Department of Computer Science and Engineering

Course Code: CSE 420	Credits: 1.5
Course Name: Compiler Design	Semester: Summer 2023

1 Introduction

In the last assignment, we have constructed a lexical analyzer to generate token streams for a subset of python language.

In this assignment, we will construct the last part of the front end of a compiler for a subset of the C language. That means we will perform syntax analysis with a grammar rule containing function implementation in this assignment. To do so, we will build a parser with the help of Lex (Flex) and Yacc (Bison).

2 Language

Our chosen subset of the C language has the following characteristics:

• There can be multiple functions. No two functions will have the same name.

• There will be no pre-processing directives like `#include` or `#define`.

• Variables can be declared at suitable places inside a function. Variables can also be declared in the global scope.

• All the operators used in the previous assignment are included. Precedence and associativity rules are as per standard. Although we will ignore consecutive logical operators or consecutive relational operators like, `a && b && c`, `a < b < c`.

• No `break` statement and `switch-case` statement will be used.

3 Tasks

You have to complete the following tasks in this assignment.

3.1 Syntax Analysis

For the syntax analysis part you have to do the following tasks:

• Incorporate the grammar given in the file "**Lab3_C_Syntax_Analyzer_Grammar.pdf**" along with this document in your Yacc file.

• You are given a modified lex file names **"lex_analyzer.l"** to use it with your Yacc file. Try to understand the syntax and tokens used.

• Use a `SymbolInfo` pointer to pass information from lexical analyzer to parser when needed. For example, if your lexical analyzer detects an identifier, it will return a token named ID and pass its symbol and type using a SymbolInfo pointer as the attribute of the token. On the other hand in the case of semicolons, it will only return the token as the parser does not need any more information. You can implement this by redefining the type of yylval (YYSTYPE) in parser and associate yylval with new type in the scanner. See the skeleton file for example.

• Handle any ambiguity in the given grammar (For example, if-else). Your Yacc file should compile with 0 conflicts.

• When a grammar matches the input from the C code, it should print the matching rule in the correct order in an output file (log.txt). For each grammar rule matched with some portion of the code, print the rule along with the relevant portion of the code.

4 Input

The input will be a text file containing a c source program. File name will be given from the command line.

5 Output

In this assignment, there will be one output file. The output file should be named as <Your_student_ID>_log.txt. This will contain matching grammar rules, and corresponding segment of source code as instructed in the previous sections. Print the line count at the end of the log file.

For more clarification about input-output check the supplied sample I/O files given in the lab folder. You are highly encouraged to produce output exactly like the sample one.

6 Submission

1. In your local machine create a new folder whose **name is your student id.**
2. Put the lex file named as **<your_student_id>.l**, the Yacc file **<your_student_id>.y** and a script named **script.sh** (modifying with your own filenames), the **symbol_info.h** file in a folder **named with your student id**. **DO NOT** put any i/o file, generated lex.yy.c file or any executable file in this folder.
3. Compress the folder in a **.zip file** which should be **named as your student id**.
4. Submit the .zip file.

Failure to follow these instructions will result in penalty.