

## Experiment 4

**Student Name:** Lipakshi

**Branch:** CSE

**Semester:** 5<sup>th</sup>

**Subject Name:** DAA Lab

**UID:** 20BCS5082

**Section/Group:** WM\_607\_B

**Date of Performance:** 02/09/2022

**Subject Code:** 21-CSP-312

### **1. Aim/Overview of the practical:**

- (i) Code to Insert and Delete an element at the beginning and at end in Doubly and Circular Linked List.
- (ii) Code to push & pop and check Iseempty, Isfull and Return top element in stacks using templates.

### **2. Task to be done/ Which logistics used:**

Insert and delete an element from a doubly circular linked list.

### **3. Algorithm/Flowchart:**

Q1. For Insertion and deletion of an element at the beginning and at end in Doubly and Circular Linked List.

1. Start.
2. For insertion in the end if the list is empty start pointer points to the first node the list. If the list is non empty previous pointer of M points to last node, next pointer of M points to first node and last node's next pointer points to this M node and first node's previous pointer points to this M node
3. For Insertion at the beginning if the list is empty T next pointer points to first node of the list, T previous pointer points to last node the list, last node's next pointer points to this T node, first node's previous pointer also points this T node and shift 'Start' pointer to this T node.
4. If the list is not empty, then we define two pointers curr and prev\_1 and initialize the pointer curr points to the first node of the list, and prev\_1 = NULL.
5. Traverse the list using the curr pointer to find the node to be deleted and before moving from curr to the next node, every time set prev\_1 = curr.
6. If the node is found, check if it is the only node in the list. If yes, set start = NULL and free the node pointing by curr.
7. If the list has more than one node, check if it is the first node of the list. The condition to check this is (curr == start). If yes, then move prev\_1 to the last node (prev\_1 = start -> prev).
8. If curr is not the first node, we check if it is the last node in the list. The condition to check this is (curr -> next == start). If yes, set prev\_1 -> next = start and start -> prev = prev\_1. Free the node pointing by curr.

9. If the node to be deleted is neither the first node nor the last node, declare one more pointer temp and initialize the pointer temp points to the next of curr pointer (temp = curr->next). Now set, prev\_1 -> next = temp and temp -> prev = prev\_1. Free the node pointing by curr. 8.
10. Stop and print the result.

#### 4. Steps for experiment/practical/Code:

```
#include<iostream>
using namespace std;

class node {
public:
    node* next;
    node* prev;
    int data;
};

void insert_front(node** head)
{
    cout << "\nEnter Data to insert at front : \n";
    node* new_node = new node;
    cin >> new_node->data;
    if (*head == NULL) {
        new_node->next = new_node;
        new_node->prev = new_node;
        *head = new_node;
    }
    else {
        new_node->next = *head;
        new_node->prev = (*head)->prev;
        ((*head)->prev)->next = new_node;
        (*head)->prev = new_node;
        *head = new_node;
    }
    cout << "Data inserted at front \n";
}

void insert_end(node** head)
{
    cout << "\nEnter Data to insert at end : \n";

    node* new_node = new node;
    cin >> new_node->data;

    if (*head == NULL) {
        new_node->next = new_node;
```

```
        new_node->prev = new_node;
        *head = new_node;
    }
    else {
        node* curr = *head;
        while (curr->next != *head)
            curr = curr->next;
        new_node->next = curr->next;

        new_node->prev = curr;
        (curr->next)->prev = new_node;
        curr->next = new_node;
    }

    cout << "Data inserted at last\n";
}

void delete_front(node** head)
{
    if (*head == NULL) {
        cout << "\nList in empty!!\n";
    }
    else if ((*head)->next == *head) {
        delete *head;
        *head = NULL;
    }
    else {
        node* curr = new node;
        curr = (*head)->next;
        curr->prev = (*head)->prev;
        ((*head)->prev)->next = curr;
        delete *head;
        *head = curr;
    }
    cout << "\nData Deleted from front\n";
}

void delete_end(node** head)
{
    if (*head == NULL) {
        cout << "\nList is Empty!!\n";
    }
    else if ((*head)->next == *head) {
        delete *head; *head = NULL;
    }
}
```

```
else {
    node* curr = new node;
    curr = *head;
    while (curr->next != (*head)) {
        curr = curr->next;
    }
    (curr->prev)->next = curr->next;
    (curr->next)->prev = curr->prev;
    delete curr;
}
cout << "\nData Deleted from last\n";
}
void display(node* head)
{
    node* curr = head; if (curr == NULL) cout << "\n List is Empty!!"; else {
        do {
            cout << curr->data << "->";
            curr = curr->next;
        } while (curr != head);
    }
}
int main()
{
    int choice;
    char menu = 'y';
    node* head = NULL;
    insert_front(&head);
    display(head);
    insert_front(&head);
    display(head);
    insert_end(&head);
    display(head);
    insert_end(&head);
    display(head);
    delete_front(&head);
    display(head);
    delete_end(&head);
    display(head);
    return 0;
}
```

## 5. Observations/Discussions/ Complexity Analysis:

**Time Complexity:**  $O(n)$

## 6. Result/Output/Writing Summary:

### Output

```
/tmp/lIlaov8Phy.o
Enter Data to insert at front :
5
Data inserted at front
5->
Enter Data to insert at front :
8
Data inserted at front
8->5->
Enter Data to insert at end :
8
Data inserted at last
8->5->8->
Enter Data to insert at end :
9
Data inserted at last
8->5->8->9->
Data Deleted from front
5->8->9->
Data Deleted from last
5->8->
```

## 1. Aim/Overview of the practical:

(ii) Code to push & pop and check Isempy, Isfull and Return top element in stacks using templates.

## 2. Task to be done/ Which logistics used:

Using C++ templates perform push and pop operation on stacks.

## 3. Algorithm/Flowchart:

1. Start.
2. First we will define the size.
3. Then we will create a class template called Stack.
4. Then we will check the top of stack using - template `<class T> Stack<T>::Stack() { top = -1;`
5. Then we will push elements into the stack using templates.
6. Using template, we will check whether the stack is empty or is full.
7. Then we will pop an element of stack using templates.
8. We will check the top element using template `<class T> T Stack<T>::topElement()`
9. print the result
10. Stop

## 4. Steps for experiment/practical/Code:

```
#include <iostream>
#include <string>
using namespace std;
#define SIZE 5
template <class T> class Stack { public:
    Stack();
    void push(T k);
    T pop();
    T topElement();
    bool isFull();
    bool isEmpty();
private:
    int top;
    T st[SIZE];
};
template <class T> Stack<T>::Stack() { top = -1; }
template <class T> void Stack<T>::push(T k)
```

```
{
    if (isFull()) {
        cout << "Stack is full\n";
    }
    cout << "Inserted element " << k << endl;
    top = top + 1;
    st[top] = k;
}
template <class T> bool Stack<T>::isEmpty()
{
    if (top == -1)
        return 1; else
        return 0;
}
template <class T> bool Stack<T>::isFull()
{
    if (top == (SIZE - 1))          return 1;
    else
        return 0;
}

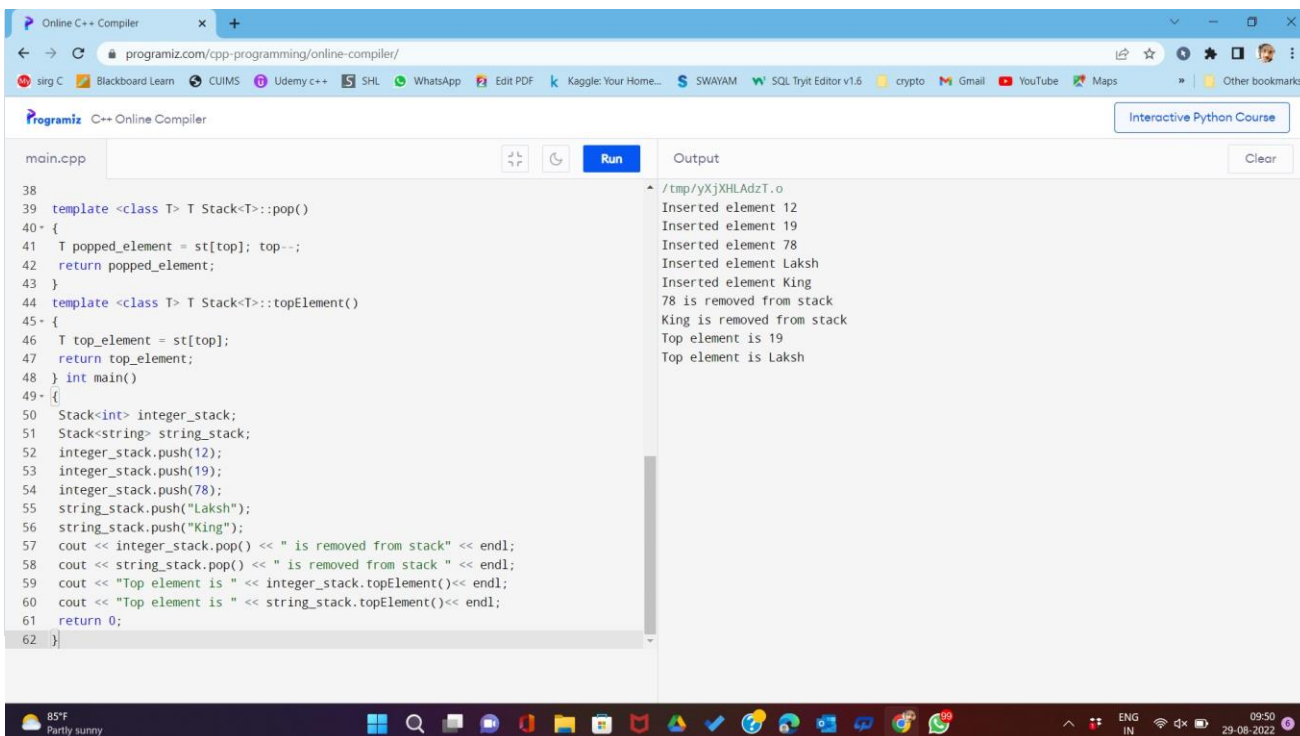
template <class T> T Stack<T>::pop()
{
    T popped_element = st[top]; top--;
    return popped_element;
}
template <class T> T Stack<T>::topElement()
{
    T top_element = st[top];
    return top_element;
}
int main()
{
    Stack<int> integer_stack;
    Stack<string> string_stack;
    integer_stack.push(10);
    integer_stack.push(15);
    integer_stack.push(65);
    string_stack.push("Elon Musk");
    string_stack.push("Bill Gates");
    cout << integer_stack.pop() << " is removed from stack" << endl;
    cout << string_stack.pop() << " is removed from stack " << endl;
    cout << "Top element is " << integer_stack.topElement() << endl;
}
```

```
cout << "Top element is " << string_stack.topElement() << endl;  
return 0;  
}
```

## 5. Observations/Discussions/ Complexity Analysis:

Time Complexity:  $O(1)$

## 6. Result/Output/Writing Summary:



The screenshot shows a web browser window with the URL `programiz.com/cpp-programming/online-compiler/`. The browser's address bar and tabs are visible. The main content area displays a C++ program in a text editor, with a 'Run' button to its right. The code defines two stacks, `integer_stack` and `string_stack`, and performs several push and pop operations. The output window on the right shows the results of these operations, including the removal of elements and the retrieval of the top element from the string stack.

```
main.cpp  
38  
39 template <class T> T Stack<T>::pop()  
40 {  
41     T popped_element = st[top]; top--;  
42     return popped_element;  
43 }  
44 template <class T> T Stack<T>::topElement()  
45 {  
46     T top_element = st[top];  
47     return top_element;  
48 } int main()  
49 {  
50     Stack<int> integer_stack;  
51     Stack<string> string_stack;  
52     integer_stack.push(12);  
53     integer_stack.push(19);  
54     integer_stack.push(78);  
55     string_stack.push("Laksh");  
56     string_stack.push("King");  
57     cout << integer_stack.pop() << " is removed from stack" << endl;  
58     cout << string_stack.pop() << " is removed from stack" << endl;  
59     cout << "Top element is " << integer_stack.topElement() << endl;  
60     cout << "Top element is " << string_stack.topElement() << endl;  
61     return 0;  
62 }
```

Output

```
/tmp/yXjXHLAdzT.o  
Inserted element 12  
Inserted element 19  
Inserted element 78  
Inserted element Laksh  
Inserted element King  
78 is removed from stack  
King is removed from stack  
Top element is 19  
Top element is Laksh
```