MODELLALAPÚ SZOFTVERFEJLESZTÉS

II. ELŐADÁS

SZÖVEGES NYELVEK

DR. SOMOGYI FERENC

A MAI ELŐADÁS

I. Szöveges nyelvek

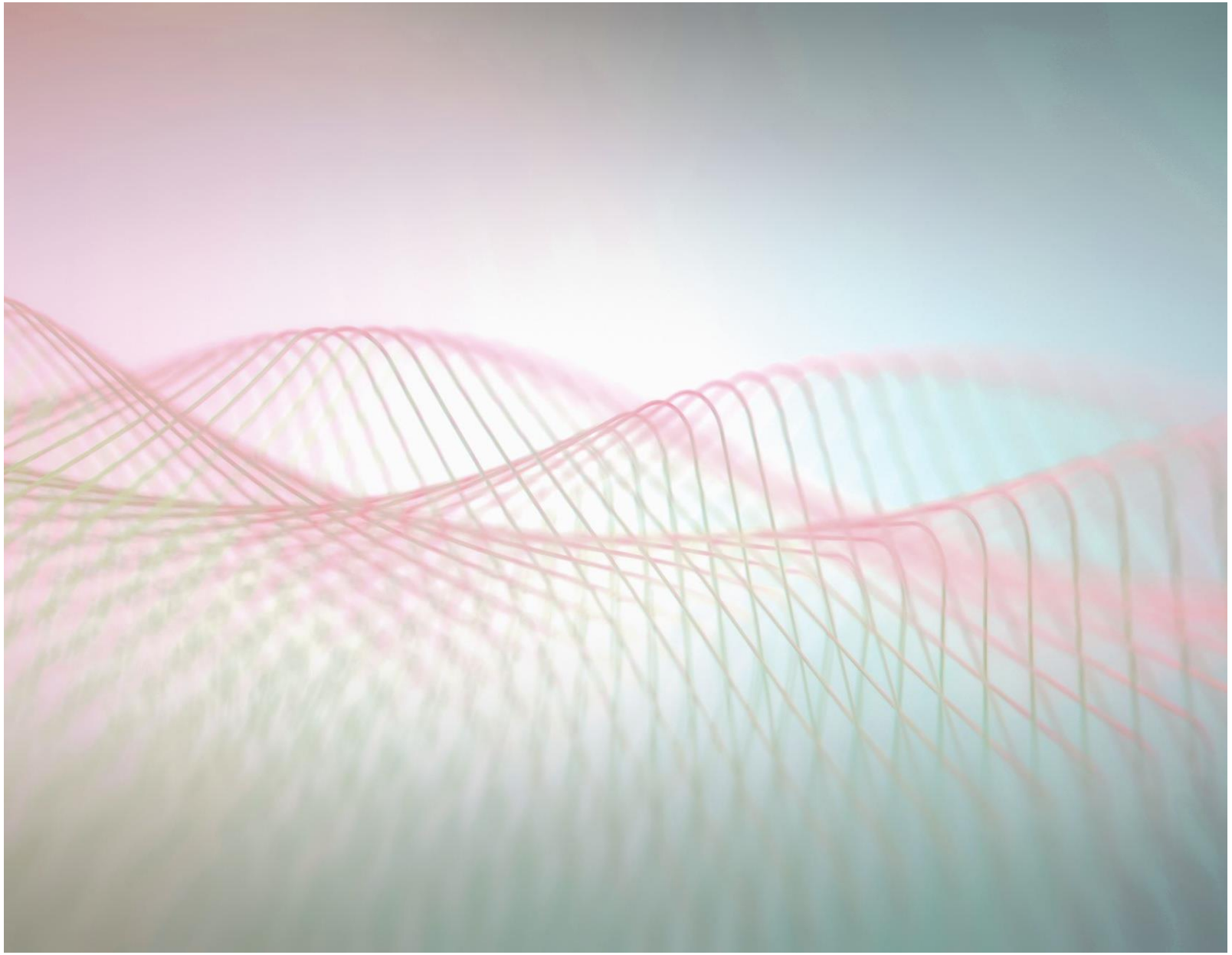
II. Fordítókról általában

III. Lexikai elemzés

IV. Reguláris kifejezések

V. Formális nyelvek

VI. Szintaktikai elemzés (bevezető)



MILYEN NYELVEKET ISMERTEK?

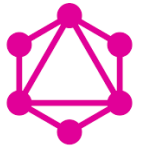
Nem csak programozási nyelveket!

SZÖVEGES NYELVEK

- Természetes nyelvek
  - Nem foglalkozunk velük a tárgy keretein belül ☺
  - Pl. magyar, angol, kínai
- Programozási nyelvek
  - Pl. C, C++, C#, Java, Kotlin, Python, Rust, Logo
- Webes világban használt nyelvek
  - Pl. HTML, CSS, JavaScript, TypeScript

SZÖVEGES NYELVEK

- Adatbázis kezelő nyelvek
  - Data Definition / Manipulation Language, stb.
  - Pl. SQL, GraphQL
- Adatstruktúra leíró nyelvek
  - Pl. XML, XAML, JSON
- Dokumentumleíró nyelvek
  - Pl. LaTeX, Markdown
- Hardverleíró nyelvek
  - Pl. Verilog, VHDL

SZÖVEGES NYELVEK

- Általános célú nyelvek
  - Általában programozási nyelvek, de nem feltétlenül
  - Melyek tekinthetők általános célú nyelvnek az előzőekből?
- Szakterületi nyelv
  - Egy adott szakterület fogalmait írja le
  - Melyek tekinthetők szakterületi nyelvnek az előzőekből?

SZAKTERÜLETI NYELVEK VS. ÁLTALÁNOS CÉLÚ NYELVEK

Szakterületi nyelvek	Általános célú nyelvek
Adott szakterület fogalmait használja (pl. bicikli, HTML input form)	Általános fogalmakat használ (pl. osztály, függvény, XML tag)
Szakértők számára készül	Programozók számára készül
Speciálisabb célok	Általánosabb célok
Szabadabb szintaxis	Kötöttebb szintaxis
Egyedi feldolgozás és környezet	Támogatott fejlesztőkörnyezet

- Szakterületi nyelv = Domain-Specific Language (DSL)
- Általános célú nyelv = General-Purpose (Programming) Language (GPL)
  - Léteznek nem programozási, általános célú nyelvek is – Id. XML, JSON

SZAKTERÜLETI NYELVEK A NYELV JELLEGE SZERINT

Internal DSL

- Általános célú programozási nyelv speciális módon használva
- A nyelvi elemek közül csak néhányat használunk
- Feldolgozás az eredeti nyelven
- Pl. script nyelvek, saját framework hívások

External DSL

- Saját nyelv, nem az adott alkalmazás programozási nyelve
- Egyedi szintaxis (vagy egy másik nyelv szintaxisa)
- Feldolgozás egy másik nyelven
- Pl. Unix parancsok, SQL, HTML, CSS

A MAI ELŐADÁS

- I. Szöveges nyelvek
- II. Fordítókról általában
- III. Lexikai elemzés
- IV. Reguláris kifejezések
- V. Formális nyelvek
- VI. Szintaktikai elemzés (bevezető)

INTERPRETER VS. COMPILER

Interpreter

- Memóriában fut, virtuális gép
- Utasításonként hajtja végre a kódot
- Gyorsan indul, lassan fut
- Csak az első hibáig fut jellemzően
- Debug könnyű
- Pl. Perl, Python, DOS, UNIX shell

Compiler

- Általában gépi kódot generál
- Az egész kódot egyben dolgozza fel
- Lassan indul, gyorsan fut
- A végén jelzi a hibákat
- Debug nehéz (instrumentált kódgenerálás segíthet)
- Pl. C, C++, C#, Kotlin, Pascal

INTERPRETER VS. COMPILER

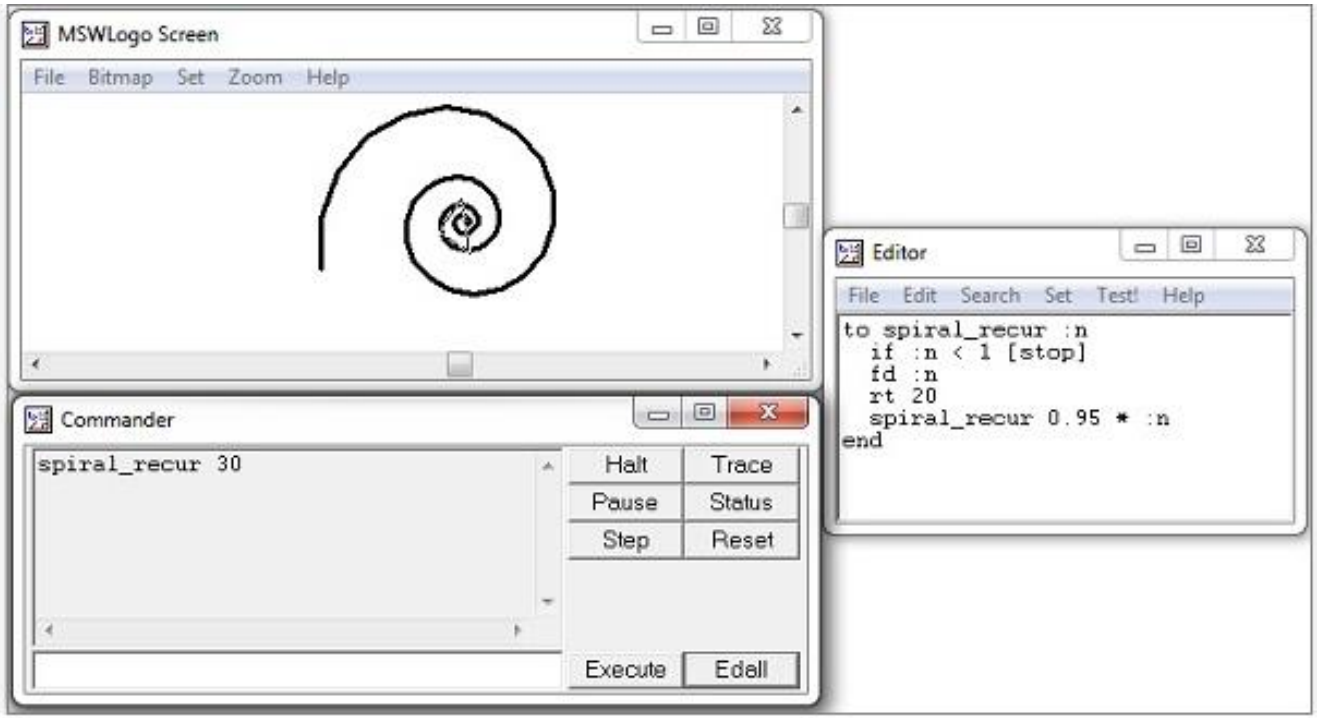
- Különleges esetek
  - Fordított (compiled) és interpretált (interpreted) nyelv egyszerre
    - Pl. Java – Just-In-Time (JIT) compiler
  - Választható, hogy fordított vagy interpretált a nyelv
    - Pl. Erlang, Prolog, SQL, Logo
- Vannak nyelvek, amelyek nem fordítottak és nem interpretáltak
  - Tipikusan markup language-ek
  - Pl. XML, JSON, HTML, XAML, UML

INTERPRETER VS. COMPILER

- Just-In-Time (JIT) compiler
  - Nem végrehajtás előtt, hanem végrehajtás közben végzi a fordítást
  - A forráskódból előállított köztes kód továbbfordítása – language interoperability
  - Gyorsabb, mint az interpretálás, de lassabb, mint a teljes fordítás
  - Kisebb memóriaigény, dinamikus futásidejű viselkedést is figyelembe tudja venni
  - Pl. Java JVM, .NET CLR
- Transpiler
  - Nem gépi kódra fordít, hanem egy másik magasabb szintű nyelvre
  - Pl. TypeScript → JavaScript, Python2 → Python3 fordítók

INTERPRETER VS. COMPILER – PÉLDA

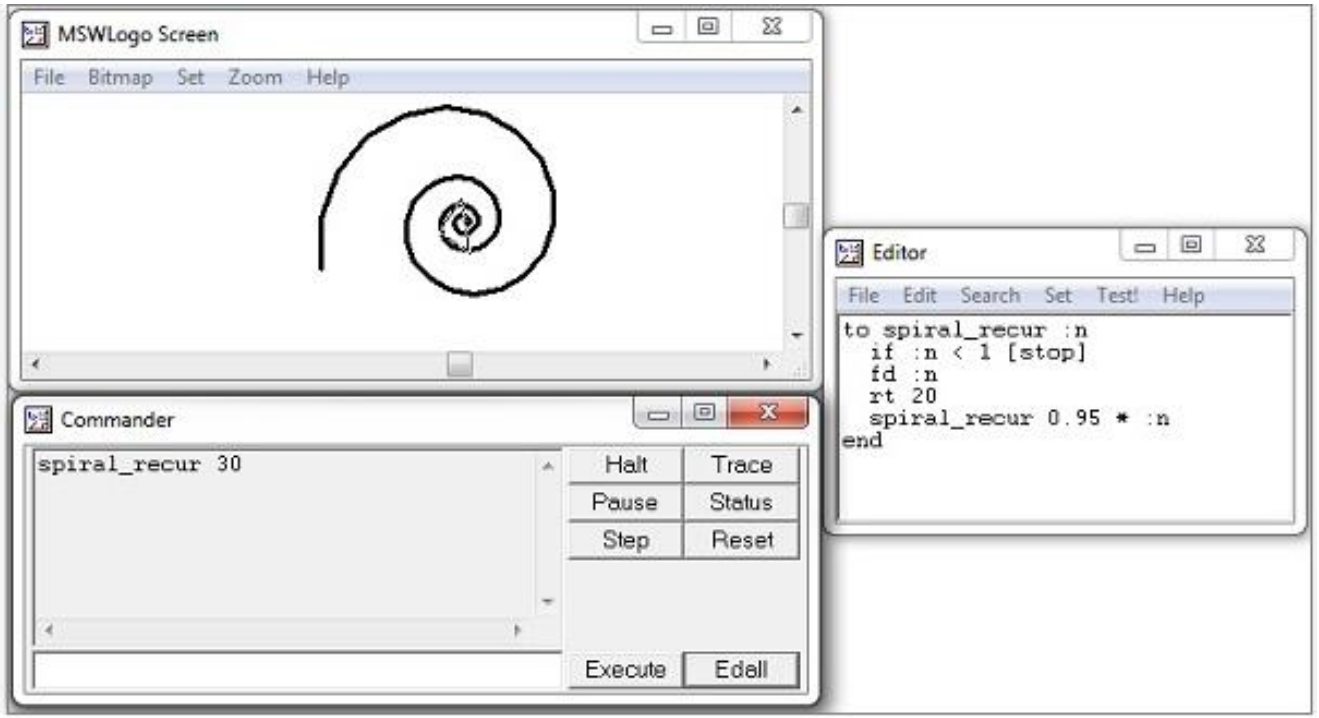
- MSWLogo
  - Programozás oktatása
  - Elemi utasítások
    - Előre / hátra megy
    - Balra / jobbra fordul
    - Tollat felemel / lerak
- Programozási koncepciók
  - Feltételes elágazás, függvény, függvény paraméter, stb.



Forrás: https://www.tutorialspoint.com/logo/logo_quick_guide.htm

INTERPRETER VS. COMPILER – PÉLDA

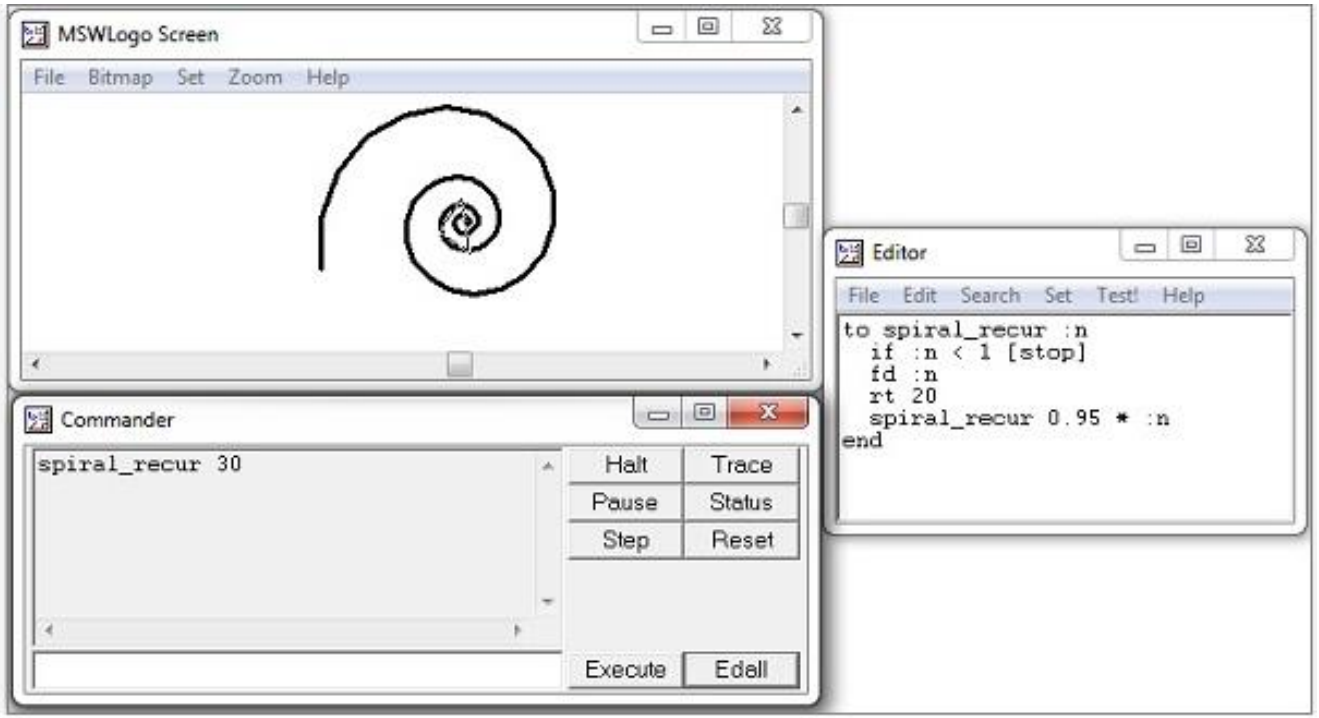
- Fordított Logo
  - Beolvassuk az egész programot
  - Tetszőleges programnyelven
    - Pl. Java (de bármi más lehet)
    - Saját fordítót írunk (kézzel vagy parser generator használatával)
  - Gépi kódra fordítunk
    - Vagy más nyelvre (ld. transpiler)
  - A fordított kódot futtatjuk



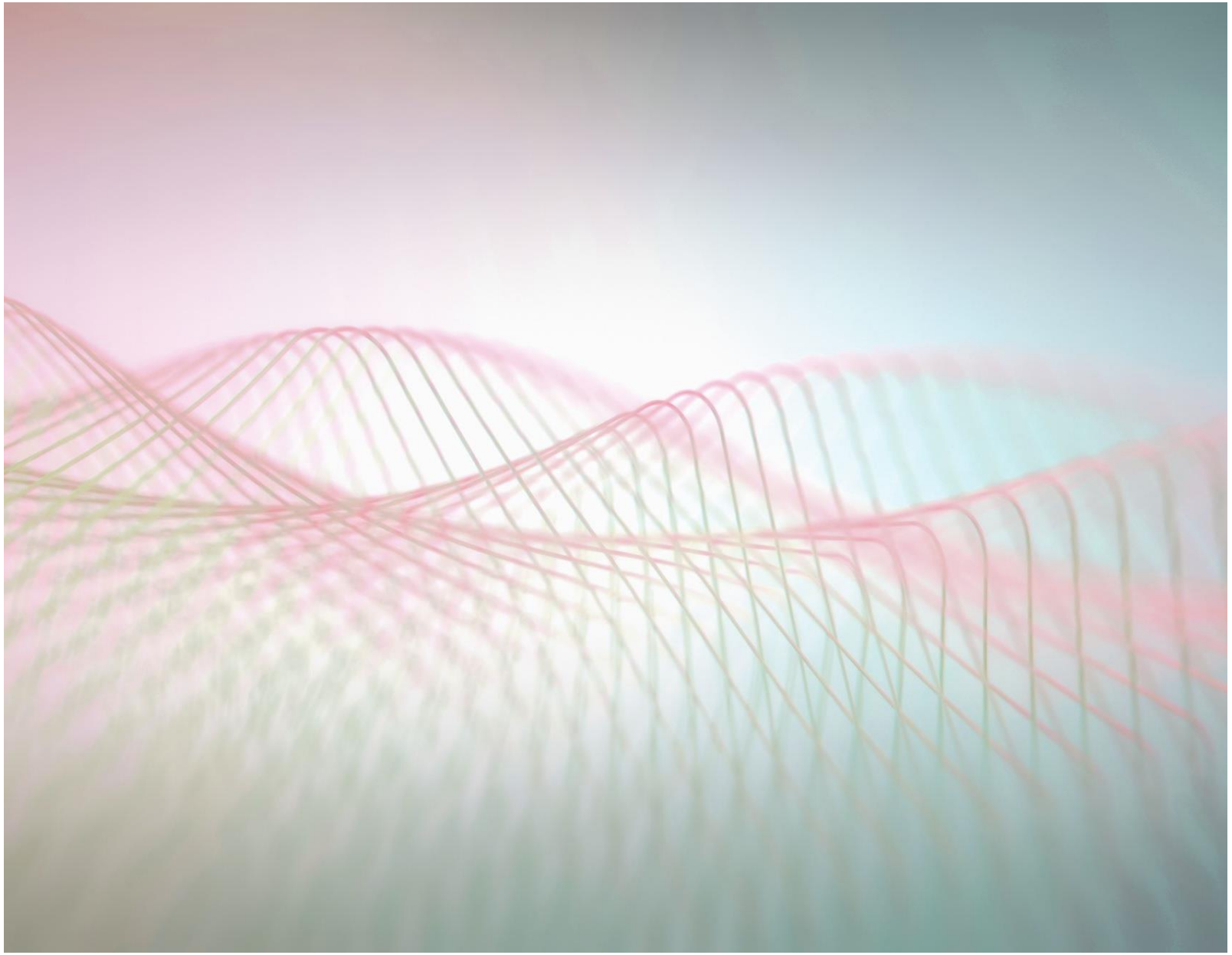
Forrás: https://www.tutorialspoint.com/logo/logo_quick_guide.htm

INTERPRETER VS. COMPILER – PÉLDA

- Interpretált Logo
  - Utasításonként dolgozzuk fel
  - Tetszőleges programnyelven
    - Pl. Java (de bármi más lehet)
    - Saját interpretert írunk
  - Végrehajtjuk az utasításokat
    - “fd :n” → Java Canvas API hívás

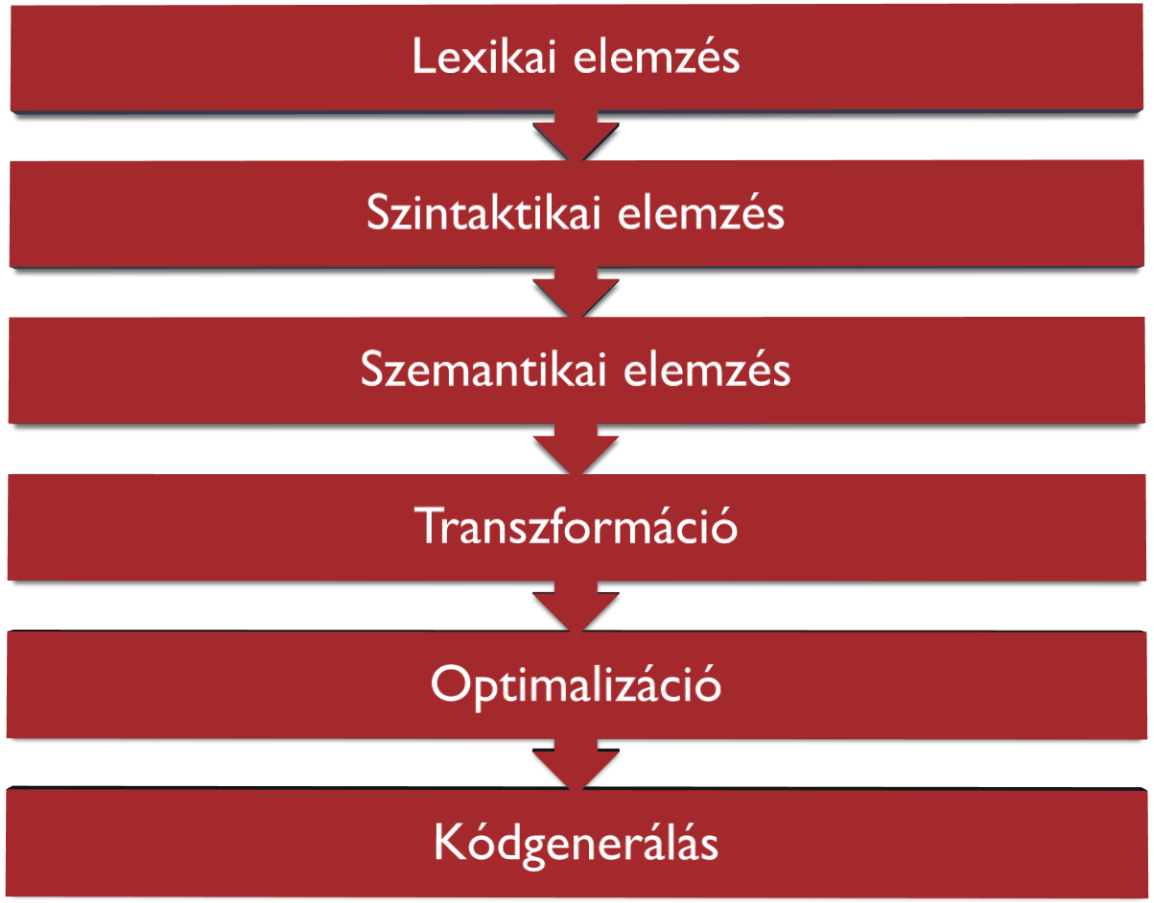


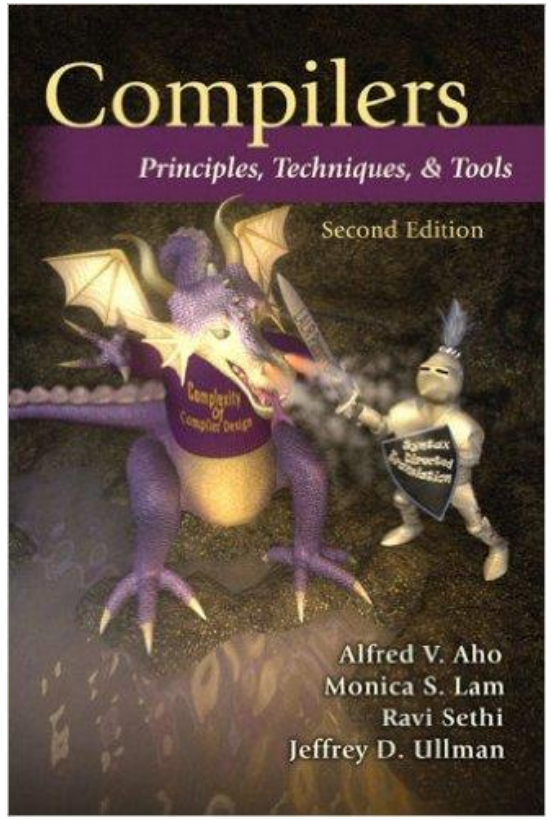
Forrás: https://www.tutorialspoint.com/logo/logo_quick_guide.htm



A FORDÍTÁS KLASSZIKUS FÁZISAI

A FORDÍTÁS KLASSZIKUS FÁZISAI





0. FÁZIS – FORRÁSKÓD

```
while (y < z) {
    x = a + b;
    y += x;
}
```

I. FÁZIS – LEXIKAI ELEMZÉS

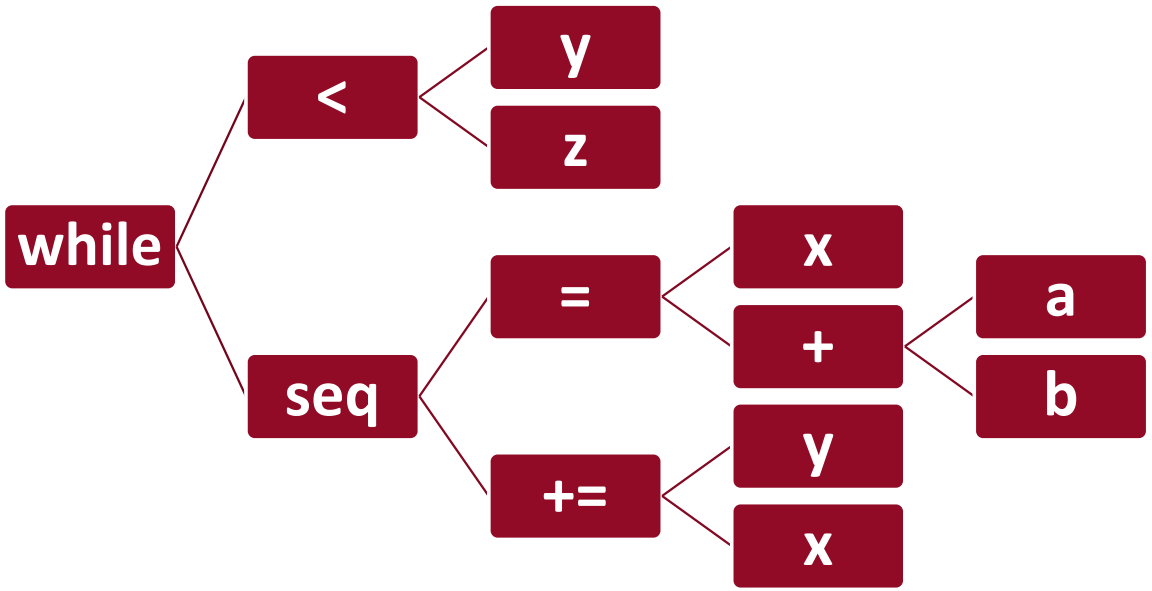
```
T_While
T_LeftParen
T_Identifier y
T_Less
T_Identifier z
T_RightParen
T_OpenBrace
T_Identifier x
T_Assign
…
```


```
while (y < z) {
  x = a + b;
  y += x;
}
```

1. FÁZIS – LEXIKAI ELEMZÉS

- Lexer végzi
- Forráskód feldarabolása elemi egységekre
  - Elemi egység = token
  - Ezek lesznek az alap építőköveink később
- Felesleges karakterek elhagyása
  - Nem mindig történik meg
  - Pl. kommentek, whitespace, stb.

II. FÁZIS – SZINTAKTIKAI ELEMZÉS

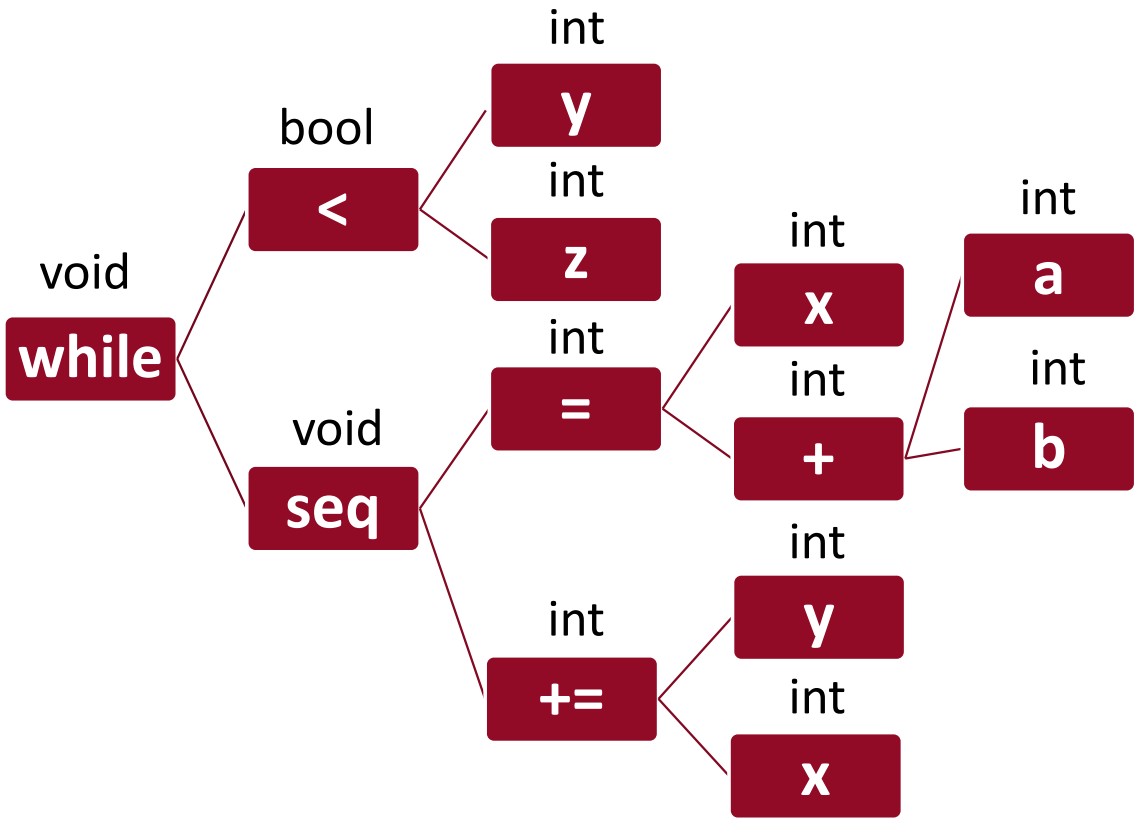



```
while (y < z) {
  x = a + b;
  y += x;
}
```

II. FÁZIS – SZINTAKTIKAI ELEMZÉS

- Parser végzi
  - A parser elnevezést néha egyben értik a lexerrel
- Olyan struktúra előállítása a kódból…
- …ami feldolgozható a későbbi fázisok által
- Tipikusan valamilyen fa struktúrát állít elő
  - Szintaxisfa

III. FÁZIS – SZEMANTIKAI ELEMZÉS

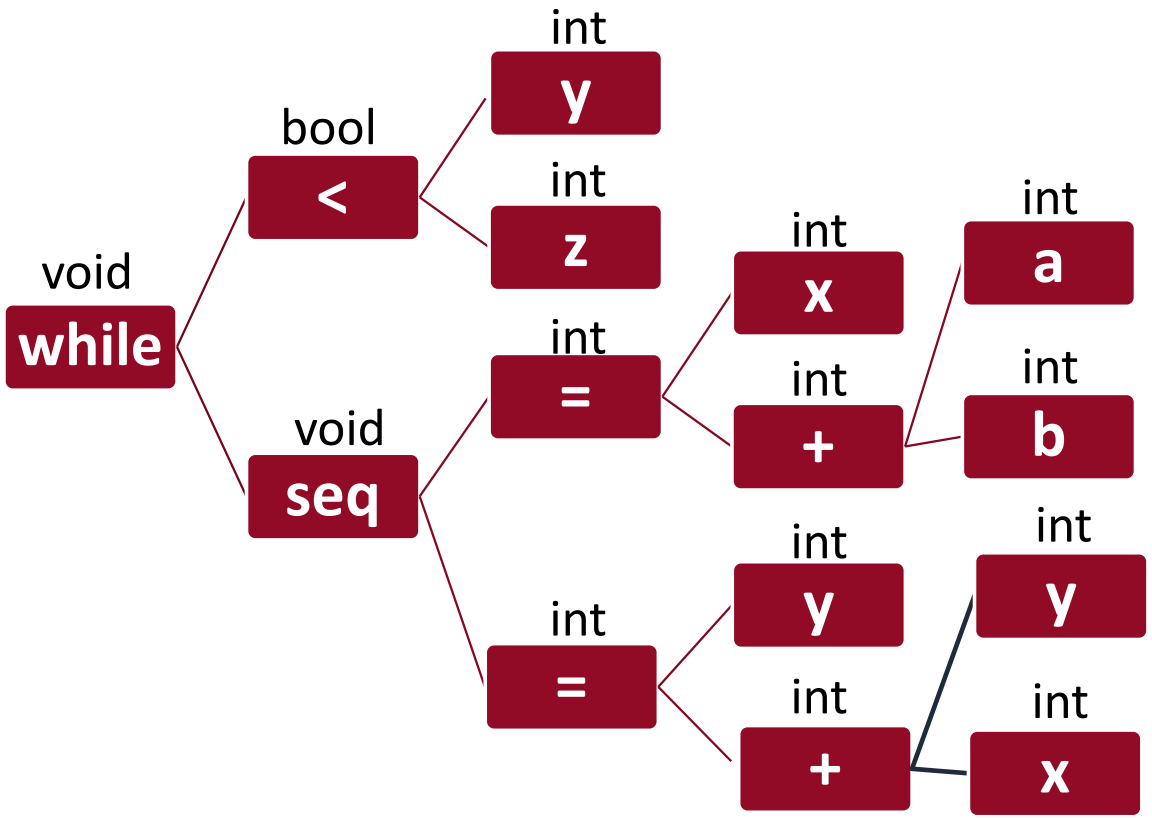



```
while (y < z) {
  x = a + b;
  y += x;
}
```

III. FÁZIS – SZEMANTIKAI ELEMZÉS

- Jelentés hozzátársítása a struktúrához
  - Pl. típusinformációk, változók
- Konzisztenciaellenőrzés
  - Parszolás közben nem megállapítható feltételek…
  - … ha látjuk a teljes struktúrát, akkor már igen
  - Pl. interfész függvényeinek implementálása, változó típusa, statikus tömbök indexelése, típushelyes kifejezések

IV. FÁZIS – TRANSZFORMÁCIÓ

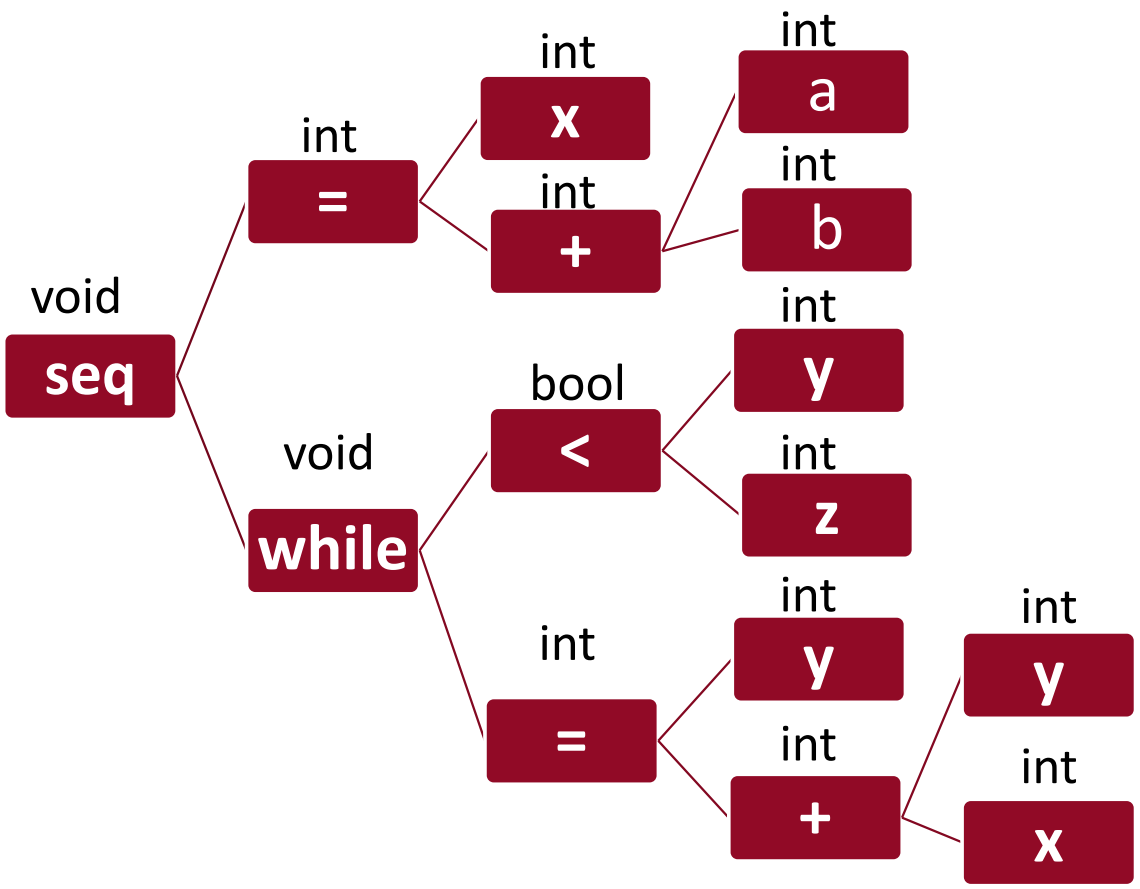



```
while (y < z) {
  x = a + b;
  y += x;
}
```

IV. FÁZIS – TRANSZFORMÁCIÓ

- Opcionális
- Köztes nyelvre való leképzés
  - Könnyebben kezelhető pl. optimalizáció során
  - Pl. Common Intermediate Language,
    Java Bytecode,
    vagy ezek előtti köztes reprezentációk
- Műveletek leképzése egységes formára
  - Pl. "*y* += *x*" → "*y* = *y* + *x*"

V. FÁZIS – OPTIMALIZÁCIÓ



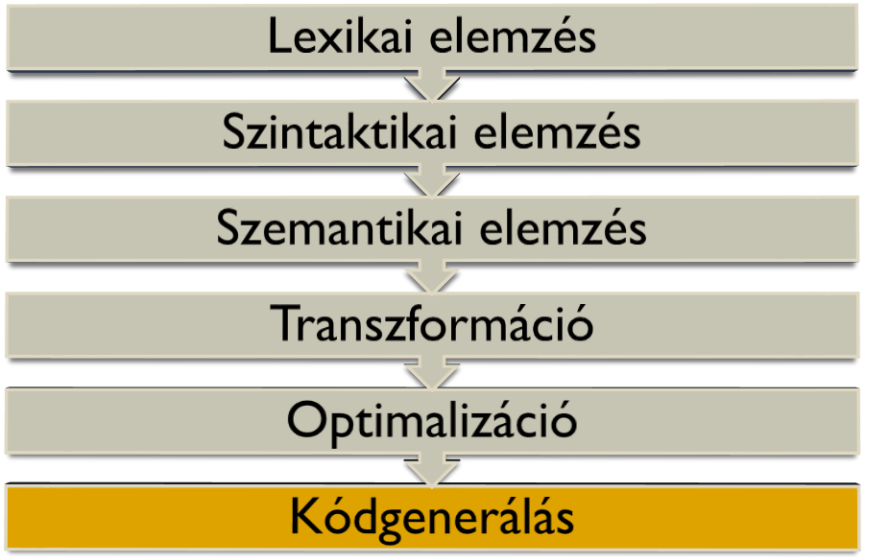

```
while (y < z) {
  x = a + b;
  y += x;
}
```

V. FÁZIS – OPTIMALIZÁCIÓ

- Opcionális
  - Általában a transzformáció eredményén végzendő
- Minél gyorsabb / kevesebb erőforrást használ
  - A kód helyességének megtartásával!
- Nincs optimális kód, csak optimalizált
  - Egymásnak ellentmondó optimalizációs technikák lehetnek

VI. FÁZIS – KÓDGENERÁLÁS

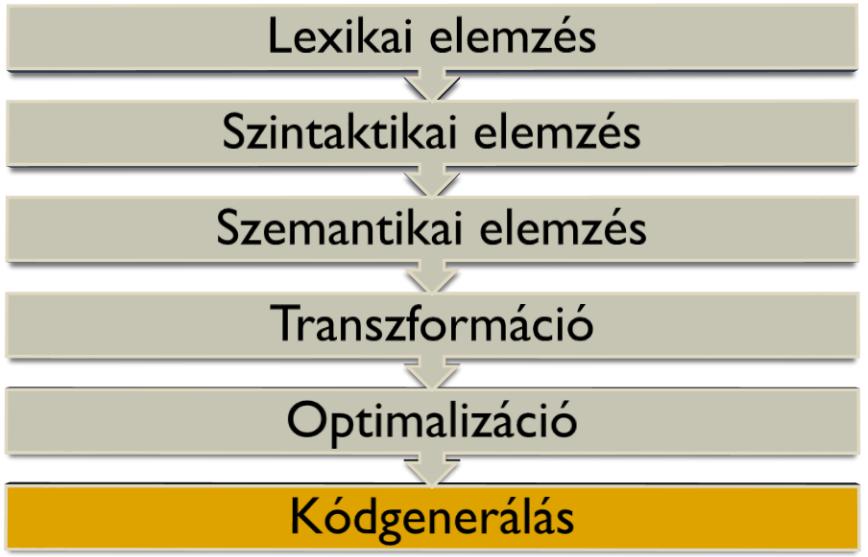
```
add   $1, $2, $3
loop: add $4, $1, $4
slt   $6, $1, $5
beq   $6, loop
-----------------------------------
x := a + b;
while y < z do
       y := y + x;
```




```
while (y < z) {
  x = a + b;
  y += x;
}
```

VI. FÁZIS – KÓDGENERÁLÁS

- Kód előállítása
  - Parancsok és végrehajtási sorrend kiválasztása
  - Nem mindig gépi kódra fordítunk (ld. transpilerek)
- Assembler
  - Gépi kódra fordításért felelős
- Linker
  - Lefordított modulok (amennyiben vannak) összekötése



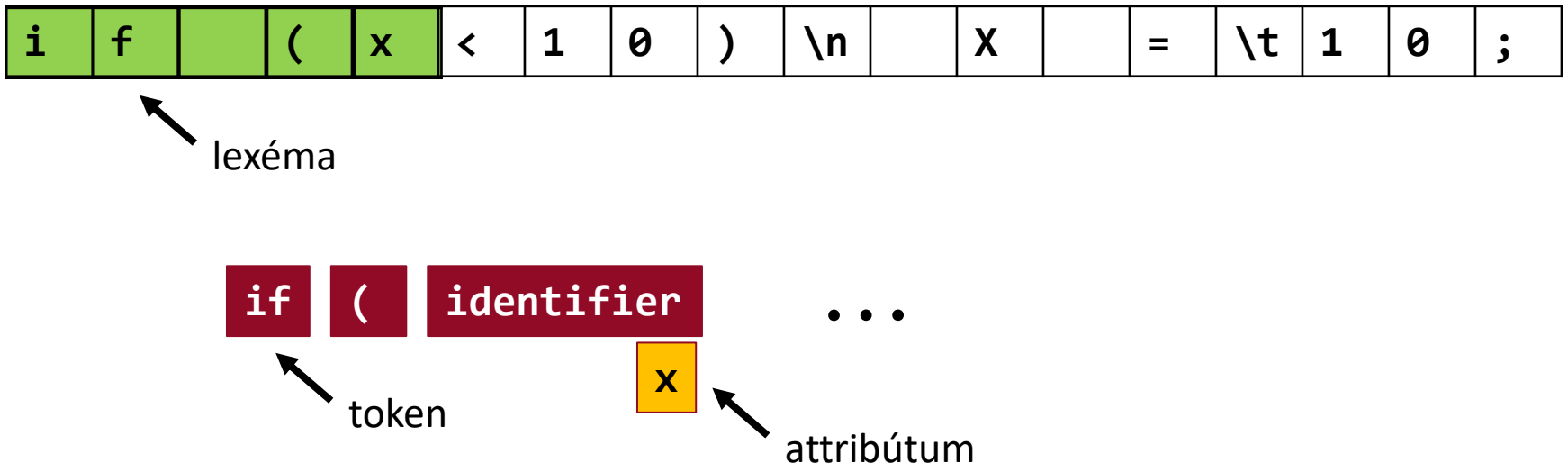
A MAI ELŐADÁS

- I. Szöveges nyelvek
- II. Fordítókról általában
- III. Lexikai elemzés
- IV. Reguláris kifejezések
- V. Formális nyelvek
- VI. Szintaktikai elemzés (bevezető)

LEXIKAI ELEMZÉS

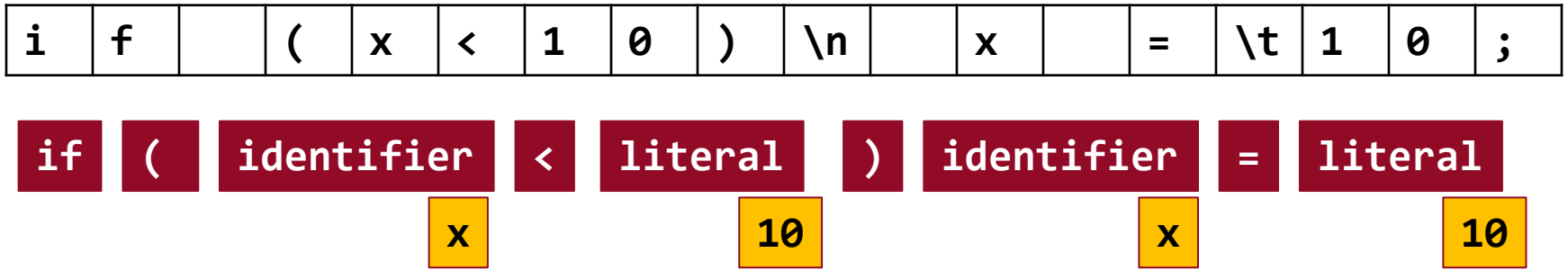
- Lexer végzi
- Bemenet: nyers szöveg (lexémák)
- Kimenet: tokenek előállítása lexémák alapján
  - Tokenizálás folyamata
  - Tokenek és lexémák összerendelése
  - Szintaktikai elemzés számára bemenet

LEXIKAI ELEMZÉS




```
if (x<10)
 x =    10;
```

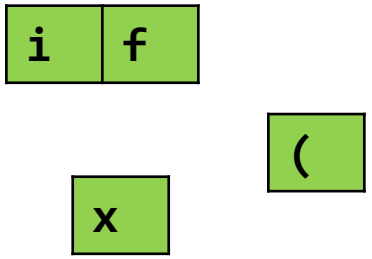
LEXIKAI ELEMZÉS




```
if (x<10)
 x =    10;
```

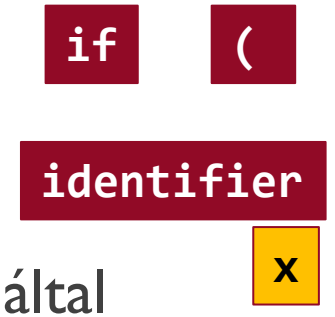
LEXÉMÁK ÉS TOKENEK

- Lexéma
  - Szavak és szimbólumok, amiknek önálló jelentésük lehet
  - Pl. kulcsszavak (*if*, *while*, *for*, stb.), zárójelek, pontosvessző, konstans értékek
  - Nem minden karaktersorozat kell, hogy lexéma legyen (pl. whitespace, comment)
- Szöveg lexémákba tördelése
  - Általános algoritmus:
    - Keressünk olyan karaktereket, amik lexémákat kezdhetnek
    - Találjuk meg a leghosszabb illeszkedő mintát (mohó stratégia)
    - Pl. *whiletrue* → *identifier* lesz belőle, nem *while*



LEXÉMÁK ÉS TOKENEK

- Token
  - Lexémákból készül, hozzátartozik a lexéma mint szöveg
  - Van típusa (pl. T_Identifier) és lehetnek attribútumai (pl. x)
- Tokenizálás
  - A nyers szövegből lexémák, majd ezekből tokenek készítése a lexer által
- Nevezéktan
  - Gyakran csak tokenekről beszélünk és "hozzágondoljuk" a lexémát

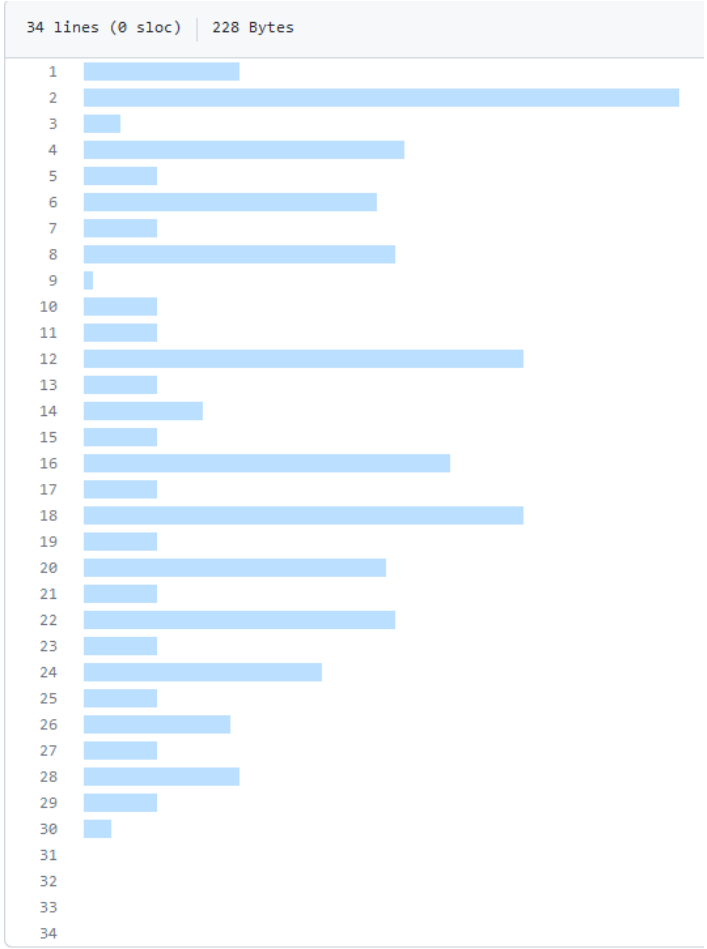


LEXÉMÁK ÉS TOKENEK

- Mi lehet egy token?
  - Kulcsszavak
  - Operátorok
  - Tagoló szavak (pl. pontosvessző)
  - Azonosítók (Id. *identifier*)
  - Konstans értékek (pl. számok, stringek)
- Felesleges karakterek (opcionális) elhagyása
  - Ezekből nem lesz token
  - Nyelvtől függ, hogy mely karakterek (lexémák) feleslegesek
  - Tipikusan whitespace, kommentek, stb.

EZOTERIKUS NYELVEK – AMIKOR KEVÉS TOKENÜNK VAN

- Whitespace nyelv
  - 2003 április 1., Turing-teljes
  - Minden nem whitespace karaktert elhagyunk
- Technikai részletek
  - Stack, heap, 22 utasítás
  - 3 token: [space], [tab], [linefeed]
  - Bináris adatreprezentáció
    - 001110 = [space][space][tab][tab][tab][space][linefeed]



Forrás: https://github.com/rdebath/whitespace/blob/master/tests/rdebath/helloworld.ws

EZOTERIKUS NYELVEK – AMIKOR KEVÉS TOKENÜNK VAN

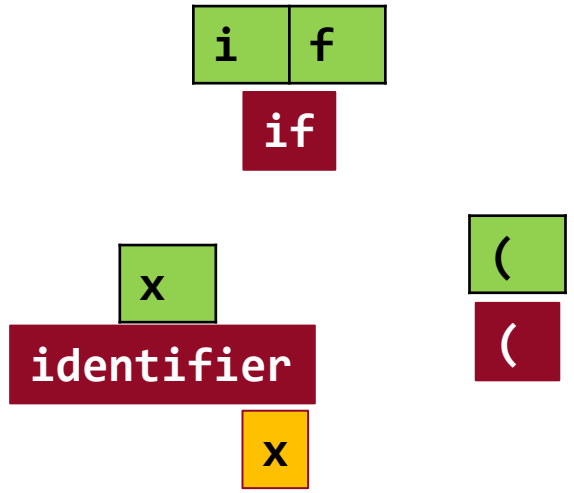
- Brainf***
  - 1993, Turing-teljes
  - 8 karakteren kívül mindent elhagyunk
- Technikai részletek
  - Memóriacellák tömbjén operál [ ]
  - 8 utasítás:
    - Pointer mozgatása: < >
    - Pointernél lévő byte növelése / csökkentése: + -
    - Byte kiírása / bekérése: . ,
    - Ugrás a kódban: [ ]

```
1.  >++++++++[<+++++++++>-]<.
2.  >++++[<+++++++>-]<+.
3.  +++++++..
4.  +++.
5.  >>++++++[<+++++++>-]<++.
6.  ------------.
7.  >++++++[<+++++++++>-]<+.
8.  <.
9.  +++.
10. ------.
11. --------.
12. >>>++++[<++++++++>-]<+.
```

Forrás: https://therenegadecoder.com/code/hello-world-in-brainfuck/

LEXÉMÁK ÉS TOKENEK

- Egy tokenhez akár végtelen lexéma is tartozhat
  - *identifier* → szimbólum (változó, függvény, stb.) nevek, karakterek végtelen kombinációja
- Hogyan rendelhető össze a token a lexémával?
  - Általában reguláris kifejezésekkel
  - Nyelvtani szabályokat írunk
  - Ezen túl – formális nyelvek, véges automaták



A MAI ELŐADÁS

- I. Szöveges nyelvek
- II. Fordítókról általában
- III. Lexikai elemzés
- IV. Reguláris kifejezések
- V. Formális nyelvek
- VI. Szintaktikai elemzés (bevezető)

REGULÁRIS KIFEJEZÉSEK

- Regular expression (regex)
- Mintafelismerésnél gyakori
  - Email cím, jelszó, bankszámlaszám, stb.

```
^[\w\-\.]+@([\w-]+\.)+[\w\-]{2,4}$
```

- Többféle jelölésrendszer
  - Matematikai jelölés
  - Implementáció-függő jelölések
- Interaktív regex validátorok
  - https://regex101.com/

REGULÁRIS KIFEJEZÉSEK – MATEMATIKAI MŰVELETEK

- Az ε szimbólum
  - Üres kifejezés
- Reguláris kifejezések műveletei
  - Unió $R_1|R_2$
  - Konkatenáció R_1R_2
  - Kleene operátor R^*
  - Csoportosítás (R)
- Műveletek prioritása lentről felfelé

REGULÁRIS KIFEJEZÉSEK A GYAKORLATBAN

- String eleje (^) és vége ($) (anchor)
  - Opcionálisan flagek lehetnek a végén (g – globális, i – case insensitive, stb.)
- Csoportosítás – (…), logikai VAGY – |
- Karakterhalmaz – […]
  - Rövidítések (\d – számjegy, \w – alfanumerikus, . – bármilyen karakter, stb.)
  - Speciális karakterek escapelése (pl. \. → pont karakter)
  - Negálás (bármely karakter, ami nem…) – ^
- Számosság (quantifier)
  - ? nulla vagy egy, * nulla vagy több, + egy vagy több, N-től M-ig – {N,M}, pl. {2,4}

REGULÁRIS KIFEJEZÉSEK – PÉLDÁK

- Törtszám
  - **^-?([0-9]*\.)?[0-9]+$**
- URL
  - **^(https?:\/\/)?([\da-z\.-]+)\.([a-z]{2,4})\/?$**
- Forrás: https://www.variables.sh/complex-regular-expression-examples/
- Megjegyzés: mikor kellenek a string eleje és vége anchorok?

REGULÁRIS KIFEJEZÉSEK – PÉLDÁK (GYAKORI TOKENEK)

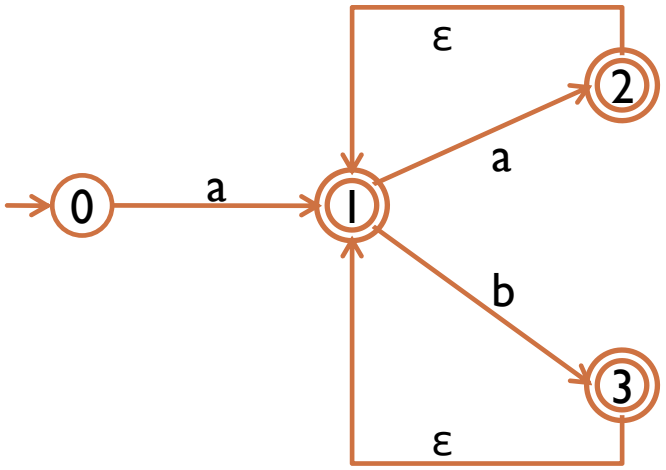
- Identifier (változónév, függvénynév, stb.)
  - **[a-zA-Z_] [a-zA-Z_0-9]***
- Komment (egy soros)
  - **(\/\/)([^\r\n]*)**
- String
  - **(")([^\r\n]*)(")** vagy **(")([^\r\n"]*)(")**
  - Mi a különbség a fenti alternatívák között?

MATEMATIKAI HÁTTÉR*

- Véges automata $M = (Q, \sum, \delta, q_0, F)$, ahol…
  - Q egy véges, nem üres halmaz – állapotok halmaza
  - $\sum$ egy véges, nem üres halmaz – ábécé (elfogadott szavak lehetséges karakterei)
  - δ az állapotátmeneti függvény (hogyan jutunk el egyik állapotból a másikba)
  - q_0 a kezdőállapot
  - $F \subseteq Q$ az elfogadó állapotok halmaza (ha itt állunk meg, elfogadjuk az adott szót)
- Más tárgyból tanuljátok / tanultátok – elvileg ☺
  - Forrás: Csima Judit, Friedl Katalin: Nyelvek és automaták – jegyzet

MATEMATIKAI HÁTTÉR*

- Reguláris kifejezésből véges automata
- a(a|b)* (matematikai jelölés)
- Állapotok (Q)
  - ${}_0a_1(a_2|b_3)^*$
- Állapotátmenetek (δ)
  - $(0,a) \rightarrow 1, (1,a) \rightarrow 2, (1,b) \rightarrow 3, (2,\varepsilon) \rightarrow 1, (3,\varepsilon) \rightarrow 1$
- Kezdőállapot (q_0): 0
- Elfogadó állapotok ($F \subseteq Q$): 1, 2, 3



A MAI ELŐADÁS

I. Szöveges nyelvek

II. Fordítókról általában

III. Lexikai elemzés

IV. Reguláris kifejezések

V. Formális nyelvek

VI. Szintaktikai elemzés (bevezető)

FELADAT

- Számológépet szeretnénk leírni reguláris kifejezés segítségével. A számológép a négy alapműveletet, illetve a zárójelezés műveletét ismeri. Az egyszerűség kedvéért csak az egyjegyű, pozitív egész számokat támogatjuk.
- Megoldás
  - [0-9]
  - [0-9][\+\-*\/][0-9]
  - [0-9]([\+\-*\/][0-9])+
  - [0-9]([\+\-*\/]\(?[0-9]\)?)+
  - **Jó ez?**

REGULÁRIS KIFEJEZÉSEK HIÁNYOSSÁGAI

- A példánkban nem tudtunk rekurziót leírni
  - Ha valahol kezdődik egy zárójel, előbb-utóbb záruljon is be!
- A reguláris kifejezések kifejezőereje néha elég…
- …de komplexebb esetekben általában nem
  - Egymásba ágyazott kifejezés blokkok
  - Helyesen zárójelezett kifejezések
  - stb.
- Erősebb formalizmusra van szükségünk!

FORMÁLIS NYELVEK (INFORMÁLISAN)

- Ábécé – tetszőleges, nem üres, véges halmaz
- Betű – az ábécé egy eleme
- Szó – betűkből felépített véges hosszú sorozat (szöveg)
- Nyelv – szavak tetszőleges (véges vagy végtelen) halmaza

CONTEXT-FREE (CF) NYELVTANOK

- Egy CF nyelvtan felépítése:
  - Produkciós szabályok
  - Nemterminális szimbólumok (változók)
    - Szintaxisfa közbenső csúcsai
  - Terminális szimbólumok (szavak)
    - Szintaxisfa levelei – tokenek (ld. lexer)
  - Kezdőváltozó (startszimbólum)
    - Tipikusan az első szabály bal oldala (de nem kötelezően)

$E \rightarrow 0 \mid 1 \mid \ldots$

$E \rightarrow E\ Op\ E$

$E \rightarrow (E)$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

$Op \rightarrow /$

CONTEXT-FREE (CF) NYELVTANOK

- A nyelvtan egy formális nyelvet ír le
- Környezetfüggetlen (*context-free, CF*)
  - Szabály bal oldalán pontosan egy nemterminális
  - Jobb oldalon terminális és / vagy nemterminális sorozat
  - CF nyelvtakon (nyelvek) a gyakorlatban gyakran használhatók
- A példán egy számológép nyelvtana látható
  - Egész számok, 4 alapművelet, zárójelezés

$E \rightarrow 0 \mid 1 \mid \ldots$

$E \rightarrow E\ Op\ E$

$E \rightarrow (E)$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

$Op \rightarrow /$

CF NYELVTANOK – JELÖLÉS

$E \rightarrow 0 \mid 1 \mid \ldots$

$E \rightarrow E\ Op\ E$

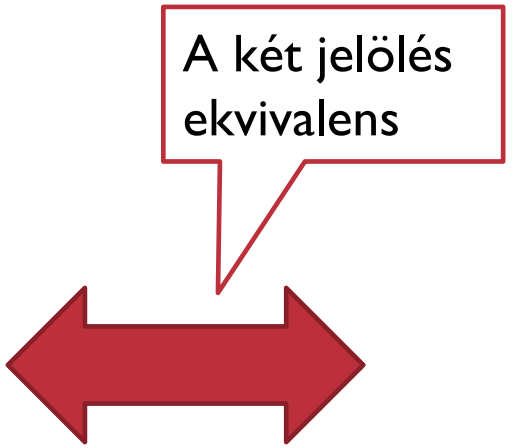
$E \rightarrow (E)$

$Op \rightarrow +$

$Op \rightarrow -$

$Op \rightarrow *$

$Op \rightarrow /$



$E \rightarrow 0 \mid 1 \mid \ldots \mid E\ Op\ E \mid (E)$

$Op \rightarrow + \mid - \mid * \mid /$

CF NYELVTANOK – LEVEZETÉS

3 * (1 + 2)

A levezetendő szöveg

$E \rightarrow 0|1|\ldots | E\ Op\ E | (E)$

$Op \rightarrow + | - | * | /$

A nyelvtan (szabályokkal leírva)

Bal oldali levezetés

E

$\Rightarrow E\ Op\ E$

$\Rightarrow 3\ Op\ E$

$\Rightarrow 3 * E$

$\Rightarrow 3 * (E)$

$\Rightarrow 3 * (E\ Op\ E)$

$\Rightarrow 3 * (1\ Op\ E)$

$\Rightarrow 3 * (1 + E)$

$\Rightarrow 3 * (1 + 2)$

CF NYELVTANOK – LEVEZETÉS

3 * (1 + 2)

A levezetendő szöveg

E → 0|1|… | E Op E | (E)

Op → + | - | * | /

A nyelvtan (szabályokkal leírva)

Jobb oldali levezetés

E

⇒ E Op E

⇒ E Op (E)

⇒ E Op (E Op E)

⇒ E Op (E Op 2)

⇒ E Op (E + 2)

⇒ E Op (1 + 2)

⇒ E * (1 + 2)

⇒ 3 * (1 + 2)

CF NYELVTANOK – LEVEZETÉS

Bal oldali levezetés

E

⇒ E Op E

⇒ 3 Op E

⇒ 3 * E

⇒ 3 * (E)

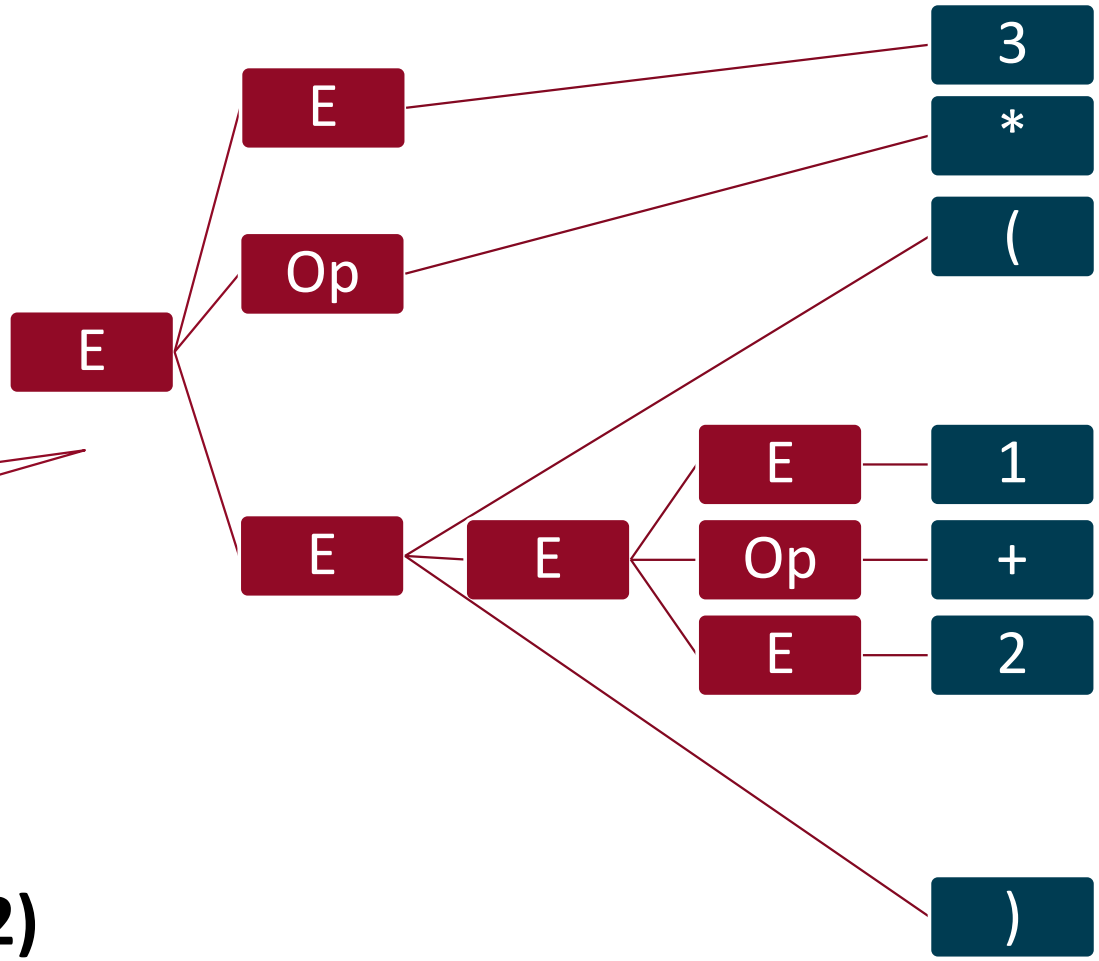
⇒ 3 * (E Op E)

⇒ 3 * (1 Op E)

⇒ 3 * (1 + E)

⇒ 3 * (1 + 2)

Levezetési fa



3 * (1 + 2)

CF NYELVTANOK – LEVEZETÉS

- Bal és jobb oldali levezetés
  - Különbség a sorrend
  - Mindig a bal vagy jobb oldali szélső nemterminális "kibontása" (behelyettesítése)
  - A gyakorlatban általában bal oldali levezetést használunk (de nem mindig)
- Levezetési fa
  - A levezetés során előálló adatstruktúra
  - Nem csak bal oldali levezetés esetén
- CF nyelvtan tesztelő tool
  - https://checker5965.github.io/toc.html

CF NYELVTANOK – PÉLDA (NYELVTAN)

```
public class MyClass {
  private int x;
  public string s;
}
```

Lexer adja az *id* tokent, nem bontjuk tovább

S → V class id { A }

A → AA | V T idE | ε

V → public | private | …

T → int | string | …

E → ;

Class → Vis class id { Att }

Att → AttAtt | Vis Typ idEoS | ε

Vis → public | private | …

Typ → int | string | …

EoS → ;

CF NYELVTANOK – PÉLDA (LEVEZETÉS)

```
public class MyClass {
  private int x;
  public string s;
}
```

$S \rightarrow V\ \text{class id}\ \{\ A\ \}$

$A \rightarrow AA \mid V\ T\ \text{id}E \mid \varepsilon$

$V \rightarrow \text{public} \mid \text{private} \mid \ldots$

$T \rightarrow \text{int} \mid \text{string} \mid \ldots$

$E \rightarrow\ ;$

S

$\Rightarrow V\ \text{class id}\ \{\ A\ \}$

$\Rightarrow \text{public class id}\ \{\ A\ \}$

$\Rightarrow \text{public class id}\ \{\ AA\ \}$

$\Rightarrow \text{public class id}\ \{V\ T\ \text{id}E\ A\ \}$

$\Rightarrow \text{public class id}\ \{\text{private}\ T\ \text{id}E\ A\ \}$

$\Rightarrow \text{public class id}\ \{\text{private int id}E\ A\ \}$

$\Rightarrow \text{public class id}\ \{\text{private int id;}\ A\ \}$

$\Rightarrow \text{public class id}\ \{\text{private int id;}\ V\ T\ \text{id}\ E\ \}$

$\Rightarrow \ldots$

BACHUS-NAUR FORM (BNF)

- Egy elterjedt, gyakorlati jelölés CF nyelvtanok leírására
- Nemterminális szimbólumok: **<…>** között
- Terminális szimbólumok: **egyszerű karakterek**
  - Vagy aposztrófok között: **‘egyszerű karakterek’**
- Produkciós szabályoknál → helyett **::=**
- Produkciós szabályokat néha ‘.’ vagy ‘;’ zárja le
  - Több variáns létezik

BACHUS-NAUR FORM (BNF)

- Példa
  - **<E> ::= <digit> | <E> <Op> <E> | (<E>)**
  - **<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
  - **<Op> ::= + | - | * | /**

E → 0|1|… | E Op E | (E)

Op → + | - | * | /

EXTENDED BACHUS-NAUR FORM (EBNF)

- Egy kényelmesebb, még gyakorlatibb jelölés CF nyelvtanok leírására
- Nemterminális szimbólumok: <…> elhagyva
- Terminális szimbólumok: mindig aposztrófok között
  'egyszerű karakterek'
- Csoportosítás: (…)
- Opcionális elem: […] (a ?-nek felel meg reguláris kifejezéseknél)
- Ismétlés nullaszor vagy többször: *
- Ismétlés egyszer vagy többször: +

EXTENDED BACHUS-NAUR FORM (EBNF)

- Példa
  - **E ::= (digit)+ | E Op E | ‘(‘ E ‘)’**
  - **digit ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’**
  - **Op ::= ‘+’ | ‘-’ | ‘*’ | ‘/’**

Több számjegyünk is lehet; vigyázat: nem szabványos jelölés!

E → (0|1|…)+ | E Op E | (E)

Op → + | - | * | /

A MAI ELŐADÁS

I. Szöveges nyelvek

II. Fordítókról általában

III. Lexikai elemzés

IV. Reguláris kifejezések

V. Formális nyelvek

VI. Szintaktikai elemzés (bevezető)

SZINTAKTIKAI ELEMZÉS

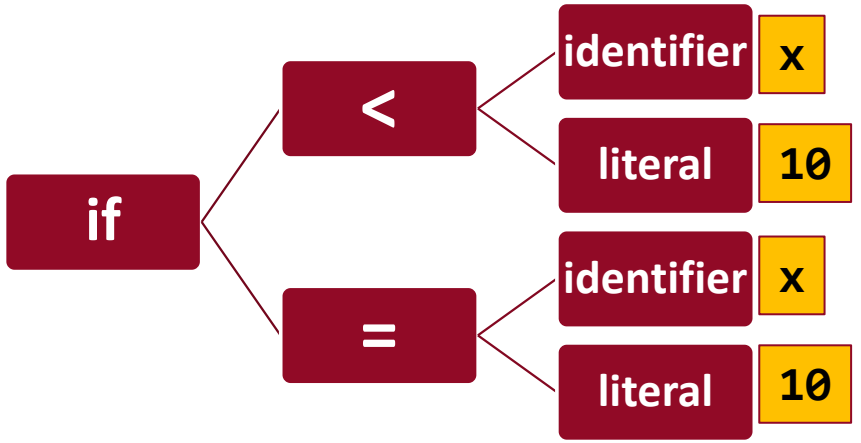
- Parser végzi
- Bemenet: lexikai elemzés során előállított tokenek
- Kimenet: (fa) struktúra előállítása a tokenek alapján
  - Hatékony feldolgozhatóság érdekében
  - Hibás struktúra felismerése – szintaktikai hibák
- Manapság tipikusan a struktúra határozza meg a kód jelentését
  - Olyan struktúrába rendezzük a kódot, hogy "olvasható" legyen
  - Később – szemantikai elemzésnél egyéb szemantika ellenőrzése

SZINTAKTIKAI ELEMZÉS

```
if (x<10)
 x = 10;
```

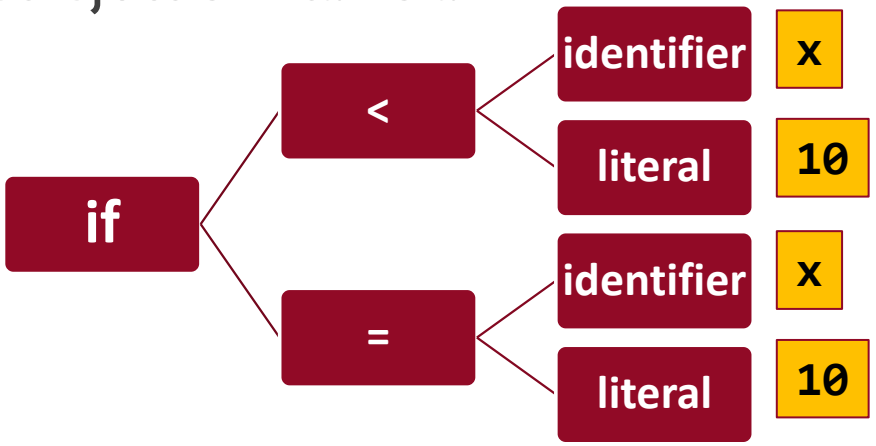
i	f		(	x	<	1	0	)	\n		x		=	\t	1	0	;

if	(	identifier	<	literal	)	identifier	=	literal
		x		10		x		10



SZINTAKTIKAI ELEMZÉS

- CF nyelvtannal való leírás esetén
  - Szintaxisfa felépítése
  - Elméletben lehetne más adatszerkezet is…
  - …de a gyakorlatban szinte mindig szintaxisfával dolgozunk
  - Levezetési fa (*parse tree*): a levezetés során létrejött szintaxisfa
- Hogyan építhető fel a szintaxisfa?
  - Különböző algoritmusok segítségével
  - Részletek a következő előadáson

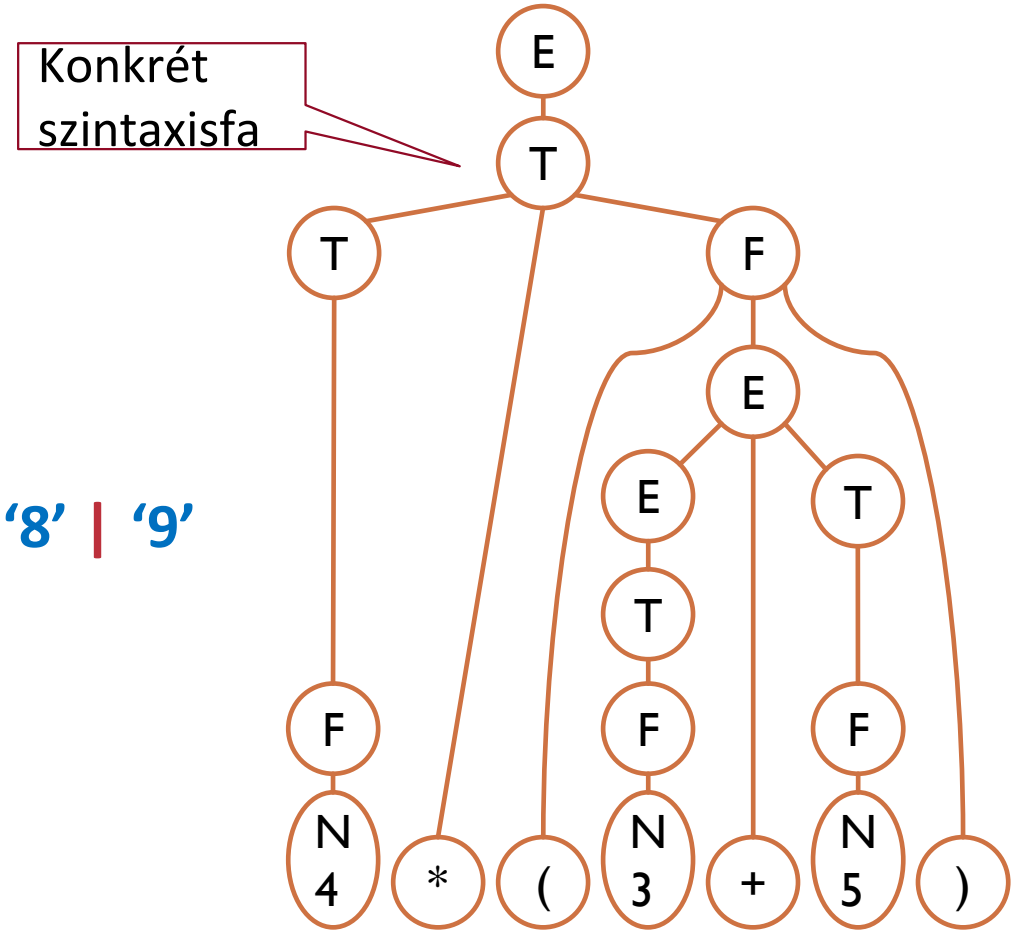


KONKRÉT ÉS ABSZTRAKT SZINTAXISFA

- Konkrét szintaxisfa (*Concrete Syntax Tree / CST*)
  - A levezetési fát szokás konkrét szintaxisfának is hívni
  - A szöveg pontos levezetését tartalmazza, benne minden kulcsszóval, stb.
- Absztrakt szintaxisfa (*Abstract Syntax Tree / AST*)
  - Csak a struktúra lényegi részét tartalmazza
  - A felesleges részeket (pl. szintaktikai kényszerek, kulcsszavak) kiszedjük a fából
  - Zárójelezést, struktúrát a fahierarchia adja
- A gyakorlatban CST és AST is előfordulhat

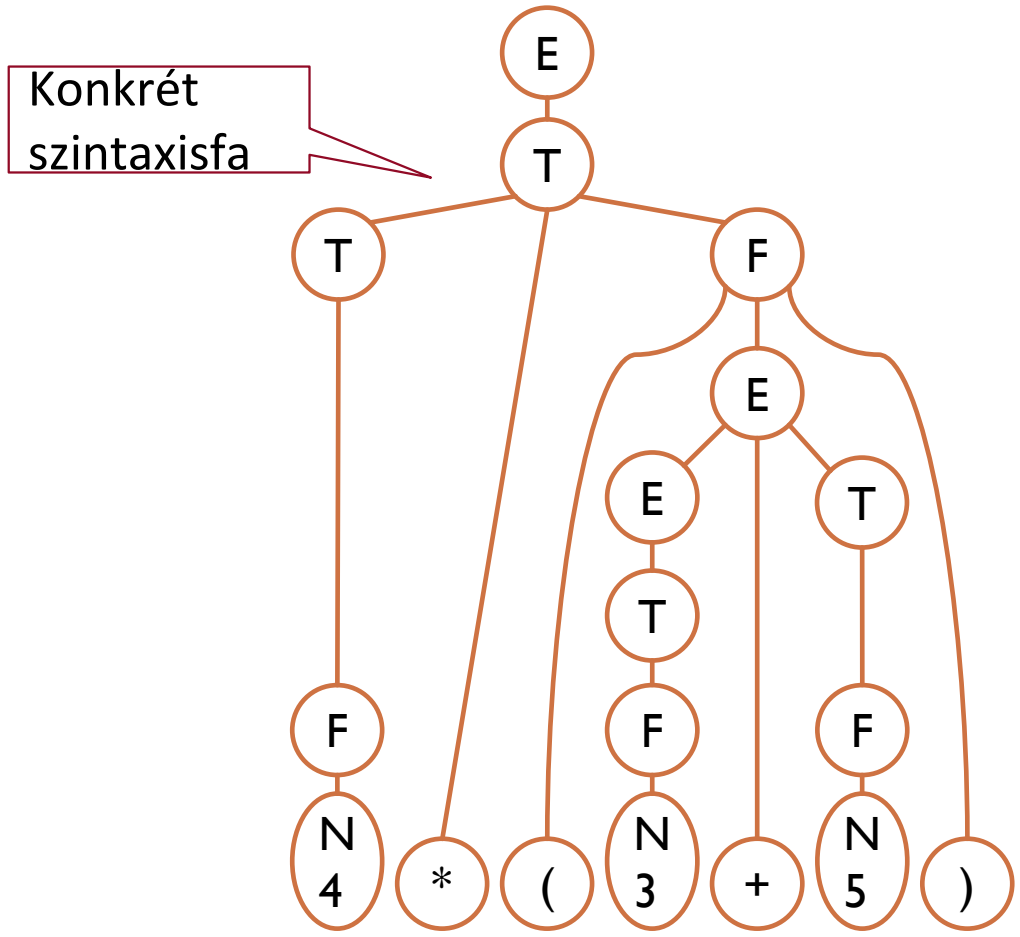
KONKRÉT ÉS ABSZTRAKT SZINTAXISFA – PÉLDA

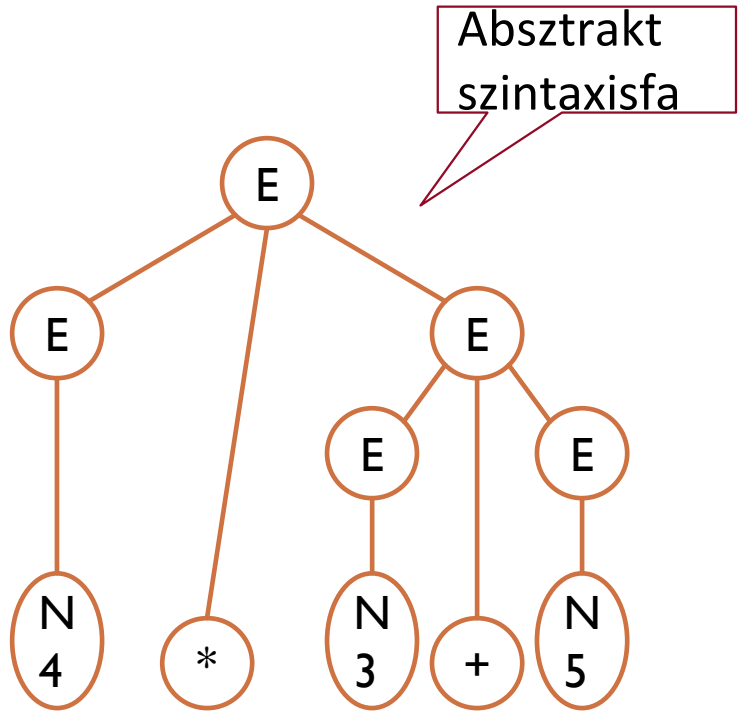
- Számológép – összeadás, szorzás, zárójelezés
- EBNF nyelvtan
  - **E ::= T (‘+‘ T)***
  - **T ::= F (‘*‘ T)***
  - **F ::= N | ‘(‘ E ‘)‘**
  - **N ::= ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | ‘8’ | ‘9’**
- Nyers szöveg: 4*(3+5)



KONKRÉT ÉS ABSZTRAKT SZINTAXISFA – PÉLDA

- Csak a fontos elemek megtartása
  - **E ::= E ‘+‘ E | E ‘*‘ E | N**





KÖSZÖNÖM A FIGYELMET!