

The background of the slide is a black and white aerial photograph of Budapest, Hungary. The Danube River flows through the center of the city, with the Buda Peninsula on the left and the Pest side on the right. The Parliament Building is visible on the Buda Peninsula. A semi-transparent white rectangle is overlaid on the center of the image, containing the title text.

Programozás 12. előadás

Tartalom

- Programozási tételek általánosítása₂
- Programkészítési elvek
- Dokumentációk
- Hatékonyágvizsgálat táblázatkezelővel

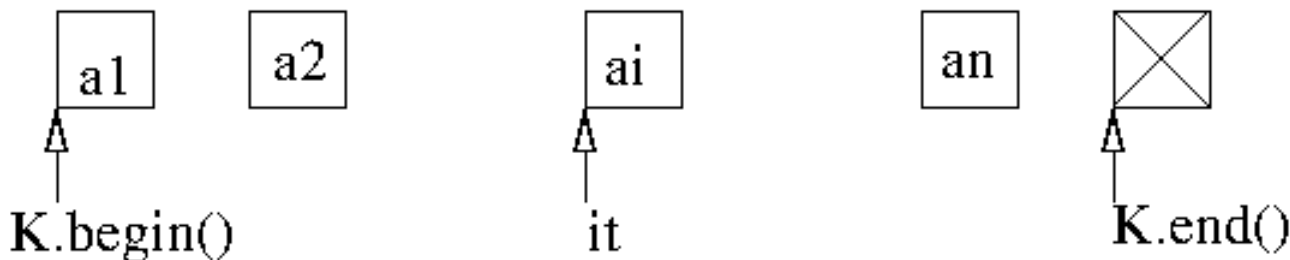


Programozási tételek általánosítása₂



Konténernek (tárolónak) nevezünk minden olyan összetett adattípust, amelynek elemei (összetevői) iterátorral bejárhatók.

Az iterátor egy referencia (hivatkozás, mutató), ami egy összetett adat valamely adatelemére való hivatkozás. Ha az összetett adat az a_1, \dots, a_n adatelemeket tartalmazza (ezekből épül fel), akkor létezik az adatelemeknek egy logikai (az adatelemek tárolásától független) sorrendje.

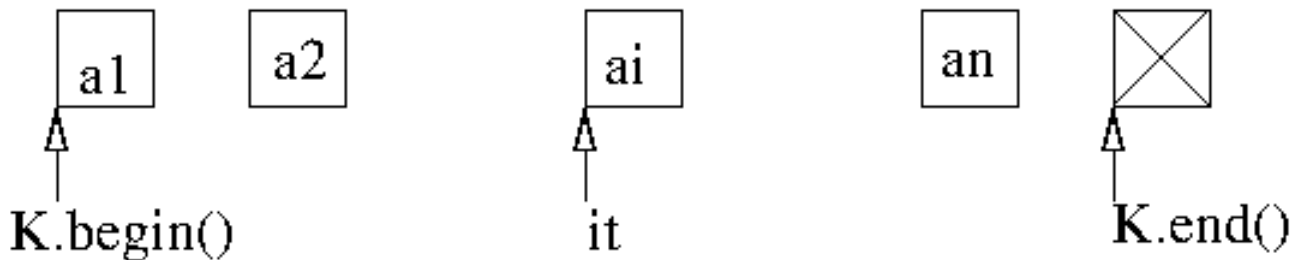


Programozási tételek általánosítása₂ – C++

Minden K konténer esetén $K.begin()$ a felsorolás első elemét tartalmazó cellára mutat, $K.end()$ pedig a felsorolás után álló fiktív cellára mutat. Ha K üres, akkor $K.end()=K.begin()$.

Egy it iterátor által mutatott cella tartalmára a C++-ban $*it$ dereferenciával hivatkozhatunk.

Az iterátorral való továbblépés a $++$ művelettel valósítható meg ($it++$).

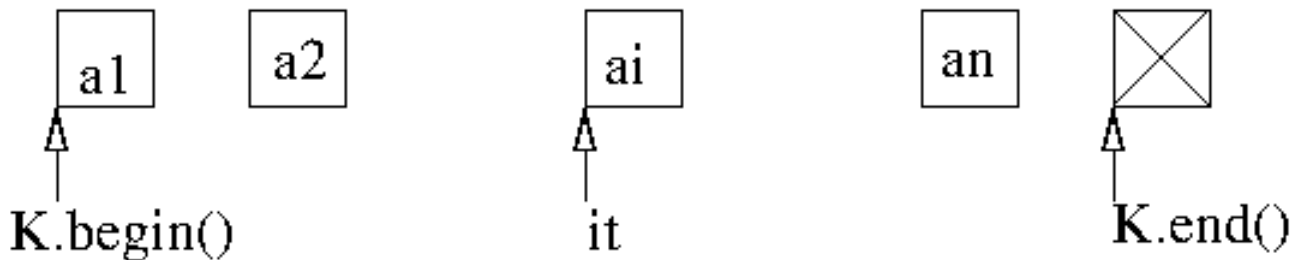


Programozási tételek általánosítása₂ – C++

```
it=K.begin();  
while (it!=K.end()) {  
    x=*it; //a kurzor által mutatott elem  
    M(x); //művelet elvégzése az adaton  
    it++; //továbblépés  
}
```

vagy

```
for (auto x:K)  
    M(x);
```

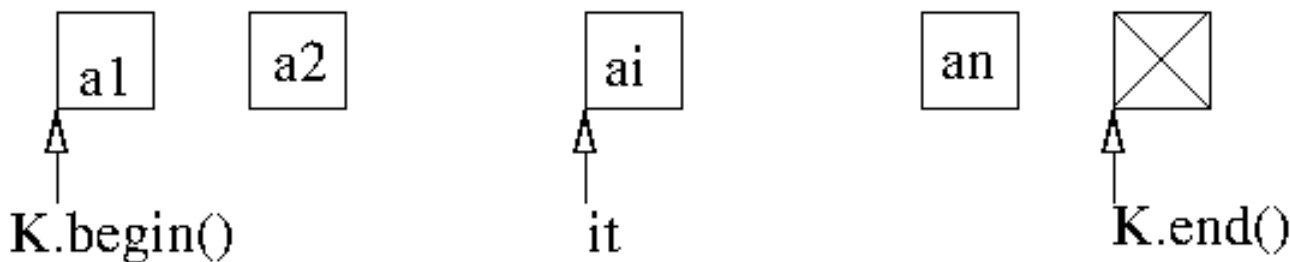


Programozási tételek általánosítása₂

Cél: a programozási tételeket tömbökön túl általánosan lehessen használni adatszerkezetek bejárására.

Módszer: függvények célszerű paraméterezése, iterátorok használata.

Elv: Az iterátor egy memóriahivatkozás, amellyel végig tudunk haladni egy struktúra (jellemzően sorozat) elemein.



Összegzés – általános sorozaton

Bemenet:	$X \in H^*$	esetleg még \rightarrow	$e, u \in \text{Cím}(H)$
Kimenet:	$S \in H$		
Előfeltétel:	–	\rightarrow	e, u X-beli cím
Utófeltétel:	$S = \sum_i X_i$	\rightarrow	$S = \sum_{i=e}^u X_i$

Az összegzés itt a sorozat elemeire (esetleg nem az összesre),
 elemeit tartalmazó képletre vonatkozik, és nem több elemet
 tartalmazó képletre (azaz $X_i * X_i$ lehet, $X_i * X_{i-1}$ nem).



Összegzés – általános sorozaton



Algoritmus (iterátorral):

e és u címek

Összeg($e, u, F0$)

$S := F0; x := e$

$x \neq u$

$S := S + \text{Adat}(x); \text{Következő}(x)$

Összeg := S

Változó
 x : Iter

e első, u utolsó utáni elem címét tartalmazza, $\text{Adat}(x)$ művelet az x címen levő adatot adja meg; $\text{Következő}(x)$ x -et a következőre lépteti.



Összegzés – általános sorozaton



C++ kód (iterátorral):

```
template<class InputIt, typename E>
E Osszeg(InputIt e, InputIt u, E F0) {
    E s=F0;
    while (e!=u) {
        s=s+*e; e++;
    };
    return s;
}
```

Paraméteres
alprogram =
„minta”

Cím
(referencia)

Két **típusparaméter**t kell alkalmazni, az egyik az iterátor típusa: **InputIt**, a másik pedig az összetett adat elemeinek **E** típusa (amire az iterátor hivatkozik).



Összegzés – általános sorozaton



C++ kód (iterátorral):

```
template<class InputIt, typename E>
E Osszeg(InputIt e, InputIt u, E F0) {
    E s=F0;
    while(e!=u) {
        s=s+*e; e++;
    };
    return s;
}
```

Használati
példák

```
...
array<int,9> AA; //9 elemű egészek tömbje
```

```
...
x=Osszeg(&AA[3], &AA[5], 0);
```

$x = 0 + AA[3] + AA[4]$

```
vector<int> V; //valahány elemű vektor
```

```
...
y=Osszeg(V.begin(), V.end(), 0);
```

$y = 0 + V[0] + \dots$



Összegzés – általános sorozaton vagy halmazon



Bemenet: $K \in H^*$ vagy $K \in 2^H$

(H -beli elemekből álló sorozat, vagy H egy részhalmaza)

Kimenet: $S \in H$

Előfeltétel: –

Utófeltétel: $S = \sum_{x \in K} x$

A bemenet egy tároló, amely elemeit a \sum -beli \in művelettel járhatjuk be, az összegzés itt is az elemekre vonatkozik.

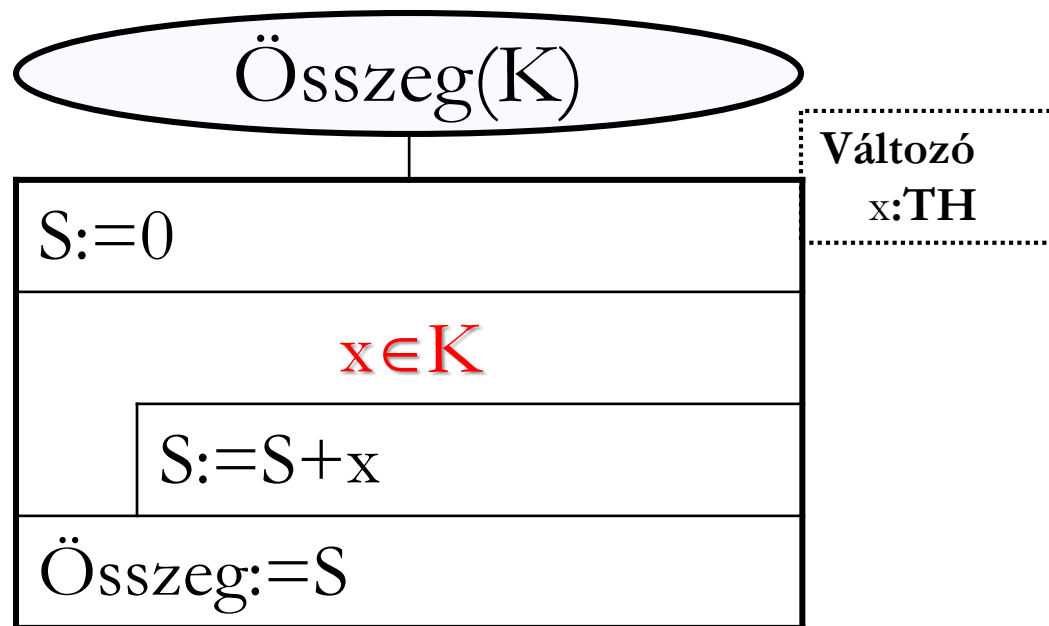


Összegzés – általános sorozaton vagy halmazon



Algoritmus (iterációs ciklussal):

Bemenet: $K \in \mathbf{H}^*$ vagy $K \in 2^{\mathbf{H}}$
(\mathbf{H} -beli elemekből álló sorozat, vagy \mathbf{H} egy részhalmaza)
Kimenet: $S \in \mathbf{H}$
Előfeltétel: –
Utófeltétel: $S = \sum_{x \in K} x$



Itt a paraméter az elemeket tartalmazó K tároló, amely elemeit az \in művelettel járhatjuk be: az \in művelet cikluslépésenként adja K elemeit. Ez egy újfajta számlálós ciklus.



Összegzés – általános sorozaton vagy halmazon



C++ kód (iterációs ciklussal):

```
template<class KontenerT>
typename KontenerT::value_type
Osszeg(const KontenerT& K,
        typename KontenerT::value_type kezd) {
    for (auto x:K)
        kezd=kezd+x;
    return kezd;
}
```

Az összegzés „mintafüggvénye”

Függvény-érték típus

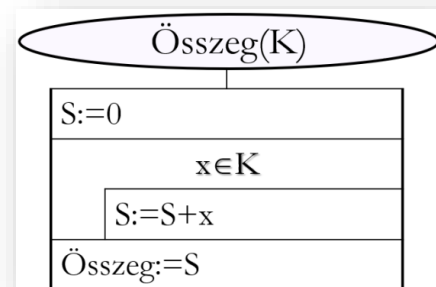
Tároló paraméter

kezdőérték paraméter

Tovább
általánosítjuk

Egy típusparamétert alkalmazunk, a konténer "**KontenerT**" típusát.

Minden összetett **T** típus esetén az adatelemek típusát jelenti a **T::value_type**.



Összegzés – általános sorozaton vagy halmazon



C++ kód (iterációs ciklussal):

```
template<class KontenerT>
typename KontenerT::value_type
Osszeg(const KontenerT& K,
        typename KontenerT::value_type kezd) {
    for (auto x:K)
        kezd=kezd+x;
    return kezd;
}
```

**Használati
példák**

```
...
array<int,9> AA; //9 elemű egészek tömbje
...
x=Osszeg(AA,0);
vector<int> V;   //valahány elemű vektor
...
y=Osszeg(V,0);
```

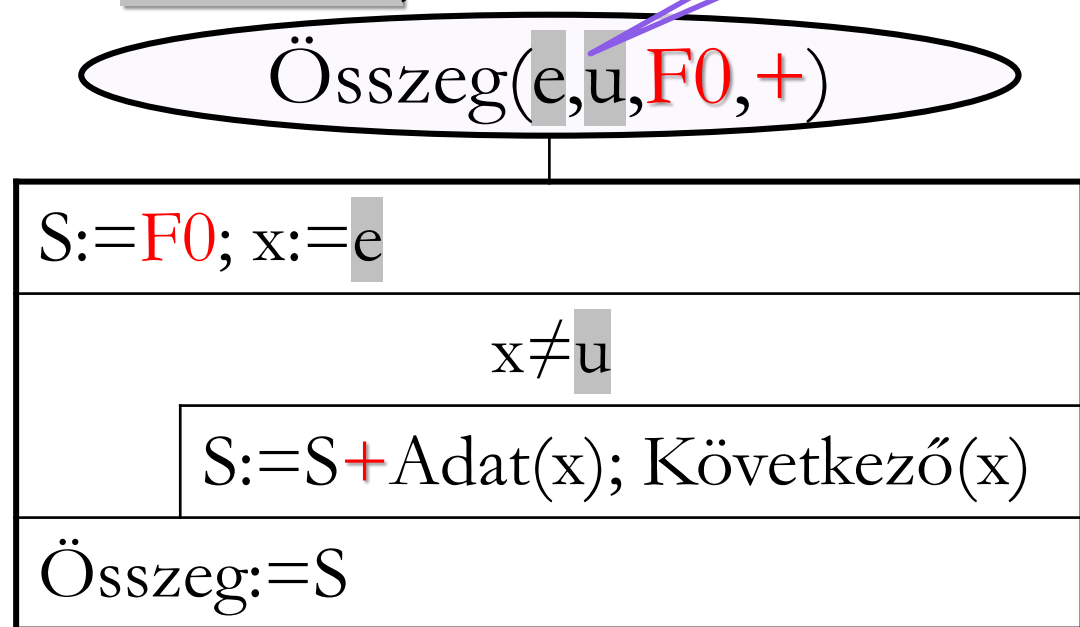


Összegzés – általános sorozaton



e és u címek

Algoritmus (általánosan, iterátorral):



Válto
x:It

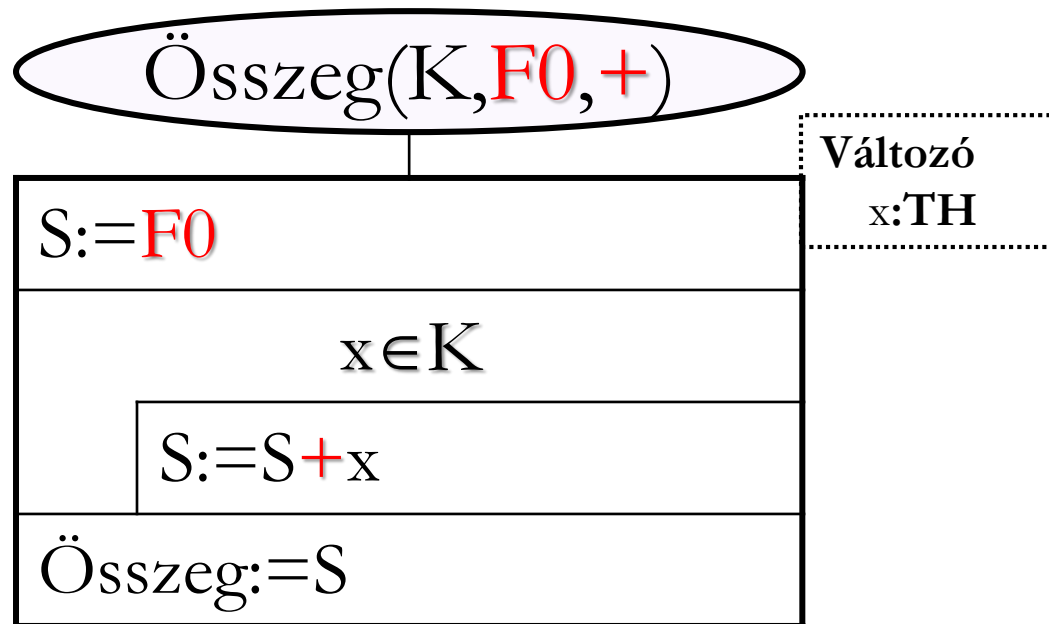
Itt paraméter a nullelem (F0) és a 2-változós művelet operátor (+). A Következő a következő elemre lépés 1-változós operátora.



Összegzés – általános sorozaton vagy halmazon



Algoritmus (általánosan, iterációs ciklussal):



Itt paraméter a nullelem ($F0$) és a 2-változós műveleti operátor ($+$).



Megszámolás – általános sorozat-intervallumon



Specifikáció:

Bemenet: $e, u \in \text{Cím}(H)$, $X \in H^*$
 $T: H \rightarrow L$

Kimenet: $D_b \in \mathbb{N}$

Előfeltétel: e, u X -beli elem címe

Utófeltétel: $D_b = \sum_{i=e}^u 1_{T(X_i)}$

Az iterátoros megoldás esetén a T tulajdonságnak egyes elemekre kell vonatkoznia!



Megszámolás – általános sorozat-intervallumon



e és **u** címek

Algoritmus (iterátorral, függvény paraméterrel):

Számol(**e**, **u**, **T**)

Változó
x:Iterátor

Db:=0; x:=**e**

$x \neq \mathbf{u}$

T(Adat(x))

Db:=Db+1

—

Következő(x)

Számol:=Db

Paraméterek: az első és az utolsó utáni elem **cím**, **T** a tulajdonság függvény. Az Adat(x): x címen levő adat; Következő(x): x iterátor a következőre lép.



Megszámolás – általános sorozat-intervallumon



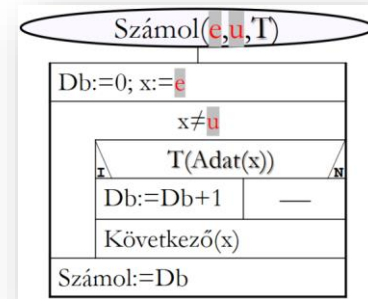
C++ kód (iterátorral, függvény paraméterrel):

```
template<class InputIt, typename E>
int Szamol(InputIt e, InputIt u, bool T(E)) {
    int db=0;
    while(e!=u) {
        if(T(*e)) db++;
        e++;
    };
    return db;
}
```

```
array<int,9> AA; //9 elemű egész tömb
vector<int> V;   //akárhány elemű vektor
```

...

```
x=Szamol(AA.begin(),AA.end(),paros);
y=Szamol(V.begin(),V.end(),paratlan);
```



Megszámolás – általános sorozat-intervallumon



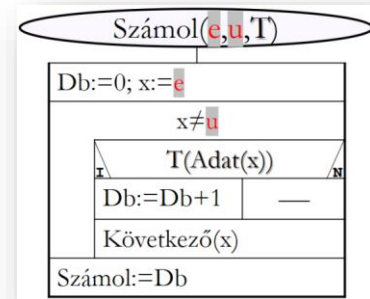
C++ kód (iterátorral, függvény paraméterrel):

```
template<class InputIt, typename E>
int Szamol(InputIt e, InputIt u, bool T(E)) {
    int db=0;
    while(e!=u) {
        if(T(*e)) db++;
        e++;
    };
    return db;
}
```

```
array<int, 9> AA; //9 elemű egész tömb
vector<int> V;    //akárhány elemű vektor
```

...

```
x=Szamol(&AA[0], &AA[9], paros);
y=Szamol(V.begin(), V.end(), paratlan);
```



Megszámolás – általános sorozaton vagy halmazon



Specifikáció:

Bemenet: $K \in \mathbf{H}^*$ vagy $K \in 2^{\mathbf{H}}$

$$T: \mathbf{H} \rightarrow \mathbf{L}$$

Kimenet: $D_b \in \mathbb{N}$

Előfeltétel: –

$$\text{Utófeltétel: } D_b = \sum_{\substack{x \in K \\ T(x)}} 1$$

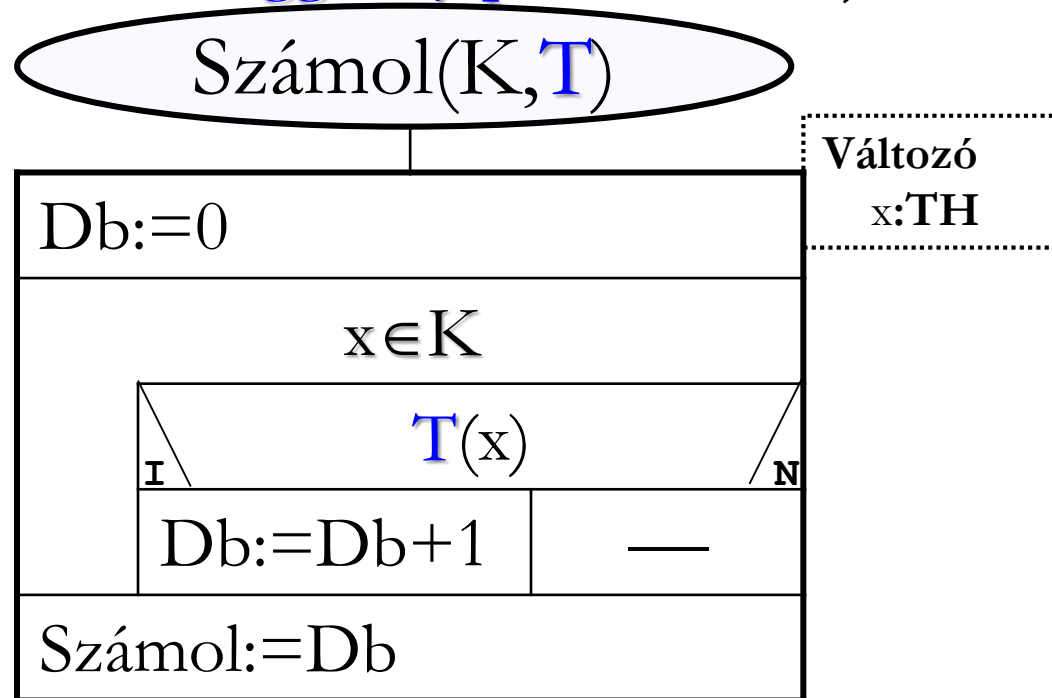
A konténeres megoldás esetén a T tulajdonságnak egyes elemekre kell vonatkoznia!



Megszámolás – általános sorozaton vagy halmazon



Algoritmus (iterációs ciklussal, **függvény paraméterrel**):



Paraméterek: az elemeket tartalmazó **K** tároló, amely elemeit az \in művelettel járhatjuk be; **T** tulajdonság függvény.



Megszámolás – általános sorozaton vagy halmazon



C++ kód (iterációs ciklussal, **függvény paraméterrel**):

```
template<class KontenerT>
int Szamol(const KontenerT& K,
           bool T(typename KontenerT::value_type)) {
    int db=0;
    for (auto x:K) {
        if(T(x)) db++;
    };
    return db;
}

array<int,9> AA; //tömb
vector<int> V;   //vektor
...
x=Szamol(AA,paros); y=Szamol(V,paratlan);
```



Maximum-kiválasztás – általános sorozat-intervallumon

Specifikáció (iterátorral):

Bemenet: $e, u \in \text{Cím}(H)$, $X \in H^*$

Kimenet: $\text{MaxÉrt} \in H$

Előfeltétel: $|X| > 0$ és

e, u X -beli elem címe és $e < u$

Utófeltétel: $e \leq \text{Max} \leq u$ és $\text{MaxÉrt} = X_{\text{Max}}$
 $\forall it (e \leq it \leq u): \text{MaxÉrt} \geq X_{it}$

Az $e \leq \dots \leq u$ a „haladási irányt”, a $|X|$ az X elemszámát jelenti.

Itt a maximális **érték** a fontos. (Elképzelhető eredményként a maximális elem memóriacíme is.)

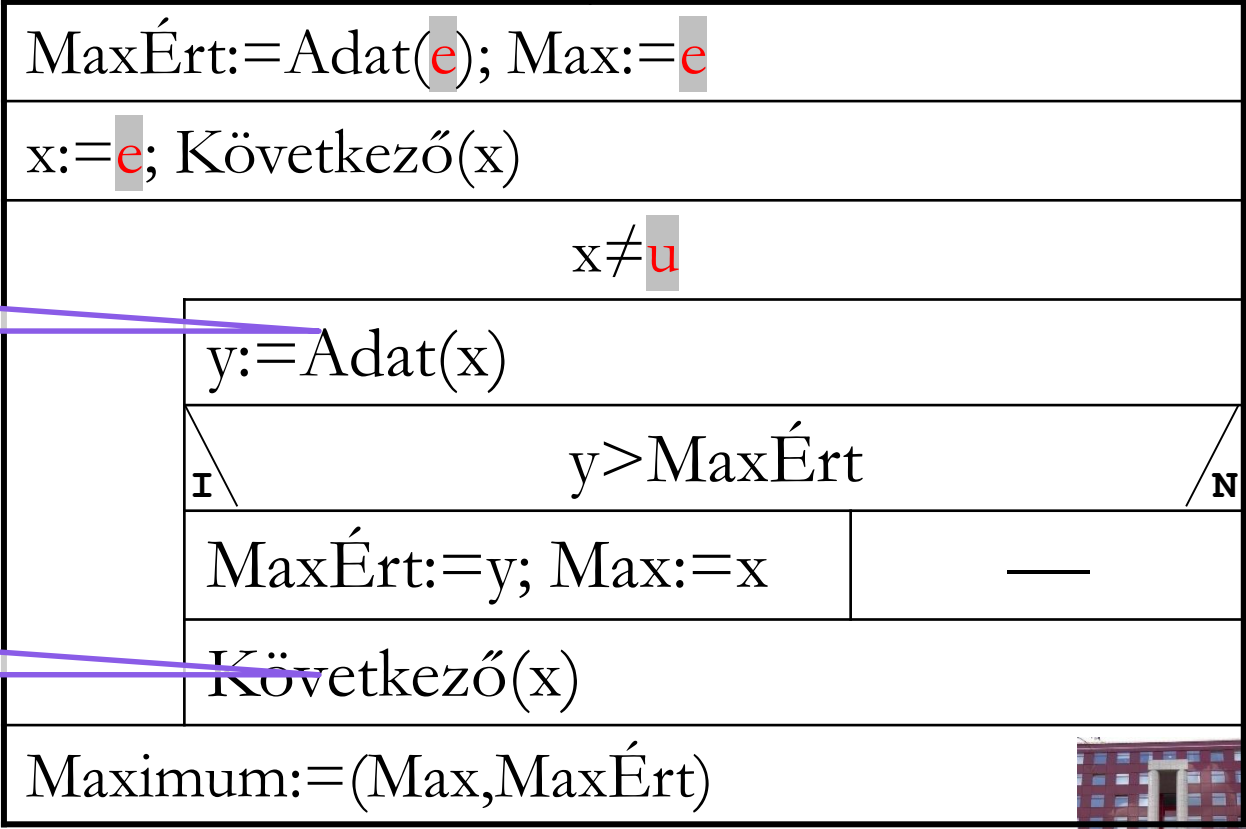
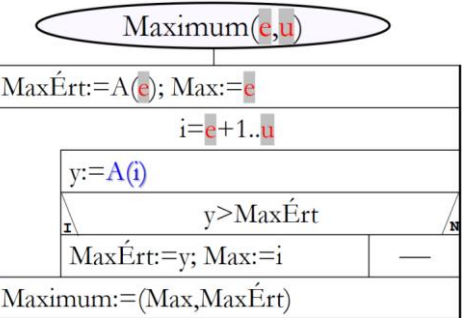


Maximum-kiválasztás – intervallumon

e és **u** címek

Algoritmus:

Maximum(**e**, **u**)



Adat(x): x **címen**
levő adat

Következő(x):
következő x **címe**



Maximum-kiválasztás – általános sorozaton vagy halmazon



Specifikáció (iterációs ciklussal):

Bemenet: $X \in H^*$ vagy $X \in 2^H$

Kimenet: **MaxÉrt** $\in H$

Előfeltétel: $|X| > 0$

Utófeltétel: **MaxÉrt** $\in X$ és $\forall y (y \in X): \text{MaxÉrt} \geq y$

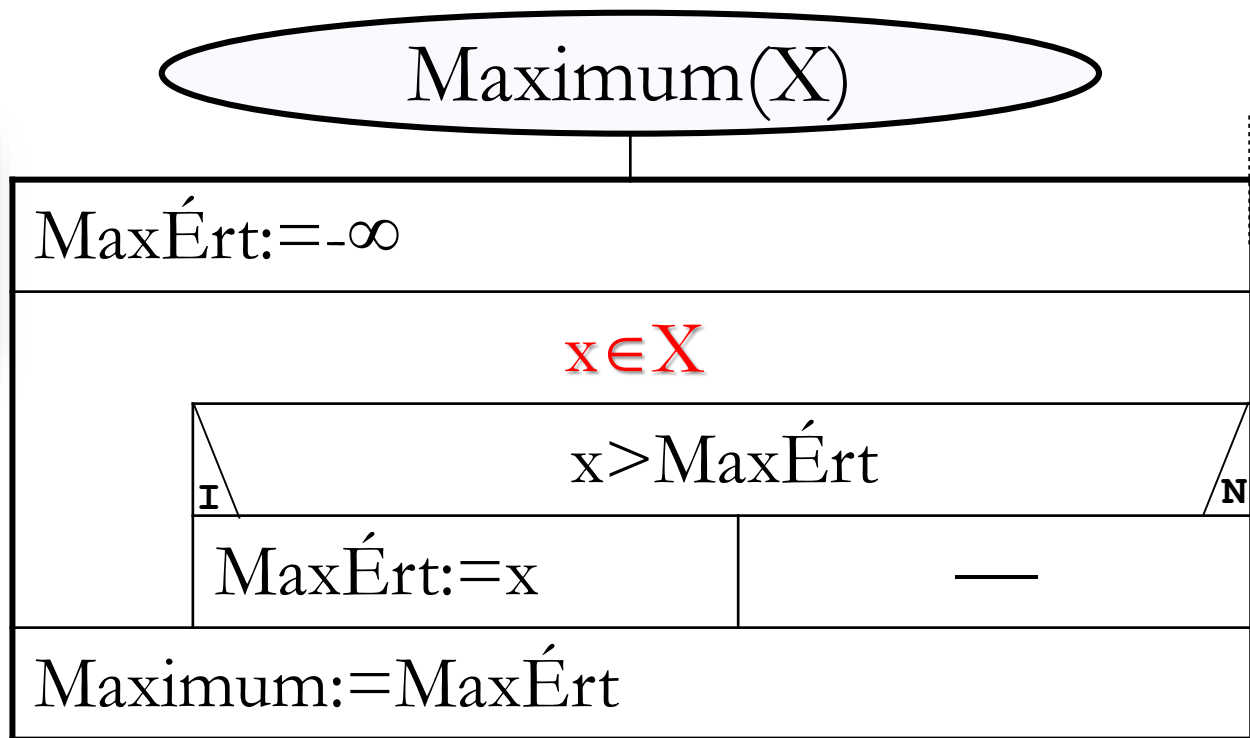
Itt csak a maximális **érték**nek van értelme, a helyének (címének) nincs.



Maximum-kiválasztás – általános sorozaton vagy halmazon

Algoritmus (iterációs ciklussal):

Specifikáció (iterációs ciklussal):
 Bemenet: $X \in H^*$ vagy $X \in 2^H$
 Kimenet: $\text{MaxÉrt} \in H$
 Előfeltétel: $|X| > 0$
 Utófeltétel: $\text{MaxÉrt} \in X$
 $\forall y (y \in X): \text{MaxÉrt} \geq y$



Változó
 $x: TH$



Kiválasztás – általános sorozat-intervallumon

Bemenet: $e, u \in \text{Cím}(\mathbf{H}), X \in \mathbf{H}^*,$
 $T: \mathbf{H} \rightarrow \mathbf{L}$

Kimenet: $\text{Ért} \in \mathbf{H}$

Előfeltétel: $|X| > 0$ és $\exists x (x \in X_{e..u}): T(x)$ és
 e, u X -beli elem címe és $e \leq u$

Utófeltétel: $\text{Ért} \in X_{e..u}$ és $T(\text{Ért})$

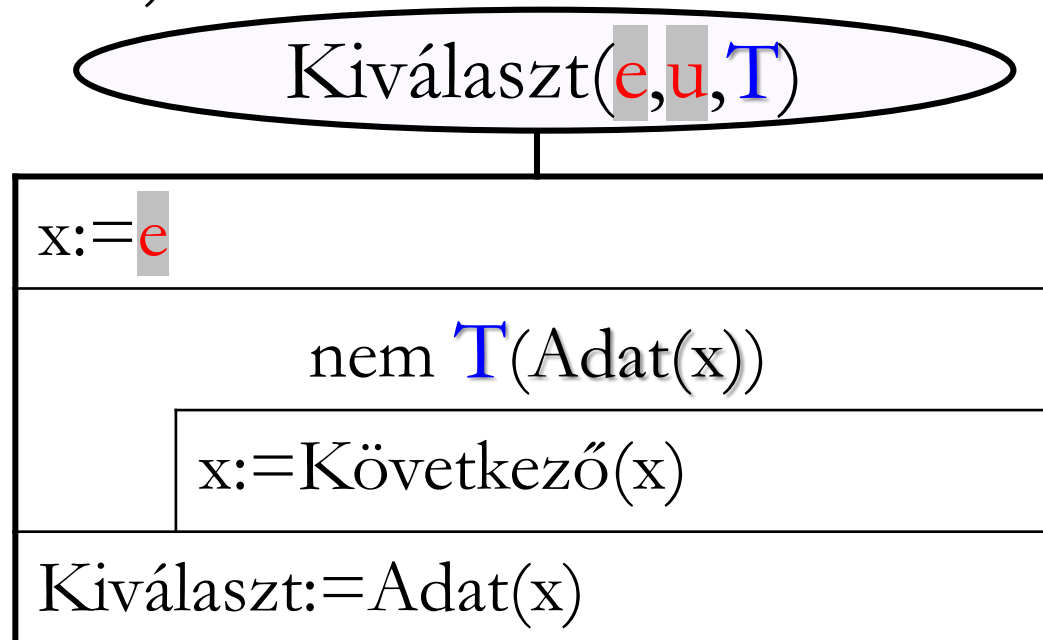
Itt is csak **érték** eredménynek van értelme, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.



Kiválasztás – általános sorozat-intervallum



Algoritmus (iterátorral):



Változó
 x : Iterátor

Az $\text{Adat}(x)$ az x memória **cím**en levő értéket jelöli, az e és az u a sorozat első és utolsó vizsgálandó eleme memória **címe**.



Kiválasztás – általános sorozat vagy **halmaz**

Bemenet: $X \in \mathbf{H}^N$ vagy $X \in \mathbf{2}^H$

$T: H \rightarrow L$

Kimenet: $\mathbf{\acute{E}rt} \in H$

Előfeltétel: $|X| > 0$ és $\exists x(\mathbf{x} \in X): T(x)$

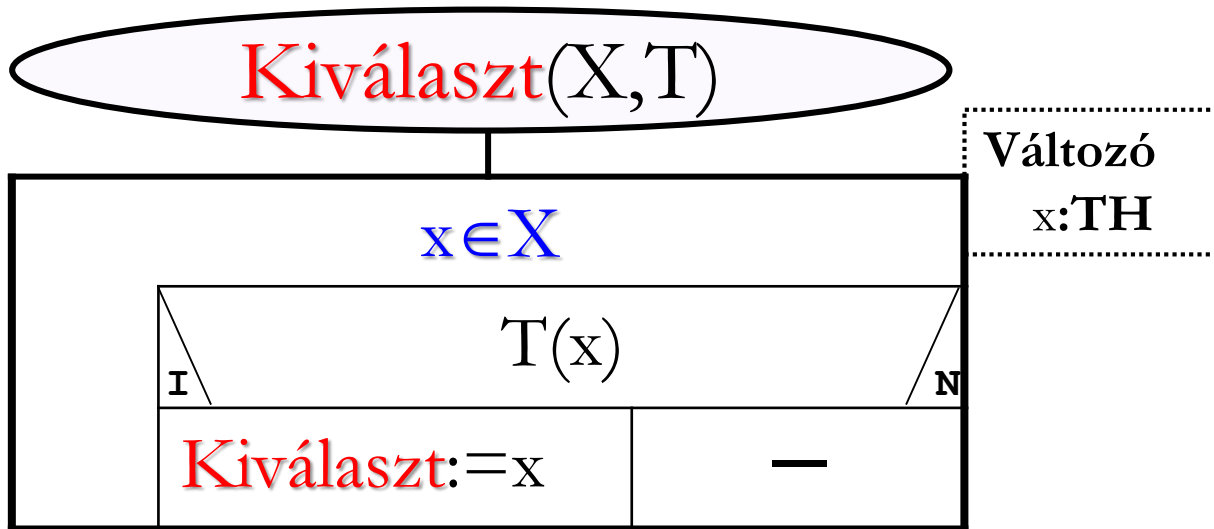
Utófeltétel: $\mathbf{\acute{E}rt} \in X$ és $T(\mathbf{\acute{E}rt})$

Itt is csak **érték** eredménynek van értelme, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.



Kiválasztás – általános sorozat vagy halmaz

Algoritmus (iterációs ciklussal)₁:



Itt az elveinknek ellentmondó, **szabálytalan** megoldásra kényszerülünk. A **függvény értékadás** egyben a függvény végrehajtásának **befejezését** is jelenti, azaz kilépést az $x \in X$ által szervezett (számlálós) ciklusból.

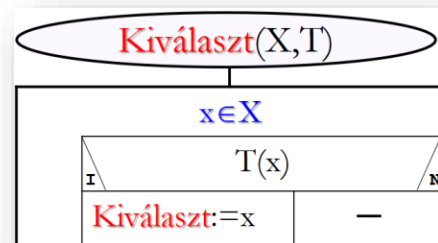


Kiválasztás – általános sorozaton vagy halmazon



C++ kód (iterációs ciklussal)₁:

```
template<class KontenerT>
typename KontenerT::value_type Kivalaszt(
    const KontenerT& K,
    bool T(typename KontenerT::value_type)) {
    for (auto x:K)
        if (T(x)) return x;
}
```

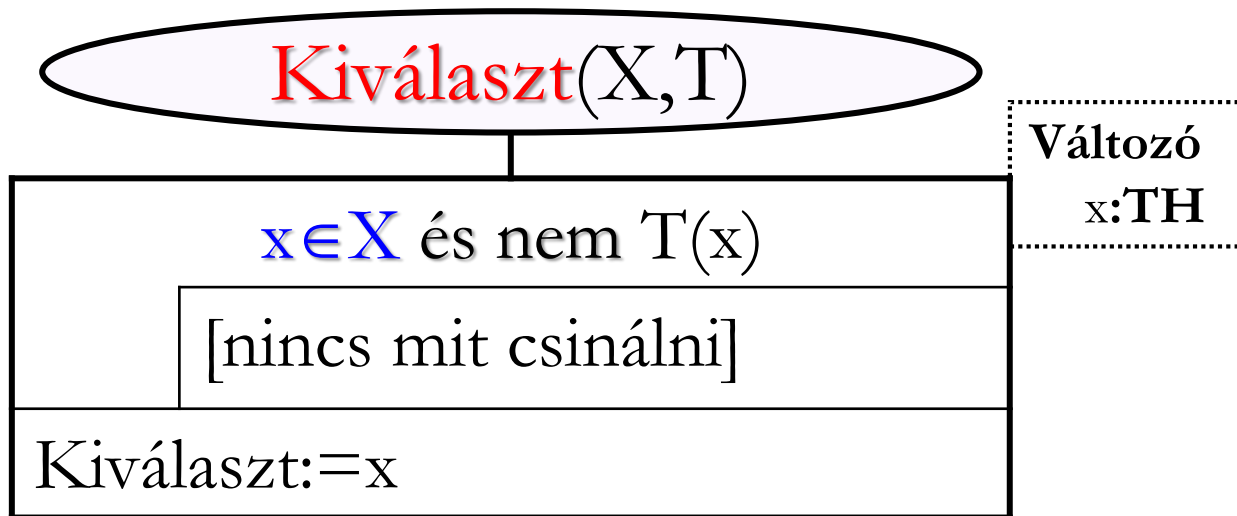


A **kódolási szabály** magyarázata: az $x:K$ által szervezett (számlálós) ciklusból (és a függvényből) kilépést a feltételnek eleget tévő elemhez éréskor a **return**-nel „erőszakoljuk ki”.



Kiválasztás – általános sorozat vagy halmaz

Algoritmus (iterációs ciklussal)₂:



E megoldásban a szabálytalanságot azzal a **szemantikus elvárással** kerüljük ki, hogy az $x \in K$ feladata „adagolni” a K -ban lévő elemeket, de a bennmaradás a $T(x)$ -től is függ. Ennek a kódolásához felhasználható az előbbi **kódolási szabály**.

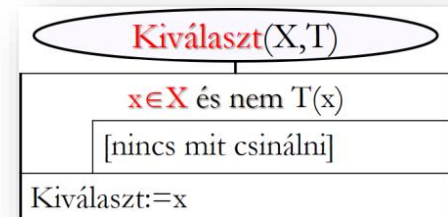


Kiválasztás – általános sorozaton vagy halmazon



C++ kód (iterációs ciklussal)₂:

```
template<class KontenerT>
typename KontenerT::value_type Kivalaszt(
    const KontenerT& K,
    bool T(typename KontenerT::value_type)) {
    typename KontenerT::value_type xU;
    for (auto x:K) {
        xU=x;
        if (T(x)) break;
    }
    return xU;
}
```



Az előbbi algoritmust jobban követő **kódolási szabály**.



Keresés – általános sorozat-intervallumon

Keresés (iterátorral):

Bemenet: $e, u \in \text{Cím}(H), X \in H^*,$

$T: H \rightarrow L$

Kimenet: $\text{Van} \in L, \text{Ért} \in H$

Előfeltétel: —

Utófeltétel: $\text{Van} = \exists x (e \leq x \leq u): T(x)$ és

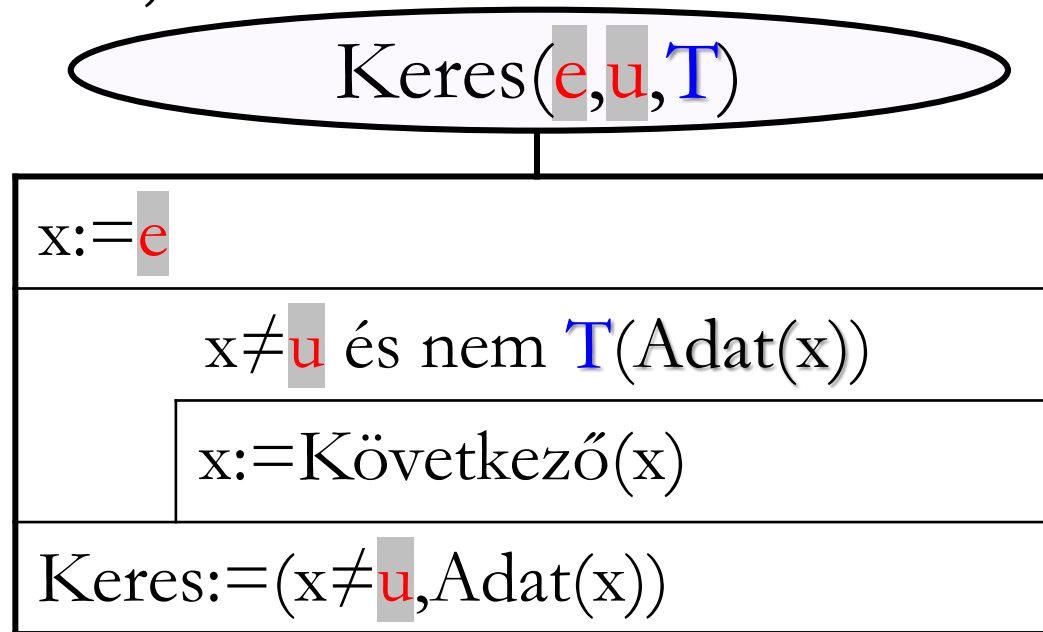
$\text{Van} \rightarrow \text{Ért} \in X$ és $T(\text{Ért})$

Itt is csak **érték** eredménynek van értelme, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.



Keresés – általános sorozat-intervallumon

Algoritmus (iterátorral):



Változó
x:Iterátor

Az $Adat(x)$ az x memória **cím**en levő értéket jelöli, az **e** és az **u** a sorozat első és utolsó vizsgálandó eleme memóriacíme.



Keresés – általános sorozaton vagy halmazon

Keresés (iterációs ciklussal):

Bemenet: $X \in \mathbf{H}^N$ vagy $X \in 2^H$
 $T: H \rightarrow L$

Kimenet: $\text{Van} \in L$, $\text{Ért} \in H$

Előfeltétel: —

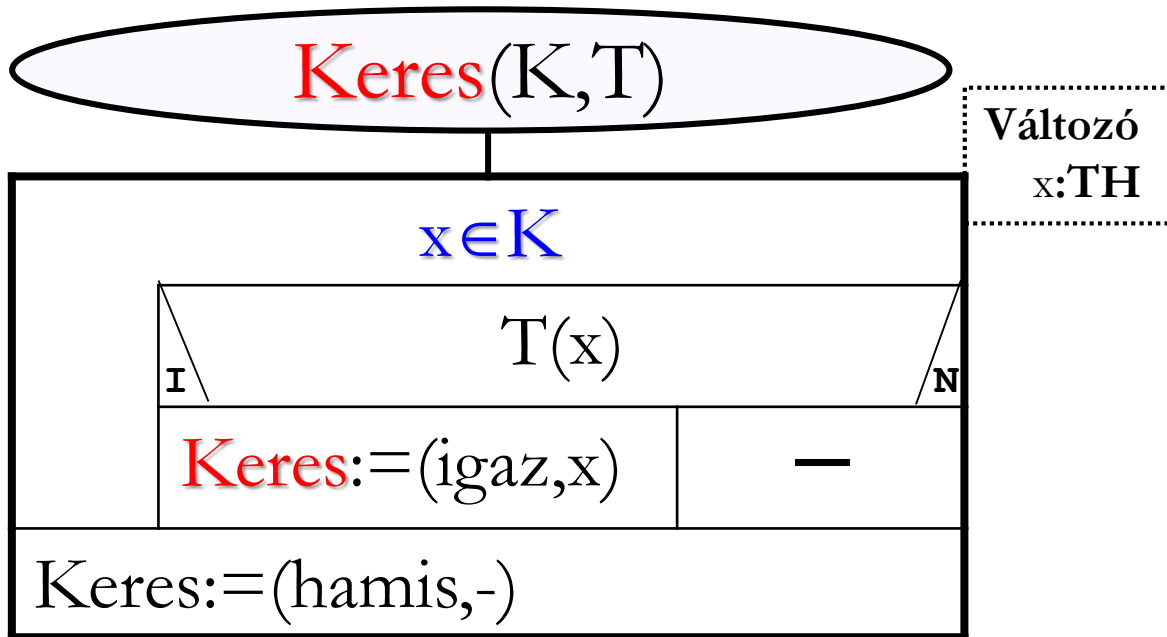
Utófeltétel: $\text{Van} = \exists x (x \in X): T(x)$ és
 $\text{Van} \rightarrow \text{Ért} \in X$ és $T(\text{Ért})$

Itt is csak **érték** eredménynek van értelme, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.



Keresés – általános sorozaton vagy halmazon

Algoritmus (iterációs ciklussal)₁:



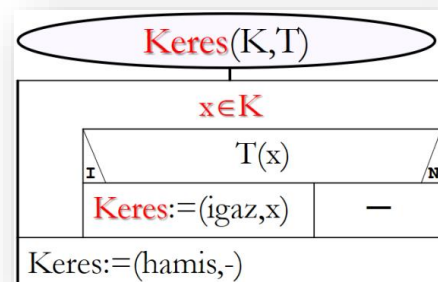
Itt az elveinknek ellentmondó, **szabálytalan** megoldásra kényszerülünk. A **függvény értékadás** a függvény végrehajtásának **befejezését** is jelenti, azaz kilépést az $x \in K$ által szervezett (számlálós) ciklusból.



Keresés – általános sorozaton vagy halmazon

C++ kód (iterációs ciklussal)₁:

```
template<class KontenerT>
void Keres(const KontenerT& K,
           bool T(typename KontenerT::value_type),
           bool& van,
           typename KontenerT::value_type& mi) {
    van=false;
    for (auto x:K) {
        if (T(x)) {
            van=true; mi=x; break;
        }
    }
}
```

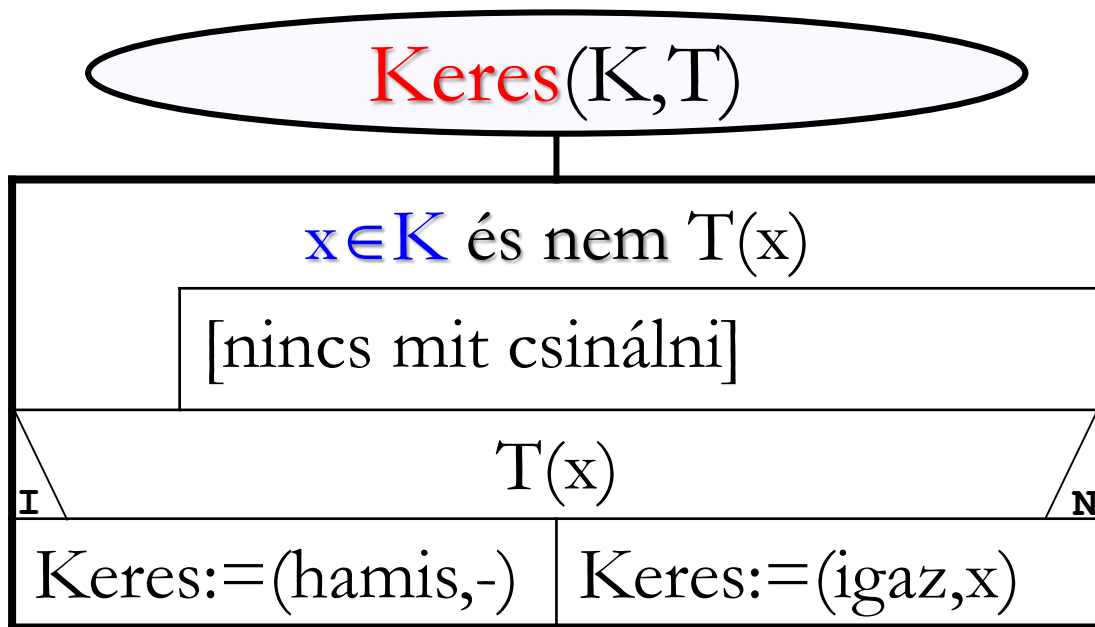


A **kódolási szabály** magyarázata: az $x:K$ által szervezett (számlálós) ciklusból kilépést a feltételnek eleget tévő elemhez éréskor egy **break**-kel „erőszakoljuk ki”.



Keresés – általános sorozaton vagy halmazon

Algoritmus (iterációs ciklussal)₂:



Változó
x:TH

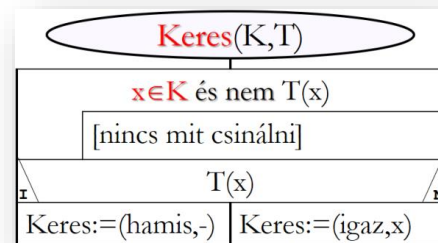
E megoldásban a szabálytalanságot azzal a **szemantikus elvárással** kerüljük ki, hogy az $x \in K$ feladata „adagolni” a K-ban lévő elemeket, de a bennmaradás a $T(x)$ -től is függ. Ennek a kódolásához felhasználható az előbbi **kódolási szabály**.



Keresés – általános sorozaton vagy halmazon

C++ kód (iterációs ciklussal)₂:

```
template<class KontenerT>
void Keres(const KontenerT& K,
           bool T(typename KontenerT::value_type),
           bool& van,
           typename KontenerT::value_type& mi) {
    typename KontenerT::value_type xU;
    for (auto x:K) {
        xU=x;
        if (T(x)) break;
    }
    if (T(xU)) {
        van=true; mi=xU;
    }else{van=false;};
}
```



Az algoritmust jobban követő **kódolási szabály**.



Programkészítési elvek

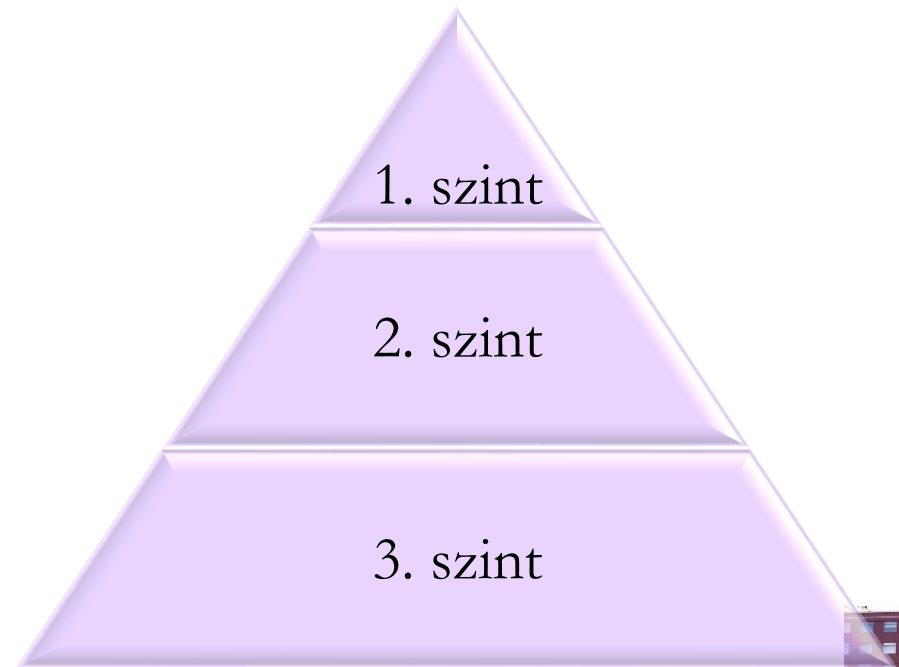
- **Stratégiai elv:** a problémamegoldás logikája – a lépésenkénti finomítás.
- **Taktikai elvek:** az algoritmuskészítés gondolati elvei a felülről lefelé kifejtéshez.
- **Technológiai elvek:** algoritmus és kód módszertani kívánalmak.
- **Technikai elvek:** kódolási technika.
- **Esztétikai, ergonómiai elvek:** emberközelség.



Stratégiai elv: lépésenkénti finomítás

- Felülről–lefelé (top–down) = probléma–dekomponálás, –analizálás.
- Alulról–felfelé (bottom–up) = probléma–szintézis.

**Nem
alternatívák!**



Taktikai elvek

- Párhuzamos finomítás
- Döntések elhalasztása
- Döntések nyilvántartása
- Vissza az ősökhöz
- Nyílt rendszer felépítés (általánosítás)
- Adatok elszigetelése (pl. alprogramokba helyezéssel + paraméterezéssel + lokális adatok deklarálásával)
- Párhuzamos ágak függetlensége
- Szintenkénti teljes kifejtés



Technológiai elvek az algoritmus készítéshez



- Struktúrák zárójelezése
- Bekezdéses struktúrák

Struktogram esetén ezek nyilvánvalóan teljesülnek

- Értelmes utasítás-csoportosítás
- Kevés algoritmusleíró szabály definiálása, de azok szigorú betartása (pl. tétel \rightarrow algoritmus)
- Beszédes azonosítók, kifejező névkonvenciók (pl. Hungarian Notation)



Technikai elvek a kódoláshoz



- Barátságosság (pl. kérdések, címek)
- Biztonságosság (pl. I/O-ellenőrzések)
- Kevés kódolási szabály definiálása, de azok következetes betartása (**algoritmus és kód koherenciája**; továbbá pl. amígos ciklusokhoz, I/O-hoz)
- Jól olvashatóság (vö.: Code::Blocks-ban a kódolási stílusok)



Esztétikai/ergonómiai elvek

- Lapokra tagolás, kiemelés, elkülönítés
- Menütechnika
- Ikontechnika, választás egérrel
- Következetesség (beolvasás, kiírás, ...)
- Hibafigyelés, hibajelzés, javíthatóság
- Súly, tájékoztató
- Ablakkezelés
- Értelmezési tartomány kijelzése
- Naplózás



Dokumentációk



Fajtái:

- *Programismertető*
- Felhasználói dokumentáció
- Fejlesztői dokumentáció
- ...

Dőlten szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.



Felhasználói dokumentáció

E nélkül be sem adható!

Tartalma:

- **feladatszöveg** (*összefoglaló és részletes is*)
- futási környezet (szg.+or.+*hw/sw-elvárások*)
- használat leírása (*telepítés, kérdések + lehetséges válaszok,...*)
- bemenő adatok, eredmények, *szolgáltatások*
- mintaalkalmazások – példafutások
- hibaüzenetek és a hibák lehetséges okai

Dőlten szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.



Fejlesztői dokumentáció

Tartalma:

- **feladatszöveg**, specifikáció, *követelményanalízis*
- fejlesztői környezet (or.+fordító program, ...)
- adatleírás (feladatparaméterek reprezentálása)
- algoritmusok leírása, döntések (pl. tételekre utalás), *más alternatívák, érvek, magyarázatok*
- kód, *implementációs szabványok*, ~ *döntések*
- tesztesetek
- *hatékonysági mérések*
- fejlesztési lehetőségek
- **szertő**(k)

E nélkül be sem adható!

Dőlten szedve, ami az aktuális nagy program estén a dokumentációból elhagyható.

Értelmesen strukturálva.

Szerző

Név: Szabó Emerencia

ETR-azonosító: SZEKAAT.ELTE

Neptun-azonosító: ESZ98A

Drótposta-cím: sze@elte.hu

Kurzus kód: IP-08PAEG/77

Gyakorlatvezető neve: Kiss-József Alfréd

Feladatsorszám: 18

E nélkül be sem adható!



Hatékonyágvizsgálat táblázatkezelővel



Ötlet:

1. A táblázatkezelők importálnak sokféle formátumú fájlt, pl. **CSV**-formátumút.
2. A **C**omma **S**eparated **V**alue (CSV) = egy „mezei” text fájl, amelyben minden önálló (cellában tárolt/tárolandó) adatot **(pontos)vessző** követ.
3. Egyszerű olyan **C++** programbetétet írni, amely a táblázatolandó adatokat „CSV-esítve” ír text fájlba.



Hatékonyágvizsgálat táblázatkezelővel



Példafeladat:

Az unió és az összefuttatás tételek hatékonyságának összevetése.

Hatékonysági „dimenzió”:

tömbbeli elemek **hasonlításszáma** esetleg futási ideje (mint a futás jellemzője)

Összefüggést keresünk a bemeneti **sorozathossz és hasonlításszám között:**

$(N,M) \rightarrow \text{hDb}_{\text{unió}}, (N,M) \rightarrow \text{hDb}_{\text{összefuttatás}}$

...



Hatékonyságvizsgálat táblázatkezelővel



Megoldásvázlat:

1. Mindkét algoritmusban számoljuk a tömbelem-összehasonlításokat (mérjük az időt).
2. Néhány (jól kiválasztott) N, M -elemű sorozatra lefuttatjuk és közben számlálunk (mérünk).
3. Majd CSV-fájlba írjuk a hatékonysági eredményeinket.

Megjegyzés: az időmérés feltétele, hogy pontosan tudjuk mérni. (Windows-ban aggályos, Unix/Linuxban OK.)



Hatékonyágvizsgálat táblázatkezelővel



Egy lehetséges eredmény a táblázatkezelőbe importálás után – **unió**:
Numerikusan

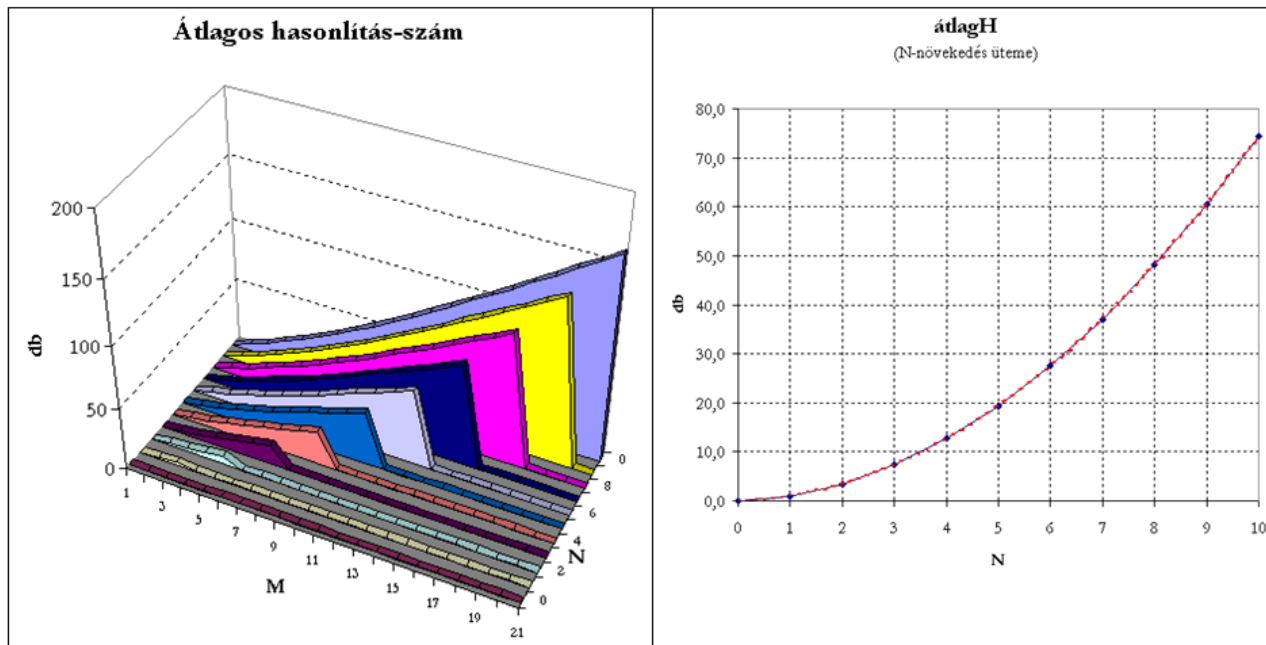
M		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
N	0	0																				
1	0	0	1	2																		
2	0	0	2	3	5	7																
3	0	0	2	4	7	10	13	16														
4	0	0	3	5	9	12	16	19	23	27												
5	0	0	3	6	10	14	18	23	28	32	37	42										
6	0	0	4	7	12	16	21	26	31	37	43	48	54	60								
7	0	0	4	8	13	18	23	29	35	41	48	54	61	67	74	81						
8	0	0	5	9	15	20	26	32	39	45	52	60	67	74	82	90	97	105				
9	0	0	5	10	16	22	28	35	42	49	57	65	73	81	90	98	107	115	124	132		
10	0	0	6	11	18	24	31	38	46	53	62	70	79	88	97	106	115	125	134	144	153	163

A program outputja táblázatkezelőbe importálás és némi szépítés után.



Hatékonyságvizsgálat táblázatkezelővel

Egy lehetséges eredmény a táblázatkezelőbe importálás után – **unió**:
Grafikusan



A program outputjának ábrázolása összetett felület-diagramokkal.

A program outputja és abból (rögzített N-hez tartozó) soronként képzett átlagok diagramja.



Hatékonyságvizsgálat táblázatkezelővel

Egy lehetséges eredmény
a táblázatkezelőbe importálás után – **összefuttatás:**
Numerikusan

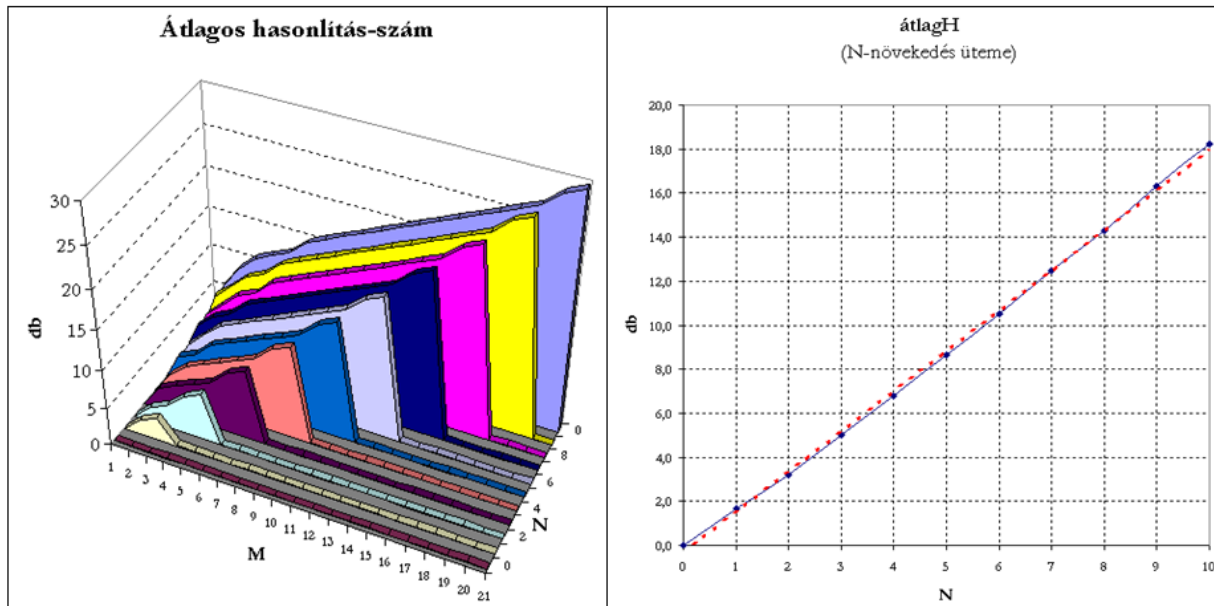
M N																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0																				
1	0	2	3																		
2	0	2	3	5	6																
3	0	3	4	5	6	8	9														
4	0	3	5	6	7	8	9	11	12												
5	0	4	5	7	8	9	10	11	12	14	15										
6	0	4	6	8	9	10	11	12	13	14	15	17	18								
7	0	5	7	8	10	11	12	13	14	15	16	17	18	20	21						
8	0	5	7	9	10	12	13	14	15	16	17	18	19	20	21	23	24				
9	0	6	8	10	11	13	14	15	16	17	18	19	20	21	22	23	24	26	27		
10	0	6	9	11	12	13	15	16	17	18	19	20	21	22	23	24	25	26	27	29	30

A program outputja táblázatkezelőbe importálás és némi szépítés után.



Hatékonyságvizsgálat táblázatkezelővel

Egy lehetséges eredmény
a táblázatkezelőbe importálás után – **összefuttatás:**
Grafikusan



A program outputjának ábrázolása
összetett felület-diagramokkal.

A program outputja és abból (rögzített N-
hez tartozó) soronként képzett átlagolok diagramja. Az N
növekedést mutató ábrán a legjobban ráilleszhető lineáris
egyenest egyenes gráfja látható piros pontokkal.

