SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering
Computer Engineering Department
Program: B.Tech Integrated. Sem V

Course: Basic Data Structures
LAB Manual

(PART A : TO BE REFFERED BY STUDENTS)

Experiment No.07

TASK 1:

Write a C/C++ program to implement the following operations on a Singly linked list

i) Create a linked list (Head node)
ii) The new node insertion at the beginning.
iii) The new node insertion at the end.
iv) The new node insertion after a given node.
v) The new node insertion before a given node.
vi) The first node deletion.
vii) The last node deletion.
viii) The node after a given node deletion.
ix) Traverse the singly linked list.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)

Roll No. C003	Name: Anuradha bansal
Class : **B**	Batch : B1

Date of Experiment: 21-10-24	Date of Submission 21-20-24
Grade :	Time of Submission:
Date of Grading:	

B.1 Software Code written by student: (Task 1)

Task1:

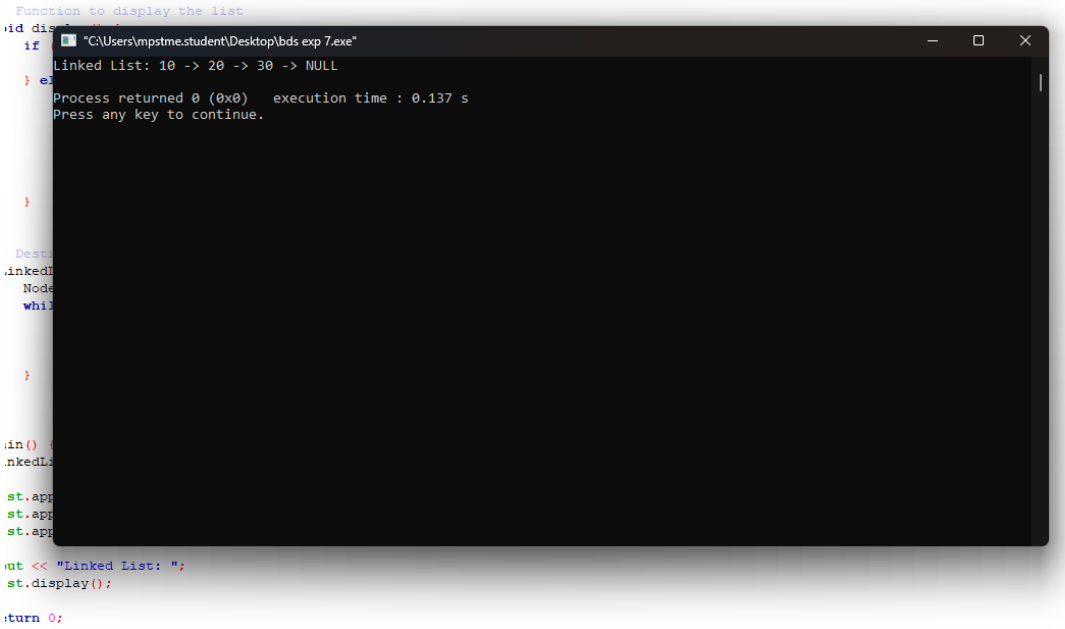
B.2 Input and Output: (Task 1)

(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)

Task1:

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

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

// Linked List class
class LinkedList {
private:
   Node* head;

public:
   LinkedList() {
      head = NULL;
   }

   // Function to add a new node at the end
   void append(int value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = NULL;

      if (head == NULL) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != NULL) {
```

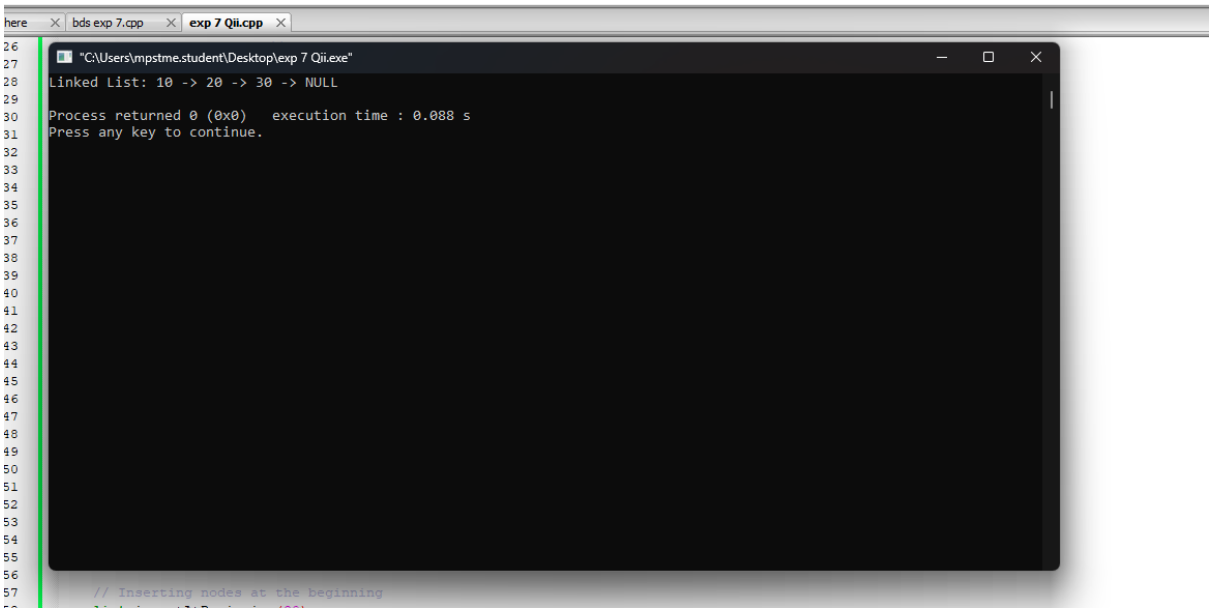
```
            temp = temp->next;
        }
        temp->next = newNode;
    }
  }

  // Function to display the list
  void display() {
    if (head == NULL) {
        cout << "List is empty." << endl;
    } else {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
  }

  // Destructor to free the memory
  ~LinkedList() {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        delete temp;
    }
  }
};

int main() {
  LinkedList list;

  list.append(10);
  list.append(20);
  list.append(30);

  cout << "Linked List: ";
  list.display();

  return 0;
}
```



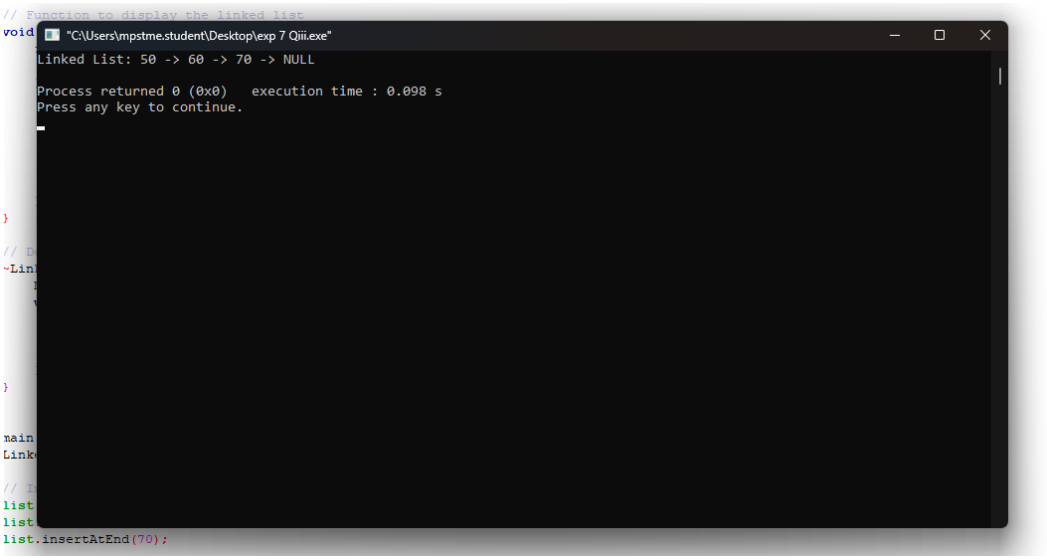
ii)

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

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

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Function to insert a node at the beginning
    void insertAtBeginning(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;  // Point the new node to the current head
        head = newNode;  // Update the head to the new node
    }
```

```
    // Function to display the linked list
    void display() {
        if (head == NULL) {
            cout << "List is empty." << endl;
        } else {
            Node* temp = head;
            while (temp != NULL) {
                cout << temp->data << " -> ";
                temp = temp->next;
            }
            cout << "NULL" << endl;
        }
    }

    // Destructor to free the memory
    ~LinkedList() {
        Node* temp;
        while (head != NULL) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;

    // Inserting nodes at the beginning
    list.insertAtBeginning(30);
    list.insertAtBeginning(20);
    list.insertAtBeginning(10);

    // Display the linked list
    cout << "Linked List: ";
    list.display();

    return 0;
}
```



iii)

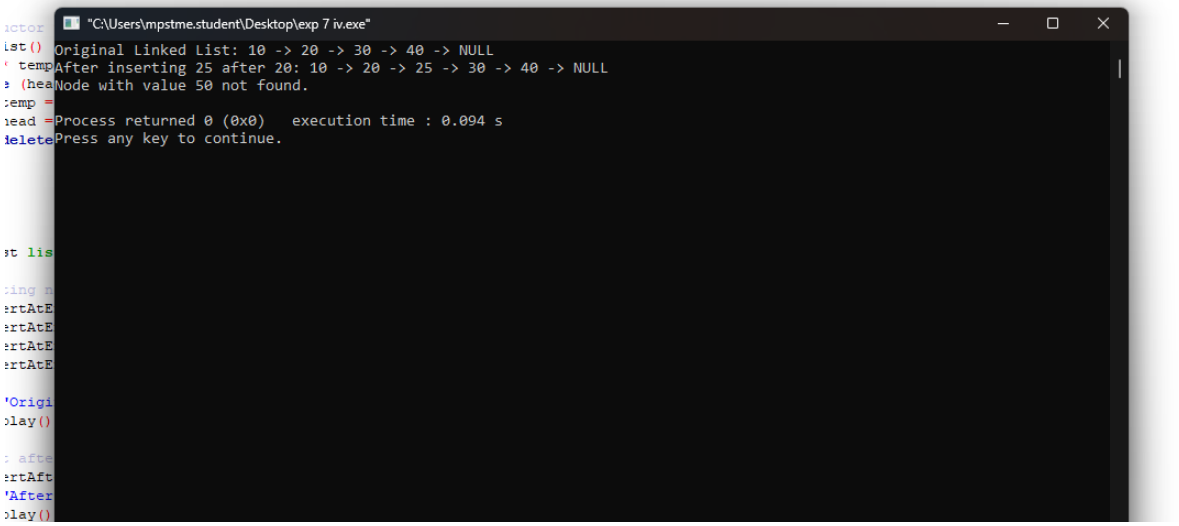



```
#include <iostream>
using namespace std;

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

// Linked List class
class LinkedList {
private:
    Node* head;

public:
```

```
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Function to insert a node at the end
    void insertAtEnd(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;  // The new node will point to nullptr since it will be the last node

        if (head == NULL) {  // If the list is empty, make the new node the head
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL) {  // Traverse to the last node
                temp = temp->next;
            }
            temp->next = newNode;  // Link the last node to the new node
        }
    }

    // Function to display the linked list
    void display() {
        if (head == NULL) {
            cout << "List is empty." << endl;
        } else {
            Node* temp = head;
            while (temp != NULL) {
                cout << temp->data << " -> ";
                temp = temp->next;
            }
            cout << "NULL" << endl;
        }
    }

    // Destructor to free the memory
    ~LinkedList() {
        Node* temp;
        while (head != NULL) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
```

```
    list.insertAtEnd(50);
    list.insertAtEnd(60);
    list.insertAtEnd(70);

    // Display the linked list
    cout << "Linked List: ";
    list.display();

    return 0;
}
```

iv)

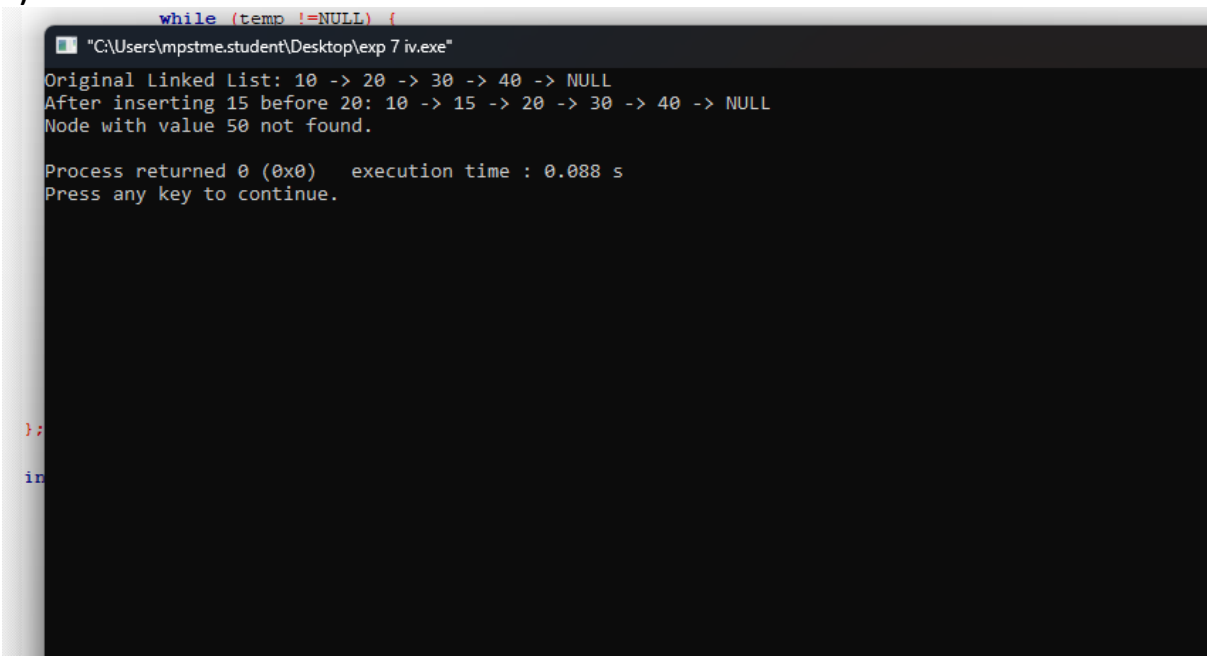



```
#include <iostream>
using namespace std;

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

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }
```

```
// Function to insert a node at the end (to populate the list for testing)
void insertAtEnd(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   } else {
      Node* temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
}

// Function to insert a new node after a given node
void insertAfterNode(int key, int value) {
   Node* temp = head;

   // Search for the node with the given key
   while (temp != NULL && temp->data != key) {
      temp = temp->next;
   }

   // If the node is found
   if (temp != NULL) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = temp->next;  // Point the new node to the next node of the given node
      temp->next = newNode;  // Link the given node to the new node
   } else {
      cout << "Node with value " << key << " not found." << endl;
   }
}

// Function to display the linked list
void display() {
   if (head == NULL) {
      cout << "List is empty." << endl;
   } else {
      Node* temp = head;
      while (temp != NULL) {
         cout << temp->data << " -> ";
         temp = temp->next;
      }
      cout << "NULL" << endl;
   }
}
```

```
    // Destructor to free the memory
    ~LinkedList() {
        Node* temp;
        while (head != NULL) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);

    cout << "Original Linked List: ";
    list.display();

    // Insert after node with value 20
    list.insertAfterNode(20, 25);
    cout << "After inserting 25 after 20: ";
    list.display();

    // Try inserting after a node that doesn't exist
    list.insertAfterNode(50, 60);

    return 0;
}
```

V)

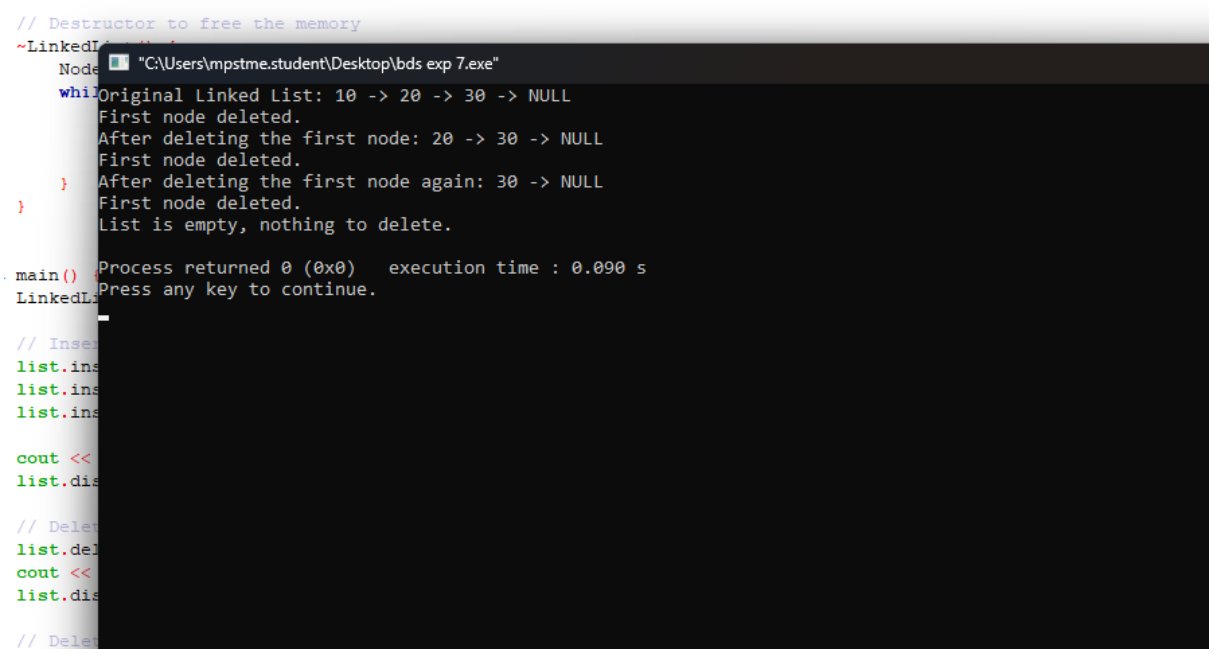



```
#include <iostream>
using namespace std;

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

// Linked List class
class LinkedList {
private:
   Node* head;

public:
   // Constructor
   LinkedList() {
      head = NULL;
   }

   // Function to insert a node at the end (to populate the list for testing)
   void insertAtEnd(int value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = NULL;

      if (head == NULL) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != NULL) {
```

```
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to insert a new node before a given node
void insertBeforeNode(int key, int value) {
    // If the list is empty
    if (head == NULL) {
        cout << "List is empty." << endl;
        return;
    }

    // If the head node is the target node
    if (head->data == key) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;  // Update the head to the new node
        return;
    }

    // Find the node before the target node
    Node* temp = head;
    while (temp->next != NULL && temp->next->data != key) {
        temp = temp->next;
    }

    // If the target node is found
    if (temp->next != NULL) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = temp->next;  // Point the new node to the target node
        temp->next = newNode;  // Link the previous node to the new node
    } else {
        cout << "Node with value " << key << " not found." << endl;
    }
}

// Function to display the linked list
void display() {
    if (head == NULL) {
        cout << "List is empty." << endl;
    } else {
        Node* temp = head;
        while (temp !=NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
```

```
        }
    }

    // Destructor to free the memory
    ~LinkedList() {
        Node* temp;
        while (head != NULL) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);

    cout << "Original Linked List: ";
    list.display();

    // Insert before node with value 20
    list.insertBeforeNode(20, 15);
    cout << "After inserting 15 before 20: ";
    list.display();

    // Try inserting before a node that doesn't exist
    list.insertBeforeNode(50, 45);

    return 0;
}
```

vi)

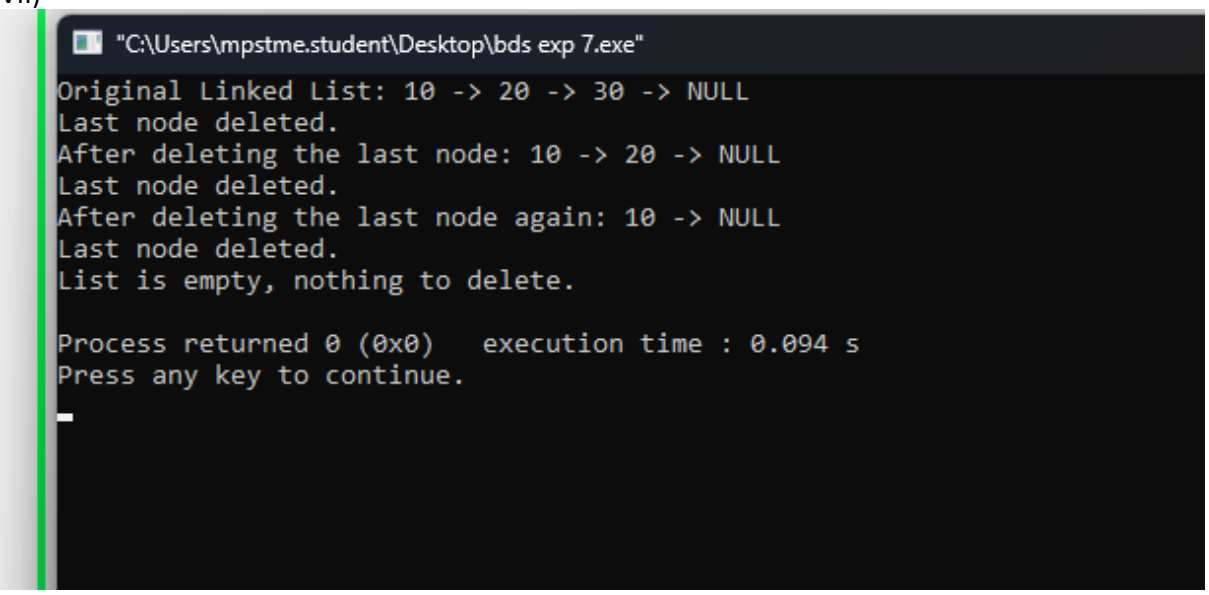



```
#include <iostream>
using namespace std;

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

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Function to insert a node at the end (to populate the list for testing)
    void insertAtEnd(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
```

```
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to delete the first node
void deleteFirstNode() {
    if (head == NULL) {
        cout << "List is empty, nothing to delete." << endl;
        return;
    }

    Node* temp = head;  // Temporary pointer to hold the head node
    head = head->next;  // Move the head to the next node
    delete temp;  // Delete the original head node
    cout << "First node deleted." << endl;
}

// Function to display the linked list
void display() {
    if (head == NULL) {
        cout << "List is empty." << endl;
    } else {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
}

// Destructor to free the memory
~LinkedList() {
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        delete temp;
    }
}
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
    list.insertAtEnd(10);
```

```
    list.insertAtEnd(20);
    list.insertAtEnd(30);

    cout << "Original Linked List: ";
    list.display();

    // Delete the first node
    list.deleteFirstNode();
    cout << "After deleting the first node: ";
    list.display();

    // Delete again to test
    list.deleteFirstNode();
    cout << "After deleting the first node again: ";
    list.display();

    // Try deleting from an empty list
    list.deleteFirstNode();
    list.deleteFirstNode();  // Additional deletion to show list is empty

    return 0;
}
```

VII)



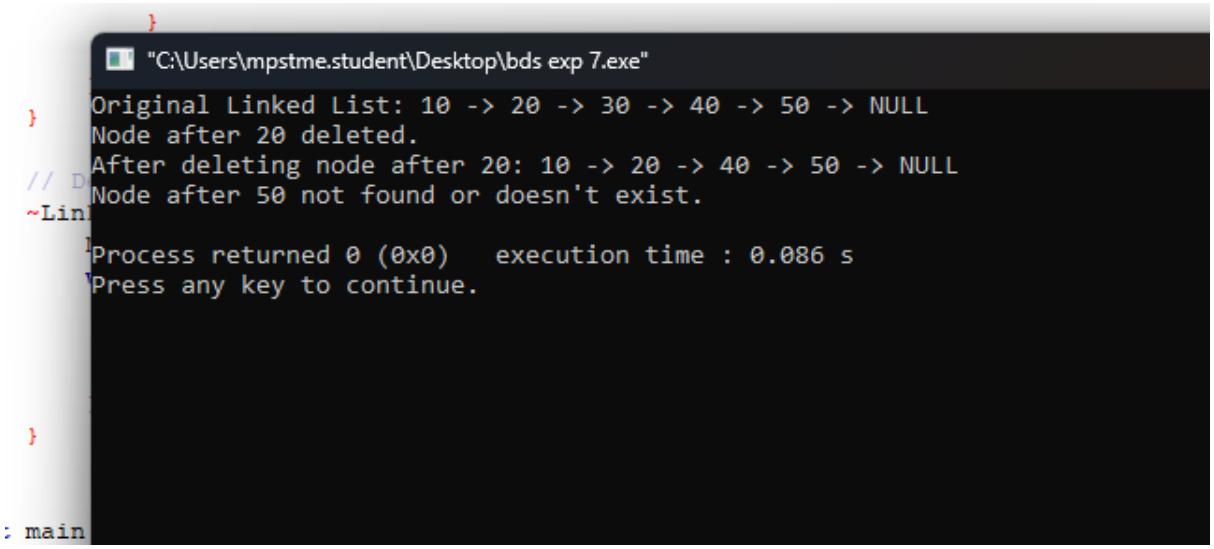

```
#include <iostream>
using namespace std;

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

// Linked List class
```

```
class LinkedList {
private:
   Node* head;

public:
   // Constructor
   LinkedList() {
      head = NULL;
   }

   // Function to insert a node at the end (to populate the list for testing)
   void insertAtEnd(int value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = NULL;

      if (head == NULL) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != NULL) {
            temp = temp->next;
         }
         temp->next = newNode;
      }
   }

   // Function to delete the last node
   void deleteLastNode() {
      if (head == NULL) {
         cout << "List is empty, nothing to delete." << endl;
         return;
      }

      // If the list has only one node
      if (head->next == NULL) {
         delete head;
         head = NULL;
         cout << "Last node deleted." << endl;
         return;
      }

      // Traverse to the second last node
      Node* temp = head;
      while (temp->next->next != NULL) {
         temp = temp->next;
      }

      // Delete the last node
      delete temp->next;
      temp->next = NULL;
```

```
        cout << "Last node deleted." << endl;
    }

    // Function to display the linked list
    void display() {
        if (head == NULL) {
            cout << "List is empty." << endl;
        } else {
            Node* temp = head;
            while (temp != NULL) {
                cout << temp->data << " -> ";
                temp = temp->next;
            }
            cout << "NULL" << endl;
        }
    }

    // Destructor to free the memory
    ~LinkedList() {
        Node* temp;
        while (head != NULL) {
            temp = head;
            head = head->next;
            delete temp;
        }
    }
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);

    cout << "Original Linked List: ";
    list.display();

    // Delete the last node
    list.deleteLastNode();
    cout << "After deleting the last node: ";
    list.display();

    // Delete again to test
    list.deleteLastNode();
    cout << "After deleting the last node again: ";
    list.display();

    // Try deleting from an empty list
    list.deleteLastNode();
```

```
    list.deleteLastNode();  // Additional deletion to show list is empty

    return 0;
}
```

VIII)

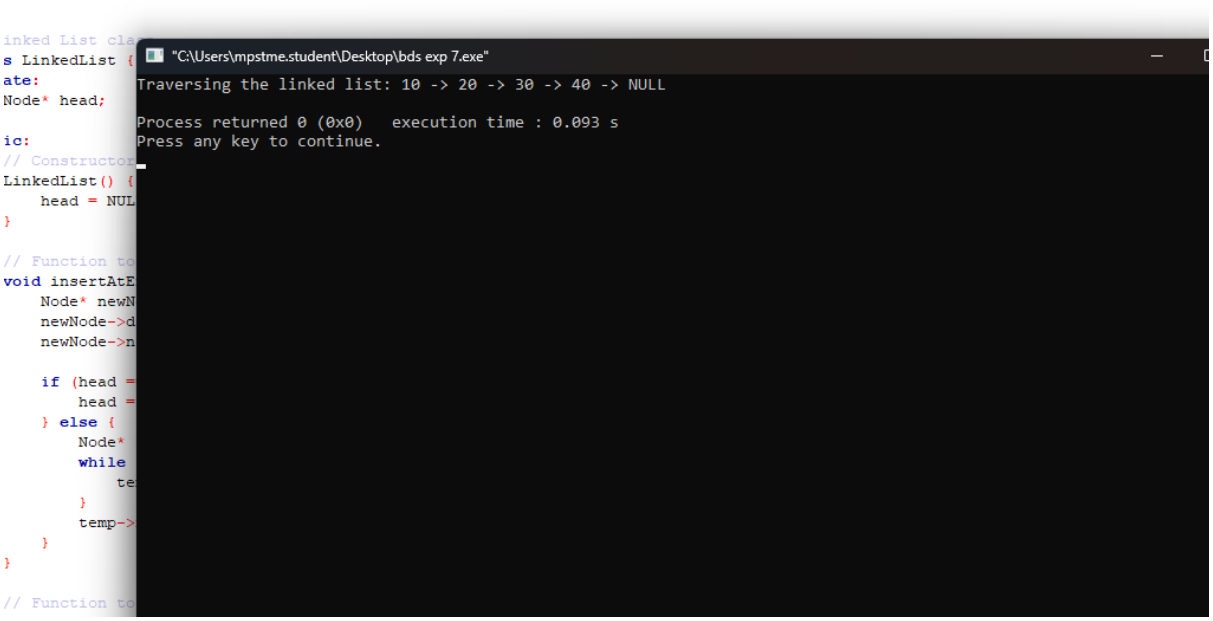



```
#include <iostream>
using namespace std;

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

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = NULL;
    }

    // Function to insert a node at the end (to populate the list for testing)
    void insertAtEnd(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next =  NULL;

        if (head ==  NULL) {
            head = newNode;
        } else {
```

```
        Node* temp = head;
        while (temp->next !=  NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to delete the node after a given node
void deleteNodeAfter(int key) {
    Node* temp = head;

    // Traverse the list to find the node with the given key
    while (temp !=  NULL && temp->data != key) {
        temp = temp->next;
    }

    // If the node is found and the next node is not null
    if (temp !=  NULL && temp->next !=  NULL) {
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;  // Bypass the node to be deleted
        delete nodeToDelete;  // Free the memory of the node to be deleted
        cout << "Node after " << key << " deleted." << endl;
    } else {
        cout << "Node after " << key << " not found or doesn't exist." << endl;
    }
}

// Function to display the linked list
void display() {
    if (head ==  NULL) {
        cout << "List is empty." << endl;
    } else {
        Node* temp = head;
        while (temp !=  NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL" << endl;
    }
}

// Destructor to free the memory
~LinkedList() {
    Node* temp;
    while (head !=  NULL) {
        temp = head;
        head = head->next;
        delete temp;
    }
}
```

```
};

int main() {
    LinkedList list;

    // Inserting nodes at the end
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);
    list.insertAtEnd(50);

    cout << "Original Linked List: ";
    list.display();

    // Delete the node after the node with value 20
    list.deleteNodeAfter(20);
    cout << "After deleting node after 20: ";
    list.display();

    // Try deleting after a node that doesn't exist or doesn't have a next node
    list.deleteNodeAfter(50);

    return 0;
}
```

ix)




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

```
// Node structure
struct Node {
   int data;
   Node* next;
};

// Linked List class
class LinkedList {
private:
   Node* head;

public:
   // Constructor
   LinkedList() {
      head = NULL;
   }

   // Function to insert a node at the end
   void insertAtEnd(int value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = NULL;

      if (head == NULL) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != NULL) {
            temp = temp->next;
         }
         temp->next = newNode;
      }
   }

   // Function to traverse and print the linked list
   void traverse() {
      if (head == NULL) {
         cout << "List is empty." << endl;
      } else {
         Node* temp = head;
         while (temp != NULL) {
            cout << temp->data << " -> ";
            temp = temp->next;
         }
         cout << "NULL" << endl;
      }
   }

   // Destructor to free the memory
   ~LinkedList() {
      Node* temp;
```

```
    while (head != NULL) {
      temp = head;
      head = head->next;
      delete temp;
    }
  }
};

int main() {
  LinkedList list;

  // Inserting nodes at the end
  list.insertAtEnd(10);
  list.insertAtEnd(20);
  list.insertAtEnd(30);
  list.insertAtEnd(40);

  // Traverse and display the linked list
  cout << "Traversing the linked list: ";
  list.traverse();

  return 0;
}
```

B.3 Observations and learning [w.r.t. all tasks]:

(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)

WE OBSERVED THAT WE CAN MANIPULATE NODES OF LINKED LISTUSING DIFFERENT CODES AND WE LEARNED EVERYTHING BASIC ABOUT SINGLE LINKED LISTS TO APPLY IN REAL LIFE APPLICATION.

B.4 Conclusion:

We concluded that the last node in the linked list always has to be NULL. We concluded that we can use different techniques to control the order of nodes in lsingle linked list.

(Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)

B.5 Question of Curiosity

(To be answered by student based on the practical performed and learning/observations)

State the advantages of linked list

- **Flexible memory allocation**: Linked lists can grow and shrink dynamically during runtime. Unlike arrays, which require predefined sizes, linked lists allocate memory as needed, avoiding wasted space and the need for resizing.
- **No need for shifting**: Inserting or deleting elements in a linked list is generally more efficient than in arrays because there's no need to shift elements. You simply adjust the pointers between nodes, making it easier to add or remove elements at the beginning or middle.

- **Constant time complexity**: Insertions and deletions at the beginning of a linked list have a time complexity of O(1)O(1)O(1), which is much faster compared to an array's O(n)O(n)O(n) where elements need to be shifted.

- **Efficient use of memory**: Since linked lists only allocate memory for elements that are actually used (one node at a time), memory allocation is more efficient, especially in situations where the exact number of elements isn't known in advance.
- **No need for contiguous memory**: Linked lists do not require contiguous memory locations, making them more flexible in scenarios with limited or fragmented memory**.**
- **Easily expand or shrink**: Linked lists don't have a fixed size like arrays. You can add or remove nodes without worrying about exceeding a pre-set array capacity or reallocating memory.