ASSIGNMENT

Submitted By: Muhammad Ahmad Mustafa

Submitted To: Miss Irsha Qureshi

Reg No: 2023-BS-AI-016

Department: BS-AI _ (A)

Doubly Linked List

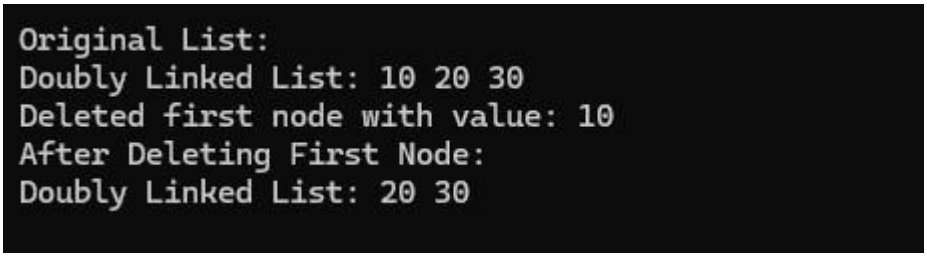
Q: Write a program to delete the first node in a doubly linked list.

```
#include <iostream>

using namespace std;


class Node {

public:

   int data;

   Node* prev;

   Node* next;


   Node(int value) {

data = value;

prev = nullptr;

next = nullptr;

   }

};


class DoublyLinkedList {

private:

   Node* head;


public:

   DoublyLinkedList() {

head = nullptr;

   }


   // Function to insert a node at the end

void insertAtEnd(int value) {

      Node* newNode = new
Node(value);
```

```
        if (head == nullptr) {
head = newNode;
            return;
        }

        Node* temp = head;
while (temp->next != nullptr) {
temp = temp->next;
        }

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

    // Function to delete the first node     void deleteFirst() {
if (head == nullptr) {            cout << "The list is empty,
nothing to delete!" << endl;
            return;
        }

        Node* temp = head;

        // If there's only one node
if (head->next == nullptr) {
head = nullptr;
        } else {            head =
head->next;            head-
>prev = nullptr;
        }

        cout << "Deleted first node with value: " << temp->data << endl;
delete temp;
```

```
    }

    // Function to display the list     void
display() {         if (head == nullptr) {
cout << "The list is empty!" << endl;
return;
        }

        Node* temp = head;
cout << "Doubly Linked List: ";
while (temp != nullptr) {
cout << temp->data << " ";
temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;

    // Insert nodes into the list
list.insertAtEnd(10);
list.insertAtEnd(20);
list.insertAtEnd(30);

    cout << "Original List:" << endl;
list.display();

    // Delete the first node     list.deleteFirst();
cout << "After Deleting First Node:" << endl;
list.display();     return 0;
}
```

Output:



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

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

class Node {
public:
   int data;
   Node* prev;
   Node* next;

   Node(int value) {
data = value;
prev = nullptr;
next = nullptr;
   }
};

class DoublyLinkedList {
private:
   Node* head;
```

```
public:
DoublyLinkedList() {
head = nullptr;
  }

  // Function to insert a node at the end
void insertAtEnd(int value) {       Node*
newNode = new Node(value);
      if (head == nullptr) {
head = newNode;
         return;
      }

      Node* temp = head;
while (temp->next != nullptr) {
temp = temp->next;
      }

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

  // Function to delete the last node     void deleteLast() {
if (head == nullptr) {           cout << "The list is empty,
nothing to delete!" << endl;
         return;
      }

      Node* temp = head;

      // If there's only one node        if (head->next == nullptr) {
cout << "Deleted last node with value: " << head->data << endl;
delete head;           head = nullptr;           return;
```

```
        }

        // Traverse to the last node
while (temp->next != nullptr) {
temp = temp->next;
        }

        // Update the second last node's `next` pointer        temp->prev-
>next = nullptr;

        cout << "Deleted last node with value: " << temp->data << endl;
delete temp;
    }

    // Function to display the list     void
display() {        if (head == nullptr) {
cout << "The list is empty!" << endl;
return;
        }

        Node* temp = head;
cout << "Doubly Linked List: ";
while (temp != nullptr) {

            cout << temp->data << " ";
temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
```

```
    // Insert nodes into the list
list.insertAtEnd(10);
list.insertAtEnd(20);
list.insertAtEnd(30);

    cout << "Original List:" << endl;
list.display();

    // Delete the last node     list.deleteLast();
cout << "After Deleting Last Node:" << endl;
    list.display();

    return 0;
}
```

Output:

```
Original List:
Doubly Linked List: 10 20 30
Deleted last node with value: 30
After Deleting Last Node:
Doubly Linked List: 10 20
```

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

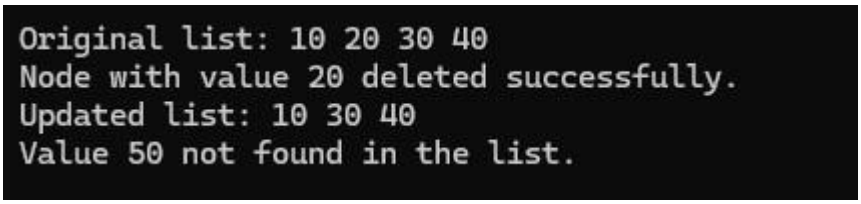
```
#include <iostream>
using namespace std;

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

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

```
};

// Function to delete a node by its value void
deleteNodeByValue(Node*& head, int value) {
   if (!head) {        cout << "The list is
empty." << endl;        return;
   }

   Node* current = head;

   // Traverse the list to find the node with the given
value     while (current && current->data != value) {
current = current->next;
   }

   // If the node is not found     if (!current) {        cout << "Value
" << value << " not found in the list." << endl;        return;
   }

   // If the node to be deleted is the
head     if (current == head) {        head =
current->next;
      if (head) {           head-
>prev = nullptr;
      }
   }
   // If the node to be deleted is in the middle or at the end
   else {        if (current->next) { // If it's not the
last node           current->next->prev = current-
>prev;
      }
      if (current->prev) { // If it's not the first node           current-
>prev->next = current->next;
```

```
        }
    }

    delete current; // Free the memory of the removed node     cout <<
"Node with value " << value << " deleted successfully." << endl;
}

// Function to display the linked list
void displayList(Node* head) {
    if (!head) {        cout << "The list is
empty." << endl;        return;
    }

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

    }
    cout << endl;
}

// Function to add a node to the end of the list
void append(Node*& head, int value) {
    Node* newNode = new Node(value);

    if (!head) {
head = newNode;
        return;
    }
```

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

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

int main() {
    Node* head = nullptr;

    // Adding nodes to the list
append(head, 10);
append(head, 20);
append(head, 30);
append(head, 40);

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

    // Deleting a node by its value
deleteNodeByValue(head, 20);

    cout << "Updated list: ";
displayList(head);

    // Trying to delete a value not in the
list     deleteNodeByValue(head, 50);

    return 0;
}
```

Output:



Q: How would you delete a node at a specific position in a doubly linked list?

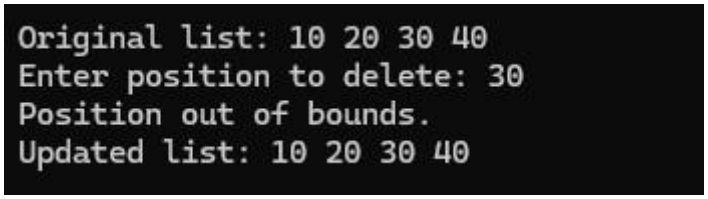
```
#include <iostream>

using namespace std;


// Node structure

struct Node {

   int data;

   Node* next;

   Node* prev;


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

};


// Function to delete a node at a specific position

void deleteAtPosition(Node*& head, int position) {

   // If the list is empty     if (head

== nullptr) {        cout << "The list

is empty.\n";        return;

   }


   // If the position is invalid     if (position <= 0) {

cout << "Invalid position. Position should be >= 1.\n";

return;

   }
```

```
   Node* current = head;
int count = 1;

   // Traverse to the node at the given position
while (current != nullptr && count < position) {
current = current->next;        count++;
   }

   // If the position is beyond the length of the
list     if (current == nullptr) {        cout <<
"Position out of bounds.\n";
      return;
   }

   // If the node to be deleted is the
head
   if (current == head) {
head = current->next;
if (head != nullptr) {
head->prev = nullptr;
      }
   }
   // If the node to be deleted is in the middle or at the end
   else {        if (current->next != nullptr) {
current->next->prev = current->prev;
      }
      if (current->prev != nullptr) {            current-
>prev->next = current->next;
      }
   }
```

```
   delete current;     cout << "Node at position " <<
position << " deleted.\n";
}

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

// Driver code
int main() {
   Node* head = new Node(10);     head-
>next = new Node(20); head->next-
>prev = head;     head->next->next =
new Node(30);     head->next->next-
>prev = head->next;     head->next-
>next->next = new Node(40);     head-
>next->next->next->prev = head->next-
>next;

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

   int position;     cout << "Enter
position to delete: ";     cin >>
position;

   deleteAtPosition(head, position);
```

```
    cout << "Updated list: ";
printList(head);

    return 0;
}
```

Output:



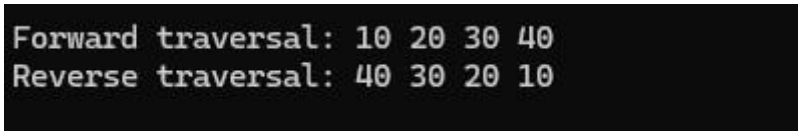
Q: After deleting a node, how will you write the forward and reverse traversal functions?

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

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

// Function to traverse the list
forward void forwardTraversal(Node*
head) {     cout << "Forward traversal:
";     while (head != nullptr) {         cout
<< head->data << " ";         head =
head->next;
    }
    cout << "\n";
```

```
}

// Function to traverse the list in
reverse void reverseTraversal(Node*
head) {    // Move to the last node    if
(head == nullptr) {        cout << "The list
is empty.\n";        return;
   }
   Node* tail = head;    while
(tail->next != nullptr) {
tail = tail->next;
   }

   // Traverse backward from the last node
   cout << "Reverse traversal:
";    while (tail != nullptr) {
cout << tail->data << " ";
tail = tail->prev;
   }
   cout << "\n";
}

// Driver code
int main() {
   Node* head = new Node(10);    head->next = new
Node(20);    head->next->prev = head;    head-
>next->next = new Node(30);    head->next->next-
>prev = head->next;    head->next->next->next =
new Node(40);    head->next->next->next->prev =
head->next->next;
```

```
   // Perform forward and reverse traversals
forwardTraversal(head);
reverseTraversal(head);

   return 0;
}
```

Output:



Circular Linked List

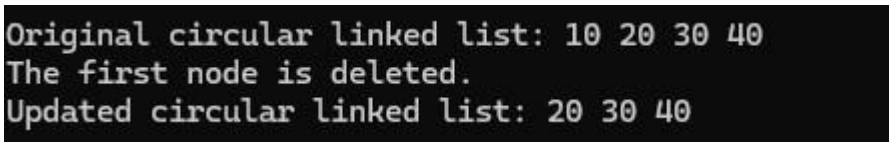
Q: Write a program to delete the first node in a circular linked list.

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

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

   Node(int value) : data(value), next(nullptr) {}
};

// Function to delete the first node in a circular linked list
void deleteFirstNode(Node*& head) {
   // If the list is empty     if (head
== nullptr) {        cout << "The list
is empty.\n";
      return;
   }

   // If the list contains only one node
if (head->next == head) {
      delete head;
head = nullptr;
      cout << "The first node is deleted. The list is now empty.\n";
return;
   }
```

```
   // For a list with multiple nodes
   Node* tail = head;

   // Find the last node
while (tail->next != head) {
      tail = tail->next;
   }

   // Update head and tail
pointers    Node* toDelete =
head;    head = head->next;
   tail->next = head;

   // Delete the first node
   delete toDelete;

   cout << "The first node is deleted.\n";
}

// Function to print the circular linked
list void printList(Node* head) {
   if (head == nullptr) {
      cout << "The list is empty.\n";
      return;
   }

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

// Driver code
int main() {
   // Creating a circular linked list: 10 -> 20 -> 30 -> 40 -> back to 10
   Node* head = new Node(10);    head-
>next = new Node(20);    head->next-
>next = new Node(30);    head->next-
>next->next = new Node(40);
   head->next->next->next->next = head; // Making it circular

   cout << "Original circular linked list: ";
printList(head);

   // Deleting the first node
   deleteFirstNode(head);

   cout << "Updated circular linked list: ";
printList(head);
```

```
    return 0;
}
```

Output:



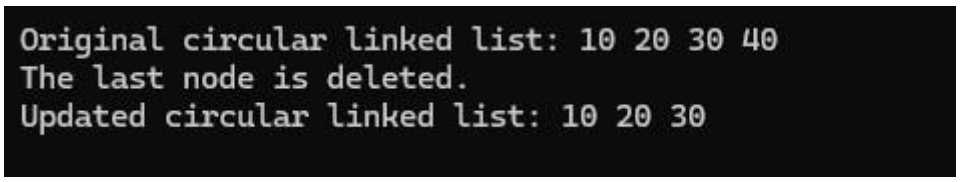
Q: How can you delete the last node in a circular linked list? Write the code.

```
#include <iostream>

using namespace std; //

Node structure struct

Node {

    int data;

    Node* next;


    Node(int value) : data(value), next(nullptr) {}

};


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

void deleteLastNode(Node*& head) {

    // If the list is empty     if (head

== nullptr) {        cout << "The list

is empty.\n";        return;

    }


    // If the list contains only one node     if (head->next == head)

{        delete head;        head = nullptr;        cout << "The last

node is deleted. The list is now empty.\n";

        return;

    }
```

```
    // For a list with multiple nodes
    Node* current = head;
    Node* prev = nullptr;

    // Traverse the list to find the last
node     while (current->next != head) {
prev = current;        current = current-
>next;
    }

    // Update the second-to-last node's next pointer to point to head     prev-
>next = head;

    // Delete the last node
delete current;

    cout << "The last node is deleted.\n";
}

// Function to print the circular linked
list void printList(Node* head) {     if
(head == nullptr) {        cout << "The list
is empty.\n";        return;
    }

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

```
// Driver code
int main() {
   // Creating a circular linked list: 10 -> 20 -> 30 -> 40 -> back to
10     Node* head = new Node(10);     head->next = new Node(20);
head->next->next = new Node(30);     head->next->next->next =
new Node(40);     head->next->next->next->next = head; //
Making it circular


   cout << "Original circular linked list: ";
printList(head);


   // Deleting the last node
deleteLastNode(head);


   cout << "Updated circular linked list: ";
printList(head);


   return 0;
}
```

Output:



Q: 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;

   Node(int value) : data(value), next(nullptr) {}
};

// Function to delete a node by its value in a circular linked list
void deleteNodeByValue(Node*& head, int value) {
   // If the list is empty     if (head
== nullptr) {        cout << "The list
is empty.\n";        return;
   }

   // If the list contains only one node     if (head->next == head && head->data ==
value) {        delete head;        head = nullptr;        cout << "The node with value " <<
value << " is deleted. The list is now empty.\n";
      return;
   }

   Node* current = head;
   Node* prev = nullptr;

   // Traverse the list to find the node with the given value
   do {        if (current->data ==
value) {
         // If the node to be deleted is the
head           if (current == head) {
prev = head;
            // Find the last node to update the next
pointer              while (prev->next != head) {
prev = prev->next;
```

```
            }
            head = current->next;
prev->next = head;
        }
else {
// If the
node to be
deleted is
not the
head
prev->next
= current-
>next;
        }

        // Delete the node          delete current;          cout <<
"The node with value " << value << " is deleted.\n";
        return;
      }

      prev = current;
current = current->next;

   } while (current != head);

   // If the value is not found     cout << "Node with value " <<
value << " not found in the list.\n";
}

// Function to print the circular linked
list void printList(Node* head) {     if
(head == nullptr) {        cout << "The list
is empty.\n";        return;
```

```
    }

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

// Driver code
int main() {
    // Creating a circular linked list: 10 -> 20 -> 30 -> 40 -> back to
10     Node* head = new Node(10);     head->next = new Node(20);
head->next->next = new Node(30);     head->next->next->next =
new Node(40);     head->next->next->next->next = head; //
Making it circular

    cout << "Original circular linked list: ";
printList(head);

    int value;     cout << "Enter value
to delete: ";     cin >> value;

    // Deleting the node by its value
deleteNodeByValue(head, value);

    cout << "Updated circular linked list: ";
printList(head);

    return 0;
}
```

Output:

```
Forward traversal: 10 20 30 40
Reverse traversal: 40 30 20 10
```

Q: How will you delete a node at a specific position in a circular linked list?

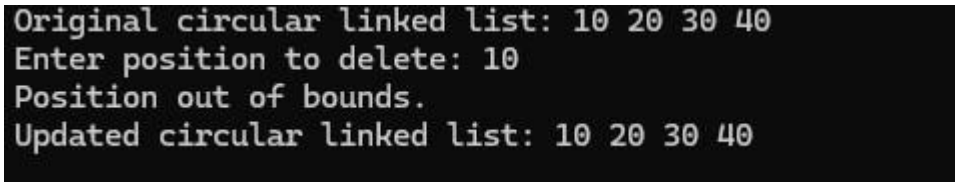
```
#include <iostream>
using namespace std;

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

    Node(int value) : data(value), next(nullptr) {}
};

// Function to delete a node at a specific position in a circular linked list
void deleteAtPosition(Node*& head, int position) {
    // If the list is empty     if (head
== nullptr) {        cout << "The list
is empty.\n";        return;
    }

    // If position is less than 1     if (position <= 0) {
cout << "Invalid position. Position must be >= 1.\n";
return;
    }

    // If the list contains only one node     if (head->next == head && position == 1) {
delete head;        head = nullptr;        cout << "The node at position " << position << " is
deleted. The list is now empty.\n";        return;
    }
```

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

    // Traverse to the node just before the position to
delete     while (current->next != head && count <
position) {        prev = current;        current = current-
>next;        count++;
    }

    // If the position is greater than the length of the
list     if (current->next == head && count < position) {
cout << "Position out of bounds.\n";
        return;
    }

    // If the node to be deleted is the
head     if (current == head) {        prev =
head;
        // Find the last node to update its next
pointer        while (prev->next != head) {
prev = prev->next;
        }
        head = current->next;        prev-
>next = head;
    }
else {
        // For any other node, just update the previous node's next pointer        prev-
>next = current->next;

    }
```

```
   // Delete the node     delete current;     cout << "The node at
position " << position << " is deleted.\n";
}

// Function to print the circular linked
list void printList(Node* head) {     if
(head == nullptr) {        cout << "The list
is empty.\n";        return;
   }

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

// Driver code
int main() {
   // Creating a circular linked list: 10 -> 20 -> 30 -> 40 -> back to
10     Node* head = new Node(10);     head->next = new Node(20);
head->next->next = new Node(30);     head->next->next->next =
new Node(40);     head->next->next->next->next = head; //
Making it circular

   cout << "Original circular linked list: ";
printList(head);

   int position;     cout << "Enter
position to delete: ";     cin >>
position;
```

```
    // Deleting the node at the specified position
deleteAtPosition(head, position);

    cout << "Updated circular linked list: ";
printList(head);

    return 0;
}
```

Output:



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

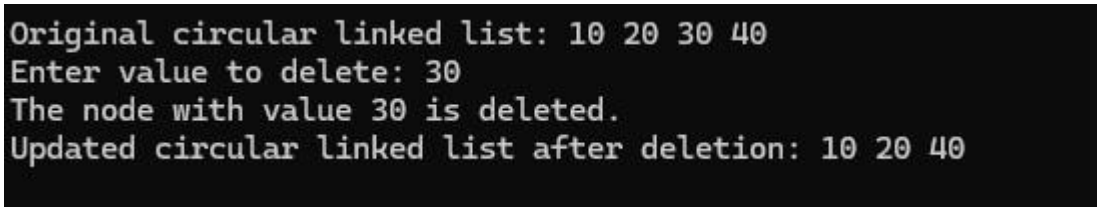
```
#include <iostream>
using namespace std;

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

    Node(int value) : data(value), next(nullptr) {}
};

// Function to delete a node by its value in a circular linked list
void deleteNodeByValue(Node*& head, int value) {
    // If the list is empty     if (head
== nullptr) {        cout << "The list
is empty.\n";        return;
```

```
    }

    // If the list contains only one node     if (head->next == head && head->data ==
value) {        delete head;        head = nullptr;        cout << "The node with value " <<
value << " is deleted. The list is now empty.\n";
        return;
    }

    Node* current = head;
    Node* prev = nullptr;

    // Traverse the list to find the node with the given value
    do {        if (current->data ==
value) {
            // If the node to be deleted is the
head            if (current == head) {
prev = head;
                // Find the last node to update the next
pointer                while (prev->next != head) {
prev = prev->next;
                }
                head = current->next;
prev->next = head;
            }
            else {
                // If the node to be deleted is not the head
prev->next = current->next;
            }

            // Delete the node            delete current;            cout <<
"The node with value " << value << " is deleted.\n";
            return;
        }
```

```
        prev = current;
current = current->next;

    } while (current != head);

    // If the value is not found     cout << "Node with value " <<
value << " not found in the list.\n";
}

// Function to print the circular linked
list void printList(Node* head) {     if
(head == nullptr) {         cout << "The list
is empty.\n";         return;
    }

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

}

// Driver code
int main() {
    // Creating a circular linked list: 10 -> 20 -> 30 -> 40 -> back to
10     Node* head = new Node(10);     head->next = new Node(20);
head->next->next = new Node(30);     head->next->next->next =
new Node(40);     head->next->next->next->next = head; //
Making it circular
```

```
    cout << "Original circular linked list: ";
printList(head);

    int value;     cout << "Enter value
to delete: ";     cin >> value;

    // Deleting the node by its value
deleteNodeByValue(head, value);

    cout << "Updated circular linked list after deletion: ";
printList(head);

    return 0;
}
```

Output:



Binary Search Tree

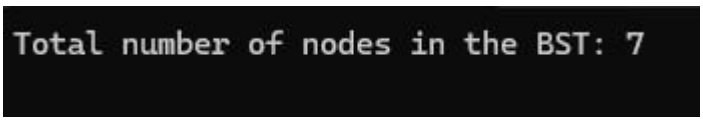
Q: Write a program to count all the nodes in a binary search tree.

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

// Definition of a TreeNode struct
TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
```

```
    // Constructor
TreeNode(int value) {
data = value;        left =
nullptr;        right =
nullptr;
    }
};

// Function to insert a node in the BST
TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {        return
new TreeNode(value);
    }

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

    return root;
}

// Function to count all the nodes in the BST int
countNodes(TreeNode* root) {
    if (root == nullptr) {
return 0;
    }
    return 1 + countNodes(root->left) + countNodes(root->right);
}
```

```
// Main function
int main() {
   TreeNode* root = nullptr;

   // Insert nodes into the BST
root = insert(root, 50);    root
= insert(root, 30);    root =
insert(root, 70);    root =
insert(root, 20);    root =
insert(root, 40);    root =
insert(root, 60);    root =
insert(root, 80);

   // Count the total number of nodes    int totalNodes =
countNodes(root);    cout << "Total number of nodes in the BST: " <<
totalNodes << endl;    return 0;
}
```

Output:



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

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

// Definition of a TreeNode
struct TreeNode {
```

```
    int data;
    TreeNode* left;
    TreeNode* right;

    // Constructor
TreeNode(int value) {
data = value;        left =
nullptr;        right =
nullptr;
    }
};

// Function to insert a node in the BST
TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {        return
new TreeNode(value); }


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

    return root;
}

// Function to search for a specific value in the BST
bool search(TreeNode* root, int value) {
    if (root == nullptr) {        return
false; // Value not found
    }
```

```
    if (root->data == value) {
return true; // Value found
    }

    if (value < root->data) {        return search(root->left, value);
// Search in the left subtree
    } else {
        return search(root->right, value); // Search in the right subtree
    }
}

// Main function
int main() {
    TreeNode* root = nullptr;

    // Insert nodes into the BST
    root = insert(root, 50);
root = insert(root, 30); root =
insert(root, 70);     root =
insert(root, 20);     root =
insert(root, 40);     root =
insert(root, 60);     root =
insert(root, 80);

    // Value to search for
int valueToSearch = 60;

    // Search for the value     if (search(root, valueToSearch)) {        cout
<< "Value " << valueToSearch << " found in the BST." << endl;
    } else {
        cout << "Value " << valueToSearch << " not found in the BST." << endl;
    }
```

```
    return 0;
}
```

Output:

```
Value 60 found in the BST.
```

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

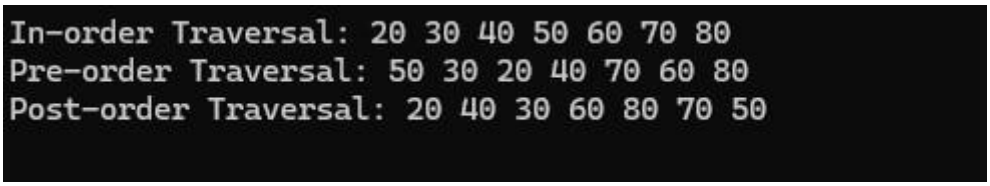
```
#include <iostream>
using namespace std;

// Definition of a TreeNode
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    // Constructor
TreeNode(int value) {
data = value;        left =
nullptr;        right =
nullptr;
    }
};

// Function to insert a node in the BST
TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {        return
new TreeNode(value);
    }
```

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

    return root;
}

// In-order Traversal (Left, Root, Right)
void inOrder(TreeNode* root) {
    if (root == nullptr) {
return;
    }
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}

// Pre-order Traversal (Root, Left, Right)
void preOrder(TreeNode* root) {
    if (root == nullptr) {
return;
    }
    cout << root->data << " ";
preOrder(root->left);     preOrder(root-
>right);
}

// Post-order Traversal (Left, Right,
Root) void postOrder(TreeNode* root) {
    if (root == nullptr) {
return;
```

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

// Main function
int main() {
    TreeNode* root = nullptr;

    // Insert nodes into the
BST     root = insert(root, 50);
root = insert(root, 30);
root = insert(root, 70);
root = insert(root, 20);
    root = insert(root, 40);
root = insert(root, 60);
root = insert(root, 80);

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

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

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

    return 0;
```

}

Output:



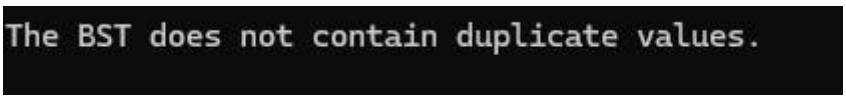
Q: Write a program to check if there are duplicate values in a binary search tree.

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

// Definition of a TreeNode
struct TreeNode {
   int data;
   TreeNode* left;
   TreeNode* right;

   // Constructor
TreeNode(int value) {
data = value;        left =
nullptr;        right =
nullptr;
   }
};

// Function to insert a node in the BST
TreeNode* insert(TreeNode* root, int value) {
   if (root == nullptr) {        return
new TreeNode(value);
   }
```

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

    // If the value is equal to root->data, do not insert (to maintain BST property)
return root;
}

// Helper function to check for
duplicates
bool checkForDuplicates(TreeNode* root, unordered_set<int>& seenValues) {
    if (root == nullptr) {        return false;
// No duplicates found
    }

    // Check if the current node's value is already in the set
if (seenValues.find(root->data) != seenValues.end()) {
return true; // Duplicate found
    }

    // Add the current node's value to the set     seenValues.insert(root-
>data);

    // Recursively check the left and right subtrees     return checkForDuplicates(root->left,
seenValues) || checkForDuplicates(root->right, seenValues);
}

// Main function
int main() {
    TreeNode* root = nullptr;
```

```
   // Insert nodes into the
BST     root = insert(root, 50);
root = insert(root, 30);
root = insert(root, 70);
root = insert(root, 20);
root = insert(root, 40);
root = insert(root, 60);
root = insert(root, 80);

   // Insert a duplicate value for testing
root = insert(root, 60);

   // Check for duplicates     unordered_set<int>
seenValues;     if (checkForDuplicates(root, seenValues))
{       cout << "The BST contains duplicate values." <<
endl;
   } else {
      cout << "The BST does not contain duplicate values." << endl;
   }

   return 0;
}
```

Output:



Q: 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;
```

```
// Definition of a TreeNode
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    // Constructor
    TreeNode(int value)
{       data = value;
left = nullptr;       right
= nullptr;

    }
};

// Function to insert a node in the BST
TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {       return
new TreeNode(value);
    }

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

    return root;
}

// Helper function to find the minimum value in a subtree
TreeNode* findMin(TreeNode* root) {
```

```
    while (root->left != nullptr) {
root = root->left;
    }
    return root;
}

// Function to delete a node from the BST
TreeNode* deleteNode(TreeNode* root, int key) {
    if (root == nullptr) {        return
root; // Node not found
    }

    if (key < root->data) {
        // Key is in the left subtree        root-
>left = deleteNode(root->left, key);
    } else if (key > root->data) {        // Key is in
the right subtree        root->right =
deleteNode(root->right, key);
    } else {
        // Node to be deleted found

        // Case 1: Node has no children (leaf node)        if
(root->left == nullptr && root->right == nullptr) {
delete root;            return nullptr;
        }

        // Case 2: Node has one child
if (root->left == nullptr) {
TreeNode* temp = root->right;
delete root;            return temp;
        } else if (root->right == nullptr)
{           TreeNode* temp = root-
```

```
>left;           delete root;
return temp;
      }

      // Case 3: Node has two children
      TreeNode* temp = findMin(root->right); // Find the in-order successor        root-
>data = temp->data;            // Replace with the successor's value        root->right =
deleteNode(root->right, temp->data); // Delete the successor
   }

   return root;
}

// In-order traversal to display the BST
void inOrder(TreeNode* root) {
   if (root == nullptr) {
return;
   }
   inOrder(root->left);
cout << root->data << " ";
inOrder(root->right);
}

// Main function
int main() {
   TreeNode* root = nullptr;

   // Insert nodes into the
BST    root = insert(root, 50);
root = insert(root, 30);
root = insert(root, 70);
root = insert(root, 20);
root = insert(root, 40);
```

```
root = insert(root, 60);
root = insert(root, 80);

   cout << "In-order Traversal of BST before deletion:
";    inOrder(root);    cout << endl;

   // Delete a leaf node    root =
deleteNode(root, 20);    cout << "After
deleting leaf node 20: ";
inOrder(root);    cout << endl;

   // Delete a node with one child    root =
deleteNode(root, 30);    cout << "After deleting
node 30 with one child: ";    inOrder(root);    cout
<< endl;

   // Delete a node with two children    root =
deleteNode(root, 50);    cout << "After deleting node
50 with two children: ";    inOrder(root);    cout <<
endl;

   return 0;
}
```

Output: