Algoritmusok és a programozás alapjai

Dr. Varga Imre
Debreceni Egyetem, Informatikai Kar

Lektorálta: Dr. Biró Piroska, Dr. Kósa Márk, Dr. Pánovics János

2018.07.25

EFOP-3.4.3-16-2016-00021

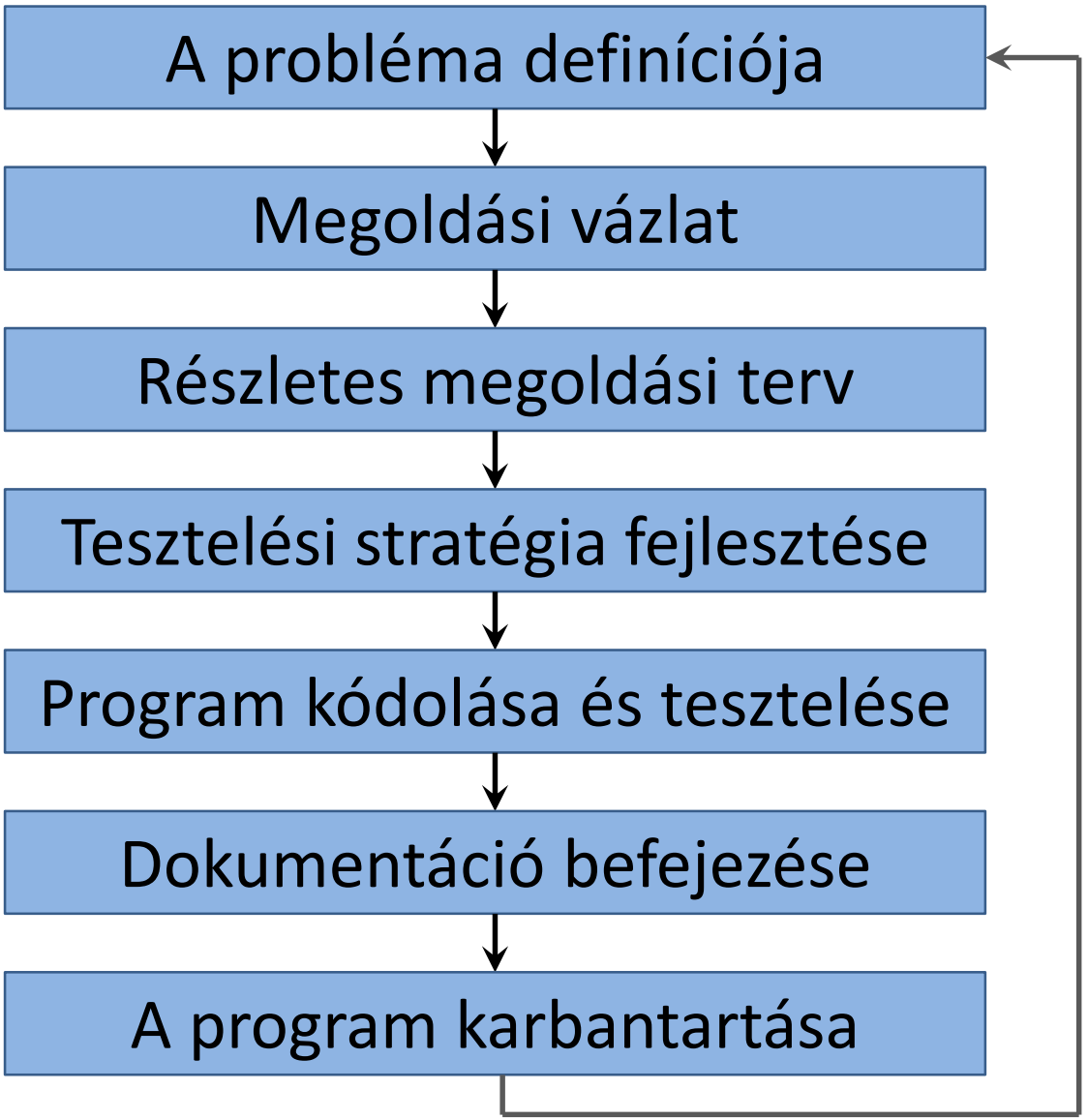
Témák

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

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

Terv jól ismert tevékenységek véges sorozatának végrehajtására, amivel elérhetjük a kívánt célt.

Precíz definíciója tevékenységeknek, amelyeknek a végrehajtása megadja a megoldási vázlat minden egyes részfeladatának a megoldását.

Néhány tulajdonsága:

- 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?
- …

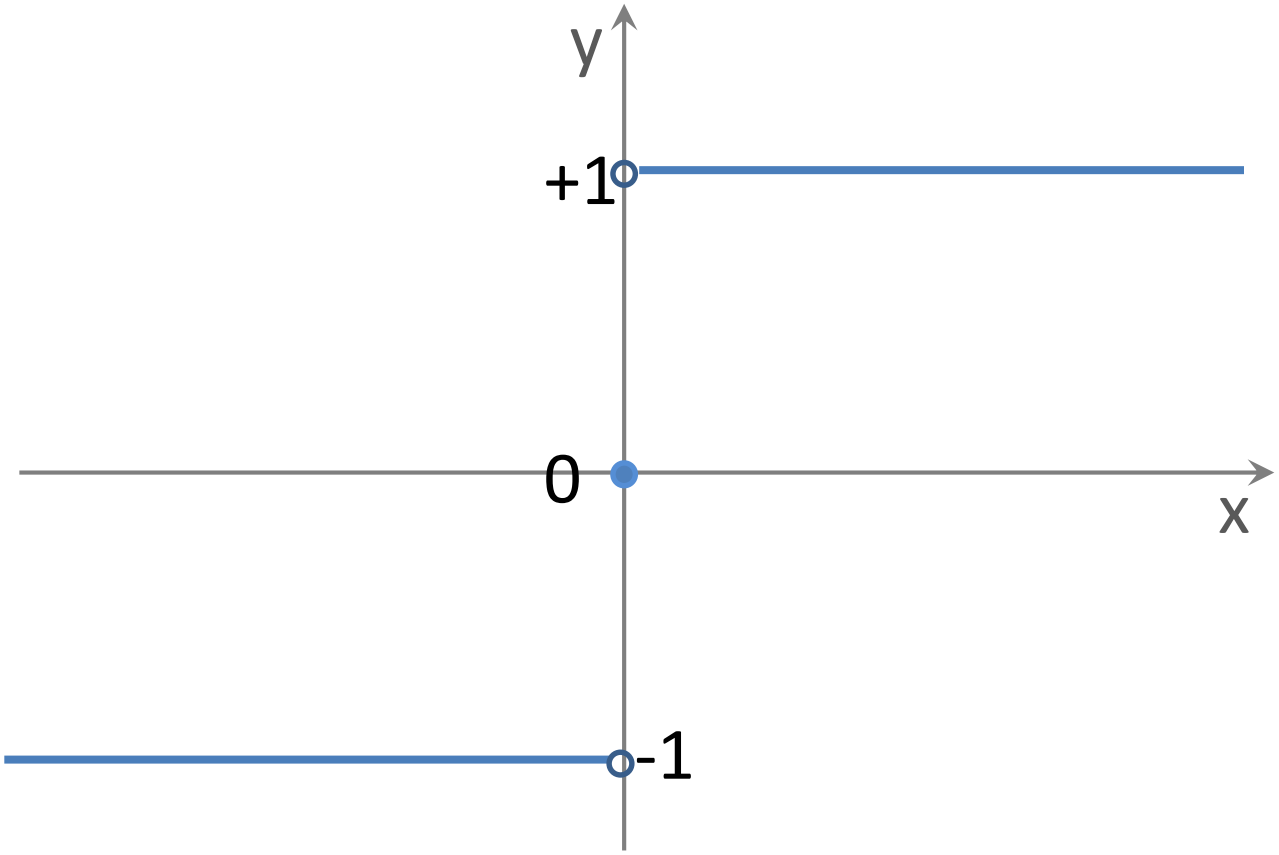
y = sign(x)

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:

$x \in \Re$

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

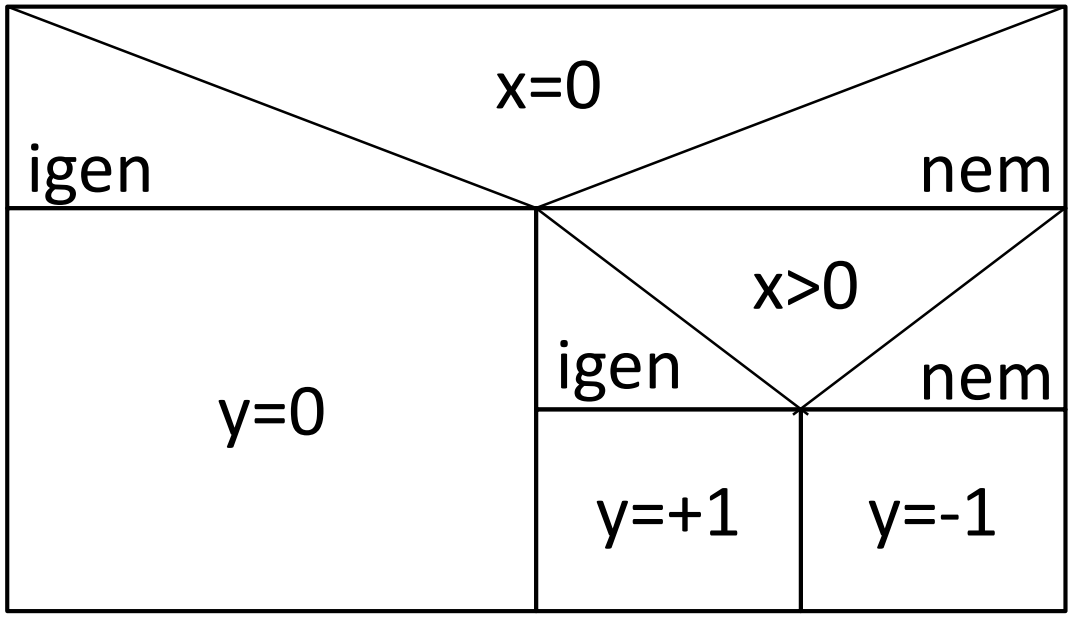
$\forall x,\ x>0 \Rightarrow y=+1$

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

$x=0 \Rightarrow y=0$

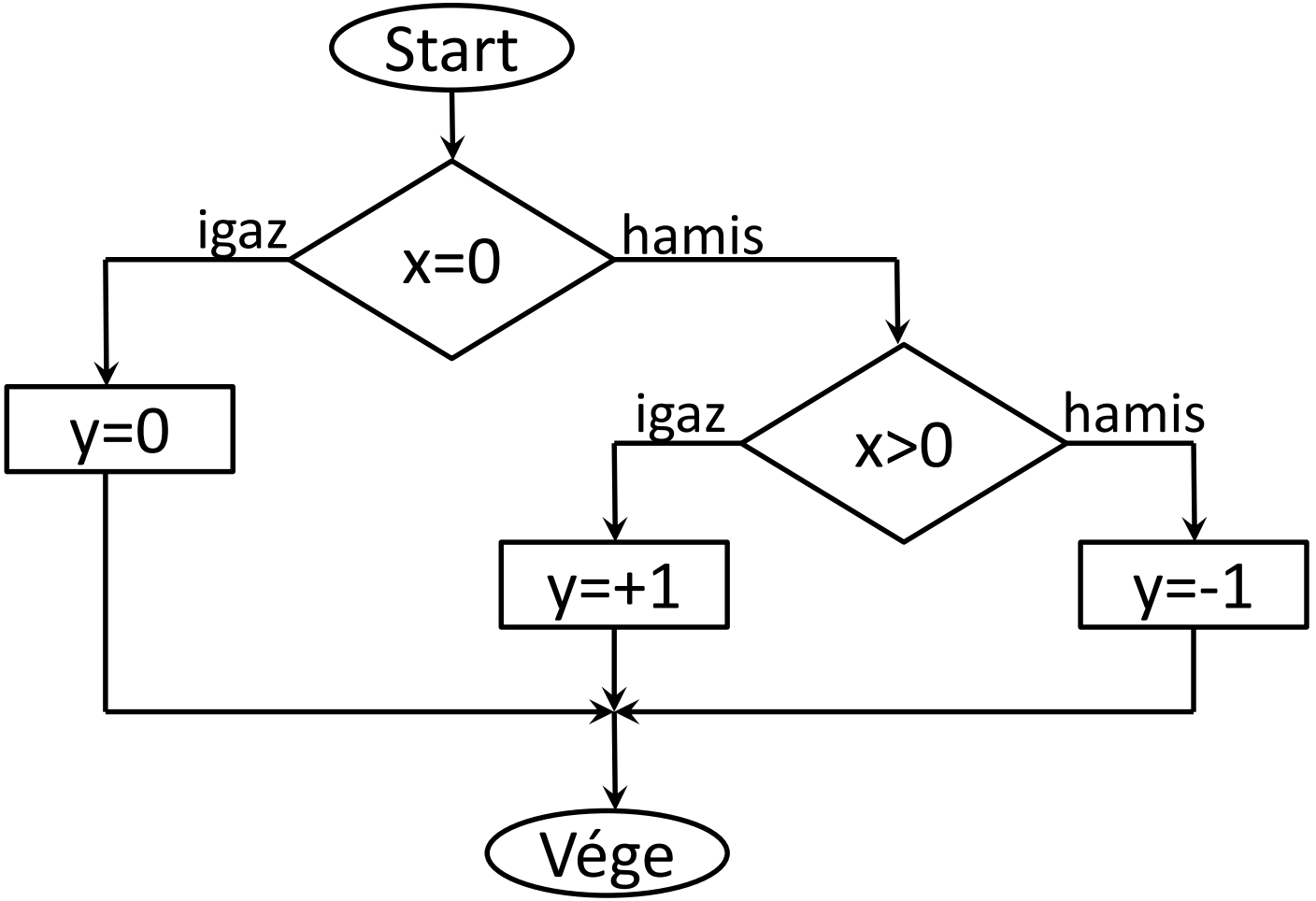
y = sign(x)

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
```

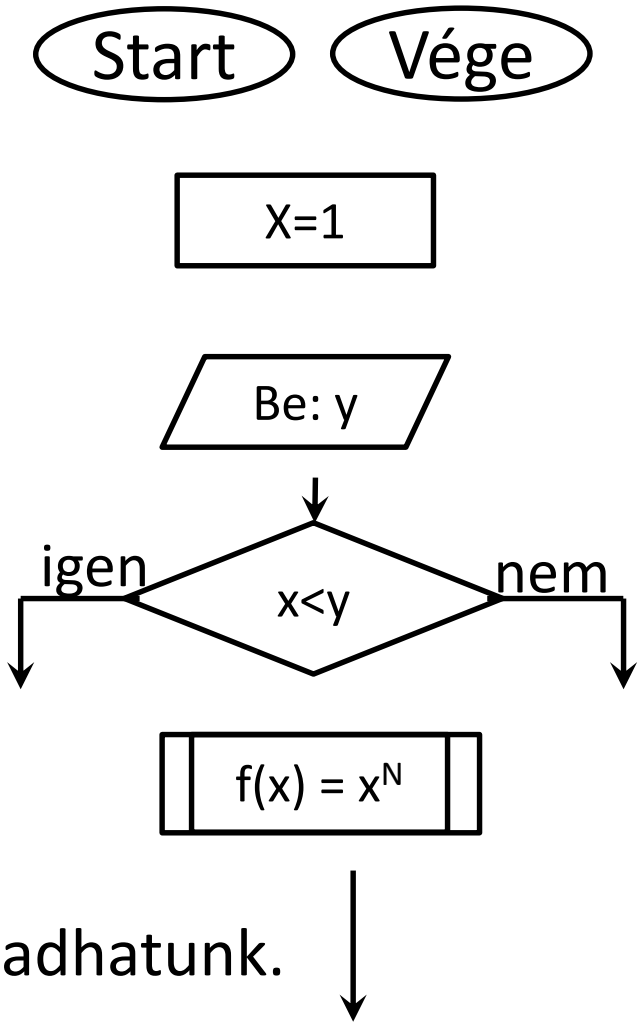
y = sign(x)

Programozási nyelven történő reprezentáció:

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

A folyamatábra elemei

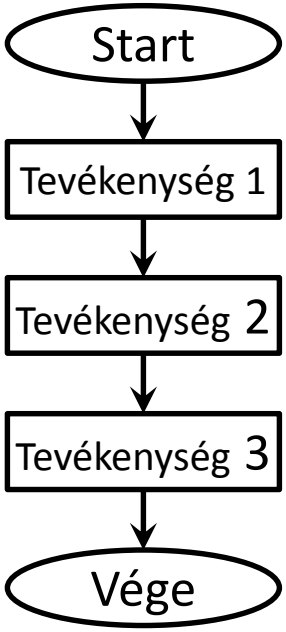
- Kiinduló- és végállapot
- Elemi utasítás
- Input/output
- Feltétel
- Beágyazás
- Csak a nyilak mentén haladhatunk.



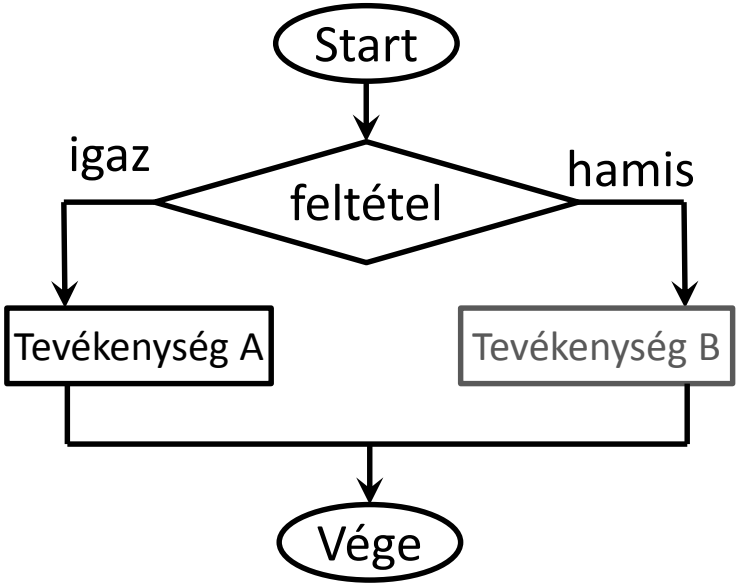
Az algoritmus alap struktúrái

folyamatábrával

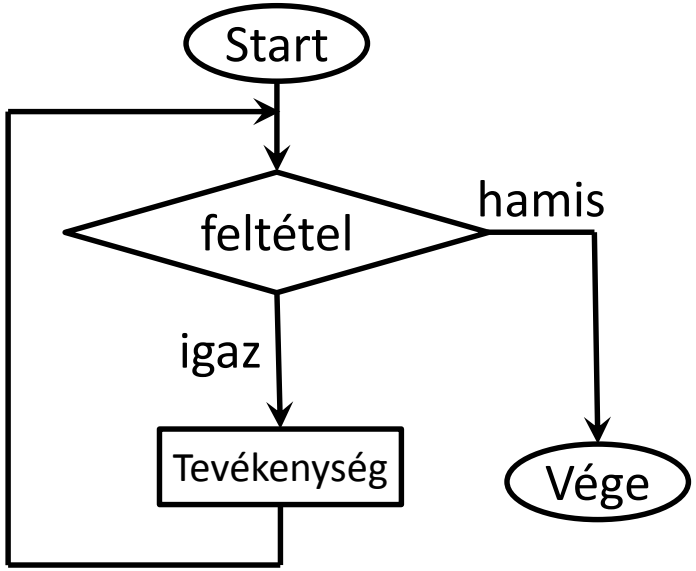
Szekvencia



Elágazás



Ismétlés



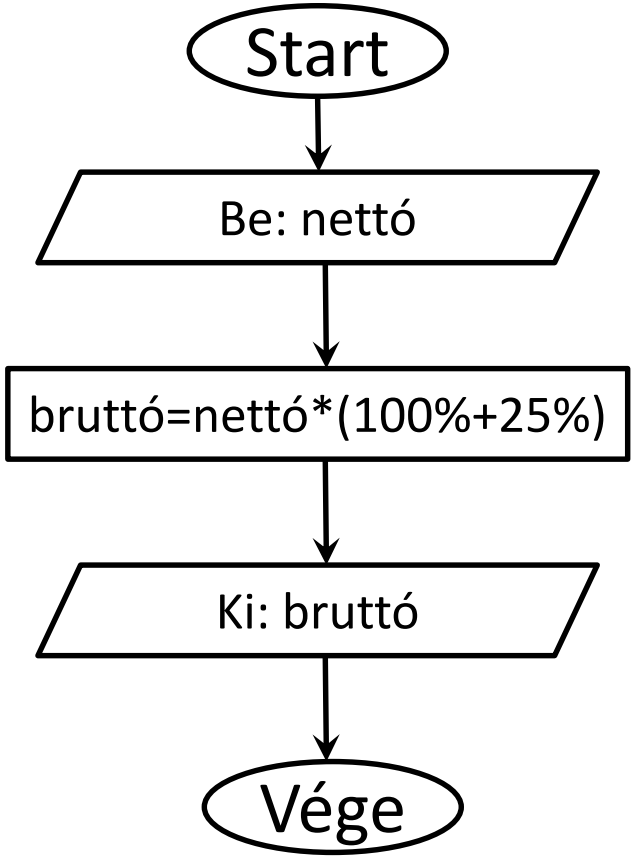
Algoritmusok módosítása

Az algoritmusokat gyakran módosítani kell, hogy jobbak legyenek.

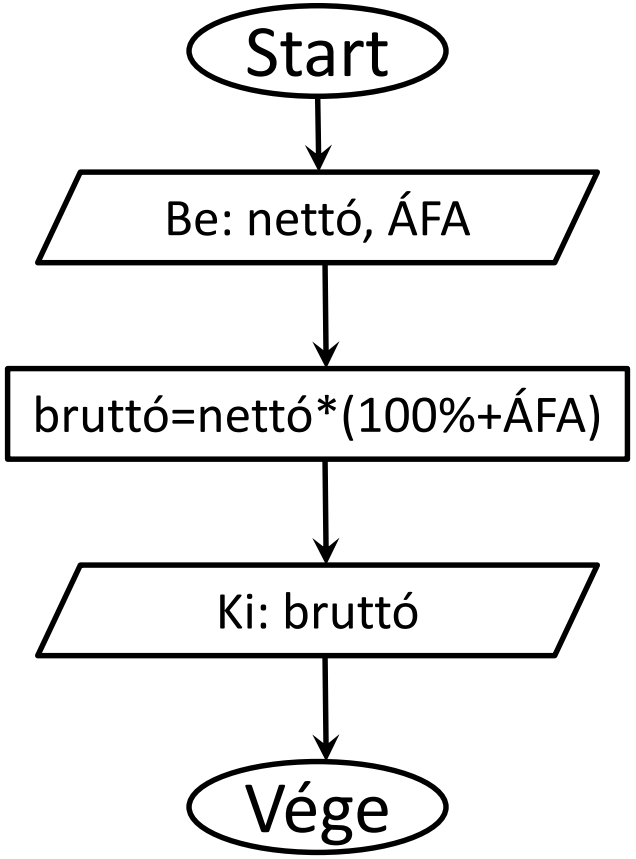
- Általánosítás:
  több esetre is alkalmazható legyen
- Kiterjesztés:
  esetek újfajta körére is alkalmazható legyen
- Beágyazás:
  egy algoritmus újrahasznosítása egy másikon belül
- ‘Bolondbiztossá’ tétel:
  megbízhatóvá, robosztussá, hibaelkerülővé tétel

Algoritmus általánosítása

Eredeti:

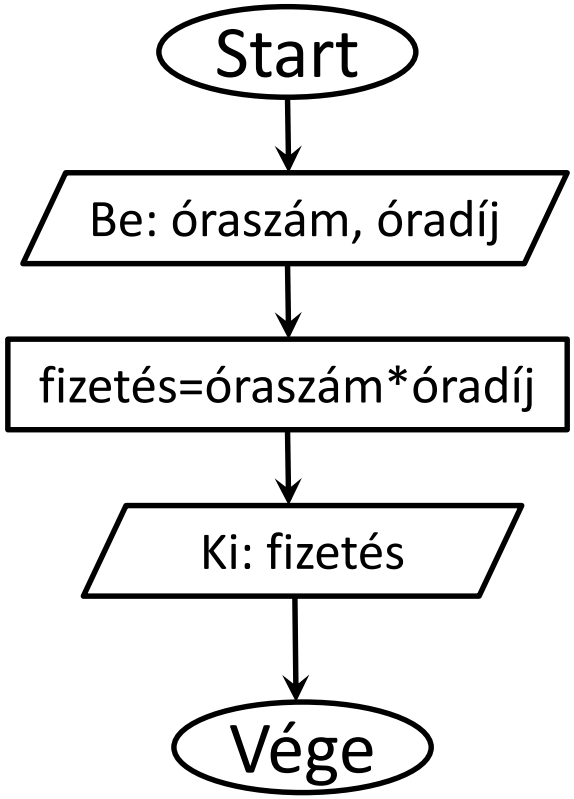


Általánosított:

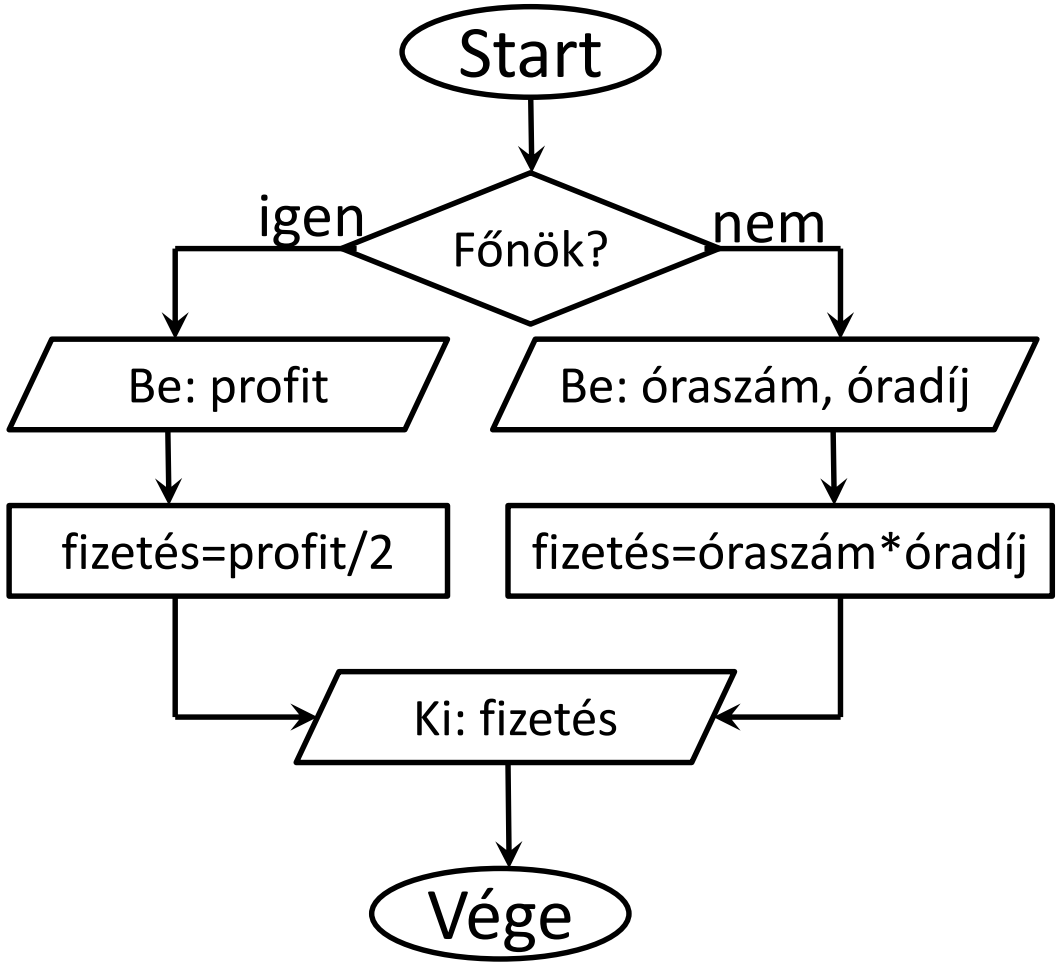


Algoritmus kiterjesztése

Eredeti:

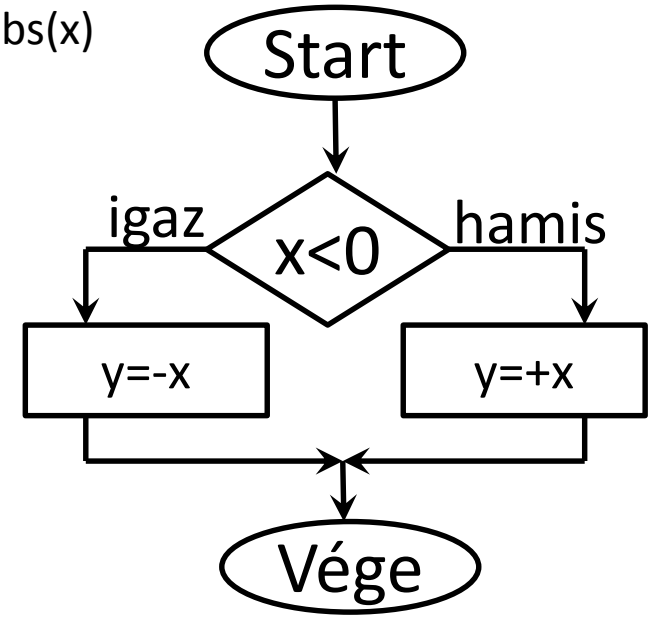


Kiterjesztett:

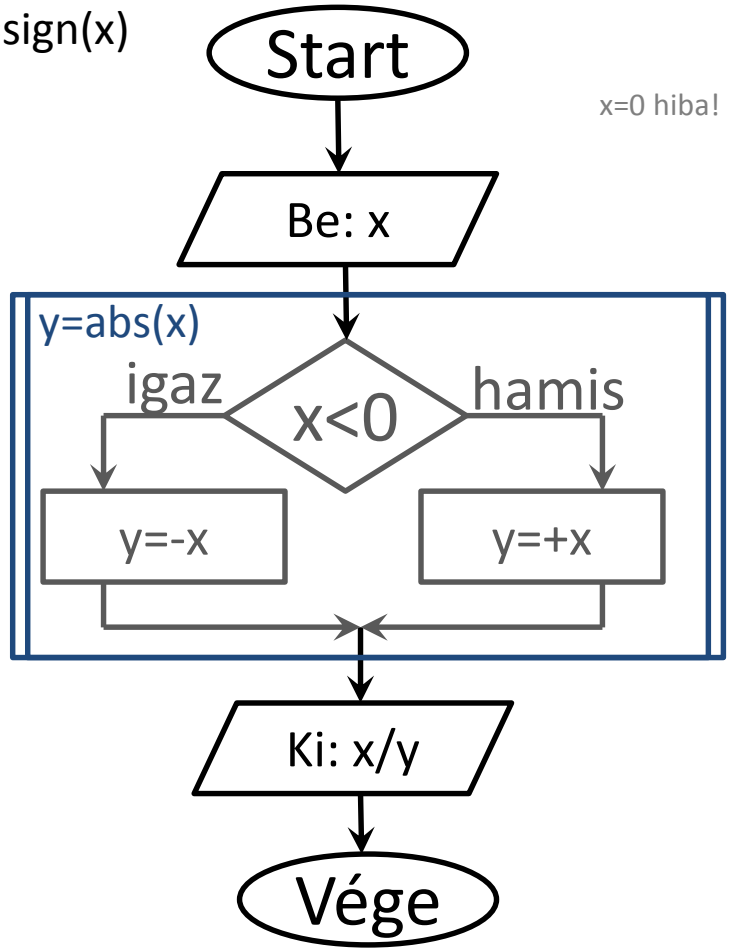


Algoritmus beágyazása

Eredeti:
y=abs(x)



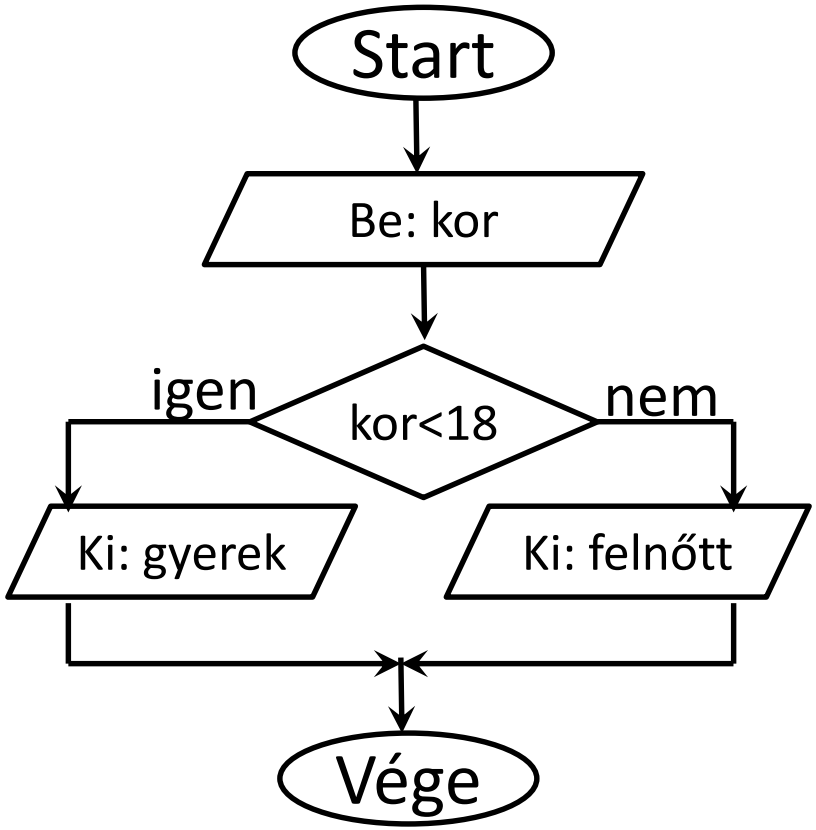
Beágyazott:
y=sign(x)



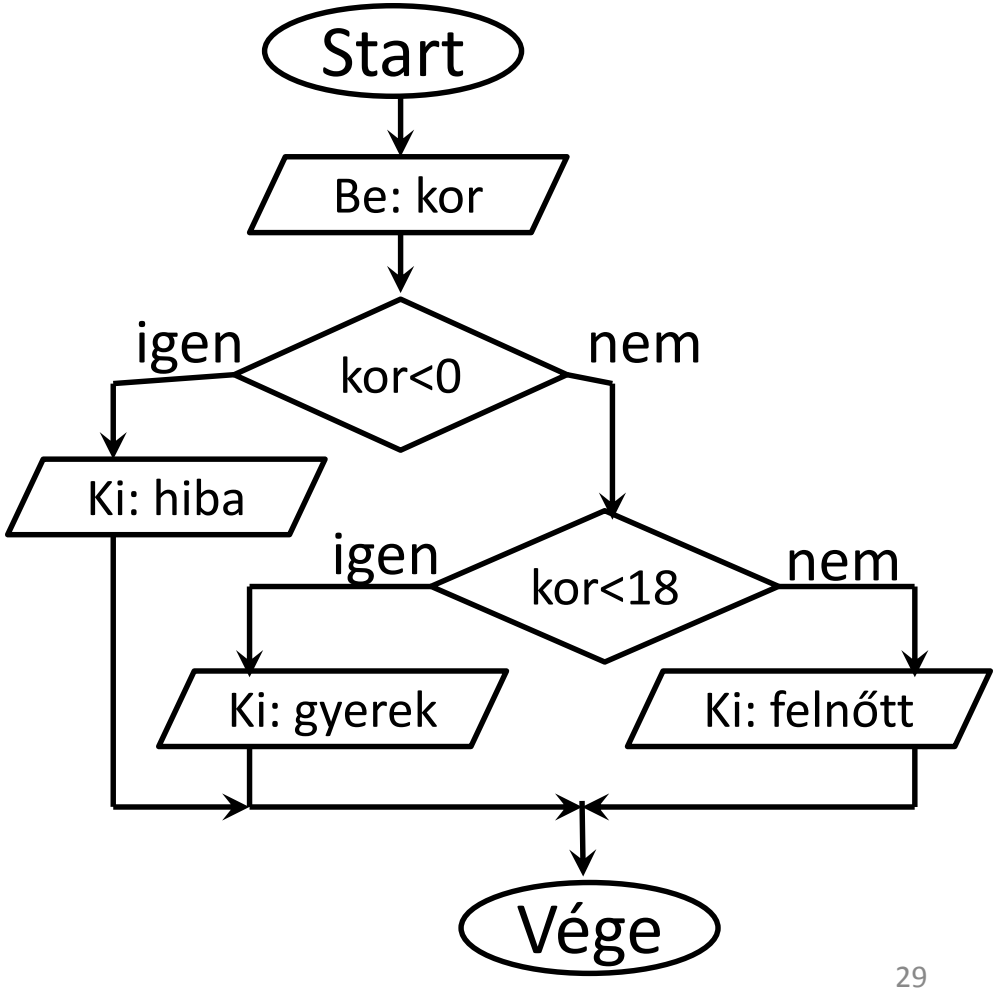
x=0 hiba!

Algoritmus ‘bolondbiztossá’ tétele

Eredeti:



Bolondbiztos:



Alternatív algoritmus

Gyakran többféleképpen is elérhetjük ugyanazt a célt.

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

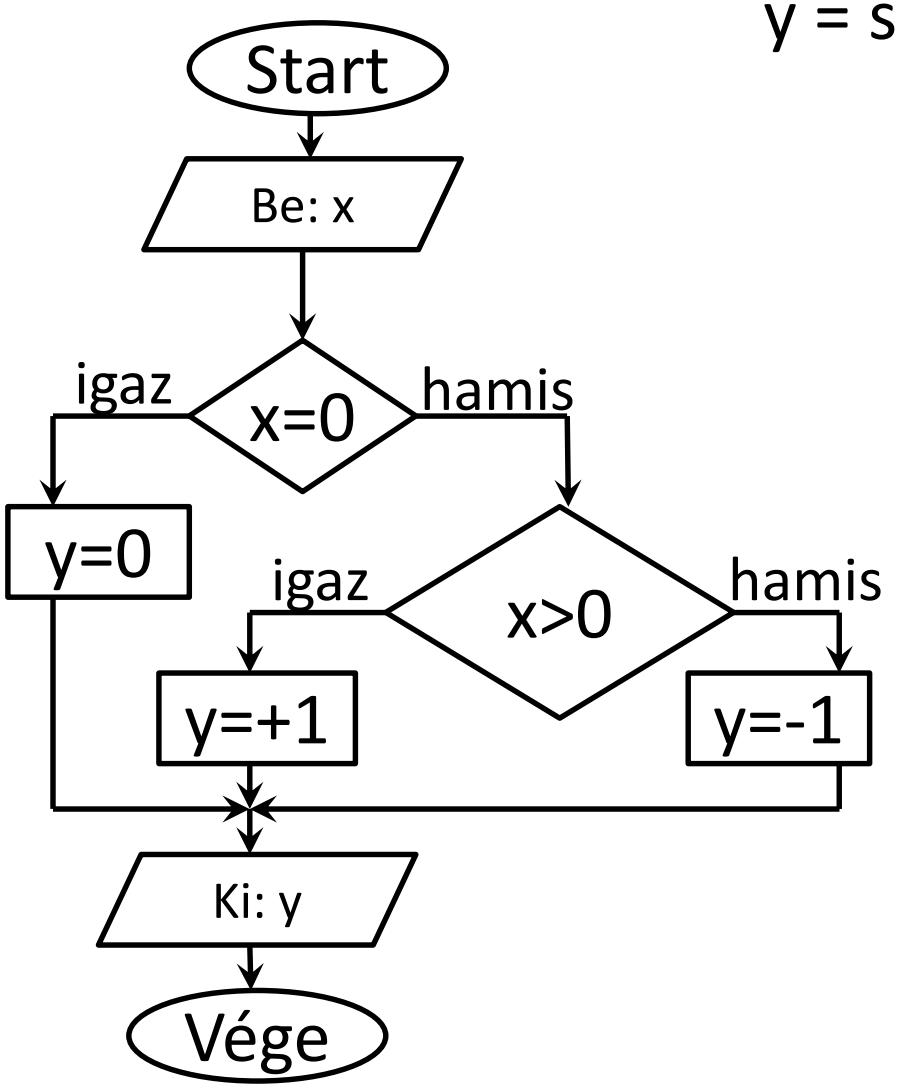
Ez azt jelenti, hogy azonos bemeneti adatokra azonos eredményeket adnak, de ezt máshogy érik el.

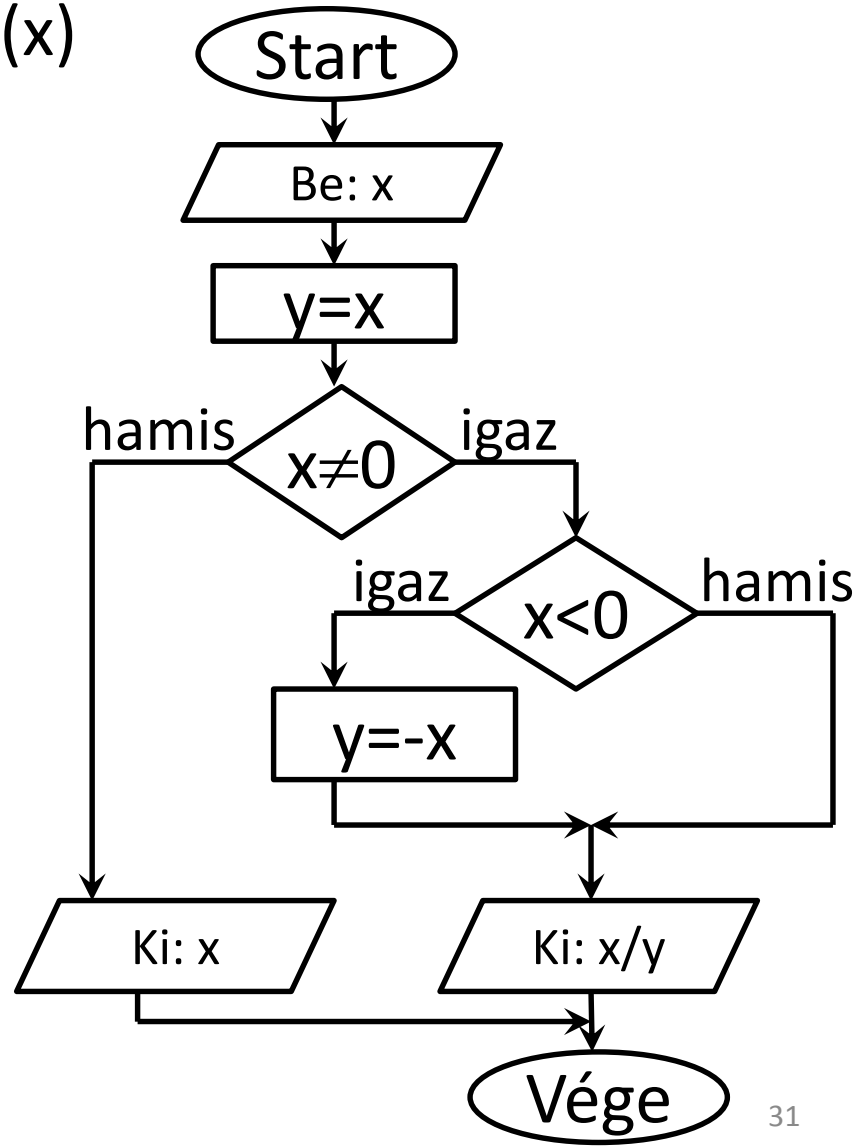
Néha hasznos az egyik algoritmust előnyben részesíteni, néha viszont mindegy, melyiket használjuk.

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

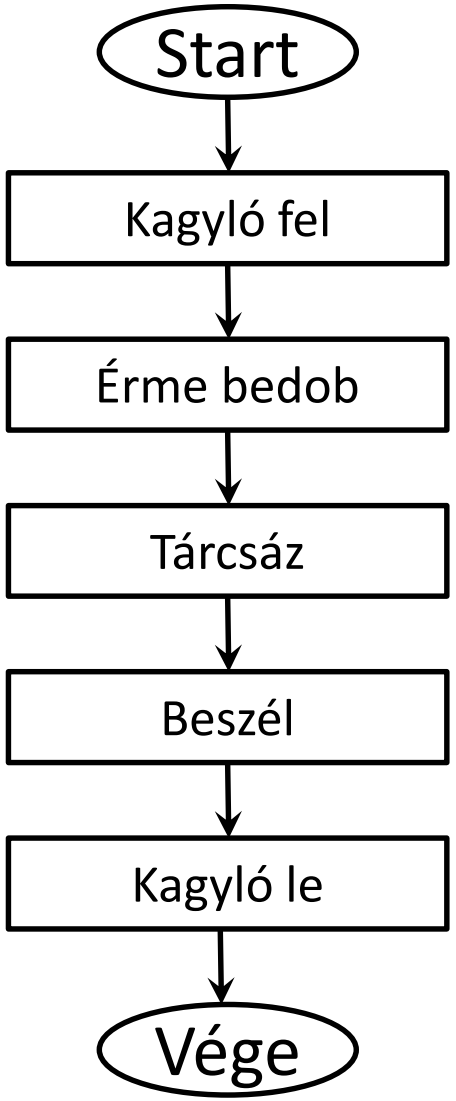
Hogyan buszozzunk?

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.

Nyilvános érmés telefon használata



Problémák:

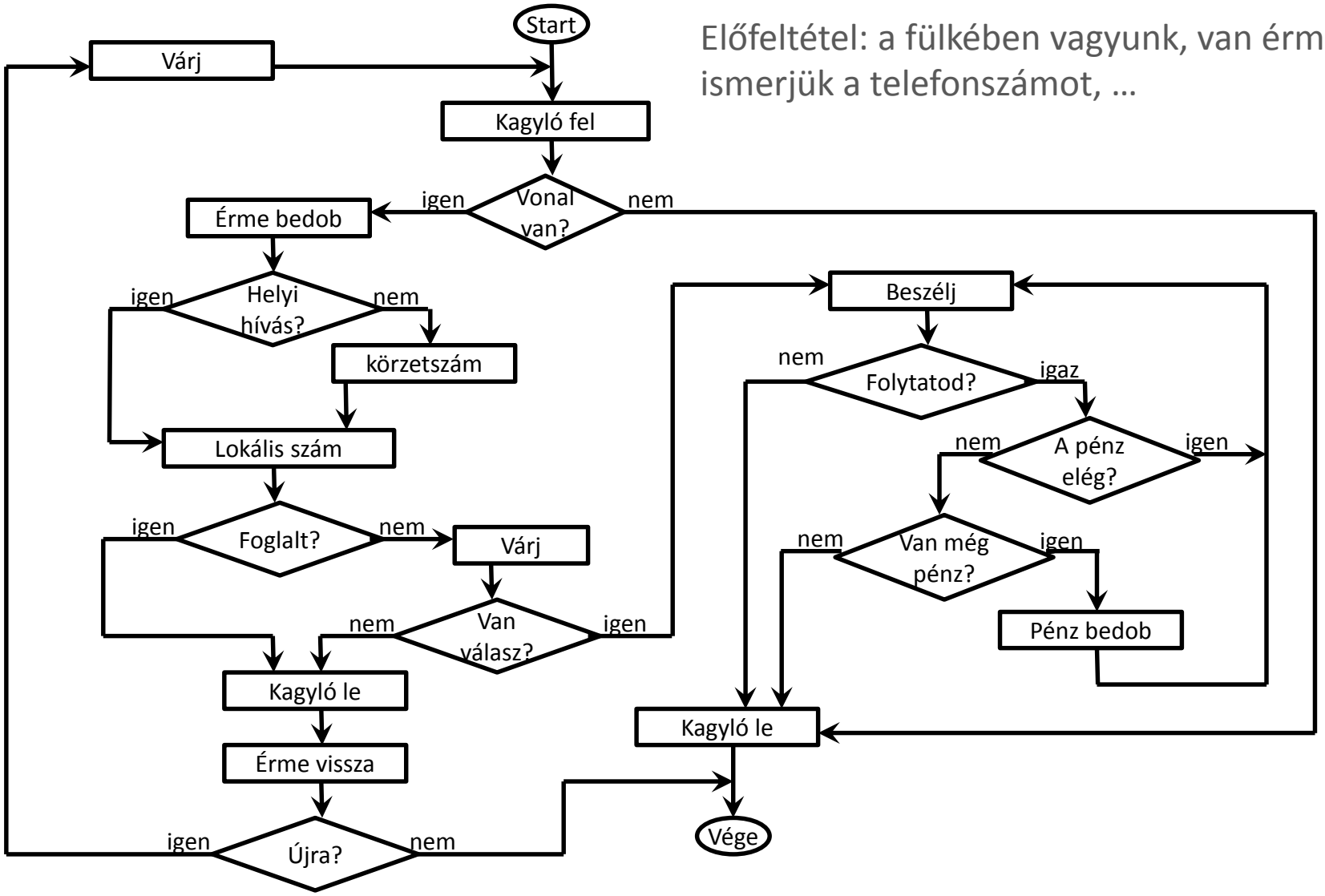
- nem teljes
- félreérthető
- …

Módosítás:

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

Nyilvános érmés telefon használata

Előfeltétel: a fülkében vagyunk, van érménk, ismerjük a telefonszámot, …



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

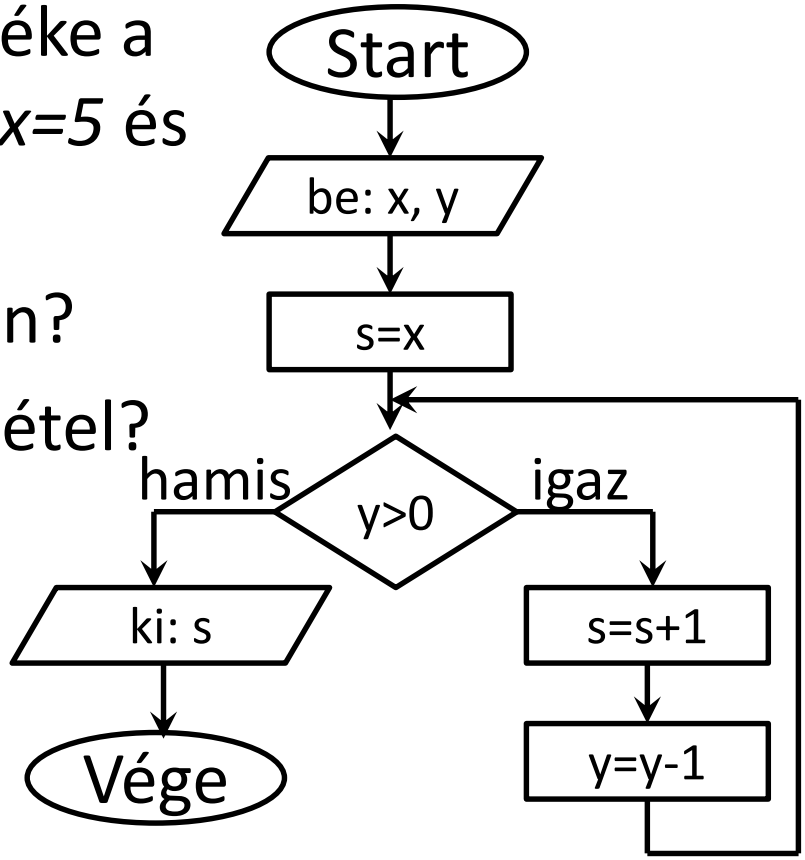
Ha ez Igaz,	**akkor ez Hamis**
=	≠
<	≥
>	≤
≠	=
≤	>
≥	<

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

Link a megoldásra

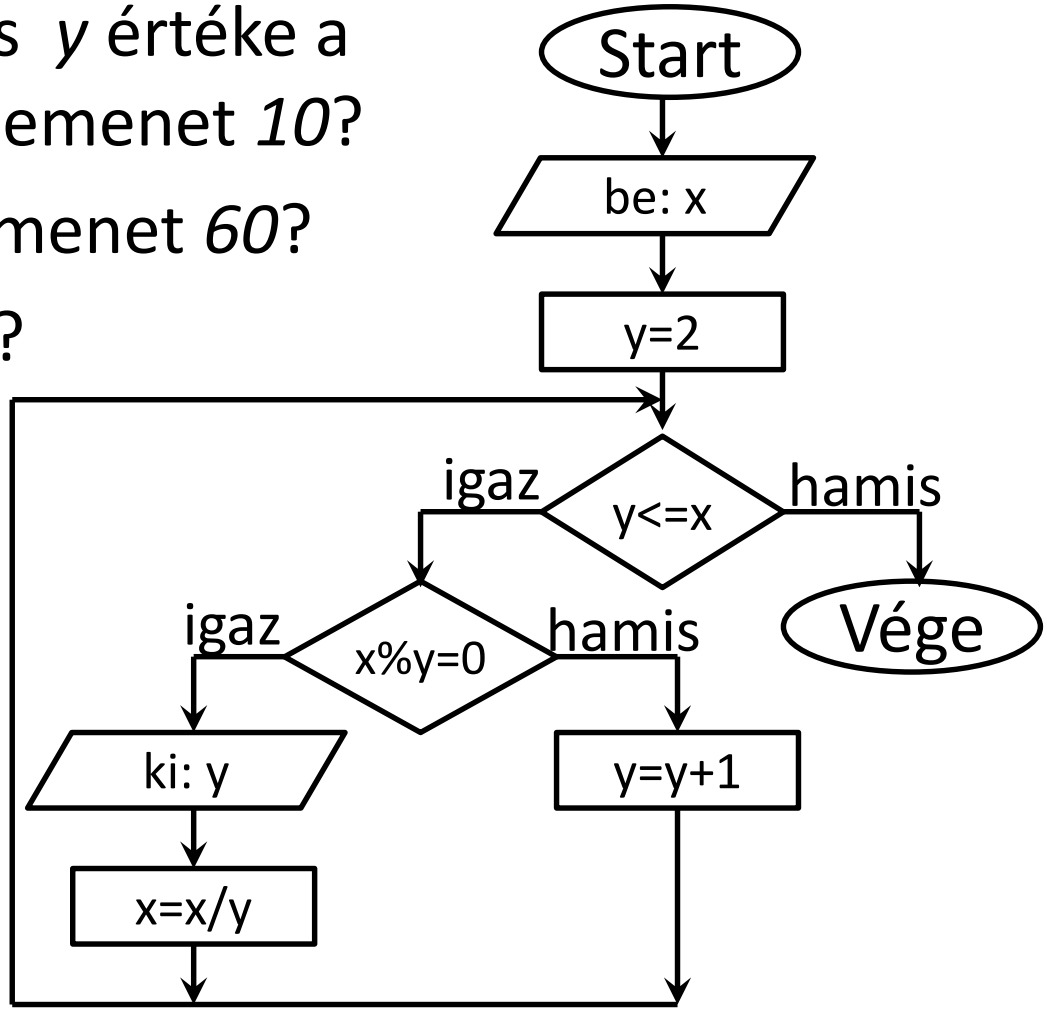
Feladat: folyamatábra

- 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?



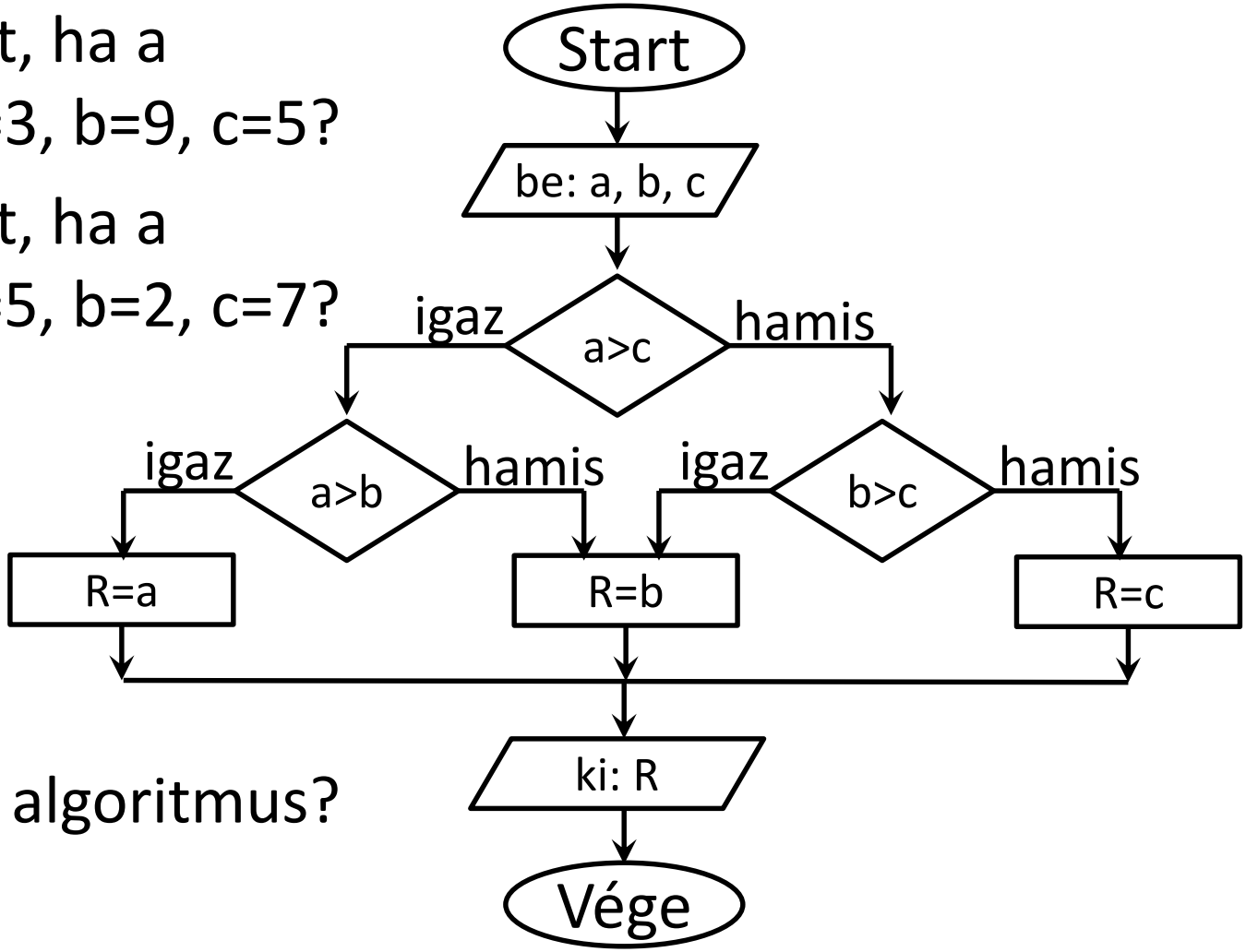
Feladat: folyamatábra

- 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?



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?

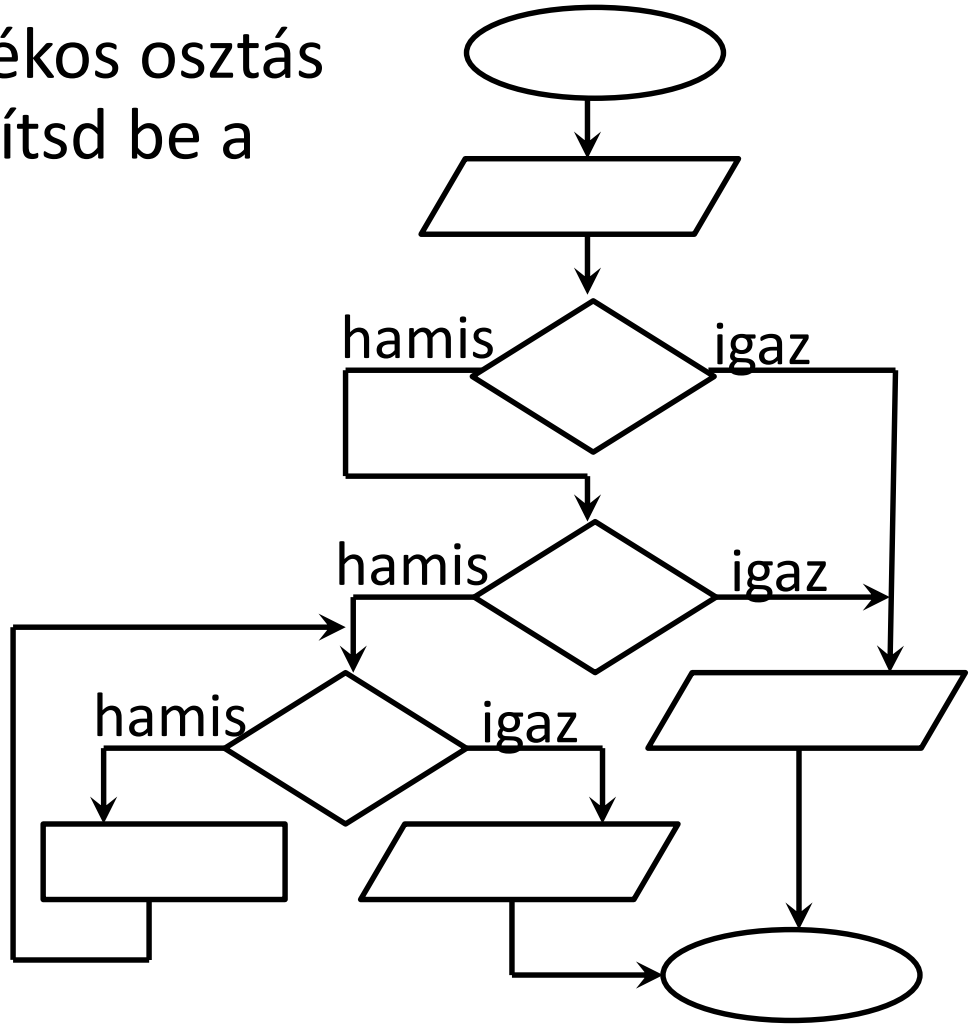


- Mit csinál az algoritmus?

Feladat: folyamatábra

Ez a folyamatábra a maradékos osztás műveletét írja le. Helyettesítsd be a kifejezéseket!

- Start
- a<b
- a<0
- b<=0
- a=a-b
- be: a, b
- ki: „hiba”
- ki: a
- Vége



Feladat: folyamatábra

- Szökőév meghatározása 
- Hatványozás
- Faktoriális kiszámítása 
- Elsőfokú egyenlet megoldása
- Az év napjának meghatározása
- Decimális szám konvertálása binárissá 
- Bináris számok inkrementálása
- Bináris számok összeadása
- Fibonacci-sorozat
- Collatz-sejtés 
- …

Pszeudokód

Szekvencia:

utasítás1
utasítás2
utasítás3
…

Elágazás:

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

Ismétlés:

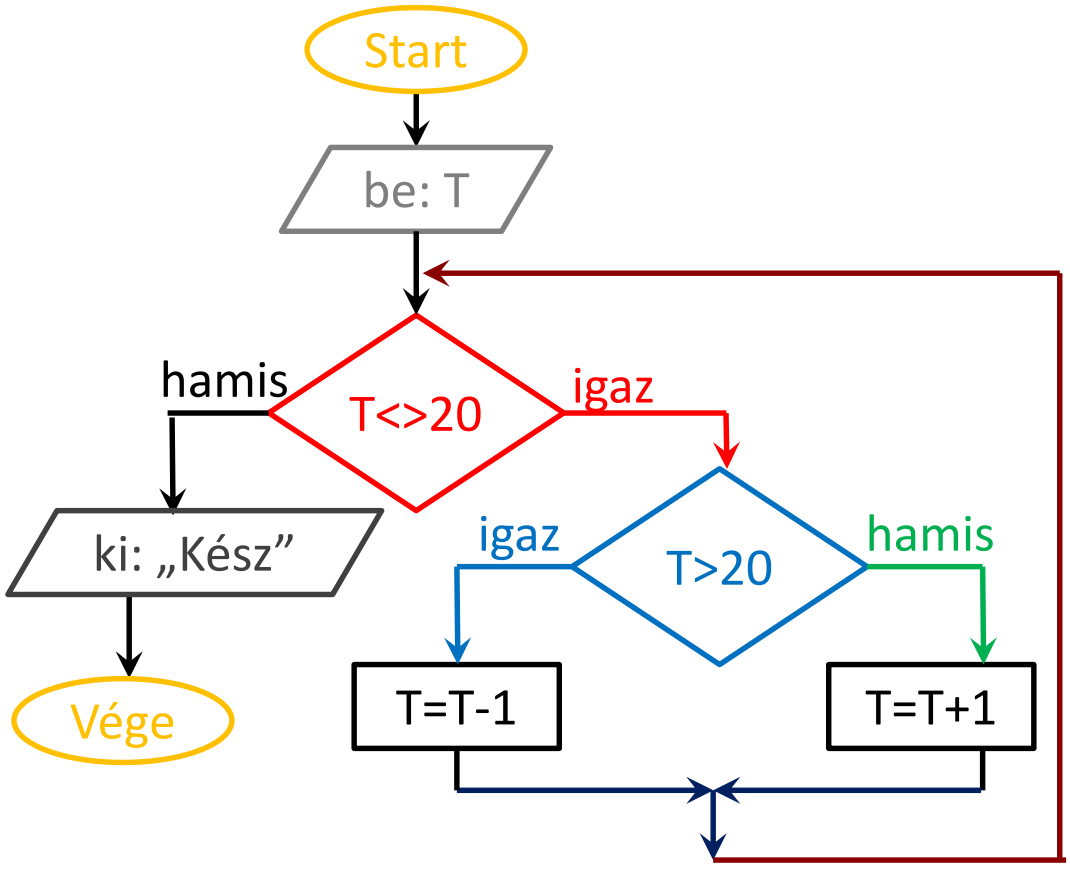
while *feltétel* do
    utasítás$_{(igaz)}$
enddo
utasítás$_{(hamis)}$
…

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



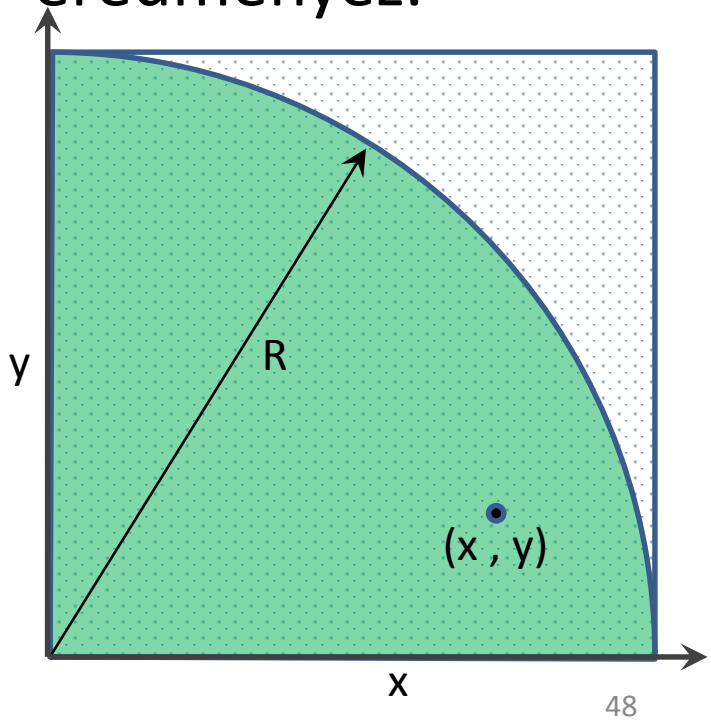

```
input T
while T<>20 do
   if T>20 then
      T=T-1
   else
      T=T+1
   endif
enddo
output ”Kész”
```

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*R)
```

A π értékének közelítése

Nagyobb R érték pontosabb közelítést eredményez.



Feladat: pszeudokód

```
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?

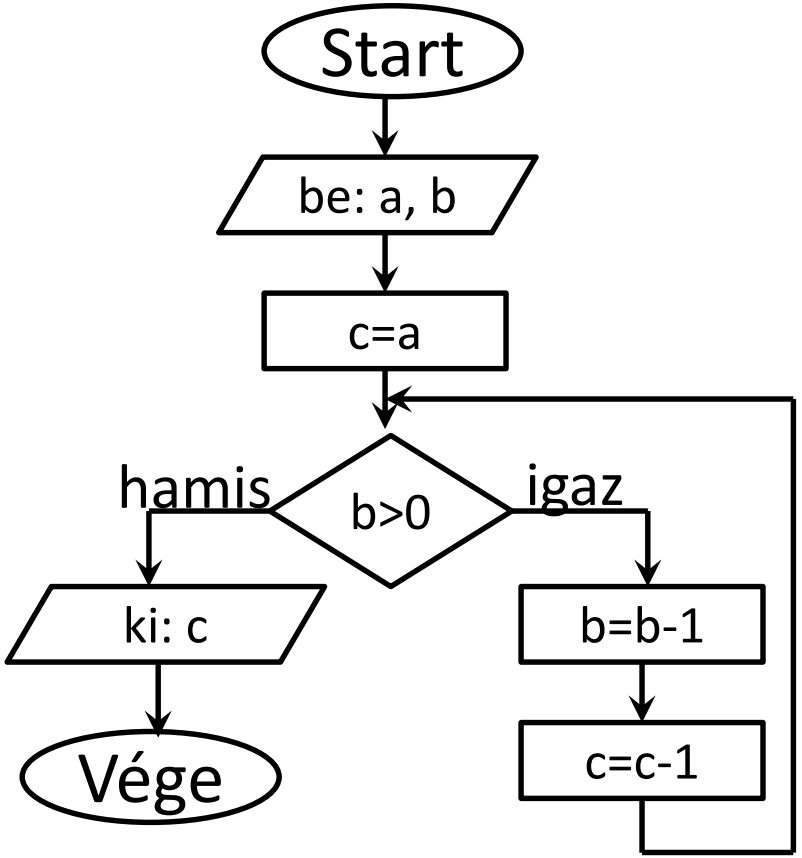
- Mit csinál ez az algoritmus?

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

Feladat: pszeudokód

```
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?



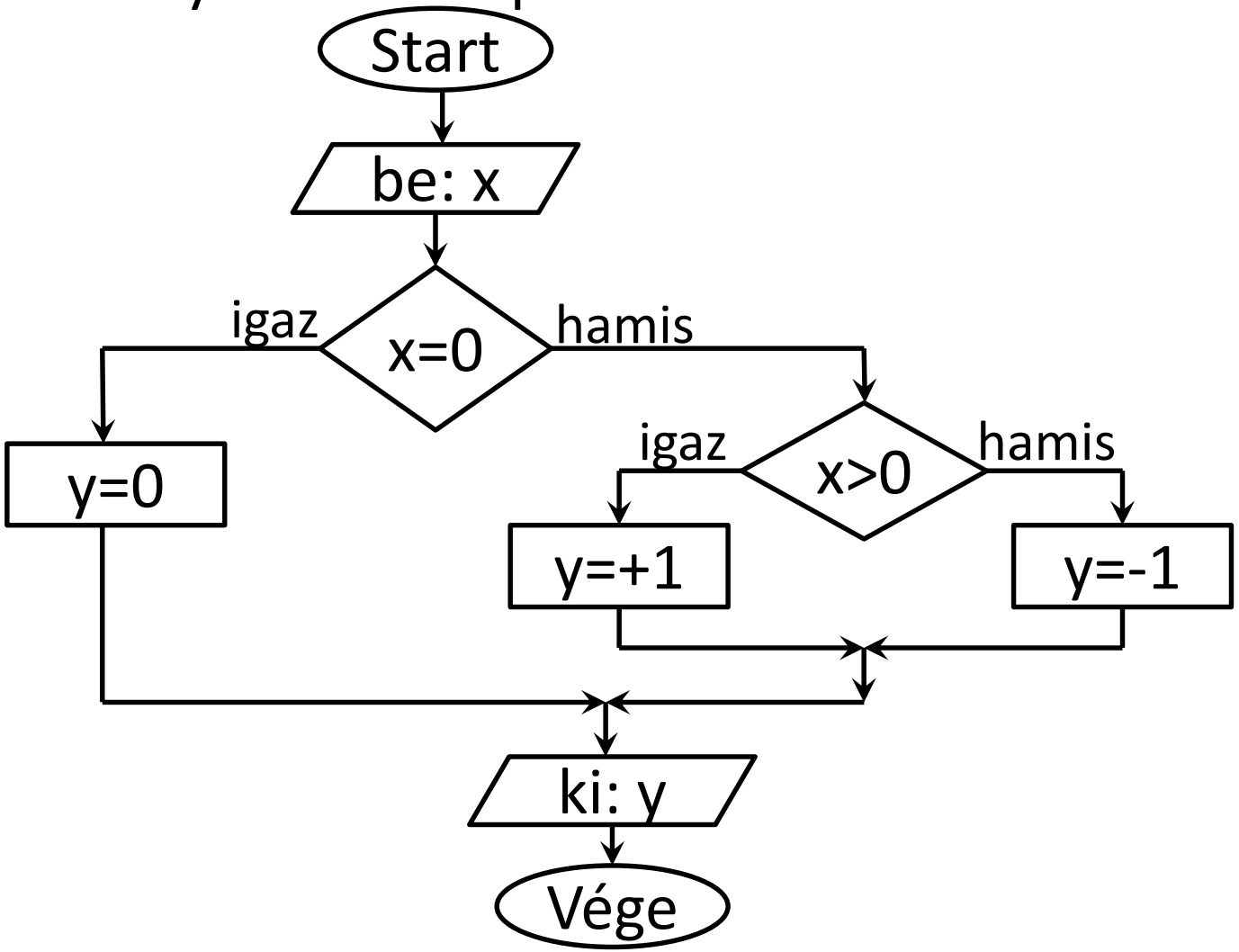
Feladat: pszeudokód

```
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?

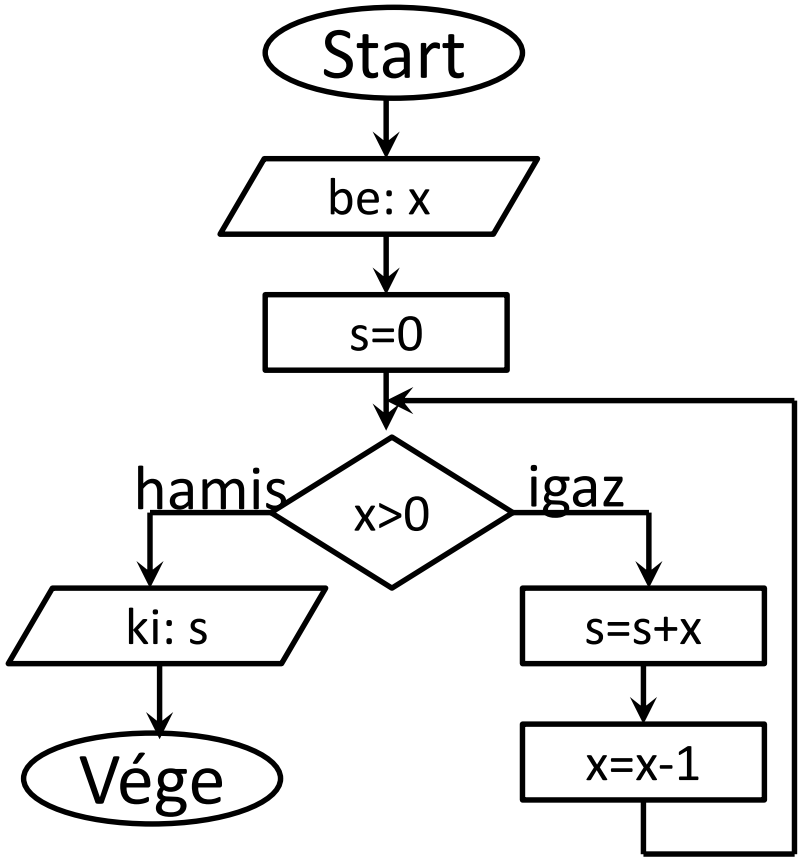
Feladat: pszeudokód

- Írd le a folyamatábrát pszeudokóddal!



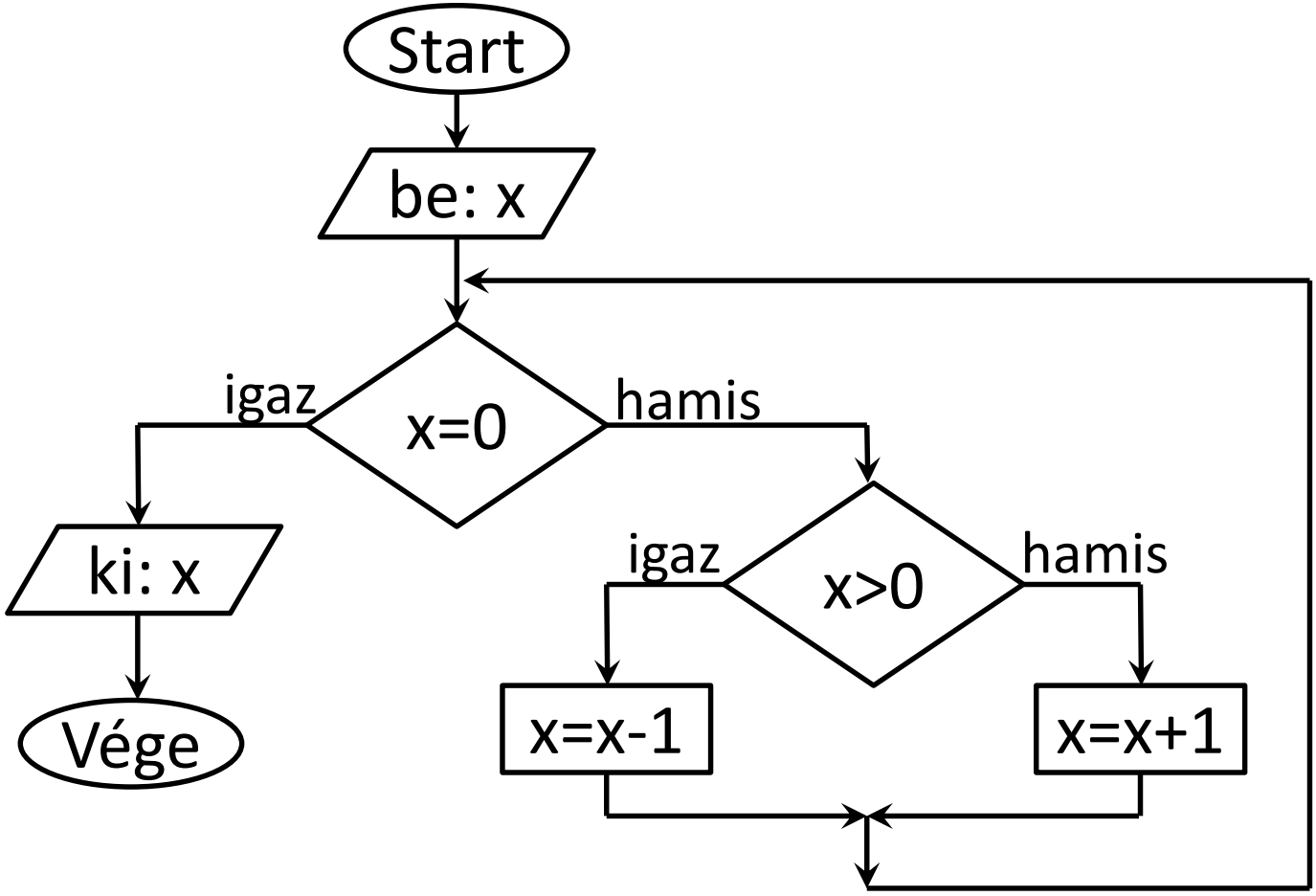
Feladat: pszeudokód

- Írd le a folyamatábrát pszeudokóddal!



Feladat: pszeudokód

- Írd le a folyamatábrát pszeudokóddal!



Feladat: pszeudokód

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=[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)

[…]: egészrész képzés
(a törtrész elhagyása)

Feladat: pszeudokód

```
input N
input B
R=0
P=1
while N<>0 do
   R=R+(N%B)*P
   P=P*10
   N=[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

```
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

```
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, 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?

- Készíts folyamatábrát ehhez az algoritmushoz!

Feladat: pszeudokód

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: pszeudokód

Írd le az alábbi algoritmusokat pszeudokóddal!

- Az abszolút érték meghatározása
- Számok összege 10-től 20-ig
- Hatványozás 
- Háromszög-egyenlőtlenség 
- Derékszögű háromszög
- Faktoriális kiszámítása
- Eldönteni egy számról, hogy prím-e 
- Prímtényezőkre bontás
- Fibonacci-sorozat 

Tömbök

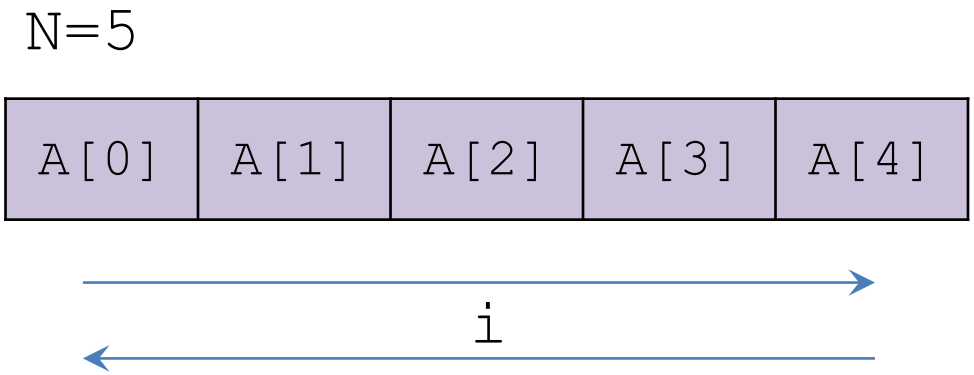
- Elemek sorozata, ahol minden elem egy (sorszám jellegű) index segítségével elérhető
- Egy N elemű tömb: A[0]=8, A[1]=-2, …, A[N-1]=98
- Példa: tömb elemeinek összege

```
sum=0
i=0
while i<N do
   sum=sum+A[i]
   i=i+1
enddo
output sum
```

Tömb példa

- N db értékek beolvasása, tárolás és kiírása fordított sorrendben

```
i=0
while i<N do
   input A[i]
   i=i+1
enddo
while i>0 do
   i=i-1
   input A[i]
enddo
```



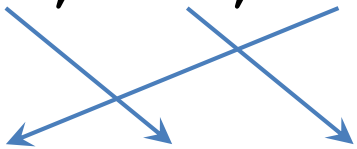
Keresés és rendezés

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

Rendezés példa

Beszúrásos rendezés

- **72**, 63, 34, 54, 91, 38, 53
- **63, 72**, 34, 54, 91, 38, 53
- **34, 63, 72**, 54, 91, 38, 53

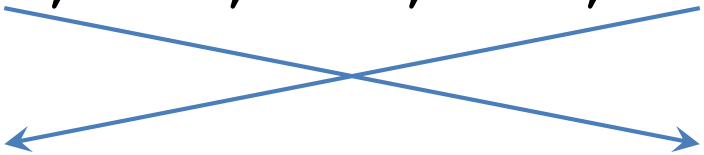


- **34, 54, 63, 72**, 91, 38, 53
- **34, 54, 63, 72, 91**, 38, 53
- **34, 38, 54, 63, 72, 91**, 53
- **34, 38, 53, 54, 63, 72, 91**

Rendezés példa

Minimum kiválasztásos rendezés

- 72, 63, 34, 54, 91, 38, 53
- **34**, 63, 72, 54, 91, 38, 53
- **34, 38**, 72, 54, 91, 63, 53

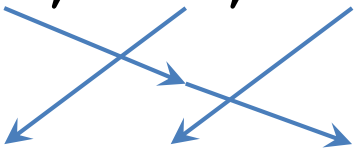


- **34, 38, 53**, 54, 91, 63, 72
- **34, 38, 53, 54**, 91, 63, 72
- **34, 38, 53, 54, 63**, 91, 72
- **34, 38, 53, 54, 63, 72, 91**

Rendezés példa

Buborékrendezés

- 72, 63, 34, 54, 91, 38, 53
- 63, 34, 54, 72, 38, 53, **91**
- 34, 54, 63, 38, 53, **72, 91**



- 34, 54, 38, 53, **63, 72, 91**
- 34, 38, 53, **54, 63, 72, 91**
- 34, 38, **53, 54, 63, 72, 91**
- **34, 38, 53, 54, 63, 72, 91**

Feladat: pszeudokód

- Tömb elemeinek átlaga 
- Megkeresni egy elemet egy (rendezett) tömbben 
- Minimum/maximum keresése 
- Szélsőérték helyének megkeresése 
- Két változó értékének felcserélése 
- Szélsőérték-kiválasztásos rendezés
- Közvetlen beszúrásos rendezés
- Buborékrendezés 
- Keresés rendezett bináris fában (azaz keresőfában)

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 *beágyazás*.
- Absztrakció: Nemcsak egy tevékenység, de hasonló tevékenységek halmazára is használható, paraméterekkel megadva a finomságokat. Az általánosítás megjelenési formája.

Alprogram

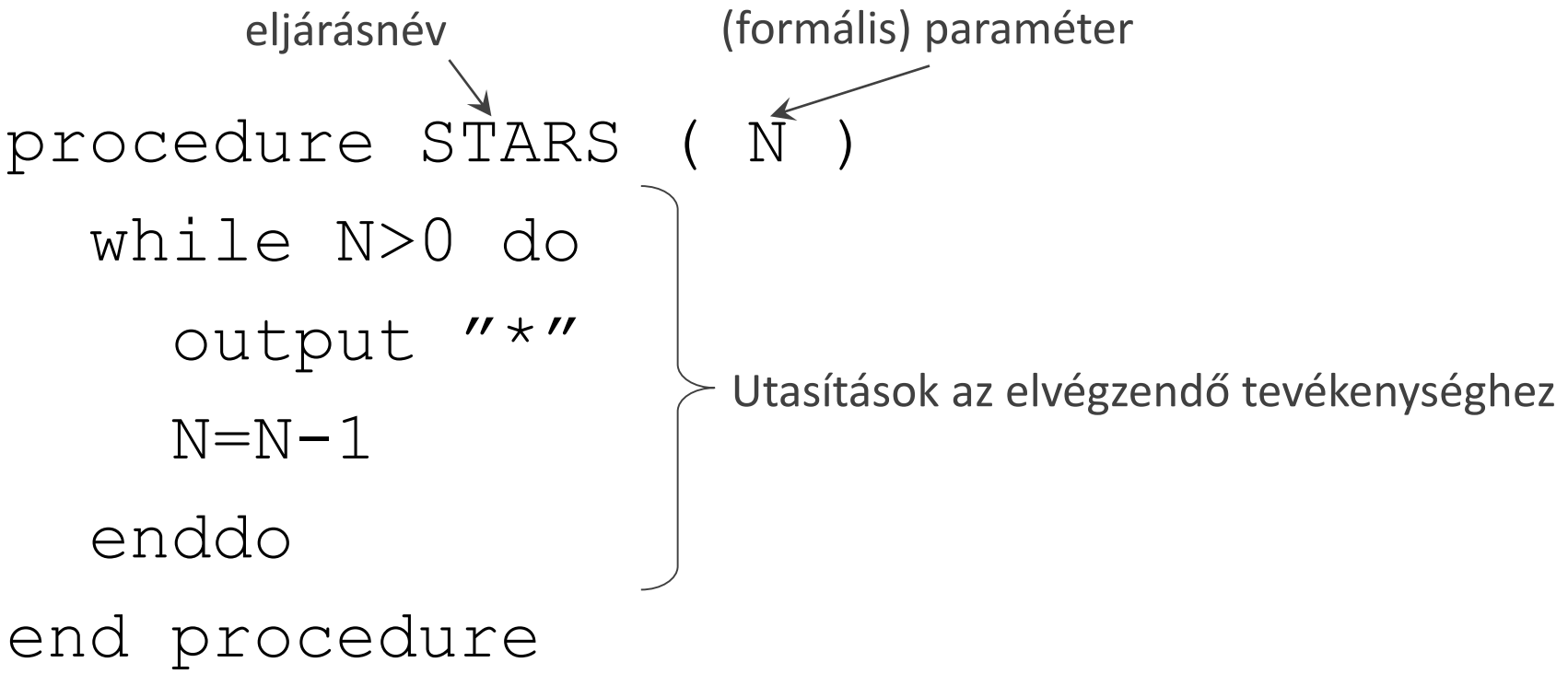
Az alprogramok két fajtája:

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

- **Eljárás**: tevékenységhalmaz valami megtételéhez a kód egy adott pontján (nincs visszatérési érték)
  Pl. Írd ki a neved!
  `ird(”Imre”)`

Eljárás példa

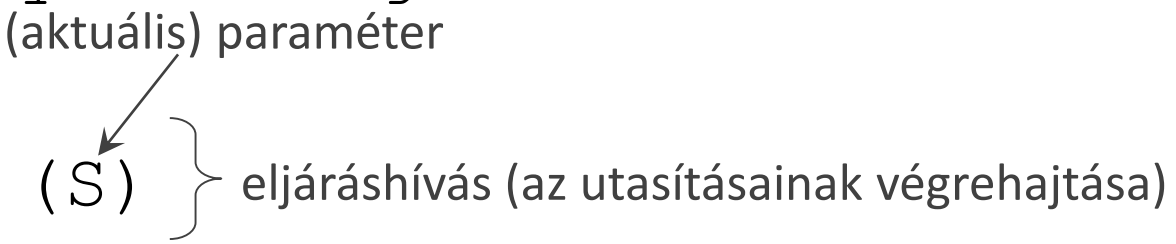
Eljárás definíció N db ’*’ karakter kiírásához



Eljárás példa

Az előző eljárás meghívása egy kódban

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

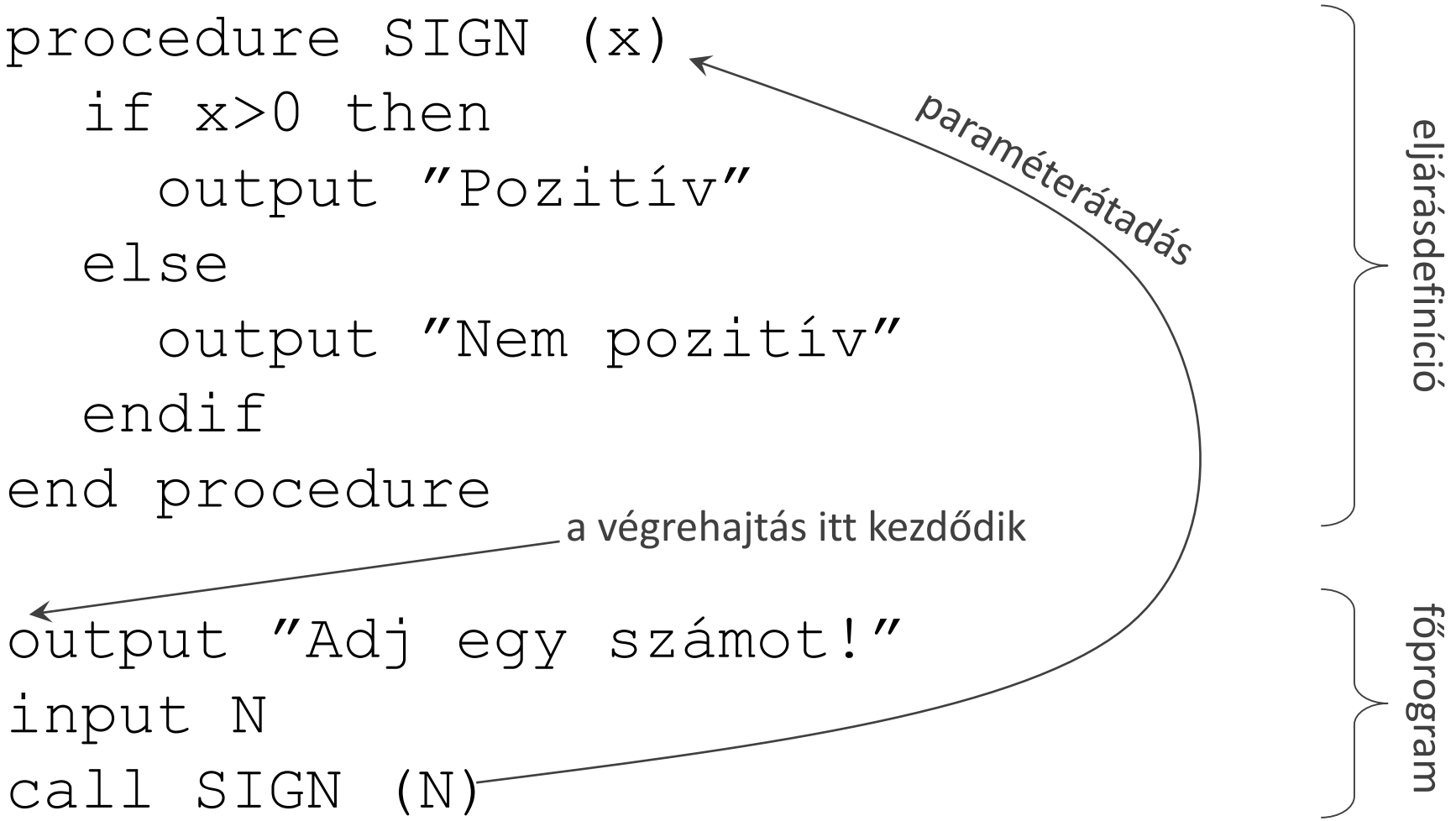


Eljárás példa

Egy eljárás, amely megmondja egy számról pozitív-e

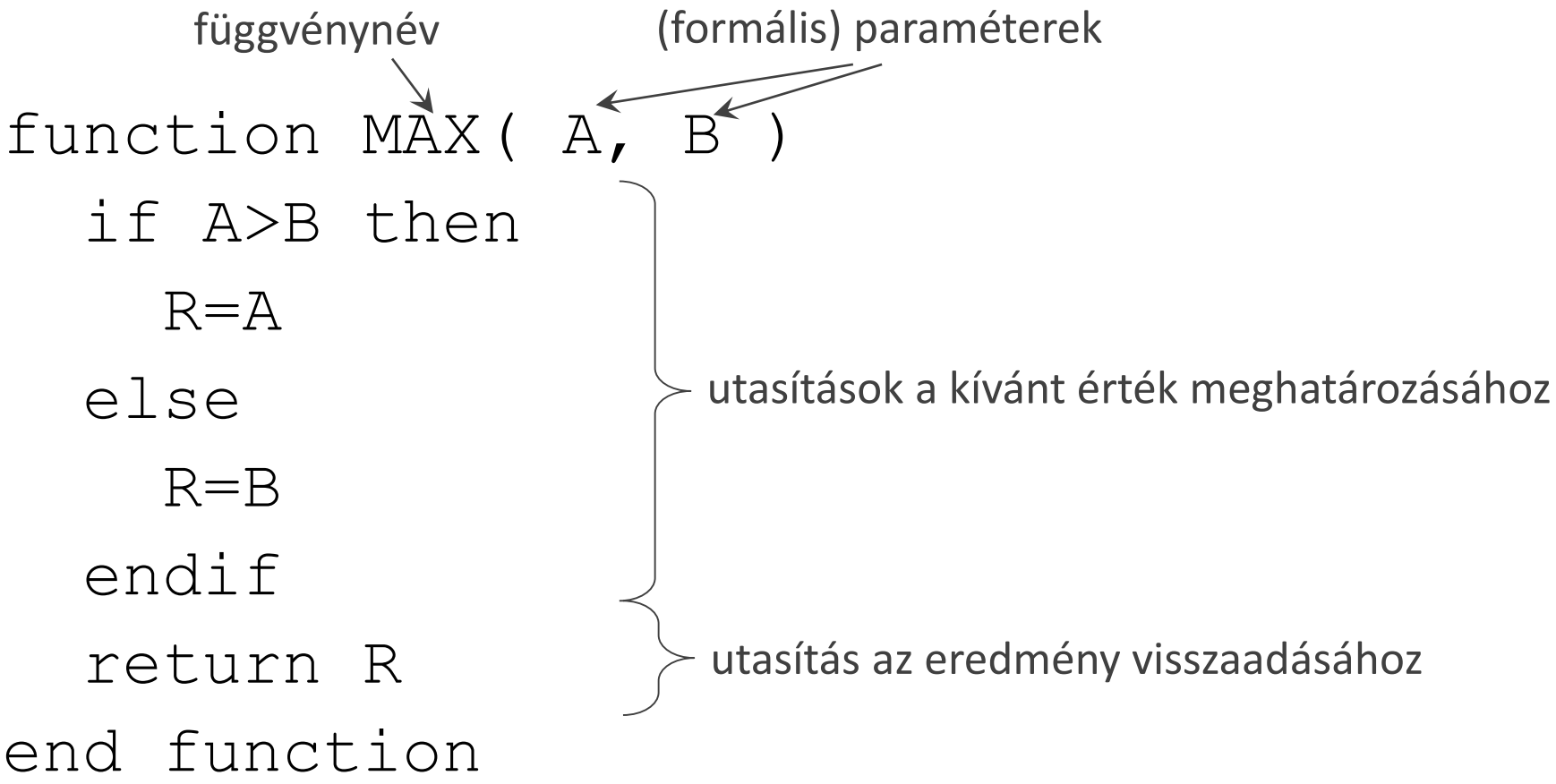
```
procedure SIGN (x)
   if x>0 then
     output ”Pozitív”
   else
     output ”Nem pozitív”
   endif
end procedure

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)

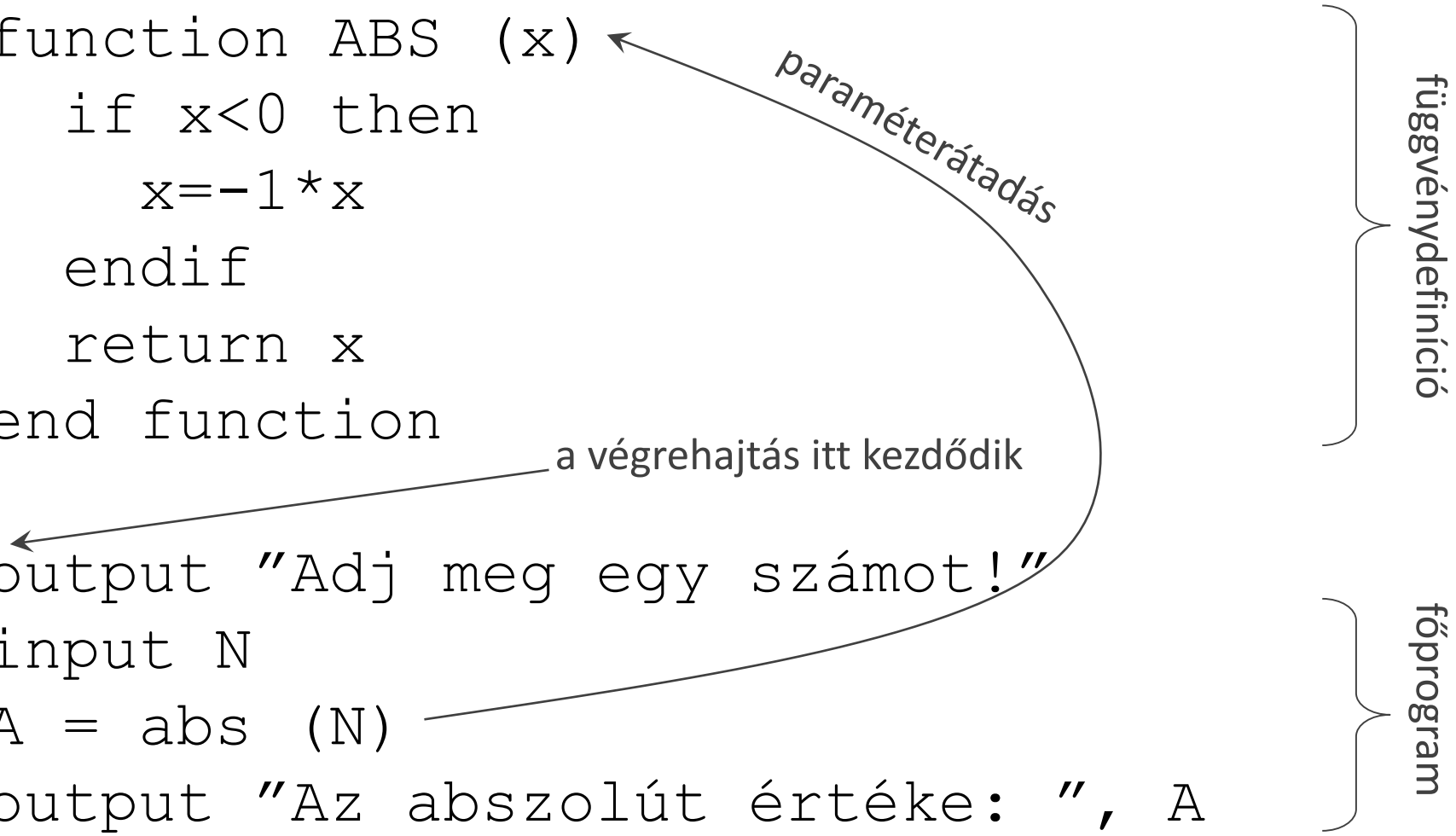
a visszaadott érték a c változóban tárolódik el

Függvény példa

Egy másik függvény az abszolút érték kiszámításához

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

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

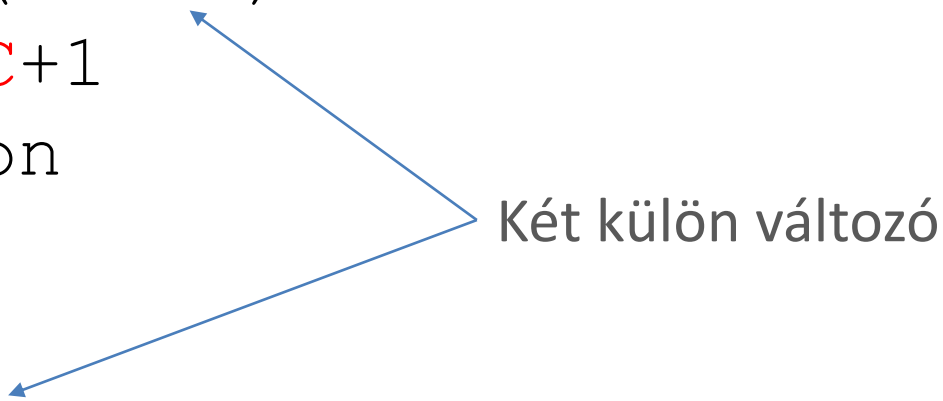


Hatáskör

- 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 )
   ABC = ABC+1
end function

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
end function

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

- Hogyan változtak 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
end function

input Max
i=0
j=0
while j<=Max do
   i = VALTOZTAT(i)
   j=j+i
   output j
enddo
```

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

Feladat: alprogram

- Írj egy algoritmust pszeudokóddal, amely tartalmaz egy függvényt két (paraméterként megadott) szám átlagának meghatározásához!
- Írj egy pszeudokódot, amely tartalmaz egy eljárást az NxN-es szorzótábla kiírásához!
  Például, ha N=4:

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

Feladat: alprogram

- 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

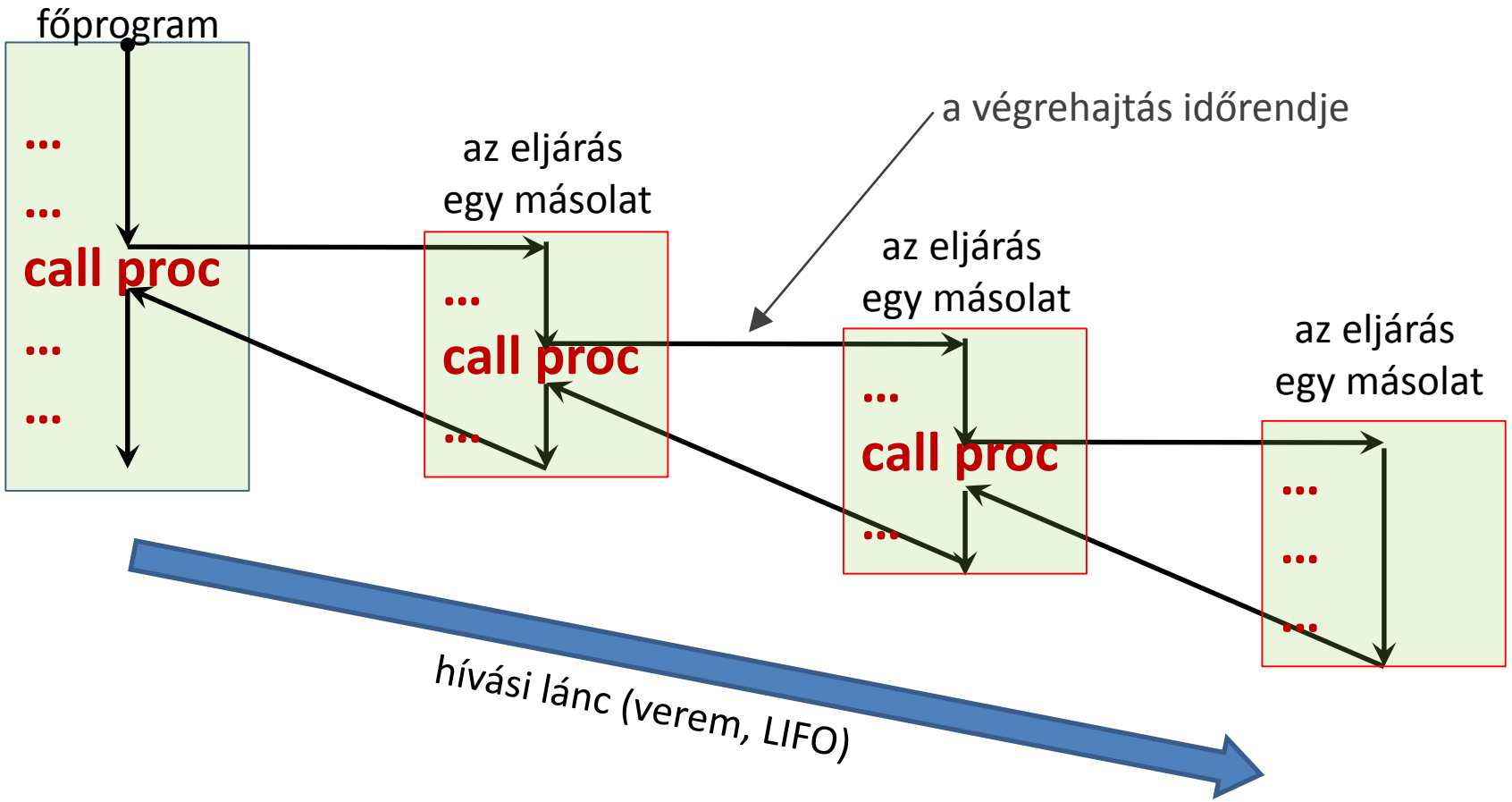
Feladat: alprogram

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!

Rekurzió

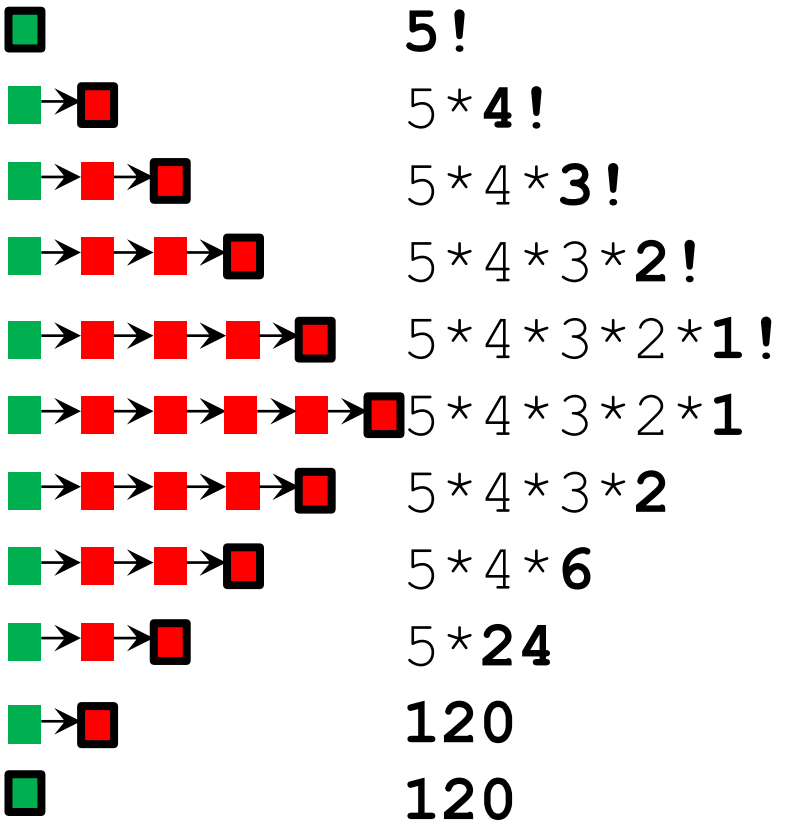
- Amikor egy alprogram saját magát hívja meg



Rekurzió példa

- Faktoriális:
  - 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
end function
output fakt(5)
```



Feladat: rekurzió

```
procedure KONVERTAL ( N , B )
   if N<>0 then
      call KONVERTAL ( [N/B] , B)
      output a%b
   endif
   output NEWLINE
end procedure

call KONVERTAL ( 16 , 8)
call KONVERTAL ( 19 , 2)
```

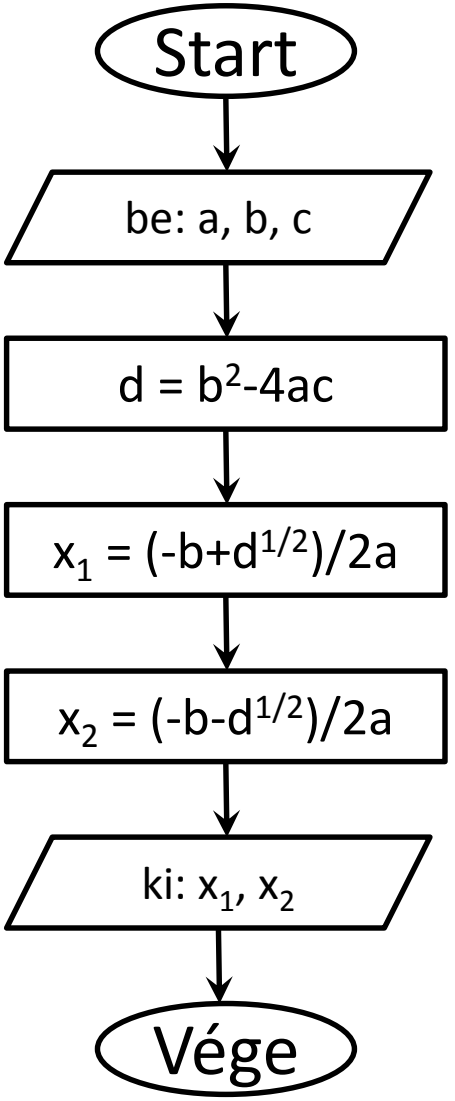
a hányados egészrésze

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

A tesztelési stratégia fejlesztése

Tesztelési stratégia példa

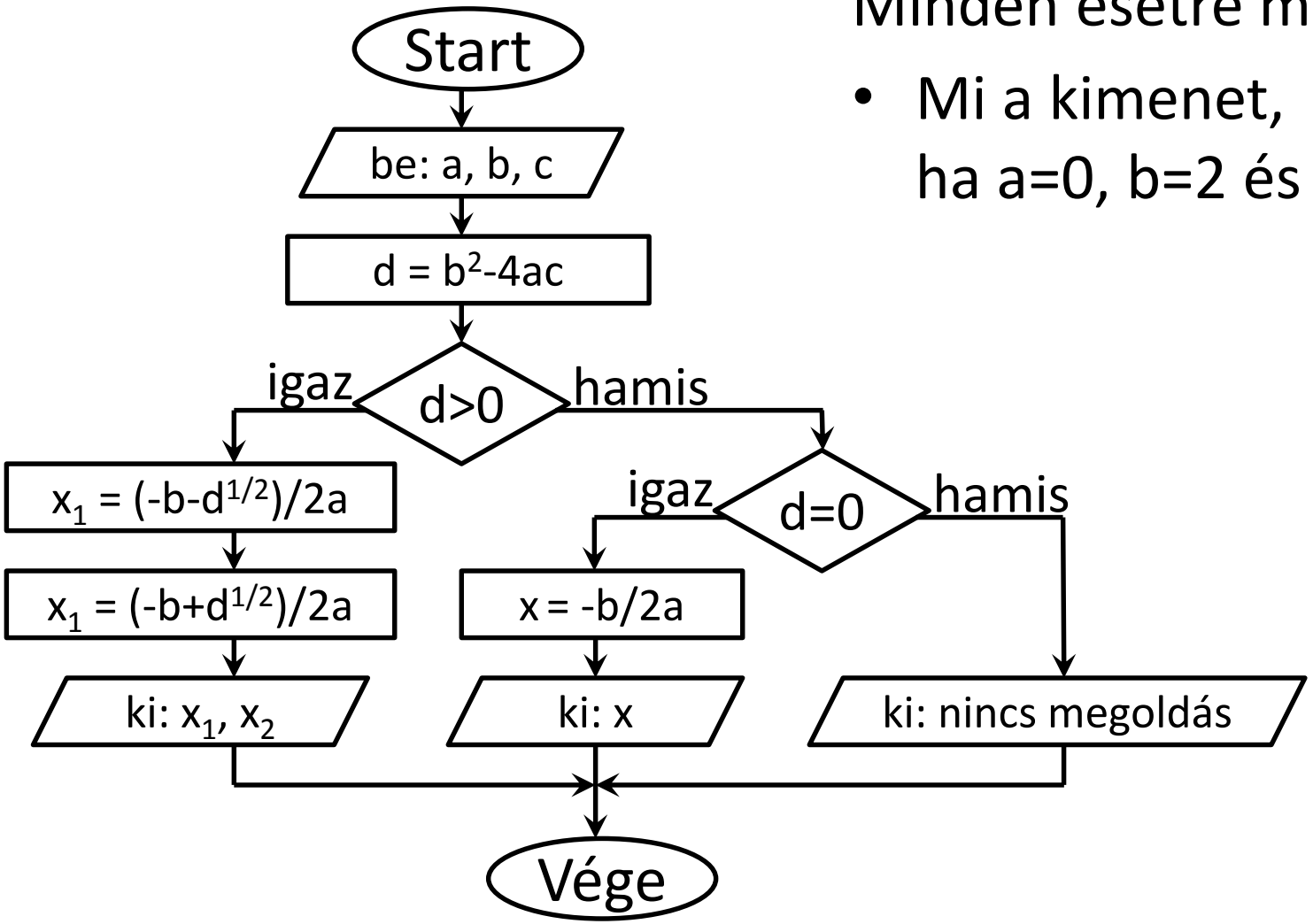
- Másodfokú egyenlet megoldása
- Általános alak: $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$
- Bemeneti paraméterek: a, b, c
- Megoldás: $\mathbf{x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}}$



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?

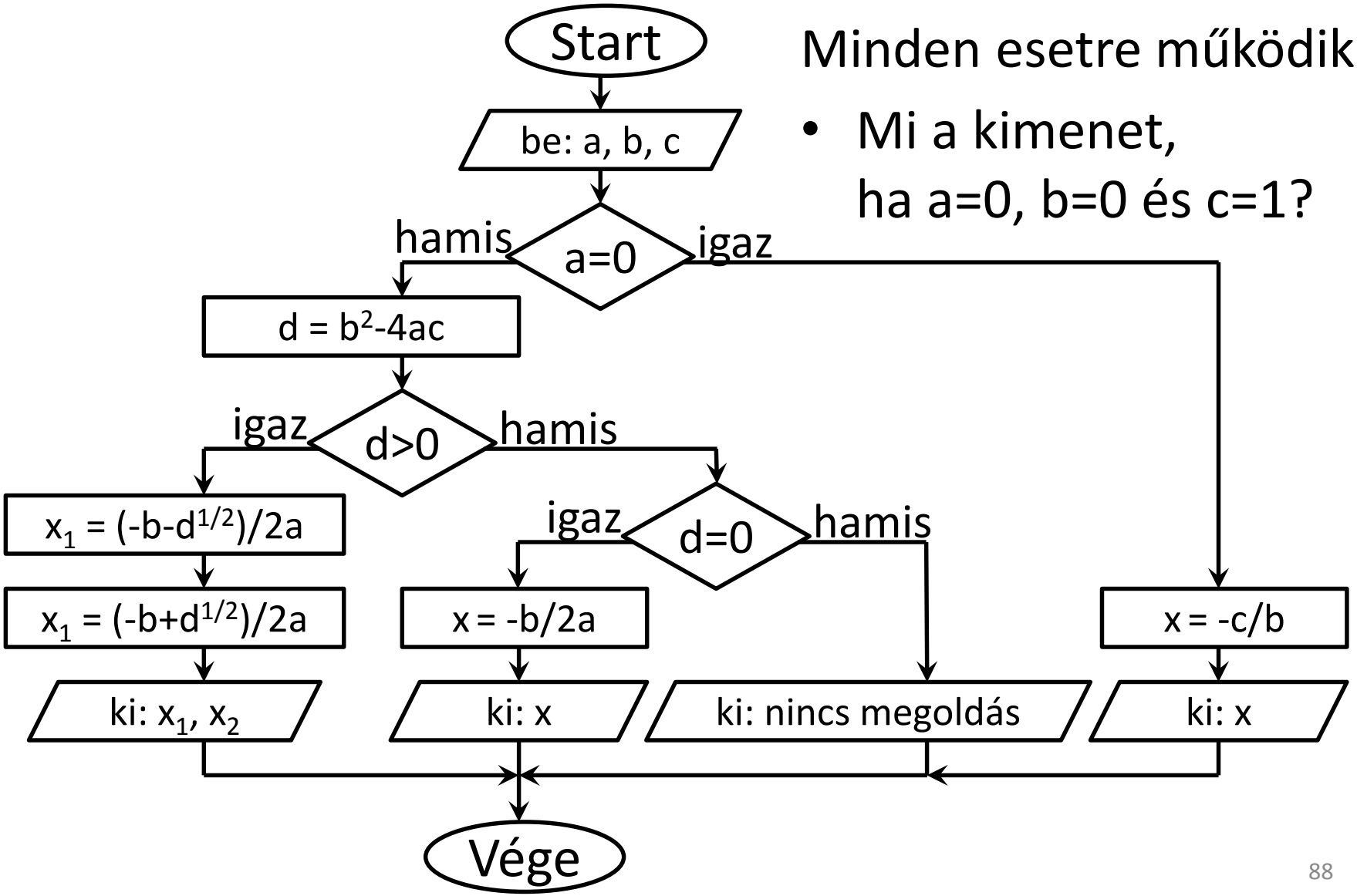
Tesztelési stratégia példa



Minden esetre működik?

- Mi a kimenet,
  ha a=0, b=2 és c=6?

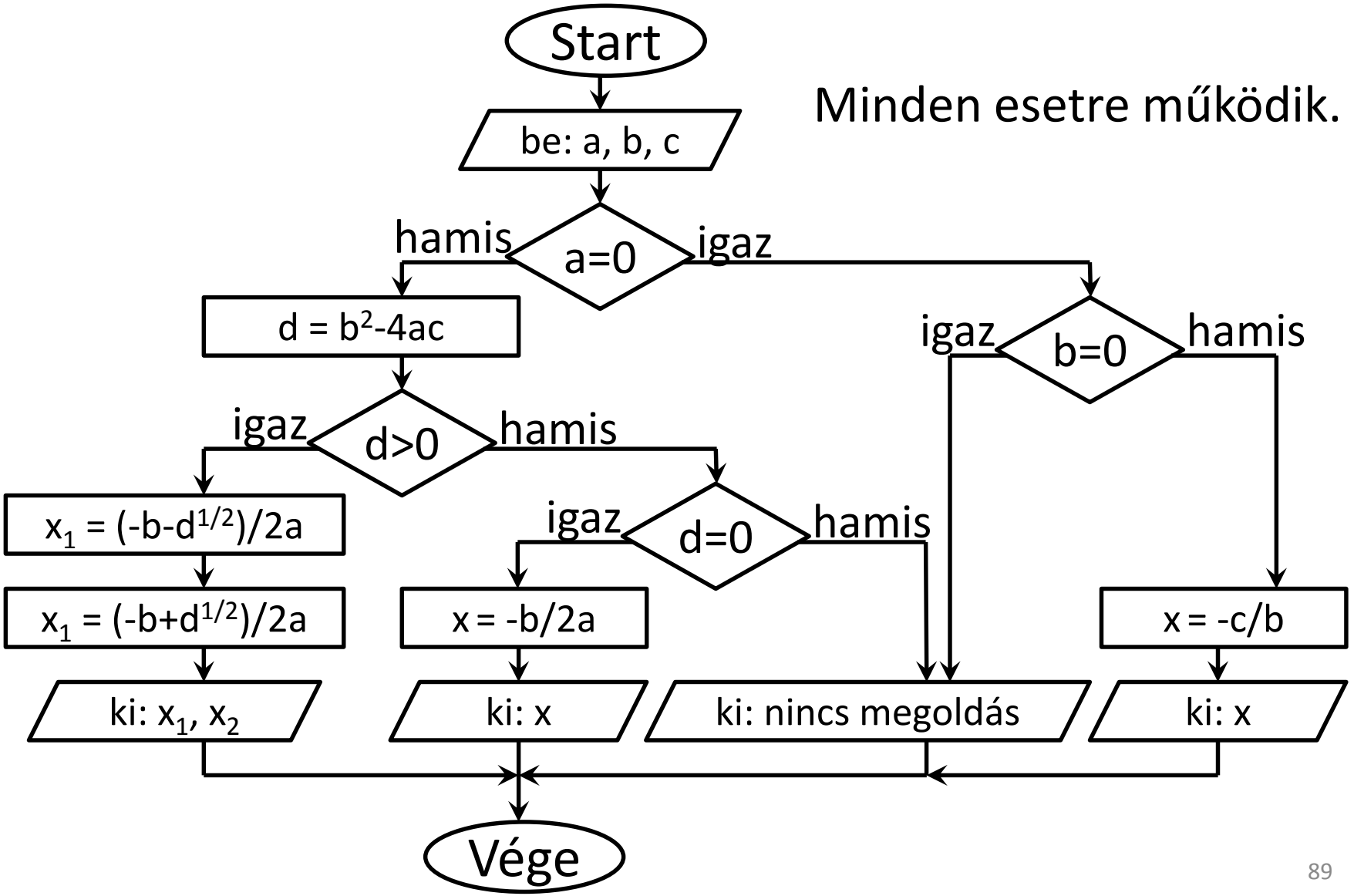
Tesztelési stratégia példa



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:

```
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
```

Minden esetre működik.

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

Tesztelési stratégiát alakítottunk ki.

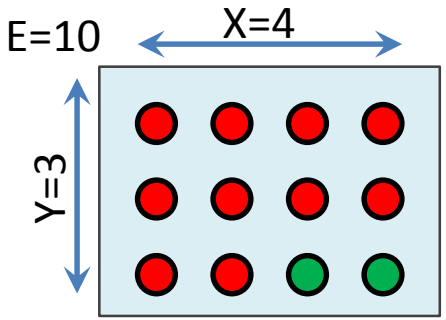
A használt tesztelési stratégia

a	b	c	Magyarázat	OK
3	7	2	általános eset (nem 0, d>0)	✓
0	2	6	**a** nulla (elsőfokú)	✓
2	0	5	**b** nulla (x^2=-c/a)	✓
1	2	0	**c** nulla (x[ax+b]=0)	✓
0	0	1	több nulla (nem egyenlet)	✓
1	2	2	**d**<0 (nincs megoldás)	✓
1	2	1	**d**=0 (csak egy megoldás)	✓
-2	-3	-9	negatív bemenetek	✓
2,3	4,2	0,83	nem egész bemenetek	✓
0,00001	1000000	1	extrém kicsi/nagy értékek	✓

Feladat: tesztelési stratégia

Ültetési rend:
Egy téglalap alakú terület rácspontjain székeket helyezünk el. Minden sor X széket tartalmaz. Hány sor szükséges E ember számára?

```
input E
input X
Y = E/X
output Y
```



- Hozz létre tesztelési stratégiát a fenti algoritmushoz!
- Az E és X mely értékei elfogadhatóak?
  (Mikor szolgáltatja az algoritmus az elvárt eredményt?)

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=[N/B]
enddo
output R
```

A program kódolása és tesztelése

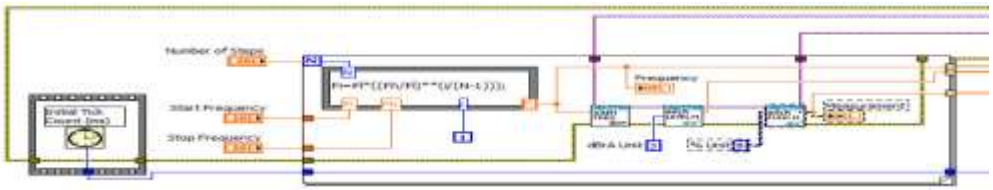
Forráskód létrehozása
egy valós programnyelven

A programozás szintjei

Magas szintű programnyelvek

Alacsony szintű programnyelvek

…



4GL SELECT name FROM people WHERE age=20

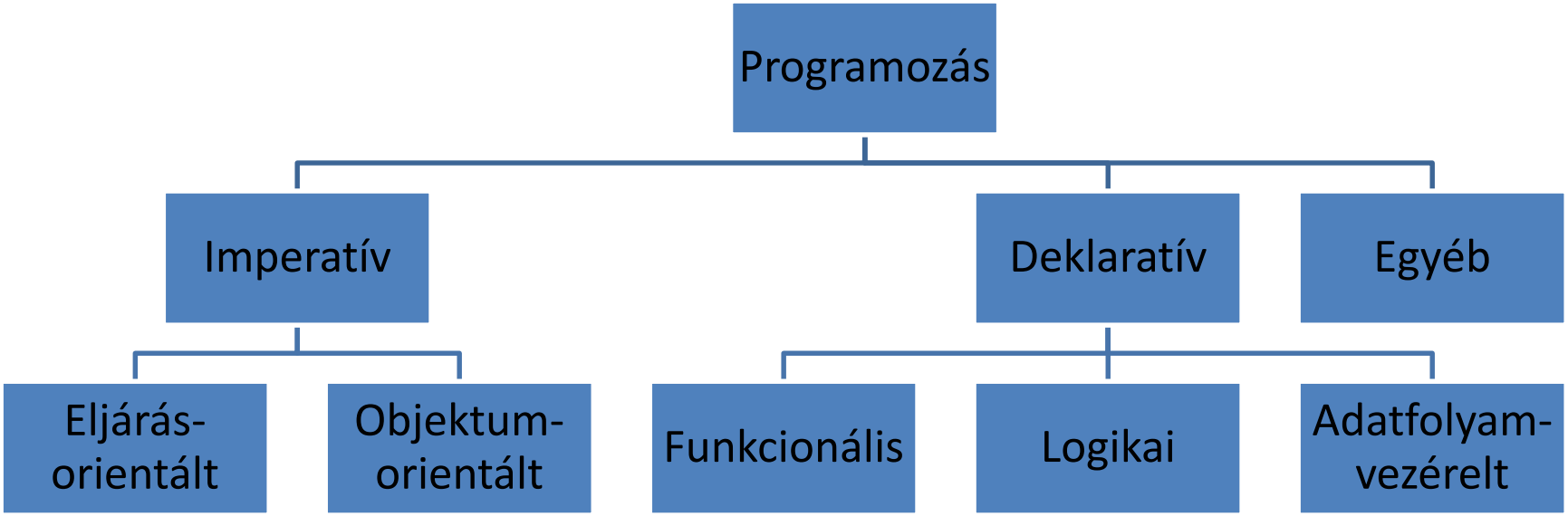
3GL if (x==0) printf(”zero”);

2GL SAVE: STA [$410A, X] assembly

1GL 10010110 11101000 10101101 10000111 gépi kód

idő

Programozási paradigmák



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

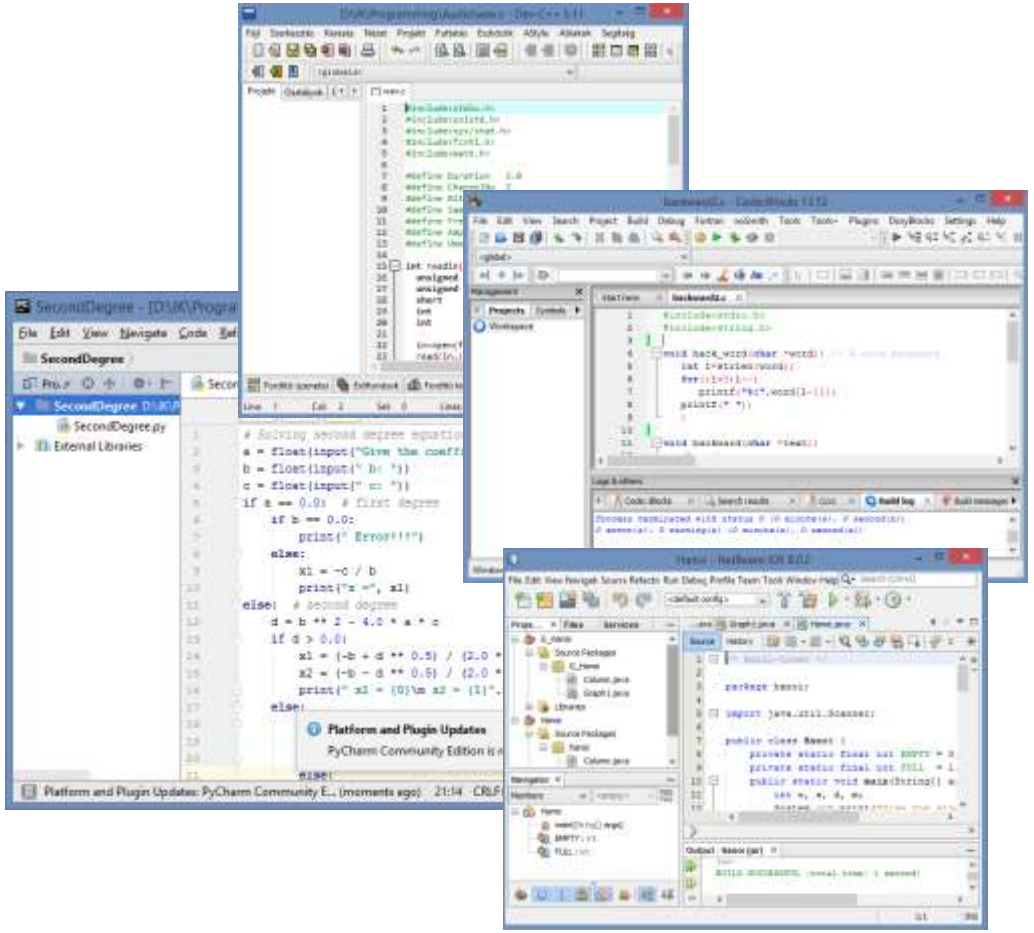
Integrált fejlesztői környezet

- 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



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
end function
```

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*N
      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;
```

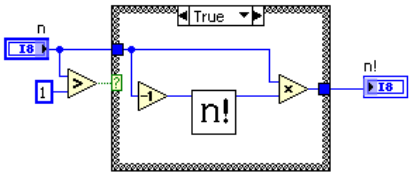
C:

```
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 kitevőre emeli a `B` számot!

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

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 (5.0) 01000000101000000000000000000000
  - sztring (”5”) 0011010100000000

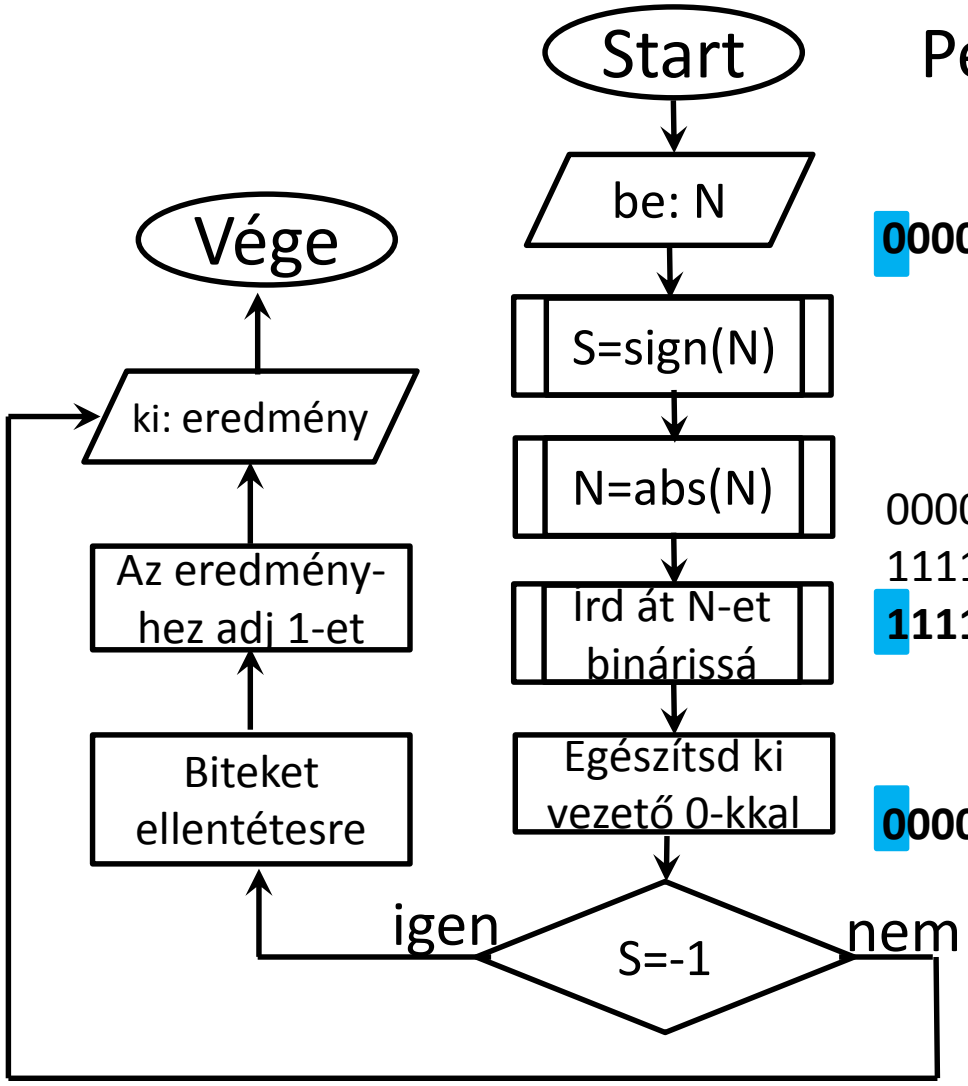
Fixpontos reprezentáció

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?

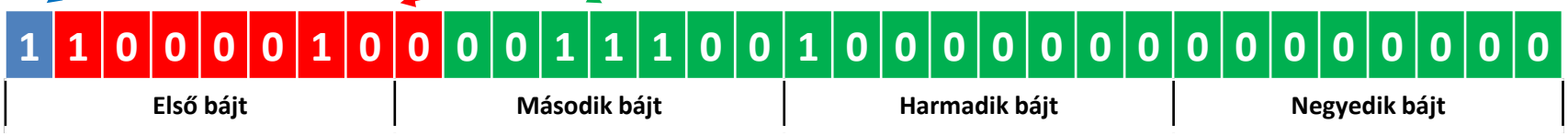
```
10000000 00000000 00000010 01001001
```

Lebegőpontos reprezentáció

Szabványosított (IEEE 754) technika valós számok (törtek, nem csak egészek) tárolására a számítógépen

Lépések egy 4 bájtos példában:

- -19.5625_{10} decimális valós
- -10011.1001_2 bináris valós
- $-1.00111001*2^4$ normálforma: számjegyek * alap $^{\text{kitevő}}$
- $(-1)^{1} * (1+.00111001) * 2^{131-127}$ $(-1)^S * (1+M) * 2^{K-127}$



Lebegőpontos reprezentáció

Reprezentáció hossz (bitszám)	4 bájt (egyszeres)	8 bájt (dupla)
Előjelbitek száma	1	1
Kitevőbitek száma	8	11
Hasznos bitek szám	23	52
Kitevőeltolás	127	1023
Minimum abszolút érték	**$1.175*10^{-38}$**	**$2.225*10^{-308}$**
Maximum abszolút érték	**$3.403*10^{+38}$**	**$1.798*10^{+308}$**
Pontosság (!)	**~7 helyiérték**	**~16 helyiérték**

Lebegőpontos reprezentáció

Egy intervallum nem minden eleme reprezentálható

Pl.: 1.0000000000; 1.0000001192; 1.0000002384; 1.0000003576

Kerekítést alkalmazunk (0.7 → 0.69999998807907…)

Speciális értékek:

- `+0.0` (pl. 1.0-1.0)
- `-0.0` (egyenlő a +0.0-val; pl. -1.0*0.0)
- `+Inf` (+∞, pl. 1.0/0.0)
- `-Inf` (-∞, pl. -1.0/0.0)
- `NaN` (nem szám, pl. +Inf*0.0)

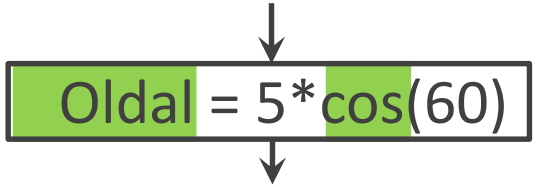
A forráskód egységei és elemei

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

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

Kulcsszó, azonosító, megjegyzés

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



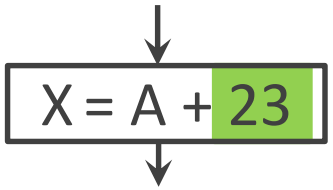
- Megjegyzés
  Olyan szöveg a forráskódban, amely csak a programozónak (olvasónak) szól, nincs hatása a program futásra

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
```



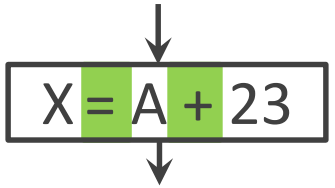
Változó

- 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);
```

A Python programozási nyelv

A nyelv néhány főbb eleme („erősen lebutítva”):

- Adattípusok: int, float, str, …
- Konstansok: 21, 21.0, ”21”, …
- Operátorok: +, -, *, /, %, **, // ,=, ==, >=, <=, !=, <<, >>, ~, &, |, not, or, and, in, is, …
- Elágaztatás: if-(elif)-else
- Ciklusszervezés: while, for
- Input/output: print(), input()
- Megjegyzés: ,’’’komment’’’, #komment
- stb. …

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
```

Algoritmusok Pythonban

- Értelmezz Python kódokat!
- Próbáld ki Pythont a PyCharm IDE segítségével!
- Olvass Python dokumentációt!
- Végezz kisebb módosításokat működőképes Python programokon!
- Írj át az eddig elkészített **összes** pszeudokódokat Pythonra!
- Gyakorold az algoritmizálást a Python segítségével!
- Írj saját algoritmust/programot Pythonban!

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)

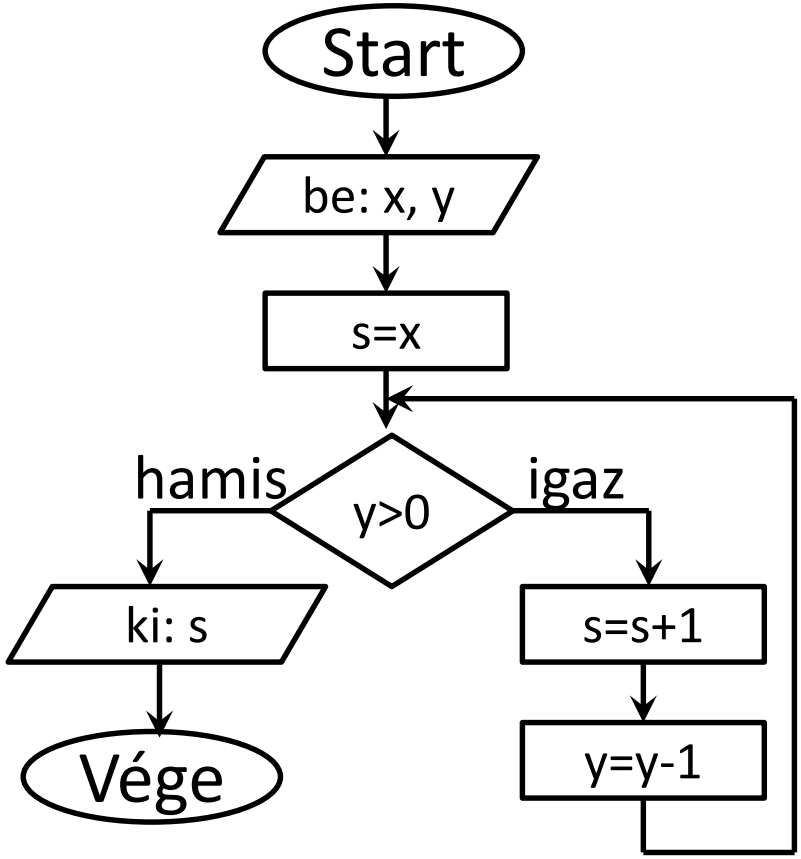
Megoldások

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**

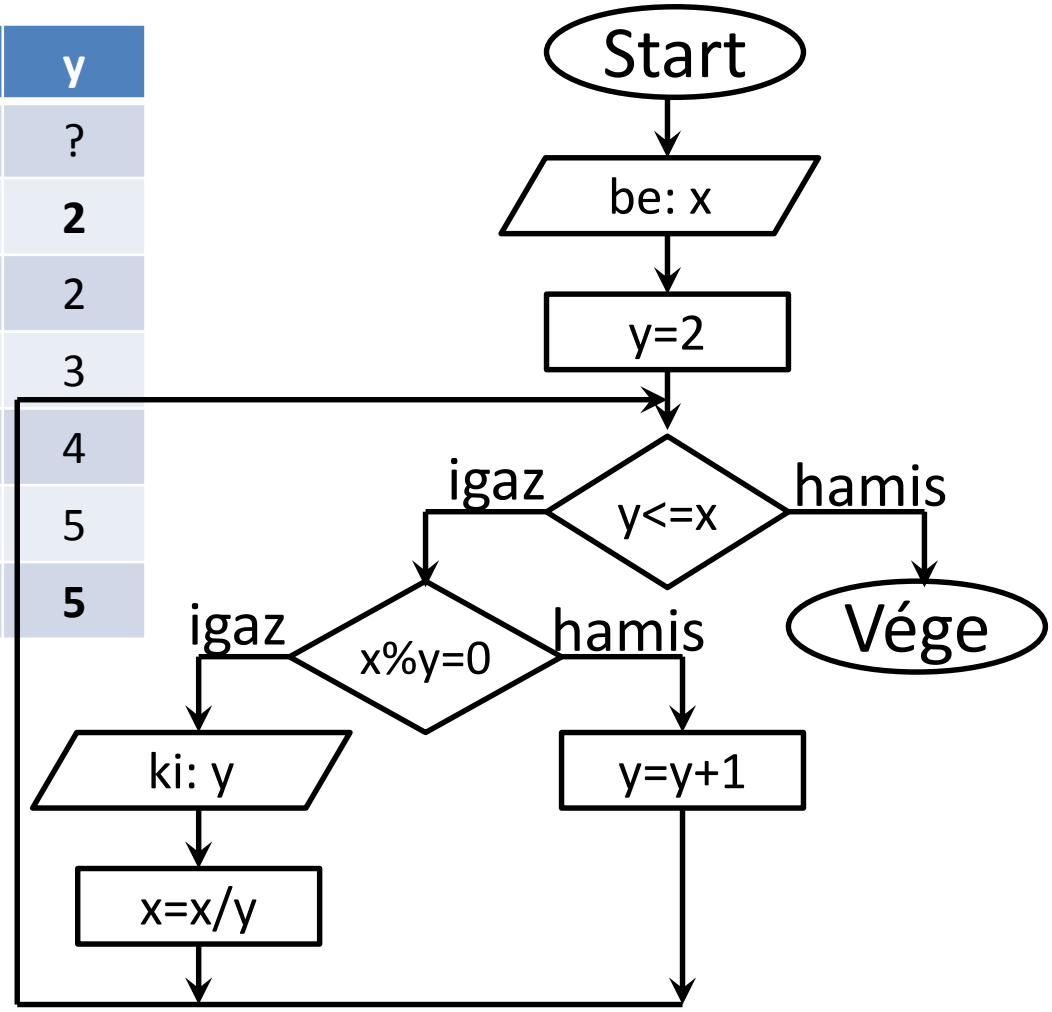


- Kimenet: 9
- 5 kifejezés kiértékelés
- Összeadás: s=x+y
- Módosítások:
  s=x → s=1 s=s+1 → s=s+x

Megoldás: folyamatábra

- Nyomkövetés:

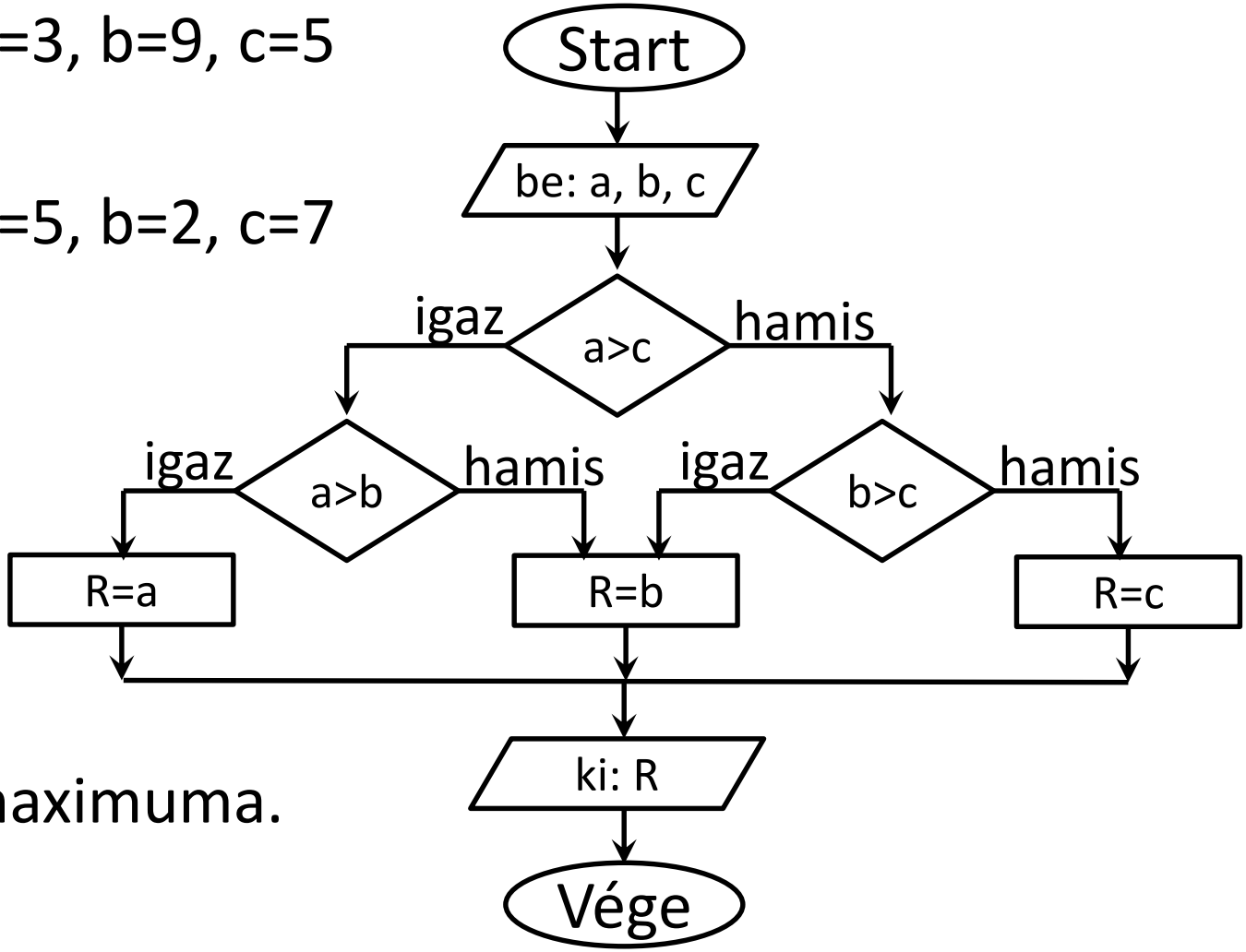
x	y
10	?
10	**2**
5	2
5	3
5	4
5	5
1	**5**



- 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

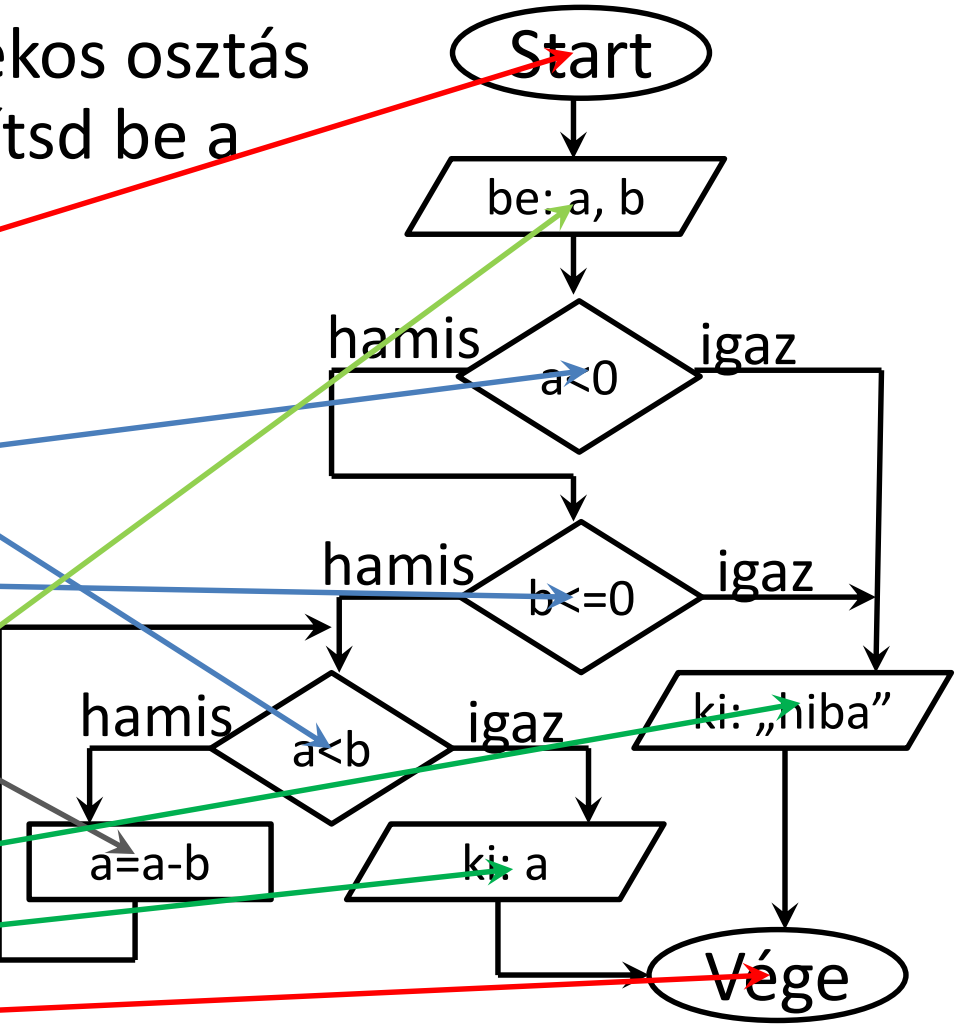


- A számok maximuma.

Megoldás: folyamatábra

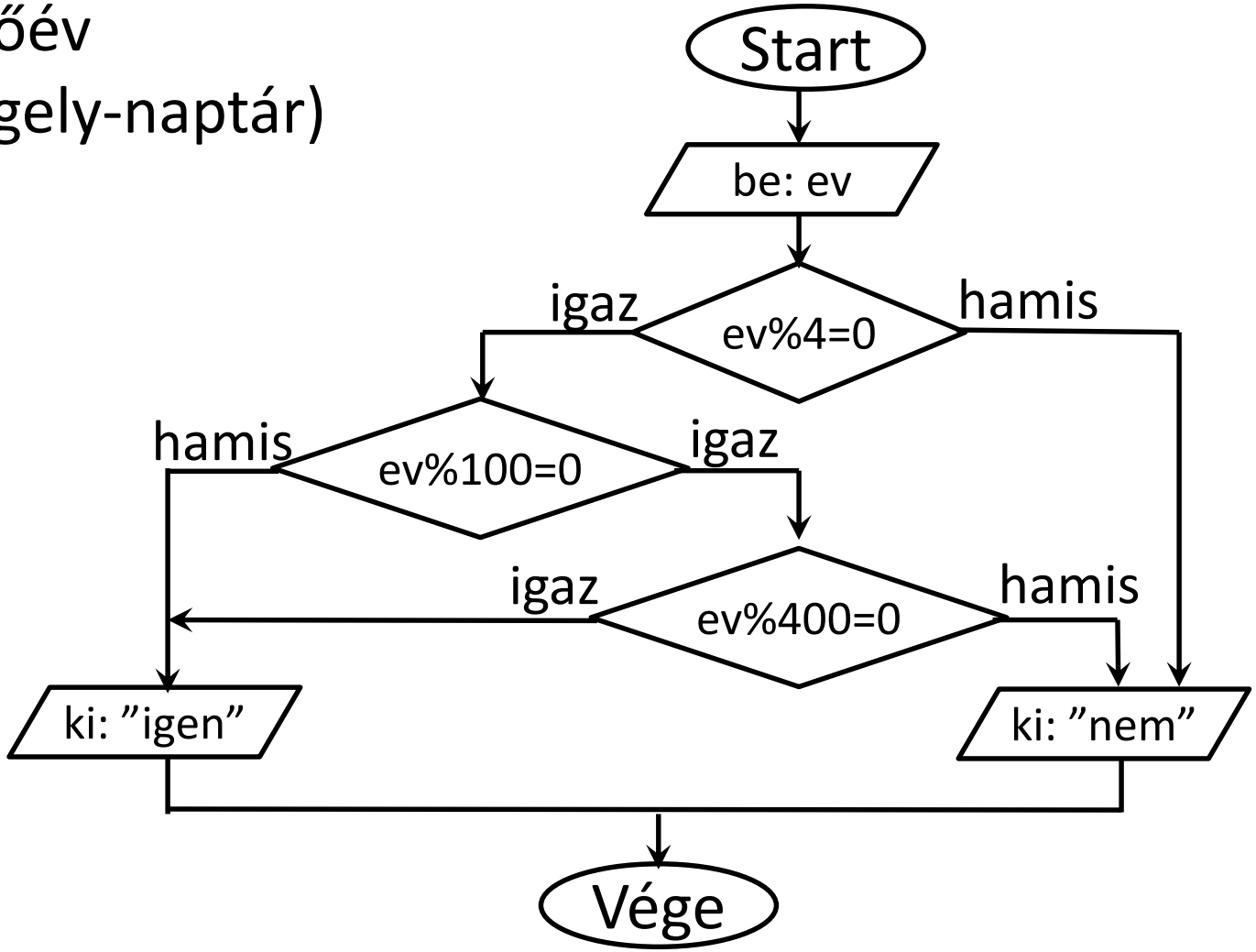
Ez a folyamatábra a maradékos osztás műveletét írja le. Helyettesítsd be a kifejezéseket!

- Start
- a<b
- a<0
- b<=0
- a=a-b
- be: a, b
- ki: „hiba”
- ki: a
- Vége



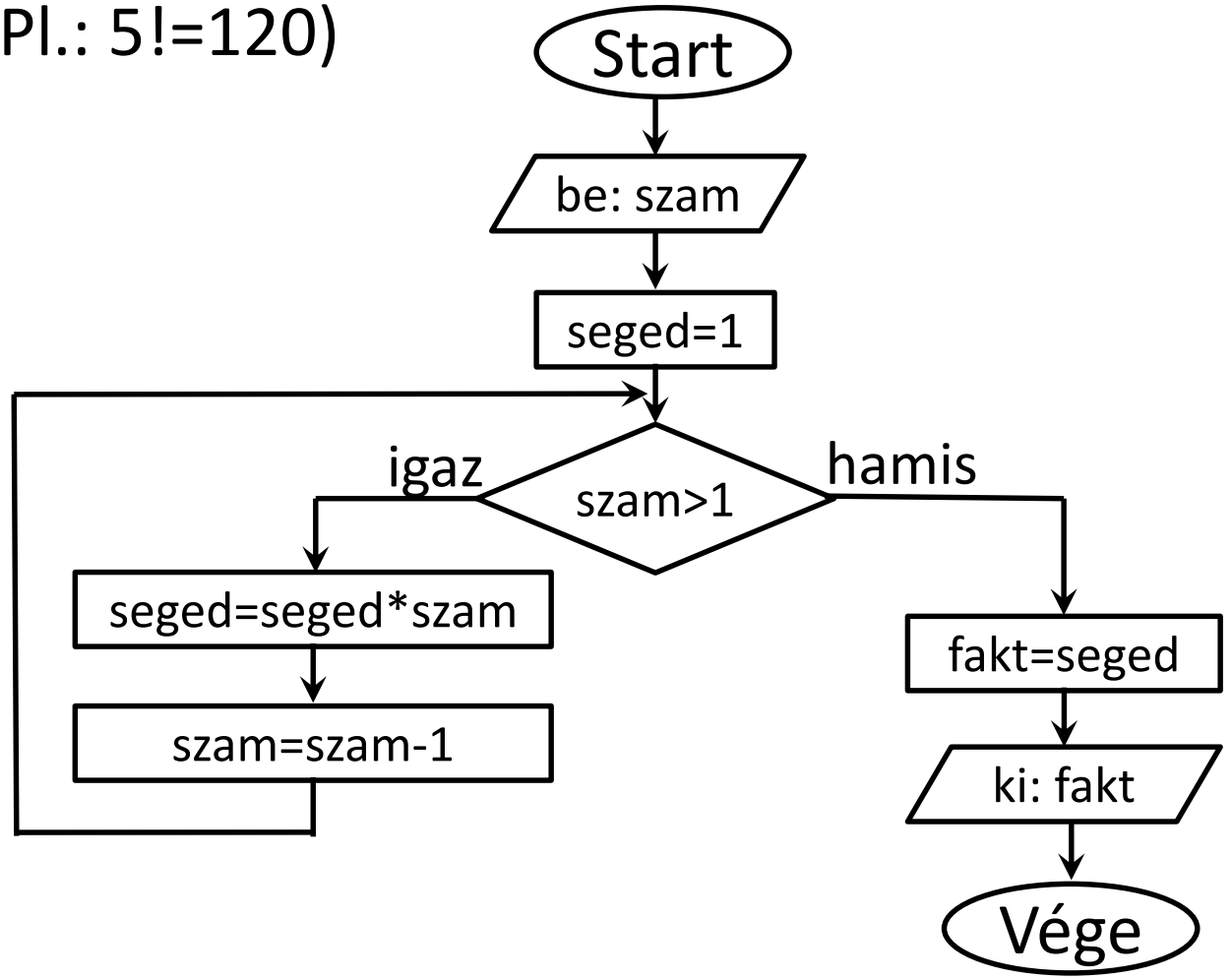
Megoldás: folyamatábra

- Szökőév
  (Gergely-naptár)



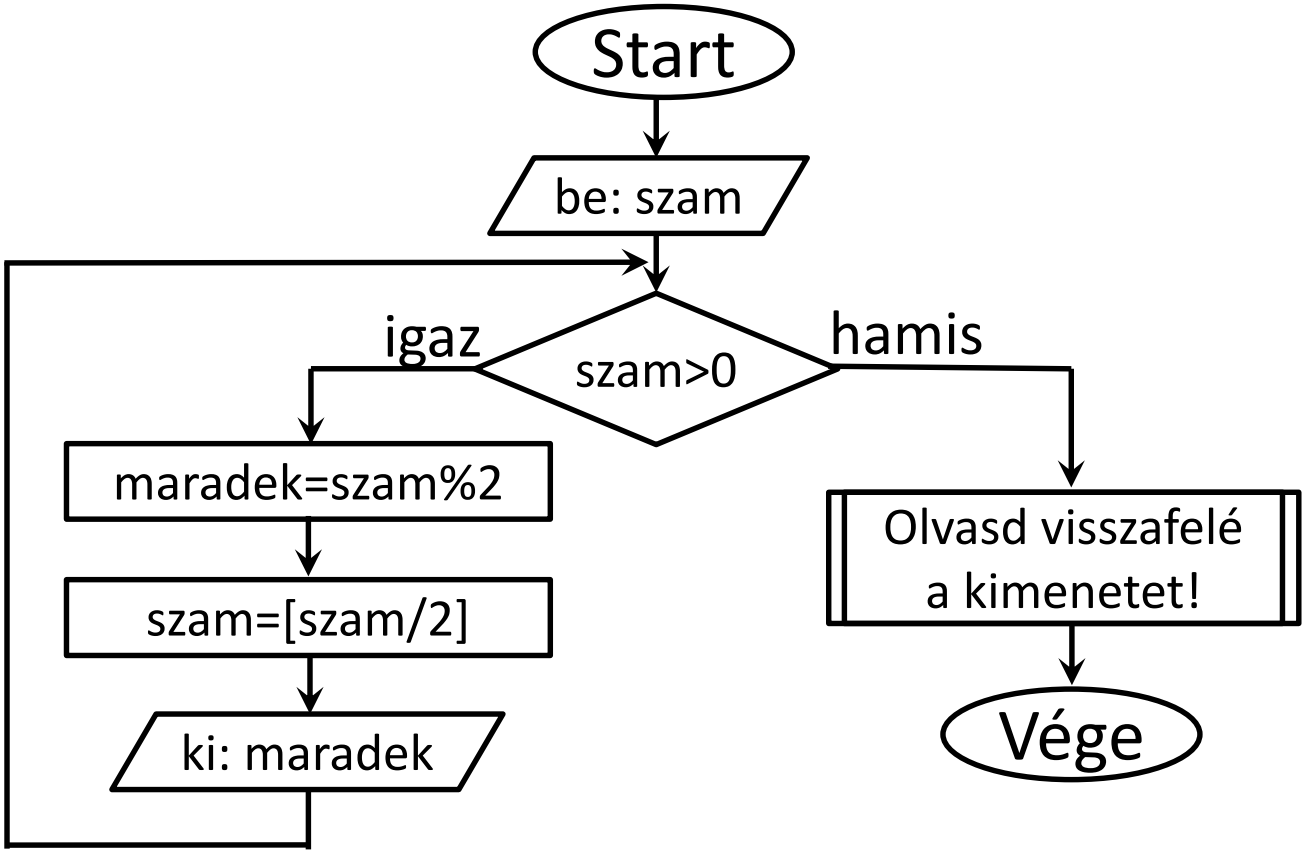
Megoldás: folyamatábra

- Faktoriális (Pl.: 5!=120)



Megoldás: folyamatábra

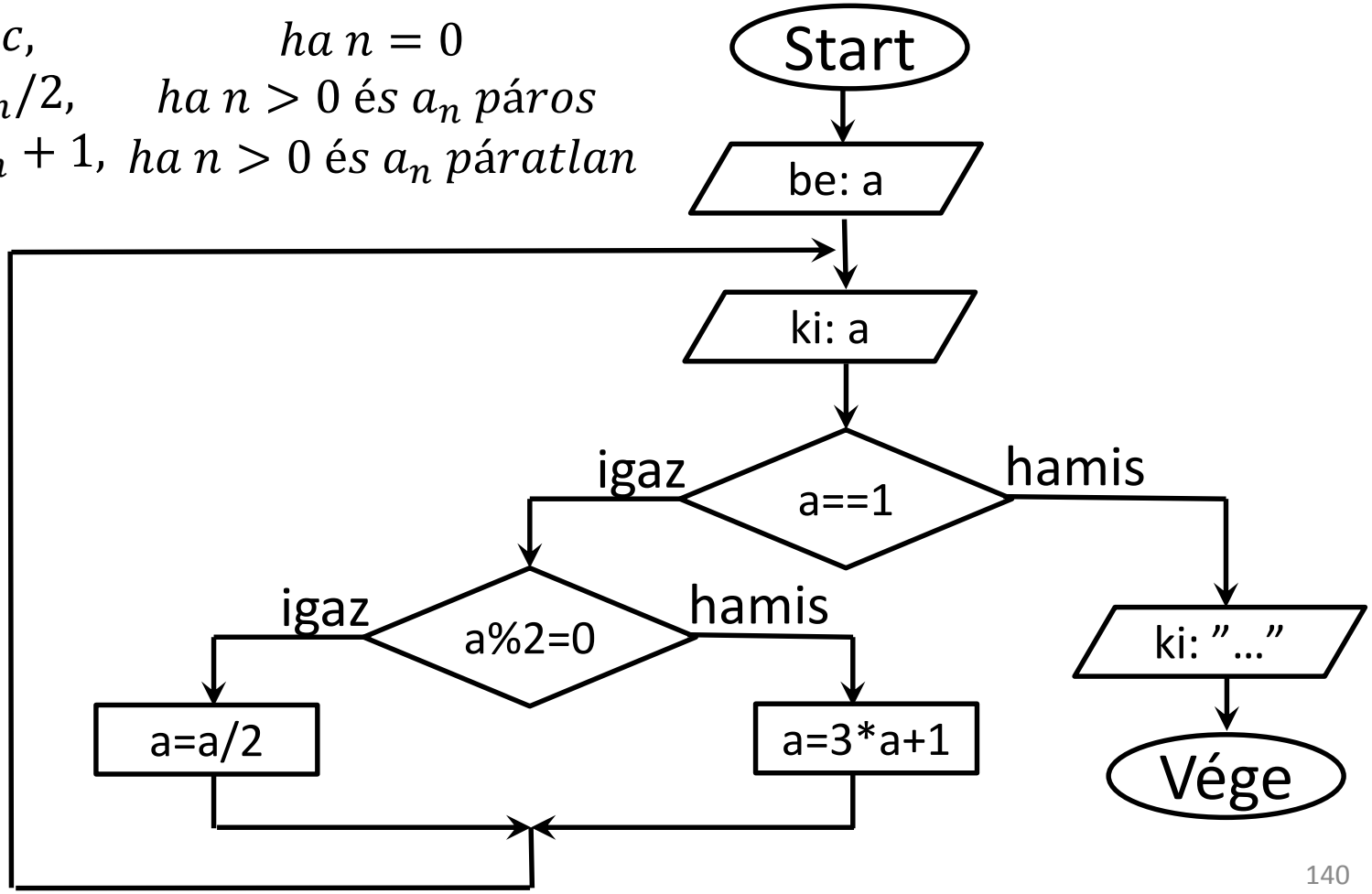
- Számrendszerváltás (10 → 2)
  Pl.: $21_{10} = 10101_2$



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

```
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
```

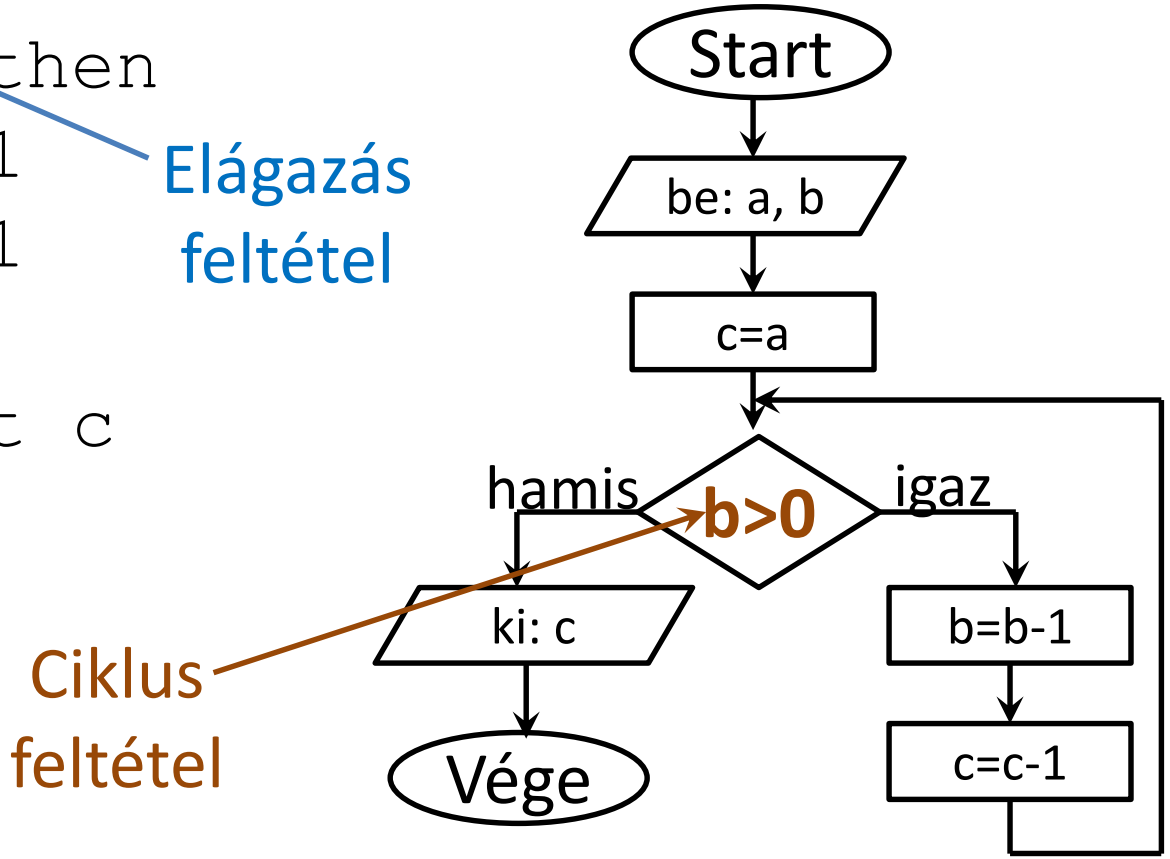
Alternatív algoritmusok!

Megoldás: pszeudokód

```
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: pszeudokód

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

- Nyomkövetés:

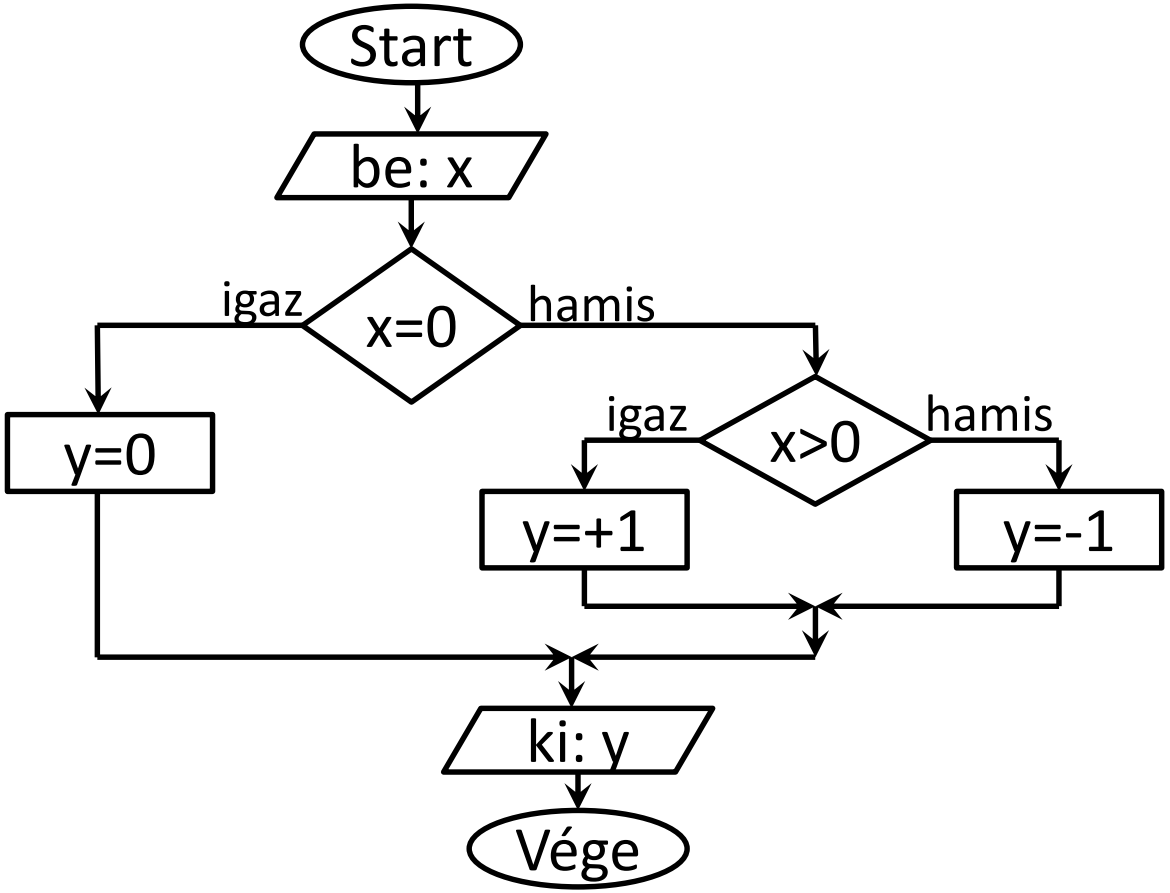
a	b	c
7	3	?
7	3	7
7	2	6
7	1	5
7	0	**4**

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

Megoldás: pszeudokód

Előjel függvény:

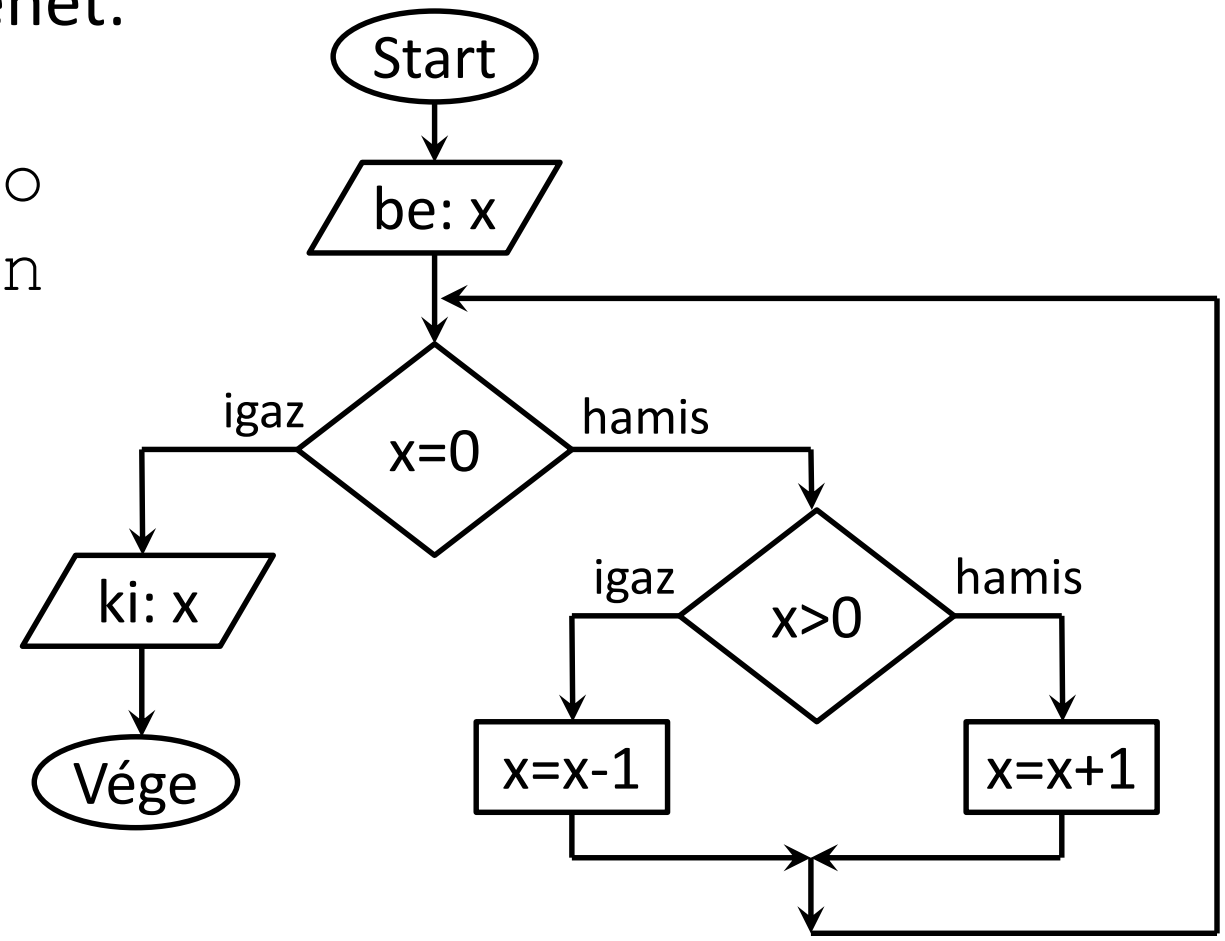
```
input x
if x=0 then
   y=0
else
   if x>0 the
      y=+1
   else
      y=-1
   endif
endif
output y
```



Megoldás: pszeudokód

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

```
input N
R=0
while N>0 do
   R=R*10+N%10
   N=[N/10]
enddo
output R
```

- Nyomkövetés:

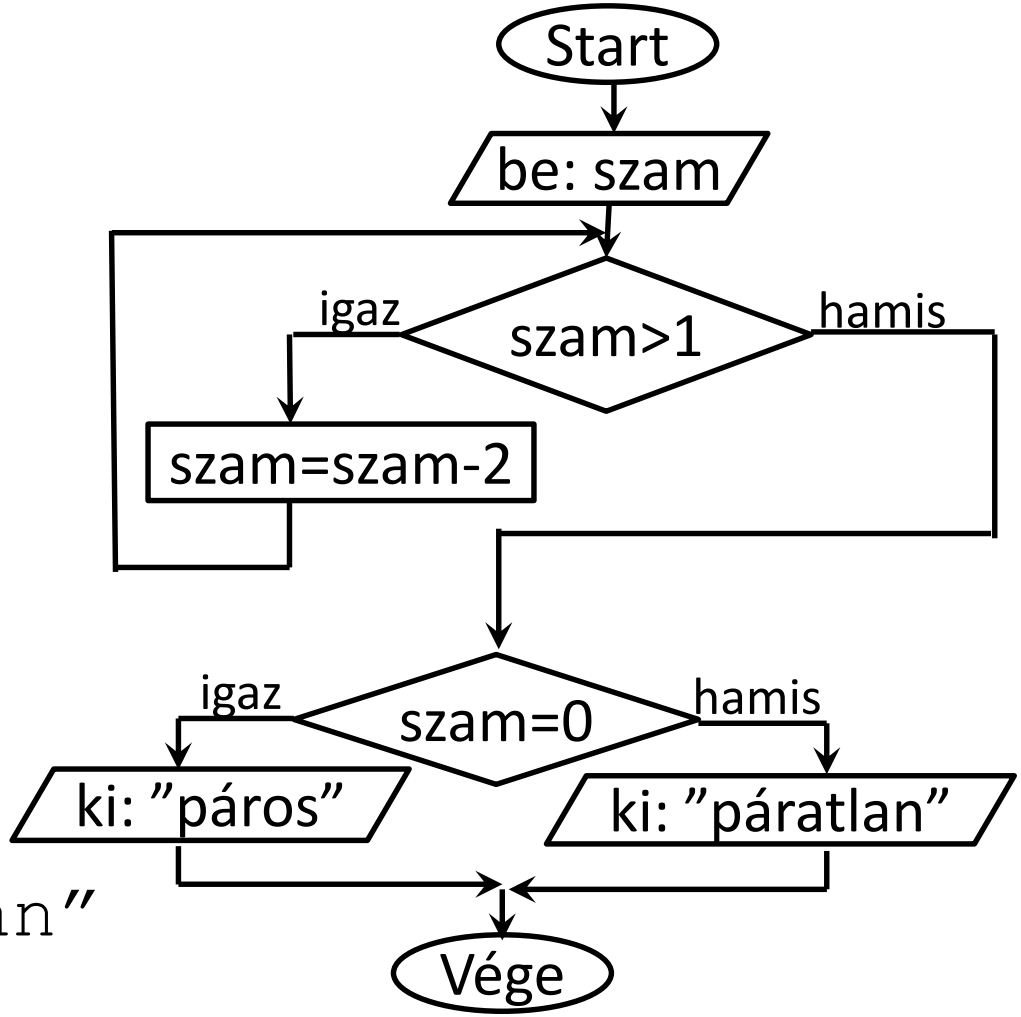
N	R
73251	0
7325	1
732	15
73	152
7	1523
0	**15237**

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

Megoldás: pszeudokód

Páros vagy páratlan:

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



Megoldás: pszeudokód

- Hatványozás ($R=A^K$):

```
input A
input K
R=1
while K>0 do
   R=R*A
   K=K-1
enddo
output R
```

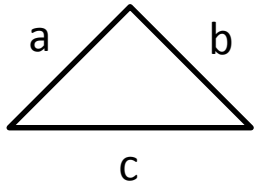
- Prímszám-e?

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

Kész, vége!

Megoldás: pszeudokód

- 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: pszeudokód

- Fibonacci-sorozat 1000-nél kisebb elemei

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

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

Megoldás: pszeudokód

- Tömbelemek átlaga
  (innentől feltesszük: `T` egy `N` elemű inicializált tömb)

```
sum=0
i=0
while i<N do
   sum=sum+T[i]
enddo
atlag=sum/N
output atlag
```

Megoldás: pszeudokód

- Elemkeresés tömbben

```
input Keresett
i=0
while i<N and T[i]<>Keresett do
   i=i+1
enddo
if T[i]=Keresett then
   output ”benne van”
else
   output ”nincs benne”
endif
```

Megoldás: pszeudokód

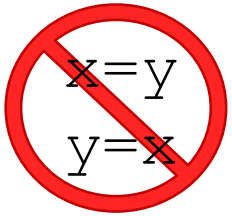
- Tömbelemek maximuma és annak indexe

```
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: pszeudokód

- 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: pszeudokód

- Buborékrendezés

```
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: 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
end function
```

```
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
end function

input Max
i=0
j=0
while j<=Max do
   i = VALTOZTAT(i)
   j=j+i
   output j
enddo
```

- Max=5 → 1 1 2 2 3 3 4 4 5 5
- Egyeseket és nullákat ír ki.
- A 0→1 és 1→0 váltogatás

Megoldás: alprogram

- Átlag függvény

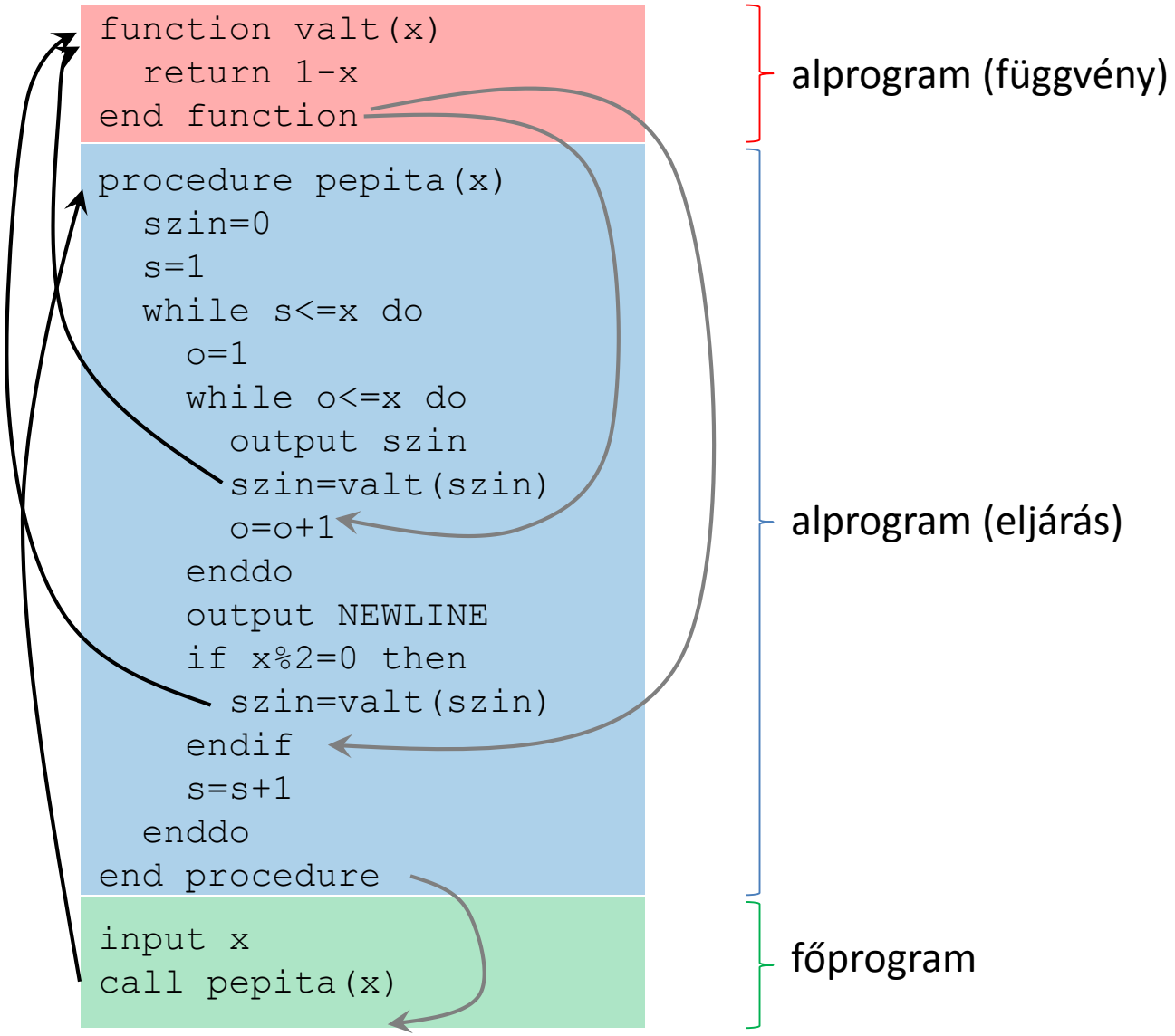
```
function atl(a,b)
   return (a+b)/2
end function


x=atl(2, 34)
output x
```

- Szorzótábla eljárás

```
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
end procedure
call SzT(8)
```

Megoldás: alprogram



Megoldás: alprogram

```
function sec(ora, perc, masodperc)
   return (ora*60+perc)*60+masodperc
end function
procedure ido(t)
   s=t%60
   m=[t/60]%60
   h=[t/3600]
   output h, m, s
end procedure
call ido(sec(12,15,30))
```

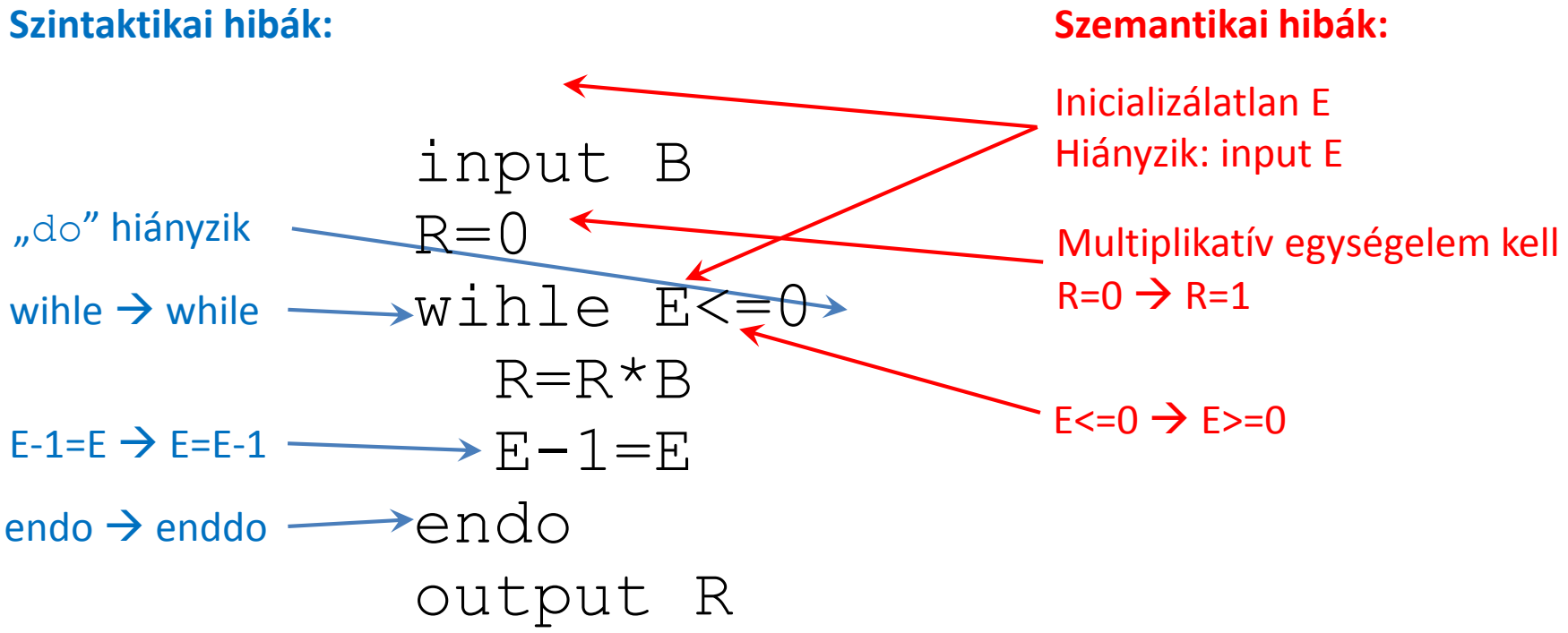
Megoldás: tesztelési stratégia

N	B	Eredmény	OK?
11	2	1011	✓
2	5	2	✓
0	9	0	✓
43	0	-	×
-10	4	-14	✓
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 egész szám (-Limit(B)<N<Limit(B))

```
input N
input B
R=0
P=1
while N<>0 do
   R=R+(N%B)*P
   P=P*10
   N=[N/B]
enddo
output R
```

Megoldás: szintaxis és szemantika



Megoldás: adatreprezentáció

- Népesség több, mint 7 000 000 000.
  Előjel nélküli fixpontos maximum 2^{32}-1 = 4294967295.
  Nem ábrázolható!
- -1:
  11111111 11111111 11111111 11111111
- 15908:
  00000000 00000000 00111110 00100100
- -666:
  11111111 11111111 11111101 01100111
- 10000000 00000000 00000010 01001001:
  2147483063

Megoldás: kifejezés

- ((9+((2*6)/3))>(8-7))
  érték: igaz
- (((2>3)&&((3*5)-(6/2)))>=(11%2))
  érték: igaz (C nyelv)
- * + 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
```