

**Preparatory data Structure (CSCI 591)**



**Evaluating General Infix Expressions**

**Project - VI**

**March 17, 2020**

**St. Cloud state university**

**Department of Computer Science**

Submitted By: Taddese Erba

# **Design Document**

## **Introduction**

Expressions that involve arithmetic operations can assume one of the three forms, postfix expression, prefix expression, or infix expression. Infix expression is an arithmetic operation in which each subexpression resides in their brackets. For example the expression is an infix expression. In C++, infix expressions Abstract Data Types (ADTs) can be implemented using the stack operation. This project will use the stack ADT to implement the general operations of infix expressions.

## **Data Structure**

To keep the program clear and distinct, the program will use three different files that define the class, implements the class methods, and a file that tests the implementation. The infix.h file contains all the declaration of the required functions. It is the framework for Infix class methods implementation. It consists of two private objects, the stack <char> sym object and the stack <int> op object. These two objects hold the bracket and other operational characters and the digit characters respectively. The Infix class contains eight functions each with their operations as discussed in the following section of this document.

## **Functions**

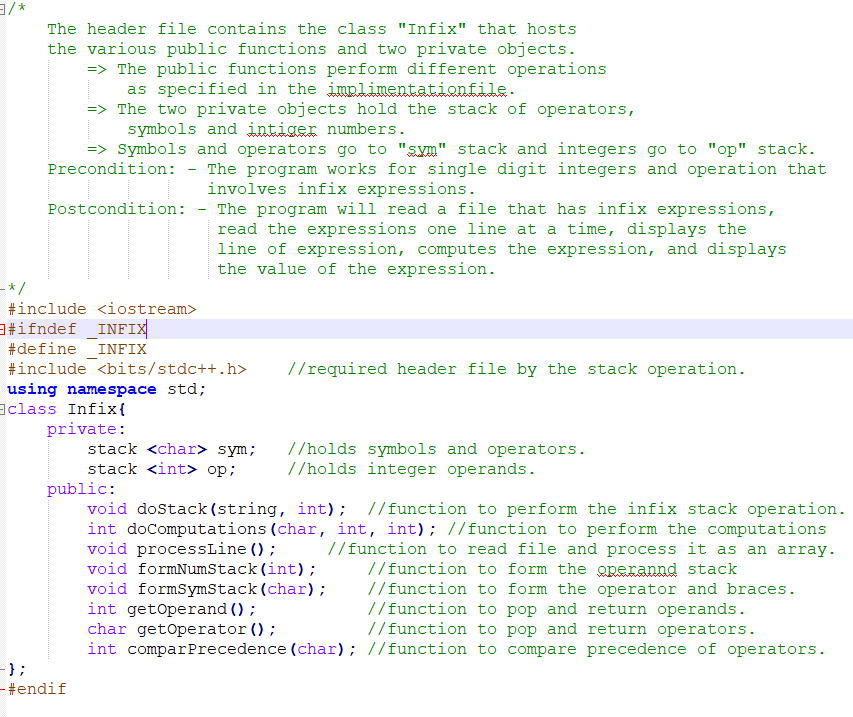
As described in the Data Structure section of this document, there are eight functions in this project. The functions void doStack(string, int) is used to perform the stacking operation. The stacking operation involves identifying and separating characters into their distinct stacks. This function takes two arguments, a string of arrays and an integer value that represents the string length. This function has no return value. The int doComputations(char, int, int) performs the respective arithmetic oppressions indicated in the infix expression. This function takes three arguments: the char type object to hold the operator and two int type objects to hold the two operands. It returns the result of the operation by the operator on the two operands. The void processLine() function is used to read and process the input file. This function will read the input file from the beginning of the file to its end. While reading it, it will process one line of the file at a time and send the string of files to the void doStack(string, int) method. This function neither takes an argument nor returns one. The two functions, void formNumStack(int) and void formSymStack(char) are used to form the numbers (operands) stack and symbol (brackets and operators) stack respectively. They both use the push() method to accomplish their task. They both have no return values and take an int and a char variable respectively. Similarly, the two functions that follow, the int getOperand() and the char getOperator() are used to obtain the two top operands and the top operator respectively. The int getOperand() function has a return type int and the char getOperator() has a return type char. They both take no arguments at all. The last function in the list is the int comparPrecedence(char) function. This function takes a char type variable and returns an int type variable. It is used to compare the operation precedence between the operators in the expression.

## **The Main Program**

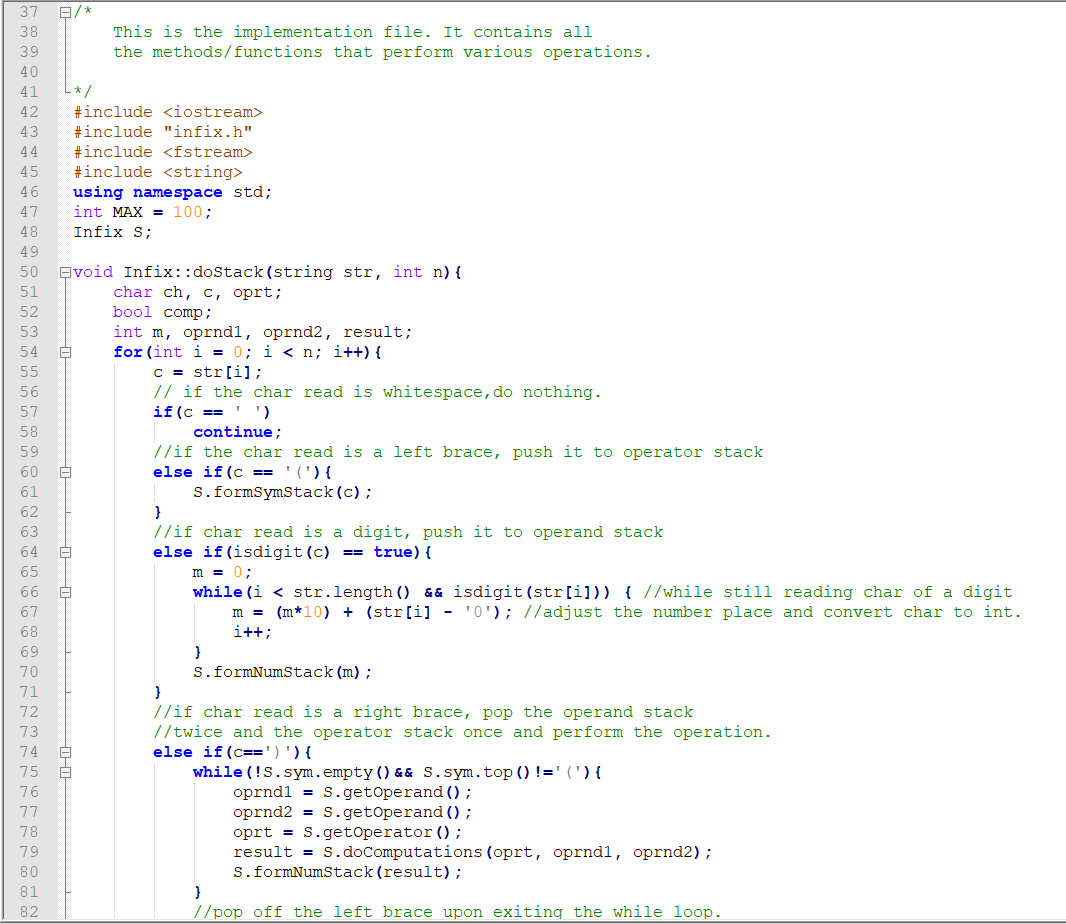
The main() function is the simplest and the shortest method for this project. It has a few lines of code in which few variables declared and outputs displayed along with input from the user, the file name. if the file name is correct, the implementation is processed and the expressions and their corresponding values are printed on the terminal. If the file name is wrong or the program is unable to locate the file name, a corresponding message is displayed on the terminal screen.

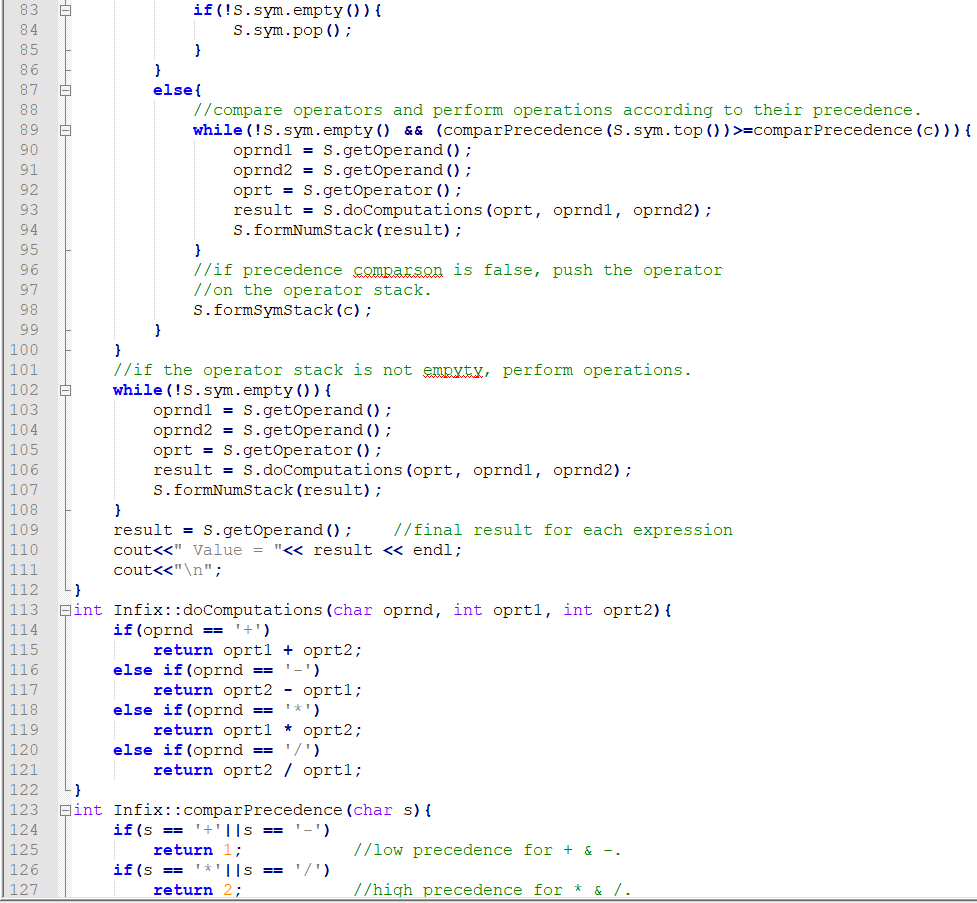
# **Code listing**

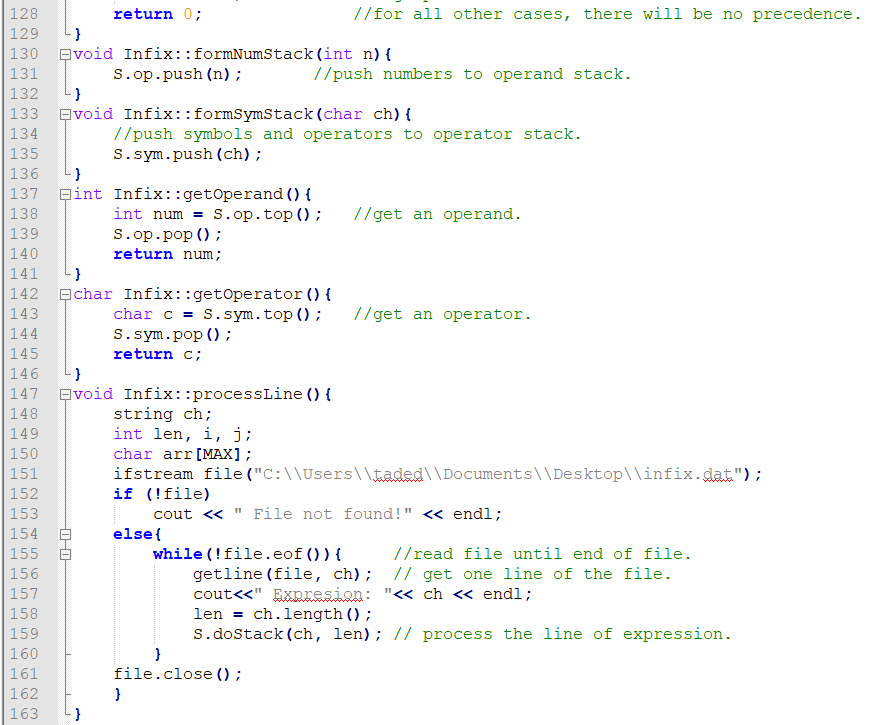
## **The header file (List.h)**



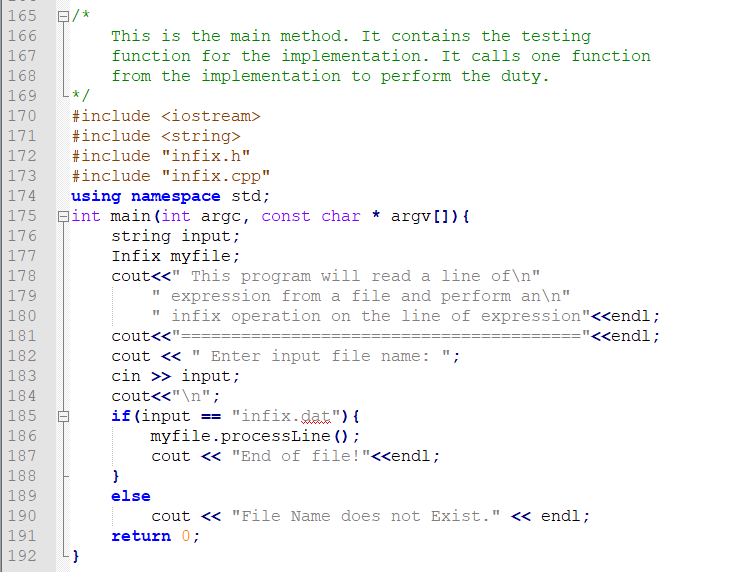
## **The implementation file (List.cpp)**





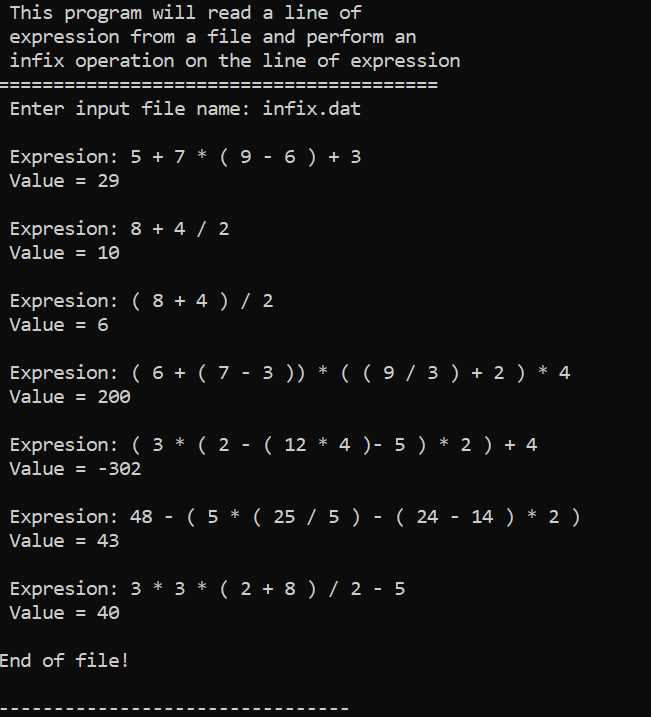


## **The testing file (main.cpp)**

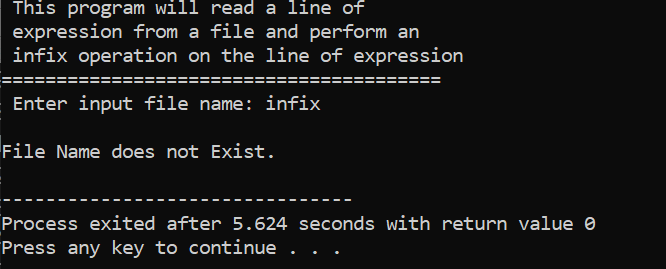


# **Test Results**

1. **Read and process a success.**



1. **Filename not correct**



# **User document**

This program can perform infix operations on expressions saved to a file. To run the program, you must perform the following steps.

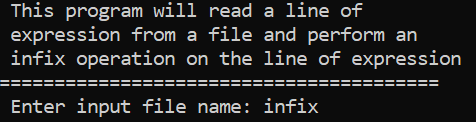
* First, save the file that contains the infix expressions to the following location.

F:\School\CSCI 301\My Projects ECE 591\Project 6\

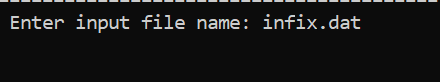
* Name the file infix.dat.
* Run the main.cpp. To compile and run the program, enter the following command to on the terminal window.

g++ -o main main.cpp

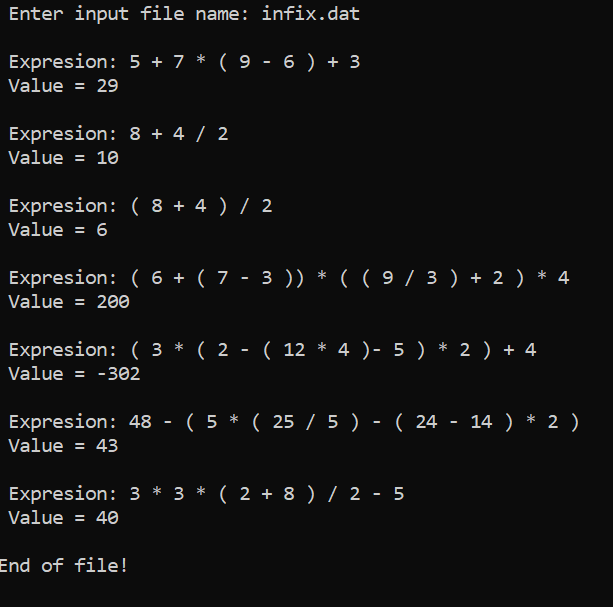
* The program will compile and open the following window:



* Once the window opens, enter the line infix.dat.
* Next, type the item you want to insert and then enter. For example, type 15 and enter.



* The program will run and display the output as shown below.



# **Summery**

The stack abstract data type is used in the implementation of the infix expression operation. The implementation used to stack operation one for character symbols and operators and another stack for numbers or operands. The two operands on the top of the operand stack are removed from the stack successively and the operator is removed from the operator stack and the operation is applied o the two operands. This process is repeated several times until the end of the expression is reached and the operator stack becomes empty.

I have improved the project by making it work for multidigit number inputs. The program works with any number of digits used. This was achieved by reding the digits of a multidigit number from left to right while the appropriate decimal place is obtained by multiplying by ten for each successive read of the digits. I believe this is a good advancement for the program which was required to work only for single-digit inputs. This program can also be slightly modified to work with postfix and prefix expressions.

By completing this project, I have gained a significant level of confidence and the necessary knowledge to work with stacks. This program could be further improved by redirecting the output stream to the end of the original file as a solution. it can also be made advanced by making it show each stem involved in simplifying the expression and obtain the result.