



# Programozási nyelvek és módszerek

---

P-ITSZT-0007

P-ITSZT-004

HÉTFŐ: 09:15-12:00(NEUMANN EA.)

# Oktatók

Dr. Tornai Kálmán – tárgyfelelős

[tornai.kalman@itk.ppke.hu](mailto:tornai.kalman@itk.ppke.hu)



Dr. Feldhoffer Gergely,



Dr. Karlócai Balázs,



Dr. Reguly István Zoltán,



Dr. Zsedrovits Tamás

# Az oktatás célja

---

Programozási módszertan alapjai

Több, különböző, magas szintű programozási nyelvek nyelvi elemeinek megismerése

# Tematika

---

- Bevezetés
- A programozási nyelvek elemei
- Beépített adattípusok, változók, kifejezések
- Utasítások
- Alprogramok és paramétereik
- Absztrakt adattípusok
- Sablonok
- Kivételek
- Objektum-orientált programozás
- Helyesség
- Párhuzamosság
- A könyvtártervezés elvei
- A funkcionális programozás alapjai

# Követelmények

---

Jegy: Vizsga

Vizsga előfeltétele az érvényes (>1) gyakorlati jegy a Java programozási nyelv tárgyból!

Az előadások látogatása legalább 80%-ban kötelező

- Maximum két hiányzás!

Az előzőekben felsorolt tematikából lesz téma otthoni, önálló feldolgozásra

- A félév során három hétfő oktatási szünet
- Várhatóan további egy alkalommal marad még el előadás, ezért a beosztás változhat

# Irodalom

---

## Kötelező irodalom (részek)

- Sebesta, Robert W., Concepts of programming languages, Edinburgh, Pearson, 2013, ISBN 978-0-273-76910-1
- Meyer, Bertrand, Object-oriented software construction, Upper Saddle River, Prentice Hall, 1997, ISBN 0-13-629155-4
- Nyékyné Gaizler Judit et al., (szerk.), Programozási nyelvek, Budapest, Kiskapu, 2003, ISBN 963-9301-47-7

## Ajánlott irodalom

- SCOTT, Michael L., Programming Language Pragmatics, Amsterdam, Morgan Kaufmann, 2009, ISBN 978-0-12-374514-9
- Stroustrup, Bjarne, A C++ programozási nyelv, 1-2kötet, Budapest, Kiskapu Kiadó, 2001, ISBN 963-9301-18-3, 963-9301-17-5
- Wilson, Leslie B., Clark, Robert G., Comparative programming languages, Harlow, Addison-Wesley, 2001, ISBN 0-20-171012-9
- Nyékyné Gaizler Judit et al., (szerk.), Java 2 útikalauz programozóknak: 5.0, Budapest, 2008, ELTE TTK Hallg. Alapítvány, ISBN 978-963-06-4092-3

# Mai óra

---

- Bevezetés
- Programozási nyelvek alapjai
- Generatív grammatika
- Programozási nyelv, mint nyelv
- Programok felépítése, részei
- Programozási nyelvek története, fontosságuk

# Miért készítünk programokat?



Joseph Marie Jacquard



# Cél, segítség, eszköz

---

Cél a jó minőségű program és termék



Ebben segít a programozási módszertan



Eszközként használjuk a programozási nyelveket

# Szoftverek minősége

---

## Szempontok

- helyesség
- megbízhatóság
- karbantarthatóság
- újrafelhasználhatóság
- kompatibilitás
- hordozhatóság
- hatékonyság
- stb.

# Helyesség

---

A program **helyes**, ha pontosan megoldja a feladatot, megfelel a kívánt specifikációnak.

Alapvető: a pontos és minél teljesebb specifikáció.

# Feladat specifikációja

---

Megadása:

- Sokszor (régen mindig) szövegesen
  - Például:
    - Számítsd ki **a** és **b** legnagyobb közös osztóját!
    - Számos információt „beleértünk”....
    - Néha félreértyük – véletlenül vagy szándékosan
      - „Találkozzunk a Blahán a metró lépcsőjénél”
      - „A királynőt megölni nem kell félnetek jó lesz”

# Feladat specifikációja

---

- Formálisan is
- Például
  - $I$  – a lehetséges bemenetek,  
 $O$  – a lehetséges kimenetek  
végesen leírható halmaza
  - $F \subseteq I \times O$  reláció, ahol  $i \in I$  és  $o \in O$  esetén  $(i,o) \in F$  azt jelenti, hogy  $F$  az  $i$  bemenethez hozzárendeli az  $o$  kimenetet.
  - Például:
    - $I = N \times N$
    - $O = N$
    - $F = \{(a,b,c), \text{ ahol } a, b, c \in N \wedge c \mid a \wedge c \mid b \wedge \forall x : x \mid a \wedge x \mid b \rightarrow c \geq x\}$
    - $c \mid a$  jelentése: a osztható  $c$ -vel.

# Feladat specifikációja

---

Szerződés a felhasználó és az implementáló között

- előfeltétel: állítás, ami leírja azt a feltételeket, ami szükséges a feladat helyes működéséhez
- utófeltétel: állítás, ami leírja azt a feltételeket, amit a függvény teljesít a helyes végrehajtás után

Helyesség a specifikáció figyelembevételével

- A program felhasználója teljesíti az előfeltételt
- A program lefut
- A program végeztével az utófeltétel igaz lesz.

Mit kell az implementációnak teljesíteni, ha a felhasználó megséríti az előfeltételt?

# Feladat specifikációja

---

Elő- és utófeltételekkel – volt korábban is

- Állapottér:  $N \times N \times N$

a   b   c

- Előfeltétel (Ef)

- $a = a' \wedge b = b'$

- Utófeltétel (Uf):

- $Ef \wedge c | a \wedge c | b \wedge \forall x : x | a \wedge x | b \rightarrow c \geq x$

# Algoritmus

---

„Az algoritmus egyértelműen előírt módon és sorrendben végrehajtandó tevékenységek véges sorozata egy feladat megoldására, ami véges idő alatt befejeződik.”

Hétköznapi életben is: egy feladat megoldási lépései teljes részletességgel

- Feladat: Süss egy gyümölcsöt!
- (Mi az input? Mi az output?)
- Algoritmus: a recept

Nem minden feladatra adható algoritmus!

# Algoritmus

---

Megadható:

- Szövegesen, természetes nyelven
- Pszeudokóddal
- Grafikusan
  - Folyamatábrával
  - Struktogrammal
  - ...
- Automatákkal vagy **Turing-gép** segítségével
- Táblázattal
- Programozási nyelv segítségével

# Algoritmus

---

Komponensei:

- Szekvencia - parancsok egymásutánja
- Elágazás – valamilyen feltételtől függően alternatívák választása
- Ciklus – utasítások ismételt végrehajtása

Fontos kérdés:

- Biztosan megoldja a feladatot?
  - Helyes?
- Mit jelent az, hogy megoldás?
- Részletesen még jön!

# Algoritmus

---

Abu Abdallah Muḥammad ibn Mūsā al-Khwārizmī (780 – ?845) arabul alkotó perzsa tudós, matematikus volt

Kidolgozta a matematikai algoritmusok fogalmát (ami miatt néhányan a számítástechnika nagyapjának nevezik), maga az algoritmus szó is nevének félrefordított latin változatából ered



# Megbízhatóság

---

Egy programot **megbízhatónak** nevezünk, ha

- *Helyes*, és
- Abnormális helyzetekben sem történik katasztrófa
  - Hanem valamilyen „ésszerű” működést produkál

Abnormális helyzet

- A specifikációban nem leírt helyzet

# Karbantarthatóság

---

A **karbantarthóság** annak mérőszáma, hogy milyen könnyű a programterméket a *specifikáció* változtatásához adaptálni.

- Egyes felmérések szerint a szoftver költségek 70 %-át szoftver karbantartásra fordítják!

A karbantarthóság növelésének két alapelve

- Tervezési egyszerűség
- Decentralizáció
  - minél önállóbb modulok létrehozása

# Újrafelhasználhatóság

---

A szoftver termék, vagy komponens egy másik, új alkalmazásban felhasználható.

Ez vonatkozik a szoftvertermék

- Egészére
- Részére

# Kompatibilitás

---

A kompatibilitás megmutatja, hogy mennyire könnyű a szoftver termékeket egymással kombinálni.

- Nem légüres térben fejlesztünk!

# Hordozhatóság

---

A program hordozhatóságának mértéke mutatja meg, mennyire könnyű a programot átalakítani

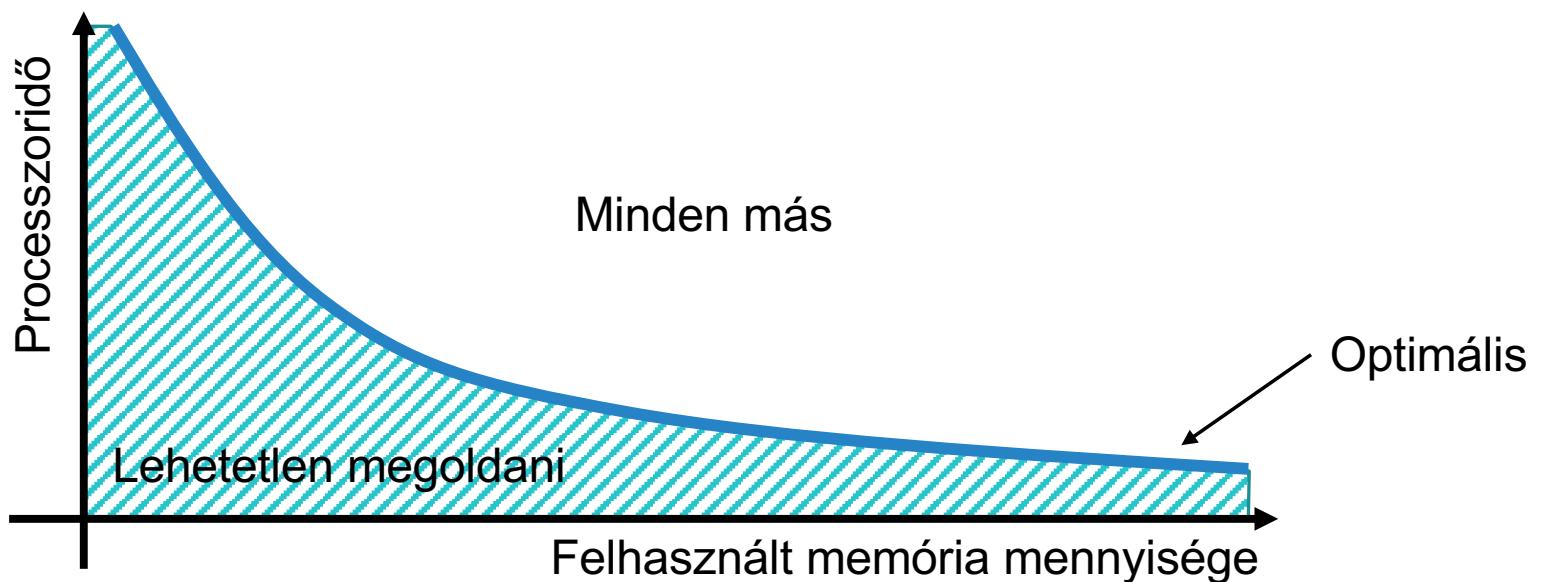
- Más géphez
- Más konfigurációhoz
- Más operációs rendszerhez
- Általában más környezethez

# Hatékonyság

A program **hatékonysága** a futási idővel és a felhasznált memória méretével arányos.

Egy program hatékony, ha

- gyorsan lefut
- kevés memóriát használ



# Barátságosság

---

A program emberközelisége, barátságossága a felhasználó számára rendkívül fontos

- Megköveteli, hogy legyen
  - az input logikus és egyszerű
  - az eredmények formája áttekinthető

# Tesztelhetőség

---

A tesztelhetőség, áttekinthetőség a program karbantartói, fejlesztői számára fontos.

# Programozási módszertan

---

## Moduláris tervezés

- Programjainkat egymástól minél inkább független, de jól definiált kapcsolatban álló programegységek segítségével tervezük.

## Paradigmák

- Procedurális
- Objektumorientált
- Függvényalgoritmusos
- Logikai
- Szimbolikus
- ...

# Moduláris dekompozíció

---

## Jelentése

- A feladat több egyszerűbb részfeladatra bontása
- A részfeladatok megoldása egymástól függetlenül elvégezhető
- Ennek segítségével csökkentjük a feladat bonyolultságát.

Általában a módszert ismételten alkalmazzuk

- Azaz az alrendszeret magukat is dekomponáljuk.
- Ez lehetővé teszi, hogy a feladat megoldásán egyszerre többen dolgozzanak

A módszer általi felbontás egy fával megadható.

- A fa csomópontjai az egyes dekompozíciós lépések

# Moduláris dekompozíció

## Feladat

Alfeladat 1

Alfeladat 2

Alfeladat 3

F.11

F.12

F.13

F.21

F.22

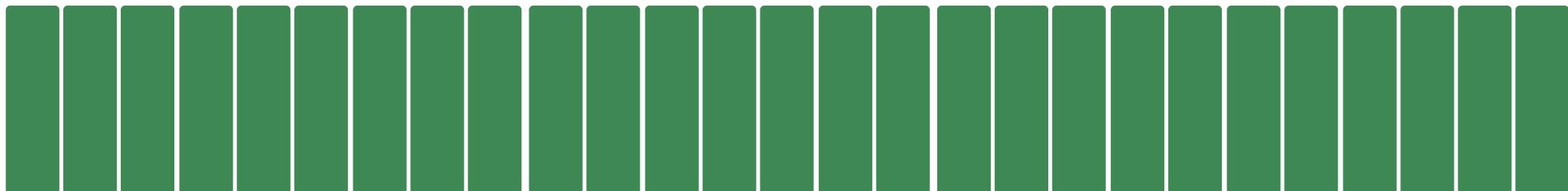
F.23

F.31

F.32

F.33

F.34



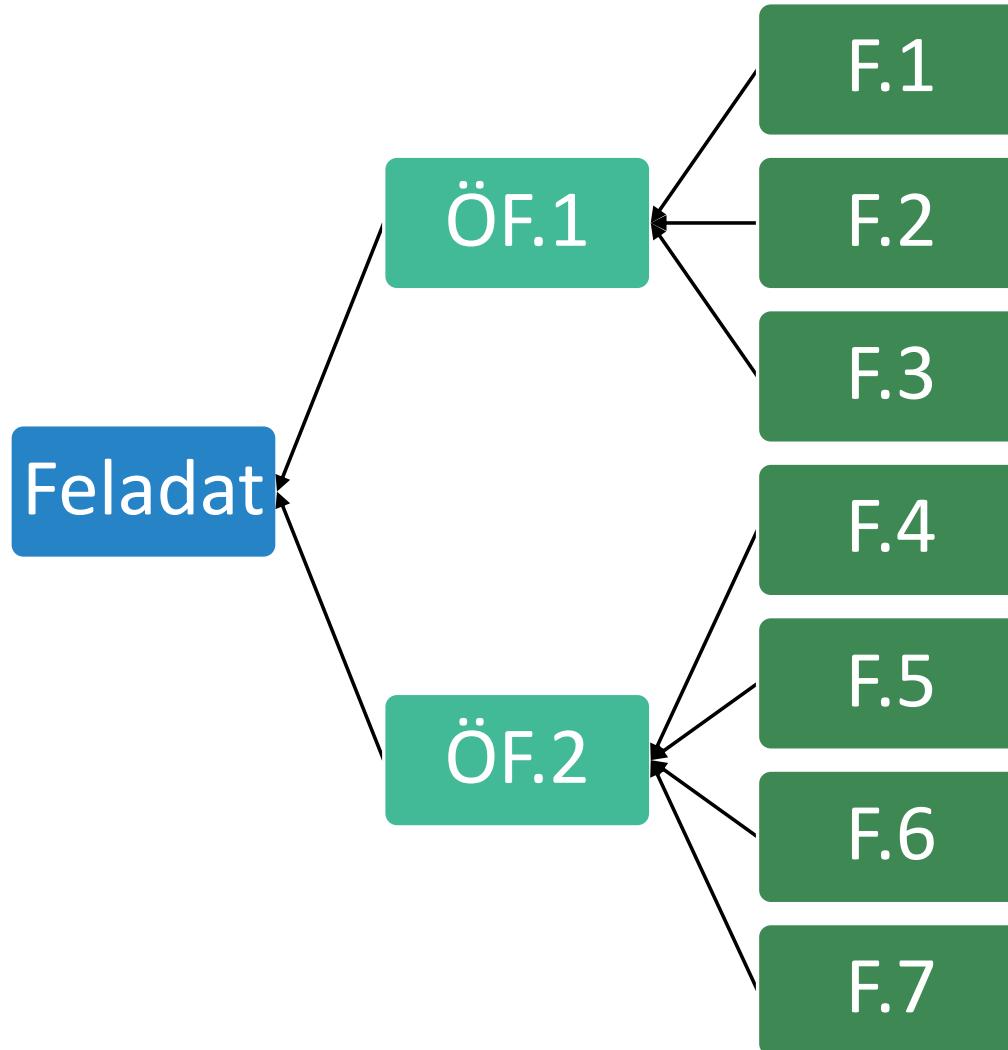
# Moduláris kompozíció

---

Olyan szoftver elemek létrehozását támogatja, amelyek szabadon kombinálhatók egymással.

- A programok lehetőleg a már meglévő programegységekből, mint építőkövekből épülnek fel.

# Moduláris kompozíció



# Moduláris érhetőség

---

A modulok önállóan is egy-egy értelmes egységet alkotnak

- Megértésükhez minél kevesebb „szomszédos” modulra van szükség

# Moduláris folytonosság

---

A specifikáció „kis” változtatása → a programban is csak „kis” változtatásra van szükség

# Moduláris védelem

---

Célunk kell legyen a program egészének védelme az abnormális helyzetek hatásaitól.

- Egy hiba hatása egy - esetleg néhány - modulra korlátozódjon!

# A modularitás alapelvei

---

A modulokat nyelvi egységek támogassák

- A modulok illeszkedjenek a használt programozási nyelv szintaktikai egységeihez.

Kevés kapcsolat legyen

- minden modul minél kevesebb másik modullal kommunikáljon!

Gyenge legyen a kapcsolat

- A modulok olyan kevés információt cseréljenek, amennyi csak lehetséges!

Explicit interface kell

- Ha két modul kommunikál egymással, akkor annak ki kell derülnie legalább az egyikük szövegéből.

# A modularitás alapelvei (folyt.)

---

## Információ elrejtés

- minden információ egy modulról rejtett kell legyen, kivéve, amit explicit módon nyilvánosnak deklaráltunk.

## Nyitott és zárt modulok

- Zárt modul: más modulok számára egy jól definiált felületen keresztül elérhető, a többi modul változatlan formában felhasználhatja.
- Nyitott modul: kiterjeszthető
  - az általa nyújtott szolgáltatások bővíthetők vagy
  - hozzávehetünk további mezőket a benne levő adatszerkezetekhez, s ennek megfelelően módosíthatjuk eddigi szolgáltatásait.

# Az újrafelhasználhatóság igényei

---

A típusok változatossága

- A modulok többféle típusra is működjenek.

Adatstruktúrák és algoritmusok változatossága

Egy típus – egy modul

- Egy típus műveletei kerüljenek egy modulba.

Reprezentáció-függetlenség

# Mi a programozási nyelv?

---

Egy jelölés ...

Eszköz

- a számítógép vezérlésére
- programozók közötti kommunikációra
- algoritmusok leírására
- magas szintű tervezésre
- a feladat bonyolultságának kezelésére

# Programozási nyelvek kialakulása

---

Gépi kód

Assembly nyelv

- Mnemonikok, címkék használata
- Assembler

Magas szintű nyelv

- Utasításkészlete független egy adott számítógép-architektúra utasításkészletétől: végrehajtásuk előtt egy fordítóprogramnak kell őket assembly kódra, illetve gépi kódra fordítani.

# Neumann János (1903 – 1957)

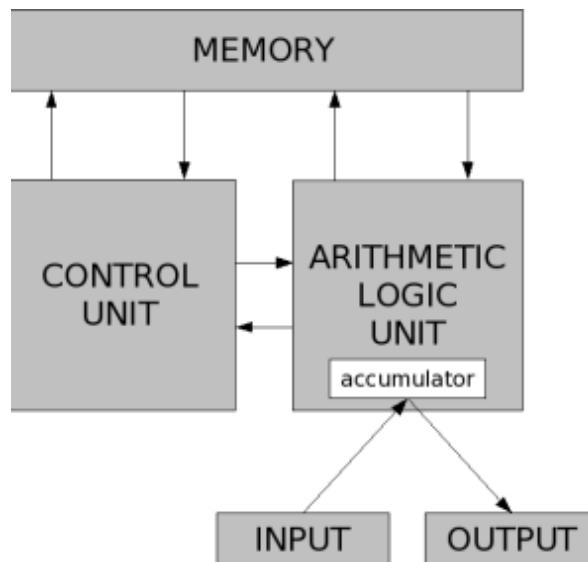


Neumann–elvek a számítógépről:

- Legyen univerzális, vagyis bármilyen feladat megoldására használható.
- Legyen soros működésű, tehát az utasításokat sorban egymás után hajtsa végre.
- Legyen teljesen elektronikus működésű.
- Az egyes utasítások és adatok leírásához és feldolgozásához a kettes számrendszeret használja.
- Legyen belső memóriája.
- Tárolt program elven működjön.  
Egy program indításakor a programot alkotó utasítások és a működéshez szükséges adatok is kerüljenek be a belső memóriába, és az utasításokat innen kiolvasva, sorban egymás után hajtsa végre a berendezés.

# Neumann János – 1946

A számítógép a következő egységekből álljon:



- Központi egység: vezérlés, műveletvégzés, adatmozgatás
- Memória: az adatok és a programok tárolása a program végrehajtása közben
- Bemeneti-kimeneti egység: kommunikáció a külvilággal, az adatbevitel- és kivitel megvalósítása

# A „Little Man Computer”

---

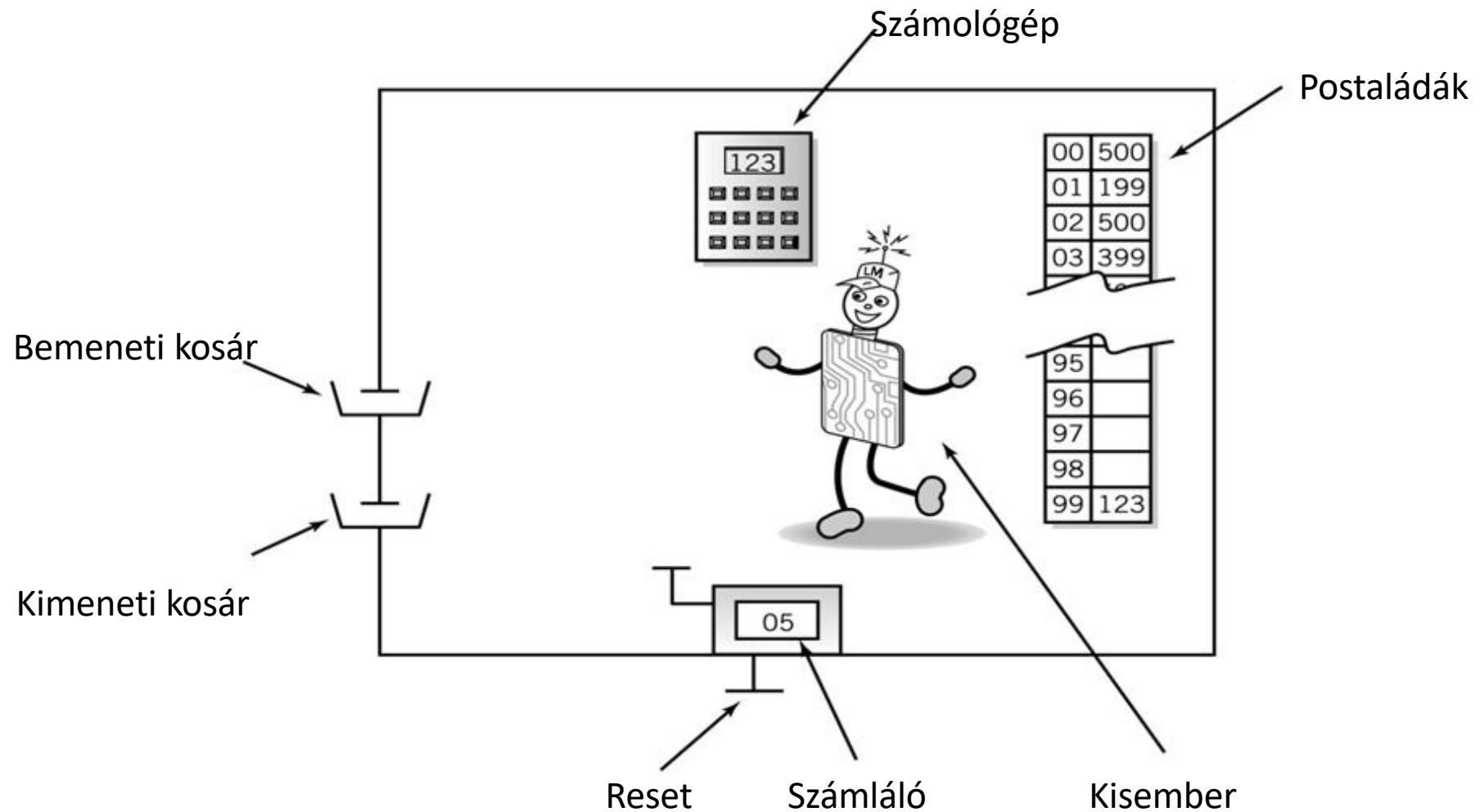
Neumann elvek az oktatásban

Stuart Madnick, MIT

Szoba, benne:

- Kisember
- 100 postaláda (00-99)
- Helyszámláló (2)
- Bemenő és kimenő kosár
- Számológép

# A „Little Man Computer”



# LMC elemei

---

Bemeneti és kimeneti kosár:

- Kisember kap/beletesz egy 3 jegyű számot tartalmazó cédulát

Számláló:

- 0 és 99 között tud számolni
- Pedál megnyomására a számlálóban tárolt szám értéke eggyel megnő
- Értékét nullára állíthatjuk egy külső úgynvezett „reset” (beállító) gombbal.

# LMC elemei (folyt.)

---

Számológép:

- Háromjegyű decimális számokat tud kezelni.
- Képes
  - Kivonni
  - Összeadni
  - A begépelt vagy a számítás eredményeként kapott értéket eltárolni.

# Postaládák: Cím - Tartalom

---

A postaládákban számokat tárolunk  
– elérésük a postaláda címe (sorszáma) alapján

A címek folyamatosan egymásután jövő értékek  
00 .. 99

A tartalom lehet:

- Adat vagy
- Utasítás

Cím	Tartalom
00	522
01	123

# Tartalom: Utasítások

---

Műveleti kód

Operandus

- A művelet operandusa: adat vagy az adat címe

Cím	Tartalom	
	Műv. kód	Operandus

# Korlátok

---

Nem foglalkozunk azzal, hogyan

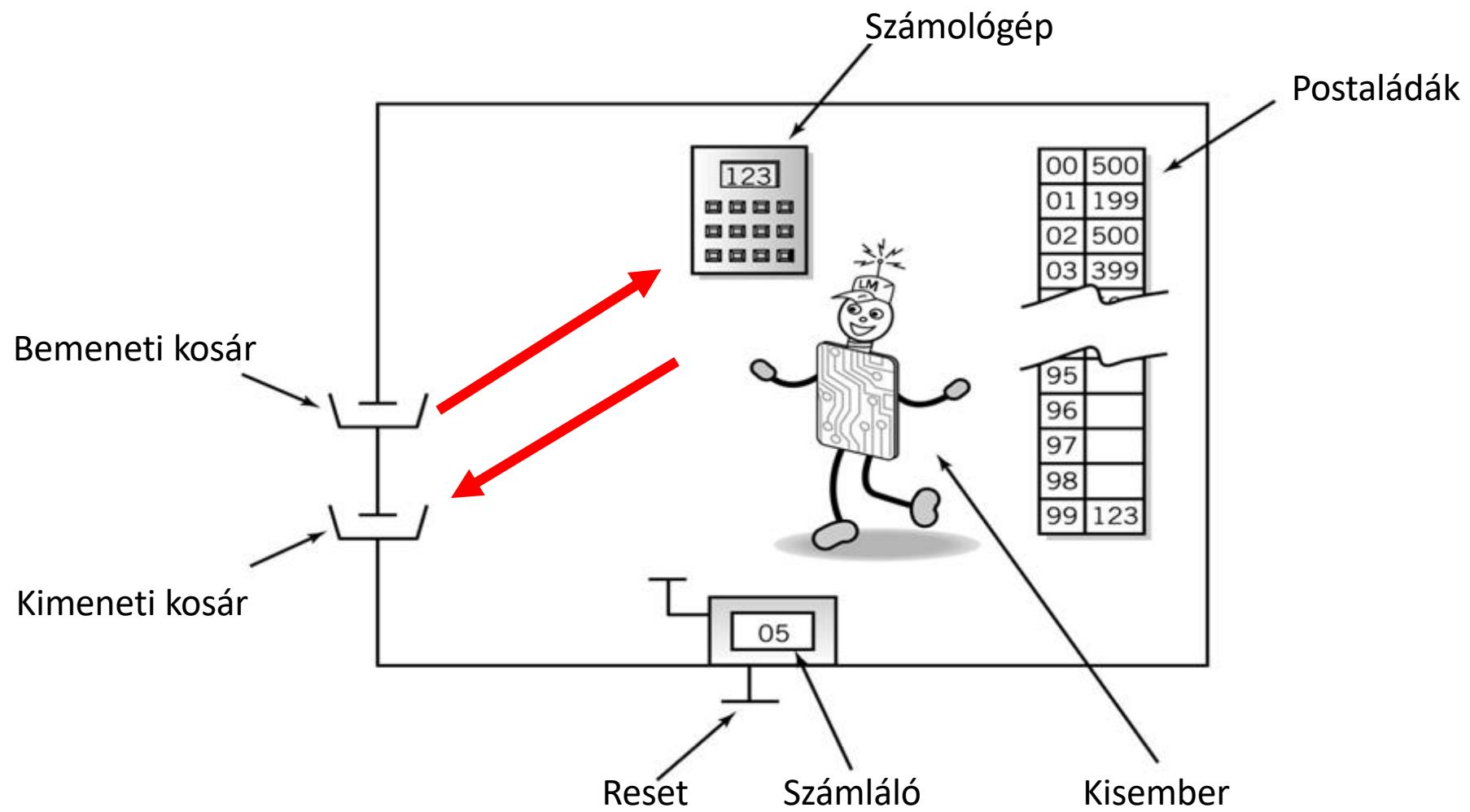
- töltük be a programot a memóriába
- tegyünk adatot a bemenő kosárba

# Utasításkészlet

---

Aritmetikai	1xx	Összeadás
	2xx	Kivonás
Adat mozgató	3xx	Tárolás
	5xx	Betöltés
Input/Output	901	Beolvasás
	902	Kiírás
Vezérlés (Kávészünet)	000	Leállás

# Input/Output



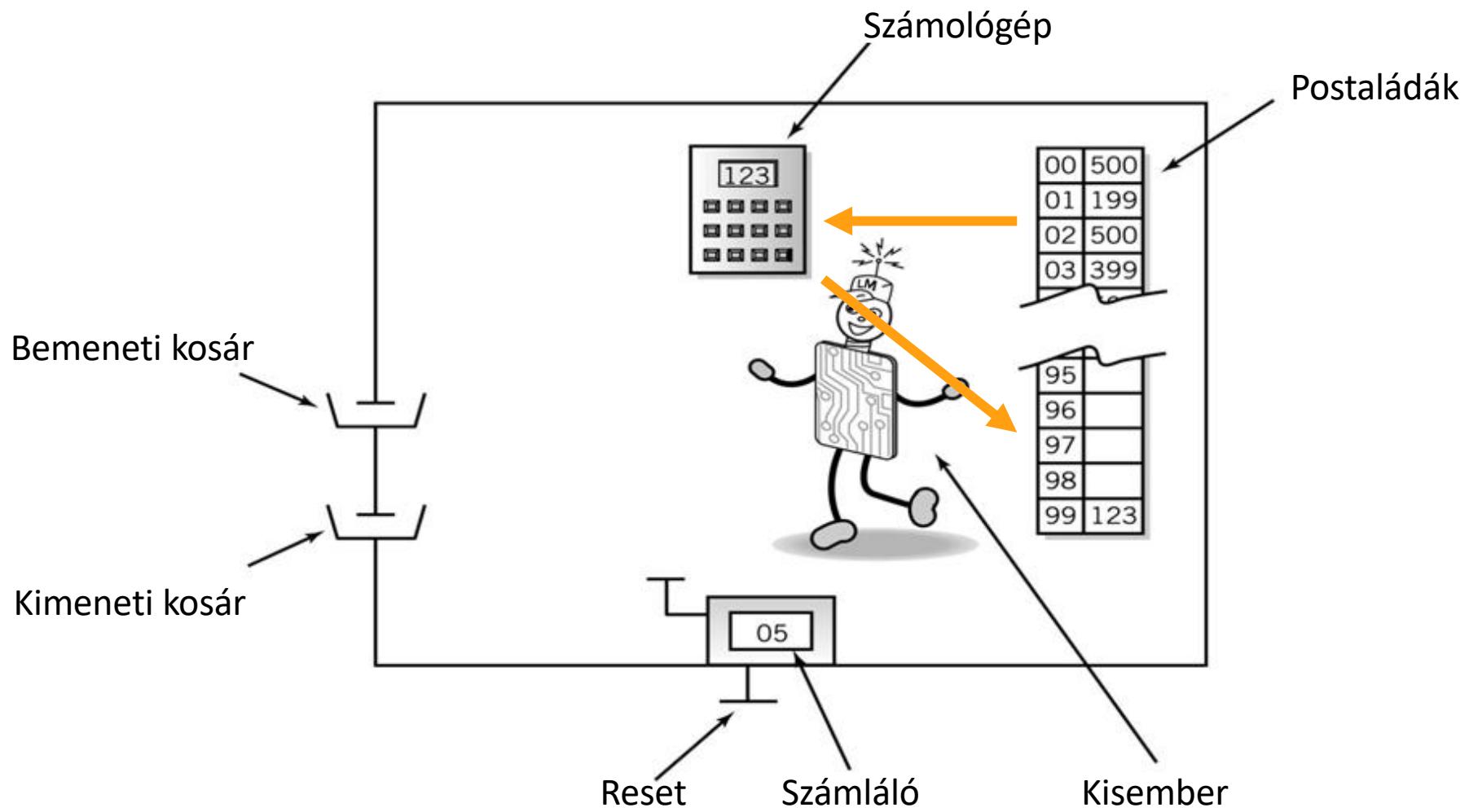
# Input/Output

Adatok mozgatása a számológép és a bemeneti/kimeneti kosarak között

Beolvasás  
Kiírás

Tartalom	
Műveleti kód	Operandus (cím)
9	01
9	02

# LMC belső adatmozgatás



# Belső adatmozgatás

A postaláda és a számológép között

Tárolás

Betöltés

Tartalom	
Műveleti kód	Operandus (cím)
3	xx
5	xx

# Adatok tárolása

---

Az utasításokat és az adatokat tároló postaláda fiókok fizikailag azonosak

Általában nincsenek beágyazva az utasítások közé

# Aritmetikai utasítások

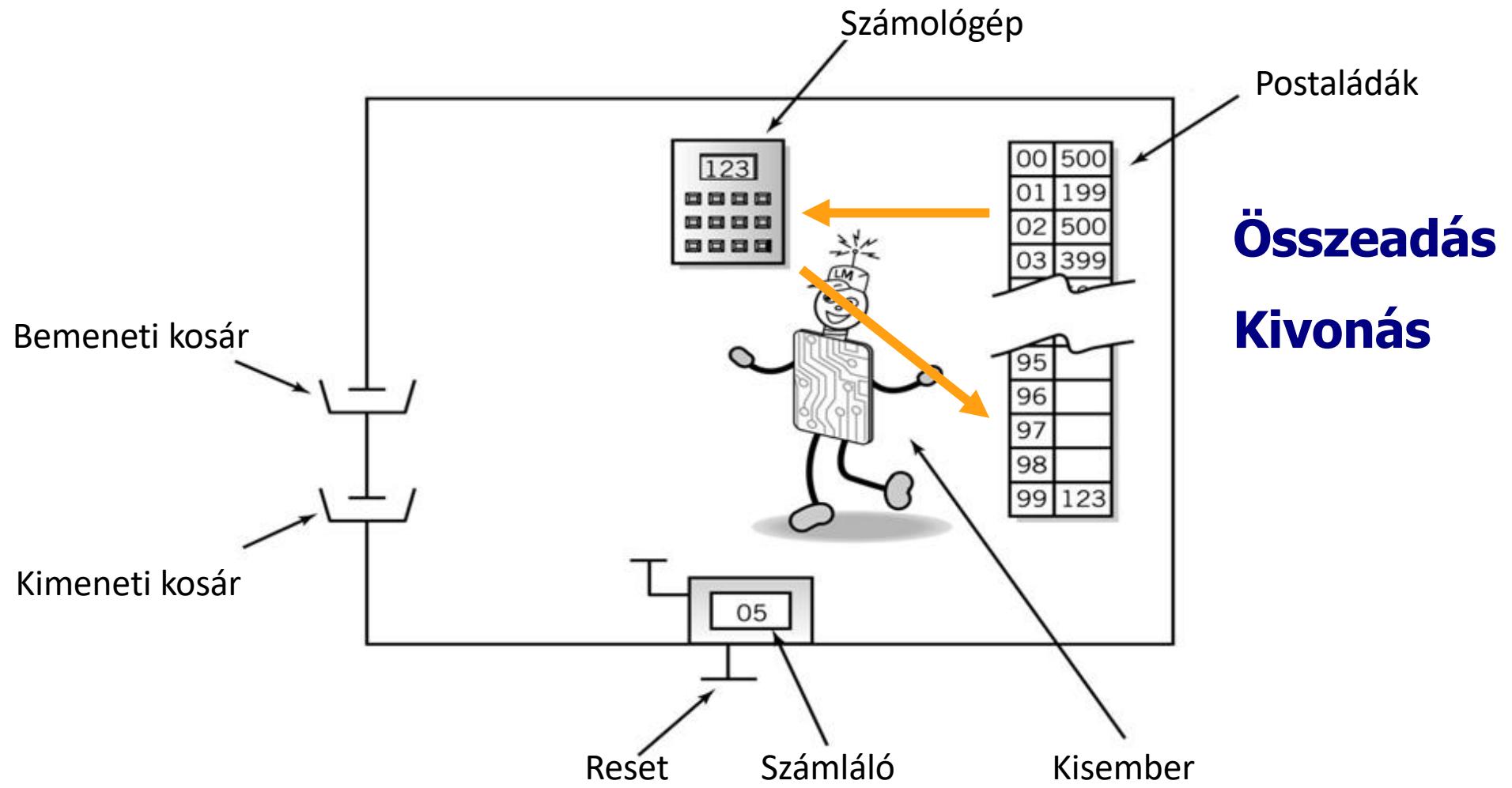
Olvassuk el az operandus által meghatározott leveleslágában lévő értéket

Hajtsuk végre a műveletet a számológéppel

Összeadás  
Kivonás

Tartalom	
Műveleti kód	Operandus (cím)
1	xx
2	xx

# LMC Aritmetikai utasítások



# Utasítás-végrehajtási ciklus

---

## Az utasítások végrehajtása

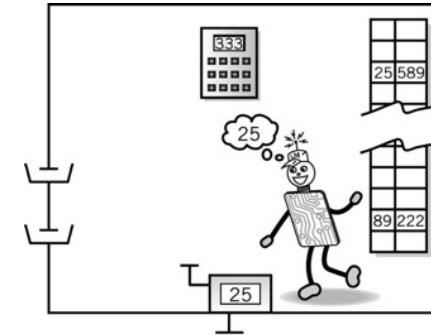
- Kikerésés (Fetch): Kisember kikeresi, hogy melyik utasítást kell végrehajtani
- Végrehajtás: Kisember elvégzi a munkát.

A számítógép által egy-egy utasítás végrehajtásakor elvégzett tevékenység-sorozatot utasítás-végrehajtási ciklusnak nevezzük

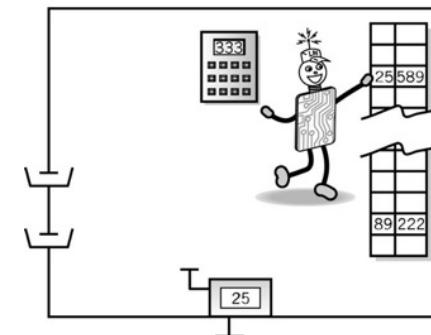
- Mivel ezek ciklikusan ismétlődnek.

# Utasítás-végrehajtási ciklus kikeresési rész

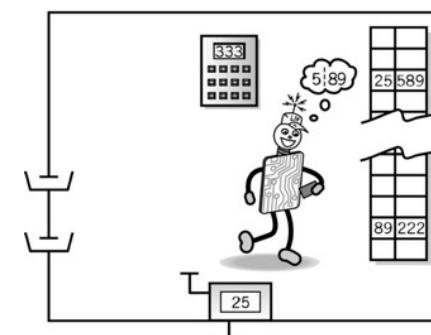
1. Kisember kiolvassa az utasításszámlálóból annak a postaládának a sorszámát (címét), ami az utasítást tartalmazza
2. Átsétál ahhoz a postaládához, ami megfelel a számláló értékének.



(1) The Little Man reads the address from the location counter



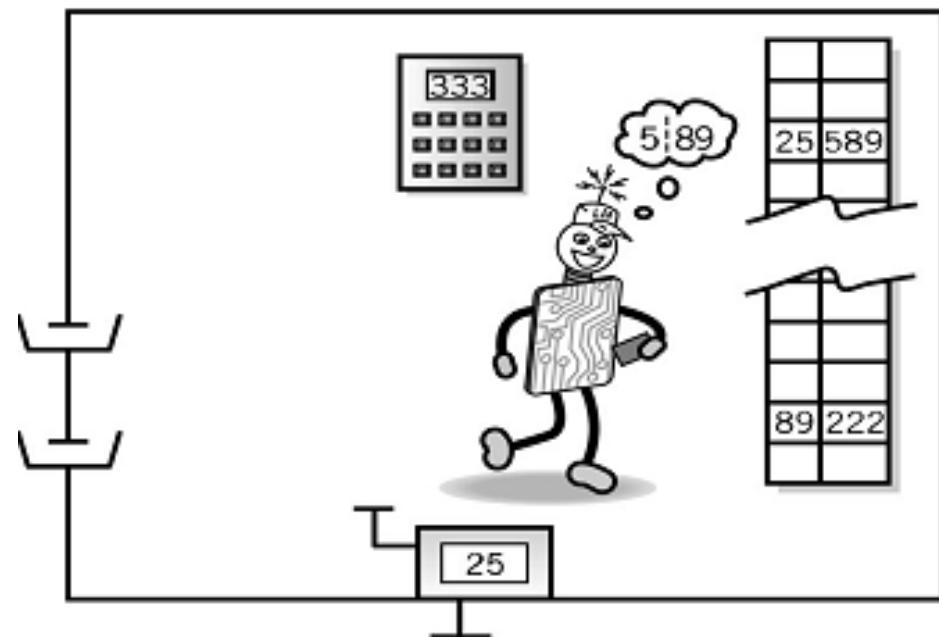
(2) . . . walks over to the mailbox that corresponds to the location counter



(3) . . . and reads the number on the slip of paper. (He then puts the slip of paper back, in case he should need to read it again later.)

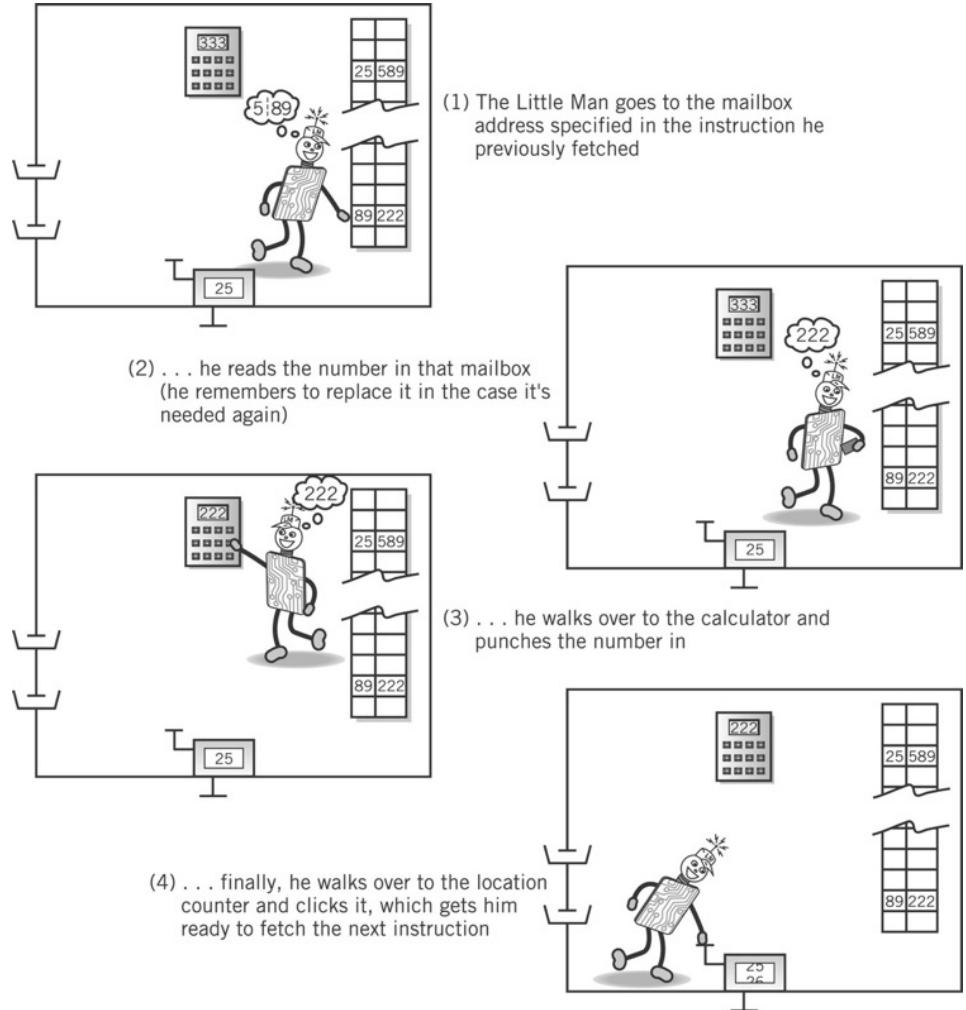
# Kikeresés (folyt.)

3. Kiveszi belőle a cédrát, elolvassa és dekódolja a számot, ami rajta van (a cetröt visszateszi, arra az esetre, ha esetleg később szükség lenne rá, hogy újra elolvassa)



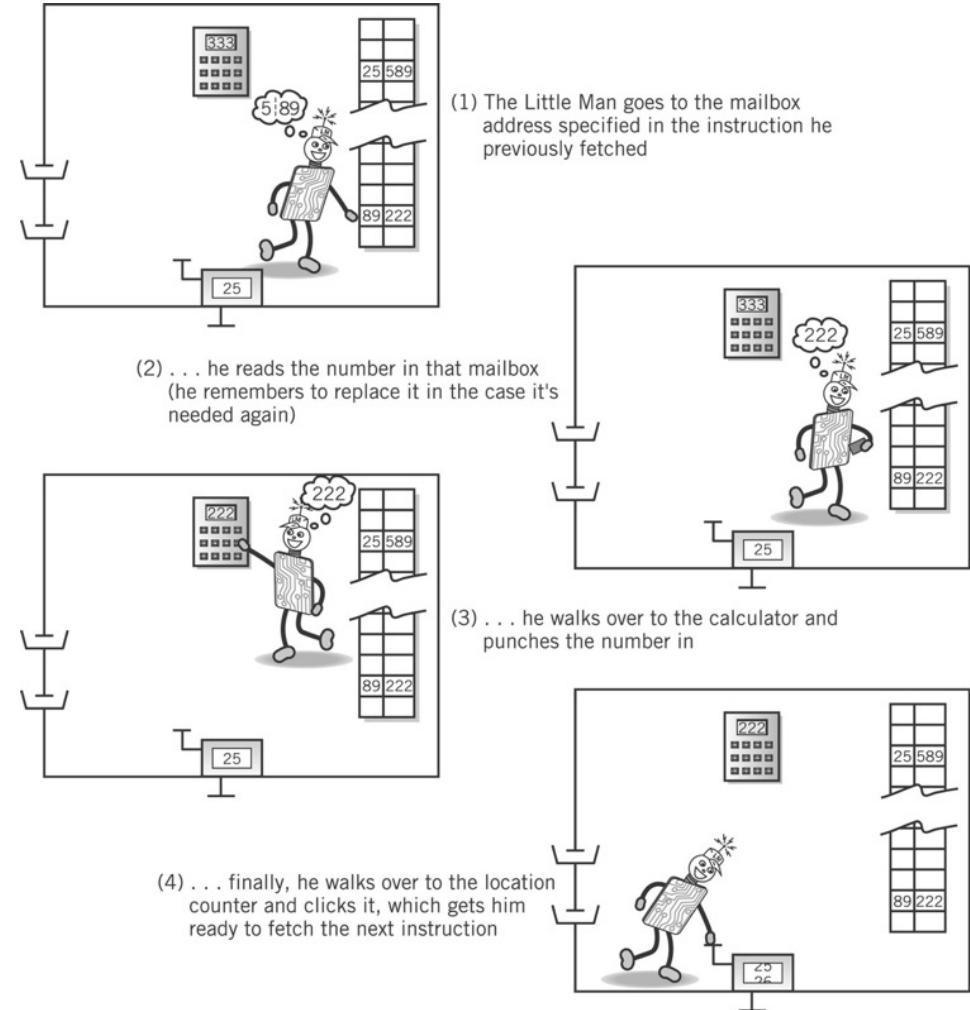
# Végrehajtási rész

1. Kisember odamegy a postaládához, aminek a sorszámát az éppen kikeresett utasítás tartalmazza.
2. Kiveszi belőle a cédratát és elolvassa a rajta lévő számot (nem felejti el a cédratát visszatenni, mert később is szükség lehet rá).



# Végrehajtási rész (folyt.)

3. A kiolvasott utasítás kódja '5' volt, tudja, hogy ez azt jelenti, hogy „betöltés”, így odamegy a számológéphez és beüti az imént kiolvasott számot.
4. Odasétál a számlálóhoz és továbblöki, így folytathatja a következő utasítás kikeresésével.



# Egyszerű program

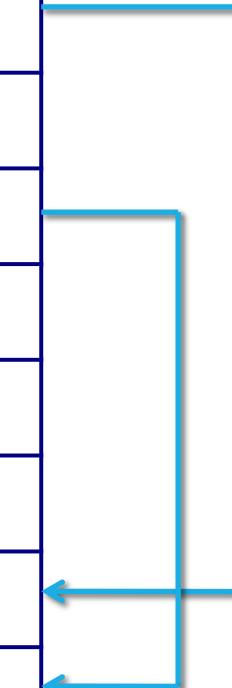
---

Adjuk meg két szám különbségét

- Olvassuk be az első számot
- Tároljuk el
- Olvassuk be a második számot
- Tároljuk el
- Vonjuk ki belőle az elsőt
- Írjuk ki az eredményt
- Fejezzük be a munkát

## Két szám különbségét kiszámító program gépi kódjának részletei

Postaláda	Kód
00	901
01	307
02	901
03	308
04	207
05	902
06	000
07	000
08	000



## Két szám különbségét kiszámító program gépi kódban

Postaláda	Kód	Utasítás leírása
00	901	Első szám beolvasása
01	307	Szám tárolása
02	901	Második szám beolvasása
03	308	Szám tárolása
04	207	Az első kivonása a 2.-ból
05	902	Eredmény kiírása
06	000	Stop
07	000	Adatrekesz
08	000	Adatrekesz

## Változtassunk rajta egy egész kicsit: Számítsuk ki két szám különbségének abszolút értékét!

---

1. Olvassuk be az első számot
2. Tároljuk el
3. Olvassuk be a másik számot
4. Tároljuk el
5. Vonjuk ki belőle az elsőt
6. Ha pozitív az eredmény, folytassuk a 9.-nél
7. (Különben) törlük be az első számot
8. Vonjuk ki belőle a másodikat
9. Írjuk ki az eredményt
10. Fejezzük be a munkát

## Számítsuk ki két szám különbségének abszolút értékét!

---

1. Olvassuk be az első számot
2. Tároljuk el
3. Olvassuk be a másik számot
4. Tároljuk el
5. Vonjuk ki belőle az elsőt
6. Ha pozitív az eredmény, folytassuk a 9.-nél
7. (Különben) törlük be az első számot
8. Vonjuk ki belőle a másodikat
9. Írjuk ki az eredményt
10. Fejezzük be a munkát

# Program vezérlés

Elágazás utasítások (az utasítások egymásutánjából való kiugrás lehetősége)

- Megváltoztatja a címet az utasítás számlálóban

## Leállás

Ugrás az xx számú postafiókra

Ha a számológép tartalma 0, ugrás az xx számú postafiókra

Ha a számológép tartalma >0, ugrás az xx számú postafiókra

Tartalom	
Műv. kód	Operandus (cím)
6	xx
7	xx
8	xx

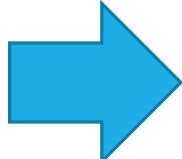
# LMC utasításkészlete

---

Hozzáadás	1xx
Kivonás	2xx
Tárolás	3xx
Betöltés	5xx
Ugrás	6xx
Ugrás 0-nál	7xx
Ugrás pozitívánál	8xx
Input	901
Output	902
Stop	000

## Számítsuk ki két szám különbségének abszolút értékét!

Postaláda	Kód
00	901
01	307
02	901
03	308
04	207
05	902
06	000
07	000
08	000



Postaláda	Kód
00	901
01	310
02	901
03	311
04	210
05	808
06	510
07	211
08	902
09	000
10	000
11	000

# Számítsuk ki két szám különbségének abszolút értékét!

---

00	901	Első szám beolvasása
01	310	Szám tárolása
02	901	Második szám beolvasása
03	311	Szám tárolása
04	210	Első számot kivonjuk
05	808	Ha pozitív, ugorj a 08-ra (a kiírásra)
06	510	(különben) töltsd be az elsőt
07	211	Vond ki belőle a másodikat
08	902	Írd ki az eredményt
09	000	Stop
10	000	Az első adat helye
11	000	A második adat helye

# Mnemonikok kellenek! LMC utasításkészlete

---

Hozzáadás	1xx	ADD xx
Kivonás	2xx	SUB xx
Tárolás	3xx	STO xx
Betöltés	5xx	LDA xx
Ugrás	6xx	BR xx
Ugrás 0-nál	7xx	BRZ xx
Ugrás pozitívnál	8xx	BRP xx
Input	901	IN
Output	902	OUT
Stop (Coffee break)	000	COB
Adattároló		DAT

# Számítsuk ki két szám különbségének abszolút értékét!

00	IN	901	Első szám beolvasása
01	STO 10	310	Szám tárolása
02	IN	901	Második szám beolvasása
03	STO 11	311	Szám tárolása
04	SUB 10	210	Első számot kivonjuk
05	BRP 08	308	Ha pozitív, ugorj a 08-ra (a kiírásra)
06	LDA 10	510	(különben) töltsd be az elsőt
07	SUB 11	211	Vond ki belőle a másodikat
08	OUT	902	Írd ki az eredményt
09	COB	000	Stop
10	DAT 00	000	Az első adat helye
11	DAT 00	000	A második adat helye

# Adjunk nevet a változóknak és az utasításoknak!

---

	00	IN	901	Első szám beolvasása
	01	STO <b>ELS</b>	310	Szám tárolása
	02	IN	901	Második szám beolvasása
	03	STO <b>MAS</b>	311	Szám tárolása
	04	SUB <b>ELS</b>	210	Első számot kivonjuk
	05	BRP <b>KI</b>	808	Ha pozitív, ugorj a 08-ra (a kiírásra)
	06	LDA <b>ELS</b>	510	(különben) töltsd be az elsőt
	07	SUB <b>MAS</b>	211	Vond ki belőle a másodikat
<b>KI</b>	08	OUT	902	Írd ki az eredményt
	09	COB	000	Stop
<b>ELS</b>	10	DAT 00	000	Az első adat helye
<b>MAS</b>	11	DAT 00	000	A második adat helye

# Példa

---

Mit csinál?

IN

STO ELS

IN

ADD ELS

OUT

COB

ELS DAT 00

# Másik példa

---

És ez mit csinál?

IN

LOOP SUB ONE

OUT

BRZ QUIT

BR LOOP

QUIT COB

ONE DAT 1

# Assembly nyelv

---

Jellemzői:

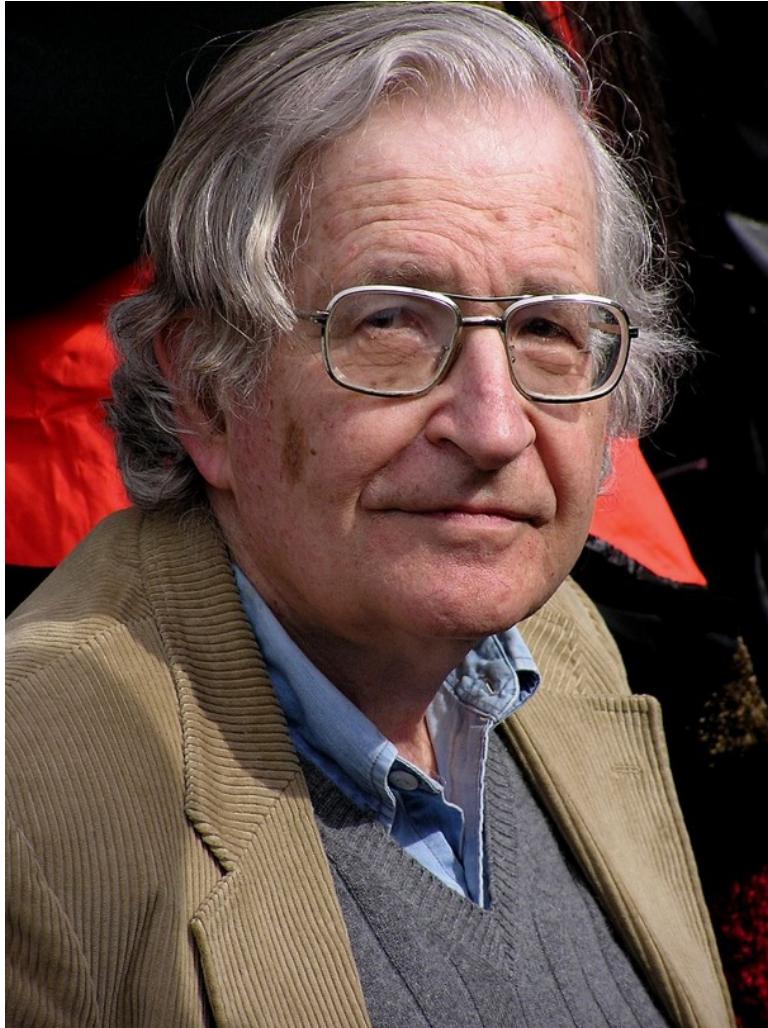
- CPU specifikus
- 1 - 1 megfelelés az assembly nyelvű utasítás és a gépi nyelvű utasítás között (makró assembly később alakult ki)
- **Mnemonikok** reprezentálják az utasításokat
- Változók deklarálhatók
- Utasítások címkézhetők

Assembler lefordítja

- inputja az assembly nyelvű program
- outputja a gépi nyelvű

# Noam Chomsky (1928-)

---



Amerikai nyelvész,  
A generatív nyelvtan  
elméletének megalkotója

# Generatív grammatikák

---

Természetes nyelvek vizsgálata

Gyerek – véges számú mondatot hall

- Képes tetszőleges nyelvtanilag helyes mondat megalkotására
- Kell legyen egy szabályrendszer minden nyelvben!

# Generatív grammatikák

---

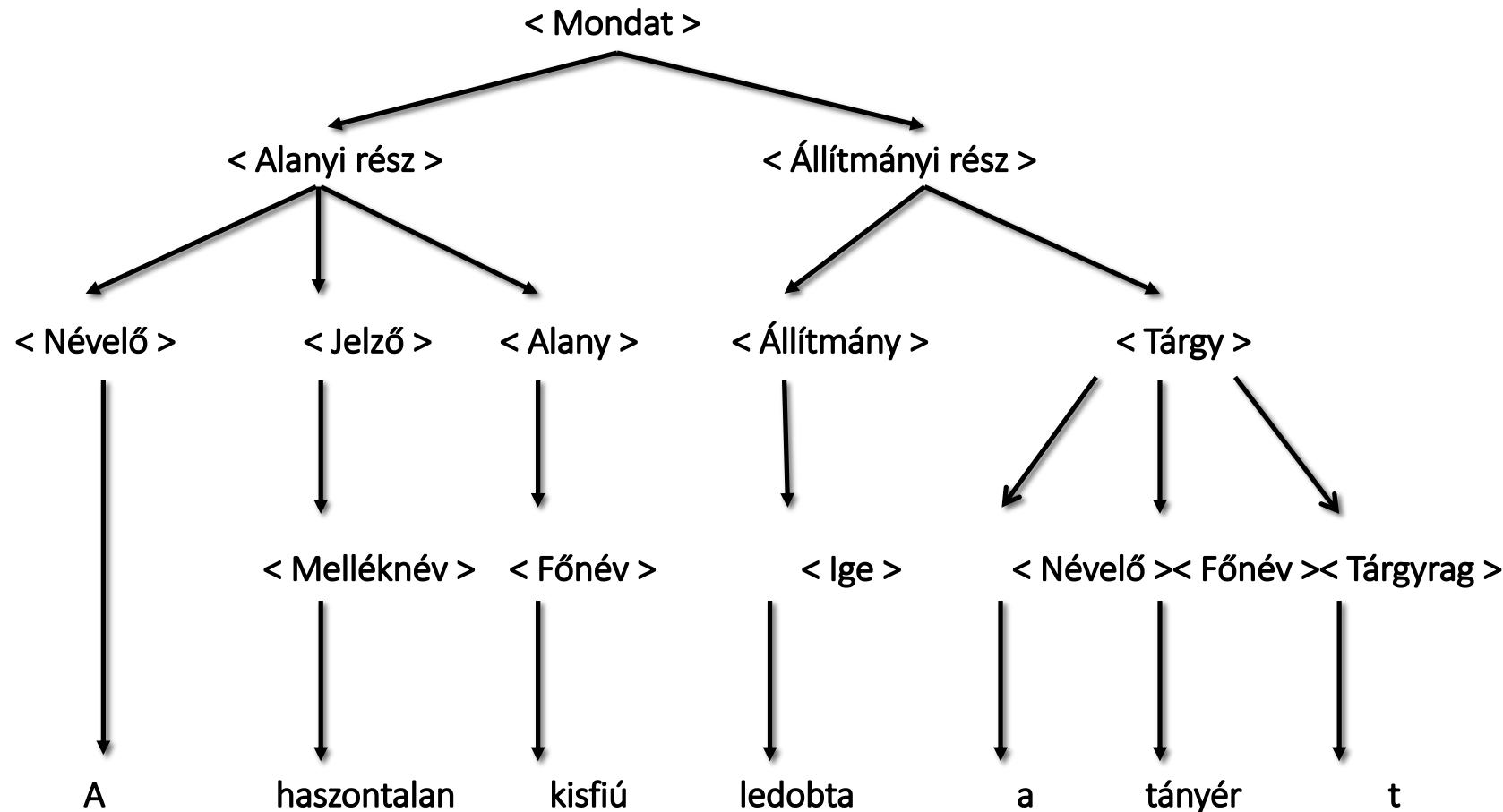
„A haszontalan kisfiú ledobta a tányért.”

- Ez egy mondat → felbontható alanyi és állítmányi részre
- Az alanyi rész névelő, jelző és alany sorozata lehet
- Az állítmányi rész az állítmány és a tárgy egymásutánja, stb.
- A struktúra leírásakor úgynévezett „grammatikai jelek” is használhatunk a mondatrészek, illetve szófajok jelzésére – ezeket <> zárójelek közé tesszük.

---

< Mondat >	→	< Alanyi rész >< Állítmányi rész>
< Alanyi rész >	→	< Névelő >< Jelző >< Alany >
< Névelő >	→	a
< Jelző >	→	< Melléknév >
< Alany >	→	< Főnév >
< Főnév >	→	kisfiú I tányér
< Melléknév >	→	haszontalan
< Állítmányi rész >	→	< Állítmány > < Tárgy >
< Állítmány >	→	< Ige >
< Ige >	→	ledobta
< Tárgy >	→	< Névelő >< Főnév >< Tárgyrag >
< Tárgyrag >	→	t

# Szintaxisfa



# Legbaloldalibb vezetés

---

< Mondat> → <Alanyi rész> <Állítmányi rész > →  
→< Névelő >< Jelző >< Alany> <Állítmányi rész > →  
→ a < Jelző >< Alany> <Állítmányi rész > →  
→ a <Melléknév>< Alany> <Állítmányi rész > →  
→ a haszontalan < Alany> <Állítmányi rész > →  
→ a haszontalan < Főnév><Állítmányi rész > →  
→ a haszontalan kisfiú <Állítmányi rész > →  
→ a haszontalan kisfiú <lge> < Tárgy > →  
→ a haszontalan kisfiú ledobta < Tárgy > > →  
→ a haszontalan kisfiú ledobta < Névelő >< Főnév >< Tárgyrag > →  
→ a haszontalan kisfiú ledobta a < Főnév >< Tárgyrag > →  
→ a haszontalan kisfiú ledobta a tányér < Tárgyrag >  
→ a haszontalan kisfiú ledobta a tányért

---

## Ugyanígy eljuthatunk a

- „a haszontalan tányér ledobta a kisfiút”
- „a haszontalan kisfiú ledobta a kisfiút”
- „a haszontalan tányér ledobta a tányért”
- mondatokhoz is ☺

## Nehéz a jó megoldás!

- Természetes nyelvek helyett vizsgáljuk most inkább a mesterséges nyelveket!

# Nyelvi elemek, alapfogalmak

---

# Általános fogalmak

---

A nyelv **szintaxisa** azoknak a szabályoknak az összessége, amelyek az adott nyelven írható összes lehetséges, **formailag helyes** programot (jelsorozatot) definiálják.

- Reguláris kifejezések, BNF forma

Az adott nyelv programjainak jelentését **leíró szabályok** összessége a nyelv **szemantikája**.

# Szintaxis és szemantika

---

Példa:

- DD / DD / DDDD
- 02 / 12 / 2018

2018. december 2. vagy 2018. február 12.?

A szintaxis befolyásolja a programok megbízhatóságát

# A programok végrehajtása

---

Az **interpreter** egy utasítás lefordítása után azonnal végrehajtja azt

A **fordítóprogram** átalakítja a programot egy vele ekvivalens formára, ez lehet a számítógép által (majdnem) közvetlenül végrehajtható forma, vagy lehet egy másik programozási nyelv.

# Fordítás

---

A lefordítandó program a **forrásprogram**.

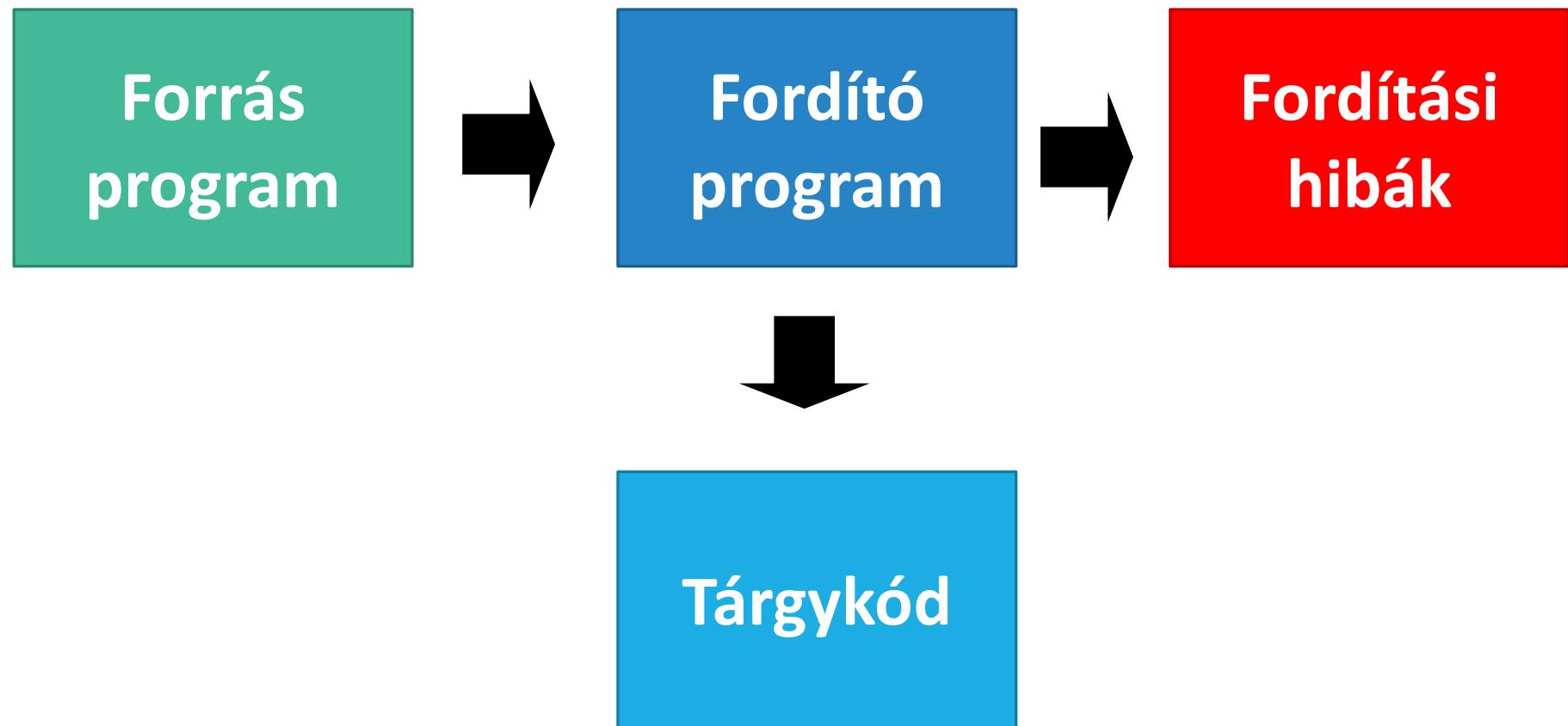
A **fordítás** eredményeként kapott program a **tárgyprogram**.

Az az idő, amikor az utasítások fordítása folyik, a **fordítási idő**.

Az az idő, amikor az utasítások végrehajtása folyik, a **végrehajtási idő**.

# Fordítás

---



# Nyelv leírása a fordítónak

---

Formális nyelv

Grammatika – generálja

Automaták – felismerik

# Formális nyelvek

---

**Definíció:** Az **abc** tetszőleges szimbólumok véges, nem üres halmaza, jelöljük  $\Sigma$ -val.

**Definíció:** A **szó** az abc elemeiből képezett  $a_1a_2\dots a_k$  alakú sorozat, ahol  $k \geq 0$ ,  $a_1, a_2, \dots, a_k \in \Sigma$

**Definíció:** A  $\Sigma^*$  a  $\Sigma$  ábécé elemeiből képezhető összes szavak halmaza:

$$\Sigma^* = \{a_1a_2\dots a_k \mid k \geq 0, a_1, a_2, \dots, a_k \in \Sigma\}$$

Ha  $k=0$ , akkor a szó üres szó, jele  $\varepsilon$  vagy  $\lambda$

**Definíció:**  $\Sigma^+$  a nem üres szavak halmaza:

$$\Sigma^+ = \Sigma^* \setminus \{\lambda\}$$

# Példa

---

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

$$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

# Grammatikák

---

## Definíció: Generatív nyelvtan (G)

$G = (N, \Sigma, S, P)$ , ahol

$N$  a nemterminálisok (grammatikai jelek) abc-je

$\Sigma$  a terminálisok abc-je,  $N \cap \Sigma = \emptyset$

$S \in N$  a kezdőszimbólum

$P: \alpha \rightarrow \beta$  alakú átírási szabályok véges halmaza

$\alpha$  a szabály baloldala,  $\beta$  a szabály jobboldala

$\alpha, \beta \in (N \cup \Sigma)^*$

$\alpha$ -ban van legalább egy nemterminális szimbólum

---

## **Definíció: Közvetlen levezetés (közvetlen levezetési reláció, $\Rightarrow_G$ )**

Legyen  $\gamma, \delta \in (N \cup \Sigma)^*$

$\gamma \Rightarrow_G \delta$ , ha van olyan  $\alpha \rightarrow \beta \in P$  szabály és vannak olyan  $\alpha', \beta' \in (N \cup \Sigma)^*$  szavak, hogy

$$\gamma = \alpha' \alpha \beta' \text{ és } \delta = \alpha' \beta \beta'$$

$\Rightarrow_G^n$  - n lépéses levezetés

$\Rightarrow_G^+$  - legalább egy lépéses levezetés

$\Rightarrow_G^*$  - levezethetőség (esetleg nulla lépésekben is)

---

## **Definíció: G grammatika által generált nyelv $L(G)$**

Legyen  $G=(N, \Sigma, S, P)$  generatív nyelvtan.

$$L(G) = \{u \in \Sigma^* \mid S \Rightarrow_G^* u\}$$

## **Definíció: Ekvivalens nyelvtanok**

Legyenek  $G=(N, \Sigma, S, P)$  és  $G'=(N', \Sigma', S', P')$  generatív nyelvtanok.

$G$  ekvivalens  $G'$ -vel, ha  $L(G)=L'(G')$ .

# Chomsky nyelvosztályok

---

## Definíció: Chomsky nyelvosztályok

0 típusú nyelvtan (**kifejezés struktúrájú**), ha semmilyen korlátozás nincs

1 típusú nyelvtan (**környezetfüggő**),  
ha P-ben minden szabály  $\alpha A\beta \rightarrow \alpha\delta\beta$  alakú, ahol  
 $\delta \neq \lambda$ , kivéve esetleg az  $S \rightarrow \lambda$  szabályt, mely esetben  
 $\alpha = \beta = \delta = \lambda$ , de ekkor S nem szerepelhet egyetlen  
szabálynak sem a jobboldalán. ( $\alpha, \beta, \gamma, \delta \in (N \cup \Sigma)^*$ )

# Chomsky nyelvosztályok (folyt.)

---

2 típusú nyelvtan (**környezetfüggetlen**),  
ha P-ben minden szabály  $A \rightarrow \alpha$  alakú.  
( $A \in N, \alpha \in (N \cup \Sigma)^*$ )

3 típusú nyelvtan (**reguláris**, jobblineáris),  
ha P-ben minden szabály  
 $A \rightarrow xB$ , vagy  $A \rightarrow x$  alakú. ( $A, B \in N, x \in \Sigma^*$ )

# Chomsky nyelvosztályok (folyt.)

---

## Definíció: Nyelv típusa

Egy  $L \subseteq \Sigma^*$  nyelv i típusú ( $i=0, 1, 2, 3$ ),  
ha van olyan i típusú nyelvtan, ami éppen  $L$ -et  
generálja.

$$L_0 \supset L_1 \supset L_2 \supset L_3$$

# Példák

---

Reguláris grammatika:  $G = (\{S\}, \{a,b\}, S, P)$

P:  $S \rightarrow abS$

$S \rightarrow a$

$S \Rightarrow a$

$S \Rightarrow abS \Rightarrow aba$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow ababa = (ab)^2a$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow abababa = (ab)^3a$

$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow ababababS \Rightarrow ababababa = (ab)^4a$

$L(G) = \{(ab)^n a \mid \text{ahol } n \geq 0\}$

$L(G) = (ab)^*a$

# Példák

---

Környezetfüggetlen (Context-free, CF ) grammatika:

$$G = (\{S\}, \{a,b\}, S, P)$$

P:  $S \rightarrow aSb$

$$S \rightarrow \lambda$$

$$S \Rightarrow \lambda$$

$$S \Rightarrow aSb \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb = a^2b^2$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb = a^3b^3$$

....

$$L(G) = \{a^n b^n \mid \text{ahol } n \geq 0\}$$

# Reguláris kifejezések

---

Definíció: Egy  $S$  abc feletti reguláris kifejezések:

1.  $S$  minden eleme.
2.  $\lambda$  (vagy e)
3.  $\emptyset$
4. Ha a és b reguláris kifejezések, akkor  $(ab)$  is az.
5. Ha a és b reguláris kifejezések, akkor  $(a \cup b)$  is az.
6. Ha a reguláris kifejezés, akkor  $a^*$  is az.

Minden reguláris kifejezés reprezentál egy nyelvet.  
Példa reguláris kifejezésre és nyelvre:

$(a^* bc)^*ab$

$(\{a\}^*\{b\}\{c\})^*\{a\}\{b\}$

---

## Minták megadása – „hagyományos” jelölések a metakarakterekre:

- Karakter megfeleltetés
  - . (pont) Bármilyen karakter
  - Pl. .a. ⇔ ‘pap’, ‘rak’, ‘lap’, ‘kap’, stb.
- [karakterek] A szögletes zárójelek között felsorolt karakterek valamelyikével megegyező karakter:
  - Pl. [abc] ⇔ ‘a’ vagy ‘b’ vagy ‘c’
  - A - (mínusz) jellel tartományt is megadhatunk.
  - Például [0-9] megfelel bármely számjegynek vagy [a-zA-Z] bármely kis vagy nagybetűnek.
- [^karakterek] az előző operátor tagadása
  - Pl. [^a-d] : bármely karakter, ami nem az ‘a’, ‘b’, ‘c’ vagy ‘d’

- 
- \d rövidítés: számjegyek 0-9
  - \w rövidítés: betűk, számjegyek, ‘\_’
  - \s rövidítés: whitespace (space,tab,CR,LF)
  - \D, \W, \S rövidítések: a fentiek negációi

## Többszörözés - ismétlő operátorok a minták után:

- ? : 0 vagy 1 egyezés - pl. \d?  $\Leftrightarrow \emptyset$ , vagy egy számjegy
- \* : 0 vagy több egyezés - pl. ab\*a  $\Leftrightarrow aa, aba, abba, \dots$
- + : 1 vagy több egyezés - pl. [0-9]+  $\Leftrightarrow 1, 45, 460, \dots$
- {n} : n egyezés ( $n \geq 1$ ) - pl. .{3}  $\Leftrightarrow aaa, 888, \dots$
- {n,m} : n és m közötti egyezés ( $n \geq 1, n \leq m$ ) –  
pl. a{2,4}  $\Leftrightarrow 'aaaa', 'aaa'$  vagy ‘aa’
- {n,} : legalább n egyezés ( $n \geq 1$ ) - pl. xa{2,}.  $\Leftrightarrow xaaaf, xaaa, \dots$
- {,m}: legfeljebb m egyezés

# Diszjunkció

---

- „|” (pipe) szimbólum két minta között: „vagy”  
pl. alA  $\Leftrightarrow$  ‘a’, ‘A’
  - lehet sorozatban több is:  
pl. alblc ugyanaz, mint [a-c]
- a „|” -jel precedenciája a legalacsonyabb az operátorok között. Ezért:
- teli  $\Leftrightarrow$  ‘te’, ‘i’
- t(elí)  $\Leftrightarrow$  ‘te’, ‘ti’

# Horgonyok

---

- ^ : az elejére illeszkedik  
pl. ^M : <‘M’ a kezdete>
- \$ : a végére illeszkedik:  
pl. \.\$ : <pont a végén>
- \b : szóhatárra illeszkedik (\w és \W karakterek közötti pozícióra,  
vagy \w és sztring kezdete/vége közé)  
pl. c\b : ‘abc’-re illeszkedik ‘abc def’-ben
- \B: \b negáltja (karakter „szó” közepén)

Speciális karakterek csak ‘\’ segítségével használhatók

- Pl. \. vagy \-

## Csoportok

- A zárójel operátorokkal tetszőleges számú csoport kijelölhető a mintában:

pl.  $a((bla)ko(k))(a)t$

- A csoportokra alkalmazhatók az operátorok:

pl.  $a(lmlkn)a$

# Hol használjuk?

---

Például a Googleban – vagy máshol – keresünk szöveget, bizonytalan az írásmódja – klasszikus példa:

- "Handel", "Händel" vagy "Haendel" ?
- H(älæ?)ndel

Fordítóprogramokban a Lexikális elemző – pl.:

- változónév: [a-zA-Z][a-zA-Z0-9]\*
- egész szám: (\+|\-)?[0-9]+
- valós szám: [0-9]+\.[0-9]\*

# Hol használjuk?

---

Webes beolvasások ellenőrzése

- születési dátum,
- bankszámlaszám, stb. is
- jelszó megbízhatóságának ellenőrzése

IP-cím formátum:

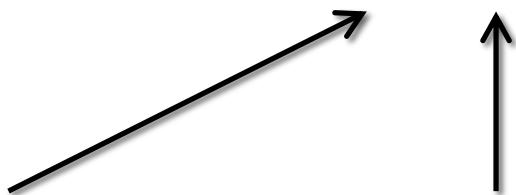
- <3-jegyű\_szám>.<3-jegyű\_szám>.<3-jegyű\_szám>.<3-jegyű\_szám>

RegKif: \b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b

# Context-Free Grammatika

$G = (N, \Sigma, S, P)$

ha  $P$ -ben minden szabály  $A \rightarrow \alpha$  alakú.



Grammatikai jel

Grammatikai jelek és  
terminálisok sorozata

$(A \in N, \alpha \in (N \cup \Sigma)^*)$

# A grammatika által generált nyelv

$G$   
grammatika       $S$   
kezdőszimbólum

$$L(G) = \{w : S \xrightarrow{*} w, w \in \Sigma^*\}$$

↑

Terminálisok sorozata vagy a  $\lambda$

# Context-Free nyelv definíció

---

Egy  $L$  nyelv környezetfüggetlen (CF) ha van egy  $G$  context-free grammatika, ahol

$$L = L(G)$$

# Levezetési fa

---

Definíció: Egy G CF gramatika szerinti levezetés egy fával ábrázolható

- A gyökere a kezdőszimbólum
- A levelek balról jobbra a levezetett szó betűi
- További csomópontjai nemterminálisok
- Az elágazások egy-egy alkalmazott szabálynak felelnek meg

Ez a levezetési fa.

# Példa

---

Context-free grammatica:

$G$

$$S \rightarrow aSa \mid bSb \mid \lambda$$

Példa levezetések:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaba$$

---

$$L(G) = \{ww^R : w \in \{a,b\}^*\}$$

Páros hosszú palindromák

# Másik példa

Context-free grammatica :

$$G : S \rightarrow aSb \mid SS \mid \lambda$$

Példa levezetések:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow ab$$

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abab$$

---

$$\begin{aligned} L(G) = \{w &: n_a(w) = n_b(w), \\ &\text{és } n_a(v) \geq n_b(v) \\ &\text{a } w \text{ bármely } v \text{ prefixében}\} \end{aligned}$$

Leírja a jó zárójelezést:

( ) (( )) ( ( ))

a = (,    b = )

# A matematikai kifejezések grammatikája

$$E \rightarrow E + E \quad | \quad E * E \quad | \quad (E) \quad | \quad a$$

Példa sorozatok:

$$(a + a) * a + (a + a * (a + a))$$


Tetszőleges szám

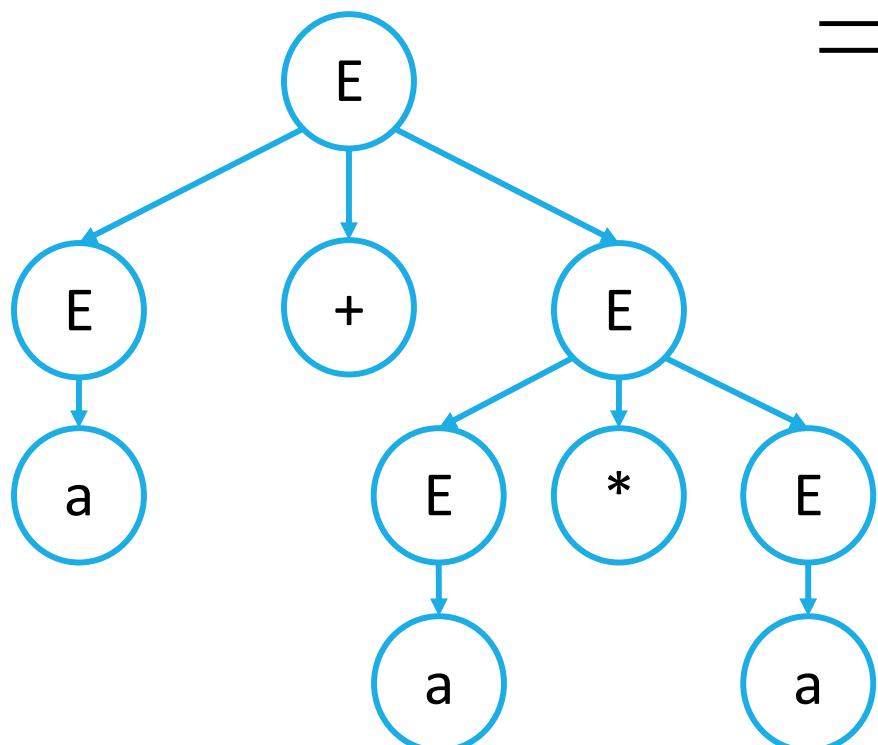
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E * E$$

$$\Rightarrow a + a * E \Rightarrow a + a * a$$

$$a + a * a$$

egy legbaloldalibb levezetése



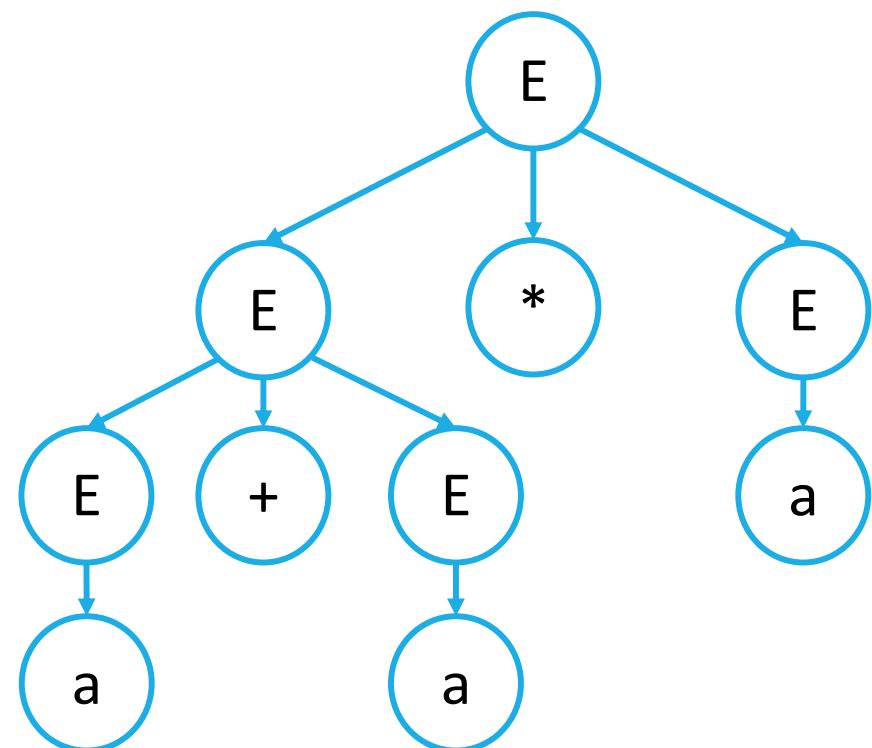
$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow a + E * E$

$\Rightarrow a + a * E \Rightarrow a + a * a$

$a + a * a$

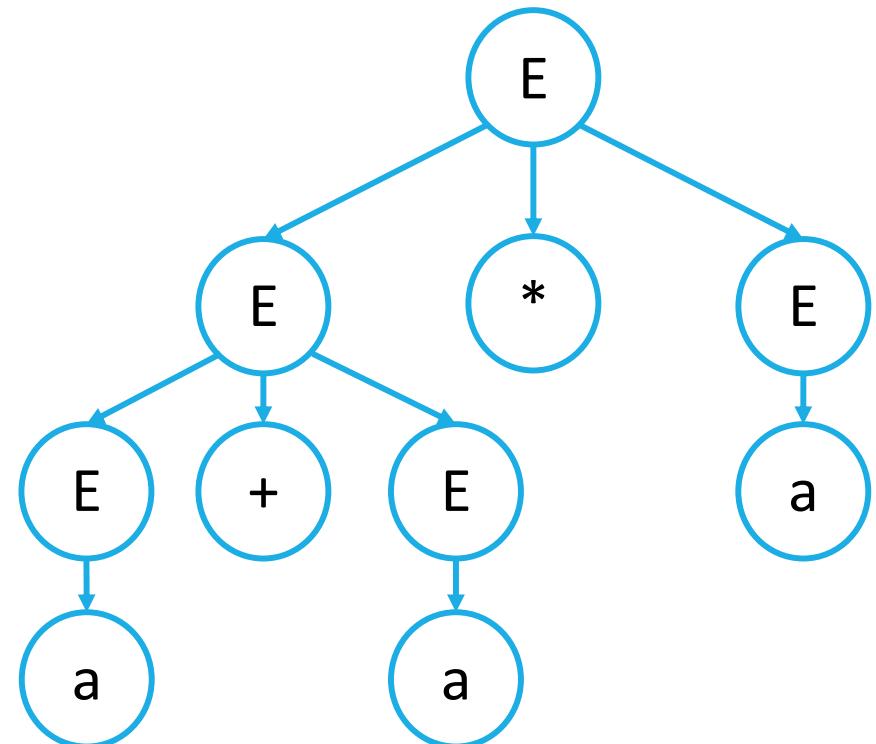
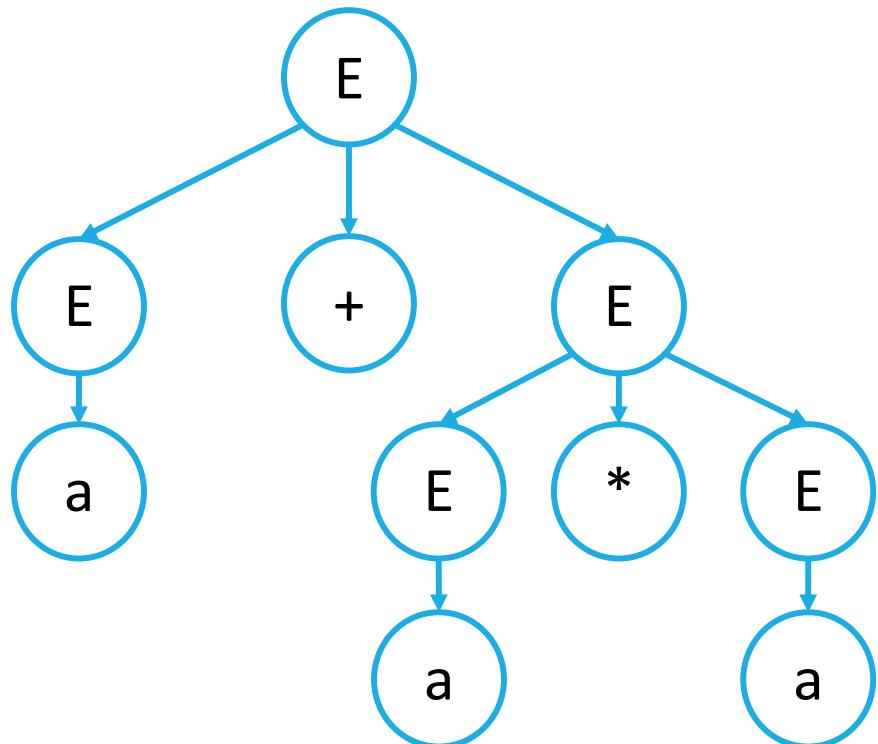
Egy másik legbaloldalibb levezetése



$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

$a + a * a$

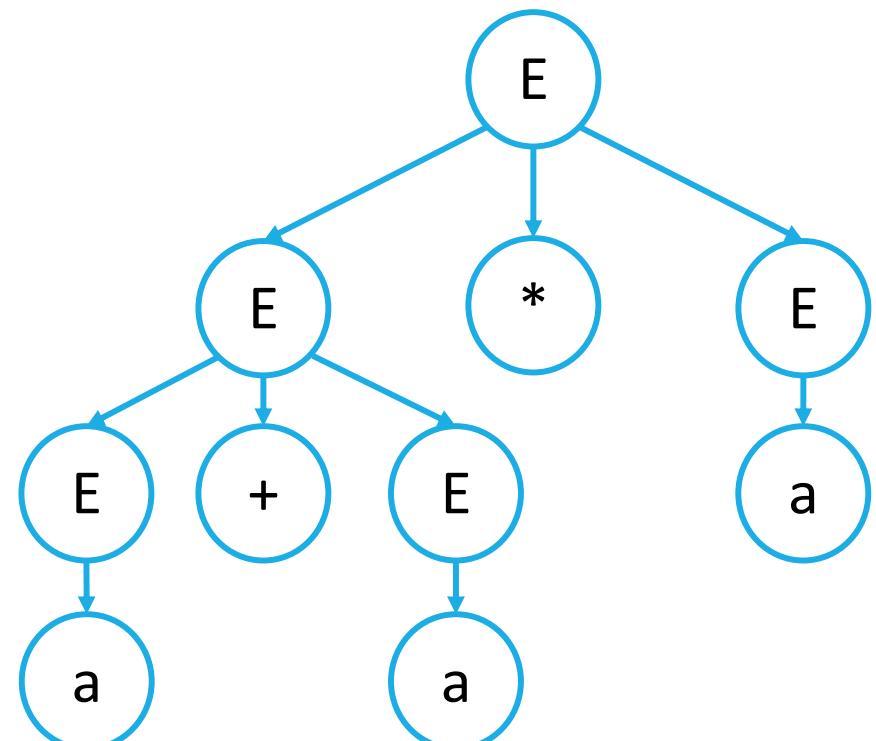
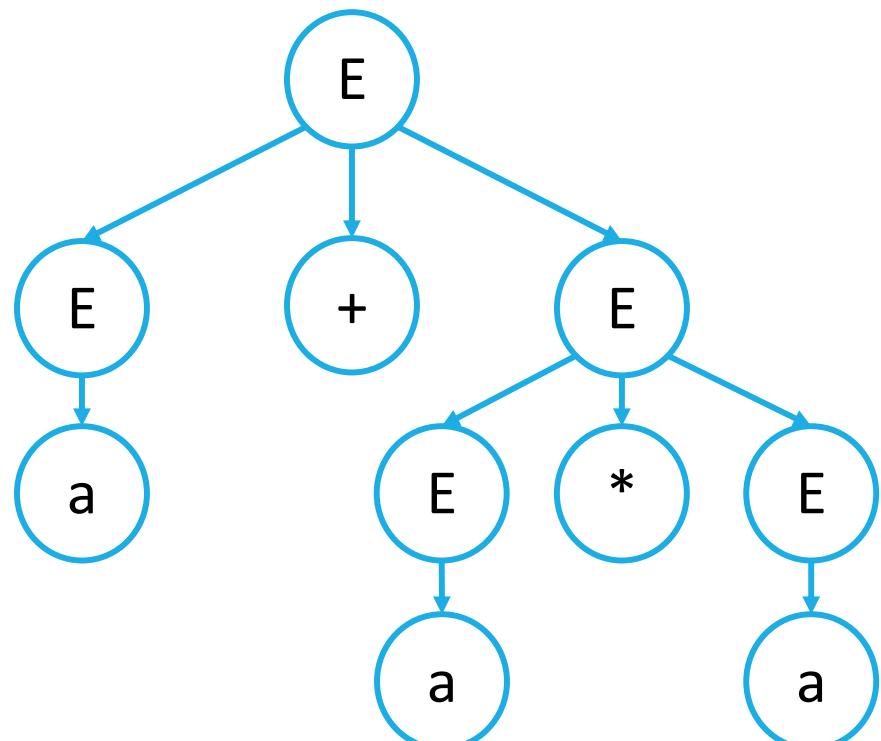
Két levezetési fa!



# Legyen $a = 2$

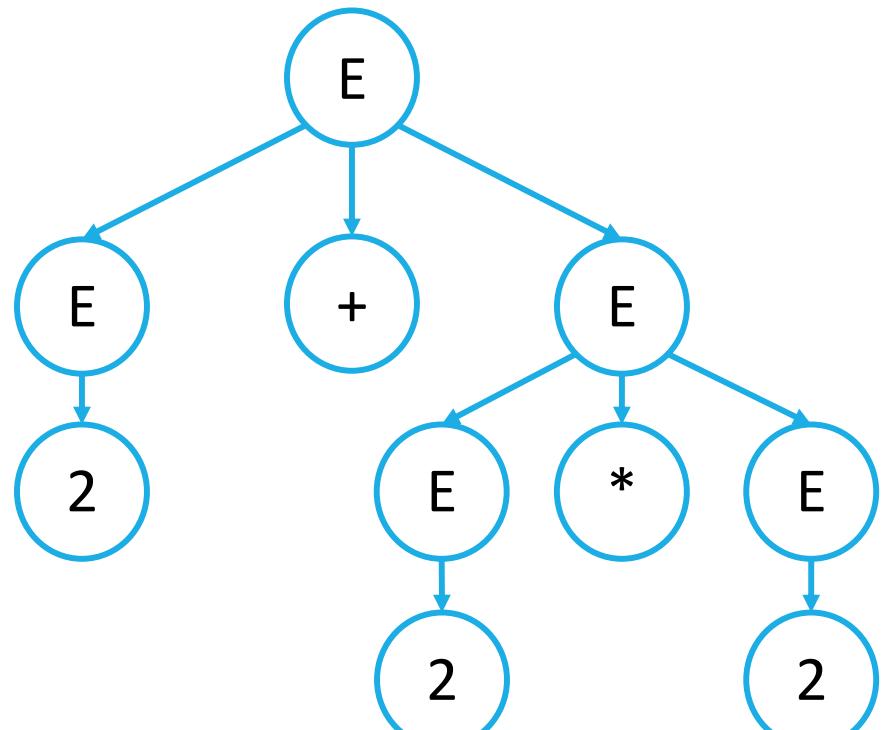
---

$$a + a * a = 2 + 2 * 2$$



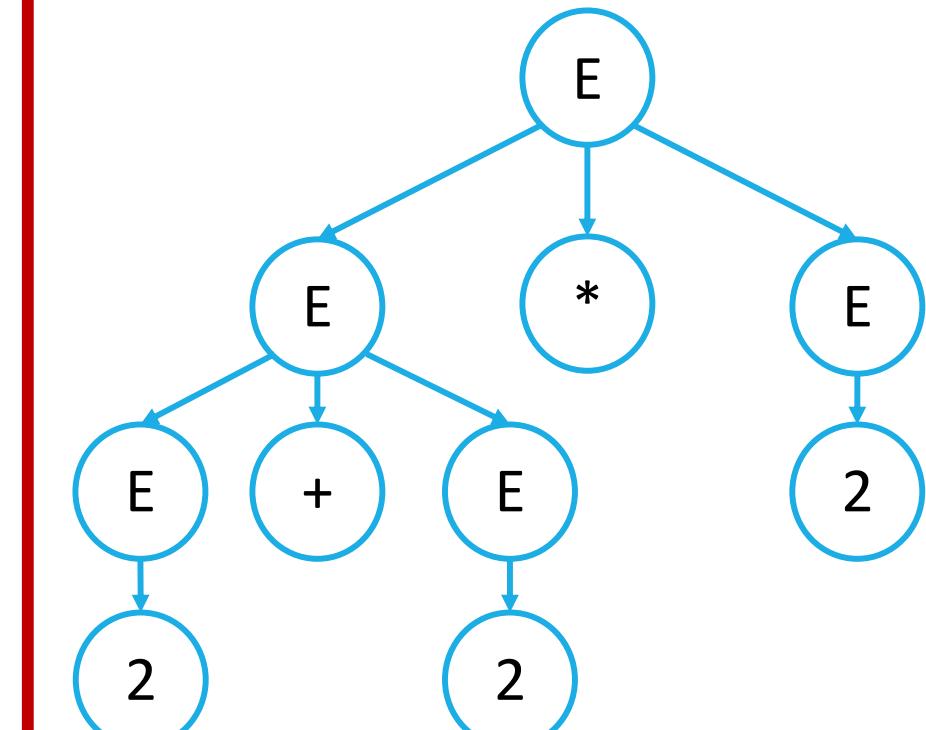
# Legyen a = 2

$$2 + 2 * 2 = 6$$



Számítsuk ki a kifejezés eredményét a fát használva

$$2 + 2 * 2 = 8$$



# Többértelmű grammatika

---

Egy  $G$  környezetfüggetlen grammatika **többértelmű**

ha van olyan  $w \in L(G)$  amelyiknek:

- két különböző levezetési fája van, vagy
- két különböző legbaloldalibb levezetése van

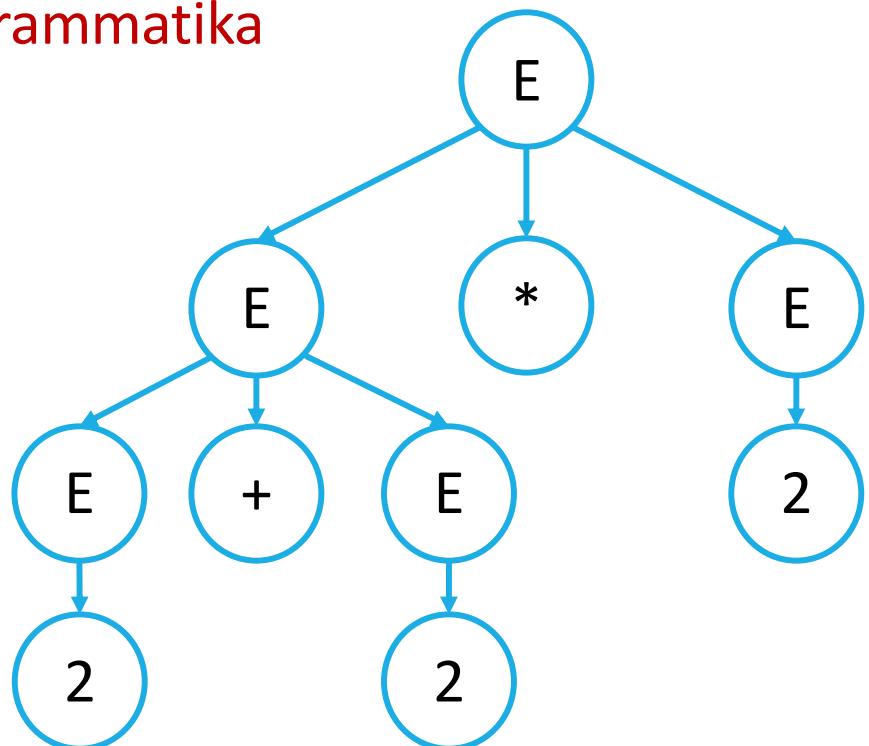
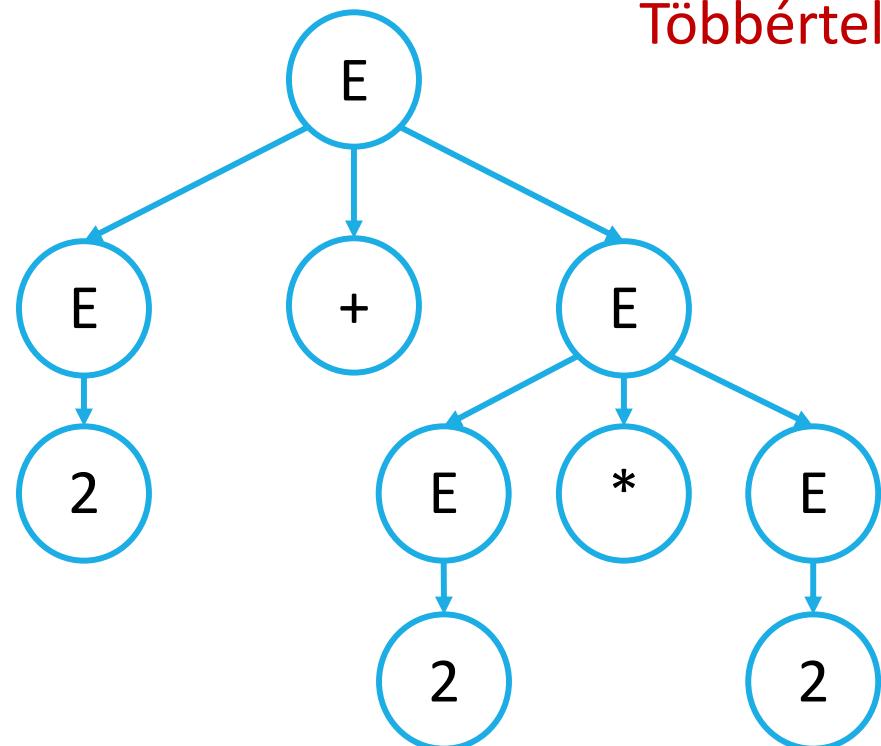
Két különböző levezetési fa problémákat okozhat

- kifejezések kiértékelésénél
- Általában - programozási nyelvek fordítóprogramjainál!

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

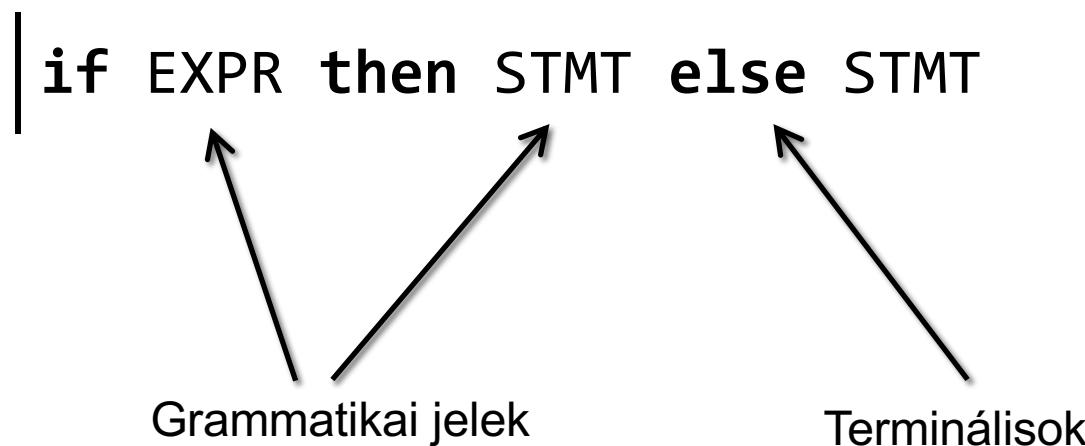
$a + a * a$

Többértelmű a grammatika



# Egy másik többérmű grammatika

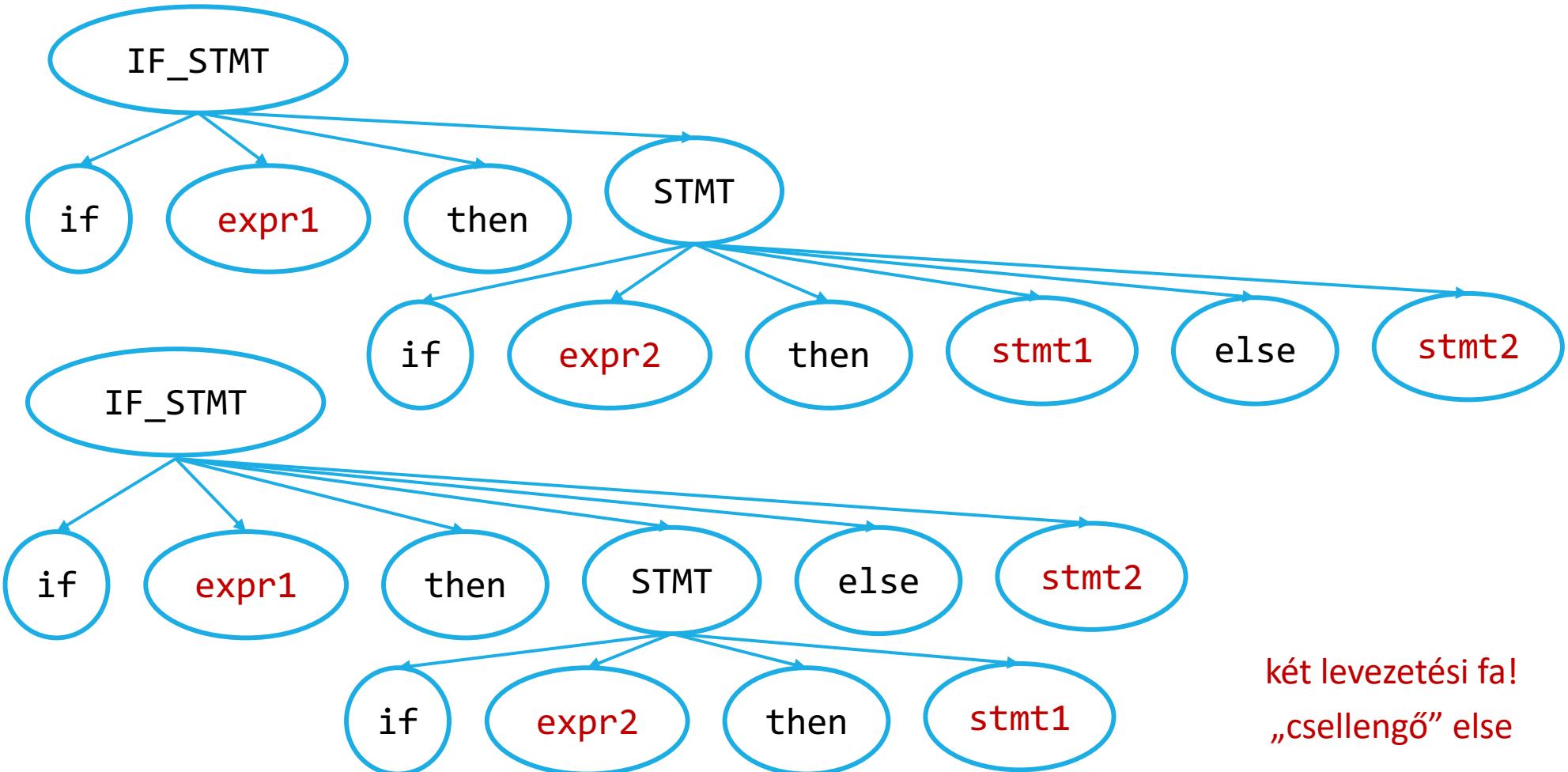
IF\_STMT → if EXPR then STMT



A programozási nyelvek grammaticájának jellemző része

# Egy másik többérmű grammatika

if expr1 then if expr2 then stmt1 else stmt2



# Többértelműség

---

Általában a többértelműség rossz, és szeretnénk megszüntetni...

Néha sikerül találni egyértelmű (nem többértelmű) grammatikát a nyelvhez

De általánosságban ezt nehéz elérni...

# Egy sikeres példa

Többértelmű grammatika:

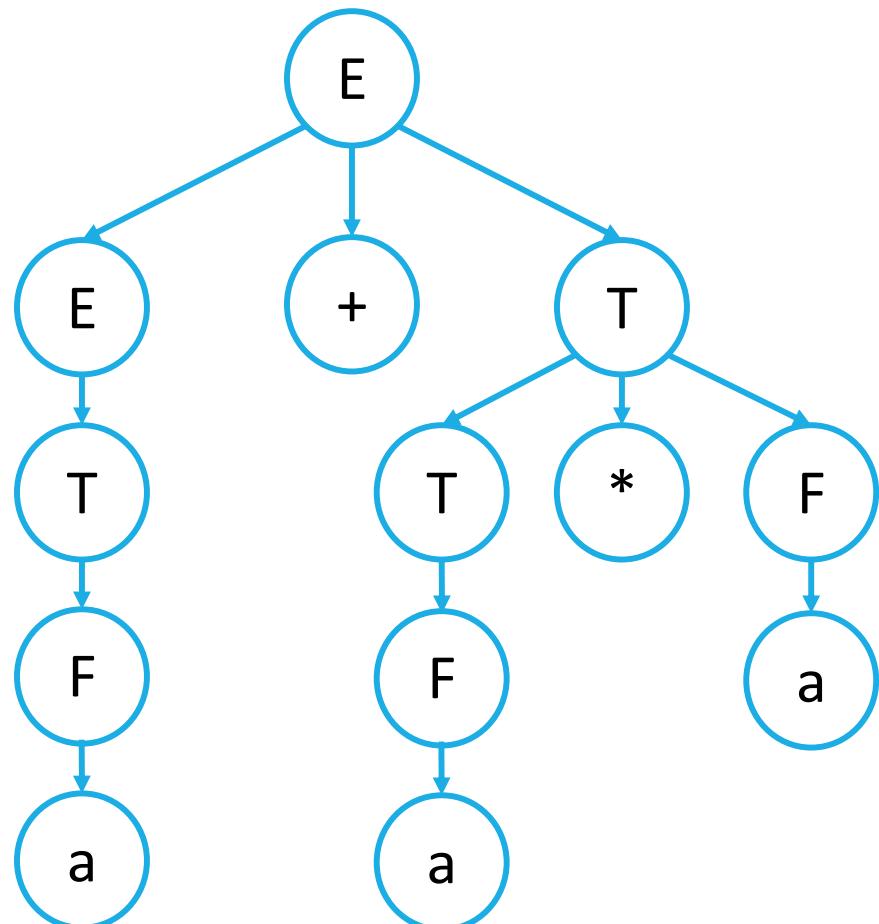
$$\begin{aligned}E &\rightarrow E + E \\E &\rightarrow E * E \\E &\rightarrow (E) \\E &\rightarrow a\end{aligned}$$

Ekvivalens

Nem-többértelmű grammatika

Ugyanazt a nyelvet generálja!

$$\begin{aligned}E &\rightarrow E + T \mid T \\T &\rightarrow T * F \mid F \\F &\rightarrow (E) \mid a\end{aligned}$$

$$\begin{aligned}
 E &\Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F \\
 &\Rightarrow a + F * F \Rightarrow a + a * F \Rightarrow a + a * a
 \end{aligned}$$


$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid a$

$a + a * a$

Egyértelmű a levezetési fa

# Programozási nyelv megadása

---

Formális szintaxis leírás – ezt használja a fordítóprogram

Formális szemantika leírás – a jelentés  
ellentmondásmentességének ellenőrzésére

# John Backus (1924-2007)

---



FORTRAN tervezője

"Munkám túlnyomó része egyszerűen a lustaságomból eredt. Amikor a rakéták röppályáját kiszámító programokon dolgoztam, hozzáfogtam egy programozási rendszer kidolgozásához, amelynek elsődleges feladata a munka megkönnyítése, leegyszerűsítése volt"

# Backus-Naur Form (BNF)

---

Nyelv a programozási nyelvek definiálására

- meta-nyelv, eredetileg 59-60-ból, az Algol és Fortran definiálására
- változatok ma is programozási nyelvek megadására
- <http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>

# BNF

---

ha egy szintaxis BNF formában adott, vannak eszközök, melyek automatikusan generálnak egy elemzőt, mely a szintaktikus helyességet ellenőrzi, és levezetési fát produkál, sőt le is fordítja a programot

[http://www.thefreecountry.com/developercity/constructcom\\_pilers.shtml](http://www.thefreecountry.com/developercity/constructcom_pilers.shtml)

Nézzük meg a Java BNF nyelvleírását!

# Fordítóprogram

Forrásprogram ⇒

Forrás-kezelő (source handler)



Lexikális elemző (scanner)



Szintaktikus elemző (parser)



Szemantikus elemző (semantic analyzer)



Belső reprezentáció



Kódgenerátor (code generator)

⇒ Tárgyprogram

# Fordítóprogram (folyt.)

---

3. Lexikális elemzés: megadható reguláris nyelvtannal
2. Szintaktikus elemzés: megadható környezetfüggetlen nyelvtannal
1. Szemantikus elemzés: környezetfüggő
  - pl. egy változó használatának helyessége függhet a deklarációjától, azaz a környezettől)

A három lépés szétválasztásának oka

- Egyszerű feladathoz ne használjunk bonyolult eszközöket.

# Programszerkezet

---

**Fordítási egység vagy modul:** A nyelvnek az az egysége, ami a fordítóprogram egyszeri lefuttatásával, a program többi részétől elkülönülten lefordítható

Több modulból álló programok esetében a fordítás és a végrehajtás között: a program **összeszerkesztése**

# Szerkesztés



# Programegység, fordítási egység

---

A **programegység** egy részfeladatot megoldó, tehát funkcionálisan összefüggő programrész.

A **fordítási egység** programegységek halmaza.

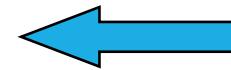
# A programegységek részei

---

A specifikációs rész írja le az egységnek a más egységből elérhető "kapcsolódási pontjait".

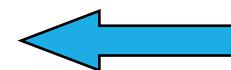
A törzs tartalmazza az egység funkcióját megvalósító programot.

```
procedure Hello;
```



specifikáció

```
with Text_IO; use Text_IO;  
procedure Hello is  
    S: String(1..40):=(others=>' ');  
    Last: Natural;  
  
begin  
    Put("Hogy hívnak?");  
    Get_Line(S, Last);  
    New_Line;  
    Put("Hello "&S(1..Last));  
  
end Hello;
```



törzs

# A törzs felépítése

---

A deklarációs rész (ha van külön) azonosítók **deklarációit**, valamint konstansok, típusok, objektumok és programegységek **definícióit** tartalmazhatja.

Az **utasítássorozat** tartalmazza azokat az utasításokat, amelyek **végrehajtása** nyomán a programegység kifejti a hatását.

# Deklarációk

---

A deklarációs rész feldolgozását, a definíciók értelmezését a deklarációs rész kiértékelésének mondjuk.

A deklarációs rész definícióinak kiértékelése történhet:

- Fordítási (pl, C, C++) vagy
- Futási (pl. PL/I, Ada) időben.

Az első esetben statikus definíció-kiértékelés,

A másodikban dinamikus definíció-kiértékelés.

# Deklarációk (folyt.)

---

Változódeklaráció – sokszor definíció is

- állapottér megadásához
- Formája általában
  - típus azonosító
  - int i;
- Vagy
  - azonosító: típus
  - N: Natural;

# Hatáskör, láthatóság

---

A programszövegnek azt a szakaszát, amelyen belül az adott deklaráció érvényben van, a deklaráció **hatáskörének** nevezzük.

A programszövegnek azt a szakaszát, ahol a deklarált azonosítóval az adott deklarációra hivatkozni lehet, a deklaráció **láthatósági körének** nevezzük.

# Blokkstruktúra

---

A **blokkstruktúra** a programegységek egymásba ágyazásával előálló hierarchikus struktúra.

Blokkstruktúrában egy programegység deklarációinak hatásköre kiterjed az összes **tartalmazott** programegységre is.

# Globális – lokális

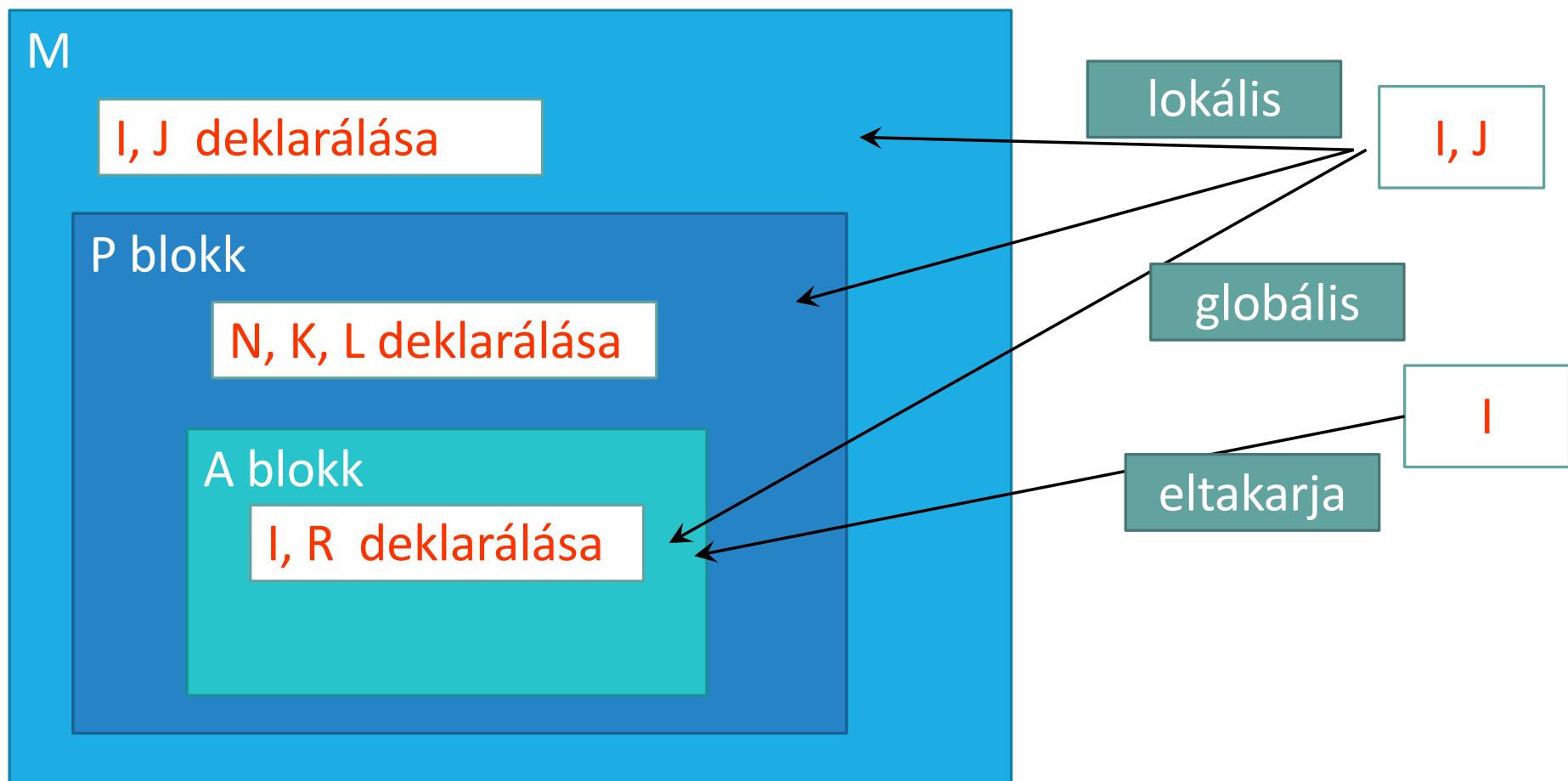
---

Egy programegységben a tartalmazó programegységek által deklarált azonosítókat (a tartalmazott programegység szemszögéből) **globális azonosítóknak** nevezzük.

A programegységben deklarált azonosítókat magában a programegységben **lokális azonosítóknak** nevezzük.

Blokkstruktúra esetén egy programegységben egy globális azonosító újra deklarálható, ekkor a lokális deklaráció **eltakarja** a globális deklarációt.

# Példa



# Példa

---

```
procedure M is
    I, J:Integer;
    procedure P is
        N, K, L:Integer;
        procedure A is
            I, R:Integer;
            begin
                I:=3; J:=9;          -- A-beli I, M-beli J
                N:=1;                 -- P-beli N
                M.I:=4;               -- minősített hivatkozás!
                ...
            end;
    begin
        I:=5;                  -- M-beli I
        A;                     --hívja A-t
        ...
    end;

begin
    I:=3;                  -- hívja P-t ...
    P;
end;
```

# Azonosító túlterhelése

---

Egy azonosítónak ugyanabban a szövegrészben több definíciója is érvényben lehet

A túlterhelést a leggyakrabban alprogramok nevei esetében teszik lehetővé – általában a paraméterek segítségével tudjuk megkülönböztetni

- `write(2)`
- `write("ez valami más")`
- `2 + 3`
- `2.1 + 3.0`

# Erősen típusos nyelv

---

Egy nyelv **erősen típusos**, ha minden érték, objektum, formális paraméter és függvény típusa **fordítási időben egyértelműen meghatározható**.

**Statikus típusosság**, ha a típusok ellenőrzése fordítási időben történik, míg a **dinamikus típusosság** esetén a típusellenőrzések (műveletekhez) futás időben történnek.

Az erősen típusos nyelv és a statikus típusosság összefonódása ebből következik.

# Memóriakezelés

---

A változóhoz a memóriaterület hozzárendelése (allokálása) történhet:

- automatikusan, a definíció kiértékelésekor, vagy
- a programozó rendelkezik róla

a változó által lefoglalt terület felszabadítása lehet:

- automatikus, vagy
- a programozó hatáskörébe tartozik.

# Élettartam

---

A változóhoz a szükséges memóriaterület lefoglalása (allokálása) és annak felszabadítása közötti időt a változó **élettartamának** nevezzük.

# Statikus változó

---

élettartama a program egész működése idejére kiterjed  
elhelyezése minden az ún. statikus (main) memória-  
területre történik

a statikus terület egyszer, a program betöltésekor kerül  
lefoglalásra

# Dinamikus változó

---

a program explicit módon foglal le területet, a **dinamikus** (heap) **memória**területen, amire a címével, úgynévezett **pointerrel** lehet hivatkozni.

egyes nyelvek tartalmaznak utasítást a dinamikus területek felszabadítására is.

blokkstruktúrában a programegységek (nem statikus) lokális változói az úgynévezett **végrehajtási veremben** helyezkednek el.

# Paraméterátadás

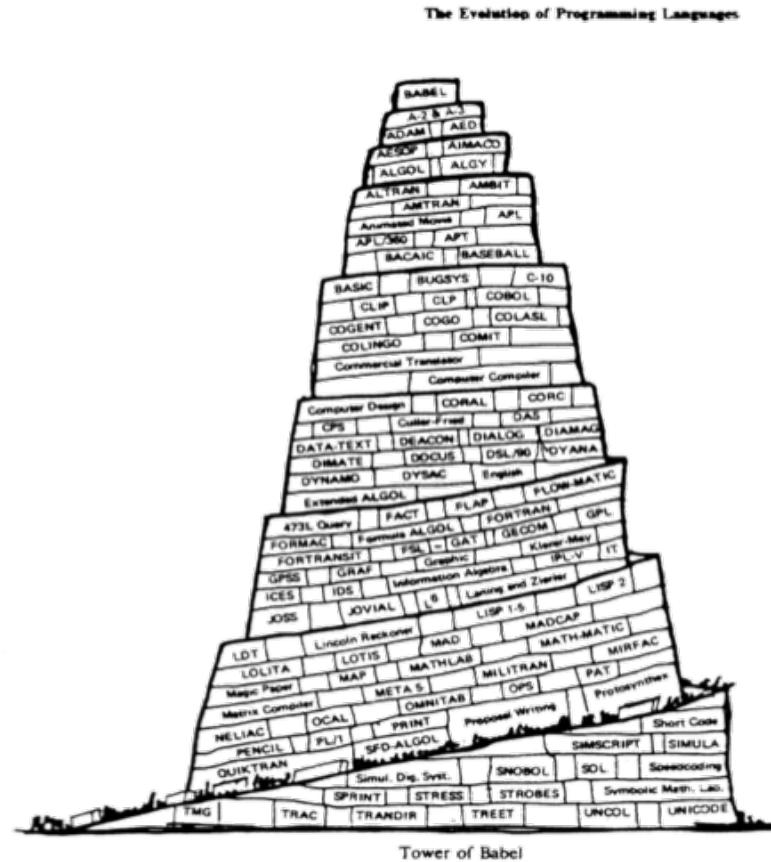
---

A programegységek legelterjedtebb fajtája az alprogram. A nyelvek kétféle alprogramot használnak, ezek az eljárások és a függvények.

Az alprogramoknál a hívóval való kommunikációt tervezéskor formális paraméterekkel írjuk le, amelyeknek az alprogram aktivizálásakor (hívásakor) aktuális paramétereket kell megfeleltetni.

Később ...

# A nyelvek „fejlődése”



Ma: több, mint 2500 nyelv –  
pedig már ez sem új...

# Mother Tongues

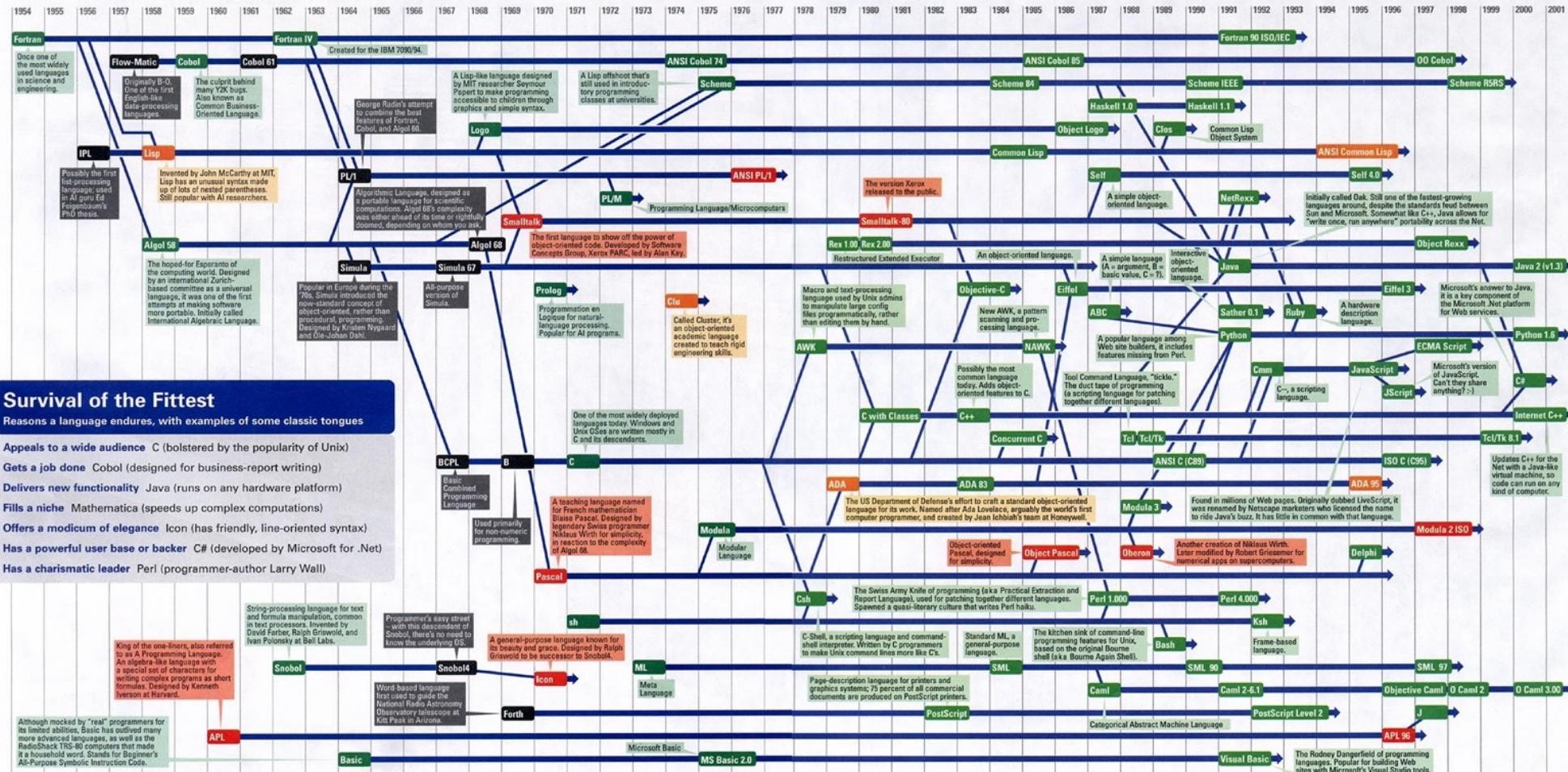
Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Snobol, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at [www.informatik.uni-freiburg.de/Java/misc/lang\\_list.html](http://www.informatik.uni-freiburg.de/Java/misc/lang_list.html). – Michael Menduno

Key	
	Year Introduced
Active:	thousands of users
Protected:	taught at universities; compilers available
Endangered:	usage dropping off
Extinct:	no known active users or up-to-date compilers
Lineage continues:	



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

# A programozási nyelvek rövid története

---

## FORTRAN (1950-es évek közepe-vége)

- hangsúly a tudományos programozáson

## LISP (1950-es évek vége)

- Szimbolikus listakezelés. Működés közösségi használja.
- van benne rekurzió, dinamikus helyfoglalás és felszabadítás (szemétgyűjtés).

## ALGOL 60 (1960)

- az adattípus fogalmának megjelenése
- blokkszerkezet
- érték- és név szerinti paraméterátadás

# A programozási nyelvek rövid története

---

## COBOL (1960)

- adatkezelés
- angol-szerű szintaxis
- hierarchikus adatszerkezetek megengedettek (rekordok rekordjai)

## BASIC (1960-as évek közepe)

### Beginner's All Purpose Symbolic Instruction Code

- érdekes tervezési cél: “a használó ideje fontosabb, mint a számítógép ideje”
- diákok programozás oktatására
- Interaktív interpretált környezet

# A programozási nyelvek rövid története

---

## PL/I (1960-as évek közepe)

- általános célúnak tervezték
- megengedi, hogy párhuzamosan végrehajtódó taszkokat hozzunk létre
- kezeli a futási idejű kivételeket
- pointerek mint adattípusok megjelenése
- tömbök részeire is hivatkozhatunk

## APL (1960 körül)

- tömb és mátrix kezelés

## SNOBOL (1960-as évek közepe)

- szövegkezelés
- műveletek szövegminták hasonlítására

# A programozási nyelvek rövid története

---

## SIMULA 67 (1967)

- szimulációs célok.
- osztály fogalmának bevezetése – az adatok és a rajta végezhető műveletek egységbezárása
- öröklődés megjelenése

## ALGOL 68 (1968)

- elsődleges tervezési cél: ortogonalitás.
- a legtöbb ezt követő programozás nyelv merített a tervezéséből

## Pascal (1970-es évek eleje)

- a programozás oktatására terveztek

# A programozási nyelvek rövid története

---

## C (1970-es évek eleje)

- rendszerprogramozásra
- rugalmas, de megbízhatatlan
- széleskörűen használják, részben a UNIX miatt

## Prolog (1970-es évek közepe)

- Logikai, nem-procedurális nyelv
- a programok állítások és szabályok halmazából állnak

# A programozási nyelvek rövid története

---

## Ada (1970-es évek vége)

- A Pentagon megbízásából
- csomagok az absztrakt adattípusok kezelésére
- Kivételkezelés
- Sablonok (Generic)
- Párhuzamosság támogatása (taszkok, randevúk)
- 2012-es szabvány: helyességellenőrzés

## Smalltalk (1980)

- Objektumorientált programozási nyelv
- nem csak egy nyelv, hanem egy grafikus környezet is

# A programozási nyelvek rövid története

---

## C++ (1980-as évek eleje óta)

- a SIMULA 67 és a Smalltalk sok objektumorientált jellemzőjét házasították össze a C-vel
- tervezési cél: ne csökkenjen a hatékonyság a C-hez viszonyítva
- Paraméterezett típusok (templates)
- Kivételkezelés
- Operátor túlterhelés

## Objective C (1980-as évek eleje)

- C és Smalltalk

## Eiffel (1980-as évek eleje)

- objektumorientált. egyszerűbb, mint a C++, nagyon jól átgondolt tervezés eredménye
- tervezés szerződéssel: állításokat használ egy szerződés kifejezésére az alprogramok és a hívóik között

# A programozási nyelvek rövid története

---

## Java (1990-es évek közepe)

- kisebb, egyszerűbb, megbízhatóbb C++
- minden kódot osztályokban kell megvalósítani
- párhuzamosság támogatása (threadek)
- szemétgyűjtés
- Kivételkezelés

## C# - a Microsoft „válasza” a Javára

- .NET alapú fejlesztés támogatása

Script nyelvek, párhuzamos nyelvek, ...



# Mi tesz egy programozási nyelvet fontossá?

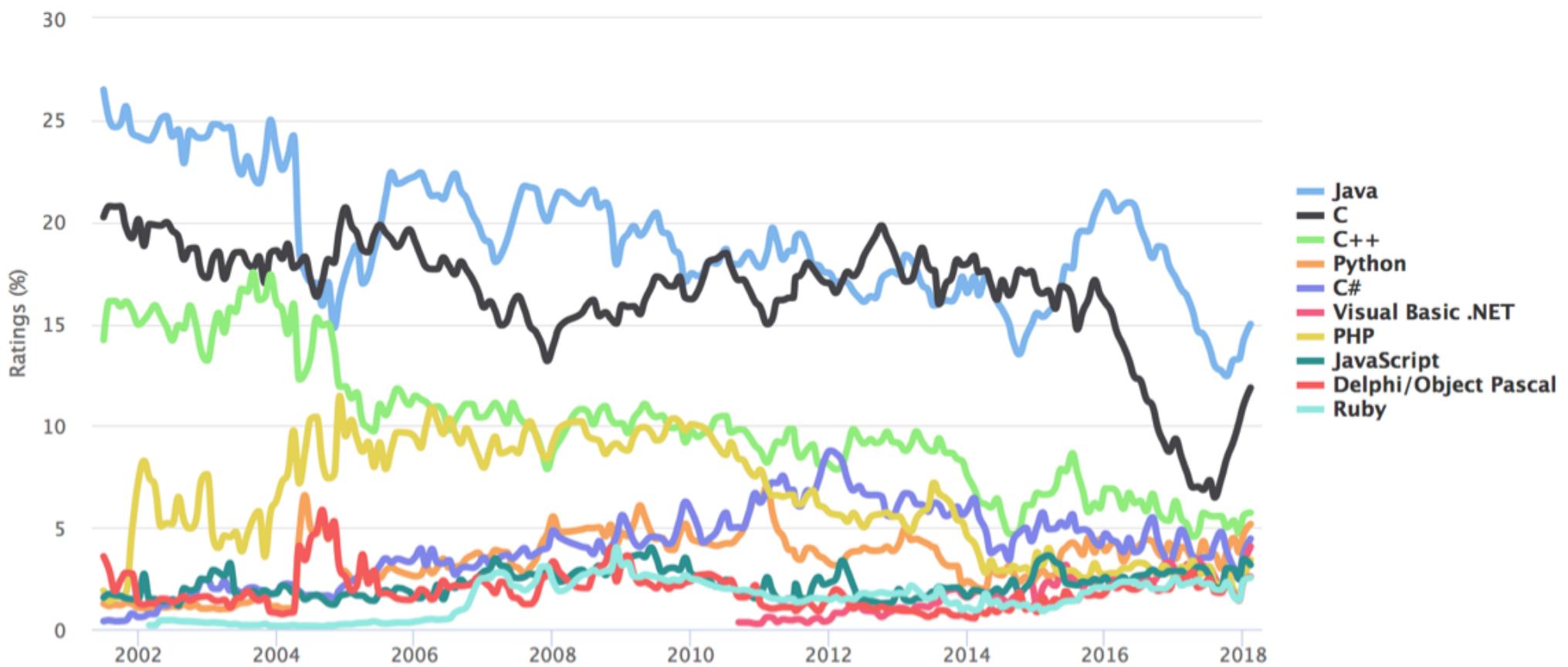
---



# TIOBE Index

TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



# Tiobe

2017	2016	2015	2014	2013	2012	2011	2010	2009	2008	Nyelv
1	1	2	2	1	2	1	1	1	1	Java
2	2	1	1	2	1	2	2	2	2	C
3	3	3	4	4	4	3	3	4	3	C++
4	4	5	5	5	5	4	6	7	8	C#
5	5	8	8	7	8	8	7	8	6	Python
6	7	9	10	12	15	39				Visual Basic .NET
7	9	6	9	11	11	10	11	9	9	JavaScript
8	11	12	13	9	9	9	9	6	7	Perl
9	13									Assembly
10	6	7	6	6	6	5	4	3	5	PHP

# Tiobe

2017	2016	2015	2014	2013	2012	2011	2010	2009	2008	Nyelv
11	10	11	20	15	12	12	12	11	10	Delphi / Object Pascal
12	11	20	12	10	10	11	10	10	11	Ruby
13										Go
14	16									SWIFT
15	12	10	7	8	7	7	5	5	4	(Visual) Basic
16	17	18	44							R
17										Dart
18	14	4	3	3	3	6	8	18	42	Objective-C
19	18	17	14	18	20	22				MATLAB
20	19	13	15	22	19	17	20	14	13	PL/SQL

# TIOBE hosszútávú összehasonlítás

Language	2018	2013	2008	2003	1998	1993	1988
Java	1	2	1	1	18	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	5
Python	4	7	6	12	26	16	-
C#	5	5	7	9	-	-	-
Visual Basic .NET	6	13	-	-	-	-	-
JavaScript	7	9	8	7	21	-	-
PHP	8	6	4	5	-	-	-
Perl	9	8	5	4	3	11	-
Ruby	10	10	9	19	-	-	-

# IEEE Rank

---

Language Rank	Types	Spectrum Ranking
1. C	  	100.0
2. Java	  	98.1
3. Python	 	98.0
4. C++	  	95.9
5. R		87.9
6. C#	  	86.7
7. PHP		82.8
8. JavaScript	 	82.2
9. Ruby	 	74.5
10. Go	 	71.9

# BlackDuck – opensource project statistics

