# Programozás
# 12. előadás

# Tartalom



# Programozási tételek további általánosításai

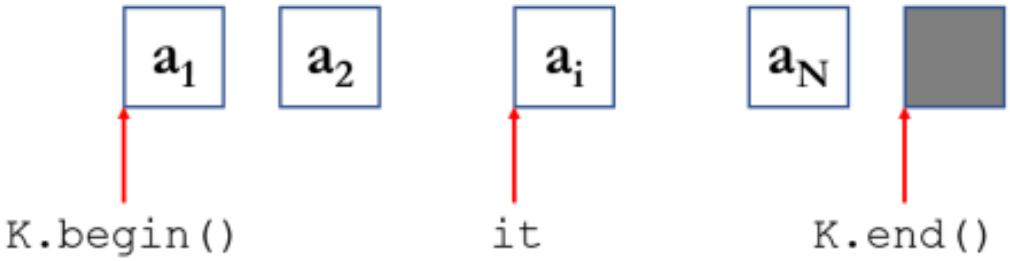

- Tételek általános(ított) sorozatokon = tárolókon
  - sokféle tároló (tömb, vektor, lista…),
  - indexelhetők (=felsorolhatók),
  - rész-sorozatra (=index-intervallumra)

- Tételek sokaságokon =
  - összes elemen,
  - nem várjuk el az indexelhetőséget (sorozat helyett halmaz)

# Programozási tételek
## általános sorozatokon

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

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



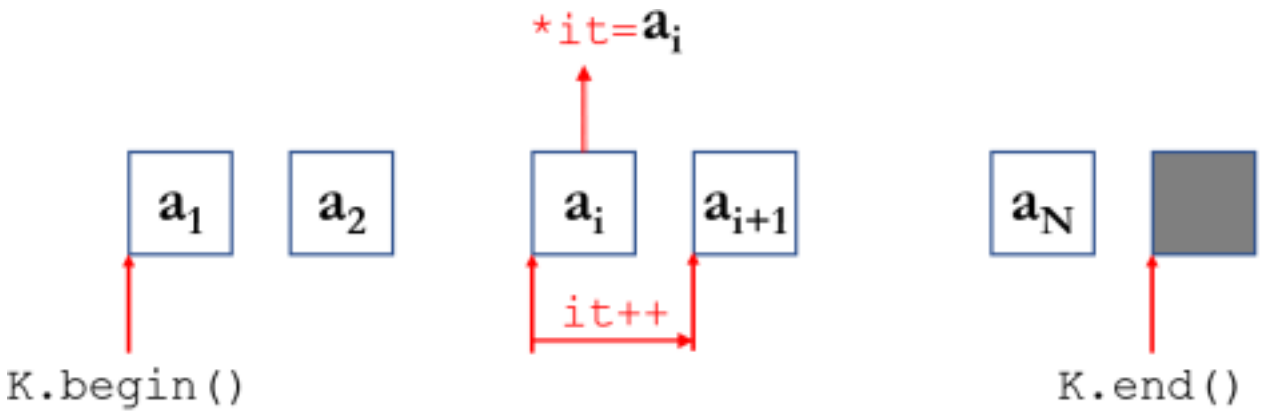

K:konténer, it:iterátor

# Programozási tételek általános sorozatokon – 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++).




K:konténer, it:iterátor

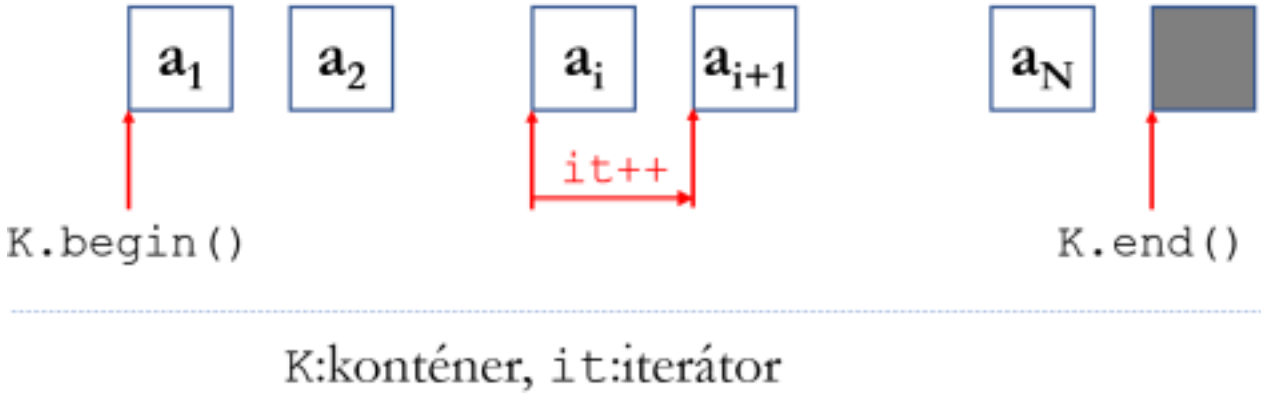
# Programozási tételek általános sorozatokon – C++

Egy konténer feldolgozása:

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

vagy

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



# Programozási tételek általánosítása$_2$

**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: $X \in \mathbb{H}^*$ esetleg még → $X \in \mathbb{H}^*$, $e,u \in$ Cím(H)

Kimenet: $S \in \mathbb{H}$

Előfeltétel: – → e,u X-hez tartozó cím

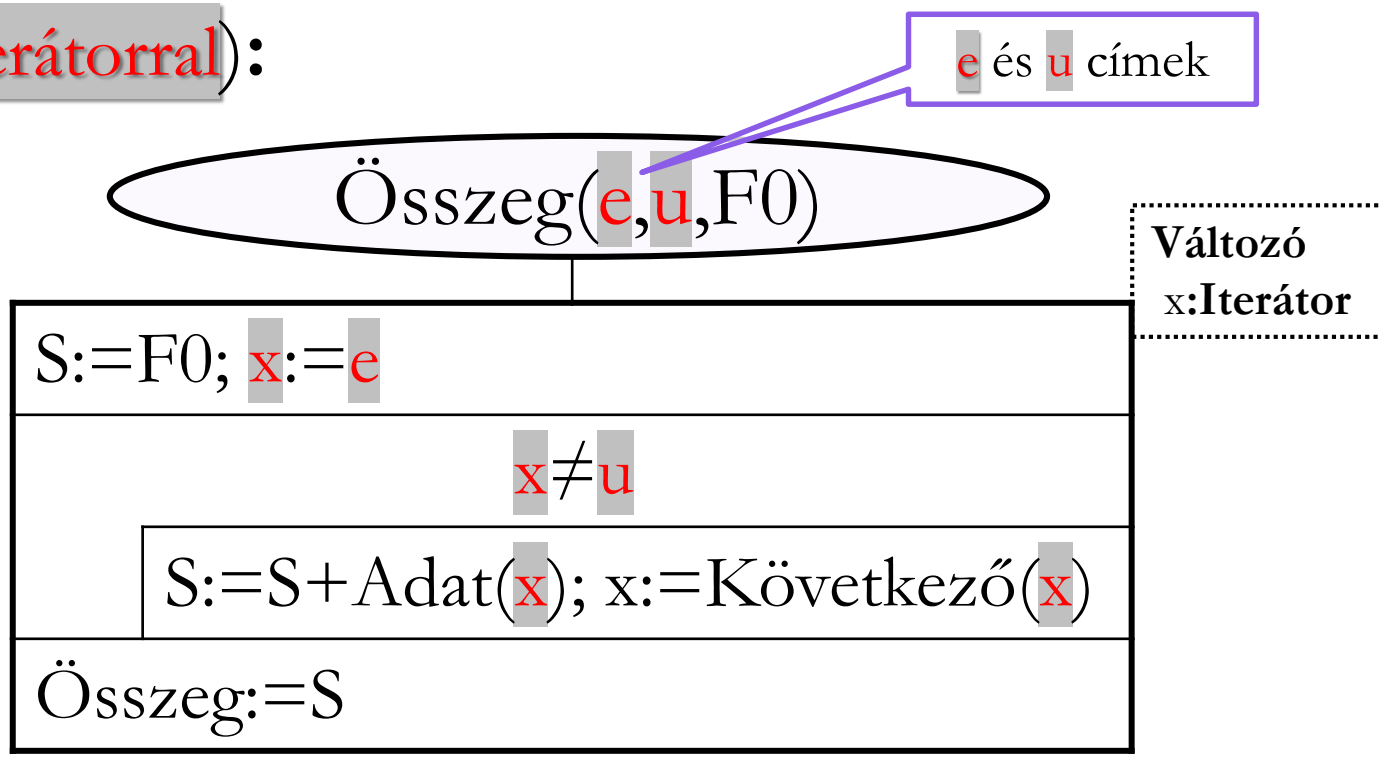
Utófeltétel: $S = \sum_{i} X_i$ → $S = \sum_{i=e}^{\text{Előző(u)}} X_i$

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).

Előző(u): az u címen kezdődő elemet megelőző elem címe.

# Összegzés – általános sorozaton

**Algoritmus** (iterátorral):




e első, u utolsó utáni elem címét tartalmazza, Adat(x) művelet az x címen levő adatot adja meg; Következő(x) függvény az x után következő címét adja értékül.

# Összegzés – általános sorozaton

**C++ kód** (iterátorral):

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

**Típusparaméteres alprogram = „minta”**

**Cím (referencia)**

Két típusparamétert kell alkalmazni, az egyik az iterátor típusa: `InputIt` (azaz egy nem megváltoztatható elemű konténer-elem referencia-típusa), a másik pedig az összetett adat elemeinek `E` típusa (amire az iterátor hivatkozik).

# Összegzés – általános sorozaton

## C++ kód (iterátorral):

```
template<typename InputIt, typename E>
E Osszeg(InputIt e,InputIt u,E F0){
   E s=F0;
   while(e!=u){
      s=s+*e; e++;
   };
   return s;
}
…
array<int,9> AA; //9 elemű egészek tömbje
…
x=Osszeg(&AA[3],&AA[5],0);

vector<int> V;   //valahány elemű vektor
…
y=Osszeg(V.begin(),V.end(),0);
```

**Használati példák**

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

**y=0+V[0]+…**

# Összegzés – általános sorozaton vagy halmazon


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

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

Kimenet: $S \in \mathbb{H}$

Előfeltétel: –

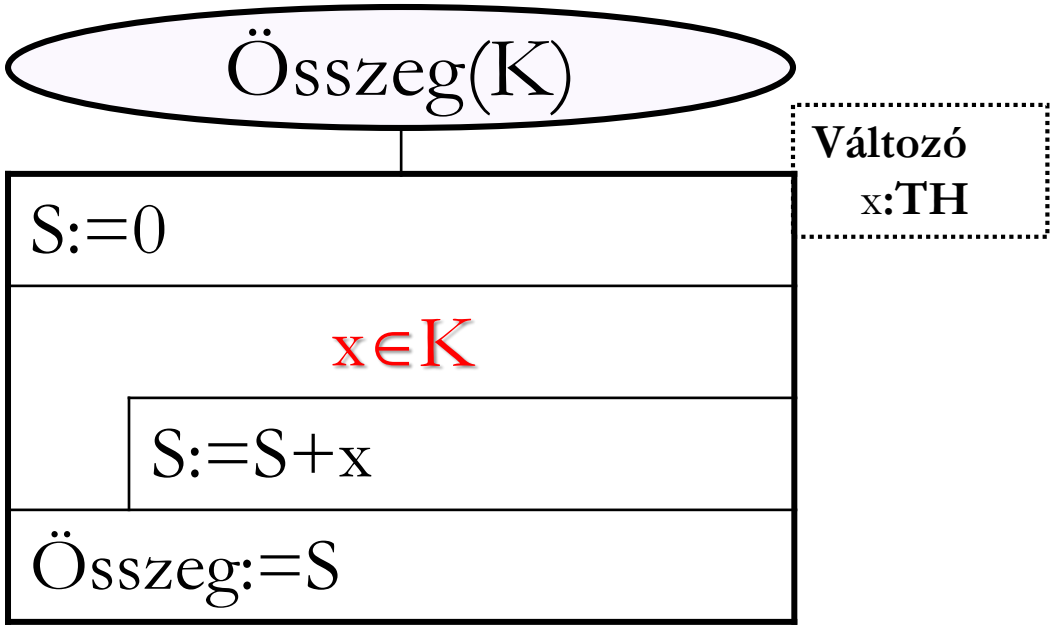
Utófeltétel: $S = \sum_{x \in K} x$

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




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

# Összegzés – általános sorozaton vagy halmazon

**C++ kód** (iterációs ciklussal):

```
template<typename 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

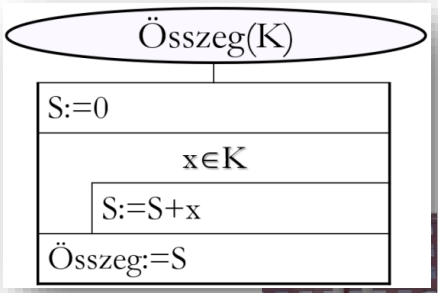
Tároló elem típusa

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):

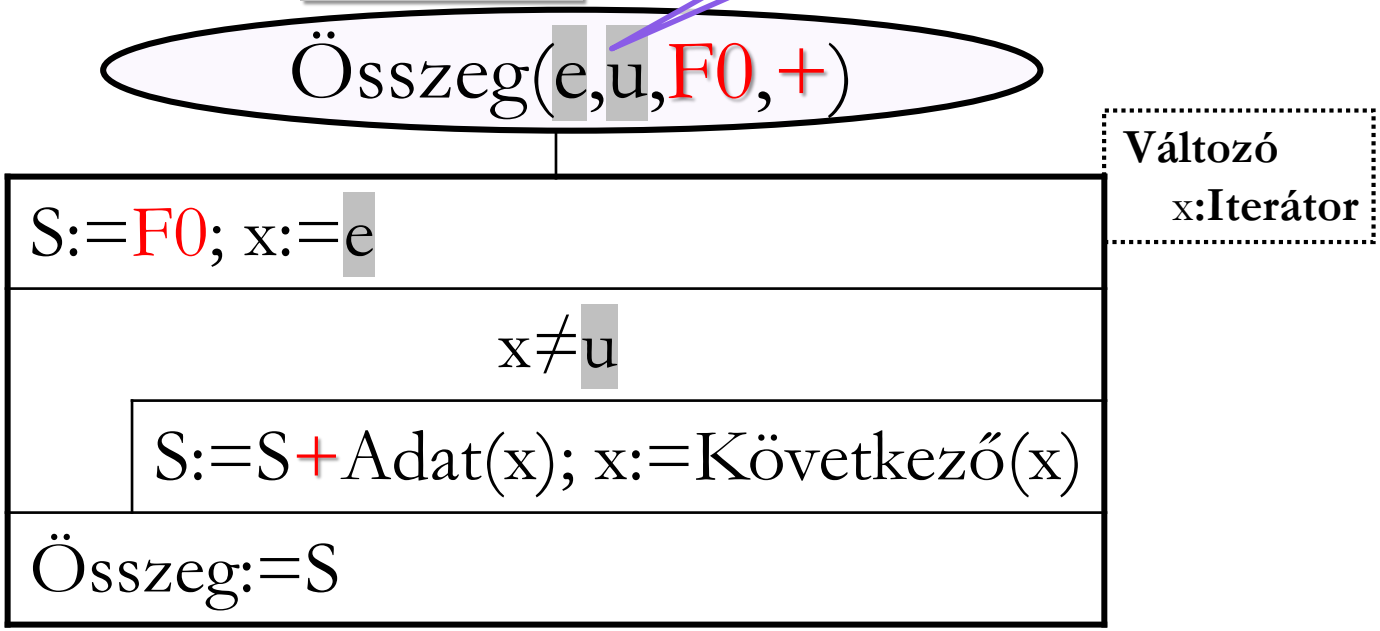
```
template<typename KontenerT>
typename KontenerT::value_type
  Osszeg(const KontenerT& K,
         typename KontenerT::value_type kezd){
   for (auto x:K)
      kezd=kezd+x;
   return kezd;
}
…
array<int,9> AA; //9 elemű egészek tömbje
…
x=Osszeg(AA,0);
vector<int> V;   //valahány elemű vektor
…
y=Osszeg(V,0);
```

**Használati példák**

# Összegzés – általános sorozaton

e és u címek

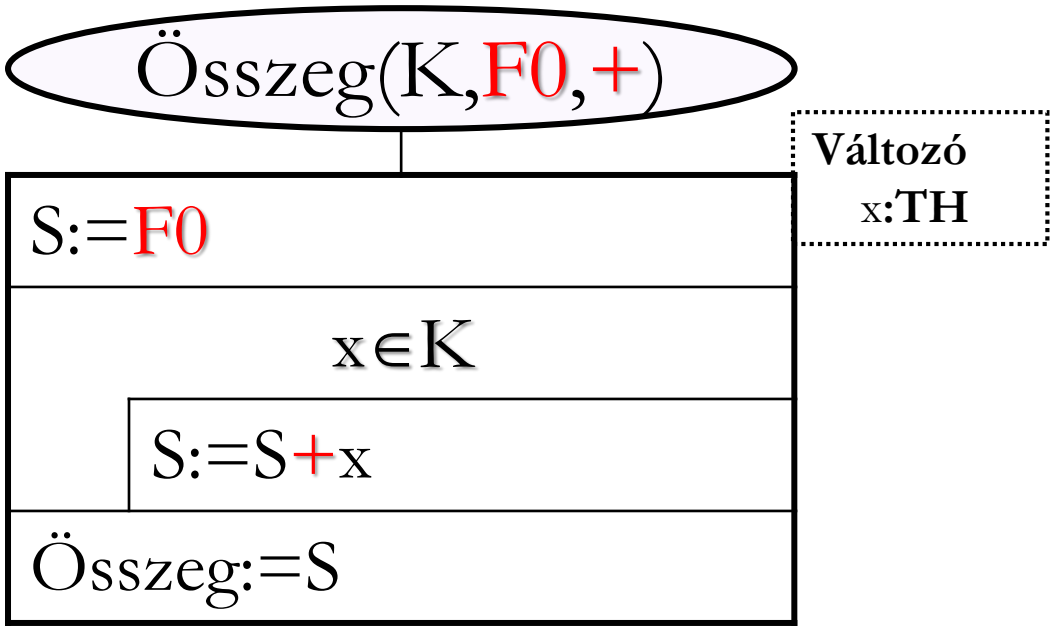
**Algoritmus** (általánosan, iterátorral):




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

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

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

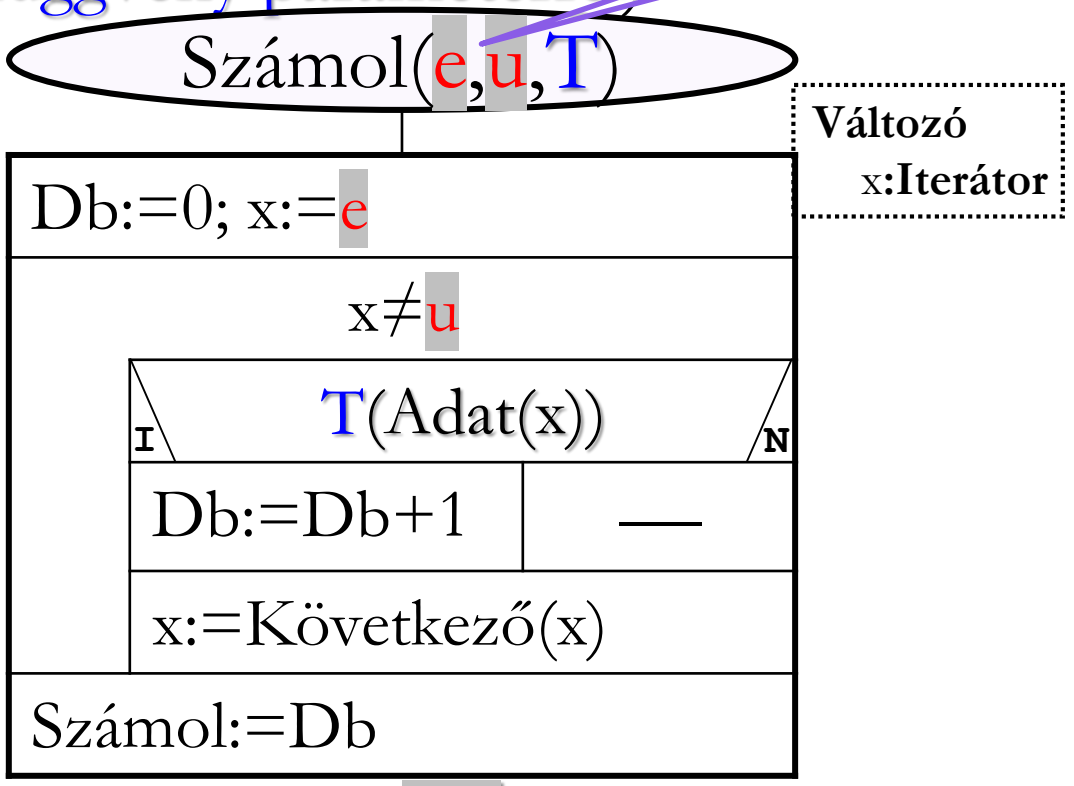
Előfeltétel: e,u X-beli elem címe

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

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

# Megszámolás – általános sorozat-intervallumon

**Algoritmus** (iterátorral, függvény paraméterrel):

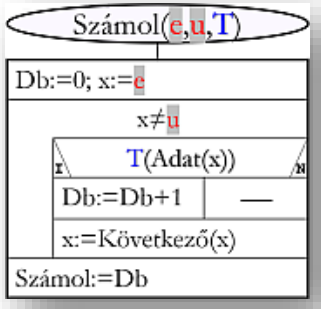



Paraméterek: az első és az utolsó utáni elem címe, T a tulajdonság függvény. Az Adat(x): x címen levő adat.

# Megszámolás – általános sorozat-intervallumon

**C++ kód** (iterátorral, függvény paraméterrel):

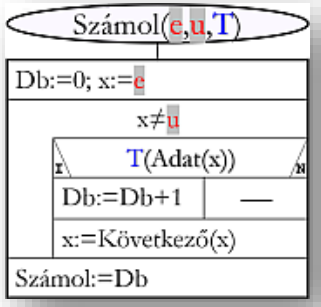
```
template<typename InputIt, typename E>
int Szamol(InputIt e,InputIt u,bool T(E)){
   int db=0;
   while(e!=u){
      if(T(*e)) db++;
      e++;
   };
   return db;
}
array<int,9> AA; //9 elemű egész tömb
vector<int> V;   //akárhány elemű vektor
…
x=Szamol(AA.begin(),AA.end(),paros);
y=Szamol(V.begin(),V.end(),paratlan);
```




# Megszámolás – általános sorozat-intervallumon

**C++ kód** (iterátorral, függvény paraméterrel):

```
template<typename 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: $K \in \mathbb{H}^*$ vagy $K \in 2^{\mathbb{H}}$

$T: \mathbb{H} \to \mathbb{L}$

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

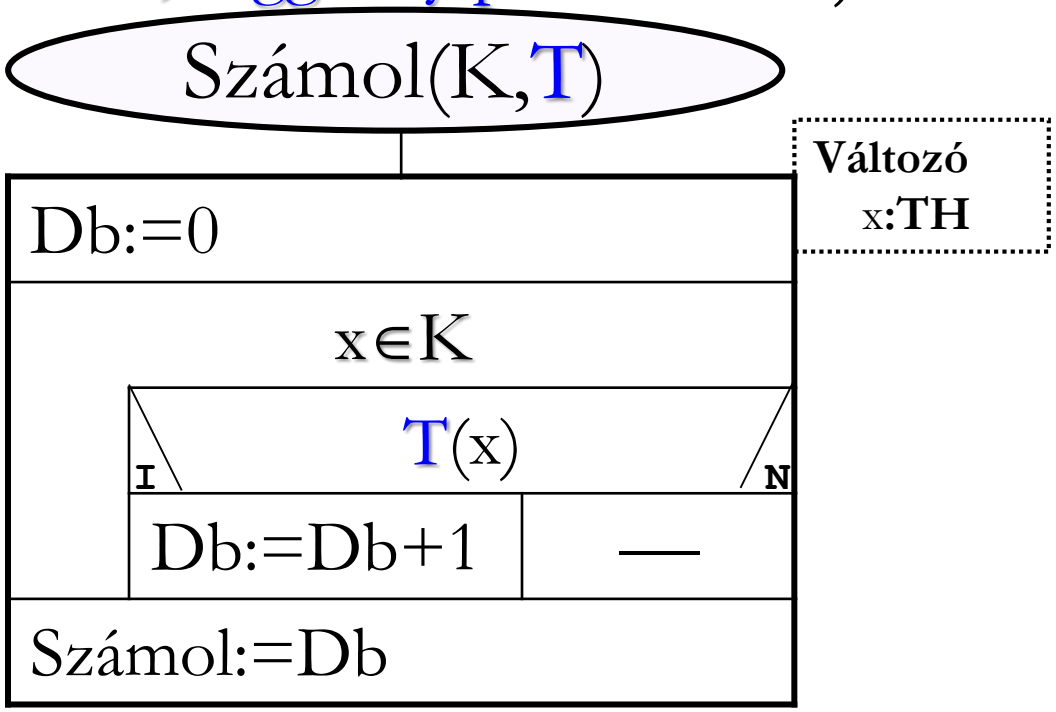
Előfeltétel: –

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

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 ∈ művelettel járhatjuk be; T tulajdonság függvény.

# Megszámolás – általános sorozaton vagy halmazon

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

```
template<typename KontenerT>
int Megszamol(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=Megszamol(AA,paros); y=Megszamol(V,paratlan);
```

# Maximum-kiválasztás – általános sorozat-intervallumon


**Specifikáció** (iterátorral):

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

Kimenet: MaxÉrt$\in \mathbb{H}$, Max$\in$ Cím($\mathbb{H}$)

Előfeltétel: $|X| > 0$ és

e,u X-beli elem címe és $e < u$

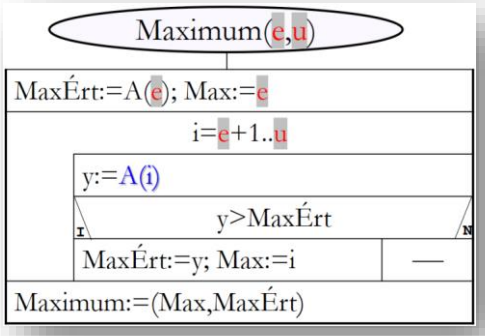
Utófeltétel: $e \leq Max < u$ és $MaxÉrt = X_{Max}$

$\forall it\ (e \leq it < u)$: $MaxÉrt \geq X_{it}$

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

# Maximum-kiválasztás – általános sorozat-intervallumon

## Algoritmus:




e és u címek

Maximum(e,u)

Változ
y:T

Max:=e; MaxÉrt:=Adat(e)

x:=e; Következő(x)

x≠u

y:=Adat(x)

I — y>MaxÉrt — N

Max:=x; MaxÉrt:=y | —

X.=Következő(x)

Maximum:=(Max,MaxÉrt)

Adat(x): x címen levő adat

Következő(x): következő x címe

# Maximum-kiválasztás – általános sorozaton vagy halmazon


**Specifikáció:**

Bemenet: $X \in \mathbb{H}^*$ vagy $X \in 2^{\mathbb{H}}$

Kimenet: MaxÉrt$\in \mathbb{H}$

Előfeltétel: $|X| > 0$

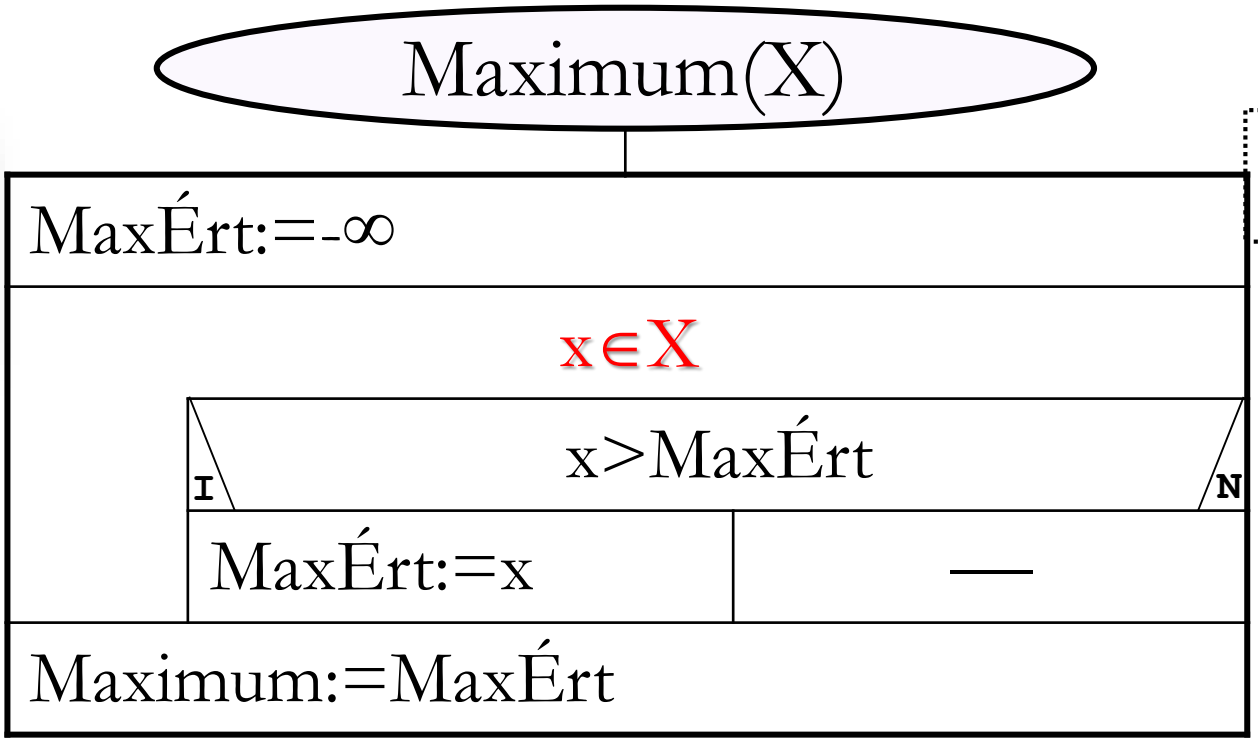
Utófeltétel: MaxÉrt$\in X$ és $\forall y\ (y \in X)$: MaxÉrt$\geq y$

Itt csak a maximális értéknek van értelme, a helyének nincs.

# Maximum-kiválasztás – általános sorozaton vagy halmazon

## Algoritmus (iterációs ciklussal):

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




**Változó**
x:TH

# Kiválasztás – általános sorozat-intervallumon


## Specifikáció:

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

Kimenet: Ért $\in \mathbb{H}$

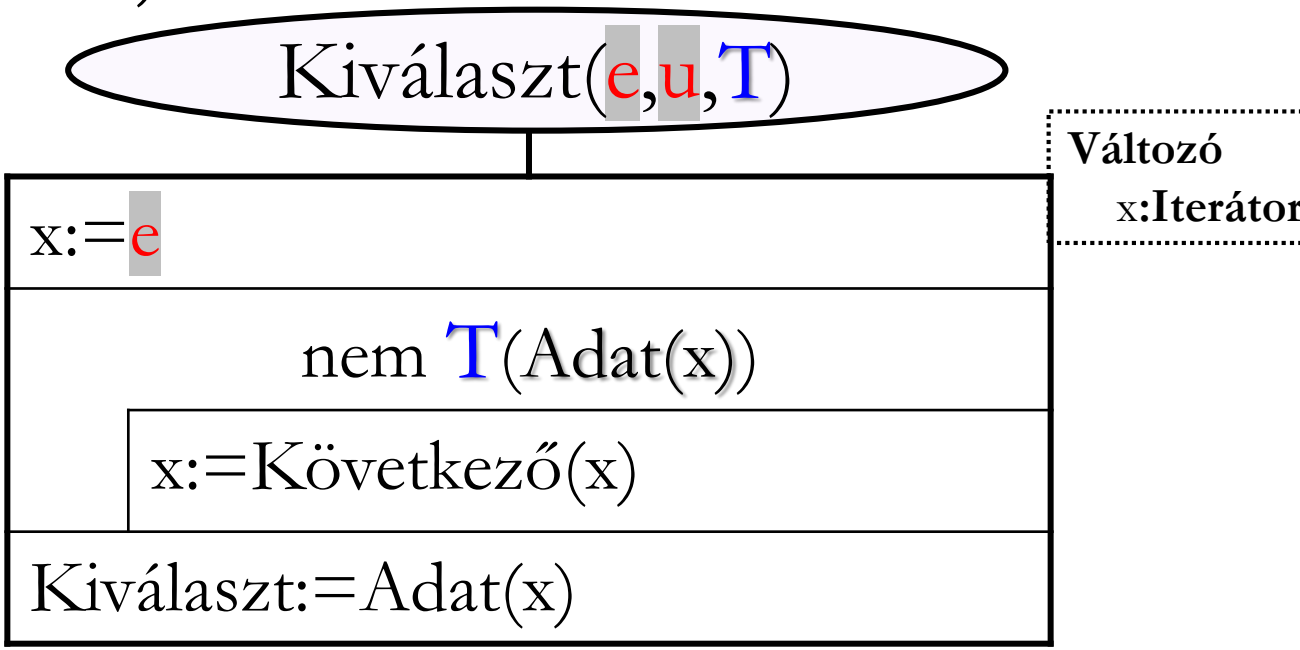
Előfeltétel: $|X| > 0$ és
e,u X-hez tartozó cím és $e < u$ és
$\exists x(x \in X_{e..\text{Előző}(u)})$: T(x)

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

Itt csak az értéket határozzuk meg, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.

# Kiválasztás – általános sorozat-intervallum

**Algoritmus** (iterátorral):




Az Adat(x) az x memóriacímen levő értéket jelöli, az e és az u a sorozat első és utolsó vizsgálandó eleme memóriacíme.

# Kiválasztás – általános sorozat vagy halmaz


**Specifikáció:**

Bemenet: $X \in \mathbb{H}^*$ vagy $X \in 2^{\mathbb{H}}$

$T: \mathbb{H} \to \mathbb{L}$

Kimenet: $Ért \in \mathbb{H}$

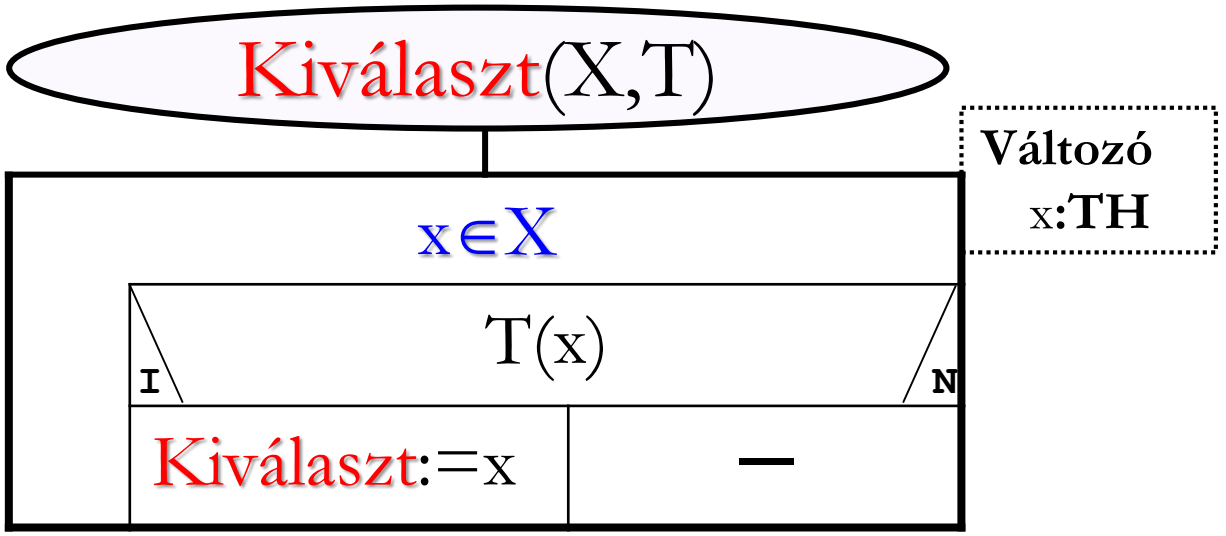
Előfeltétel: $|X| > 0$ és $\exists x (x \in X)$: $T(x)$

Utófeltétel: $Ért \in X$ és $T(Ért)$

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

# Kiválasztás – általános sorozat vagy halmaz

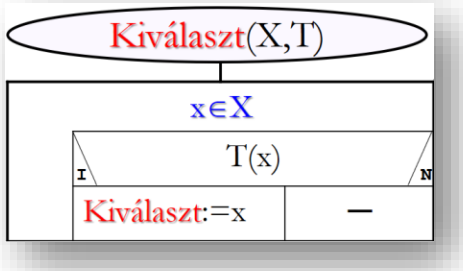
**Algoritmus** (iterációs ciklussal)$_1$**:**




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

# Kiválasztás – általános sorozaton vagy halmazon

## C++ kód (iterációs ciklussal)$_1$:

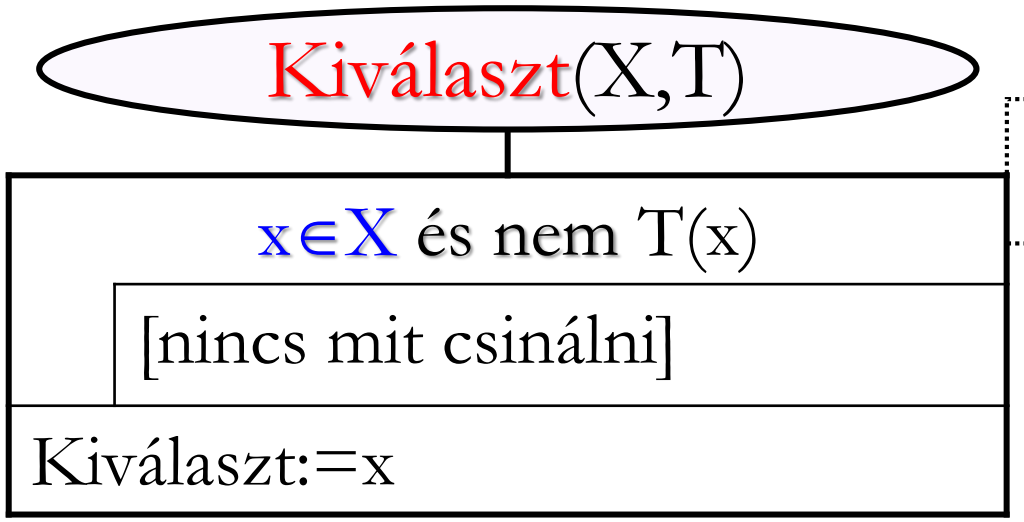



```
template<typename KontenerT>
typename KontenerT::value_type Kivalaszt(
        const KontenerT& K,
        bool T(typename KontenerT::value_type)){
   for (auto x:K)
      if (T(x)) return x;
}
```

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”.

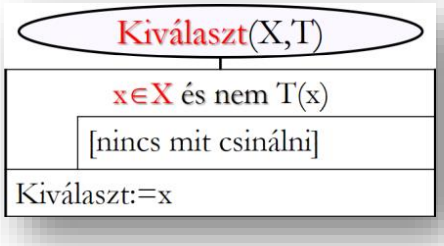
# Kiválasztás – általános sorozat vagy halmaz

## Algoritmus (iterációs ciklussal)$_2$:




E megoldásban a szabálytalanságot azzal a szemantikus elvárással kerüljük ki, hogy az x∈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 is.

# Kiválasztás – általános sorozaton vagy halmazon




## C++ kód (iterációs ciklussal)$_2$:

```
template<typename 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

## Specifikáció:

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

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

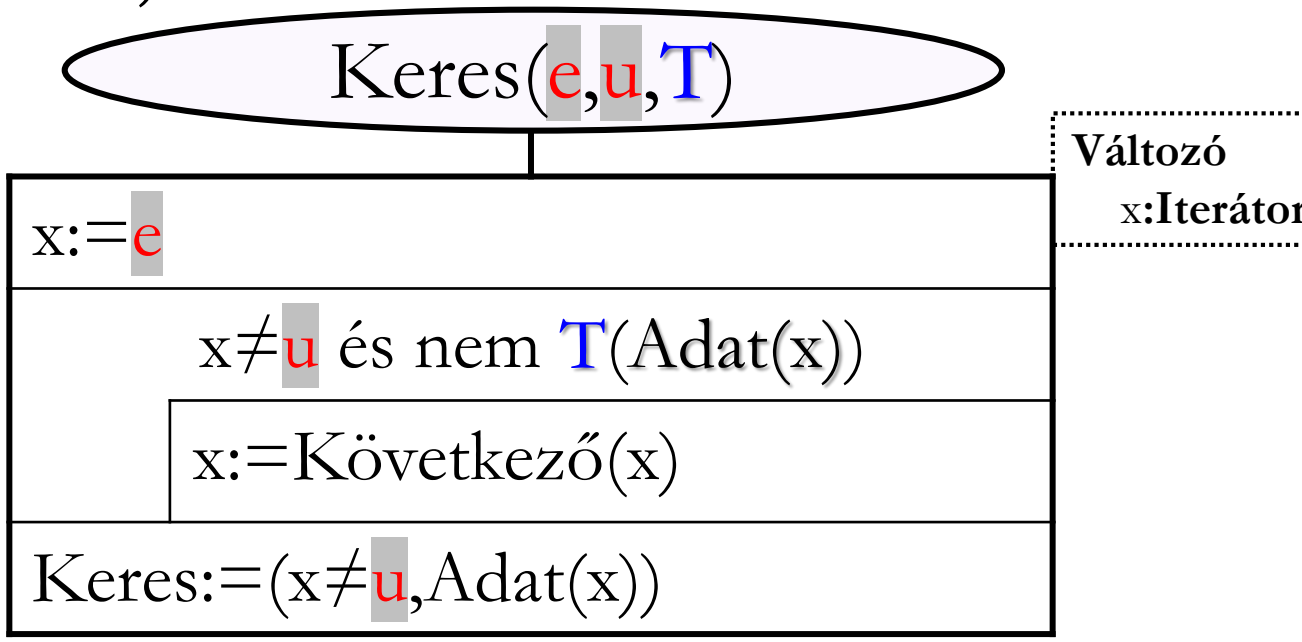
Előfeltétel: e,u X-beli elem címe

Utófeltétel: $\text{Van} = \exists x\ (x \in X_{e..\text{Előző}(u)})$: $T(x)$ és
$\text{Van} \rightarrow \text{Ért} \in X$ és $T(\text{Ért})$

Itt is csak értéket keressük, és a vizsgált tulajdonság csak egyes elemekre vonatkozhat.

# Keresés – általános sorozat-intervallumon

**Algoritmus** (iterátorral):




Az Adat(x) az x memóriacímen levő értéket jelöli, az e és az u a sorozat első és utolsó vizsgálandó eleme memóriacíme.

# Keresés – általános sorozaton vagy halmazon


**Specifikáció:**

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

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

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

Előfeltétel: —

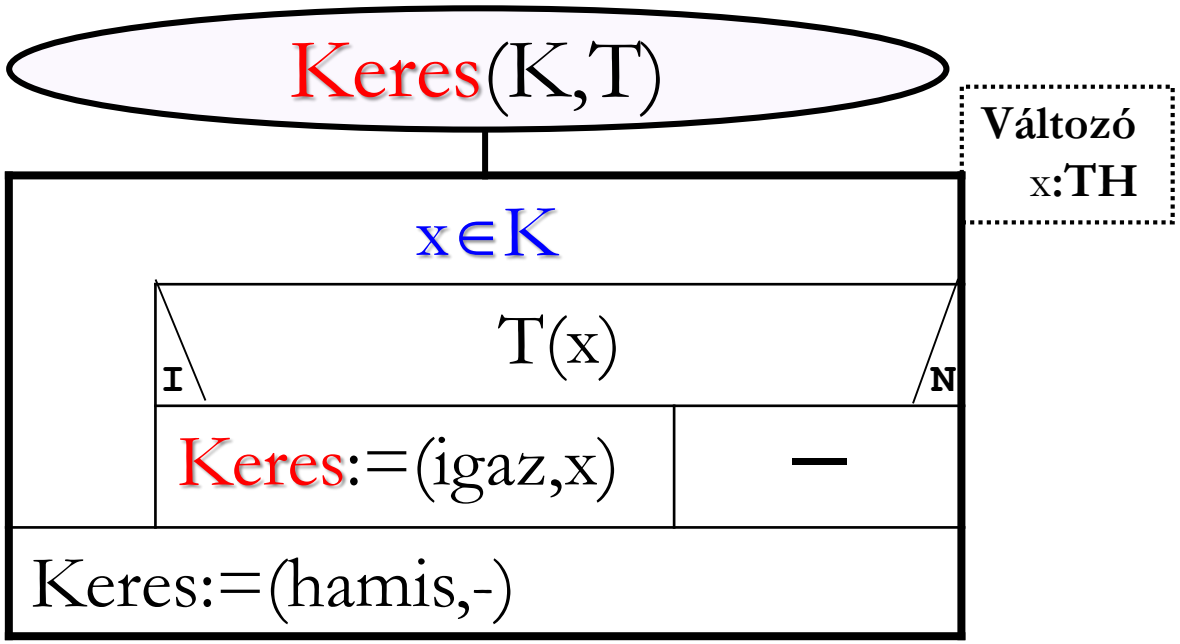
Utófeltétel: $Van = \exists x\ (x \in X)$: $T(x)$ és

$Van \rightarrow Ért \in X$ és $T(Ért)$

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

# Keresés – általános sorozaton vagy halmazon

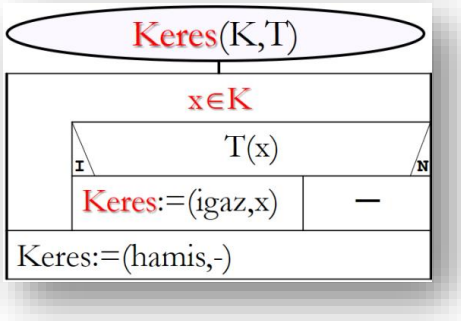
## Algoritmus (iterációs ciklussal)$_1$:




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

# Keresés – általános sorozaton vagy halmazon

## C++ kód (iterációs ciklussal)$_1$:

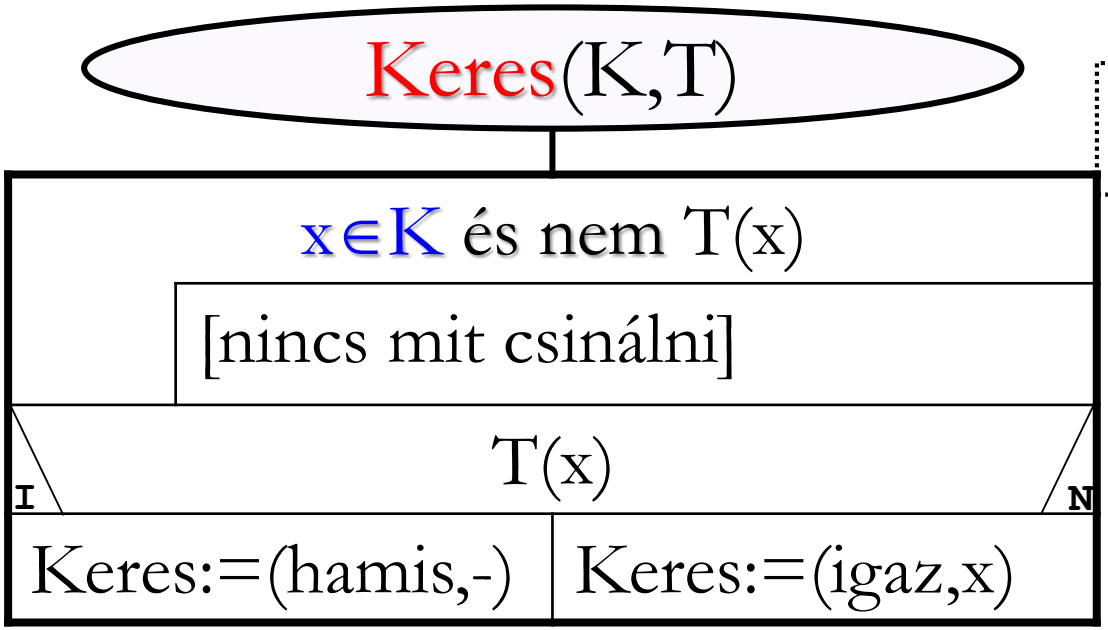



```
template<typename KontenerT>
void Keres(const KontenerT& K,
           bool T(typename KontenerT::value_type),
           bool& van,
           typename KontenerT::value_type& mi){
     van=false;
     for (auto x:K) {
      if (T(x)) {
         van=true; mi=x; break;
      };
    }
}
```

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

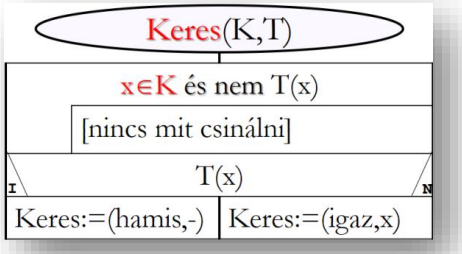
## Algoritmus (iterációs ciklussal)$_2$:




E megoldásban a szabálytalanságot azzal a szemantikus elvárással kerüljük ki, hogy az x∈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<typename 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.

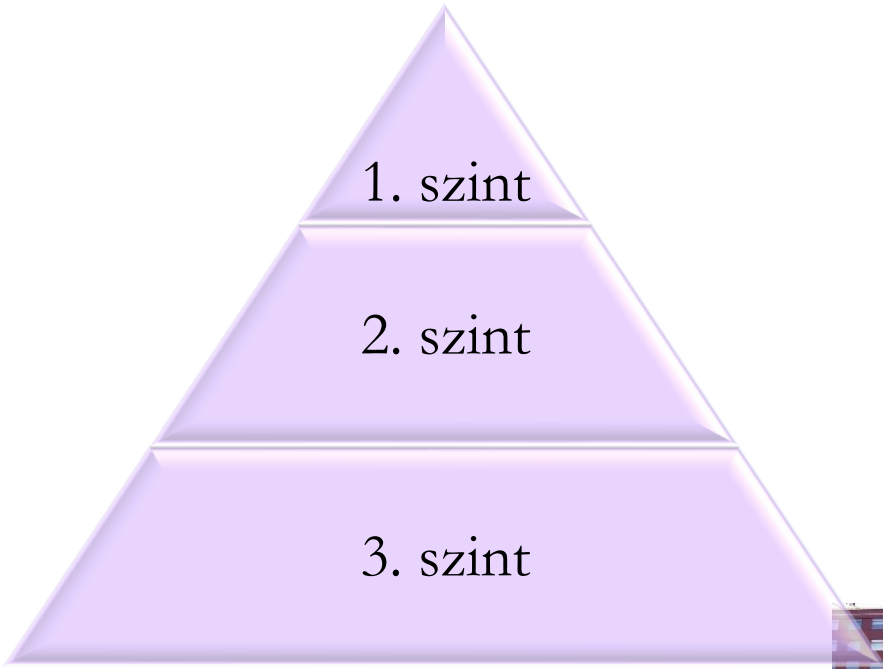
# Programkészítési elvek

- **Stratégiai elv**: a problémamegoldás logikája – a lépésenkénti finomítás.
- **Taktikai elvek**: az algoritmuskészítés gondolati elvei a felülről lefelé kifejtéshez.
- **Technológiai elvek**: algoritmus és kód módszertani kívánalmai.
- **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 → 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
- Súgó, 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

## Tartalma:

E nélkül be sem adható!

- 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

E nélkül be sem adható!

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

## 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
- szerző(k)

Értelmesen strukturálva.

**Szerző**

Név: **Szabó Emerencia**
ETR-azonosító: **SZEKAAT.ELTE**
Neptun-azonosító: **ESZ98A**
Drótposta-cím: **sze@elte.hu**
Kurzuskód: **IP-08PAEG/77**
Gyakorlatvezető neve: **Kiss-József Alfréd**
Feladatsorszám: **18**

E nélkül be sem adható!

# Hatékonyságvizsgálat táblázatkezelővel

Ötlet:

1. A táblázatkezelők importálnak sokféle formátumú fájlt, pl. CSV-formátumút.
2. A Comma Separated Value (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.

# Hatékonyságvizsgálat 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:

$(N,M) \rightarrow hDb_{unió}$, $(N,M) \rightarrow hDb_{összefuttatás}$

…

# 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.)

# Hatékonyságvizsgálat táblázatkezelővel

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

Numerikusan

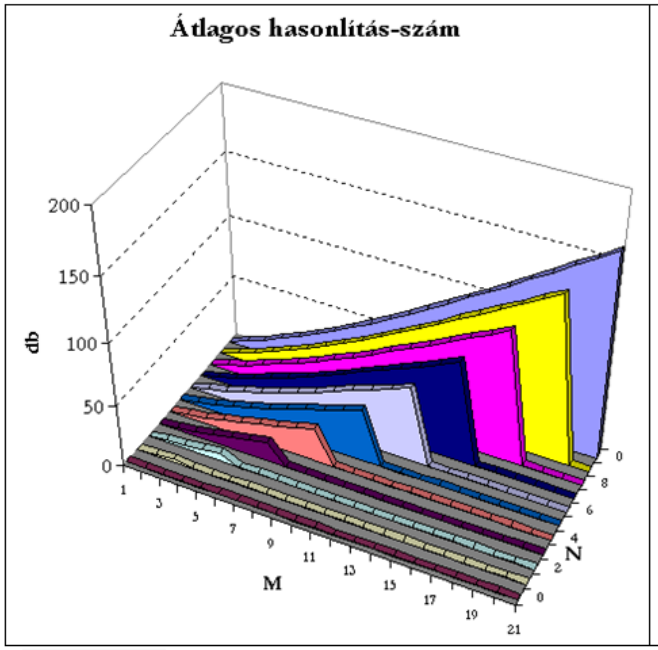
| N \ M | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 1 | 2 | | | | | | | | | | | | | | | | | | |
| 2 | 0 | 2 | 3 | 5 | 7 | | | | | | | | | | | | | | | | |
| 3 | 0 | 2 | 4 | 7 | 10 | 13 | 16 | | | | | | | | | | | | | | |
| 4 | 0 | 3 | 5 | 9 | 12 | 16 | 19 | 23 | 27 | | | | | | | | | | | | |
| 5 | 0 | 3 | 6 | 10 | 14 | 18 | 23 | 28 | 32 | 37 | 42 | | | | | | | | | | |
| 6 | 0 | 4 | 7 | 12 | 16 | 21 | 26 | 31 | 37 | 43 | 48 | 54 | 60 | | | | | | | | |
| 7 | 0 | 4 | 8 | 13 | 18 | 23 | 29 | 35 | 41 | 48 | 54 | 61 | 67 | 74 | 81 | | | | | | |
| 8 | 0 | 5 | 9 | 15 | 20 | 26 | 32 | 39 | 45 | 52 | 60 | 67 | 74 | 82 | 90 | 97 | 105 | | | | |
| 9 | 0 | 5 | 10 | 16 | 22 | 28 | 35 | 42 | 49 | 57 | 65 | 73 | 81 | 90 | 98 | 107 | 115 | 124 | 132 | | |
| 10 | 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.

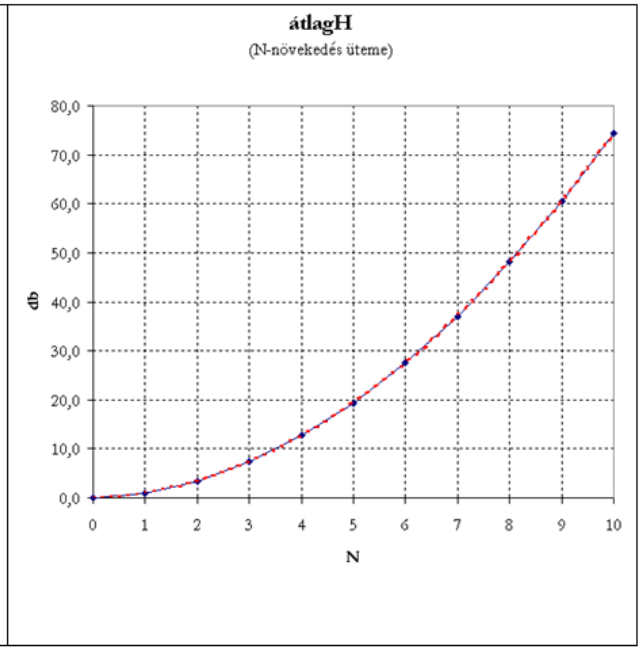
# Hatékonyságvizsgálat táblázatkezelővel

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

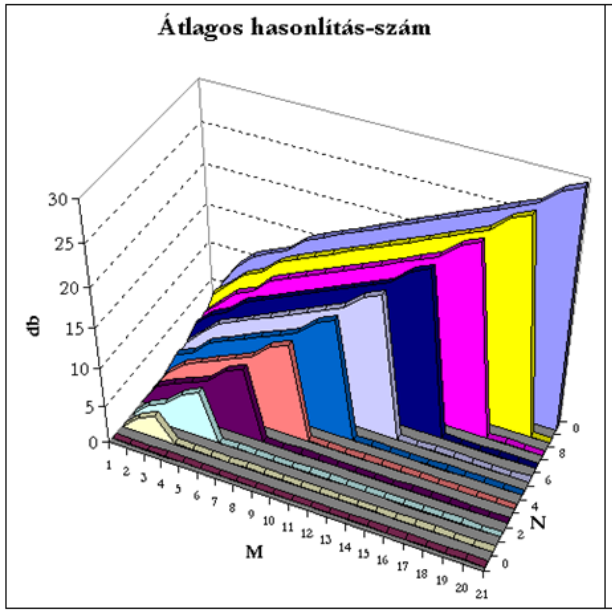
Egy lehetséges eredmény
a táblázatkezelőbe importálás után – **összefuttatás**:
Numerikusan

| N \ M | 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 |

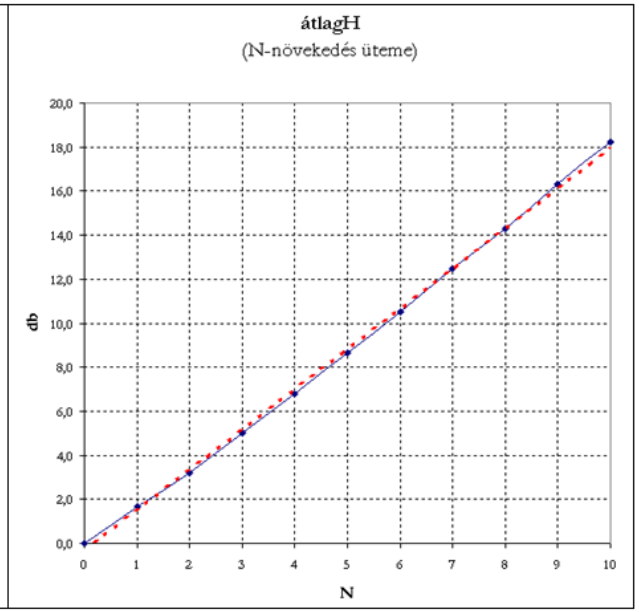
A program outputja táblázatkezelőbe importálás és némi szépítés után.

# 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 ráilleszthető lineáris egyenes gráfja látható piros pontokkal.