Eric Blasko

Data Structures CSE330

Winter 2018

Lab 1 - Infix to Postfix

* **Status**

100% complete

* **Complexity Analaysis**

O(n) where n is the size of input.

* **Source code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Eric Blasko

\*lab1.cpp

\*01/17/2018

\*This program runs an algorithm to convert infix expressions into a postfix expression.

\*A stack is used to hold operators in order to reverse the order of the operators in

\*the expression.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool IsOperator(char a);

int Precedence(char a);

void postfix(string);

//Main function to get infix expression and run algorithm to convert to postfix expression

int main()

{

string infix = "";

cout << "Enter Expression: ";

getline(cin,infix);

postfix(infix);

return 0;

}

//This function determines if the current char is a operator and returns a boolean

bool IsOperator(char a)

{

if(a == '+' ||

a == '-' ||

a == '\*' ||

a == '/' ||

a == '(' ||

a == ')')

return true;

else

return false;

}

//This function returns a numerical value based on operators precedence scale

int Precedence(char a)

{

if(a == '(' || a == ')')

return 1;

else if(a == '+' || a == '-')

return 2;

else if(a == '\*' || a == '/')

return 3;

else

{

cerr << "Invalid input" << endl;

return 0;

}

}

//This function takes a infix expression and prints the converted postfix expression

void postfix(string expression)

{

stack<char> operators;

while(!cin.eof())

{

for(int i = 0; i < expression.size(); i++)

{

if(IsOperator(expression[i]) == false)

{

cout << expression[i];

}

else

{

if(expression[i] == '(')

{

operators.push(expression[i]);

}

else if(expression[i] == ')')

{

while (!operators.empty() && operators.top() != '(')

{

cout << operators.top();

if(!operators.empty())

{

operators.pop();

}

else

{

cerr << "no matching (" << endl;

}

}

operators.pop();

}

else if(IsOperator(expression[i]) == true)

{

if(operators.empty() || Precedence(operators.top()) < Precedence(expression[i]))

{

operators.push(expression[i]);

}

else

{

while(!operators.empty() && Precedence(operators.top()) >= Precedence(expression[i]))

{

cout << operators.top();

operators.pop();

}

operators.push(expression[i]);

}

}

else

{

cerr << "Invalid Input" << endl;

}

}

}

while(!operators.empty())

{

cout << operators.top();

operators.pop();

}

cout << endl << "Enter Expression: ";

getline(cin,expression);

}

}

* **Sample Run**

Script started on 2018-01-15 12:34:24-0800

]0;005670557@csusb.edu@csevnc:~/lab1[005670557@csusb.edu@csevnc lab1]$ g++ -c lab1.cpp

]0;005670557@csusb.edu@csevnc:~/lab1[005670557@csusb.edu@csevnc lab1]$ g++ lab1.cpp

]0;005670557@csusb.edu@csevnc:~/lab1[005670557@csusb.edu@csevnc lab1]$ ./a.out

Enter Expression: a+b-c

ab+c-

Enter Expression: a+b\*c

abc\*+

Enter Expression: (a+2)/(5-d)

a2+5d-/

Enter Expression: a+((b-d c)\*d)/e

abc-d\*e/+

Enter Expression: ^C

]0;005670557@csusb.edu@csevnc:~/lab1[005670557@csusb.edu@csevnc lab1]$ exit

Script done on 2018-01-15 12:35:44-0800

* **Discussion**

By adding another stack which only holds values, it could be possible to solve

expressions as it performs the postfix algorithm. Each time the operator stack encountered

a operator with a lower presidence it would pop off two from the values stack and perform that

operator action. An example would be

2 + ((5-2) \* 3)/3

2 -> value stack

+ -> operand stack

( -> operator stack presedence 0

( -> operator stack presedence 0

5 -> value stack

- -> operator stack presedence 1

2 -> value stack

) -> remove 5,2 from value stack and -. 5-2 = 3 -> value stack

\* -> operator stack presedence 2

3 -> value stack

) -> remove 3,3 from value stack and \*. 3\*3 = 9 -> value stack

/ -> operator stack presedence 2

3 -> value stack

EOF - remove 9,3 from value stack and /. 9/3 = 3 -> value stack

remove 3,2 from value stack and +. 3+2 = 5 = answer