# 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**

# MUHAMMAD IBRAHIM – 333426 – BSCS 10 A

# Instructor: Dr. Yasir Faheem

# Lab Engineer: Aftab Farooq

# 

# 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**

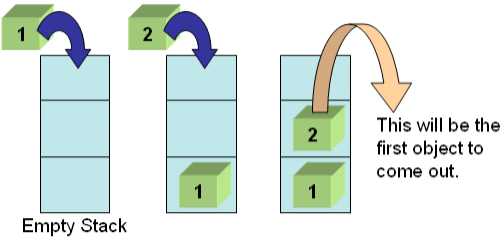
Visual Studio c++, Eclipse C++ IDE

**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

|  |
| --- |
| Array-Based Implementation: |
| #include <iostream>  #define MAX\_SIZE 10 /\* setting max size for array size \*/  using namespace std;  /\* Array based implementation of stack \*/  class stack  {  int \*arr = new int[MAX\_SIZE];  int top;  int length;  public:  /\* Constructor \*/  stack()  {  top = -1;  length = 0;  }  /\* Methods \*/  bool isEmpty()  {  return length == 0;  }  bool isFull()  {  return length == MAX\_SIZE;  }  void push(int data)  {  if (isFull())  {  cout << "\tStack is full.." << endl;  return;  }  cout << "\tValue has been added" << endl;  arr[length] = data;  length++;  top++;  }  int pop()  {  int data = arr[top];  top--;  length--;  return data;  }  void clear()  {  delete[] arr;  }  int peek()  {  return arr[top];  }  };  /\* Main Function \*/  int main()  {  stack st;  int option;  while (true)  {  system("cls");  cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STACK - ARRAY IMPLEMENTATION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  cout << "\t1.Push into stack\n\t2.Pop from the stack\n\t3.Peek from the stack\n\t4.Clear Stack" << endl << endl;  cout << "\tYour command: ";  cin >> option;  cout << endl;  /\* Operation according to the command \*/  switch (option)  {  int data;  case 1:  cout << "\tEnter the number you want to push: ";  cin >> data;  st.push(data);  break;  case 2:  if (!st.isEmpty())  {  st.pop();  cout << "\tValue has been popped from the stack" << endl;  }  else  {  cout << "\tStack is empty" << endl;  }  break;  case 3:  if (!st.isEmpty())  {  cout << "\tValue at the top of the stack is: " << st.peek() << endl;  }  else  {  cout << "\tStack is empty" << endl;  }  break;  case 4:  st.clear();  cout << "\tStack has been cleared" << endl;  break;  default:  cout << "\tEnter a valid option..." << endl;  break;  }  cout << "\n\t";  system("pause");  }  return 0;  } |
| OUTPUT: |
|  |
|  |

|  |
| --- |
| Linked List-Based Implementation: |
| #include <iostream>  using namespace std;  /\* Node Class \*/  class node  {  public:  int data;  node\* next;  public:  /\* Constructor \*/  node()  {  next = NULL;  data = 0;  }  };  /\* Linked List based implementation of stack \*/  class stack  {    node\* top;  int length;  public:  /\* Constructor \*/  stack()  {  top = NULL;  length = 0;  }  /\* Methods \*/  bool isEmpty()  {  return length == 0;  }  void push(int data)  {  cout << "\tValue has been added" << endl;  node\* newnode = new node;  newnode->data = data;  newnode->next = top;  top = newnode;  length++;  }  int pop()  {  int data = top->data;  node \*todelete = top;  top = top->next;    length--;  delete todelete;  return data;  }  void clear()  {  while (!isEmpty())  {  pop();  }  }  int peek()  {  return top->data;  }  };  /\* Main Function \*/  int main()  {  stack st;  int option;  while (true)  {  system("cls");  cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STACK - LINKED LIST IMPLEMENTATION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  cout << "\t1.Push into stack\n\t2.Pop from the stack\n\t3.Peek from the stack\n\t4.Clear Stack" << endl << endl;  cout << "\tYour command: ";  cin >> option;  cout << endl;  /\* Operation according to the command \*/  switch (option)  {  int data;  case 1:  cout << "\tEnter the number you want to push: ";  cin >> data;  st.push(data);  break;  case 2:  if (!st.isEmpty())  {  st.pop();  cout << "\tValue has been popped from the stack" << endl;  }  else  {  cout << "\tStack is empty" << endl;  }  break;  case 3:  if (!st.isEmpty())  {  cout << "\tValue at the top of the stack is: " << st.peek() << endl;  }  else  {  cout << "\tStack is empty" << endl;  }  break;  case 4:  st.clear();  cout << "\tStack has been cleared" << endl;  break;  default:  cout << "\tEnter a valid option..." << endl;  break;  }  cout << "\n\t";  system("pause");  }  return 0;  } |
| 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.

**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 <string>  #define MAX\_SIZE 50 /\* setting max size for array size \*/  using namespace std;  /\* Function to be used \*/  void parenthesesCheck(string);  /\* Array based implementation of stack \*/  template<class T> //using templates for generic stack implementation  class stack  {  T \*arr = new T[MAX\_SIZE];  int top;  int length;  public:  /\* Constructor \*/  stack()  {  top = -1;  length = 0;  }  /\* Methods \*/  bool isEmpty()  {  return length == 0;  }  bool isFull()  {  return length == MAX\_SIZE;  }  void push(T data)  {  if (isFull())  {  cout << "\tStack is full.." << endl;  return;  }  arr[length] = data;  length++;  top++;  }  T pop()  {  T data = arr[top];  top--;  length--;  return data;  }  void clear()  {  delete[] arr;  }  T peek()  {  return arr[top];  }  };  /\* Main Function \*/  int main()  {  string exp;  cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PARENTHESES CHECK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  cout << "\tEnter an expression: ";  getline(cin, exp);  parenthesesCheck(exp);  cout << endl << endl;  return 0;  }  void parenthesesCheck(string exp)  {  stack<char> st;  stack<int> pos;  int i;  for (i = 0; i < exp.length(); i++)  {  if (exp[i] == '[' || exp[i] == '{' || exp[i] == '(')  {  st.push(exp[i]);  pos.push(i+1);  }  else if (exp[i] == ')')  {  if (st.peek() == '(')  {  st.pop();  pos.pop();  }  else  break;  }  else if (exp[i] == '}')  {  if (st.peek() == '{')  {  st.pop();  pos.pop();  }  else  break;  }  else if (exp[i] == ']')  {  if (st.peek() == '[')  {  st.pop();  pos.pop();  }  else  break;  }  }  if (i == exp.length())  cout << "\n\tThis expression is correct." << endl;  else  cout << "\n\tThis expression is not correct. Error at character #" << pos.peek() << " \"" << st.peek() << "\" is not closed" << endl;  } |
| OUTPUT: |
|  |

**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 <string>  #include <sstream>  #include <cmath>  using namespace std;  /\* Function Used \*/  string Infix\_to\_Postfix(string);  int prec(char);  /\* Node Class \*/  class node  {  public:  char data;  node\* next;  public:  /\* Constructor \*/  node()  {  next = NULL;  }  };  /\* Linked List based implementation of stack \*/  class stack  {  node\* top;  int length;  public:  /\* Constructor \*/  stack()  {  top = NULL;  length = 0;  }  /\* Methods \*/  bool isEmpty()  {  return length == 0;  }  void push(char data)  {  node\* newnode = new node;  newnode->data = data;  newnode->next = top;  top = newnode;  length++;  }  char pop()  {  char data = top->data;  node \*todelete = top;  top = top->next;  length--;  delete todelete;  return data;  }  void clear()  {  while (!isEmpty())  {  pop();  }  }  char peek()  {  return top->data;  }  };  /\* Global Variable \*/  stack st;  /\* Main Function \*/  int main()  {  string exp;  cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INFIX TO POSTFIX\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  cout << "\tEnter an infix expression: ";  getline(cin, exp);  cout << "\n\tThe following expression in Postfix form: " << Infix\_to\_Postfix(exp) << endl << endl;  return 0;  }  /\* Function to convert \*/  string Infix\_to\_Postfix(string exp)  {  string postfixExp = "";    for (int i = 0; i < exp.length(); i++)  {  if (isdigit(exp[i]))  {  postfixExp += exp[i];  }  else  {  postfixExp += ' '; // append a space after every number  if (exp[i] == ' ')  {  continue;  }  else if (exp[i] == '(')  {  st.push(exp[i]);  }  else if (exp[i] == ')')  {  while (!st.isEmpty() && st.peek() != '(')  {  postfixExp += st.pop();  }  if (!st.isEmpty())  st.pop();  }  else  {  while (!st.isEmpty() && prec(st.peek()) >= prec(exp[i]))  {  postfixExp += st.peek();  st.pop();  }  st.push(exp[i]);  }  }  }    while (!st.isEmpty())  {  postfixExp += st.peek();  st.pop();  }  return postfixExp;  }  /\* Function for precedence check \*/  int prec(char op)  {  if (op == '^')  return 3;  else if (op == '\*' || op == '/')  return 2;  else if (op == '+' || op == '-')  return 1;  else  return 0;  } |
| OUTPUT: |
|  |

**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 <sstream>  #include <cmath>  using namespace std;  /\* Function Used \*/  int postfixEvaluation(string);  /\* Node Class \*/  class node  {  public:  string data;  node\* next;  public:  /\* Constructor \*/  node()  {  next = NULL;  }  };  /\* Linked List based implementation of stack \*/  class stack  {  node\* top;  int length;  public:  /\* Constructor \*/  stack()  {  top = NULL;  length = 0;  }  /\* Methods \*/  bool isEmpty()  {  return length == 0;  }  void push(string data)  {  node\* newnode = new node;  newnode->data = data;  newnode->next = top;  top = newnode;  length++;  }  string pop()  {  string data = top->data;  node \*todelete = top;  top = top->next;  length--;  delete todelete;  return data;  }  void clear()  {  while (!isEmpty())  {  pop();  }  }  string peek()  {  return top->data;  }  };  /\* Global Variable \*/  stack st;  /\* Main Function \*/  int main()  {  string exp;  cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*POSTFIX EXPRESSION SOLVER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;  cout << "\tEnter a postfix expression: ";  getline(cin, exp);  cout << "\n\tAnswer to the following expression is: " << postfixEvaluation(exp) << endl << endl;  return 0;  }  int postfixEvaluation(string exp)  {  string num;  for (int i = 0; i < exp.length(); i++)  {  while (isdigit(exp[i]))  {  num.append(to\_string(exp[i] - 48));  i++;  }  if (num != "")  {  st.push(num);  num = "";  }  if (exp[i] == '+' || exp[i] == '-' || exp[i] == '\*' || exp[i] == '/' || exp[i] == '^')  {  int op1 = stoi(st.pop());  int op2 = stoi(st.pop());  int res = 0;  switch (exp[i])  {  case '+':  res = op2 + op1;  break;  case '-':  res = op2 - op1;  break;  case '\*':  res = op2 \* op1;  break;  case '/':  res = op2 / op1;  break;  case '^':  res = pow(op2, op1);  break;  default:  break;  }  st.push(to\_string(res));  }  }  return stoi(st.pop());  } |
| OUTPUT: |
|  |

**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.