



PROJECT AND TEAM INFORMATION


Project Title

(Try to choose a catchy title. Max 20 words).

Pseudo Transpiler

Student / Team Information

Team Name: Team #	
Team member 1 (Team Lead) (Last Name, name: student ID: email, picture):	Shubham Negi – 22021844 snegi4260@gmail.com 
Team member 2 (Last Name, name: student ID: email, picture):	Neha Dobriyal - 220221247 nehadobriyal70@gmail.com 
Team member 3 (Last Name, name: student ID: email, picture):	Gokul Singh – 220211572

	jaie6252@gmail.com 

PROPOSAL DESCRIPTION (10 pts)

Motivation (1 pt)

(Describe the problem you want to solve and why it is important. Max 300 words).

The problem is the lack of tools that can help students learn pseudocode , which is widely used for education and algorithm design purposes . Students and developers often write pseudocode to outline logic but cant quickly check the legitimacy of their logic without writing that pseudocode into an actual programming language, which is time-consuming and error-prone, especially for beginners.

This project aims to automate this process by making a transpiler to convert the pseudocode like language to python code , enhancing productivity and learning efficiency. It is will allow students to focus on algorithmic thinking and provides a foundation for further language translation tools it can also serve as an accessible learning tool for those who want to explore compiler design but may find working with real programming languages intimidating at first.

Overall, the project bridges the gap between conceptual logic and practical implementation, offering value to both students and educators.

State of the Art / Current solution (1 pt)

(Describe how the problem is solved today (if it is). Max 200 words).

Currently pseudocode to code conversion relies on manual translation by programmers or limited automated tools . Students typically rely on their personal knowledge of programming languages to check their pseudocode's logic .

Some online tools, like Pseudo Editor (pseudoeditor.com), offer basic conversion using ai but lack full compiler phases , don't share implementation details and charge to use full features while also lacking support for structures like functions.

Open-source projects on GitHub, such as MugilanGN/Pseudocode-Compiler, target specific pseudocode standards (e.g., IGCSE) but output to LLVM IR, which is not beginner friendly.

Solution available are either incomplete , or require technical expertise to use and understand leaving room for a beginner-accessible transpiler .

Project Goals and Milestones (2 pts)

(Describe the project general goals. Include initial milestones as well any other milestones. Max 300 words).

The goal of this project is to develop a pseudocode-to-Python transpiler using a compiler design pipeline (lexical analysis, syntax analysis, semantic analysis, code generation), with a GUI to make it beginner friendly . This will serve as a tool that will improve learning for beginners.

Initial milestones include :

- (WEEK 1-2) Defining a standard (or use already present standards like *IGCSE-style pseudocode* to write pseudocode and define grammar and rules to generate it.
- (WEEK 1-2) Implement a lexical analyser to tokenize the input .
- (WEEK 3-5) Create a parser and use grammar rules to generate the AST.
- (WEEK 6-7) Perform basic semantic checks on variable declaration , usage and type compatibility.
- (WEEK 8-9)Implement a code generator to convert the AST into valid Python code.
- Build a GUI to showcase our work

Initially the grammar would be kept limited to variables , input , output , conditionals , iteratives , basic operators and Functions for simplicity of project and completion before time.

Other milestones include :

- Extend support for complex tasks like arrays , more operators
And basic data structures that are widely used.

This approach allows us to finish the project first with less trouble and then keep in expanding it just like any real world application does .

Project Approach (3 pts)

(Describe how you plan to articulate and design a solution. Including platforms and technologies that you will use. Max 300 words).

To develop this project, we will follow the classical phases of compiler design: lexical analysis, syntax analysis, semantic analysis, and code generation.

The system will be implemented in Python due to its simplicity, strong text processing capabilities, and availability of parser libraries. The project aims to mirror the compiler pipeline while keeping the design beginner-friendly .

APPROACH :

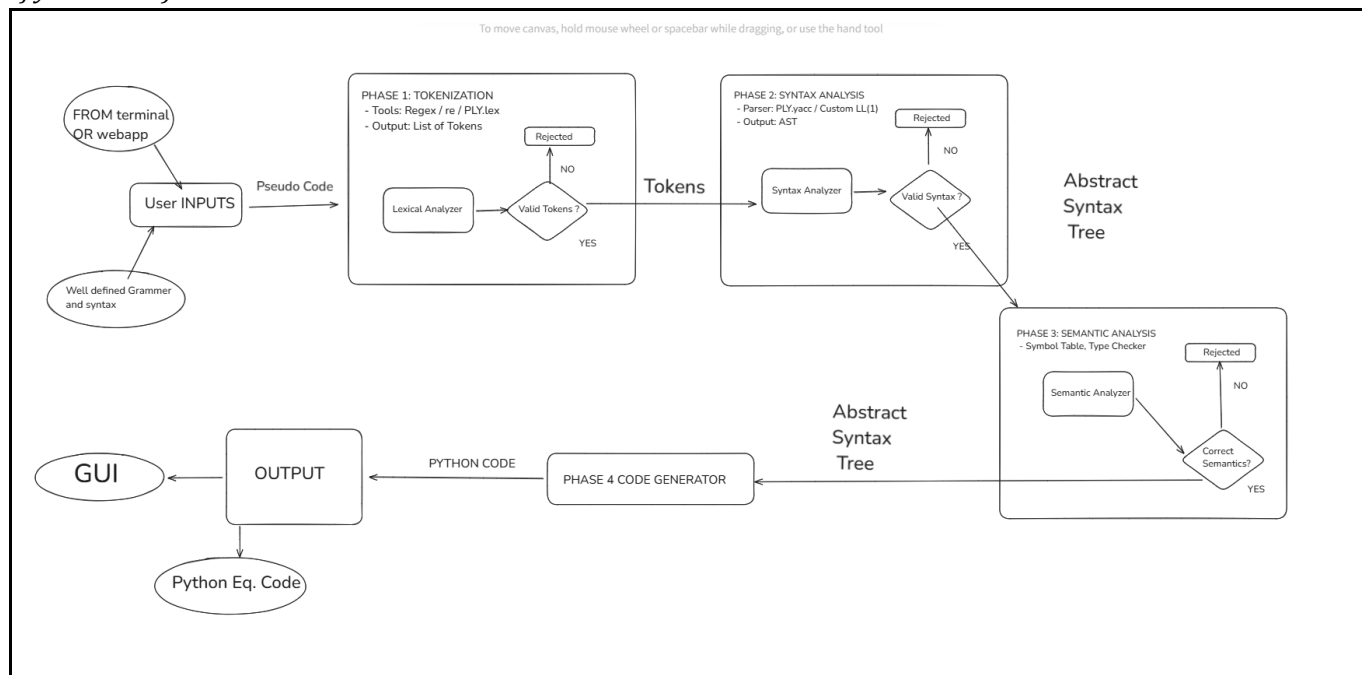
1. **Input** : We begin with a well-defined grammar and syntax for our pseudocode language (inspired by common educational styles like pseudocode) which will be inputted from terminal / file / web app
2. **Lexical Analysis** : Using this grammar, we will implement a lexical analyzer using Python's re module or libraries like PLY.lex (Python Lex-Yacc) to tokenize the input into well defined tokens of the language.
3. **Syntax Analysis** : These tokens will be passed to a parser built with PLY.yacc or a custom parser (top down or bottom up) that constructs an Abstract Syntax Tree (AST) representing the structure of the pseudocode.
4. **Semantic Analysis** : The semantic analyzer will traverse the AST to perform type checks, scope validation, and other logical validations using symbol tables.
5. **Code Generation** : Once verified, the code generation phase will translate the AST into clean, equivalent Python code using a template-based or tree-walk approach.

We will also develop a simple graphical user interface (GUI) using Tkinter or a Web Application using Web Development to allow users to input pseudocode, view phase-by-phase transformation (tokens, AST, semantic checks), and see the final Python code output

This modular and layered approach will ensure that each component can be tested independently.

System Architecture (High Level Diagram)(2 pts)

(Provide an overview of the system, identifying its main components and interfaces in the form of a diagram using a tool of your choice).



Project Outcome / Deliverables (1 pts)

(Describe what are the outcomes / deliverables of the project. Max 200 words).

The project will deliver a functional pseudocode-to-Python transpiler with the following outcomes:

- A core transpiler supporting basic constructs like input, conditionals, iterative ,output with basic operators implemented using a complete compiler pipeline .
- A tkinter-based or Web based GUI for user-friendly input of pseudocode and display of Python output, showcased midway.
- An extended version supporting functions, arrays, and additional operators completed by the end .
- A comprehensively test project to validate functionality .
- Documentation including a README with grammar specification, usage guide, and sample inputs/outputs.
- The deliverables will be a Python package with scripts for different phases of compiler like lexer.py, parser.py, semantic.py, codegen.py, main.py, transpiler_gui.py etc hosted on GitHub, along with a high-level architecture diagram.

Assumptions

(Describe the assumptions (if any) you are making to solve the problem. Max 100 words)

We assume pseudocode follows a consistent grammar (e.g., IGCSE-style or any else) defined upfront, with keywords in uppercase and clear structure.

Users have basic Python 3.9+ installed, and the system runs on a standard desktop/laptop environment.

We assume no complex concept and type systems like Data Structures and Classes are NOT needed initially, focusing on only the things mentioned in the initial goals .

Input errors are kept minimal by the user initially , and users will test output Python code in a compatible environment.

These assumptions simplify development, with plans to address variations in later iterations if needed.

References

(Provide a list of resources or references you utilised for the completion of this deliverable. You may provide links).

- GeeksforGeeks. "Compiler Design Tutorial." <https://www.geeksforgeeks.org/compiler-design-tutorials/>
- Strumenta. "How to write a transpiler." <https://tomassetti.me/how-to-write-a-transpiler/>
- GitHub: MugilanGN/Pseudocode-Compiler, mbyx/psu, Khushi-Balia/le-transpiler.
- Python Software Foundation. "PLY (Python Lex-Yacc)." <https://ply.readthedocs.io/>
 - Youtube playlist on how to build you own language in python
https://www.youtube.com/playlist?list=PLZQftyCk7_SdoVexSmwy_tBgs7P0b97yD