# Department of Computing

# School of Electrical Engineering and Computer Science

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

**Class: BSCS 10C**

# 

# Lab 05 : Stack and its Applications

**Date: 15th October, 2021**

**Time: 09:00 am – 11:50 pm**

# Instructor: Dr. Faisal Shafait

# Lab Engineer: Aftab Farooq

# 

# Lab 05 : 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**

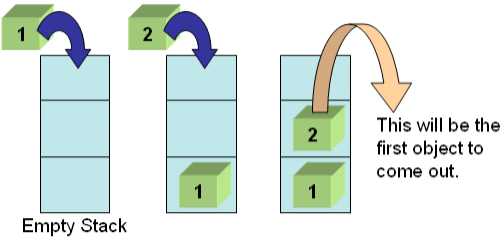
Visual Studio c++, Eclipse C++ IDE

**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

**Code:**

#include<iostream>

#include<conio.h>

using namespace std;

class Node{

public:

int data;

Node\* next;

};

Node\* head = NULL;

class linkStack{

public:

void push(int data\_value){

Node\* temp = new Node;

temp->data = data\_value;

temp->next = head;

head = temp;

}

void display(){

Node\* temp = new Node;

temp = head;

while (temp != NULL){

cout << temp->data << endl;

temp = temp->next;

}

}

int Pop(){

Node\* temp = new Node;

int tempo;

temp = head;

tempo = head->data;

head = head->next;

delete temp;

return tempo;

}

bool isEmpty(){

if (head == NULL){

return true;

}

else{

return false;

}

}

bool isFull(){

return false;

}

void clear(){

Node\* temp = new Node;

temp = head;

while (temp!=NULL){

delete temp;

temp = temp->next;

}

}

void peek(){

cout << head->data;

}

};

int main(){

linkStack s;

cout << "Entering Element in to the stack via link list" << endl << endl;

s.push(1);

s.push(2);

s.push(3);

s.push(4);

s.push(5);

cout << endl;

s.display();

cout << "Calling the function peek to see the top of the stack: " << endl;

cout << endl;

s.peek();

cout << endl;

cout << "Calling the function pop to pop up the top element of the stack and the remaining stack will be:" << endl;

cout << endl;

s.Pop();

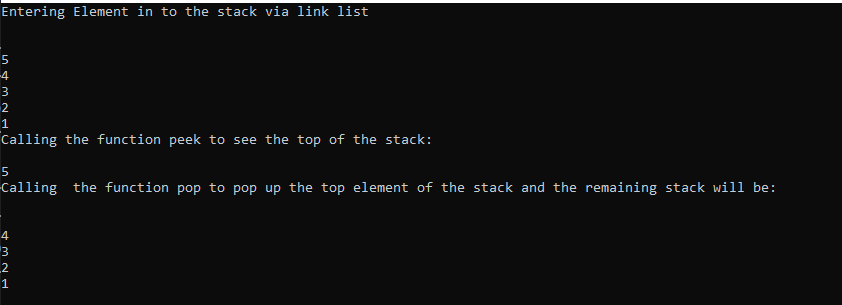
cout << endl;

s.display();

\_getch();

}

**Output:**

**For The Array Stack:**

#include<iostream>

#include<conio.h>

using namespace std;

class Node{

public:

int data;

Node\* next;

};

Node\* head = NULL;

class arrayStack{

public:

int arr[10];

int size=0, capacity=10;

void push(int value){

if (size < capacity){

arr[size] = value;

size++;

}

else{

cout << "The array is full and we can't push further elements" << endl;

}

}

void display(){

for (int i = size-1; i>=0; i--){

cout << arr[i] << endl;

}

}

int Pop(){

size--;

return arr[size];

}

bool isEmpty(){

if (size == 0){

return true;

}

return false;

}

bool isFull(){

if (size == capacity){

return true;

}

return false;

}

void clear(){

delete[] arr;

}

void peek(){

cout<< arr[size - 1]<<endl;

}

};

int main(){

arrayStack as;

cout << "Entering elements into the array{1,2,3,4,5} " << endl << endl;

as.push(1);

as.push(2);

as.push(3);

as.push(4);

as.push(5);

as.display();

cout << endl;

cout << "Calling the peek function to have the top of the array :" << endl;

as.peek();

cout << endl;

cout << "Calling the Pop function to remove the top element of the array" << endl;

as.Pop();

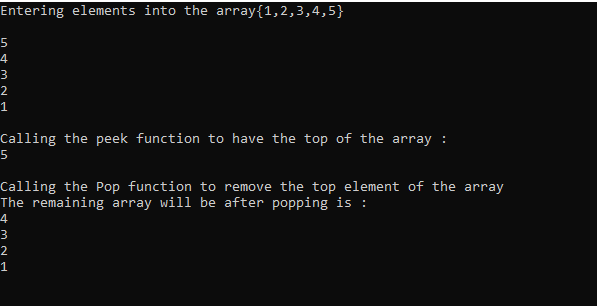
cout << "The remaining array will be after popping is : " << endl;

as.display();

\_getch();

}

**Output:**



**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.

**Code:**

**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.

**Code:**

|  |
| --- |
| #include<iostream>  #include<conio.h>  #include<string>  using namespace std;  class Node {  public:  int data;  Node\* next;  };  Node\* head = NULL;  /\*Using template to make it general so we can use it as int or char whatever we want\*/  template<class T>  class arrayStack {  public:  T\* arr = new T[100];  int size = 0, capacity = 100;  void push(T value) {  if (size < capacity) {  arr[size] = value;  size++;  }  else {  cout << "The array is full and we can't push further elements" << endl;  }  }  void display() {  for (int i = size - 1; i >= 0; i--) {  cout << arr[i] << endl;  }  }  T pop() {  size--;  return arr[size];  }  bool isEmpty() {  if (size == 0) {  return true;  }  return false;  }  bool isFull() {  if (size == capacity) {  return true;  }  return false;  }  void clear() {  delete arr;  }  T peek() {  return arr[size - 1];    }    };  bool checkExpression(string exp) {  int open = 0;  int close = 0;    arrayStack<int> stackInt;  arrayStack<char> stackChar;  cout << endl;    for (int i = 0; i < exp.length(); i++)  {  if (exp[i] == '[' || exp[i] == '{' || exp[i] == '(')  {  stackChar.push(exp[i]);  stackInt.push(i+1);  }  else if (exp[i] == ')')  {  if (stackChar.peek() == '(')  {  stackChar.pop();  stackInt.pop();  }  else  break;  }  else if (exp[i] == '}')  {  if (stackChar.peek() == '{')  {  stackChar.pop();  stackInt.pop();  }  else  break;  }  else if (exp[i] == ']')  {  if (stackChar.peek() == '[')  {  stackChar.pop();  stackInt.pop();  }  else  break;  }  }  if (!stackInt.isEmpty()) {  cout << endl;  cout << "The Expression is not correct at Index # " << stackInt.peek() << " '" << stackChar.peek() << "' is not closed!" << endl;  }  if (stackInt.isEmpty() && stackChar.isEmpty() && open==close ) {  cout << "Expression is Correct";  return true;  }  return false;  }  int main() {    string expression;  cout << "Enter the expression : " << endl;  getline(cin, expression);  checkExpression(expression);  } |

**Output:**

**Text

Description automatically generated**

**Text

Description automatically generatedText

Description automatically generated**

**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^ |

**Code:**

|  |
| --- |
| #include<iostream>  #include<conio.h>  #include<string>  int presedenceCheck(char Operator) {  if (Operator == '^') {  return 3;  }  if (Operator == '\*' || Operator == '/') {  return 2;  }  if (Operator == '+' || Operator == '-') {  return 1;  }  else  return 0;  }  using namespace std;  class Node {  public:  int data;  Node\* next;  };  Node\* head = NULL;  template<class T>  class arrayStack {  public:  T\* arr = new T[100];  int size = 0, capacity = 100;  void push(T value) {  if (size < capacity) {  arr[size] = value;  size++;  }  else {  cout << "The array is full and we can't push further elements" << endl;  }  }  void display() {  for (int i = size - 1; i >= 0; i--) {  cout << arr[i] << endl;  }  }  T pop() {  size--;  return arr[size];  }  bool isEmpty() {  if (size == 0) {  return true;  }  return false;  }  bool isFull() {  if (size == capacity) {  return true;  }  return false;  }  void clear() {  delete arr;  }  T peek() {  return arr[size - 1];    }    };  void postFixToPreFix(string prefix) {  arrayStack<char> stack;  string postfixExp = "";  for (int i = 0; i < prefix.length(); i++)  {  if (isdigit(prefix[i]))  {  postfixExp += prefix[i];  }  else  {  postfixExp += ' '; // append a space after every number  if (prefix[i] == ' ')  {  continue;  }  else if (prefix[i] == '('|| prefix[i] == '{')  {  stack.push(prefix[i]);  }  else if (prefix[i] == ')' || prefix[i] == '}')  {  while (!stack.isEmpty() && stack.peek() != '(' && stack.peek() != '{')  {  postfixExp += stack.pop();  }  if (!stack.isEmpty())  stack.pop();  }  else  {  while (!stack.isEmpty() && presedenceCheck(stack.peek()) >= presedenceCheck(prefix[i]))  {  postfixExp += stack.peek();  stack.pop();  }  stack.push(prefix[i]);  }  }  }  while (!stack.isEmpty())  {  postfixExp += stack.peek();  stack.pop();  }  cout << postfixExp;  }  int main() {    string expression;  cout << "Enter the expression : " << endl;  getline(cin, expression);  postFixToPreFix(expression);  } |

**Output:**

**Text

Description automatically generated**

**Text

Description automatically generated with medium confidence**

**Text

Description automatically generated**

****

**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.