V2.1

Algoritmusok és Adatstruktúrák

I. félév

Hernyák Zoltán

A jegyzet e változata még tartalmazhat mind gépelési, mind helyességi hibákat. Az állítások nem mindegyike lett tesztelve teljes körűen. Minden észrevételt, amely valamilyen hibára vonatkozik, örömmel fogadok.

Eger, 2003. február 16.

Algoritmusok és adatstruktúrák I. félév

Hernyák Zoltán

Algoritmus neve:

Muhammad Ibn Músza Al-Hvárizmi csillagász, matematikus, ie. I. századi perzsa tudós nevéből származik, akinek egyik könyvét latinra fordították, és a nevét pontatlanul **Algorithmus**-nak írták.

Algoritmus fogalma

Műveletek tartalmát és sorrendjét meghatározó egyértelmű utasításrendszer, amely a megfelelő kiinduló adatokból a kívánt eredményre vezet.

Az algoritmus tulajdonságai

- Végrehajtható.
- Minden lépés egy elemi utasítás vagy további algoritmus végrehajtását igényli.
- Meghatározott sorrendet követ.
- Véges számú lépésben véget ér.

Az algoritmus megadása két fő összetevő megadásából áll: **adatok**, és a velük kapcsolatos **műveletek** megadásából.

Adatok (mennyiségek)

Az algoritmus működéséhez adatokra van szükség. Egyrészt a fő műveleti adatokból, másrészt a működés során fellépő segédadatokra. Az informatikában az adatokat **változók** és **konstansok** formájában jelentkezik.

Alapfogalmak

Változó: Azok a mennyiségek, amelyek az algoritmus végrehajtása során megváltoznak.
Konstans: Azok a mennyiségek, amelyek az algoritmus végrehajtása során nem változnak meg.
Azonosító: Az a jelsorozat, amellyel hivatkozhatunk a változó tartalmára, és módosíthatjuk azt.
Értékadás: Az a művelet, amelynek során a változók értéket kapnak, vagy megváltozik az értékük. **Kezdőérték**: A változónak és konstans az induló értéke.
Deklaráció: A bejelentés helye, az algoritmusnak az a helye, ahol a változót először elhelyeztük.
Definiált: Az azonosító mögöttes értéke egyértelműen ismert.
Definiálatlan: nem definiált.
Értéktípus: Az adatoknak az a tulajdonsága, amely meghatározza az adat által felvehető értékeket és a rajtuk értelmezett műveleteket.

- Az azonosító képzésére komoly szabályok vannak:
  - csak az angol abc karakterei, számjegyek, és az aláhúzás jel használható
  - az első betű nem lehet számjegy
  - az azonosító hossza tetszőleges lehet, de nem ajánlatos 16 karakternél hosszabbat használni

Értéktípusok

1. Egyszerű értéktípusok *(egy azonosító pontosan egy értéket takar)*
1/1 : Egész típus
Értékek: pozitív és negatív egész számok tárolására alkalmas típus.
Műveletek: + - * / % < > = <= >= <>
az osztás kivezethetne a tartományból, de két egész típusú adaton elvégzett osztási művelet eredményét egészre *kerekítvén* kell értelmezni (csonkolás művelete) pl. 4/3=1
- a % művelet az osztási maradék képzése. Pl. 4%3=1, 8%3=2, 8%2=0)

1/2: Valós típus
Értékek: tetszőleges egész vagy tört szám
Műveletek: + - * / < > = <= >= <>

1/3: Karakter típus
Értékek: egy karakter (pl. egy betű, számjegy, egyéb jel)
Műveletek: < > = <= >= <>
- karakter konstansok megadása egyszeres aposztrófok között történhet, pl. 'a'
- az összehasonlítás az ASCII kódtábla alapján történik, pl. az 'a' kisebb mint 'b'
- az összehasonlítás kisbetű-nagybetű értzékeny (Case-Sensitive), vagyis 'alma'<>'ALMA'

1/4: Logikai típus
Értékek: IGAZ, HAMIS
Műveletek: ÉS VAGY NEM XOR =
- a logikai műveletek részletes leírását lásd később

<u>2. Összetett értéktípusok</u> *(egy azonosító több értéket is takar*

2/1: Tömb típus *(egy azonosító név mögött több érték is tárolható, melyek viszont kötelezően azonos típusúak kell hogy legyenek. Ez a típus a tömb alaptípusa. A mögöttes több érték közül egyszerre általában csak egyhez férhetünk hozzá (egyesével kezelhető). A konkrét érték azonosításához meg kell adni az érték azonosító sorszámát (indexét))*

2/1/1: Egydimenziós tömb (<u>vektor</u>) *pl.: **X: tömb [1..10] egész-ből**. A konkrét értékhez hozzáférés egyetlen index megadásával történik: X[1] a tíz lehetséges érték közül az elsőt jelenti. A tíz érték ábrázolása egy sorban történhet.*

2/1/2: Két dimenziós tömb (<u>mátrix</u>) *pl.: **X:tömb [1..10,1..20] egész-ből.** A konkrét értékhez hozzáférés a két index megadásával történik: X[4,15] a 200 (10*20) lehetéges érték közül a negyedik sor 15. elemét jelenti. A 200 érték ábrázolása 10 sor, soronként 20 oszlop formájában történhet (gyak. Egy rácsos táblázat).*

2/1/3: Több dimenziós tömb (<u>tömb</u>) *pl.: **X:tömb [1..10,1..20,1..10] egész-ből.** Ez konkrétan egy három dimenziós tömb, de ugyanezen az elven lehet felépíteni négy, öt, stb… dimenziós tömböket is. A konkrét elemhez hozzáférés során a dimenziók számának megfelelő számú index megadása szükséges. Pl. X[1,14,6] egy konkrét elem a lehetséges 2000-ből (10*20*10). Egy háromdimenziós tömb elemei egy térbeli egységkockákból felépített kockaként ábrázolható. Az ennél több dimenziós tömb nehezen elképzelhető, és ábrázolható.*

➢ elméletileg elképzelhető, hogy a tömb indexei nem 1-nél kezdődnek.
➢ Pl. ***X: tömb [-10..+10] egész-ből***. Ez pl. 21 elem tárolását jelenti, ahol a konkrét elem elérése során a sorszám -10 és +10 között lehet (0 is!).
➢ a lehetségestől eltérő index megadása hibára vezet. Az előbbi példa esetén az X[15] hibás, mert nincs ilyen eleme a vektornak. Több dimenziós tömbök esetén a megfelelő index a megfelelő értékhatárok között kell legyen. Pl.: ***X: tömb [-5..+5,1..10,-10..20] egész-ből*** esetén az első index csak -5 és +5 közötti lehet, a második index csak 1 és 10 közötti lehet, a harmadik pedig -10 és +20 közötti. Ez alapján az X[-5, 6, -8] helyes, míg az X[4,12,15] helytelen, mert a második index értéke túl nagy.
➢ egy esetben képzelhető el az, hogy nem csak egy tömbelemmel végzünk műveletet, amikor a tömb minden elemével egyszerre. Ez csak egy esetben lehetséges, két egyforma típusú tömb közötti értékadást úgy értelmezünk, hogy az egyik tömb minden egyes elemét átmásoljuk a másik tömb minden egyes eleme helyére. Pl.: ***X,Y: tömb [1..10] egész-ből*** esetén az X:=Y jelentése: X minden egyes eleme egyenlő lesz az Y megfelelő elemével X[1]:=Y[1], X[2]:=Y[2], stb…
➢ ezen utóbbi művelet csakis teljesen egyforma tömök között lehetséges. Tömb-részre nem működik.
➢ Pl.: ***X:tömb [1..10,1..20] egész-ből, Y:tömb [1..20] egész-ből*** esetén hibás az X[1]:=Y értékadás, pedig X[1] egy 20 egészet tároló vektorként is felfogható lenne (ami éppen az Y).

2/2: Szöveg típus (string típus) *(egynél több karakter tárolására alkalmas)*
➢ - a szöveg konstansok a dupla aposztróf között ábrázolódnak. Pl.: "almafa"
➢ - a szöveg hossza az őt alkotó karakterek számával egyenlő
➢ - megengedett az ún. üres string is (nulla hosszú string), melyet ""-ként jelzünk.
➢ - a szöveg típus felfogható karakterekből álló vektorként is. Ennek megfelelően ha
➢ ***X:szöveg***, akkor az X[1] a szöveget alkotó első karaktert jelzi. Pl. ha X="almafa", akkor az X[1] az 'a' karaktert jelzi. Emiatt soroljuk a szöveg típust az összetett típusok közé.

Logikai alapfogalmak

Logikai érték: Az IGAZ, HAMIS tulajdonság.
Logikai művelet: A logikai változókat összekapcsoló művelet.
Kijelentés: logikai kifejezés, mely logikai változókból, és a logikai műveletekből építhető fel.

Logikai műveletek

NOT (NEM) *A művelet értéke az ellentétes logikai érték lesz*

NEM *IGAZ* => HAMIS
NEM *HAMIS* => IGAZ

AND (ÉS) *A művelet eredménye akkor és csak akkor igaz, ha mindkét operandus igaz*

HAMIS **ÉS** *HAMIS* => HAMIS
HAMIS **ÉS** *IGAZ* => HAMIS
IGAZ **ÉS** *HAMIS* => HAMIS
IGAZ **ÉS** *IGAZ* => IGAZ

OR (VAGY) *A művelet eredménye akkor és csak akkor hamis, ha mindkét operandus hamis*

HAMIS **VAGY** *HAMIS* => HAMIS
HAMIS **VAGY** *IGAZ* => IGAZ
IGAZ **VAGY** *HAMIS* => IGAZ
IGAZ **VAGY** *IGAZ* => IGAZ

XOR (kizáró vagy) *A művelet eredménye akkor és csak akkor hamis, ha mindkét operandus értéke ugyanaz*

HAMIS **XOR** *HAMIS* => HAMIS
HAMIS **XOR** *IGAZ* => IGAZ
IGAZ **XOR** *HAMIS* => IGAZ
IGAZ **XOR** *IGAZ* => HAMIS

Az A és B az alábbiakban logikai típusú változókat jelöl, melyek értékei IGAZ ill. HAMIS lehet. Az alábbiakban tárgyaljuk, hogy mely konkrét értékek esetén milyen eredményre vezet a művelet.

Logikai azonosságok

De Morgan azonosságok

NEM (A **VAGY** B)	=	(**NEM** A) **ÉS** (**NEM** B)
NEM (A **ÉS** B)	=	(**NEM** A) **VAGY** (**NEM** B)

Általános azonosságok

A **ÉS** (**NEM** A)	=	mindig HAMIS
A **VAGY** (**NEM** A)	=	mindig IGAZ
A **ÉS** *IGAZ*	=	A *a művelet eredménye csak A-tól függ (megegyezik A-val)*
A **ÉS** *HAMIS*	=	HAMIS
A **VAGY** *IGAZ*	=	IGAZ
A **VAGY** *HAMIS*	=	A *a művelet eredménye csak A-tól függ (megegyezik A-val)*
NEM (**NEM** A)	=	A *kétszeres tagadás eredménye maga az érték*

Az algoritmus megadása során különböző műveleteket végezhetünk az adatokon, az adatok segítségével. A műveleteket (tevékenységeket) az alábbi módon csoportosíthatjuk:

A tevékenységek csoportosítása

- **Elemi műveletek**
  Azok a tevékenységek, amelyek nem igényelnek magyarázatot, azonnal végrehajthatók. *Ezen műveleteket a végrehajtó (a számítógép) ismeri, és azokat végre tudja hajtani.*

- **Összetett műveletek**
  Azok a tevékenységek, amelyek elemi tevékenységekből épülnek föl, tartalmukat mindig meg kell magyarázni, maguk is egyszerűbb algoritmusokból épülnek föl. *Ezen tevékenységeket a végrehajtó (a számítógép) nem ismeri, azok további magyarázatra várnak, ki kell bontani őket.*

Tevékenységszerkezetek *(több művelet végrehajtása során a műveletek során a végrehajtás sorrendjét az alábbi módokon lehet szervezni)*

1. Szekvencia *a szekvenciát alkotó utasítások a megadás (leírás) sorrendjében végrehajtandók*

utasítás 1
utasítás 2
…

2. Elágazás *két (vagy több) műveletcsoport közül legfeljebb csak az egyiket kell végrehajtani. A döntés mindig valamilyen logikai feltételtől függenek, és annak ismeretében egyértelmű a döntés.*

2/1: Egyszerű elágazás *(egy utasításblokkból áll)*

az utasításblokk a feltételtől függően vagy végrehajtásra kerül, vagy nem.

```
HA sötét_van AKKOR
    kapcsold fel a villanyt
HVÉGE
```

2/2: Összetett elágazás

2/2/a: két utasításblokkból álló összetett elágazás

A két utasításblokk közül a feltételtől függően pontosan az egyik utasításblokk hajtódik végre.

```
HA meleg_van AKKOR
    nyisd_ki_az_ablakot
KÜLÖNBEN
    kapcsol_le_a_kazánt
HVÉGE
```

2/2/b: több utasításblokkból álló összetett elágazás

A több utasításblokk közül legfeljebb az egyik kerül végrehajtásra
- elképzelhető, hogy egyik feltétel sem teljesül. Ekkor
- ha van KÜLÖNBEN ág, akkor az hajtódik végre
- ha nincs KÜLÖNBEN ág, akkor egyik blokk sem hajtódik végre
- ha több feltétel is teljesül, akkor sorrendben csak az első hajtódik végre

```
ELÁGAZÁS KEZD
    HA kapható_túró AKKOR
            süss_túrós_sütit
    HA kapható_mák AKKOR
            süss_mákos_sütit
    HA kapható_dió AKKOR
            süss_diós_sütit
    KÜLÖNBEN
            süss_almás_sütit
ELÁGAZÁS VÉGE
```

3. Ciklus *egy feltételtől függően egy adott utasításblokk többszöri ismételt végrehajtását jelenti. Az utasításblokkot **ciklusmag**nak nevezzük. A feltételt **ciklus vezérlő feltétel**nek.*

Pozitív vezérlés elve: a ciklusmagot akkor kell végrehajtani, ha a vezérlő feltétel **igaz.**

Negatív vezérlés elve: a ciklusmagot akkor kell végrehajtani, ha a vezérlő feltétel **hamis.**

Előltesztelő ciklusok: *a feltétel előbb értékelődik ki, majd megfelelő esetben végrehajtásra kerül a ciklusmag „… előbb tesztel, aztán ciklusmag…”.*

Hátultesztelő ciklusok: a ciklusmag végrehajtódik, majd kiértékelődik a ciklus vezérlő feltétel, és megfelelő esetben újra végrehajtásra kerül a ciklusmag.

3/1: logikai ciklusok

3/1/a: logikai előltesztelő pozitív vezérlésű ciklusok *működése:*

1. HA a ciklus_vezérlő_feltétel igaz AKKOR
2. ciklusmag_utasításaiank_végrehajtása
3. ugorj újra az 1. Sorra
4. KÜLÖNBEN ciklus vége

3/1/b: logikai előltesztelő negatív vezérlésű ciklusok *működése:*

1. HA a ciklus_vezérlő_feltétel hamis AKKOR
2. ciklusmag_utasításaiank_végrehajtása
3. ugorj újra az 1. sorra
4. KÜLÖNBEN ciklus vége

3/1/c: logikai hátultesztelő pozitív vezérlésű ciklusok *működése:*

1. ciklusmag_utasításainak_végrehajtása
2. HA a ciklus_vezérlő_feltétel igaz AKKOR ugorj újra az 1. sorra
3. KÜLÖNBEN ciklus vége

3/1/d: logikai hátultesztelő negatív vezérlésű ciklusok *működése:*

1. ciklusmag_utasításaiank_végrehajtása
2. HA a ciklus_vezérlő_feltétel hamis AKKOR ugorj újra az 1. sorra
3. KÜLÖNBEN ciklus vége

- az előltesztelő ciklus esetén (a működésből fakadóan) elképzelhető, hogy a ciklusmag egyszer sem hajtódik végre, hátultesztelők esetén a ciklusmag legalább egyszer biztosan végrehajtódik
- pozitív vezérlésű ciklusoknál a vezérlő feltétel <u>igaz</u> értéke esetén ismétlődik a ciklusmag végrehajtása (ekkor a vezérlő feltételt a ciklusban maradás feltételének is hívhatjuk)
- negatív vezérlésű ciklusoknál a vezérlő feltétel <u>hamis</u> értéke esetén ismétlődik a ciklusmag végrehajtása (ekkor a vezérlő feltételt a ciklusból kilépés feltételének is hívhatjuk)
- gondoskodni kell róla, hogy a ciklusmag kihatással legyen a vezérlő feltétel értékére, különben végtelen ciklus léphet fel (a vezérlő feltétel értéke sohasem lesz képes megváltozni)
- a helytelenül megfogalmazott logikai ciklus könnyen végtelen ciklussá válhat

3/2 fix ismétlésszámú
3/2/a: fix ismétlésszámú ciklus

ISMÉTELD n-szer
    ciklusmag utasításai
IVÉGE
- a ciklusmag utasításai a fejrészben megadott számszor hajtódnak végre

3/2/b: számlálós ciklus

ISMÉTELD cv:=kezdőérték -től végérték -ig
    ciklusmag utasításai
IVÉGE
végrehajtása:
1. lépés: ciklus változó (cv, amely egy egész típusú változó) felveszi a kezdőértéket
2. lépés: HA a ciklusváltozó értéke kisebb, vagy egyenlő, mint a végérték AKKOR
3. lépés: ciklusmag utasításainak végrehajtása
4. lépés: ciklusváltozó értékének automatikus növelése 1-el
5. lépés. ugorj újra a 2. Lépésre
6. lépés: KÜLÖNBEN ciklus vége

a ciklusváltozó értéke a ciklus belsejében hozzáférhető (olvasható), de meg nem változtatható (nem írható)
a ciklusváltozó értéke a 4. Lépésben automatikusan változik. Általában 1-el nő, de ha ettől eltérő viselkedést várunk a ciklustól (pl. 1-el csökkenjen), azt a ciklus megadásakor jelezni kell.
- a számlálós ciklusok esetén (elvileg) a végtelen ciklusba esés kizárt

3/2/c: halmaz-alapú ciklus

A ciklus egy adott (nem végtelen) intervallum minden egyes elemét felveszi. Hasonló a számlálós ciklushoz, de itt nem csak adott egész intervallumra működik, hanem kissé kiterjesztett értelemben akár egy halmazon is futtatható, melynek során a ciklusváltozó felveszi a halmaz minden egyes elemét, és közben mindig végrehajtja a ciklusmagot. A halmaz elemein általában tudunk értelmezni egy rendező leképezést, mely alapján meg tudjuk határozni, hogy melyik elem után melyik elem értékét veszi fel a ciklusváltozó.

3/3: végtelen ciklusok

- a ciklusmag utasításai a végtelenségig ismétlődnek
- az algoritmusok egyik alaptulajdonsága a végesség. De speciális algoritmusok esetén éppen hogy a végtelenség a szükséges tulajdonság (pl. operációs rendszerek futása elvileg a végtelenségig zajlik).

Struktúrált programozás: olyan algoritmusok készítése, amely csak a fenti három tevékenységszerkezet tartalmaz (szekvencia, szelekció, iteráció).

Matematikusok bizonyították, hogy minden algoritmus elkészíthető csak ezen három szerkezet segítségével.

Algoritmusleíró eszközök

1. Folyamatábra

Szabályok:

- Minden folyamatábra pontosan egy START szimbólumot tartalmaz.
- Minden folyamatábra legalább egy STOP szimbólumot tartalmaz.
- Minden folyamatábra-szimbólumba pontosan egy irányított él vezet be, kivéve a START szimbólumot, melybe nem vezet be él.
- Minden folyamatábra-szimbólumból pontosan annyi irányított él vezet ki, amennyit a mellékelt ábrán jelöltünk (vagyis mindegyikből pontosan egy vezet ki, kivéve az elágazás szimbólumot, melyből kettő, és kivéve a STOP szimbólumot, melyből egy sem).
- Minden folyamatábra-szimbólumba vezet irányított élsorozat a START szimbólumból.
- Minden folyamatábra szimbólumból vezet irányított élsorozat legalább egy STOP szimbólumba.

A folyamatábrával nem csak struktúrált algoritmusok írhatók le. Ahhoz, hogy ellenkezője ne történhessen, az alábbiakra kell ügyelni:
- azon feltételes elágazások, melyek nem ciklust jelölnek, a „Nem” és az „Igen” ágak egy pontban kell hogy találkozzanak
- Nem vezethet be irányított él a ciklusok magjába, sem feltételes elágazások ágainak belsejébe.

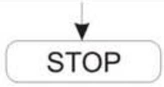
2. Struktogramm

Dobozolós módszer. Csak struktúrált algoritmusok írhatók le segítségével. Hátránya, hogy előre „jól” kell dönteni a kiinduló téglalap méretéről. Az algoritmus bővítése során ugyanis könnyen előfordulhat, hogy „kicsi” lesz valamelyik téglalap.

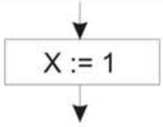
3. Leíró nyelv

Mondatszerű leíró eszköz, mely nem teljesen szabványosított, de könnyen elsajátítható. Csak struktúrált algoritmusok írhatók le vele.

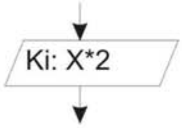
Start: folyamatábra kezdőszimbóluma, ebből pontosan egy van minden folyamatábrában.



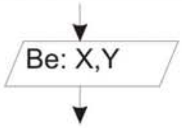
Stop: folyamatábra záró szimbóluma. Ebből elvileg csak egy van de gyakorlatilag többet is szoktunk használni a könnyebbség végett.



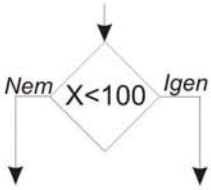
Értékadás szimbóluma.



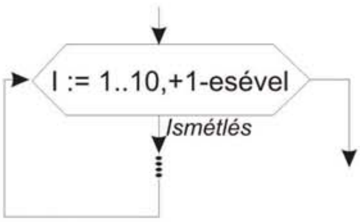
Output. A megadott kifejezés eredménye kiíródik a képernyőre.



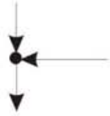
Input. A megadott változók értékeit a gép sorban bekéri a billentyűzetről.



Feltételes elágazás. Ha a megadott feltétel igaz, az “Igen” ágon kell továbbhaladni. Ellentétes esetben a “Nem” ágon. Mindíg két ág van. Az “Igen” és “Nem” ágak fordítva is elhelyezkednek, ezért felirattal mindíg jelezni kell melyik melyik. A rombusz belsejébe egy egyértelműen eldönthető Logikai kifejezést kell írni.

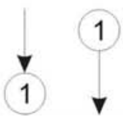


Számlálós ciklus. A cilusváltozó (I) felveszi a kezdőértéket, amennyiben ez kisebb, vagy egyenlő mint a végérték, az “Ismétlés” ágon kell haladni. Mielőtt visszatérnénk a szimbólumba, a ciklusváltozó értéke automatikusan nő egyel. Amennyiben még mindíg kisebb, vagy egyenlő a végértékkel, újra az “Ismétlés” ágon kell haladni. Az “Ismétlés” ágon kell elhelyezni a ciklusmag utasításait. Amennyiben a ciklusváltozó értéke meghaladja a végértéket, a ciklus futása befejeződik, és a másik ágon folytatódik a folyamatábra végrehajtása.



Folyamatábra vonalak csatlakozása. Az ágak közös irányban haladnak tovább.

Vonalak csatlakozás nélküli keresztezése. A két dimenziós ábrázolás miatt a vonalak időnként keresztezik egymást.



Algoritmus folytatása. Áttekinthetőségi okok miatt, vagy mert az algoritmus eléri a lap szélét, folytatást jelölhetünk. A megszakítás helyére egy baloldali szimbólum kerül, a folytatást egy jobb oldali szimbólummal kezdjük. A szám helyébe természetesen tetszőleges szám, ill. egyéb egyedi azonosító kerülhet.

X := 1

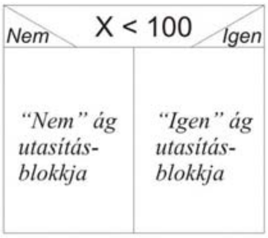
Értékadás szimbóluma.

Be: X,Y

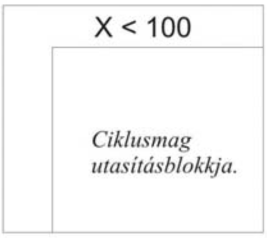
Input. A megadott változók értékeit a gép sorban bekéri a billentyűzetről.

Ki: X*2

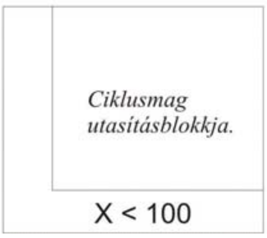
Output. A megadott kifejezés eredménye kiíródik a képernyőre.



Feltételes elágazás. A megadott feltétel "Igen" értéke esetén a jobb oldali rész-blokkba zárt utasításblokk hajtódik végre. "Nem" esetén a bal oldali blokk utasításai hajtódnak végre. Az "Igen" és "Nem" ág helyzetét fel lehet cserélni, ezért mindíg jelölni kell melyik melyik.



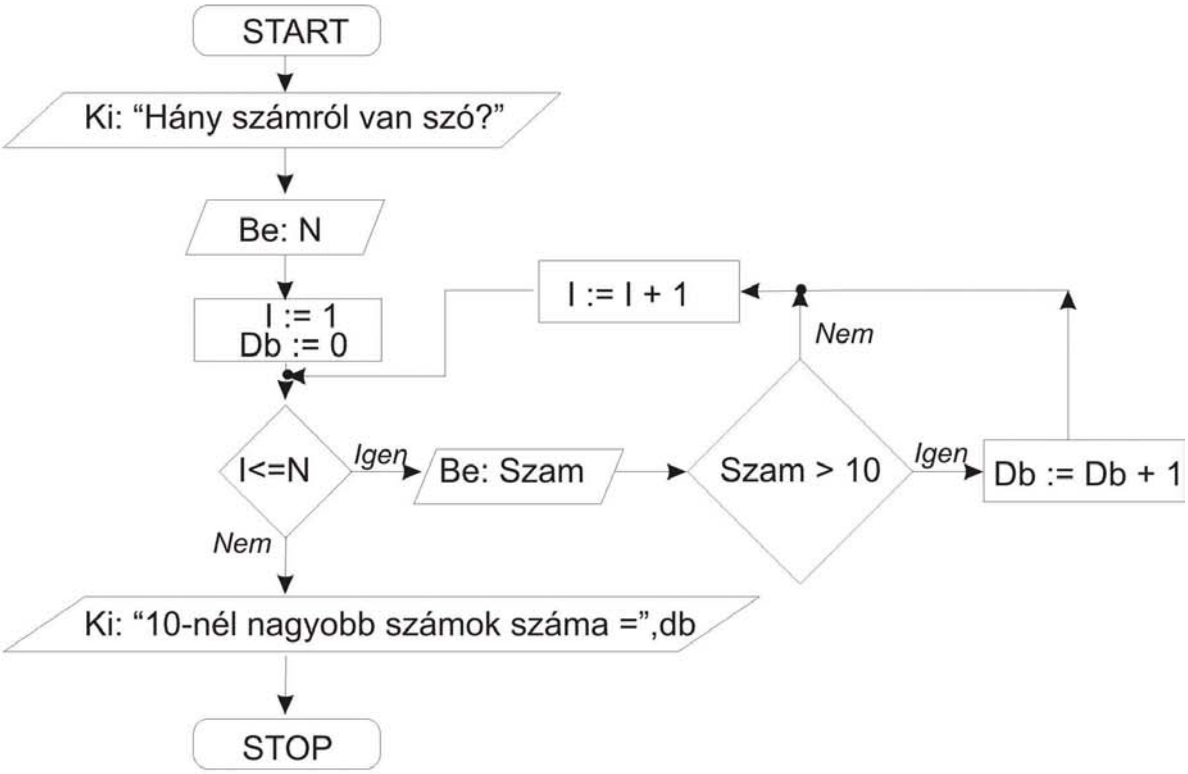
Előltesztelő logikai ciklus. A vezértlő feltétel "Igaz" értéke esetén az al-blokkba zárt ciklusmag utasításai hajtódnak végre. A vezérlő feltétel "Nem" értéke esetén tovább folytatódik az algoritmus végrehajtása a következő utasításon.

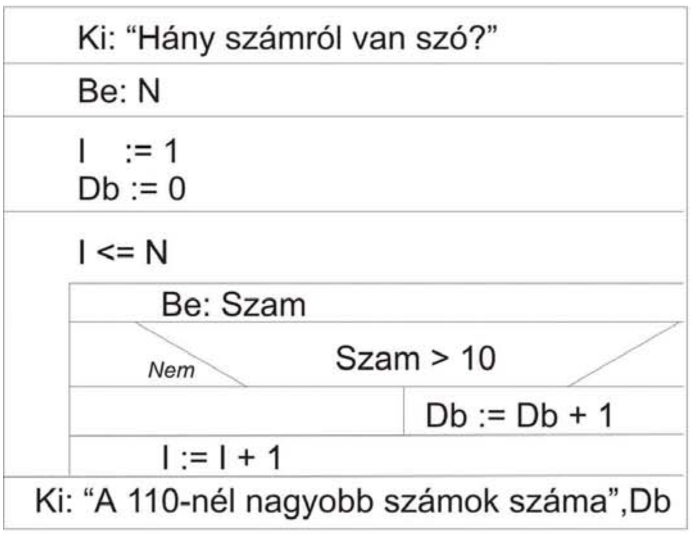


Hátultesztelő pozitv bezérlésű ciklus. A ciklusmag egyszer biztosan végrehajtódik. Amennyiben a vezérlő feltétel "Igen" értékű, a ciklusmag utasításai újra végrehajtódnak. Amennyiben a vezérlő feltétel értéke "Nem"-re vált, az algoritmus végrehajtása a következő utasításon folytatódik.

<u>Leíró nyelv:</u>

ALGORITMUS	Vannak bemenő, kimenő, és átmenő paraméterei
DEKLARÁCIÓ **Változódeklarációk**	Ezek után kell megadni a lokális változókat
AKEZD **Utasítások** **Utasítások**	Algoritmus kezd. Utána következnek az utasítások.
AVÉGE	Algoritmus vége. Utána már nem állhat semmi.
vá := ***kifejezés***	Értékadás
Ki: ***kifejezés***	Output, kiírás a képernyőre. A képernyőre kiíródik a kifejezés értéke.
Be: ***változólista***	Input, bekérés billentyűzetről. A megadott változók értékeit a gép sorban bekéri a billentyűzetről.
HA ***feltétel*** **AKKOR** **Utasítás** **Utasítás** **HVÉGE**	Feltételes elágazás. Ha a feltétel „Igaz”, akkor az „AKKOR” ág utasításai hajtódnak végre. Ha a feltétel „Hamis”, akkor a HVÉGE után következő utasításon folytatódik a végrehajtás.
HA ***feltétel*** **AKKOR** **Utasítás** **Utasítás** **KÜLÖNBEN** **Utasítás** **Utasítás** **HVÉGE**	Feltételes elágazás. Ha a feltétel „Igaz”, akkor az „AKKOR” ág utasításai hajtódnak végre. Ha a feltétel „Hamis”, akkor a „KÜLÖNBEN” ág utasításai hajtódnak végre. A megfelelő ág végrehajtása után a HVÉGE után következő utasításon folytatódik a végrehajtás.
ELÁGAZÁS **HA** ***feltétel*** **Utasítás** **HA feltétel** **Utasítás** **KÜLÖNBEN** **Utasítás** **EVÉGE**	Összetett elágazás. Az első teljesülő feltételhez tartozó utasításblokk hajtódik végre. Ha több feltétel is teljesülne egyszerre, akkor is csak a sorrendben először teljesülőhöz tartozó utasításblokk hajtódik végre. Ha egyik sem teljesül, akkor a „KÜLÖNBEN” ág utasításblokkja hajtódik végre. Ha nincs „KÜLÖNBEN” ág (ez ugyanis nem kötelező), és egyik feltétel sem teljesül, akkor egyik programág sem hajtódik végre. Akár így, akár úgy, az algoritmus végrehajtása az „EVÉGE” utáni soron folytatódik.
CIKLUS AMÍG ***feltétel*** **Utasítás** **Utasítás** **CVÉGE**	Pozitív vezérlésű logikai előltesztelős ciklus. Ha a feltétel „Igaz”, végrehajtódnak a ciklusmag utasításai. Ha a feltétel „Hamis”-sá válik, akkor a CVÉGE utá- ni utasításon folytatódik a végrehajtás. A ciklusmag nem biztos hogy egyszer is végrehajtódik.
CIKLUS **Utasítás** **Utasítás** **AMÍG** ***feltétel***	Pozitív vezérlésű logikai hátultesztelős ciklus. Végrehajtódik a ciklusmag, majd ha a feltétel „Igaz”, újra végrehajtódik ciklusmag utasításai. Ha a feltétel „Hamis”-sá válik, az algoritmus végrehajtása a MIALATT sor után folytatódik.
CIKLUS **Utasítás** **Utasítás** **CVÉGE_HA** ***feltétel***	Negatív vezérlésű logikai hátultesztelős ciklus. Végrehajtódik a ciklusmag, majd ha a feltétel „Igaz”-zá válik, a ciklus kilép. Ha a feltétel „Hamis”, akkor újra végrehajtódnak a ciklusmag utasításai.
CIKLUS cv := ké-tól vé-ig **Utasítás** **Utasítás** **CVÉGE**	Számlálós ciklus. A ciklusváltozó (cv) felveszi a kezdőértéket (ké). Amennyiben ez kisebb vagy egyenlő a végértéknél, úgy végrehajtásra kerül a ciklusmag, majd a cv értéke automatikusan nő egyel. Ha ez még nem nagyobb a vé-nél, akkor újra végrehajtódik a ciklusmag. Ha a lépésköz nem +1, hanem –1, akkor azt külön jelölni kell.






```
AKezd
   Ki: "Hány számról van szó?"
   Be: N
   I := 1
   Db := 0
   Ciklus Amíg I<=N
      Be: Szam
      Ha Szam>10
         Db := Db + 1
      HVége
      I := I + 1
   CVége
   Ki: "A 110-nél nagyobb számok száma",Db
AVége
```

Algoritmusok osztályozása:

- **Egy elemhez egy elemet rendelő algoritmusok osztálya**
  Egy (ill. néhány, de jellegében különböző) bemenő adathoz egy (ill. néhány, de jellegében különböző) kimenő adatot rendel. Pl: másodfokú egyenlet megoldásának algoritmusa.

- **Egy elemhez egy sorozatot rendelő algoritmusok osztálya**
  Egy (ill. néhány, de jellegében különböző) bemenő adathoz egy sorozatot (vagy tömböt) rendel hozzá. Pl: Egy kezdőérték megadása után a tömb feltöltése adott lépésközzel.

- **Egy sorozathoz egy elemet rendelő algoritmusok osztálya**
  Egy bemenő sorozathoz (vagy tömbhöz) egy konkrét elemet rendel hozzá. Pl: lineáris keresés tétele.

- **Egy sorozathoz egy sorozatot rendelő algoritmusok osztálya**
  Egy bemenő sorozathoz egy másik kimenő sorozatot rendel hozzá. Pl: másolás tétele.

- **Egy sorozathoz több sorozatot rendelő algoritmusok osztálya**
  Egy bemenő sorozathoz több kimenő sorozatot rendel hozzá. Pl: szétválogatás tétele.

- **Több sorozathoz egy sorozatot rendelő algoritmusok osztálya**
  Több bemenő sorozathoz egy kimenő sorozatot rendel hozzá. Pl: összefuttatás tétele.

„be”: *Az algoritmusok bemenő paraméterei azt jelzik, hogy az adott változónak van értéke már az algoritmus indulása előtt is (és ezt ki is használjuk az algoritmusban)*
„ki”: *Az algoritmusok kimenő paramétere azt jelzi, hogy ezen változóknak elvileg lehet kezdőértéke az algoritmus indulásakor, de ezt az algoritmusban nem használjuk ki. Viszont az algoritmus ezen változóknak beállít egy értéket, mely érték az algoritmusok lefutása után a további feldolgozások miatt később is érdekes és fontos.*
„átmenő” *: Ezen paraméterek a „be” és „ki” keveredése, vagyis azt jelzik, hogy az algoritmus indulásakor ezen változóknak már van kezdőértékük (amit ki is használunk), de az algoritmus futása közben ezen értéket megváltoztatja, és ezen új érték később is fontos lehet, ezért egyben kimenő paraméter is.*

„előfeltétel”: a bemenő vagy átmenő változók kezdőértékére vonatkozó állítások, melyek szükségesek az algoritmusok helyes működéséhez. Ez azt jelenti, hogy ha ezen változók kezdőértéke ennek megfelelő, akkor az algoritmus is megfelelően fog működni. Ha ennek nem felel meg, akkor az algoritmus még működhet (véletlenül) helyesen, de nem garantált.

„utófeltétel”: a kimenő vagy átmenő változók értékére vonatkozó állítások. Ezek csak akkor garantáltak, ha az „előfeltétel” megfelelő volt.

Pl: „$\forall\ i \in \mathbb{N}$, $1 \le i \le N$-re $T[i] \in \mathbb{N}$, definiált” olvasása: minden olyan egész számra, mely 1..N közé esik (1 és N is lehet) a T[] vektor ezen eleme egy egész számot tartalmaz, és tényleg tartalmaz számot, az algoritmus azt beállította.
Pl: „$N \ge 1$, $N \in \mathbb{N}$” olvasása: N változó induláskor egy egész számot tartalmaz, amely nagyobb, vagy egyenlő 1-el
Pl: „$\forall\ i,j \in \mathbb{N}$, $1 \le i < j \le N$-re: $T[i] \le T[j]$, és T[1..N] új a T[1..N] eredeti egy permutációja” olvasása: A T[i] elem nem nagyobb a T[j] elemnél, ha a j. elem később van, mint az i. elem (vagyis pl. a T[2] elemnél a T[3] csak nagyobb vagy egyenlő lehet, mivel ez minden ilyen esetre igaz kell legyen, ezért a T vektor rendezett). A T az erdetei T egy permutációja azt jelenti, hogy az új T elemek a régi elemek sorrendjének átrendezésével kaphatóak, más elem nem kerülhet bele.

1. Algoritmus Vektorok feltöltése I. - billentyűzetről

***Feladat**: egy adott (N elemű) vektor feltöltése billentyűzetről.*
Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$
***Utófeltétel**:* $\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$*, definiált*

```
1. ALGORITMUS ( be N:konstans egész; ki T : tömb [1..N] egész-ből )
2. DEKLARÁCIÓ
3.    I : egész
4. AKEZD
5.   Ciklus I:=1-től N-ig
6.      Be:  T [ I ]
7.   CVÉGE
8. AVÉGE
```

Megjegyzések:

- az I ciklusváltozó minden egész értéket felvesz 1 és N között, így az algoritmus sorban bekéri az 1., 2., 3., …, N-1., N. tömbértéket.

Ellenőrző kérdések:

1. mi történne, ha a T tömb nem „egész-ből”, hanem „valós-ból” lenne deklarálva ?
2. mi történne, ha az I nem „egész”, hanem „valós” lenne ?
3. mi történne, ha az 5. sorban a ciklus nem 1-től, hanem 2-től indulna ?
4. mi történne, ha az 5. sorban a ciklus nem N-ig hanem N+1-ig menne ?
5. mi történne, ha az 5. sor „Ciklus I:=2-től N+1-ig” lenne, a 6. sor „Be: T[I-1]” lenne ?

2. Algoritmus Véletlen érték meghatározása

***Feladat**: egy adott (A,B) egész számintervallumba eső véletlen szám előállítása*
Előfeltétel: $A \leq B,\ A,B \in \mathbb{N}$
***Utófeltétel**:* $X \in \mathbb{N},\ A \leq X \leq B$

```
ALGORITMUS ( be A,B:konstans egész; ki X:egész )
DEKLARÁCIÓ
AKEZD
  X := Véletlen_Érték( B-A+1 ) + A
AVÉGE
```

Megj: *A* Véletlen_Érték(N) *egy 0..N-1 közötti véletlen egész számot ad meg.*

Megj: *A továbbiakban a* Véletlen_Érték(A és B között) *jelölésen a fenti algoritmust értjük.*

3. Algoritmus — Vektorok feltöltése II. - véletlenszám-generátorral

***Feladat**: egy adott (N elemű) vektor feltöltése véletlen értékekkel*

Előfeltétel: $A \le B,\ A,B \in \mathbb{N}$ és $N \ge 1,\ N \in \mathbb{N}$

Utófeltétel: $\forall\ i \in \mathbb{N},\ 1 \le i \le N$-re: $T[i] \in \mathbb{N}$, definiált, és $A \le T[i] \le B$

```
ALGORITMUS (be N,A,B:konstans egész; ki T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
   I : egész
AKEZD
  Ciklus I:=1-től N-ig
     T [ I ] := Véletlen_Érték( A és B között )
  CVÉGE
AVÉGE
```

4. Algoritmus — Vektorok feltöltése III. - végértékig

***Feladat**: egy adott (N elemű) vektor feltöltése billentyűzetről úgy, hogy egy speciális érték jelzi a vektor végét (leggyakrabban 0).*

Előfeltétel: VegeJel $\in \mathbb{N},\ 1 \le N,\ N \in \mathbb{N}$

Utófeltétel: $Db \in \mathbb{N},\ 1 \le Db \le N$, és
$\forall\ i \in \mathbb{N},\ 1 \le i \le Db$-re: $T[i] \in \mathbb{N}$, definiált, és $T[i] \neq Vege\_Konstans$

```
ALGORITMUS (   be N,VegeJel:konstans egész;
               ki  T : tömb [1..N] egész-ből;
               ki Db:egész )
DEKLARÁCIÓ
    Befejeztuk: logikai
    X : egész
AKEZD
  Db := 0
  Befejeztuk := HAMIS
  Ciklus amíg (Db<=N) és nem Befejeztuk
     Be: X
     HA X = VegeJel AKKOR
          Befejeztuk := IGAZ
     KÜLÖNBEN
          Db := Db + 1
          T[ Db ] := X
     HVÉGE
  CVÉGE
AVÉGE
```

5. Algoritmus Vektorok feltöltése IV. – rendezett módon

***Feladat**: egy adott (N elemű) vektor feltöltése véletlen értékekkel úgy, hogy a vektor eleve rendezett legyen szigorúan növekvő formában*

Előfeltétel: $A \le B,\ A,B \in \mathbb{N},\ 1 \le C,\ C \in \mathbb{N},\ 1 \le N,\ N \in \mathbb{N}$

Utófeltétel: $\forall\ i \in \mathbb{N},\ 1 \le i \le N$-re: $T[i] \in \mathbb{N}$, definiált, és

$T[i] \in [A\ ..\ B+(N-1)*C]$, és

$\forall\ i,j \in \mathbb{N},\ 1 \le i < j \le N$-re: $T[i] < T[j]$

```
1. ALGORITMUS (be A,B,C,N :konstans egész; ki T : tömb [1..N] egész-ből)
2. DEKLARÁCIÓ
3.    I,J : egész
4. AKEZD
5.    T [ 1 ] := Véletlen_Érték( A és B között )
6.    Ciklus I:=2-től N-ig
7.       T [ I ] := T [ I -1 ] + Véletlen_Érték( 1 és C között )
8.    CVÉGE
9. AVÉGE
```

Megjegyzések:

- Amennyiben a 7. sorban „+ *Véletlen_Érték(0 és C között)"* lenne írva, úgy a vektoron belül előfordulhatna ismétlődés (csak növekvő formátum, nem szigorúan növekvő!)

6. Algoritmus Mátrixok feltöltése billentyűzetről.

***Feladat**: egy adott (NxM-s rögzített) méretű mátrix feltöltése billentyűzetről.*

Előfeltétel: $N,M \ge 1;\ N,M \in \mathbb{N}$

Utófeltétel: $\forall\ i,j \in \mathbb{N},\ 1 \le i \le N,\ 1 \le j \le M$-re: $T[i,j] \in \mathbb{N}$, definiált

```
1. ALGORITMUS ( N,M:konstans egész; ki T : tömb [1..N,1..M] egész-ből )
2. DEKLARÁCIÓ
3.      I,J : egész
4. AKEZD
5.    Ciklus I:=1-től N-ig
6.       Ciklus J:=1-től M-ig
7.          Be:  T [ I , J ]
8.       CVÉGE
9.    CVÉGE
10. AVÉGE
```

Megjegyzések:

- A belső ciklus (J) egy teljes sor, az (I. sor) bekérését végzi el.
- A külső ciklus (I) gondoskodik, hogy minden sor bekérésre kerüljön.

Ellenőrző kérdések:

1. mi történne, ha az 5. sor csak egy „I := 3"-ből állna, és a 9. sort elhagynánk
2. mi történne, ha az 5. és 6. sort felcserélnénk az algoritmusban ?
3. mi történne, ha fordítva neveznénk el a ciklusokat ?
4. mi történne, ha a J ciklust is N-ig futtatnánk, vagy ha az I ciklust is M-ig futtatnánk ?

7. Algoritmus Vektorok összeadása konstanssal

***Feladat**: egy adott (adatokkal már feltöltött) N elemű vektor minden egyes elemének értékének megnövelése egy adott másik értékkel*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N},\ Ertek \in \mathbb{N}$

$\forall i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$, *definiált*

***Utófeltétel**:* $\forall i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$, *és*

$T[i] = T[\ i]$ *eredeti értéke + a konstans*

```
ALGORITMUS (be N,Ertek:konstans egész; átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I: egész
AKEZD
  Ciklus I:=1-től N-ig
     T [ I ] := T [ I ] + Ertek
  CVÉGE
AVÉGE
```

Megjegyzések:

- Az „Érték” konstans értékkel kerül minden egyes tömbelem értéke növelésre

Ellenőrző kérdések:

8. Algoritmus Vektorok szorzása konstanssal

***Feladat**: egy adott (adatokkal már feltöltött) vektor minden egyes elemének értékének megszorzása egy konstanssal.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N},\ Ertek \in \mathbb{N}$

$\forall i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$, *definiált*

***Utófeltétel**:* $\forall i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$, *és*

$T[i] = T[\ i]$ *eredeti értéke * a konstans*

```
ALGORITMUS (be N,Ertek:konstans egész; átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I : egész
AKEZD
  Ciklus I:=1-től N-ig
     T [ I ] := T [ I ] * Ertek
  CVÉGE
AVÉGE
```

Megjegyzések:

- Az „Érték” konstans értékkel kerül minden egyes tömbelem értéke szorzásra

Ellenőrző kérdések:

9. Algoritmus Mátrix összeadása konstanssal

***Feladat**: egy adott (adatokkal már feltöltött) mátrix minden egyes elemének értékének megnövelése egy konstanssal.*

Előfeltétel: $N,M \geq 1$, $N,M \in \mathbb{N}$, $Ertek \in \mathbb{N}$

$\forall\ i,j \in \mathbb{N}$, $1 \leq i \leq N$, $1 \leq j \leq M$ -re: $T[i,j] \in \mathbb{N}$, definiált

Utófeltétel: $\forall\ i,j \in \mathbb{N}$, $1 \leq i \leq N$, $1 \leq j \leq M$ -re: $T[i,j] \in \mathbb{N}$, és

$T[i] = T[\ i]$ eredeti értéke + a konstans

```
1.  ALGORITMUS (be N,Ertek:konstans egész; átmenő T : tömb [1..N] egész-ből)
2.  DEKLARÁCIÓ
3.     I,J : egész
4.  AKEZD
5.     Ciklus I:=1-től N-ig
6.        Ciklus J:=1-től M-ig
7.           T [ I , J ] := T [ I , J ] + Ertek
8.        CVÉGE
9.     CVÉGE
10. AVÉGE
```

Megjegyzések:

- A belső ciklus (J) a mátrix adott sorának minden egyes elemére elvégzi a műveletet.
- A külső ciklus (I) gondoskodik róla, hogy minden egyes sorra lefusson a belső ciklus

Ellenőrző kérdések:

- Mi történne, ha felcserélnénk az 5. és 6. sort ?
- Mi történne, ha az I ciklust 1-től M-ig futtatnák ?
- Mi történne, ha felcserélnénk az 5. és 6. sort, de a 7. sorban azt írnánk: T[J,I]:=T[J,I]+Ertek

10. Algoritmus Mátrix szorzása konstanssal

***Feladat**: egy adott (adatokkal már feltöltött) mátrix minden egyes elemének szorzása egy konstanssal.*

Előfeltétel: $N,M \geq 1$, $N,M \in \mathbb{N}$, $Ertek \in \mathbb{N}$

$\forall\ i,j \in \mathbb{N}$, $1 \leq i \leq N$, $1 \leq j \leq M$ -re: $T[i,j] \in \mathbb{N}$, definiált

Utófeltétel: $\forall\ i,j \in \mathbb{N}$, $1 \leq i \leq N$, $1 \leq j \leq M$ -re: $T[i,j] \in \mathbb{N}$, és

$T[i] = T[\ i]$ eredeti értéke * a konstans

```
ALGORITMUS (be N,Ertek:konstans egész; átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
   I,J : egész
AKEZD
   Ciklus I:=1-től N-ig
      Ciklus J:=1-től M-ig
         T [ I , J ] := T [ I , J ] * Ertek
      CVÉGE
   CVÉGE
AVÉGE
```

11. Algoritmus — Vektorok másolása

***Feladat**: egy adott (adatokkal már feltöltött) vektor minden eleméről másolat készítése egy másik, azonos méretű vektorba.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$

$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

Utófeltétel: $\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$ -re: $A[i] \in \mathbb{N}$, definiált, és $A[i]=T[i]$

```
1.  ALGORITMUS (   be N:konstans egész;
                   be T : tömb [1..N] egész-ből;
                   ki A : tömb [1..N] egész-ből )
2.  DEKLARÁCIÓ
3.     I : egész
4.  AKEZD
5.    Ciklus I:=1-től N-ig
6.       A [ I ] := T [ I ]
7.    CVÉGE
8.  AVÉGE
```

Megjegyzések:

- A két vektornak természetesen kompatíbilis alaptípusból kell felépülnie, és az „A" vektornak legalább akkorának kell lennie, mint a „T" vektornak
- Egyes programozási nyelvek megengedik egyforma típusú vektorok esetén az A := T formát, amely ugyanezt jelenti: a „T" vektorról másolatot kell készíteni az „A" vektorba.

Ellenőrző kérdések:

12. Algoritmus — Mátrixok másolása

***Feladat**: egy adott (adatokkal már feltöltött) mátrix minden eleméről másolat készítése egy másik, azonos méretű mátrixba.*

Előfeltétel: $N,M \geq 1,\ N,M \in \mathbb{N}$

$\forall\ i,j \in \mathbb{N},\ 1 \leq i \leq N,\ 1 \leq j \leq M$-re: $T[i,j] \in \mathbb{N}$, definiált

Utófeltétel: $\forall\ i,j \in \mathbb{N},\ 1 \leq i \leq N,\ 1 \leq j \leq M$ -re: $A[i,j] \in \mathbb{N}$, definiált, és $A[i,j]=T[i,j]$

```
1.  ALGORITMUS (be N,M:konstans egész;
                   be T : tömb [1..N,1..m] egész-ből;
                   ki A : tömb [1..N,1..M] egész-ből )
2.  DEKLARÁCIÓ
3.     I,J : egész
4.  AKEZD
5.    Ciklus I:=1-től N-ig
6.      Ciklus J:=1-től M-ig
7.        A [ I , J ] := T [ I , J ]
8.      CVÉGE
9.    CVÉGE
10. AVÉGE
```

Megjegyzések:

Ellenőrző kérdések:

13. Algoritmus — Összegképzés

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemeinek összegének kiszámítása.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$

$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* $Osszeg \in \mathbb{N},\ Osszeg = \Sigma\, T[i]\ \ (i \in \mathbb{N},\ 1 \leq i \leq N)$

```
1.  ALGORITMUS (   be N:konstans egész;
                   be T: tömb [1..N] egész-ből;
                   Ki Osszeg:egész)
2.  DEKLARÁCIÓ
3.      I: egész
4.  AKEZD
5.     Osszeg := 0
6.     Ciklus I:=1-től N-ig
7.          Osszeg := Osszeg + T [ I ]
8.     CVÉGE
9.  AVÉGE
```

Megjegyzések:

- A „0” a szorzat műveletre nézve neutrális elem!
- A vektornak természetesen valamely szám alaptípusból kell felépülnie. Az „összeg” nevű változónak hasonló alaptípusból kell állnia, de ne feledjük, hogy a benne képződő szám már nagyon nagy is lehet.
- Az összegképzést szokás „szumma meghatározás”-nak is nevezni.

Ellenőrző kérdések:

- Mi történne, ha az 5. sort kihagynánk az algoritmusból ?
- Mi történne, ha a 7. sort „Osszeg := T[I] + Osszeg” formában írnánk fel ?
- Mi történne, ha az 5. sort Osszeg := -1 formában írnánk fel ?
- Mi történne, ha az 5. sort „Osszeg:=T[1]” és a 6. sort „Ciklus I:=2-től N-ig” formában írnánk fel ?

Az alábbi algoritmus szintén megoldja a fenti problémát:

```
10. ALGORITMUS (   be N:konstans egész;
                   be T: tömb [1..N] egész-ből;
                   Ki Osszeg:egész)
11. DEKLARÁCIÓ
12.     I: egész
13. AKEZD
14.    Osszeg := T[1]
15.    Ciklus I:=2-től N-ig
16.         Osszeg := Osszeg + T [ I ]
17.    CVÉGE
18. AVÉGE
```

14. Algoritmus Szorzatképzés

Feladat: *egy adott (adatokkal már feltöltött) vektor elemeinek szorzatának kiszámítása.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$
$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált*

Utófeltétel: $Osszeg \in \mathbb{N},\ Osszeg = \pi\, T[i]\ (i \in \mathbb{N},\ 1 \leq i \leq N)$

```
1. ALGORITMUS (   be N:konstans egész;
                  be T: tömb [1..N] egész-ből;
                  ki Szorzat:egész)
2. DEKLARÁCIÓ
3.    I: egész
4. AKEZD
5.    Szorzat := 1
6.    Ciklus I:=1-től N-ig
7.       Szorzat := Szorzat * T [ I ]
8.    CVÉGE
9. AVÉGE
```

Megjegyzések:
- Az „1” a szorzat műveletre nézve neutrális elem!

Ellenőrző kérdések:
- Mi történne, ha az 5. sort kihagynánk az algoritmusból ?
- Mi történne, ha az 5. sort Szorzat := 0 formában írnánk fel ?

Az alábbi algoritmus szintén megoldja a fenti problémát:

```
1. ALGORITMUS (   be N:konstans egész;
                  be T: tömb [1..N] egész-ből;
                  ki Szorzat:egész)
2. DEKLARÁCIÓ
3.    I: egész
4. AKEZD
5.    Szorzat := T [ 1 ]
6.    Ciklus I:=2-től N-ig
7.       Szorzat := Szorzat * T [ I ]
8.    CVÉGE
9. AVÉGE
```

15. Algoritmus Kiválasztás tétele (logikai ciklussal, legkisebb sorszám)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni azon elem sorszámát, amely megfelel egy adott P tulajdonságnak (pl. páros szám-e). Ha több ilyen elem is lenne, a legkisebb sorszámot határozzuk meg. Ha egyetlen ilyen elem sincs, akkor az észrevehető legyen.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$

$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* $Sorszam \in \mathbb{N},\ 0 \leq Sorszam \leq N$, és

$Sorszam = 0$, ha $\nexists\, i \in \mathbb{N}$, hogy $T[i]$ P tulajdonságú,

$0 < Sorszam = j$, ha $T[j]$ P tulajdonságú, és

$\nexists\, i \in \mathbb{N}$, hogy melyre $T[i]$ P tulajdonságú, és $i<j$

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Sorszam:egész)
DEKLARÁCIÓ
  I : egész
AKEZD
  I := 1
  Ciklus Amíg I <= N ÉS NEM PTulajdonságú( T [ I ] )
     I := I + 1
  CVÉGE
  HA  I <= N AKKOR
     Sorszam := I  //„Van közötte P tulajdonságú elem, mégpedig az „ I” sorszámú elem
  KÜLÖNBEN
     Sorszam := 0 // Nincs a vektor elemei között P tulajdonságú elem.
  HVÉGE
AVÉGE
```

Megjegyzések:

- A „PTulajdonságú(elem)” egy fv, amely eldönti az átadott „elem”-ről, hogy ő megfelelő-e, vagyis P tulajdonságú-e
- Ha a vektor egyetlen eleme sem P tulajdonságú, akkor a Sorszam értéke kilépéskor 0 lesz, egyébként pedig a legkisebb sorszámú elem

16. Algoritmus Kiválasztás tétele (logikai ciklussal, legnagyobb sorszám)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni azon elem sorszámát, amely megfelel egy adott P tulajdonságnak (pl. páros szám-e). Ha több ilyen elem is lenne, a legnagyobb sorszámot határozzuk meg. Ha egyetlen ilyen elem sincs, akkor az észrevehető legyen.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$
$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált*

***Utófeltétel**:* $Sorszam \in \mathbb{N},\ 0 \leq Sorszam \leq N$, *és*
$Sorszam = 0$, *ha* $\nexists\ i \in \mathbb{N}$, *hogy* $T[i]$ *P tulajdonságú,*
$0 < Sorszam = j$, *ha* $T[j]$ *P tulajdonságú, és*
$\nexists\ i \in \mathbb{N}$, *hogy melyre* $T[i]$ *P tulajdonságú, és* $i>j$

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Sorszam:egész)
DEKLARÁCIÓ
  I : egész
AKEZD
  I := N
  Ciklus Amíg I >= 1  ÉS NEM PTulajdonságú( T [ I ] )
     I := I - 1
  CVÉGE
  HA  I  >= 1 AKKOR
     Sorszam := I // Van közötte P tulajdonságú elem, mégpedig az „I" sorszámú elem
  KÜLÖNBEN
     Sorszam := 0 // Nincs a vektor elemei között P tulajdonságú elem.
  HVÉGE
AVÉGE
```

17. Algoritmus Kiválasztás tétele (számlálós ciklussal, legkisebb sorszám)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni azon elem sorszámát, amely megfelel egy adott P tulajdonságnak (pl. páros szám-e). Ha több ilyen elem is lenne, a legkisebb sorszámot határozzuk meg. Ha egyetlen ilyen elem sincs, akkor az észrevehető legyen.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$

$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált*

***Utófeltétel**:* $Sorszam \in \mathbb{N},\ 0 \leq Sorszam \leq N$, *és*

Sorszam = 0, ha $\nexists\, i \in \mathbb{N}$, *hogy T[i] P tulajdonságú,*

$0 < Sorszam = j$, *ha T[j] P tulajdonságú, és*

$\nexists\, i \in \mathbb{N}$, *hogy melyre T[i] P tulajdonságú, és* $i<j$

```
ALGORITMUS (  be N:konstans egész;
              be T: tömb [1..N] egész-ből;
              ki Sorszam:egész)
DEKLARÁCIÓ
    I: egész
AKEZD
    Sorszam := 0
    Ciklus I:=1-től N-ig
        HA PTulajdonságú( T [ I ] ) AKKOR
            Sorszam := I
        HVÉGE
    CVÉGE
AVÉGE
```

18. Algoritmus Kiválasztás tétele (számlálós ciklussal, legnagyobb sorszám)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni azon elem sorszámát, amely megfelel egy adott P tulajdonságnak (pl. páros szám-e). Ha több ilyen elem is lenne, a legnagyobb sorszámot határozzuk meg. Ha egyetlen ilyen elem sincs, akkor az észrevehető legyen.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$
$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* $Sorszam \in \mathbb{N},\ 0 \leq Sorszam \leq N$, és
$Sorszam = 0$, ha $\nexists\, i \in \mathbb{N}$, hogy $T[i]$ P tulajdonságú,
$0 < Sorszam = j$, ha $T[j]$ P tulajdonságú, és
$\nexists\, i \in \mathbb{N}$, hogy melyre $T[i]$ P tulajdonságú, és $i>j$

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Sorszam:egész)
DEKLARÁCIÓ
      I: egész
AKEZD
      Sorszam := 0
      Ciklus I:=N-től 1-ig -1esével
            HA PTulajdonságú( T [ I ] ) AKKOR
                  Sorszam := I
            HVÉGE
      CVÉGE
AVÉGE
```

19. Algoritmus — Eldöntés tétele (logikai ciklussal)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni, van-e egyáltalán olyan elem, amely megfelel egy adott P tulajdonságnak (pl. páros szám-e).*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N}$

$\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$*, definiált*

***Utófeltétel**:* $Vane\_e \in \mathcal{H}$

$Van\_e = HAMIS$*, ha* $\nexists\, i \in \mathbb{N}$*, hogy* $T[i]$ *P tulajdonságú,*

$Van\_e = IGAZ$*, ha* $\exists\ i \in \mathbb{N}$*, hogy* $T[i]$ *P tulajdonságú,*

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Van_e:logikai)
DEKLARÁCIÓ
       I: egész
AKEZD
  I := 1
  Ciklus Amíg I <= N ÉS NEM PTulajdonságú( T [ I ] )
       I := I - 1
  CVÉGE
  Van_e := (I  <=  N)
AVÉGE
```

Megj.: a kiválasztás és az eldöntés tétele közel azonos feladattal foglalkozik. Az eldöntés lényege, hogy döntsük el, hogy van-e a vektor elemei között P tulajdonságú elem (csak Igen/Nem válasz). A kiválasztás tétele során ha van, akkor azt is meg kell határozni, mi az elem sorszáma.

Megj. Az eldöntéshez felhasználhatnánk a kiválasztás tételét, és a „HA Sorszam=0 AKKOR Van_E:=HAMIS KÜLÖNBEN Van_e:=IGAZ” sorral befejezhetnénk a problémát.

Megj: Ez utóbbit a „Van_e := (Sorszam=0)” formában írva elegánsabb.

20. Algoritmus — Eldöntés tétele (számlálós ciklussal)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni, vane- egyáltalán olyan elem, amely amely megfelel egy adott P tulajdonságnak (pl. páros szám-e).*

Előfeltétel: $N \geq 1, N \in \mathbb{N}$

$\forall\, i \in \mathbb{N}, 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált*

***Utófeltétel**:* $Vane\_e \in \mathcal{H}$

$Van\_e = HAMIS$, *ha* $\nexists\, i \in \mathbb{N}$, *hogy* $T[i]$ *P tulajdonságú,*

$Van\_e = IGAZ$, *ha* $\exists\, i \in \mathbb{N}$, *hogy* $T[i]$ *P tulajdonságú,*

```
ALGORITMUS (    be N:konstans egész;
                be T: tömb [1..N] egész-ből;
                ki Van_e:logikai)
DEKLARÁCIÓ
      I: egész
AKEZD
  Van_e := HAMIS
  Ciklus I := 1-től N-ig
     HA PTulajdonságú( T [ I ] )
          Van_e := IGAZ
    HVÉGE
   CVÉGE
AVÉGE
```

Megj: ez az implementáció nem olyan hatékony, mint az előző, mert ha már eldőlt, hogy van-e ilyen elem, akkor is folytatja a vizsgálatot, a vektor elemeinek feldolgozását.

Megj: a két megoldás egyforma ideig fut, ha a vektorban egyáltalán nincs P tulajdonságú elem, vagy ha az első ilyen elem sorszáma éppen N.

21. Algoritmus — Vektorok feltöltése V. – ismétlődés kizárt

***Feladat**: egy adott (N elemű) vektor feltöltése véletlen értékekkel úgy, hogy nem lehet a vektoron belül ismétlődés (egy érték kétszer nem fordulhat elő)*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N},\ A,B \in \mathbb{N},\ N \leq B-A,\ A<B$

***Utófeltétel**:* $\forall\ i \in \mathbb{N},\ 1 \leq i \leq N$*-re:* $T[i] \in \mathbb{N}$*, definiált, és* $A \leq T[i] \leq B$*, és* $\nexists\, i,j \in \mathbb{N},\ i \neq j$ *hogy* $T[i]=T[j]$

```
ALGORITMUS (     be A,B,N:konstans egész;
                 ki T: tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J : egész
AKEZD
   I := 1
  Ciklus amíg (I<=N)
     T [ I ] := Véletlen_Érték( A és B között )
     J := 1
     Ciklus amíg (J<I) és (T[ J] <> T[ I ])
        J := J + 1
     CVÉGE
     Ha J>=I AKKOR
        I := I + 1
     HVÉGE
  CVÉGE
AVÉGE
```

22. Algoritmus   Megszámlálás tétele

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei között megszámolni, hogy hány db adott P tulajdonságú eleme (pl. páros szám) van.*
Előfeltétel: *$N \geq 1$, $N \in \mathbb{N}$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált*
***Utófeltétel**:* *Ha $\mathcal{A} := \{ i \mid i \in \mathbb{N},\ T[i]$ elem P tulajdonságú $\}$, akkor $Db = |\mathcal{A}|$.*

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Db:egész)
DEKLARÁCIÓ
  I : egész
AKEZD
   Db := 0
  Ciklus I:=1-től N-ig
     HA PTulajdonságú( T [ I ] ) AKKOR
           Db := Db + 1
     HVÉGE
  CVÉGE
AVÉGE
```

Megj: az eldöntés tételét ezen tétel segítségével is meg lehet valósítani, mert ha Db=0, akkor nem volt ilyen P tulajdonságú elem a vektorban, egyébként pedig volt.

23. Algoritmus   Maximális elem kiválasztás tétele (érték)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni a legnagyobb elem <u>értékét</u>.*
Előfeltétel: *$N \geq 1$, $N \in \mathbb{N}$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált*
***Utófeltétel**:* *$Max := maximum\{ T[i] \mid i \in \mathbb{N} \}$*

```
1.  ALGORITMUS (   be N:konstans egész;
                   be T: tömb [1..N] egész-ből;
                   ki Max:egész)
2.  DEKLARÁCIÓ
3.    I: egész
4.  AKEZD
5.    Max := T [ 1 ]
6.    Ciklus I:=2-től N-ig
7.       HA Max < T [ I ] AKKOR
8.             Max := T [ I ]
9.       HVÉGE
10.   CVÉGE
11. AVÉGE
```

Kérdés: Mely esetekben nem fog az algoritmus helyes működést produkálni, ha az 5. sort „Max:=0”-ra javítjuk, és a 6. sorban a ciklus 1-től indul?

24. Algoritmus — Maximális elem kiválasztás tétele (pozíció)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni a legnagyobb elem sorszámát. Ha ezen maximális érték többször is előfordulna, úgy a legkisebb ilyennek a sorszámát.*

Előfeltétel: $N \geq 1$, $N \in \mathbb{N}$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* Ha $Max := \text{maximum}\{\ T[i] \mid i \in \mathbb{N}\ \}$, akkor

$MaxI := \text{minimum}\{\ i \mid\ i \in \mathbb{N},\ T[i] = Max\ \}$

```
ALGORITMUS (    be N:konstans egész;
                be T: tömb [1..N] egész-ből;
                ki Max:egész)
DEKLARÁCIÓ
   I: egész
AKEZD
  MaxI := 1
  Ciklus I:=2-től N-ig
     HA T [ MaxI ] < T [ I ] AKKOR
           MaxI  := I
     HVÉGE
  CVÉGE
AVÉGE
```

Megj: Ha ismert a legnagyobb elem indexe, akkor az értéke is, hiszen az nem más, mint T[*MaxI*] !

25. Algoritmus — Minimális elem kiválasztás tétele (érték)

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni a legkisebb elem <u>értékét</u>.*

Előfeltétel: $N \geq 1$, $N \in \mathbb{N}$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* $Min := \text{minimum}\{\ T[i] \mid i \in \mathbb{N}\ \}$

```
ALGORITMUS (    be N:konstans egész;
                be T: tömb [1..N] egész-ből;
                ki Min:egész)
DEKLARÁCIÓ
   I: egész
AKEZD
  Min := T [ 1 ]
  Ciklus I:=2-től N-ig
     HA Min > T [ I ] AKKOR
           Min := T [ I ]
     HVÉGE
  CVÉGE
AVÉGE
```

26. Algoritmus — Minimális elem kiválasztás tétele (pozíció)

Feladat: *egy adott (adatokkal már feltöltött) vektor elemei közül meghatározni a legkisebb elem sorszámát. Ha ezen minimum érték többször is előfordulna, úgy a legkisebb ilyennek a sorszámát.*

Előfeltétel: $N \geq 1$, $N \in \mathbb{N}$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

Utófeltétel: Ha $Min := \text{minimum}\{\ T[i] \mid i \in \mathbb{N}\ \}$, akkor
$MinI := \text{minimum}\{\ i \mid i \in \mathbb{N},\ T[i]=Min\ \}$

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki MinI:egész)
DEKLARÁCIÓ
  I: egész
AKEZD
  MinI := 1
  Ciklus I:=2-től N-ig
     HA T [ MinI ] < T [ I ] AKKOR
           MinI  := I
     HVÉGE
  CVÉGE
AVÉGE
```

27. Algoritmus — Kiválogatás tétele

***Feladat**: egy adott (adatokkal már feltöltött) vektor elemei közül egy másik vektorba átemelni azokat az elemeket, amelyek adott P tulajdonsággal rendelkeznek (pl. párosak).*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N},\ \forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

***Utófeltétel**:* Ha $\mathcal{F} := \{\ T[i] \mid i \in \mathbb{N},\ T[i]$ elem P tulajdonságú $\}$,
$ADb := \mid \mathcal{F} \mid$, és $\mathcal{A}[i] := \mathcal{F}$ elemei (rendre) $1 \leq i \leq ADb$-ra

```
ALGORITMUS (   be N:konstans egész;
               be T : tömb [1..N] egész-ből;
               ki ADb : egész;
               ki A : tömb [1..N] egész-ből )
DEKLARÁCIÓ
   I: egész
AKEZD
   ADb := 0
   Ciklus I:=1-től N-ig
      HA PTulajdonságú( T [ I ] ) AKKOR
            ADb := ADb + 1
            A [ ADb ] := T [ I ]
      HVÉGE
   CVÉGE
AVÉGE
```

Megj: Az „A” vektor további feldolgozásához szükséges az „ADb” értékének ismerete, hiszen az „A” vektor elemei csak az 1..ADb részen definiáltak! Pl:

```
HA ADb = 0 AKKOR
   Ki: „Nem volt a T tömbben P tulajdonságú elem!”
KÜLÖNBEN
   Ciklus I := 1-től ADb-ig
      Ki: A [ I ]
   CVÉGE
HVÉGE
```

__Megj:__ A formális specifikációban adott „F” halmaz ismétlődő elemeket is tartalmazhat, ún. multiset, hiszen az „A” vektorban lehet ismétlődés, ezek közül lehetnek olyan elemek, amelyek P tulajdonságúak. Ekkor ezek többszörösen szerepelnek az „F” halmazban is.

28. Algoritmus — Szétválogatás tétele

Feladat: *egy adott (adatokkal már feltöltött) vektor elemei közül egy másik vektorba átemelni azokat az elemeket, amelyek adott P tulajdonsággal rendelkeznek (pl. párosak). Az átemelés során a P tulajdonságú elemeket a másik vektor elejére kell elhelyezni, a többit a vektor végére.*

Előfeltétel: $N \geq 1,\ N \in \mathbb{N},\ \forall\ i \in \mathbb{N},\ 1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, definiált

Utófeltétel: Ha $\mathcal{F}1 := \{\ T[i] \mid i \in \mathbb{N},\ T[i]$ elem P tulajdonságú $\}$,
és $\mathcal{F}2 := \{\ T[i] \mid i \in \mathbb{N},\ T[i]$ elem nem P tulajdonságú $\}$,
akkor $Ae := |\ \mathcal{F}1\ |$, és az $A\ [1..Ae.] := \mathcal{F}1$ elemei (rendre), és
$A[Ae+1..N] := \mathcal{F}2$ elemei (fordított sorrendben rendre)

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               ki Ae: egész;
               ki A : tömb [1..N] egész-ből)
DEKLARÁCIÓ
    I,Av: egész
AKEZD
  Ae := 0
  Av := N
  Ciklus I:=1-től N-ig
     HA PTulajdonságú( T [ I ] ) AKKOR
          Ae := Ae + 1
          A [ Ae ] := T [ I ]
     KÜLÖNBEN
          A [ Av ] := T [ I ]
          Av := Av - 1
     HVÉGE
  CVÉGE
AVÉGE
```

Megj: Az „A” vektor további feldolgozásához szükséges lesz az „Ae” ismerete:

```
  Ciklus I := 1-től De-ig
        Ki : „P tulajdonságú elemek:”, A [ I ]
  CVÉGE
  Ciklus I := N-től De+1-ig –1-esével
          Ki: „Nem P tulajdonságú elemek:”, A [ I ]
  CVÉGE
```

Megj: a fenti kiírások (feldolgozások) akkor is helyesen működnek, ha a T vektorban minden elem/egy elem sem volt P tulajdonságú!

Megj: ha Ae (A tömb eleje) db P tulajdonságú elem volt, akkor a nem P tulajdonságú elemek számát könnyű meghatározni: N-Ae!

29. Algoritmus Logaritmikus keresés (bináris keresés) tétele

***Feladat**: egy adott (adatokkal már feltöltött) **rendezett** vektor elemei között egy adott (X) érték megkeresése. Ha az elem előfordul, adjuk meg a sorszámát (K). Ha nincs a vektor elemei között, akkor jelezzük (K=0).*

Előfeltétel: $N \geq 1$, $N \in \mathbb{N}$, $\forall\, i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált, és*
$\forall\, i,j \in \mathbb{N}$, $i<j$-re $T[i] <= T[j]$ *(T rendezett)*
$X \in \mathbb{N}$, *definiált (nem feltétlenül eleme a vektornak)*

***Utófeltétel**:* $K \in \mathbb{N}$, $0 \leq K \leq N$, *és*
$K=0$, *ha* $\not\exists\, i \in \mathbb{N}$, *hogy* $T[i]=X$,
$1 \leq K=j$, *ha* $T[j]=X$

```
ALGORITMUS (   be N:konstans egész;
               be T: tömb [1..N] egész-ből;
               be: X: egész;
               ki K: egész)
DEKLARÁCIÓ
   A,F: egész
AKEZD
  A := 1
  F := N
  Ciklus
     K :=  ( A + F ) / 2
     ELÁGAZÁS
         HA    T[K] < X   AKKOR
               A := K + 1
         HA    T[K] > X   AKKOR
               F := K - 1
     EVÉGE
  CVÉGE_HA  T[K] = X    VAGY   A > F
  HA  A > F  AKKOR
       K := 0
  HVÉGE
AVÉGE
```

***Megj:** A keresés (a ciklusmag végrehajtásának) maximális időigénye* $log_2(N)$*, ezért ezen keresést logaritmikus keresésnek hívják.*
***Megj:** Mivel a* $log_2(N)$ *azt a számot adja meg, hogy 2-t hanyadik hatványra kell emelni, hogy éppen N-t kapjunk, s ez akár tört szám is lehet, az lépésszám ezen* $log_2(N)$ *pozitív valós szám felfelé kerekítése a legközelebbi tőle nem kisebb egész számra. Ezt* $\lceil \log_2(N) \rceil$*-el jelöljük.*
***Megj:** 1024 elemből álló vektorban a fenti keresés 10 lépésen belül véget ér, mivel* $2^{10}=1024$*. 65536 elemű vektor esetén maximum 16 lépés kell az eredmény előállításához. Ez nagyon kedvező idő! Ezen fenti keresést bináris keresésének is hívják.*

Megj: Az „elágazás” helyett sokszor két <u>külön</u> „ha”-t írnak. Ha egy „ha”-t írunk, és az „F:=K-1”-et egy „különben”-be tesszük, az algoritmus hibásan fog működni!!!!!

30. Algoritmus Két változó cseréje segédváltozóval

***Feladat**: adott két elemi változó. Cseréljük meg a két változó értékét.*
Előfeltétel: $A,B \in \mathbb{N}$, *definiáltak*
***Utófeltétel**:* *A=B eredeti értéke, B = A eredeti értéke*

```
ALGORITMUS ( átmenő A,B: egész )
DEKLARÁCIÓ
    Seged: egész
AKEZD
   Seged := A
   A     := B
   B     := Seged
AVÉGE
```

Megj: *A fenti algoritmusra a továbbiakban Csere(A,B) formában fogunk hivatkozni.*

31. Algoritmus Vektorok feltöltése VI. – intervallum minden eleme

***Feladat**: egy adott (N elemű) vektor feltöltése véletlen értékekkel úgy, hogy minden érték szerepeljen 1..N között, és nem lehet ismétlődés*
Előfeltétel: $N \in \mathbb{N}$, $N \geq 1$
Utófeltétel: $\forall\, i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált*, $1 \leq T[i] \leq N$, *és*
$\nexists\, i,j \in \mathbb{N}$, $i \neq j$, *hogy* $T[i]=T[j]$

```
ALGORITMUS (  be N: konstans egész;
              ki T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
   I,J,K : egész
AKEZD
     Ciklus I := 1-től N-ig
         T [ I ] := I
    CVÉGE
    Ciklus I:=1-től N*10-ig
       J := Véletlen_Érték( 1-N között )
       K := Véletlen_Érték( 1-N között )
       Csere( T[J], T[K])
    CVÉGE
AVÉGE
```

Megj: Ha J=K, akkor a Csere semmit nem változtat a vektoron belül (saját magával történt megcserélés). A csere elé el lehetne helyezni egy J<>K feltételt. De elég ritkán történik meg a J=K egyenlőség, mely időben felesleges csere hajtódna végre, ugyanakkor ezen feltétel kiértékelésére mindig sort kerítene az algoritmus, ezért ezen vizsgálat beillesztése elképzelhető, hogy nem okoz komoly sebességnövekedést.
Megj: A N*10 kifejezésben a 10-es szorzó tapasztalati érték. Alaposabb keveréshez használjunk alkalmasint nagyobb konstans szorzót.
Megj: ha az intervallum értékeit véletlenszám-generálással sorsolnánk, nagyon lassú futásidejű algoritmust kapnánk!

Nagyság szerint növekvőben rendező algoritmusok feladata, előfeltétele, és utófeltétele közös, ezért ezt csak egyszer adjuk meg: Csökkenőbe rendezés esetén ezek átírása triviális.

***Feladat**: egy adott (adatokkal már feltöltött) <u>rendezetlen</u> vektor elemeit rendezni kell nagyság szerint növekvő sorrendbe.*

Előfeltétel: $N \in \mathbb{N}$, $N \geq 1$, $\forall\ i \in \mathbb{N}$, $1 \leq i \leq N$-re: $T[i] \in \mathbb{N}$, *definiált,*

Utófeltétel: $\forall\ i,j \in \mathbb{N}$, $1 \leq i < j \leq N$-re: $T[i] \leq T[j]$, *és*
$T[1..N]$ *új a* $T[1..N]$ *eredeti egy permutációja*

32. Algoritmus Cserélő rendezés (rendezés közvetlen elemkiválasztással)

```
ALGORITMUS (    be N: konstans egész;
                átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
   I,J : egész
AKEZD
    Ciklus I := 1-től  N-1-ig
      Ciklus J := I + 1-től   N-ig
           HA   T[ I ]  >  T[ J ]   AKKOR
                Csere( T[i], T[j] )
           HVÉGE
      CVÉGE
    CVÉGE
AVÉGE
```

Megj: *a relációjel megfordítása esetén a rendezés csökkenő sorrendbe fog történni.*

Ciklusinvariás: *A belső ciklus (J) minden egyes lefutásának végére a tömb I. eleme a rendezettségben I. elem értékét fogja tartalmazni.*

Megj: *A belső ciklus (J) minden egyes lefutásának végére a tömb I. eleme a „helyére kerül", hiszen összehasonlításra kerül a többi elemmel, és ha azok közül valamelyik kisebb lenne nála, akkor a csere folytán ő kerül az I. elem helyére.*

Megj: Ennek megfelelően **I**=1 esetén (első lefutás) az 1. elem helyére kerül a vektor legkisebb eleme, I=2-re a 2. elem helyére a vektor második legkisebb eleme, stb…

Megj: Utoljára I=N-1-re fut le a belső ciklus, melynek hatására A vektor N-1 helyére kerül a megfelelő elem. De ekkor a vektor N-edik eleme is a helyére kerül!

Megj: *Egy J menetben több csere is történhet, mire az I. elem a megfelelőre elemre cserélődik. Ezért ezen rendezés kis hatékonyságú (lassú).*

Kérdés: Mi történne, ha a J ciklus nem **I**+1-től, hanem 1-től indulna?

33. Algoritmus Maximumkiválasztásos rendezés

```
ALGORITMUS (    be N: konstans egész;
                átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J,MaxI : egész
AKEZD
    Ciklus I := N-től  2-ig –1-esével
        MaxI := I
        Ciklus J := I - 1-től   1-ig  -1-esével
            HA   T [ MaxI ]  <  T [ J ]   AKKOR
                 MaxI  := J
            HVÉGE
        CVÉGE
        HA MaxI <> I AKKOR
            Csere( T[MaxI], T[I] )
        HVÉGE
    CVÉGE
AVÉGE
```

Megjegyzés: a relációjel megfordítása esetén a rendezés csökkenő sorrendbe fog történni, bár ekkor a „MaxI” változót inkább „MinI” változónak kellene hívni . Ekkor gyakorlatilag a minimumkiválasztásos rendezést kapjuk, annak is a csökkenő sorrendbe rendező változatát).

34. Algoritmus Minimumkiválasztásos rendezés

```
ALGORITMUS (    be N: konstans egész;
                átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J,MinI : egész
AKEZD
    Ciklus I := 1-től  N-1-ig
        MinI := I
        Ciklus J := I+ 1-től   N-ig
            HA   T [ MinI ]  >  T [ J ]   AKKOR
                 MinI  := J
            HVÉGE
        CVÉGE
        HA MinI <> I AKKOR
            Csere( T[ MinI ], T[ I ] )
        HVÉGE
    CVÉGE
AVÉGE
```

Megjegyzés: a relációjel megfordítása esetén a rendezés csökkenő sorrendbe fog történni, bár ekkor a „MinI” változót inkább „MaxI” változónak kellene hívni. Ekkor gyakorlatilag a maximumkiválasztásos rendezést kapjuk, annak is a csökkenő sorrendbe rendező változatát).

35. Algoritmus **Buborék-rendezés I. változat**

```
ALGORITMUS (    be N: konstans egész;
                átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J : egész
AKEZD
    Ciklus I := N-1-től  1-ig –1-esével
        Ciklus J := 1-től   I-ig
            HA   T [ J ]  >  T [ J + 1 ]   AKKOR
                Csere( T[ J ], T[ J+1 ] )
            HVÉGE
        CVÉGE
    CVÉGE
AVÉGE
```

Megj: kulcsszó: szomszédos elemek összehasonlítása!
Megj: egy eredetileg rendezett tömbön is elég sokáig dolgozik!

36. Algoritmus **Buborék-rendezés II. változat**

```
ALGORITMUS (    be N: konstans egész;
                átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J : egész
  Vege:logikai
AKEZD
    I  :=  N - 1
    Vege  := HAMIS
    Ciklus Amíg    I >= 1   ÉS   NEM Vege
        Vege  :=  IGAZ
        Ciklus  J := 1-től  I-ig
            HA   T [ J ]  >  T [ J + 1 ]   AKKOR
                Csere( T[ J ], T[ J+1 ] )
                Vege  :=  HAMIS
             HVÉGE
        CVÉGE
        I := I - 1
    CVÉGE
AVÉGE
```

Megj: *A rendezés menet közben abbamarad, ha egy belső ciklus menetben (J) már nem történik csere (ekkor később sem fog már!).*
Megj: *az előrendezettséget is figyeli, ha a tömb már „félúton" rendezetté válik, azonnal kilép.*

37. Algoritmus Buborék-rendezés III. változat

```
ALGORITMUS (   be N: konstans egész;
               átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  I,J : egész
  UtolsoCsere : egész
AKEZD
    I := N - 1
    Ciklus Amíg   I >= 1
         UtolsoCsere := 0
         Ciklus J := 1-től I-ig
              HA  T[ J ] > T[ J + 1 ]  AKKOR
                   C sere( T[ J ], T[ J+1 ] )
                   UtolsoCsere := J
              HVÉGE
         CVÉGE
         I := UtolsoCsere
    CVÉGE
AVÉGE
```

Megj: *A belső ciklus hossza megfelelő esetben nem csak 1-el csökkenhet minden végrehajtás után (rohamos csökkenés).*

Megj: *Ha a tömb már „félúton” rendezetté válik, azonnal kilép.*

38. Algoritmus Beszúró rendezés (Póker-rendezés)

```
ALGORITMUS (   be N: konstans egész;
               átmenő T : tömb [1..N] egész-ből)
DEKLARÁCIÓ
  X,I,J : egész
AKEZD
    Ciklus I := 2-től N-ig
         J :=  I – 1
         X := T[ I ]
         Ciklus Amíg J >= 1   ÉS   X < T[ J ]
              T[ J + 1 ] := T[ J ]
              J := J - 1
         CVÉGE
         T[ J + 1 ]  :=  X
    CVÉGE
AVÉGE
```

Megj: a rendezés elve a kártyák sorbarendezésére hasonlít pókerezés közben a kézben.

39. Algoritmus — Listás rendezés (MU-ME)

***Feladat**: Meg kell határozni egy adott (adatokkal már feltöltött) rendezetlen vektor elemeinek rendezett sorrendjét a nélkül, hogy az eredeti sorrenden változtatnánk!*

```
ALGORITMUS (   be N: konstans egész;
               be T : tömb [1..N] egész-ből;
               ki: Mu : tömb [0..N] egész-ből)
DEKLARÁCIÓ
  Me,Mk,X,I  : egész
AKEZD
     Ciklus I  :=  0-től  N-ig
           Mu [ I ] := 0
     CVÉGE
     MU [ 0 ]  :=  1
     Ciklus I  :=  2-től  N-ig
           Me :=  0
           Mk := MU[ ME ]
           X :=  T [ I ]
           Ciklus  Amíg  MK > 0   ÉS   X > T [ Mk ]
                 Me :=  Mk
                 Mk := MU[ ME ]
           CVÉGE
           MU [ ME ]  :=  I
           MU [ I ] := MK
     CVÉGE
AVÉGE
```

Megj: Me: „Mutató az Előzőre”, Mk: „Mutató a következőre”
Megj: A „MU” vektor elemei valójában a T vektor tömbindexei, 1..N terjedő számok. A 0 érték is előfordul a MU vektorban, jelentése speciális, a lánc végét jelenti.
Megj:*A MU elemei láncolat formában tartalmazzák a T következő elemének sorszámát:*

Pl:
MU[0] = 2
MU[1] = 3
MU[2] = 1
MU[3] = 0

A láncot a 0. elemnél kell elkezdeni kiolvasni. Ez azt mutatja, hogy a T rendezett sorrendjében a 2. elem a legkisebb. A MU[2] szering a következő legkisebb elem a T 1. eleme. MU[1] szerint a rákövetkező elem a T 3. eleme. A MU[3] 0-t tartalmaz, ezzel jelzi, hogy vége a láncnak.
***Megj:** A MU[N]=M azt jelöli, hogy a rendezettségben a T[M] elem következik, és hogy tudjuk folytatni, el kell olvasni a MU[M] értékét. Ezt addig, amíg 0-t nem tartalmaz a MU vektor.*
***Megj:** A „MU” vektor segítségével pl. az alábbi módon lehet a T vektor elemit rendezetten feldolgozni:*

```
     // ……………………………………. kiíratás a MU tömb segítségével
     Mk := 0
     Ciklus  J := 1-től N-ig
          Ki:  T [  MU [ Mk ]  ]
          Mk :=  MU [ Mk ]
     CVÉGE
```

40. Algoritmus Összefuttatás tétele (nincs strázsa elem)

***Feladat**: két, eleve rendezett vektor elemeinek összefésülése egy újabb vektorba úgy, hogy az is rendezett legyen.*

```
ALGORITMUS (   be N,M: konstans egész;
               be A : tömb [1..N] egész-ből;
               be B: tömb [1..M] egész-ből;
               ki C : tömb [1..N+M] egész-ből)
DEKLARÁCIÓ
  Adb,Bdb,Cdb : egész
AKEZD
   Adb := 1
   BDb := 1
   CDb := 0
   Ciklus Amíg  Adb <= N  ÉS   Bdb <= M
      HA A [ Adb ] > B [ Bdb ] AKKOR
         Cdb := Cdb + 1
         C [ Cdb ] := B [ Bdb ]
         Bdb := Bdb + 1
      KÜLÖNBEN
         Cdb := Cdb + 1
         C [ Cdb ] := A [ Adb ]
         Adb := Adb + 1
      HVÉGE
   CVÉGE
   Ciklus Amíg  Adb <= N
         Cdb := Cdb + 1
         C [ Cdb ] := A [ Adb ]
         Adb := Adb + 1
   CVÉGE
   Ciklus Amíg  Bdb <= M
         Cdb := Cdb + 1
         C [ Cdb ] := B [ Bdb ]
         Bdb := Bdb + 1
   CVÉGE
AVÉGE
```

***Megj:** Az első „nagy” ciklus valamelyik (A vagy B) vektor minden elemét átmásolja a C vektorba. A másik két „kis” vektor a maradék elemeket is hozzáteszi. A két kis ciklus közül egy konkrét futás esetén csak az egyik fog elindulni – amelyik vektort a nagy ciklus nem fejezett be! De mivel általános esetben nem lehet tudni, melyik lesz az, ezért kell mindkettőt beletervezni az algoritmusba!*

***Megj:** Az algoritmusnak van olyan változata is, amelyben a „nagy” ciklus belsejében külön esetként kezelik az „A[Adb]=B[Bdb]” esetet, s ekkor ezen egyenlő értékek közül csak az egyiket teszik át a C-be, de mindkét számlálót (Adb, Bdb) növelik. Ezt akkor használjuk, ha az A és B vektor elemei között nem lehet ismétlődés, és azt szeretnénk, hogy a C-ben se legyen. Ekkor a C elemszáma nem feltétlenül lesz N+M! Ekkor az algoritmusnak kimenő adatként kell a Cdb-t feltüntetnie!*

41. Algoritmus — Összefuttatás tétele (strázsa elemmel)

***Feladat**: két, eleve rendezett vektor elemeinek összefésülése egy újabb vektorba úgy, hogy az is rendezett legyen.*
***Előfeltétel**: Az A[] és B[] vektorok 1-el nagyobb méretűek!*

```
ALGORITMUS (   be N,M: konstans egész;
               be A : tömb [1..N+1] egész-ből;
               be B: tömb [1..M+1] egész-ből;
               ki C : tömb [1..N+M] egész-ből)
DEKLARÁCIÓ
  Adb,Bdb,Cdb : egész
AKEZD
   Adb  := 1
   BDb  := 1
   CDb  := 0
   A [ N + 1 ] := +∞
   B [ M + 1 ] := +∞
   Ciklus Amíg   Adb <= N   VAGY   Bdb <= M
      HA  A [ Adb ] > B [ Bdb ]  AKKOR
         Cdb := Cdb + 1
         C [  Cdb  ]  := B [ Bdb ]
         Bdb := Bdb + 1
      KÜLÖNBEN
         Cdb := Cdb + 1
         C [  Cdb  ]  := A [ Adb ]
         Adb := Adb + 1
      HVÉGE
   CVÉGE
AVÉGE
```

Megj: *Az algoritmusból hiányzik a két „kis” ciklus, köszönhetően annak, hogy a „nagy” ciklus mindkét vektor mindkét elemét átemeli a C vektorba.*

Megj: *Indulás előtt az A[N+1] és B[M+1] elemét be kell állítani egy olyan extrémen nagy értékre, melynél csak kisebb elemek fordulnak elő a vektorokban. Ezzel oldjuk azt meg, hogy amikor valamely vektor utolsó elemét is betesszük a C vektorba, és léptetjük a számlálóját, a következő cikluslépésben az összehasonlítás során mindenképpen a másik vektor soron következő eleme legyen a kisebb, az átrakandó.*

Megj: ezen funkció betöltéséhez az A[N+1] := „B legnagyobb eleme”+1, beállítás is elegendő lenne, de ehhez egy menetben meg kellene határozni a B legnagyobb (maximális) elemének értékét, amely csak lassítaná a futást (és hasonlóan az A vektorra is)

Megj: Az algoritmus konkrét nyelvi implementációjában a **+∞** természetesen nem írható le. Helyette a konkrét feladat ismeretében valamely elképzelhetetlenül nagy értéket használnak (pl ha az A és B vektorok egy folyó vizének hőmérséklet-adatait tartalmazzák, akkor a 101 is megfelelő lesz!). Ha ilyen nincs, akkor az „egész” típus adott programnyelvében ábrázolható legnagyobb „egész” számot használják.

42. Algoritmus **Veremkezelés - STACK**

A verem egy olyan összetett adatszerkezet, amely véges mennyiségű homogén adat tárolására képes úgy, hogy az adathozzáférés mechanizmusa LIFO rendszerű (LIFO = Last In First Out, Utolsó Be Először Ki).
A vermet egy vektor segítségével szimuláljuk. A veremkezelő algoritmus-részletek a feltüntetett paramétereken kívül minden esetben átmenő paraméterként megkapják a T vektort, és a VM változót is.

```
DEKLARÁLÁS
        T : tömb [1..N] egész-ből
        VM : egész
```

Feladat: a verem kezdőállapotba hozása. (INIT)

```
ELJÁRÁS Verem_Inic
        VM := 0
EVÉGE
```

Feladat: megállapítani, hogy a verem üres-e (EMPTY)

```
ELJÁRÁS Verem_Ures( ki Ures_e:logikai)
        Ures_e := (VM=0)
EVÉGE
```

Feladat: megállapítani, hogy a verem tele van-e (FULL)

```
ELJÁRÁS Verem_Tele(ki Tele_e:logikai)
        Tele_e := (VM=N)
EVÉGE
```

Feladat: az „X" elem elhelyezése a verembe (PUSH)

```
ELJÁRÁS VeremBe_Elhelyezes ( be X : egész; ki Sikeres_volt:logikai )
  HA NEM Verem_Tele AKKOR
       VM := VM + 1
       T [ VM ] := X
         Sikeres_volt := IGAZ
  KÜLÖNBEN
         Sikeres_volt := HAMIS
  HVÉGE
EVÉGE
```

Feladat: elem kiolvasása a veremből (POP)

```
ELJÁRÁS VeremBol_Olvasas( ki X:egész; ki Sikeres_volt:logikai )
  HA NEM Verem_Ures AKKOR
       X := T [ VM ]
       VM := VM - 1
         Sikeres_volt := IGAZ
  KÜLÖNBEN
         Sikeres_volt := HAMIS
  HVÉGE
EVÉGE
```

Léptetéses sor kezelése – ADVANCING QUEUE

A sor egy olyan összetett adatszerkezet, amely véges mennyiségű homogén adat tárolására képes úgy, hogy az adathozzáférés mechanizmusa FIFO rendszerű (FIFO = First In First Out, Első Be Először Ki). *A sort egy vektor segítségével szimuláljuk. A sorkezelő algoritmus-részletek a feltüntetett paramétereken kívül minden esetben átmenő paraméterként megkapják a T vektort, és a Berak és TaroltDb változókat is.*

```
DEKLARÁLÁS
        T : tömb [1..N] egész-ből
        Berak, TaroltDb : egész

ELJÁRÁS Sor_Inic
  Berak := 0          // hová tettük be az utolsó elemet
  TaroltDb := 0       // jelenleg hány tárolt elem van a sorban
EVÉGE

ELJÁRÁS Sor_Ures( ki Ures_e:logikai)
  Ures_e := (TaroltDb = 0)
EVÉGE

ELJÁRÁS Sor_Tele (ki Tele_e:logikai)
  Tele_e := (TaroltDb = N)
EVÉGE

ELJÁRÁS SorBa_Elhelyezes ( be X : egész; ki Sikeres_volt:logikai )
  HA NEM Sor_Tele AKKOR
      Berak := Berak + 1
      T [ Berak ] := X
      TaroltDb := TaroltDb + 1
      Sikeres_volt := IGAZ
  KÜLÖNBEN
      Sikeres_volt := HAMIS
  HVÉGE
EVÉGE

ELJÁRÁS SorBol_Olvasas( ki X:egész; ki Sikeres_volt:logikai)
  HA NEM Sor_Ures AKKOR
      X := T [ 1 ]
      Ciklus I := 1-től TaroltDb-1 -ig
        T [ I ] := T [ I+1  ]
      CVÉGE
      TaroltDb := TaroltDb - 1
      Sikeres_volt := IGAZ
  KÜLÖNBEN
      Sikeres_volt := HAMIS
  HVÉGE
EVÉGE
```

Ciklikus sorok – CYCLIC QUEUE

A sort egy vektor segítségével szimuláljuk. A sorkezelő algoritmus-részletek a feltüntetett paramétereken kívül minden esetben átmenő paraméterként megkapják a T vektort, és a Berak, TaroltDb és Kivesz változókat is.

```
DEKLARÁLÁS
  T : tömb [1..N] egész-ből
  Berak, TaroltDb : egész

ELJÁRÁS Sor_Inic
  Berak    := 0          // hová tettük be az utolsó elemet
  TaroltDb := 0          // jelenleg hány tárolt elem van a sorban
  Kivesz   := 0          // honnan vettük ki az utolsó elemet
EVÉGE

ELJÁRÁS SorBa_Elhelyezes ( be X : egész; ki Sikeres_volt: logikai )
  HA NEM Sor_Tele AKKOR
    HA Berak=N AKKOR
      Berak := 0
    HVÉGE
    Berak := Berak + 1
    T [ Berak ] := X
    TaroltDb := TaroltDb + 1
    Sikeres_volt := IGAZ
  KÜLÖNBEN
    Sikeres_volt := HAMIS
  HVÉGE
EVÉGE

ELJÁRÁS SorBol_Olvasas( ki X:egész; ki Sikeres_volt:logikai )
  HA NEM Sor_Ures AKKOR
    Ha Kivesz=N AKKOR
      Kivesz := 0
    HVÉGE
    Kivesz := Kivesz + 1
    X := T [ Kivesz ]
    TaroltDb := TaroltDb – 1
    Sikeres_volt := IGAZ
  KÜLÖNBEN
    Sikeres_volt := HAMIS
  HVÉGE
EVÉGE
```

Megj: *a „Sor_tele” és „Sor_ures” eljárások változatlan formában átvehetők az előző változatból*

Témakörök / Algoritmusok

Esti tagozat, I. félév

- Algoritmus fogalma, tulajdonságai. Adatok, alapfogalmak. Értéktípusok (egyszerű, összetett). Logaritmikus keresés (Bináris keresés)
- Logikai alapfogalmak. Logikai műveletek. Logikai azonosságok. Ciklusok. Beszúró rendezés (Póker)
- Tevékenységszerkezetek. Szekvencia. Elágazás. Kiválogatás tétele, Buborék-III.
- Algoritmus leíró eszközök. Leíró nyelv. Folyamatábra. Struktogramm. Szétválogatás tétele.
- Vektorok feltöltése I. II. III. Mátrixok feltöltése. Vektorok és mátrixok kiírása képernyőre. Ciklikus sorok.
- Egydimenziós és kétdimenziós tömb összeadása, szorzása konstanssal Megszámlálás tétele, Buborék-II,
- Összegképzés és szorzatképzés tétele. Maximum kiválasztás tétele 1, 2. Cserélő rendezés (Rendezés közvetlen kiválasztással)
- Vektorok feltöltése IV. V. VI. Eldöntés tétele 1, 2. Összefuttatás tétele 1, 2.
- Minimum kiválasztás tétele 1, 2. Lineáris keresés tétele 1, 2.
- Másolás tétele, Buborék-I., Maximumkiválasztásos rendezés
- Listás rendezés (MU-ME), Veremkezelés.
- Léptetős sorok. Minimumkiválasztásos rendezés

Hernyák Zoltán
főisk. adj.