Joseph Baruch

CS121

Professor: Bruce Bolden

Program 3

Due: 3/3/2023

Program Design

Objective

Apply stacks and classes in a program to test our current knowledge from material taught during lecture. Through this, become more familiar and gain experience with the typical stack ADT and classes.

Program Description

Infix notation, such as (2 + 5), can be more complicated for computers to process since the operator is between the operands. So, computers typically convert a given expression from infix to postfix ( 2 5 +) internally using compilers or interpreters.

Using the given infix to postfix expression algorithm, write a program exactly matching the given algorithm to convert infix expressions to postfix. Make sure all requirements listed below are met.

Requirements

* Exactly follow the algorithm given in the program assignment description document so that the user interface directly models it.
* Loop continuously until the program process the correct output.
* Must use a linked list and stack class that work with each other (Stack made up of linked lists).
* Must have the file with the algorithm (main file), separate from the link declaration, stack declaration, link implementation and stack implementation files.
* Program should be able to evaluate all the following inputs.
  + 3+4
  + (3+4)
  + (2+3)\*4
  + 2+((5)\*3)
  + 2\*((3+4\*7)/2+4)

Program Contents

* Classes
  + Link.h: Linked List class that works with Stack class.
  + Stack.h: Stack class used within main and with Linked List class.
* Methods (link.h)
  + Struct node:
    - Char info: info will be assigned a value in the infix expression.
    - Node \* next: used for linked list.
  + Tyedef node \* nodeptr: creates a nodeptr variable name.
  + Nodeptr head: declares a nodeptr named head
  + Int count: will be used to keep count of how many nodes in the linked list.
  + LinkedList(): Constructor which sets head = NULL and count = 0.
  + ~LinkedList(): clears the linked list and deallocates memory.
* Methods (stack.h)
  + LinkedList topPtr: Declares a variable of type LinkedList named topPtr.
* Methods (Program3.cpp)
  + Char AddEndParen( char a[], int &s ): Adds a ‘)’ to the end of the Infix expression.
  + Bool OpPrecedence( char &a, char &k): Checks the operator precedence of Token and the character at the top of the stack.
  + Char Infix[]: Character array for the users input (infix expression).
  + Int SizeIn: Assigned the value of the size of the infix expression.
  + Int CurrInCell: Current position of the infix expression
  + Int CurrPoCell: Current position of the postfix expression.
  + Char Token: Current value of the infix expression.
  + Char c: Set as character popped off the top of the stack.
  + Char Topstack:
  + Stack S1: Instance of the stack class.
  + Char Postfix[]: Charater array for the output (postfix expression).
* Functions (link.cpp)
  + Void AddNode( char x ): Adds a node to the linked list.
  + Void DeleteNode( char x ): Deletes a node from the linked list.
  + Char FirstNode(): Returns the first node of the linked list.
  + Bool IsInList( char x ): Checks to see if argument char is in the linked list.
  + Int Size(): Returns the size of the linked list.
* Functions (stack.cpp)
  + Stack(): Does nothing.
  + ~Stack(): Deallocates stack by deallocating the link list in each stack location.
  + Void Push( char n ): Adds char in argument to the top of the stack.
  + Char Pop(): Removes and returns the char on the top of the stack.
  + Char Peek(): Returns the char on the top of the stack (does not remove).
  + Int IsEmpty(): Checks to see if the stack is empty.
* Functions (Program3.cpp)
  + Char AddEndParen( char a[], int &s); : Adds a ‘)’ to the end of the infix expression. This function takes the infix expression and size of char array as arguments.
  + Bool OpPrecedence( char &a, char &k ); : Checks the precedence of the current infix expression char and the top of the stack. Returns true if token value is less than or greater than stack value.

Program Operation Description

1. Get input from the user using a get() function. The infix input should be stored in a char array.
2. Push a ‘(‘ onto the stack.
3. Add a ‘)’ onto the infix expression by going through the array until it reaches ‘\0’. After this go to the next cell and declare it as ‘\0’.
4. Find the size of the infix expression by using a counter in the above process and then declare the postfix expression to the same size of the infix expression.
5. Begin while loop until the stack is empty.
6. Get the next token from the infix expression.
7. Determine the value of the token and using the provided algorithm, preform the correct process for the determined token.
   * If the token is a ‘(‘, push it onto the stack.
   * If the token is a number, add the number to the end of the postfix expression.
   * If the token is a ‘)’, pop the top element of the stack.
     + While that element is not a ‘(‘.
       - Add it to the end of the postfix expression.
       - Grab the next element from the stack and go until while loop constraints are not met.
       - This works because if the top stack element is a ‘(‘ then the program ends.
   * If the token is an operator
     + While the operator is of greater precedence then the top of the stack operator
       - Pop the top of the stack off and add it to the end of postfix expression.
     + If the while loop doesn’t process the token will be added to the stack.
8. Print output because the program should be complete.

Programming Log

Total Time: ~16 hours

2/23/2023: 15 minutes

* Read programming assignment guidelines.

2/24/2023: 30 minutes

* Reread programming assignment guidelines.
* Begin Program Design document.
* Begin programming log and got it up to date.
* Try to understand assignment.

2/25/2023: 15 min

* Continue program design.

2/27/2023: 1.5 hours

* Began working on the main program and adding all the pieces. I spent most of the day being very confused on what I had to do. I decided to try to get to a point where I had the outline of the functions complete so the next day when I went into office hours, I could get most of the confusion complete.

2/28/2023: 3 hours

* Spent the morning preparing for going into office hours so that I had comments prepared, and I could focus on asking the right questions.
* Went into office hours and asked some basic questions on how to work getline() with character arrays. I didn’t really get to the point where it worked.
* Also asked if we would use the link.h, stack.h, link.cpp, stack.cpp exactly from the notes. Bolden said yes but change some things like the int to char.
* The last thing I asked and got clarification on was how to compile. Bolden told me to compile all of the .cpp files at once.
* After office hours I got to work implementing all the files working together. I ended up getting most of the program set up but ended up making the mistake of changing the link.cpp file. Later on I had to go in and changed this.

3/1/2023: 4 hours

* Added function to check precedence of different variables.
* Added function to add a ‘)’ to the end of the Infix expression.
* Found a way to include a function (using a library) to check to see if a char is a digit or not.
* Visited Bruce Bolden in his office hours to find out what form the infix and postfix expression should take. Before this point I thought that the postfix should take the form of a linked list so I had to revert my code so that it would work for a char array.
* Once I fixed it after this point, I ended for the day with everything put together so that I could focus on other things. I decided that the next day I would sweep my program for errors, try to compile it and get rid of all the errors the compiler gives me.

3/2/2023: 8 hours

* I did some passing by reference to some functions and had to relearn some of those concepts. I reminded myself that you don’t need to pass variables by reference because they automatically are.
* Figured out that file guards are necessary to protect program from the stack.h and link.h files being read more than 1 time.
* I had to redo link.h and stack.h because I initially changed them because I thought that not all the functions in link.h had to be used. I also adjusted stack.h references to link.h so I had to fix those back.
* Fixed all the issues within the main program so that it finally compiled. This included removing variables that I initially made when I first started but changed later.
* Visited Bruce Bolden in his office hours to find the solution to file guard problem, found out that you don’t need to declare an instance of the linked list class and other issues.
* While experimenting where I was getting some other errors I noticed a placed c in an argument to be added to the stack when it should have been the token. This was a result of not following the algorithm correctly. This resolved a ton of issues.
* I was having a lot of issues with segmentation faults. I determined the issue was because I was initializing the postfix expression to be slightly smaller or larger than the infix expression. This would either result in a segmentation fault within the printing process or a segmentation fault within the while loop of the main program. After this issue was resolved my program compiled correctly and gave the correct outputs.
* Finished Programming Log and Program Design.
* Cleaned up code to look nice and added any comments that were missed.

3/3/2023: 5 min

* Assignment due
* Turn in printed deliverables at 9:30 in class.

Outputs

-bash-4.2$ g++ Program3.cpp link.cpp stack.cpp

-bash-4.2$ ./a.out

Infix: 3+4

Size of Infix: 3

Postfix: 34+

-bash-4.2$ g++ Program3.cpp link.cpp stack.cpp

-bash-4.2$ ./a.out

Infix: (3+4)

Size of Infix: 5

Postfix: 34+

-bash-4.2$ g++ Program3.cpp link.cpp stack.cpp

-bash-4.2$ ./a.out

Infix: (2+3)\*4

Size of Infix: 7

Postfix: 23+4\*

-bash-4.2$ g++ Program3.cpp link.cpp stack.cpp

-bash-4.2$ ./a.out

Infix: 2+((5)\*3)

Size of Infix: 9

Postfix: 253\*+

-bash-4.2$ g++ Program3.cpp link.cpp stack.cpp

-bash-4.2$ ./a.out

Infix: 2\*((3+4\*7)/2+4)

Size of Infix: 15

Postfix: 2347\*+24+/\*

Program

\* Joseph Baruch

Program 3

CS 121

Bruce Bolden

Due: 3/3/2023

\*/

#include <iostream>

#include <ctype.h>

using namespace std;

#include "stack.h"

char AddEndParen( char a[], int &s, int &P );

bool OpPrecedence( char &a, char &k );

int main(){

char Infix[250];

int SizeIn = 0;

int CurrInCell = 0;

int CurrPoCell = 0;

int ParenNum = 0;

char Token, c, TopStack;

Stack S1; // declare an instance of the Stack class

// get character string

cout << "Infix: ";

cin.get( Infix, 250 );

// Add a ')' to the end of infix

AddEndParen( Infix, SizeIn, ParenNum );

S1.Push('('); // Push a '(' onto the stack

cout << "Size of Infix: " << SizeIn << endl;

SizeIn = SizeIn - ParenNum;

// Initialize Postfix to size of infix expression

char Postfix[SizeIn];

while( !S1.IsEmpty() ){ // while the stack is not empty

// get (read) the next token from the infix expression

Token = Infix[CurrInCell];

if( Token == '(' ){ // if the token is a '('

S1.Push( Token ); // Push the token onto the stack

}else if( isdigit(Token) ){ // else if the token is a number

// Add the number to the end of the postfix expression

Postfix[CurrPoCell] = Token;

CurrPoCell++;

}else if( Token == ')' ){ // else if the token is a ')'

c = S1.Pop(); // Pop the element c from the stack

while( c != '(' ){ // while c is not a '('

// Place c at the end of the postfix expression

Postfix[CurrPoCell] = c;

CurrPoCell++;

c = S1.Pop(); // Pop another element c from the stack

}

}else{ // else (the token must be an operator)

TopStack = S1.Peek();

// check precedence

while( OpPrecedence( Token, TopStack )){

c = S1.Pop(); // pop the element c from the stack

// Place c at the end of the postfix expression

Postfix[CurrPoCell] = c;

CurrPoCell++;

}

S1.Push( Token ); // Push the token onto the stack

}

CurrInCell++; // increment current infix cell

}

// Print postfix expression

cout << "Postfix: ";

for( int i = 0; i <= CurrPoCell; i++ ){

cout << Postfix[i];

}

cout << endl;

return 0;

}

char AddEndParen( char a[], int &s, int &P ){

while( a[s] != '\0' ){ // Go until cell is empty

if( a[s] == '(' || a[s] == ')' ){

P++;

}

s++;

}

a[s] = ')'; // Add a ')' to the end of the infix expression

s++;

a[s] = '\0'; // Set cell after ')' to '\0'

s--;

}

bool OpPrecedence( char &a, char &k ){

// checks precedence of stack operator and current infix operator

int T; // value of Token operator

int S; // value of Stack operator

if( k == '(' || k == ')' ){

return false;

}

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

T = 1; // if Token operator is '\*' or '/"

}

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

T = 0; // if Token operator is '+' or '-'

}

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

S = 1; // if Stack operator is '\*' or '/'

}

if( k == '-' || k == '+' ){

S = 0; // if Stack operator is '+' or '-'

}

if( S >= T ){ // if Token value is <= Stack operator

return true;

}

if( T > S ){ // if Token value is > Stack operator

return false;

}

}

/\* link.h --- interface for a linked list of integers class. \*/

#ifndef LINK\_H

#define LINK\_H

//#include <bool.h>

#include <iostream>

using namespace std;

class LinkedList{

private:

struct node{

char info;

node \* next;

};

typedef node \* nodeptr;

nodeptr head;

int count;

public:

// Constructor

LinkedList(){

head = NULL;

count = 0;

}

// Destructor

~LinkedList(){

nodeptr p = head, n;

while( p != NULL ){

n = p;

p = p->next;

delete n;

}

}

// Add a node onto the front of the linked list.

void AddNode( char x );

// Delete the first node.

void DeleteNode( char x );

// Return the first node found in the list

char FirstNode();

// Return true if there is a node with the value x.

bool IsInList( char x );

// Return a count of the number of nodes in the list.

int Size();

};

#endif

/\* link.cpp: Class for a sorted linked list of integers. \*/

#include <iostream>

#include "link.h"

using namespace std;

// Add an item to the FRONT of the list

void LinkedList::AddNode( char x ){

nodeptr n; // allocate new node

n = new node;

n->info = x;

count++;

if( head == NULL ){

head = n;

n->next = NULL;

}else{

nodeptr tmp = head;

n->next = tmp;

head = n;

}

}

void LinkedList::DeleteNode( char x ){

nodeptr prev, curr = head;

while( curr != NULL && x > curr->info ){

prev = curr;

curr = curr->next;

}

if( x == curr->info ){

if( curr == head ){

head = head->next;

}else{

prev->next = curr->next;

}

delete curr;

count--;

}

}

char LinkedList::FirstNode(){

return head->info;

}

bool LinkedList::IsInList( char x ){

nodeptr p = head;

while( p != NULL && x > p->info ){

p = p->next;

}

return (x == p->info);

}

int LinkedList::Size(){

return count;

}

/\* stack.h Stack class interface \*/

#ifndef STACK\_H

#define STACK\_H

#include <iostream>

#include "link.h"

using namespace std;

class Stack{

public:

Stack();

~Stack();

void Push( char n );

char Pop();

char Peek();

int IsEmpty();

private:

LinkedList topPtr;

};

#endif

/\* stack.cpp --- Definition of Stack class member functions. \*/

#include <iostream>

#include "stack.h"

using namespace std;

Stack::Stack(){

}

Stack::~Stack(){

// delete topPtr;

while( !IsEmpty() ) {

char n = topPtr.FirstNode();

topPtr.DeleteNode( n );

}

}

void Stack::Push( char n ){

topPtr.AddNode( n );

}

char Stack::Pop(){

char n = topPtr.FirstNode();

topPtr.DeleteNode( n );

return n;

}

char Stack::Peek(){

char n = topPtr.FirstNode();

return n;

}

int Stack::IsEmpty(){

int n = topPtr.Size();

return (n == 0);

}