Experiment: 4

Student Name: Rohit Kumar

UID: 20BCS1922

Branch: CSE

Section/Group: 20BCS-MM-808/B

Semester: 5th

Date of Performance: 01/08/2022

Subject Name: DAA Lab

Subject Code: 21-CSP-312

1.Aim/Overview of the practical:

a) Code to Insert and Delete an element at the beginning and at end in doubly and Circular Linked List

b) Code to push & pop and check Is empty, Is full and Return top element in stacks using templates

2.Task to be done/ Which logistics used:

Code to Insert and Delete an element at the beginning and at end in doubly and Circular Linked List

Code to push & pop and check Is empty, Is full and Return top element in stacks using templates

3.Algorithm/Flowchart (For programming based labs):

```
Step-1: [Check for overflow]
                if(rear==MAX)
                        Print("Queue is Overflow");
                        return;
Step-2: [Insert Element]
                else
                        rear=rear+1;
                        q[rear]=no;
                [Set rear and front pointer]
                if rear=0
                        rear=1;
                if front=0
                        front=1;
Step-3: return
```

4.Steps for experiment/practical/Code:

```
///  Code to Insert and Delete an element at the beginning and at end in doubly and Circular Linked List
// C++ program to illustrate inserting a Node in
// a Circular Doubly Linked list in begging, end
// and middle
#include <bits/stdc++.h>
using namespace std;

// Structure of a Node
struct Node
{
	int data;
	struct Node *next;
	struct Node *prev;
};

// Function to insert at the end
void insertEnd(struct Node** start, int value)
{
	// If the list is empty, create a single node
	// circular and doubly list
	if (*start == NULL)
	{
		struct Node* new_node = new Node;
		new_node->data = value;
		new_node->next = new_node->prev = new_node;
		*start = new_node;
		return;
	}

	// If list is not empty
```

```
        /* Find last node */
        Node *last = (*start)->prev;

        // Create Node dynamically
        struct Node* new_node = new Node;
        new_node->data = value;

        // Start is going to be next of new_node
        new_node->next = *start;

        // Make new node previous of start
        (*start)->prev = new_node;

        // Make last previous of new node
        new_node->prev = last;

        // Make new node next of old last
        last->next = new_node;
}

// Function to insert Node at the beginning
// of the List,
void insertBegin(struct Node** start, int value)
{
        // Pointer points to last Node
        struct Node *last = (*start)->prev;

        struct Node* new_node = new Node;
        new_node->data = value; // Inserting the data

        // setting up previous and next of new node
        new_node->next = *start;
        new_node->prev = last;
```

```
        // Update next and previous pointers of start
        // and last.
        last->next = (*start)->prev = new_node;

        // Update start pointer
        *start = new_node;
}

// Function to insert node with value as value1.
// The new node is inserted after the node with
// with value2
void insertAfter(struct Node** start, int value1,
                                                    int value2)
{
        struct Node* new_node = new Node;
        new_node->data = value1; // Inserting the data

        // Find node having value2 and next node of it
        struct Node *temp = *start;
        while (temp->data != value2)
                temp = temp->next;
        struct Node *next = temp->next;

        // insert new_node between temp and next.
        temp->next = new_node;
        new_node->prev = temp;
        new_node->next = next;
        next->prev = new_node;
}


void display(struct Node* start)
```

```
{
        struct Node *temp = start;

        printf("\nTraversal in forward direction \n");
        while (temp->next != start)
        {
                printf("%d ", temp->data);
                temp = temp->next;
        }
        printf("%d ", temp->data);

        printf("\nTraversal in reverse direction \n");
        Node *last = start->prev;
        temp = last;
        while (temp->prev != last)
        {
                printf("%d ", temp->data);
                temp = temp->prev;
        }
        printf("%d ", temp->data);
}

/* Driver program to test above functions*/
int main()
{
        /* Start with the empty list */
        struct Node* start = NULL;

        // Insert 5. So linked list becomes 5->NULL
        insertEnd(&start, 5);

        // Insert 4 at the beginning. So linked
        // list becomes 4->5
```

```
    insertBegin(&start, 4);

    // Insert 7 at the end. So linked list
    // becomes 4->5->7
    insertEnd(&start, 7);

    // Insert 8 at the end. So linked list
    // becomes 4->5->7->8
    insertEnd(&start, 8);

    // Insert 6, after 5. So linked list
    // becomes 4->5->6->7->8
    insertAfter(&start, 6, 5);

    printf("Created circular doubly linked list is: ");
    display(start);

    return 0;
}
```

b)

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

#define MAX 1000 //max size for stack

class Stack
{
  int top;
  public:
  int myStack[MAX]; //stack array

  Stack() { top = -1; }
  bool push(int x);
  int pop();
  bool isEmpty();
```

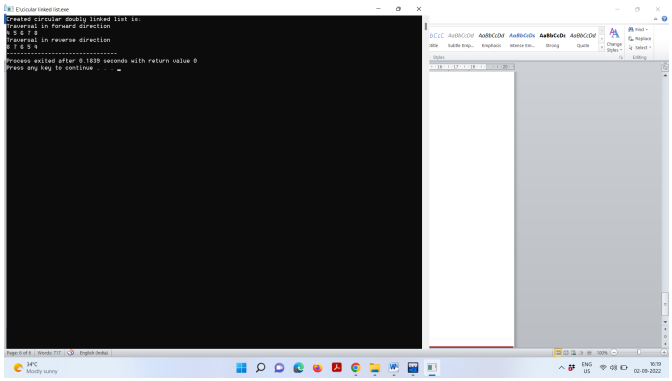
```
};
  //pushes element on to the stack
  bool Stack::push(int item)
  {
    if (top >= (MAX-1)) {
    cout << "Stack Overflow!!!";
    return false;
  }
else {
  myStack[++top] = item;
  cout<<item<<endl;
  return true;
  }
}

//removes or pops elements out of the stack
int Stack::pop()
{
  if (top < 0) {
    cout << "Stack Underflow!!";
    return 0;
  }
else {
     int item = myStack[top--];
    return item;
  }
}

//check if stack is empty
bool Stack::isEmpty()
{
  return (top < 0);
}

// main program to demonstrate stack functions
int main()
{
```

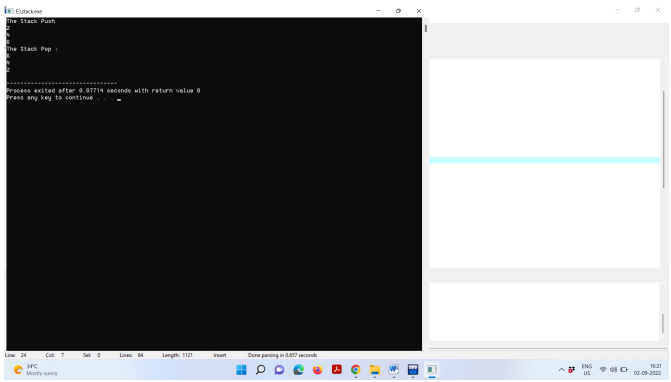
```
class Stack stack;
cout<<"The Stack Push "<<endl;
stack.push(2);
stack.push(4);
stack.push(6);
cout<<"The Stack Pop : "<<endl;
while(!stack.isEmpty())
  {
  cout<<stack.pop()<<endl;
  }
return 0;
}
```

5.Observations/Discussions/ Complexity Analysis:

Time complexity of finding GCD of two number using Euclidean method is O(log n).

6. Result/Output/Writing Summary:





Learning outcomes (What I have learnt):

1. To know how Euclidean algorithm works.

2. To learn how to use recursion for solving problems.