# Lab Part

## QL1.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}

struct Node* insert(struct Node* root, int data) {
   if (root == NULL)
      return createNode(data);
```

```
  if (data < root->data)
    root->left = insert(root->left, data);
  else if (data > root->data)
    root->right = insert(root->right, data);

  return root;
}

void inorder(struct Node* root) {
  if (root != NULL) {
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
  }
}

struct Node* search(struct Node* root, int key) {
  if (root == NULL || root->data == key)
    return root;

  if (key < root->data)
    return search(root->left, key);

  return search(root->right, key);
}
```

```
struct Node* findMin(struct Node* root) {
  while (root->left != NULL)
    root = root->left;
  return root;
}

struct Node* deleteNode(struct Node* root, int key) {
  if (root == NULL) return root;

  if (key < root->data)
    root->left = deleteNode(root->left, key);
  else if (key > root->data)
    root->right = deleteNode(root->right, key);
  else {
   if (root->left == NULL) {
      struct Node* temp = root->right;
      free(root);
      return temp;
    }
    else if (root->right == NULL) {
      struct Node* temp = root->left;
      free(root);
      return temp;
    }
```

```
        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

int main() {
    struct Node* root = NULL;

    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 60);
    root = insert(root, 80);

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

    int key = 40;
    root = deleteNode(root, key);
    printf("Inorder after deleting %d: ", key);
    inorder(root);
```

```
    printf("\n");

    int searchKey = 60;
    struct Node* found = search(root, searchKey);
    if (found)
        printf("Found %d in the BST.\n", searchKey);
    else
        printf("%d not found in the BST.\n", searchKey);

    return 0;
}
```

## QL2.

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

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

// Function to create a new node
Node* createNode(float value) {
```

```
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Function to delete a node with value XX
void deleteNode(Node*& head, float value) {
    if (head == nullptr) return;

    Node* temp = head;
    Node* prev = nullptr;

    // If head needs to be deleted
    if (temp->data == value) {
        head = temp->next;
        delete temp;
        return;
    }

    // Search for the node to delete
    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }
```

```
    // If node not found
    if (temp == nullptr) return;

    // Unlink the node
    prev->next = temp->next;
    delete temp;
}

// Function to display the list in reverse using stack
void displayReverse(Node* head) {
    stack<float> s;
    Node* temp = head;

    while (temp != nullptr) {
        s.push(temp->data);
        temp = temp->next;
    }

    cout << "Linked List in Reverse Order: ";
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;
}
```

```
// Function to display the list normally
void displayList(Node* head) {
   cout << "Linked List: ";
   while (head != nullptr) {
      cout << head->data << " ";
      head = head->next;
   }
   cout << endl;
}

int main() {
   // Creating linked list manually from image
   Node* head = createNode(10.5);
   head->next = createNode(7.21);    // XX = SGPA + 3.21
   head->next->next = createNode(15.7);
   head->next->next->next = createNode(34.3);
   cout << "Before Deletion:" << endl;
   displayList(head);

   // QA: Delete node with value 7.21
   deleteNode(head, 7.21);

  cout << "After Deletion of 7.21:" << endl;sss
   displayList(head);
   // QB: Display reverse
   displayReverse(head);
```

```
    return 0;
}
```