CMPE 230: SYSTEMS PROGRAMMING

In this project, the primary objective is developing an interpreter for an advanced calculator, using C programming language. The calculator has basic arithmetic operations, binary operations and functions. The interpreter is responsible for checking the input for possible syntax errors and generating the correct result unless the input is invalid.

Execution of program is done by using the terminal, so it can be said that the program interface is the terminal of OS. The program doesn’t need any parameters. Inserting the file path of the executable file of program to the terminal is sufficient for execution of the program. Termination of the program is done by the termination command of terminal (e.g. Ctrl + C for Windows).

As it was written earlier, the main functionality of the program is calculating the result of given input by a user. The input is directly taken from the terminal, since the program doesn’t take any parameters, all the user has to do is inserting the operation to the terminal and pressing the Enter button. Unless the input has an error, given operation is done by the program. In case of an invalid input, “Error!” message will be printed to the terminal.

The operations can be categorized into two groups: equations (a = 2 \* 5) and non-equations (2 \* 5). For equations, left hand side of an operation must be a single variable. Right hand side of an operation is considered as the value of the variable, which is stored in the program for the upcoming operations. After the assignment, nothing will be printed to the terminal. Whereas for non-equations, the result is printed to the terminal.

The program is consisted of only one file. This file has the main method and all other methods which constructs the whole algorithm. In the beginning of the file, there are type definitions and structs, which are essential for the program. Method declarations, global variables and main method are right after these type definitions. Main method doesn’t take any parameters and return anything, it takes input and calls the necessary methods in order. When the method is done, frees the memory.

Before going further into the algorithm, it is useful to understand the structs which were defined in the beginning of file. The first one of them is Token, the smallest unit of the given inputs, it can also be named as lexeme. Token has 4 members and the first one of these members is TokenType. It is the terminals in the scientific calculator. It is helpful to classify and specialize the Token, whether it is an operator, function call, variable etc. There are 18 different TokenType’s, their functionality can be easily understood by its name. (e.g. ADDITION for “+”, COMMA for “,“ ).

Tokens have 3 other members: Id, name and number. Id is the string form of TokenType. It makes debugging very easy, since C doesn’t give structs a default “toString” method. Name and number isn’t significant for all Tokens, however they are very useful for specific types of Tokens and other methods. Name is very functional for VARIABLE TokenType and another struct whose has the same name, Variable(will be explained explicitly). Number is also very useful for the CONST TokenType, it stores the corresponding integer value of the integer part of the string in the given input.