/\* Assignment: PrefixToPostFix

File: Main.cpp

Author: Ryan Wood

Created On: 3/3/2018

\*/

#include "../Containers/GenericContainers/Stack/Stack.h"

#include "../Containers/GenericContainers/Queue/Queue.h"

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

ifstream inFile ("Infix.in");

ofstream outFile("postFix.out");

const char endToken = '#';

/\* Function: infixPriority(char)

Purpose: determines the infix priority of the given character.

each operator has an infix priority and a stack

priority, which may not be the same

Parameters: the operator

Return: the infix priority

\*/

int infixPriority(char op)

{

int priority = 0;

switch(op)

{

case '^':

case '(':

priority = 3;

break;

case endToken:

case ')':

priority = 0;

break;

case '+':

case '-':

priority = 1;

break;

case '\*':

case '/':

priority = 2;

break;

default:

priority = -1;

}

return priority;

}

/\* Function: stackPriority(char)

Purpose: determines the stack priority of the given character.

Each operator has a infix priority and a stack priority

which may not be the same.

Parameters: the operator

Return: the stack priorty

\*/

int stackPriority(char op)

{

int priority = 0;

switch(op)

{

case endToken:

case '(':

priority = 0;

break;

case '+':

case '-':

priority = 1;

break;

case '\*':

case '/':

priority = 2;

break;

case '^':

priority = 3;

break;

default:

priority = -1;

}

return priority;

}

/\* Function: isValidOperator(char)

Purpose: determines whether the given character is

one of the operators this program supports

Parameters: the character

Return: whether it is valid as an arithmetic operator

\*/

bool isValidOperator(char op)

{

bool isValid = false;

switch(op)

{

case '(':

case ')':

case '+':

case '-':

case '\*':

case '/':

case '^':

isValid = true;

break;

default:

isValid = false;

}

return isValid;

}

/\* Function: isSpace(char)

Purpose: determines whether the given character is

a space as spaces are skipped by this program

Parameters: the character

Return: whether the character is a space

\*/

bool isSpace(char op)

{

return (op == ' ');

}

/\* Function: isOperand(char)

Purpose: determines whether the given character is

a number between 0 and 9. As designed, this program

only handles single digit numbers

Parameters: the character

Return: whether the character is a valid number

\*/

bool isOperand(char op)

{

return(op >= 48 && op <= 57);//0 through 9 falls within this ascii range

}

/\* Function: isRightParen(char)

Purpose: determines whether the given character

is a closing parenthasis

Parameters: the character

Return: whether it is a right parenthasis

\*/

bool isRightParen(char op)

{

return(op == ')');

}

/\* Function: isLeftParen(char)

Purpose: determines whether the given character is

an opening parenthasis

Paramters: the character

Return: whether it is a left parenthasis

\*/

bool isLeftParen(char op)

{

return(op == '(');

}

/\* Function: getPostFix(string)

Purpose: converts the given infix equation string

to be a postfix equation and returns

the result.

Parameters: the infix equation

Return: the postfix equation

\*/

string getPostFix(string inFix)

{

Queue<char, 100> que;

Stack<char, 100> stack;

string postFix = "";

int index = 0;

char token;

//1. Initialize stack by pushing EndToken

stack.push(endToken);

//2. Dequeue token from infix expression

for(index = 0; index < inFix.length(); index++)

{

token = inFix[index];

//3. if(token)

if(isSpace(token))

continue;

//a. operand: enqueue it on the postfix expression

if(isOperand(token))

que.push(token);

else if(isValidOperator(token))

{

/\*b. right paren, pop entries from operator stack and enque

them on postfix until a matching left paran is popped.\*/

if(isRightParen(token))

{

while(!isLeftParen(stack.top()) && stack.top() != endToken)

{

que.push(stack.top());

stack.pop();

}

if(isLeftParen(stack.top()))

stack.pop();//pop the left paren off

}

/\*d. otherwise, pop from the stack and enque on the postfix queue

operators whose stack priority is greater than or equal to the

infix priority of the token.\*/

else if(infixPriority(token) <= stackPriority(stack.top()))

{

while(infixPriority(token) <= stackPriority(stack.top()) && stack.top() != endToken)

{

que.push(stack.top());

stack.pop();

}

stack.push(token);

}

else

stack.push(token);

}//else if(isValidOperator(token))

else if(token == endToken)

{

/\*c. EndToken, pop all entries that remain on the stack and enque

them on the postfix queue.\*/

while(!stack.empty() && stack.top() != endToken)

{

que.push(stack.top());

stack.pop();

}

}//token is endToken

}//for(index = 0; index < inFix.length(); index++)

while(!que.empty())

{

postFix += que.top();

que.pop();

}

return postFix;

}

/\* Function: getInFix()

Purpose: Propmts the user for an infix expression

This function is used in conjunction with

getPostFix and displayResults for testing

Return: the equation input by the user at the keyboard

\*/

string getInFix()

{

string inFix;

cout << "Please enter an equation to be converted. End the equation with #: ";

getline(cin, inFix);

inFix += endToken;

return inFix;

}

/\* Function: displayResults(string, string)

Purpose: outputs the given infix and postfix strings

to both the console and the output file

Parameters: the infix equation, the postfix equation

\*/

void displayResults(string inFix, string postFix)

{

cout << "The original infix equation was " << inFix << endl;

cout << "The postfix equation is " << postFix << endl << endl;

outFile << "The original infix equation was " << inFix << endl;

outFile << "The postfix equation is " << postFix << endl << endl;

}

/\* Function: processInputFile()

Purpose: processes the infix expressions defined

in the input file, converts them each to

postfix format and displays the results

to the console and writes them to the file

\*/

void processInputFile()

{

string inFix;

string postFix;

while(getline(inFile, inFix))

{

inFix += endToken;

postFix = getPostFix(inFix);

displayResults(inFix, postFix);

}

}

int main()

{

processInputFile();

inFile.close();

outFile.close();

return 0;

}

The original infix equation was 5 + 7#

The postfix equation is 57+

The original infix equation was 6 ^ 2 + 1#

The postfix equation is 62^1+

The original infix equation was 3 \* 8 + 6#

The postfix equation is 38\*6+

The original infix equation was 5 ^ 4 - 6 ^ 4#

The postfix equation is 54^64^-

The original infix equation was 5 - 3 \* 7 / 4 - 9#

The postfix equation is 537\*4/-9-

The original infix equation was 8 \* ( 9 / 3 - 2 ) / 4 + 5 \* 6#

The postfix equation is 893/2-\*4/56\*+

The original infix equation was 5 ^ 3 \* 4 + ( 2 + ( 9 \* 8 / ( 2 \* 6 \* ( 8 / 4 ) ) ) ^ 2 \* 8 - 5 ) / 5 ^ 2 - 4#

The postfix equation is 53^4\*298\*26\*84/\*/2^8\*+5-52^/+4-

The original infix equation was 5 - 3 \* 8 / 2 ^ 3#

The postfix equation is 538\*23^/-

The original infix equation was 3 ^ 3 ^ 2 \*3#

The postfix equation is 33^2^3\*

The original infix equation was 8 \* ( 6 / 3 - 2 ) / 4 + 5 \* 6 +3#

The postfix equation is 863/2-\*4/56\*+3+

The original infix equation was ( ( ( ( ( 9 \* 5 ) ) ) ) )#

The postfix equation is 95\*