



# MODELLALAPÚ SZOFTVERFEJLESZTÉS

## II. ELŐADÁS SZÖVEGES NYELVEK

DR. SOMOGYI FERENC

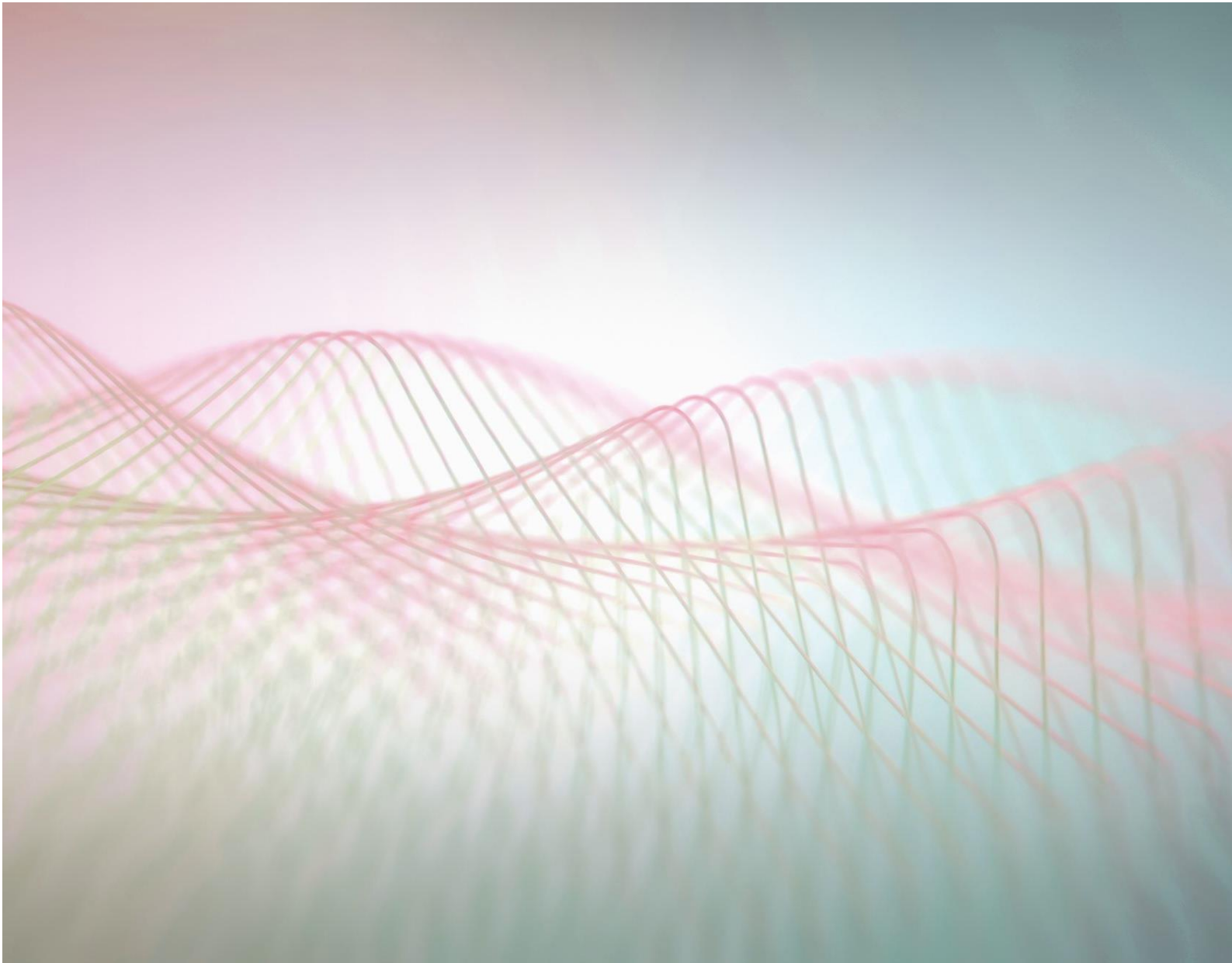
# A MAI ELŐADÁS

**I. Szöveges nyelvek**

**II. Interpreter és compiler**

**III. Editor támogatás**





# MILYEN NYELVEKET ISMERTEK?

Nem csak programozási nyelveket!

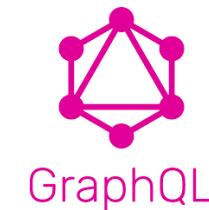
# SZÖVEGES NYELVEK

- 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?
- Editor support – mindkét típusnál fontos!



# 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
Platformfüggetlenség támogatott	Platformfüggő (általában)

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

# PROGRAMOZÁSI PARADIGMÁK (ÁLTALÁNOS CÉLÚ NYELVEKNÉL)

## ■ Imperatív megközelítés

- Procedurális (pl. COBOL, C)
- Objektum-orientált (pl. C#, Java, Kotlin)
- stb.

```
public int fib(int n) {  
    if (n <= 1) return n;  
    return fib(n - 1) + fib(n - 2);  
}
```

*OO (procedurális) - Java*

## ■ Deklaratív megközelítés

- Funkcionális (pl. Erlang, Haskell)
- Logikai (pl. Prolog)
- stb.

```
fib(N) -> fib_iter(N, 0, 1).  
  
fib_iter(0, Result, _Next) -> Result;  
  
fib_iter(Iter, Result, Next) ->  
    fib_iter(Iter-1, Next, Result+Next).
```

*Funkcionális - Erlang*



# PROGRAMOZÁSI PARADIGMÁK (ÁLTALÁNOS CÉLÚ NYELVEKNÉL)

## ■ Imperatív megközelítés

- Procedurális (pl. COBOL, C)
- Objektum-orientált (pl. C#, Java, Kotlin)
- stb.

## ■ Deklaratív megközelítés

- Funkcionális (pl. Erlang, Haskell)
- Logikai (pl. Prolog)
- stb.

```
fib(0, 1) :- !.  
fib(N, F) :- fib(1, N, 1, 1, F).  
  
fib(N, N, _, F, F) :- !.  
fib(N0, N, F0, F1, F) :-  
    N1 is N0 + 1,  
    F2 is F0 + F1,  
    fib(N1, N, F1, F2, F).
```

*Logikai - Prolog*

# SZAKTERÜLETI NYELVEK PROGRAMOZÁSI PARADIGMA SZERINT

- Hasonlóan az általános célú nyelvekhez, itt is vannak külön kategóriák
- Imperatív DSL
  - A végrehajtás módját adjuk meg
  - Pl. LOGO, egy tetszőleges vállalati workflow-t leíró nyelv
- Deklaratív DSL
  - A végrehajtás módja helyett a kívánt eredmény a fontos
  - Pl. HTML, SQL, XAML

# 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, XML

# ESETTANULMÁNY – A PICTUREPROCESSOR NYELV

- Képeket szeretnék feldolgozni (több százat)
  - Meg akarom adni a képfájl nevét és opciókat a feldolgozáshoz
    - Az eredmény felbontása (maximális méret)
    - Kimenet formátuma (pl. jpg) és esetleges opciói (pl. 95%-os minőség)
    - Kimenet fájl neve (ha nincs megadva, felülírja az eredetit)
  - Egyszerre sok fájlt akarok feldolgozni

```
Process DSC3456.jpg -size =1024 -out=JPG -outfile = DSC3456s.jpg
Process DSC3457.jpg -size =1024 -out=JPG -outfile = DSC3457s.jpg
...
Process DSC3465.jpg -size =1024 -out=JPG -outfile = DSC3465s.jpg
Process DSC3466.jpg -size =1600 -out=PNG -outfile = DSC3465s.jpg
```

# ESETTANULMÁNY – A PICTUREPROCESSOR NYELV

- Ne kelljen egyenként felsorolni a fájlneveket!
  - Legyen *From* és *To* paraméter, a feldolgozó találja ki a közbenső fájlneveket, ha azokban csak egy szám különbség van
- Szeretném az opciókat összevonni (ahol lehet)
  - A kimeneti fájlnévénél \$\$\$ jelentse az eredeti fájl nevét
  - Továbbra is tudjak egyéni opciókat is megadni!

```
Process -from=DSC3456.jpg -to=DSC3465.jpg -size=1024 -out=JPG  
-outfile=$$$s.jpg
```

```
Process DSC3466.jpg -size=1600 -out=PNG -outfile=DSC3466s.jpg
```

# ESETTANULMÁNY – A PICTUREPROCESSOR NYELV

- Később felmerülő igények
  - Lehesse egyéni, szöveges, vagy grafikus vízjelet tenni a képekre!
  - Lehesse megadni, hogy alkalmazzon filtereket a képeken, ha szükséges (pl. fényerő állítás, hisztogram információk alapján)!
  - Tudjon több képből indexképet (mozaik képet) készíteni!
  - Az EXIF információk alapján másolja a képeket különböző könyvtárakba!
  - A különböző effektek sorrendjét lehesse megadni!
  - Lehesse elágazásokat definiálni – pl. kis képre jobb minőségű kódolás ideális

# ESETTANULMÁNY – A PICTUREPROCESSOR NYELV

- Gondoljuk végig mi történt!
  - Volt egy kezdetben egyszerű, később bővülő, speciális igényünk (kötegetelt képfeldolgozás)!
  - Az igényhez kapcsolódó opciókat meg akartuk adni, amennyire lehet egyszerűen
  - Nem akartunk megtanulni programozni
  - Ezért...

Készítettünk egy szöveges szakterületi nyelvet!



# A MAI ELŐADÁS

**I. Szöveges nyelvek**

**II. Interpreter és compiler**

**III. Editor támogatás**



# XML, MINT MODELLEZŐ NYELV

- Gyakori megoldás
  - Konfigurációs fájlok
  - Platformfüggetlen tárolás
- XML Schema Definition (XSD)
  - XML dokumentum struktúrája
    - tag, attribútum, tartalmazás,
    - számosság, sorrend, adat típus,
    - default érték, stb.
  - W3C ajánlás
- Programkönyvtárak

```
<book category="COOKING">  
  <title lang="en">Everyday Italian</title>  
  <author>Giada De Laurentiis</author>  
  <year>2005</year>  
  <price>30.00</price>  
</book>
```

# JSON, MINT MODELLEZŐ NYELV

- Nagy népszerűség
- Rengeteg alkalmazási terület
- Széles támogatottság
- JSON Schema

```
"image": {  
  "src": "Images/Sun.png",  
  "name": "sun1",  
  "hOffset": 250,  
  "vOffset": 250,  
  "alignment": "center"  
}
```

# 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 (de: instrumentált kódgenerálás)
- Pl. C, C++, C#, Kotlin, Pascal

# INTERPRETER VS. COMPILER

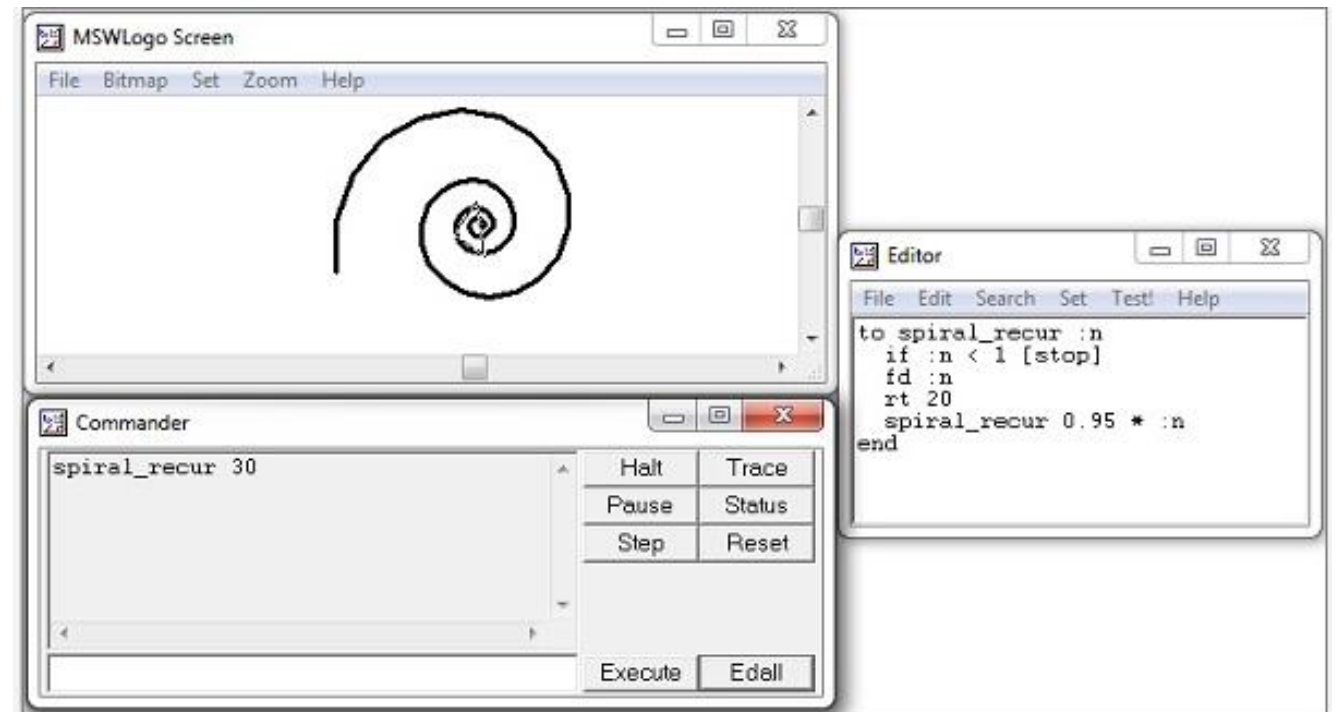
- Just-In-Time (JIT) compiler
  - Nem végrehajtás előtt, hanem végrehajtás közben végzi a fordítást
  - A forráskódból előállított köztes kód továbbfordítása – language interoperability
  - Gyorsabb, mint az interpretálás, de lassabb, mint a teljes fordítás
  - Kisebb memóriaigény, dinamikus futásidejű 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

- 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 – PÉLDA

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

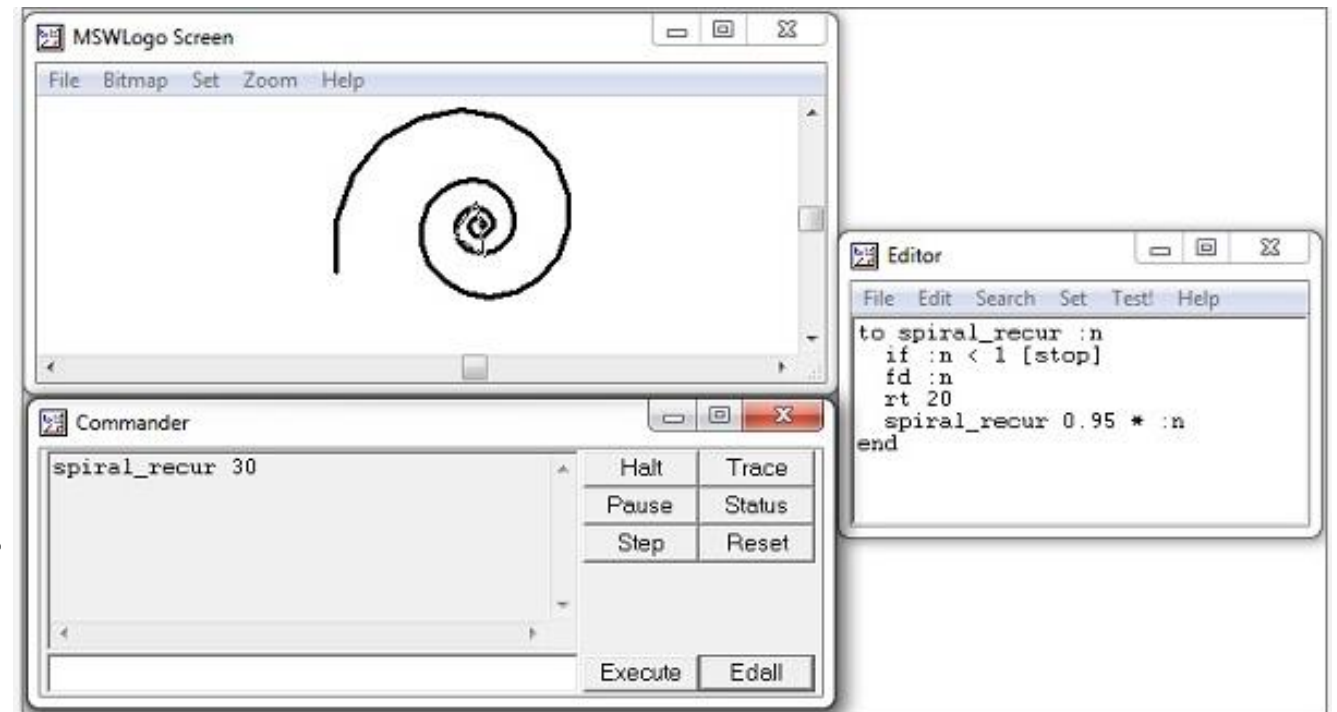


Forrás: [https://www.tutorialspoint.com/logo/logo\\_quick\\_guide.htm](https://www.tutorialspoint.com/logo/logo_quick_guide.htm)



# INTERPRETER VS. COMPILER – PÉLDA

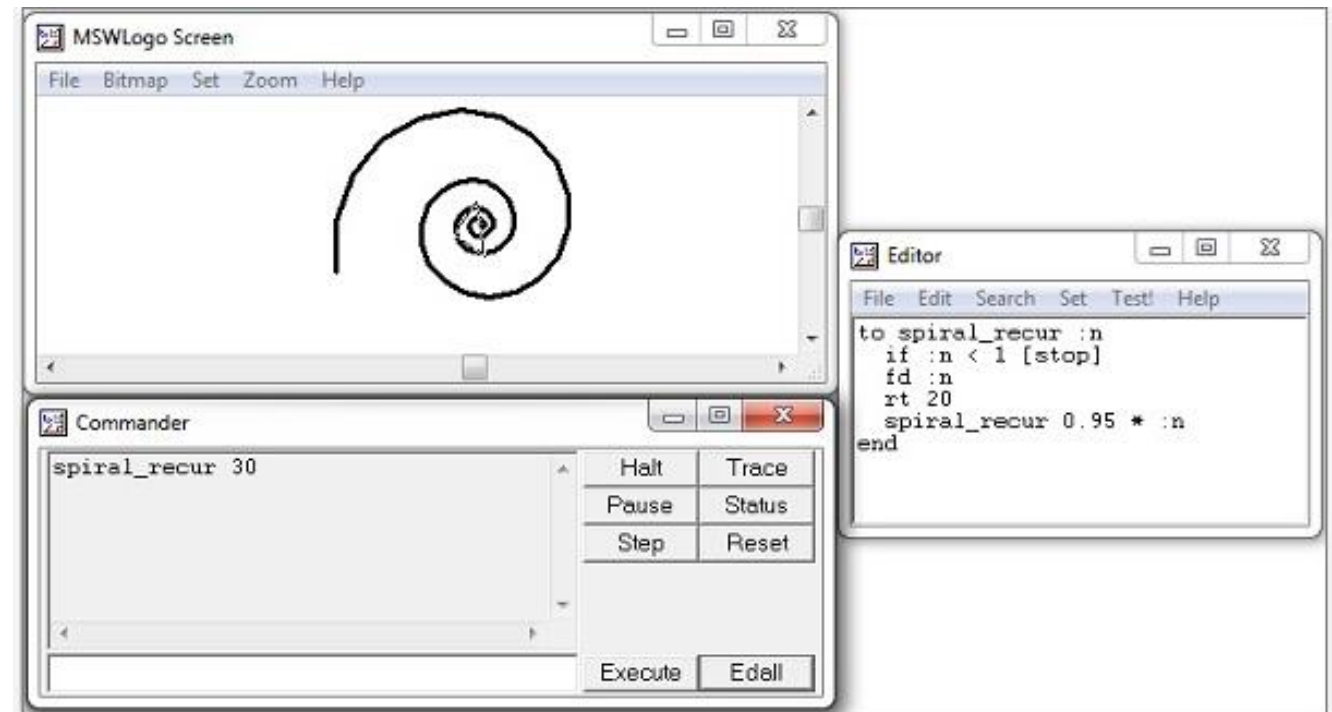
- Interpretált Logo
  - Utasításonként dolgozzuk fel
  - Tetszőleges programnyelven
    - Saját interpretert írunk
    - Pl. Java (de bármi más lehet)
  - Végrehajtjuk az utasításokat
    - Pl. “fd :n” → Java Canvas API hívás



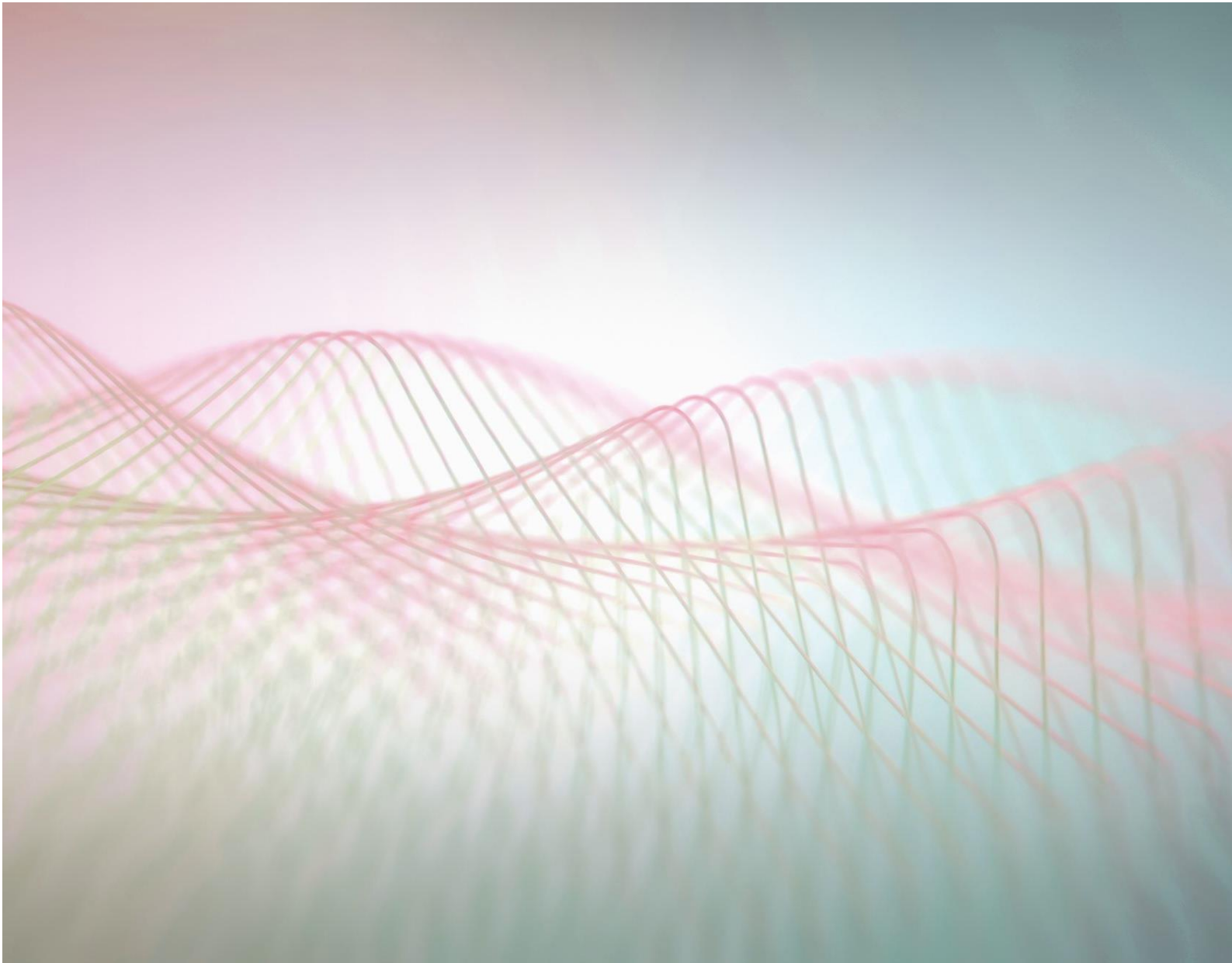
Forrás: [https://www.tutorialspoint.com/logo/logo\\_quick\\_guide.htm](https://www.tutorialspoint.com/logo/logo_quick_guide.htm)

# 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

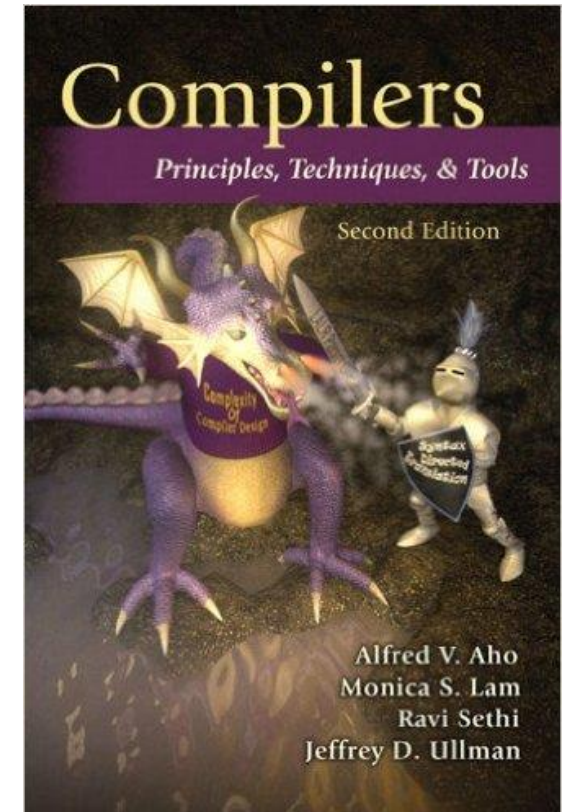


Forrás: [https://www.tutorialspoint.com/logo/logo\\_quick\\_guide.htm](https://www.tutorialspoint.com/logo/logo_quick_guide.htm)



# 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;  
}
```

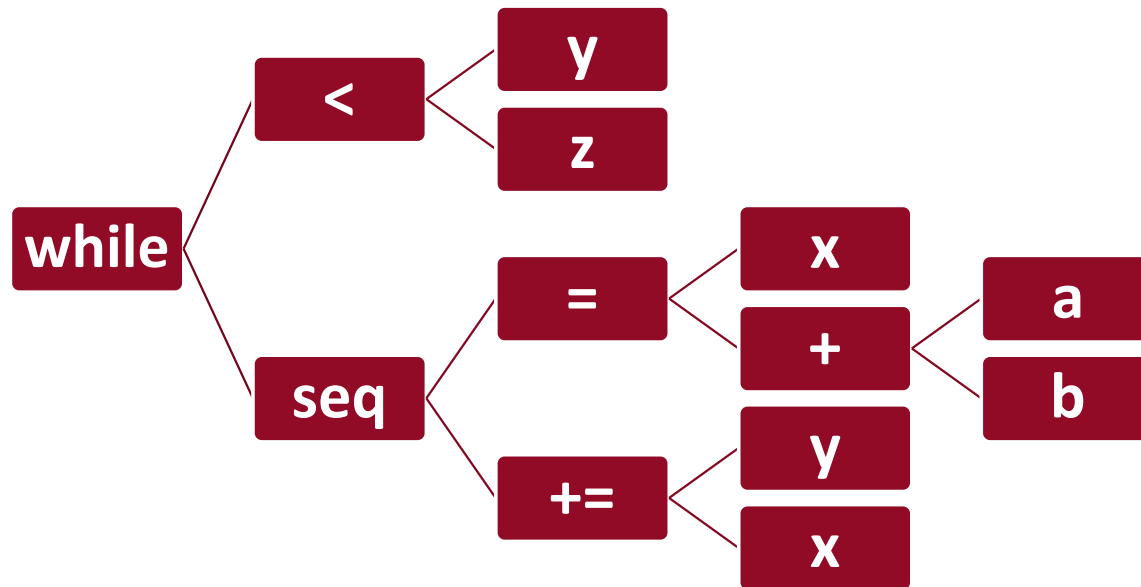
# I. FÁZIS – LEXIKAI ELEMZÉS

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





## II. FÁZIS – SZINTAKTIKAI ELEMZÉS



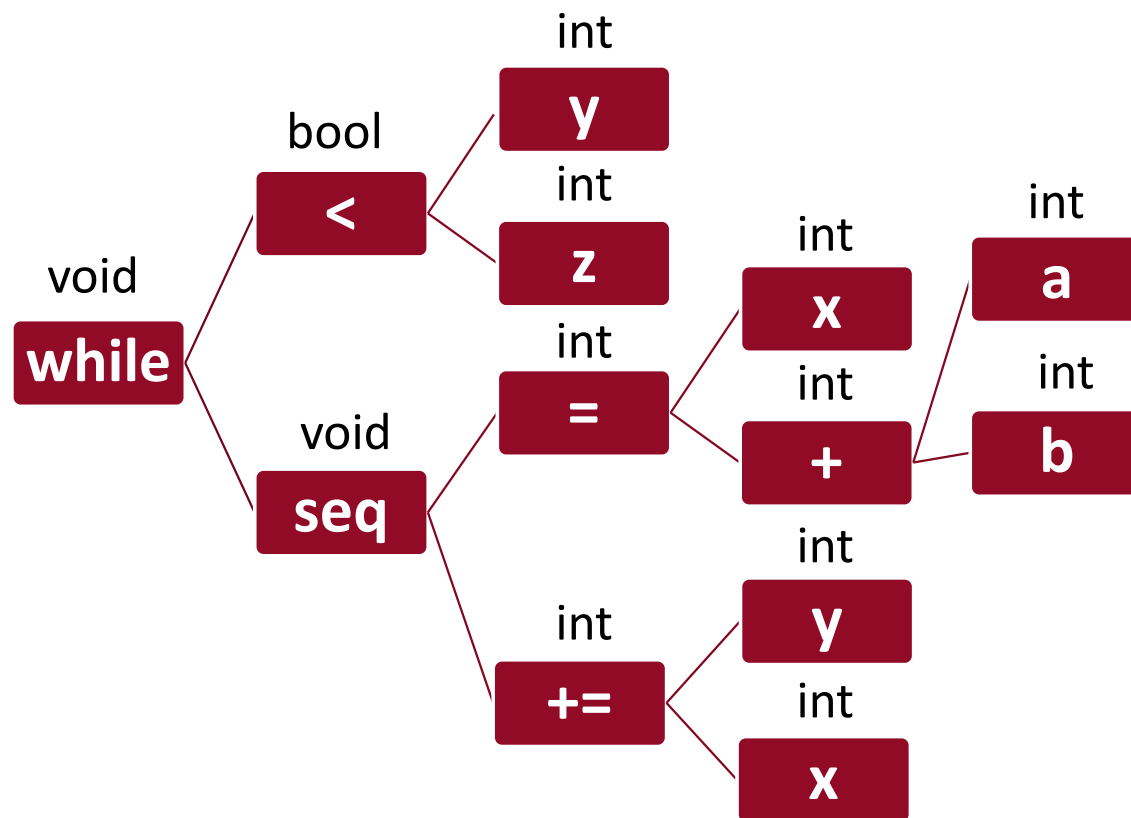
```
while (y < z) {  
    x = a + b;  
    y += x;  
}
```

## 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



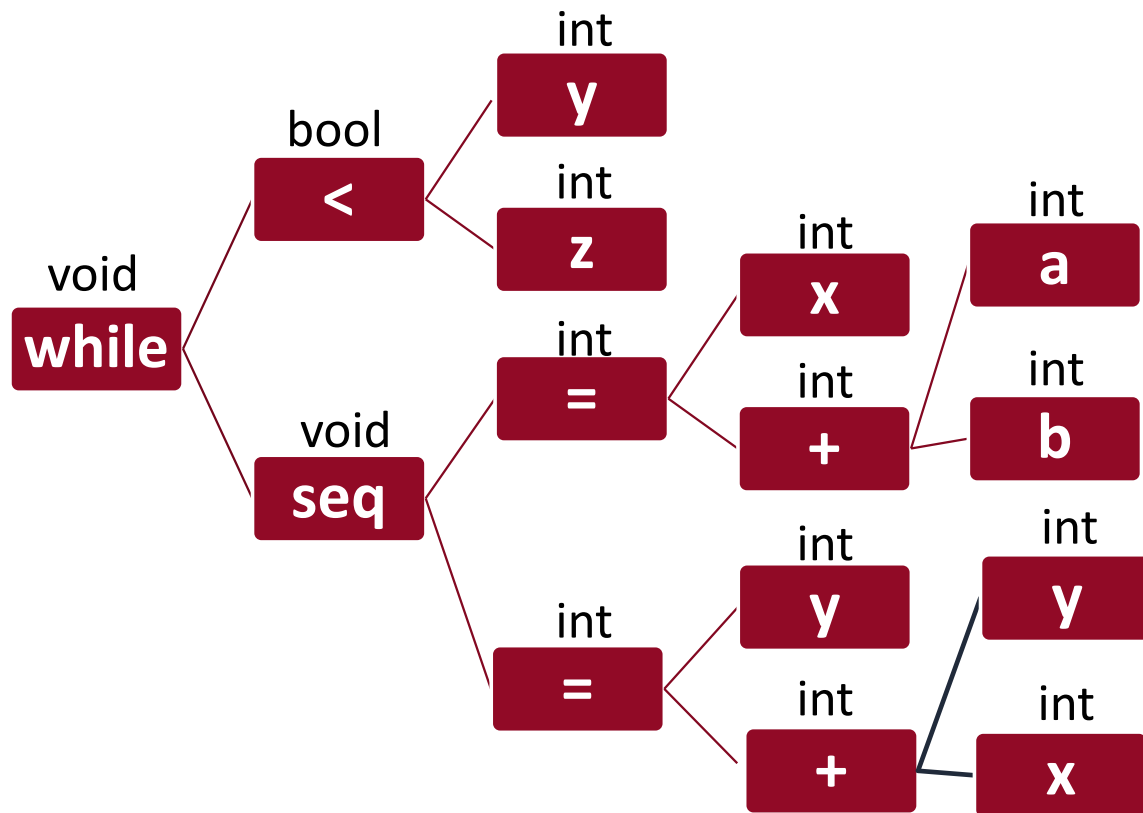
```
while (y < z) {
    x = a + b;
    y += x;
}
```

### 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



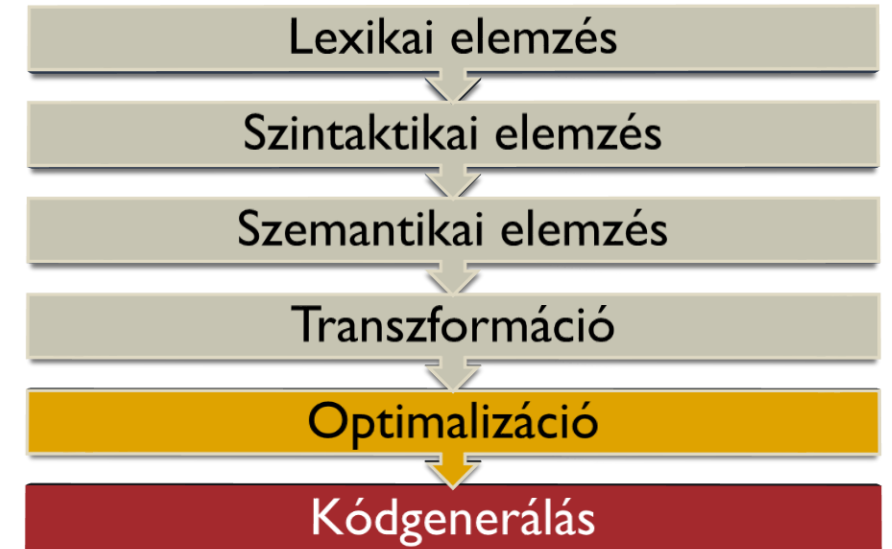
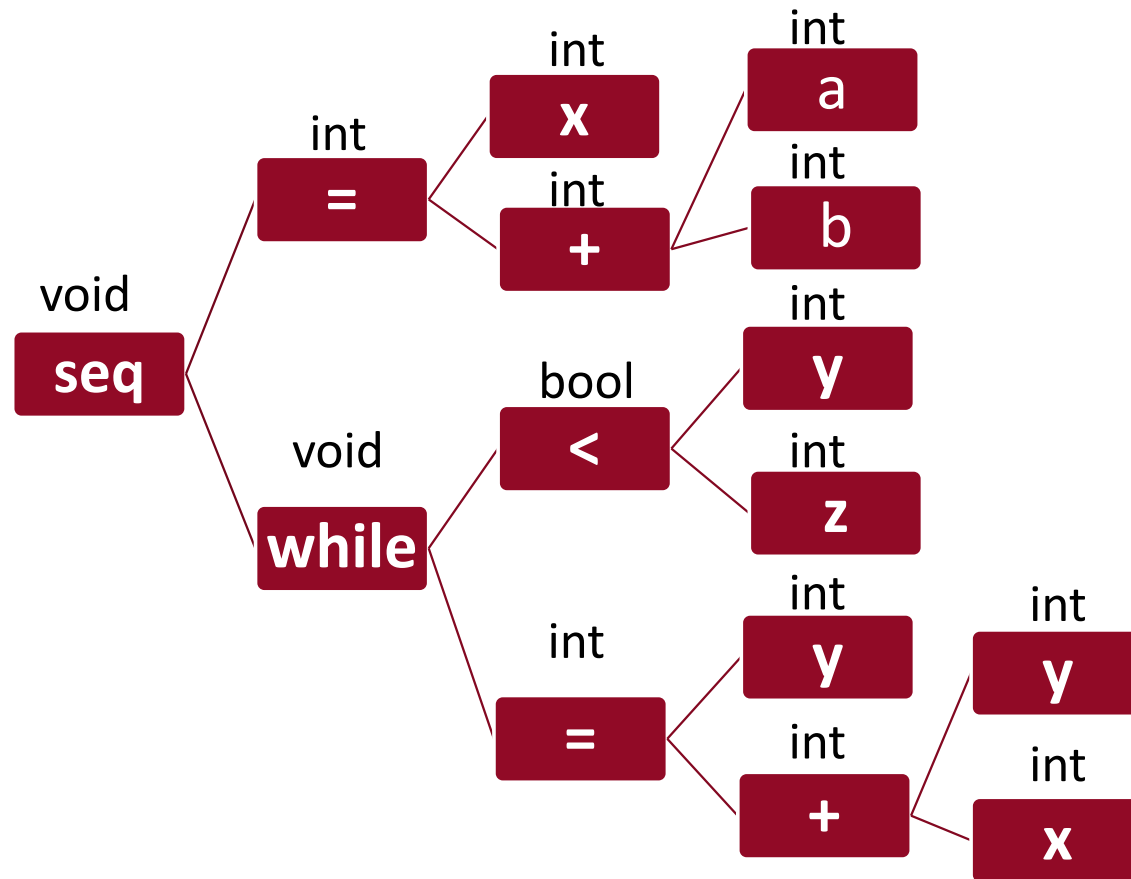
```
while (y < z) {
    x = a + b;
    y += x;
}
```

## IV. FÁZIS – TRANSZFORMÁCIÓ

- Opcionális
- Köztes nyelvre való leképezés
  - Könnyebben kezelhető pl. optimalizáció során
  - Pl. Common Intermediate Language, Java Bytecode, vagy ezek előtti köztes reprezentációk
- Műveletek leképezése egységes formára
  - Pl.  $y += x \rightarrow y = y + x$



# V. FÁZIS – OPTIMALIZÁCIÓ

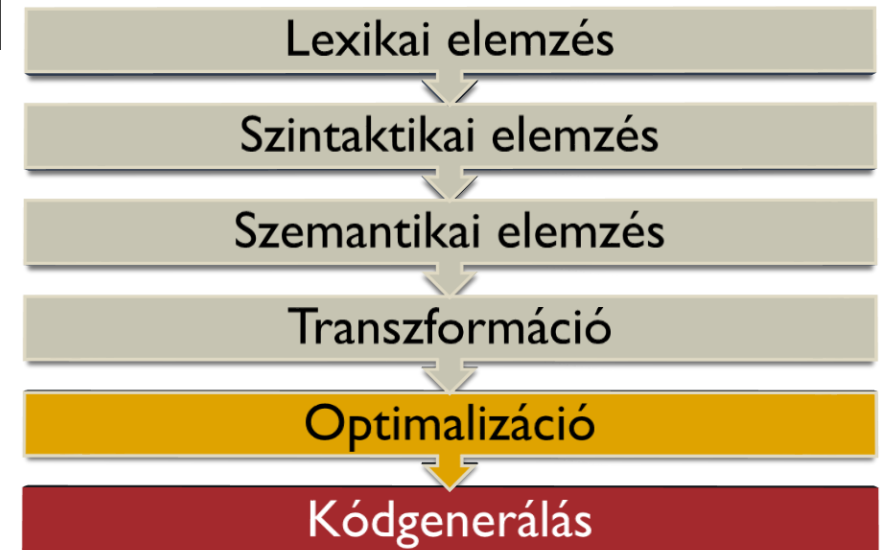


```
while (y < z) {
    x = a + b;
    y += x;
}
```



## V. FÁZIS – OPTIMALIZÁCIÓ

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

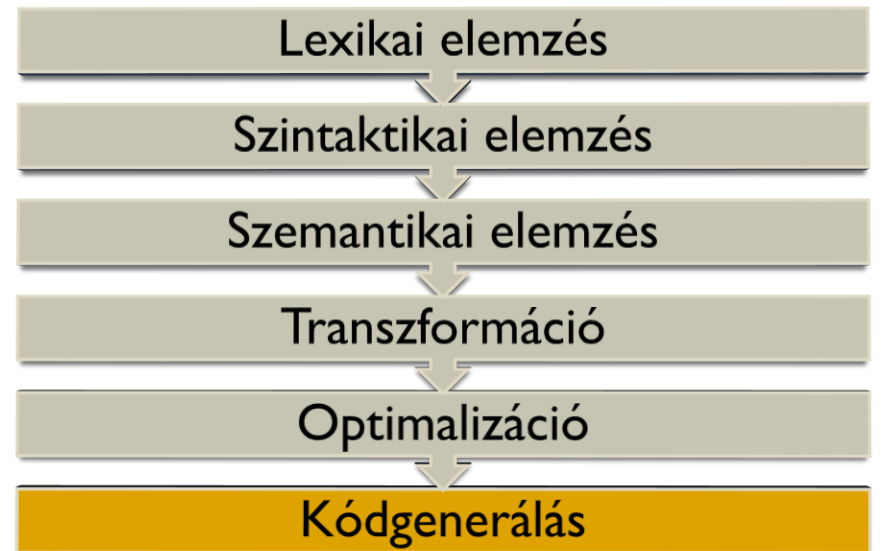


## VI. FÁZIS – KÓDGENERÁLÁS

```
add    $1, $2, $3
loop:  add $4, $1, $4
slt     $6, $1, $5
beq     $6, loop
```

-----

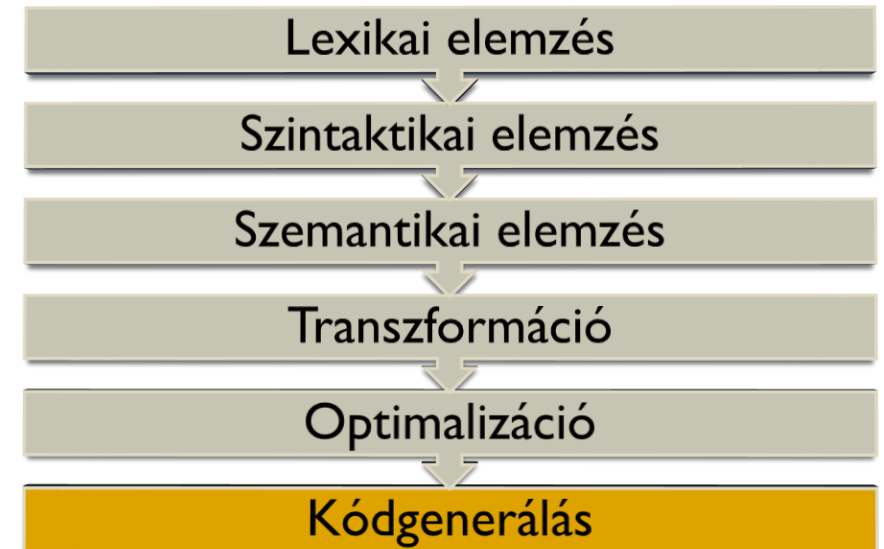
```
x := a + b;
while y < z do
    y := y + x;
```



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

## VI. FÁZIS – KÓDGENERÁLÁS

- Kód előállítás
  - 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. Interpreter és compiler**

**III. Editor támogatás**



# SZÖVEGES SZERKESZTŐK (EDITOROK)

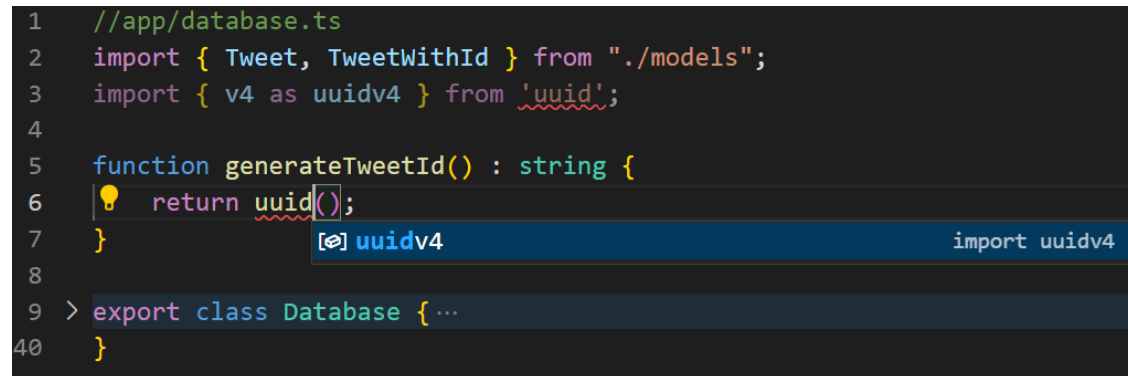
- Szöveges nyelvek – jó tool support
  - Írhatnánk Notepad-ben is, de nem ajánlott 😊
  - Vannak kiforrottabb szövegszerkesztők (Pl. Vim, Emacs, Atom)
    - De ezeket összekötni a nyelvünkkel nem túl egyszerű
  - Fejlesztői környezetek (IDE)  
(Pl. VSCode, Eclipse, Eclipse Theia, IntelliJ IDEA, NetBeans)
    - Ezek közül több kiterjeszthető (plugin, extension, stb.), ami megkönnyíti az összekötést
- Nyelv és editor összekötése
  - A fordítás fázisaival összhangban (elsősorban szemantikai elemzésnél)



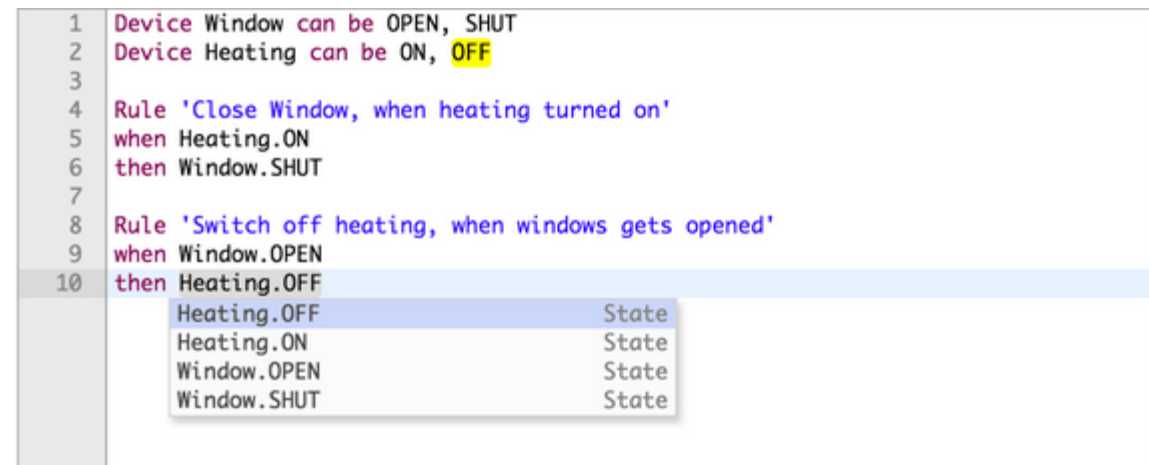
# SZÖVEGES SZERKESZTŐK (EDITOROK)

- Editor funkciók
  - Syntax highlighting
  - Code completion
  - Hibák jelzése
  - Refaktorálás támogatása
  - Folding
  - Quick fixek
  - Stb.

```
1 //app/database.ts
2 import { Tweet, TweetWithId } from "../models";
3 import { v4 as uuidv4 } from 'uuid';
4
5 function generateTweetId() : string {
6   return uuid();
7 }
8
9 > export class Database { ...
40 }
```



```
1 Device Window can be OPEN, SHUT
2 Device Heating can be ON, OFF
3
4 Rule 'Close Window, when heating turned on'
5 when Heating.ON
6 then Window.SHUT
7
8 Rule 'Switch off heating, when windows gets opened'
9 when Window.OPEN
10 then Heating.OFF
```



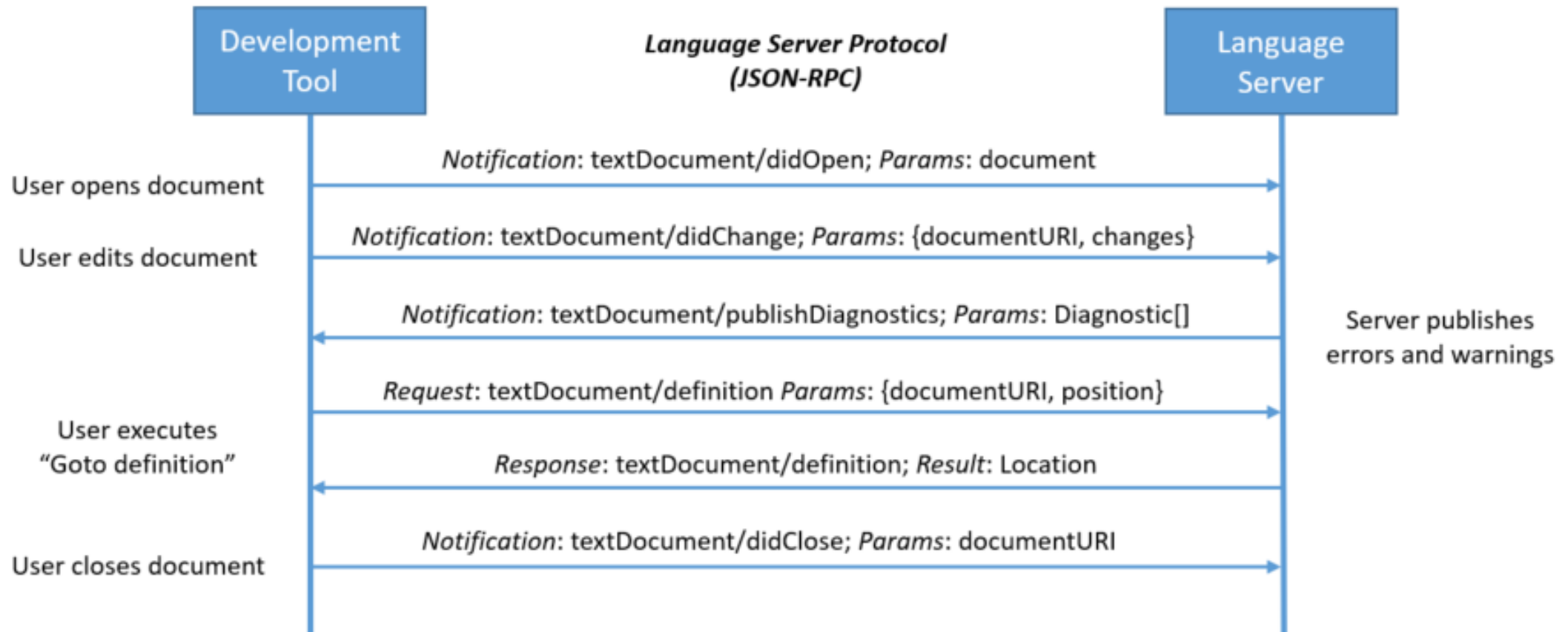
Heating.OFF	State
Heating.ON	State
Window.OPEN	State
Window.SHUT	State

Forrás: <https://www.eclipse.org/Xtext/>

# LANGUAGE SERVER PROTOCOL (LSP)

- JSON-RPC alapú protokoll
- Szöveges editorok és language serverek közötti kommunikáció
- Főbb editor funkciók támogatása
  - Syntax highlighting, hibák jelzése, code completion, refaktorálás, stb.
- Miért jó ez nekünk?
  - Egyszer kell megírni, több editorral is összeköthető
  - Sok népszerű editor támogatja (Pl.VSCode, Monaco, Eclipse IDE, Eclipse Theia)
  - Sok nyelvhez van létező Language Server implementáció
    - <https://langserver.org/>

# LANGUAGE SERVER PROTOCOL (LSP)



Forrás <https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/>

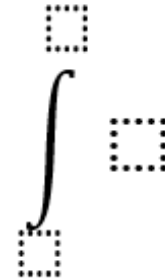


# PROJEKCIÓS EDITOROK

- Hagyományos editorok
  - Amikről eddig szó volt
  - Bevált, sok helyen használják
  - Bármilyen szövegszerkesztő használható (ajánlott editor funkciókkal)
- Projekciós editorok
  - A szintaxisfa elemei közvetlenül vizualizálva vannak
  - *Nem kell szintaxisfát építeni!*
  - Speciális tooling kell hozzá
  - Pl. JetBrains MPS, Gentleman

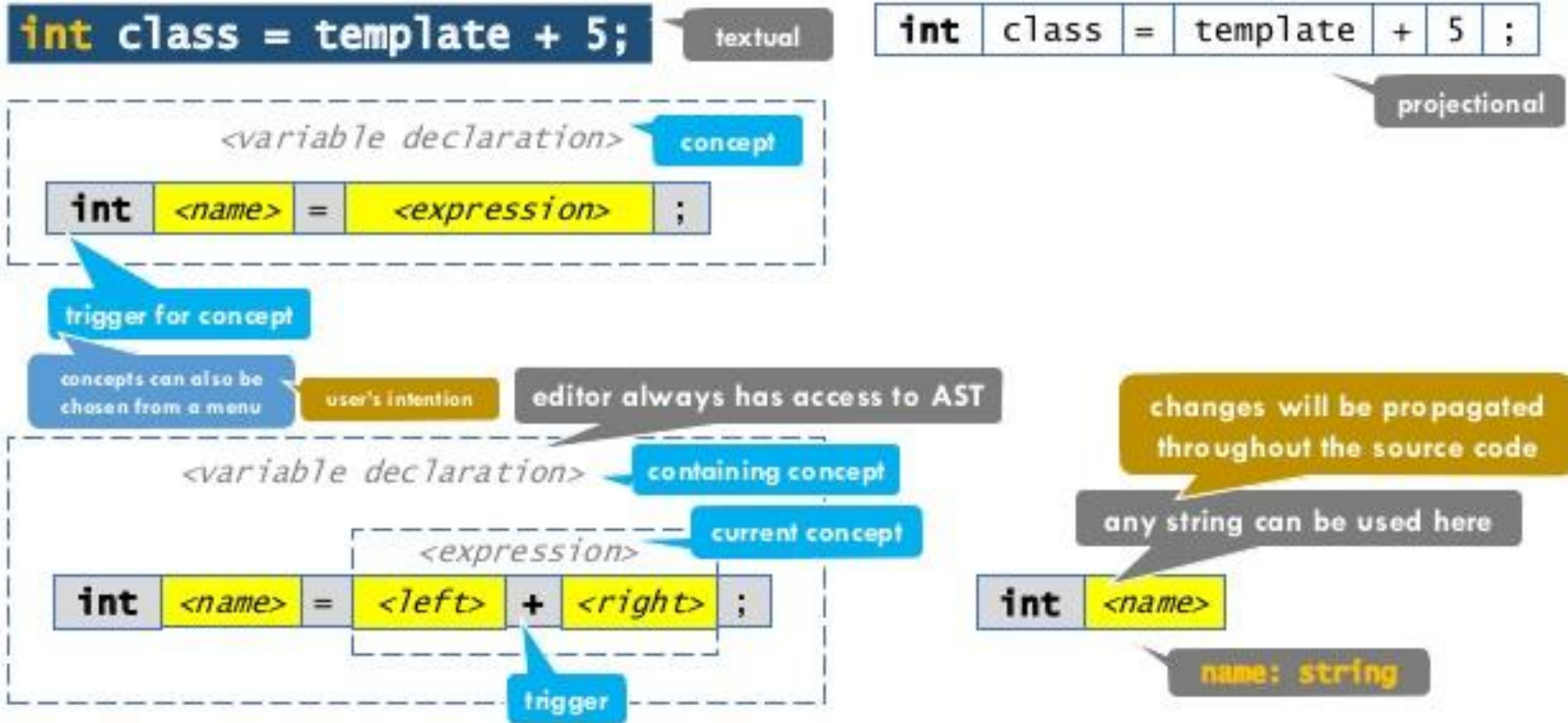
# PROJEKCIÓS EDITOROK – PÉLDA

- Szemantikus modell
  - intervallum alsó, felső határa
  - integrálandó kifejezés
- Konkrét kifejezés:  $\int_{-\infty}^{\infty} e^{-x^2} dx$
- Közvetlenül leképezhető a szemantikus modellre



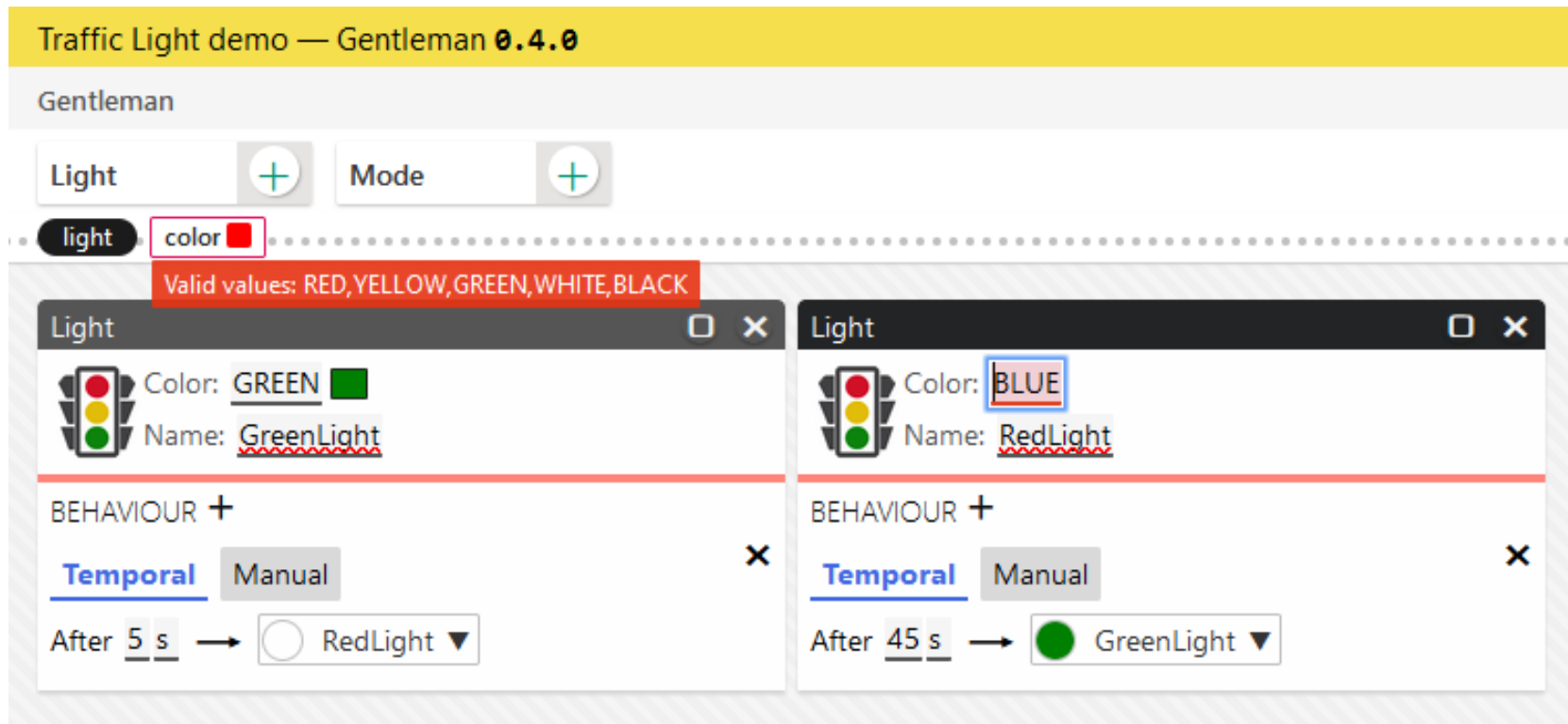
# PROJEKCIÓS EDITOROK – JETBRAINS MPS

## Parsing vs. projectional editing



Forrás: Mikhail Barash: Reflections on teaching JetBrains MPS within a university course

# PROJEKCIÓS EDITOROK – GENTLEMAN



Forrás: <https://geodes.iro.umontreal.ca/gentleman/demo/traffic-light/index.html>



KÖSZÖNÖM A FIGYELMET!