Algoritmusok és a programozás alapjai

Dr. Varga Imre
Debreceni Egyetem, Informatikai Kar

2.2.0 verzió
2024.12.11

EFOP-3.4.3-16-2016-00021

Témák

- Hogyan írjunk le és oldjunk meg problémákat?
- Mit jelent az **algoritmikus gondolkodás**?
- Mi is az az algoritmus?
- Milyen tulajdonságai vannak?
- Hogyan írhatunk le algoritmusokat?
- Mit jelent a ‘programírás’?
- A programozáshoz kell absztrakt gondolkodás?
- Példák, példák, példák…
- *És még sok minden más…*

Költő vs Programozó

Jó programozó:

nyelv ismeret + algoritmikus gondolkodás

(forma) *(tartalom)*

Lenni vagy nem lenni: az itt a kérdés.
Akkor nemesb-e a lélek, ha tűri
Balsorsa minden nyűgét s nyilait;
Vagy ha kiszáll tenger fájdalma ellen,
S fegyvert ragadva véget vet neki?

William Shakespeare

Lenni s lenni: az itt nem kérdés.
Akkor véget vet-e neki, a nemesb lélek
Ha kiszáll tenger a fájdalma ellen,
Vagy minden balsors fegyvert s nyilait
Vagy nyűgét ragadva; ha tűri?

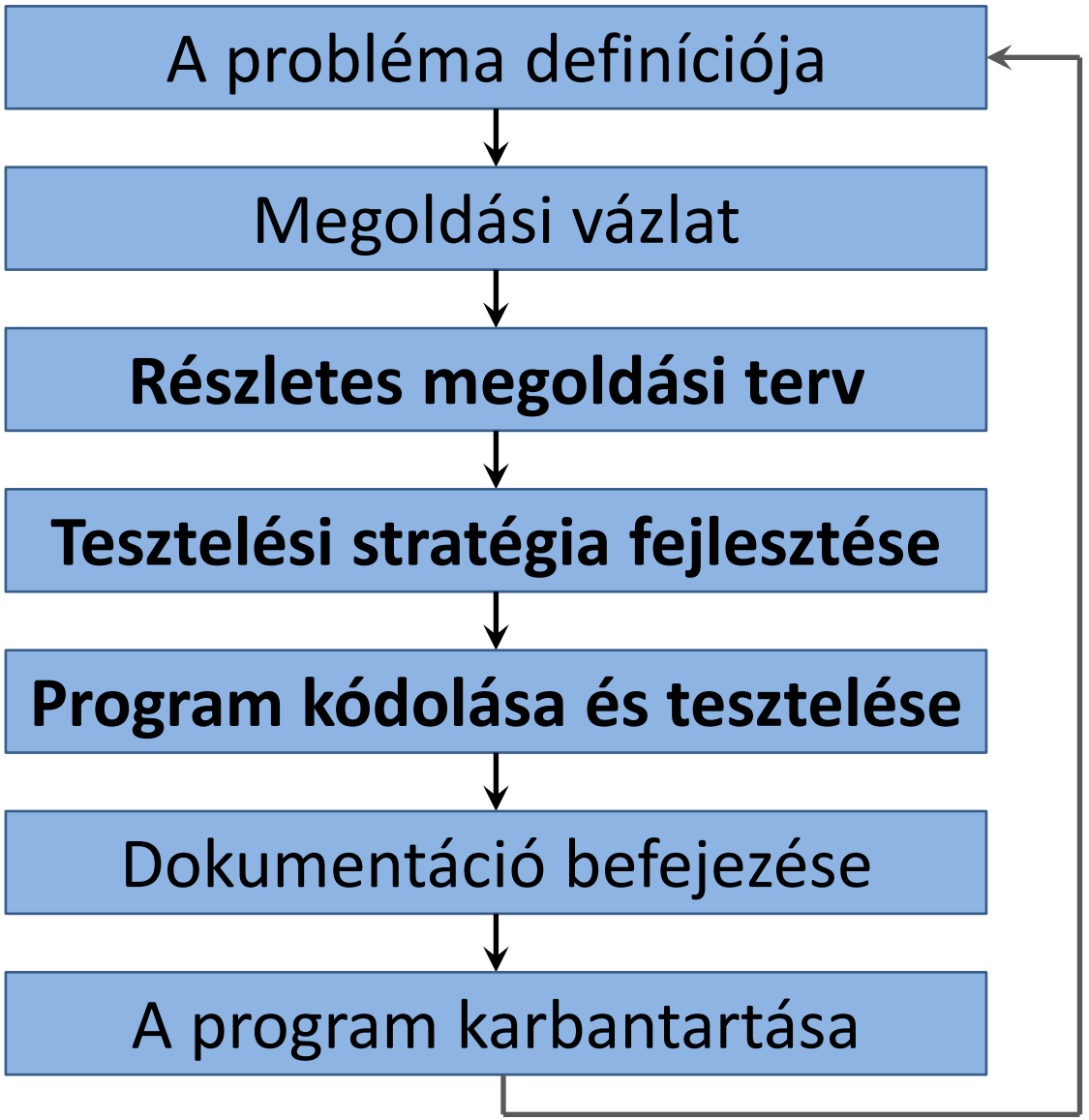
biztosan nem William Shakespeare

azonos szavak
eltérő jelentés

Számítógépes problémamegoldás

Szorosan kapcsolódva a **szoftver életciklus**ához

Számítógépes problémamegoldás



1: A probléma definíciója

- Mi az *ismeretlen* (az elvárt eredmény)? Mi az összefüggés a rendelkezésre álló információk és az ismeretlen között?
- A megadott információk elegendőek egyáltalán a probléma megoldásához?
- A probléma leírásnak precíznek, pontosnak kell lennie.
- A felhasználónak és a programozónak együtt kell működnie.
- A probléma teljes specifikációjára szükség van, az inputot és a kívánt outputot is beleértve.

2: Megoldási vázlat

- A probléma körvonalának definiálása.
- Az eredeti probléma több részproblémára osztása.
- A részproblémák legyenek kisebbek és könnyebben megoldhatóak.
- Ezek megoldásai a végső megoldás komponensei lesznek.
- "Oszd meg és uralkodj!"

3: Részletes megoldási terv

- Az előző lépésben nincs megmondva, hogyan kell a részfeladatokat végrehajtani.
- Finomítás szükséges a részletek megadásával.
- Kerülni kell a félreérthetőséget.
- A pontos módszer tartalmazza a jól meghatározott lépések sorozatát, amit **algoritmus**nak hívunk.
- Különböző formalizmusokkal írható le.

4: Tesztelési stratégia fejlesztése

- Ki kell próbálni az algoritmust több különböző inputkombinációval, hogy biztosak legyünk, hogy jó eredményt ad minden esetben.
- A bemeneti adatok különböző kombinációit **teszteset**eknek nevezzük.
- Ez nemcsak normál adatokat foglal magába, hanem extrém bemeneteket is, hogy teszteljük a határokat.
- Komplett tesztesetekkel ellenőrizhetjük magát az algoritmust.

5: Program kódolása és tesztelése

- Az előző szinteken leírt algoritmus nem hajtható végre közvetlenül a számítógép által.
- Át kell alakítani egy konkrét **programnyelv**re.
- A kódolás közben/után tesztelni kell a programot a kidolgozott tesztelési stratégia szerint.
- Ha hibára derül fény, akkor megfelelő átdolgozásra és újratesztelésre van szükség, amíg a program minden körülmények között jó eredményt nem ad.
- A kódolás és tesztelés folyamatát **implementáció**nak hívjuk.

6: Dokumentáció befejezése

- A dokumentáció már az első lépésnél elkezdődik és a program egész élettartamán át tart.
- Tartalmazza:
  - az összes lépés magyarázatát,
  - a meghozott strukturális döntéseket,
  - a felmerült problémákat,
  - az alkalmazott algoritmusok leírását,
  - a program szövegét és annak magyarázatát,
  - felhasználói kézikönyvet,
  - stb.

7: A program karbantartása

- A program nem megy tönkre.
- Néha viszont hibázhat, összeomolhat, elbukhat.
- A programhibák oka, hogy sosem volt tesztelve az adott körülmények között.
- Az újonnan felfedezett hibákat ki kell küszöbölni (átadás után is).
- Néha a felhasználóknak új igényei, elvárásai vannak a programmal szemben.
- Ennek megfelelően átalakítás, bővítés lehet szükséges.
- Ezek után frissíteni kell a dokumentációt is.

Részletes megoldási terv

Az algoritmus

Algoritmus

Jól ismert, megengedett lépésekből, tevékenységekből álló véges utasítássorozat, amelyet követve mindig egyértelműen elérjük a kívánt célt.

Tevékenységek precíz definíciója, amelyeknek a végrehajtása megadja a megoldási vázlat, minden egyes részfeladatának a megoldását.

Néhány jellemző:

- pontos, félreérthetetlen,
- minden eshetőségre felkészített,
- a tevékenységek sorozata véges,
- elérhető vele a cél, megadja a megoldást,
- hatékony, elegáns, egyszerű, …

Algoritmusok leírása

- **Természetes (beszélt) emberi nyelv**
- **Mondatszerű leírás**
- **Folyamatábra**
- **Pszeudokód**
- Struktogram
- Grafikus
- Algebraszerű
- Adat-folyam diagram
- Hierarchikus
- Táblázatos
- Programnyelv

Példa

y = sign(x) függvény

- Mi az?
- Mit jelent?
- Mi az eredmény?
- Hogyan működik?
- Hogyan tudjuk meghatározni az értékét?
- Ha x egyenlő -4, mi az y értéke?
- …

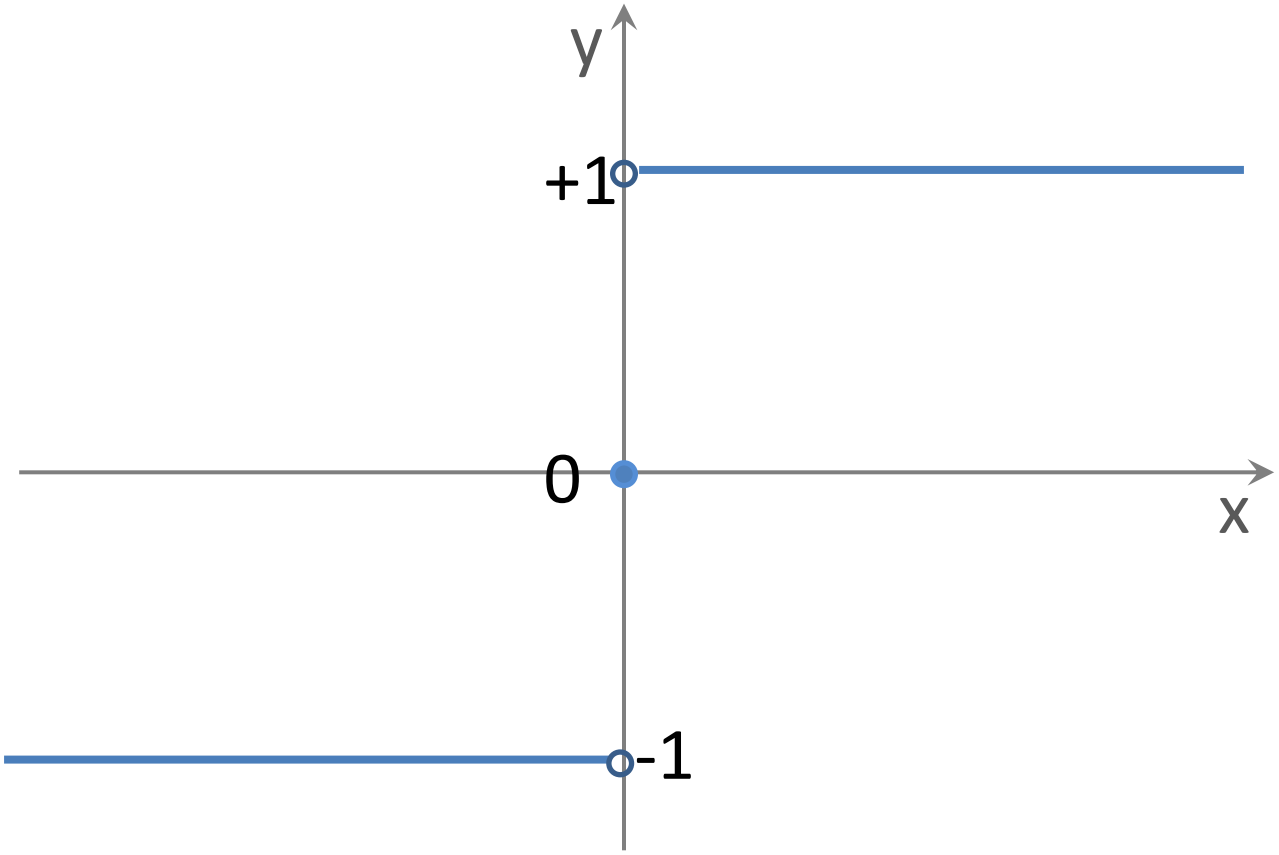
y = sign(x)

Mondatszerű reprezentáció:

1. Ha x egyenlő 0, y értéke legyen 0!
2. Különben ha x nagyobb, mint 0, y értéke legyen +1!
3. Egyébként ha x kisebb, mint 0, akkor a függvény adjon -1 értéket!

y = sign(x)

Grafikus reprezentáció:



y = sign(x)

Algebraszerű reprezentáció:

$x \in \mathfrak{R}$

$y \in \{-1, 0, +1\}$

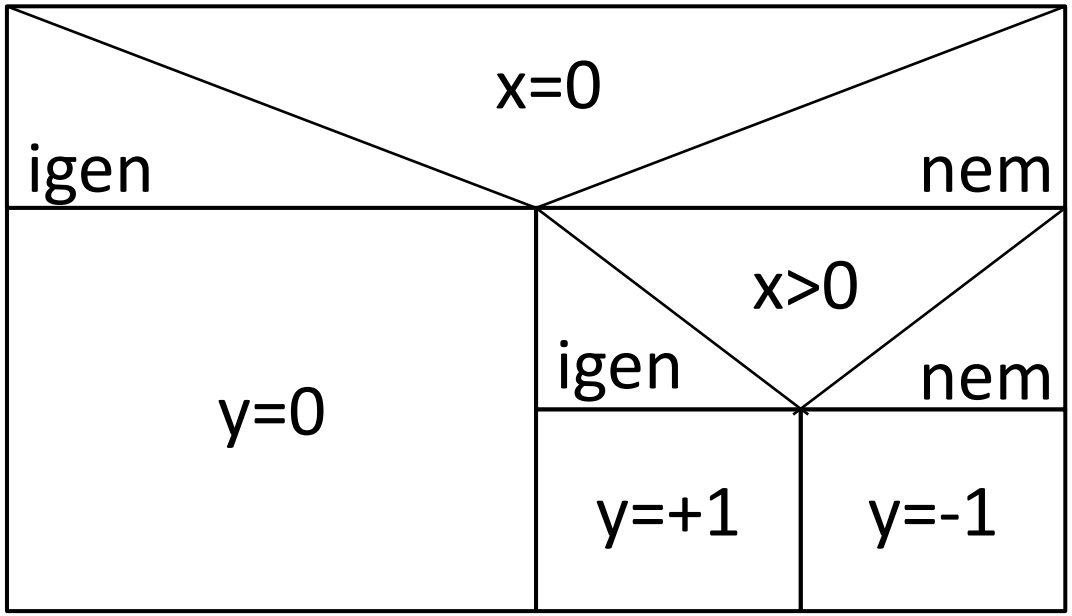
$\forall x,\ x>0 \Rightarrow y=+1$

$\forall x,\ x<0 \Rightarrow y=-1$

$x=0 \Rightarrow y=0$

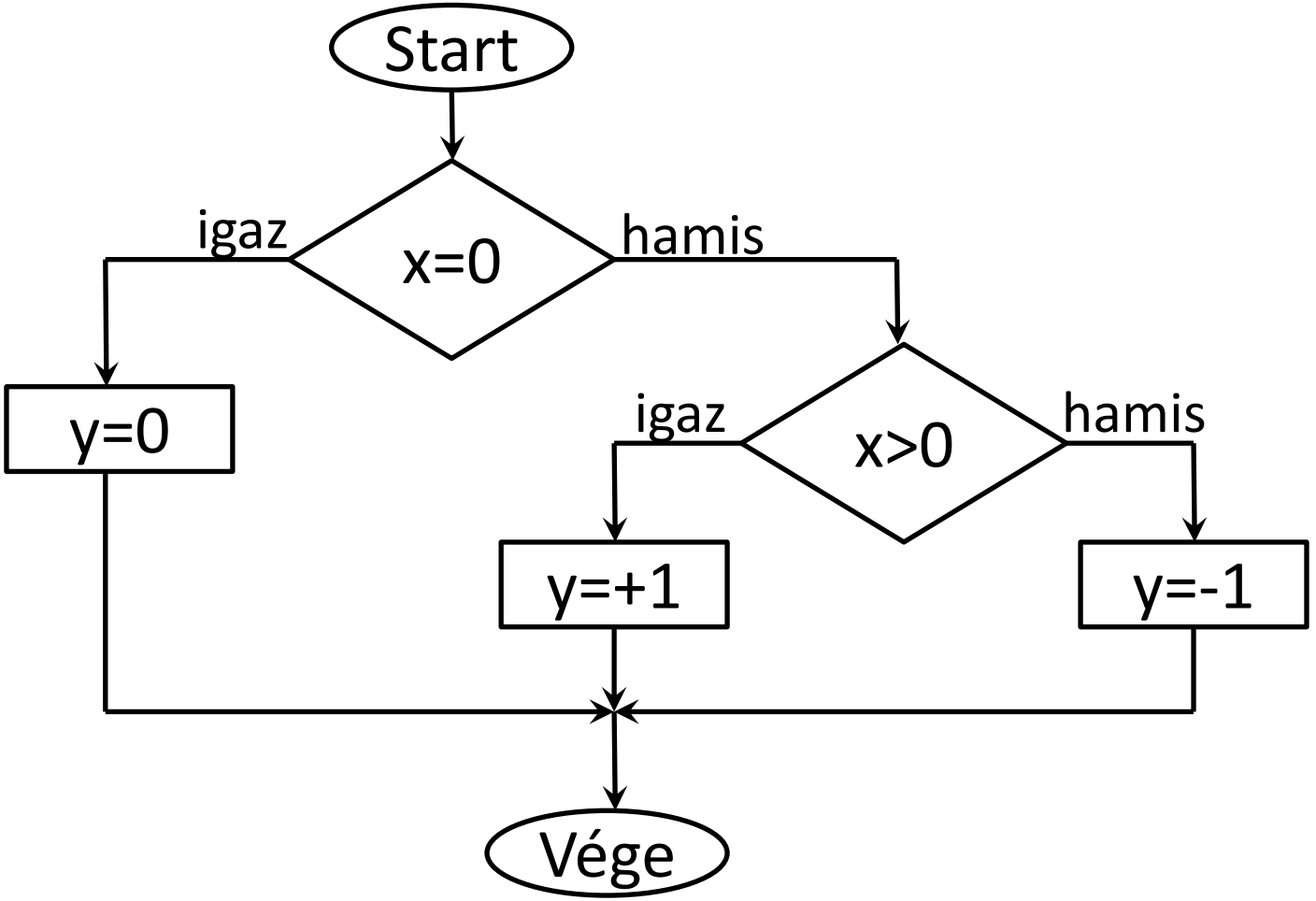
y = sign(x)

Struktogramos reprezentáció:



y = sign(x)

Folyamatábrás reprezentáció:



y = sign(x)

Pszeudokódos reprezentáció:

```
if x==0 then
  y=0
else
  if x>0 then
     y=+1
  else
     y=-1
  endif
endif
```

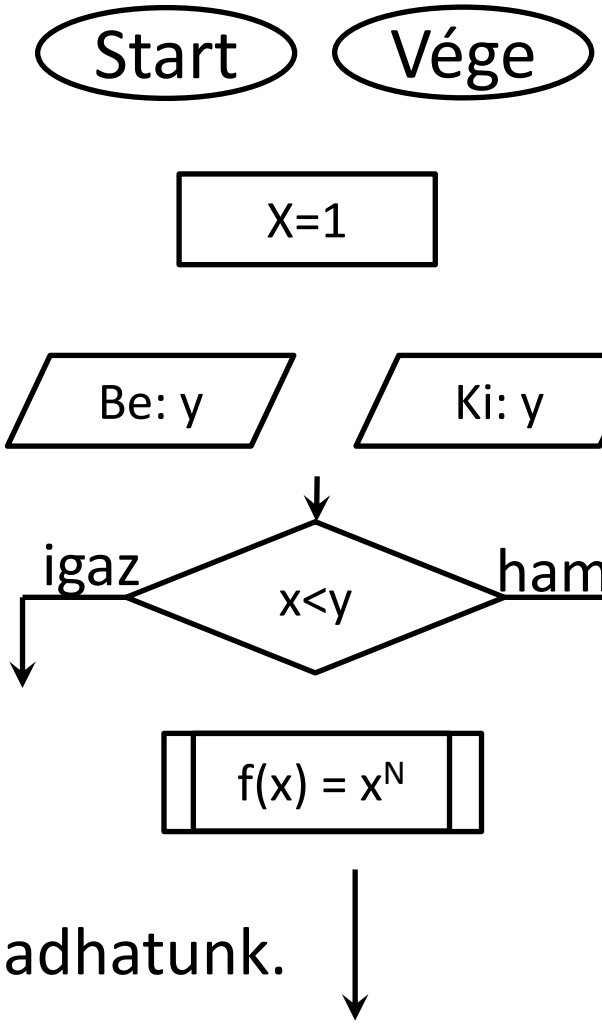
y = sign(x)

Programozási nyelven történő reprezentáció
(például Python nyelven):

```
if x == 0:
    y = 0
else:
    if x>0:
        y = +1
    else:
        y = -1
```

A folyamatábra elemei

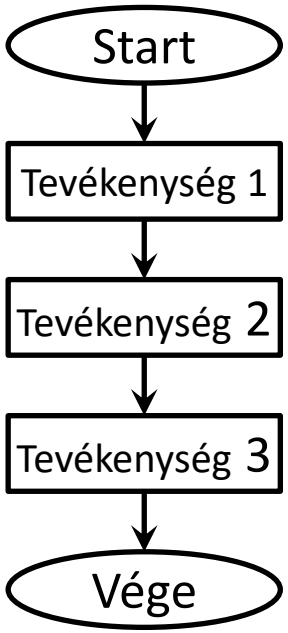
- Kiinduló- és végállapot
- Elemi utasítás
- Input/output
- Feltétel
- Beágyazás
- Csak a nyilak mentén haladhatunk.



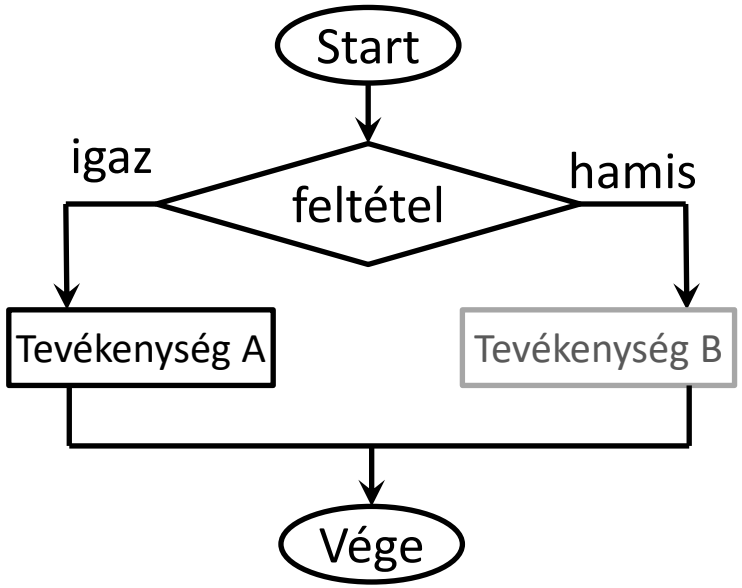
Az algoritmus alap struktúrái

folyamatábrával

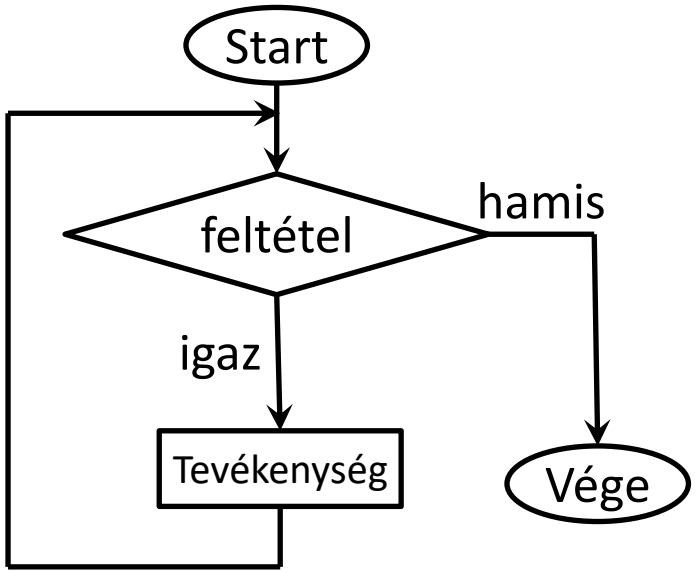
Szekvencia



Elágazás



Ismétlés



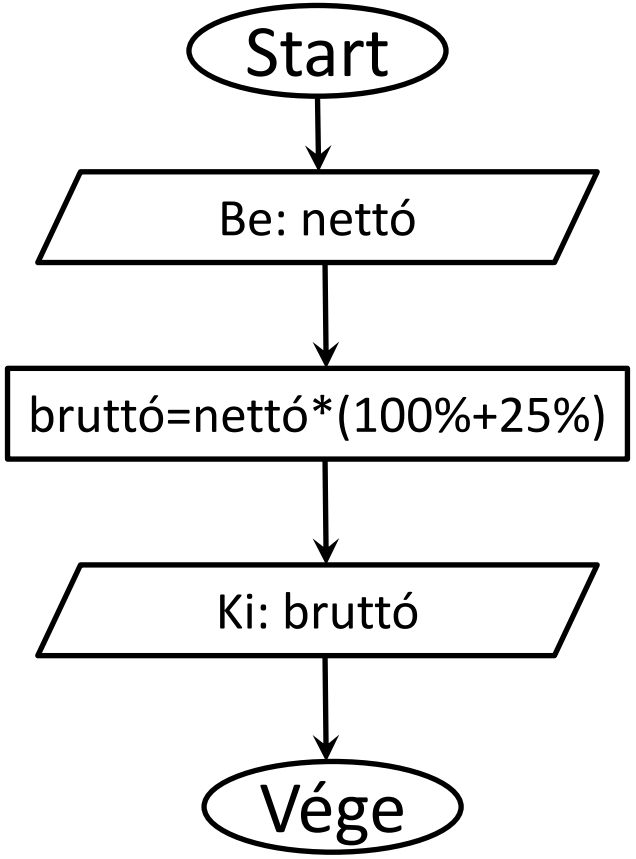
Algoritmusok módosítása

Az algoritmusokat gyakran módosítani kell, hogy jobbak legyenek.

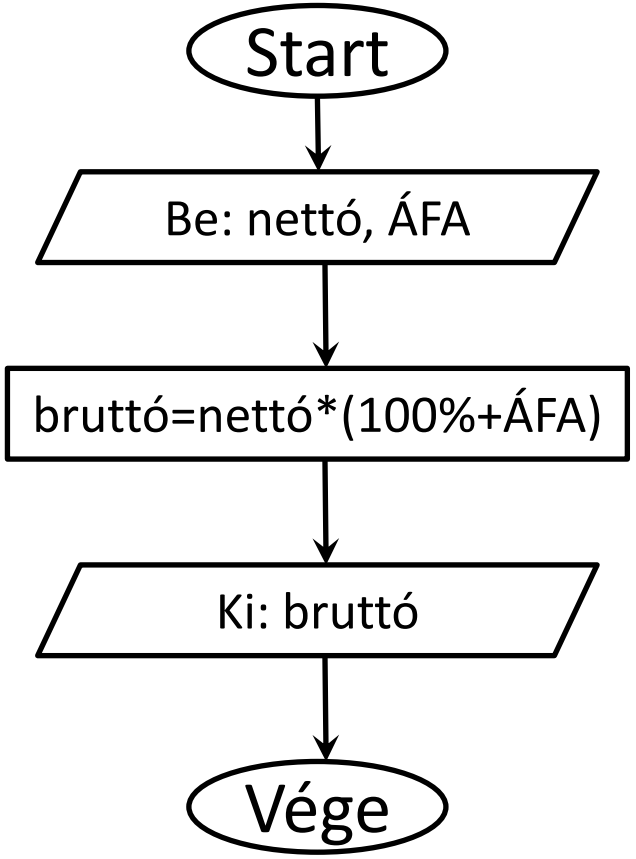
- Általánosítás:
  több esetre is alkalmazható legyen
- Kiterjesztés:
  esetek újfajta körére is alkalmazható legyen
- Beágyazás:
  egy algoritmus újrahasznosítása egy másikon belül
- ‘Bolondbiztossá’ tétel:
  megbízhatóvá, robosztussá, hibaelkerülővé tétel

Algoritmus általánosítása

Eredeti:

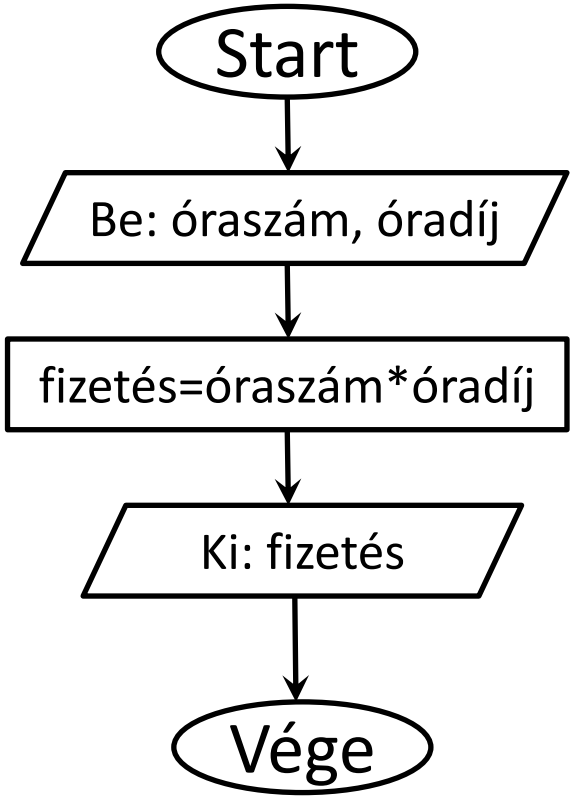


Általánosított:

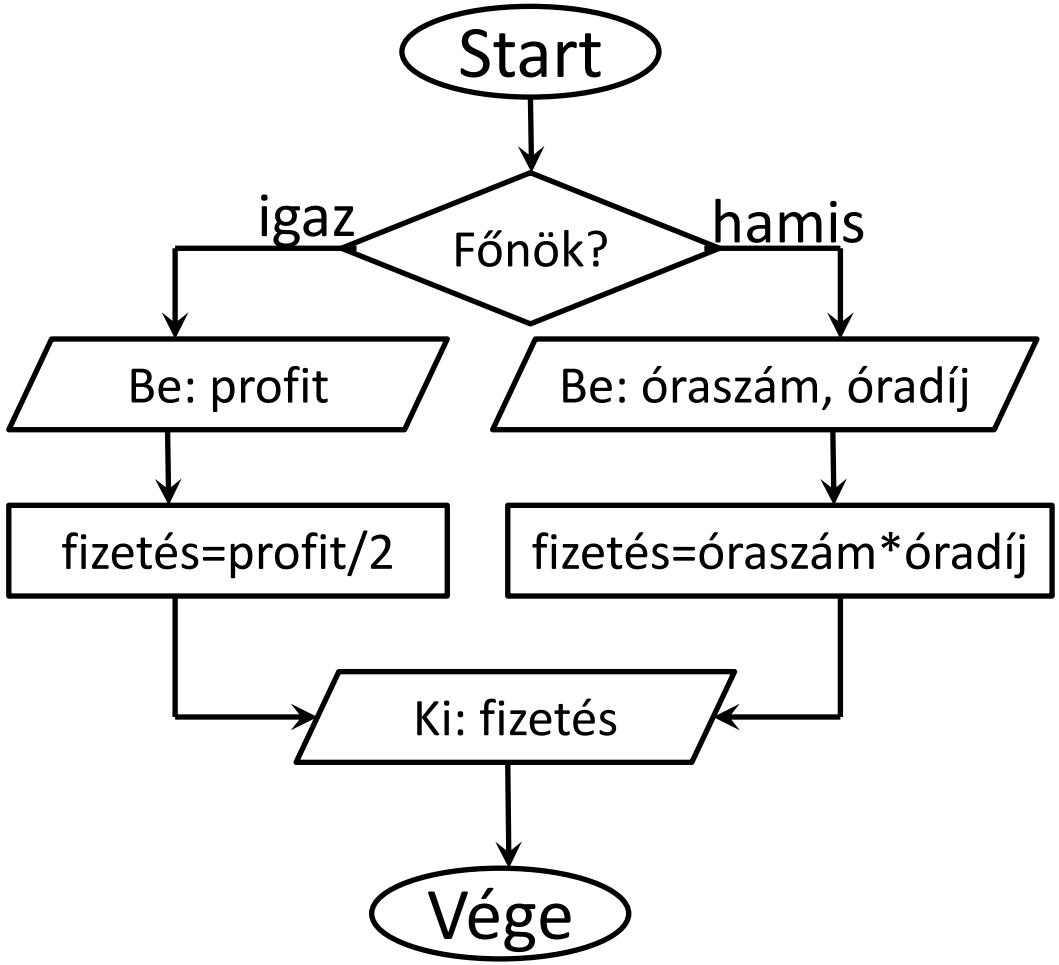


Algoritmus kiterjesztése

Eredeti:

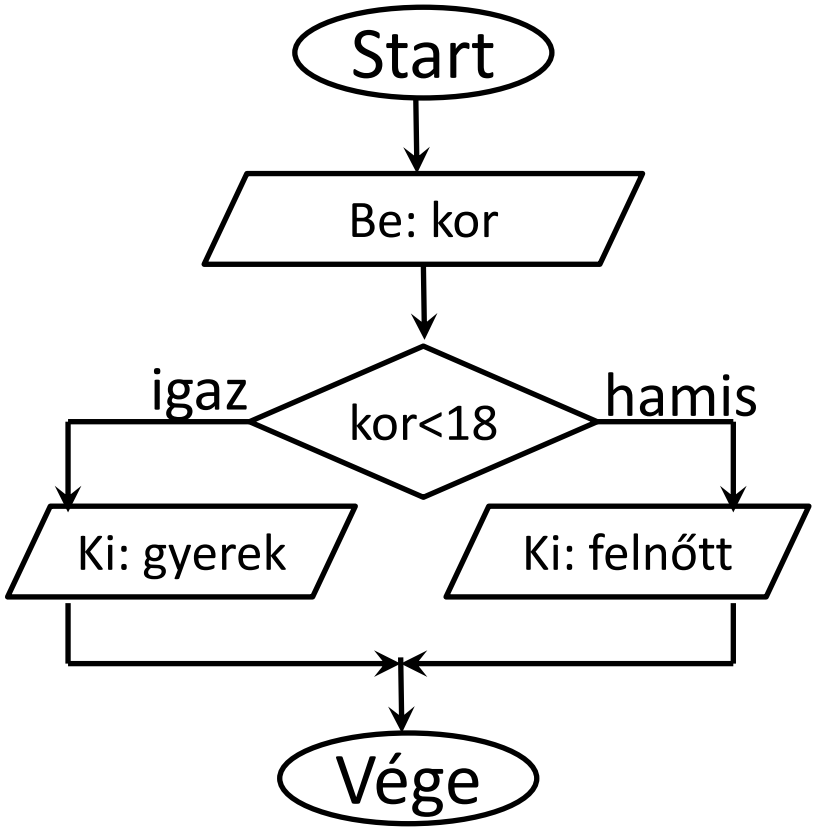


Kiterjesztett:

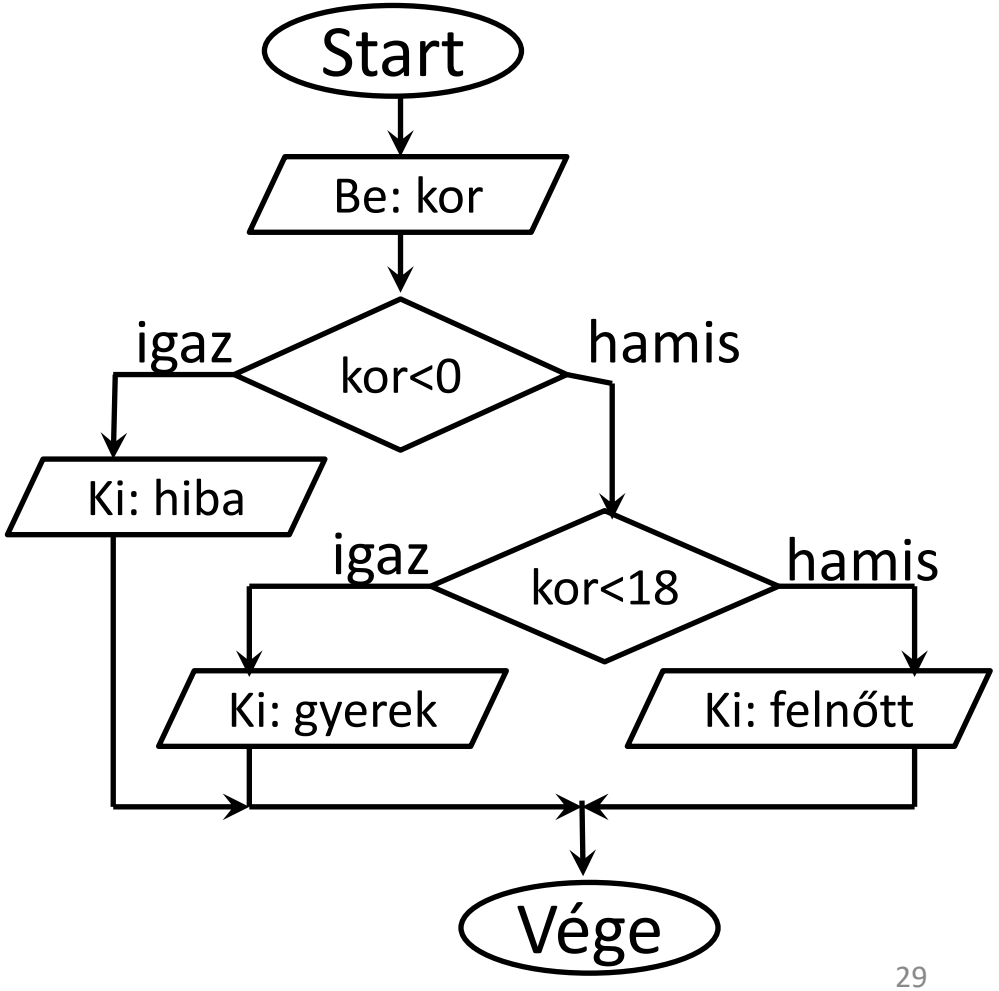


Algoritmus ‘bolondbiztossá’ tétele

Eredeti:

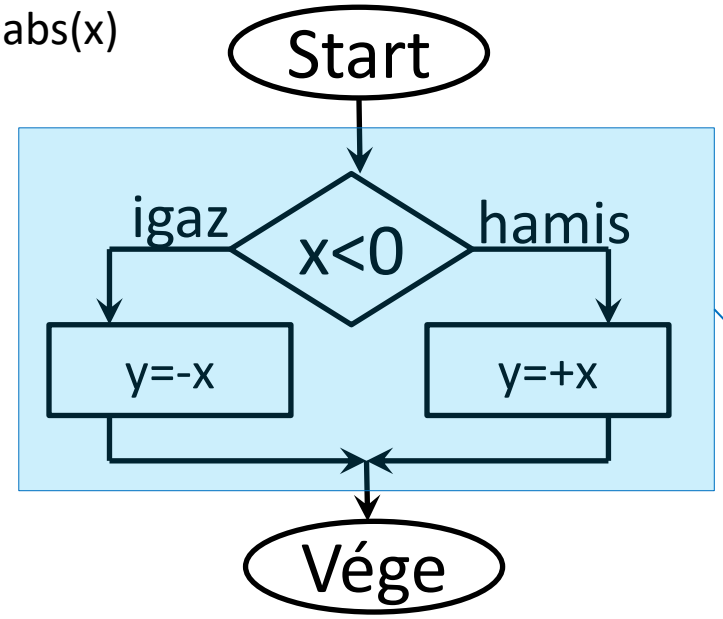


Bolondbiztos:

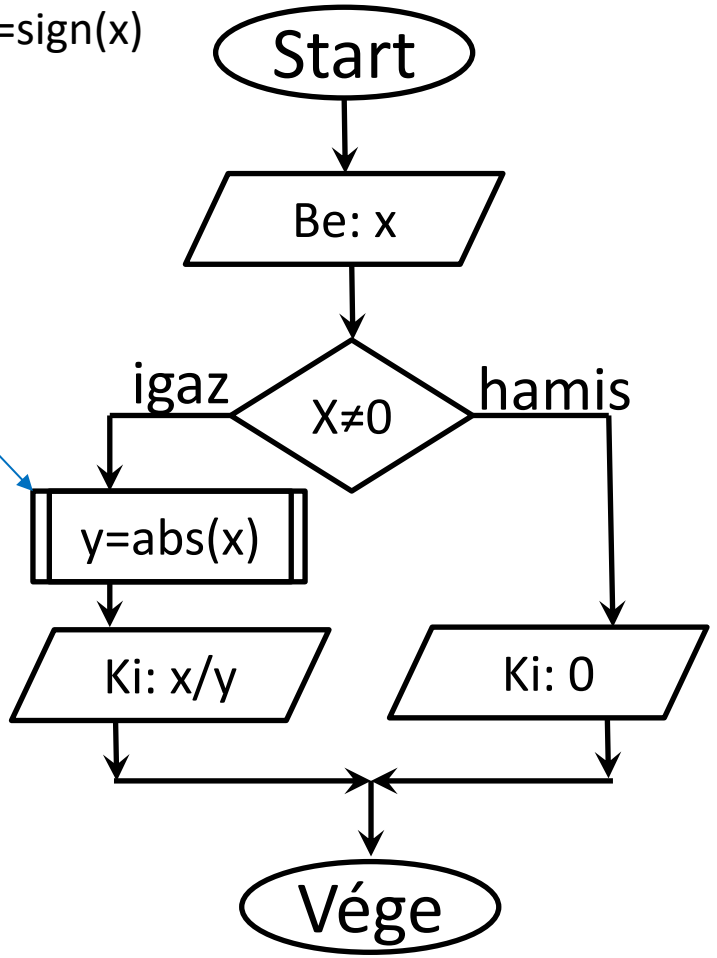


Algoritmus beágyazása

Eredeti:
y=abs(x)



Beágyazott:
y=sign(x)



Alternatív algoritmus

Gyakran többféleképpen is elérhetjük ugyanazt a célt.

Az algoritmusoknak különböző lehet a szerkezete, de azonos lehet a viselkedése.

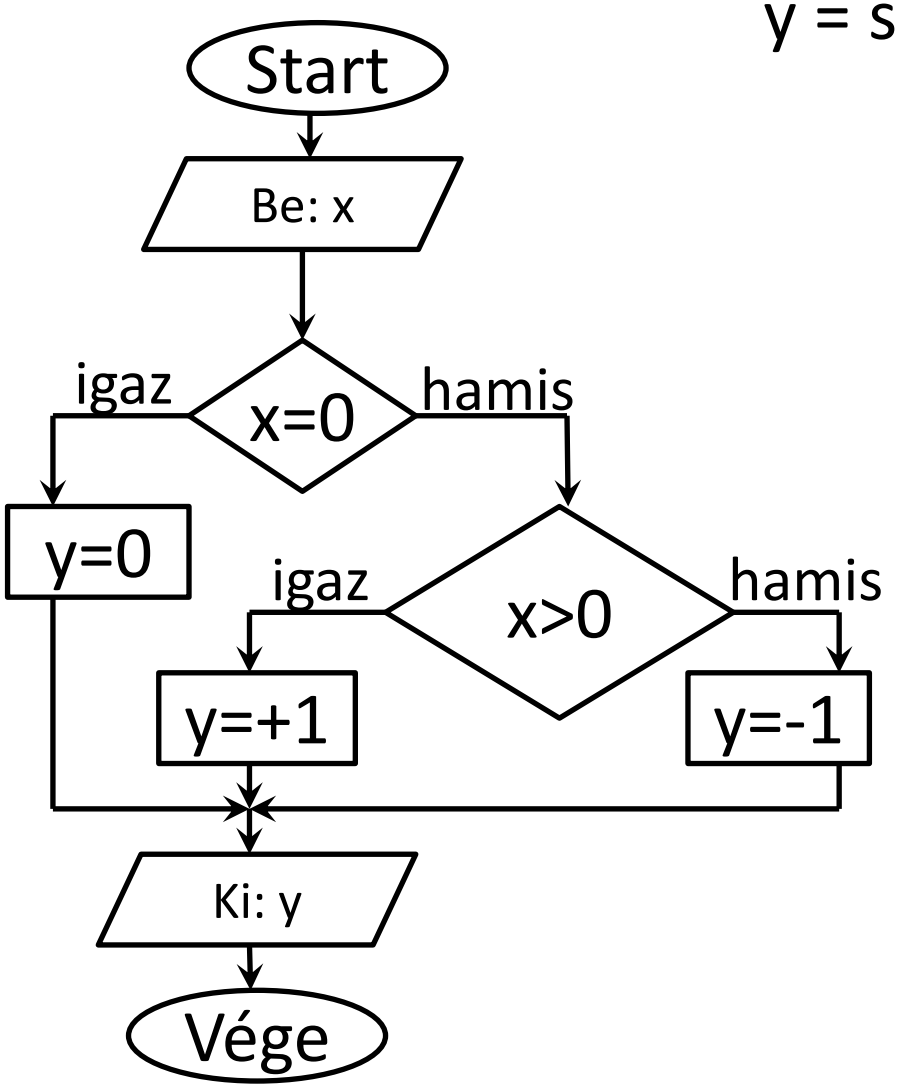
Ez azt jelenti, hogy azonos bemeneti adatokra azonos eredményeket adnak, de ezt máshogy érik el.

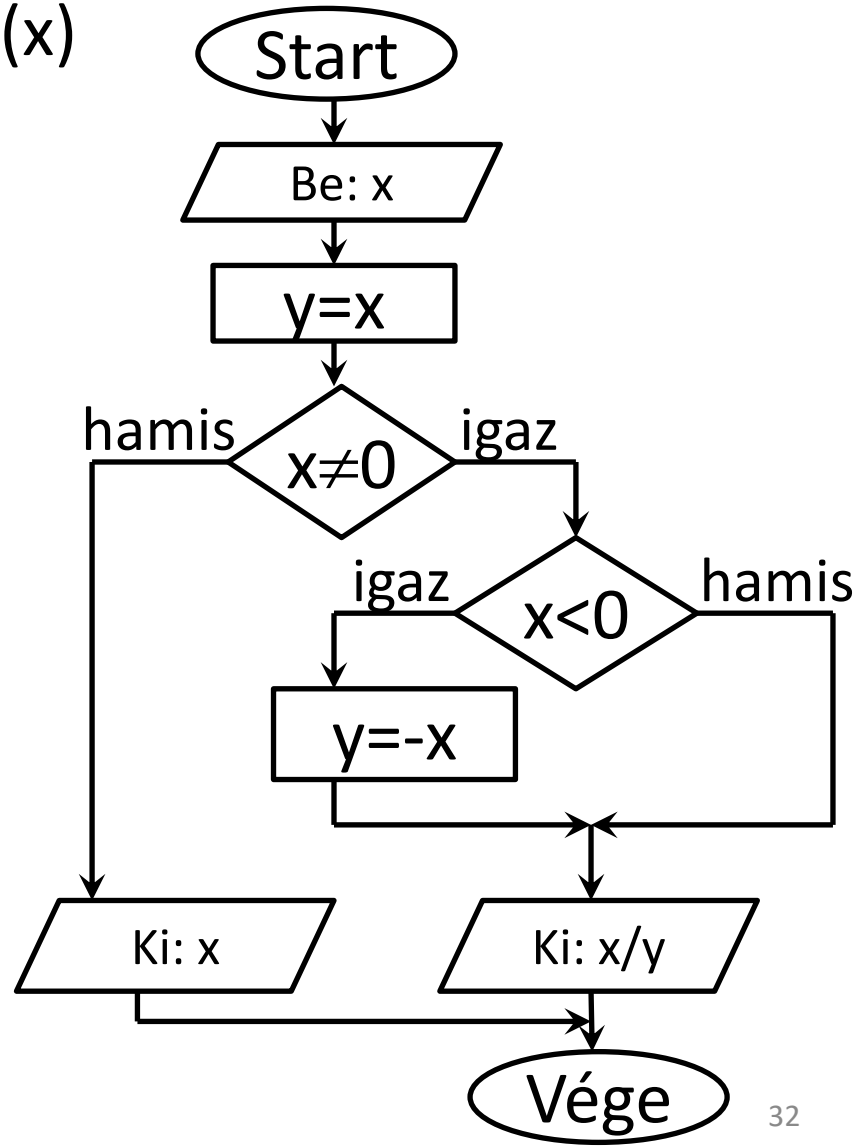
Néha hasznos az egyik algoritmust előnyben részesíteni, néha viszont mindegy, melyiket használjuk.

Olykor az egyik algoritmus határozottan egyszerűbb, kisebb, gyorsabb, megbízhatóbb lehet, mint a többi.

Alternatív algoritmus

y = sign(x)





Az algoritmusok tulajdonságai

- **Teljes**:
  Minden lehetséges esetre/részletre tekintettel pontosan megadott
- **Félreérthetetlen**:
  csak egyféleképpen értelmezhető tevékenységek
- **Determinisztikus**:
  az utasításokat követve mindig biztosan elérhető a cél, a megoldás
- **Véges**:
  korlátozott számú lépés után véget ér az utasítássorozat

Rossz algoritmus

Hogyan jussunk el a 2.-ról az 5. emeletre lifttel?

1. Nyomd meg a lift hívógombját!
2. Szállj be!
3. Nyomd meg az ‘5’ gombot!
4. Várj!
5. Ha az ajtó kinyílik, szállj ki!

Probléma (nem teljes):

- Mi van, ha az érkező lift lefelé megy?
- Mi van, ha valaki miatt megáll a lift a 3. emeleten is?

Rossz algoritmus

Hogyan készítsünk sült csirkét?

1. Tedd be a csirkét a sütőbe!
2. Állítsd be a hőmérsékletet!
3. Várj, amíg kész lesz!
4. Szolgáld fel!

Problémák (félreérthetőség):

- Mi a megfelelő hőmérséklet (50°C vagy 200°C)?
- Fagyasztott vagy élő csirke?
- Honnan tudjuk, hogy "kész" van?

Rossz algoritmus

Hogyan legyünk milliomosok?

1. Vegyél egy lottót!
2. Húzd le a kívánt számokat!
3. Várj a nyereményre
   (vagy szomorkodj)!

Problémák (véletlenszerű, nem determinisztikus):

- Az esetek többségében nem leszünk milliomosok.
- Csak néha működik, pedig mindig ugyanazt csináljuk.

Rossz algoritmus

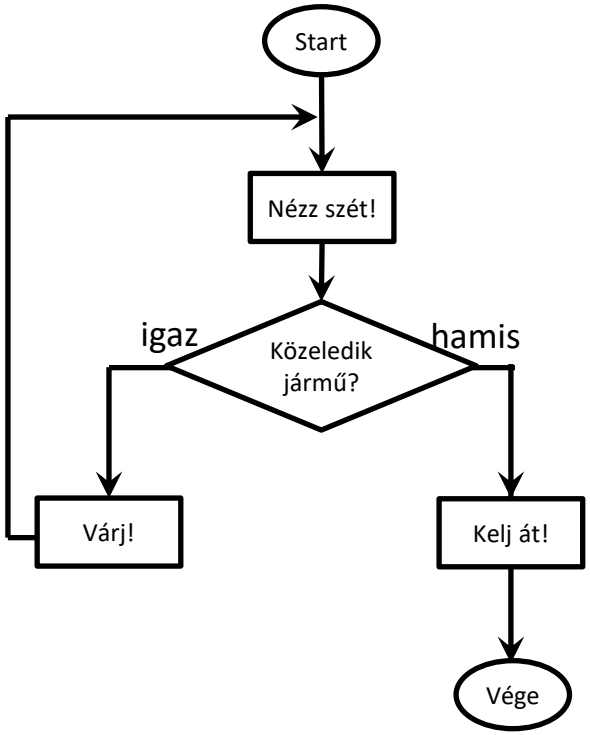
Hogyan buszozzunk?

1. Várj a buszra!
2. Szállj fel!
3. Vegyél jegyet!
4. Ülj le!
5. Ha megérkeztél, szállj le!

Problémák (végtelen):

- Ha nem megállóban vársz, a busz sosem áll meg.
- Ha rossz buszra szálltunk, sosem szállhatunk le róla.

Gyalogos átkelés egyenes úttesten



Problémák:

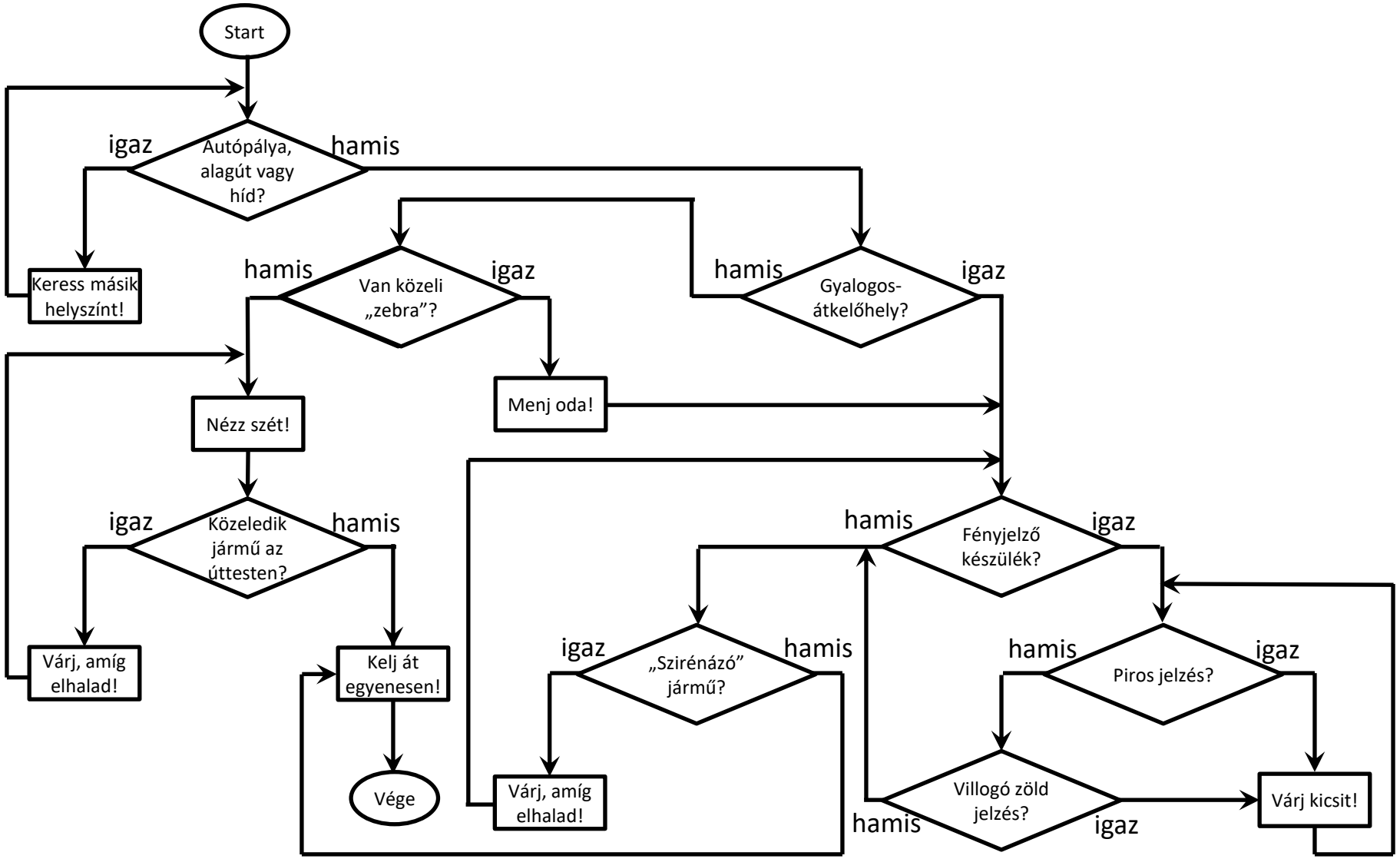
- nem teljes
- félreérthető
- …

Módosítás:

- általánosítás
- kiterjesztés
- ‘bolondbiztosá’ tétel
- teljessé tétel

Pontosabb algoritmus kell

Gyalogos átkelés egyenes úttesten



Logikai műveletek és kifejezések

- Logikai műveletek

ÉS	Hamis	Igaz
Hamis	Hamis	Hamis
Igaz	Hamis	Igaz

VAGY	Hamis	Igaz
Hamis	Hamis	Igaz
Igaz	Igaz	Igaz

XOR	Hamis	Igaz
Hamis	Hamis	Igaz
Igaz	Igaz	Hamis

- Logikai ellentétek

Ha ez Igaz,		**akkor ez Hamis**
=	==	≠
<	<	≥
>	>	≤
≠	!=	=
≤	<=	>
≥	>=	<

Igaz	=	NEM Hamis
Hamis	=	NEM Igaz
X	=	NEM NEM X

Megengedett műveletek

+	Összeadás, pozitív előjel
-	Kivonás, negatív előjel
*	Szorzás
/	Osztás
%	Maradékos osztás (modulo)
(int)	Egészre alakítás csonkolással
[...]	Tömb indexelés
{...}	Tömb inicializálás
(...)	Precedencia, paraméter lista
=	Értékadás

==	Egyenlőség vizsgálat
!=	Nem egyenlő
<	Kisebb
>	Nagyobb
<=	Kisebb vagy egyenlő
>=	Nagyobb vagy egyenlő
AND	Logikai ÉS művelet
OR	Logikai VAGY művelet
NOT	Tagadás (logikai NEM művelet)
,	Elválasztó listaelem között

A számkonstansok és műveletek **tízes számrendszer**ben vannak értelmezve.
Az `Hossz` egy mennyiség értékét jelentheti, míg a `"Hossz"` csak egy szöveg.

Műveletek sorrendje

I. Zárójel
   1. `( )`
II. Egy operandusú műveletek
   1. `előjel (+ -)`
   2. `NOT`
   3. `(int)`
III. Aritmetikai műveletek
   1. `* / %`
   2. `+ -`
IV. Relációs műveletek
   1. `< <= > >=`
   2. `== !=`
V. Logikai műveletek
   1. `AND`
   2. `OR`

Műveletek sorrendje

Gyakori hibák:

- Mennyi a kétszerese 3 és 4 összegének?
  ~~2*3+4~~ 2*(3+4)
- Az `x` változó értéke 5 és 10 között van-e?
  ~~5 < x < 10~~ (5 < x) AND (x < 10)
- Az `x` változó értéke egyenlő-e 5-tel vagy 10-zel?
  ~~x == 5 OR 10~~ (x == 5) OR (x == 10)
- Írd ki a `size` szót!
  ~~output size~~ output "size"

Feladat: műveletek sorrendje

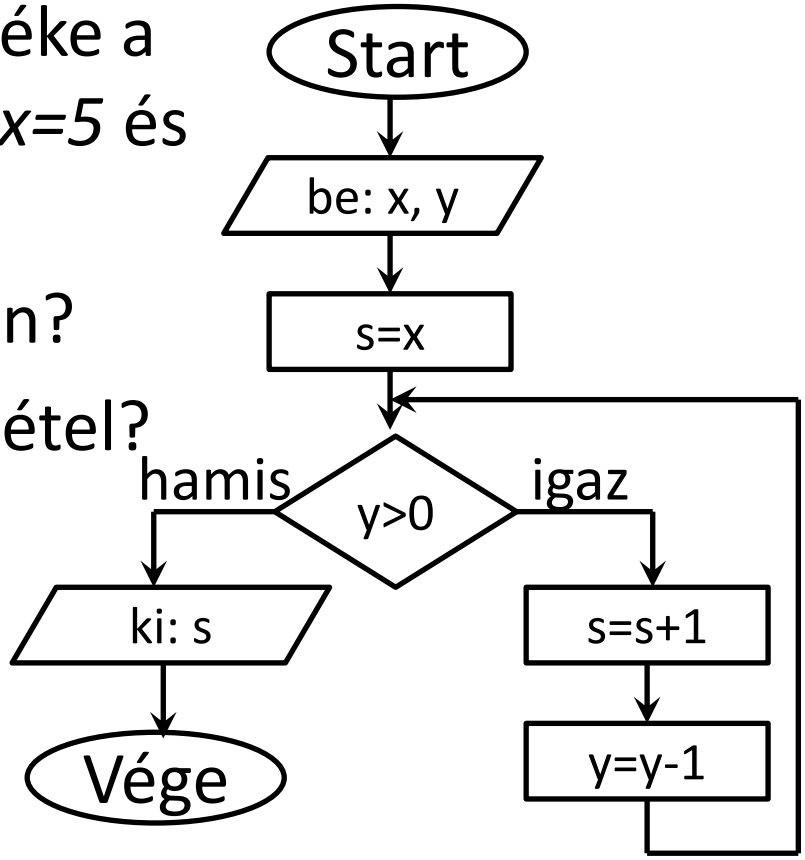
Mi az értéke az alábbi kifejezéseknek,
ha kezdetben `a=10` és `b=20`?

- `a-1/2`
- `2+b / a+1`
- `(int)a/b`
- `-a-b/-2`
- `NOT ( a <= b )`
- `a == 10 OR b > a AND a * b != 200`
- `b+-a*2!=1/2 OR (int)(a/4)==2,5`

Link a megoldásra

Feladat: folyamatábra

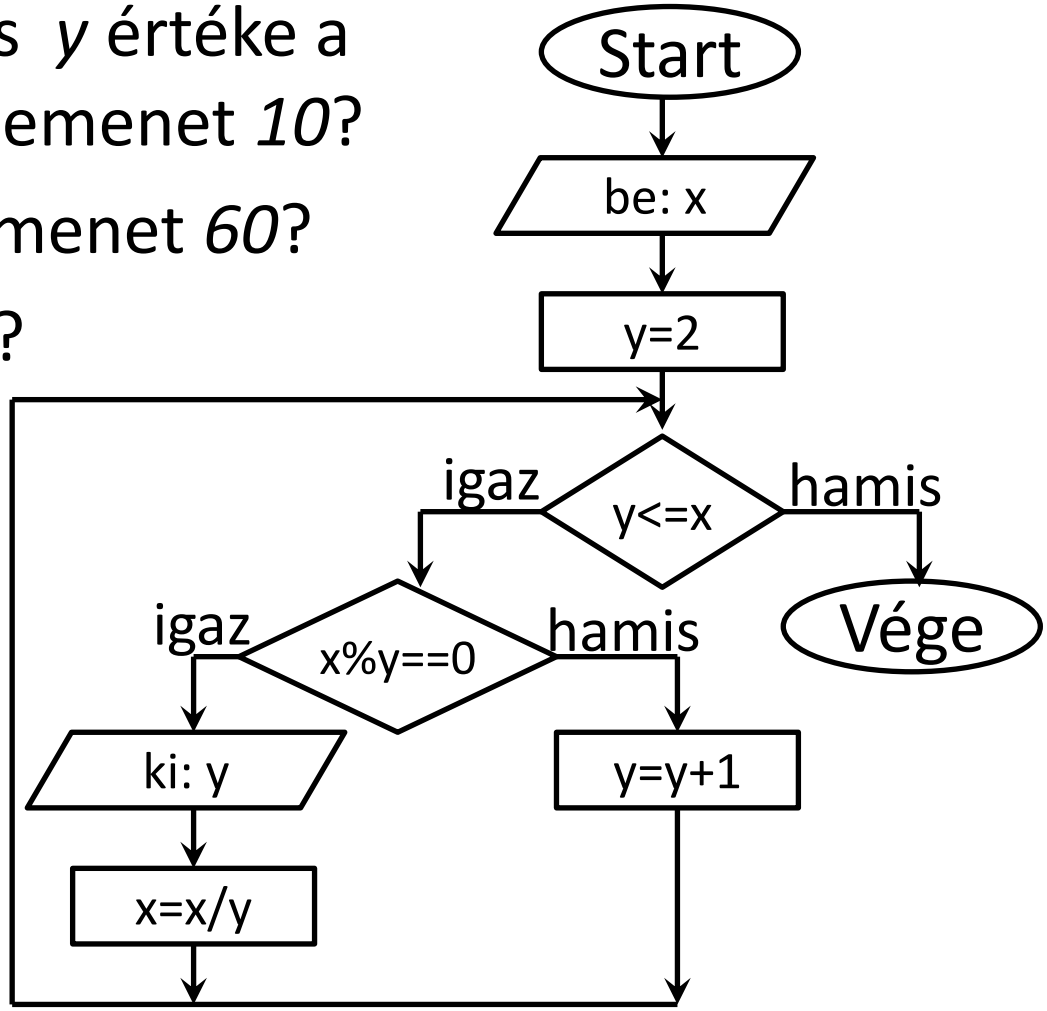
- Hogyan változik az *x, y* és *s* értéke a folyamat során, ha kezdetben *x=5* és *y=4*?
- Mi a kimenet ebben az esetben?
- Hányszor lett kiértékelve a feltétel?
- Mit csinál az algoritmus?
- Hogyan változtatnád meg az algoritmust, hogy *x* és *y* szorzatát határozza meg?



Feladat: folyamatábra

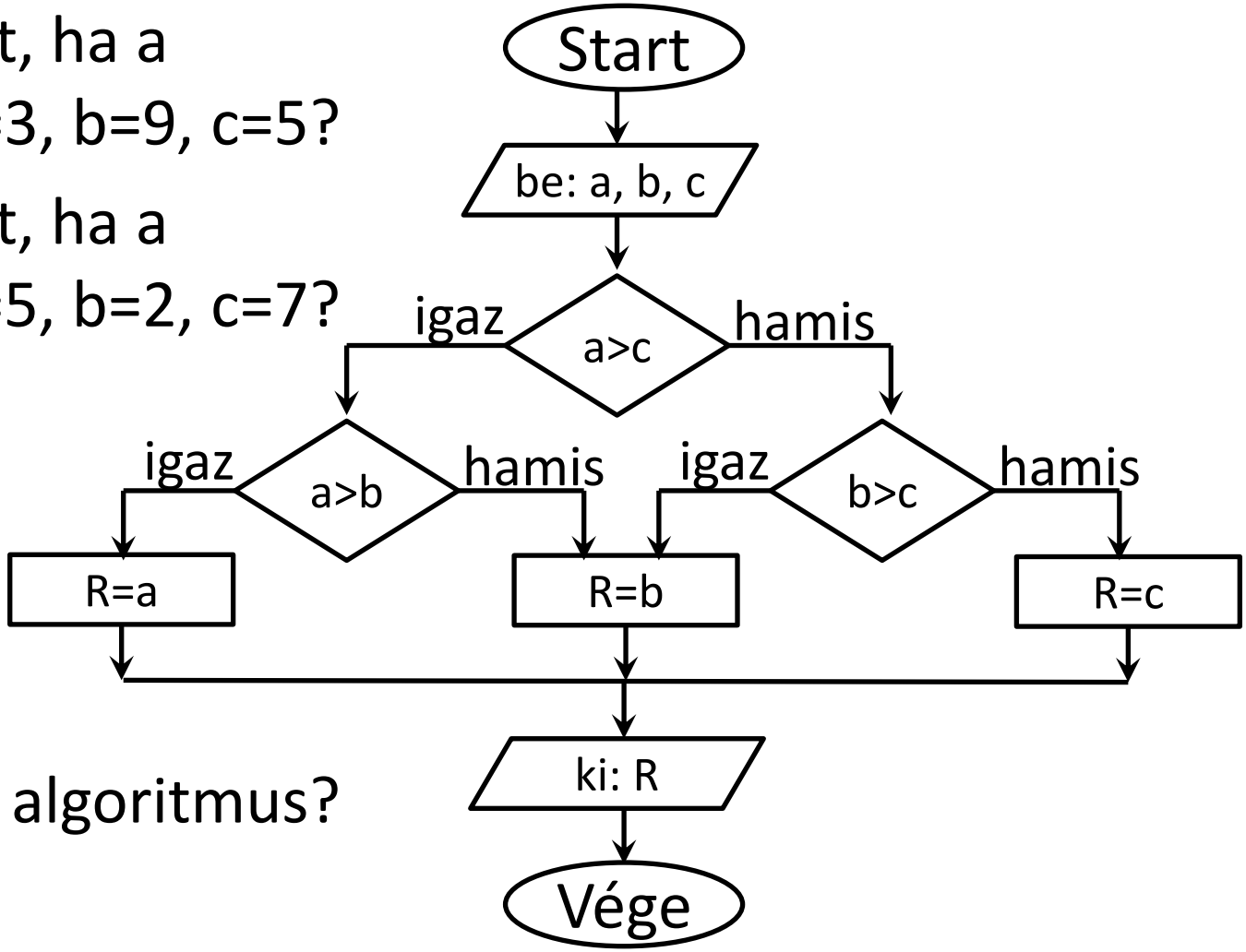
- Hogyan változik az *x* és *y* értéke a folyamat során, ha a bemenet *10*?
- Mi a kimenet, ha a bemenet *60*?
- Mit ír le az algoritmus?
- Működik, ha *x=1*?
- Ha az input *24*, hányszor ismétlődik a ciklus?

%: maradékos osztás operátora



Feladat: folyamatábra

- Mi a kimenet, ha a bemenet: a=3, b=9, c=5?
- Mi a kimenet, ha a bemenet: a=5, b=2, c=7?
- Mit csinál az algoritmus?



Feladat: folyamatábra

- Add meg azt az algoritmust, amellyel egy **a**x+**b**=0 alakú elsőfokú egyenlet megoldása megkapható!
  Például: a=0,5; b=-6 → x=12
- Add meg azt az algoritmust, amely egy évszámról megmondja, hogy az szökőév-e!
  Például: 2023 → nem; 2024 → igen
- Add meg azt az algoritmust, amely a felhasználó által megadott 3 számot kiírja csökkenő sorrendben!
- Folyamatábra segítségével határozd meg egy pozitív egész szám faktoriálisát!
- Add meg folyamatábrával, egy pozitív egész decimális számot binárissá konvertáló algoritmust!


Pszeudokód

Szekvencia:

utasítás1
utasítás2
utasítás3
…

Elágazás:

if *feltétel* then
    utasítás$_{(igaz)}$
else
    utasítás$_{(hamis)}$
endif
…

Ismétlés:

while *feltétel* do
    utasítás$_{(igaz)}$
enddo
utasítás$_{(hamis)}$
…

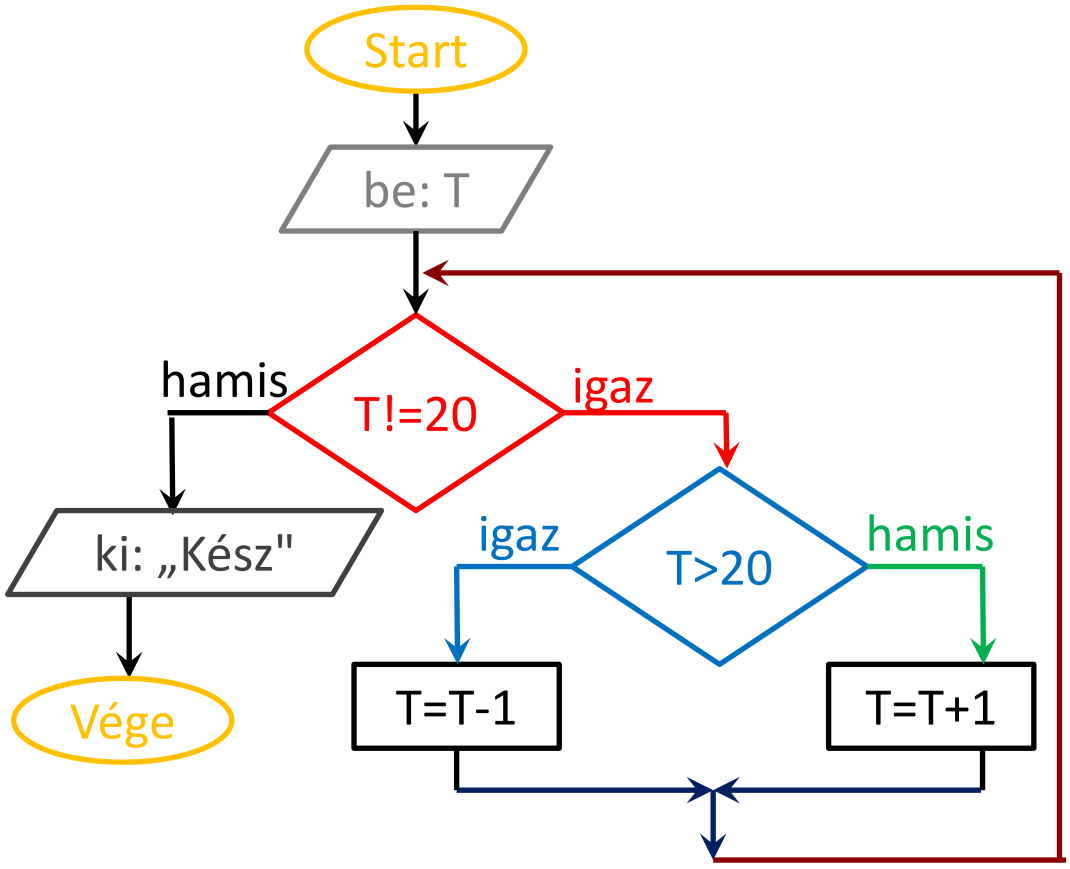
További kulcsszavak: input, output, stop, break, call function, endfunction, procedure, endprocedure, return

Konverzió

Folyamatábra ⟺ pszeudokód

- Egy feltétel tartozhat elágazáshoz és ismétléshez is
  - Ismétlés: ha van vissza nyíl (`enddo`) a feltétel elé
  - Elágazás: ha nincs vissza út a feltétel elé a két ág összekapcsolódik (`endif`)
- Az elágazás hamis (`else`) ága kihagyható
- Az ismétlés mindig csak igaz feltétel esetén történik
- Egy feltétel két ága felcserélhető a feltétel tagadásával
  - Egyes ismétlések és elágazások megadásához szükséges lehet

Konverzió




```
input T
while T!=20 do
   if T>20 then
      T=T-1
   else
      T=T+1
   endif
enddo
output "Kész"
```

Behúzás (indentálás)

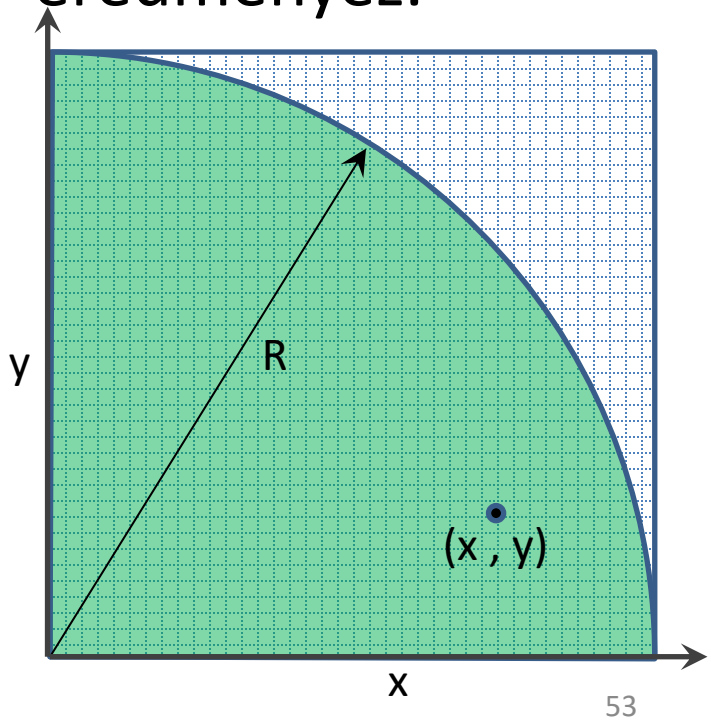
- Egy szekvencia sorai azonos mértékű behúzással írandók (azaz azonos helyen kezdődnek)
- A `while` sorhoz képest egy ciklus magja behúzandók
  - A `do` és az `enddo` között
- Az `if` sorhoz képest egy elágazás ágai behúzandók
  - A `then` és `else` között, valamint az `else` és az `endif` között
  - A `then` és az `endif` között (ha nincs `else` ág)
- Egymásba ágyazott szerkezeteknél többszörös behúzás

Pszeudokód példa

```
input R
i=0
x=0
while x<=R do
   y=0
   while y<=R do
      if x*x+y*y<=R*R then
         i=i+1
      endif
      y=y+1
   enddo
   x=x+1
enddo
output 4*i/((R+1)*(R+1))
```

A π értékének közelítése

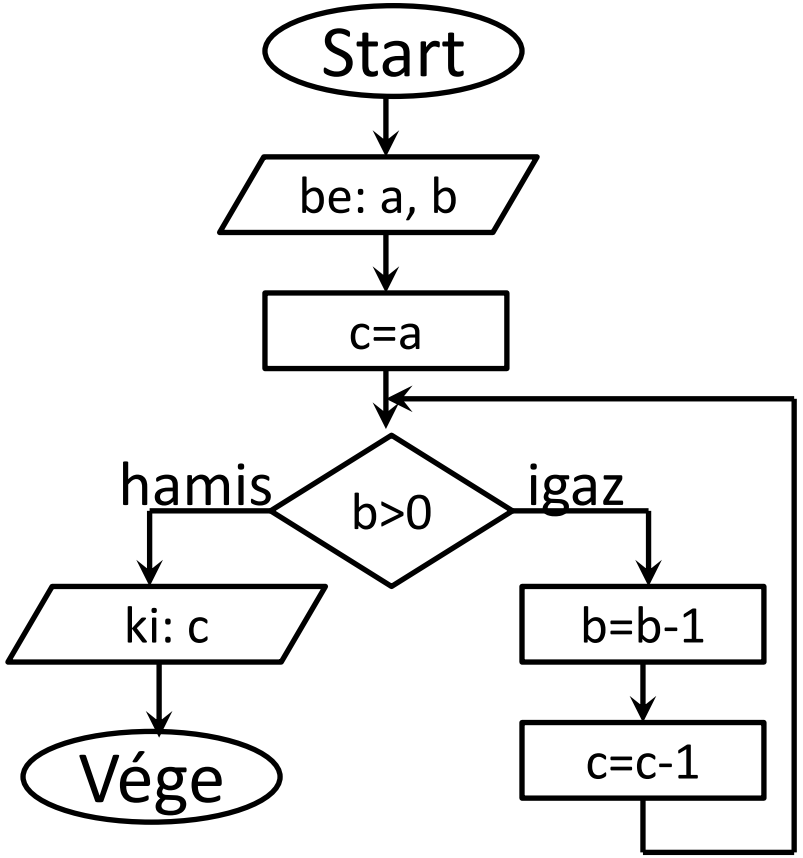
Nagyobb R érték pontosabb közelítést eredményez.



Feladat: Azonosak?

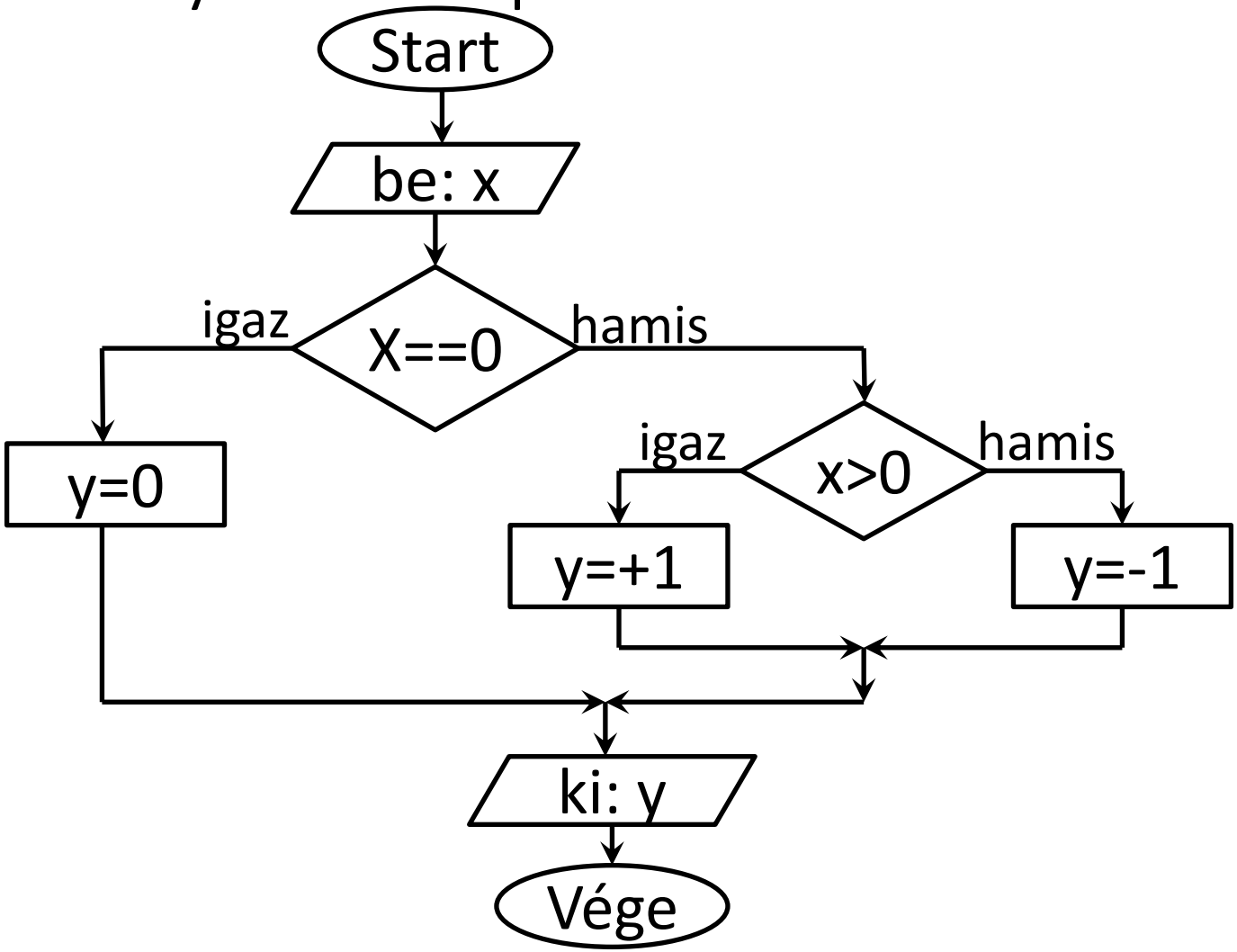
```
input a
input b
c=a
if b>0 then
   b=b-1
   c=c-1
else
  output c
endif
```

- A folyamatábra és a pszeudokód ugyanazt az algoritmust írják le?



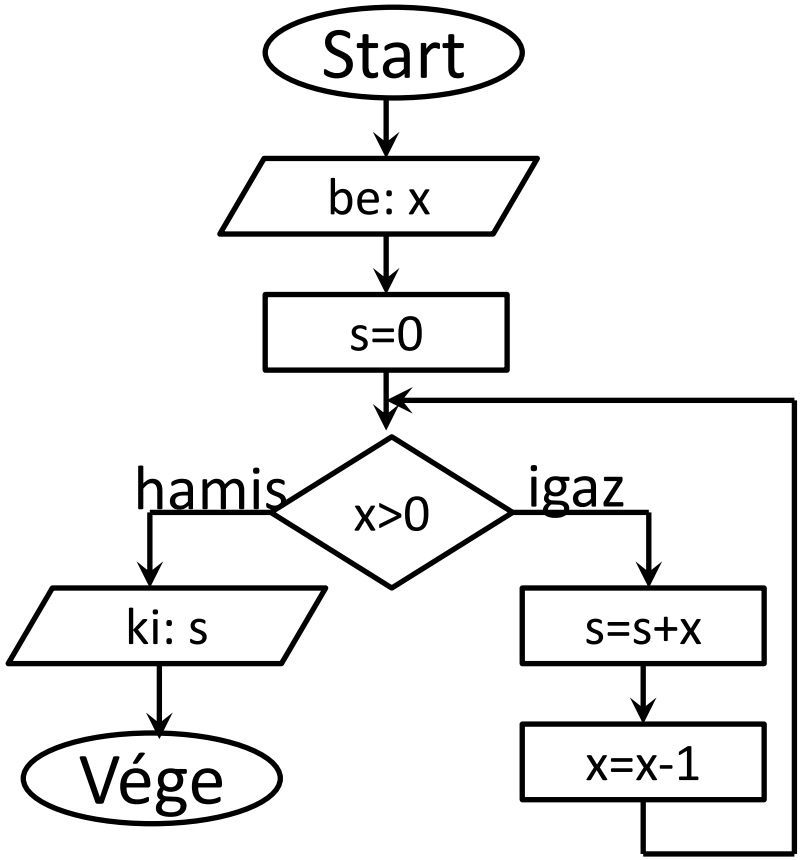
Feladat: konverzió 1

- Írd le a folyamatábrát pszeudokóddal!



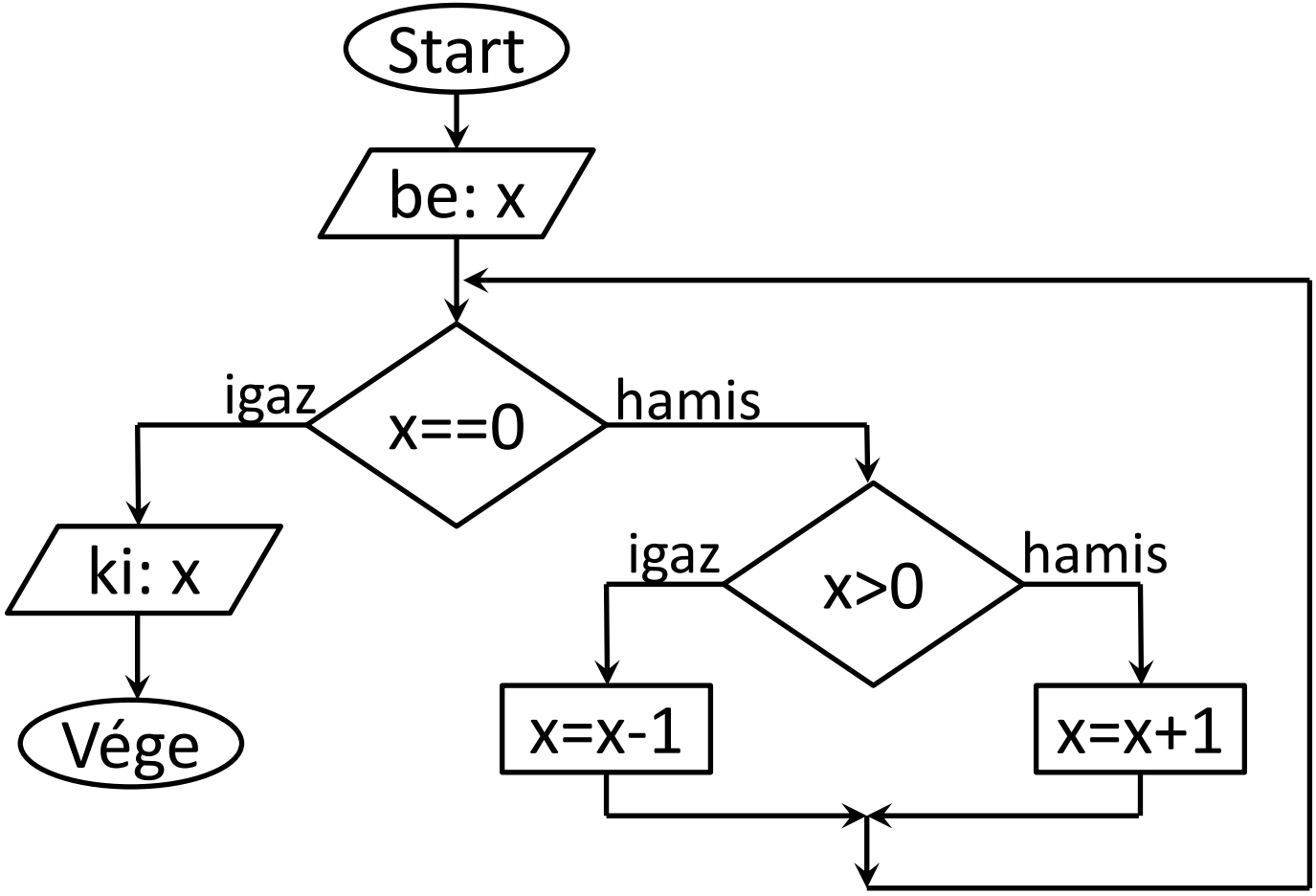
Feladat: konverzió 2

- Írd le a folyamatábrát pszeudokóddal!



Feladat: konverzió 3

- Írd le a folyamatábrát pszeudokóddal!



Feladat: pszeudokód 1

```
input a
if a<0 then
   b=-1*a
else
   b=a
endif
output b
```

- Mi a kimenet, ha *a=10*?
- Mi a kimenet, ha *a=-4*?
- Mit csinál az algoritmus?

- Mit csinál ez az algoritmus?

```
input a
if a<0 then
   a=-1*a
endif
output a
```

Feladat: pszeudokód 2

```
input a
input b
c=a
while b>0 do
    b=b-1
    c=c-1
enddo
output c
```

- Hogyan változik *a, b* és *c* értéke a folyamat során, ha a=7 és b=3?
- Mi a kimenet ebben az esetben?
- Hányszor ismétlődik a ciklus?
- Hányszor kell kiértékelni a feltételt?
- Mit csinál az algoritmus?

Feladat: pszeudokód 3

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=(int)(N/10)
enddo
output R
```

- Hogyan változik N és R értéke a folyamat során, ha kezdetben N=73251?
- Mi a kimenet ebben az esetben?
- Mit csinál az algoritmus?

Jelmagyarázat:

%: maradékos osztás (modulo)

(int): egészrész képzés
(a törtrész elhagyása)

Feladat: pszeudokód 4

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

- Mi a kimenet, ha N=15, B=2?
- Mi a kimenet, ha N=16, B=2?
- Mi a kimenet, ha N=10, B=2?
- Mi a kimenet, ha N=5, B=2?
- Mi a kimenet, ha N=30, B=3?
- Mi a kimenet, ha N=20, B=3?
- Mi a kimenet, ha N=64, B=8?

- Mit csinál az algoritmus?

Feladat: pszeudokód 5

```
input A
input B
while B>0 do
   C=B
   B=A%B
   A=C
enddo
output A
```

- Hogyan változik A, B és C értéke a folyamat során, ha kezdetben A=24 és B=18?
- Mi a kimenet ebben az esetben?
- Próbáld ki: A=30 és B=105!
- Próbáld ki: A=165 és B=48!
- Mit csinál az algoritmus?

(Euklideszi algoritmus)

Feladat: pszeudokód 6

```
input x,y
z=x*y
while y>=1 do
   w=y
   y=x%y
   x=w
enddo
output z/x
```

- Hogyan változik x, y, z és w értéke a folyamat során, ha kezdetben x=12 és y=30?
- Mi a kimenet ebben az esetben?
- Próbáld ki: x=14 és y=15!
- Próbáld ki: x=18 és y=18!
- Mit csinál az algoritmus?

Feladat: pszeudokód 7

```
input A
input B
while A!=B do
   if A>B then
      A=A-B
   else
      B=B-A
   endif
enddo
output B
```

- Hogyan változik A és B értéke a folyamat során, ha kezdetben A=24 és B=18?
- Mi a kimenet ebben az esetben?
- Próbáld ki: A=30 és B=105!
- Próbáld ki: A=165 és B=48!
- Mit csinál az algoritmus?

- Készíts folyamatábrát ehhez az algoritmushoz!

Feladat: lineáris kongruencia generátor

```
input m, a, x
x0=x
while 1==1 do
   x=(a*x)%m
   output x, " "
   if x==x0 then
     break
   endif
enddo
```

Mi a kimenete az algoritmusnak az alábbi bemenetek esetén?

- a=5, m=16, x=1
- a=12, m=7, x=6
- a=12, m=7, x=1
- a=16807
  m=2147483647
  x=1672552800

Feladat: páros vagy páratlan

Algoritmus mondatszerű leírása:

1. Mondj egy nemnegatív egész számot!
2. Ellenőrizd, hogy nagyobb, mint 1, vagy nem!
3. Ha nagyobb, vonj ki belőle kettőt, és menj a 2. lépéshez!
4. Különben ellenőrizd, hogy 0-e az érték!
5. Ha 0, akkor írd ki, hogy ‚*páros*'!
6. Különben írd ki, hogy ‚*páratlan*'!

Írd le az algoritmust pszeudokóddal (és folyamatábrával)!

Feladat: elágazások

- Adott 3 szám. Pszeudokóddal add meg azt az algoritmust, amely meghatározza mi a legkisebb érték ezek közül!

- Adott 3 pozitív szám. El kell dönteni, hogy szerkeszthető-e olyan háromszög, amelynek oldalai ilyen hosszúságúak. Pszeudokóddal írd le a háromszög-egyenlőtlenség probléma megoldását!

- Adott 3 pozitív szám. Add meg azt az algoritmust, amely megmondja létezik-e olyan derékszögű háromszög, amelynek oldalai ilyen hosszúságúak!

Feladat: iterációk

- Írd le pszeudokóddal azt az algoritmust, amely megadja, hogy mennyi az egész számok összege a [10;20] zárt intervallumban!
- Írd le pszeudokóddal azt az algoritmust, amely megadja, egy a felhasználó által megadott pozitív egész szám faktoriálisát!
- Írd le pszeudokóddal azt az algoritmust, amely segítségével meghatározható az x^y hatvány értéke! Az x és y értékeket felhasználó adja meg.

Feladat: négyzetgyök

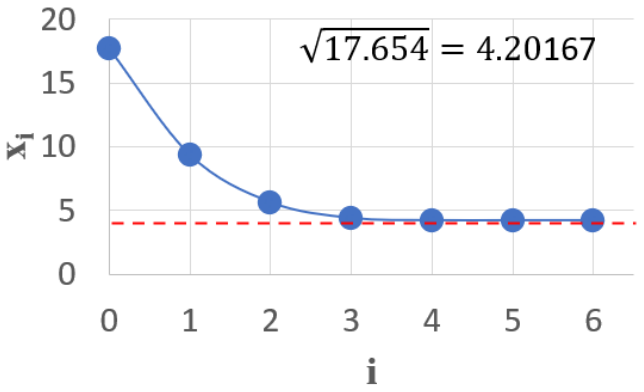
Egy szám négyzetgyöke meghatározható a Newton-Raphson módszert alkalmazó közelítéssel. Ehhez az alábbi iteratív formulára van szükségünk, amely végtelen lépés során konvergál a megoldáshoz.

- Írj egy algoritmust, amely az iterációt addig folytatja, amíg 2 egymás utáni érték kevésbé tér el egymástól, mint a felhasználó által megadott küszöbérték!

$$x_0 = N$$

$$x_{i+1} = x_i - \frac{x_i^2 - N}{2x_i}$$

$$\lim_{i \to \infty} x_i = \sqrt{N}$$



Feladat: prímek

- Írd le pszeudokóddal azt az algoritmust, amely eldönteni egy felhasználó által megadott 1-nél nagyobb egész számról, hogy prím szám-e!

- Írd le pszeudokóddal azt az algoritmust, amely kiírja azokat a prím számokat amelyeknek a szorzata megegyezik egy a felhasználó által megadott pozitív egész számmal! (Prímtényezőkre bontás)

Feladat: szökőnapok

- A felhasználó (növekvő sorrendben) két évszámot ad meg 1901 és 2099 között. Írd meg azt az algoritmust, amely megmondja hány szökőnap van a két megadott év első napjai között!

Például

bemenet:

1979 2023

kimenet:

***11** szökőnap van 1979.01.01 és 2023.01.01 között*

Feladat: törtek

Egy hagyományos tört megadható 2 egész számmal.

- Írd le azt az algoritmust pszeudokóddal, amely hagyományos törtek egyszerűsítésére képes!
  Például: 36/90 → 2/5
- Írd le azt az algoritmust pszeudokóddal, amely tud hagyományos törtekkel műveleteket (+ - * /) végezni
  Például: 4/6+2/8 →11/12; (2/3) / (4/5) → 5/6
- Írd le azt az algoritmust pszeudokóddal, amely képes egy tizedes törtet hagyományos törtként felírni.
  Például: 0,375 → 3/8

Feladat: sorozatok

A Fibonacci-sorozat egész számok egy sorozata, amely 0-val és 1-gyel kezdődik majd minden további eleme az előző két elem összege.

- Írd le azt az algoritmust pszeudokóddal, amely …
  … megadja a Fibonacci-sorozat első 100 elemét!
  … megadja a sorozat 1000-nél kisebb elemeit!

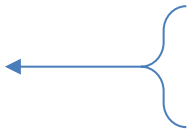

A Collatz-sejtés definiál sorozatot, amely egy tetszőleges pozitív egész számmal kezdődik. Minden további elemet (a_{i+1}) ez előtte levő (a_i) határoz meg. Ha a_i páros, akkor $a_{i+1}=a_i/2$, különben $a_{i+1}=3a_i+1$.

- Írd le a generáló algoritmust, amely $a_i=1$ esetén megáll.

Tömbök

- N db elem sorozata, ahol minden elem egy (0-val kezdődő sorszám jellegű) index segítségével érhető el

```
N=5
A[]={2,93,4,-1,8}
sum=0
i=0
while i<N do
   sum=sum+A[i]
   i=i+1
enddo
output sum/N
```

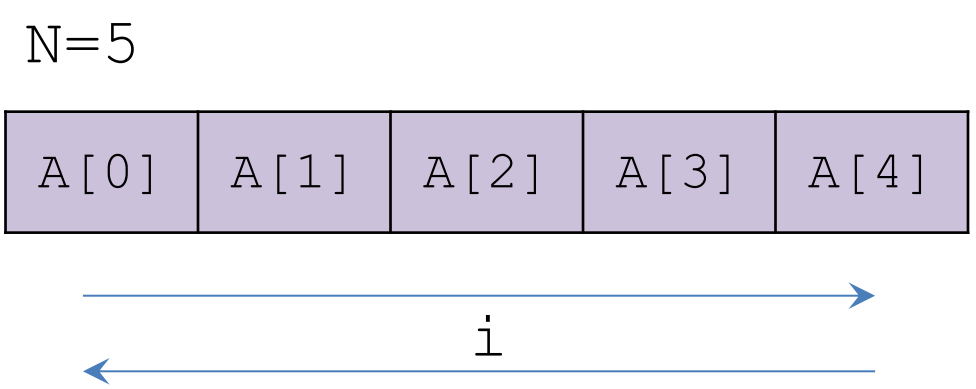


kezdő értékadás:
`A[0]=2, A[1]=93, …, A[4]=8`

Tömb példa

- 5 db értékek beolvasása, tárolás és kiírása fordított sorrendben

```
N=5
A[]={0,0,0,0,0}
i=0
while i<N do
   input A[i]
   i=i+1
enddo
while i>0 do
   i=i-1
   output A[i], ” ”
enddo
```



Feladat: jelkódolás

Adott egy N elemű tömb, amelyben 0 és 1 értékek (bitek) szerepelnek tetszőleges sorrendben.

- Kódold le ezeket az értékeket Manchester kódolással, azaz ha az érték 1, akkor írj ki egy nullát és egy egyest, különben fordított sorrendben írd ki ezeket!
- Kódold le az értéket NRZI kódolással, azaz először írj ki egy 0-t (függetlenül az első elem értékétől) majd ha az adott tömbelem 0, akkor írd ki a legutóbb kiírt értéket, különben a legutóbb kiírttól eltérő bit értéket!

Feladat: érmék száma

Hozz létre egy algoritmust pszeudokód segítségével, amely egy felhasználó által megadott összegről megmondja, hogy legkevesebb hány érmével lehet azt kifizetni!

- Lehetséges érmék: 1, 2, 5, 10, 20, 50, 100
- Lehetséges érmék: 5, 10, 20, 50, 100, 200
  (bolti kerekítési szabály alkalmazásával)

Keresés és rendezés

- **Keresés**
  Az adott elem benne van-e a tömbben (és ha igen hol fordul elő először)?
  - Lineáris keresés (őrszemmel/őrszem nélkül), bináris keresés, stb.
- **Rendezés**
  Elemek érték szerint növekvő/csökkenő rendjének kialakítása (az értékek eredeti helyén).
  - Kiválasztásos rendezés, beszúrásos rendezés, buborékrendezés, radix rendezés, gyors rendezés, kupacrendezés stb.

Feladat: keresés

Tegyük fel van `T` nevű `N` elemű inicializált tömbünk.

- Írj egy pszeudokódot, amely meghatározza, hogy egy felhasználó által keresett érték benne van-e a (rendezetlen/rendezett) tömbben!
  (lineáris keresés, őrszem alkalmazás, bináris keresés)
- Hogyan függ a keresés lépésszáma N értékétől?
- Írj egy pszeudokódot, amely meghatározza a tömb legnagyobb/legkisebb elemének értékét!
- Írj egy pszeudokódot, amely meghatározza a legnagyobb elem első előfordulásának indexét!
- Hogyan kereshető meg a második legnagyobb elem?

Feladat: rendezés

- Hogyan cserélnéd meg két változó értékét!

Tegyük fel van `T` nevű `N` elemű inicializált tömbünk.

- Írj egy pszeudokódot, amely a tömb elemet növekvő sorrendbe rendezi …
  … a kiválasztásos rendezés elvei alapján
  … a beszúrásos rendezés elvei alapján
  … a buborék rendezés elvei alapján
  … a radix rendezés elvei alapján

- Hogyan függ a rendezés lépésszáma N értékétől?
- Hogyan csinálnál két rendezett tömbből egyetlen rendezettet, ami az összes elemet tartalmazza?

Feladat: bankkártya száma

Bank Kártya Szám (BKSz): $a_1, a_2 \ldots, a_{16}$ $\quad 0 \leq a_i \leq 9$

Egy BKSz minden páratlan pozíciójú számjegyét cseréld le a duplájára vagy ha ez több, mint 9 akkor annál 9-cel kisebb számra! Add össze számjegyeket! Egy BKSz akkor és csak akkor érvényes, ha az így kapott szám osztható 10-zel.

Azaz egy BKSz akkor érvényes, ha

$$\left(\sum_{i=1}^{16}\left((i\%2+1)*a_i-\left\{\frac{a_i}{5}\right\}*(i\%2)*9\right)\right)\%\,10==0$$

A BKSz 16 számjegye egy tömb egyes elemeiben van. Írj egy pszeudokódot, ami ellenőrzi a BKSz érvényességét!

Feladat: leggyakoribb kor

- Egy városban lakó 18 év alattiak száma `N`. Az egyes gyerekek kora az `Age` nevű tömbben van eltárolva. Írj egy algoritmust pszeudokóddal, amely megmondja, hogy milyen korú gyerekből van a legtöbb!

Például:

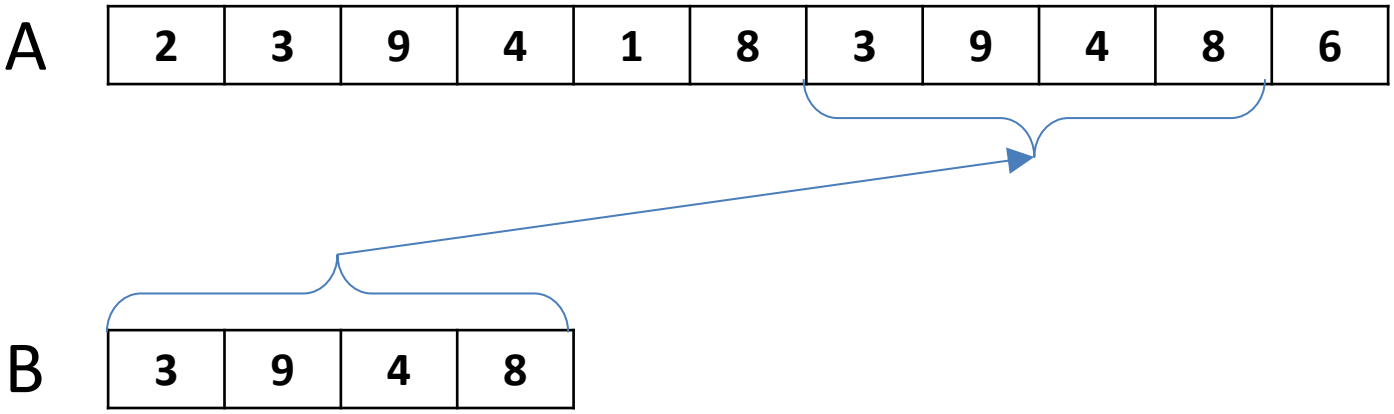
```
N=15
Age[]={1,3,2,0,5,10,17,3,10,0,3,2,15,1,3}
```

3 éves gyerekből van a legtöbb.

Feladat: mintaillesztés

- Adott egy A nevű N elemű tömb és egy B nevű M elemű tömb. Írd le azt a pszeudokódot, amely azt az algoritmust reprezentálja, ami megmondja, hogy a B tömb összes eleme ugyanebben a sorrendben egymás mellett előfordul-e az A tömbben!

Például:

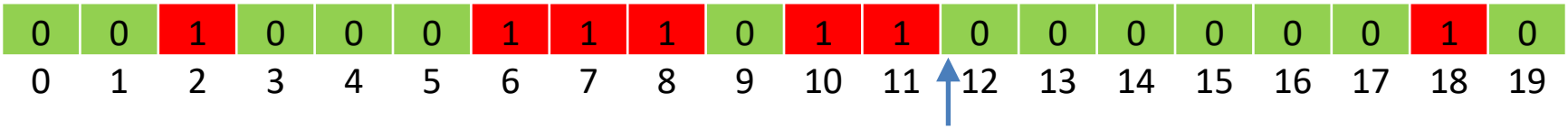


Feladat: first-fit allokáció

A `M` tömb elemei (`N` darab) a memória egyes cellák foglaltságát jellemzik. `M[i]=1` azt jelenti, hogy az `i`. memória cella foglalt, a `M[i]=0` pedig azt, hogy az `i`. cella szabad.

- Írj egy programot, amely bekér egy pozitív egész számot (`S`) és megmondja, hogy a memóriában melyik cellával kezdődik az első olyan memóriaterület, amely folytonosan legalább `S` darab szabad cellát tartalmaz!

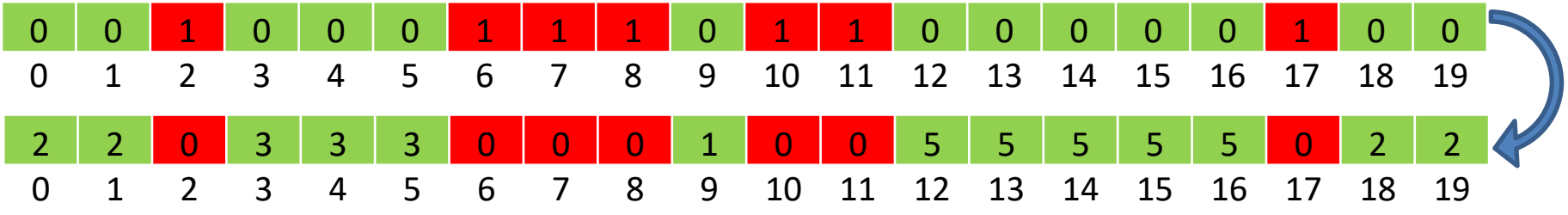
  Például S=4 esetén a válasz 12.



Feladat: memory-map konverzió

A `M` tömb elemei (`N` darab) a memória egyes cellák foglaltságát jellemzik. `M[i]=1` azt jelenti, hogy az `i`. memória cella foglalt, a `M[i]=0` pedig azt, hogy az `i`. cella szabad.

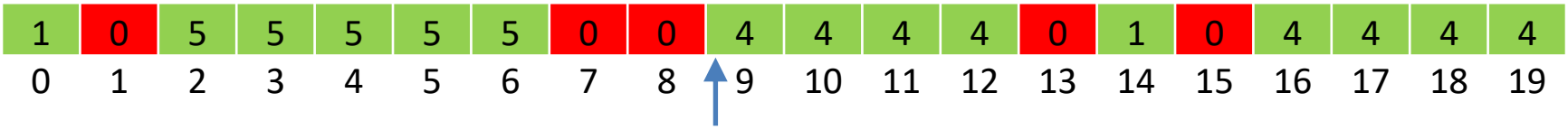
- Írj egy programot, amely tömböt átalakítja úgy, hogy az eredeti 1 értékek helyett 0 tárol, míg a 0-k helyett azt a pozitív egész számot, ahány darab 0 volt folyamatosan az adott cella környezetét jelentő területen.



Feladat: best-fit allokáció

A `M` tömb elemei (`N` darab) a memória egyes cellák foglaltságát jellemzik. `M[i]=0` azt jelenti, hogy az `i`. memória cella foglalt, a `M[i]=x (x>0)` pedig azt, hogy az `i`. cella egy `x` darab szabad cellából álló folytonos terület része.

- Írj egy programot, amely bekér egy számot (`S`) és megmondja, hogy a memóriában melyik cellával kezdődik az első olyan memóriaterület, amely a legalább `S` méretű szabad területek közül legkisebbnek az első előfordulása (ha létezik)!
  Például S=3 esetén a válasz 9.



Alprogram

Az algoritmus elkülönített egysége

Az **újrafelhasználás** és az **absztrakció** eszköze

- Újrafelhasználás: Nem szükséges ugyanazt az utasítássorozatot a kód több helyén is megírni. Be tudjuk tanítani a tevékenységet, hogy egyszerű utasításként működjön. Egy példa a *beágyazásra*.
- Absztrakció: Nemcsak egy tevékenység, de hasonló tevékenységek halmazára is használható, paraméterekkel megadva a finomságokat. Az *általánosítás* megjelenési formája.

Alprogram

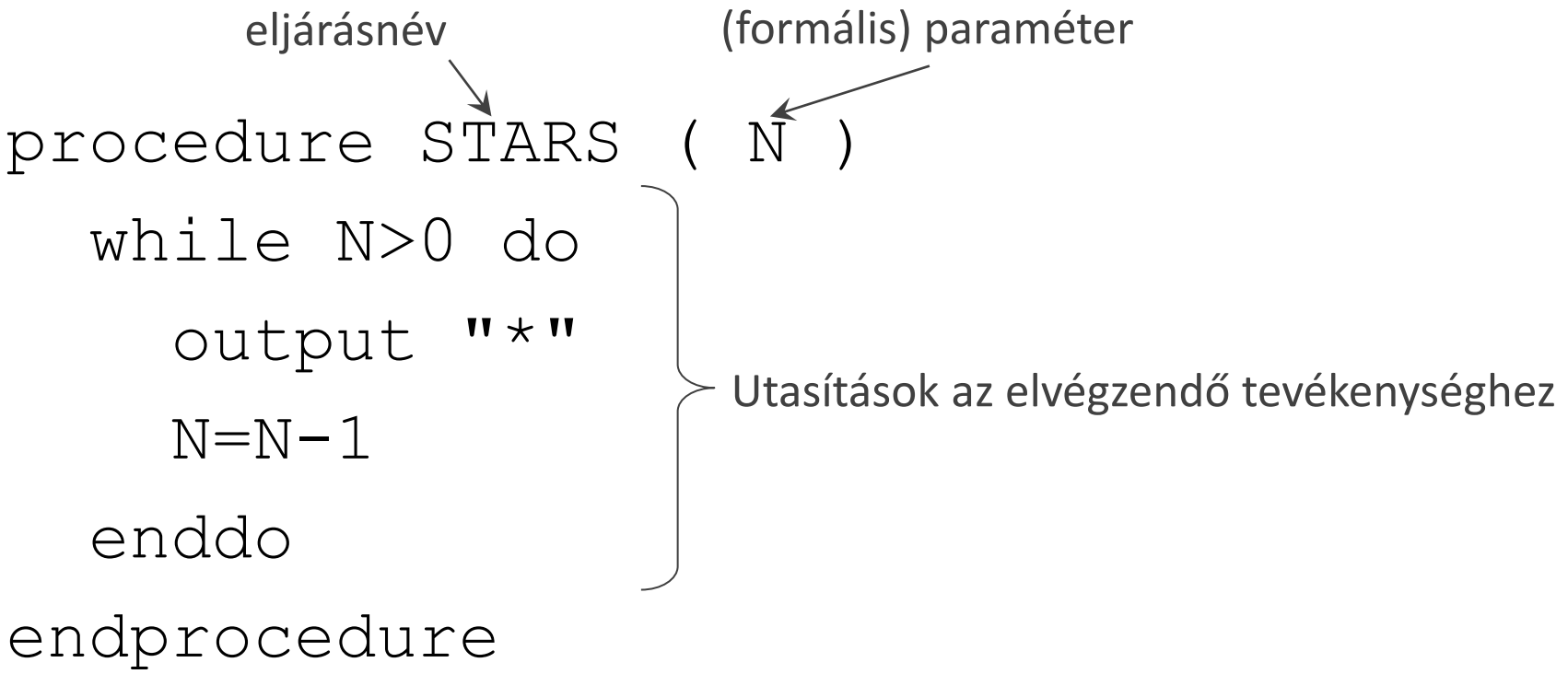
Az alprogramok két fajtája:

- **Függvény**: utasítássorozat egy érték (a visszatérési érték) előállítására valahol a kódban.
  Pl. Mi a koszinusza a 45°-nak?
  `x=cos(45)`

- **Eljárás**: tevékenységsorozat valami megtételéhez a kód egy adott pontján (nincs visszatérési érték)
  Pl. Rendezd az N elemű T tömb elemeit növekvőleg!
  `call Rendezes(T,N)`

Eljárás példa

Eljárás definíció N db ’*’ karakter kiírásához



Eljárás példa

Az előző eljárás meghívása egy kódban

```
output "Hány csillag kell?"
input S
call STARS (S)
output "Remélem tetszik!"
```

(aktuális) paraméter

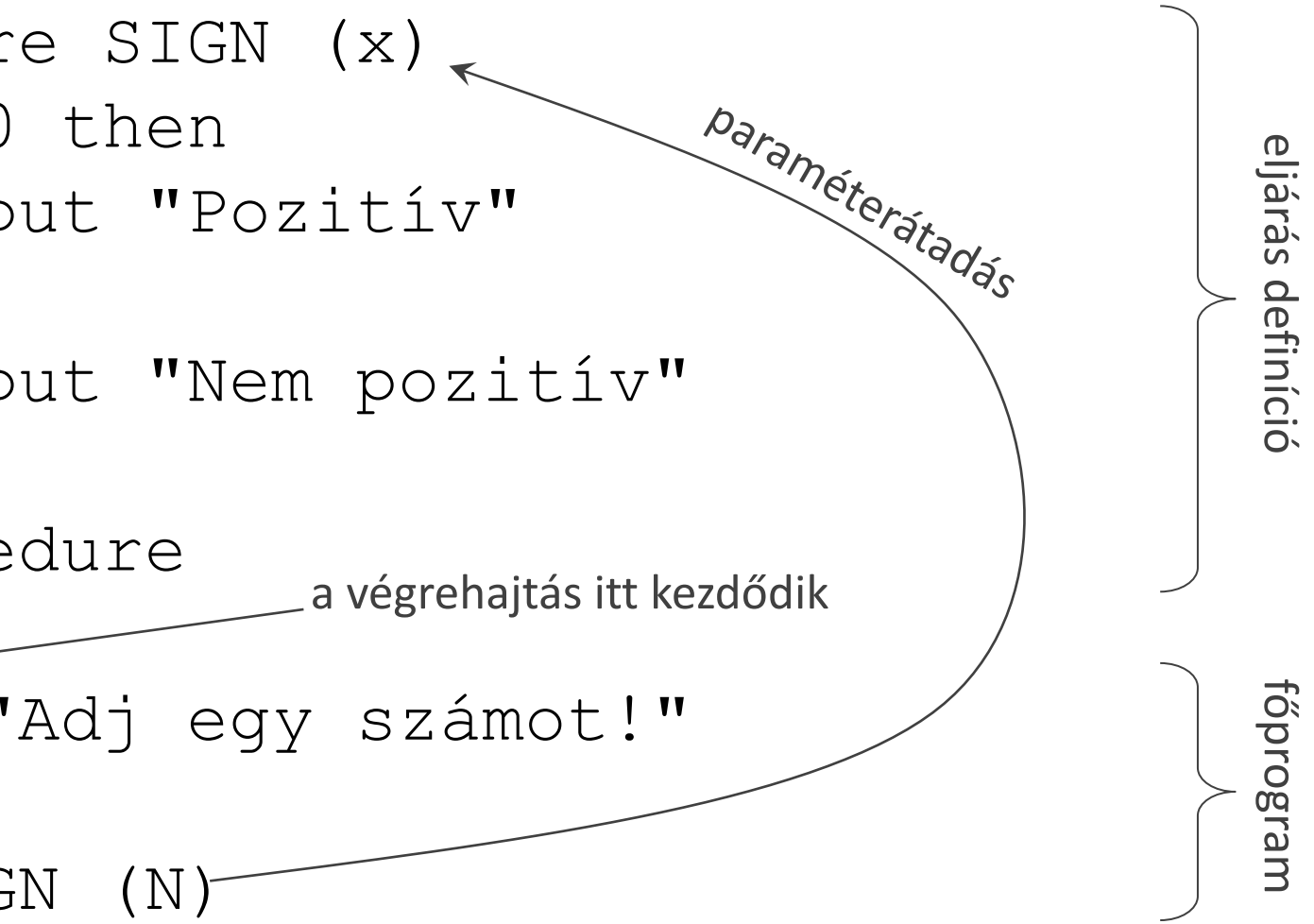
eljáráshívás (az utasításainak végrehajtása)

Eljárás példa

Egy eljárás, amely megmondja egy számról pozitív-e

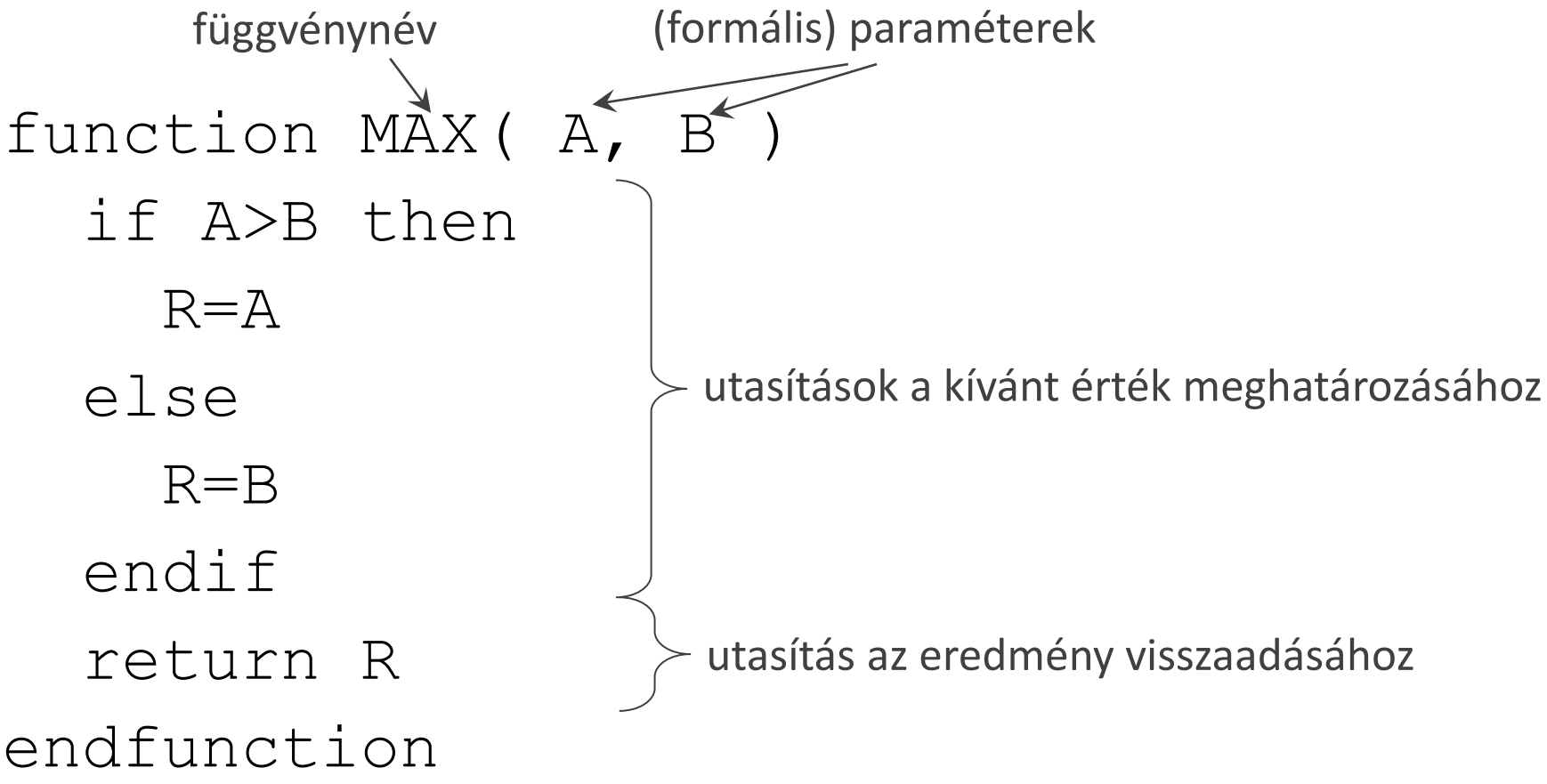
```
procedure SIGN (x)
   if x>0 then
     output "Pozitív"
   else
     output "Nem pozitív"
   endif
endprocedure

output "Adj egy számot!"
input N
call SIGN (N)
```



Függvény példa

Függvénydefiníció két érték maximumának meghatározásához



Függvény példa

Az előző függvény meghívása egy kódban

```
output "Adj meg két számot!"
input a, b
c = MAX (a,b)
output "A nagyobb: ", c
```

(aktuális) paraméterek

függvényhívás (az érték meghatározása)

a visszaadott érték a c változóban tárolódik el

Függvény példa

Egy másik függvény az abszolút érték kiszámításához

```
function ABS (x)
   if x<0 then
     x=-1*x
   endif
   return x
endfunction

output "Adj meg egy számot!"
input N
A = ABS (N)
output "Az abszolút értéke: ", A
```

paraméterátadás

a végrehajtás itt kezdődik

függvénydefiníció

főprogram

Eljárás kontra függvény

```
procedure Avg(x,y,z)
  a=(x+y+z)/3
  output a
endprocedure
call Avg(2,5,4)
```

≠

```
function Avg(x,y,z)
  a=(x+y+z)/3
  return a
endfunction
if Avg(2,5,4)<4 then
  output "Week!"
else
  output "Good job!"
endif
```

Az átlag
a képernyőre írva.

Az átlag **nincs**
a képernyőre írva.

Hatáskör

- A különböző programegységek (függvények, eljárások, főprogram) saját változókkal rendelkeznek
- Egy változónév más és más változóra hivatkozik két eltérő programegységben
  - Még akkor is, ha esetleg azonos a név alakja.

```
function F( ABC )
   return ABC+1
endfunction

ABC = 3
ABC = F(7)
ABC = F(ABC)*2
```

Feladat: alprogram

```
function PP( a )
  b=0
  while b<a do
    a = a-1
    b = b+1
  enddo
  if a==b then
    return 1
  else
    return 0
  endif
endfunction

input a, b
a = a*2
b = PP(a+b)+1
output b
```

- Hogyan változnak a változók értékei a végrehajtás során, ha a felhasználó az 1 és 4 értékeket adja meg bemenetként?
- Mi a kimenet ekkor?
- Mit csinál a függvény pozitív egész paraméter esetén?

Feladat: alprogram

```
function VALTOZTAT( a )
   return 1-a
endfunction
input Max
i=0
j=0
while j<Max do
   i = VALTOZTAT(i)
   j = j+i
   output j, " "
enddo
output j
```

- Mi a kimenet, ha a bemenet: 5?
- Mit csinál a program?
- Mi a függvény szerepe?

Feladat: alprogram

```
function min(x,y)
  if x<y then
    return x
  endif
  return y
endfunction
function even(N)
  return 1-N%2
endfunction
C[]={2,3}
input x,y
N=(min(min(x,C[1]),x-3)+C[even(y)]*5)%6
output N
```

- Mi van a képernyőn, ha a felhasználó által adott adatok: 4 és 5?

Feladat: szorzótábla

- Írj egy pszeudokódot, amely tartalmaz egy eljárást az NxN-es szorzótábla kiírásához!
  Például, ha N=4:

Új sor:

```
output NEWLINE
```

1	2	3	4
2	4	6	8
3	6	9	12
4	8	12	16

Feladat: pepita minta

- Írj egy pszeudokódot, amely tartalmaz egy eljárást egy ’0’ és ’1’ karakterekből álló NxN-es pepita minta megalkotásához!
  Például

N=3 esetén:

0	1	0
1	0	1
0	1	0

N=4 esetén:

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

Feladat: idő

Írj egy algoritmust pszeudokóddal az alábbiak szerint:

- A kód tartalmazzon egy függvényt, amely három paraméterrel rendelkezik (`óra, perc, másodperc`) és ez alapján megadja, hogy az adott időpont a nap hányadik másodperce!
- A kód tartalmazzon egy eljárást, amely egy egész számot kap paraméterként és ezt az adott napon eltelt másodpercek számaként értelmezve kiírja az időpontot `óra:perc:másodperc` formátumban!
- Hívd meg az alprogramokat ellenőrzésként!

Feladat: képlet

- Írj egy pszeudokód függvényt, amely egyetlen paraméterrel rendelkezik (N) és visszaadja az alábbi képlet által meghatározott érték (x) faktoriálisát! A függvényt hívd meg egy felhasználó által megadott pozitív egész értékre és írasd ki a függvény által visszaadott értéket a főprogramban!

$$x = \left(\frac{\sum_{i=1}^{N} (2i - 1)}{N^2} \right)$$

Feladat: DCB aritmetika

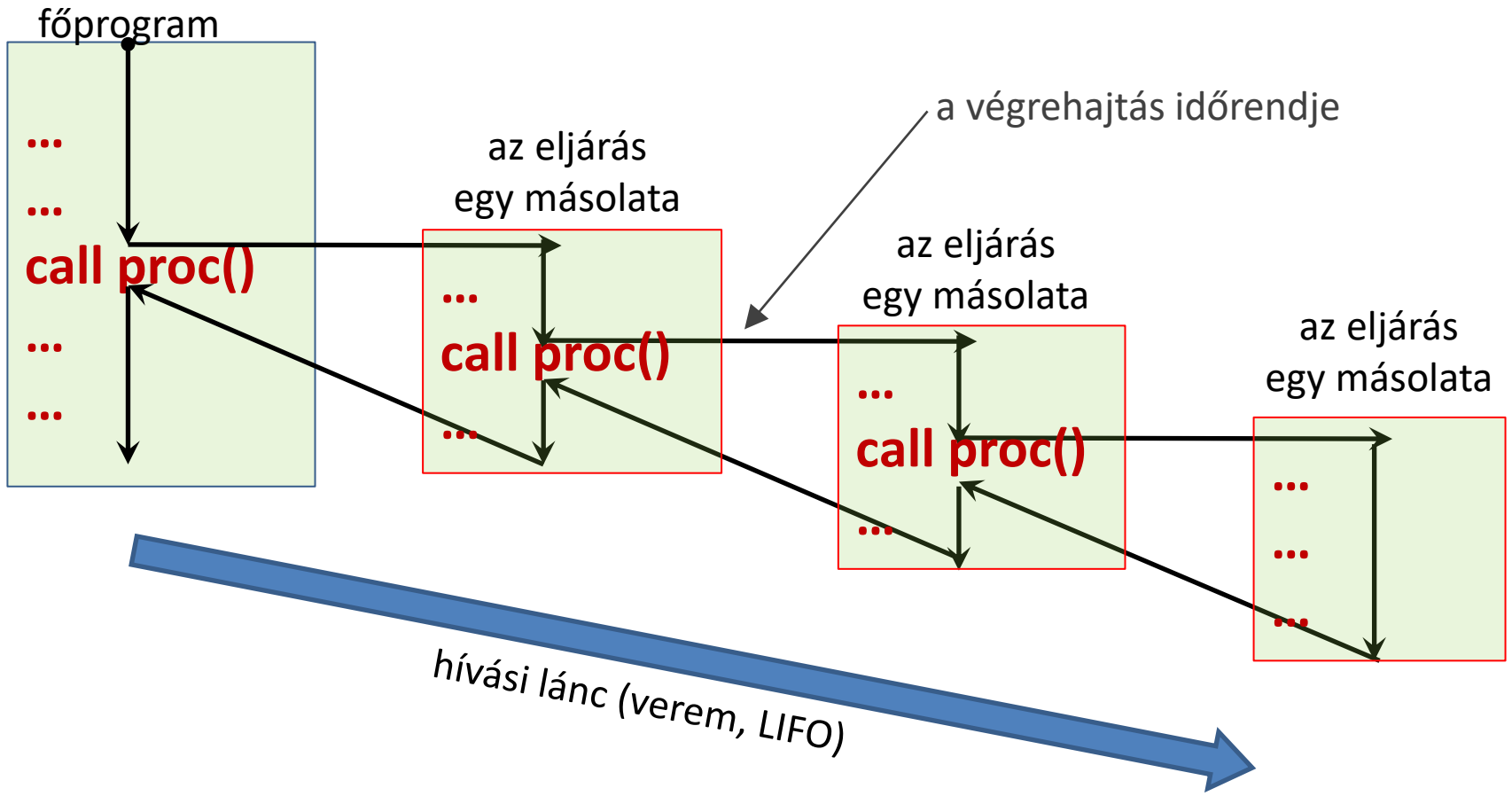
Bináris számokkal szeretnénk matematikai műveleteket végezni, de a pszeudokód csak decimális számokat használ. Alkalmazzuk egy "trükköt"! A kívánt műveletek csak decimális számokat használjanak, amelyekben kizárólag 0 és 1 számjegyek találhatóak, mintha bináris értékek lennének. Használj alprogramokat!

- Írj egy pszeudokóddal reprezentált algoritmust, amely képes nem negatív, bináris egész számok halmazán meghatározni az alábbi értéket!

$$Q = \frac{X^Y}{Z}$$

Rekurzió

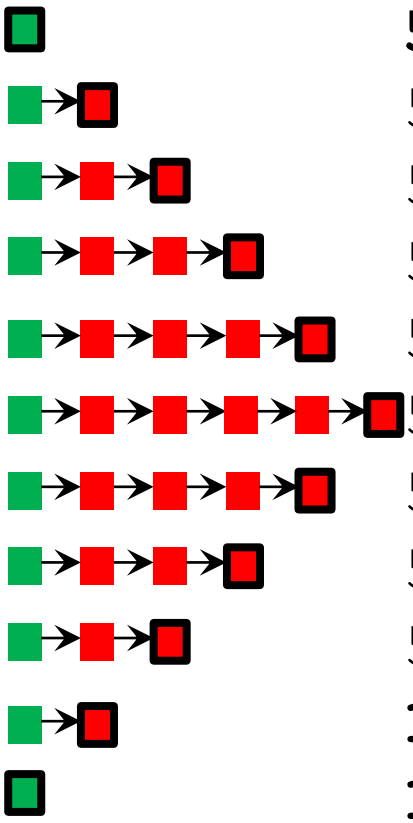
- Amikor egy alprogram saját magát hívja meg



Rekurzió példa

- Faktoriális (N!):
  - Ha a szám (N) az 1, akkor a faktoriálisa is 1.
  - Különben N! = N * (N-1)!

```
function fakt(N)
   if N==1 then
      return 1
   else
      return N*fakt(N-1)
   endif
endfunction
output fakt(5)
```



Feladat: rekurzió 1

```
procedure KONV ( N , B )
   if N!=0 then
      call KONV ( (int)(N/B) , B)
      output N%B
   endif
endprocedure

call KONV (16,8)
output NEWLINE
call KONV (19,2)
```

a hányados egészrésze

- Próbáld meg fejben futtatni!
- Mi a kimenet?
- Milyen hosszú a hívási lánc?
- Hol van a számjegyek fordított sorrendje lekódolva?

Feladat: rekurzió 2

- Milyen kimenetet szolgáltat az alábbi rekurzív algoritmus?

```
procedure something(B,E,P)
   if E>0 then
      call something(B,E-1,P*B)
   else
      output P
   endif
endprocedure
call something(5,3,1)
```

Feladat: rekurzió 3

- Írj egy algoritmust, amely bekér a felhasználótól számokat, mindaddig, amíg 0-t nem kap ezután a program írja ki a kapott értékeket (a 0 kivételével) fordított sorrendben! A beadott számok darabszáma nem ismert. Az algoritmusban ne használj tömböt!

Például:

bemenet: 1, 4, 65, -9, 2, 9, 2, 0

kimenet: 2, 9, 2, -9, 65, 4, 1

Algoritmikus gondolkodás

Szükséges hozzá:

- Tudás, ismeret (*tanulás*)
- Tapasztalat (*gyakorlás*)
- Kreativitás, intelligencia (*adottság, tehetség*)

Nincs útmutató (algoritmus) arra, hogy hogyan kell algoritmust írni.

Tisztán memorizálással, magolással nem lehet elsajátítani az algoritmikus gondolkodás képességét.

Algoritmikus gondolkodás nélkül nincs programozó.

Gondolatébresztő kérdések

- Mi az, amit a felhasználó fog megadni? (bemenet)
- Mi az, amit látnia kell a felhasználónak? (kimenet)
- Kell-e valamit tárolni, adatra emlékezni? (változó)
- Kapott-e már értéket egy változó, mielőtt felhasználnánk azt? (kezdőérték)
- Hogyan kell kiszámolni egy értéket? Milyen sorrendben, mely műveletek szükségesek? (kifejezés kiértékelés)
- Van-e olyan lépés, amely csak néha bizonyos esetekben szükséges? (elágazás)

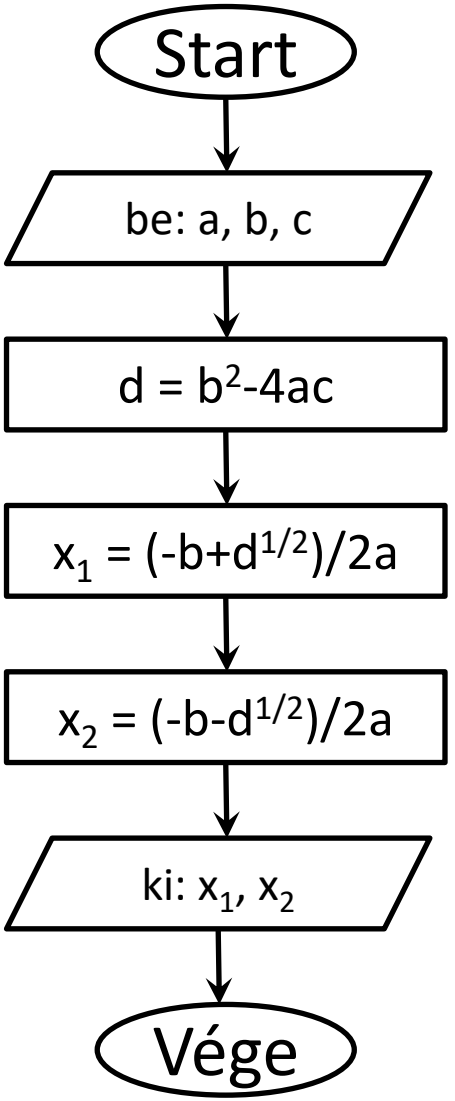
Gondolatébresztő kérdések

- Kell-e ismételni egyes lépéseket? (ciklus)
- Milyen esetben kell valamit végrehajtani? (feltétel)
- Több helyzet megléte egyszerre szükséges vagy elég az egyik is magában? (logikai operátor)
- Kell-e kezelni összefüggő érték sorozatot? (tömb)
- Van-e olyan tevékenység csoport, amely több helyen is szükséges lehet? (eljárás, függvény)
- Melyek azok az értékek, amelyek befolyásolják egy tevékenységcsoport működését? (paraméter)

A tesztelési stratégia fejlesztése

Tesztelési stratégia példa

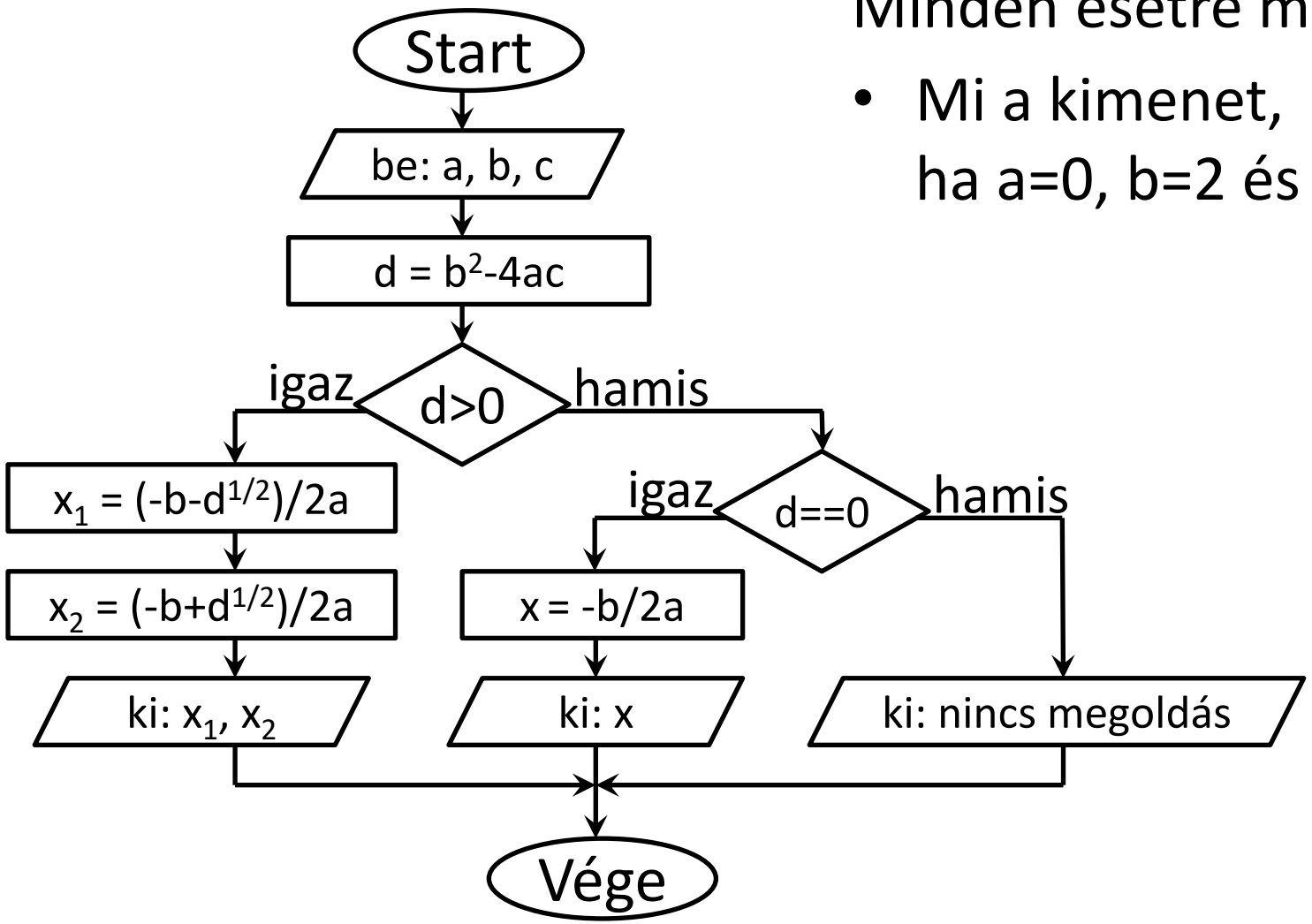
- Másodfokú egyenlet megoldása
- Általános alak: $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$
- Bemeneti paraméterek: a, b, c
- Megoldás: $\mathbf{x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}}$



Minden esetre működik?

- Mi a kimenet,
  ha a=1, b=2 és c=1?
- Mi a kimenet,
  ha a=1, b=2 és c=2?

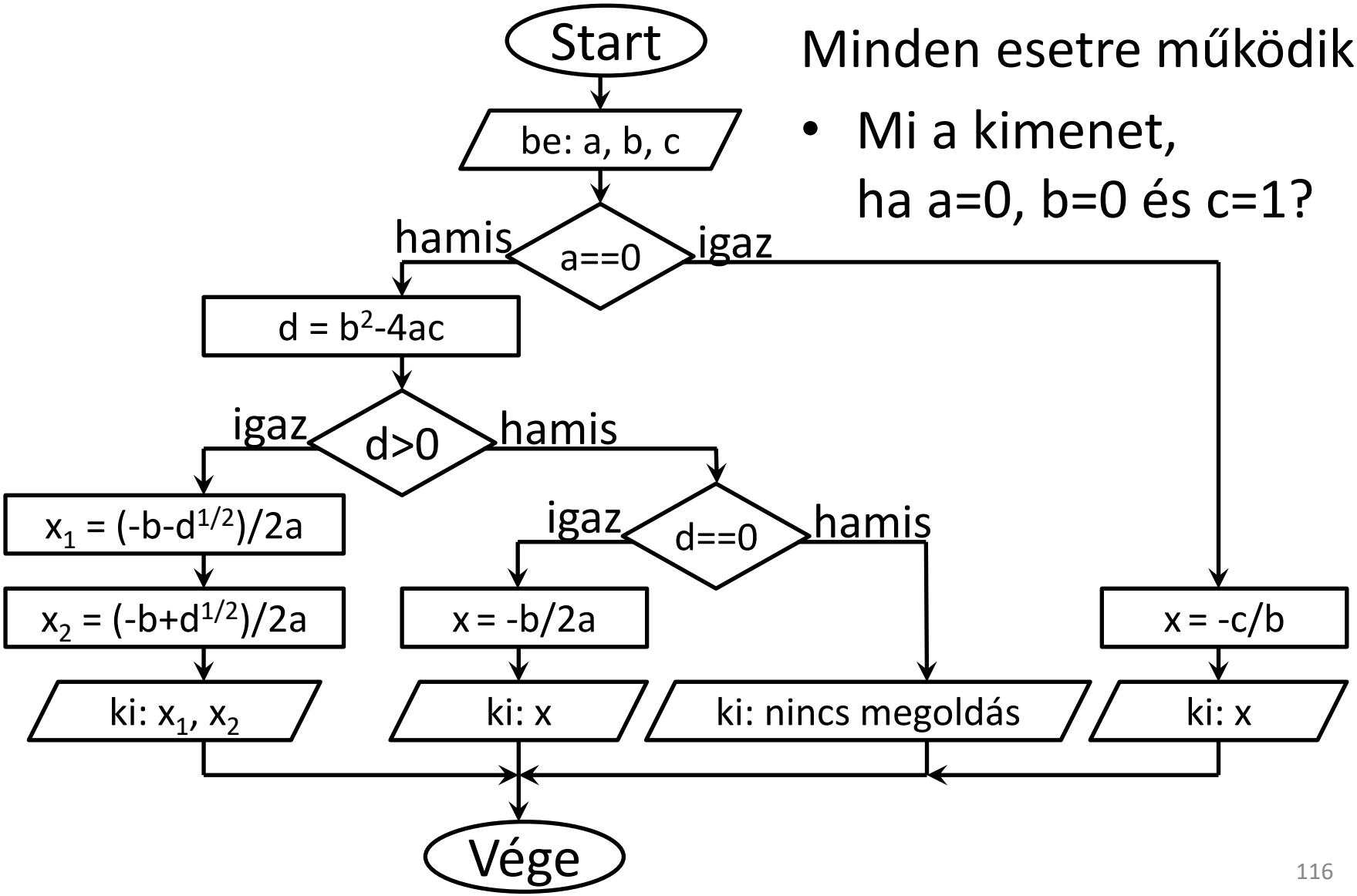
Tesztelési stratégia példa



Minden esetre működik?

- Mi a kimenet,
  ha a=0, b=2 és c=6?

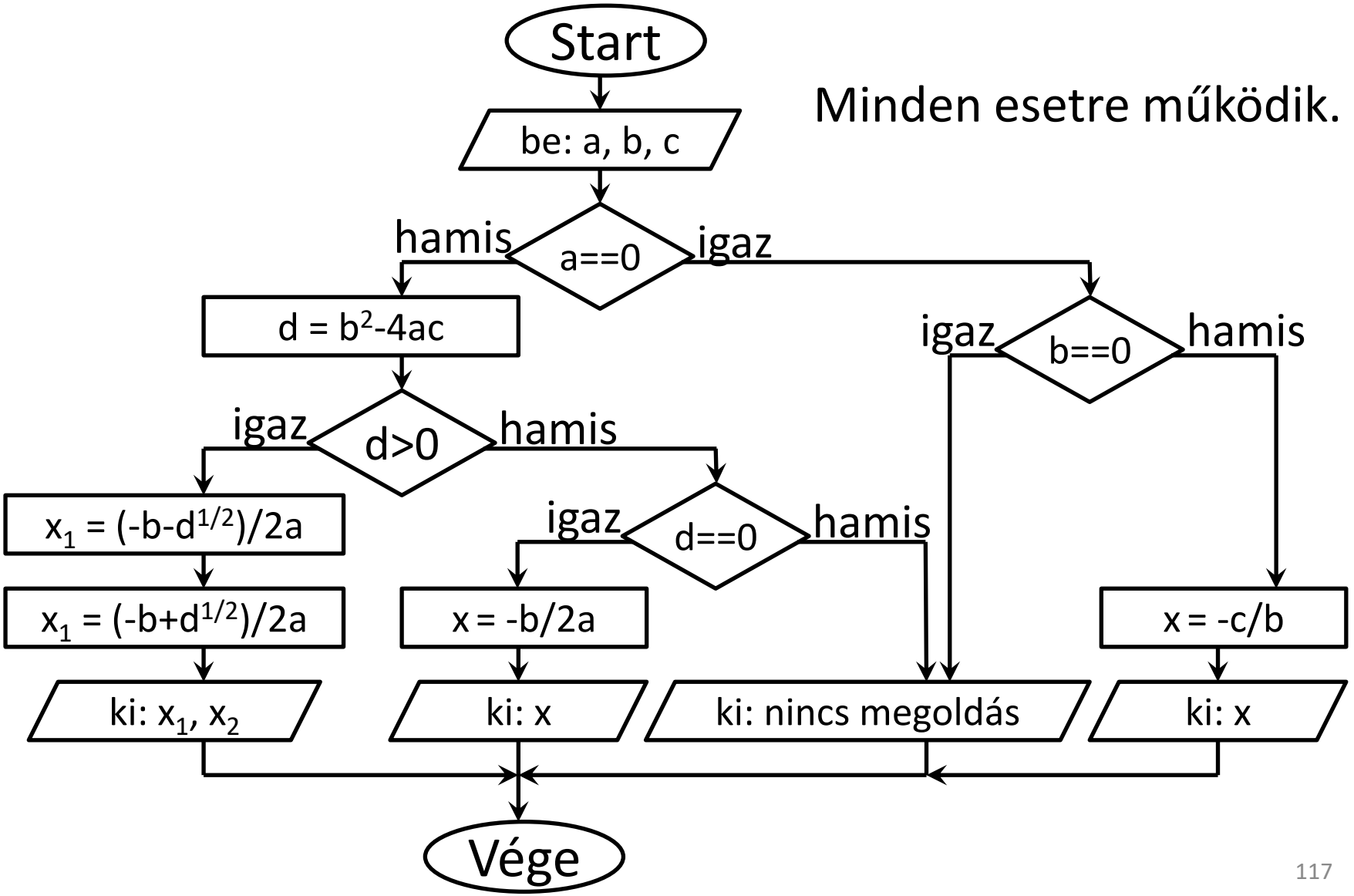
Tesztelési stratégia példa



Minden esetre működik?

- Mi a kimenet,
  ha a=0, b=0 és c=1?

Tesztelési stratégia példa



Minden esetre működik.

Tesztelési stratégia példa

A jó megoldás pszeudokóddal:

Minden esetre működik.

Hogy elérjük ezt az állapotot, tesztelnünk kellett az algoritmust különböző input kombinációkra azaz **tesztesetek**re, és folyamatosan módosítanunk kellett azt.

Tesztelési stratégiát alakítottunk ki.

```
input a, b, c
if a==0 then
  if b==0 then
    output "error"
  else
    x=-c/b
    output x
  endif
else
  d=b*b-4*a*c
  if d>0 then
    x1=(-b+sqrt(d))/(2*a)
    x2=(-b-sqrt(d))/(2*a)
    output x1, x2
  else
    if d==0 then
      x=-b/(2*a)
      output x
    else
      output "error"
    endif
  endif
endif
```

A használt tesztelési stratégia

a	b	c	output	Magyarázat	OK
3	7	2	-1/3 ; -2	általános eset (nem 0, d>0)	✓
0	2	6	-3	**a** nulla (elsőfokú)	✓
2	0	-8	2 ; -2	**b** nulla (x^2=-c/a)	✓
1	2	0	0 ; -2	**c** nulla (x[ax+b]=0)	✓
0	0	1	"error"	több nulla (nem egyenlet)	✓
1	2	2	"error"	**d**<0 (nincs megoldás)	✓
1	2	1	-1	**d**=0 (csak egy megoldás)	✓
-2	-10	-1	-4.9 ; -0.1	negatív bemenetek	✓
2.3	-4.2	0.83	1.6 ; -0.23	nem egész bemenetek	✓
10^{-5}	10^5	1	-10^{-5} ; 10^{10}	extrém kicsi/nagy értékek	✓

teszt esetek

Feladat: tesztelési stratégia

Számrendszerátváltás

- Hozz létre tesztelési stratégiát az algoritmushoz!
- Az N és B mely értékei elfogadhatóak? (Mikor szolgáltatja az algoritmus az elvárt eredményt?)

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

Feladat: tesztelési stratégia

- Adott egy pszeudokód, amely két véges paraméterrel rendelkezik (x és y) és kiírja az x^y értékét. Az algoritmusnak működnie kell bármely y egész számra és minden valós x értékre. Ha az elméleti matematikai eredmény definiálatlan vagy végtelen érték, akkor egy hibaüzenet kiírása szükséges. Ha y egy nem egész szám, hibaüzenet akkor is elvárt.

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \boldsymbol{x}^{\boldsymbol{y}}$$
$$x \in \mathbb{R} \setminus \{\infty\}$$
$$y \in \mathbb{Z} \setminus \{\infty\}$$
$$f(x, y) \in \mathbb{R} \setminus \{\infty\}$$

A program kódolása és tesztelése

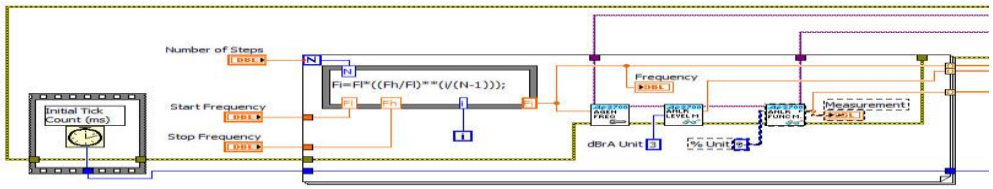
Forráskód létrehozása
egy valós programnyelven

A programozás szintjei

Magas szintű programnyelvek

Alacsony szintű programnyelvek

…



4GL SELECT name FROM people WHERE age=20

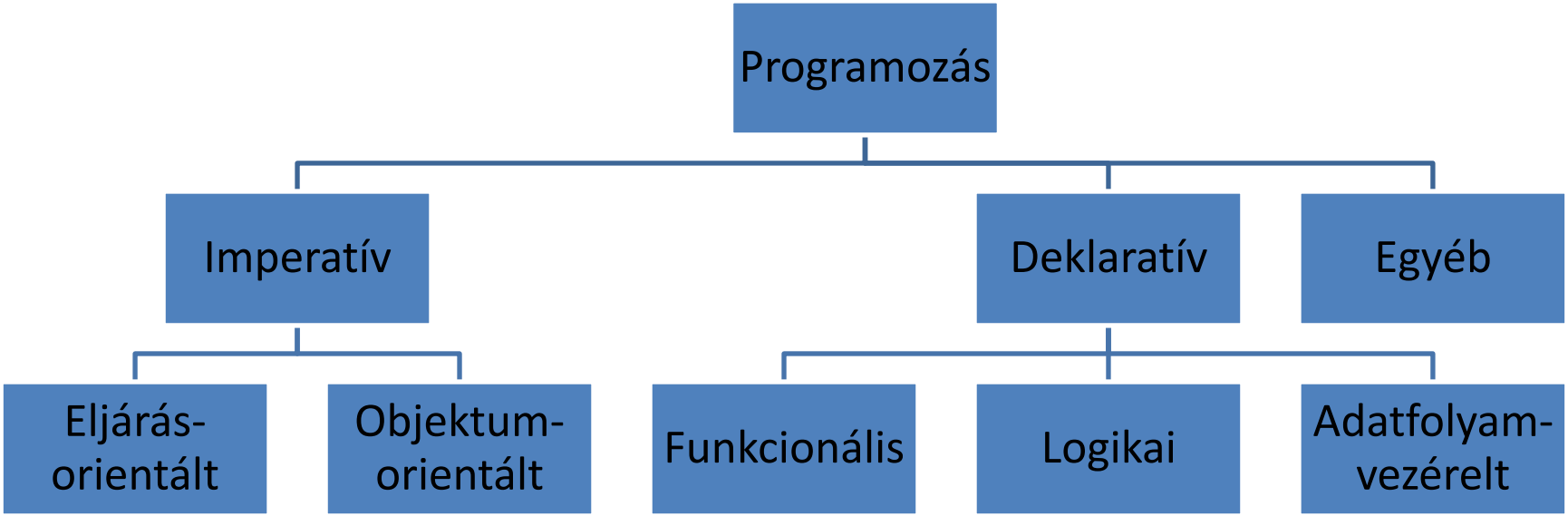
3GL if (x==0) printf("zero");

2GL mov eax, DWORD PTR [rbp-4] assembly

1GL 10001011 01000101 11111100 gépi kód

idő

Programozási paradigmák



Szintaxis és szemantika

Szintaxis: A program szövegének formai szabályai.

Szemantika: A nyelvi elemek és az algoritmus értelmezése, jelentése.

Példa (előjel függvény):

```
function sign (a)
    b=a
    if a>0 then
       b=-1*a
    enidf
    return a/b
endfunction
```

Szemantikai hiba (`a<0`)

Szintaktikai hiba (`endif`)

Futásidejű hiba (ha `/0`)

Programozási nyelvek szintaxisa

Fortran:

```
      REAL FUNCTION FACT(N)
      FACT=1
      IF (N .EQ. 0 .OR. N .EQ. 1) RETURN
      DO 20 K=2,N
20    FACT=FACT*K
      RETURN
      END
```

Python:

```
def Fact(N):
    f=1
    for i in range(1,N+1):
        f*=i
    return f
```

Pascal:

```
FUNCTION FACT(N:INTEGER):REAL;
BEGIN
  IF N=0 THEN FACT:=1
  ELSE FACT:=FACT(N-1)*N;
END;
```

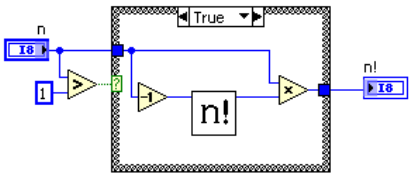
C:

```
int Fact(unsigned N){
  return N<=1?1:N*Fact(N-1);
  }
```

APL:

```
FACT←{×/⍳⍵}
```

LabView:



Feladat: szintaxis és szemantika

- Találj szintaktikai és szemantikai hibákat a következő pszeudokódban, amely az `E` nem negatív egész kitevőre emeli a `B` számot!

```
input B
R=0
wihle E<=0
  R=R*B
  E-1=E
endo
output R
```

Fordító és interpreter

- A processzorok csak a gépi kódot értik meg
  - A magas szintű kódokat át kell alakítani
- A nyelvek különböző technikákat használnak
  - Fordítóprogram
    Pl.: Pascal, C, C++, Labview
  - Interpreter
    Pl.: PHP, JavaScript, Python
  - Kombinált (virtuális gép bájtkód)
    Pl.: Java, C#

Fordítóprogram

- Egy szoftver, amely a **forráskód**ból egy **tárgykód**ot generál
- A fordítóprogram lexikai, szintaktikai, szemantikai ellenőrzést, valamint kódgenerálást végez
  - A forráskódnak hibátlannak kell lennie
- Egy **kapcsolatszerkesztő** program készít futtatható állományt a tárgykódból, majd a **betöltő** tölti be a RAM-ba futtatáshoz
- Fordítsd egyszer, futtasd később többször!
  - A fordítás és a futtatás külön történik
- A futtatás gyors

Interpreter

- Közvetlen végrehajtás
  - Analízis és kódgenerálás futási időben
- Nincs tárgykód
- Utasítások értelmezése egyesével
  - Egyedüli utasítások sorozata a bemenet
- Szintaktikailag helytelen kód is futtatható lehet
  - Hibák rejtve maradnak
- Minden futtatáshoz kell az interpreter
  - Az értelmezés és futtatás összetartozik
- A végrehajtás gyakran lassú

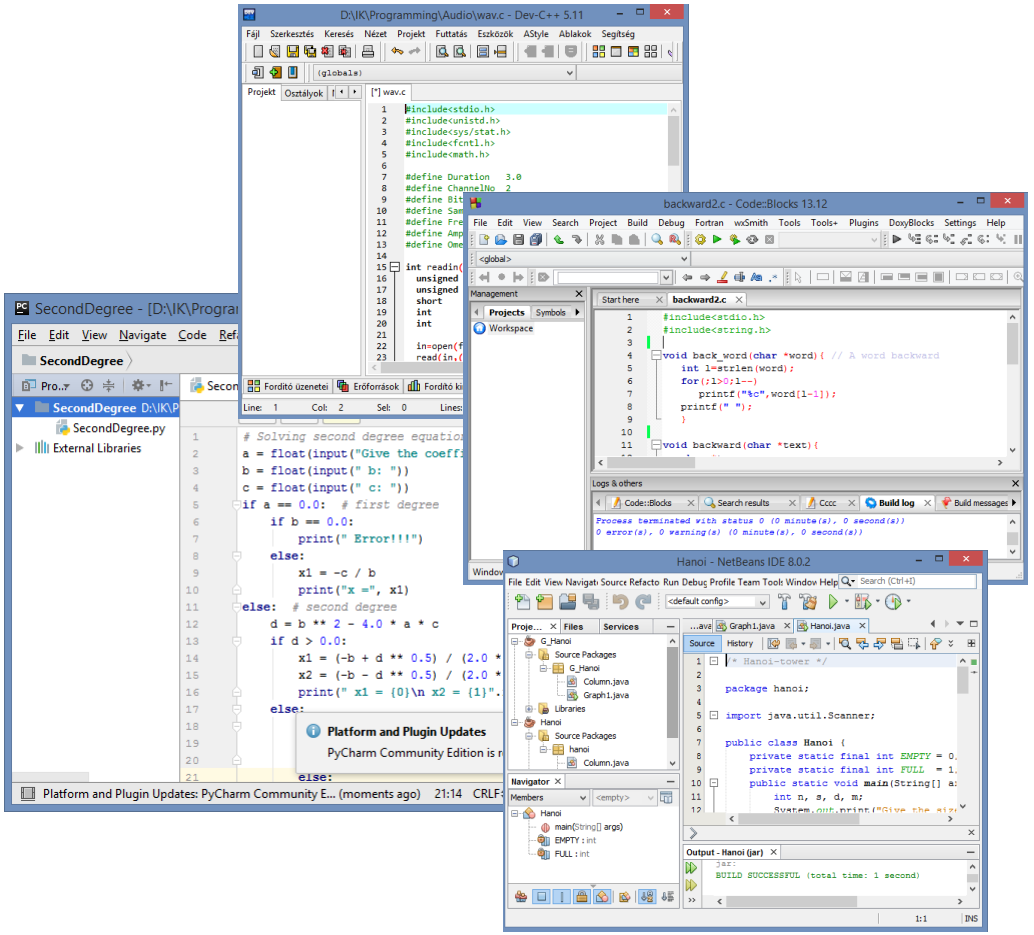
Integrált fejlesztői környezet

- Egy (grafikus) program, ami egyszerűvé és gyorssá teszi a szoftverfejlesztést, eszközöket és segítséget nyújt a programozónak
- Az **IDE** (Integrated Development Environment) tartalmaz:
  - nyelvérzékeny szövegszerkesztőt
  - fordítót/interpretert
  - nyomkövető rendszert
  - verziókezelő eszközöket
  - projektkezelési lehetőséget
  - szimulátort

Integrált fejlesztői környezet

Gyakran használt IDE-k:

- Code::Blocks
- Dev-C++
- NetBeans
- Eclipse
- PyCharm
- MS Visual Studio
- Jbuilder
- MPLAB



Adatreprezentáció, adattípusok

- A memóriában minden adat binárisan van tárolva.
- Különböző adattípusokat használunk
  - eltérő tárolási mód
  - eltérő adattartomány
  - eltérő műveletek
- A leggyakrabban alkalmazott típusok:
  - egész (5)
    00000000000000000000000000000101
  - valós (5.)
    01000000001010000000000000000000000000000000000000000000000000000
  - karakter (’5’)
    00110101
  - sztring ("5")
    0011010100000000

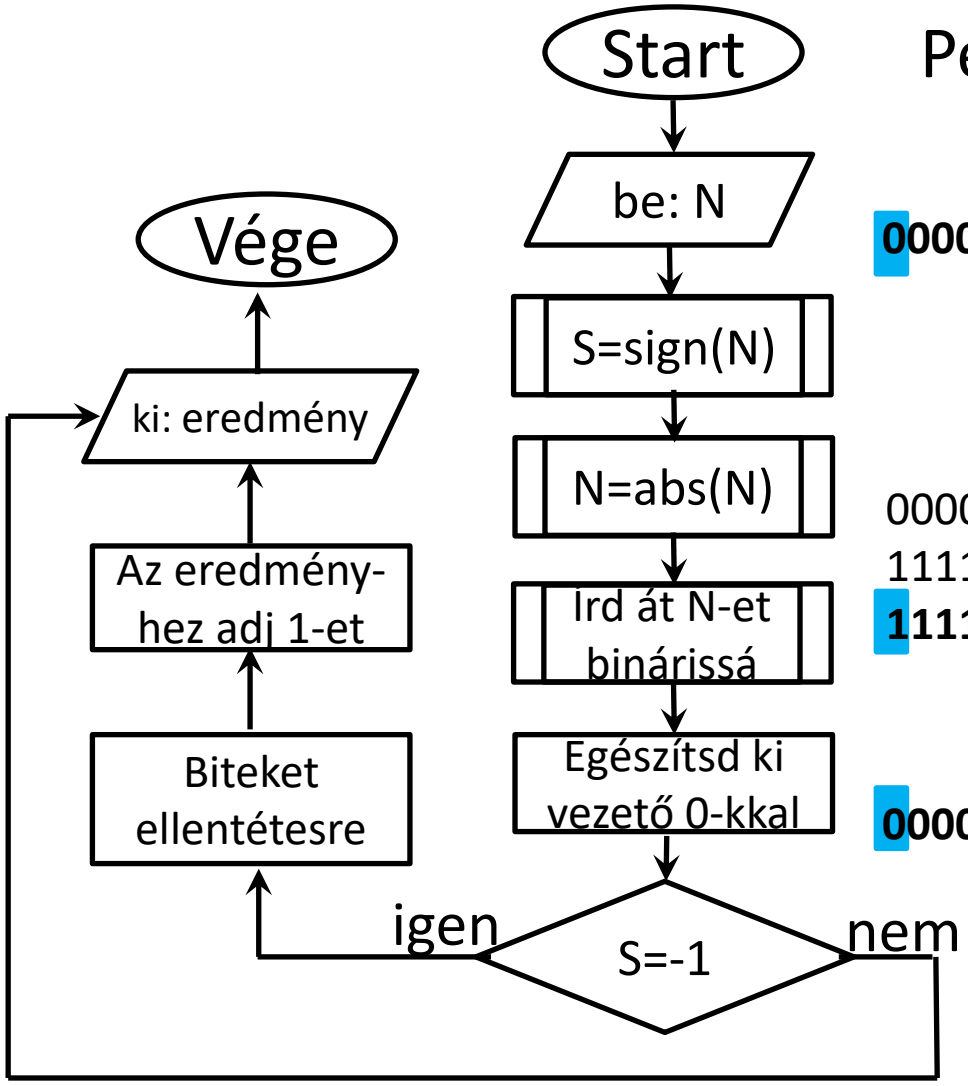
Fixpontos reprezentáció

Hogyan tárolja a számítógép az (előjeles) **egész**eket?

Lépések (az algoritmus):

- Jegyezd meg a szám előjelét, és az abszolút értékét váltsd át kettes számrendszerbe!
- Írj elé vezető nullákat (ha szükséges), hogy elérd a kívánt helyiértéket (bitet)!
- Ha az előjel negatív (a sign függvény -1 értéket ad), akkor
  - minden bitet állíts át az ellentétére,
  - adj hozzá az eredményhez 1-et (binárisan)!
- Ez a szám fixpontos reprezentációja.

Fixpontos reprezentáció



Példa

+195
11000011
00000000 00000000 00000000 11000011

-195
+195
11000011
00000000 00000000 00000000 11000011
11111111 11111111 11111111 00111100
11111111 11111111 11111111 00111101

0
0
00000000 00000000 00000000 00000000

MSB: előjelbit

Fixpontos reprezentáció

	Reprezentáció hossz (bitszám)	Minimum érték	Maximum érték
előjel nélküli	1 bájt	0	255
	2 bájt	0	65 535
	4 bájt	0	4 294 967 295
előjeles	1 bájt	-128	127
	2 bájt	-32 768	32 767
	4 bájt	-2 147 483 648	2 147 483 647

Feladat: adatreprezentáció

- 32 bites fixpontos reprezentációval add meg a Föld lakosságának (hozzávetőleges) számát!
- Add meg a -1 értékét 32 bites fixpontos reprezentációval!
- Melyik 4 bájtos, fixpontos bitsorozat jelenti: 15908?
- Melyik 4 bájtos, fixpontos bitsorozat jelenti: -666?
- Mi az értéke az alábbi bitsorozatnak 4 bájtos, fixpontos számábrázolás esetén?

```
10000000 00000000 00000010 01001001
```

A forráskód egységei és elemei

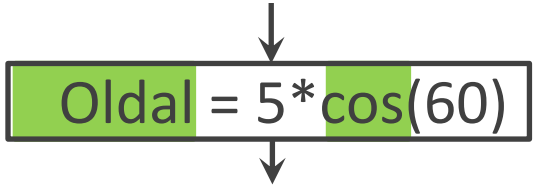
- Karakterkészlet
- Lexikai egység
- Szintaktikai egység
- Utasítás
- Programegység
- Fordítási egység
- Program

Minden nyelvben különböző karaktereket, szimbólumokat, speciális kulcsszavakat, kifejezéseket és szabályokat használunk.

Kulcsszó, azonosító, megjegyzés

A példák C nyelven vannak.

- Kulcsszó
  Speciális jelentésű karaktersorozat
  Pl.: `if, else, while, for, return`
- Azonosító
  Karaktersorozat, amely egy programozói eszköz neveként szerepel
  Pl.: `i, Count, var2, abs_val`



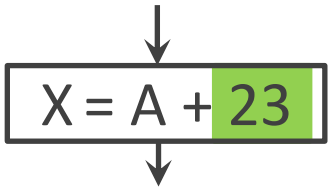
- Megjegyzés
  Olyan szöveg a forráskódban, amely csak a programozónak (olvasónak) szól, nincs hatása a program futásra

Konstans

- A konstans (literál) egy fix érték a programban, amely futásidőben nem változtatható.
- Értéke és típusa van, amelyeket a felírás módja határoz meg.
- Speciális: nevesített konstans, fix érték azonosítóval.
- Példák

```
-12.34, 5, .5, 5., 0x1F, 'F', "F",
"alma"
```

```
while x>100 do
```



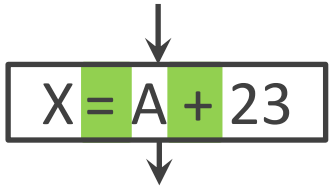
Változó

- Egy memóriarekesz adattárolásra, amelyre az azonosítóval hivatkozhatunk
- Az eljárásorientált nyelvek legfontosabb eszköze
- Komponensei
  - név (azonosító)
  - érték (bitsorozat a RAM-ban)
  - attribútum (típus)
  - cím (RAM-beli pozíció)
- Példa (C nyelven)

```
int A = 10;
float Ugras1 = 11.5;
```

Operátorok

- Adatokon végzett egyszerű műveleteket jelentenek
- Lehetnek egy-, kettő- vagy háromoperandusúak
- Általános operátor csoportok:
  - aritmetikai (pl.: `+, -, *, /, %`)
  - hasonlító (pl.: `>, <, ==, >=, <=, !=`)
  - logikai (pl.: `&&, ||, !`)
  - bitenkénti (pl.: `&, |, ^, ~, <<, >>`)
  - értékadó (pl.: `=, +=, *=`)
  - egyéb (pl.: `*, &, ? :, ., ->`)



Kifejezés

- Operátorok & operandusok & zárójelek
- Operandus lehet: konstans, nevesített konstans, változó, függvényhívás
- A kifejezésnek értéke és típusa van (kiértékelés)
- Az alak lehet:
  - infix (a precedencia/erősség fontos)
    például: `4 + 3 * 2`
  - prefix
    például: `+ * 3 2 4`
  - postfix
    például: `4 3 2 * +`

Feladatok: kifejezés

- Mi a következő infix kifejezés értéke?
  `9+2*6/3>8-7`
- Mi a következő infix kifejezés értéke (C nyelv esetén)?
  `2>3&&3*5-6/2>=11%2`
- Mi a következő prefix kifejezések értéke?
  `* + 1 2 - 9 6`
  `+ 1 - * 2 13 / 25 5`
- Mi a következő postfix kifejezések értéke? Alakítsd át őket infix alakúakká!
  `30 2 15 4 6 + - * /`
  `1 2 13 * 25 5 / - +`

Utasítások

Csoportosítás

- Deklaráció
- Értékadás
- Feltételes utasítás
  - kétirányú elágazás
  - többirányú elágazás
- Iteráció
  - feltételes ciklus
  - előírt lépésszámú ciklus
- Egyéb

nem futtatható

futtatható

Deklaráció, értékadás

Deklaráció:

- Azonosító és típus összerendelése.
- Memóriafoglalás, kezdőértékadás (néha).
- `int i = 0;`
- `float Suly;`

Értékadás:

- Változóhoz érték rendelése.
- `i = 6;`
- `Suly = 80.3*i;`

Feltételes utasítás

Választás két tevékenység közül.

- Két külön utasításssorozat.
- Az egyik utasításssorozat lehet üres.
- ```
  if(N<0.0)   S=1;
  else        S=0;
  ```

Választás több tevékenység közül.

- ```
  switch (i){
     case 1:  X=1;  break;
     case 2:  X=10; break;
     default: X=100;
  }
  ```

Iteráció

Utasítások, tevékenységek többszöri ismétlése.

Működési szempontból (szemantikailag) határesetek:

- üres ciklus (a mag egyszer sem hajtódik vége)
- végtelen ciklus (soha nem áll meg, szemantikai hiba)

Iteráció típusok (szintaktikailag)

- feltételes ciklus
  - kezdőfeltételes
  - végfeltételes
- előírt lépésszámú
- egyéb (végtelen, összetett)

Kezdőfeltételes ciklus

A **fej** tartalmazz egy **feltételt**

Szemantika:

1. A feltétel kiértékelése.
2. Ha igaz, a **mag** (törzs) végrehajtása és ismét kiértékelés (1.)
   Különben a ciklus véget, és folytatás a ciklus utáni utasítással.

Lehet üres ciklus is,
ha a feltétel kezdetben hamis.

A mag végrehajtása hatással lehet a feltételre. →

```
while (x<2) {
    i=i+1;
    x=x-2;
}
```

Végfeltételes ciklus

A **vég** tartalmazza a **feltételt**.

Szemantika:

1. A **mag** végrehajtása egyszer
2. A feltétel kiértékelése
3. Ha ez igaz (hamis), hajtsd végre a **mag**ot újra (1.)
   Különben a ciklus véget ér, és folytatás a ciklus utáni utasítással

Nem lehet üres ciklus,
a mag legalább egyszer végrehajtódik.

```
do{
   i=i+1;
   x=x-2;
  }
while (x<2);
```

Pszeudokód → Python

Pszeudokód:

```
input a
if a>0 then
  b=a
else
  b=-1*a
endif
while b!=0 do
  b=b-1
enddo
output b
```

Python:

```
a = int(input())
if a > 0:
    b = a
else:
    b = -1*a

while b != 0:
    b = b-1

print(b)
```

A Python programozási nyelv

```
# Solving second degree equations
a = float(input("Give the coefficients\n a: "))
b = float(input(" b: "))
c = float(input(" c: "))
if a == 0.0:  # first degree
    if b == 0.0:
        print(" Error!!!")
    else:
        x1 = -c / b
        print("x =", x1)
else:  # second degree
    d = b ** 2 - 4.0 * a * c
    if d > 0.0:
        x1 = (-b + d ** 0.5) / (2.0 * a)
        x2 = (-b - d ** 0.5) / (2.0 * a)
        print(" x1 = {0}\n x2 = {1}".format(x1, x2))
    else:
        if d == 0.0:
            x1 = -b / (2.0 * a)
            print(" x =", x1)
        else:
            print(" Error!!!")
```

Feladat: Python

A következő Python kódrészletben találd meg az alábbi fogalmak előfordulásait!

- kulcsszó
- megjegyzés
- azonosító
- konstans
- változó
- operátor
- kifejezés
- utasítás

```
# Valami számolás
Sum=0
for i in range(N):
    Sum+=i
if(Sum==0):
    print("összeg"+Sum)
else:
    z=10%2+N/N+cos(90)
#return z
```

Ajánlott irodalom

- Simon Harris, James Ross: *Kezdőkönyv az algoritmusokról*, (Szak Kiadó, 2006)
- Gérard Swinnen: *Tanuljunk meg programozni Python nyelven*, (O'Reilly, 2005)
- Narasimha Karumanchi: *Data Structures and Algorithmic Thinking with Python*, (CareerMonk, 2017)
- Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers: *How to Think Like a Computer Scientist: Learning with Python 3,* (online, 2012)
- Metrowerks CodeWarrior: *Principles of Programing* (online, 1995)

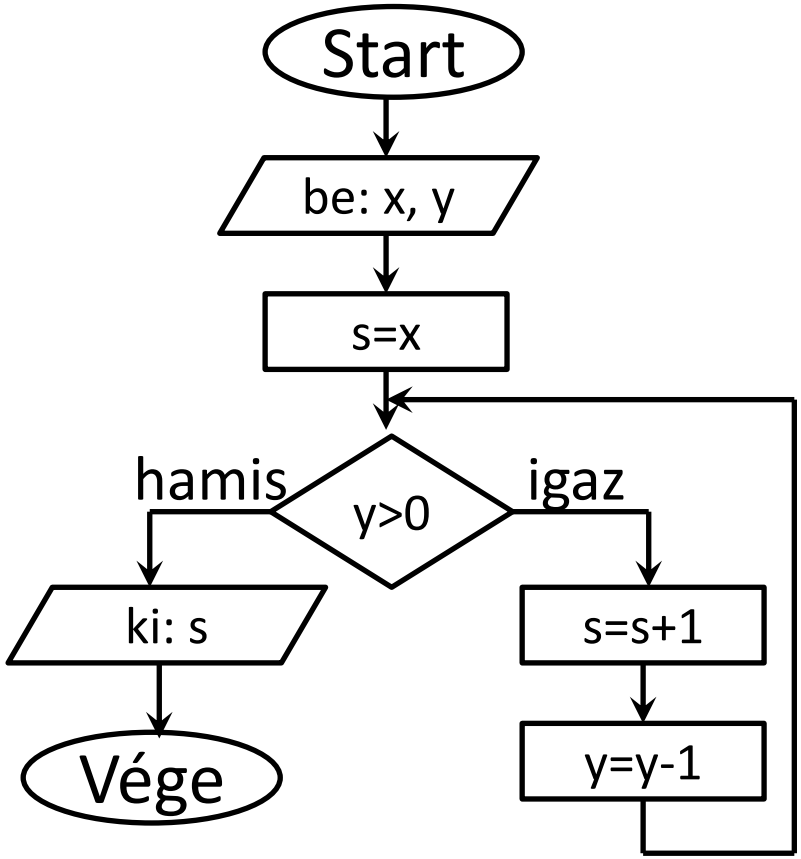
Megoldások

Vissza a feladathoz

Megoldás: folyamatábra

- Nyomkövetés:

x	y	s
5	4	?
5	4	5
5	3	6
5	2	7
5	1	8
5	0	**9**



- Kimenet: 9
- 5 kifejezés kiértékelés
- Összeadás: s=x+y
- Módosítások:

  s=s+1 → s=s+x s=x → s=0

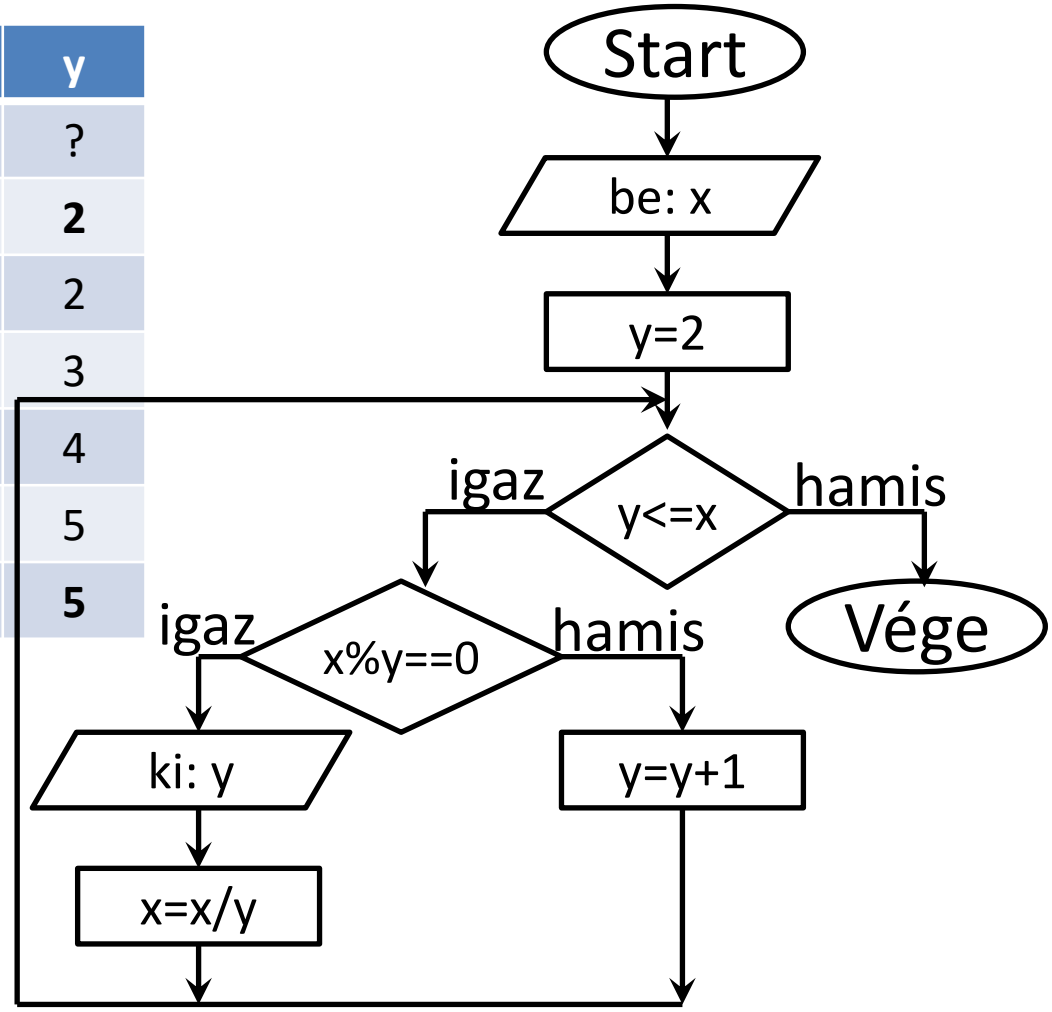
  s=s+1 → s=s+x y>0 → y>1

  s=s+1 → s=s+x Out: s → Out: s-x

Megoldás: folyamatábra

- Nyomkövetés:

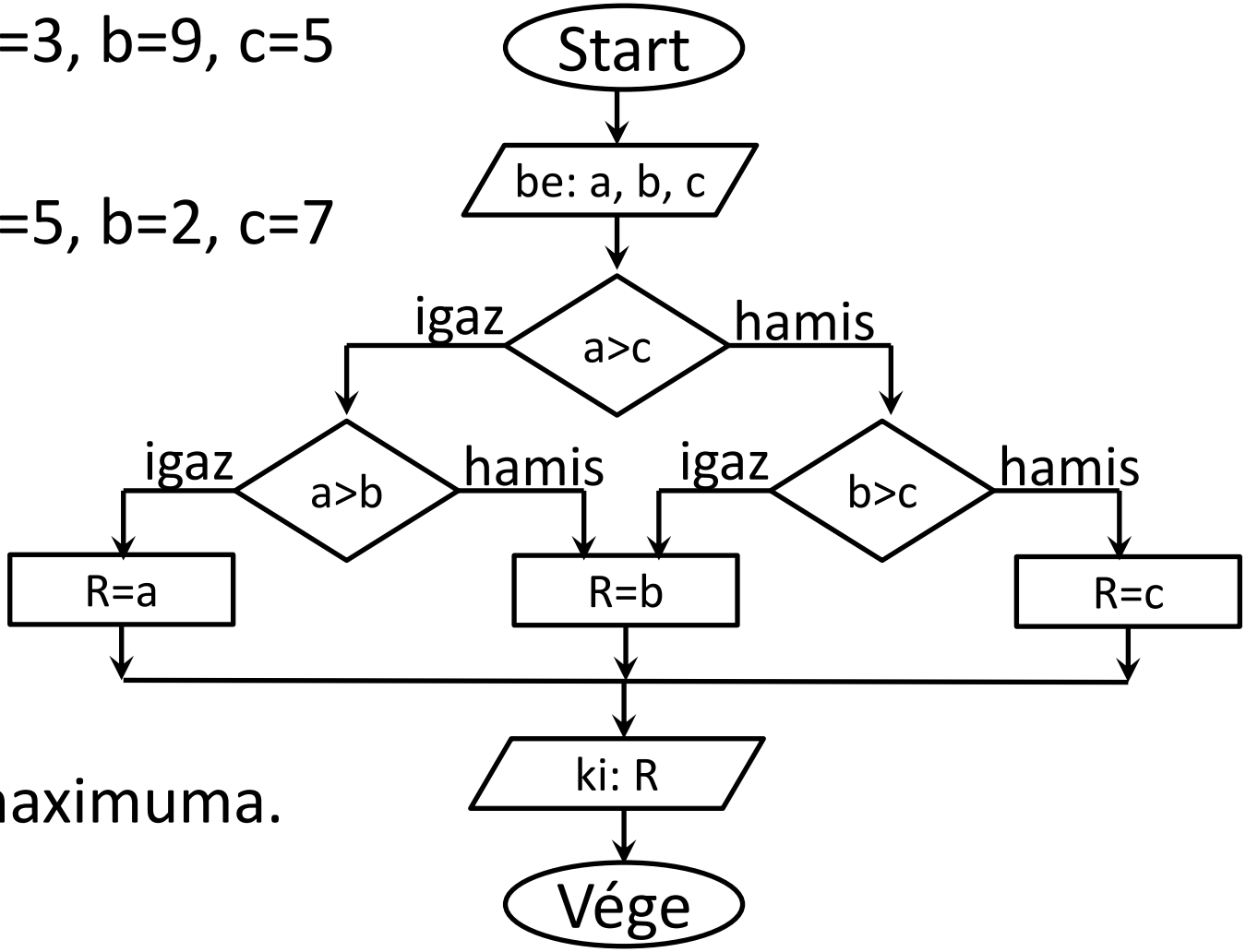
x	y
10	?
10	**2**
5	2
5	3
5	4
5	5
1	**5**



- Kimenet a 60 esetén:
  2 2 3 5
- Prímtényezőkre bontás
- Nincs kimenet
- Ismétlésszám: 5

Megoldás: folyamatábra

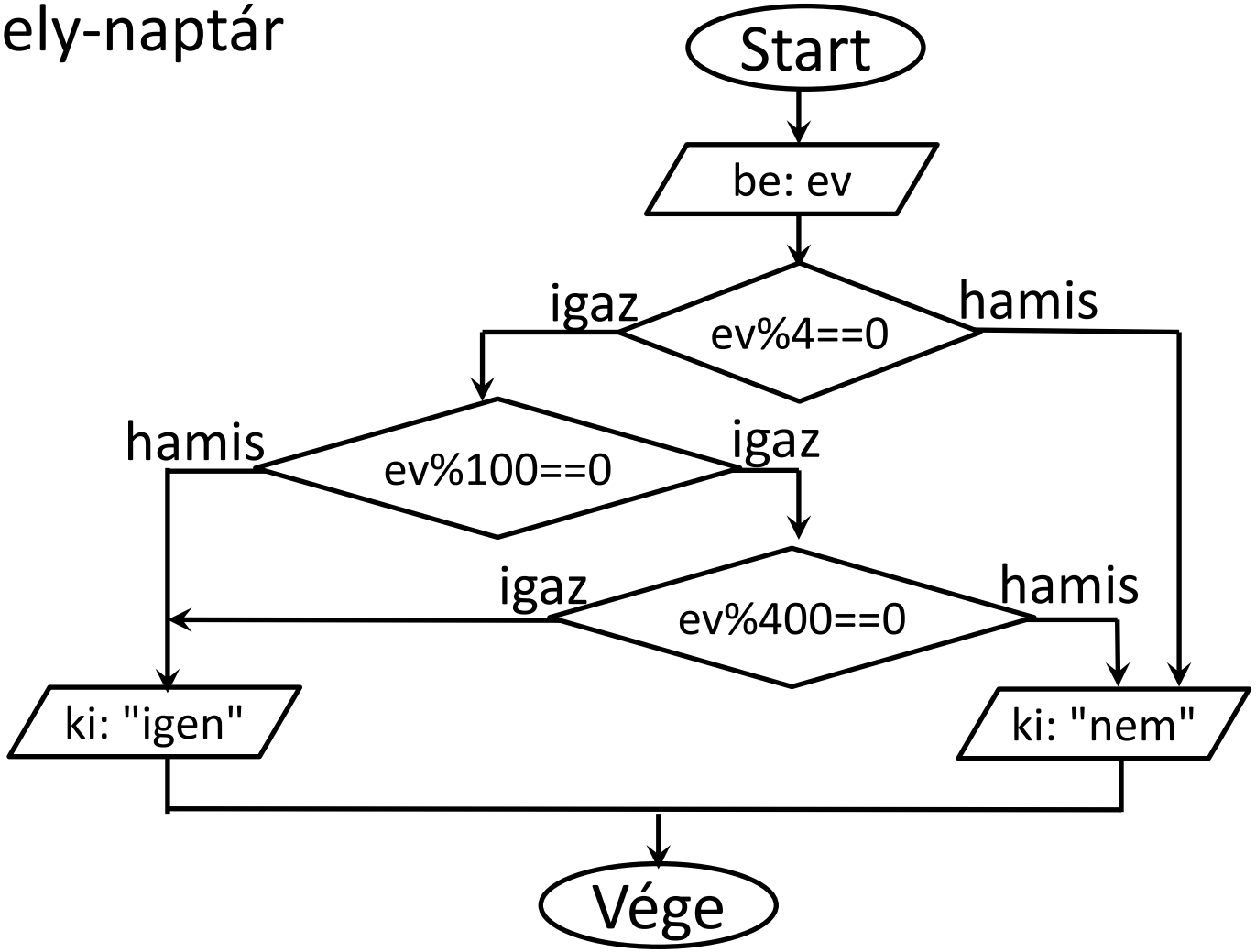
- Bemenet: a=3, b=9, c=5
  Kimenet: 9
- Bemenet: a=5, b=2, c=7
  Kimenet: 7



- A számok maximuma.

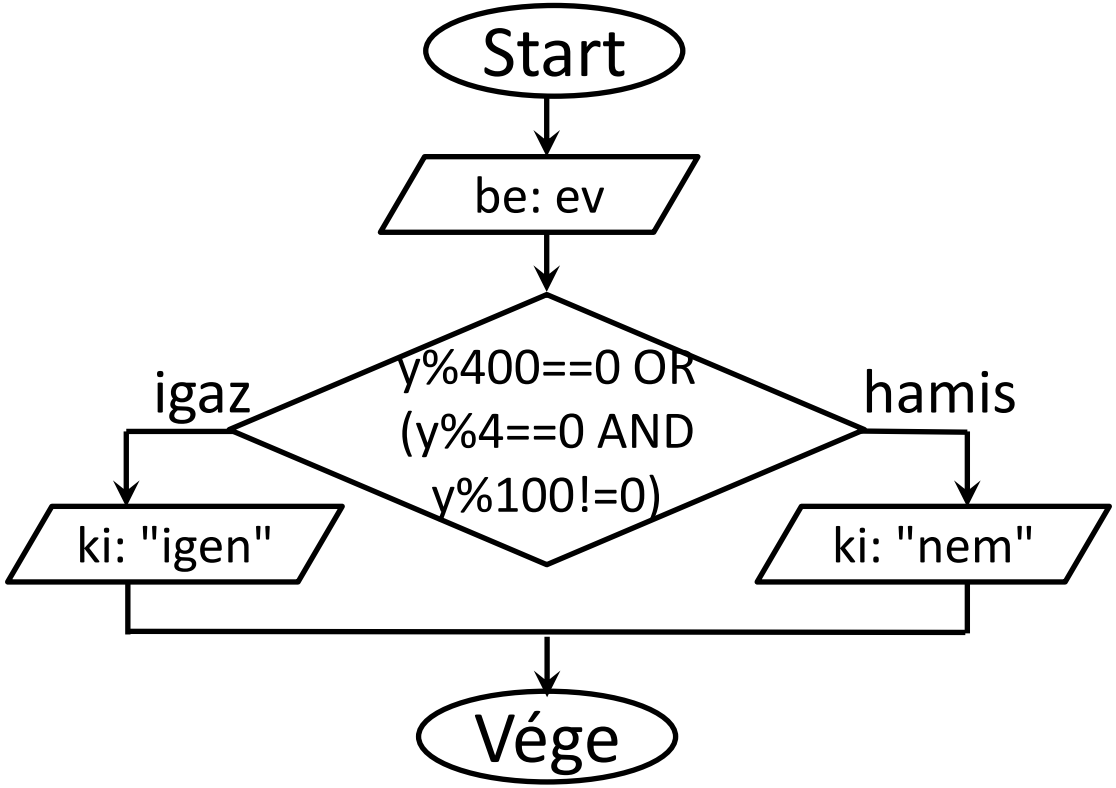
Megoldás: szökőév 1

- Gergely-naptár



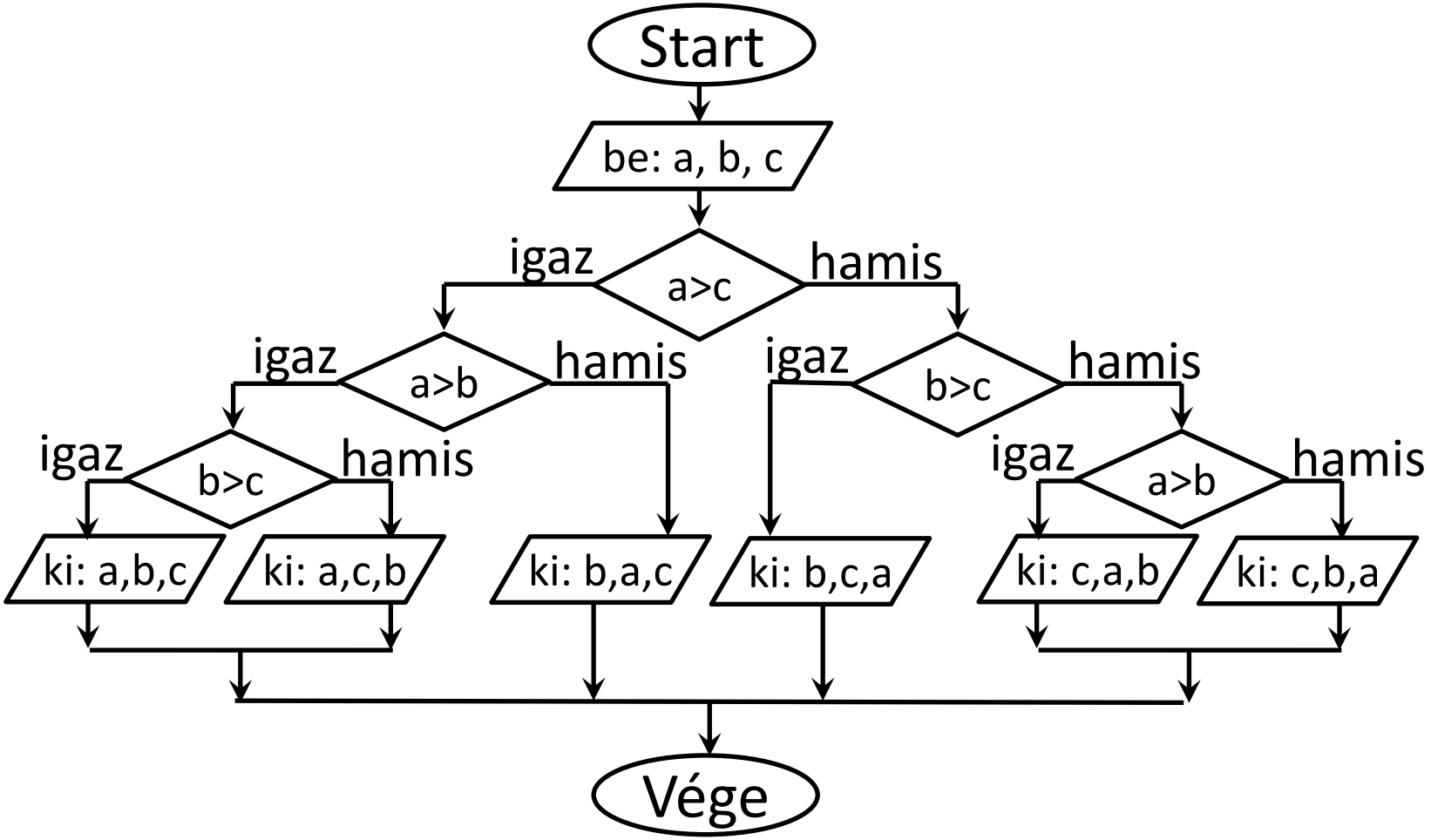
Megoldás: szökőév 2

- Gergely-naptár

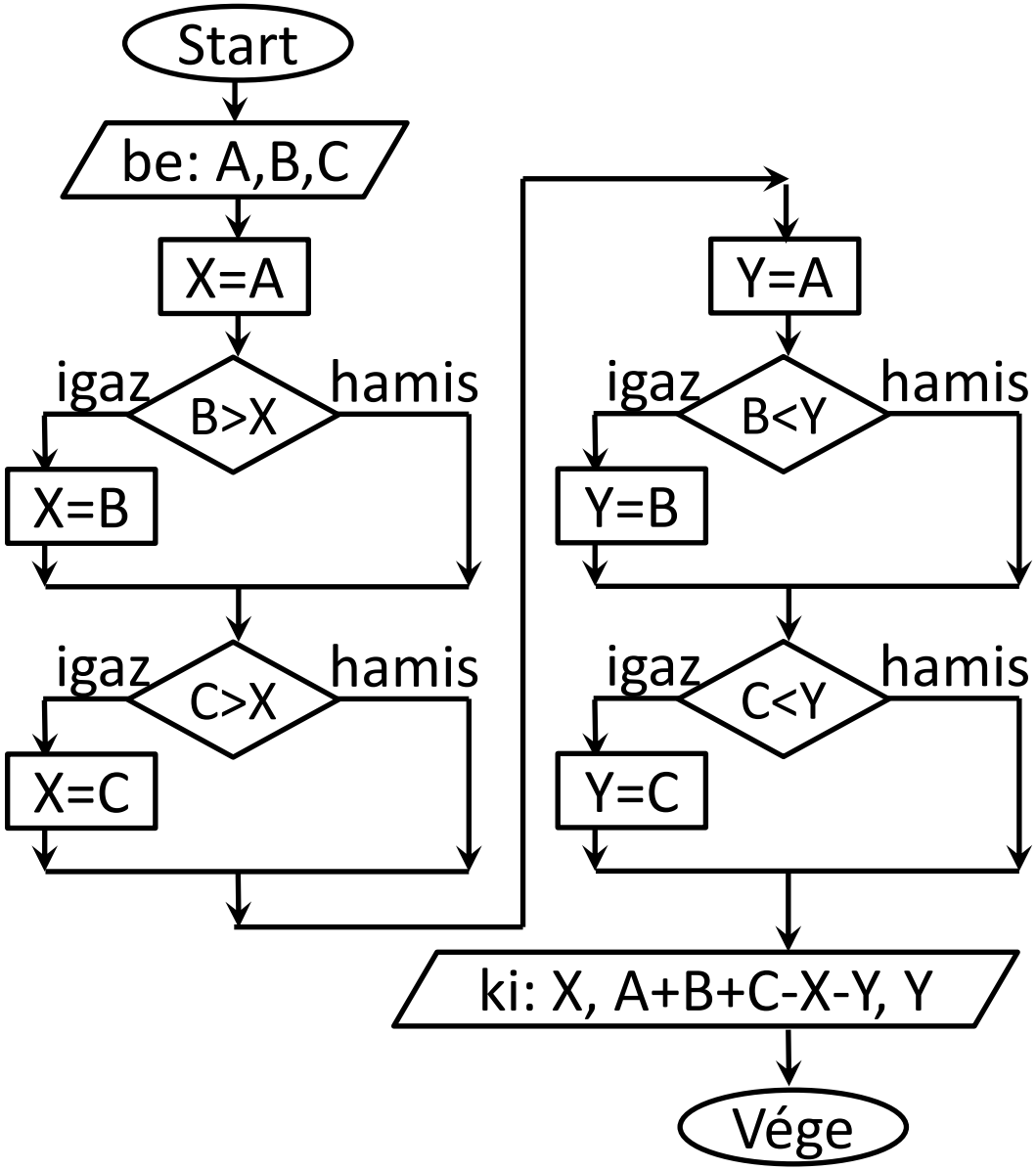


Megoldás: három érték 1

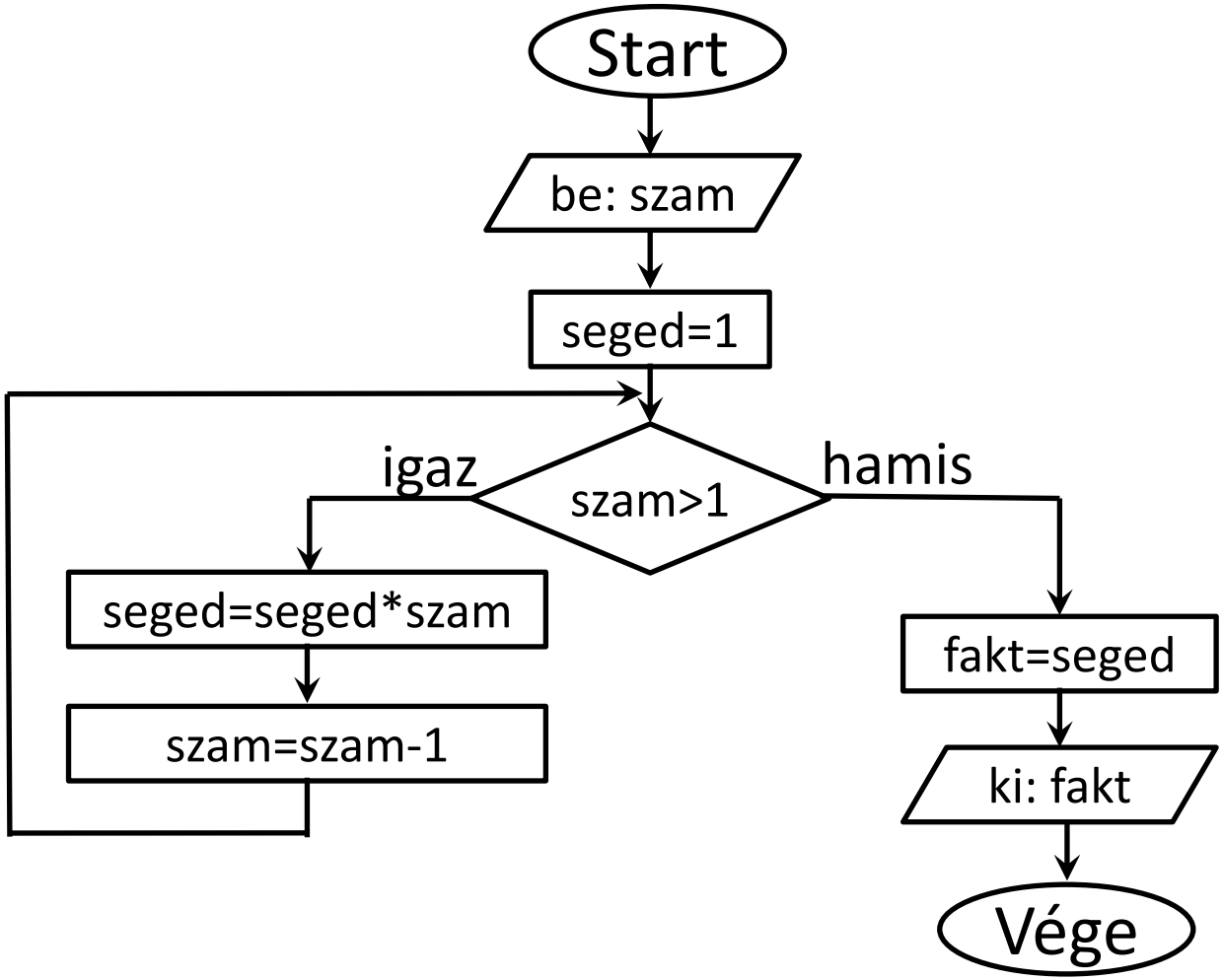
- 3 érték csökkenő sorrendben



Megoldás: három érték 2

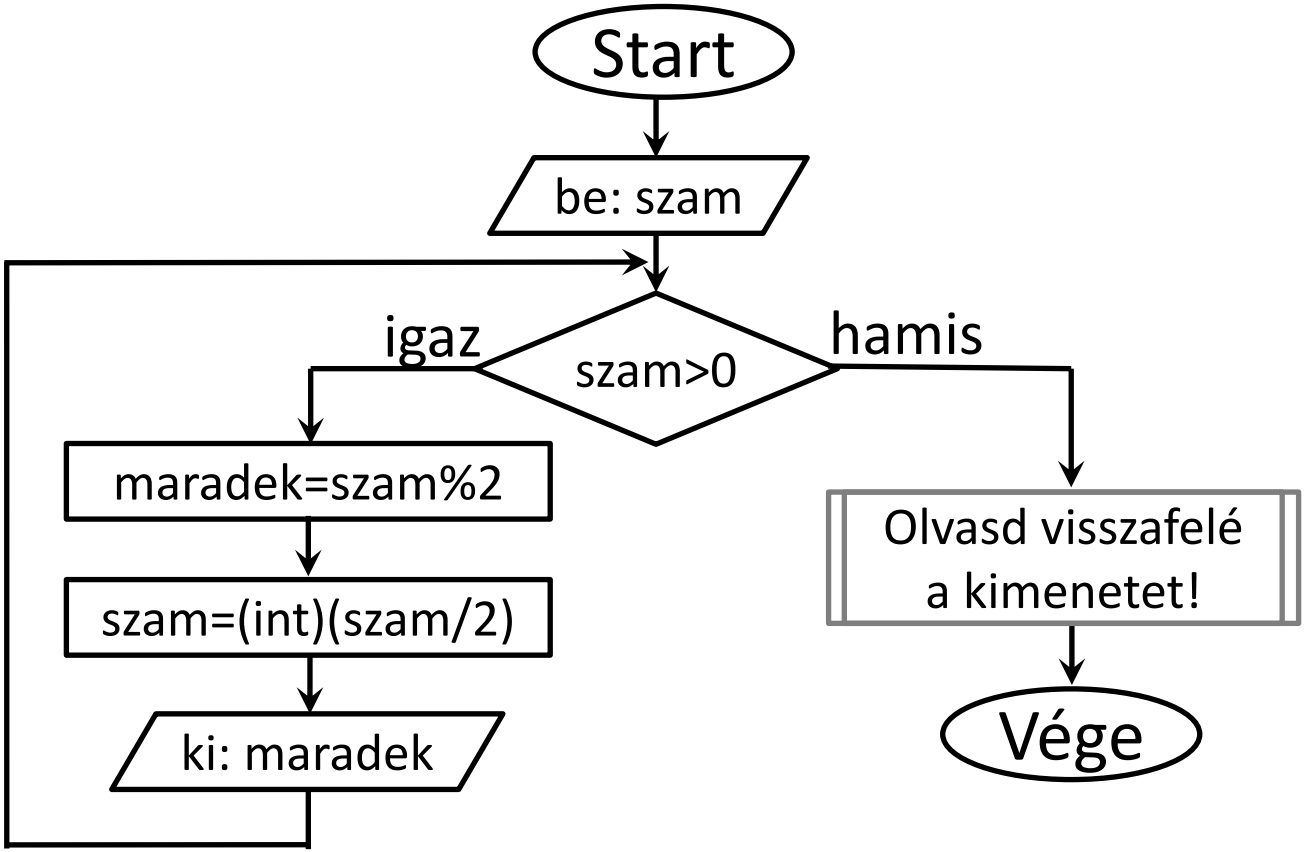


Megoldás: faktoriális



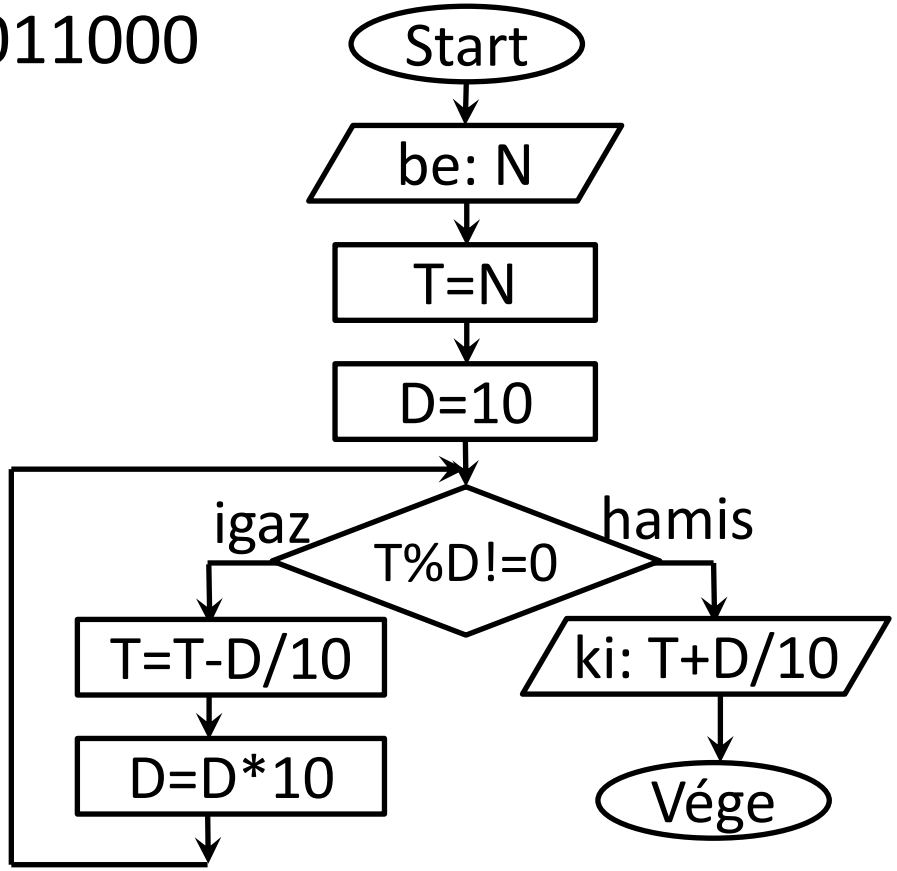
Megoldás: bináris szám

- Számrendszerváltás (10 → 2)
  Pl.: $21_{10} = 10101_2$



Megoldás: folyamatábra

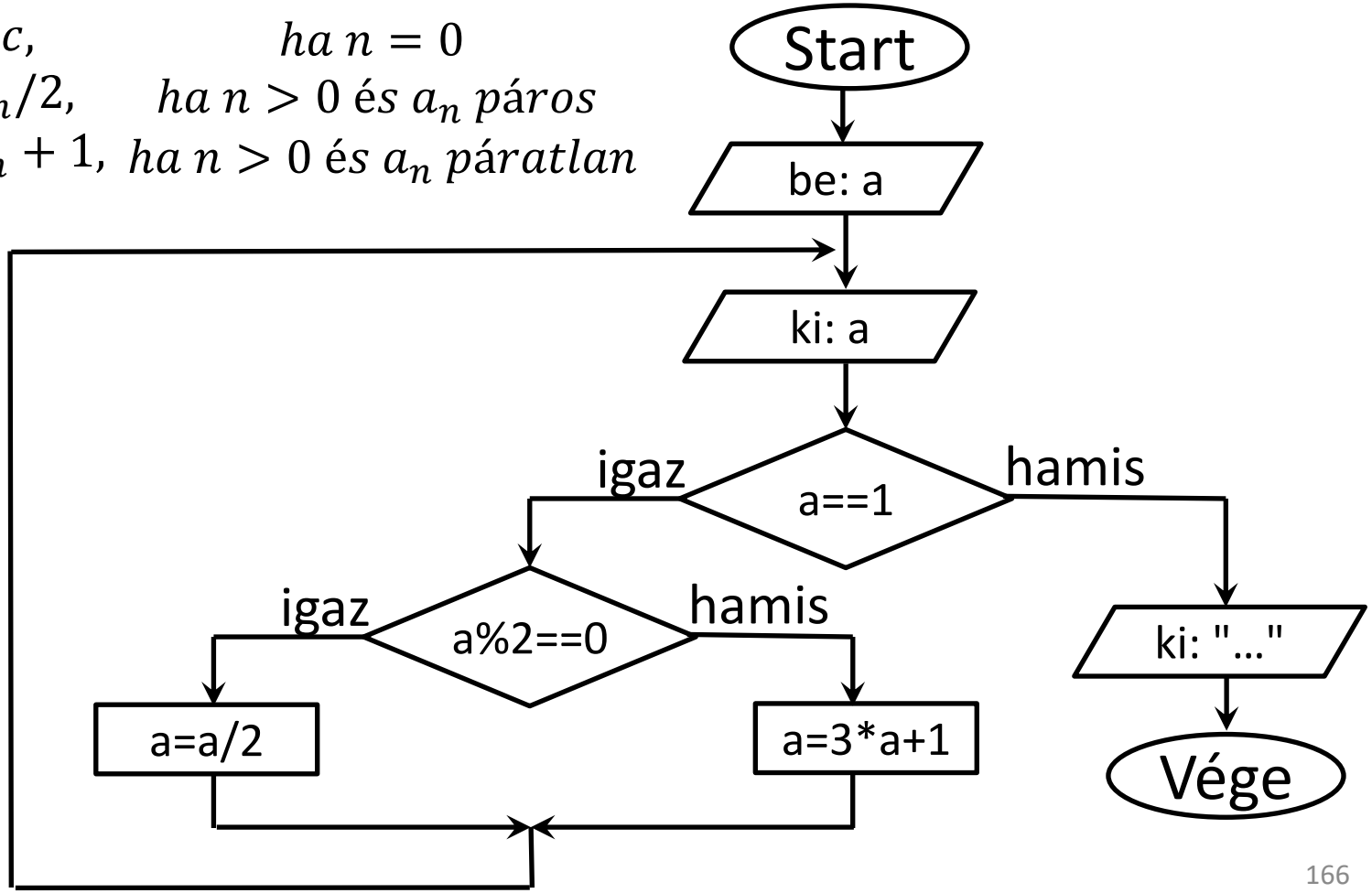
- Bináris szám inkrementálása
  Pl.: 1010111 → 1011000



Megoldás: folyamatábra

- Collatz-sejtés

$$a_{n+1} = \begin{cases} c, & ha\ n = 0 \\ a_n/2, & ha\ n > 0\ és\ a_n\ páros \\ 3a_n + 1, & ha\ n > 0\ és\ a_n\ páratlan \end{cases}$$



Megoldás: pszeudokód 1

```
input a
if a<0 then
   b=-1*a
else
   b=a
endif
output b
```

- Bemenet: 10; Kimenet: *10*
- Bemenet: -4; Kimenet: *4*
- Egy szám abszolút értéke

- Egy szám abszolút értéke

```
input a
if a<0 then
   a=-1*a
endif
output a
```

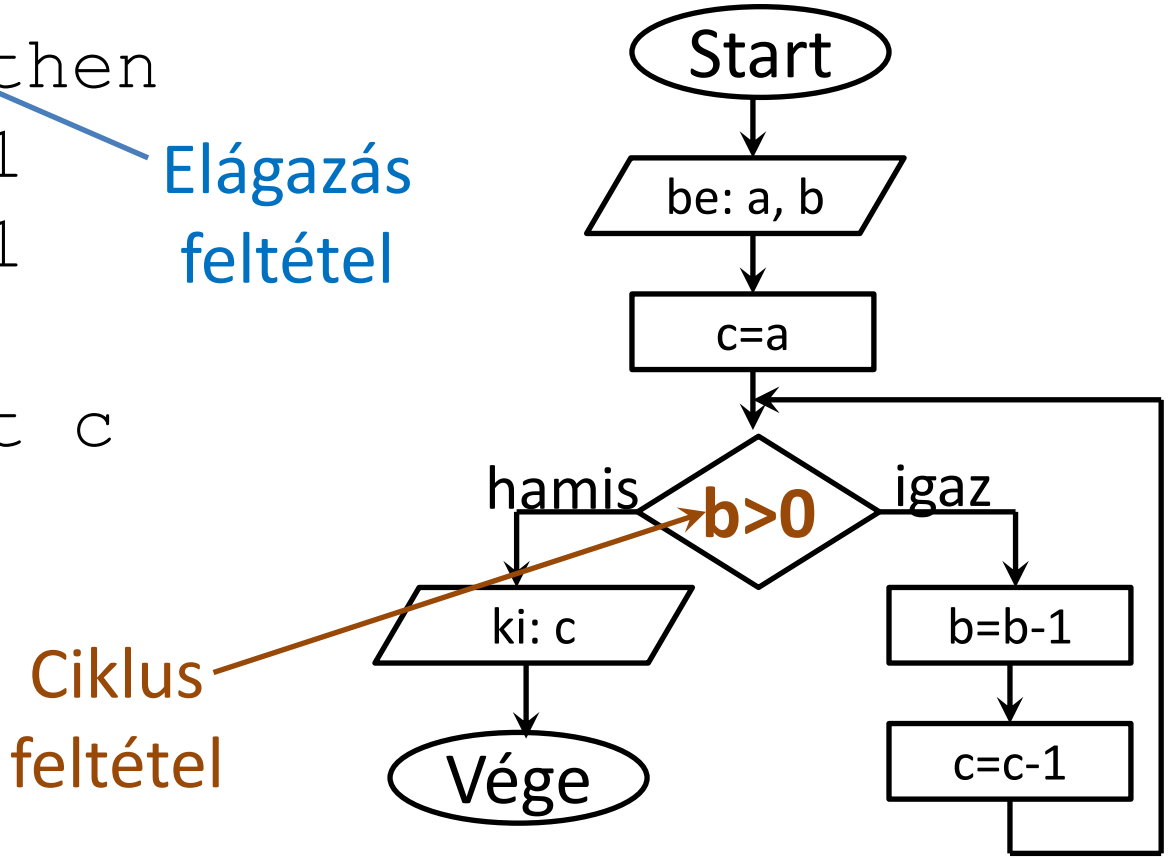
Alternatív algoritmusok!

Megoldás: Azonosak?

```
input a
input b
c=a
if b>0 then
    b=b-1
    c=c-1
else
   output c
endif
```

- Nem ugyanaz az algoritmus (nem alternatívák)

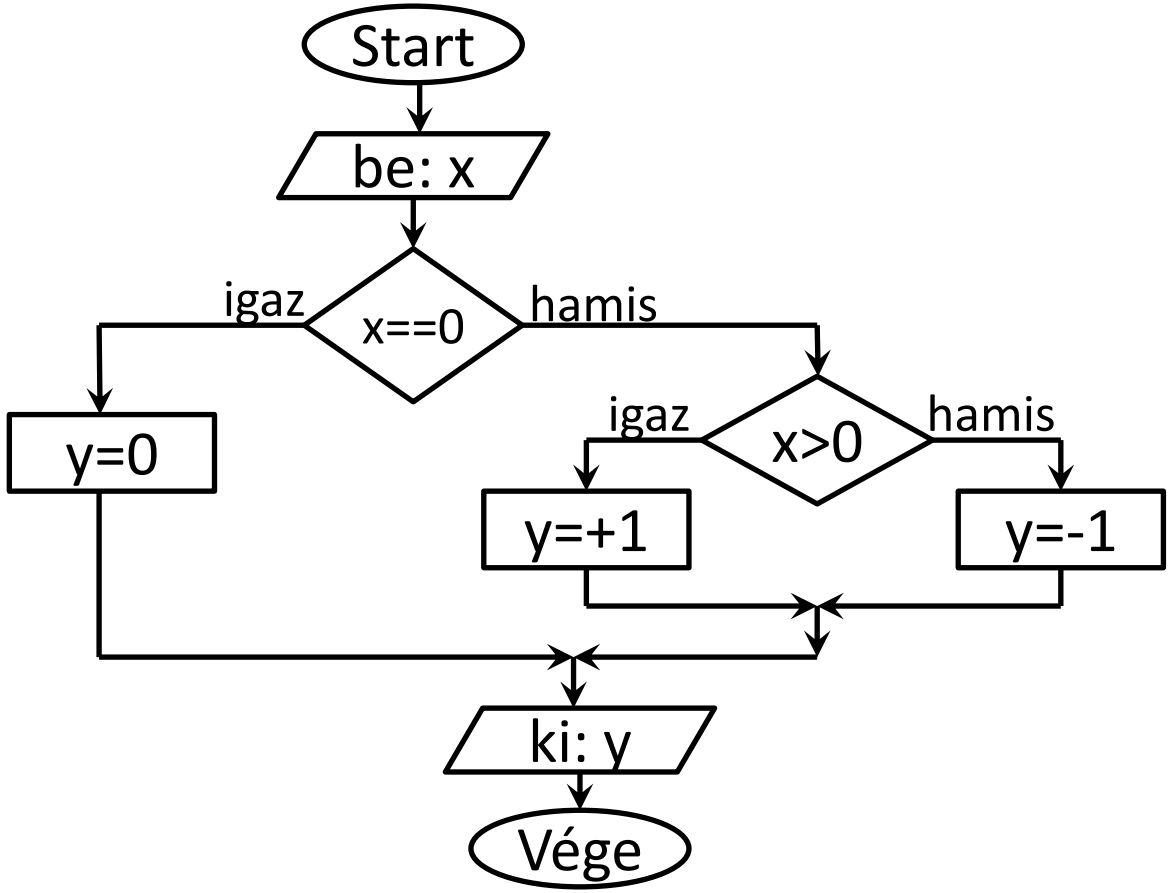
Elágazás feltétel



Megoldás: konverzió 1

Előjel függvény:

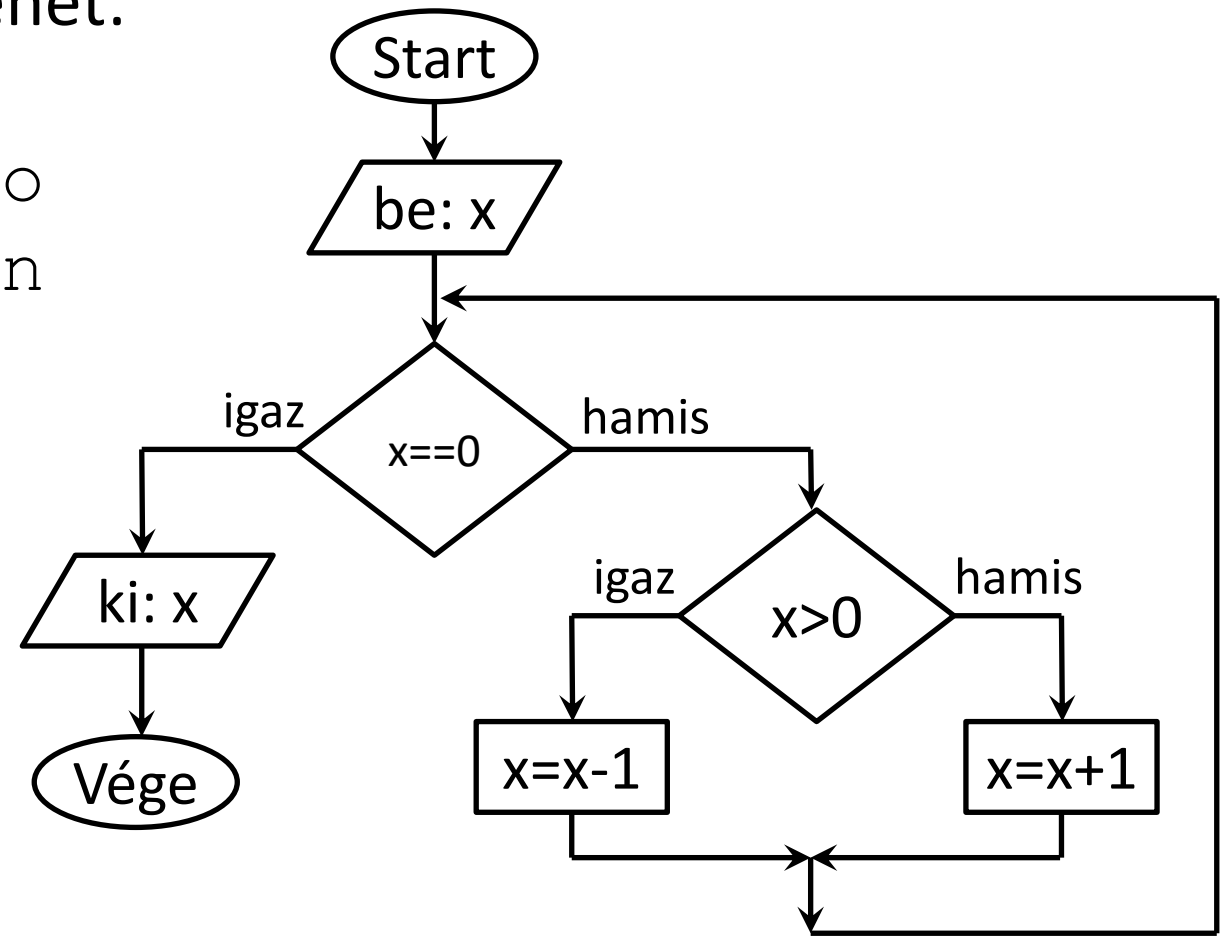
```
input x
if x==0 then
   y=0
else
   if x>0 the
      y=+1
   else
      y=-1
   endif
endif
output y
```



Megoldás: konverzió 3

Mindig nulla kimenet:

```
input x
while x!=0 do
   if x>0 then
      x=x-1
   else
      x=x+1
   endif
enddo
output x
```



Megoldás: pszeudokód 2

```
input a
input b
c=a
while b>0 do
    b=b-1
    c=c-1
enddo
output c
```

- Nyomkövetés:

a	b	c
7	3	?
7	3	7
7	2	6
7	1	5
7	0	**4**

- Bemenet: a=7, b=3; Kimenet: 4
- 3 ismétlés
- 4 feltétel kiértékelés
- Különbség számítás
  c=a-b

Megoldás: pszeudokód 3

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=(int)(N/10)
enddo
output R
```

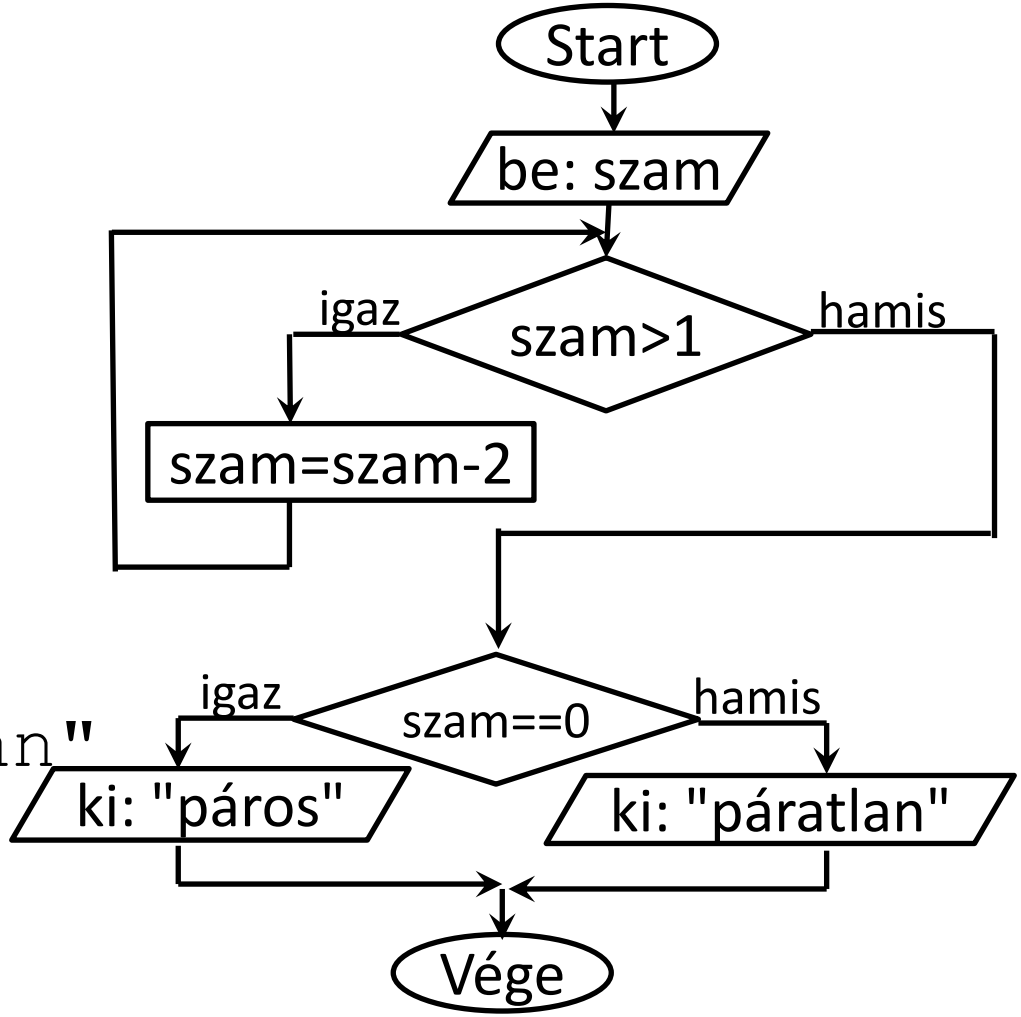
- Nyomkövetés:

N	R
73251	0
7325	1
732	15
73	152
7	1523
0	**15237**

- Kimenet: 15237
- Megfordítja egy szám számjegyeinek a sorrendjét.

Megoldás: páros vagy páratlan

```
input szam
while szam>1 do
   szam=szam-2
enddo
if szam==0 then
   output "páros"
else
   output "páratlan"
endif
```



Megoldás: elágazások

```
input a,b,c
if a<b then
  if a<c then
    output a
  else
    output c
  endif
else
  if b<c then
    output b
  else
    output c
  endif
endif
```

```
input a,b,c
min=a
if b<min then
  min=b
endif
if c<min then
  min=c
endif
output min
```

```
input a
input b
input c
if a<b AND a<c then
  output a
else
  if b<c then
    output b
  else
    output c
  endif
endif
```

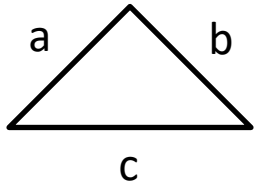
```
input a,b,c
if a<b then
  d=a
else
  d=b
endif
if c<d then
  output c
else
  output d
endif
```

```
input a,b,c
if a<=b AND a<=c then
  output a
  stop
endif
if b<=a AND b<=c then
  output b
  stop
endif
if c<=b AND c<=a then
  output c
endif
```

```
input a,b,c
if c<a AND c<b then
  output c
  stop
endif
if b<a then
  output b
  stop
endif
output a
```

3 szám minimuma: 6 különböző megoldás

Megoldás: elágazások



- Háromszög-egyenlőtlenség

```
input a, b, c
if a<b+c AND b<c+a AND c<a+b then
 output "Szerkeszthető háromszög"
else
 output "Nem szerkeszthető háromszög"
endif
```

Megoldás: iterációk

```
input A
input K
if K==0 AND A==0 then
  output "Határozatlan!"
  stop
endif
if K<0 then
  if A!=0 then
    A=1/A
  else
    output "Végtelen!"
    stop
  endif
  K=K*(-1)
endif
H=1
while K>0 do
  H=H*A
  K=K-1
enddo
output H
```

Hatványozás ($H=A^K$)

Általános megoldás:

$$f(x,y,z) = x^{y/z}$$

Megoldás: iterációk

```
output "Base"
input B
output "Exponent numerator"
input EN
if EN!=0 then
  output "Exponent
denominator"
  input ED
else
  ED=1
endif

if B==0 AND EN==0 then
  output "Undefined!"
  stop
endif

if ED<0 then
  ED=-1*ED
  EN=-1*EN
endif

if EN<0 then
  if B!=0 then
    EN=-1*EN
    B=1/B
  else
    output "Intinity!"
    stop
  endif
endif
```

```
T1=EN
T2=ED
while T2>0 do
  T3=T2
  T2=T1%T2
  T1=T3
enddo
EN=EN/T1
ED=ED/T1

P=1
TE=EN
while TE>0 do
  P=P*B
  TE=TE-1
enddo

if ED==1 do
  output P
  stop
endif

if P<0 AND ED%2==0 then
  output "No real root!"
  stop
endif

if P<0 AND ED%2==0 then
  output "No real root!"
  stop
endif
```

```
R=0
D=1
if P<0 then
  S=-1*P
else
  S=P
endif
Th=0.0000001
while S>Th do
  A=1
  TE=ED
  while TE>0 do
    A=A*R
    TE=TE-1
  enddo
  F=0
  if D==1 AND A>P then
    F=1
  endif
  if D==-1 AND A<P then
    F=1
  endif
  if F==1 do
    S=S/2.0
    D=-1*D
  endif
  R=R+D*S
enddo
output R
```

Megoldás: négyzetgyök

```
output „Mi a szám?"
input Num
output „Mi a pontosság?"
input Threshold
Old=Num
Diff=Num
while Diff>Threshold do
  New=Old-(Old*Old-Num)/(2*Old)
  Diff=Old-New
  Old=New
enddo
output „Négyzetgyök:" Old
```

Megoldás: prímek

```
input N
S=2
while S<N do
   if N%S==0 then
      output "nem"
      stop
   endif
   S=S+1
enddo
output "igen"
```

Kész, vége!

Megoldás: sorozatok

- Fibonacci-sorozat első 100 eleme

```
F1=0
F2=1
output F1, F2
N=2
while N<=100 do
   F3=F1+F2
   output F3, " "
   F1=F2
   F2=F3
   N=N+1
enddo
```

```
F1=0
F2=1
output F1, F2
N=2
while N<=100 do
   F1=F1+F2
   F2=F1+F2
   output F1, " "
   output F2, " "
   N=N+2
enddo
```

Megoldás: sorozatok

- Fibonacci-sorozat 1000-nél kisebb elemei

```
F1=0
F2=1
output F1, F2
F3=F1+F2
while F3<1000 do
   output F3, " "
   F1=F2
   F2=F3
   F3=F1+F2
enddo
```

$$f_i = \begin{cases} 0, & ha\ i = 1, \\ 1, & ha\ i = 2, \\ f_{i-1} + f_{i-2} & egyébként \end{cases}$$

Megoldás: szökőnapok

```
input y1, y2
n=0
while y1<y2 do
   if y1%4==0 then
     n=n+1
   endif
   y1=y1+1
enddo
output n
```

```
input y1, y2
if y1%4!=0 then
   y1=y1+4-y1%4
endif
if y2%4==0 then
   y2=y2-4
else
   y2=y2-y2%4
endif
output (y2-y1)/4+1
```

Hatékonyság?

Megoldás: keresés

- Elemkeresés tömbben

```
N=10
T[]={3,5,2,-7,0,34,5,3,567,9}
input Keresett
i=0
while i<N AND T[i]!=Keresett do
  i=i+1
enddo
if i<N then
  output "benne van, az indexe:", i
else
  output "nincs benne"
endif
```

Megoldás: keresés

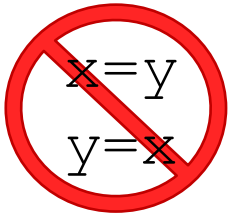
- Tömbelemek maximuma és annak indexe

```
N=10
T[]={3,5,2,-7,0,34,5,3,567,9}
max_i=0
i=1
while i<N do
   if T[i]>T[max_i] then
     max_i=i
   endif
   i=i+1
enddo
output max_i, T[max_i]
```

Megoldás: rendezés

- Változótartalmak megcserélése (x=5; y=8 → x=8; y=5)

```
z=x
x=y
y=z
```



- Alternatív megoldás, köztes átmeneti változó nélkül

```
x=x+y
y=x-y
x=x-y
```

Megoldás: rendezés

- Buborékrendezés

```
N=10
T[]={3,5,2,-7,0,34,5,3,567,9}
veg=N-1
while veg>0 do
  i=0
  while i<veg do
    if T[i]>T[i+1] then
      x=T[i]
      T[i]=T[i+1]
      T[i+1]=x
    endif
    i=i+1
  enddo
  veg=veg-1
enddo
```

Lásd még:
https://hu.wikipedia.org/wiki/Buborékrendezés

Megoldás: bankkártya száma

```
a[]={1,3,4,2,6,5,7,8,9,7,6,5,1,0,5,7}
```

```
i = 0
while i<=15 do
 a[i] = 2*a[i]
 if a[i]>=10 then
  a[i] = a[i]-9
 endif
 i = i+2
enddo
i = 0
s = 0
while i<=15 do
 s = s+a[i]
 i = i+1
enddo
if s%10==0 then
 output "valid"
else
 output "invalid"
endif
```

```
i = 0
s = 0
while i<=15 do
 s=s+(2-i%2)*a[i]
 s=s-(int)(a[i]/5)*(1-i%2)*9
enddo
if s%10!=0 then
 output "not "
endif
output "valid"
```

Luhn algoritmus

Megoldás: leggyakoribb kor

```
N=15
Age[]={1,3,2,0,5,10,17,3,10,0,3,2,15,1,3}
Distr[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
i=0
while i<N do
  Distr[Age[i]]=Distr[Age[i]]+1
  i=i+1
enddo
freq=0
i=1
while i<18 do
  if Distr[i]>Distr[freq] then
    freq=i
  endif
  i=i+1
enddo
output freq, " éves gyerekből van a legtöbb. "
```

Megoldás: best-fit allokáció

```
N=20
M[]={1,0,5,5,5,5,5,0,0,4,4,4,4,0,1,0,4,4,4,4}
input S
i=0
pos=0
while i<N do
  if M[i]==S then
    pos=i
    break
  endif
  if M[i]>S AND M[i]<M[pos] then
    pos=i
  endif
  i=i+1
enddo
if pos>0 OR M[pos]>=S then
  output pos
endif
```

Megoldás: mintaillesztés

```
N=11
A[]={2,3,9,4,1,8,3,9,4,8,6}
M=4
B[]={3,9,4,8}
iA=0
iB=0
while iA<N AND iB<M do
  if A[iA]==B[iB] then
    iB=iB+1
    iA=iA+1
  else
    iA=iA-iB+1
    iB=0
  endif
enddo
if iB==M then
  output "Pattern found."
else
  output "Not found."
endif
```

Megoldás: alprogram

```
function PP( a )
   b=0
   while b<a do
      a = a-1
      b = b+1
   enddo
   if a==b then
      return 1
   else
      return 0
   endif
endfunction
```

```
input a, b
a = a*2
b = PP(a+b)+1
output b
```

a	b	a	b
1	4	-	-
2	4	-	-
2	4	6	
2	4	6	0
2	4	5	1
2	4	4	2
2	4	3	3
2	2	-	-

- A kimenet: 2
- Páros paraméter esetén 1-t ad, páratlan esetben 0-et.

Megoldás: alprogram

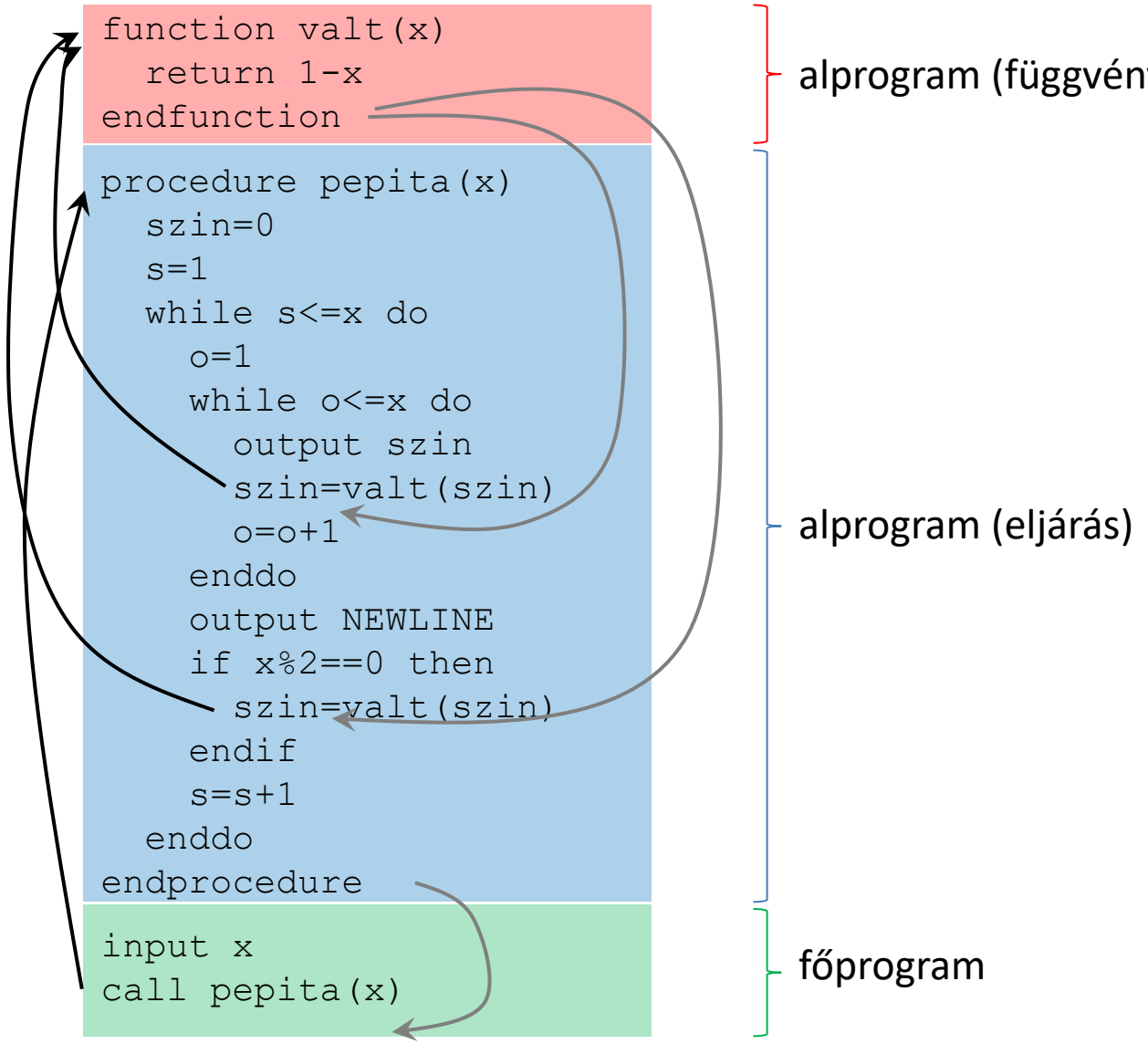
```
function VALTOZTAT( a )
   return 1-a
endfunction
input Max
i=0
j=0
while j<Max do
   i = VALTOZTAT(i)
   j=j+i
   output j
enddo
output j
```

- Max=5 → 1 1 2 2 3 3 4 4 5 5
- Természetes számokat duplán ír ki
- A 0→1 és 1→0 váltogatás

Megoldás: szorzótábla

```
procedure SzT(N)
   i=1
   while i<=N do
      j=1
      while j<=N do
         output i*j, " "
         j=j+1
      enddo
      output NEWLINE
      i=i+1
   enddo
endprocedure
call SzT(8)
```

Megoldás: pepita minta



Megoldás: idő

```
function sec(ora, perc, masodperc)
  return (ora*60+perc)*60+masodperc
endfunction
procedure ido(t)
  s=t%60
  m=((int)(t/60))%60
  h=(int)(t/3600)
  output h, ":", m, ":", s
endprocedure
call ido(sec(12,15,30))
```

Megoldás: DCB aritmetika

```
function get_digit(Num,Pos)
  return (int)(Num/Pos)%10
endfunction

function set_digit(Num,Pos,Bit)
  big=(int)(Num/(Pos*10))
  little=Num%Pos
  return big*Pos*10+Bit*Pos+little
endfunction

function magnitude(Num)
  Pos=10
  while Num>=Pos do
    Pos=Pos*10
  enddo
  return Pos/10
endfunction

function max(A,B)
  if A>B then
    return A
  else
    return B
  endif
endfunction

function add(A,B)
  bigpos=max(magnitude(A),magnitude(B))
  sum=0
  carry=0
  pos=1
  while pos<=bigpos do
    dA=get_digit(A,pos)
    dB=get_digit(B,pos)
    digit=dA+(dB+carry)
    if digit>1 then
      digit=digit-2
      carry=1
    else
      carry=0
    endif
    sum=set_digit(sum,pos,digit)
    pos=pos*10
  enddo
  if carry==1 then
    sum=set_digit(sum,pos,1)
  endif
  return sum
endfunction
```

```
function sub(A,B)
  bigpos=max(magnitude(A),magnitude(B))
  diff=0
  carry=0
  pos=1
  while pos<=bigpos do
    dA=get_digit(A,pos)
    dB=get_digit(B,pos)
    digit=dA-(dB+carry)
    if digit<0 then
      digit=digit+2
      carry=1
    else
      carry=0
    endif
    diff=set_digit(diff,pos,digit)
    pos=pos*10
  enddo
  if carry==1 then
    output "Negative difference"
  endif
  return diff
endfunction

function mul(A,B)
  pos=magnitude(B)
  prod=0
  while pos>=1 do
    prod=add(prod,pos*A*get_digit(B,pos))
    pos=pos/10
  enddo
  return prod
endfunction

function div(A,B)
  pos=magnitude(A)
  quotient=0
  D=(int)(A/pos)
  while pos>=1 do
    if D>=B then
      digit=1
      D=sub(D,B)
    else
      digit=0
    endif
    quotient=quotient*10+digit
    pos=pos/10
    D=D*10+get_digit(A,pos)
  enddo
  return quotient
endfunction
```

```
function pow(A,B)
  p=1
  while B>0 do
    p=mul(p,A)
    B=sub(B,1)
  enddo
  return p
endfunction

output "Enter 3 binary values: "
input X,Y,Z
output div(pow(X,Y),Z), NEWLINE
output "Oh, it was so easy, isn't it?"
```

Megoldás: tesztelési stratégia

N	B	Eredmény	OK?
11	2	1011	✓
2	5	2	✓
0	9	0	✓
43	0	-	×
-10	4	22	×
9	-2	-999	×
10	1.5	-	×
3.5	4	-	×
31	16	25	×
64	1	-	×
1024	2	1410065408	×

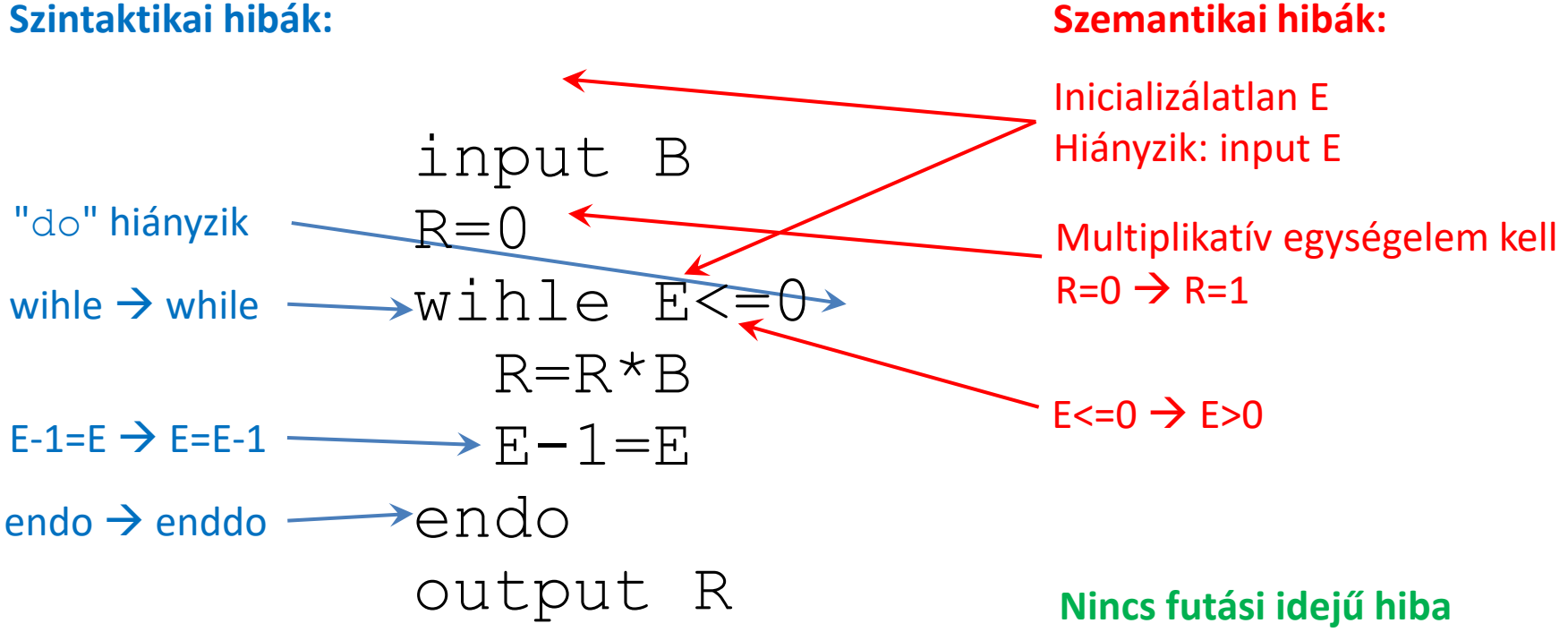
B ∈{2,3,4,5,6,7,8,9,10}
N nem negatív egész szám (N<Limit(B))

```
input N
input B
R=0
P=1
while N!=0 do
   R=R+(N%B)*P
   P=P*10
   N=(int)(N/B)
enddo
output R
```

Megoldás: tesztelési stratégia

Bemenet: x	Bemenet: y	Elvárt kimenet
2	3	8
0	3	0
2	0	1
0	0	Definiálatlan érték.
-2	3	-8
-2	4	16
2	-3	0.125
-2	-3	-0.125
0	-3	Végtelen
0.2	3	0.008
-0.2	3	-0.008
0.2	-3	125
-0.2	-3	-125
2	0.3	A kitevő nem egész szám.
100	13	11 991 163 848 716 906 297 072 721

Megoldás: szintaxis és szemantika



Megoldás: adatreprezentáció

- Népesség több, mint 7 000 000 000.
  Előjel nélküli fixpontos maximum 2^{32}-1 = 4294967295.
  Nem ábrázolható! (33bit szükséges)
- `-1:`
  `11111111 11111111 11111111 11111111`
- `15908:`
  `00000000 00000000 00111110 00100100`
- `-666:`
  `11111111 11111111 11111101 01100110`
- `10000000 00000000 00000010 01001001:`
  `-2147483063` (signed), `2147484233` (unsigned)

Megoldás: kifejezés

- `((9+((2*6)/3))>(8-7))`
  érték: igaz
- `((2>3)&&(((3*5)-(6/2))>=(11%2)))`
  érték: hamis (C nyelv: 0)
- `* + 1 2 - 9 6`
  érték: 9, azaz ((1+2)*(9-6))
  `+ 1 - * 2 13 / 25 5`
  érték: 22, azaz (1+((2*13)-(25/5)))
- `30 2 15 4 6 + - * /`
  érték: 3, azaz (30/(2*(15-(4+6))))
  `1 2 13 * 25 5 / - +`
  érték: 22, azaz (1+((2*13)-(25/5)))

Megoldás: Python

- Kulcsszó: for, in, if, else
- Megjegyzés: # Valami számolás, #return z
- Azonosító: Sum, N, print, z, cos
- Konstans: 0, 10, 2, 90
- Változó: Sum, N, z
- Operátor: =, +=, ==, %, +, /
- Kifejezés: Sum=0, Sum+=i, Sum==0, "összeg"+Sum, z=10%2+N/N+cos(90)
- Függvényhívás: cos(90)
- Utasítás: Sum=0, for …, if …else…, Sum+=1, print(…), z=10%2+N/N+cos(90)

```
# Valami számolás
Sum=0
for i in range(N):
    Sum+=i
if(Sum==0):
    print("összeg"+Sum)
else:
    z=10%2+N/N+cos(90)
#return z
```