**Worksheet-4**

**Student Name:-** Pushpraj Roy **UID:-** 20BCS9866

**Branch:-** BE- CSE **Section/Group:-** WM\_617 “A”

**Subjetct Code:-** 20CSP-312 **Semester:-** 5th

**Subject Name:-** DAA Lab

1. **Aim/Overview of the practical: -**
2. Code for inserting and removing elements at the start and end of a doubly and circular linked list.

# Task to be done/ Which logistics used :-

1. Code to Insert and Delete an element at the beginning and at end in doubly and Circular Linked List
2. **Algorithm/Flowchart :-**
3. Start.
4. 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
5. 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.
6. If the list is pty, 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.
7. 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.
8. 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.
9. 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).
10. 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.
11. 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.
12. Stop and print the result.
13. **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;

}

else {

}

new\_node->prev = new\_node;

\*head = new\_node;

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);

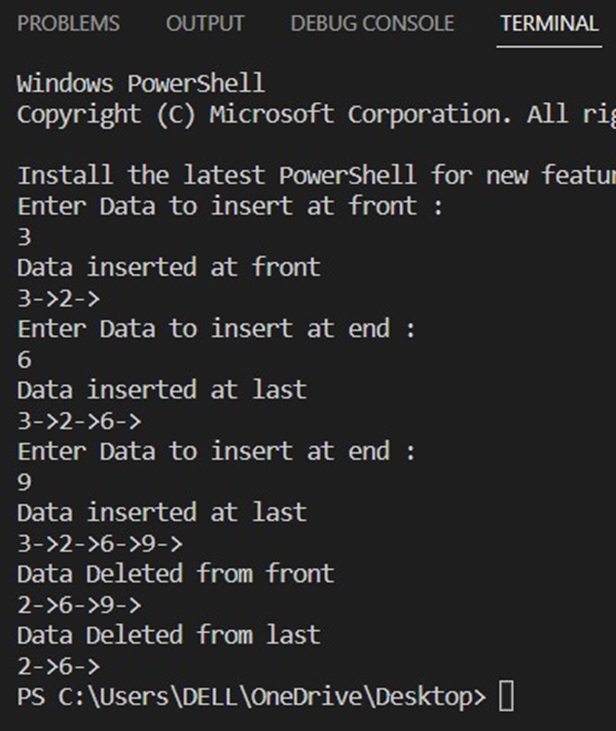
delete\_end(&head);

display(head);

return 0;

}

1. **Result/Output :-**

****

1. **Aim/Overview of the practical: -**
2. Using templates, write code to push and pop elements, check is empty and is full, and return the top element in stacks.

# Task to be done/ Which logistics used :-

1. Code to push & pop and check Is empty, Is full and Return top element in stacks using templates
2. **Algorithm/Flowchart :-**
3. Start.
4. First we will define the size.
5. Then we will create a class template called Stack.
6. Then we will check the top of stack using - template <class T> Stack<T>::Stack() { top = -1;
7. Then we will push elements into the stack using templates.
8. Using template, we will check whether the stack is empty or is full.
9. The we will pop an element of stack using templates.
10. We will check the top element using template <class T> T Stack<T>::topElement().
11. Print the result.
12. Stop.
13. **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(20);

integer\_stack.push(30);

string\_stack.push("PUSHPRAJ");

string\_stack.push("ROY");

cout << integer\_stack.pop() << " is removed from stack" << endl;

cout << string\_stack.pop() << " is removed from stack " << endl;

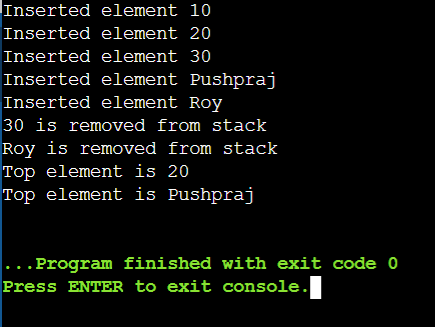
cout << "Top element is " << integer\_stack.topElement()<< endl;

cout << "Top element is " << string\_stack.topElement()<< endl;

return 0;

}

1. **Result/Output :-**

****

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |