# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS 10AB**

# 

# Lab 04 : Stack and its Applications

**Date: 12th October, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# MASHAL ASHFAQUE

# 337203

# CS 10-A

# Lab 04 : Stack and its Applications

**Introduction**

This lab consists of stacks implementation and some of its applications.

**Objectives**

Objective of this lab is to enable students to build stack ADT using linked list and arrays, perform the following tasks on it and analyze the performance of each implementation.

**Tools/Software Requirement**

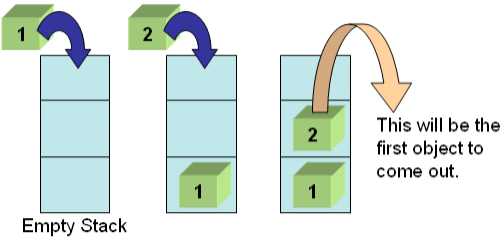
Visual Studio c++, Eclipse C++ IDE

**Helping Material**

Lecture slides, text book

**Description**

A stack is a data structure in which the insertion and deletion operations are performed only at one end referred to as the top end. Resultingly, the stack is referred to as Last-In First-Out (LIFO) or First-In Last-Out (FILO) data structure. The purpose of this lab is to build stack using arrays and linked lists.



**Tasks:**

**Task 1:**

Your first task is to implement all the following operations of Stack ADT using both the arrays and linked lists:

1. void Push(element) – pushes an element on the top of stack
2. element Pop() – removes and display the element on the top of stack
3. boolisEmpty() – checks if the stack is empty or not
4. boolisFull() – checks if the stack is full or not
5. void Clear() – release the memory allocated by stack
6. void Peak() – display the contents of the top element of stack

**Solution:**

**Array based implementation:**

#include <iostream>

using namespace std;

#define SIZE 5

class stack

{

public:

int arr[SIZE];

int top = -1;

//required functions

bool isfull();

bool isempty();

void push(int);

int pop();

int peak();

//other utility functions

void print\_stack()

{

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

{

cout << pop() << "\t";

}

}

};

bool stack::isfull()

{

return top == SIZE - 1;

}

bool stack::isempty()

{

return top < 0;

}

void stack::push(int val)

{

if (!isfull())

{

arr[++top] = val; //pre increment top and put in it the desired value

}

else

cout << "List already full" << endl;

}

int stack::pop()

{

int element = -1; //flag

if (!isempty())

{

element = arr[top];

arr[top--] = -1;

}

return element; //deletes and displays top element

}

int stack::peak()

{

return arr[top]; //return top element without deleting it;

}

int main()

{

stack Array;

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

{

int val;

cout << "Enter element # " << i + 1 << endl;

cin >> val;

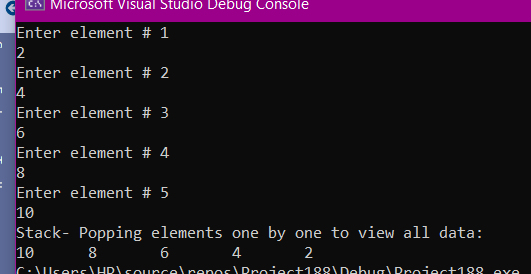
Array.push(val);

}

cout << "Stack- Popping elements one by one to view all data:" << endl;

Array.print\_stack();

}



**Linked list based implementation:**

#include <iostream>

using namespace std;

class node

{public:

int data;

node\* next;

node()

{

next = NULL;

}

};

class stack

{

public:

int size;

node\* top;

stack()

{

top = NULL; size = 0;

}

//required functions

bool isempty();

void push(int);

int pop();

int peak();

void clear();

//other utility functions

void printstack();

};

bool stack::isempty()

{

return size == NULL;

}

void stack::push(int val)

{

node\* temp = new node;

temp->data = val;

temp->next = top;

top = temp;

size++;

}

int stack::pop()

{

int val;

if (!isempty())

{

node\* temp = top;

val = top->data;

top = top->next;

delete temp; //delete the top element and save its value

}

return val;

}

int stack::peak()

{

if (!isempty())

{

return top->data; //return the top element without deleting it

}

}

void stack::clear()

{

while (!isempty())

{

pop();

}

}

void stack::printstack()

{

if (isempty())

return;

else

{

while (top != NULL)

{

cout << pop()<<"\t";

}

}

}

int main()

{

stack linklist;

cout << "How many values you want to add to stack?"; //user defined size of array

int size; cin >> size;

//enetering data

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

{

int val;

cout << "\nEnter element# " << i + 1;

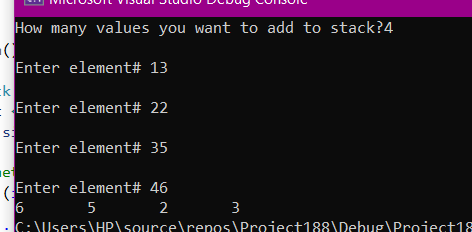
cin >> val;

linklist.push(val);

}

linklist.printstack();

}



**Task 2:**

The idea is rather simple: You keep a Stack of braces, and every time you encounter an open brace, you push it into your stack. Every time you encounter a close brace, you pop the top element from your stack. At the end, you check your stack for being empty. If so, indeed your input string contained balanced braces. Otherwise, it didn't.

**Expected Input**

1. 1 + 2 \* (3 / 4)
2. 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10) – 11 + (12\*8)] + 14
3. 1 + 2 \* [3 \* 3 + {4 – 5 (6 (7/8/9) + 10)} – 11 + (12\*8) / {13 +13}] + 14

Your program will determine whether the open brackets (the square brackets, curly braces and the parentheses) are closed in the correct order.

**Expected Output**

1. This expression is correct.
2. This expression is NOT correct. Error at character # 10. ‘{‘- not closed.
3. This expression is correct.

Your program should be able to take generic input expression from user

Solve the above problem using an array-based stack.

#include <iostream>

#include <string>

using namespace std;

#define SIZE 5

class stack

{

public:

int arr[SIZE];

int top = -1;

//required functions

bool isfull();

bool isempty();

void push(int);

int pop();

int peak();

//other utility functions

void print\_stack()

{

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

{

cout << pop() << "\t";

}

}

};

bool stack::isfull()

{

return top == SIZE - 1;

}

bool stack::isempty()

{

return top < 0;

}

void stack::push(int val)

{

if (!isfull())

{

arr[++top] = val; //pre increment top and put in it the desired value

}

else

cout << "List already full" << endl;

}

int stack::pop()

{

int element = -1; //flag

if (!isempty())

{

element = arr[top];

arr[top--] = -1;

}

return element; //deletes and displays top element

}

int stack::peak()

{

return arr[top]; //return top element without deleting it;

}

int main()

{

stack braces;

bool valid = true;

while (1)

{

cout << "Enter the operation:" << endl;

string ope;

getline(cin, ope);

int char\_no;

for (int i = 0; i < size(ope); i++)

{

if (ope[i] == '(' || ope[i] == '[' || ope[i] == '{')

{

braces.push(ope[i]);

char\_no = i;

}

if (ope[i] == ')' || ope[i] == ']' || ope[i] == '}')

{

if (braces.isempty())

{

valid = false;

}

else if(ope[i]==')' || ope[i] == '}' || ope[i] == ']' )

{

char l = ope[i];

if (l == ')' && braces.peak() != '(')

{

valid = false;

}

else if (l == '}' && braces.peak() != '{')

{

valid = false;

}

else if (l == ']' && braces.peak() != '[')

{

valid = false;

}

else

{

braces.pop(); //matching closing bracket found so clear that from brackets/braces stack

}

}

}

}

if (braces.isempty() == false)

{

valid = false;

}

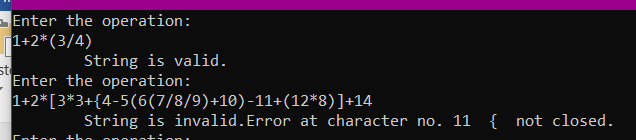
if (valid)

cout << "\tString is valid." << endl;

else

cout << "\tString is invalid.Error at character no. " <<char\_no<<" "<< static\_cast<char>(braces.pop())<< " " << " not closed." << endl;

}

}

**Task 3:**

A mathematical expression can be written in prefix, infix and postfix notations. Your task is to implement the algorithm that converts a mathematical expression from Infix notation to its equivalent postfix notation using stack. Your function should take as input the string in which an expression in the infix notation is stored. It should return the final postfix expression which a user may store for further processing.

**Note:** You should add a special character such as space, comma etc. between operands to avoid confusion. For instance, if an infix expression 23+45 is written in equivalent postfix form 2345+ without adding special character between operands, it isn’t clear whether addition applies to 2 and 345, 23 and 45 or any other combination.

|  |  |
| --- | --- |
| **Sample Inputs** | **Sample Outputs** |
| 10+3-5 | 10 3+5- |
| 12+30/5 | 12 30 5/+ |
| 430+10^3 | 430 10 3^+ |
| {2\*(430+10)}^3 | 2 430 10+\*3^ |

#include <iostream>

using namespace std;

#define SIZE 10

int Precedence(char c) //function to check the priority

{

if (c == '^') //highest priority

return 3;

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

return 2;

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

return 1;

else

return -1;

}

class stack

{

public:

int arr[SIZE];

int top = -1;

//required functions

bool isfull();

bool isempty();

void push(int);

int pop();

int peak();

string converttopostfix(string infix);

//other utility functions

void print\_stack()

{

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

{

cout << pop() << "\t";

}

}

};

bool stack::isfull()

{

return top == SIZE - 1;

}

bool stack::isempty()

{

return top < 0;

}

void stack::push(int val)

{

if (!isfull())

{

arr[++top] = val; //pre increment top and put in it the desired value

}

else

cout << "List already full" << endl;

}

int stack::pop()

{

int element = -1; //flag

if (!isempty())

{

element = arr[top];

arr[top--] = -1;

}

return element; //deletes and displays top element

}

int stack::peak()

{

return arr[top]; //return top element without deleting it;

}

string stack::converttopostfix(string infix)

{

string postfix;// to get the postfix expression at the end

for (int i = 0; i < infix.length(); i++) //for loop

{

//if there are operators in infix string

if (infix[i] == '+' || infix[i] == '-' || infix[i] == '/' || infix[i] == '\*' || infix[i] == '^')

{

//if the element at the top of the stack is '(' then push it in the infix string

if (isempty() || peak() == '(')

{

push(infix[i]);

}

//if the precedence of the element in infix string is greater than the precedance of the element at the top of the stack then push it in the stack

else if (Precedence(infix[i]) > Precedence(peak()))

{

push(infix[i]);

}

//if the precedance of element in infix string is equal to the precedance of the element at the top of the array then push it in the stack

else if (Precedence(infix[i]) == peak() && infix[i] == '^')

{

push(infix[i]);

}

else

{

//until the stack is empty & precedance of element in infix string is less than or equal to precedance of top element of array

while (!isempty() && Precedence(infix[i]) <= Precedence(peak()))

{

//append the pop element in the postfix expression

postfix += pop();

}

push(infix[i]);

}

}

else if (infix[i] == '(') //if the element in infix string is opening bracket then push it in the string

{

push(infix[i]);

}

else if (infix[i] == ')') //if the element in infix string is closing bracket then pop it

{

while (!isempty() && peak() != '(') //append in postfix string until stack is empty & stack top is not

{

postfix += pop();

}

pop();

}

else

{

if (infix[i + 1] == '+' || infix[i + 1] == '-' || infix[i + 1] == '/' || infix[i + 1] == '\*' || infix[i + 1] == '^')

{

postfix += infix[i];

postfix += " ";

}

//if the operand is multi-digit, donot concatenate space

else

{

postfix += infix[i];

}

}

}

while (!isempty())

{

postfix += pop();

}

return postfix;

}

int main()

{

stack obj;

// char stackarray[size];

string infix, postfix;

cout << "\n\nEnter the Infix Mathematical Expression::";

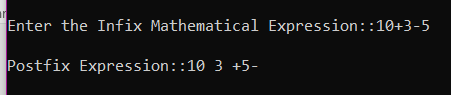
cin >> infix;

postfix = obj.converttopostfix(infix);

cout << "\nPostfix Expression::" << postfix << endl;

return 0;

}



**Task 4:**

Your task is to implement the algorithm that solves a mathematical expression stored in the postfix notation using stack. It should return the final answer.

**Note:** An expression is stored in a string. You should convert operands to the relevant datatype before solving an expression. For instance, characters 430, 10 and 3 in the string 430 10 3^+ should be converted to int prior to applying arithmetic operations. A stack of type **int** should be for the below given inputs.

|  |  |
| --- | --- |
| **Sample Inputs** | **Sample Outputs** |
| 10 3+5- | 8 |
| 12 30 5/+ | 18 |
| 430 10 3^+ | 1430 |

**Code:**

#include<iostream>

#include<string>

#include <cstdlib>

using namespace std;

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

node()

{

next = NULL;

}

};

class stack

{

public:

int size;

node\* top;

stack()

{

top = NULL; size = 0;

}

//required functions

bool isempty();

void push(int);

int pop();

int peak();

void clear();

void evaluate(string);

//other utility functions

void printstack();

};

bool stack::isempty()

{

return size == NULL;

}

void stack::push(int val)

{

node\* temp = new node;

temp->data = val;

temp->next = top;

top = temp;

size++;

}

int stack::pop()

{

int val;

if (!isempty())

{

node\* temp = top;

val = top->data;

top = top->next;

delete temp; //delete the top element and save its value

}

return val;

}

int stack::peak()

{

if (!isempty())

{

return top->data; //return the top element without deleting it

}

}

void stack::clear()

{

while (!isempty())

{

pop();

}

}

void stack::printstack()

{

if (isempty())

return;

else

{

while (top != NULL)

{

cout << pop() << "\t";

}

}

}

void stack::evaluate(string exp)

{

int Opnd1 = 0;

int Opnd2 = 0;

int ans = 0;

int num = 0;

string val = "";//initialize empty

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

{

if (isdigit(exp[i]))//if its a digit then add it to the postfix string

{

val.push\_back(exp[i]); //appends the digit to preveious digit in the string

}

else //symbol is an operator or blank

{

if (isdigit(exp[i - 1]) || isblank(exp[i - 1]))//edit

{

//if second last char is blank or number it means number has ended so convert it to int

int num = stoi(val); //convert the string to int

push(num);//pushes number to stack

val = ""; //reinitialise

}

if (!isdigit(exp[i]) && !isblank(exp[i]))

{

//if char is operator then pop 2 numbers as the operands for this operator

Opnd1 = pop(); Opnd2 = pop();

if (exp[i] == '+')

{

ans = Opnd2 + Opnd1;

}

else if (exp[i] == '^')

{

ans = pow(Opnd2, Opnd1);//edit

}

else if (exp[i] == '-')

{

ans = Opnd2 - Opnd1;

}

else if (exp[i] == '\*')

{

ans = Opnd2 \* Opnd1;

}

else if (exp[i] == '/')

{

ans = Opnd2 / Opnd1;

}

push(ans);// push answer to the stack side by side

}

}

}

}

int main()

{

stack list;

string postfix;

for (int i = 0; i < 3; i++) //to check 3 sample outputs

{

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

getline(cin, postfix); //getting input from user

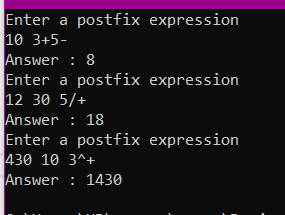
list.evaluate(postfix);

cout << "Answer : " << list.pop()<<endl;

}

return 0;

}



**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.