DATA STRUCTURE LAB
FINAL ASSINGMENT

Doubly Link List

1) Write a program to delete the first node in a doubly linked list.

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

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

void deleteFirstNode(Node*& head) {
   if (!head) return;

   Node* temp = head;
   head = head->next;
   if (head) head->prev = nullptr;

   delete temp;
}

void printList(Node* head) {
   while (head) {
      cout << head->data << " ";
      head = head->next;
   }
   cout << endl;
}

int main() {
   Node* head = new Node{1, nullptr, new Node{2, nullptr, new Node{3,
nullptr, nullptr}}};
   head->next->prev = head;
   head->next->next->prev = head->next;

   cout << "Original List: ";
   printList(head);
```

```
    deleteFirstNode(head);

    cout << "After Deleting First Node: ";
    printList(head);

    return 0;
}
```

2) **How can you delete the last node in a doubly linked list? Write the code.**

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

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

void deleteLastNode(Node*& head) {
    if (!head) return;

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

    Node* temp = head;
    while (temp->next) temp = temp->next;

    temp->prev->next = nullptr;
    delete temp;
}

void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{1, nullptr, new Node{2, nullptr, new Node{3,
nullptr, nullptr}}};
    head->next->prev = head;
    head->next->next->prev = head->next;

    cout << "Original List: ";
    printList(head);

    deleteLastNode(head);

    cout << "After Deleting Last Node: ";
    printList(head);

    return 0;
}
```

3) Write code to delete a node by its value in a doubly linked list.

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

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

void deleteNodeByValue(Node*& head, int value) {
    if (!head) return;

    Node* temp = head;
    while (temp && temp->data != value) temp = temp->next;

    if (!temp) return; // Value not found

    if (temp->prev) temp->prev->next = temp->next;
    if (temp->next) temp->next->prev = temp->prev;

    if (temp == head) head = temp->next;
    delete temp;
}

void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{1, nullptr, new Node{2, nullptr, new Node{3,
nullptr, nullptr}}};
    head->next->prev = head;
    head->next->next->prev = head->next;

    cout << "Original List: ";
    printList(head);

    deleteNodeByValue(head, 2);

    cout << "After Deleting Node with Value 2: ";
    printList(head);

    return 0;
}
```

4) How would you delete a node at a specific position in a doubly linked list?

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

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

void deleteNodeAtPosition(Node*& head, int position) {
    if (!head || position < 1) return;
    Node* temp = head;
    for (int i = 1; temp && i < position; ++i)
        temp = temp->next;

    if (!temp) return; // Position out of range

    if (temp->prev) temp->prev->next = temp->next;
    if (temp->next) temp->next->prev = temp->prev;

    if (temp == head) head = temp->next;

    delete temp;
}

void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{1, nullptr, new Node{2, nullptr, new Node{3,
nullptr, nullptr}}};
    head->next->prev = head;
    head->next->next->prev = head->next;

    cout << "Original List: ";
    printList(head);

    deleteNodeAtPosition(head, 2);

    cout << "After Deleting Node at Position 2: ";
    printList(head);

    return 0;
}
```

5) After deleting a node, how will you write the forward and reverse traversal functions?

```
#include <iostream>
```

```
using namespace std;

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

void forwardTraversal(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

void reverseTraversal(Node* tail) {
    while (tail) {
        cout << tail->data << " ";
        tail = tail->prev;
    }
    cout << endl;
}

Node* getTail(Node* head) {
    while (head && head->next) head = head->next;
    return head;
}

int main() {
    Node* head = new Node{1, nullptr, new Node{2, nullptr, new Node{3,
nullptr, nullptr}}};
    head->next->prev = head;
    head->next->next->prev = head->next;

    cout << "Forward Traversal: ";
    forwardTraversal(head);

    Node* tail = getTail(head);
    cout << "Reverse Traversal: ";
    reverseTraversal(tail);

    return 0;
}
```

Circular Linked List

1) Write a program to delete the first node in a circular linked list

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

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

void deleteFirstNode(Node*& head) {
   if (!head) return;

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

   Node* temp = head;
   Node* last = head;

   while (last->next != head) last = last->next;

   head = head->next;
   last->next = head;

   delete temp;
}

void printList(Node* head) {
   if (!head) return;

   Node* temp = head;
   do {
      cout << temp->data << " ";
      temp = temp->next;
   } while (temp != head);

   cout << endl;
}

int main() {
   Node* head = new Node{1, new Node{2, new Node{3, nullptr}}};
   head->next->next->next = head;

   cout << "Original List: ";
   printList(head);

   deleteFirstNode(head);

   cout << "After Deleting First Node: ";
   printList(head);

   return 0;
}
```

2) **How can you delete the last node in a circular linked list?**

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

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

void deleteLastNode(Node*& head) {
   if (!head) return;

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

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

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

```
    prev->next = head;

    delete temp;
}

void printList(Node* head) {
    if (!head) return;

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);

    cout << endl;
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, nullptr}}};
    head->next->next->next = head;

    cout << "Original List: ";
    printList(head);

    deleteLastNode(head);

    cout << "After Deleting Last Node: ";
    printList(head);

    return 0;
}
```

3) Write a function to delete a node by its value in a circular linked list

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

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

```
void deleteNodeByValue(Node*& head, int value) {
    if (!head) return;

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

    // Handle the case if the head node contains the value
    if (head->data == value) {
        if (head->next == head) {
            delete head;
            head = nullptr;
            return;
        }

        while (temp->next != head) temp = temp->next;

        temp->next = head->next;
        delete head;
        head = temp->next;
        return;
    }

    // Traverse to find the value
    do {
        prev = temp;
        temp = temp->next;
    } while (temp != head && temp->data != value);

    if (temp == head) return; // Value not found

    prev->next = temp->next;
    delete temp;
}

void printList(Node* head) {
    if (!head) return;

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, nullptr}}};
    head->next->next->next = head;

    cout << "Original List: ";
    printList(head);

    deleteNodeByValue(head, 2);

    cout << "After Deleting Node with Value 2: ";
    printList(head);

    return 0;
}
```

4) How will you delete a node at a specific position in a circular linked list?

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

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

void deleteNodeAtPosition(Node*& head, int position) {
    if (!head || position < 1) return;

    Node* temp = head;

    // Handle deleting the first node
    if (position == 1) {
        if (head->next == head) {
            delete head;
            head = nullptr;
            return;
        }

        Node* last = head;
        while (last->next != head) last = last->next;
        last->next = head->next;
        delete head;
        head = last->next;
        return;
    }

    // Traverse to the node at the given position
    Node* prev = nullptr;
    for (int i = 1; temp->next != head && i < position; ++i) {
        prev = temp;
        temp = temp->next;
    }

    if (temp->next == head && position > 1) return; // Position out of
range

    prev->next = temp->next;
    delete temp;
}

void printList(Node* head) {
    if (!head) return;

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);

    cout << endl;
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, nullptr}}};
    head->next->next->next = head;

    cout << "Original List: ";
    printList(head);

    deleteNodeAtPosition(head, 2);

    cout << "After Deleting Node at Position 2: ";
    printList(head);
    return 0;
}
```

5) **Write a program to show forward traversal after deleting a node in a circular linked list.**

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

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

void forwardTraversal(Node* head) {
    if (!head) return;

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);

    cout << endl;
}

int main() {
    Node* head = new Node{1, new Node{2, new Node{3, nullptr}}};
    head->next->next->next = head;

    cout << "Original List: ";
    forwardTraversal(head);

    // Deleting a node (for example, the second node)
    Node* temp = head->next;
    head->next = temp->next;
    delete temp;

    cout << "Forward Traversal After Deleting a Node: ";
    forwardTraversal(head);

    return 0;
}
```

Binary Search Tree

1) Write a program to count all the nodes in a binary search tree.

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

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

   Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

int countNodes(Node* root) {
   if (!root) return 0;
   return 1 + countNodes(root->left) + countNodes(root->right);
}

int main() {
   Node* root = new Node(10);
   root->left = new Node(5);
   root->right = new Node(15);
   root->left->left = new Node(3);
   root->left->right = new Node(7);

   cout << "Total Nodes in BST: " << countNodes(root) << endl;

   return 0;
}
```

2) How can you search for a specific value in a binary search tree?

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

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

   Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

bool searchBST(Node* root, int value) {
   if (!root) return false;
   if (root->data == value) return true;
   if (value < root->data) return searchBST(root->left, value);
   return searchBST(root->right, value);
}

int main() {
   Node* root = new Node(10);
   root->left = new Node(5);
   root->right = new Node(15);
   root->left->left = new Node(3);
   root->left->right = new Node(7);

   int value = 7;
   if (searchBST(root, value)) {
       cout << "Value " << value << " found in BST." << endl;
   } else {
       cout << "Value " << value << " not found in BST." << endl;
   }

   return 0;
}
```

3) Write code to traverse a binary search tree in in-order, pre-order, and postorder.

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

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

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

void inOrder(Node* root) {
    if (!root) return;
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

void preOrder(Node* root) {
    if (!root) return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}

void postOrder(Node* root) {
    if (!root) return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(3);
    root->left->right = new Node(7);

    cout << "In-order: ";
    inOrder(root);
    cout << "\nPre-order: ";
    preOrder(root);
    cout << "\nPost-order: ";
    postOrder(root);

    return 0;
}
```

4) How will you write reverse in-order traversal for a binary search tree?

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

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

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

void reverseInOrder(Node* root) {
    if (!root) return;
    reverseInOrder(root->right);
    cout << root->data << " ";
    reverseInOrder(root->left);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(3);
    root->left->right = new Node(7);

    cout << "Reverse In-order: ";
    reverseInOrder(root);

    return 0;
}
```

5) Write a program to check if there are duplicate values in a binary search tree.

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

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

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

bool checkDuplicates(Node* root, unordered_set<int>& seen) {
    if (!root) return false;
    if (seen.count(root->data)) return true;
    seen.insert(root->data);
    return checkDuplicates(root->left, seen) || checkDuplicates(root->right,
seen);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(3);
    root->left->right = new Node(7);
    root->right->left = new Node(7); // Duplicate value

    unordered_set<int> seen;
    if (checkDuplicates(root, seen)) {
        cout << "Duplicates found in BST." << endl;
    } else {
        cout << "No duplicates in BST." << endl;
    }

    return 0;
}
```

6) How can you delete a node from a binary search tree? Write code for deleting a leaf, a node with one child, and a node with two children.

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

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

    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

Node* findMin(Node* root) {
    while (root->left) root = root->left;
    return root;
}

Node* deleteNode(Node* root, int value) {
    if (!root) return nullptr;

    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        // Node with one child or no child
        if (!root->left) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (!root->right) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // Node with two children
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

void inOrder(Node* root) {
    if (!root) return;
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    root->left->left = new Node(3);
    root->left->right = new Node(7);

    cout << "Original BST (In-order): ";
    inOrder(root);
    cout << endl;

    root = deleteNode(root, 5); // Delete a node with one child

    cout << "After Deleting Node 5 (In-order): ";
    inOrder(root);
    cout << endl;

    return 0;
```

```
}
```