EXPERIMENT – 4

Student Name: Gaganjot Singh **UID: 22BCS14843**
Branch: BE-CSE **Section/Group:** 22BCS_JT_802-B
Semester: 5th **Date of Performance:** 8 August 2024
Subject Name: Design & Analysis of Algorithms **Subject Code:** 22CSH-311

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

2. **Objective:** To understand doubly circular linked list Input. The objective of this exercise is to implement and demonstrate the operations of insertion and deletion at both the beginning and end of a Doubly Linked List. A Doubly Linked List (DLL) is a data structure where each node contains a reference to both the next and previous node in the sequence. This allows traversal in both directions and enables efficient insertion and deletion operations.

3. **Algorithm**
   - Start
   - Inserting at end:
     - If list empty, set head, tail to new node
     - Else prev to tail and tail's next to new node
     - Update tail in new node
   - Inserting at beginning:
     - If list empty, set head, tail to new node
     - Else set next to head and head's prev to new node
     - Update head in new node
   - Delete at end:
     - If only 1 element, delete head and set head, tail to nullptr
     - Else update tail to tail's prev and delete tail's next
   - Delete at Beginning:
     - If only 1 element, delete head and set head, tail to nullptr
     - Else update head and head's next and delete head's prev
   - Printing:
     - Use head traverse till nullptr when printing fwd
     - Use tail and traverse till nullptr when printing backward
   - End

4. Implementation/Code:

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

// Define node structure
class Element
{
public:
    int data;
    Element *next;
    Element *prev;

    Element(int data)
    {
        this->data = data;
        this->next = nullptr;
        this->prev = nullptr;}};

// Doubly Linked List class
class DoublyLL
{private:
    Element *head;
    Element *tail;

public:
    DoublyLL()
    {head = nullptr;
     tail = nullptr;}

    // Insert at the end of the list
    void insertAtEnd(int data)
    {Element *newNode = new Element(data);
        if (head == nullptr)
        {head = newNode;
         tail = newNode;}
        else
        {newNode->prev = tail;
          tail->next = newNode;
          tail = newNode;}}

    // Insert at the beginning of the list
    void insertAtBeginning(int data)
    {Element *newNode = new Element(data);

        if (head == nullptr)
        {head = newNode;
            tail = newNode;}
        else
```

```
    {  newNode->next = head;
       head->prev = newNode;
       head = newNode;}}

// Delete from the end of the list
void deleteAtEnd()
{if (head == nullptr)
   {cout << "List is empty. Cannot delete." << endl;
    return;}

   if (head == tail)
   {delete head;
    head = nullptr;
    tail = nullptr;}
   else
   {Element *temp = tail;
    tail = tail->prev;
    tail->next = nullptr;
    delete temp;}}

// Delete from the beginning of the list
void deleteAtBeginning()
{if (head == nullptr)
   {cout << "List is empty. Cannot delete." << endl; return;}

   if (head == tail)
   {delete head;
    head = nullptr;
    tail = nullptr;}
   else
   {  Element *temp = head;
      head = head->next;
      head->prev = nullptr;
      delete temp;}}

// Print the list in forward direction
void printForward()
{Element *current = head;
   while (current != nullptr)
   {cout << current->data << " ";
    current = current->next;}
   cout << endl;
}

// Print the list in backward direction
void printBackward()
{Element *current = tail;
 while (current != nullptr)
   {cout << current->data << " ";
```

```
        current = current->prev;
      }
      cout << endl;}};

int main()
{
   DoublyLL list;

   list.insertAtEnd(1);
   list.insertAtEnd(2);
   list.insertAtEnd(3);
   list.insertAtEnd(4);

   cout << "*** Native List ***" << endl;
   cout << "Forward" << endl;
   list.printForward();
   cout << "Backward" << endl;
   list.printBackward();
   cout << " " << endl;

   list.insertAtEnd(5);
   cout << "*** List after inserting 5 at end ***" << endl;
   cout << "Forward" << endl;
   list.printForward();
   cout << "Backward" << endl;
   list.printBackward();
   cout << " " << endl;

   list.insertAtBeginning(6);
   cout << "*** List after inserting 6 at beginning ***" << endl;
   cout << "Forward" << endl;
   list.printForward();
   cout << "Backward" << endl;
   list.printBackward();
   cout << " " << endl;

   list.deleteAtBeginning();
   cout << "*** List after deleting 6 at beginning ***" << endl;
   cout << "Forward" << endl;
   list.printForward();
   cout << "Backward" << endl;
   list.printBackward();
   cout << " " << endl;

   list.deleteAtEnd();
   cout << "*** List after deleting 5 at end ***" << endl;
   cout << "Forward" << endl;
   list.printForward();
   cout << "Backward" << endl;
```

```
list.printBackward();
cout << " " << endl;
return 0;}
```

5. Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS E:\CU Study\22CSH 311 DAA\codes> cd "e:\CU Study\22CSH 311 DAA\codes\" ; if ($?) { g++ ex4_doubly.cpp -o ex4_doubly
*** Native List ***
Forward
1 2 3 4
Backward
4 3 2 1

*** List after inserting 5 at end ***
Forward
1 2 3 4 5
Backward
5 4 3 2 1

*** List after inserting 6 at beginning ***
Forward
6 1 2 3 4 5
Backward
5 4 3 2 1 6

*** List after deleting 6 at beginning ***
Forward
1 2 3 4 5
Backward
5 4 3 2 1

*** List after deleting 5 at end ***
Forward
1 2 3 4
Backward
4 3 2 1
```

6. Time Complexity :

The insertAtEnd and deleteAtEnd functions have a time complexity of O(1) for inserting and deleting elements at the end of the doubly linked list. The printForward and printBackward functions have a time complexity of O(n) as they iterate through all elements in the list. Therefore, the overall time complexity of the operations in this code snippet is O(n).

7. Learning Outcomes:

- Understood doubly linked list as a data structure in which each element points to both the next and previous elements.
- The class Element has data, next, and prev members to represent a node.
- insertAtEnd method adds a new element to the end of the list, updates the tail pointer.
- insertAtBeginning method adds a new element to the beginning of the list, updates the head pointer.
- deleteAtEnd method to removes last element from the list, updates the tail pointer.
- deleteAtBeginning method removes the first element from the list, updates the head pointer.
- printForward & printBackward method to traverse the list and print it.
- Learned the usage of the DoublyLL class by creating a list, inserting elements, printing the list, deleting elements, and printing the list again to show the changes.