



# **VISUALCOMP: EDUCATIONAL COMPILER WITH VISUALIZATION IN JAVA**

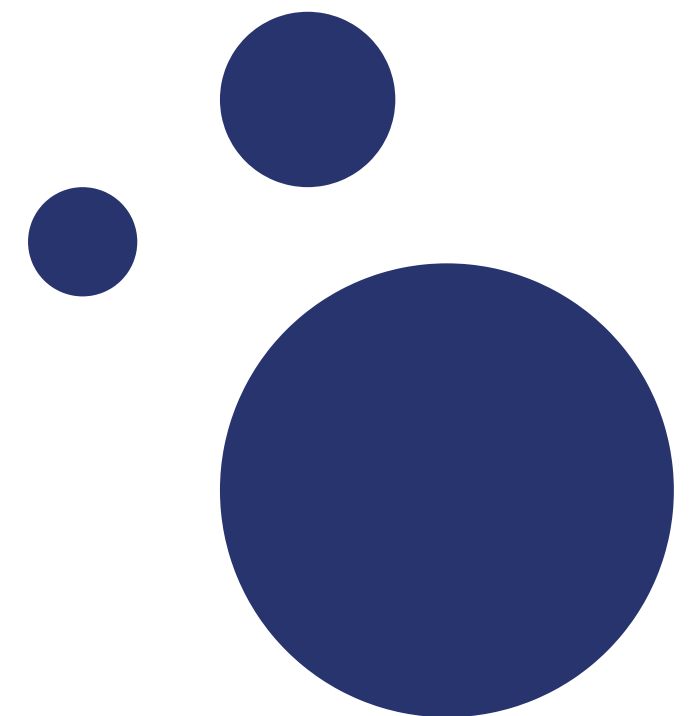
*Juan Bustos - 20221020114*

*Maria Restrepo - 20221020044*

*COMPUTER SCIENCE III*

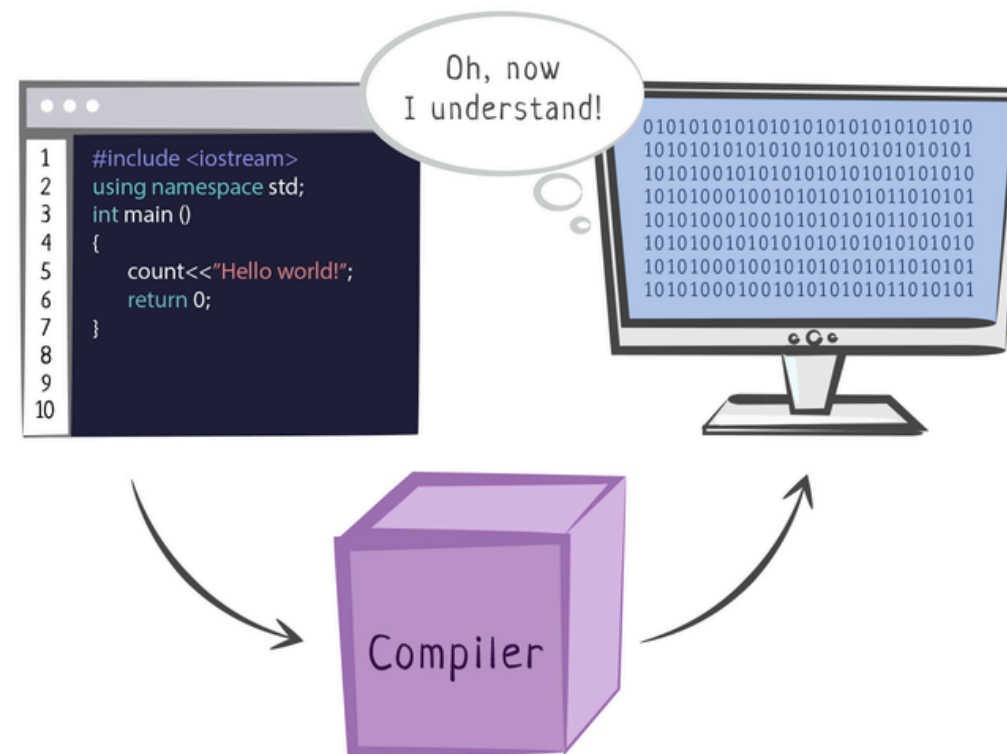
# OUTLINE

• Introduction	<b>01</b>
• Project Goal	<b>02</b>
• System Architecture	<b>03</b>
• User Interface	<b>04</b>
• Language Design	<b>05</b>
• Grammar Rules	<b>06</b>
• Error Handling	<b>07</b>
• Achieved Result	<b>08</b>
• Conclusions	<b>09</b>
• References	<b>10</b>



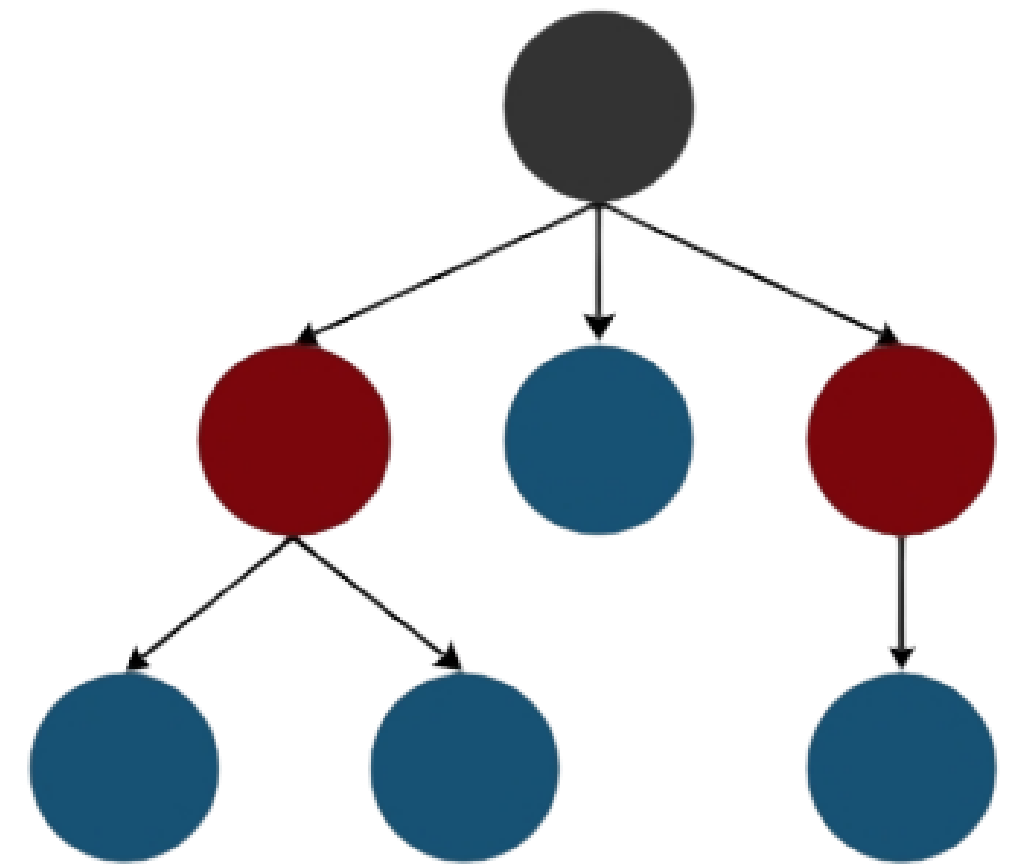
# INTRODUCTION

- Compiler design is abstract and hard to visualize.
- Industrial tools hide inner steps and overwhelm beginners.
- Students often struggle to understand compilation phases.



# PROJECT GOAL

- Develop an educational compiler in Java.
- Supports input in a simplified programming language.
- Performs lexical, syntactical, and semantic analysis.
- The derivation tree is displayed after compilation.



# SYSTEM ARCHITECTURE

User Code Input



Lexical Analyzer



Syntactic Analyzer



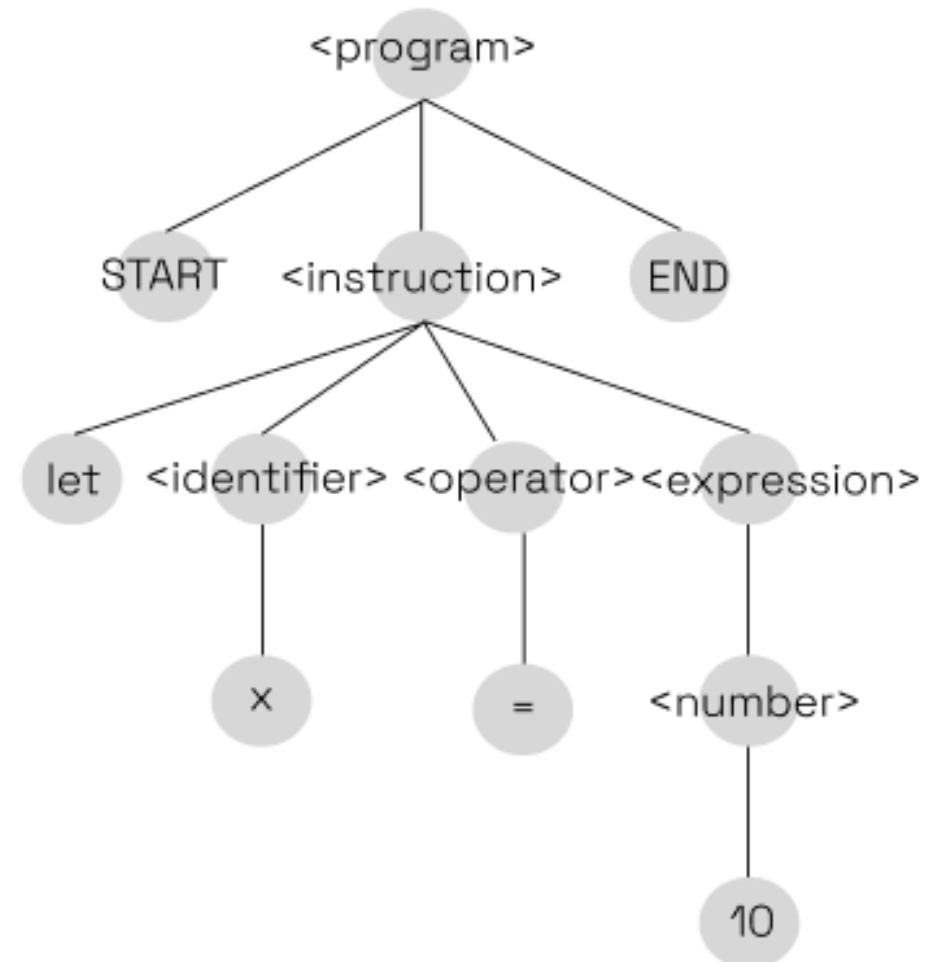
Semantic Analyzer



**Derivation Tree  
Visualization**

# USER INTERFACE

Derivation Tree Simulator



START  
let x = 10  
END

Compile

# LANGUAGE DESIGN

- Keywords: START, END, let, if, while, print
- Tokens: KEYWORD, IDENTIFIER, NUMBER, OPERATOR, PARENTHESIS, BRACES, etc.
- Grammar: defined using context-free productions.

$\Sigma = \{ \text{a-z, A-Z, 0-9, + - * / = < > ( ) \{ \} } \}$

START

let x = 10

if ( x < 5 ) {

    print x

}

END

# GRAMMAR RULES

$\langle S \rangle \rightarrow \text{START } \langle \text{Instructions} \rangle \text{ END}$

$\langle \text{Instructions} \rangle \rightarrow \langle \text{Instruction} \rangle \langle \text{Instructions} \rangle \mid \varepsilon$

$\langle \text{Instruction} \rangle \rightarrow \text{let } \langle \text{Identifier} \rangle = \langle \text{Expression} \rangle$

$\mid \text{print } \langle \text{Identifier} \rangle$

$\mid \text{if } ( \langle \text{Condition} \rangle ) \langle \text{Block} \rangle$

$\mid \text{while } ( \langle \text{Condition} \rangle ) \langle \text{Block} \rangle$

$\langle \text{Block} \rangle \rightarrow \{ \langle \text{Instructions} \rangle \}$

$\langle \text{Expression} \rangle \rightarrow \langle \text{Identifier} \rangle \langle \text{Operator} \rangle \langle \text{Expression} \rangle$

$\mid \langle \text{Number} \rangle$

$\langle \text{Condition} \rangle \rightarrow \langle \text{Expression} \rangle$

$\langle \text{Identifier} \rangle \rightarrow [a-zA-Z][a-zA-Z0-9]^*$

$\langle \text{Number} \rangle \rightarrow [0-9]^+$

$\langle \text{Operator} \rangle \rightarrow + \mid - \mid * \mid / \mid = \mid < \mid >$



# ERROR HANDLING

## Lexical Errors

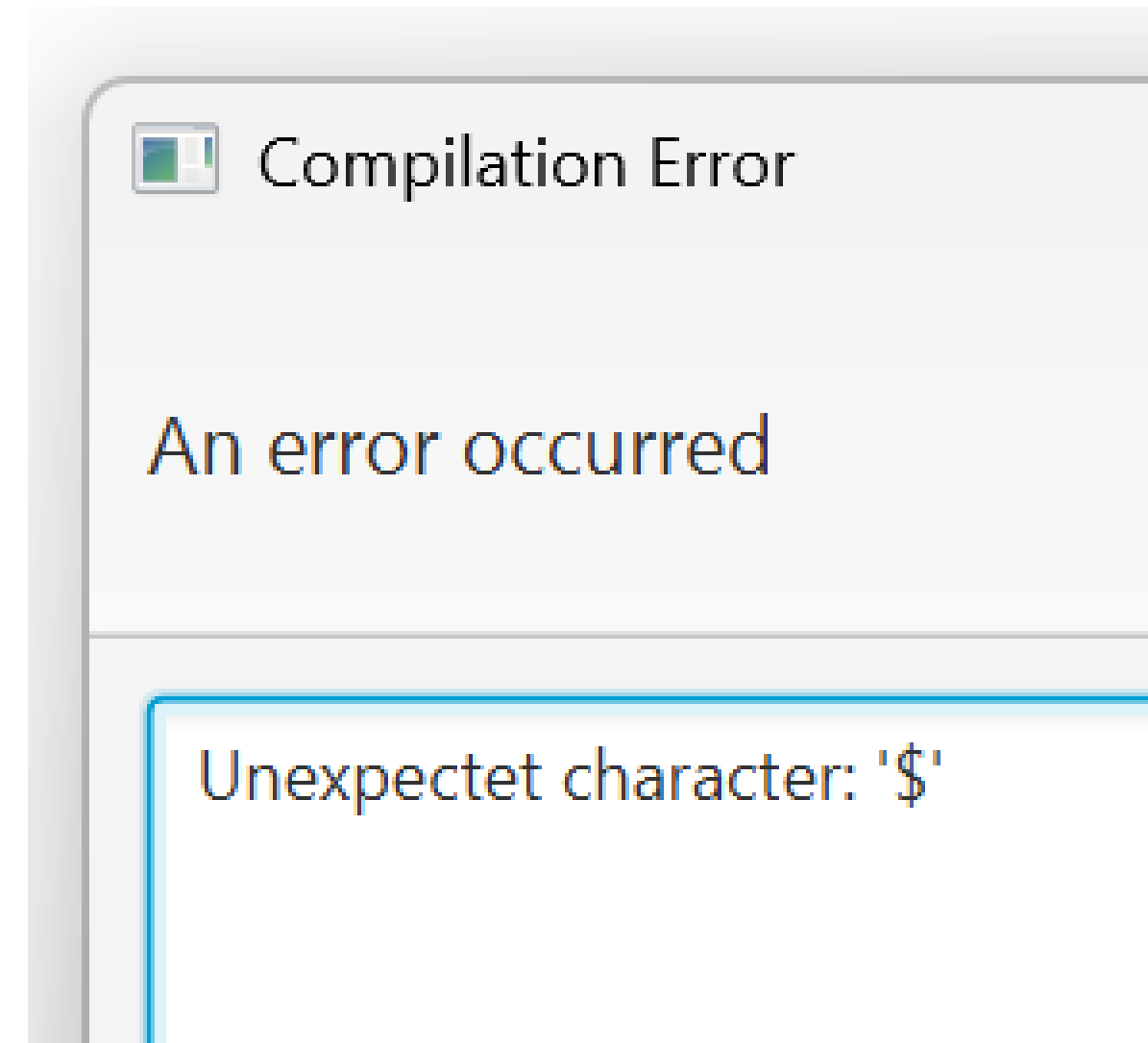
- Unexpected symbols → `let x = 10$`

## Syntactic Errors

- Structure mismatches → `while (x < 5 {`

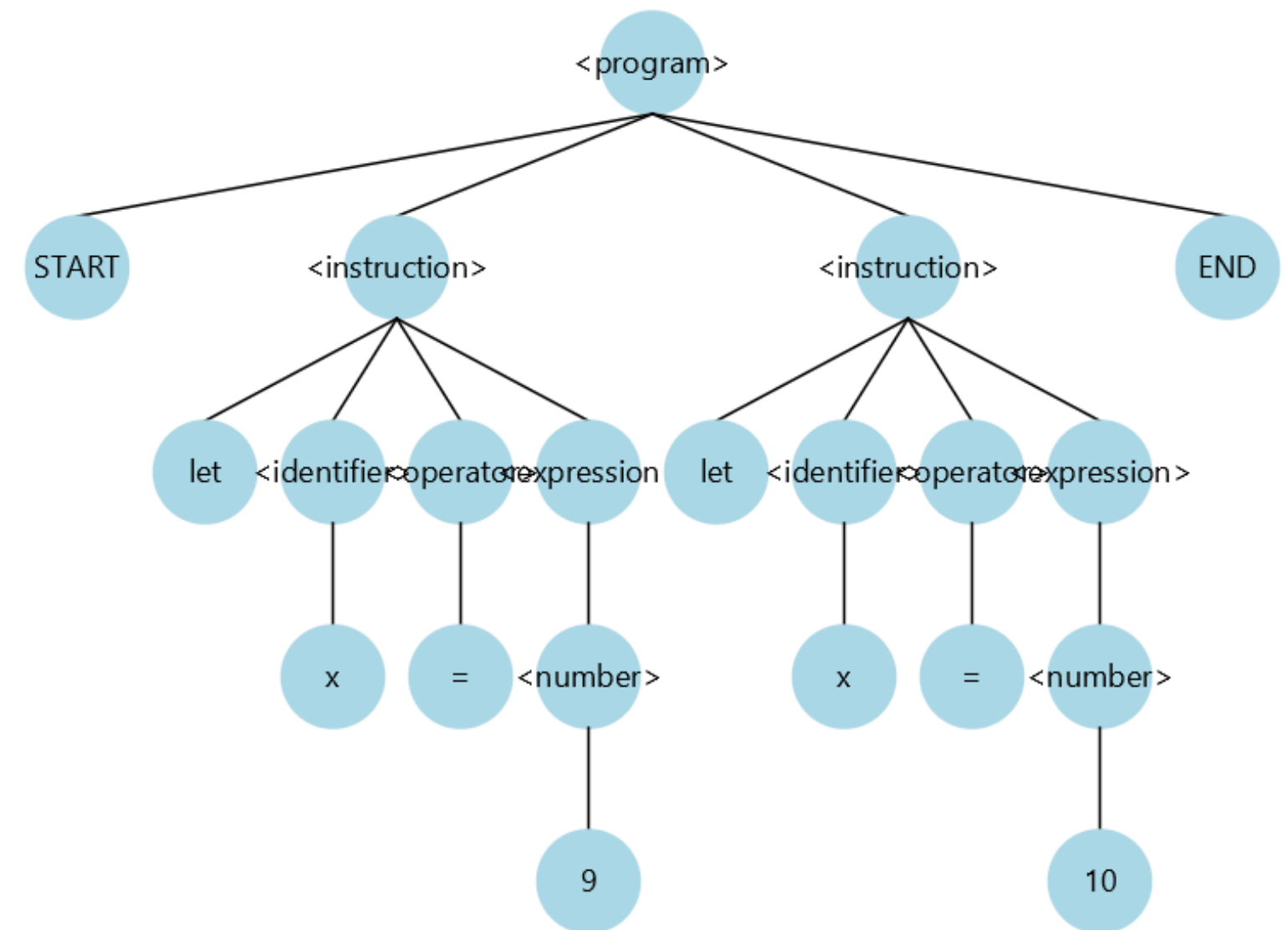
## Semantic Errors

- Incorrect block usage → `while x < 5 {  
 print x }`



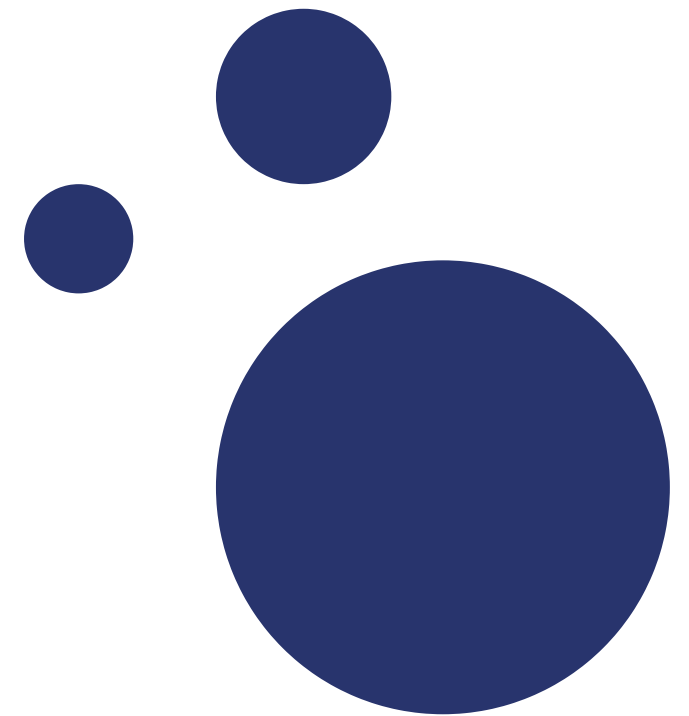
# ACHIEVED RESULTS

- Successfully analyzes code: lexically, syntactically, semantically
- Tree visualization reflects real grammar derivation
- Interface is clean, animated, and interactive



# CONCLUSIONS

- VisualComp makes learning compilers easier.
- It helps connect theory with practice.
- Visualization enables understanding of each compilation phase.
- Provides clear feedback on errors, supporting debugging skills.



# REFERENCES

- Aho et al. Compilers: Principles, Techniques, and Tools, 2nd ed.
- T. Parr, The Definitive ANTLR 4 Reference. Dallas, TX, USA: The Pragmatic Bookshelf.
- Appel. Modern Compiler Implementation in Java

