Program No 1:

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

Code:

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

struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *n, *first = NULL, *last = NULL, *current, *temp;

void deletefirst()
{
    if (first == NULL)
        return; // List is empty
    temp = first;
    first = first->next;
    if (first != NULL)
        first->prev = NULL;
    else
        last = NULL; // List is now empty
    delete temp;
}

void insertlast(int data)
{
    n = new node();
    n->data = data;
    n->prev = last;
    n->next = NULL;
    if (last != NULL)
        last->next = n;
    last = n;
    if (first == NULL)
        first = n;
}

void display()
{
    current = first;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main()
```

```
{
    for (int i = 1; i <= 5; i++)
    {
        insertlast(i);
    }

    deletefirst();

    cout << "Display after Deletion at First:\n";
    display();

    return 0;
}
```

Output:

```
Display after Deletion at First:
2 3 4 5
```

Program No 2:

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

Code:

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

struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *n, *first = NULL, *last = NULL, *current, *temp;

void insertlast(int data)
{
    n = new node();
    n->data = data;
    n->prev = last;
    n->next = NULL;
    if (last != NULL)
        last->next = n;
    last = n;
    if (first == NULL)
        first = n;
}

void deletelast()
{
    if (last == NULL)
        return; // List is empty
```

```
    temp = last;
    last = last->prev;
    if (last != NULL)
        last->next = NULL;
    else
        first = NULL; // List is now empty
    delete temp;
}

void display()
{
    current = first;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main()
{
    for (int i = 1; i <= 5; i++)
    {
        insertlast(i);
    }

    deletelast();

    cout << "Display after Deletion at Last:\n";
    display();

    return 0;
}
```

Output:

```
Display after Deletion at Last:
1 2 3 4
```

Program No 3:

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

Code:

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

struct node
{
    int data;
    struct node *prev;
    struct node *next;
```

```
};
struct node *n, *first = NULL, *last = NULL, *current, *temp;

void insertlast(int data)
{
    n = new node();
    n->data = data;
    n->prev = last;
    n->next = NULL;
    if (last != NULL)
        last->next = n;
    last = n;
    if (first == NULL)
        first = n;
}

void deletebyvalue(int value)
{
    current = first;
    while (current != NULL && current->data != value)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        if (current->prev != NULL)
            current->prev->next = current->next;
        else
            first = current->next;

        if (current->next != NULL)
            current->next->prev = current->prev;
        else
            last = current->prev;

        delete current;
    }
}

void display()
{
    current = first;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main()
{
    for (int i = 1; i <= 5; i++)
```

```
    {
        insertlast(i);
    }

    cout << "Original List:\n";
    display();

    deletebyvalue(3);

    cout << "Display after Deleting Node with Value 3:\n";
    display();

    return 0;
}
```

Output:

```
Original List:
1 2 3 4 5
Display after Deleting Node with Value 3:
1 2 4 5
```

Program No 4:

How would you delete a node at a specific position in a doubly linked list? Show it in code.

Code:

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

struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *n, *first = NULL, *last = NULL, *current, *temp;

void insertlast(int data)
{
    n = new node();
    n->data = data;
    n->prev = last;
    n->next = NULL;
    if (last != NULL)
        last->next = n;
    last = n;
    if (first == NULL)
        first = n;
}

void deletebyposition(int position)
{
    if (position < 1)
```

```
    {
        cout << "Invalid position!" << endl;
        return;
    }

    current = first;
    int index = 1;

    while (current != NULL && index < position)
    {
        current = current->next;
        index++;
    }

    if (current == NULL)
    {
        cout << "Position out of range!" << endl;
        return;
    }

    if (current->prev != NULL)
        current->prev->next = current->next;
    else
        first = current->next;

    if (current->next != NULL)
        current->next->prev = current->prev;
    else
        last = current->prev;

    delete current;
}

void display()
{
    current = first;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main()
{
    for (int i = 1; i <= 5; i++)
    {
        insertlast(i);
    }

    cout << "Original List:\n";
    display();
```

```
    deletebyposition(3);

    cout << "Display after Deleting Node at Position 3:\n";
    display();

    return 0;
}
```

Output:

```
Original List:
1 2 3 4 5
Display after Deleting Node at Position 3:
1 2 4 5
```

Program No 5:

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

Code:

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

struct node
{
    int data;
    struct node *prev;
    struct node *next;
};
struct node *n, *first = NULL, *last = NULL, *current, *temp;

void insertlast(int data)
{
    n = new node();
    n->data = data;
    n->prev = last;
    n->next = NULL;
    if (last != NULL)
        last->next = n;
    last = n;
    if (first == NULL)
        first = n;
}

void deleteafternode(int value)
{
    current = first;
    while (current != NULL && current->data != value)
    {
        current = current->next;
    }

    if (current != NULL && current->next != NULL)
    {
```

```
        temp = current->next;
        current->next = temp->next;
        if (temp->next != NULL)
            temp->next->prev = current;
        else
            last = current;

        delete temp;
    }
}

void display()
{
    current = first;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void reverseDisplay()
{
    current = last;
    while (current != NULL)
    {
        cout << current->data << " ";
        current = current->prev;
    }
    cout << endl;
}

int main()
{
    for (int i = 1; i <= 5; i++)
    {
        insertlast(i);
    }
    cout << "Display before Deletion after Node:\n";
    display();

    deleteafternode(3);

    cout << "Display after Deletion after Node:\n";
    display();

    cout << "Reverse Display after Deletion after Node:\n";
    reverseDisplay();

    return 0;
}
```

Output:

```
Display before Deletion after Node:
1 2 3 4 5
Display after Deletion after Node:
1 2 3 5
Reverse Display after Deletion after Node:
5 3 2 1
```

Program No 6:

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

Code:

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

struct node {
    int data;
    struct node* link;
};

struct node *first = NULL, *last = NULL, *n;

void create(int data) {
    n = new node();
    n->data = data;

    if (first == NULL) {
        n->link = n;
        first = last = n;
    } else {
        n->link = first;
        last->link = n;
        last = n;
    }
}

void deletefirst() {
    if (first == NULL) return;

    struct node* temp = first;

    if (first == last) {
        first = last = NULL;
    } else {
        first = first->link;
        last->link = first;
    }

    delete temp;
}
```

```
void display() {
    if (first == NULL) return;

    struct node* current = first;
    do {
        cout << current->data << " ";
        current = current->link;
    } while (current != first);

    cout << endl;
}

int main() {
    create(1);
    create(2);
    create(3);

    cout << "Before Delete First:" << endl;
    display();

    deletefirst();

    cout << "After Delete First:" << endl;
    display();

    return 0;
}
```

Output:

```
Before Delete First:
1 2 3
After Delete First:
2 3
```

Program No 7:

How can you delete the last node in a circular linked list? Write the code.

Code:

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

struct node {
    int data;
    struct node* link;
};

struct node *first = NULL, *last = NULL, *n;

void create(int data) {
    n = new node();
    n->data = data;
```

```
    if (first == NULL) {
        n->link = n;
        first = last = n;
    } else {
        n->link = first;
        last->link = n;
        last = n;
    }
}

void deletelast() {
    if (first == NULL) return;

    if (first == last) {
        delete last;
        first = last = NULL;
    } else {
        struct node* current = first;

        while (current->link != last)
            current = current->link;

        delete last;
        last = current;
        last->link = first;
    }
}

void display() {
    if (first == NULL) return;

    struct node* current = first;
    do {
        cout << current->data << " ";
        current = current->link;
    } while (current != first);

    cout << endl;
}

int main() {
    create(1);
    create(2);
    create(3);

    cout << "Before Delete Last:" << endl;
    display();

    deletelast();

    cout << "After Delete Last:" << endl;
    display();

    return 0;
```

```
}
```

Output:

```
Before Delete Last:
1 2 3
After Delete Last:
1 2
```

Program No 8:

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

Code:

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

struct node {
    int data;
    struct node* link;
};

struct node *first = NULL, *last = NULL, *n;

void create(int data) {
    n = new node();
    n->data = data;

    if (first == NULL) {
        n->link = n;
        first = last = n;
    } else {
        n->link = first;
        last->link = n;
        last = n;
    }
}

void deletebyvalue(int value) {
    if (first == NULL) return;

    struct node* current = first;
    struct node* previous = NULL;

    do {
        if (current->data == value) {
            if (previous == NULL) {  // Deleting the first node
                if (first == last) {
                    delete first;
                    first = last = NULL;
                } else {
                    first = first->link;
                    last->link = first;
                    delete current;
```

```
                }
            } else {
                previous->link = current->link;
                if (current == last) last = previous;
                delete current;
            }
            return;
        }
        previous = current;
        current = current->link;
    } while (current != first);
}

void display() {
    if (first == NULL) return;

    struct node* current = first;
    do {
        cout << current->data << " ";
        current = current->link;
    } while (current != first);

    cout << endl;
}

int main() {
    create(1);
    create(2);
    create(3);

    cout << "Before Delete By Value:" << endl;
    display();

    deletebyvalue(2);

    cout << "After Delete By Value:" << endl;
    display();

    return 0;
}
```

Output:

```
Before Delete By Value:
1 2 3
After Delete By Value:
1 3
```

Program No 9:

How will you delete a node at a specific position in a circular linked list? Write code for it.

Code:

```
#include <iostream>
```

```cpp
using namespace std;

struct node {
    int data;
    struct node* link;
};

struct node *first = NULL, *last = NULL, *n;

void create(int data) {
    n = new node();
    n->data = data;

    if (first == NULL) {
        n->link = n;
        first = last = n;
    } else {
        n->link = first;
        last->link = n;
        last = n;
    }
}

void deleteatposition(int position) {
    if (first == NULL || position < 1) return;

    struct node* current = first;
    struct node* previous = NULL;

    int count = 1;

    do {
        if (count == position) {
            if (previous == NULL) {  // Deleting the first node
                if (first == last) {
                    delete first;
                    first = last = NULL;
                } else {
                    first = first->link;
                    last->link = first;
                    delete current;
                }
            } else {
                previous->link = current->link;
                if (current == last) last = previous;
                delete current;
            }
            return;
        }
        previous = current;
        current = current->link;
        count++;
    } while (current != first);
}
```

```
void display() {
    if (first == NULL) return;

    struct node* current = first;
    do {
        cout << current->data << " ";
        current = current->link;
    } while (current != first);

    cout << endl;
}

int main() {
    create(1);
    create(2);
    create(3);
    create(4);

    cout << "Before Deleting Node at Position 2:" << endl;
    display();

    deleteatposition(2);

    cout << "After Deleting Node at Position 2:" << endl;
    display();

    return 0;
}
```

Output:

```
Before Deleting Node at Position 2:
1 2 3 4
After Deleting Node at Position 2:
1 3 4
```

Program No 10:

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

Code:

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

struct node {
    int data;
    struct node* link;
};

struct node *first = NULL, *last = NULL, *n;

void create(int data) {
    n = new node();
```

```
    n->data = data;

    if (first == NULL) {
        n->link = n;
        first = last = n;
    } else {
        n->link = first;
        last->link = n;
        last = n;
    }
}

void deletelast() {
    if (first == NULL) return;

    if (first == last) {
        delete last;
        first = last = NULL;
    } else {
        struct node* current = first;

        while (current->link != last)
            current = current->link;

        delete last;
        last = current;
        last->link = first;
    }
}

void display() {
    if (first == NULL) return;

    struct node* current = first;
    do {
        cout << current->data << " ";
        current = current->link;
    } while (current != first);

    cout << endl;
}

int main() {
    create(1);
    create(2);
    create(3);

    cout << "Before Delete Last:" << endl;
    display();

    deletelast();

    cout << "After Delete Last:" << endl;
    display();
```

```
    return 0;
}
```

Output:

```
Before Delete Last:
1 2 3
After Delete Last:
1 2
```

Program No 11:

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

Code:

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

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

// Function to create a new node
void createNode(Node *&n, int a)
{
    n = new Node();
    n->data = a;
    n->left = NULL;
    n->right = NULL;
}

// Function to insert a value into the tree
void insert(Node *&root, int b)
{
    if (root == NULL)
    {
        createNode(root, b);
        return;
    }

    if (b < root->data)
    {
        insert(root->left, b);
    }
    else if (b > root->data)
```

```
        {
            insert(root->right, b);
        }
    }

    // Function to count the total number of nodes in the tree
    int countNodes(Node *root)
    {
        if (root == NULL)
        {
            return 0; // Base case: no nodes
        }
        return 1 + countNodes(root->left) + countNodes(root->right);
    }

    int main()
    {
        Node *root = NULL;

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

        cout << "Total number of nodes in the tree: " << countNodes(root)
    << endl;

        return 0;
    }
```

Output:

```
Total number of nodes in the tree: 7
```

Program No 12:

How can you search for a specific value in a binary search tree? Write the code.

Code:

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

// Node structure
struct Node
{
    int data;
    Node *left;
```

```
    Node *right;
};

// Function to create a new node
void createNode(Node *&n, int a)
{
    n = new Node();
    n->data = a;
    n->left = NULL;
    n->right = NULL;
}

// Function to insert a value into the tree
void insert(Node *&root, int b)
{
    if (root == NULL)
    {
        createNode(root, b);
        return;
    }

    if (b < root->data)
    {
        insert(root->left, b);
    }
    else if (b > root->data)
    {
        insert(root->right, b);
    }
}

// Function to search for a value in the tree
bool search(Node *root, int c)
{
    if (root == NULL)
    {
        return false;
    }

    if (c == root->data)
    {
        return true;
    }
    else if (c < root->data)
    {
        return search(root->left, c);
    }
    else
    {
        return search(root->right, c);
    }
}

int main()
```

```
{
    Node *root = NULL;

    insert(root, 50);
    insert(root, 30);
    insert(root, 70);

    if (search(root, 30))

    {
        cout << "Value found." << endl;
    }
    else
    {
        cout << "Value not found." << endl;
    }

    return 0;
}
```

Output:

```
Value found.
```

Program No 13:

Write code to traverse a binary search tree in in-order, pre-order, and post-order.

Code:

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

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

void createNode(Node *&n, int a)
{
    n = new Node();
    n->data = a;
    n->left = NULL;
    n->right = NULL;
}

void insert(Node *&root, int b)
{
    if (root == NULL)
    {
        createNode(root, b);
        return;
```

```
    }

    if (b < root->data)
    {
        insert(root->left, b);
    }
    else if (b > root->data)
    {
        insert(root->right, b);
    }
}

void inOrderTraversal(Node *root)
{
    if (root != NULL)
    {
        inOrderTraversal(root->left);
        cout << root->data << " ";
        inOrderTraversal(root->right);
    }
}

void preOrderTraversal(Node *root)
{
    if (root != NULL)
    {
        cout << root->data << " ";
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

void postOrderTraversal(Node *root)
{
    if (root != NULL)
    {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        cout << root->data << " ";
    }
}

int main()
{
    Node *root = NULL;

    insert(root, 50);
    insert(root, 30);
    insert(root, 70);

    cout << "In-Order Traversal: ";
    inOrderTraversal(root);
    cout << endl;
```

```
    cout << "Pre-Order                    Traversal: ";
    preOrderTraversal(root);
    cout << endl;

    cout << "Post-Order Traversal: ";
    postOrderTraversal(root);
    cout << endl;

    return 0;
}
```

Output:

```
In-Order Traversal: 30 50 70
Pre-Order Traversal: 50 30 70
Post-Order Traversal: 30 70 50
```

Program No 14:

Write code to traverse a binary search tree in in-order, pre-order, and post-order.

Code:

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

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

// Function to create a new node
void createNode(Node *&n, int a)
{
    n = new Node();
    n->data = a;
    n->left = NULL;
    n->right = NULL;
}

// Function to insert a value into the tree
void insert(Node *&root, int b)
{
    if (root == NULL)
    {
        createNode(root, b);
        return;
    }
```

```
    if (b < root->data)
    {
        insert(root->left, b);
    }
    else if (b > root->data)
    {
        insert(root->right, b);
    }
}

// Function for in-order traversal (ascending order)
void inOrder(Node *root)
{
    if (root == NULL)
    {
        return;
    }

    // Traverse the left subtree
    inOrder(root->left);

    // Visit the current node
    cout << root->data << " ";

    // Traverse the right subtree
    inOrder(root->right);
}

// Function for reverse in-order traversal (descending order)
void reverseInOrder(Node *root)
{
    if (root == NULL)
    {
        return;
    }

    // Traverse the right subtree
    reverseInOrder(root->right);

    // Visit the current node
    cout << root->data << " ";

    // Traverse the left subtree
    reverseInOrder(root->left);
}

int main()
```

```
{
    Node *root = NULL;

    // Inserting elements into the BST
    insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    // Display in-order traversal
    cout << "In-order traversal: ";
    inOrder(root);
    cout << endl;

    // Display reverse in-order traversal
    cout << "Reverse in-order traversal: ";
    reverseInOrder(root);
    cout << endl;

    return 0;
}
```

Output:

```
In-order traversal: 20 30 40 50 60 70 80
Reverse in-order traversal: 80 70 60 50 40 30 20
```

Program No 14:

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

Code:

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

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

void createNode (Node *&n, int a)
{
    n = new Node();
    n->data = a;
    n->left = NULL;
```

```
    n->right = NULL;
}

void insert (Node *&root, int c)
{
    if (root == NULL)
    {
        createNode(root, c);
        return;
    }

    if (c < root->data)
    {
        insert(root->left, c);
    }
    else if (c > root->data)
    {
        insert(root->right, c);
    }
}

bool checkDup (Node *&root, int v)
{
    if (root == NULL)
    {
        createNode(root, v);
        return true;
    }

    if (v < root->data)
    {
        return checkDup(root->left, v);
    }
    else if (v > root->data)
    {
        return checkDup(root->right, v);
    }
    else
    {
        return false;
    }
}

int main()
{
    Node *root = NULL;

    createNode(root, 90);
    createNode(root, 70);
    createNode(root, 50);

    if (checkDup(root, 50))
    {
        cout << "a 50 inserted." << endl;
```

```
    }
    else
    {
        cout << "Duplicate a not inserted." << endl;
    }

    return 0;
}
```

Output:
```
Duplicate a not inserted.
```

Program No 15:

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.

Code:

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

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

void createNode(Node *&n, int d)
{
    n = new Node();
    n->data = d;
    n->left = NULL;
    n->right = NULL;
}

void insert(Node *&root, int e)
{
    if (root == NULL)
    {
        createNode(root, e);
        return;
    }

    if (e < root->data)
    {
        insert(root->left, e);
    }
    else if (e > root->data)
    {
        insert(root->right, e);
    }
```

```
}

int findMin(Node *root)
{
    if (root == NULL)
    {
        return -1;
    }
    if (root->left == NULL)
    {
        return root->data;
    }
    return findMin(root->left);
}

Node *deleteNode(Node *root, int f)
{
    int minValue = -1; // Define minValue at the start of the function

    if (root == NULL) // Base case: Tree is empty or value not found
    {
        return root; // Return NULL or the unchanged root
    }

    if (f < root->data) // If the value is smaller, recur on the left
subtree
    {
        root->left = deleteNode(root->left, f);
    }
    else if (f > root->data) // If the value is larger, recur on the
right subtree
    {
        root->right = deleteNode(root->right, f);
    }
    else
    {
        // Node with the value is found
        if (root->left == NULL && root->right == NULL) // Case 1: No
children (leaf node)
        {
            delete root; // Free the memory for the current node
            root = NULL; // Set the node pointer to NULL
        }
        else if (root->left == NULL) // Case 2: Only right child
        {
            Node *temp = root;  // Store the current node in a
temporary variable
            root = root->right; // Replace the current node with its
right child
            delete temp;        // Delete the temporary node
        }
        else if (root->right == NULL) // Case 3: Only left child
        {
```

```
            Node *temp =                    root; // Store the current
node in a temporary variable
            root = root->left; // Replace the current node with its
left child
            delete temp;       // Delete the temporary node
        }
        else
        {
            // Case 4: Two children
            minValue = findMin(root->right);                // Find
the minimum value in the right subtree
            root->data = minValue;                          //
Replace data with the minimum value
            root->right = deleteNode(root->right, minValue); // Recur
to delete the minimum node
        }
    }
    return root; // Return the modified root
}

int main()
{
    Node *root = NULL;

    insert(root, 50);
    insert(root, 30);
    insert(root, 70);

    root = deleteNode(root, 30);

    cout << "Deleted 30 from the tree." << endl;

    return 0;
}
```

Output:
Deleted 30 from the tree.