**Experiment Title 1.4**

**Student Name: Sahul Kumar Parida UID: 20BCS4919**

**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 23/08/2022**

**Subject Name: Design and Analysis Algorithm Lab**

**Subject Code: 20CSP-312**

# Aim/Overview of the practical:

**1.4(i)**

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

1. **Task to be done/ Which logistics used:**

Using various methods, implementation of doubly linked list.

1. **Algorithm/Flowchart (For programming based labs):**

 Start.

 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.

 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.

 For deletion at the end.

 If the list is empty, simply return it.

 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.

 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.

 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.

 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).

 After prev\_1 reaches the last node, set start = start -> next and prev\_1 -> next = start

and start ->prev = prev\_1. Free the node pointing by curr.

 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.

 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. print the result.

 Stop

1. **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){

}

else{

new\_node->next = new\_node; 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 at 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 at last\n";

}

void display(node \*head){ node \*curr = head;

if (curr == NULL)

cout << "\nList 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;

}

# Observations/Discussions/ Complexity Analysis:

Time Complexity = O(n).

1. **Result/Output/Writing Summary:**

****

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

**1.4(ii)**

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

1. **Task to be done/ Which logistics used:**

Using various methods, implementation of stack.

1. **Algorithm/Flowchart (For programming based labs):**

 Start.

 First we will define the size.

 Then we will create a class template called Stack.

 Then we will check the top of stack using - template

<class T> Stack<T>::Stack() { top = - 1}.

 Then we will push elements into the stack using templates.

 Using template we will check whether the stack is empty or is full.

 The we will pop an element of stack using templates.

 We will check the top element using template

<class T> T Stack<T>::topElement()

 print the result

 Stop

1. **Steps for experiment/practical/Code:**

#include <iostream> #include <cstdlib> using namespace std; #define SIZE 10

template <class X> class stack{

X \*arr; int top;

int capacity;

public:

stack(int size = SIZE);

void push(X); X pop();

X peek(); int size();

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;

}

int main(){ stack<string> pt(2);

pt.push("Sahul"); pt.push("Parida");

pt.pop();

pt.pop(); pt.push("20BCS4919;

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";

}

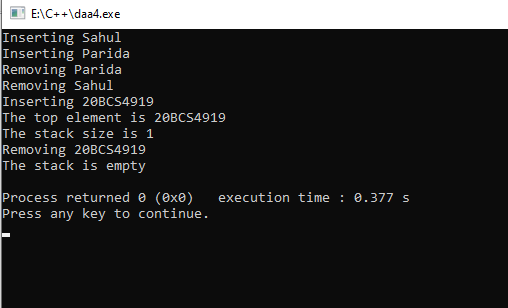
return 0;

}

# Observations/Discussions/ Complexity Analysis:

Time Complexity = O(1).

1. **Result/Output/Writing Summary:**

****

**Learning outcomes (What I have learnt):**

1. Learn Implementation of linked list and stack.
2. Learnt about optimization of the problem.
3. Learnt about time complexities and space complexities.