Data Structures Lab 6

Stack Manual

Name: Ahmed Kasteer

Section 3D

Roll Number: 20F-0336

Task 1 & 2

#include <iostream>

using namespace std;

class Stack {

private:

int\* Stackarray;

int Stacksize;

int Top;

public:

Stack(int Size)

{

Stacksize = Size;

Stackarray = new int[Size];

Top = -1;

}

~Stack()

{

delete Stackarray;

}

bool isEmptyStack()

{

if (Top <= -1)

{

cout << "Stack is empty" << endl;

return true;

}

else

{

return false;

}

}

bool isFullStack()

{

if (Top >= Stacksize - 1)

{

cout << "Stack is full" << endl;

return true;

}

else

{

return false;

}

}

void Push(int value)

{

if (isFullStack() == false)

{

Top++;

Stackarray[Top] = value;

}

}

void Pop()

{

if (isEmptyStack() == false)

{

cout << "The popped element is : " << Stackarray[Top] << endl;

Top--;

}

}

void top()

{

cout << "Top Element is : " << endl;

cout << Stackarray[Top] << endl;

}

void Display()

{

if (isEmptyStack() == false)

{

for (int i = Top; i >= 0; i--)

{

cout << Stackarray[i] << " ";

}

cout << endl;

}

}

void Reverse()

{

int temp = 0;

int j = 0;

for (int i = Top; i >= (Top / 2) + 1; i--)

{

temp = Stackarray[i];

Stackarray[i] = Stackarray[j];

Stackarray[j] = temp;

j++;

}

}

};

int main()

{

int size;

cout << "------------Stack Program--------------" << endl;

cout << "Enter size of stack: " << endl;

cin >> size;

Stack obj(size);

for (int i = 1; i <= size; i++)

{

obj.Push(rand()%10+1);

}

obj.Display();

obj.top();

cout << "Reversing the stack" << endl;

obj.Reverse();

obj.Display();

obj.top();

return 0;

}

Text

Description automatically generated

Task 3

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

};

class Stack {

private:

Node\* top;

public:

Stack()

{

top = NULL;

}

~Stack()

{

delete top;

}

bool isEmpty()

{

if (top == NULL)

{

cout << "Stack is empty" << endl;

return true;

}

else

{

return false;

}

}

void Push(int value)

{

Node\* new\_node = new Node;

new\_node->data = value;

new\_node->next = top;

top = new\_node;

}

bool Pop(int& val)

{

if (isEmpty() == true)

{

return false;

}

else

{

Node\* temp = top;

val = top->data;

top = top->next;

delete temp;

return true;

}

}

void Display\_stack()

{

Node\* temp = top;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

void makeEmpty()

{

int value = 0 ;

while (Pop(value));

cout << "Stack has been emptied" << endl;

}

void Reverse\_stack()

{

if (isEmpty() == false)

{

Node\* temp = top;

int nodecount = 1;

while (temp != NULL)

{

nodecount++;

temp = temp->next;

}

Node\* temp2 = top;

int\* temparr = new int[nodecount];

int x = 0;

while (temp2 != NULL)

{

temparr[x] = temp2->data;

temp2 = temp2->next;

x++;

}

makeEmpty();

for (int i = 0; i < nodecount - 1; i++)

{

Push(temparr[i]);

}

}

}

};

int main()

{

int val = 0;

Stack obj;

obj.Push(5);

obj.Push(6);

obj.Push(8);

obj.Push(10);

obj.Display\_stack();

cout << "Popping top from stack" << endl;

obj.Pop(val);

cout << "Element Popped: " << val << endl;

cout << endl;

cout << "Current state of Stack:" << endl;

obj.Display\_stack();

cout << endl;

cout << "Stack being reversed:" << endl;

obj.Reverse\_stack();

cout << endl;

cout << "Stack Reversed" << endl;

obj.Display\_stack();

return 0;

}

Text

Description automatically generated

Task 4

#include<iostream>

#include<string>

using namespace std;

class stack

{

public:

int stacksize;

char\* stackarray;

int top;

stack(int size)

{

top = -1;

stacksize = size;

stackarray = new char[stacksize];

for (int i = 0; i < stacksize; i++)

{

stackarray[i] = ' ';

}

}

};

bool isempty(stack s)

{

if (s.top == -1)

{

cout << "Stack is empty" << endl;

return true;

}

else

{

return false;

}

}

bool isfull(stack s)

{

if (s.top == s.stacksize - 1)

{

cout << "Stack is full" << endl;

return true;

}

else

{

return false;

}

}

void push(stack s, char val)

{

if (isfull(s) == true)

{

return;

}

else

{

s.top++;

s.stackarray[s.top] = val;

}

}

void pop(stack s)

{

if (isempty(s) == true)

{

cout << "Cannot pop" << endl;

}

else

{

cout << s.stackarray[s.top] << endl;

s.top--;

}

}

bool isOperator(char val)

{

if (val == ')' || val == '(' || val == '+' || val == '-' || val == '/' || val == '\*' || val == '^')

{

return true;

}

else

{

return false;

}

}

int prec(char val)

{

if (isOperator(val) == true)

{

if (val == '^')

return 3;

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

return 2;

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

return 1;

else

return -1;

}

}

void convertToPostfix(stack s, string val)

{

string res;

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

{

if (isOperator(val[i]) == false)

{

res += val[i];

}

else if (val[i] == '(' || isOperator(val[i]) == true)

{

push(s, val[i]);

}

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

{

while (!isempty(s) && s.top != '(')

{

res += s.top;

pop(s);

}

if (!isempty(s))

{

pop(s);

}

}

else

{

while (!isempty(s) && prec(s.top) >= prec(val[i]))

{

res += s.top;

pop(s);

}

push(s, val[i]);

}

}

while (!isempty(s))

{

res += s.top;

pop(s);

}

cout << "Result: " << res << endl;

}

void printstack(stack s)

{

cout << "Printing stack: ";

for (int i = 0; i < s.stacksize; i++)

{

cout << s.stackarray[i] << " ";

}

}

int main()

{

string c = "(6+2)\*5-8/4";

stack s(c.size());

convertToPostfix(s, c);

printstack(s);

}

Text

Description automatically generated