PL Project - Abstract Tree Generator for C

Mani Nandadeep, IMT2019051
R Prasannavenkatesh, IMT2019063
Vijay Jaisankar, IMT2019525
M Dhanush, IMT2019049

I. Problem statement

The aim of this project is to implement a program that computes the abstract syntax tree of an input program written in C. The AST Generator is written in C++.

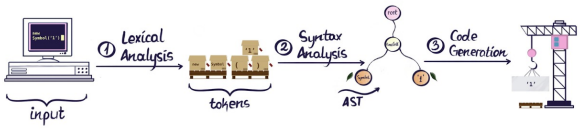

Fig. 1. Transformation of a code by a compiler [1].

II. Problem Specifics

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program that performs lexical analysis may be termed a **lexer**. A lexer is generally combined with a parser, which together analyze the syntax of programming languages.

A **parser** is a component of a compiler or interpreter that divides data into smaller elements for easier translation into another language. A parser takes input in the form of a sequence of tokens, interactive commands, or programm instructions and divides it into parts that can be used by other programming components.

Abstract syntax trees are structures used in program analysis and program transformation systems. It is a tree that represents the syntactic structure of a language construct according to our grammar definition. Typically, the most general implementation of ASTs are n-ary trees. Each node holds a token and pointers to its first child and next sibling. They frequently serve as an intermediate representation of the program and has a significant impact on the compiler's final output.

In our project, We will be looking into writing our custom parser, lexer, and AST Generator using C++. This will only include a certain subset of the C programming language considering the vastness of the language.

III. Solution Outline

A. Deliverables

- A Github Repository containing the source code of Abstract syntax tree generator for the C language in C++.
- Project Report explaining the project and its working in detail.
- README file containing the instructions on how to run the project.
- A Video Demo explaining the code and main features of the project.

B. Work Split

- Mani Nandadeep - Lexer and Grammar implementation
- R Prasanna - Parser, AST, and Grammar implementation
- Vijay Jaisankar - Parser, AST, and Grammar implementation
- M Dhanush - Lexer implementation, Parser

C. References

- **Language Implementation Patterns** [2]: This book explains how existing language applications work and how certain concepts are implemented.

- **Compilers: Principles and Practice** [3]: This book explains the phases and implementation of compilers and interpreters, using a large number of real-life examples. We will be using this book to understand the various stages of compiling and interpretting, as well as use it as a reference book for getting test cases and examples.
- **GNU Bison** [4]: Yacc-compatible parser generator. Bison is a general purpose parser generator that converts a grammar description for an LALR(1) context-free grammar into a C program to parse that grammar. We will be using Bison as a tool for inspiration; we can validate our outputs based on this tool, and see how we can model our outputs based off of it.
- **Lex and YACC** [5]: Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream, i.e, Split the source file into tokens. Yacc provides a general tool for describing the input to a computer program. The Yacc user specifies the structures of his input, together with code to be invoked as each such structure is recognized. Yacc turns such a specification into a subroutine that handles the input process; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine,i.e, Find the hierarchical structure of the program.
- **AST Explorer** [6]: We will use this tool to analyse visualisations of parse trees.

REFERENCES

[1] D. Kundel, "Introduction to abstract syntax trees," Jun 2020. Available at https://www.twilio.com/blog/abstract-syntax-trees.

[2] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Pragmatic Bookshelf, 1st ed., 2009.

[3] H. B. D. Parag H. Dave, *Compilers: Principles and Practice*. Pearson Education, 2012.

[4] "Gnu bison." https://www.gnu.org/software/bison/.

[5] "The lex yacc page." Available at http://dinosaur.compilertools.net/.

[6] "Ast explorer." Available at https://astexplorer.net/.