Doubly Linked List

1. **Delete the first node in a doubly linked list**

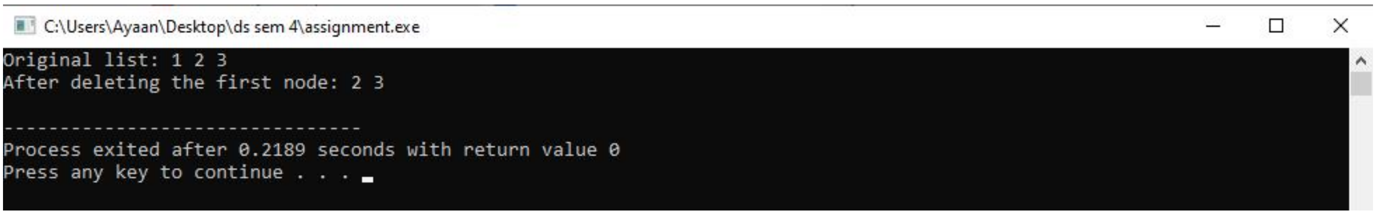
Explaination:

To delete the first node, we update the head pointer to point to the second node and adjust the prev pointer of the new head to NULL.

Code:

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

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

void deleteFirst(Node*& head) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   Node* temp = head;
   head = head->next;
   if (head != NULL)
      head->prev = NULL;
   delete temp;
}

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

```
int main() {
    Node* head = new Node{1, NULL, NULL};
    Node* second = new Node{2, NULL, head};
    Node* third = new Node{3, NULL, second};
    head->next = second;
    second->next = third;

    cout << "Original list: ";
    displayForward(head);

    deleteFirst(head);
    cout << "After deleting the first node: ";
    displayForward(head);

    return 0;
}
```



2. Delete the last node in a doubly linked list

Explaination:

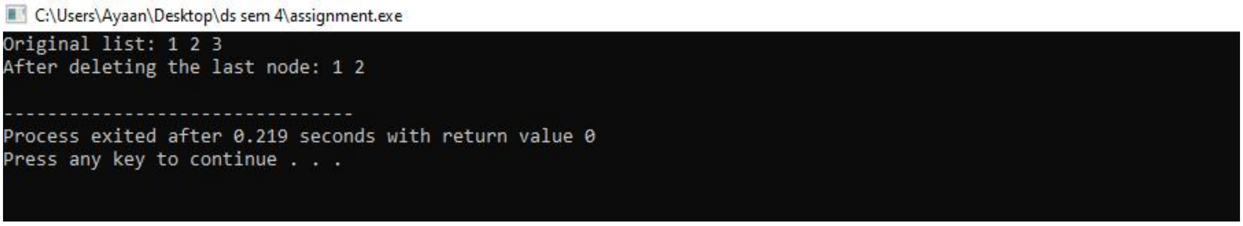
To delete the last node, we traverse the list to the last node, update the next pointer of the second-last node to NULL, and delete the last node.

Code:

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

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

```
void deleteLast(Node*& head) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   if (head->next == NULL) { // Only one node in the list
      delete head;
      head = NULL;
      return;
   }
   Node* temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->prev->next = NULL;
   delete temp;
}
void displayForward(Node* head) {
   Node* temp = head;
   while (temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}

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

   cout << "Original list: ";
   displayForward(head);

   deleteLast(head);
   cout << "After deleting the last node: ";
   displayForward(head);

   return 0;
```

}



3. Delete a node by its value in a doubly linked list

Explaination:

To delete a node by its value, we traverse the list to find the node, then adjust the next and prev pointers of the adjacent nodes and delete the target node.

Code:

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

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

void deleteByValue(Node*& head, int value) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   Node* temp = head;

   while (temp != NULL && temp->data != value) {
      temp = temp->next;
   }
   if (temp == NULL) {
      cout << "Value not found in the list.\n";
      return;
```

```
    }
    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next; // If deleting the first node
    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    delete temp;
}
void displayForward(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}


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

    cout << "Original list: ";
    displayForward(head);

    deleteByValue(head, 2);
    cout << "After deleting node with value 2: ";
    displayForward(head);

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 1 2 3
After deleting node with value 2: 1 3

--------------------------------
Process exited after 0.224 seconds with return value 0
Press any key to continue . . .
```

4. Delete a node at a specific position

Explaination:

To delete a node at a specific position, we traverse to the node, adjust its adjacent nodes, and delete it.

Code:

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

struct Node {
   int data;
   Node* next;
   Node* prev;
};
void deleteFirst(Node*& head) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   Node* temp = head;
   head = head->next;
   if (head != NULL)
      head->prev = NULL;
   delete temp;
}
void deleteAtPosition(Node*& head, int position) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   if (position == 1) { // Deleting the first node
      deleteFirst(head);
      return;
   }
   Node* temp = head;
   for (int i = 1; temp != NULL && i < position; i++) {
      temp = temp->next;
```

```
    }
    if (temp == NULL) {
        cout << "Position out of range.\n";
        return;
    }
    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    delete temp;
}
void displayForward(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = new Node{1, NULL, NULL};
    Node* second = new Node{4, NULL, head};
    Node* third = new Node{3, NULL, second};
    head->next = second;
    second->next = third;

    cout << "Original list: ";
    displayForward(head);

    deleteAtPosition(head, 2);
    cout << "After deleting node at position 2: ";
    displayForward(head);

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original list: 1 4 3
After deleting node at position 2: 1 3

--------------------------------
Process exited after 0.2303 seconds with return value 0
Press any key to continue . . .
```

5. Forward and reverse traversal functions

Explaination:

Forward traversal starts at the head and follows next pointers to the end. Reverse traversal finds the tail (last node) and follows prev pointers back to the beginning. The deletion process itself maintains the correct next and prev links, so the traversal logic remains the same.

Code:

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

struct Node {
   int data;
   Node* next;
   Node* prev;
};
void deleteFirst(Node*& head) {
   if (head == NULL) {
      cout << "List is empty, nothing to delete.\n";
      return;
   }
   Node* temp = head;
   head = head->next;
   if (head != NULL)
      head->prev = NULL;
   delete temp;
}
void displayForward(Node* head) {
   Node* temp = head;
   while (temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
void displayReverse(Node* head) {
   if (head == NULL) {
```

```
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next; // Go to the last node
    }
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
    Node* head = new Node{1, NULL, NULL};
    Node* second = new Node{2, NULL, head};
    Node* third = new Node{3, NULL, second};
    head->next = second;
    second->next = third;
    deleteFirst( head) ;
    cout<<"after deleting a node :"<<endl;
    cout << "Forward traversal: ";
    displayForward(head);

    cout << "Reverse traversal: ";
    displayReverse(head);

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
after deleting a node :
Forward traversal: 2 3
Reverse traversal: 3 2

--------------------------------
Process exited after 0.2153 seconds with return value 0
Press any key to continue . . .
```

Circular Linked List

1. Delete the first node in a circular linked list

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        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 display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    Node* temp = head;
```

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

    Node* node1 = new Node{10, nullptr};
    Node* node2 = new Node{20, nullptr};
    Node* node3 = new Node{30, nullptr};

    head = node1;
    node1->next = node2;
    node2->next = node3;
    node3->next = head;

    cout << "Original list: ";
    display(head);

    deleteFirst(head);

    cout << "After deleting the first node: ";
    display(head);

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 10 20 30
After deleting the first node: 20 30

--------------------------------
Process exited after 0.2078 seconds with return value 0
Press any key to continue . . .
```

2. Delete the last node in a circular linked list

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

struct Node {
```

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

void deleteLast(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        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 display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

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

```
int main() {
    Node* head = nullptr;

    Node* node1 = new Node{10, nullptr};
    Node* node2 = new Node{20, nullptr};
    Node* node3 = new Node{30, nullptr};

    head = node1;
    node1->next = node2;
    node2->next = node3;
    node3->next = head;

    cout << "Original list: ";
    display(head);

   deleteLast(head);

    cout << "After deleting the first node: ";
    display(head);

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original list: 10 20 30
After deleting the first node: 10 20

--------------------------------
Process exited after 0.2187 seconds with return value 0
Press any key to continue . . .
```

3. Delete a node by its value in a circular linked list

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

struct Node {
    int data;
    Node* next;
};
void deleteFirst(Node*& head) {
```

```cpp
    if (head == nullptr) {
        cout << "List is empty.\n";
        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 deleteByValue(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

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

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

    do {
        if (temp->data == value) {
            if (prev == nullptr) { // Deleting head
```

```
            deleteFirst(head);
        } else {
            prev->next = temp->next;
            delete temp;
        }
        return;
    }
    prev = temp;
    temp = temp->next;
  } while (temp != head);

  cout << "Value not found.\n";
}

void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty.\n";
    return;
  }

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

int main() {
  Node* head = nullptr;

  Node* node1 = new Node{10, nullptr};
  Node* node2 = new Node{20, nullptr};
  Node* node3 = new Node{30, nullptr};

  head = node1;
  node1->next = node2;
  node2->next = node3;
  node3->next = head;

  cout << "Original list: ";
```

```
    display(head);

    deleteByValue(head,20);

    cout << "After deleting the first node: ";
    display(head);

    return 0;
}
```

4. Delete a node at a specific position in a circular linked list

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

struct Node {
    int data;
    Node* next;
};
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        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 deleteAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    if (position == 1) {
        deleteFirst(head);
        return;
    }

    Node* temp = head;
    Node* prev = nullptr;
    int count = 1;

    do {
        if (count == position) {
            prev->next = temp->next;
            delete temp;
            return;
        }
        prev = temp;
        temp = temp->next;
        count++;
    } while (temp != head);

    cout << "Position out of range.\n";
}

void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
```

```
    }

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

int main() {
    Node* head = nullptr;

    Node* node1 = new Node{10, nullptr};
    Node* node2 = new Node{40, nullptr};
    Node* node3 = new Node{30, nullptr};

    head = node1;
    node1->next = node2;
    node2->next = node3;
    node3->next = head;

    cout << "Original list: ";
    display(head);

     deleteAtPosition(head,2);

    cout << "After deleting the first node: ";
    display(head);

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 10 40 30
After deleting the first node: 10 30

--------------------------------
Process exited after 0.2766 seconds with return value 0
Press any key to continue . . .
```

6. Show forward traversal after deleting a node in a circular linked list

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

struct Node {
    int data;
    Node* next;
};
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        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 deleteAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

    if (position == 1) {
        deleteFirst(head);
        return;
    }
```

```
    Node* temp = head;
    Node* prev = nullptr;
    int count = 1;

    do {
        if (count == position) {
            prev->next = temp->next;
            delete temp;
            return;
        }
        prev = temp;
        temp = temp->next;
        count++;
    } while (temp != head);

    cout << "Position out of range.\n";
}

void deleteByValue(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List is empty.\n";
        return;
    }

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

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

    do {
        if (temp->data == value) {
            if (prev == nullptr) { // Deleting head
                deleteFirst(head);
            } else {
                prev->next = temp->next;
                delete temp;
```

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

    cout << "Value not found.\n";
}
void deleteLast(Node*& head) {
    if (head == nullptr) {
      cout << "List is empty.\n";
      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 display(Node* head) {
    if (head == nullptr) {
      cout << "List is empty.\n";
      return;
    }

    Node* temp = head;
    do {
```

```
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

int main() {
    Node* head = nullptr;

    Node* node1 = new Node{10, nullptr};
    Node* node2 = new Node{20, nullptr};
    Node* node3 = new Node{30, nullptr};
    Node* node4 = new Node{40, nullptr};

    head = node1;
    node1->next = node2;
    node2->next = node3;
    node3->next = node4;
    node4->next = head;

    cout << "Original list: ";
    display(head);

    deleteFirst(head);
    cout << "After deleting the first node: ";
    display(head);

    deleteLast(head);
    cout << "After deleting the last node: ";
    display(head);

    deleteByValue(head, 20);
    cout << "After deleting node with value 20: ";
    display(head);

    deleteAtPosition(head, 2);
    cout << "After deleting node at position 2: ";
    display(head);

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original list: 10 20 30 40
After deleting the first node: 20 30 40
After deleting the last node: 20 30
After deleting node with value 20: 30
Position out of range.
After deleting node at position 2: 30

--------------------------------
Process exited after 0.2446 seconds with return value 0
Press any key to continue . . .
```

Binary Search Tree

1. Program to Count All the Nodes in a Binary Search Tree

Explanation:

To count the nodes, use a recursive function that traverses the entire tree and adds 1 for each visited node.

Code:

```
#include <iostream>using namespace std;
struct Node {
   int data;
   Node* left;
   Node* right;
};
Node* createNode(int value) {
   Node* newNode = new Node;
   newNode->data = value;
   newNode->left = newNode->right = nullptr;
   return newNode;
}
Node* insert(Node* root, int value) {
   if (root == nullptr) {
      return createNode(value);
   }
   if (value < root->data) {
      root->left = insert(root->left, value);
   } else if (value > root->data) {
      root->right = insert(root->right, value);
   }
```

```
    return root;
}
int countNodes(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    return 1 + countNodes(root->left) + countNodes(root->right);
}
int main() {
    Node* root = nullptr;
    root = 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 BST: " << countNodes(root) << endl;

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Total number of nodes in the BST: 7

--------------------------------
Process exited after 0.2315 seconds with return value 0
Press any key to continue . . .
```

2. Search for a Specific Value in a Binary Search Tree

Explanation:

To search for a value, recursively traverse the tree. If the current node's value matches the target, return true; otherwise, search in the left or right subtree based on the value.

Code:

```
#include <iostream>
```

```cpp
using namespace std;

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

Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

bool search(Node* root, int key) {
    if (root == nullptr) {
        return false;
    }
    if (root->data == key) {
        return true;
    } else if (key < root->data) {
        return search(root->left, key);
    } else {
        return search(root->right, key);
    }
}

int main() {
```

```
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    int key = 40;
    if (search(root, key)) {
        cout << key << " is found in the BST." << endl;
    } else {
        cout << key << " is not found in the BST." << endl;
    }

    return 0;
}
```

3. Traverse a Binary Search Tree (In-order, Pre-order, Post-order)

Explanation:

- **In-order:** Traverse left subtree, visit root, traverse right subtree.
- **Pre-order:** Visit root, traverse left subtree, traverse right subtree.
- **Post-order:** Traverse left subtree, traverse right subtree, visit root.

Code:

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

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

```
    Node* right;
};

Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

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

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

void postOrder(Node* root) {
    if (root != nullptr) {
        postOrder(root->left);
```

```
        postOrder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

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

    cout << "Pre-order traversal: ";
    preOrder(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postOrder(root);
    cout << endl;

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
In-order traversal: 20 30 40 50 60 70 80
Pre-order traversal: 50 30 20 40 70 60 80
Post-order traversal: 20 40 30 60 80 70 50

--------------------------------
Process exited after 0.2218 seconds with return value 0
Press any key to continue . . .
```

4. Reverse In-order Traversal

Explanation:

Traverse the right subtree first, then visit the root, and finally traverse the left subtree.

Code:

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

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

Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}

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

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

int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);
    cout<<"original inorder:";
    inOrder( root);
    cout<<endl;


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

    return 0;
}
```

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

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

--------------------------------
Process exited after 0.2347 seconds with return value 0
Press any key to continue . . .
```

5. Check for Duplicate Values in a Binary Search Tree

Explanation:

Use a helper function to compare the values of nodes while inserting. If a duplicate is found, return true.

Code:

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

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

Node* createNode(int value) {
   Node* newNode = new Node;
   newNode->data = value;
   newNode->left = newNode->right = nullptr;
   return newNode;
}


bool insertAndCheckDuplicate(Node*& root, int value) {
   if (root == nullptr) {
      root = createNode(value);
      return false;
   }
   if (value == root->data) {
      return true; // Duplicate found
   } else if (value < root->data) {
      return insertAndCheckDuplicate(root->left, value);
   } else {
      return insertAndCheckDuplicate(root->right, value);
   }
}

int main() {
   Node* root = nullptr;
   int values[] = {50, 30, 70, 20, 40, 60, 80, 30}; // 30 is duplicate
   bool duplicate = false;
```

```
    for (int value : values) {
        if (insertAndCheckDuplicate(root, value)) {
            duplicate = true;
            break;
        }
    }

    if (duplicate) {
        cout << "Duplicate values found in the BST." << endl;
    } else {
        cout << "No duplicate values in the BST." << endl;
    }

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Duplicate values found in the BST.

--------------------------------
Process exited after 0.2619 seconds with return value 0
Press any key to continue . . .
```

6. Delete a Node from a Binary Search Tree

Explanation:

- **Case 1:** Deleting a leaf node.
- **Case 2:** Deleting a node with one child.
- **Case 3:** Deleting a node with two children (find in-order successor).

Code:

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

struct Node {
    int data;
```

```
    Node* left;
    Node* right;
};

Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
Node* findMin(Node* root) {
    while (root->left != nullptr) {
        root = root->left;
    }
    return root;
}

Node* deleteNode(Node* root, int value) {
    if (root == nullptr) {
        return root;
    }
    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        if (root->left == nullptr && root->right == nullptr) {
            delete root;
            return nullptr;
        } else if (root->left == nullptr) {
```

```
        Node* temp = root->right;
        delete root;
        return temp;
    } else if (root->right == nullptr) {
        Node* temp = root->left;
        delete root;
        return temp;
    } else {
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
  }
  return root;
}
void inOrder(Node* root) {
  if (root != nullptr) {
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
  }
}


int main() {
  Node* root = nullptr;
  root = insert(root, 50);
  insert(root, 30);
  insert(root, 70);
  insert(root, 20);
  insert(root, 40);
  insert(root, 60);
  insert(root, 80);

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

  root = deleteNode(root, 20); // Leaf node
  root = deleteNode(root, 30); // Node with one child
  root = deleteNode(root, 50); // Node with two children
```

```
    cout << "After deletions In-order: ";
    inOrder(root);
    cout << endl;

    return 0;
}
```

```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original In-order: 20 30 40 50 60 70 80
After deletions In-order: 40 60 70 80

--------------------------------
Process exited after 0.2438 seconds with return value 0
Press any key to continue . . .
```