Algoritmikus gondolkodás

Varga Imre
Debreceni Egyetem, Informatikai Kar

2015.12.02

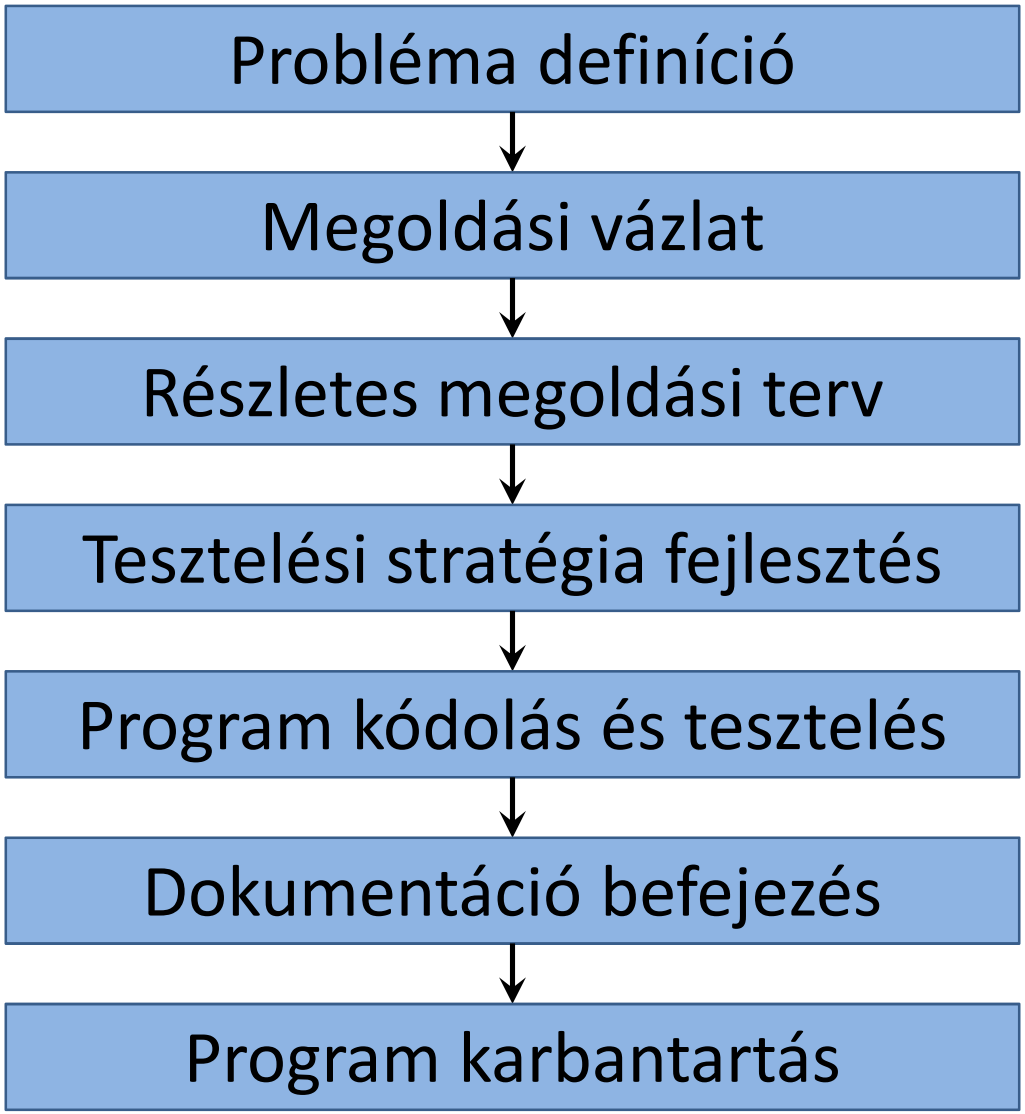
Témák

- Hogyan írjunk le és oldjunk meg problémákat?
- Hogyan osszunk fel egy problémát kisebbekre?
- Mi is az az algoritmus?
- Milyen tulajdonságai vannak?
- Hogyan írhatunk le algoritmusokat?
- Mit jelent a ‘program írás’?
- *És még sok minden más…*

Számítógépes probléma megoldás

Szorosan kapcsolódva a **szoftver életciklus**hoz

Számítógépes probléma megoldás



1: Probléma definíció

- 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 a félreérthetőséget
- A pontos módszer tartalmazza a jól meghatározott lépések sorozatát, amit **algoritmus**nak hívunk
- Különböző formalizmusokkal írható le

4: Tesztelési stratégia fejlesztés

- Ki kell próbálni az algoritmust több különböző input kombinációval hogy biztosak legyünk, hogy jó eredményt ad minden esetben
- Ezeket a különböző bemeneti adat kombinációkat **teszt eset**eknek nevezzük
- Ez nemcsak normál adatokat foglal magába, hanem extrém bemeneteket is, hogy teszteljük a határokat
- Komplett teszt esetekkel ellenőrizhetjük magát az algoritmust

5: Program kódolás és tesztelés

- 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
  - A program szövegét és annak magyarázatát
  - Felhasználói utasításokat
  - stb.

7: Program karbantartás

- 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 át átalakítás, bővítés lehet szükséges
- Ezek után frissíteni kell a dokumentációt is

Részletes megoldási terv

Algoritmus

Algoritmus

Terv jól ismert tevékenységek 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
- Tevékenységek véges sorozata
- Elérhető vele a cél, megadja a megoldást
- Hatékony, elegáns, egyszerű, …

Algoritmusok leírása

- **Verbális**
- **Folyamatábra**
- **Pszeudokód**
- Struktogram
- Grafikus
- Algebra-szerű
- Adat-folyam diagram
- Hierarchikus
- Táblázatos

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?
- …

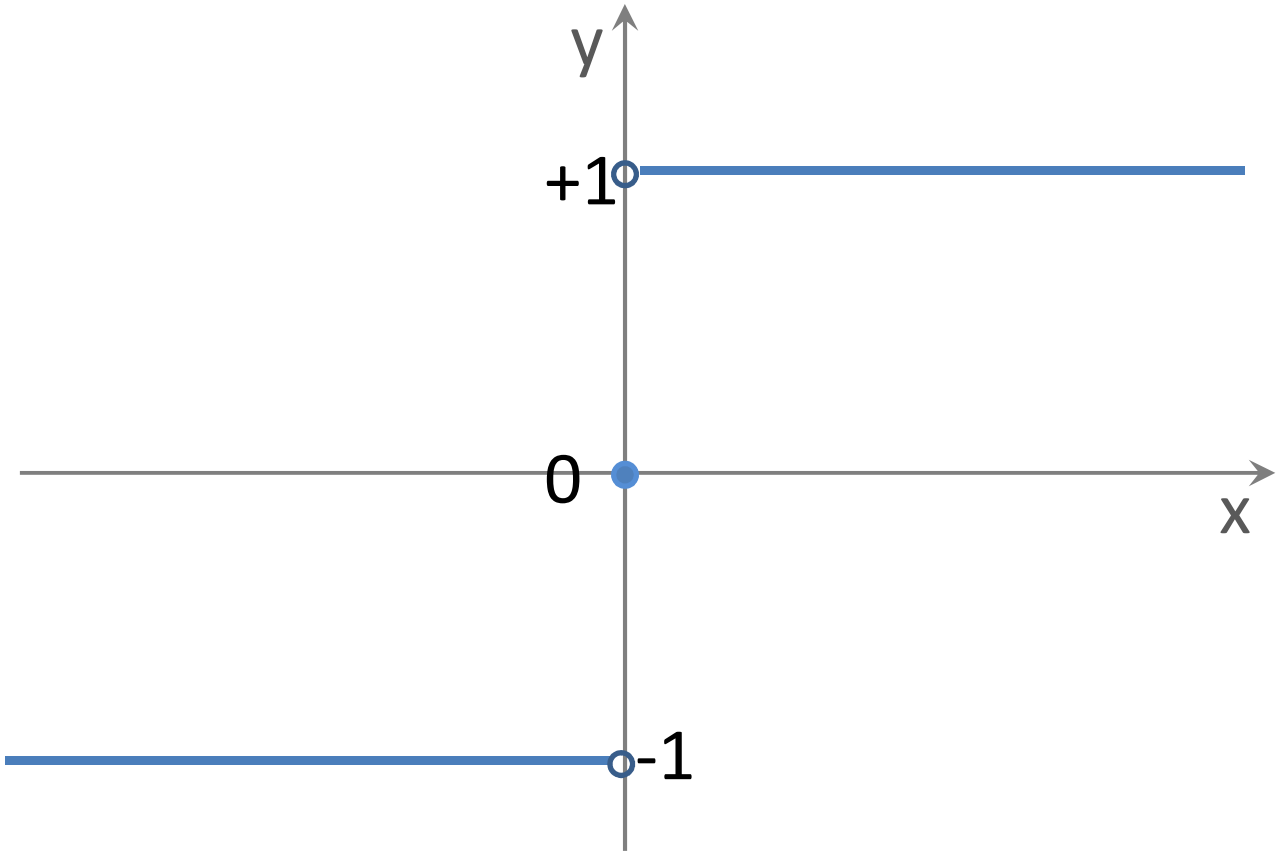
y=sign(x)

Verbális 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)

‘Algebra-szerű’ reprezentáció:

$x \in \Re$

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

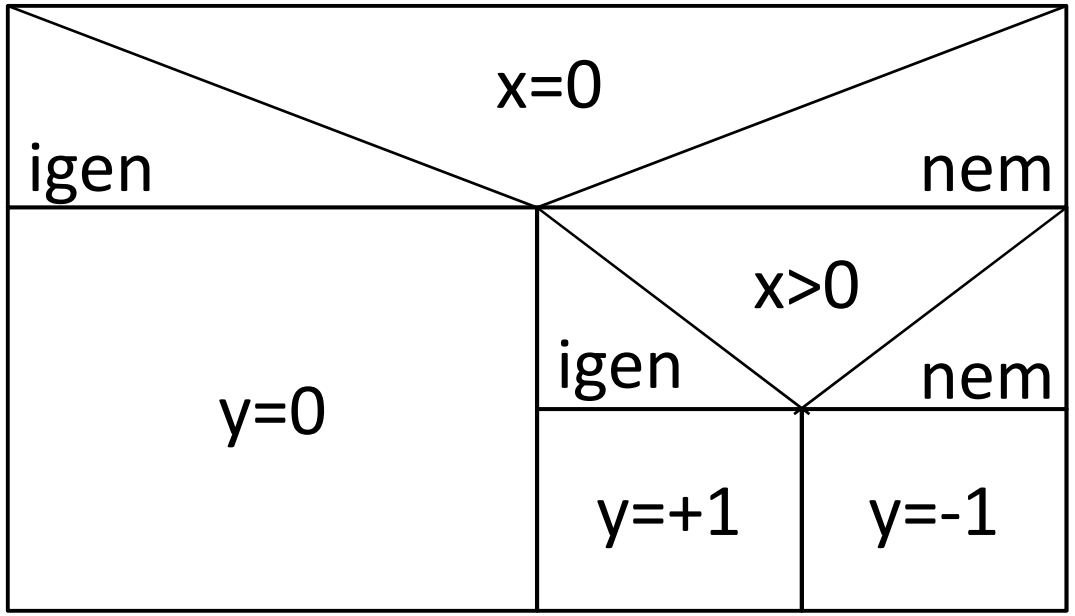
$\forall x,\ x>0 \Rightarrow y=+1$

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

$x=0 \Rightarrow y=0$

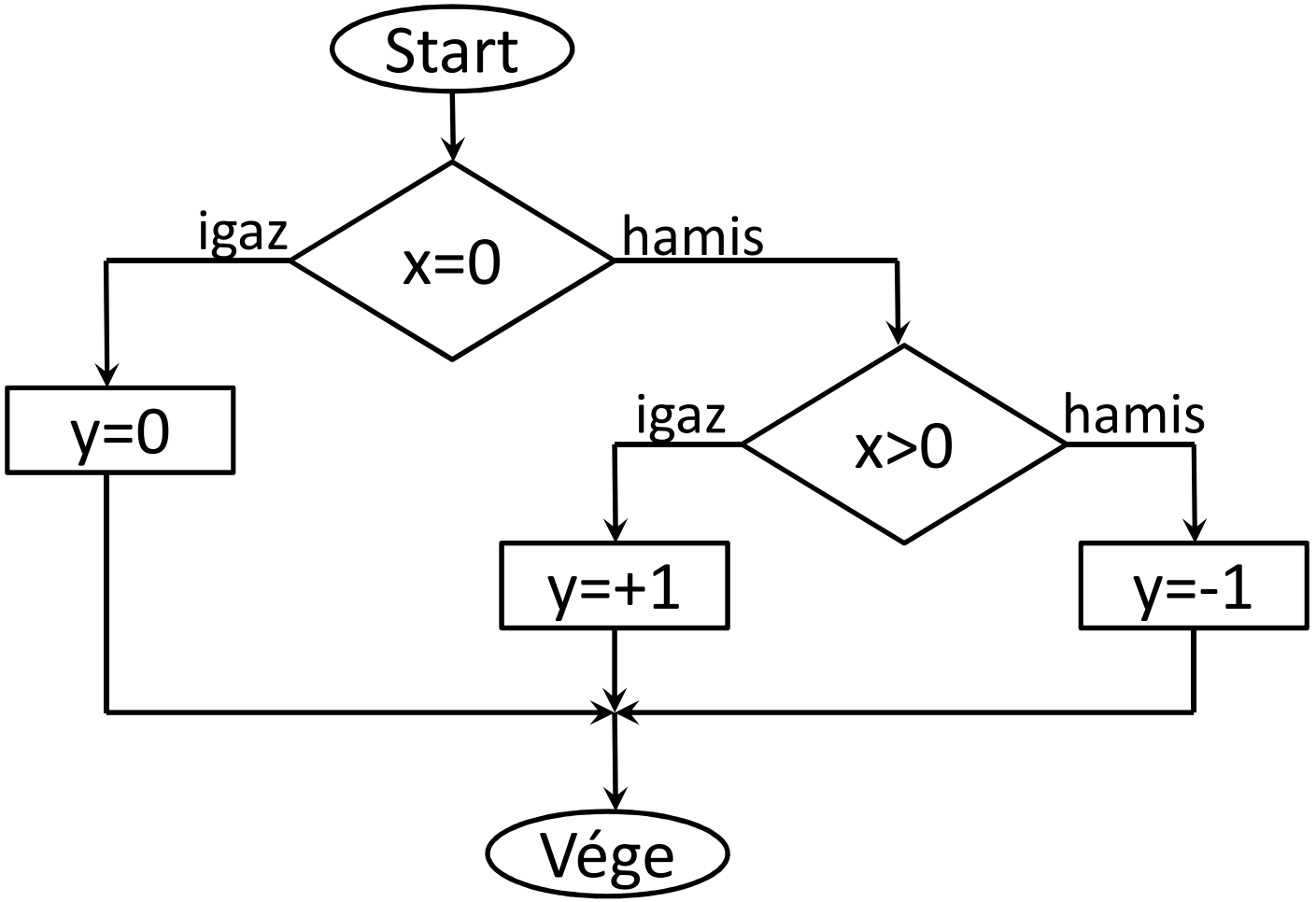
y=sign(x)

Struktogramos reprezentáció:



y=sign(x)

Folyamatábrás reprezentáció:



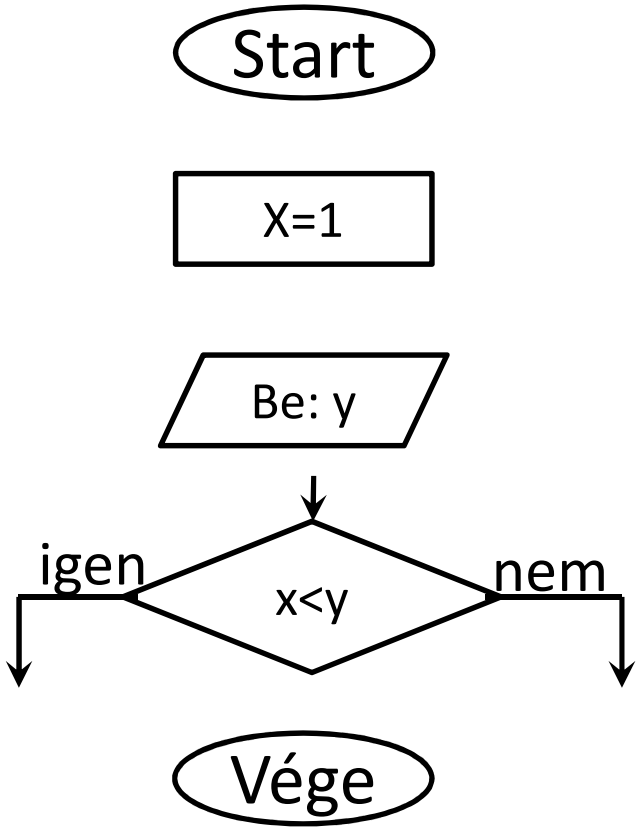
y=sign(x)

Pszeudokódos reprezentáció:

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

Folyamatábra

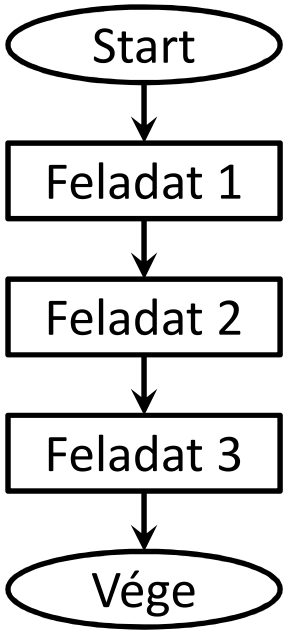
- Kiindulópont
- Elemi utasítás
- Input/output
- Feltétel
- Végállapot
- Csak a nyilak mentén haladhatunk.



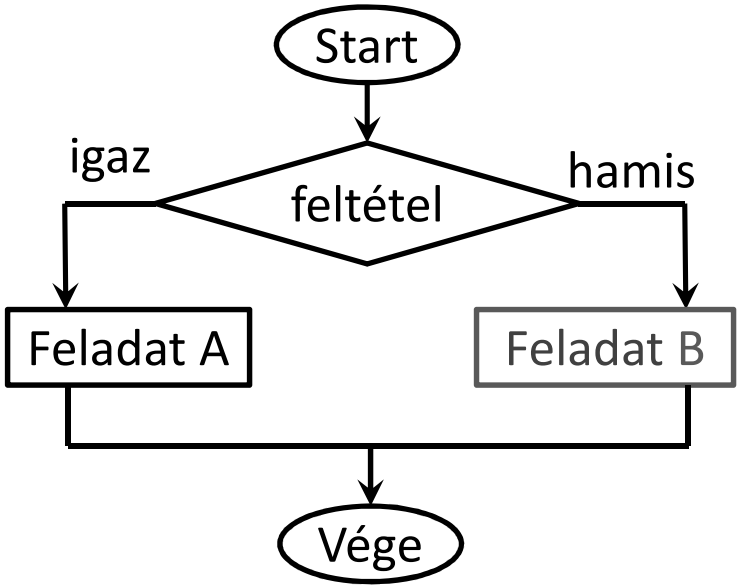
Az algoritmus alap struktúrái

folyamatábrával

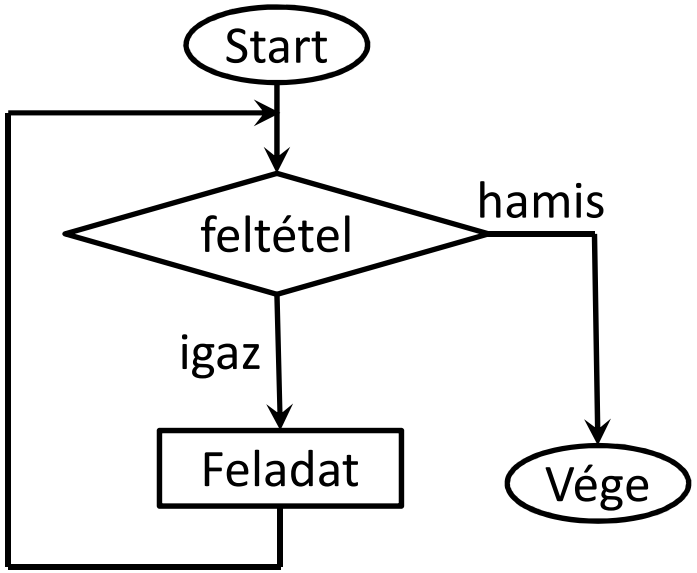
Szekvencia



Elágazás



Ismétlés



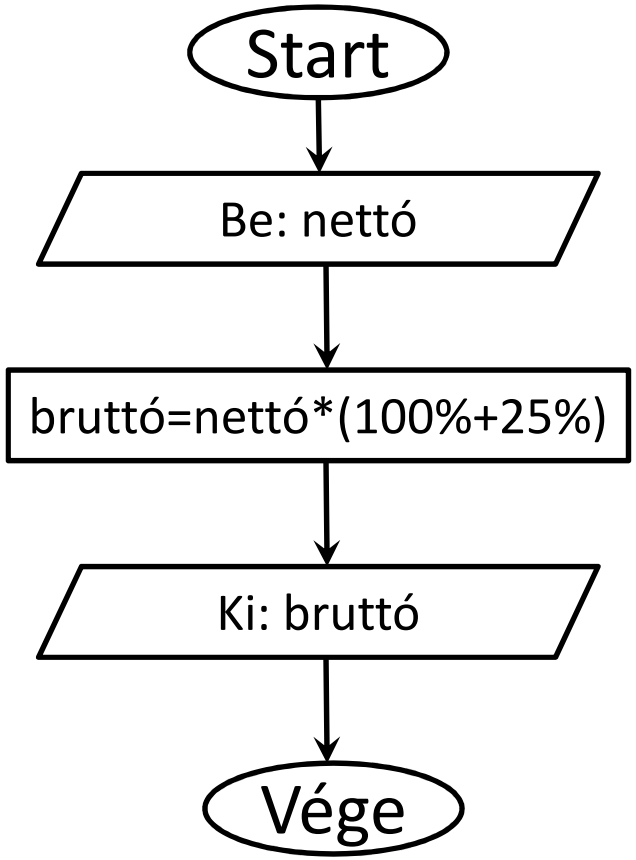
Algoritmusok módosítása

Az algoritmusokat gyakran módosítani kell, hogy jobbak legyenek.

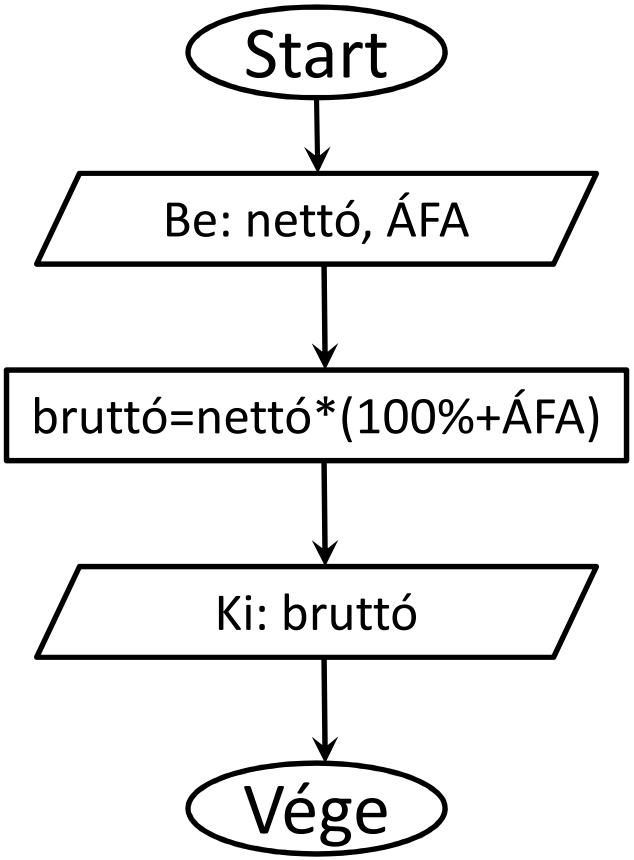
- Általánosítás:
  több esetre is alkalmazható legyen
- Kiterjesztés:
  esetek újfajta körére is alkalmazható legyen
- Beágyazás:
  egy algoritmus újrahasznosítása egy másikon belül
- ‘Bolondbiztossá’ tétel:
  megbízhatóvá, robosztussá, hibaelkerülővé tétel

Algoritmus általánosítása

Eredeti:

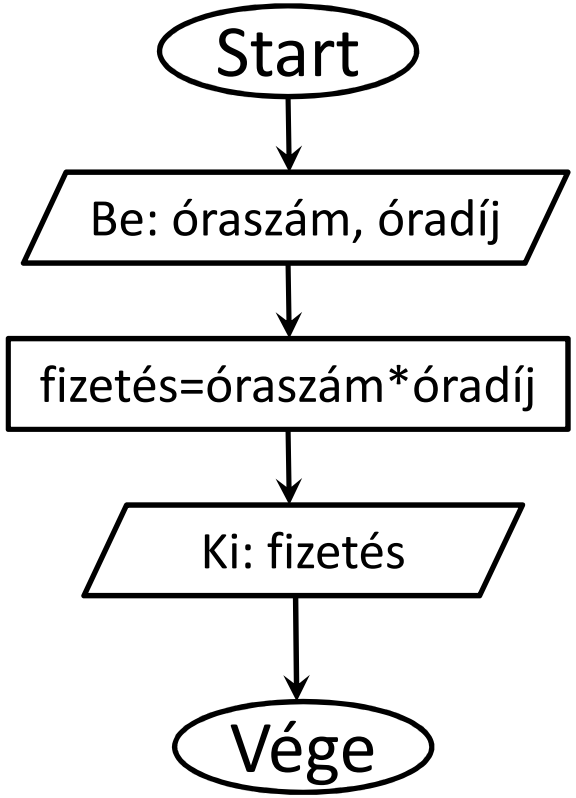


Általánosított:

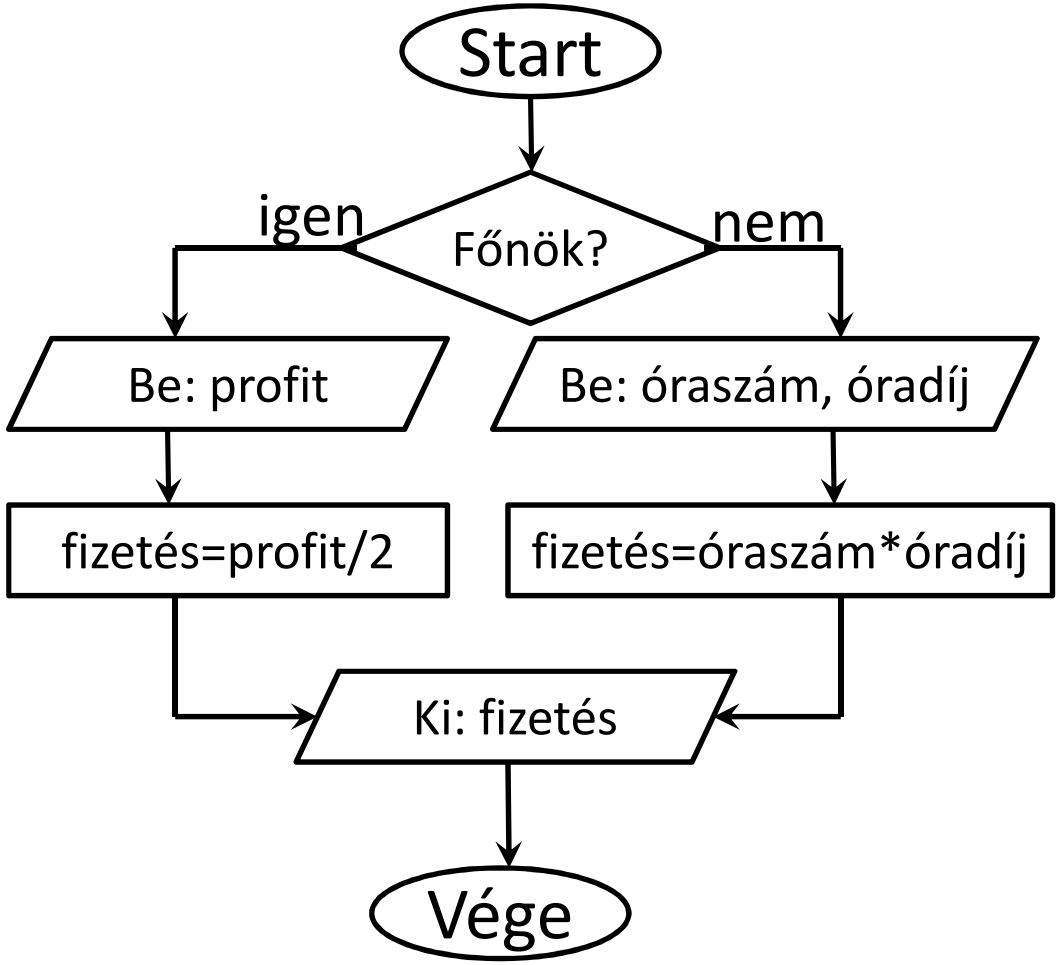


Algoritmus kiterjesztése

Eredeti:

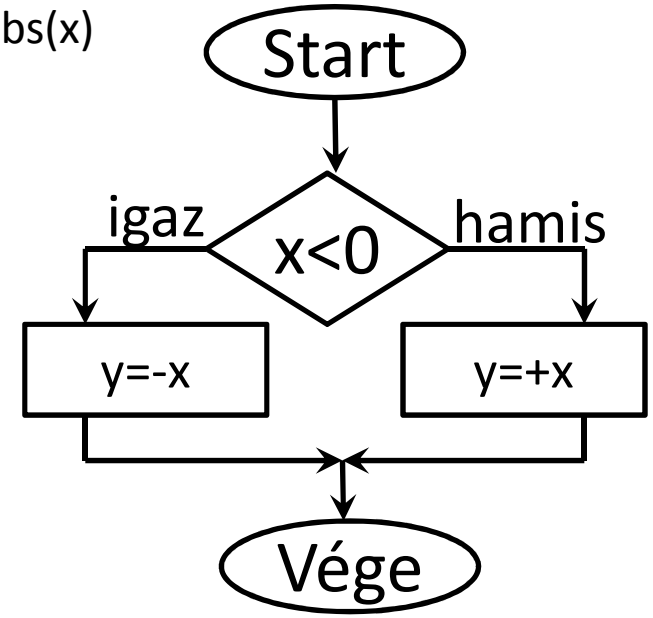


Kiterjesztett:



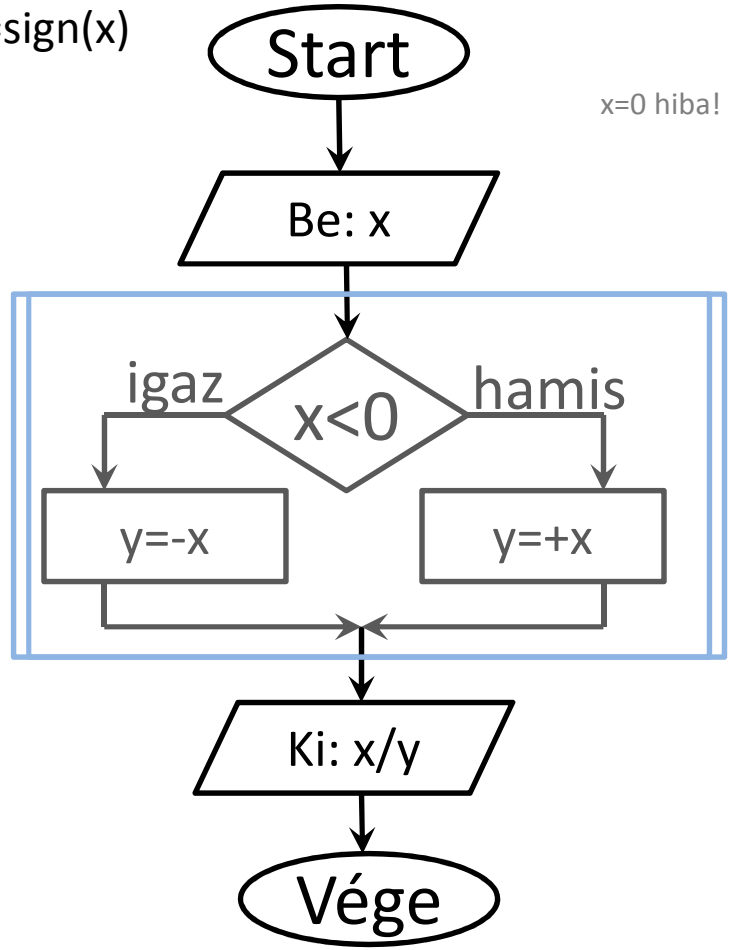
Algoritmus beágyazása

Eredeti:
y=abs(x)



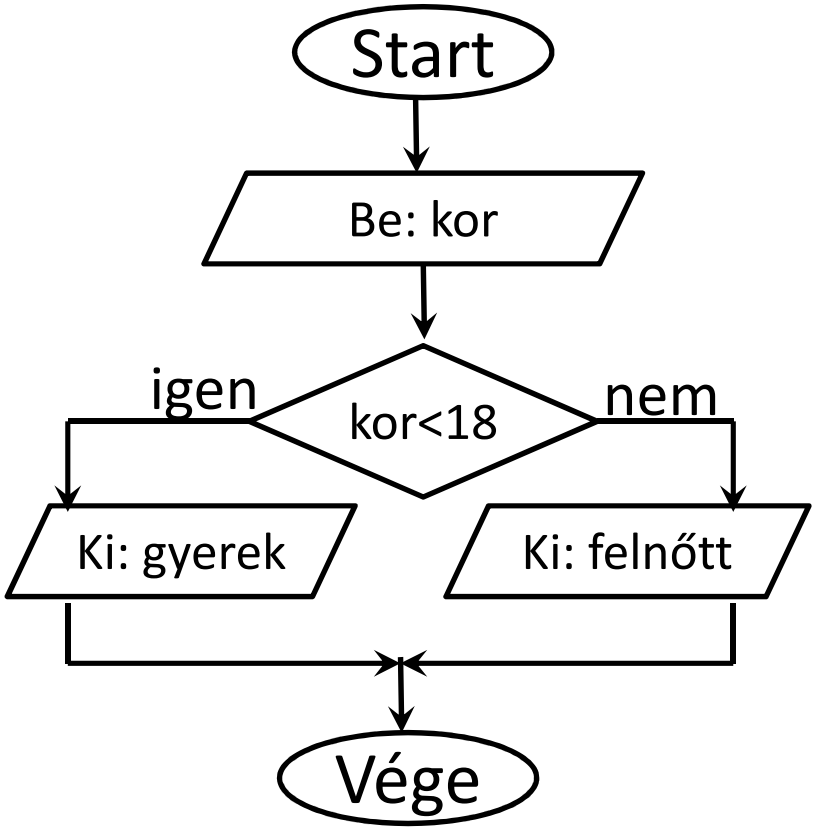
Beágyazott:
y=sign(x)

x=0 hiba!

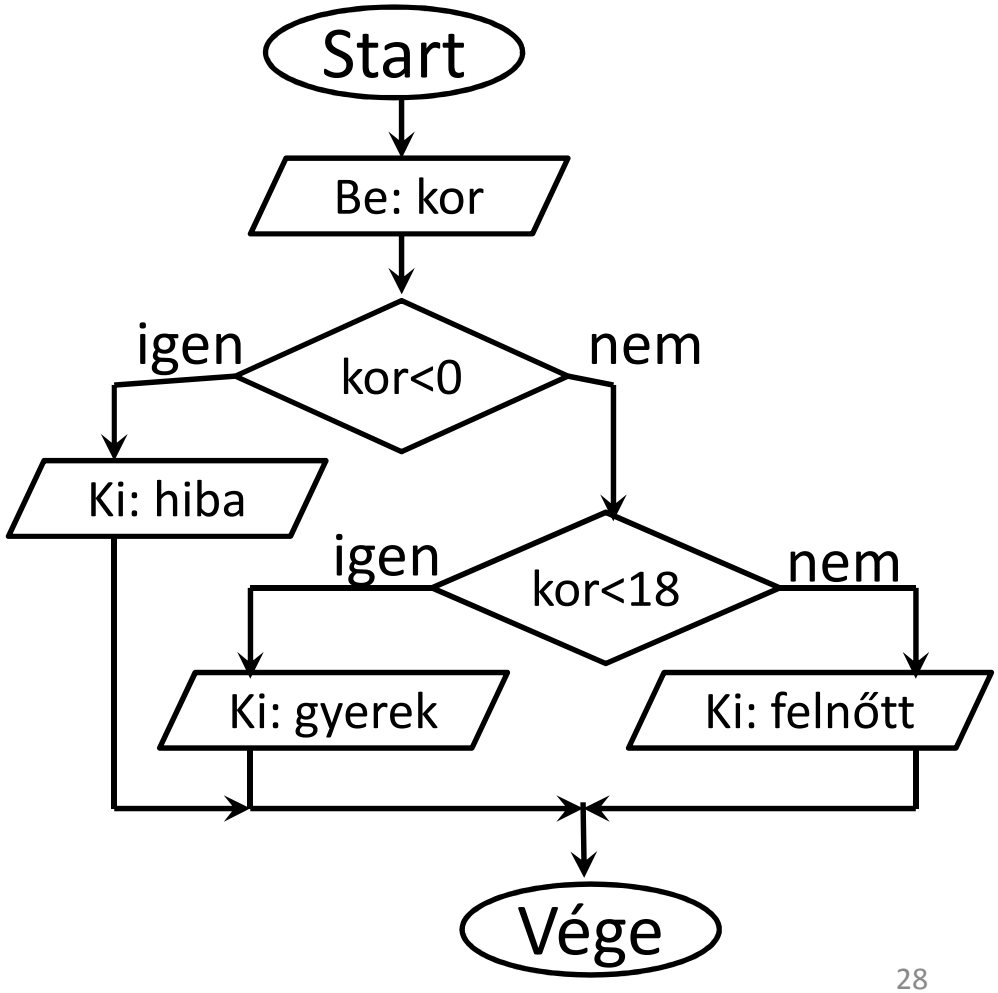


Algoritmus ‘bolondbiztossá’ tétele

Eredeti:



Bolondbiztos:



Alternatív algoritmus

Gyakran többféleképpen is elérhetjük ugyanazt a célt.

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

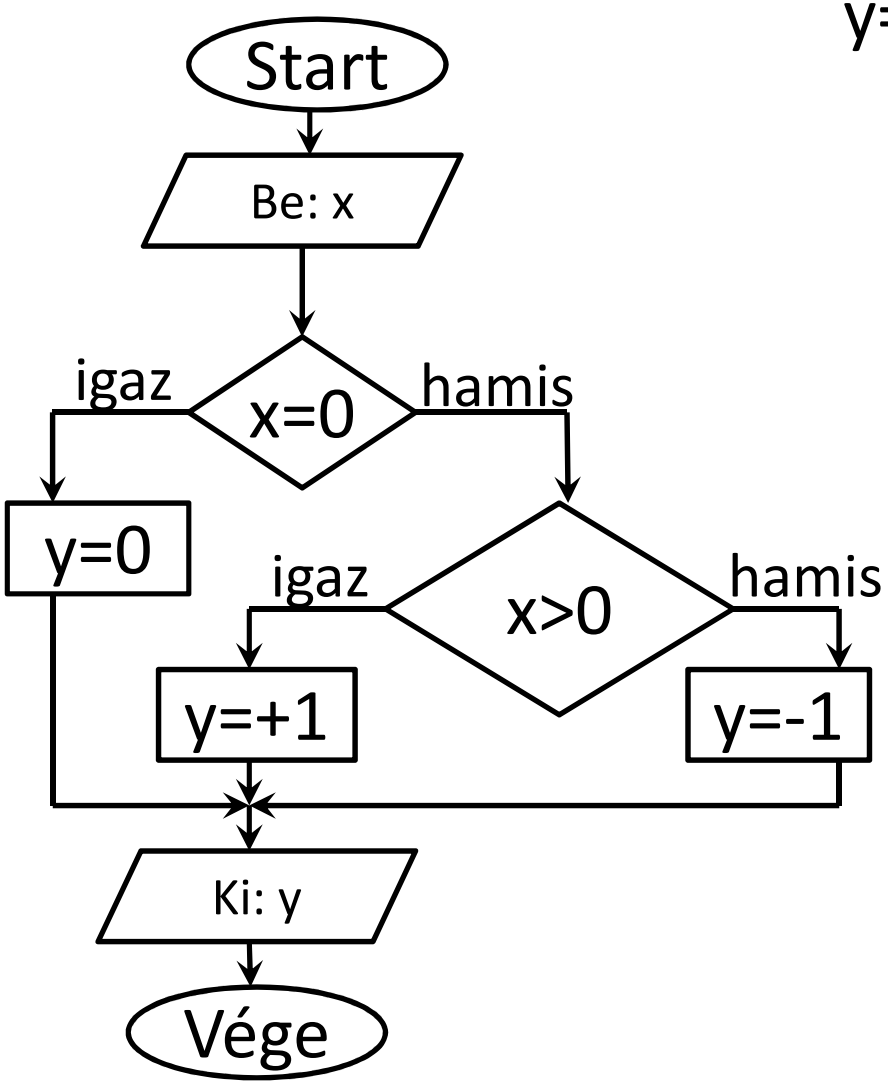
Ez azt jelenti, hogy azonos bemeneti adatokra azonos eredményeket adnak, de ezt máshogy érik el.

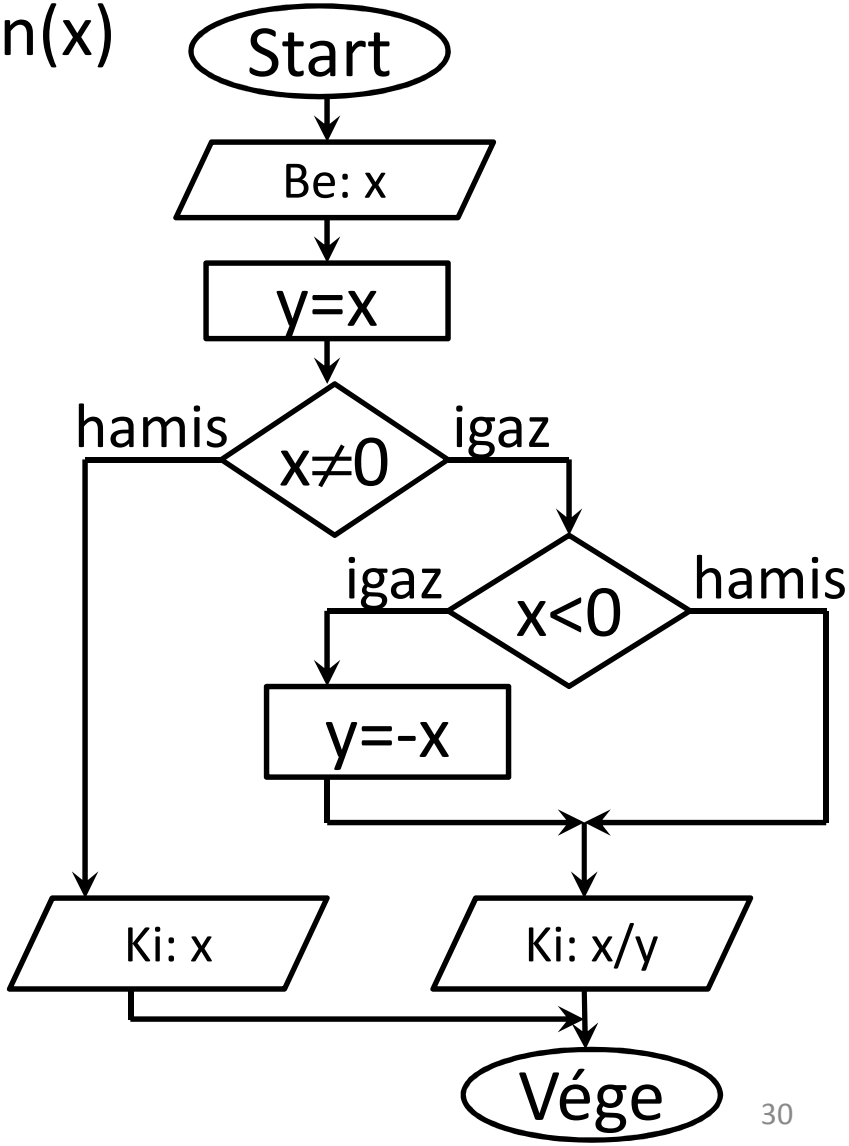
Néha hasznos az egyik algoritmust előnyben részesíteni, néha viszont mindegy melyiket használjuk.

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, not determinisztikus):

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

Rossz algoritmus

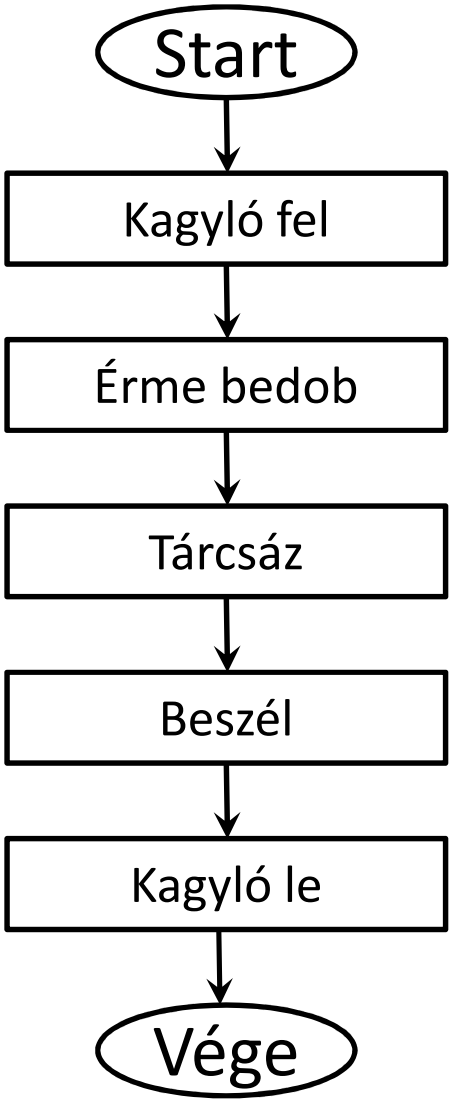
Hogyan buszozzunk?

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



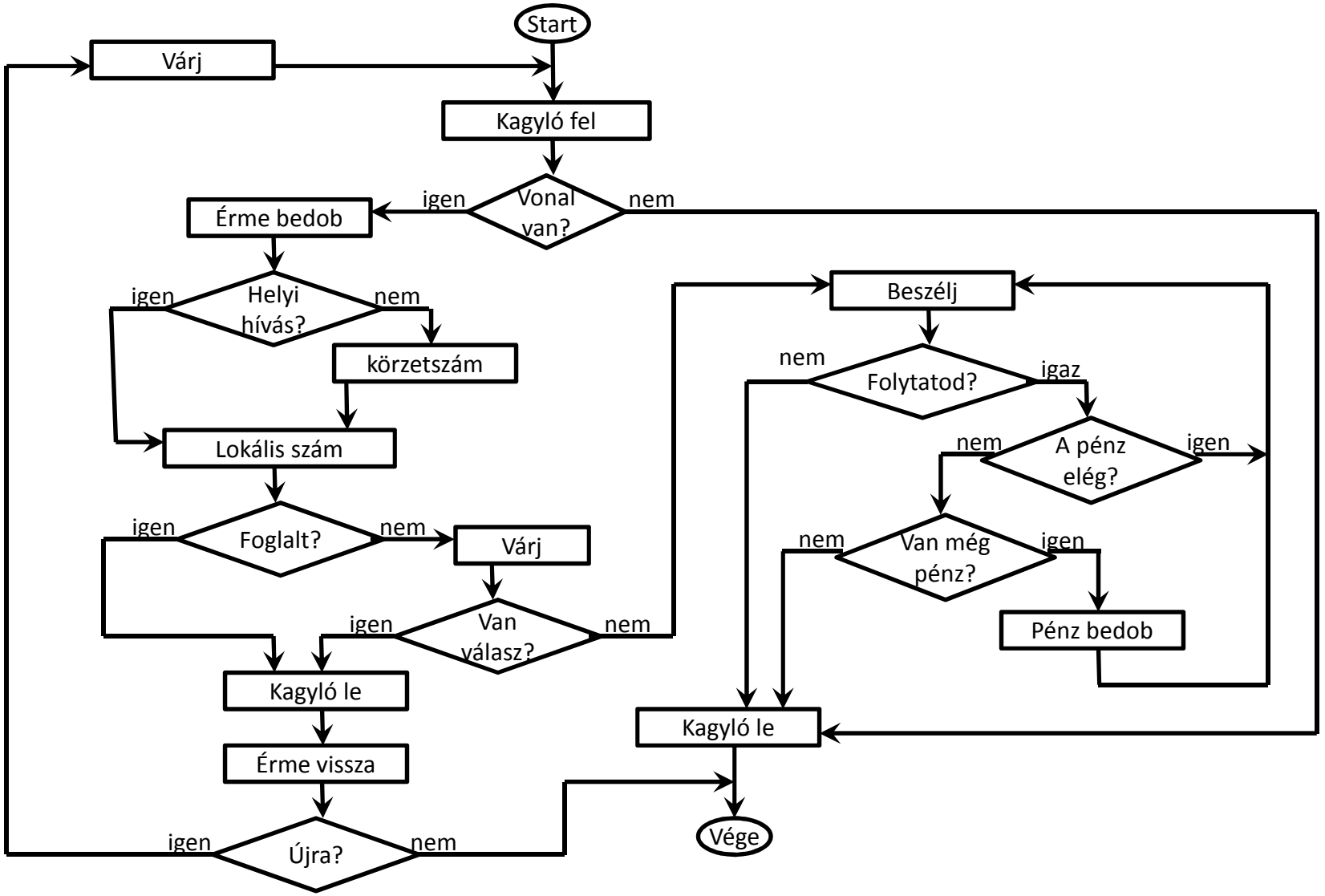
Problémák:

- Nem teljes
- Félreérthető
- …

Módosítás:

- Általánosítás
- Kiterjesztés
- ‘Bolondbiztosá’ tétel
- Teljessé tétel
- Félreértések elkerülése

Nyilvános érmés telefon használata

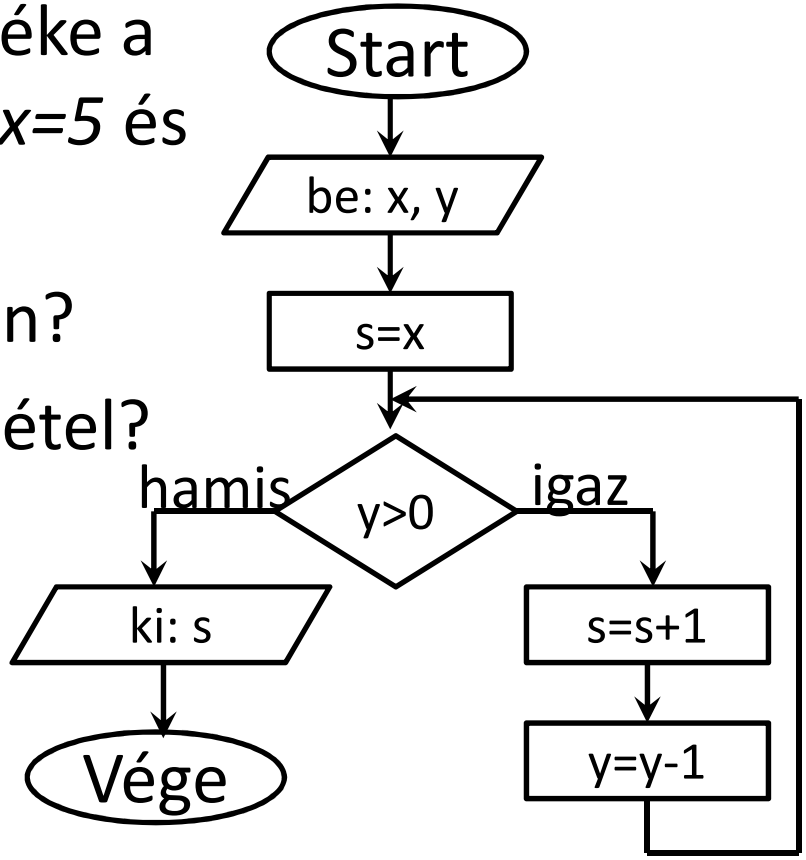


Feladatok és példák

- E-mail írás
- Cipővásárlás
- TV nézés
- Tárgyteljesítés
- Mikrohullámú sütő használat
- Fizetés a kasszánál
- Telefonálás mobillal
- Gyalogosként átkelni az úttesten
- Járművel áthajtani a kereszteződésen
- …

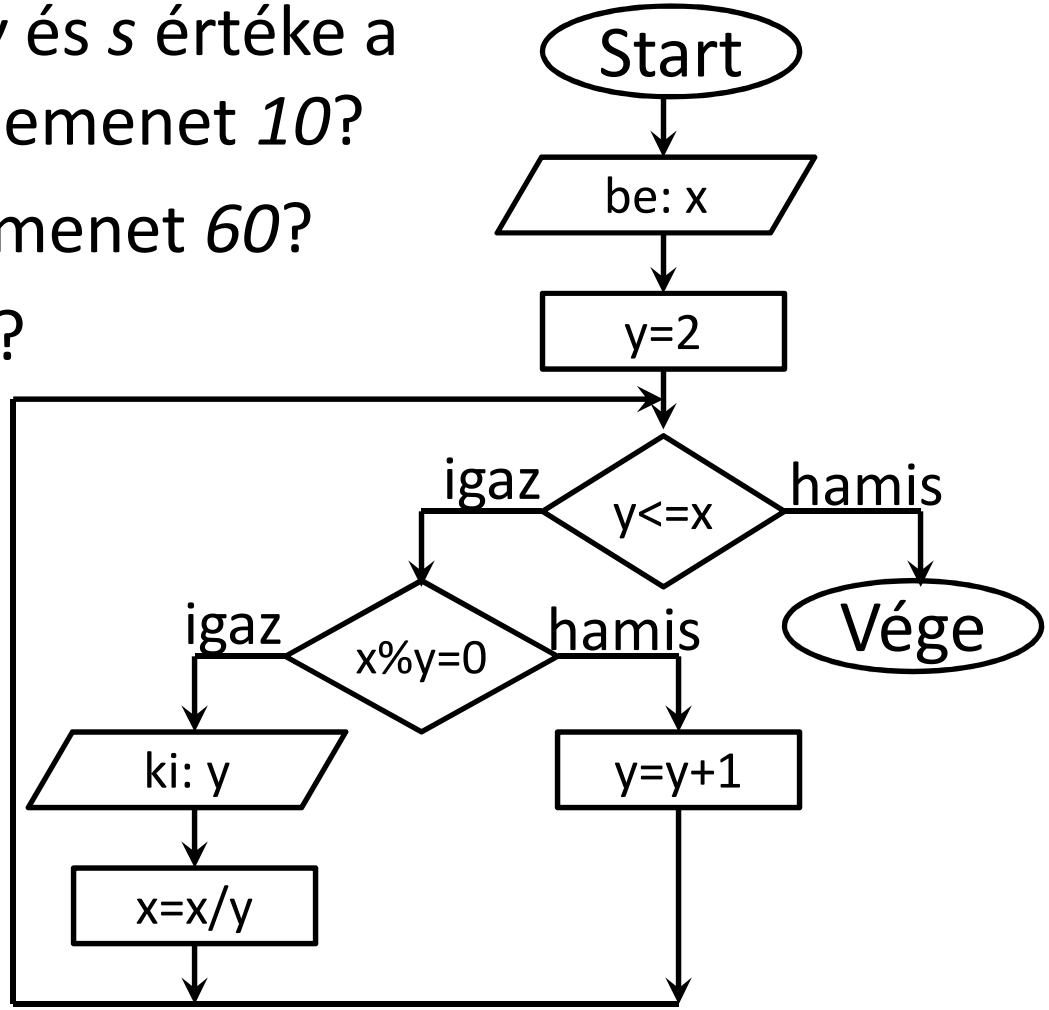
Feladatok és példák

- 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?



Feladatok és példák

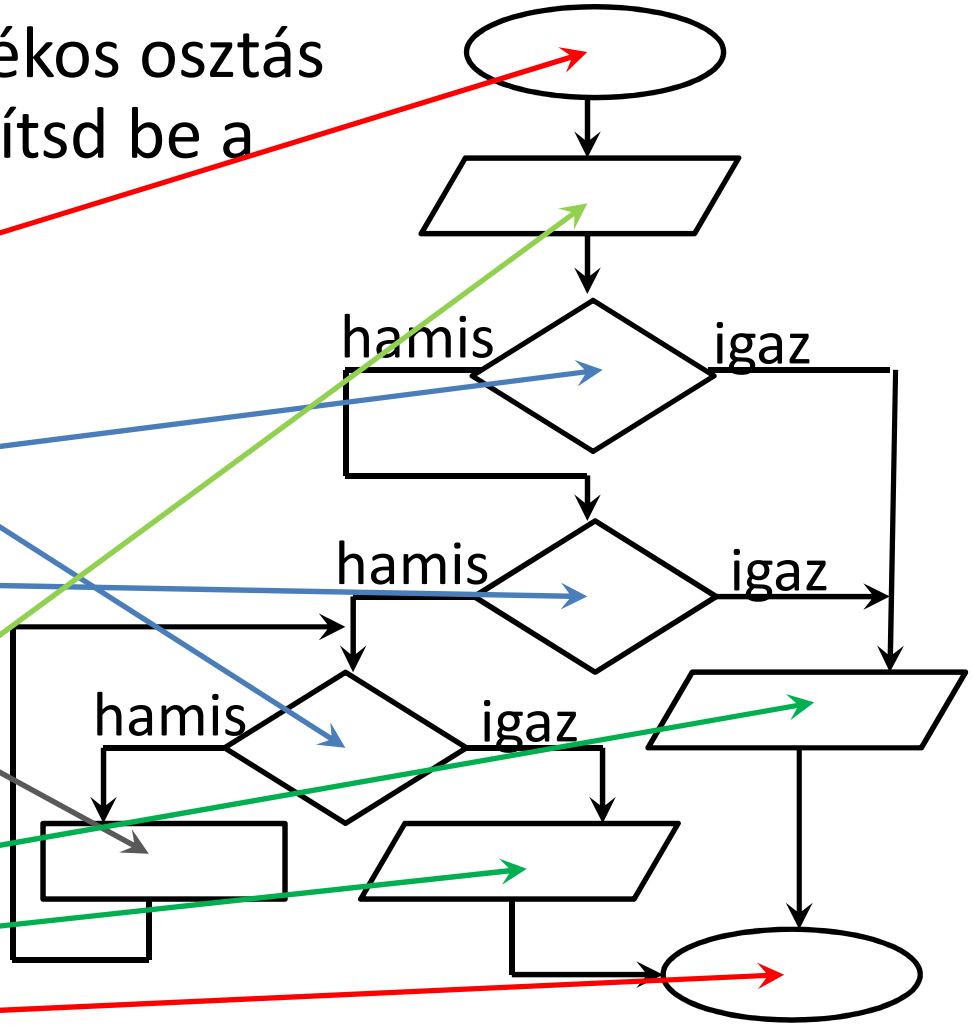
- Hogyan változik az *x, y* és *s* értéke a folyamat során, ha a bemenet *10*?
- Mi a kimenet, ha a bemenet *60*?
- Mit ír le az algoritmus?
- Működik, ha *x=1*?
- Ha az input *24*, hányszor ismétlődik a ciklus?
- Hogy lehetne gyorsabbá tenni?



Feladatok és példák

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: error
- ki: a
- Vége



Feladatok és példák

- Szökőév meghatározása
- Hatványozás
- Faktoriális kiszámítása
- Elsőfokú egyenlet megoldása
- Az év hányadik napja
- Decimális szám konvertálása binárissá
- Bináris számok inkrementálása
- Bináris számok összeadása
- Keresés rendezett bináris fában
- Fibonacchi sorozat
- …

Pszeudokód

Szekvencia:

utasítás1
utasítás2
utasítás3
…

Elágazás:

if *feltétel* then
    utasítás1
else
    utasítás2
endif
…

Ismétlés:

while *feltétel* do
    utasítás
enddo
…

Feladatok és példák

```
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?

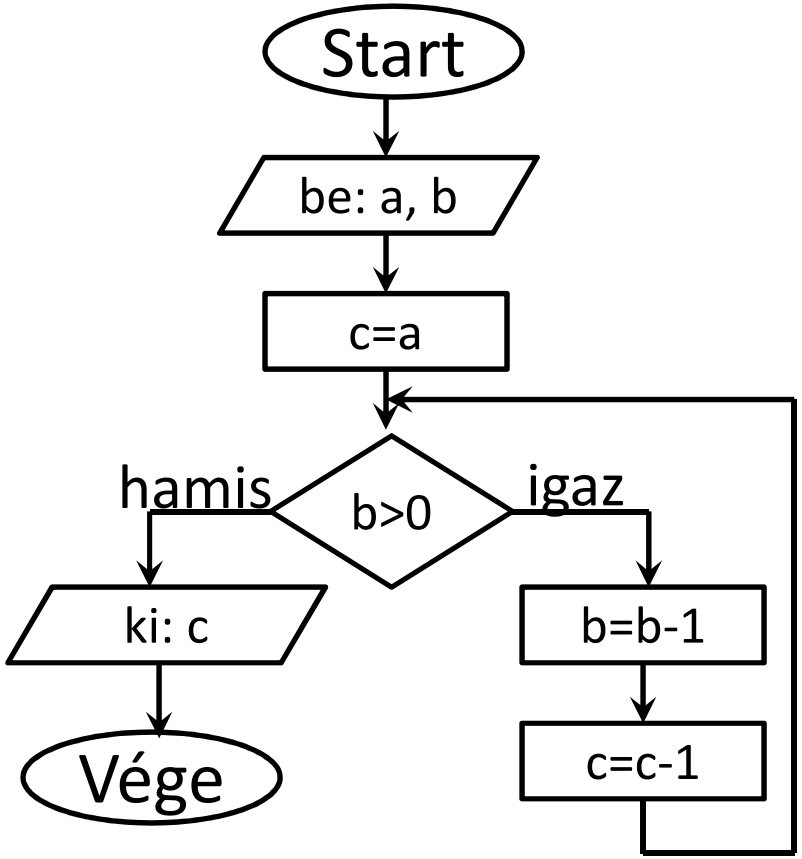
- Mit csinál ez az algoritmus?

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

Feladatok és példák

```
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?



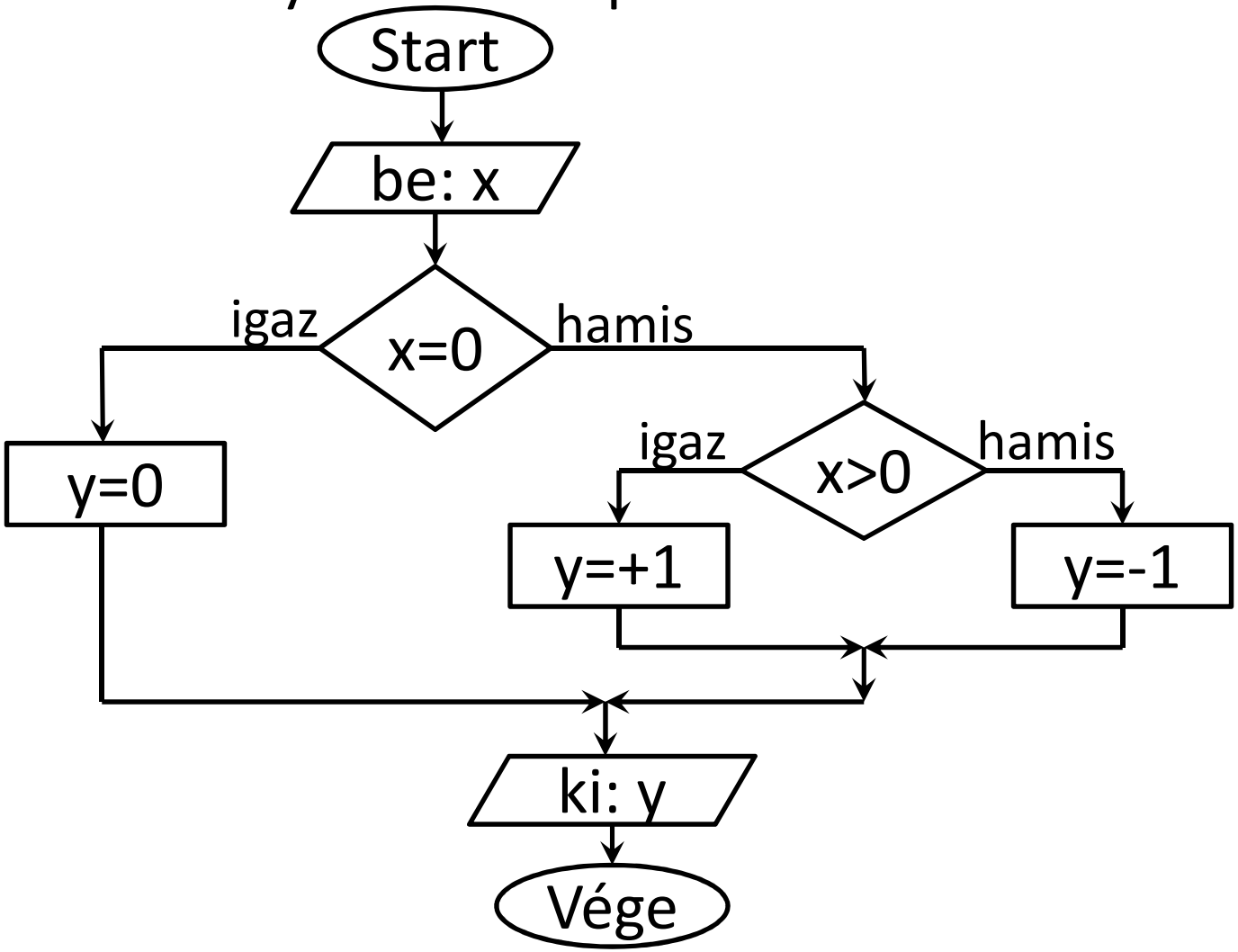
Feladatok és példák

```
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 kell kiértékelni a feltételt?
- Mit csinál az algoritmus?

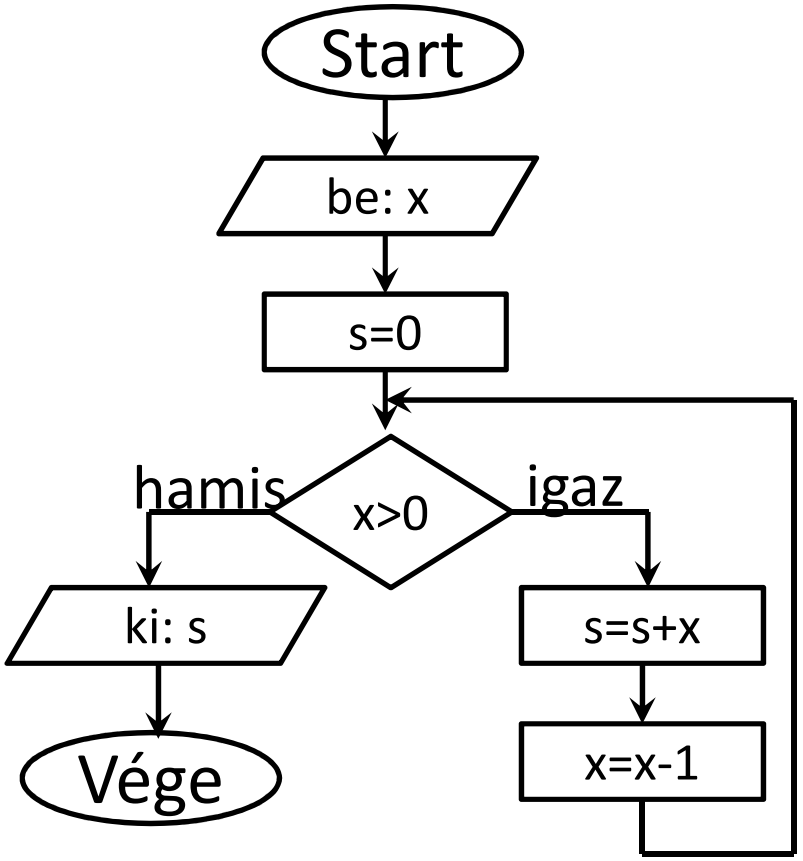
Feladatok és példák

- Írd le azt a folyamatábrát pszeudokóddal!



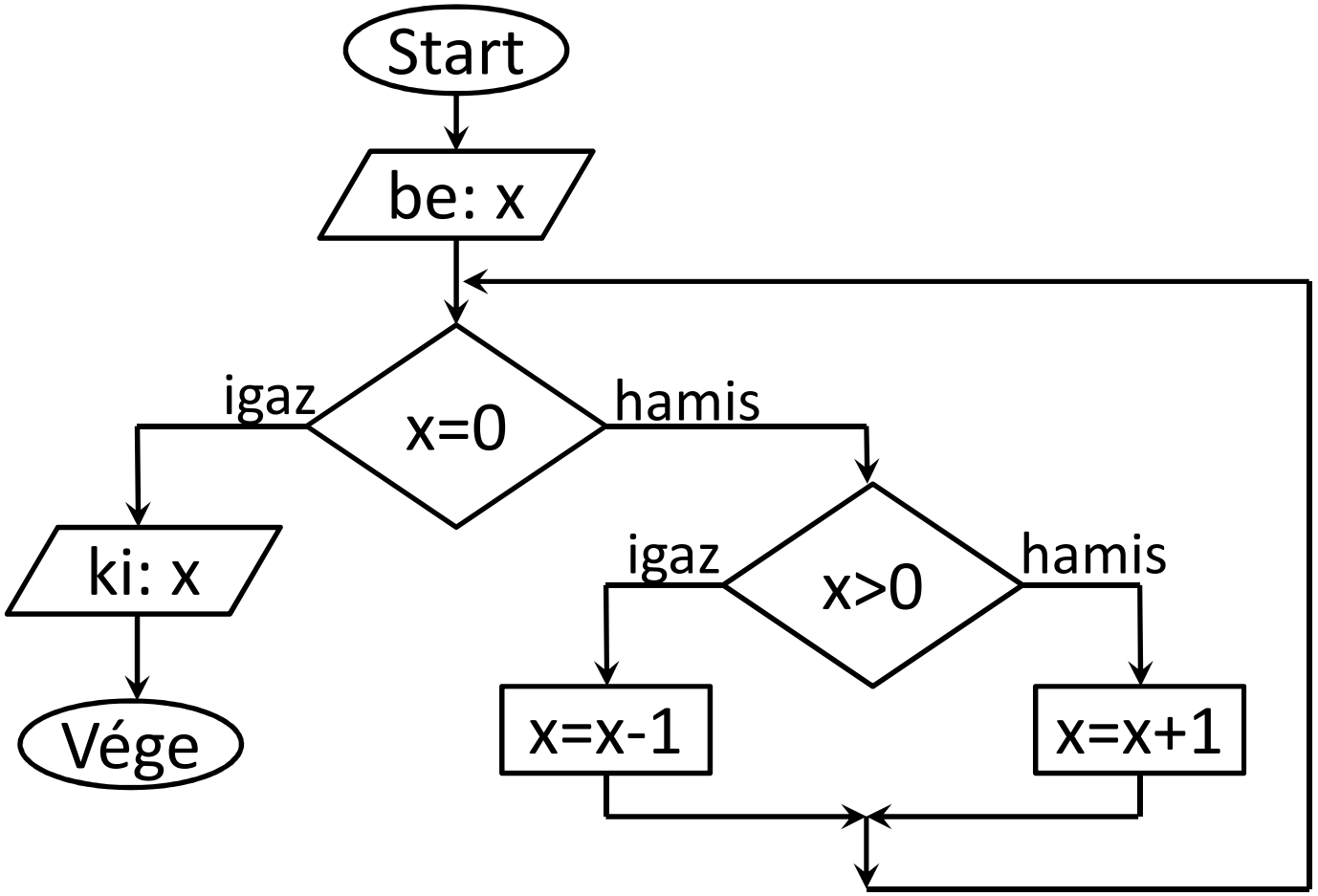
Feladatok és példák

- Írd le azt a folyamatábrát pszeudokóddal!



Feladatok és példák

- Írd le azt a folyamatábrát pszeudokóddal!



Feladatok és példák

Algoritmus verbális leírása:

1. Mondj egy számot!
2. Ellenőrizd, hogy nagyobb mint egy 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!

Feladatok és példák

Írd le az alábbi algoritmusokat pszeudokóddal!

- Abszolút érték meghatározás
- Számok összege 10-től 20-ig
- Hatványozás
- Elsőfokú egyenlet megoldása
- Faktoriális kiszámítása
- Eldönteni egy számról, hogy prím-e
- Prím tényezőkre bontás
- Fibonacchi sorozat

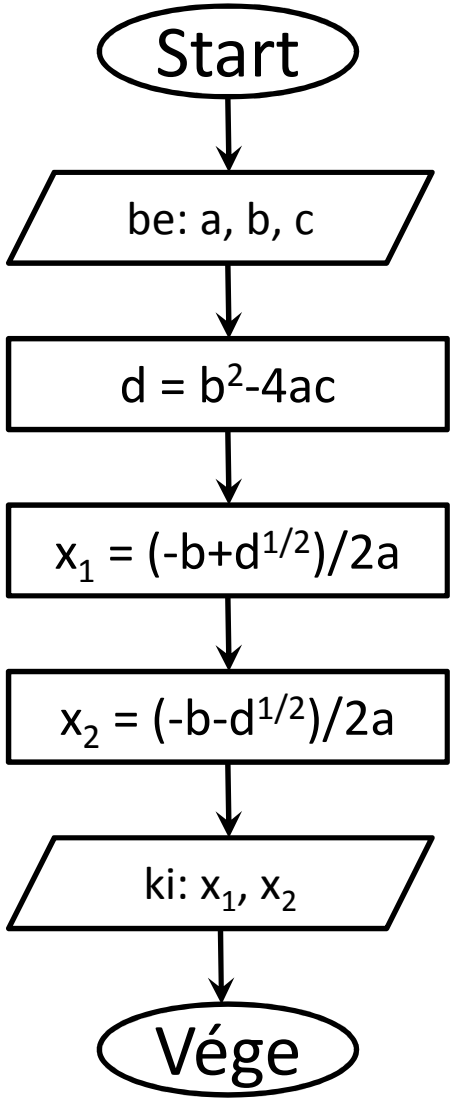
Feladatok és példák

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

Tesztelési stratégia fejlesztés

Tesztelési stratégia példa

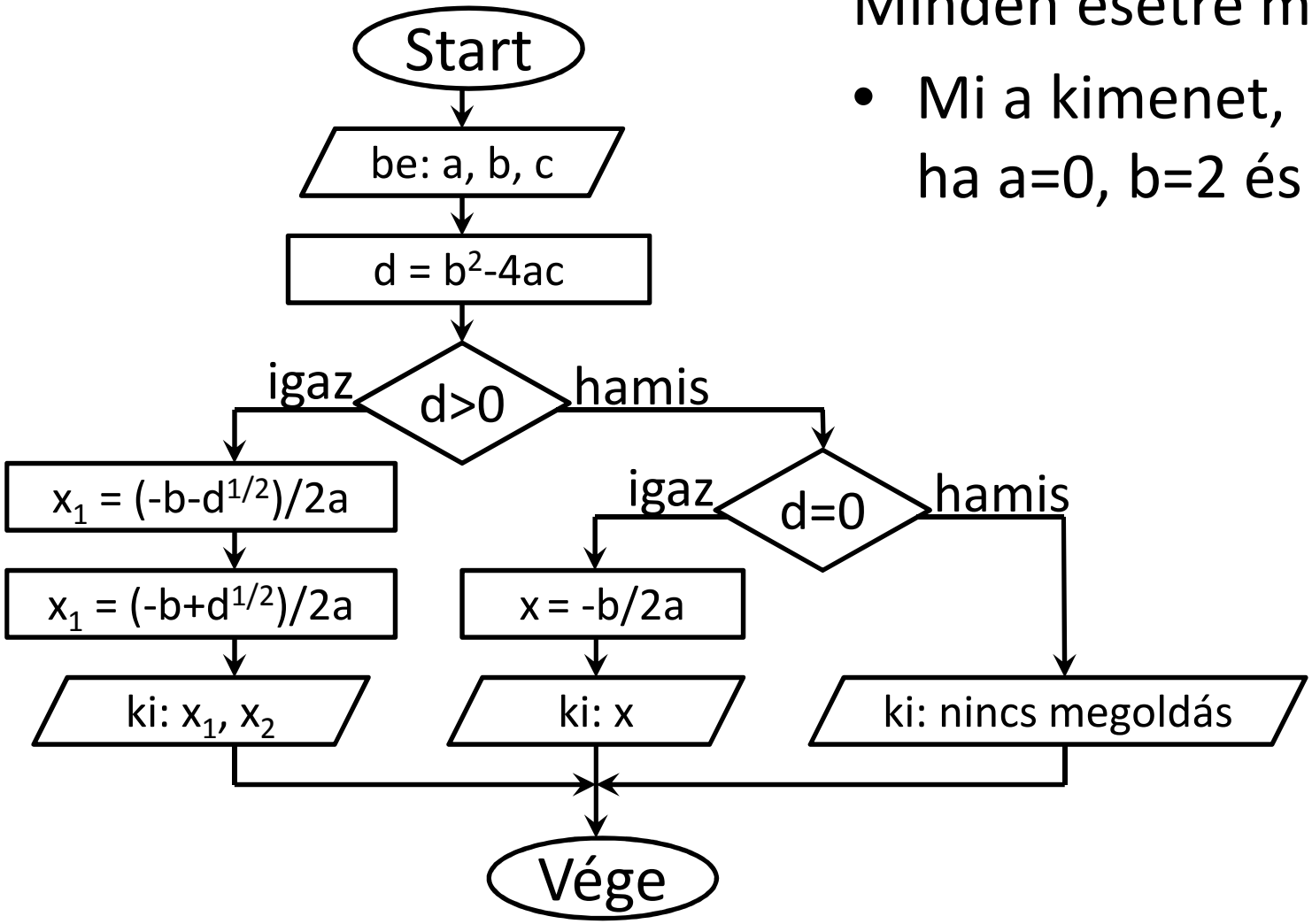
- Másodfokú egyenlet megoldása
- Általános alak: $\mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} = 0$
- Bemeneti paraméterek: a, b, c
- Megoldás: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$



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?

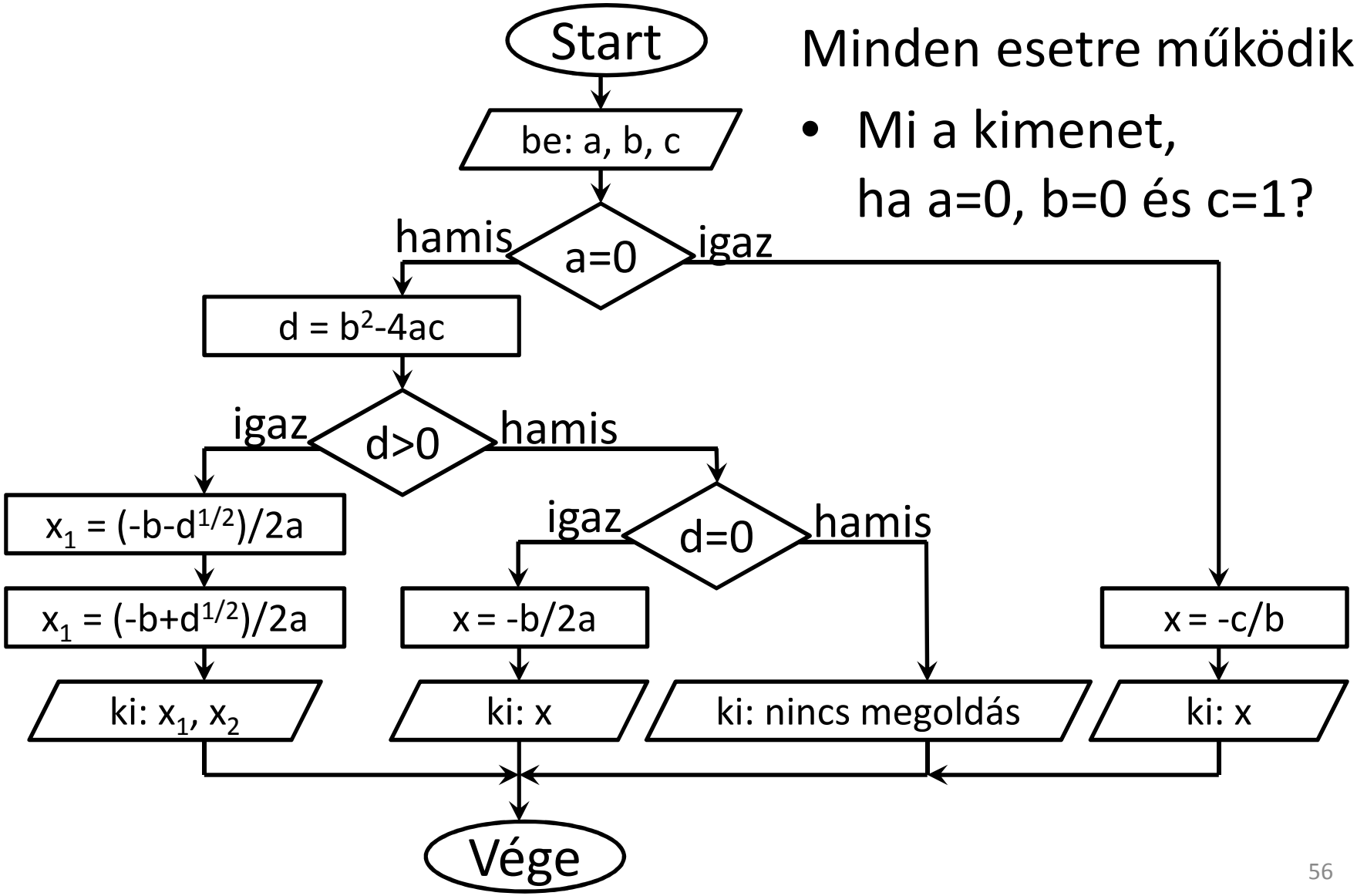
Tesztelési stratégia példa



Minden esetre működik?

- Mi a kimenet,
  ha a=0, b=2 és c=6?

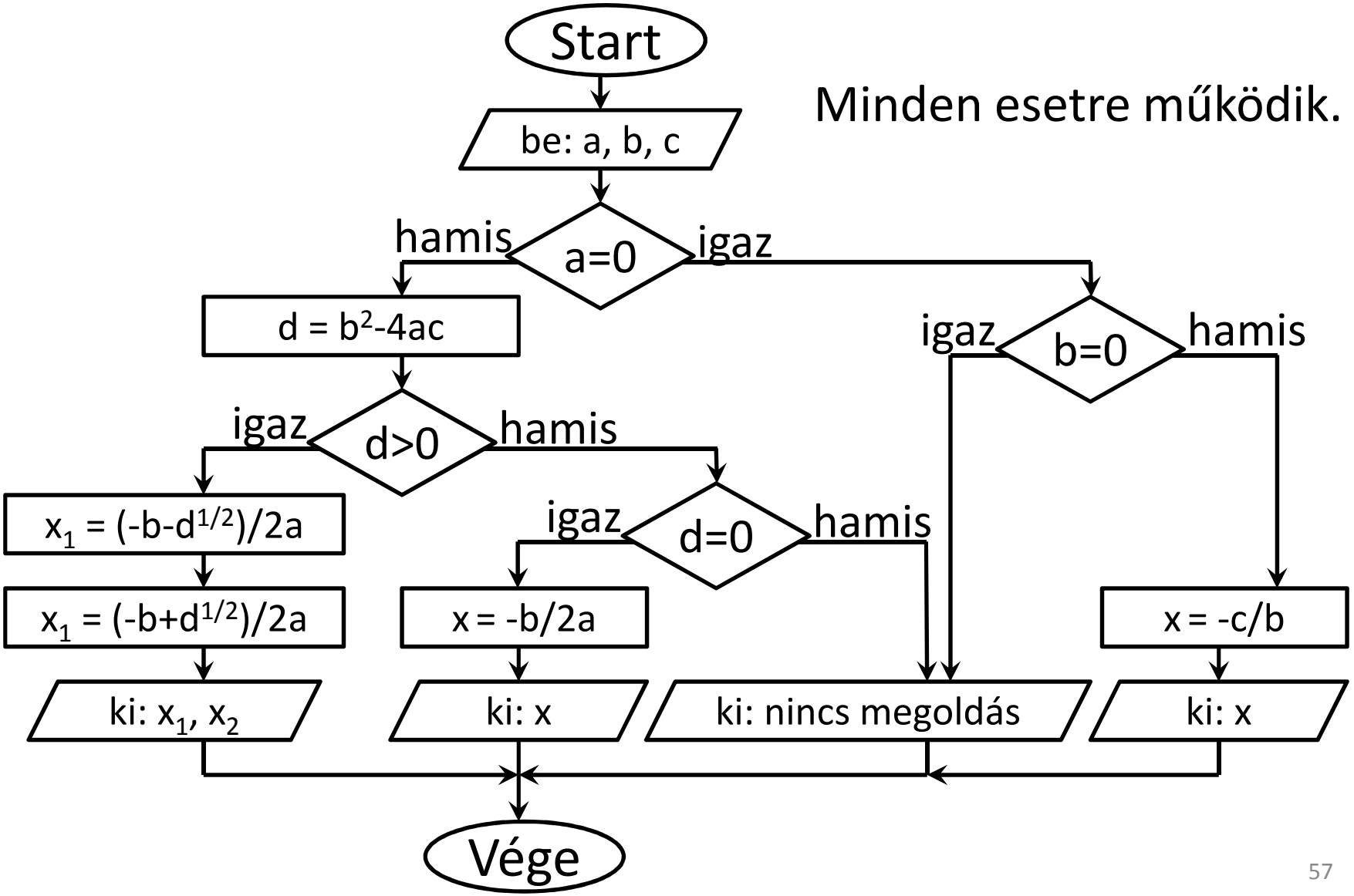
Tesztelési stratégia példa



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 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	Magyarázat	OK
3	7	2	Általános eset (nem 0, d>0)	✓
0	2	6	**a** nulla (első fokú)	✓
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	✓

Program kódolás és tesztelés

Forráskód létrehozása
egy valós programnyelven

Szintaxis és szemantika

Szintaxis: A program szövegének formai szabályai.

Szemantika: A kívánt algoritmust írja le?

Példa (abszolút érték):

```
input a
if a>0 then
   a=-1*a
enidf
output a
```

Szemantikai hiba

Szintaktikai hiba

Programozási nyelvek szintaxisa

Fortran:

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

Pascal:

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

C:

```
long fakt(long n){
  if (n<=1) return 1;
  else return n*fakt(n-1);
  }
```

A forráskód egységei és elemei

- Karakter készlet
- Lexikai egység
- Szintaktikai egység
- Utasítás
- Program egység
- Fordítási egység
- Program

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

C programozási nyelv

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

- Adattípusok: int, float, char, …
- Konstansok: 21, 34.5, ’A’, ”alma”, …
- Operátorok: +, -, *, /, %, =, ==, >=, <=, !=, &&, ||, …
- Elágaztatás: if-else, switch-case
- Ciklusszervezés: while, for, do-while
- Input/output: printf(), scanf() (beépített alprogramok)
- Megjegyzés: /* komment */, //komment
- Stb. …

C programozási nyelv

```c
/*** Solving second degree equation ***/
#include<stdio.h>
#include<math.h>
int main(){
  float a,b,c,d,x1,x2;
  printf("Give the coefficients!\n");
  scanf("%f %f %f",&a,&b,&c);
  if (a==0.0) //first degree
    if (b==0.0)
      printf("Error!\n");
    else{
      x1=-c/b;
      printf("x=%f\n",x1);}
  else{  //second degree
    d=b*b-4*a*c;
    if (d>0.0){  //two solution
      x1=-b+sqrt(d)/(2*a);
      x1=-b+sqrt(d)/(2*a);
      printf("x1=%f\nx2=%f\n",x1,x2);}
    else
      if (d==0.0){  // one solution
        x1=-b/(2*a);
        printf("x=%f\n",x1);}
      else  // no solution
        printf("Error!\n");}
  return 0;}
```