Practical Assignment

- **Name:**

  Abdul Haseeb Arif.

- **Reg no.:**

  2023-BS-AI-033.

- **Submitted to:**

  Mam Irsha Qureshi

- **Subect:**

  Data Structures and Algorithms.

- **Department:**

  BS Artificial Intelligence. (Section-A)

- **Semester:**

  3rd

➢ <u>Doubly Linked List</u>

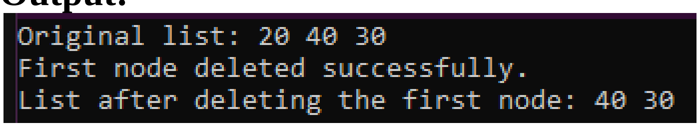
• 1.Write a program to delete the first node in a doubly linked list.

```
#include <iostream>

using namespace std;

// Define a Node of the doubly linked list

struct Node {

    int data;

    Node* prev;

    Node* next;

    Node(int val) : data(val), prev(nullptr), next(nullptr) {}

};

// Function to delete the first node of a doubly linked list

void deleteFirstNode(Node*& head) {

    if (head == nullptr) { // If the list is empty

        cout << "The list is already empty." << endl;

        return;

    }

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

    head = head->next; // Move the head pointer to the next node

    if (head != nullptr) { // If the list is not empty after deletion

        head->prev = nullptr;

    }

    delete temp; // Free the memory of the old head node

    cout << "First node deleted successfully." << endl;

}


// Function to display the doubly linked list
```

```
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to insert a node at the end of the doubly linked list
void appendNode(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 20);
    appendNode(head, 40);
```

```
    appendNode(head, 30);

    cout << "Original list: ";

    displayList(head);

    // Delete the first node

    deleteFirstNode(head);

    cout << "List after deleting the first node: ";

    displayList(head);

    return 0;

}
```

- **Output:**



```
Original list: 20 40 30
First node deleted successfully.
List after deleting the first node: 40 30
```

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

```
#include <iostream>

using namespace std;

// Define a Node of the doubly linked list

struct Node {

    int data;

    Node* prev;

    Node* next;

    Node(int val) : data(val), prev(nullptr), next(nullptr) {}

};

// Function to delete the last node of a doubly linked list

void deleteLastNode(Node*& head) {
```

```
    if (head == nullptr) { // If the list is empty
        cout << "The list is already empty." << endl;
        return;
    }
    if (head->next == nullptr) { // If the list has only one node
        delete head;
        head = nullptr;
        cout << "Last node deleted successfully." << endl;
        return;
    }
    Node* temp = head;
    // Traverse to the last node
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    // Update the previous node's next pointer
    temp->prev->next = nullptr;
    delete temp; // Free the memory of the last node
    cout << "Last node deleted successfully." << endl;
}
// Function to display the doubly linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
```

```
    cout << endl;
}
// Function to insert a node at the end of the doubly linked list
void appendNode(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 20);
    appendNode(head, 40);
    appendNode(head, 30);
    cout << "Original list: ";
    displayList(head);
    // Delete the last node
    deleteLastNode(head);
```

```
    cout << "List after deleting the last node: ";
    displayList(head);
    return 0;
}
```

- **Output:**

```
Original list: 20 40 30
Last node deleted successfully.
List after deleting the last node: 20 40
```

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

```
#include <iostream>
using namespace std;
// Define a Node of the doubly linked list
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};
// Function to delete a node by its value in a doubly linked list
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // If the list is empty
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    // Search for the node with the specified value
    while (temp != nullptr && temp->data != value) {
```

```
        temp = temp->next;
    }

    if (temp == nullptr) { // Value not found
        cout << "Value " << value << " not found in the list." << endl;
        return;
    }
    // Node with the value found
    if (temp == head) { // If it's the head node
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) { // If it's the last node
        temp->prev->next = nullptr;
    } else { // If it's a middle node
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp; // Free the memory of the node
    cout << "Node with value " << value << " deleted successfully." << endl;
}
// Function to display the doubly linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
```

```
        temp = temp->next;
    }
    cout << endl;
}
// Function to insert a node at the end of the doubly linked list
void appendNode(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 20);
    appendNode(head, 40);
    appendNode(head, 30);
    appendNode(head, 50);
    cout << "Original list: ";
    displayList(head);
```

```
    // Delete a node by value
    deleteNodeByValue(head, 20);
    cout << "List after deleting the node with value 20: ";
    displayList(head);
    // Try deleting a non-existent value
    deleteNodeByValue(head, 50);
    return 0;
}
```

- **Output:**

```
Original list: 20 40 30 50
Node with value 20 deleted successfully.
List after deleting the node with value 20: 40 30 50
Node with value 50 deleted successfully.
```

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

```
#include <iostream>
using namespace std;
// Define a Node of the doubly linked list
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};

// Function to delete a node at a specific position in a doubly linked list
```

```
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // If the list is empty
        cout << "The list is empty." << endl;
        return;
    }
    if (position <= 0) { // Invalid position
        cout << "Invalid position. Position should be greater than 0." << endl;
        return;
    }
    Node* temp = head;
    int currentIndex = 1;
    // Traverse the list to find the node at the specified position
    while (temp != nullptr && currentIndex < position) {
        temp = temp->next;
        currentIndex++;
    }
    if (temp == nullptr) { // Position exceeds the size of the list
        cout << "Position " << position << " exceeds the list size." << endl;
        return;
    }
    // If the node to be deleted is the head
    if (temp == head) {
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) { // If it's the last node
```

```
        temp->prev->next = nullptr;
    } else { // If it's a middle node
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp; // Free the memory of the node
    cout << "Node at position " << position << " deleted successfully." << endl;
}
// Function to display the doubly linked list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Function to insert a node at the end of the doubly linked list
void appendNode(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
```

```
    }
    temp->next = newNode;
    newNode->prev = temp;
}
int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    appendNode(head, 40);
    cout << "Original list: ";
    displayList(head);
    // Delete a node at position 2
    deleteNodeAtPosition(head, 2);
    cout << "List after deleting the node at position 2: ";
    displayList(head);
    // Delete the head node
    deleteNodeAtPosition(head, 1);
    cout << "List after deleting the node at position 1: ";
    displayList(head);
    // Try deleting at an invalid position
    deleteNodeAtPosition(head, 10);
    return 0;
}
```

- **Output:**

```
Original list: 10 20 30 40
Node at position 2 deleted successfully.
List after deleting the node at position 2: 10 30 40
Node at position 1 deleted successfully.
List after deleting the node at position 1: 30 40
Position 10 exceeds the list size.
```

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

```
#include <iostream>

using namespace std;

// Define a Node of the doubly linked list

struct Node {

    int data;

    Node* prev;

    Node* next;

    Node(int val) : data(val), prev(nullptr), next(nullptr) {}

};

// Function to perform forward traversal of the doubly linked list

void forwardTraversal(Node* head) {

    cout << "Forward Traversal: ";

    Node* temp = head;

    while (temp != nullptr) {

        cout << temp->data << " ";

        temp = temp->next;

    }

    cout << endl;

}
```

```
// Function to perform reverse traversal of the doubly linked list
void reverseTraversal(Node* head) {
    if (head == nullptr) { // If the list is empty
        cout << "Reverse Traversal: List is empty." << endl;
        return;
    }
    // Move to the last node
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    // Traverse backward from the last node
    cout << "Reverse Traversal: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}
// Function to append a node to the end of the doubly linked list
void appendNode(Node*& head, int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) { // If the list is empty
        head = newNode;
        return;
    }
```

```
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
// Function to delete a node at a specific position in the doubly linked list
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // If the list is empty
        cout << "The list is empty." << endl;
        return;
    }
    if (position <= 0) { // Invalid position
        cout << "Invalid position. Position should be greater than 0." << endl;
        return;
    }
    Node* temp = head;
    int currentIndex = 1;
    // Traverse to the node at the specified position
    while (temp != nullptr && currentIndex < position) {
        temp = temp->next;
        currentIndex++;
    }
    if (temp == nullptr) { // Position exceeds the list size
        cout << "Position " << position << " exceeds the list size." << endl;
        return;
```

```
    }
    if (temp == head) { // Deleting the head node
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        }
    } else if (temp->next == nullptr) { // Deleting the last node
        temp->prev->next = nullptr;
    } else { // Deleting a middle node
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }
    delete temp; // Free the memory of the node
    cout << "Node at position " << position << " deleted successfully." << endl;
}
// Main function to demonstrate forward and reverse traversal
int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    appendNode(head, 40);
    // Perform forward and reverse traversal before deletion
    forwardTraversal(head);
    reverseTraversal(head);
    // Delete a node at position 2
```

```
    deleteNodeAtPosition(head, 2);

    // Perform forward and reverse traversal after deletion

    forwardTraversal(head);

    reverseTraversal(head);

    return 0;

}
```

- **Output:**

```
Forward Traversal: 10 20 30 40
Reverse Traversal: 40 30 20 10
Node at position 2 deleted successfully.
Forward Traversal: 10 30 40
Reverse Traversal: 40 30 10
```

➢ **Circular Linked List**

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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to delete the first node of a circular linked list
void deleteFirstNode(Node*& head) {
    if (head == nullptr) { // List is empty
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }

    if (head->next == head) { // Only one node in the list
        delete head;
        head = nullptr;
        return;
    }
    // Find the last node in the list
    Node* last = head;
    while (last->next != head) {
        last = last->next;
    }
    // Point the last node to the second node
```

```
    Node* temp = head;
    head = head->next;
    last->next = head;
    // Delete the first node
    delete temp;
}
// Function to insert a node at the end of the circular linked list
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }

    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
```

```
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
// Main function
int main() {
    Node* head = nullptr;
    // Insert some nodes
    insert(head, 10);
    insert(head, 20);
    insert(head, 30);
    insert(head, 40);
    cout << "Original list: ";
    display(head);
    // Delete the first node
    deleteFirstNode(head);
    cout << "After deleting the first node: ";
    display(head);
    return 0;
}
```

- **Output:**

```
Original list: 10 20 30 40
After deleting the first node: 20 30 40
```

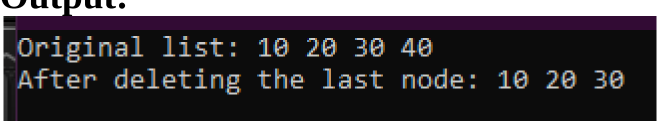
•2. How can you delete the last node in a circular linked list? Write the code.

```
#include <iostream>

using namespace std;

// Node structure

struct Node {

    int data;

    Node* next;

};

// Function to delete the last node of a circular linked list

void deleteLastNode(Node*& head) {

    if (head == nullptr) { // List is empty

        cout << "List is empty. Nothing to delete." << endl;

        return;

    }

    if (head->next == head) { // Only one node in the list

        delete head;

        head = nullptr;

        return;

    }

    // Traverse the list to find the second last node

    Node* current = head;

    while (current->next->next != head) {

        current = current->next;

    }
```

```
    // Adjust pointers and delete the last node
    Node* last = current->next;
    current->next = head;
    delete last;
}
// Function to insert a node at the end of the circular linked list
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
```

```
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
// Main function
int main() {
    Node* head = nullptr;
    // Insert some nodes
    insert(head, 10);
    insert(head, 20);
    insert(head, 30);
    insert(head, 40);
    cout << "Original list: ";
    display(head);
    // Delete the last node
    deleteLastNode(head);
    cout << "After deleting the last node: ";
    display(head);
    return 0;
}
```

- **Output:**



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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to delete a node by its value in a circular linked list
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // List is empty
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    Node* current = head;
    Node* previous = nullptr;
    // Case 1: The node to be deleted is the only node in the list
    if (head->data == value && head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    // Case 2: The node to be deleted is the head node
    if (head->data == value) {
        // Find the last node
        while (current->next != head) {
            current = current->next;
        }
```

```
        Node* temp = head;
        head = head->next;
        current->next = head;
        delete temp;
        return;
    }
    // Case 3: The node to be deleted is in the middle or end of the list
    do {
        previous = current;
        current = current->next;
        if (current->data == value) {
            previous->next = current->next;
            delete current;
            return;
        }
    } while (current != head);

    // If the value was not found
    cout << "Value " << value << " not found in the list." << endl;
}
// Function to insert a node at the end of the circular linked list
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
```

```
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
// Main function
int main() {
    Node* head = nullptr;
    // Insert some nodes
    insert(head, 10);
```

```
    insert(head, 20);
    insert(head, 30);
    insert(head, 40);
    cout << "Original list: ";
    display(head);
    // Delete a node by value
    deleteNodeByValue(head, 20);
    cout << "After deleting node with value 20: ";
    display(head);
    // Try to delete a node not in the list
    deleteNodeByValue(head, 50);
    cout << "After attempting to delete node with value 50: ";
    display(head);
    return 0;
}
```

- **Output:**

```
Original list: 10 20 30 40
After deleting node with value 20: 10 30 40
Value 50 not found in the list.
After attempting to delete node with value 50: 10 30 40
```

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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
```

```
};
// Function to delete a node at a specific position in a circular linked list
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // List is empty
        cout << "List is empty. Nothing to delete." << endl;
        return;
    }
    // If the position is 0, delete the head node
    if (position == 0) {
        // If there's only one node
        if (head->next == head) {
            delete head;
            head = nullptr;
        } else {
            Node* last = head;
            while (last->next != head) { // Find the last node
                last = last->next;
            }
            Node* temp = head;
            head = head->next; // Move the head pointer
            last->next = head; // Adjust the last node's next pointer
            delete temp; // Delete the old head
        }
        return;
    }
    Node* current = head;
    Node* previous = nullptr;
```

```
    int count = 0;
    // Traverse to the desired position
    while (current->next != head && count < position) {
        previous = current;
        current = current->next;
        count++;
    }
    // If position is out of bounds
    if (current->next == head && count < position) {
        cout << "Position out of bounds." << endl;
        return;
    }
    // Delete the node
    previous->next = current->next;
    delete current;
}
// Function to insert a node at the end of the circular linked list
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
```

```
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head;
}
// Function to display the circular linked list
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
// Main function
int main() {
    Node* head = nullptr;
    // Insert some nodes
    insert(head, 10);
    insert(head, 20);
    insert(head, 30);
    insert(head, 40);
    cout << "Original list: ";
```

```
    display(head);
    // Delete node at position 2
    deleteNodeAtPosition(head, 2);
    cout << "After deleting node at position 2: ";
    display(head);
    // Try to delete node at an invalid position
    deleteNodeAtPosition(head, 5);
    cout << "After attempting to delete node at position 5: ";
    display(head);
    return 0;
}
```

- **Output:**

```
Original list: 10 20 30 40
After deleting node at position 2: 10 20 40
Position out of bounds.
After attempting to delete node at position 5: 10 20 40
```

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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};
// Function to delete a node at a specific position in a circular linked list
void deleteNodeAtPosition(Node*& head, int position) {
```

```
if (head == nullptr) { // List is empty
    cout << "List is empty. Nothing to delete." << endl;
    return;
}
// If the position is 0, delete the head node
if (position == 0) {
    // If there's only one node
    if (head->next == head) {
        delete head;
        head = nullptr;
    } else {
        Node* last = head;
        while (last->next != head) { // Find the last node
            last = last->next;
        }
        Node* temp = head;
        head = head->next; // Move the head pointer
        last->next = head; // Adjust the last node's next pointer
        delete temp; // Delete the old head
    }
    return;
}
Node* current = head;
Node* previous = nullptr;
int count = 0;
// Traverse to the desired position
while (current->next != head && count < position) {
```

```
        previous = current;
        current = current->next;
        count++;
    }
    // If position is out of bounds
    if (current->next == head && count < position) {
        cout << "Position out of bounds." << endl;
        return;
    }
    // Delete the node
    previous->next = current->next;
    delete current;
}
// Function to insert a node at the end of the circular linked list
void insert(Node*& head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (head == nullptr) {
        head = newNode;
        newNode->next = head;
        return;
    }
    Node* temp = head;
    while (temp->next != head) {
        temp = temp->next;
    }
    temp->next = newNode;
```

```
    newNode->next = head;
}
// Function to display the circular linked list in forward traversal
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
// Main function
int main() {
    Node* head = nullptr;
    // Insert some nodes
    insert(head, 10);
    insert(head, 20);
    insert(head, 30);
    insert(head, 40);
    cout << "Original list: ";
    display(head);
    // Delete node at position 2
    deleteNodeAtPosition(head, 2);
```

```
    cout << "After deleting node at position 2: ";
    display(head);
    // Delete node at position 0 (head node)
    deleteNodeAtPosition(head, 0);
    cout << "Forward Traversal: ";
    display(head);
    return 0;
}
```

- **Output:**

```
Original list: 10 20 30 40
After deleting node at position 2: 10 20 40
Forward Traversal: 20 40
```

➢ Binary Search Tree

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

```
#include <iostream>
using namespace std;
// Node structure for Binary Search Tree
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};
// Function to insert a node in the Binary Search Tree
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
// Function to count the nodes in the Binary Search Tree
```

```
int countNodes(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    // Recursively count nodes in the left and right subtrees, and add 1 for the current node
    return 1 + countNodes(root->left) + countNodes(root->right);
}
// Function to perform an in-order traversal and print the tree
void inorderTraversal(Node* root) {
    if (root != nullptr) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}
// Main function
int main() {
    Node* root = nullptr;
    // Insert nodes into the BST
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
    cout << "In-order traversal of the Binary Search Tree: ";
```

```
    inorderTraversal(root);
    cout << endl;
    // Count the nodes in the BST
    int nodeCount = countNodes(root);
    cout << "Total number of nodes in the BST: " << nodeCount << endl;

    return 0;
}
```

- **Output:**

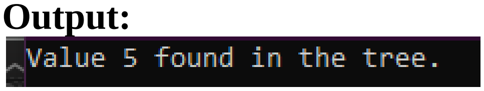
```
In-order traversal of the Binary Search Tree: 20 30 40 50 60 70 80
Total number of nodes in the BST: 7
```

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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Function to search for a value
```

```
bool search(Node* root, int key) {
    if (root == NULL) return false;
    if (root->data == key) return true;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);
}
// Main function to test searching
int main() {
    Node* root = new Node(10);
    root->left = new Node(5);
    root->right = new Node(15);
    int searchKey = 5;
    if (search(root, searchKey)) {
        cout << "Value " << searchKey << " found in the tree." << endl;
    } else {
        cout << "Value " << searchKey << " not found in the tree." << endl;
    }
    return 0;
}
```

- **Output:**



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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
```

```
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Traversal functions
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
void preorder(Node* root) {
    if (root == NULL) return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}
void postorder(Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
```

```
}
// Main function to test traversals
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(12);
    root->right->right = new Node(18);
    cout << "Inorder: ";
    inorder(root);
    cout << endl;
    cout << "Preorder: ";
    preorder(root);
    cout << endl;
    cout << "Postorder: ";
    postorder(root);
    cout << endl;
    return 0;
}
```

- **Output:**

```
Inorder: 3 5 7 10 12 15 18
Preorder: 10 5 3 7 15 12 18
Postorder: 3 7 5 12 18 15 10
```

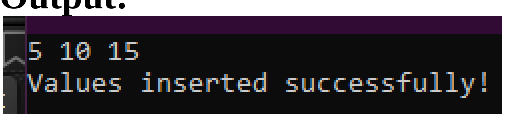
• 4.How will you write reverse in-order traversal for a binary search tree? Show

it in code.

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
            void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
// Function to insert a new value
Node* insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }
    if (val < root->data) {
        root->left = insert(root->left, val);
    } else if (val > root->data) {
```

```
        root->right = insert(root->right, val);
    }
    return root;
}

// Main function to test insertion
int main() {
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 5);
    root = insert(root, 15);
        inorder(root);
    cout << "\nValues inserted successfully!" << endl;
    return 0;
}
```

- **Output:**



```
5 10 15
Values inserted successfully!
```

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

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* left;
    Node* right;
```

```
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    }
};
// Function to insert a value (with duplicate check)
Node* insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }
    if (val < root->data) {
        root->left = insert(root->left, val);
    } else if (val > root->data) {
        root->right = insert(root->right, val);
    } else {
        cout << "Duplicate value " << val << " not allowed." << endl;
    }
    return root;
}
// Main function to test duplication handling
int main() {
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 3);
```

```
    root = insert(root, 7);

    root = insert(root, 12);

    root = insert(root, 10); // Duplicate Values

    root = insert(root, 18);

    return 0;}
```

- **Output:**

```
Duplicate value 10 not allowed.
```

• 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;

// Node structure

struct Node {

    int data;

    Node* left;

    Node* right;

    Node(int val) {

        data = val;

        left = NULL;

        right = NULL;

    }

};

// Function to find the minimum value node

Node* findMin(Node* root) {

    while (root && root->left != NULL) {

        root = root->left;

    }
```

```
    return root;
}
// Function to delete a node
Node* deleteNode(Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
// Main function to test deletion
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(12);
    root->right->right = new Node(18);
    cout << "Deleting value 5." << endl;
    root = deleteNode(root, 5);
    return 0;
}
```

- **Output:**

```
Deleting value 5.
```

>---<