

DATA STRUCTURE AND ALGO **ASSIGNMENT**

(Double Link List, Circular Link List, BST)

(DOUBLE LINKED LIST)

- 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 *next;

    struct node *prev;
};

struct node *n, *first, *last, *current, * temp;

Void create(int data)
{
    n = new node();

    n->data = data;

    if(first == NULL)
    {
        n->next = n->prev = NULL;

        first = last = n;
```

```

    }
    else
    {
        n->next = NULL;
        n->prev = last;
        last->next = n;
        last = n;
    }
}

void deleteFirst()
{
    temp = first;
    first = first->next;
    first->prev = NULL;
    delete(first);
}

void display()
{
    current = first;

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

int main()

```

```

{
    for(int i=1; i<6; i++)
    {
        create(i);
    }

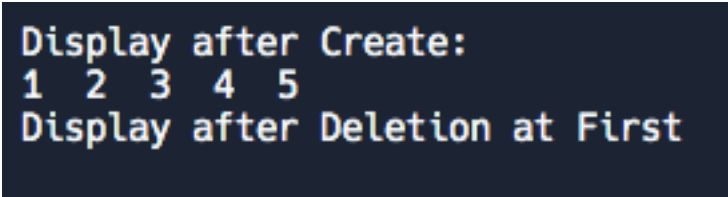
    cout<<"Display after Create: "<<endl;
    display();

    cout<<"\nDisplay after Deletion at First "<<endl;
    deleteFirst();
    display();

    return 0;
}

```

OUTPUT:



```

Display after Create:
1 2 3 4 5
Display after Deletion at First

```

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 *next;

```

```
    struct node *prev;

};

struct node *n, *first, *last, *current, * temp;

void create(int data)

{
    n = new node();
    n->data = data;
    if(first == NULL)
    {
        n->next = n->prev = NULL;
        first = last = n;
    }
    else
    {
        n->next = NULL;
        n->prev = last;
        last->next = n;
        last = n;
    }
}

void deleteLast()

{
    current = first;
    while(current->next != last)
    {
        current = current->next;
    }
}
```

```
    delete last;

    last = current;

    last->next = NULL;
}

void display()
{
    current = first;

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

int main()
{
    for(int i=1; i<6; i++)
    {
        create(i);
    }

    cout<<"Display after Creation: "<<endl;
    display();

    cout<<"\nDisplay after Deletion at last "<<endl;
    deleteLast();
    display();

    return 0;
}
```

OUTPUT

```
Display after Creation:
1 2 3 4 5
Display after Deletion at last
1 2 3 4
```

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 *next;
    struct node *prev;
};

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

// Function to create a new node and add it to the list
void create(int data)
{
    n = new node();
    n->data = data;
```

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

// Function to delete the first node

```
void deletefirst()
{
    if(first != NULL)
    {
        current = first;
        first = first->next;
        if(first != NULL)
            first->prev = NULL;
        delete(current);
    }
}
```

```
// Function to delete the last node
```

```
void deletelast()
```

```
{  
    if(last != NULL)  
    {  
        current = last;  
        last = last->prev;  
        if(last != NULL)  
            last->next = NULL;  
        delete(current);  
    }  
}
```

```
// Function to delete a node by its value
```

```
void deleteByValue(int value)
```

```
{  
    current = first;  
    struct node *previous = NULL;  
  
    while(current != NULL)  
    {  
        if(current->data == value)  
        {  
            if(current == first) // Deleting the first node  
            {  
                deletefirst();  
            }  
        }  
    }
```



```

        else if(current == last) // Deleting the last node
        {
            deletelast();
        }
        else // Deleting a middle node
        {
            previous->next = current->next;
            current->next->prev = previous;
            delete(current);
        }
        break; // After deletion, break the loop
    }
    previous = current;
    current = current->next;
}
}

```

// Function to display the linked list

```

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

```

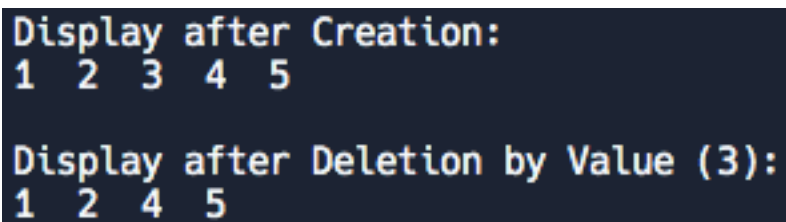
```
}

int main()
{
    // Creating a doubly linked list with values 1 to 5
    for(int i = 1; i < 6; i++)
    {
        create(i);
    }

    cout << "Display after Creation: " << endl;
    display();

    // Deleting node with value 3
    cout << "\nDisplay after Deletion by Value (3): " << endl;
    deleteByValue(3);
    display();
    return 0;
}
```

OUTPUT:



```
Display after Creation:
1 2 3 4 5

Display after Deletion by Value (3):
1 2 4 5
```

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;

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

// Function to delete a node at a specific position
void deleteNode(Node** head_ref, int position) {
    // If the linked list is empty
    if (*head_ref == nullptr)
        return;

    // Store the head node
    Node* temp = *head_ref;

    // If the head needs to be removed
    if (position == 0) {
        *head_ref = temp->next; // Change head
        if (*head_ref != nullptr)
            (*head_ref)->prev = nullptr; // Change prev of new head node to nullptr
        delete temp; // Free old head
    }
```

```

    return;
}

// Traverse to the node to be deleted
for (int i = 0; temp != nullptr && i < position; i++)
    temp = temp->next;

// If the position is more than the number of nodes
if (temp == nullptr)
    return;

// Change next only if the node to be deleted is NOT the last node
if (temp->next != nullptr)
    temp->next->prev = temp->prev;

// Change prev only if the node to be deleted is NOT the first node
if (temp->prev != nullptr)
    temp->prev->next = temp->next;

// Free the memory occupied by the node
delete temp;
}

// Function to push a node at the beginning
void push(Node** head_ref, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = (*head_ref);

```

```

    new_node->prev = nullptr;
if ((*head_ref) != nullptr)
    (*head_ref)->prev = new_node;

    (*head_ref) = new_node;
}

// Function to print the doubly linked list
void printList(Node* node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
}

int main() {
    Node* head = nullptr;

    // Create the doubly linked list: 10<->8<->4<->2
    push(&head, 2);
    push(&head, 4);
    push(&head, 8);
    push(&head, 10);

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

    // Delete node at position 2

```

```

deleteNode(&head, 2);

cout << "\nDoubly Linked List after deletion at position 2: ";
printList(head);

return 0;
}

```

OUTPUT:

```

Original Doubly Linked List: 10 8 4 2
Doubly Linked List after deletion at position 2: 10 8 2

```

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

FUNCTIONS

The printListForward function traverses the list from the head to the end, printing each node's data. The printListReverse function first moves to the end of the list and then traverses it backward, printing each node's data.

(CIRCULAR LINKED LIST)

1. 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 *n, *first, *last, *current, * temp;

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()
{
    temp = first;
    first = first->link;
    last->link = first;
```

```
    delete(temp);
}

void display()
{
    current = first;

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

int main()
{
    for(int i=1; i<6; i++)
    {
        create(i);
    }

    cout<<"Display after Creation: "<<endl;
    display();
    cout<<"\nDisplay after Deletion at First "<<endl;
    deleteFirst();
    display();
    return 0;
}
```


OUTPUT:

```
Display after Creation:
1 2 3 4 5
Display after Deletion at First
2 3 4 5
```

2. 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 *n, *first = NULL, *last = NULL, *current, *temp;

// Function to create and insert a node in the circular linked list
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;
    }
}

// Function to delete the first node in the circular linked list
void deleteFirst() {
    if(first == NULL) {
        cout << "List is empty, cannot delete first node." << endl;
        return;
    }

    if(first == last) { // Case where there's only one node
        delete first;
        first = last = NULL;
    } else {
        current = first;
        first = first->link;
        last->link = first; // Update the link of the last node
        delete current;
    }
}

// Function to delete the last node in the circular linked list
void deleteLast() {

```

```

if(first == NULL) {
    cout << "List is empty, cannot delete last node." << endl;
    return;
}

if(first == last) { // Case where there's only one node
    delete first;
    first = last = NULL;
} else {
    current = first;
    while(current->link != last) {
        current = current->link;
    }
    delete last;
    current->link = first;
    last = current;
}
}

// Function to display the circular linked list
void display() {
    if(first == NULL) {
        cout << "List is empty!" << endl;
        return;
    }
    current = first;
    do {

```

```

        cout << current->data << " ";

        current = current->link;
    } while(current != first);

    cout << endl;
}

int main() {

    // Create circular linked list with values from 1 to 5
    for(int i = 1; i < 6; i++) {
        create(i);
    }

    cout << "Display after Creation: " << endl;

    display();

    // Delete the first and last nodes
    cout << "\nDisplay after Deletion at First and Last: " << endl;

    deleteFirst();
    deleteLast();
    display();

    return 0;
}

```

OUTPUT

```

Display after Creation:
1 2 3 4 5

Display after Deletion at First and Last:
2 3 4

```

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

FUNCTION TO DELETE NODE BY VALUE

Code:

```
#include<iostream>

using namespace std;

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

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

// Function to create and insert a node into the circular linked list
void create(int data) {
    n = new node();
    n->data = data;

    if (first == NULL) {
        n->link = n; // Points to itself, circular list
        first = last = n;
    } else {
        n->link = first;
        last->link = n; // Last node points to new node
        last = n; // Update last pointer to the new node
    }
}
```

```

    }
}

// Function to delete a node by its value in the circular linked list
void deleteByValue(int value) {
    if (first == NULL) {
        cout << "List is empty!" << endl;
        return;
    }

    current = first;
    previous = NULL;

    do {
        // If the current node contains the value, delete it
        if (current->data == value) {
            // Case 1: Deleting the first node
            if (current == first) {
                if (first == last) { // Only one node in the list
                    delete first;
                    first = last = NULL;
                } else {
                    first = first->link;
                    last->link = first;
                    delete current;
                }
            }
        }
    }
}

```

```

// Case 2: Deleting the last node
else if (current == last) {
    previous->link = first;
    last = previous;
    delete current;
}

// Case 3: Deleting a middle node
else {
    previous->link = current->link;
    delete current;
}

return; // Value found and deleted, exit function
}

previous = current;
current = current->link;
} while (current != first); // Continue loop until we come back to first node

cout << "Value not found in the list!" << endl;
}

// Function to display the circular linked list
void display() {
    if (first == NULL) {
        cout << "List is empty!" << endl;
        return;
    }
}

```

```
current = first;

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

cout << endl;
}

int main() {
    // Create a circular linked list
    for (int i = 1; i <= 5; i++) {
        create(i);
    }

    cout << "Display after Creation: " << endl;
    display();

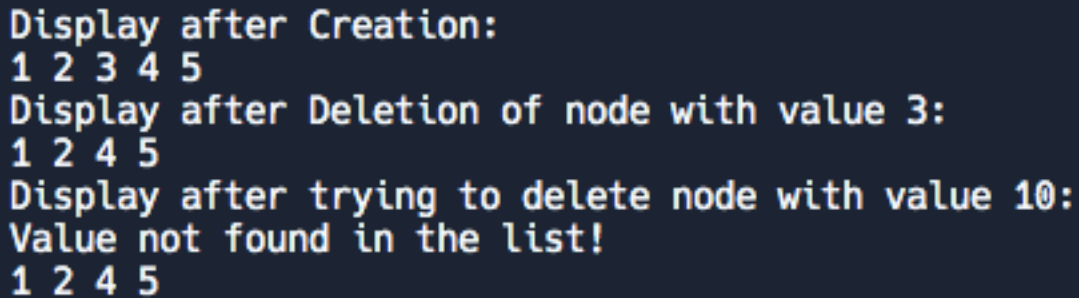
    // Deleting a node by value
    cout << "Display after Deletion of node with value 3: " << endl;
    deleteByValue(3);
    display();

    // Try deleting a non-existing value
    cout << "Display after trying to delete node with value 10: " << endl;
    deleteByValue(10);
    display();
}
```



```
    return 0;
}
```

OUTPUT:



```
Display after Creation:
1 2 3 4 5
Display after Deletion of node with value 3:
1 2 4 5
Display after trying to delete node with value 10:
Value not found in the list!
1 2 4 5
```

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

CODE

```
#include<iostream>

using namespace std;

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

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

// Function to create and insert a node into the circular linked list
void create(int data) {
    n = new node();
    n->data = data;
```

```
if (first == NULL) {
    n->link = n; // Points to itself, circular list
    first = last = n;
} else {
    n->link = first;
    last->link = n; // Last node points to new node
    last = n; // Update last pointer to the new node
}
}

// Function to delete the first node in the circular linked list
void deleteFirst() {
    if (first == NULL) return; // If the list is empty, do nothing

    temp = first;

    // If there is only one node in the list
    if (first == last) {
        first = last = NULL;
    } else {
        first = first->link;
        last->link = first; // Last node's link should point to the new first node
    }

    delete temp;
}
```

```
// Function to delete the last node in the circular linked list

void deleteLast() {
    if (first == NULL) return; // If the list is empty, do nothing

    current = first;

    // If there's only one node in the list
    if (first == last) {
        delete first;
        first = last = NULL;
        return;
    }

    // Traverse to the second last node
    while (current->link != last) {
        current = current->link;
    }

    delete last; // Delete the last node
    current->link = first; // Update the second last node's link to point to the first node
    last = current; // Update the last node pointer to the second last node
}

// Function to delete a node by its value in the circular linked list
void deleteByValue(int value) {
    if (first == NULL) return; // If the list is empty, do nothing
```

```

struct node *previous = NULL;
current = first;

do {
    if (current->data == value) {
        if (current == first) {
            deleteFirst(); // Call deleteFirst() to delete the first node
        } else if (current == last) {
            deleteLast(); // Call deleteLast() to delete the last node
        } else {
            previous->link = current->link; // Bypass the current node
            delete current;
        }
        return; // After deleting, no need to continue the loop
    }
    previous = current;
    current = current->link;
} while (current != first); // Continue until we loop back to the first node

cout << "Value not found in the list!" << endl; // Value not found
}

// Function to display the circular linked list
void display() {
    if (first == NULL) {
        cout << "List is empty!" << endl;
        return;
    }

```

```

    }

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

int main() {
    // Create circular linked list with values 1 to 5
    for (int i = 1; i <= 5; i++) {
        create(i);
    }

    cout << "Display after Creation: " << endl;
    display();

    // Deleting a node by value (in this case, value 1)
    cout << "\nDisplay after Deletion by Value (1): " << endl;
    deleteByValue(1);
    display();

    // Deleting a node that doesn't exist
    cout << "\nDisplay after trying to delete a non-existing value (10): " << endl;
    deleteByValue(10);
}

```

```
display();

return 0;
}
```

OUTPUT

```
Display after Creation:
1 2 3 4 5

Display after Deletion by Value (1):
2 3 4 5

Display after trying to delete a non-existing value (10):
Value not found in the list!
2 3 4 5
```

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 *n, *first, *last, *current, * temp;
```

```
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()
{
    temp = first;
    first = first->link;
    last->link = first;
    delete(temp);
}

void displaytraversed()
{
    current = first;
```

```

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

int main()
{
    for(int i=1; i<6; i++)
    {
        create(i);
    }
    cout<<"Display after Creation: "<<endl;
    displaytraversed();
    cout<<"\nDisplay traversed array after Deletion at First "<<endl;
    deleteFirst();
    displaytraversed();
    return 0;
}

```

OUTPUT:

```

Display after Creation:
1 2 3 4 5
Display traversed array after Deletion at First
2 3 4 5

```


(BINARY SEARCH TREE)

1. 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
Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->left = nullptr;
    node->right = nullptr;
    return node;
}

// Function to insert a new node in the binary search tree
Node* insert(Node* node, int data) {
    if (node == nullptr)
```

```
        return newNode(data);

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

    return node;
}

// Function to count all the nodes in the binary search tree
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, 5);
    insert(root, 3);
    insert(root, 2);
    insert(root, 4);
    insert(root, 7);
    insert(root, 6);

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

```
    return 0;
}
```

OUTPUT

```
Total number of nodes in the binary search tree: 6
```

2. 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
Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->left = nullptr;
    node->right = nullptr;
    return node;
}
```

```
// Function to insert a new node in the binary search tree
```

```
Node* insert(Node* node, int data) {
```

```
    if (node == nullptr)
```

```
        return newNode(data);
```

```
    if (data < node->data)
```

```
        node->left = insert(node->left, data);
```

```
    else if (data > node->data)
```

```
        node->right = insert(node->right, data);
```

```
    return node;
```

```
}
```

```
// Function to search for a specific value in the binary search tree
```

```
bool search(Node* root, int key) {
```

```
    if (root == nullptr)
```

```
        return false;
```

```
    if (root->data == key)
```

```
        return true;
```

```
    if (key < root->data)
```

```
        return search(root->left, key);
```

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

```
}
```

```

int main() {
    Node* root = nullptr;
    root = insert(root, 5);
    insert(root, 3);
    insert(root, 2);
    insert(root, 4);
    insert(root, 7);
    insert(root, 6);

    int key = 4;
    if (search(root, key))
        cout << "Value " << key << " found in the binary search tree." << endl;
    else
        cout << "Value " << key << " not found in the binary search tree." << endl;

    return 0;
}

```

OUTPUT

Value 4 found in the binary search tree.

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

CODE

```

#include <iostream>
using namespace std;

```

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

// Function to create a new node
Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->left = nullptr;
    node->right = nullptr;
    return node;
}

// Function to insert a new node in the binary search tree
Node* insert(Node* node, int data) {
    if (node == nullptr)
        return newNode(data);

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

```
    return node;
}

// Function for in-order traversal
void inorderTraversal(Node* root) {
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}

// Function for pre-order traversal
void preorderTraversal(Node* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Function for post-order traversal
void postorderTraversal(Node* root) {
    if (root != nullptr) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        cout << root->data << " ";
    }
}
```

```
    }  
}  
  
int main() {  
    Node* root = nullptr;  
    root = insert(root, 5);  
    insert(root, 3);  
    insert(root, 2);  
    insert(root, 4);  
    insert(root, 7);  
    insert(root, 6);  
    insert(root, 8);  
  
    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: 2 3 4 5 6 7 8
Pre-order traversal: 5 3 2 4 7 6 8
Post-order traversal: 2 4 3 6 8 7 5
```

**4. How will you write reverse in-order traversal for a binary search tree?
Show it in code.**

CODE

```
#include <iostream>

using namespace std;

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

// Function to create a new node
Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->left = nullptr;
    node->right = nullptr;
    return node;
}
```

```
// Function to insert a new node in the binary search tree
```

```
Node* insert(Node* node, int data) {  
    if (node == nullptr)  
        return newNode(data);  
  
    if (data < node->data)  
        node->left = insert(node->left, data);  
    else if (data > node->data)  
        node->right = insert(node->right, data);  
  
    return node;  
}
```

```
// Function for reverse in-order traversal
```

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

```
int main() {  
    Node* root = nullptr;  
    root = insert(root, 5);  
    insert(root, 3);  
}
```

```

insert(root, 2);
insert(root, 4);
insert(root, 7);
insert(root, 6);
insert(root, 8);

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

return 0;
}

```

OUTPUT

```
Reverse In-order traversal: 8 7 6 5 4 3 2
```

5. 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;
};

```

```
// Function to create a new node
```

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

```
// Function to insert a new node in the binary search tree
```

```
Node* insert(Node* node, int data) {  
    if (node == nullptr)  
        return newNode(data);  
  
    if (data < node->data)  
        node->left = insert(node->left, data);  
    else if (data > node->data)  
        node->right = insert(node->right, data);  
  
    return node;  
}
```

```
// Function to check for duplicate values in the binary search tree
```

```
bool checkDuplicates(Node* root, Node* &prev) {  
    if (root == nullptr)  
        return false;
```

```

// Check the left subtree
if (checkDuplicates(root->left, prev))
    return true;

// Check the current node
if (prev != nullptr && root->data == prev->data)
    return true;

// Update the previous node
prev = root;

// Check the right subtree
return checkDuplicates(root->right, prev);
}

int main() {
    Node* root = nullptr;
    root = insert(root, 5);
    insert(root, 3);
    insert(root, 2);
    insert(root, 4);
    insert(root, 7);
    insert(root, 6);
    insert(root, 8);
    insert(root, 3); // Duplicate value

```

```

Node* prev = nullptr;
if (checkDuplicates(root, prev))
    cout << "The binary search tree contains duplicate values." << endl;
else
    cout << "The binary search tree does not contain duplicate values." << endl;

return 0;
}

```

OUTPUT

The binary search tree does not contain duplicate values.

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.

CODE

```

#include <iostream>
using namespace std;

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

// Function to create a new node
Node* newNode(int data) {
    Node* node = new Node();
}

```

```
node->data = data;
node->left = nullptr;
node->right = nullptr;
return node;
}

// Function to insert a new node in the binary search tree
Node* insert(Node* node, int data) {
    if (node == nullptr)
        return newNode(data);

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

    return node;
}

// Function to find the minimum value node in the tree
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current && current->left != nullptr)
        current = current->left;
    return current;
}
```

```
// Function to delete a node from the binary search tree
Node* deleteNode(Node* root, int key) {
    // Base case
    if (root == nullptr)
        return root;

    // If the key to be deleted is smaller than the root's key, then it lies in the left subtree
    if (key < root->data)
        root->left = deleteNode(root->left, key);

    // If the key to be deleted is greater than the root's key, then it lies in the right subtree
    else if (key > root->data)
        root->right = deleteNode(root->right, key);

    // If the key is the same as the root's key, then this is the node to be deleted
    else {
        // Node with only one child or no child
        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;
        }
    }
}
```



```

    // Node with two children: Get the inorder successor (smallest in the right subtree)
    Node* temp = minValueNode(root->right);

    // Copy the inorder successor's content to this node
    root->data = temp->data;

    // Delete the inorder successor
    root->right = deleteNode(root->right, temp->data);
}
return root;
}

// Function to print the binary search tree in in-order traversal
void inorderTraversal(Node* root) {
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 5);
    insert(root, 3);
    insert(root, 2);
    insert(root, 4);
}

```

```
insert(root, 7);
insert(root, 6);
insert(root, 8);

cout << "In-order traversal of the original tree: ";
inorderTraversal(root);
cout << endl;

// Delete a leaf node
root = deleteNode(root, 2);
cout << "In-order traversal after deleting leaf node 2: ";
inorderTraversal(root);
cout << endl;

// Delete a node with one child
root = deleteNode(root, 3);
cout << "In-order traversal after deleting node 3 with one child: ";
inorderTraversal(root);
cout << endl;

// Delete a node with two children
root = deleteNode(root, 5);
cout << "In-order traversal after deleting node 5 with two children: ";
inorderTraversal(root);
cout << endl;

return 0;
```

}

OUTPUT

```
In-order traversal of the original tree: 2 3 4 5 6 7 8  
In-order traversal after deleting leaf node 2: 3 4 5 6 7 8  
In-order traversal after deleting node 3 with one child: 4 5 6 7 8  
In-order traversal after deleting node 5 with two children: 4 6 7 8
```