Design and implementation of a simple compiler in Python for generating personalized avatars

Juan David Guevara García - 20202020001
Universidad Distrital
Bogotá, Colombia
judguevarag@udistrital.edu.co

Javier Alejandro Alvarez Marin - 20202020028
Universidad Distrital
Bogotá, Colombia
jaalvarem@udistrital.edu.co

***Abstract*—This paper presents a practical exercise in computer science aimed at developing a conceptual understanding of compiler functionality without delving into low-level implementation details. To achieve this, we designed and implemented a Python-based command interpreter capable of rendering digital avatars. The primary objective is to facilitate the internalization of core compiler principles through hands-on experience. Furthermore, this technology could have potential for broader applications, such as enhancing user interaction in e-commerce platforms, or serving as an educational tool to support the teaching of programming and compiler concepts.**

***Index Terms*—Compiler Design, Command Interpreter, Python**

I. Introduction

Compilers are fundamental tools in computer science, responsible for translating high-level programming languages into machine-readable instructions. Traditionally, compilers operate invisibly, transforming code into executable programs without providing a visual understanding of their internal processes. In this project, we propose an approach that lets Python simulate the functionality of a compiler in a visual manner. By utilizing concepts from formal language theory—specifically, context-free grammars [1]—we aim to build a system that not only parses and processes input commands but also generates corresponding visual avatars as output [2]. This approach serves both as a practical demonstration of compiler principles and as an engaging educational tool that helps users grasp the abstract concepts behind syntax analysis, parsing, and code generation through interactive visual representation.

There is a wide variety of avatar generation libraries across different programming languages, such as Java and modern versions of Python, each offering different levels of customization and control over visual output. However, these solutions often rely on direct code calls and lack a user-friendly language structure. The unique challenge we address in this project is the integration of context-free grammars to build a compiler-like environment in Python, allowing users to define and modify avatars using a simplified syntax. This not only streamlines the creation process but also introduces the principles of compiler theory into a visual and interactive domain.

II. Methods and Materials

The development of the project was based on the Python programming language due to its accessibility, readability, and rich ecosystem [3], [4]. For the generation of digital avatars, the `py_avataaars` library was used [5], which allows programmatic creation of SVG images with customizable attributes such as hairstyle, skin tone, accessories, and clothing.

The system was divided into three main modules:

1) **Lexical, syntactic, and semantic analysis**: These modules simulate the fundamental phases of a compiler [1].
2) **Graphical user interface (GUI)**: Allows users to input commands and receive visual feedback through avatars [6].
3) **Avatar rendering with py_avataaars**: Converts validated attributes into graphical avatars [5].

A. General Structure of the System

The main objective was to build an **educational compiler**, aimed at supporting the learning process of compilation. To achieve this, a tool was designed that simulates the internal functioning of a real compiler, allowing users to observe how its various components operate and how the instructions they enter are validated.

1. Compiler Modules: **Lexical Analysis:** The input text is read and broken down into tokens. Implemented with regular expressions in `lexicalProject.py`, it identifies elements such as keywords (`inicio`, `final`), verbs (`teñir`, `ajustar`), and attributes (`cabello`, `ropa`).

Syntactic Analysis: Implemented in `sintaticProject.py`, this module checks the structure of token sequences against a defined grammar using a top-down parser.

Semantic Analysis: Implemented in `semanticProject.py`, it verifies logical consistency, such as verb-attribute compatibility, and generates a configuration object for avatar rendering.

2. Graphical User Interface (GUI): The GUI allows users to input commands intuitively and view the step-by-step compilation process. Built using Tkinter [**?**], it processes text input and, upon successful validation, triggers avatar generation.

3. Avatar Generation with `py_avataaars`: Once validation is complete, the `py_avataaars` library [5] generates a visual avatar based on the defined attributes:

- Hairstyle and color
- Skin tone
- Clothing and accessories
- Facial expressions

The avatar serves as positive visual feedback, motivating the user and reinforcing learning.

III. Results

To evaluate the functionality and performance of the educational compiler, a series of tests were carried out using both valid and invalid input sequences. These tests aimed to verify the correct behavior of the lexical, syntactic, and semantic modules, as well as the successful integration of the graphical interface and avatar generation.

A. Validation of Compiler Modules

The system was tested with a set of well-formed instructions such as:

```
inicio
teñir cabello castaño_oscuro;
ajustar ropa hoodie;
añadir accesorio gafas_redondas;
expresar boca sonriente;
final
```

The lexical analyzer correctly identified all tokens and their types. The syntactic analyzer confirmed the structure was valid according to the defined grammar, and the semantic analyzer verified the compatibility between verbs and attributes. The avatar configuration was successfully generated based on the parsed values.

When tested with incorrect input, such as:

```
inicio
teñir ropa azul;
final
```

The semantic analyzer detected an invalid attribute-verb pairing (”teñir” cannot be used with ”ropa”), and an appropriate error message was returned, demonstrating robust error handling.

B. Interface and User Interaction

The graphical user interface allowed users to input text and receive immediate visual and textual feedback. Successful compilation resulted in the automatic generation of an avatar using the `py_avataaars` library. This confirmed that the integration between the compiler and the avatar system was working correctly.

C. Educational Value and Usability

Informal testing with users unfamiliar with compilers showed that the system effectively communicated compilation stages through both textual validation and visual representation. The inclusion of the avatar acted as an incentive and provided intuitive feedback, reinforcing understanding of how each instruction affected the result.

Overall, the system performed as expected across all test cases, validating the effectiveness of the architecture and the learning-oriented design.

Preliminary tests showed that the system successfully processed valid input commands and rejected invalid ones with appropriate error messages. For example, using the command:

```
inicio
teñir cabello castaño_oscuro;
ajustar ropa hoodie;
final
```

produced a correctly configured avatar. In contrast, invalid instructions such as `teñir ropa azul;` generated semantic errors. This confirmed the effectiveness of the compiler modules. The GUI performed as expected and avatar generation was visually correct.

Furthermore, tools of this type have shown value in other research projects as educational aids for learning compiler theory [7].

Results:

This input generates an avatar with:

- Afro hairstyle with red color
- Trigueña skin tone
- Hoodie clothing in pastel blue
- Sunglasses
- Furrowed eyebrows, surprised eyes, and shouting mouth

Output:

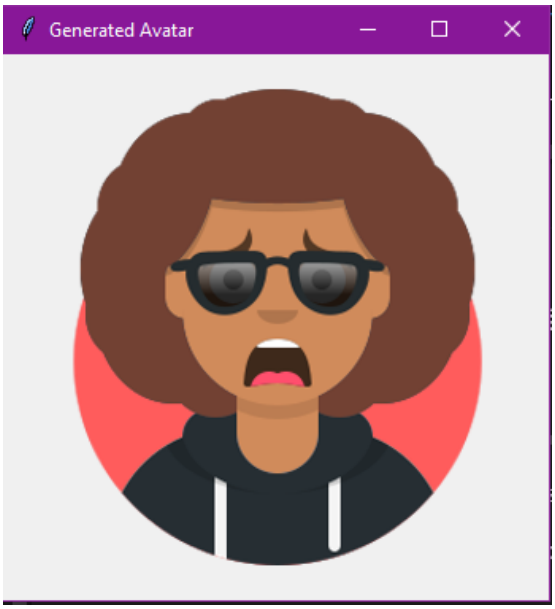

Fig. 1. Generated avatar from Example 1 input.

IV. Conclusions

1) **Effective simulation of the compilation process:**
   The implementation of the lexical, syntactic, and semantic modules successfully replicated the behavior of a real compiler in a didactic context [1], [2].
2) **Educational usability enhanced by the graphical interface:**
   The inclusion of a basic GUI using Tkinter [6] made the tool accessible even to users without programming experience.
3) **Visual feedback with `py_avataaars` as a motivational tool:**
   The integration of `py_avataaars` [5] provided engaging visual output that reinforced the learning process.
4) **Scalability and adaptability of the system:**
   Thanks to Python's flexibility [3], [4], the system can be expanded to support new attributes or even new domains.

References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Addison-Wesley, 2006. [Online]. Available: https://www.pearson.com/en-us/subject-catalog/p/compilers-principles-techniques-and-tools/P200000003444/9780136067054

[2] D. Grune, H. E. Bal, C. J. Jacobs, and K. G. Langendoen, *Modern Compiler Design*, 2nd ed. Springer, 2012. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4471-2300-0

[3] G. van Rossum and F. L. D. Jr., "The python language reference manual," https://docs.python.org/3/reference/index.html, accessed: 2025-07-08.

[4] M. Lutz, *Learning Python*, 5th ed. O'Reilly Media, 2013. [Online]. Available: https://learning.oreilly.com/library/view/learning-python-5th/9781449355722/

[5] PyPI, "py_avataaars: A python library for generating avatars," https://pypi.org/project/py-avataaars/, accessed: 2025-07-08.

[6] P. S. Foundation, "Tkinter - python interface to tcl/tk," https://docs.python.org/3/library/tkinter.html, accessed: 2025-07-08.

[7] M. González and J. Salas, "Design and implementation of an educational compiler for learning grammatical structures," *Revista de Ingeniería y Tecnología*, vol. 16, no. 2, pp. 45–56, 2018. [Online]. Available: https://example-journal.org/article/educational-compiler