

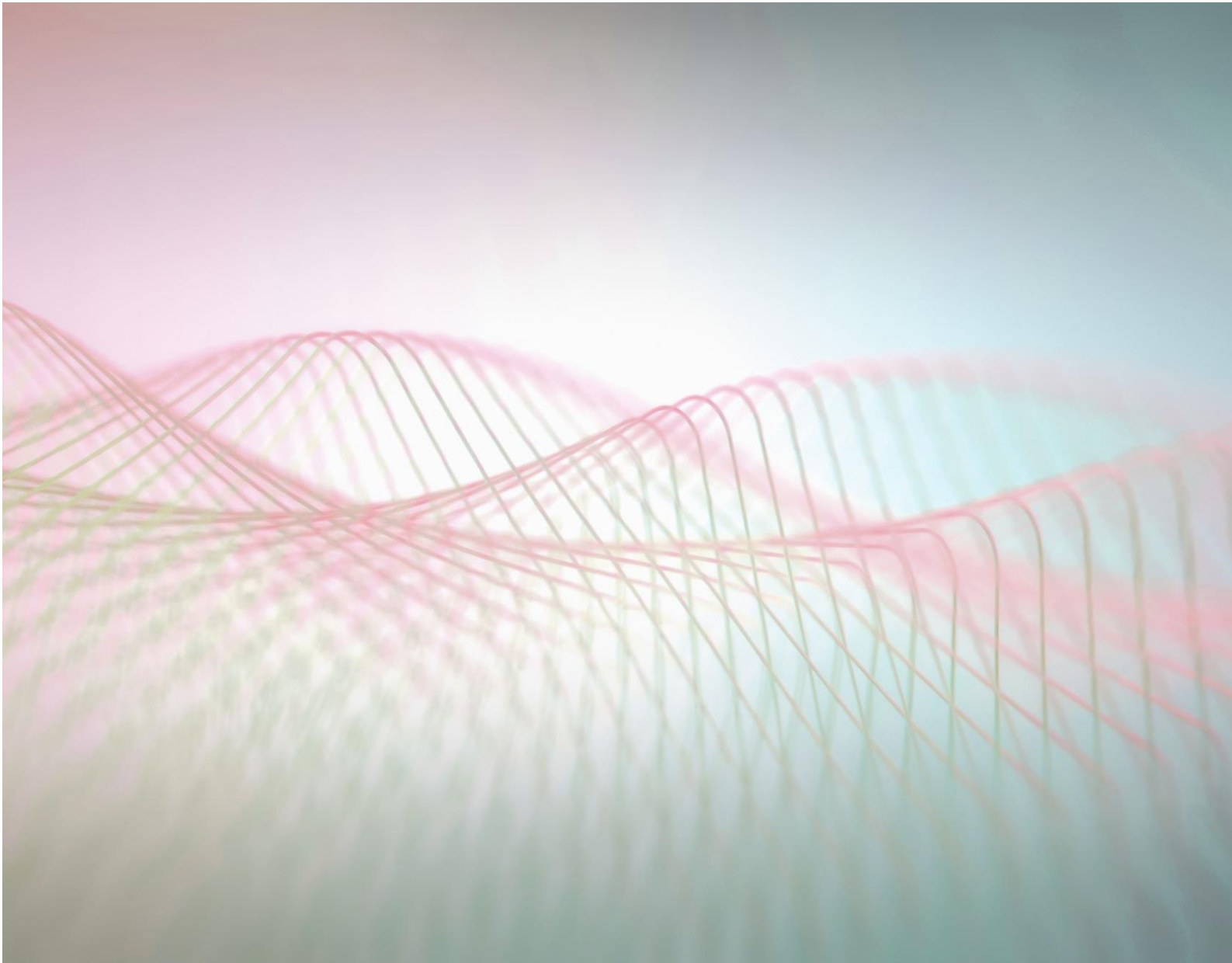


MODELLALAPÚ SZOFTVERFEJLESZTÉS

III. ELŐADÁS

LEXIKAI ÉS SZINTAKTIKAI ELEMZÉS

DR. SOMOGYI FERENC



MIELŐTT ELKEZDENÉNK

ZH információk

ZH INFORMÁCIÓK

- **ZH** – 2023-04-26, Szerda, 18-20
- **PótZH** – 2023-05-08, Hétfő, 18-20

A MAI ELŐADÁS

I. Lexikai elemzés

II. Reguláris kifejezések

III. Formális nyelvek

IV. 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

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

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

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

LEXIKAI ELEMZÉS

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

lexéma

if (identifier ...

token

x

attribútum

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

LEXÉMÁK ÉS TOKENEK

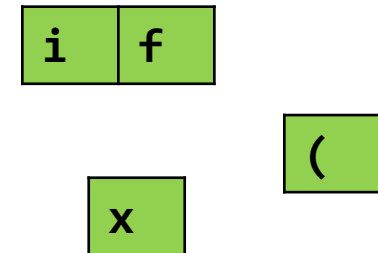
■ Lexéma

- 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, a szövege a lexéma
- Van típusa (pl. `T_Identifier`) és lehetnek attribútumai (pl. `x`)

■ Tokenizálás

- A nyers szövegből lexémák, majd tokenek készítése a lexer által

■ Nevezéktan

- Gyakran csak tokenekről beszélünk és “hozzágondoljuk” a lexémát

if

(

identifier

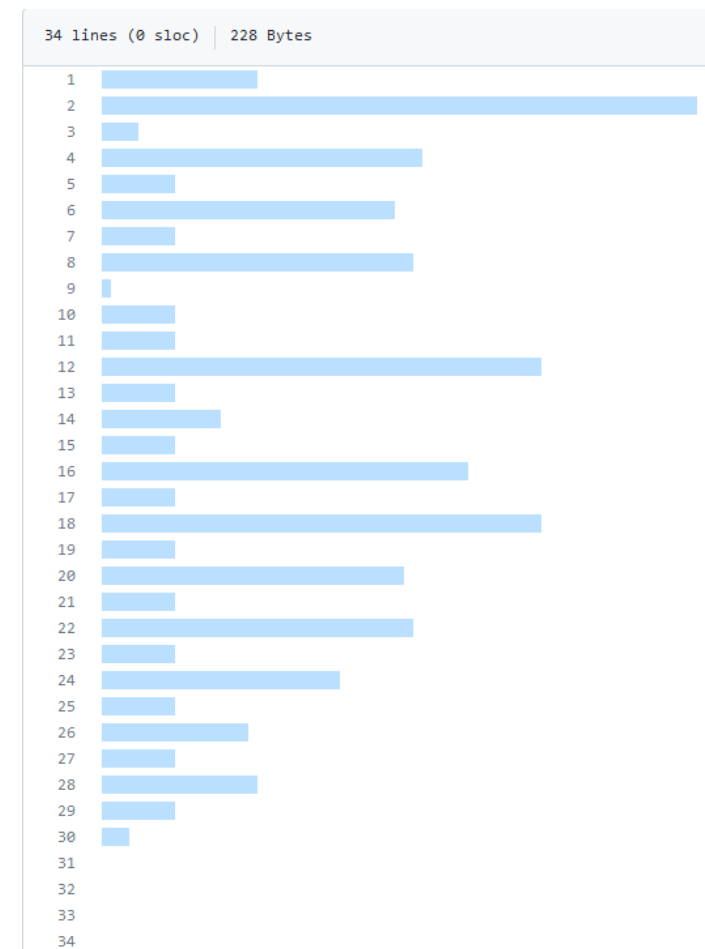
x

LEXÉMÁK ÉS TOKENEK

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

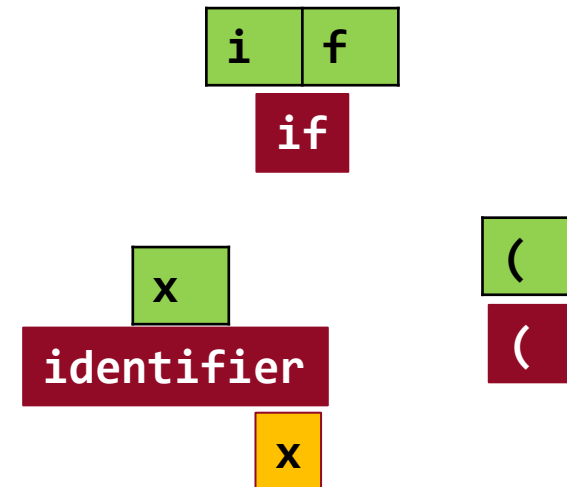
- Pointer mozgatása: < >
- Pointernél lévő byte növelése / csökkentése: + -
- Byte kiírása / bekérése: . ,
- Ugrás a kódban: []

```
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
 - Pl. *identifier* → szimbólum (változó, függvény, stb.) nevek, végtelen kombináció
- Hogyan rendelhetők össze?
 - Általában reguláris kifejezésekkel
 - Nyelvtani szabályokat írunk
 - Ezen túl – formális nyelvek, véges automaták



A MAI ELŐADÁS

I. Lexikai elemzés

II. Reguláris kifejezések

III. Formális nyelvek

IV. Szintaktikai elemzés (bevezető)



REGULÁRIS KIFEJEZÉSEK

- Regular expression (regex) – biztosan ismerős
- Mintafelismerésnél gyakori
 - Email cím, jelszó, bankszámlaszám, stb.
- Többféle jelölésrendszer
 - Matematikai jelölés
 - Implementáció-függő jelölések is lehetnek
- Interaktív regex validator
 - <https://regex101.com/>
 - Stb.

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

REGULÁRIS KIFEJEZÉSEK – MATEMATIKAI MŰVELETEK

- Az ε szimbólum
- Reguláris kifejezések műveletei
 - Unió $R_1 | R_2$
 - Konkatenáció $R_1 R_2$
 - 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 – |
- Adott karakter halmaz – [...]
 - Rövidítések (\d – számjegy, \w – alfanumerikus, . – bármilyen karakter, stb.)
 - Speciális karakterek escapelése (pl. \. → pont karakter)
 - Negálás (bármi, ami nem az adott karakter halmaz) – ^
- 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})$`

- IP cím

- `^(((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$`

- Forrás: <https://www.variables.sh/complex-regular-expression-examples/>

- Megjegyzés: string eleje és vége anchorok: mikor kell?

REGULÁRIS KIFEJEZÉSEK – PÉLDÁK (TIPIKUS TOKENEK)

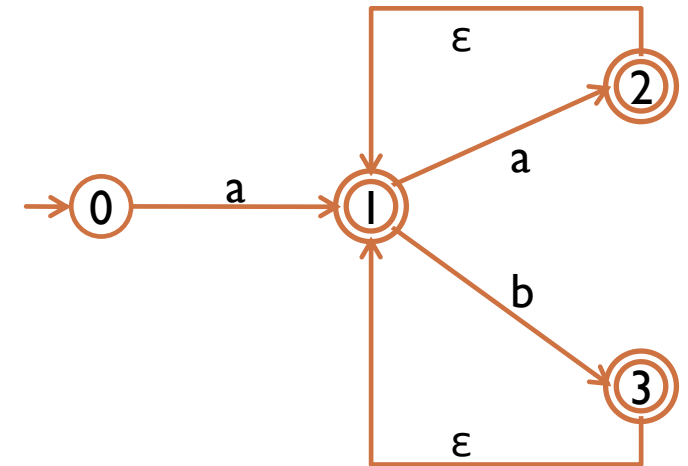
- Identifier (változónév, függvéynév, stb.)
 - **`[a-zA-Z_] [a-zA-Z_0-9]*`**
- Komment (egy soros)
 - **`(//)([^\r\n]*)`**
- String
 - **`(")([^\r\n]*)(")` vagy **`(")([^\r\n"]*)(")`****
 - **`^(")([^\r\n]*)("$` vagy **`^(")([^\r\n"]*)(")$`****
 - Mi a különbség a fentiek között?

LEXIKAI ELEMZÉS – MATEMATIKAI HÁTTÉR*

- Véges automata $M = (Q, \Sigma, \delta, q_0, F)$, ahol...
 - Q egy véges, nem üres halmaz – állapotok halmaza
 - Σ 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
- Ennél a tárgynál nem fontosak a mélyebb matematikai részletek

LEXIKAI ELEMZÉS – MATEMATIKAI HÁTTÉR*

- Reguláris kifejezésből véges automata
- Példa: $a(a|b)^*$ (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. Lexikai elemzés

II. Reguláris kifejezések

III. Formális nyelvek

IV. 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 kezdődik egy zárójel, egyszer záródjon is!
- 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

- Ábécé – tetszőleges, nem üres, véges halmaz
- Betű – az ábécé egy eleme
- Szó – betűkből felépített véges hosszú sorozat (szöveg)
- Nyelv – szavak tetszőleges (véges vagy végtelen) részhalmaza
- Formális definíciókat itt nem tárgyaljuk
 - Bővebben: Bach Iván: Formális nyelvek
Csima Judit, Friedl Katalin: Nyelvek és automaták – jegyzet

CONTEXT-FREE (CF) NYELVTANOK

- A 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

$E \rightarrow 0 | 1 | \dots$

$E \rightarrow E \text{ Op } E$

$E \rightarrow (E)$

$\text{Op} \rightarrow +$

$\text{Op} \rightarrow -$

$\text{Op} \rightarrow *$

$\text{Op} \rightarrow /$

CONTEXT-FREE (CF) NYELVTANOK

- A nyelvtan egy formális nyelvet ír le
- Környezetfüggetlen (*context-free*, *CF*)
 - Bal oldalon egy **nemterminális**
 - Jobb oldalon **terminális** és / vagy **nemterminális** sorozat
 - A gyakorlatban gyakran előfordul
- Az ábrán egy számológép nyelvtana látható
 - Egész számok, 4 alapl művelet, zárójelezés

$E \rightarrow 0 | 1 | \dots$

$E \rightarrow E \text{ Op } E$

$E \rightarrow (E)$

$\text{Op} \rightarrow +$

$\text{Op} \rightarrow -$

$\text{Op} \rightarrow *$

$\text{Op} \rightarrow /$

CF NYELVTANOK – ELMÉLETI JELÖLÉS

$E \rightarrow 0|1|...$

$E \rightarrow E \text{ Op } E$

$E \rightarrow (E)$

$\text{Op} \rightarrow +$

$\text{Op} \rightarrow -$

$\text{Op} \rightarrow *$

$\text{Op} \rightarrow /$

A két jelölés
ekvivalens



$E \rightarrow 0|1|... | E \text{ Op } E | (E)$

$\text{Op} \rightarrow + | - | * | /$

CF NYELVTANOK – LEVEZETÉS

3 * (1 + 2)

A levezetendő
szöveg

Bal oldali
levezetés

E → **0** | **1** | ... | **E Op E** | **(E)**

Op → **+** | **-** | ***** | **/**

A nyelvтанunk
(szabályokkal leírva)

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)**

CF NYELVTANOK – LEVEZETÉS

3 * (1 + 2)

A levezetendő
szöveg

Jobb oldali
levezetés

$E \rightarrow 0 | 1 | \dots | E \text{ Op } E | (E)$

$\text{Op} \rightarrow + | - | * | /$

A nyelvtanunk
(szabályokkal leírva)

E
 $\Rightarrow E \text{ Op } E$
 $\Rightarrow E \text{ Op } (E)$
 $\Rightarrow E \text{ Op } (E \text{ Op } E)$
 $\Rightarrow E \text{ Op } (E \text{ Op } 2)$
 $\Rightarrow E \text{ Op } (E + 2)$
 $\Rightarrow E \text{ Op } (1 + 2)$
 $\Rightarrow E * (1 + 2)$
 $\Rightarrow 3 * (1 + 2)$

CF NYELVTANOK – 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**)

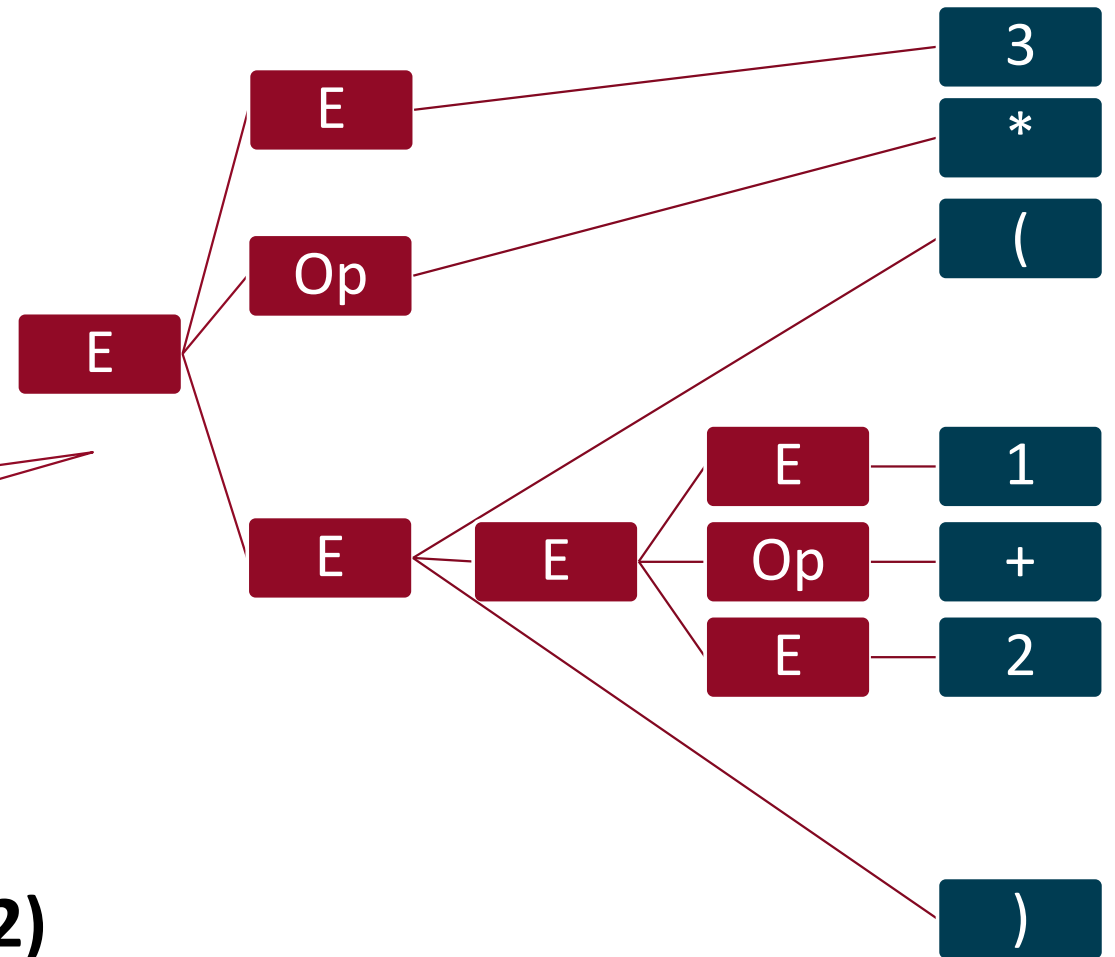
\Rightarrow **3** * (**1** + **E**)

\Rightarrow **3** * (**1** + **2**)

Bal oldali
levezetés

Levezetési
fa

3 * (1 + 2)



CF NYELVTANOK – LEVEZETÉS

- Bal és jobb oldali levezetés
 - Különbség a sorrend
 - Mindig a bal vagy jobb oldali szélső nemterminális (változó) “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* tokenet,
most nem írjuk le külön

$S \rightarrow V \text{ class id } \{ A \}$
 $A \rightarrow AA \mid V T \text{ id} E \mid \epsilon$
 $V \rightarrow \text{public} \mid \text{private} \mid \dots$
 $T \rightarrow \text{int} \mid \text{string} \mid \dots$
 $E \rightarrow ;$

$\text{Class} \rightarrow \text{Vis class id } \{ \text{Att} \}$
 $\text{Att} \rightarrow \text{AttAtt} \mid \text{Vis Typ idEoS} \mid \epsilon$
 $\text{Vis} \rightarrow \text{public} \mid \text{private} \mid \dots$
 $\text{Typ} \rightarrow \text{int} \mid \text{string} \mid \dots$
 $\text{EoS} \rightarrow ;$

CF NYELVTANOK – PÉLDA (LEVEZETÉS)

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

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

$A \rightarrow AA \mid V T \text{ idE } \mid \epsilon$

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

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

$E \rightarrow ;$

S

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

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

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

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

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

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

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

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

$\Rightarrow \dots$

BACHUS-NAUR FORM (BNF)

- Egy elterjedt, gyakorlati jelölés CF nyelvtanok leírására
- Nemterminális szimbólumok: $\langle \dots \rangle$ között
- Terminális szimbólumok: **egyszerű karakterek**
 - Vagy aposztrófok között: **'egyszerű karakterek'**
- Produkciós szabályoknál \rightarrow helyett $::=$
- Produkciós szabályokat néha $'$ vagy $;$ zárja le
 - Több variáns létezik

BACHUS-NAUR FORM (BNF)

■ Példa

- $\langle E \rangle ::= \langle \text{digit} \rangle \mid \langle E \rangle \langle \text{Op} \rangle \langle E \rangle \mid (\langle E \rangle)$
- $\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- $\langle \text{Op} \rangle ::= + \mid - \mid * \mid /$

$E \rightarrow 0 \mid 1 \mid \dots \mid E \text{ Op } E \mid (E)$
 $\text{Op} \rightarrow + \mid - \mid * \mid /$

EXTENDED BACHUS-NAUR FORM (EBNF)

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

EXTENDED BACHUS-NAUR FORM (EBNF)

■ Példa

- $E ::= (\text{digit})^+ \mid E \text{ Op } E \mid '(' E ')'$
- $\text{digit} ::= '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9'$
- $\text{Op} ::= '+' \mid '-' \mid '*' \mid '/'$

Több számjegyük is lehet; vigyázat: nem szabványos elméleti jelölés!

$E \rightarrow (0|1|\dots)^+ \mid E \text{ Op } E \mid (E)$
 $\text{Op} \rightarrow + \mid - \mid * \mid /$

A MAI ELŐADÁS

I. Lexikai elemzés

II. Reguláris kifejezések

III. Formális nyelvek

IV. 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

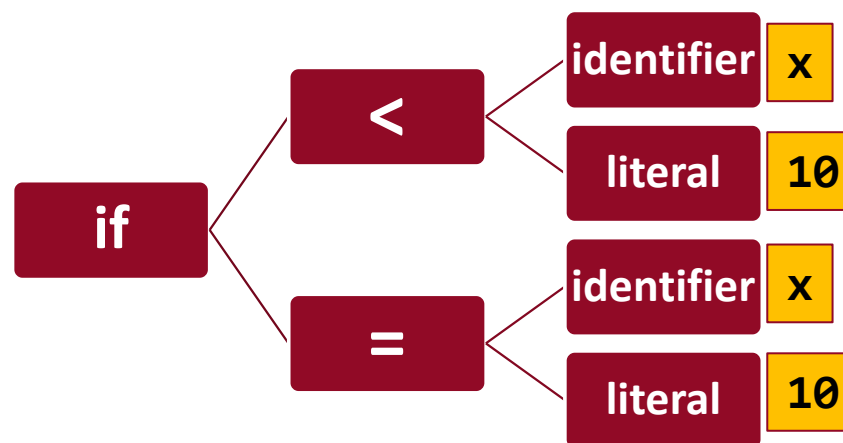


SZINTAKTIKAI ELEMZÉS

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

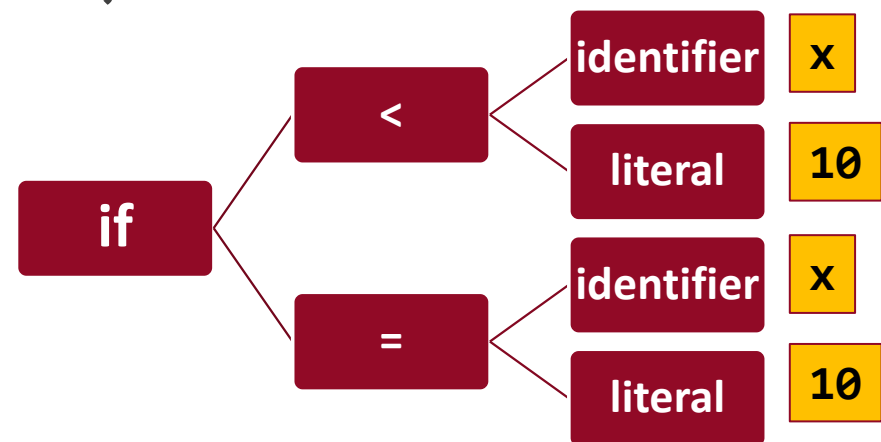
i	f		(x	<	1	0)	\n		x		=	\t	1	0	;
---	---	--	---	---	---	---	---	---	----	--	---	--	---	----	---	---	---

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



SZINTAKTIKAI ELEMZÉS

- CF nyelvtannal való leírás esetén
 - Szintaxisfa előállítása
 - 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

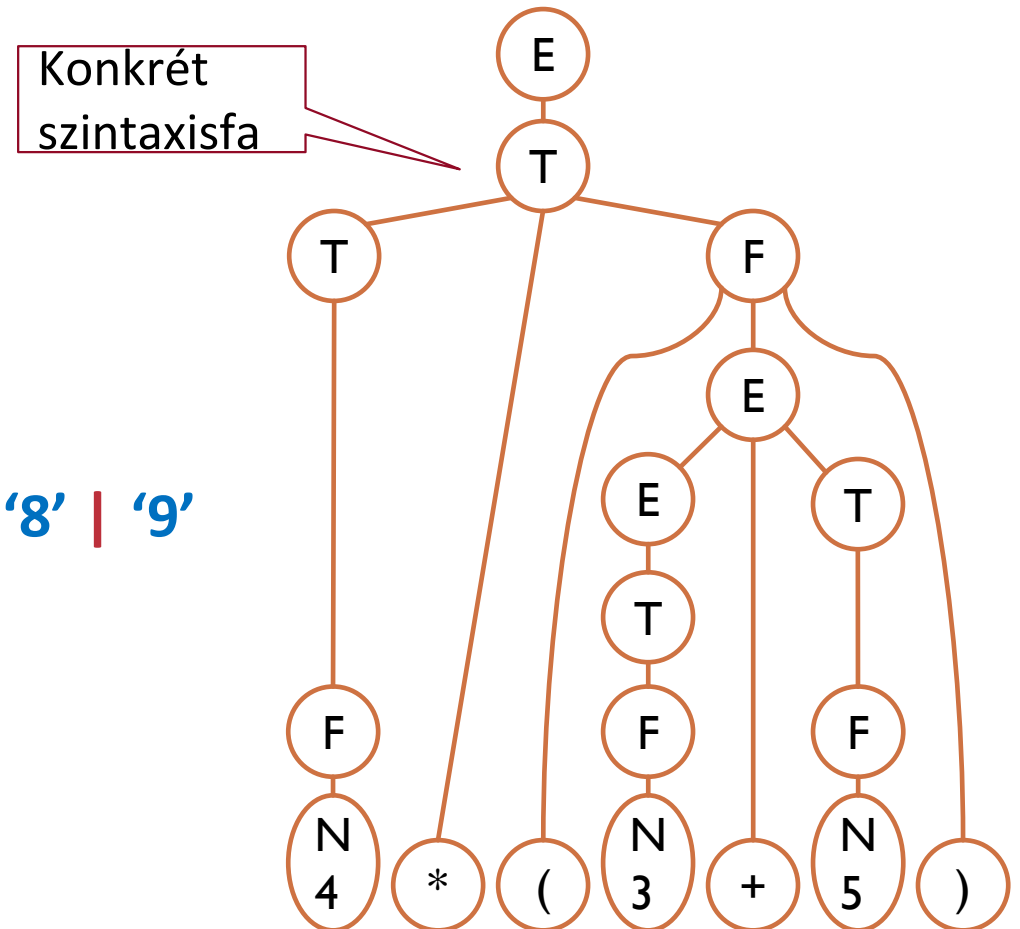
- Konkrét szintaxisfa (*Concrete Syntax Tree / CST*)
 - A levezetési fát szokás konkrét szintaxisfának is hívni
 - A szöveg pontos levezetését tartalmazza, benne minden kulcsszóval, stb.
- Absztrakt szintaxisfa (*Abstract Syntax Tree / AST*)
 - Csak a struktúra lényegi részét tartalmazza
 - A felesleges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

KONKRÉT ÉS ABSZTRAKT SZINTAXISFA – PÉLDA

- Számológép – összeadás, szorzás, zárójelezés
- EBNF nyelvtan

Konkrét szintaxi

 - $E ::= T ('+' T)^*$
 - $T ::= F ('*' T)^*$
 - $F ::= N | '(' E ')'$
 - $N ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'$
- Nyers szöveg: $4*(3+5)$

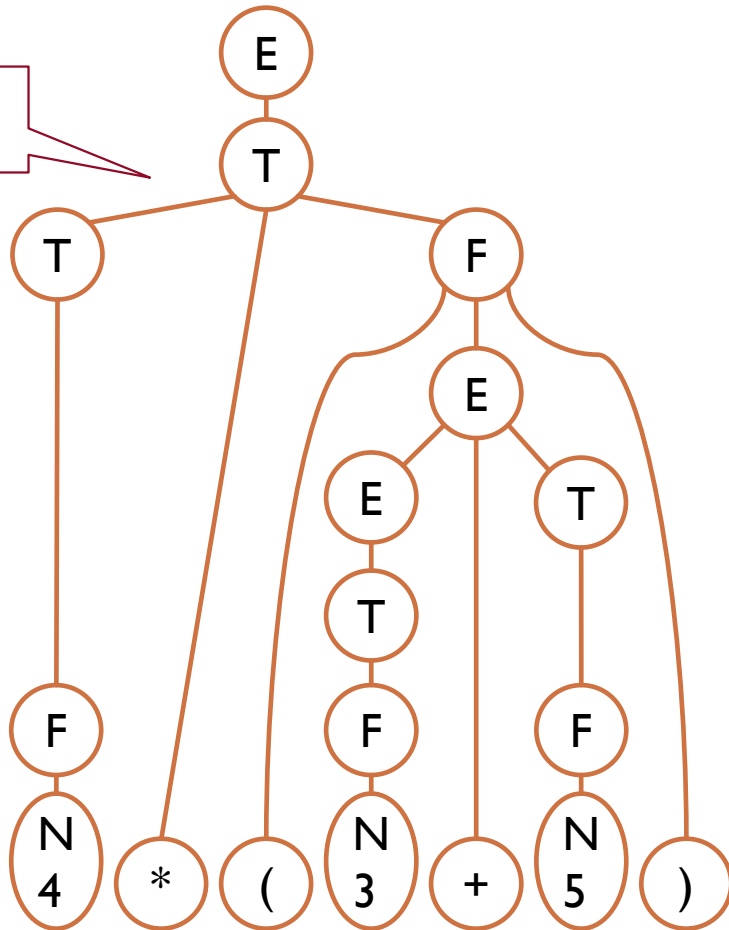


KONKRÉT ÉS ABSZTRAKT SZINTAXISFA – PÉLDA

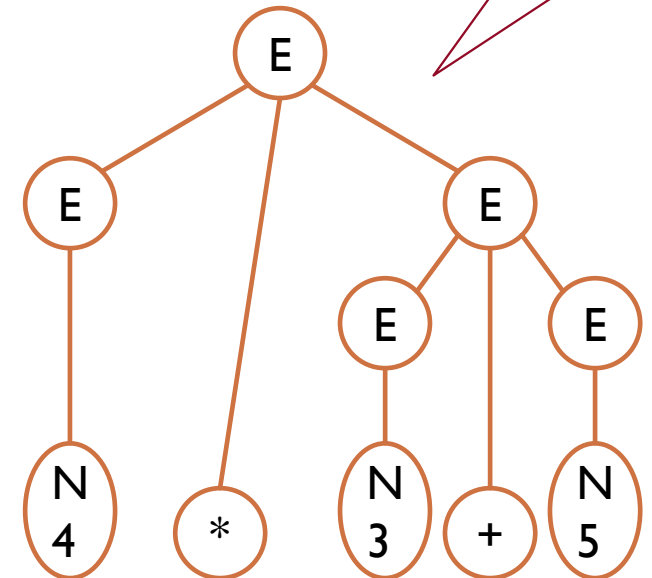
- Csak a fontos elemek megtartása

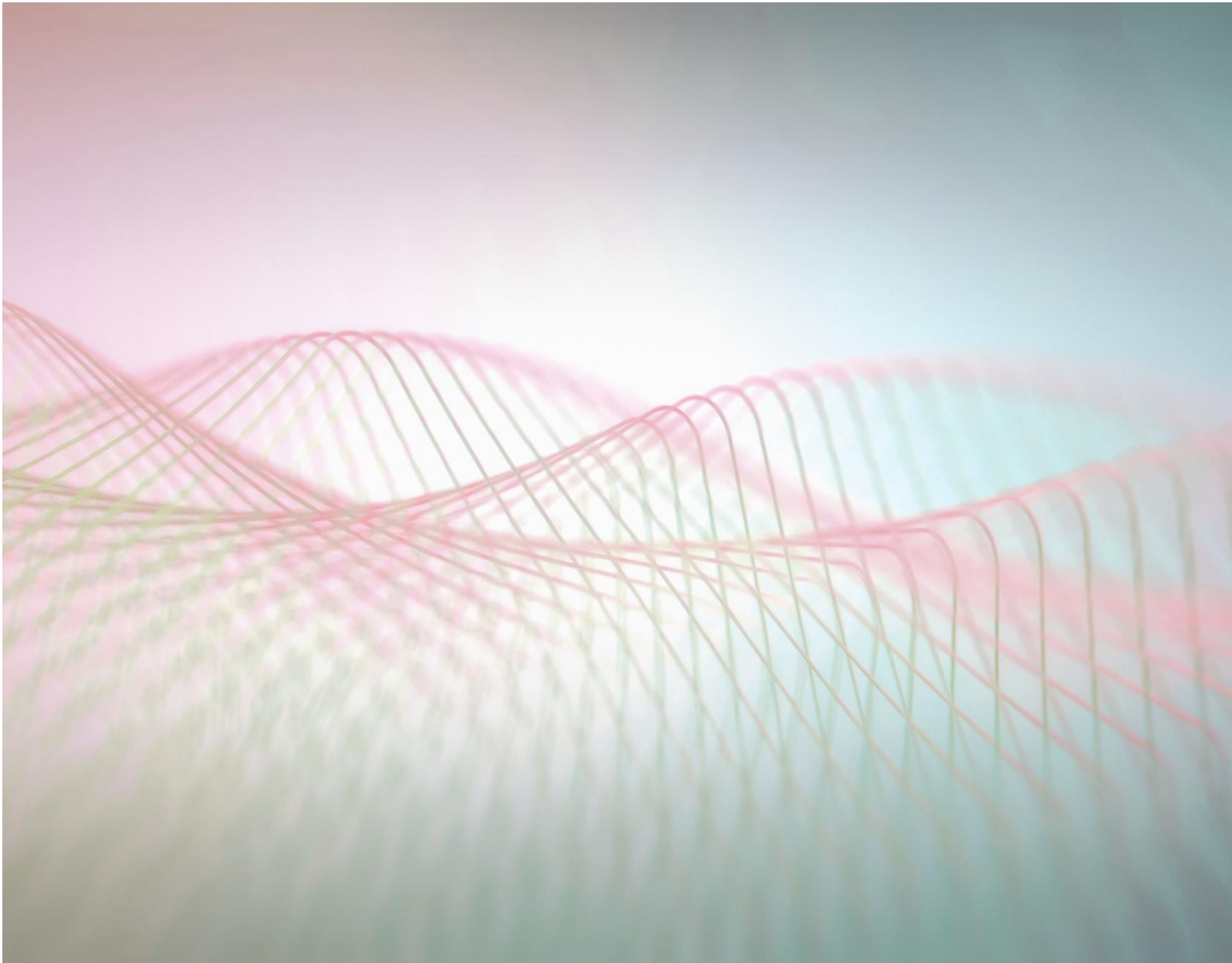
■ $E ::= E '+' E \mid E '*' E \mid N$

Konkrét
szintaxisfa



Absztrakt
szintaxisfa

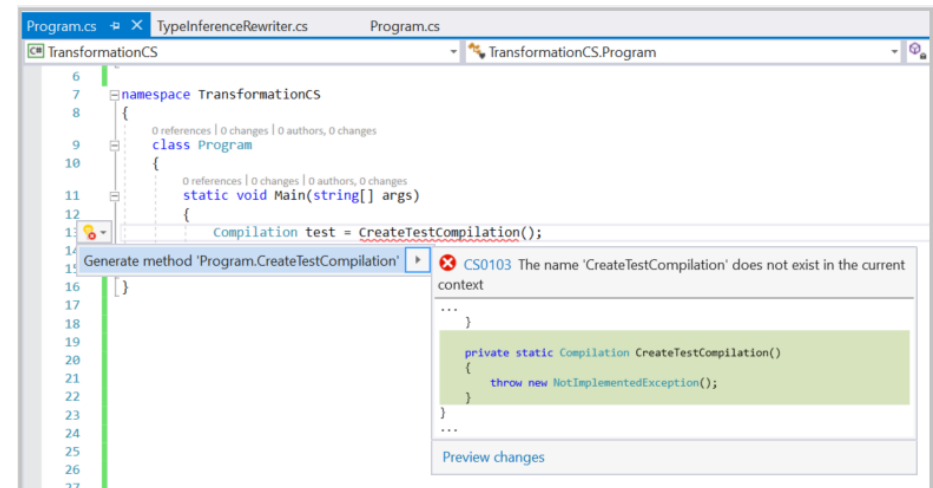




A MÁSODIK GYAKORLAT

A KÖVETKEZŐ RÉSZ TARTALMÁBÓL...

- Témakör: Fordítók a gyakorlatban
- Compiler as a Service – Roslyn
 - A fordító és szerkesztő kiterjesztése Visual Studio-ban
 - Hogyan reagálhatunk a fordítási folyamatra?
- Projekciós szerkesztők – MPS
 - DSL tervezés – hogyan is kezdődik ez?
 - Led szalag vezérlése projekciós editorral

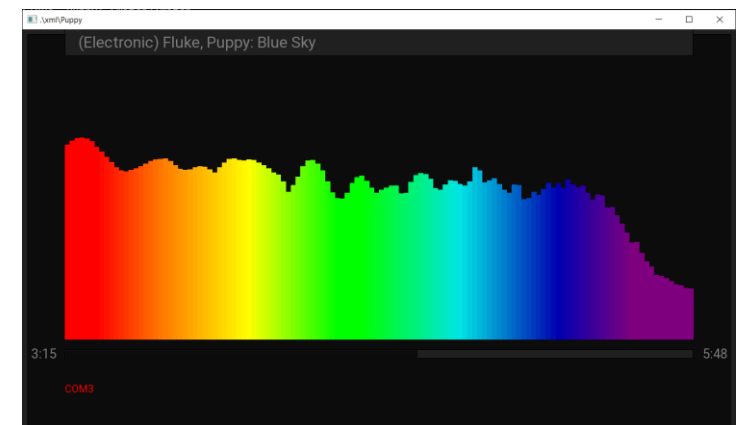
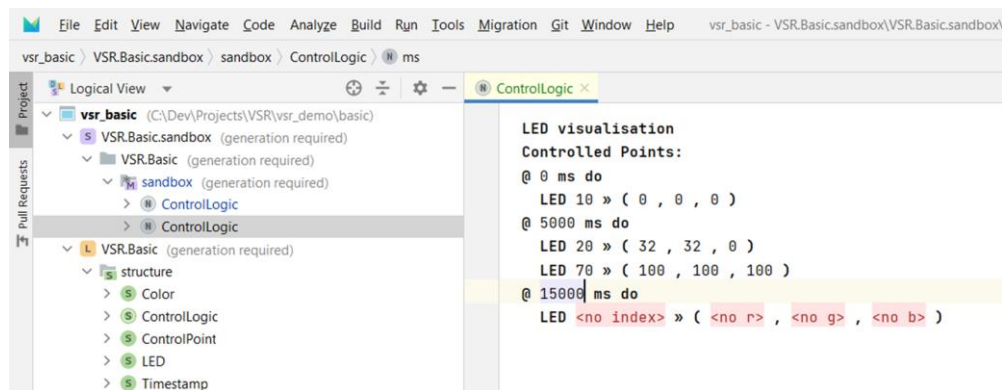


```
6 |  
7 | namespace TransformationCS  
8 | {  
9 |     0 references | 0 changes | 0 authors, 0 changes  
10 |     class Program  
11 |     {  
12 |         0 references | 0 changes | 0 authors, 0 changes  
13 |         static void Main(string[] args)  
14 |         {  
15 |             Compilation test = CreateTestCompilation();  
16 |         }  
17 |     }  
18 | }  
19 |  
20 |  
21 |  
22 |  
23 |  
24 |  
25 |  
26 |  
27 |
```

Generate method 'Program.CreateTestCompilation' CS0103 The name 'CreateTestCompilation' does not exist in the current context

```
private static Compilation CreateTestCompilation()  
{  
    throw new NotImplementedException();  
}
```

Preview changes





KÖSZÖNÖM A FIGYELMET!