Experiment-1.4

Student Name: Yash Kumar **UID: 20BCS9256**
Branch: CSE **Section/Group: 616-B**
Semester: 5 **Date of Performance: 12/09/2022**
Subject Name: DAA Lab

1. Aim/Overviewofthepractical:

(i) CodetoInsertand Delete anelement atthebeginning andatend inDoublyand CircularLinkedList.

2. Tasktobedone/Whichlogisticsused:

Insertanddeleteanelementfrom adoubly circularlinkedlist.

3. Algorithm/Flowchart:

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 nonempty previous pointer of M points to last node, next pointer of M points to first node and last node'snextpointerpoints to thisMnodeandfirstnode's previouspointer points to thisMnode
3. For Insertion atthebeginning ifthelist isempty Tnextpointerpoints tofirst node ofthe list, Tpreviouspointer points to last node the list, last node's next pointer points to this T node, first node's previouspointeralso pointsthisTnodeand shift 'Start' pointerto thisT node.
4. If the list is not empty, then we define two pointers curr and prev_1 and initialize the pointer curr pointstothefirst nodeof thelist, and prev_1 =NULL.
5. Traversethe listusing thecurrpointerto findthenodeto bedeletedandbeforemovingfrom currtothenextnode, every timesetprev_1 =curr.
6. Ifthe nodeis found,checkif it istheonly nodein the list.If yes, setstart = NULLand freethenodepointingby curr.
7. Ifthe list hasmorethan onenode,check if it isthefirst nodeof thelist. The condition to check thisis(curr==start).Ifyes, then move prev_1to thelastnode(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). Ifyes,setprev_1->next= startand start ->prev =prev_1.Freethe nodepointing bycurr.
9. If the node to be deleted is neither the first node nor the last node, declare one more pointer temp andinitializethepointertemppointsto thenextof currpointer(temp=curr>next).Now set,prev_1 ->next =tempand temp->prev=prev_1.Freethe nodepointing bycurr.8.
10. Stopandprinttheresult.

4. Stepsforexperiment/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";
```

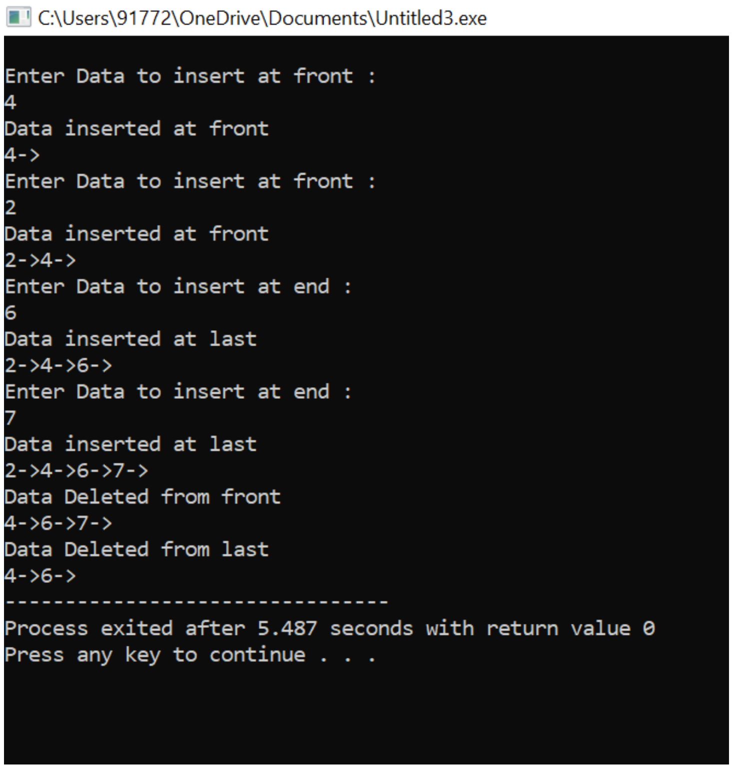
```
}
void delete_front(node** head)
{
if (*head == NULL) {
cout<< "\nList in empty!!\n";
}
else if ((*head)->next == *head) { delete *head;
*head = NULL;
}    else {
node* curr = new node; curr = (*head)->next;
curr->prev = (*head)->prev; ((*head)->prev)->next = curr; delete *head;
*head = curr;
}
cout<< "\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);
}
}
intmain()
{
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/ComplexityAnalysis:

TimeComplexity:O(n)

6. Result/Output/WritingSummary:



```
C:\Users\91772\OneDrive\Documents\Untitled3.exe

Enter Data to insert at front :
4
Data inserted at front
4->
Enter Data to insert at front :
2
Data inserted at front
2->4->
Enter Data to insert at end :
6
Data inserted at last
2->4->6->
Enter Data to insert at end :
7
Data inserted at last
2->4->6->7->
Data Deleted from front
4->6->7->
Data Deleted from last
4->6->
--------------------------------
Process exited after 5.487 seconds with return value 0
Press any key to continue . . .
```

1. Aim/Overview of the practical:

Code to push & pop and check Isempty, Isfull and Return top element in stacks using templates.

2. Task to be done/ which logistics used:

Using templates, perform all stack operations such as push, pop, peek, isfull, isempty, overflow and underflow condition also.

3. Steps for experiment/practical/Code

Algorithm:

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 Stack::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 T Stack::topElement()
9. Print the result
10. Stop

Program Code:

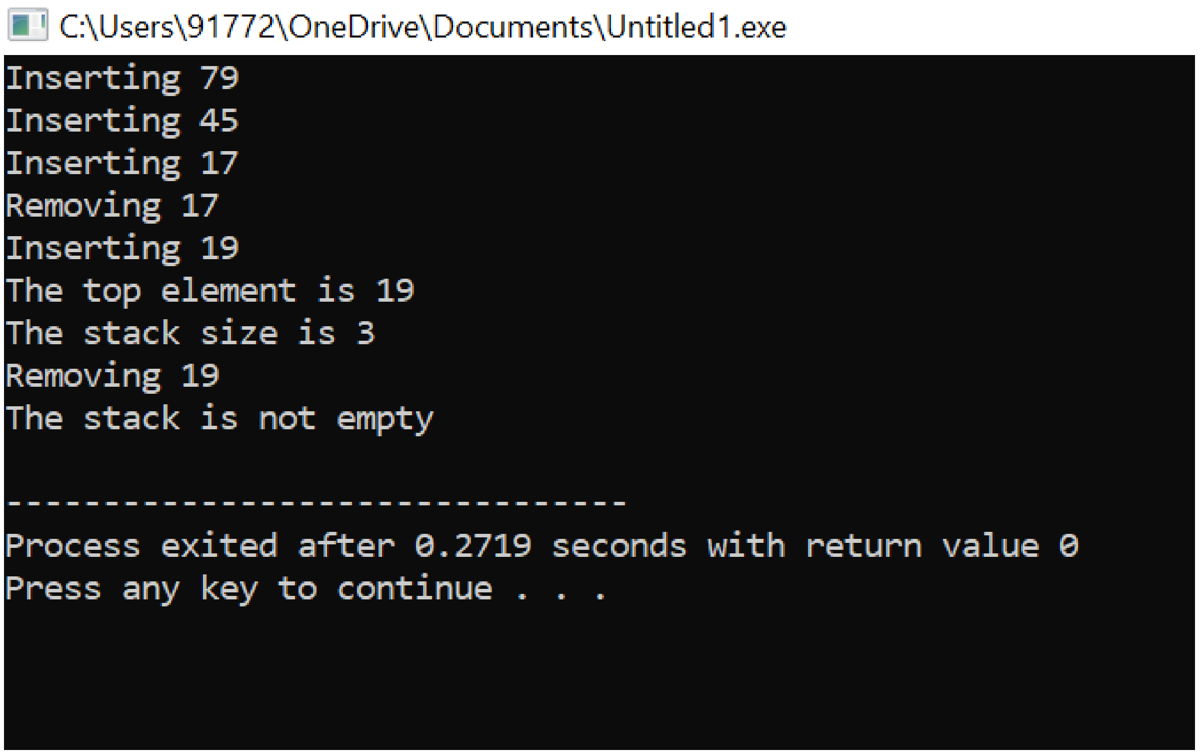
```
#include <iostream>
#include <cstdlib>
using namespace std;

template <class X>
class stack
{
   X *arr;
int top;
int capacity;

public:
stack(int size = SIZE);

   void push(X);
   X pop();
   X peek();

intsize();
   bool isEmpty();
```

```
    bool isFull();


    ~stack() {
delete[] arr;
    }
};

template <class X>
stack<X>::stack(int size)
{
arr = new X[size];
    capacity = size;
    top = -1;
}

template <class X>
void stack<X>::push(X x)
{
    if (isFull())
    {
cout<< "Overflow\nProgram Terminated\n";
        exit(EXIT_FAILURE);
    }

cout<< "Inserting " << x <<endl;
arr[++top] = x;
}

template <class X>
X stack<X>::pop()
{
    if (isEmpty())
    {
cout<< "Underflow\nProgram Terminated\n";
```

```
        exit(EXIT_FAILURE);
    }

cout<< "Removing " <<peek() <<endl;

    return arr[top--];
}

template <class X>
X stack<X>::peek()
{
    if (!isEmpty()) {
        return arr[top];
    }
    else {
        exit(EXIT_FAILURE);
    }
}

template <class X>
int stack<X>::size() {
    return top + 1;
}
template <class X>
bool stack<X>::isEmpty() {
    return top == -1;
}
template <class X>
bool stack<X>::isFull() {
    return top == capacity - 1;
}
intmain()
{
    stack<int>pt(5);
```

```
pt.push(79);
pt.push(45);
pt.push(17);
pt.pop();
pt.push(19);
cout<< "The top element is " <<pt.peek() <<endl;
cout<< "The stack size is " <<pt.size() <<endl;
pt.pop();

   if (pt.isEmpty()) {
cout<< "The stack is empty\n";
   }
   else {
cout<< "The stack is not empty\n";
   }
   return 0;
}
```

4. Result/Output/Writing Summary:



C:\Users\91772\OneDrive\Documents\Untitled1.exe

```
Inserting 79
Inserting 45
Inserting 17
Removing 17
Inserting 19
The top element is 19
The stack size is 3
Removing 19
The stack is not empty

--------------------------------
Process exited after 0.2719 seconds with return value 0
Press any key to continue . . .
```