Aiswarya, 2016, VCE

**Algorithm:**

I have designed my Expression evaluator by dividing them into phases, these phases are similar to the phases of compiler evaluating.

First phase is to read the file and identify the set of characters as operands and operators. This is called Lexical analysis phase.

Second phase is to check if the syntax is correct or not. The rules which I used to check are, the operator and operand should be occurring alternatively and it should start and end with an operand. This is called Syntax analysis phase.

Third phase is to check the allowed type of characters. My algorithm allows any integer which has less size than Long Double for an operand and allowed operators are \*\*, /, \*, +, - . Spaces and tab spaces are ignored. If any other input is given then it returns error. This is called Semantic analysis phase.

Forth phase is to put those operands and operators in a stack, in the order they appear and the precedence order. The **Data structure** I used for stack is **Linked list**, because the number of operands and operators used in an expression is not fixed and the linked is a better way to handle such situations rather than reallocating size of array every time. I have used two stacks ne for operand and other for operator. They both push the read values into the respective stacks. Every time while pushing an operator it checks and computes all the less precedence operators from the stack. The precedence of the operators are given an integer value and are compared with those values. This phase is called Intermediate code generation.

Features provided are to remove the unwanted data like extra spaces, tab spaces and leave if the is line with no data. If there is any error in a line, then that line returns error and the next line is executed normally. It can accept operands of decimal type also. Returns appropriate errors.