Chris Dang Lab Experience Nine

Do the following problems:

Pages 384-385:

**2 – 10 evens, 16 – 22 evens, 24, 26**

Programming Problem:

You are to implement the algorithm, **Evaluate PostFix Expressions,**  found on page 373

to solve **problem number 12 on page 387**.

In exercise 1-11, assume that a = 7.0, b = 4.0, c = 3.0, and d = -2.0. Evaluate the postfix expression.

2. a b c + / d \*

7 4 3 + / -2 \*

7 7 / -2 \*

1 -2 \*

-2

4. a b + c + d +

7 4 + 3 + -2 +

11 3 + -2 +

14 -2 +

12

6. a b c + + d +

7 4 3 + + -2 +

7 7 + -2 +

14 -2 +

12

8. a b - c - d -

7 4 - 3 - -2 -

3 3 - -2 -

0 -2 -

2

10. a b c - - d -

7 4 3 - - -2 -

7 1 - -2 -

6 -2 -

8

16. (( a + ( b / c ) ) + d )

(( a + (b c / ) + d

(( a ( b c / + + d )

(( a ( b c / + d +

a b c / + d +

18. ( a + ( b / ( c + d ) ) )

( a + ( b / ( c d + ) )

( a + ( b ( c d + / )

( a ( b ( c d + / +

a b c d + / +

20. ( ( a - b ) \* ( c - ( d + e ) ) )

( ( a b - \* ( c - ( d + e ) ) )

( ( a b - \* ( c - ( d e + ) )

( ( a b - \* ( c ( d e + - )

( ( a b - ( c ( d e + - \*

a b - c d e + - \*

22. ( a - ( b - ( c - ( d - e ) ) ) )

( a - ( b - ( c - ( d e - ) ) )

( a - ( b - ( c ( d e - - ) )

( a - ( b ( c ( d e - - - )

( a ( b ( c ( d e - - - -

a b c d e - - - -

24.

Expression Stack Display

c - d + e / a b +

d + e - a b + c /

a b + c / d - e +

26.

Expression Stack Display

(

/

- d ) \* e + a b c

/

\* e + a b c d - /

\*

e + a b c d - /

a b c d - / e \* +

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

// Programmer: Chris Dang Class: CSCI 1107 Spring 2015

//

// Description: This program takes in a post fix expression and evaulates

// the expression while showing the process of pushing to and popping from

// the stack.

// |

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

#include <iostream>

#include <string>

using namespace std;

#include "LStack.h"

const string TOKEN = " Token = " ;

const string PUSH = " Push " ;

const string POP = " Pop " ;

int main() {

Stack st ; // initialize empty stack for operators

string expression ;

cout << "Please enter a postfix expression: " ;

getline(cin, expression) ; // get expression into string

// loop through expression from start to end

for (int i = 0; expression[i] != expression[expression.size()]; i++) {

int token = expression[i] ; // holds a character for processing

if (token != 32) { // skips spaces

// 48 is the ascii value of 0, and 57 is the ascii value of 9

if (token >= 48 && token <= 57) { // if token is operand

token -= '0' ; // subtract ascii val 48 to get

cout << TOKEN << token ; // true integer value

st.push(token) ; // push operands to stack

cout << PUSH << token ;

}// end if

else if (token == '+') {

cout << TOKEN << static\_cast<char>(token) ; // Display operator

cout << POP << st.top() ; // Display operand 1

int sum = st.top() ; // get operand 1

st.pop() ; // pop op 1

cout << POP << st.top() ; // Display operand 2

sum += st.top() ; // add operand 2

st.pop() ; // pop op 2

st.push(sum) ; // push sum

cout << PUSH << sum ; // Display sum

} // end else if addition operator

else if (token == '-') {

cout << TOKEN << static\_cast<char>(token) ; // Display operator

cout << POP << st.top(); // Display operand 2

int op2 = st.top() ; // get operand 2

st.pop() ; // pop op 2

cout << POP << st.top() ; // Display operand 1

int op1 = st.top(); // get operand 1

st.pop() ; // pop op 1

op1 -= op2 ; // subtract op2 from op1

st.push(op1) ; // push the difference

cout << PUSH << op1 ; // Display difference

} // end else if subtraction operator

else if (token == '\*') {

cout << TOKEN << static\_cast<char>(token) ; // Display operator

cout << POP << st.top(); // Display operand 1

int total = st.top(); // get operand 1

st.pop() ; // pop op 1

cout << POP << static\_cast<int>(st.top()) ; // Display operand 2

total \*= st.top() ; // mult operand 2

st.pop() ; // pop op 2

st.push(total) ; // push total

cout << PUSH << total ; // Display total

} // end else if multiplication operator

else if (token == '/') {

cout << TOKEN << static\_cast<char>(token) ; // Display operator

cout << POP << st.top() ; // Display operand 2

int op2 = st.top() ; // get operand 2

st.pop() ; // pop op 2

cout << POP << st.top() ; // Display operand 1

int op1 = st.top() ; // get operand 1

st.pop() ; // pop op 1

op1 /= op2 ; // divide op 1 by op 2

st.push(op1) ; // push the quotient

cout << PUSH << op1 ; // Display quotient

} // end else if division operator

cout << endl ;

} // end if

} // end for

// Display final pop

cout << TOKEN << ' ' << POP << static\_cast<int>(st.top()) << endl ;

return 0 ;

} // end main ;

//--- LStack.cpp -------------------------------------------------

// utilized by: Chris Dang

#include <new>

using namespace std;

#include "LStack.h"

//--- Definition of Stack constructor

Stack::Stack(){myTop = NULL;}

//--- Definition of Stack copy constructor

Stack::Stack(const Stack & original)

{

myTop = 0;

if (!original.empty())

{

// Copy first node

myTop = new Stack::Node(original.top());

// Set pointers to run through the stacksÕ linked lists

Stack::NodePointer lastPtr = myTop,

origPtr = original.myTop->next;

while (origPtr != 0)

{

lastPtr->next = new Stack::Node(origPtr->data);

lastPtr = lastPtr->next;

origPtr = origPtr->next;

}

}

}

//--- Definition of Stack destructor

Stack::~Stack()

{

// Set pointers to run through the stack

Stack::NodePointer currPtr = myTop, // node to be deallocated

nextPtr; // its successor

while (currPtr != 0)

{

nextPtr = currPtr->next;

delete currPtr;

currPtr = nextPtr;

}

}

//--- Definition of assignment operator

const Stack & Stack::operator=(const Stack & rightHandSide)

{

if (this != &rightHandSide) // check that not st = st

{

this->~Stack(); // destroy current linked list

if (rightHandSide.empty()) // empty stack

myTop = 0;

else

{ // copy rightHandSide's list

// Copy first node

myTop = new Stack::Node(rightHandSide.top());

// Set pointers to run through the stacks' linked lists

Stack::NodePointer lastPtr = myTop,

rhsPtr = rightHandSide.myTop->next;

while (rhsPtr != 0)

{

lastPtr->next = new Stack::Node(rhsPtr->data);

lastPtr = lastPtr->next;

rhsPtr = rhsPtr->next;

}

}

}

return \*this;

}

//--- Definition of empty()

bool Stack::empty() const

{

return (myTop == 0);

}

//--- Definition of push()

void Stack::push(const StackElement & value)

{

myTop = new Stack::Node(value, myTop);

}

//--- Definition of display()

void Stack::display(ostream & out) const

{

Stack::NodePointer ptr;

for (ptr = myTop; ptr != 0; ptr = ptr->next)

out << ptr->data << endl;

}

//--- Definition of top()

StackElement Stack::top() const

{

if (!empty())

return (myTop->data);

else

{

cerr << "\*\*\* Stack is empty "

" -- returning garbage \*\*\*\n";

StackElement \* temp = new(StackElement);

StackElement garbage = \*temp; // "Garbage" value

delete temp;

return garbage;

}

}

//--- Definition of pop()

void Stack::pop()

{

if (!empty())

{

Stack::NodePointer ptr = myTop;

myTop = myTop->next;

delete ptr;

}

else

cerr << "\*\*\* Stack is empty -- can't remove a value \*\*\*\n";

}

/\*-- LStack.h --------------------------------------------------------------

// utilized by: Chris Dang

This header file defines a Stack data type.

Basic operations:

constructor: Constructs an empty stack

empty: Checks if a stack is empty

push: Modifies a stack by adding a value at the top

top: Accesses the top stack value; leaves stack unchanged

pop: Modifies stack by removing the value at the top

display: Displays all the stack elements

Note: Execution terminates if memory isn't available for a stack element.

--------------------------------------------------------------------------\*/

#include <iostream>

#include <string>

using namespace std;

#ifndef LSTACK

#define LSTACK

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

// Uncomment the following section to use the class for your lab

// assignment.

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

/\*class IdentOrTemp

{

public:

int temp; // used as as counter for temp1, temp2 etc.

string ident; // used to specify a , b, c, etc.

};

//-- Empty output operator to use for Stack display()

inline ostream & operator<<(ostream & out, IdentOrTemp it)

{ out << ""; return out; }

typedef IdentOrTemp StackElement;

\*/

typedef int StackElement; // comment out or remove after testing.

class Stack

{

public:

/\*\*\*\*\* Methods \*\*\*\*\*/

/\*\*\*\*\* Constructors \*\*\*\*\*/

Stack();

/\*-----------------------------------------------------------------------

Construct a Stack object.

Precondition: None.

Postcondition: An empty Stack object has been constructed

(myTop is initialized to a null pointer).

------------------------------------------------------------------------\*/

Stack(const Stack & original);

/\*-----------------------------------------------------------------------

Copy constructor to create another Stack object.

Precondition: Another stack

Postcondition: A copy of the stack has been created.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Destructor \*\*\*\*\*/

~Stack();

/\*------------------------------------------------------------------------

Class destructor

Precondition: None

Postcondition: The linked list in the stack has been deallocated.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Assignment \*\*\*\*\*/

const Stack & operator= (const Stack & rightHandSide);

/\*------------------------------------------------------------------------

Assignment Operator

Precondition: rightHandSide is the stack to be assigned and is

received as a const reference parameter.

Postcondition: The current stack becomes a copy of rightHandSide

and a const reference to it is returned.

------------------------------------------------------------------------\*/

bool empty() const;

/\*------------------------------------------------------------------------

Check if stack is empty.

Precondition: None

Postcondition: Returns true if stack is empty and false otherwise.

-----------------------------------------------------------------------\*/

void push(const StackElement & value);

/\*------------------------------------------------------------------------

Add a value to a stack.

Precondition: value is to be added to this stack

Postcondition: value is added at top of stack provided there is space;

otherwise, a stack-full message is displayed and execution is

terminated.

-----------------------------------------------------------------------\*/

void display(ostream & out) const;

/\*------------------------------------------------------------------------

Display values stored in the stack.

Precondition: ostream out is open.

Postcondition: Stack's contents, from top down, have been output to out.

-----------------------------------------------------------------------\*/

StackElement top() const;

/\*------------------------------------------------------------------------

Retrieve value at top of stack (if any).

Precondition: Stack is nonempty

Postcondition: Value at top of stack is returned, unless the stack is

empty; in that case, an error message is displayed and a "garbage

value" is returned.

-----------------------------------------------------------------------\*/

void pop();

/\*------------------------------------------------------------------------

Remove value at top of stack (if any).

Precondition: Stack is nonempty.

Postcondition: Value at top of stack has been removed, unless the stack

is empty; in that case, an error message is displayed and

execution allowed to proceed.

-----------------------------------------------------------------------\*/

private:

/\*\*\* Node class \*\*\*/

class Node

{

public:

StackElement data;

Node \* next;

//--- Node constructor

Node(StackElement value, Node \* link = NULL){data = value; next = link;}

/\*-------------------------------------------------------------------

Precondition: None.

Postcondition: A Node has been constructed with value in its data

part and its next part set to link (default 0).

-------------------------------------------------------------------\*/

}; // end of class Node

typedef Node \* NodePointer;

/\*\*\*\*\* Data Members \*\*\*\*\*/

NodePointer myTop; // pointer to top of stack

}; // end of class declaration

#endif



