Preparatory data Structure (CSCI 591)

Project - VI

Evaluating General Infix Expressions

Submitted By: Taddese Erba

March 17, 2020

St. Cloud state university

Department of Computer Science

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

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 $(2 + (5 * (8 - 4) * 2)/ 2 + 3)$ 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

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

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.

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

Code listing

a. The header file (List.h)

```
/*
    The header file contains the class "Infix" that hosts
    the various public functions and two private objects.
        => The public functions perform different operations
            as specified in the implimentationfile.
        => The two private objects hold the stack of operators,
            symbols and intiger numbers.
        => Symbols and operators go to "sym" stack and integers go to "op" stack.
    Precondition: - The program works for single digit integers and operation that
                    involves infix expressions.
    Postcondition: - The program will read a file that has infix expressions,
                     read the expressions one line at a time, displays the
                     line of expression, computes the expression, and displays
                     the value of the expression.
*/
#include <iostream>
#ifndef _INFIX
#define _INFIX
#include <bits/stdc++.h>    //required header file by the stack operation.
using namespace std;
class Infix{
    private:
        stack <char> sym;   //holds symbols and operators.
        stack <int> op;     //holds integer operands.
    public:
        void doStack(string, int);  //function to perform the infix stack operation.
        int doComputations(char, int, int); //function to perform the computations
        void processLine();     //function to read file and process it as an array.
        void formNumStack(int);     //function to form the operannd stack
        void formSymStack(char);    //function to form the operator and braces.
        int getOperand();           //function to pop and return operands.
        char getOperator();         //function to pop and return operators.
        int comparPrecedence(char); //function to compare precedence of operators.
};
#endif
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

b. The implementation file (`List.cpp`)

```
/*
    This is the implementation file. It contains all
    the methods/functions that perform various operations.

*/
#include <iostream>
#include "infix.h"
#include <fstream>
#include <string>
using namespace std;
int MAX = 100;
Infix S;

void Infix::doStack(string str, int n){
    char ch, c, oprt;
    bool comp;
    int m, oprnd1, oprnd2, result;
    for(int i = 0; i < n; i++){
        c = str[i];
        // if the char read is whitespace,do nothing.
        if(c == ' ')
            continue;
        //if the char read is a left brace, push it to operator stack
        else if(c == '('){
            S.formSymStack(c);
        }
        //if char read is a digit, push it to operand stack
        else if(isdigit(c) == true){
            m = 0;
            while(i < str.length() && isdigit(str[i])) { //while still reading char of a digit
                m = (m*10) + (str[i] - '0'); //adjust the number place and convert char to int.
                i++;
            }
            S.formNumStack(m);
        }
        //if char read is a right brace, pop the operand stack
        //twice and the operator stack once and perform the operation.
        else if(c==')'){
            while(!S.sym.empty()&& S.sym.top()!='('){
                oprnd1 = S.getOperand();
                oprnd2 = S.getOperand();
                oprt = S.getOperator();
                result = S.doComputations(oprt, oprnd1, oprnd2);
                S.formNumStack(result);
            }
            //pop off the left brace upon exiting the while loop.
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

```
83              if(!S.sym.empty()){
84                  S.sym.pop();
85              }
86          }
87          else{
88              //compare operators and perform operations according to their precedence.
89              while(!S.sym.empty() && (comparPrecedence(S.sym.top())>=comparPrecedence(c))){
90                  oprnd1 = S.getOperand();
91                  oprnd2 = S.getOperand();
92                  oprt = S.getOperator();
93                  result = S.doComputations(oprt, oprnd1, oprnd2);
94                  S.formNumStack(result);
95              }
96              //if precedence comparson is false, push the operator
97              //on the operator stack.
98              S.formSymStack(c);
99          }
100     }
101     //if the operator stack is not empyty, perform operations.
102     while(!S.sym.empty()){
103         oprnd1 = S.getOperand();
104         oprnd2 = S.getOperand();
105         oprt = S.getOperator();
106         result = S.doComputations(oprt, oprnd1, oprnd2);
107         S.formNumStack(result);
108     }
109     result = S.getOperand();    //final result for each expression
110     cout<<" Value = "<< result << endl;
111     cout<<"\n";
112 }
113 int Infix::doComputations(char oprnd, int oprt1, int oprt2){
114     if(oprnd == '+')
115         return oprt1 + oprt2;
116     else if(oprnd == '-')
117         return oprt2 - oprt1;
118     else if(oprnd == '*')
119         return oprt1 * oprt2;
120     else if(oprnd == '/')
121         return oprt2 / oprt1;
122 }
123 int Infix::comparPrecedence(char s){
124     if(s == '+'||s == '-')
125         return 1;           //low precedence for + & -.
126     if(s == '*'||s == '/')
127         return 2;           //high precedence for * & /.
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

```
128     return 0;               //for all other cases, there will be no precedence.
129 }
130 void Infix::formNumStack(int n){
131     S.op.push(n);       //push numbers to operand stack.
132 }
133 void Infix::formSymStack(char ch){
134     //push symbols and operators to operator stack.
135     S.sym.push(ch);
136 }
137 int Infix::getOperand(){
138     int num = S.op.top();   //get an operand.
139     S.op.pop();
140     return num;
141 }
142 char Infix::getOperator(){
143     char c = S.sym.top();   //get an operator.
144     S.sym.pop();
145     return c;
146 }
147 void Infix::processLine(){
148     string ch;
149     int len, i, j;
150     char arr[MAX];
151     ifstream file("C:\\Users\\taded\\Documents\\Desktop\\infix.dat");
152     if (!file)
153         cout << " File not found!" << endl;
154     else{
155         while(!file.eof()){     //read file until end of file.
156             getline(file, ch);  // get one line of the file.
157             cout<<" Expresion: "<< ch << endl;
158             len = ch.length();
159             S.doStack(ch, len); // process the line of expression.
160         }
161     file.close();
162     }
163 }
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

c. The testing file (main.cpp)

```
/*
    This is the main method. It contains the testing
    function for the implementation. It calls one function
    from the implementation to perform the duty.
*/
#include <iostream>
#include <string>
#include "infix.h"
#include "infix.cpp"
using namespace std;
int main(int argc, const char * argv[]){
    string input;
    Infix myfile;
    cout<<" This program will read a line of\n"
        " expression from a file and perform an\n"
        " infix operation on the line of expression"<<endl;
    cout<<"========================================"<<endl;
    cout << " Enter input file name: ";
    cin >> input;
    cout<<"\n";
    if(input == "infix.dat"){
        myfile.processLine();
        cout << "End of file!"<<endl;
    }
    else
        cout << "File Name does not Exist." << endl;
    return 0;
}
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

Test Results

1. Read and process a success.

```
This program will read a line of
expression from a file and perform an
infix operation on the line of expression
========================================
Enter input file name: infix.dat

Expresion: 5 + 7 * ( 9 - 6 ) + 3
Value = 29

Expresion: 8 + 4 / 2
Value = 10

Expresion: ( 8 + 4 ) / 2
Value = 6

Expresion: ( 6 + ( 7 - 3 )) * ( ( 9 / 3 ) + 2 ) * 4
Value = 200

Expresion: ( 3 * ( 2 - ( 12 * 4 )- 5 ) * 2 ) + 4
Value = -302

Expresion: 48 - ( 5 * ( 25 / 5 ) - ( 24 - 14 ) * 2 )
Value = 43

Expresion: 3 * 3 * ( 2 + 8 ) / 2 - 5
Value = 40

End of file!

------------------------------
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

2. Filename not correct

```
 This program will read a line of
 expression from a file and perform an
 infix operation on the line of expression
=======================================
 Enter input file name: infix

File Name does not Exist.


--------------------------------
Process exited after 5.624 seconds with return value 0
Press any key to continue . . .
```

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:

```
 This program will read a line of
 expression from a file and perform an
 infix operation on the line of expression
=======================================
 Enter input file name: infix
```

- 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.

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

```
Enter input file name: infix.dat
```

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

```
Enter input file name: infix.dat

Expresion: 5 + 7 * ( 9 - 6 ) + 3
Value = 29

Expresion: 8 + 4 / 2
Value = 10

Expresion: ( 8 + 4 ) / 2
Value = 6

Expresion: ( 6 + ( 7 - 3 )) * ( ( 9 / 3 ) + 2 ) * 4
Value = 200

Expresion: ( 3 * ( 2 - ( 12 * 4 )- 5 ) * 2 ) + 4
Value = -302

Expresion: 48 - ( 5 * ( 25 / 5 ) - ( 24 - 14 ) * 2 )
Value = 43

Expresion: 3 * 3 * ( 2 + 8 ) / 2 - 5
Value = 40

End of file!
```

Taddese Erba
Section – I
Project – Six
Due: March 17, 2020

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.