VisualComp: Educational Compiler With Visualization in Java

1st Juan Pablo Bustos Urueña - 20221020114
2nd Maria Camila Restrepo Silva - 20221020044
Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia

***Abstract*—Teaching compiler construction is a persistent challenge in computer science education due to the complexity and abstract nature of its internal phases, such as lexical analysis, parsing, semantic checking, and code generation. Students often struggle to understand how high level code is transformed internally, especially when learning is based solely on theoretical material or industrial-grade tools that obscure intermediate steps. To address this issue, VisualComp is presented, an educational compiler developed in Java that includes an interactive graphical interface using JavaFX to visually represent the compilation process. This tool allows students to input code in a simplified custom language and observe animated representations of tokens, syntax trees and semantic validations. VisualComp is designed using traditional compiler design techniques such as recursive-descent parsing, symbol table construction, and three-address code generation. The system will be tested in an academic setting with undergraduate students to bring the needed feedback for the improvement of the project. The main expected result of the project is that students can better understand the internal functioning of a compiler, the way it works. Also the expected result of the project is the correct operation of the compiler and the correct graphics that are wanted to be presented**

***Index Terms*—compiler education, visualization, JavaFX, programming languages, lexical analysis, derivation tree, educational tools**

I. Introduction

The compilation process is one of the fundamental areas in computer science, particularly in the education of professionals in software engineering, computer science, and related fields. However, for many students, the concepts that make up a compiler are abstract and difficult to visualize. This difficulty is exacerbated when learning relies exclusively on theory or on industrial tools that hide the internal details of the translation process from a source language to a lower-level representation. In response to this issue, there is a growing need for educational tools that make the inner workings of a compiler visible, comprehensible, and interactive in a progressive and engaging manner.

Over the past decades, several tools have been developed to support compiler instruction. Among them are JFLAP [3], which enables the visualization of automata and grammars; ANTLR [2], a powerful lexer and parser generator; and classic tools such as Yacc and Flex [5], which are widely used in academic environments. Additionally, platforms like WinFlex/Bison and visualizers such as SyntaxTreeViewer allow graphical representations of derivation trees. However, most of these solutions are intended for users with intermediate to advanced technical knowledge and do not integrate all components of a compiler into a single interactive interface—limiting their pedagogical effectiveness.

In response to these limitations, this project proposes the development of an educational compiler built from scratch in Java, coupled with a graphical interface using JavaFX that animates each stage of the compilation process in real time. The tool aims to help students understand how high-level code is transformed into lower-level representations through visual and interactive feedback. Java is chosen due to its wide adoption in educational contexts, its robust graphical ecosystem, and its object-oriented structure, which facilitates modular and readable design.

From a computational standpoint, the project implements classical compiler construction techniques such as lexical analysis using finite automata, recursive-descent parsing, derivation tree generation, type checking through a symbol table, and the generation of intermediate code in three-address format [1], [4]. The interface uses graphical components such as animated transitions, derivation trees, and a message console to guide students through each stage in an intuitive and progressive way.

II. Methods and Materials

A. Programming Languaje

Java was selected as the implementation language for this project due to its clarity, object-oriented structure, platform independence, and wide adoption in both industry and education. These features make it especially suitable for modularizing the different phases of the compiler—lexical analysis, parsing, semantic checking, and code generation. Its standard libraries (such as java.util and java.io) provide essential support for data structures and file handling, while regular expressions (java.util.regex) facilitate token recognition during lexical analysis.

To enrich the educational experience with real-time visual feedback, the project integrates JavaFX, a modern graphical library for building rich user interfaces. JavaFX enables the creation of interactive components such as code editors, message consoles, and syntax tree viewers. The visual representation of the Derivation Tree is handled using TreeView structures, while animations of lexical and syntactic processing are implemented through Timeline and Transition classes.

B. Topics

The main topic that would be used for the project (clearly apart of the programming languaje) relies on the use of generative grammar, tokens, and derivation trees, which are essential components in the construction of any compiler. Generative grammars formally define the syntax of the language being compiled, allowing the system to recognize valid sequences of symbols and enforce structural rules. This makes it possible to implement deterministic parsing algorithms capable of transforming input code into structured representations. Tokens, produced during lexical analysis, serve as the fundamental units processed by the parser, enabling the transition from raw text to syntactically meaningful constructs. The derivation tree, built according to the grammar rules, provides a structured view of how source code elements are hierarchically organized, which is critical for further stages such as semantic analysis and code generation.

The generative grammar implemented in this project serves as the foundational structure for parsing the input code and constructing the derivation tree. Designed as a context-free grammar (CFG), it defines the syntactic rules and constructs permitted within the simplified programming language. The grammar was intentionally developed with clarity and minimalism in mind, allowing students new to compiler theory to observe how input strings can be broken down and interpreted based on predefined patterns. Each rule in the grammar corresponds to a common programming instruction such as variable declarations, print statements, conditional blocks, or loops, and outlines how these statements are composed using identifiers, numbers, operators, and control symbols like parentheses and braces. The grammar supports recursive patterns and hierarchical nesting, enabling the compiler to build a structured tree representation of complex expressions and control flows.

By employing these tools, it becomes possible to provide an effective solution to the problem through the implementation of generative grammar, tokens, and derivation trees using Java. These elements allow for the creation of a compiler capable of recognizing basic user inputs and, by applying formal grammar rules, breaking down lines of code into a structured derivation tree. This tree reveals the hierarchical composition of the input according to the language's syntax. Additionally, the system can generate a corresponding list that explicitly displays the sequence of grammar productions applied during the parsing process. This not only facilitates the internal organization of the compiler but also enables clear tracking of the syntactic analysis performed on the code.

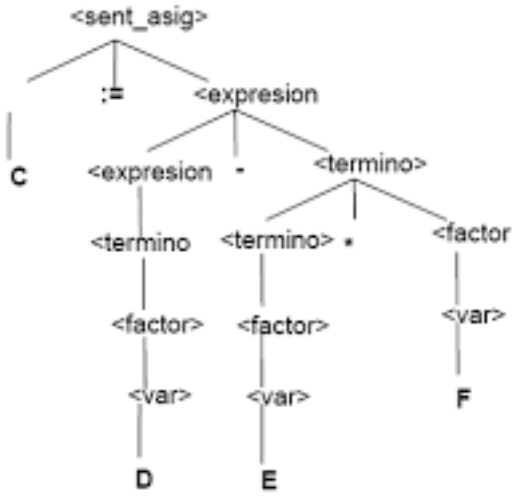

Fig. 1. Example of a derivation tree.

III. Results

- The main achieved result of the project is a fully functional educational compiler that successfully interprets simplified input code and transforms it into a corresponding derivation tree. The compiler is capable of identifying valid tokens through lexical analysis, applying defined grammar rules through a syntax parser, and generating a clear hierarchical tree structure that reflects the syntactic composition of the input. Additionally, it provides a graphical visualization of the derivation tree, offering students a tangible view of the compilation process. The tool also logs the applied grammar productions, ensuring transparency and traceability throughout parsing.
- The expected result for the contextual problem is that students will be able to understand the basic operations performed by a compiler during the compilation process. Through the use of a simplified syntax and the visualization of the corresponding derivation tree, the tool aims to make key stages—such as lexical analysis, syntax parsing, and grammar rule application—more accessible and comprehensible. Thanks to this, students can gain a clearer understanding of how compilers analyze structure, apply grammar, and generate intermediate representations, thus bridging the gap between theoretical concepts and practical implementation.
- Combining lexical, syntactic, and semantic analysis within a single Java application allows users to navigate the entire compilation process in a modular and clear way. Every phase of the compiler—ranging from tokenization to the creation of the derivation tree—was developed with a distinct object oriented approach, which guarantees both maintainability and scalability while also functioning as a model for grasping compiler architecture
- The visualization component of the compiler successfully enhances user engagement and comprehension by displaying the derivation tree. This graphical output allows users to see how their input is processed step by step, providing immediate feedback and reinforcing theoretical knowledge through visual representation. It proved to be an effective didactic aid for introducing the concept of syntax parsing and grammar rule application.

IV. Conclusions

- This project lies at the intersection of educational software development and compiler theory, contributing to the efforts to improve the teaching of this complex area. By integrating visualizations, modular design, and

a simplified source language, this tool seeks to bridge the gap between theoretical knowledge and practical understanding of compilers at the undergraduate level.

- The successful implementation of a fully functional compiler in Java demonstrates the feasibility of building educational tools that offer transparency in the compilation process. The integration of lexical, syntactic, and semantic analysis into a single environment promotes a holistic understanding of compiler phases and reinforces structured thinking in students.
- The visualization of derivation trees not only enhances student engagement but also improves their ability to internalize abstract compiler concepts. By observing the transformation of input code into a hierarchical structure governed by grammar rules, users gain a deeper appreciation for the inner workings of compilers.
- In addition to its direct academic benefits, the project cultivates abilities in software engineering, formal language theory, and graphical user interface design. Through the integration of theoretical precision and practical relevance, it enables students to grasp and expand upon the principles of compiler development.

REFERENCES

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools, 2nd ed. Boston, MA, USA: Pearson, 2007.

[2] T. Parr, The Definitive ANTLR 4 Reference. Dallas, TX, USA: The Pragmatic Bookshelf, 2013..

[3] S. H. Rodger and T. W. Finley, JFLAP: An Interactive Formal Languages and Automata Package. Sudbury, MA, USA: Jones & Bartlett Learning, 2006.

[4] K. C. Louden, Compiler Construction: Principles and Practice. Boston, MA: PWS Publishing Company, 1997.

[5] J. R. Levine, T. Mason, and D. Brown, lex & yacc. Sebastopol, CA, USA: O'Reilly Media, 1992.