



Submitted to:

Ma'am Irsha Qureshi

Submitted by:

Hanzla Bhatti

Registration no:

2023-BS-AI-046

Department:

BS- Artificial Intelligence

Semester:

03

FINAL ASSINGMENT

Doubly 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;
    Node* next;
    Node* prev;
};

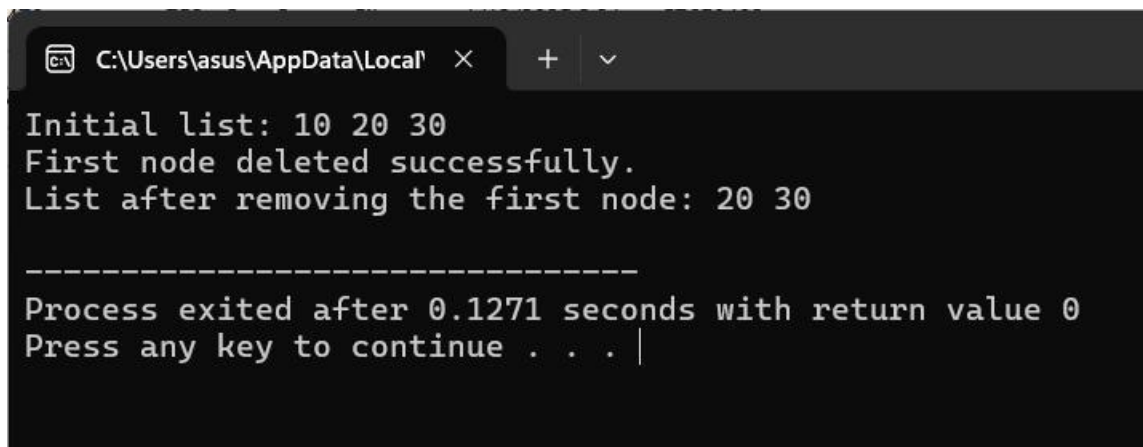
void deleteFirstNode(Node*& head) {
    if (head == nullptr) return; // List is empty
    Node* temp = head;
    if (head->next != nullptr) {
        head = head->next;
        head->prev = nullptr;
    } else {
        head = nullptr; // Only one node in the list, set head to nullptr
    }
    delete temp;
}

void forwardTraversal(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
```

```

head = head->next;
}
cout << endl;
}
int main() {
// Create a doubly linked list with 3 nodes
Node* head = new Node{1, nullptr, nullptr};
head->next = new Node{2, nullptr, head};
head->next->next = new Node{3, nullptr, head->next};
cout << "Original list: ";
forwardTraversal(head);
// Delete the first node
deleteFirstNode(head);
cout << "After deleting the first node: ";
forwardTraversal(head);
return 0;
}

```



```

C:\Users\asus\AppData\Local' x + v
Initial list: 10 20 30
First node deleted successfully.
List after removing the first node: 20 30

-----
Process exited after 0.1271 seconds with return value 0
Press any key to continue . . . |

```

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

Code:

```

#include <iostream>

using namespace std;

```

```

// Structure for a Node in a doubly linked list
struct Node {
    int value;
    Node* prev;
    Node* next;
    // Constructor to initialize the node
    Node(int val) : value(val), prev(nullptr), next(nullptr) {}
}; // Function to remove the last node from the doubly linked list
void removeLastNode(Node*& head) {
    if (!head) { // Check if the list is empty
        cout << "The list is empty, no nodes to delete." << endl;
        return;
    }
    if (head->next == nullptr) { // Only one node in the list
        delete head;
        head = nullptr;
        cout << "The last node was deleted." << endl;
        return;
    }
    Node* current = head;
    // Traverse to the last node
    while (current->next != nullptr) {
        current = current->next;
    }
    // Update the previous node's next pointer to null
    current->prev->next = nullptr;
    delete current; // Free the memory of the last node
}

```

```

cout << "The last node was deleted." << endl;
}

// Function to print the elements of the doubly linked list
void printList(Node* head) {
    Node* current = head;
    while (current != nullptr) { // Traverse the list and print values
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}

// Function to append a new node at the end of the doubly linked list
void appendToEnd(Node*& head, int value) {Node* newNode = new Node(value);
    if (!head) { // If the list is empty
        head = newNode;
        return;
    }
    Node* current = head;
    while (current->next != nullptr) { // Traverse to the last node
        current = current->next;
    }
    current->next = newNode; // Link the new node at the end
    newNode->prev = current;
}

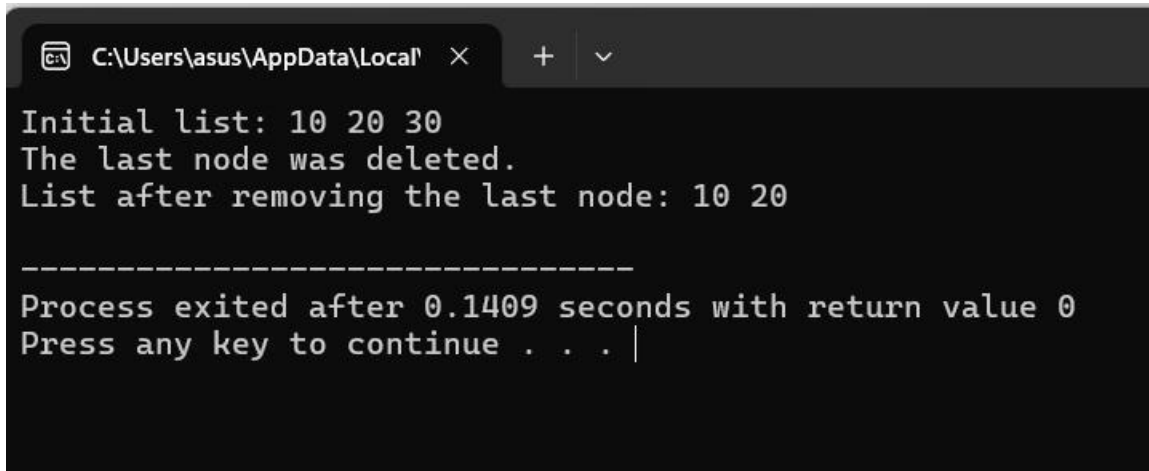
// Main function to demonstrate the doubly linked list operations
int main() {
    Node* head = nullptr;

```

```

// Add nodes to the list
appendToEnd(head, 10);
appendToEnd(head, 20);
appendToEnd(head, 30);
cout << "Initial list: ";
printList(head);
// Remove the last node
removeLastNode(head);
cout << "List after removing the last node: ";
printList(head);
return 0;}

```



```

Initial list: 10 20 30
The last node was deleted.
List after removing the last node: 10 20

-----
Process exited after 0.1409 seconds with return value 0
Press any key to continue . . . |

```

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

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 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;
}

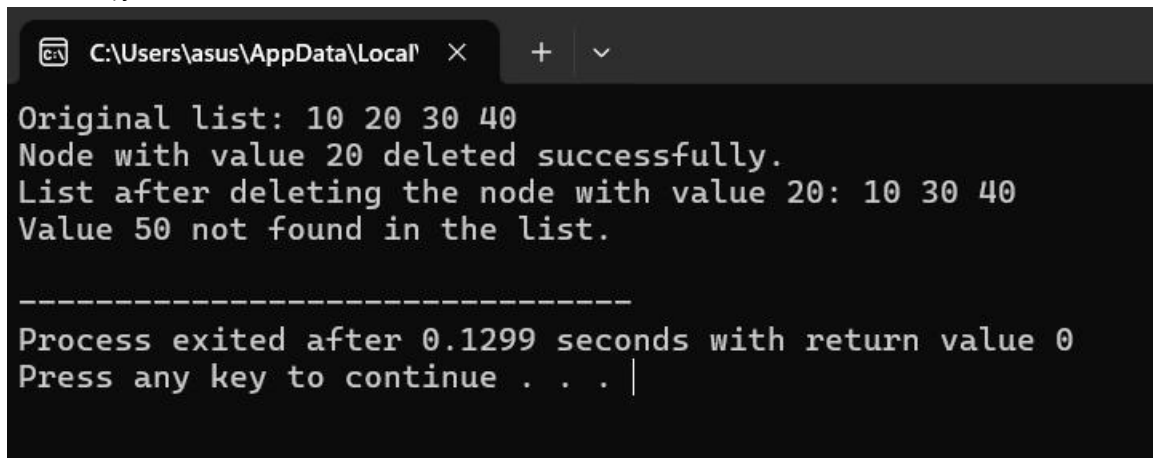
```



```

int main() {
    Node* head = nullptr;
    // Add nodes to the doubly linked list
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    appendNode(head, 40);
    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;}

```



```

C:\Users\asus\AppData\Local
Original list: 10 20 30 40
Node with value 20 deleted successfully.
List after deleting the node with value 20: 10 30 40
Value 50 not found in the list.

-----
Process exited after 0.1299 seconds with return value 0
Press any key to continue . . . |

```

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;

// Structure for a Node in a doubly linked list
struct Node {
    int value;
    Node* previous;
    Node* next;

    // Constructor to initialize a node
    Node(int val) : value(val), previous(nullptr), next(nullptr) {}
};

// Function to remove a node from a specified position in the doubly linked list
void removeNodeAtPosition(Node*& head, int position) {
    if (!head) { // Check if the list is empty
        cout << "The list is currently empty." << endl;
        return;
    }

    if (position <= 0) { // Validate the position
        cout << "Invalid position. Please provide a position greater than 0." << endl;
        return;
    }

    Node* current = head;
    int index = 1;

    // Traverse the list to locate the target position
    while (current != nullptr && index < position) {
        current = current->next;
        index++;
    }

    if (!current) { // Position exceeds the number of nodes in the list
        cout << "Position " << position << " is out of range." << endl;
    }
}

```

```

return;
}

// Handle the removal of the node
if (current == head) { // Case: Remove the first node
    head = head->next;
    if (head) {head->previous = nullptr;
}
} else if (!current->next) { // Case: Remove the last node
    current->previous->next = nullptr;
} else { // Case: Remove a node in the middle
    current->previous->next = current->next;
    current->next->previous = current->previous;
}
delete current; // Deallocate the memory
cout << "Node at position " << position << " removed successfully." << endl;
}

// Function to display the elements of the doubly linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp) { // Traverse the list and print values
        cout << temp->value << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Function to add a new node to the end of the doubly linked list
void addNodeToEnd(Node*& head, int value) {

```

```

Node* newNode = new Node(value);if (!head) { // Check if the list is empty
head = newNode;
return;
}
Node* current = head;
while (current->next) { // Traverse to the last node
current = current->next;
}
current->next = newNode; // Link the new node to the end
newNode->previous = current;
}
// Main function to test the doubly linked list operations
int main() {
Node* head = nullptr;
// Add nodes to the list
addNodeToEnd(head, 10);
addNodeToEnd(head, 20);
addNodeToEnd(head, 30);
addNodeToEnd(head, 40);
cout << "Initial list: ";
printList(head);// Remove the node at position 2
removeNodeAtPosition(head, 2);
cout << "List after removing node at position 2: ";
printList(head);
// Remove the head node
removeNodeAtPosition(head, 1);
cout << "List after removing node at position 1: ";

```

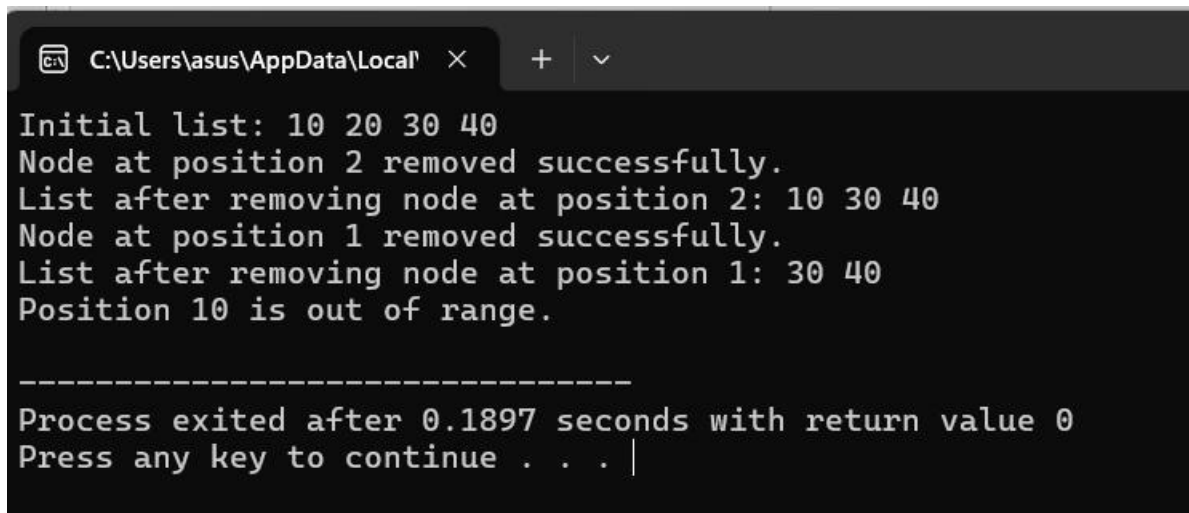
```

printList(head);

// Attempt to remove a node at an invalid position
removeNodeAtPosition(head, 10);

return 0;
}

```



```

Initial list: 10 20 30 40
Node at position 2 removed successfully.
List after removing node at position 2: 10 30 40
Node at position 1 removed successfully.
List after removing node at position 1: 30 40
Position 10 is out of range.

-----
Process exited after 0.1897 seconds with return value 0
Press any key to continue . . . |

```

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

Code:

```

#include <iostream>

using namespace std; struct Node {
int data;
Node* prev;
Node* next;
Node(int val) : data(val), prev(nullptr), next(nullptr) {}
};

// Function to perform forward traversal
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
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

```

```

void appendNode(Node*& head, int data) {Node* newNode = new Node(data);
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 a node at a specific position
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;
}

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

```



```

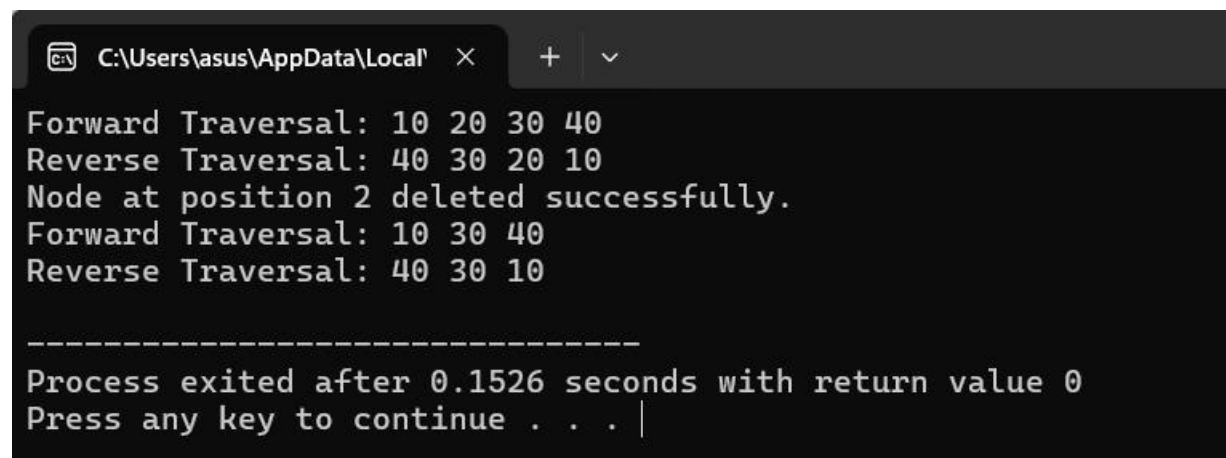
forwardTraversal(head);
reverseTraversal(head);

// Delete a node at position 2
deleteNodeAtPosition(head, 2);

// Perform forward and reverse after
forwardTraversal(head);
reverseTraversal(head);

return 0;}

```



```

C:\Users\asus\AppData\Local
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

-----
Process exited after 0.1526 seconds with return value 0
Press any key to continue . . . |

```

Circular Linked List

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

Code:

```

#include <iostream>

using namespace std;

// Structure to define a node
struct Node {
    int value;
    Node* nextNode;
};

// Function to remove the first node

```

```

void removeFirstNode(Node*& head) {
    if (!head) { // Check if the list is empty
        cout << "The list is empty. Nothing to remove." << endl;
        return;
    }
    if (head->nextNode == head) { // Single node in the list
        delete head;
        head = nullptr;
        return;
    }
    // Locate the last node in the list
    Node* lastNode = head;
    while (lastNode->nextNode != head) {
        lastNode = lastNode->nextNode;
    }
    // Update the head and adjust the last node's pointer
    Node* temp = head;
    head = head->nextNode;
    lastNode->nextNode = head;
    // Delete the old head node
    delete temp;
}

// Function to add a new node at the end of the circular linked list
void appendNode(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->value = value;
    if (!head) { // If the list is empty
        head = newNode;
    }
}

```

```

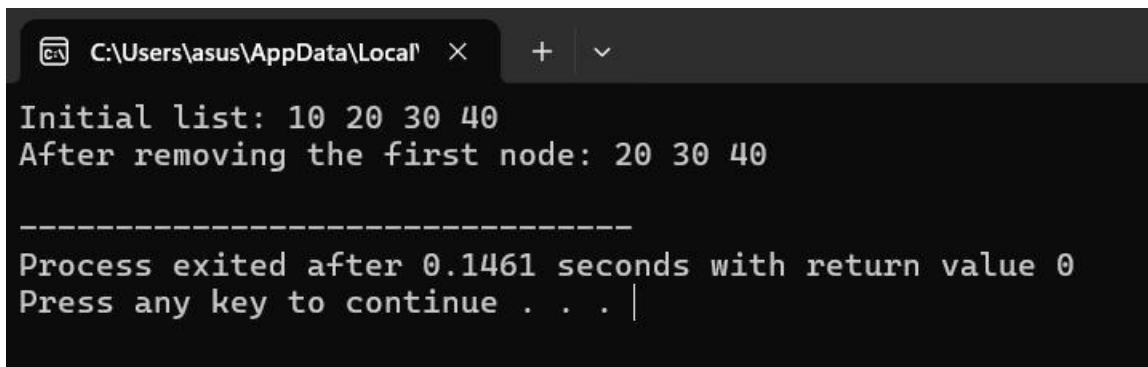
newNode->nextNode = head;
return;
}
Node* current = head;
while (current->nextNode != head) { // Traverse to the last node
current = current->nextNode;
}
current->nextNode = newNode;
newNode->nextNode = head; // Complete the circular link
}
// Function to print the contents of the circular linked list
void printList(Node* head) {
if (!head) { // Check if the list is empty
cout << "The list is empty." << endl;
return;
}
Node* current = head;do {
cout << current->value << " ";
current = current->nextNode;
} while (current != head);
cout << endl;
}
// Main function to test
int main() {
Node* head = nullptr;
// Add nodes to the list
appendNode(head, 10);

```

```

appendNode(head, 20);
appendNode(head, 30);
appendNode(head, 40);
cout << "Initial list: ";
printList(head);
// Remove the first node
removeFirstNode(head);
cout << "After removing the first node: ";
printList(head);
return 0;
}

```



```

C:\Users\asus\AppData\Local' x + v
Initial list: 10 20 30 40
After removing the first node: 20 30 40

-----
Process exited after 0.1461 seconds with return value 0
Press any key to continue . . . |

```

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

Code:

```

#include <iostream>using namespace std;

// Node structure for circular linked list
struct Node {
    int data;
    Node* next;
};

// Function to delete a node with a specific value
void deleteNodeByValue(Node*& head, int target) {

```

```

if (!head) { // Check if the list is empty
    cout << "The list is empty" << endl;
    return;
}

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

// Case 1: The list contains a single node
if (head->data == target && head->next == head) {
    delete head;
    head = nullptr;
    return;
}

// Case 2: The node to delete is the head node
if (head->data == target) {
    // Locate the last node
    while (current->next != head) {
        current = current->next;
    }
    Node* toDelete = head;
    head = head->next; // Update the head pointer
    current->next = head; // Link the last node to the new head
    delete toDelete;
    return;
}

// Case 3: The node to delete is in the middle or at the end
do {
    prev = current; current = current->next;

```

```

if (current->data == target) {
    prev->next = current->next;
    delete current;
    return;
}
} while (current != head);
cout << "Value " << target << " not found in the list." << endl;
}

void addNodeToEnd(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    if (!head) {
        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 displayList(Node* head) {
    if (!head) {
        cout << "The list is empty." << endl;
    }
}

```

```

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

int main() {
Node* head = nullptr;
// Add nodes to the list
addNodeToEnd(head, 10);
addNodeToEnd(head, 20);
addNodeToEnd(head, 30);
addNodeToEnd(head, 40);
cout << "Initial list: ";
displayList(head);
// Remove a node
deleteNodeByValue(head, 20);
cout << "After removing node with value 20: ";
displayList(head);
// delete a non-existent node
deleteNodeByValue(head, 50);
cout << "After attempting to delete node with value 50: ";
displayList(head);
return 0;

```

```
C:\Users\asus\AppData\Local' x + v
Initial list: 10 20 30 40
After removing 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

-----
Process exited after 0.1763 seconds with return value 0
Press any key to continue . . . |
}
```

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

Code:

```
#include <iostream>

using namespace std;

struct Node {
    int value;
    Node* nextNode;
};

// Function to remove a node by its value in a circular linked list
void removeNodeByValue(Node*& head, int targetValue) {
    if (head == nullptr) { // Check if the list is empty
        cout << "The list is empty. Nothing to remove." << endl;
        return;
    }

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



```
// Case 1: The list contains only one node
if (head->value == targetValue && head->nextNode == head) {
    delete head;
    head = nullptr;
    return;
}
```

```
// Case 2: The target node is the head node
if (head->value == targetValue) {
    // Find the last node
    while (current->nextNode != head) {
        current = current->nextNode;
    }
    Node* toDelete = head;
    head = head->nextNode; // Update the head pointer
    current->nextNode = head; // Adjust the last node's link
    delete toDelete;
    return;
}
```

```
// Case 3: The target node is in the middle or end of the list
do {
    prev = current;
    current = current->nextNode;

    if (current->value == targetValue) {
        prev->nextNode = current->nextNode;
```

```

        delete current;

        return;
    }
} while (current != head);

// If the target not found
cout << "Value " << targetValue << " not found in the list." << endl;
}

// Function to add a new node
void addNode(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->value = value;
    if (head == nullptr) {
        head = newNode;
        newNode->nextNode = head;
        return;
    }

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

    temp->nextNode = newNode;
    newNode->nextNode = head;
}

```

```
// Function to print the contents  
void printList(Node* head) {  
    if (head == nullptr) {  
        cout << "The list is empty." << endl;  
        return;  
    }  
}
```

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

```
// Main function for operations  
int main() {  
    Node* head = nullptr;  
  
    addNode(head, 10);  
    addNode(head, 20);  
    addNode(head, 30);  
    addNode(head, 40);  
  
    cout << "Initial list: ";  
    printList(head);  
}
```

```

// Remove a node by its value
removeNodeByValue(head, 20);

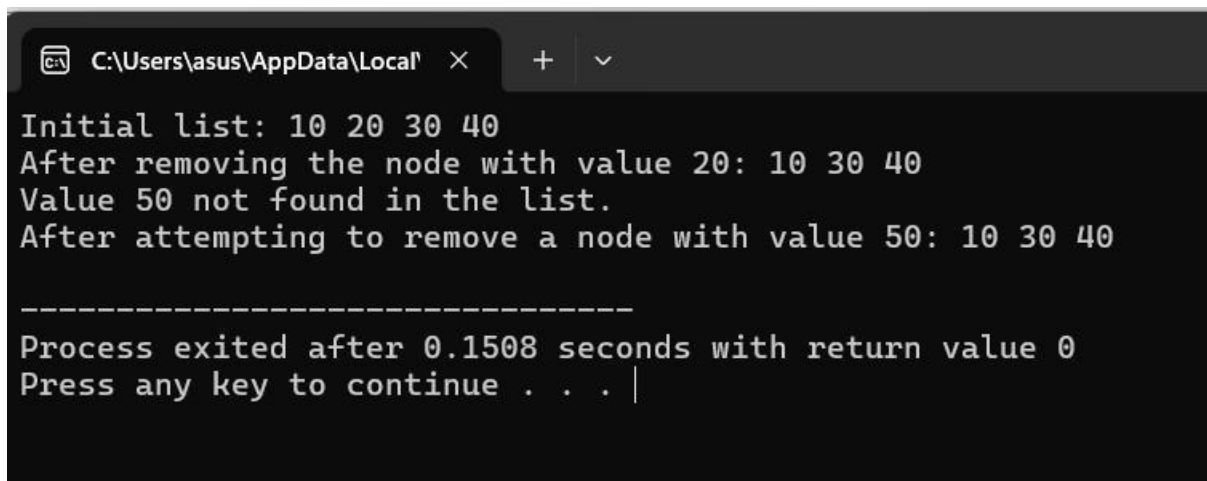
cout << "After removing the node with value 20: ";
printList(head);

// Attempt to remove a value that does not exist
removeNodeByValue(head, 50);

cout << "After attempting to remove a node with value 50: ";
printList(head);

return 0;
}

```



```

C:\Users\asus\AppData\Local
Initial list: 10 20 30 40
After removing the node with value 20: 10 30 40
Value 50 not found in the list.
After attempting to remove a node with value 50: 10 30 40

-----
Process exited after 0.1508 seconds with return value 0
Press any key to continue . . . |

```

4. 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 value;
    Node* nextNode;
};

// Function to remove a node from specific pos
void removeNodeAt(Node*& head, int position) {
    if (head == nullptr) {
        cout << "The list is empty" << endl;
        return;
    }
    // Handle removal of the head node
    if (position == 0) {
        if (head->nextNode == head) {
            delete head;
            head = nullptr;
        } else {
            Node* last = head;
            while (last->nextNode != head) {
                last = last->nextNode;
            }
            Node* toDelete = head;
            head = head->nextNode;
            last->nextNode = head;
            delete toDelete;
        }
    }
    return;
}
```

```

Node* current = head;
Node* prev = nullptr;
int currentIndex = 0;
// Traverse the list to find
while (current->nextNode != head && currentIndex < position) {
    prev = current;
    current = current->nextNode;
    currentIndex++;
}
if (current->nextNode == head && currentIndex < position) { // Out of bounds
    cout << "Position exceeds the list size." << endl;
    return;
}
//remove
prev->nextNode = current->nextNode;
delete current;
}
// Function to add node at end
void addNode(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->value = value;
    if (head == nullptr) {
        head = newNode;
        newNode->nextNode = head;
    }
    return;
}
Node* temp = head;

```

```

while (temp->nextNode != head) {temp = temp->nextNode;
}
temp->nextNode = newNode;
newNode->nextNode = head;
}
// Function to display the contents of the circular linked list
void printList(Node* head) {
if (head == nullptr) {
cout << "The list is empty." << endl;
return;
}
Node* temp = head;
do {
cout << temp->value << " ";
temp = temp->nextNode;
} while (temp != head);
cout << endl;
}
// Main function
int main() {
Node* head = nullptr;
// Add nodes to the list
addNode(head, 10);
addNode(head, 20);
addNode(head, 30);
addNode(head, 40);
cout << "Initial list: ";

```

```

printList(head);

// Remove the node at position 2
removeNodeAt(head, 2);

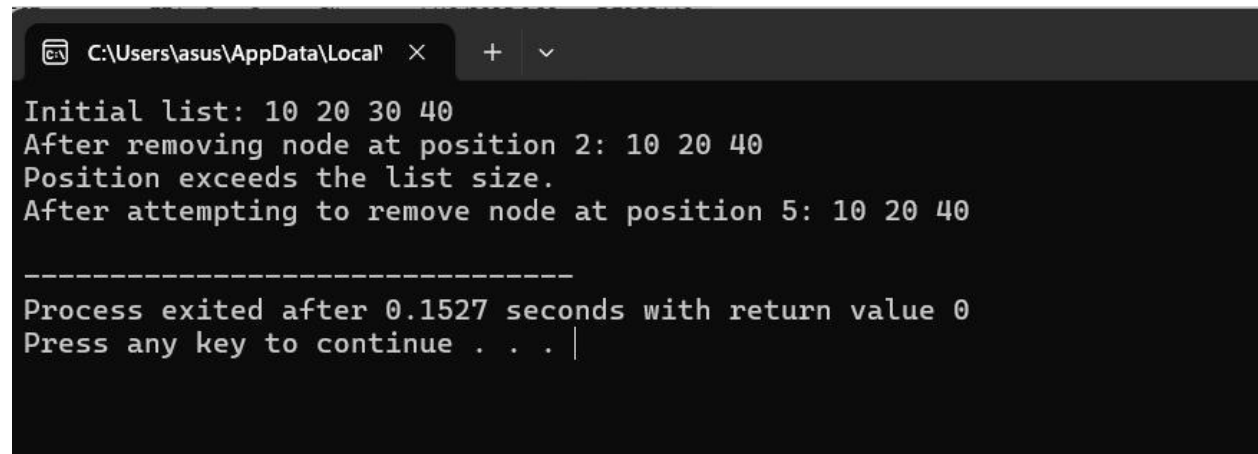
cout << "After removing node at position 2: ";
printList(head);

// Attempt to remove a node at an invalid position
removeNodeAt(head, 5);

cout << "After attempting to remove node at position 5: ";
printList(head);

return 0;
}

```



```

Initial list: 10 20 30 40
After removing node at position 2: 10 20 40
Position exceeds the list size.
After attempting to remove node at position 5: 10 20 40

-----
Process exited after 0.1527 seconds with return value 0
Press any key to continue . . . |

```

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

Code:

```

#include <iostream>

using namespace std;

// Definition of a node in a circular linked list
struct Node {
    int value;
    Node* nextNode;
};

```



```

void removeNodeAtPosition(Node*& head, int position) {
    if (!head) { // Check if the list is empty
        cout << "The list is empty. Nothing to remove." << endl;
        return;
    }
    if (position == 0) {
        // If there's only one node in the list
        if (head->nextNode == head) {
            delete head;
            head = nullptr;
        } else {
            Node* lastNode = head; while (lastNode->nextNode != head) { // Locate the last node
                lastNode = lastNode->nextNode;
            }
            Node* temp = head;
            head = head->nextNode;
            lastNode->nextNode = head;
            delete temp;
        }
        return;
    }
    Node* current = head;
    Node* previous = nullptr;
    int index = 0;
    // reach the specified position
    while (current->nextNode != head && index < position) {
        previous = current;

```

```

current = current->nextNode;
index++;
}
// If the position is out of bounds
if (current->nextNode == head && index < position) {
cout << "Invalid position: Out of bounds." << endl;
return;
}
// Adjust links and delete the node
previous->nextNode = current->nextNode;
delete current;
}
// Function to add a new node to the end
void addNode(Node*& head, int value) {
Node* newNode = new Node();
newNode->value = value;
if (!head) { // Check if the list is empty
head = newNode;
newNode->nextNode = head;
return;}
Node* current = head;
while (current->nextNode != head) {
current = current->nextNode;
}
current->nextNode = newNode;
newNode->nextNode = head; }
// Function to display the circular linked list

```

```

void printList(Node* head) {
    if (!head) { // Check if the list is empty
        cout << "The list is empty." << endl;
        return;
    }
    Node* current = head;
    do {
        cout << current->value << " ";
        current = current->nextNode;
    } while (current != head);
    cout << endl;
}

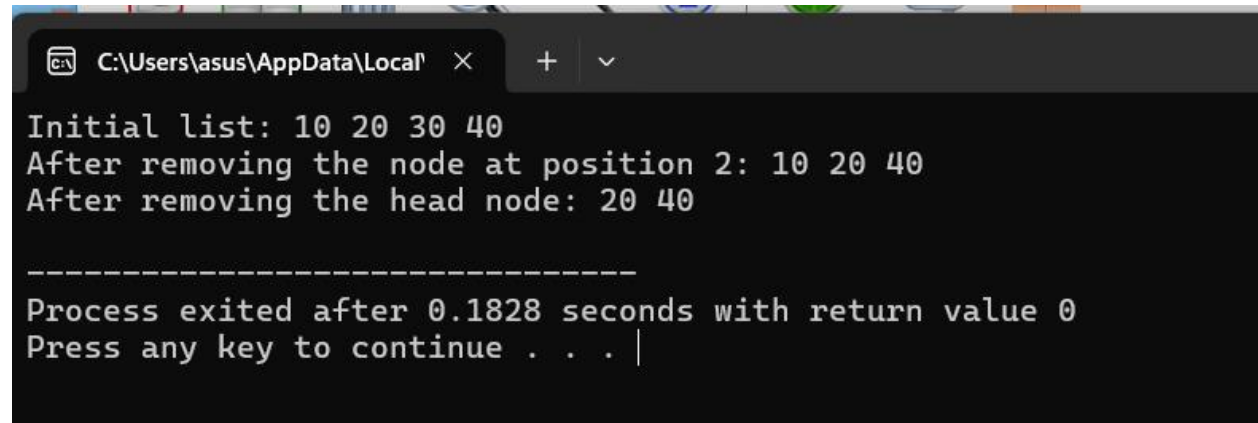
// Main function to test the functionality
int main() {
    Node* head = nullptr;
    // Adding nodes to the list
    addNode(head, 10);
    addNode(head, 20);
    addNode(head, 30);
    addNode(head, 40);
    cout << "Initial list: ";
    printList(head);

    // Remove the node at position 2
    removeNodeAtPosition(head, 2);
    cout << "After removing the node at position 2: "; printList(head);

    // Remove the node at position 0 (head node)
    removeNodeAtPosition(head, 0);
}

```

```
cout << "After removing the head node: ";  
printList(head);  
return 0;  
}
```



```
Initial list: 10 20 30 40  
After removing the node at position 2: 10 20 40  
After removing the head node: 20 40  
  
-----  
Process exited after 0.1828 seconds with return value 0  
Press any key to continue . . . |
```

Binary Search Tree

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

Code:

```
#include <iostream>  
  
using namespace std;  
  
// Structure for a node in a Binary Search Tree (BST)  
  
struct TreeNode {  
    int value;  
    TreeNode* left;  
    TreeNode* right;  
    TreeNode(int val) {  
        value = val;  
        left = right = nullptr;  
    }  
};
```

```

// Function to add a node to the Binary Search Tree
TreeNode* insertNode(TreeNode* root, int val) {
    if (root == nullptr) {
        return new TreeNode(val);
    }
    if (val < root->value) {
        root->left = insertNode(root->left, val);
    } else if (val > root->value) {
        root->right = insertNode(root->right, val);
    }
    return root;
}

// Function to calculate the total number of nodes in the Binary Search Tree
int getNodeCount(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }
    // Recursively count nodes in both the left and right subtrees, and add 1 for the
    // current node
    return 1 + getNodeCount(root->left) + getNodeCount(root->right);
}

// Function for in-order traversal of the BST and printing node values
void inorderPrint(TreeNode* root) {
    if (root != nullptr) {
        inorderPrint(root->left);
        cout << root->value << " ";
        inorderPrint(root->right);
    }
}

```

```

}
}

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

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

    cout << "In-order traversal of the Binary Search Tree: ";
    inorderPrint(root);

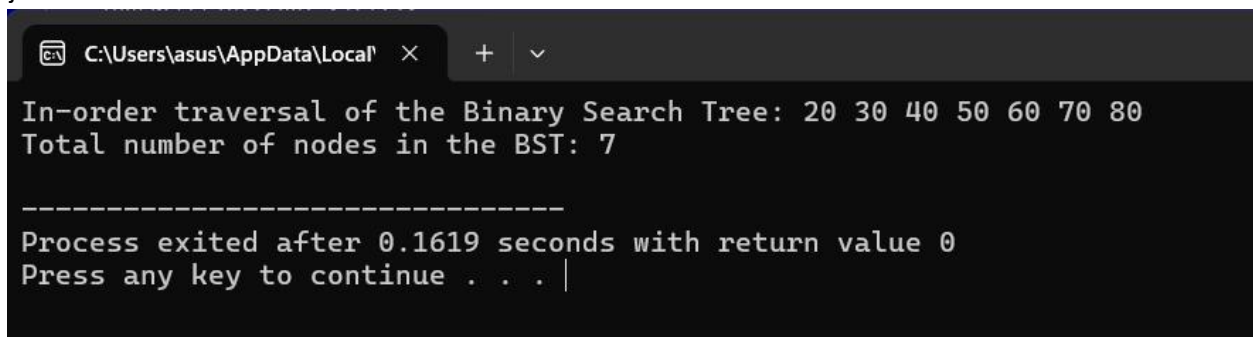
    cout << endl;

    // Calculate and print the number of nodes in the BST
    int totalNodes = getNodeCount(root);

    cout << "Total number of nodes in the BST: " << totalNodes << endl;

    return 0;
}

```



```

C:\Users\asus\AppData\Local
In-order traversal of the Binary Search Tree: 20 30 40 50 60 70 80
Total number of nodes in the BST: 7

-----
Process exited after 0.1619 seconds with return value 0
Press any key to continue . . . |

```

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;
    Node(int val) {
        data = val;
        left = NULL;
        right = NULL;
    };
};

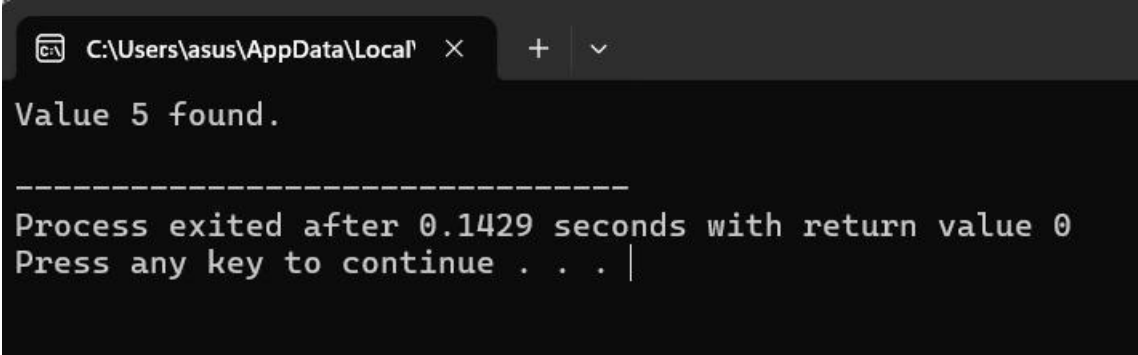
// Function to search for a value
bool search(Node* root, int key) {
    if (root == NULL) return false;
    if (root->data == key) return true;
    if (key < root->data) return search(root->left, key);
    return search(root->right, key);
}

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." << endl;
```

```

} else {
cout << "Value " << searchKey << " not found" << endl;
}
return 0;

```



```

C:\Users\asus\AppData\Local
Value 5 found.
-----
Process exited after 0.1429 seconds with return value 0
Press any key to continue . . . |

```

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

Code:

```

#include <iostream>

using namespace std;

// Define a TreeNode structure for the binary tree
struct TreeNode {int value;
TreeNode* left;
TreeNode* right;
TreeNode(int val) {
value = val;
left = nullptr;
right = nullptr;
}
};

// In-order Traversal (Left, Root, Right)
void inorderTraversal(TreeNode* root) {

```



```

if (root == nullptr) return;
inorderTraversal(root->left);
cout << root->value << " ";
inorderTraversal(root->right);
}

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

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

// Main function to test the traversal functions
int main() {
TreeNode* root = new TreeNode(10);
root->left = new TreeNode(5);
root->right = new TreeNode(15);
root->left->left = new TreeNode(3); root->left->right = new TreeNode(7);
root->right->left = new TreeNode(12);
root->right->right = new TreeNode(18);

```

```

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

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

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

return 0;
}

```

```

C:\Users\asus\AppData\Local
In-order traversal: 3 5 7 10 12 15 18
Pre-order traversal: 10 5 3 7 15 12 18
Post-order traversal: 3 7 5 12 18 15 10

-----
Process exited after 0.1313 seconds with return value 0
Press any key to continue . . . |

```

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

Show it in code.

Code:

```

#include <iostream>

using namespace std;

```

```

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);
}

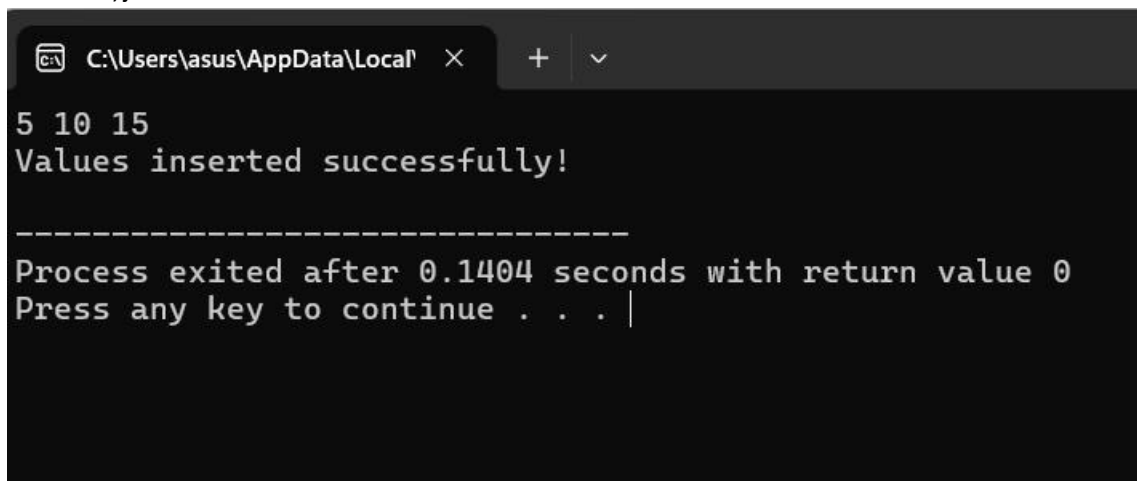
return root;
}

```

```

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;}

```



```

C:\Users\asus\AppData\Local
5
10
15
Values inserted successfully!

-----
Process exited after 0.1404 seconds with return value 0
Press any key to continue . . . |

```

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;
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;

```

```

}
C:\Users\asus\AppData\Local\
Duplicate value 10 not allowed.
-----
Process exited after 0.1727 seconds with return value 0
Press any key to continue . . . |

```

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;
// Structure representing a node in the Binary Search Tree
struct TreeNode {
    int value;TreeNode* left;
    TreeNode* right;
    TreeNode(int val) {
        value = val;
        left = nullptr;
        right = nullptr;
    }
};
// Function to find the node with the minimum value
TreeNode* findMinNode(TreeNode* root) {
    while (root && root->left != nullptr) {

```

```

root = root->left;
}
return root;
}

// Function to remove a node from the Binary Search Tree
TreeNode* removeNode(TreeNode* root, int key) {
if (root == nullptr) return root;
if (key < root->value) {
root->left = removeNode(root->left, key);
} else if (key > root->value) {
root->right = removeNode(root->right, key);
} else {
// Node to be deleted found
// Case 1: Node has no left child
if (root->left == nullptr) {
TreeNode* temp = root->right;
delete root;
return temp;
}
// Case 2: Node has no right child
else if (root->right == nullptr) {
TreeNode* temp = root->left;
delete root;
return temp;
}
// Case 3: Node has two children
TreeNode* temp = findMinNode(root->right); // Get the minimum node from the
right subtree

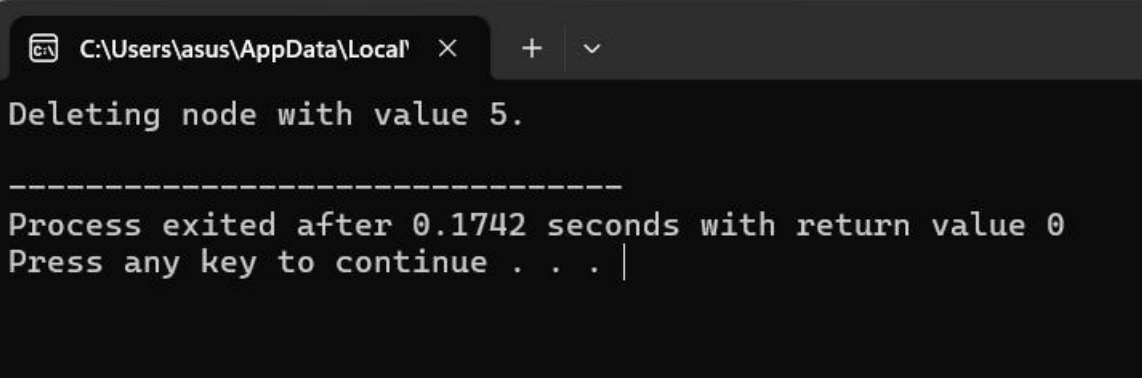
```

```

root->value = temp->value; // Replace current node's value with the in-order
successor's value
root->right = removeNode(root->right, temp->value); // Delete the in-order
successor
}
return root;
}

// Main function to demonstrate node deletion in the Binary Search Tree
int main() {
TreeNode* root = new TreeNode(10);
root->left = new TreeNode(5);
root->right = new TreeNode(15);
root->left->left = new TreeNode(3);
root->left->right = new TreeNode(7);
root->right->left = new TreeNode(12);
root->right->right = new TreeNode(18);
cout << "Deleting node with value 5." << endl;
root = removeNode(root, 5);
return 0;
}

```



```

C:\Users\asus\AppData\Local' × + ▾
Deleting node with value 5.
-----
Process exited after 0.1742 seconds with return value 0
Press any key to continue . . . |

```