Final Assignment

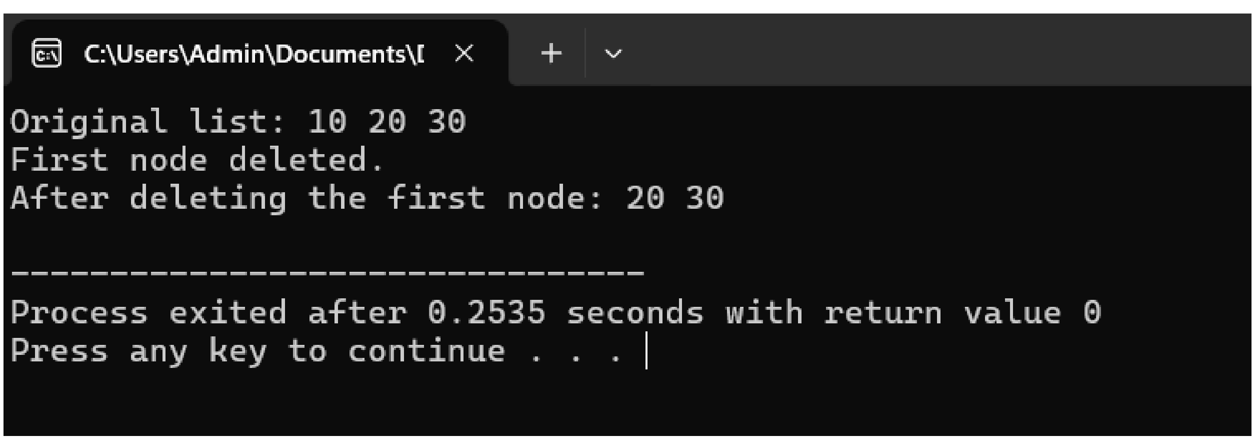
“ ﻉ DOUBLY ﻉ ”

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

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

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

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

// Function to delete the first node of the doubly linked list
void deleteFirstNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "List is already empty." << endl;
        return;
    }
    Node* temp = head;      // Store the current head
    head = head->next;      // Move head to the next node
    if (head != nullptr) {  // If there's a next node, update its prev pointer
        head->prev = nullptr;
    }
```

```
    delete temp;           // Delete the original head node
    cout << "First node deleted." << endl;
}

// Function to display the list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Function to add a node at the end of the 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) { // Traverse to the last node
        temp = temp->next;
    }

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

```
}

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

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

    // Delete the first node
    deleteFirstNode(head);

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

    return 0;
}
```



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

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

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

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

// Function to delete the first node of the doubly linked list
void deleteFirstNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "List is already empty." << endl;
        return;
    }

    Node* temp = head;      // Store the current head
    head = head->next;      // Move head to the next node
    if (head != nullptr) {  // If there's a next node, update its prev pointer
        head->prev = nullptr;
    }

    delete temp;            // Delete the original head node
    cout << "First node deleted." << endl;
}
```

```
// Function to delete the last node of the doubly linked list
void deleteLastNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "List is already empty." << endl;
        return;
    }

    if (head->next == nullptr) { // If there's only one node
        delete head;
        head = nullptr;
        cout << "Last node deleted." << endl;
        return;
    }

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

    temp->prev->next = nullptr; // Update the second last node's next pointer
    delete temp;                // Delete the last node
    cout << "Last node deleted." << endl;
}

// Function to display the list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
```

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

// Function to add a node at the end of the 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) { // Traverse to the last node
        temp = temp->next;
    }

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

int main() {
    Node* head = nullptr;

    // Add some nodes to the list
    appendNode(head, 10);
    appendNode(head, 20);
```

```cpp
    appendNode(head, 30);

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

    // Delete the first node
    deleteFirstNode(head);

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

    // Delete the last node
    deleteLastNode(head);

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

    return 0;
}
```

```
C:\Users\Admin\Documents\I
Original list: 10 20 30
Last node deleted.
After deleting the last node: 10 20

--------------------------------
Process exited after 0.2123 seconds with return value 0
Press any key to continue . . .
```

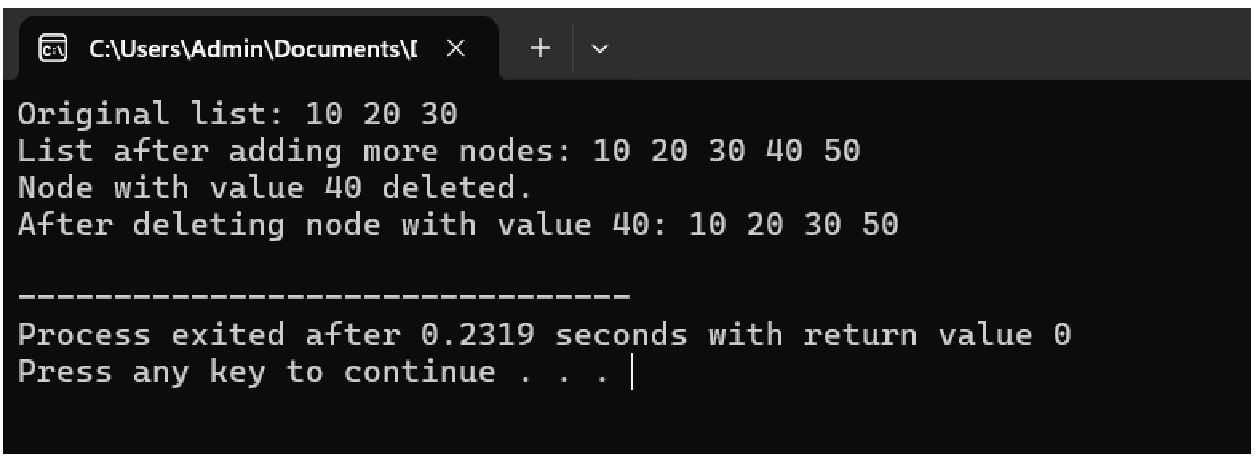
Write code to delete a node by its value in a doubly linked list.

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

// Define the Node structure
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 the doubly linked list
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // If the list is empty
        cout << "List is empty." << endl;
        return;
    }

    Node* temp = head;

    while (temp != nullptr && temp->data != value) { // Traverse to find the node
        temp = temp->next;
    }

    if (temp == nullptr) { // Value not found
        cout << "Value " << value << " not found in the list." << endl;
        return;
```

```cpp
    }

    if (temp->prev != nullptr) { // If it's not the first node
        temp->prev->next = temp->next;
    } else { // If it's the first node
        head = temp->next;
    }

    if (temp->next != nullptr) { // If it's not the last node
        temp->next->prev = temp->prev;
    }

    delete temp; // Delete the node
    cout << "Node with value " << value << " deleted." << endl;
}

// Function to display the list
void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Function to add a node at the end of the 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) { // Traverse to the last node
        temp = temp->next;
    }

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

int main() {
    Node* head = nullptr;

    // Add some nodes to the list
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);

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


    // Delete a node by its value
    appendNode(head, 40);
    appendNode(head, 50);
```

```
    cout << "List after adding more nodes: ";
    displayList(head);

    deleteNodeByValue(head, 40);

    cout << "After deleting node with value 40: ";
    displayList(head);

    return 0;
}
```



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 the Node structure
struct Node {
    int data;
    Node* prev;
```

```
    Node* next;

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

// Function to delete the first node of the doubly linked list
void deleteFirstNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "List is already empty." << endl;
        return;
    }

    Node* temp = head;      // Store the current head
    head = head->next;      // Move head to the next node
    if (head != nullptr) {  // If there's a next node, update its prev pointer
        head->prev = nullptr;
    }

    delete temp;            // Delete the original head node
    cout << "First node deleted." << endl;
}

// Function to delete the last node of the doubly linked list
void deleteLastNode(Node*& head) {
    if (head == nullptr) { // If the list is empty
        cout << "List is already empty." << endl;
        return;
    }
```

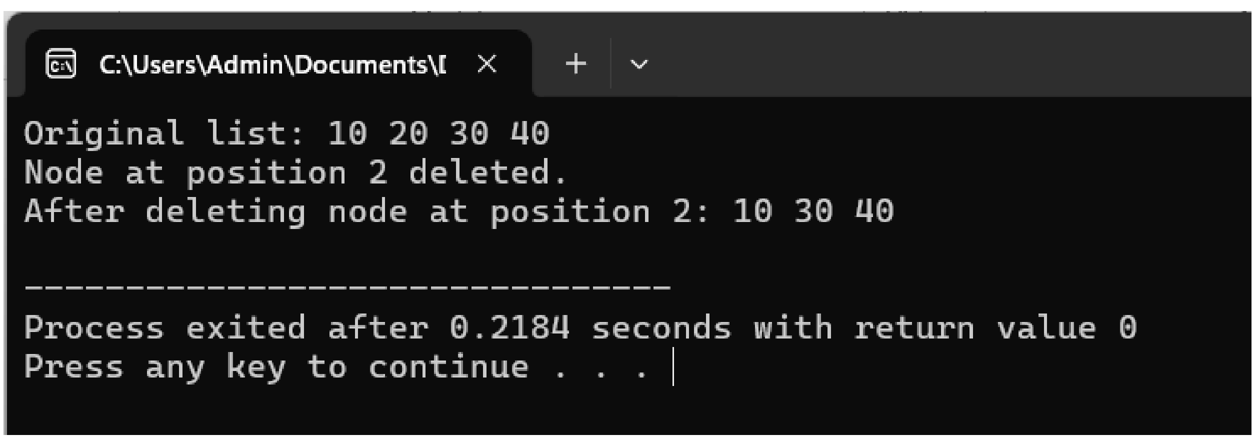
```
    if (head->next == nullptr) { // If there's only one node
        delete head;
        head = nullptr;
        cout << "Last node deleted." << endl;
        return;
    }

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

    temp->prev->next = nullptr; // Update the second last node's next pointer
    delete temp;              // Delete the last node
    cout << "Last node deleted." << endl;
}

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

    Node* temp = head;

    while (temp != nullptr && temp->data != value) { // Traverse to find the node
        temp = temp->next;
    }
```

```
    if (temp == nullptr) { // Value not found
        cout << "Value " << value << " not found in the list." << endl;
        return;
    }

    if (temp->prev != nullptr) { // If it's not the first node
        temp->prev->next = temp->next;
    } else { // If it's the first node
        head = temp->next;
    }

    if (temp->next != nullptr) { // If it's not the last node
        temp->next->prev = temp->prev;
    }

    delete temp; // Delete the node
    cout << "Node with value " << value << " deleted." << endl;
}

// 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 << "List is empty." << endl;
        return;
    }

    if (position < 1) { // Invalid position
        cout << "Invalid position." << endl;
```

```
        return;
    }

    Node* temp = head;

    for (int i = 1; temp != nullptr && i < position; i++) { // Traverse to the specified position
        temp = temp->next;
    }

    if (temp == nullptr) { // Position is out of bounds
        cout << "Position " << position << " is out of bounds." << endl;
        return;
    }

    if (temp->prev != nullptr) { // If it's not the first node
        temp->prev->next = temp->next;
    } else { // If it's the first node
        head = temp->next;
    }

    if (temp->next != nullptr) { // If it's not the last node
        temp->next->prev = temp->prev;
    }

    delete temp; // Delete the node
    cout << "Node at position " << position << " deleted." << endl;
}

// Function to display the list
```

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

// Function to add a node at the end of the 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) { // Traverse to the last node
        temp = temp->next;
    }

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

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

```
    // Add some nodes to the list
    appendNode(head, 10);
    appendNode(head, 20);
    appendNode(head, 30);
    appendNode(head, 40);

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

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

    cout << "After deleting node at position 2: ";
    displayList(head);

    return 0;
}
```



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

To implement **forward** and **reverse** traversal after deleting a node in a doubly linked list, you can follow these approaches. Both functions are simple loops that iterate through the nodes of the list in the desired direction.

1. Forward Traversal

Start from the head of the list and move forward using the next pointer until you reach the end of the list.

```
void forwardTraversal(Node* head) {
    Node* temp = head; // Start from the head
    while (temp != nullptr) { // Traverse until the end
        std::cout << temp->data << " "; // Print data of the current node
        temp = temp->next; // Move to the next node
    }
    std::cout << std::endl;
}
```

2. Reverse Traversal

Start from the tail of the list (after ensuring you have a reference to it) and move backward using the prev pointer until you reach the beginning of the list.

```
void reverseTraversal(Node* tail) {
    Node* temp = tail; // Start from the tail
    while (temp != nullptr) { // Traverse until the start
        std::cout << temp->data << " "; // Print data of the current node
        temp = temp->prev; // Move to the previous node
    }
    std::cout << std::endl;
}
```

“꒰CIRCULAR꒱”

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 create a new node

Node* createNode(int data) {

    Node* newNode = new Node();

    newNode->data = data;

    newNode->next = nullptr;

    return newNode;

}


// Function to delete the first node

void deleteFirstNode(Node*& head) {

    if (head == nullptr) {

        cout << "List is empty, nothing to delete." << endl;

        return;

    }


    // If the list has only one node

    if (head->next == head) {
```

```
        delete head;
        head = nullptr;
        return;
    }

    // If the list has more than one node
    Node* temp = head;
    Node* tail = head;

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

    // Update the head to the next node and adjust the last node's next pointer
    head = head->next;
    tail->next = head;

    // Delete the old head
    delete temp;
}

// Function to display the list
void displayList(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() {
    // Create a circular linked list
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = head; // Complete the circular link

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

    // Delete the first node
    deleteFirstNode(head);

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

    return 0;
}
```

```
C:\Users\Admin\Documents\[

Original list: 1 2 3 4
After deleting the first node: 2 3 4

--------------------------------
Process exited after 0.3005 seconds with return value 0
Press any key to continue . . .
```

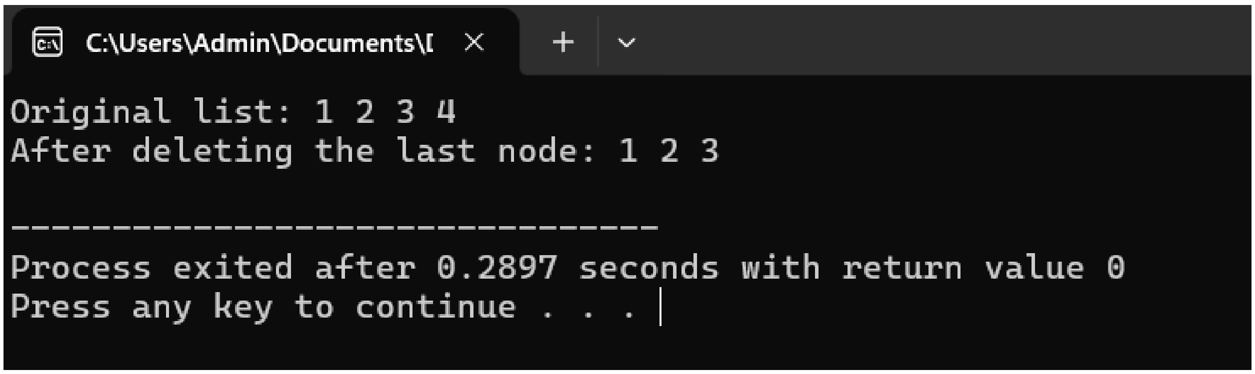
How can you delete the last node in a circular linked list? Write the code.

```
#include <iostream>

using namespace std;


// Node structure

struct Node {

    int data;

    Node* next;

};


// Function to create a new node

Node* createNode(int data) {

    Node* newNode = new Node();

    newNode->data = data;

    newNode->next = nullptr;

    return newNode;

}


// Function to delete the last node

void deleteLastNode(Node*& head) {

    if (head == nullptr) { // Case: Empty list
```

```
        cout << "List is empty, nothing to delete." << endl;
        return;
    }

    // Case: Only one node in the list
    if (head->next == head) {
        delete head;
        head = nullptr;
        return;
    }

    // Case: More than one node
    Node* temp = head;
    Node* prev = nullptr;

    // Traverse to the last node
    while (temp->next != head) {
        prev = temp;       // Keep track of the previous node
        temp = temp->next; // Move to the next node
    }

    // Update the previous node's next pointer to the head
    prev->next = head;

    // Delete the last node
    delete temp;
}

// Function to display the list
```

```
void displayList(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() {
    // Create a circular linked list
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = head; // Complete the circular link

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

    // Delete the last node
    deleteLastNode(head);
```

```
    cout << "After deleting the last node: ";
    displayList(head);

    return 0;
}
```



```
Original list: 1 2 3 4
After deleting the last node: 1 2 3

--------------------------------
Process exited after 0.2897 seconds with return value 0
Press any key to continue . . .
```

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

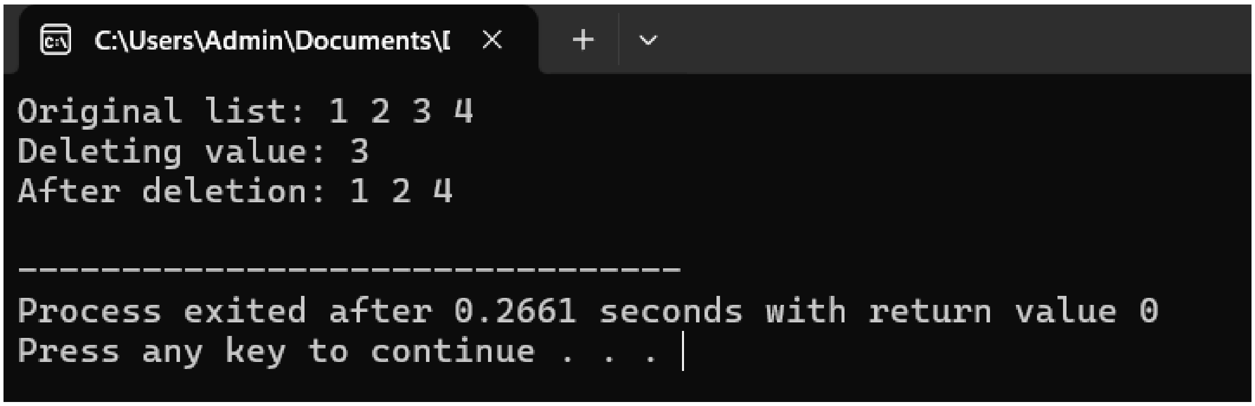
```
#include <iostream>
using namespace std;

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

// Function to delete a node by its value
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // Case: Empty list
        cout << "List is empty, nothing to delete." << endl;
        return;
```

```
}

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

// Case: Single node list
if (head->next == head) {
   if (head->data == value) { // If the single node contains the value
      delete head;
      head = nullptr;
   } else {
      cout << "Value not found in the list." << endl;
   }
   return;
}

// Case: Value is in the first node
if (head->data == value) {
   // Find the last node to update its next pointer
   Node* tail = head;
   while (tail->next != head) {
      tail = tail->next;
   }
   // Update the head and tail pointers
   Node* temp = head;
   head = head->next;
   tail->next = head;
   delete temp;
   return;
```

```
    }

    // Case: Value is in a node other than the first
    do {
        prev = current;
        current = current->next;
        if (current->data == value) {
            prev->next = current->next;
            delete current;
            return;
        }
    } while (current != head);

    // Case: Value not found
    cout << "Value not found in the list." << endl;
}

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

// Function to display the list
void displayList(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() {
    // Create a circular linked list
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = head; // Complete the circular link

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

    // Delete a node by its value
    int valueToDelete = 3;
    cout << "Deleting value: " << valueToDelete << endl;
    deleteNodeByValue(head, valueToDelete);

    cout << "After deletion: ";
```

```
    displayList(head);

    return 0;
}
```



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

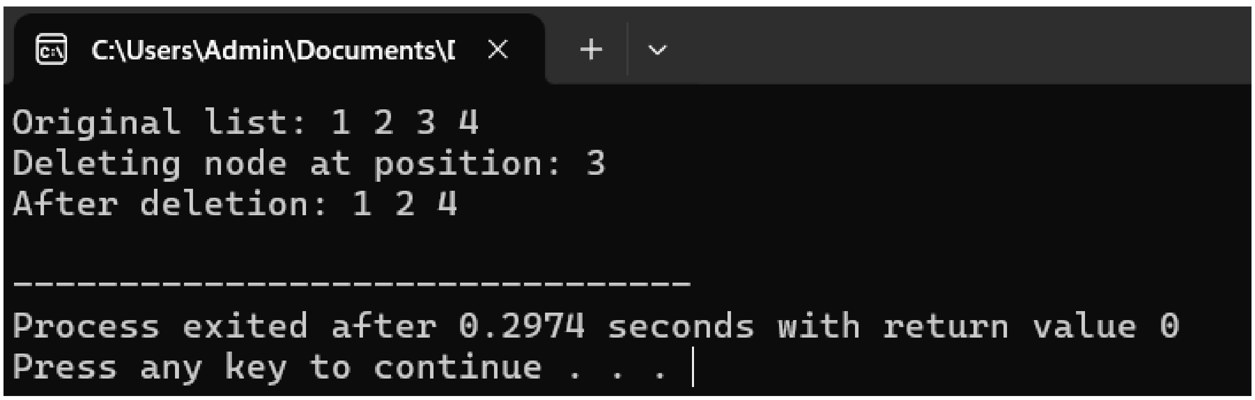
```
#include <iostream>
using namespace std;

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

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
```

```
    return newNode;
}

// Function to delete a node at a specific position
void deleteNodeAtPosition(Node*& head, int position) {
    if (head == nullptr) { // Case: Empty list
        cout << "List is empty, nothing to delete." << endl;
        return;
    }

    int length = 0;
    Node* temp = head;
    do {
        length++;
        temp = temp->next;
    } while (temp != head);

    if (position < 1 || position > length) { // Invalid position
        cout << "Invalid position. Position must be between 1 and " << length << "." << endl;
        return;
    }

    // Case: Deleting the first node
    if (position == 1) {
        Node* tail = head;
        while (tail->next != head) { // Find the last node
            tail = tail->next;
        }
```

```
        // If only one node exists
        if (head->next == head) {
            delete head;
            head = nullptr;
            return;
        }

        // Update the head and delete the old head
        Node* oldHead = head;
        head = head->next;
        tail->next = head;
        delete oldHead;
        return;
    }

    // Case: Deleting a node at a specific position (not the first node)
    Node* current = head;
    Node* prev = nullptr;

    for (int i = 1; i < position; i++) {
        prev = current;
        current = current->next;
    }

    // Remove the current node
    prev->next = current->next;
    delete current;
}
```

```
// Function to display the list
void displayList(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() {
    // Create a circular linked list
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = head; // Complete the circular link

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

    // Delete node at a specific position
    int position = 3; // For example, delete the 3rd node
```

```
    cout << "Deleting node at position: " << position << endl;
    deleteNodeAtPosition(head, position);

    cout << "After deletion: ";
    displayList(head);

    return 0;
}
```



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

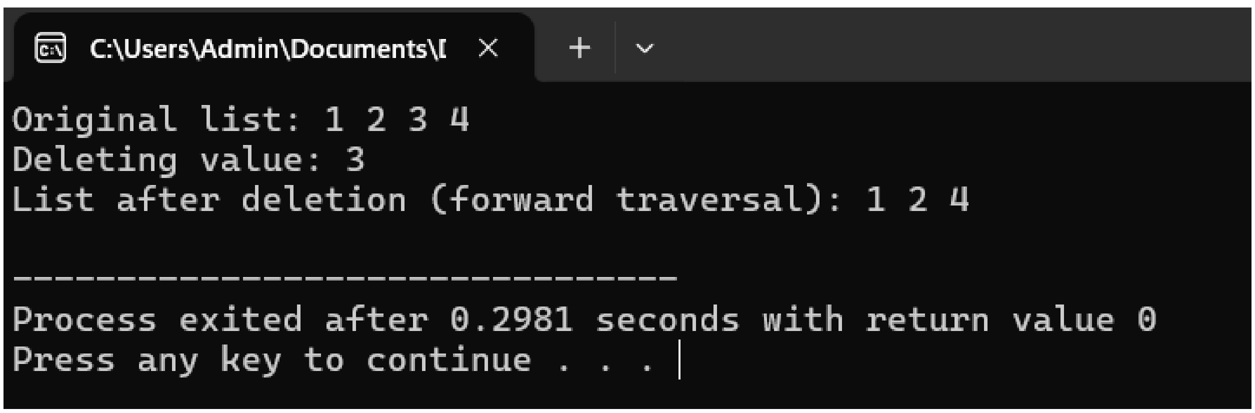
```
#include <iostream>
using namespace std;

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

// Function to create a new node
Node* createNode(int data) {
```

```
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to delete a node by its value
void deleteNodeByValue(Node*& head, int value) {
    if (head == nullptr) { // Case: Empty list
        cout << "List is empty, nothing to delete." << endl;
        return;
    }

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

    // Case: Single node list
    if (head->next == head) {
        if (head->data == value) { // If the single node contains the value
            delete head;
            head = nullptr;
        } else {
            cout << "Value not found in the list." << endl;
        }
        return;
    }

    // Case: Value is in the first node
    if (head->data == value) {
```

```
        // Find the last node to update its next pointer
        Node* tail = head;
        while (tail->next != head) {
            tail = tail->next;
        }

        // Update the head and tail pointers
        Node* temp = head;
        head = head->next;
        tail->next = head;
        delete temp;
        return;
    }

    // Case: Value is in a node other than the first
    do {
        prev = current;
        current = current->next;
        if (current->data == value) {
            prev->next = current->next;
            delete current;
            return;
        }
    } while (current != head);

    // Case: Value not found
    cout << "Value not found in the list." << endl;
}
```

```cpp
// Function to display the list (forward traversal)
void displayList(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() {
    // Create a circular linked list
    Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = head; // Complete the circular link

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

    // Delete a node by its value
    int valueToDelete = 3; // Change this value to test other cases
```

```
    cout << "Deleting value: " << valueToDelete << endl;
    deleteNodeByValue(head, valueToDelete);

    // Display the list after deletion
    cout << "List after deletion (forward traversal): ";
    displayList(head);

    return 0;
}
```



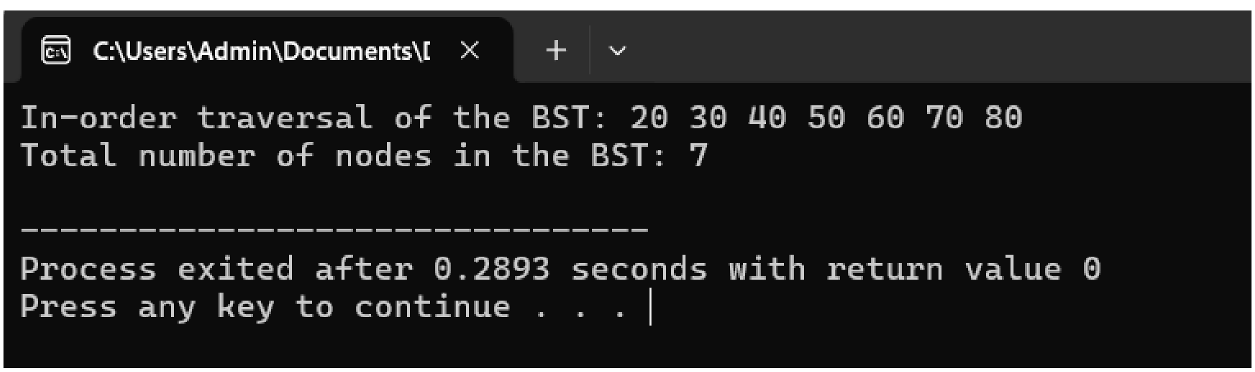
“꒰BST꒱”

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

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

// Structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

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

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

```
    return root;
}

// Function to count the nodes in the BST
int countNodes(Node* root) {
    if (root == nullptr) {
        return 0;
    }

    // Count the node and recursively count in left and right subtrees
    return 1 + countNodes(root->left) + countNodes(root->right);
}

// Function to display the tree in-order (for verification)
void inOrderTraversal(Node* root) {
    if (root == nullptr) return;
    inOrderTraversal(root->left);
    cout << root->data << " ";
    inOrderTraversal(root->right);
}

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

    // Insert nodes in the BST
    root = insert(root, 50);
    insert(root, 30);
```

```
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    cout << "In-order traversal of the BST: ";
    inOrderTraversal(root);
    cout << endl;

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

    return 0;
}
```



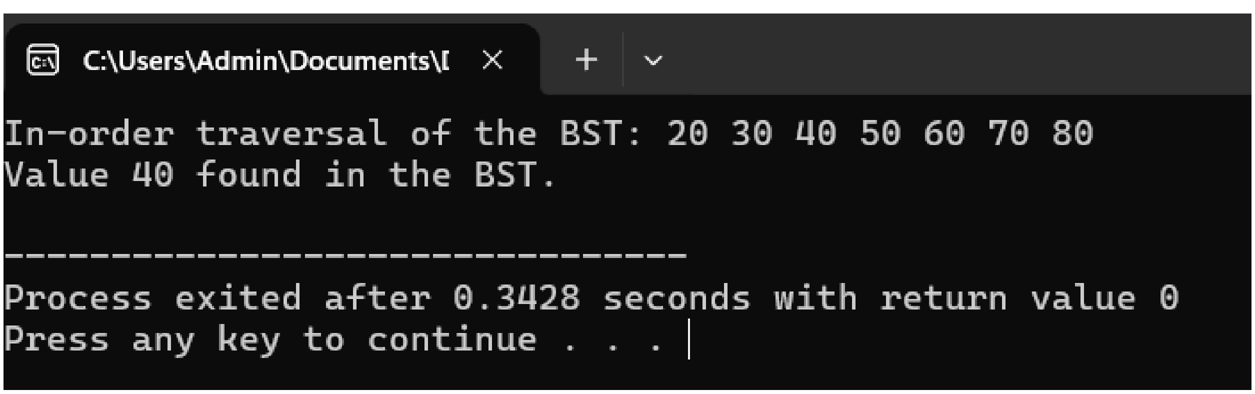
How can you search for a specific value in a binary search tree? Write the code.

```
#include <iostream>

using namespace std;
```

```
// Structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

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

```
    // Base case: root is null or the value is present at the root
    if (root == nullptr || root->data == value) {
        return root;
    }

    // If value is greater, search in the right subtree
    if (value > root->data) {
        return search(root->right, value);
    }

    // Otherwise, search in the left subtree
    return search(root->left, value);
}

// Function to display the tree in-order (for verification)
void inOrderTraversal(Node* root) {
    if (root == nullptr) return;
    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);
    insert(root, 30);
```

```
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    cout << "In-order traversal of the BST: ";
    inOrderTraversal(root);
    cout << endl;

    // Search for a value
    int valueToSearch = 40;
    Node* result = search(root, valueToSearch);

    if (result != nullptr) {
        cout << "Value " << valueToSearch << " found in the BST." << endl;
    } else {
        cout << "Value " << valueToSearch << " not found in the BST." << endl;
    }
}
```

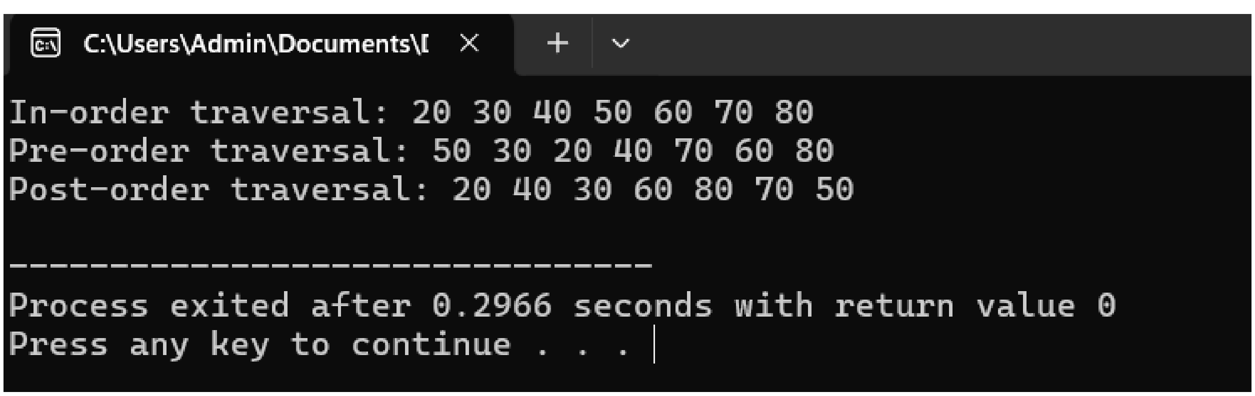


Write code to traverse a binary search tree in in-order, pre-order, and post order.

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

// Structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

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

    inOrderTraversal(root->left);  // Traverse left subtree
    cout << root->data << " ";     // Visit root
    inOrderTraversal(root->right); // Traverse right subtree
}

// Pre-Order Traversal: Root, Left, Right
void preOrderTraversal(Node* root) {
    if (root == nullptr) return;

    cout << root->data << " ";     // Visit root
    preOrderTraversal(root->left); // Traverse left subtree
    preOrderTraversal(root->right);// Traverse right subtree
}

// Post-Order Traversal: Left, Right, Root
void postOrderTraversal(Node* root) {
    if (root == nullptr) return;

    postOrderTraversal(root->left);  // Traverse left subtree
    postOrderTraversal(root->right); // Traverse right subtree
    cout << root->data << " ";       // Visit root
}
```

```
// Main function
int main() {
   Node* root = nullptr;
   // Insert nodes into the BST
   root = insert(root, 50);
   insert(root, 30);
   insert(root, 20);
   insert(root, 40);
   insert(root, 70);
   insert(root, 60);
   insert(root, 80);
   cout << "In-order traversal: ";
   inOrderTraversal(root);
   cout << endl;
   cout << "Pre-order traversal: ";
   preOrderTraversal(root);
   cout << endl;
   cout << "Post-order traversal: ";
   postOrderTraversal(root);
   cout << endl;
}
```



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

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

// Structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

// Function to insert a node in the BST
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else {
        root->right = insert(root->right, value);
    }
    return root;
}
```

```cpp
// Reverse In-Order Traversal: Right, Root, Left
void reverseInOrderTraversal(Node* root) {
    if (root == nullptr) return;
    reverseInOrderTraversal(root->right); // Traverse right subtree
    cout << root->data << " ";           // Visit root
    reverseInOrderTraversal(root->left);  // Traverse left subtree
}
// Main function
int main() {
    Node* root = nullptr;
    // Insert nodes into the BST
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
    cout << "Reverse In-order traversal: ";
    reverseInOrderTraversal(root);
    cout << endl;
}
```

C:\Users\Admin\Documents\I

```
Reverse In-order traversal: 80 70 60 50 40 30 20
--------------------------------
Process exited after 0.2608 seconds with return value 0
Press any key to continue . . .
```

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

```
#include <iostream>
#include <unordered_set> // For using set to track visited nodes
using namespace std;

// Structure for a tree node
struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

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

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

```
    return root;
}

// Function to check if there are duplicates in the BST using a set
bool checkDuplicates(Node* root, unordered_set<int>& nodeValues) {
    if (root == nullptr) {
        return false; // No duplicates in the empty subtree
    }

    // If the value already exists in the set, return true (duplicate found)
    if (nodeValues.find(root->data) != nodeValues.end()) {
        return true;
    }

    // Insert the current node's value into the set
    nodeValues.insert(root->data);

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

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

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

```
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // Insert a duplicate value (e.g., 40)
    insert(root, 40);

    // Set to track visited nodes
    unordered_set<int> nodeValues;

    // Check for duplicates
    if (checkDuplicates(root, nodeValues)) {
        cout << "Duplicate values found in the BST." << endl;
    } else {
        cout << "No duplicate values found in the BST." << endl;
    }

    return 0;
}
```

```
C:\Users\Admin\Documents\[
Duplicate values found in the BST.

--------------------------------
Process exited after 0.3481 seconds with return value 0
Press any key to continue . . .
```