# Experiment 1.4

## Student Name: UID: 21BCS

**Branch: CSE Section/Group:**

## Semester: 5th Date of Performance: 22/8/23 Subject Name: Design and analysis of algorithm

**Subject Code: 21CSH311**

1. **Aim:** Code to perform operation on singly and doubly Linked list
2. **Objective:** To perform insertion and deletion on singly and doubly Linked list

.

# Algorithm:

**Singly Linked List:**

**Insertion at the Beginning:**

1. Create a new node with the given value.
2. Set the new node's next pointer to the current head.
3. Update the head pointer to point to the new node.

**Insertion at the End:**

1. Create a new node with the given value.
2. If the list is empty, set the head pointer to the new node.
3. Otherwise, traverse the list until you reach the last node.
4. Set the last node's next pointer to the new node.

**Deletion at the Beginning:**

1. If the list is empty, return.
2. Store the head node in a temporary variable.
3. Update the head pointer to point to the next node.
4. Delete the temporary variable (old head).

**Deletion at the End:**

1. If the list is empty, return.
2. If there's only one node, delete the node and set the head pointer to null.
3. Traverse the list until you reach the second-to-last node.
4. Set the second-to-last node's next pointer to null.
5. Delete the last node.

**Doubly Linked List Algorithms:**

**Insertion at the Beginning:**

1. Create a new node with the given value.
2. If the list is empty, set both head and tail pointers to the new node.
3. Otherwise, update the new node's next pointer to the current head.
4. Update the current head's prev pointer to the new node.
5. Update the head pointer to point to the new node.

**Insertion at the End:**

1. Create a new node with the given value.
2. If the list is empty, set both head and tail pointers to the new node.
3. Otherwise, set the new node's prev pointer to the current tail.
4. Update the current tail's next pointer to the new node.
5. Update the tail pointer to point to the new node.

**Deletion at the Beginning:**

1. If the list is empty, return.
2. Store the head node in a temporary variable.
3. Update the head pointer to point to the next node.
4. If there's a next node, update its prev pointer to null.
5. Delete the temporary variable (old head).

**Deletion at the End:**

1. If the list is empty, return.
2. If there's only one node, delete the node and set both head and tail pointers to null.
3. Store the tail node in a temporary variable.
4. Update the tail pointer to point to the second-to-last node.
5. If there's a new tail node, update its next pointer to null.
6. Delete the temporary variable (old tail).

# Code and output: A.

//singly

#include <iostream> class Node {

public:

int value; Node\* next; Node(int val) {

value = val; next = nullptr;

}

};

Node\* insertAtBeginning(Node\* head, int value) { Node\* newNode = new Node(value);

newNode->next = head; head = newNode; return head;

}

Node\* insertAtEnd(Node\* head, int value) { Node\* newNode = new Node(value);

if (!head) {

head = newNode;

} else {

Node\* current = head; while (current->next) {

current = current->next;

}

current->next = newNode;

}

return head;

}

Node\* deleteAtBeginning(Node\* head) { if (!head) {

return nullptr;

}

Node\* newHead = head->next; delete head;

return newHead;

}

Node\* deleteAtEnd(Node\* head) { if (!head) {

return nullptr;

}

if (!head->next) { delete head;

return nullptr;

}

Node\* current = head;

while (current->next->next) { current = current->next;

}

delete current->next; current->next = nullptr; return head;

}

void printLinkedList(Node\* head) { Node\* current = head;

while (current) {

std::cout << current->value << " "; current = current->next;

}

std::cout << std::endl;

}

int main() {

Node\* head = nullptr;

head = insertAtBeginning(head, 10); head = insertAtBeginning(head, 5); head = insertAtEnd(head, 20); printLinkedList(head);

head = deleteAtBeginning(head); head = deleteAtEnd(head); printLinkedList(head);

while (head) {

Node\* temp = head; head = head->next; delete temp;

}

return 0;

}

## B.

//doubly

#include <iostream>

class Node { public:

int value; Node\* prev; Node\* next;

Node(int val) { value = val; prev = nullptr; next = nullptr;

}

};

Node\* insertAtBeginning(Node\* head, int value) { Node\* newNode = new Node(value);

if (!head) {

head = newNode;

} else {

newNode->next = head; head->prev = newNode; head = newNode;

}

return head;

}

Node\* insertAtEnd(Node\* head, int value) { Node\* newNode = new Node(value);

if (!head) {

head = newNode;

} else {

Node\* current = head;

while (current->next) { current = current->next;

}

current->next = newNode; newNode->prev = current;

}

return head;

}

Node\* deleteAtBeginning(Node\* head) { if (!head) {

return nullptr;

}

Node\* newHead = head->next; if (newHead) {

newHead->prev = nullptr;

}

delete head; return newHead;

}

Node\* deleteAtEnd(Node\* head) { if (!head) {

return nullptr;

}

if (!head->next) { delete head; return nullptr;

}

Node\* current = head; while (current->next) {

current = current->next;

}

if (current->prev) {

current->prev->next = nullptr;

}

delete current;

return head;

}

void printLinkedListForward(Node\* head) { Node\* current = head;

while (current) {

std::cout << current->value << " "; current = current->next;

}

std::cout << std::endl;

}

void printLinkedListBackward(Node\* tail) { Node\* current = tail;

while (current) {

std::cout << current->value << " "; current = current->prev;

}

std::cout << std::endl;

}

int main() {

Node\* head = nullptr;

head = insertAtBeginning(head, 10); head = insertAtBeginning(head, 5); head = insertAtEnd(head, 20); printLinkedListForward(head); Node\* tail = head;

while (tail->next) { tail = tail->next;

}

printLinkedListBackward(tail); head = deleteAtBeginning(head); head = deleteAtEnd(head); printLinkedListForward(head); while (head) {

Node\* temp = head;

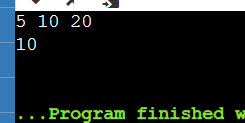
head = head->next; delete temp;

}

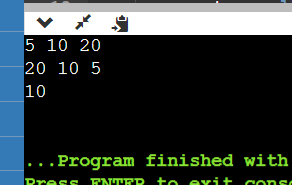
return 0;

}

Singly Linkedlist:-



Doubly Linkedlist:-



# Complexity: O(logn)