04/30/2021



**Department Of Computer Science**

|  |  |
| --- | --- |
| **Subject:** Data Structure and Algorithm | **Instructor:** Ma’am Zainab Malik |
| **Lab No:** 7  **Class:** BSCS-3B  **Students’ Name**  Madina Javed Iqbal Khokhar 2426  Esha Mansoor 2413  Sultan Zahid 2411  Abdul Moeed 2419 | **Date:** 04-30-2021 |

**Lab Repot 7**

**Task:**

**What is three tier database architecture? How does cloud architecture differ from three tier database architecture?**

**Explanation:**

In this lab we will continue from the previous program and add following new functions in the main class.

* postfixEvaluation
* infixToPostfix
* postfixToinfix

**PostfixEvaluation:**

For postfix, we had perform following steps in sequence:

First and foremost, we had created an empty stack of type double and read input postfix expression char by char till the end of input using for loop.Then, we had use if -else statement to analysis if char is an operand then we had converted it into its double equivalent and then push it on the stack. However,if char is an operator then pop two elements from the stack, perform the operation, we had use switch statement to perform operations and push the result back on the stack. At the end of input, Pop the final result and return it

**InfixToPostfix:**

Firstly, we had created an empty stack of type char then we read input expression char by char till the end of input using for loop.Then, we used If-else- if statement for following cases :

•If Operand then display it

•If opening Parenthesis then push

•In case of operator we will have following conditions:

❖If stack is empty then push it

❖If stack is non-empty then pop characters from stack and display them until

we find an operator of lower precedence or an opening parenthesis or stack

become empty. When popping is done push the current operator on to the

stack.

•Closing Parenthesis: Pop operators from stack and display them until we pop

an opening parenthesis which will be popped but not displayed.

Lastly,at the end of input, if stack is non-empty, pop operators from stack and

display them until the stack becomes empty

**PostfixToinfix:**

First of all create an empty stack of type string and read the input postfix expression char by char till the end of input,then use if –else for following cases:

* If the char is an operand then push it on the stack
* It the char is operator then pop two operands from stack from an infix sub-expression and push the sub-expression back on the stack.

At the end of the input pop the resultant infix expression from the stack and display it

Then, we will call each function using switch statement after creating a menu.

**Code:**

**Node Class:**

#include<iostream>

using namespace std;

template<class T> // Templates actually

//increase flexibility, they're easy to update, and they provide consistency

class Node

{

private:

T info; // variable name use to store information

Node<T> \*next; // variable use to store address of next node,that's why its data type

public:

Node(T i=0,Node<T> \*n=0):info(i),next(n)

{

// constructor { having

}

void setInfo(T i); // using setter and getter

T getInfo();

void setNext(Node<T> \*n);// using setter and getter

Node<T>\* getNext();

};//EOC

template<class T>

void Node<T>::setInfo(T i)

{

info=i; // setting our info

}

template<class T>

void Node<T>::setNext(Node<T> \*n)

{

next=n; // setting our next

}

template<class T>

T Node<T>::getInfo() // getting our info

{

return info;

}

template<class T>

Node<T>\* Node<T>::getNext() // getting our next

{

return next;

}

**LinkedStack Class:**

#include<iostream>

#include "Node.h" // To include node.h file

using namespace std;

template<class T> // Templates actually

//increase flexibility, they're easy to update, and they provide consistency

class LinkedStack

{

private:

Node<T> \*top; // top is declared which is the fudamental characteristics of linked stack

public:

LinkedStack() // constructor

{

top=0; // setting top to 0

}

void push(T element); //add to head

T pop(); // remove from head

bool isEmpty();

T topValue(); // tp access top value

void removeAndDisplayAll(); // to remove and display

};

template<class T>

void LinkedStack<T>::push(T element)

{

Node<T> \*ptr=new Node<T>(element);// creating a node

//info=5 & next=0

if(top==0)// if list is empty

{

top=ptr; // then top will be set to ptr

}

else //only one element or >1 element

{

ptr->setNext(top); // Increment top //next of 9 is 23

top=ptr;//head will now poT Node<T> with value 9

}

}//push

template<class T>

T LinkedStack<T>::pop() // to pop en elemnet

{

if(top==0) // is stack is empty

{

cerr<<"nothing to delete"<<endl; // then error occur

}

else if(top->getNext()==0) // if only one element exist

{

T info=top->getInfo(); //storing value of top info in info variable

delete top; // deleting value at tho top

top=0;

return info;

}

else//more than one element

{

Node<T> \*temp=top; // storing top in temp

top=top->getNext(); // increment top

T info=temp->getInfo();

delete temp;

return info; // it will return info which is being deleted

}

}//pop

template<class T>

bool LinkedStack<T>::isEmpty()

{

return top==0;

}//isEmpty

template<class T>

T LinkedStack<T>::topValue()

{

return top->getInfo(); // getting value at the top and then return

} //topValue

template<class T>

void LinkedStack<T>::removeAndDisplayAll()

{

while(top!=0) // // until stack become empty and passes through all nodes

{

cout<<pop()<<" ";

//cout<<endl;

}

}//removeAndDisplayAll

**Main Function:**

#include <iostream>

#include "LinkedStack.h"

#include <ctype.h>// for check its operater or digit

#include <stdlib.h>// for

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

// declaring functions

void numberConversion(int num);

void symbolBalancing(string text);

void postfixEvaluation(string exp);

string infixToPostfix(string exp);

string PostfixToinfix(string exp);

int prec(char op);

int main(int argc, char\*\* argv) {

LinkedStack<char>list1;

list1.push('n');

list1.push('u');

list1.push('m');

list1.push('l');

list1.removeAndDisplayAll(); // calling removeAndDisplayAll() function

cout<<endl;

int opt;

do

{

// creating a menu

cout<<"Press 1 for number conversion "<<endl;

cout<<"Press 2 for check sequence of symbols "<<endl;

cout<<"Press 3 to evaluate a postfix expression"<<endl;

cout<<"Press 4 to convert infix to postfix "<<endl;

cout<<"Press 5 to convert postfix to infix "<<endl;

cout<<"Press 0 to exit "<<endl;

cin>>opt;

string text;

switch (opt)

{

case 1:

// call function for number conversion

int num;

cout<<"Provide number in decimal "<<endl;

cin>>num; // getting a number for conversion

cout<<endl;

numberConversion(num); // calling a number conversion function

break;

case 2:

cout<<"Provide sequence of symbols"<<endl;

cin>>text; // getting a string of symbols

symbolBalancing(text); // calling a symbol balancing funnction

break;

case 3:

cout<<"Provide a postfix expression"<<endl;

cin>>text; // getting a expressionn for postfix expression

postfixEvaluation(text); // calling a postfixEvaluation

case 4:

cout<<"Provide an infix expression"<<endl;

cin>>text; // getting an infix expression

cout<<"Conversion from infix to postfix "<<endl;

infixToPostfix(text); // calling a infixToPostfix

break;

case 5:

cout<<"Provide an postfix expression"<<endl;

cin>>text;// getting an postfix expression

cout<<"Conversion from postfix to infix "<<endl;

PostfixToinfix(text);

break;

case 0:

exit(-1);

}//switch

}while(true); // do while

return 0;

}

void numberConversion(int num) // number conversion function

{

LinkedStack<int> stack;

while(num>=1) //it should be greater than 0

{

stack.push(num%2); // for remiander

num=num/2; // for quotient

}

cout<<"conversion from decimal to binary";

cout<<endl;

stack.removeAndDisplayAll(); // calling remove and display function

cout<<endl;

}// numbertConversion

void symbolBalancing(string text) // symbolBalancing function

{

LinkedStack<char> stack;

for(int i=0;i<text.length();i++)

{

if(text[i]=='('||text[i]=='['||text[i]=='{'||text[i]=='<')

{ // If char is an opening symbol, push it on the stack

stack.push(text[i]);

}//2.1

// If char is a closing symbol, then we will two situations

else if(text[i]==')'||text[i]==']'||text[i]=='}'||text[i]=='>')

{

if(stack.isEmpty())//2.2.1

{

// If stack is empty, report an error ("Opening symbol missing"

cout<<"opening symbol missing";

}

else//2.2.2

{

//If stack is non-empty, pop a char from stack and match it with input char

char stackVal=stack.pop();

if((text[i]==']'&& stackVal!='[')||(text[i]=='}'&& stackVal!='{')||(text[i]==')'&& stackVal!='(')||(text[i]=='>'&& stackVal!='<'))

{ // If both characters do not match then report an error ("Symbol mismatch")

cout<<"Symbol mismatch"<<endl;

return;

}

}

}

}

if(!stack.isEmpty())

{

cout<<"valid sequence"<<endl;

}

else

{

cout<<"Valid Sequence \n";

}

}

void postfixEvaluation(string exp)

{

LinkedStack<double>s1;//Create an empty stack of type double

for(int i=0;i<exp.length();i++)//Read input postfix expression char by char till the end of input

{

if(isdigit(exp[i]))//operan 2.1

{

// If char is an operand, convert it into its double equivalent

char operand=exp[i];

double convertedOperand=atof(&operand);

s1.push(convertedOperand); //then push it on the stack

}

else//operater

{

// If char is an operator then pop two elements from the stack,

double op1=s1.pop();

double op2=s1.pop();

switch(exp[i])

{

// perfoming operation

case '+':

s1.push(op2+op1);

break;

case '-':

s1.push(op2-op1);

break;

case '\*':

s1.push(op2\*op1);

break;

case '/':

s1.push(op2/op1);

break;

}//switch end

}//else

}//for

// pop the result and return

cout<<"Answer is : "<<s1.pop()<<endl;

}//postfixEvaluation

int prec(char op) // set precedence

{

if (op=='^')

{

return 3;

}

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

{

return 2;

}

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

{

return 1;

}

}//prec

string infixToPostfix(string exp)

{

LinkedStack<char>s1; //Create an empty stack of type char

string output="";

for(int i=0;i<exp.length();i++)//Read input expression char by char till the end of input

{

if(isdigit(exp[i]))// If operand then display it

{

output+=exp[i];

}

else if(exp[i]=='(')// If opening Parenthesis then push

{

s1.push(exp[i]);

}

else if(exp[i]=='+'||exp[i]=='-'||exp[i]=='\*'||exp[i]=='/'||exp[i]=='^')//operater 2.3

{

if(s1.isEmpty())//empty then push

{

s1.push(exp[i]);

}

else//nonempty then pop characters from stack and display them until we find an operator of lower precedence or an opening parenthesis or stack become empty.

{

while(!s1.isEmpty()&&s1.topValue()!='('&&prec(s1.topValue())>=prec(exp[i]))

{

output+=s1.pop();

}//popping done

s1.push(exp[i]);// When popping is done push the current operator on to the stack.

}

}

else if(exp[i]==')')//closing symbol then Pop operators from stack and display them until

//we pop an opening parenthesis which will be popped but not displayed

{

while(s1.topValue()!='(')

{

output+=s1.pop();

}

//at top there is opening symbol

s1.pop();

}

}//for

while(!s1.isEmpty()) // if stack is non empty

{

output+=s1.pop(); //, pop operators from stack and display them until the stack becomes empty.

}

cout<<"Postfix expression is : "<<output<<endl;

return output;

}// end string infixToPostfix

string PostfixToinfix(string exp)

{

LinkedStack<string>s1; //Create an empty stack of type string

for(int i=0;i<exp.length();i++)//Read the input postfix expression char by char till the end of input

{

if(isdigit(exp[i]))//operand

{

string s(1,exp[i]);

s1.push(s); // push char in stack

}

else//operater

{

string op1=s1.pop(); // pop two operands

string op2=s1.pop();

s1.push("("+op2+exp[i]+op1+")");// push the sub expression back on the stack

}

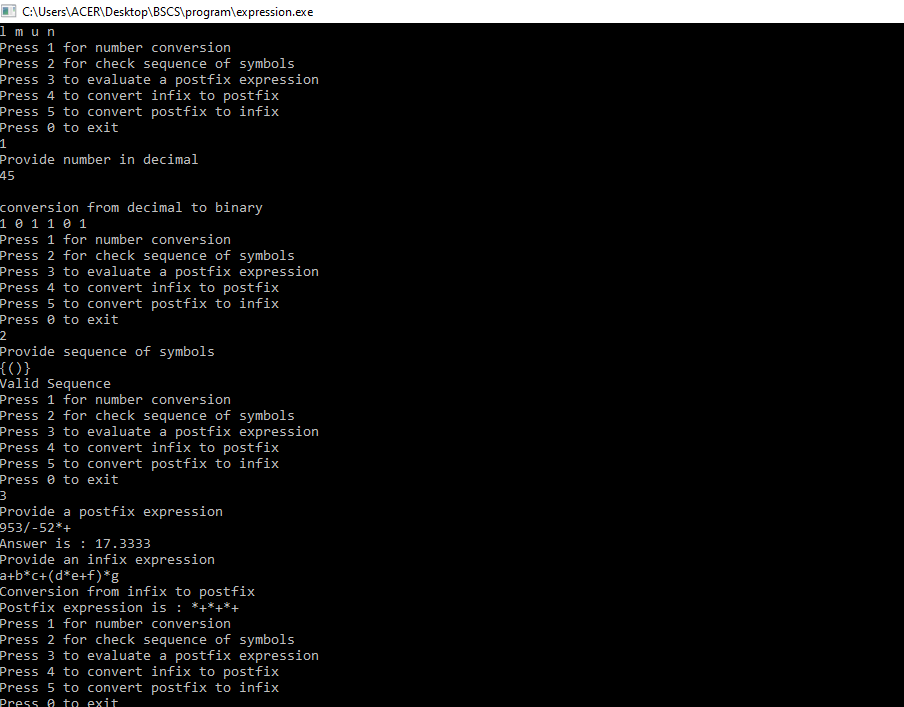
}//for

cout<<"Infix expression is : "<<s1.topValue()<<endl;

return s1.pop();

}

**Output:**



# Thanks