Experiment 1.4

Student Name : Utkarsh Joshi	UID : 21BCS9158
Branch : CSE	Section/Group : ST- 802 A
Semester : 5th	Date Of Performance : 10 Sept,2023
Subject Name : DAA Lab	Subject Code : 21CSH - 311

Aim : Apply the concept of Linked list and write code to Insert and Delete an element at the beginning and at end in Doubly and Singly Linked List. Objectives :

a) To make a Singly Linked list and perform the insertion and deletion at the beginning and at the end.
b) To make a Doubly Linked list and perform the insertion and deletion at the beginning and at the end.

Input/Apparatus Used :

1. C++ Compiler

Procedure/Algorithm :

a)Singly Linked List

Insert at beginning:

- Create a new node with the given data.
- Set the next pointer of the new node to the current head node. □
- Update the head pointer to point to the new node.

Insert at end:

- Create a new node with the given data.
- Set the next pointer of the new node to null.

- If the current head node is null, set the head pointer to the new node.
- Otherwise, traverse the linked list until you reach the last node.
- Set the next pointer of the last node to the new node.

Delete at beginning:

- If the current head node is null, return.
- Store the current head node in a temporary variable.
- Update the head pointer to point to the next node.
- Delete the temporary node.

Delete at end:

- If the current head node is null, return.
- Traverse the linked list until you reach the last node.
- Store the last node in a temporary variable.
- Set the next pointer of the previous node to null.
- Delete the temporary node.

b)Doubly Linked List

Insertion at the beginning:

- Create a new node with the given data.
- Set the previous pointer of the new node to null.
- Set the next pointer of the new node to the current head node.
- If the current head node is null, set the tail pointer to the new node.
- Update the head pointer to point to the new node.

Insertion at the end:

- Create a new node with the given data.
- Set the next pointer of the new node to null.

- If the current tail node is null, set the head pointer to the new node.

- Otherwise, set the next pointer of the current tail node to point to the new node.
- Update the tail pointer to point to the new node.

Deletion at the beginning:

- If the current head node is null, return.
- Store a pointer to the current head node in a temporary variable.
- Update the head pointer to point to the next node.
- If the next node is not null, update its previous pointer to null.
- If the head pointer is now null, set the tail pointer to null.
- Delete the temporary node.

Deletion at the end:

- If the current tail node is null, return.
- Store a pointer to the current tail node in a temporary variable.
- Update the tail pointer to point to the previous node.
- If the previous node is not null, update its next pointer to null.
- If the tail pointer is now null, set the head pointer to null.
- Delete the temporary node.

Sample Code :

a)Singly linked list

```
#include <iostream>


using namespace std;


struct Node {
int data;
  Node* next;
};


void insert_at_beginning(Node*& head, int data) {
Node* new_node = new Node();   new_node->data
= data;   new_node->next = head;   head =
new_node;
}


void insert_at_end(Node*& head, int data) {
Node* new_node = new Node();
new_node->data = data;   new_node->next =
nullptr;


  if (head == nullptr) {
head = new_node;     return;
  }


  Node* current_node = head;   while
(current_node->next != nullptr) {
current_node = current_node->next;
  }


  current_node->next = new_node;
}

void delete_at_beginning(Node*& head) {
if (head == nullptr) {     return;
  }

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

void delete_at_end(Node*& head) {
if (head == nullptr) {     return;
  }

  Node* current_node = head;
  Node* previous_node = nullptr;

  while (current_node->next != nullptr) {
previous_node = current_node;
current_node = current_node->next;
  }

  previous_node->next = nullptr;
delete current_node;
}

void print_linked_list(Node* head) {
Node* current_node = head;   while
(current_node != nullptr) {     cout <<
current_node->data << " ";
current_node = current_node->next;
  }   cout <<
endl;
}
```

```
 int main() {   Node*
 head = nullptr;

  insert_at_beginning(head, 1);
 insert_at_end(head, 2);   insert_at_end(head,
 3);   insert_at_end(head, 4);

  cout << "Singly linked list : ";
 print_linked_list(head);

  insert_at_beginning(head, 1);
  cout << "Singly linked list after insertion at the beginning: ";   print_linked_list(head);

  insert_at_end(head, 5);   cout << "Singly linked list
 after insertion at the end: ";   print_linked_list(head);

  delete_at_beginning(head);   cout << "Singly linked list after deleting
 element at the beginning: ";   print_linked_list(head);

  delete_at_end(head);
  cout << "Singly linked list after deleting element at the end: ";   print_linked_list(head);

  cout<<"Utkarsh Joshi"<<" 21BCS9158";
  return 0;
}
```

b)Doubly linked list

```
 #include <iostream>

 using namespace std;

 struct  Node  {
 int          data;
```

```
Node*     prev;
Node* next;
};

class DoublyLinkedList {
public:
DoublyLinkedList() {
head = nullptr;         tail =
nullptr;
  }

  void insert_at_beginning(int data) {
Node* new_node = new Node();
new_node->data = data;         new_node-
>prev = nullptr;         new_node->next =
head;

    if (head == nullptr) {
tail = new_node;
    } else {             head->prev
= new_node;
    }

    head = new_node;
  }

  void insert_at_end(int data) {
Node* new_node = new Node();
new_node->data = data;
new_node->prev = tail;
new_node->next = nullptr;

    if (tail == nullptr) {
head = new_node;         } else
{            tail->next =
new_node;
    }
```

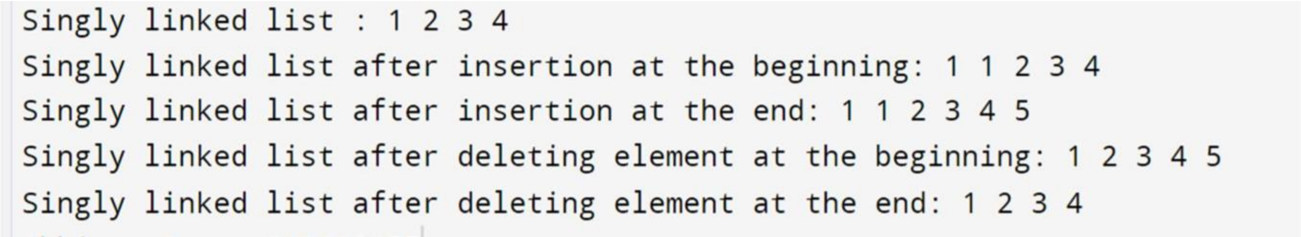
```
        tail = new_node;
    }

    void delete_at_beginning() {
if (head == nullptr) {
return;
        }

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

        if (head == nullptr) {
tail = nullptr;         } else {
head->prev = nullptr;
        }

        delete temp;
    }

    void delete_at_end() {
if (tail == nullptr) {
return;
        }

        Node* temp = tail;
tail = tail->prev;

        if (tail == nullptr) {
head = nullptr;         } else
{             tail->next =
nullptr;
        }

        delete temp;
    }
```

```
   void print_list() {         Node*
current_node = head;         while
(current_node != nullptr) {             cout <<
current_node->data << " ";
current_node = current_node->next;
        }
        cout << endl;
    }

private:
Node* head;
    Node* tail;
};

int main() {
    DoublyLinkedList doubly_linked_list;
doubly_linked_list.insert_at_beginning(3);
doubly_linked_list.insert_at_end(5);
doubly_linked_list.insert_at_end(7);
doubly_linked_list.insert_at_end(9);

    cout << "Doubly linked list:" << endl;
doubly_linked_list.print_list();

    doubly_linked_list.insert_at_beginning(1);     cout << "Doubly linked
list after insertion at the beginning:" << endl;
doubly_linked_list.print_list();

    doubly_linked_list.insert_at_end(5);     cout << "Doubly linked
list after insertion at the end:" << endl;
doubly_linked_list.print_list();

    doubly_linked_list.delete_at_beginning();     cout << "Doubly linked
list after deletion at the beginning:" << endl;
doubly_linked_list.print_list();
```

```
    doubly_linked_list.delete_at_end();     cout << "Doubly linked
  list after deletion at the end:" << endl;
  doubly_linked_list.print_list();


    cout<<"Utkarsh Joshi"<<" 21BCS9158";
  return 0;
}
```

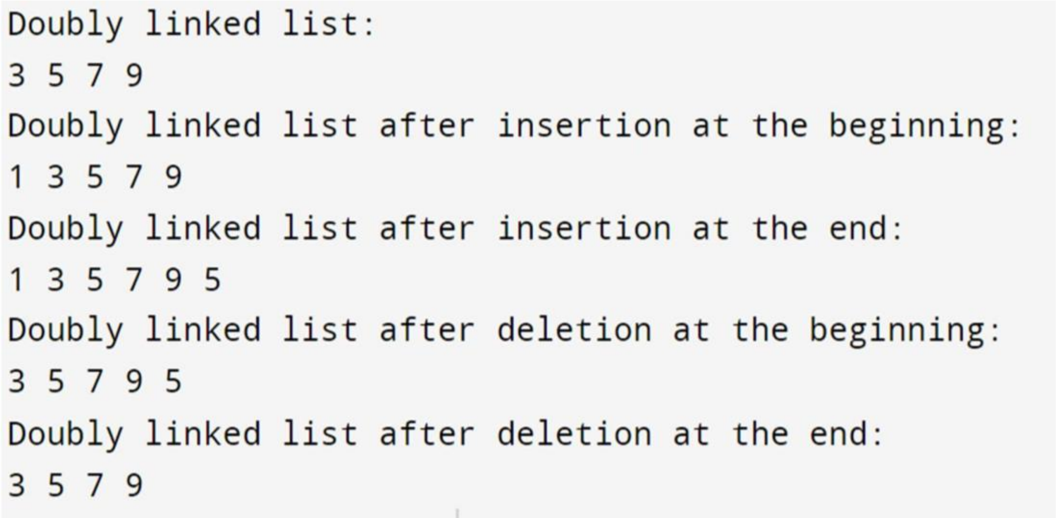
Observations/Outcome :

a)



```
Singly linked list : 1 2 3 4
Singly linked list after insertion at the beginning: 1 1 2 3 4
Singly linked list after insertion at the end: 1 1 2 3 4 5
Singly linked list after deleting element at the beginning: 1 2 3 4 5
Singly linked list after deleting element at the end: 1 2 3 4
```

b)



```
Doubly linked list:
3 5 7 9
Doubly linked list after insertion at the beginning:
1 3 5 7 9
Doubly linked list after insertion at the end:
1 3 5 7 9 5
Doubly linked list after deletion at the beginning:
3 5 7 9 5
Doubly linked list after deletion at the end:
3 5 7 9
```

Time Complexity :

a) The time complexity of the singly linked list is as follows:

- Insert at beginning: O(1)
- Insert at end: O(n)
- Delete at beginning: O(1)
- Delete at end: O(n)

b) The time complexity of the doubly linked list is as follows:

- Insert at beginning: O(1)
- Insert at end: O(1)
- Delete at beginning: O(1)
- Delete at end: O(1)