Michelle Novenda Hartono 莊晴雯
108062281
Department of Computer Science

OJ Account: s108062281

During the process of doing this first mini-project, which is to implement a simple calculator, I encountered a few problems, one of which is to understand the template given. First thing I did to understand the template was to write down the flow of the code and figuring out what each line is meant to do. After exchanging some thoughts and ideas with my other friends too, we eventually got this all figured out and understood the main key to designing this code.

It is said there are at least three variables x, y, z that are registered from the beginning, therefore we need to set their values to 1 because our idea is that every registered variable's value will not be equal to 0, in this case, the value is 1. In the main function, the function *statement()* is called, here is where the function determines what to do after reading the input that is either EOF, enter, or anything else. Before that, we need to get to the *match()* function first to get the token type to know which *if statement* inside the *statement()* function satisfies. The program will directly exit if the input reaches EOF. When it reaches the end of line ('\n'), we will go to the *advance()* function to get the token for the next input. Otherwise, we need to compute the expression by getting into other functions.

Following the flow, we first need to solve for the function *expr()*. Inside the *expr()* function, the *term()* function was called, and following is the *factor()* function. After we get the token type from the token function, we need to pass it to *term_tail(BTNode *)* function. In this function, following the precedence rule of the mathematical operations, multiplication and division come first, therefore we need to construct a tree and go to the *advance()* function to get the operator and set it as the root. Then for the left child, we set is as our input on the left-hand side of the equation. As for the right child, we set it as our input on the right-hand side of the equation. Not to forget, in the division we need to directly exit if the denominator's value is equal to 0, following the basic mathematical rules. Also, there might be a case where the denominator's value equals to 0 not when we directly type 0 as the input, but by other operations such as "(2-2)" in the expression "x = 5/(2-2)". To solve this, I made a new function called *division_error(BTNode *)* to check the right-hand side of the expression, and if the denominator equals 0, the program will directly exit. The program will also be able to compute for multiple occurrences of the division symbol (/) because of the variable *div_flag*.

After we finish executing these functions stated above and returning each of them, we get to *expr_tail(BTNode *)* function. We can calculate addition and subtraction here because the multiplication and division were already solved in the *term_tail(BTNode *left)* function, then return the value back to *expr()* function again for the program to continue executing.

The function *getval()* and *setval()* deals with the variable's input and its value. Specifically, *getval()* first checks if the input is INT or ID. If the input is INT, we need to use *atoi* to convert the string to an integer, then assign it to *retval* to be returned in the end. If the input is ID, we first check that the variable is registered or not. If the variable is registered, the variable's value needs to be set to anything but 0, in this case, we set it to 1, then assign the value to *retval*. We use i++ to check for other variables. For new variables, we first need to copy the new variable to the *arr[].alphabet[]*, then set the value to 0 first because the variable is still not yet registered. The function *setval()* is used to return the value of the equation on the right-hand side and set it to the left-hand side. Not to forget, we also need to do some changes to the *evaluateTree()* function in order to get the outputs we want.

Aside from understanding the template itself, I also encountered some problems during the work process, such as running out of ideas on what to do with the code, figuring out the wrong test cases, or even how to implement my thoughts to the code. Eventually, after reviewing the code again and again and exchanging some thoughts with the others, I was able to fully understand the idea and the concepts implemented on this simple calculator project.