]))`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int[] y = Mintak.Masol(1, n, i => Math.Abs(szamok[i – 1]));
```

Minták általánosított függvényei
kiválogatás

- Uf: (db,y)=**KIVÁLOGAT**(i=e..u,T(i),f(i))
- Példa:
  - Uf: (db,y)=**KIVÁLOGAT**(i=1..n,szamok[i]<0,
    szamok[i])
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int[] y = Mintak.Kivalogat(1, n, i => szamok[i – 1] < 0,
                                 i => szamok[i – 1]);
```

Példa
legjobb jó tanuló

	1	2	3	m=4
1	5	5	5	3
2	5	4	4	4
3	5	3	2	4
4	5	4	5	5
n=5	5	5	4	4

Feladat:

Egy egész számokat tartalmazó mátrixban melyik az a sor, amiben csak 4-es és 5-ös van, és az összege a legnagyobb?

Specifikáció:

	1	2	3	m=4	
1	5	5	5	3	
2	5	4	4	4	17
3	5	3	2	4	
4	5	4	5	5	19
n=5	5	5	4	4	18

→ van=igaz
→ legjobb=4

```
Be: n∈N, m∈N, jegyek∈N[1..n,1..m]
Ki: van∈L, legjobb∈N
Fv: összeg:N->N,
    összeg(diák)=SZUMMA(tantárgy=1..m,jegyek[diák,tantárgy])
Fv: jó:N->L,
    jó(diák)=MIND(tantárgy=1..m,4<=jegyek[diák,tantárgy]<=5)
Ef: ∀sor∈[1..n]:(∀oszlop∈[1..m]:(1<=jegyek[sor,oszlop]<=5))
Uf: (van,legjobb,)=FELTMAX(diák=1..n,összeg(diák),jó(diák))
```

Példa
legjobb jó tanuló

	1	2	3	m=4
1	5	5	5	3
2	5	4	4	4
3	5	3	2	4
4	5	4	5	5
n=5	5	5	4	4

Csak a külső szinten

```
static void Main(string[] args) {
  int n; int m; int[,] jegyek;
  bool van; int legjobb;

  (n, m, jegyek) = beolvas();
  (van,legjobb,_)= Mintak.FeltMax(1, n,
    diak => osszeg(diak, m, jegyek),
    diak => jo(diak, m, jegyek)
  );
  kiir(van, legjobb);
}
```

Specifikáció:

```
Be: n∈N, m∈N, jegyek∈N[1..n,1..m]
Ki: van∈L, legjobb∈N
Fv: összeg:N->N,
    összeg(diák)=SZUMMA(tantárgy=1..m,jegyek[diák,tantárgy])
Fv: jó:N->L,
    jó(diák)=MIND(tantárgy=1..m,4<=jegyek[diák,tantárgy]<=5)
Ef: ∀sor∈[1..n]:(∀oszlop∈[1..m]:(1<=jegyek[sor,oszlop]<=5))
Uf: (van,legjobb,)=FELTMAX(diák=1..n,összeg(diák),jó(diák))
```

Minden szinten

```
static void Main(string[] args) {
  int n; int m; int[,] jegyek;
  bool van; int legjobb;

  (n, m, jegyek) = beolvas();
  (van, legjobb, _) = Mintak.FeltMax(1, n,
    diak => Mintak.Szumma(1, m, tantargy => jegyek[diak - 1, tantargy - 1]),
    diak => Mintak.Mind(1, m, tantargy => 4 <= jegyek[diak - 1, tantargy - 1] &&
                                              jegyek[diak - 1, tantargy - 1] <= 5)
  );
  kiir(van, legjobb);
}
```

Programozási tételek megvalósítása általánosított függvényekkel

Programozási tételek általánosított függvényei

- Az intervallumra kimondott programozási mintákhoz hasonló általánosítás elvégezhető a tömbökre kimondott programozási tételekre is.
- Az így kapott függvények ugyanabban a segédosztályban találhatók (Mintak.cs)
  - A segédosztályt vagy a feladat osztálya mellé másoljuk a Program.cs fájlban, vagy
  - fájlszinten hozzáadjuk a Mintak.cs fájlt a projekthez.
- Az általánosított függvényeknek ugyanaz a neve, csak más típusú paraméterekkel dolgozik (függvény túlterhelés):
  - `Mintak.Keres(x, e => e < 0)`

Programozási tételek általánosított függvényei

- Ahogy a tömbökre kimondott tételek lehetőségei szűkebbbek az intervallumra kimondott mintákénál, úgy az általánosított függvényeknél is limitációkkal kell számolnunk:
  - a tömbök 0-tól indexelődnek
  - az intervallum nem határozható meg, az a tömb indextartománya
  - a tételek egyes függvényei csak az aktuális elemhez férnek hozzá, az indexhez nem (kivétel a kiválogatás)
  - mátrix helyett tömbök tömbje kell (jagged array)
    - int[,] → int[][]

Programozási tételek általánosított függvényei

- Azért foglalkozunk a tételek nyelvi általánosításával, mert a legtöbb programozási nyelvben beépítve megtalálhatók ezek az általánosított tételek tömbfüggvények formájában.
- C#-ban ez az ún. LINQ metódusokon keresztül, a tömbök metódusaiként érhetők el.
- A saját megoldásunk mellett megmutatjuk az adott tétel LINQ megfelelőjét is.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

Tételek általánosított függvényei
összegzés

- Uf: `s=SZUMMA(i=1..n, x[i])`
- Példa:
  - Uf: `s=SZUMMA(i=1..n, szamok[i])`
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;

int s = Mintak.Szumma(szamok);
```

```
int s = szamok.Sum();
```

- Példa (transzformátorfüggvény):
  - Uf: `s=SZUMMA(i=1..n, diákok[i].jegy)`
  - Kód:

```
int s = Mintak.Szumma(diakok, diak => diak.jegy);
```

```
int s = diakok.Sum(diak => diak.jegy);
```

Tételek általánosított függvényei
általános összegzés

- Uf: `s=`**`SZUMMA`**`(i=e..u, x[i], kezd, add)`
- Példa:
  - Uf: `p=`**`SZUMMA`**`(i=1..n, szamok[i], 1, add)`
    `add(s,p)=s*p`
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;

int s = Mintak.Szumma(szamok, 1, (s, p) => s * p);
```

```
int s = szamok.Aggregate(1, (s, p) => s * p);
```

Tételek általánosított függvényei
megszámolás

- Uf: `db=DARAB(i=e..u, T(x[i]))`
- Példa:
  - Uf: `db=SZUMMA(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { 2, 4, -7, 5, 3, -2, 1 };
int n = szamok.Length;

int db = Mintak.Darab(szamok, e => e < 0);
```

```
int db = szamok.Count(e => e < 0);
```

Tételek általánosított függvényei
maximumkiválasztás

- Uf: (maxind,maxért)=**MAX**(i=1..n,x[i])
- Példa: (maxi,maxé)=**MAX**(i=1..n, szamok[i])
  - Kód:

```
int[] szamok = { 2, 4, 7, 5, 3, 2, 1 };
int n = szamok.Length;
(int maxi, int maxe) = Mintak.Max(szamok);
```

```
int maxe = szamok.Max();
```

- Példa (transzformátorfüggvény):
  - Uf: (ind,diák)=**MAX**(i=1..n, diákok[i].jegy)
  - Kód:

```
struct Diak { public string nev; public int jegy; }
(int ind,int diak) = Mintak.Max(diakok, diak => diak.jegy);
```

```
int maxe = diakok.Max(diak => diak.jegy);
```

- Példa (összehasonlító függvény):
  - Uf: (ind,diák)=**MAX**(i=1..n, diákok[i].jegy)
  - Kód:

```
(int ind,int diak) = Mintak.Max(diakok,
                                (d1,d2) => d1.jegy > d2.jegy);
```

LINQ-ban a Diak struct-nak implementálnia kellene az IComparable interfészt.

Tételek általánosított függvényei
feltételes maximumkeresés

- Uf: `(van,maxind,maxért)=FELTMAX(i=1..n,x[i],T(x[i]))`
- Példa: `(van,maxi,maxé)=FELTMAX(i=1..n,szamok[i],szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;
(bool van, int maxi, int maxe) = Mintak.FeltMax(szamok, e => e<0);
```

```
int maxe = szamok.Where(e => e < 0).Max();
```

- Példa (transzformátorfüggvény):
  - Uf: `(van,maxi,maxé)=FELTMAX(i=1..n, diakok[i].jegy,diakok[i].jegy<5)`
  - Kód:

```
struct Diak { public string nev; public int jegy; }
(bool van,int ind,int diak) = Mintak.FeltMax(diakok,
                              diak => diak.jegy, diak => diak.jegy<5);
```

```
int diak = diakok.Where(dk=>dk.jegy<5).Max(dk=>dk.jegy);
```

- Példa (összehasonlító függvény):
  - Uf: `(van,maxi,maxé)=FELTMAX(i=1..n, diakok[i].jegy, diakok[i].jegy<5)`
  - Kód:

```
(bool van,int ind,int diak) = Mintak.FeltMax(diakok,
          diak => diak.jegy<5, (d1, d2) => d1.jegy > d2.jegy);
```

Tételek általánosított függvényei
keresés

- Uf: `(van,ind)=KERES(i=1..n,T(x[i]))`
- Példa:
  - Uf: `(van,ind)=KERES(i=1..n,szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

(bool van,int ind) = Mintak.Keres(szamok, e => e < 0);
```

```
int ertek = szamok.FirstOrDefault(e => e < 0, 0);
```

Tételek általánosított függvényei
eldöntés

- Uf: `van=VAN(i=1..n,T(x[i]))`
- Példa:
  - Uf: `van=VAN(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;
bool van = Mintak.Van(szamok, e => e < 0);
```

```
bool van = szamok.Any(e => e < 0);
```

- Példa:
  - Uf: `mind=MIND(i=1..n, szamok[i]>0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;
bool mind = Mintak.Mind(szamok, e => e > 0);
```

```
bool van = szamok.All(e => e > 0);
```

Tételek általánosított függvényei
kiválasztás

- Uf: `ind=KIVÁLASZT(i>=e,T(x[i]))`
- Példa:
  - Uf: `ind=KIVÁLASZT(i>=1, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int ind = Mintak.Kivalaszt(szamok, e => e < 0);
```

```
int ertek = szamok.First(e => e < 0);
```

Tételek általánosított függvényei
másolás

- Uf: `y=MÁSOL(i=1..n, f(x[i]))`
- Példa:
  - Uf: `y=MÁSOL(i=1..n, abs(szamok[i]))`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int[] y = Mintak.Masol(szamok, e => Math.Abs(e));
```

```
int[] y = szamok.Select(e => Math.Abs(e)).ToArray();
```

- Példa (értékek és indexek):

```
int[] y = diakok
  .Select((e, i) => (ertek: e, ind: i))
  .Select(e => e.ind)
  .ToArray();
```

Tételek általánosított függvényei
kiválogatás

- Uf: `(db,y)=`**`KIVÁLOGAT`**`(i=1..n,T(x[i]),x[i])`
- Példa:
  - Uf: `(db,y)=`**`KIVÁLOGAT`**`(i=1..n, szamok[i]<0, szamok[i])`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;

int[] y = Mintak.Kivalogat(szamok, e => e < 0);
```

```
int[] y = szamok.Where(e => e < 0).ToArray();
```

- Példa:
  - Uf: `(db,y)=`**`KIVÁLOGAT`**`(i=1..n, szamok[i]<0, i)`
  - Kód:

```
int[] y = Mintak.Kivalogat(szamok, e => e < 0, (e, i) => i);
```

```
int[] y = szamok
  .Select((e,i) => (ertek: e, ind: i))
  .Where(e => e.ertek < 0)
  .Select(e => e.ind).ToArray();
```

Példa
legjobb jó tanuló

	1	2	3	m=4
1	5	5	5	3
2	5	4	4	4
3	5	3	2	4
4	5	4	5	5
n=5	5	5	4	4

Feladat:

Egy egész számokat tartalmazó mátrixban melyik az a sor, amiben csak 4-es és 5-ös van, és az összege a legnagyobb?

Specifikáció:

	1	2	3	m=4	
1	5	5	5	3	
2	5	4	4	4	17
3	5	3	2	4	
4	5	4	5	5	19
n=5	5	5	4	4	18

→ van=igaz
→ legjobb=4

```
Be: n∈N, m∈N, jegyek∈N[1..n,1..m]
Ki: van∈L, legjobb∈N
Fv: összeg:N->N,
    összeg(diák)=SZUMMA(tantárgy=1..m,jegyek[diák,tantárgy])
Fv: jó:N->L,
    jó(diák)=MIND(tantárgy=1..m,4<=jegyek[diák,tantárgy]<=5)
Ef: ∀sor∈[1..n]:(∀oszlop∈[1..m]:(1<=jegyek[sor,oszlop]<=5))
Uf: (van,legjobb,)=FELTMAX(diák=1..n,összeg(diák),jó(diák))
```

Specifikáció:

```
Be: n∈N, m∈N, jegyek∈N[1..n,1..m]
Ki: van∈L, legjobb∈N
Fv: összeg:N->N,
    összeg(diák)=SZUMMA(tantárgy=1..m,jegyek[diák,tantárgy])
Fv: jó:N->L,
    jó(diák)=MIND(tantárgy=1..m,4<=jegyek[diák,tantárgy]<=5)
Ef: ∀sor∈[1..n]:(∀oszlop∈[1..m]:(1<=jegyek[sor,oszlop]<=5))
Uf: (van,legjobb,)=FELTMAX(diák=1..n,összeg(diák),jó(diák))
```

```
static void Main(string[] args) {
  int n; int m; int[][] jegyek;
  bool van; int legjobb;

  (n, m, jegyek) = beolvas();
  (van, legjobb, _) = Mintak.Max(jegyek,
    diak => Mintak.Szumma(diak),
    diak => Mintak.Mind(diak, jegy => 4 <= jegy && jegy <= 5)
  );
  kiir(van, legjobb);
}
static (int n, int m, int[][] jegyek) beolvas() {
  int n, m;
  int[][] jegyek;

  Console.Write("Varazstanoncok szama = ");
  int.TryParse(Console.ReadLine(), out n);
  Console.Write("Jegyek szama = ");
  int.TryParse(Console.ReadLine(), out m);
  jegyek = new int[n][];
  for (int i = 1; i <= n; i++) {
    jegyek[i - 1] = new int[m];
    for (int j = 1; j <= m; j++) {
      Console.Write("{0}. varazstanonc {1} jegye = ", i, j);
      int.TryParse(Console.ReadLine(), out jegyek[i - 1][j - 1]);
    }
  }
  return (n, m, jegyek);
}
```

Specifikáció:

Be: n∈N, m∈N, jegyek∈N[1..n,1..m]
Ki: van∈L, legjobb∈N
Fv: összeg:N->N,
    összeg(diák)=**SZUMMA**(tantárgy=1..m,jegyek[diák,tantárgy])
Fv: jó:N->L,
    jó(diák)=**MIND**(tantárgy=1..m,4<=jegyek[diák,tantárgy]<=5)
Ef: ∀sor∈[1..n]:(∀oszlop∈[1..m]:(1<=jegyek[sor,oszlop]<=5))
Uf: (van,legjobb,)=**FELTMAX**(diák=1..n,összeg(diák),jó(diák))

```
static void Main(string[] args) {
  int n; int m; int[][] jegyek;
  bool van; int legjobb;

  (n, m, jegyek) = beolvas();
  (van, legjobb, _) = Mintak.Max(jegyek,
    diak => diak.Sum(),
    diak => diak.All(jegy => 4 <= jegy && jegy <= 5)
  );
  kiir(van, legjobb);
}
static (int n, int m, int[][] jegyek) beolvas() {
  int n, m;
  int[][] jegyek;

  Console.Write("Varazstanoncok szama = ");
  int.TryParse(Console.ReadLine(), out n);
  Console.Write("Jegyek szama = ");
  int.TryParse(Console.ReadLine(), out m);
  jegyek = new int[n][];
  for (int i = 1; i <= n; i++) {
    jegyek[i - 1] = new int[m];
    for (int j = 1; j <= m; j++) {
      Console.Write("{0}. varazstanonc {1} jegye = ", i, j);
      int.TryParse(Console.ReadLine(), out jegyek[i - 1][j - 1]);
    }
  }
  return (n, m, jegyek);
}
```

Összefoglalás

Minták általánosított függvényei

- Uf: `van=VAN(i=e..u,T(i))`
- Példa:
  - Uf: `van=VAN(i=1..n, szamok[i]<0)`
  - Kód:

```
int[] szamok = { -2, 4, 7, -5, 3, 2, -1 };
int n = szamok.Length;
```

```
bool van = Mintak.Van(1, n, i => szamok[i – 1] < 0);
```

```
bool van = Mintak.Van(szamok, e => e < 0);
```

```
bool van = szamok.Any(e => e < 0);
```