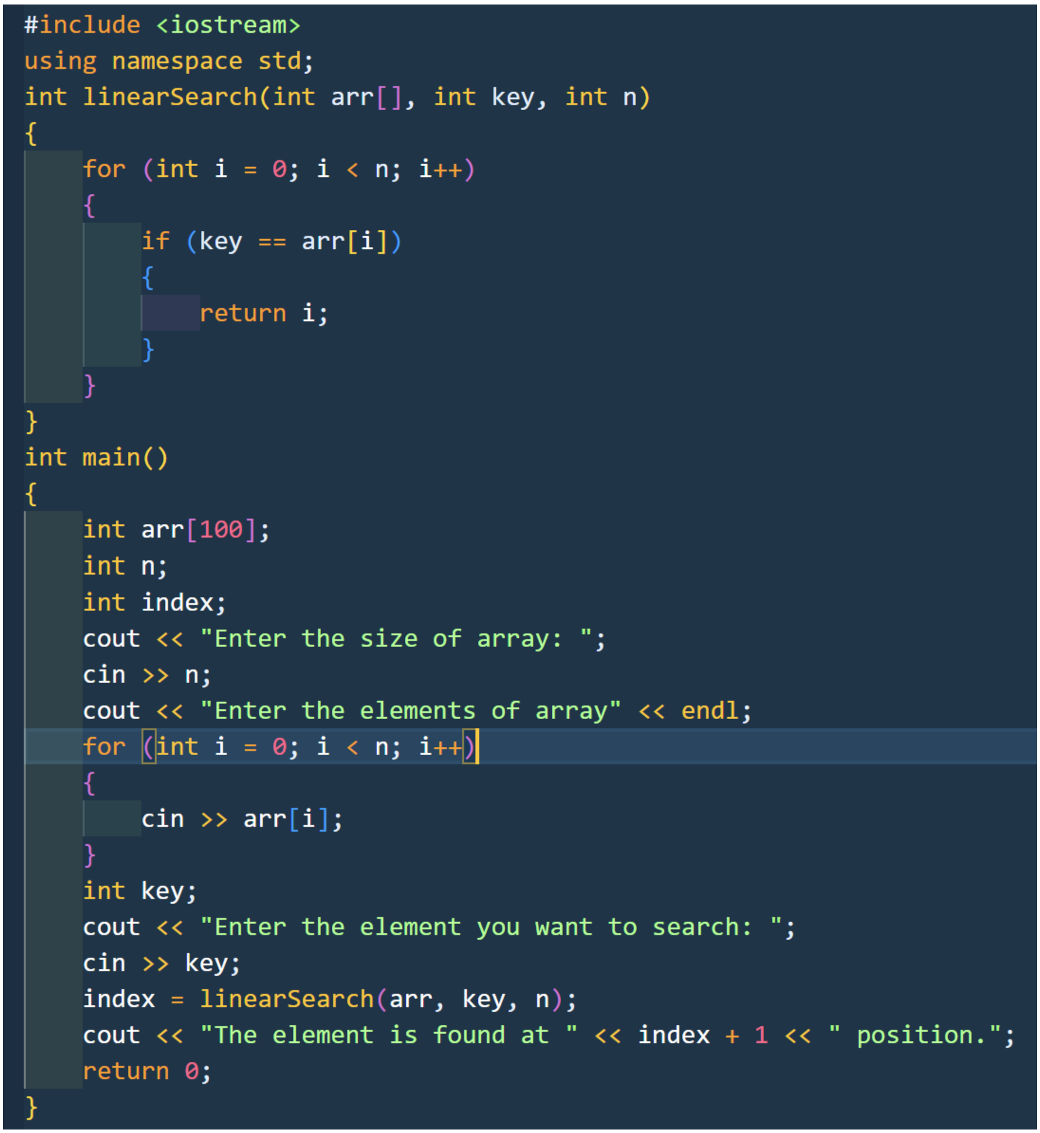
COMPUTER SCIENCE ASSIGNMENT

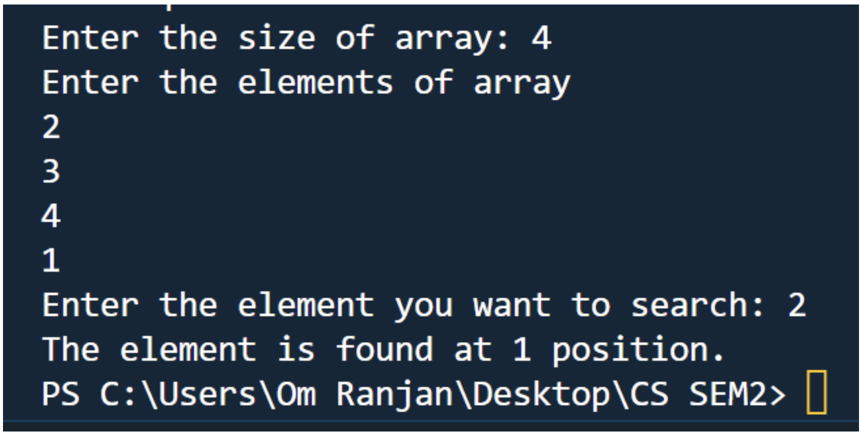
Data Structures in C++

Name: Om Ranjan
Roll No.: 22/25085
Course: B.Sc. (Prog.) Physical Science with Computer Science

1) Write a program in C++ to find a particular element in a given array location of a given element by linear search.

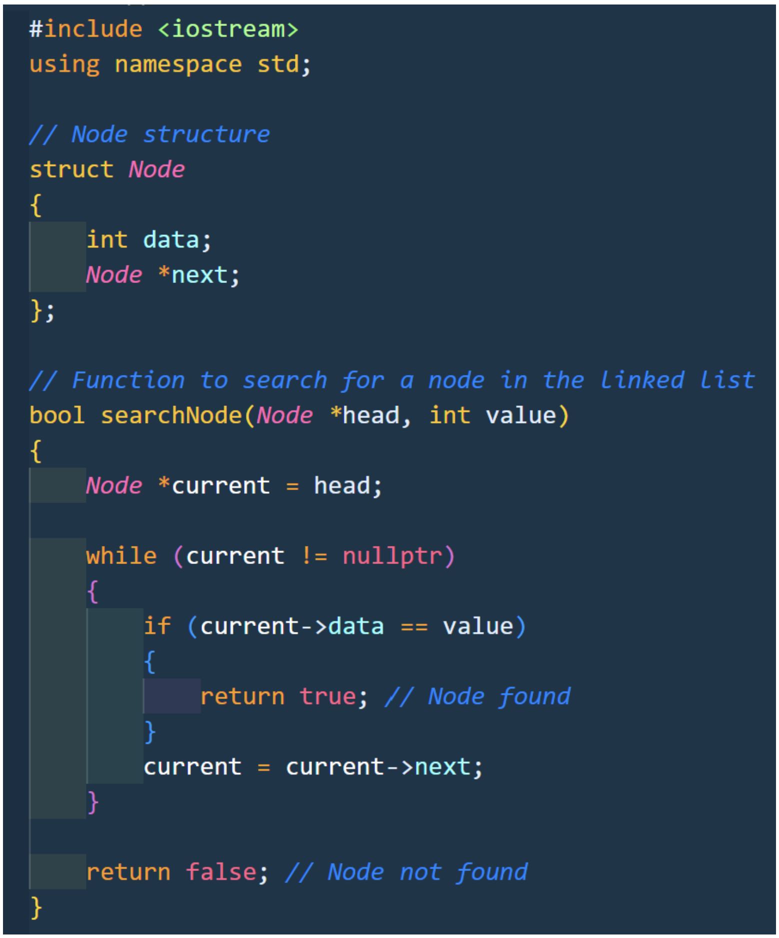


```cpp
#include <iostream>
using namespace std;
int linearSearch(int arr[], int key, int n)
{
    for (int i = 0; i < n; i++)
    {
        if (key == arr[i])
        {
            return i;
        }
    }
}
int main()
{
    int arr[100];
    int n;
    int index;
    cout << "Enter the size of array: ";
    cin >> n;
    cout << "Enter the elements of array" << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    int key;
    cout << "Enter the element you want to search: ";
    cin >> key;
    index = linearSearch(arr, key, n);
    cout << "The element is found at " << index + 1 << " position.";
    return 0;
}
```

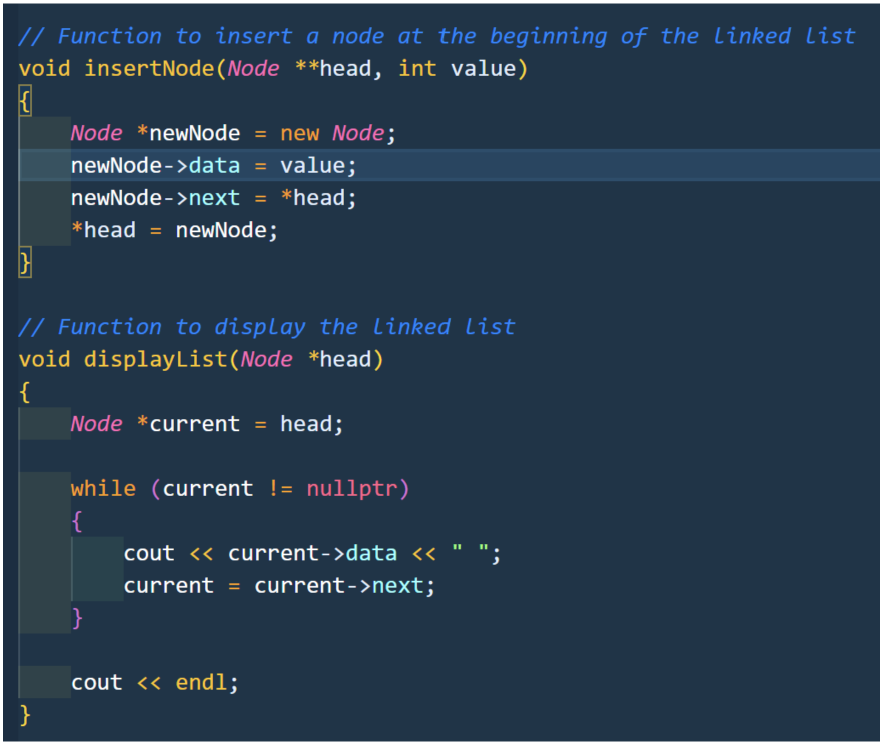


```
Enter the size of array: 4
Enter the elements of array
2
3
4
1
Enter the element you want to search: 2
The element is found at 1 position.
PS C:\Users\Om Ranjan\Desktop\CS SEM2> 
```

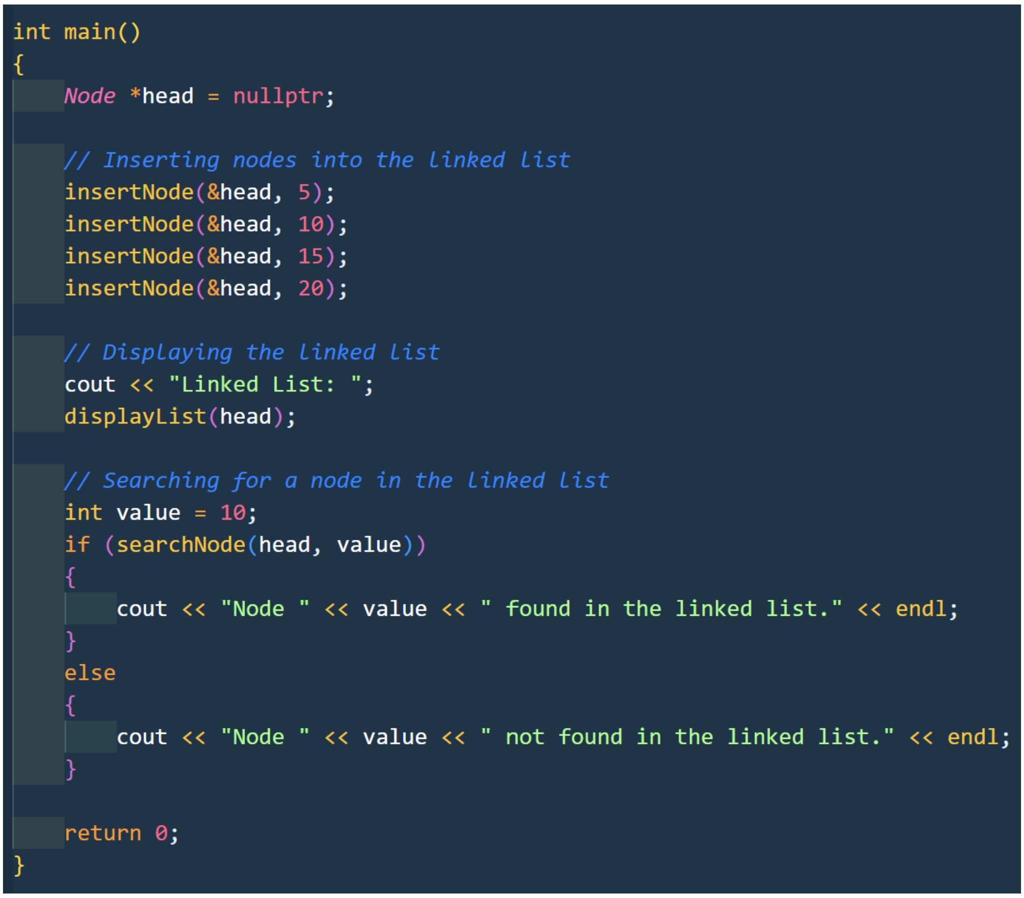
2) Write a program in C++ to search a particular node in given in singly linked list.



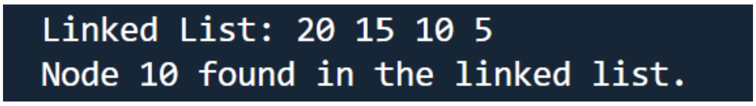
```cpp
#include <iostream>
using namespace std;

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

// Function to search for a node in the linked list
bool searchNode(Node *head, int value)
{
    Node *current = head;

    while (current != nullptr)
    {
        if (current->data == value)
        {
            return true; // Node found
        }
        current = current->next;
    }

    return false; // Node not found
}
```



```cpp
// Function to insert a node at the beginning of the linked list
void insertNode(Node **head, int value)
{
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

// Function to display the linked list
void displayList(Node *head)
{
    Node *current = head;

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

    cout << endl;
}
```



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

    // Inserting nodes into the linked list
    insertNode(&head, 5);
    insertNode(&head, 10);
    insertNode(&head, 15);
    insertNode(&head, 20);

    // Displaying the linked list
    cout << "Linked List: ";
    displayList(head);

    // Searching for a node in the linked list
    int value = 10;
    if (searchNode(head, value))
    {
        cout << "Node " << value << " found in the linked list." << endl;
    }
    else
    {
        cout << "Node " << value << " not found in the linked list." << endl;
    }

    return 0;
}
```



```
Linked List: 20 15 10 5
Node 10 found in the linked list.
```

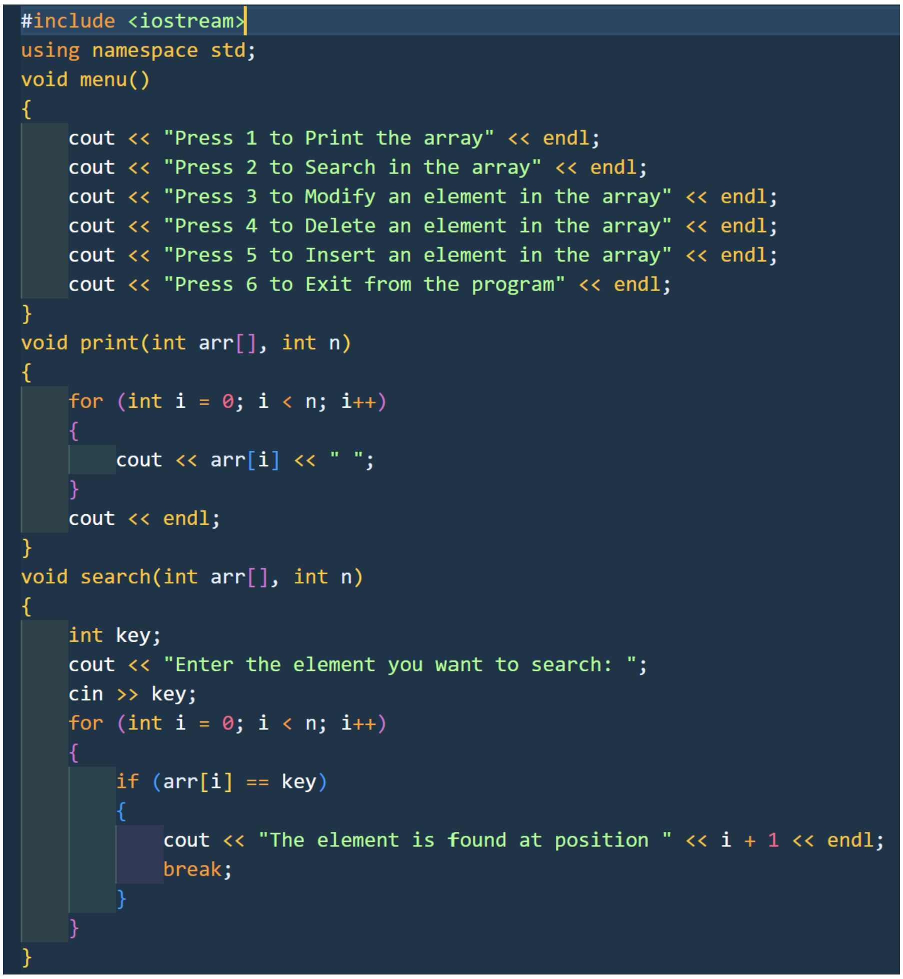
3) Write a program in C++ to perform the following operations using arrays:

i. Insertion of element

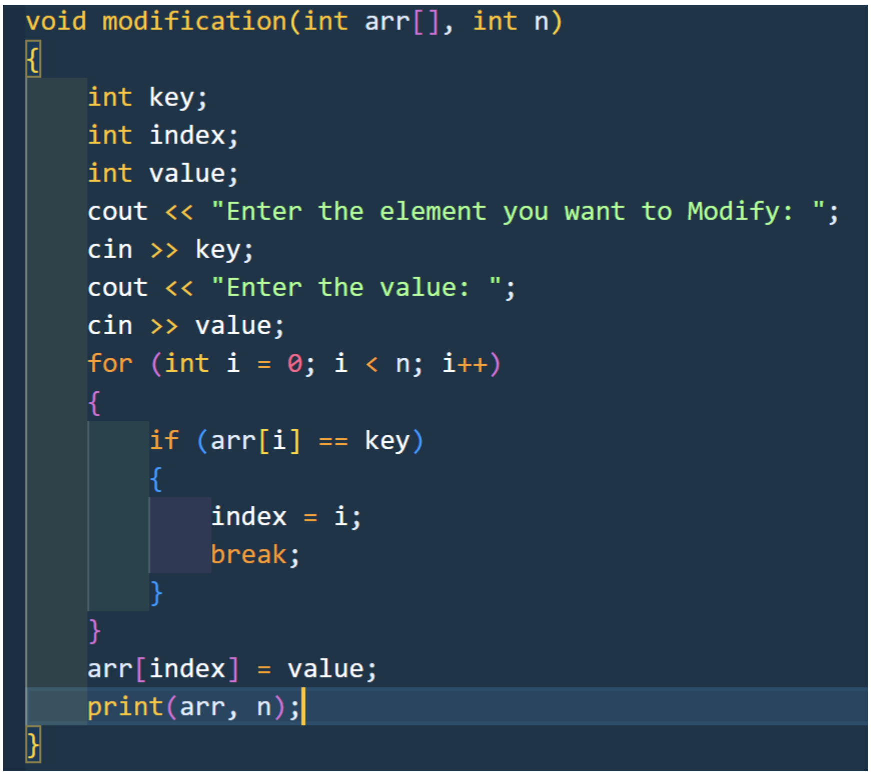
ii. Deletion of element

iii. Traversing the array

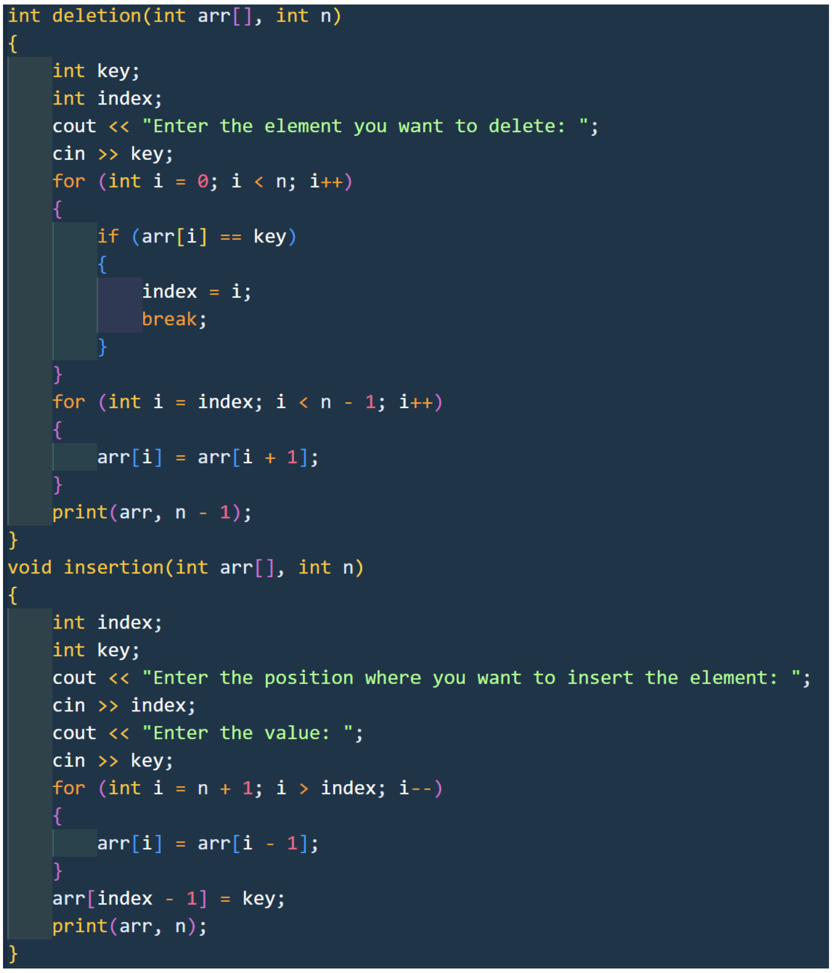
iv. Updating the value of an array



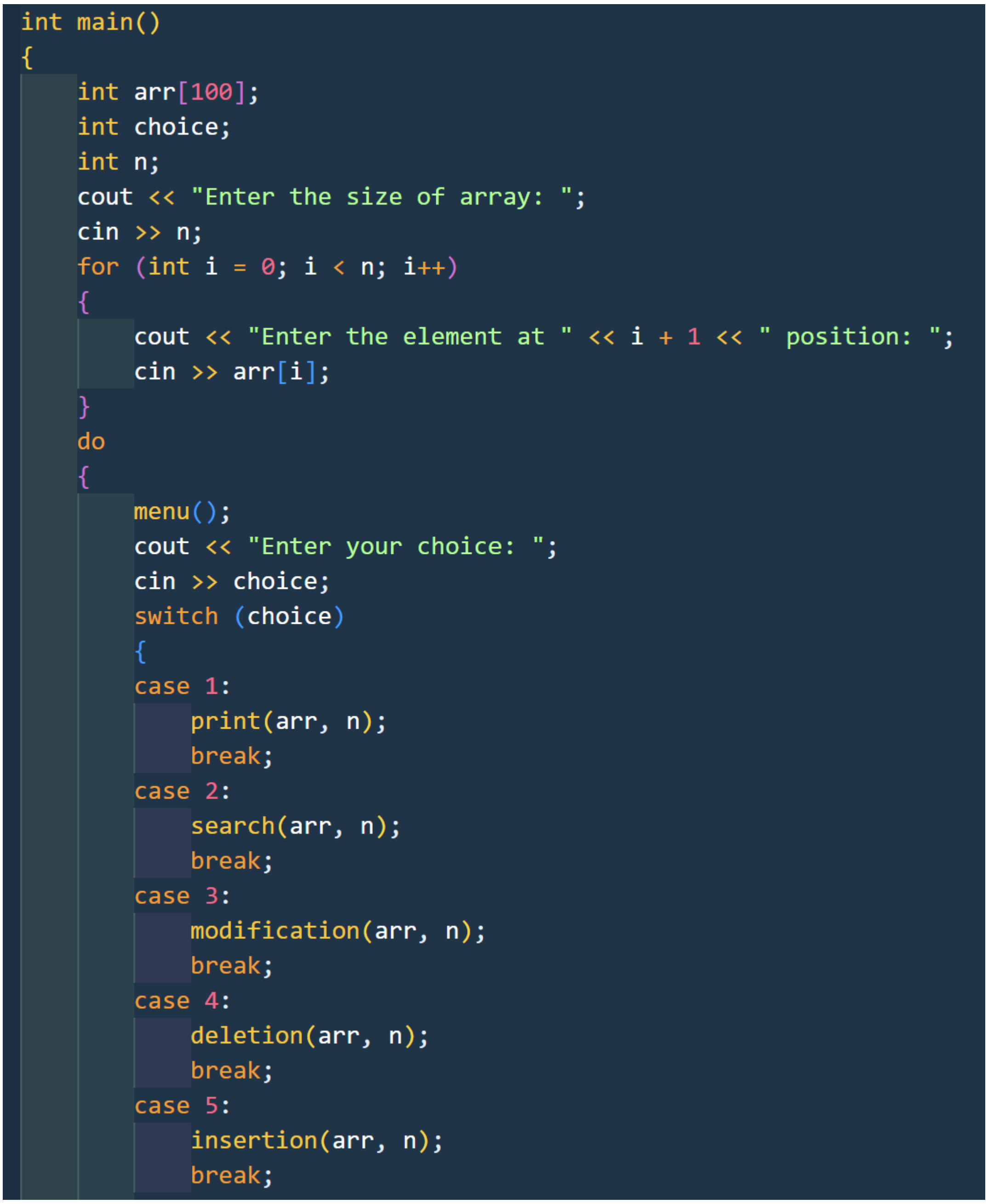
```cpp
#include <iostream>
using namespace std;
void menu()
{
    cout << "Press 1 to Print the array" << endl;
    cout << "Press 2 to Search in the array" << endl;
    cout << "Press 3 to Modify an element in the array" << endl;
    cout << "Press 4 to Delete an element in the array" << endl;
    cout << "Press 5 to Insert an element in the array" << endl;
    cout << "Press 6 to Exit from the program" << endl;
}
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
void search(int arr[], int n)
{
    int key;
    cout << "Enter the element you want to search: ";
    cin >> key;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == key)
        {
            cout << "The element is found at position " << i + 1 << endl;
            break;
        }
    }
}
```



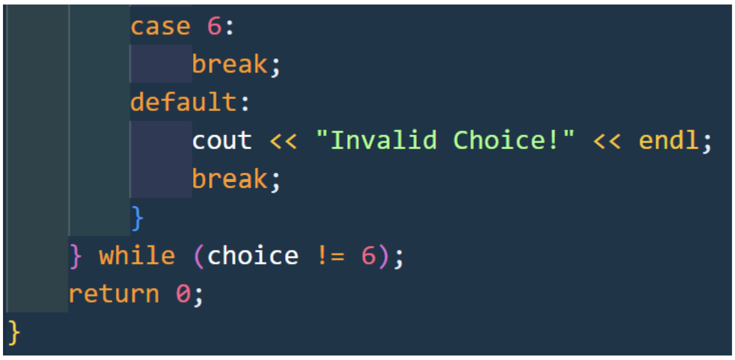
```
void modification(int arr[], int n)
{
    int key;
    int index;
    int value;
    cout << "Enter the element you want to Modify: ";
    cin >> key;
    cout << "Enter the value: ";
    cin >> value;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == key)
        {
            index = i;
            break;
        }
    }
    arr[index] = value;
    print(arr, n);
}
```



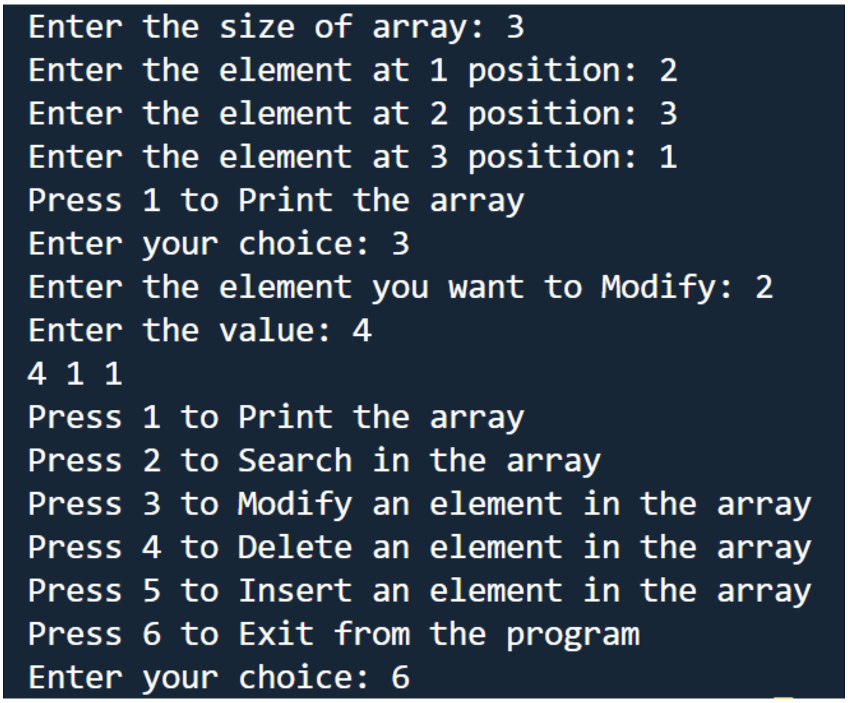
```
int deletion(int arr[], int n)
{
    int key;
    int index;
    cout << "Enter the element you want to delete: ";
    cin >> key;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == key)
        {
            index = i;
            break;
        }
    }
    for (int i = index; i < n - 1; i++)
    {
        arr[i] = arr[i + 1];
    }
    print(arr, n - 1);
}
void insertion(int arr[], int n)
{
    int index;
    int key;
    cout << "Enter the position where you want to insert the element: ";
    cin >> index;
    cout << "Enter the value: ";
    cin >> key;
    for (int i = n + 1; i > index; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[index - 1] = key;
    print(arr, n);
}
```



```cpp
int main()
{
    int arr[100];
    int choice;
    int n;
    cout << "Enter the size of array: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cout << "Enter the element at " << i + 1 << " position: ";
        cin >> arr[i];
    }
    do
    {
        menu();
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            print(arr, n);
            break;
        case 2:
            search(arr, n);
            break;
        case 3:
            modification(arr, n);
            break;
        case 4:
            deletion(arr, n);
            break;
        case 5:
            insertion(arr, n);
            break;
```



```cpp
        case 6:
            break;
        default:
            cout << "Invalid Choice!" << endl;
            break;
        }
    } while (choice != 6);
    return 0;
}
```



```
Enter the size of array: 3
Enter the element at 1 position: 2
Enter the element at 2 position: 3
Enter the element at 3 position: 1
Press 1 to Print the array
Enter your choice: 3
Enter the element you want to Modify: 2
Enter the value: 4
4 1 1
Press 1 to Print the array
Press 2 to Search in the array
Press 3 to Modify an element in the array
Press 4 to Delete an element in the array
Press 5 to Insert an element in the array
Press 6 to Exit from the program
Enter your choice: 6
```

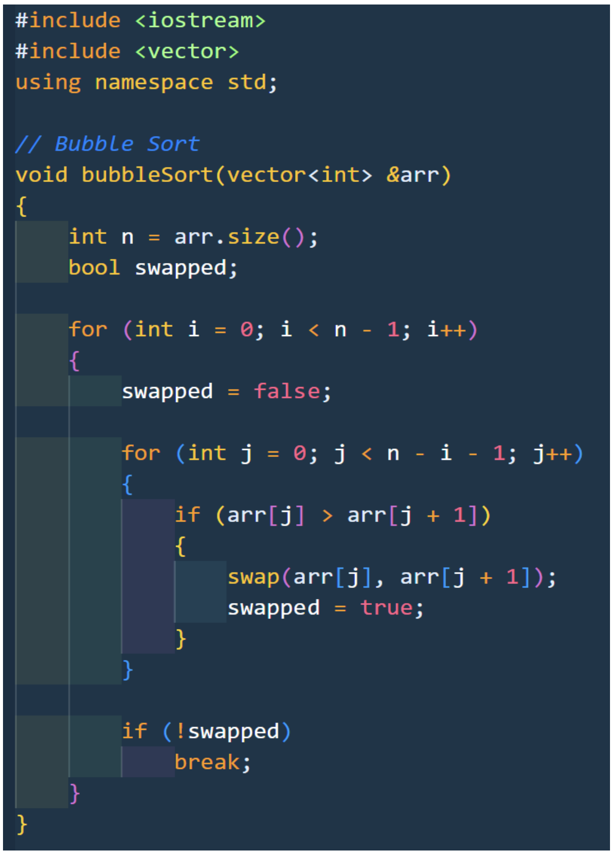
4) Write a program in C++ to perform the following:

i. Bubble Sort

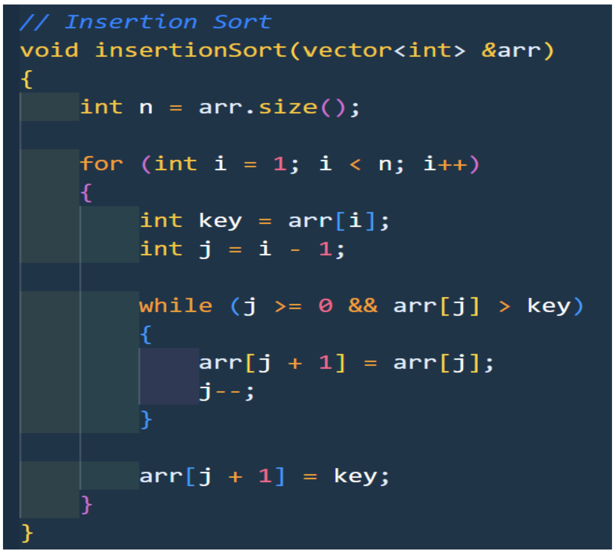
ii. Insertion Sort

iii. Counting Sort

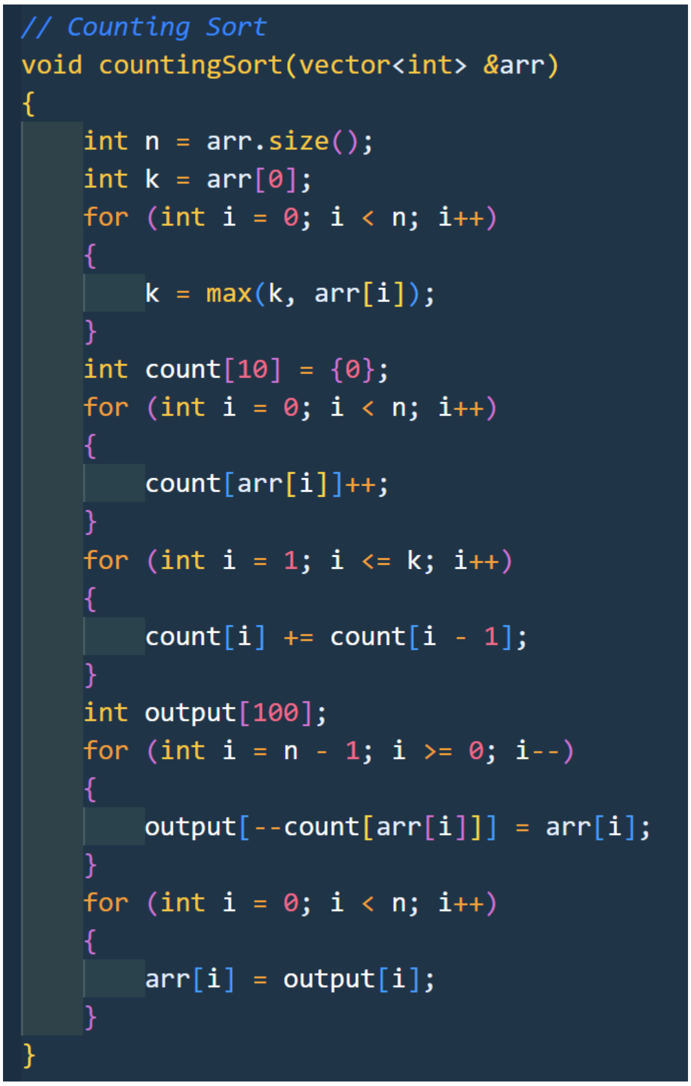
iv. Selection Sort



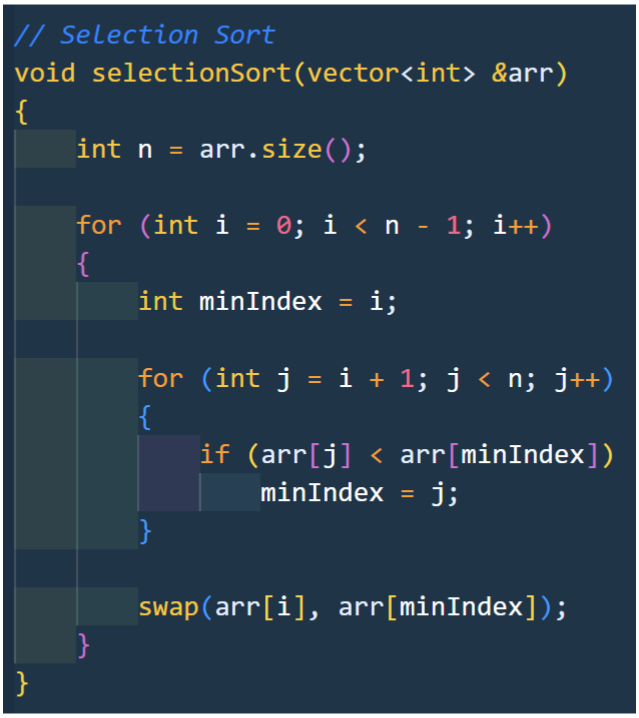
```
#include <iostream>
#include <vector>
using namespace std;

// Bubble Sort
void bubbleSort(vector<int> &arr)
{
    int n = arr.size();
    bool swapped;

    for (int i = 0; i < n - 1; i++)
    {
        swapped = false;

        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }

        if (!swapped)
            break;
    }
}
```



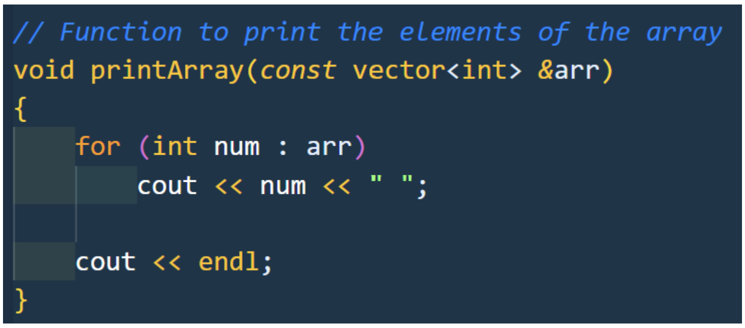
```
// Insertion Sort
void insertionSort(vector<int> &arr)
{
    int n = arr.size();

    for (int i = 1; i < n; i++)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}
```



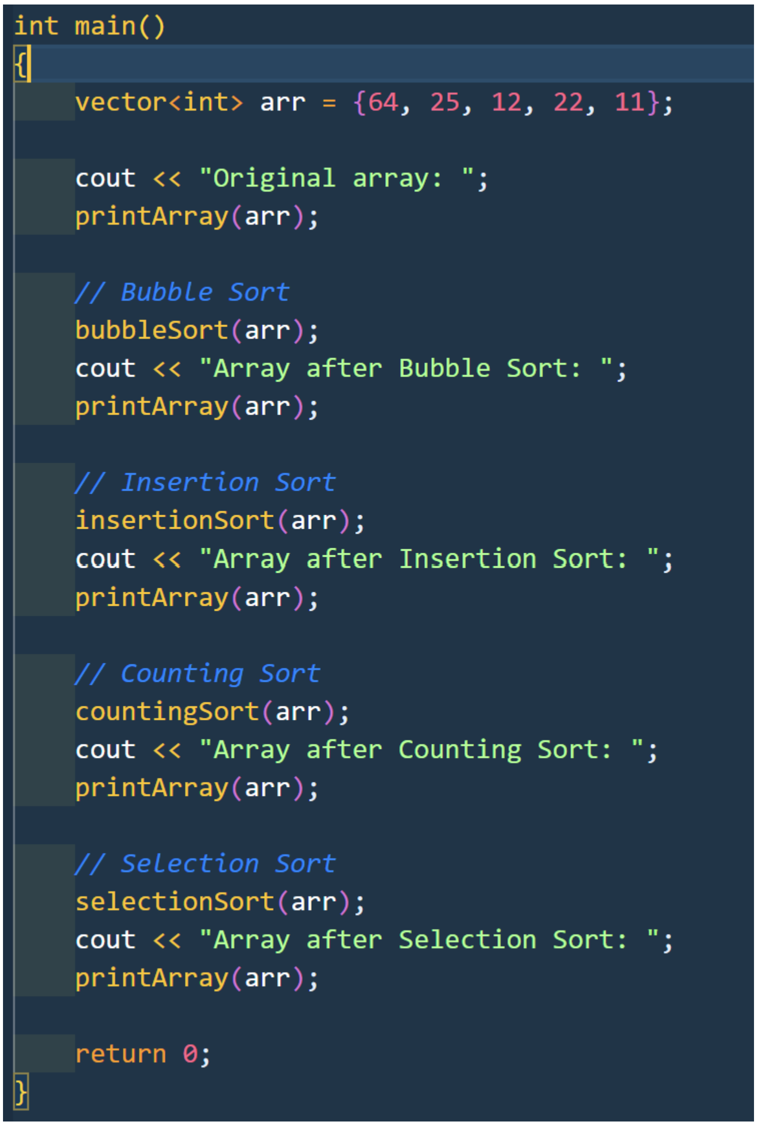
```cpp
// Counting Sort
void countingSort(vector<int> &arr)
{
    int n = arr.size();
    int k = arr[0];
    for (int i = 0; i < n; i++)
    {
        k = max(k, arr[i]);
    }
    int count[10] = {0};
    for (int i = 0; i < n; i++)
    {
        count[arr[i]]++;
    }
    for (int i = 1; i <= k; i++)
    {
        count[i] += count[i - 1];
    }
    int output[100];
    for (int i = n - 1; i >= 0; i--)
    {
        output[--count[arr[i]]] = arr[i];
    }
    for (int i = 0; i < n; i++)
    {
        arr[i] = output[i];
    }
}
```



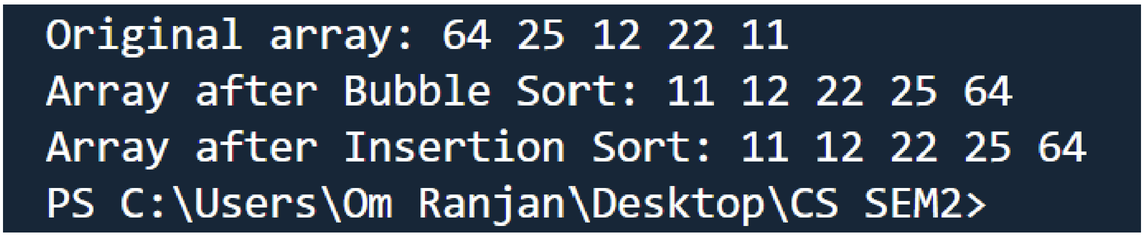
```cpp
// Selection Sort
void selectionSort(vector<int> &arr)
{
    int n = arr.size();

    for (int i = 0; i < n - 1; i++)
    {
        int minIndex = i;

        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }

        swap(arr[i], arr[minIndex]);
    }
}
```



```cpp
// Function to print the elements of the array
void printArray(const vector<int> &arr)
{
    for (int num : arr)
        cout << num << " ";

    cout << endl;
}
```

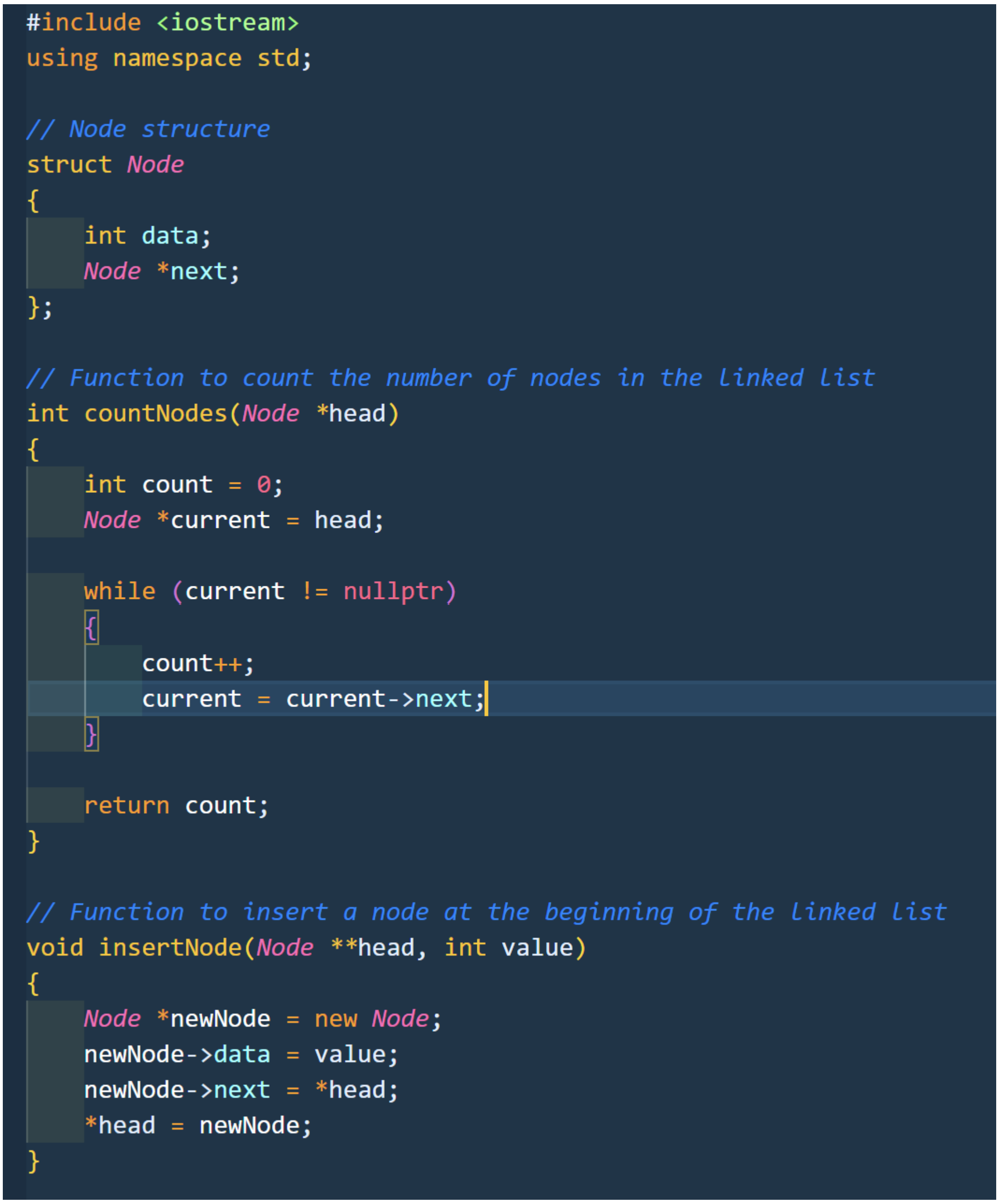


```cpp
int main()
{
    vector<int> arr = {64, 25, 12, 22, 11};

    cout << "Original array: ";
    printArray(arr);

    // Bubble Sort
    bubbleSort(arr);
    cout << "Array after Bubble Sort: ";
    printArray(arr);

    // Insertion Sort
    insertionSort(arr);
    cout << "Array after Insertion Sort: ";
    printArray(arr);

    // Counting Sort
    countingSort(arr);
    cout << "Array after Counting Sort: ";
    printArray(arr);

    // Selection Sort
    selectionSort(arr);
    cout << "Array after Selection Sort: ";
    printArray(arr);

    return 0;
}
```

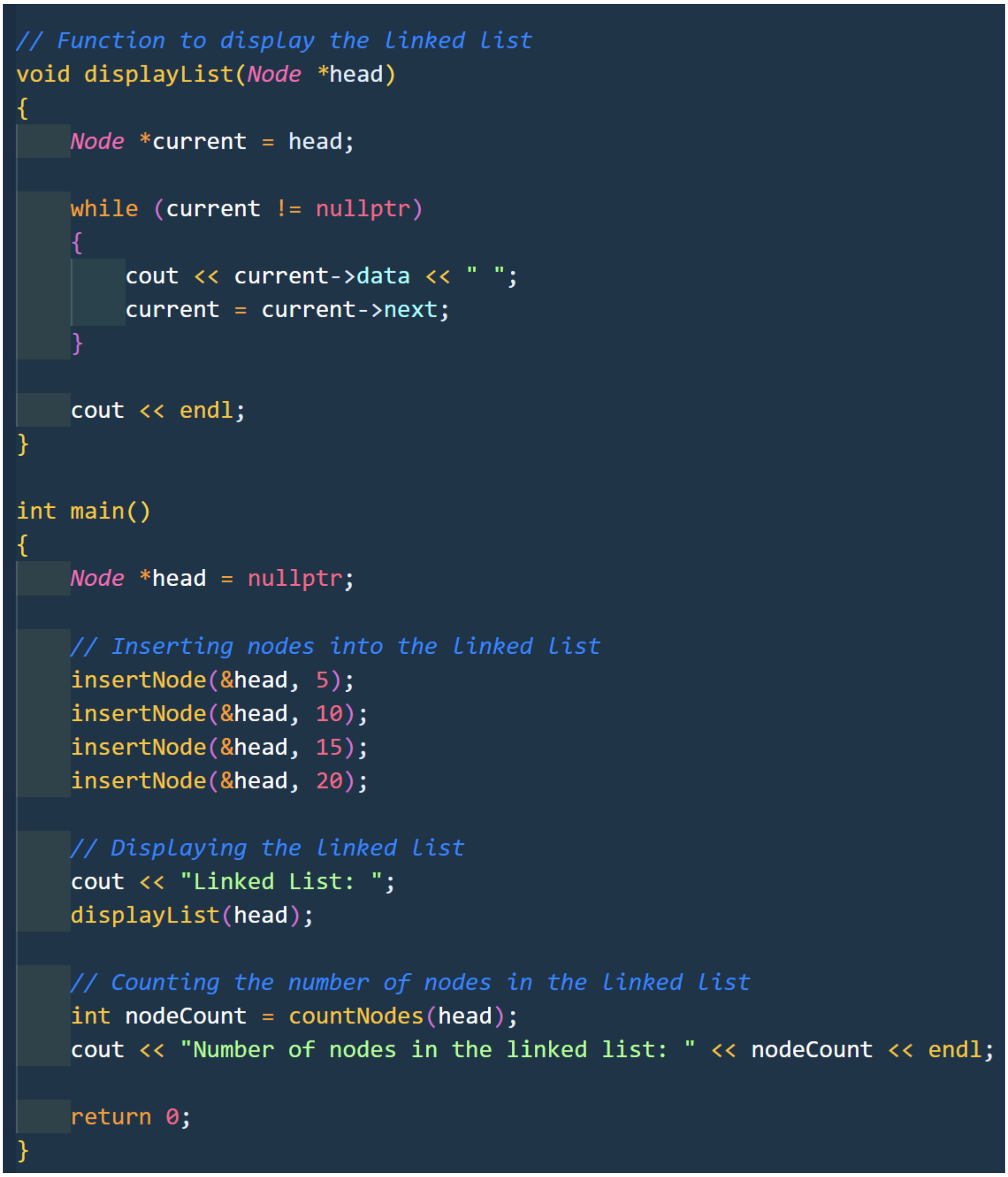


```
Original array: 64 25 12 22 11
Array after Bubble Sort: 11 12 22 25 64
Array after Insertion Sort: 11 12 22 25 64
PS C:\Users\Om Ranjan\Desktop\CS SEM2>
```

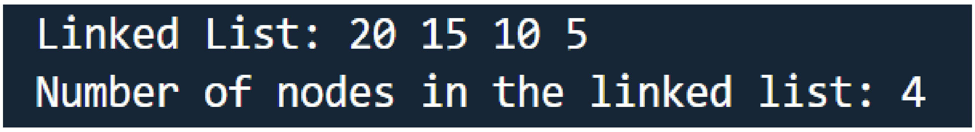
5) Write a program in C++ to count the number of nodes in a linked list.



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

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

// Function to count the number of nodes in the linked list
int countNodes(Node *head)
{
    int count = 0;
    Node *current = head;

    while (current != nullptr)
    {
        count++;
        current = current->next;
    }

    return count;
}

// Function to insert a node at the beginning of the linked list
void insertNode(Node **head, int value)
{
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}
```

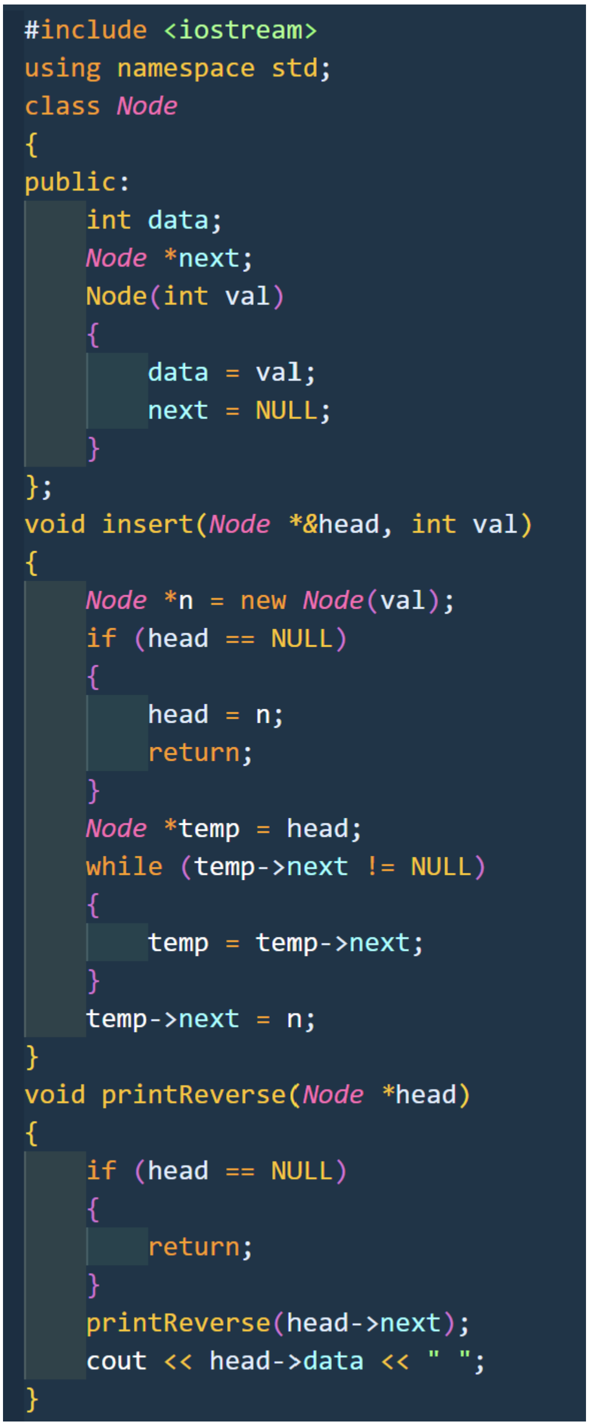


```cpp
// Function to display the linked list
void displayList(Node *head)
{
    Node *current = head;

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

    cout << endl;
}

int main()
{
    Node *head = nullptr;

    // Inserting nodes into the linked list
    insertNode(&head, 5);
    insertNode(&head, 10);
    insertNode(&head, 15);
    insertNode(&head, 20);

    // Displaying the linked list
    cout << "Linked List: ";
    displayList(head);

    // Counting the number of nodes in the linked list
    int nodeCount = countNodes(head);
    cout << "Number of nodes in the linked list: " << nodeCount << endl;

    return 0;
}
```

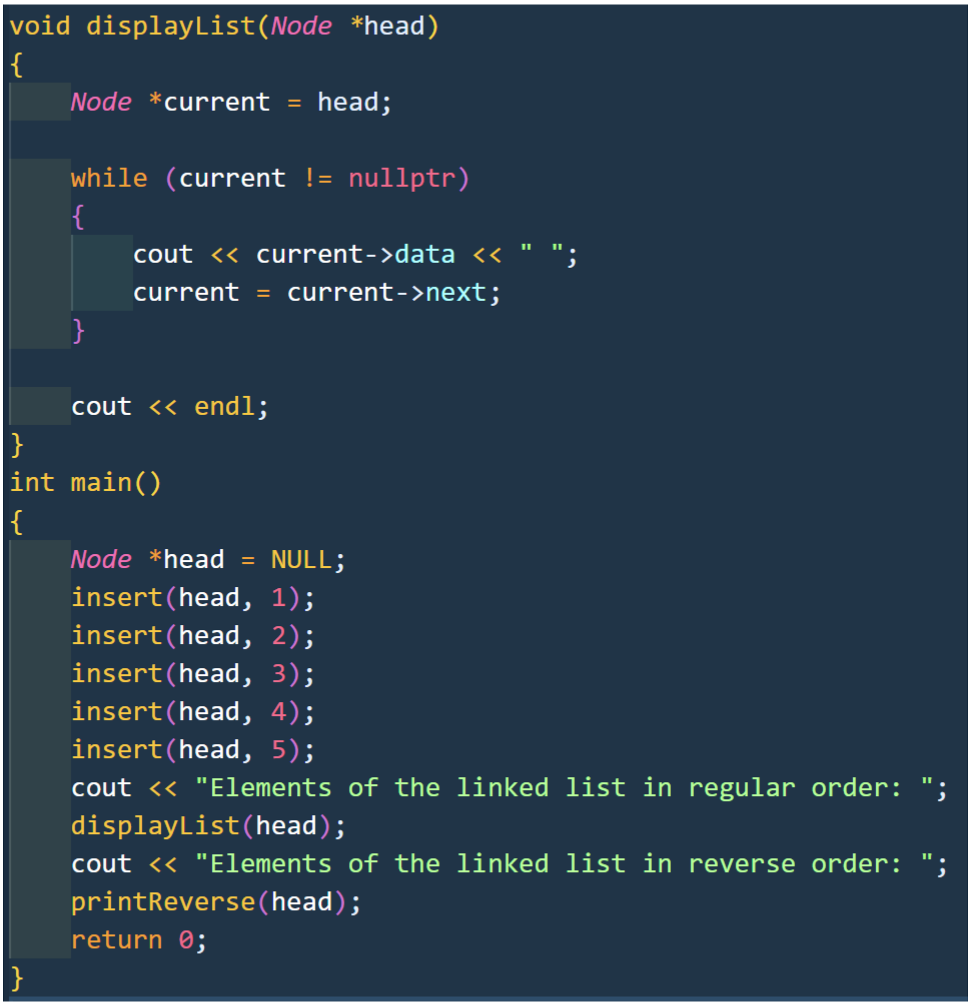


```
Linked List: 20 15 10 5
Number of nodes in the linked list: 4
```

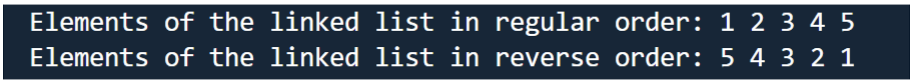
6) Write a program in C++ to print the elements of a linked list in reverse order.



```cpp
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node(int val)
    {
        data = val;
        next = NULL;
    }
};
void insert(Node *&head, int val)
{
    Node *n = new Node(val);
    if (head == NULL)
    {
        head = n;
        return;
    }
    Node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = n;
}
void printReverse(Node *head)
{
    if (head == NULL)
    {
        return;
    }
    printReverse(head->next);
    cout << head->data << " ";
}
```

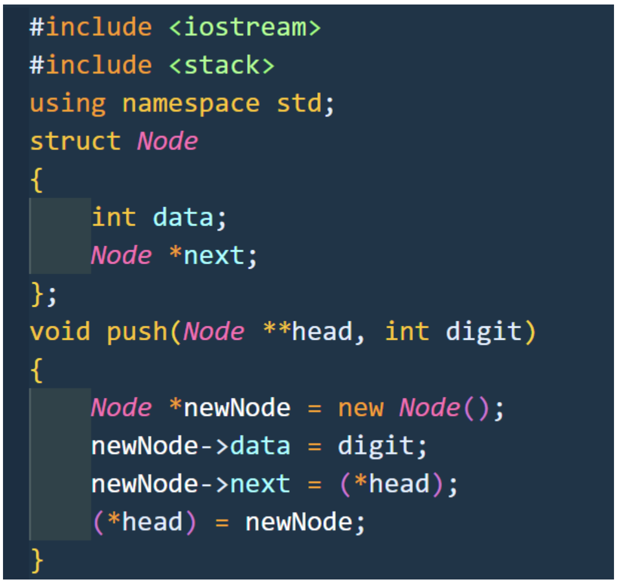


```cpp
void displayList(Node *head)
{
    Node *current = head;

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

    cout << endl;
}
int main()
{
    Node *head = NULL;
    insert(head, 1);
    insert(head, 2);
    insert(head, 3);
    insert(head, 4);
    insert(head, 5);
    cout << "Elements of the linked list in regular order: ";
    displayList(head);
    cout << "Elements of the linked list in reverse order: ";
    printReverse(head);
    return 0;
}
```

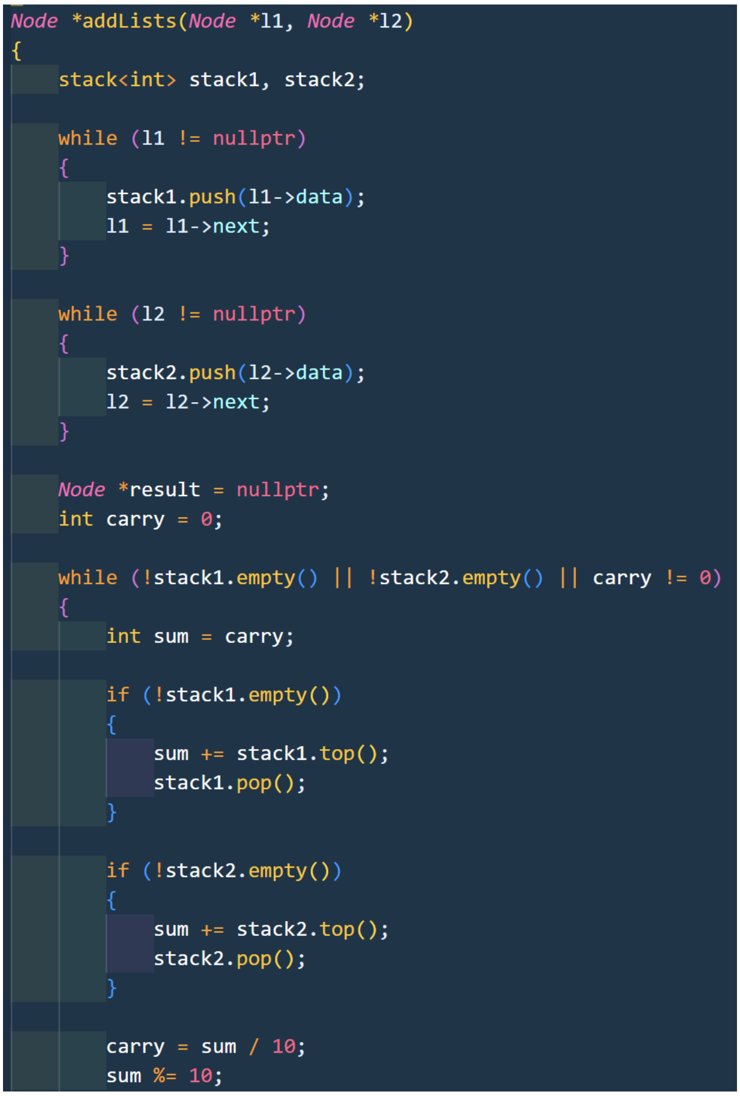


```
Elements of the linked list in regular order: 1 2 3 4 5
Elements of the linked list in reverse order: 5 4 3 2 1
```

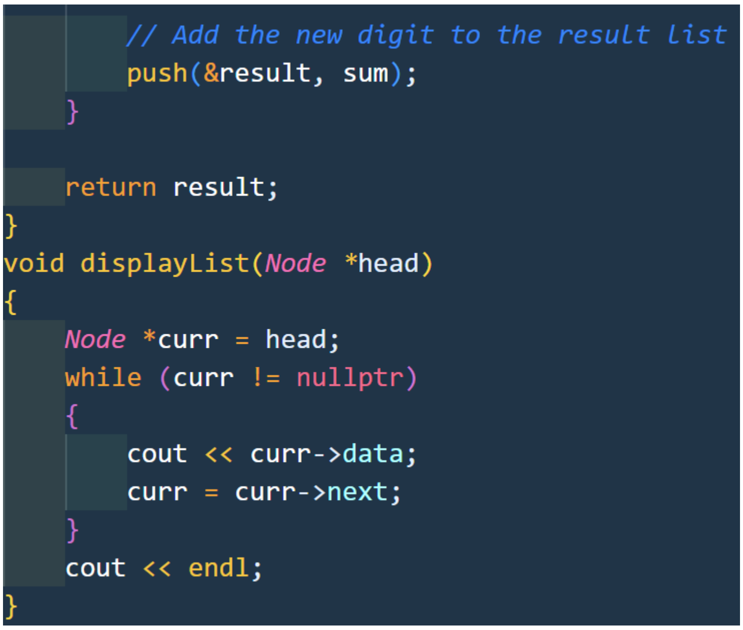
7) Write a program in C++ to add two very large numbers using the stack and linked list.



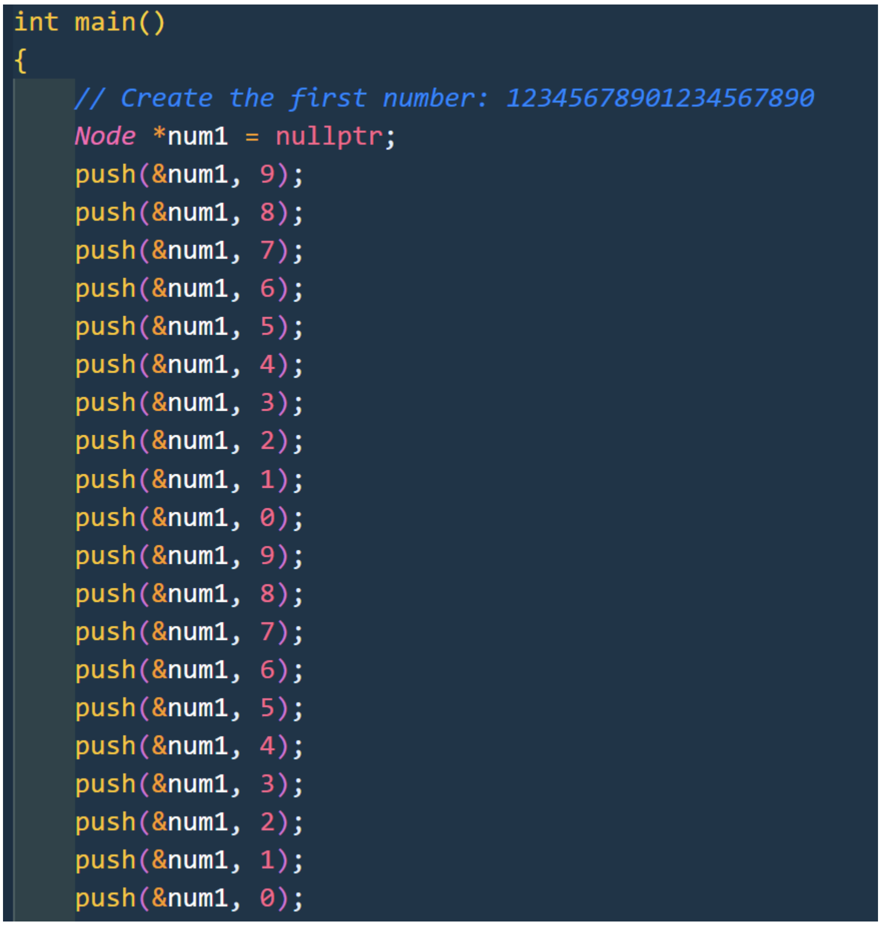
```cpp
#include <iostream>
#include <stack>
using namespace std;
struct Node
{
    int data;
    Node *next;
};
void push(Node **head, int digit)
{
    Node *newNode = new Node();
    newNode->data = digit;
    newNode->next = (*head);
    (*head) = newNode;
}
```



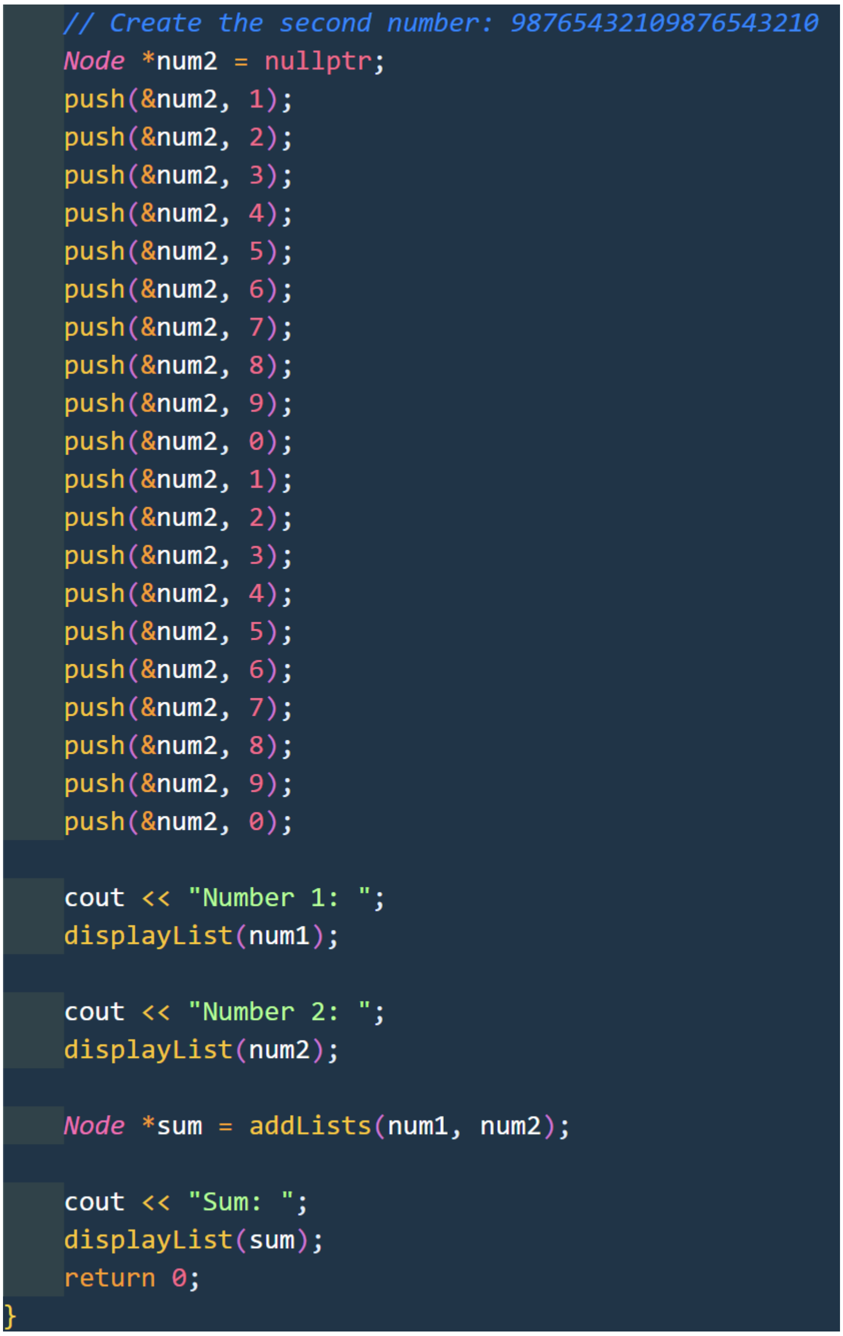
```cpp
Node *addLists(Node *l1, Node *l2)
{
    stack<int> stack1, stack2;

    while (l1 != nullptr)
    {
        stack1.push(l1->data);
        l1 = l1->next;
    }

    while (l2 != nullptr)
    {
        stack2.push(l2->data);
        l2 = l2->next;
    }

    Node *result = nullptr;
    int carry = 0;

    while (!stack1.empty() || !stack2.empty() || carry != 0)
    {
        int sum = carry;

        if (!stack1.empty())
        {
            sum += stack1.top();
            stack1.pop();
        }

        if (!stack2.empty())
        {
            sum += stack2.top();
            stack2.pop();
        }

        carry = sum / 10;
        sum %= 10;
```



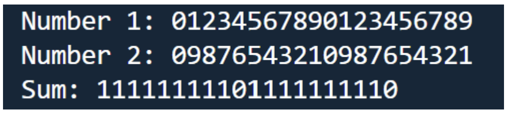
```
        // Add the new digit to the result list
        push(&result, sum);
    }

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



```
int main()
{
    // Create the first number: 12345678901234567890
    Node *num1 = nullptr;
    push(&num1, 9);
    push(&num1, 8);
    push(&num1, 7);
    push(&num1, 6);
    push(&num1, 5);
    push(&num1, 4);
    push(&num1, 3);
    push(&num1, 2);
    push(&num1, 1);
    push(&num1, 0);
    push(&num1, 9);
    push(&num1, 8);
    push(&num1, 7);
    push(&num1, 6);
    push(&num1, 5);
    push(&num1, 4);
    push(&num1, 3);
    push(&num1, 2);
    push(&num1, 1);
    push(&num1, 0);
```

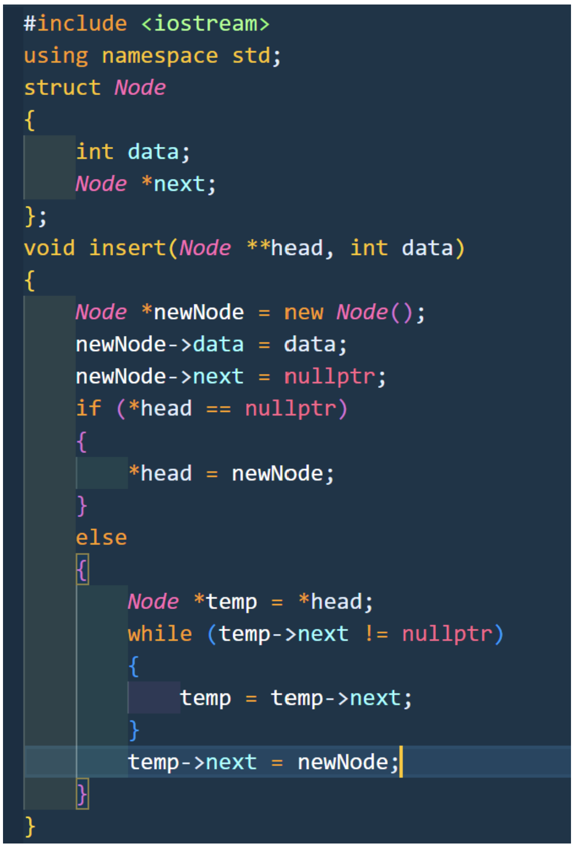


```cpp
    // Create the second number: 98765432109876543210
    Node *num2 = nullptr;
    push(&num2, 1);
    push(&num2, 2);
    push(&num2, 3);
    push(&num2, 4);
    push(&num2, 5);
    push(&num2, 6);
    push(&num2, 7);
    push(&num2, 8);
    push(&num2, 9);
    push(&num2, 0);
    push(&num2, 1);
    push(&num2, 2);
    push(&num2, 3);
    push(&num2, 4);
    push(&num2, 5);
    push(&num2, 6);
    push(&num2, 7);
    push(&num2, 8);
    push(&num2, 9);
    push(&num2, 0);

    cout << "Number 1: ";
    displayList(num1);

    cout << "Number 2: ";
    displayList(num2);

    Node *sum = addLists(num1, num2);

    cout << "Sum: ";
    displayList(sum);
    return 0;
}
```

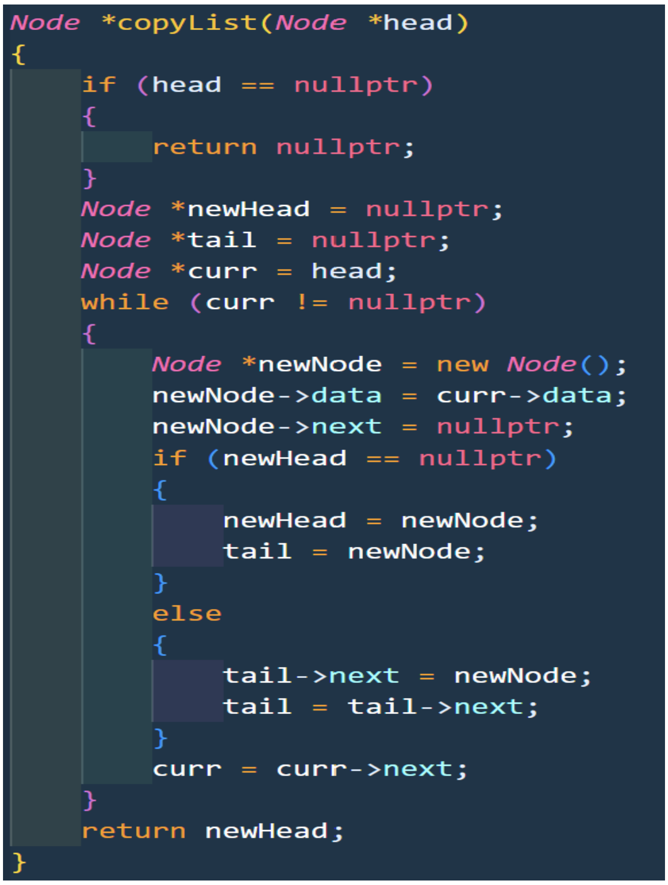


```
Number 1: 01234567890123456789
Number 2: 09876543210987654321
Sum: 11111111101111111110
```

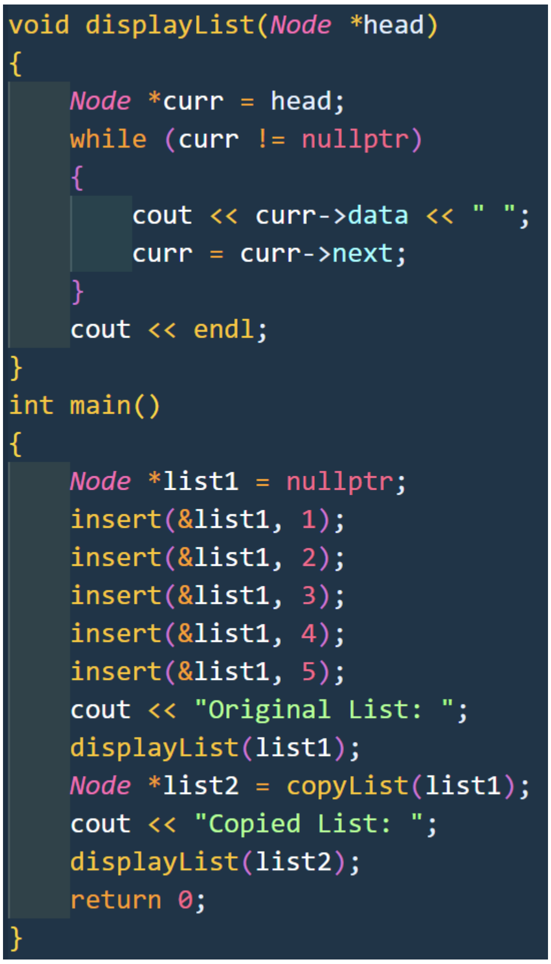
8) Write a program in C++ to copy element of linked list to another linked list.



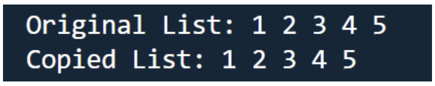
```cpp
#include <iostream>
using namespace std;
struct Node
{
    int data;
    Node *next;
};
void insert(Node **head, int data)
{
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    if (*head == nullptr)
    {
        *head = newNode;
    }
    else
    {
        Node *temp = *head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```



```cpp
Node *copyList(Node *head)
{
    if (head == nullptr)
    {
        return nullptr;
    }
    Node *newHead = nullptr;
    Node *tail = nullptr;
    Node *curr = head;
    while (curr != nullptr)
    {
        Node *newNode = new Node();
        newNode->data = curr->data;
        newNode->next = nullptr;
        if (newHead == nullptr)
        {
            newHead = newNode;
            tail = newNode;
        }
        else
        {
            tail->next = newNode;
            tail = tail->next;
        }
        curr = curr->next;
    }
    return newHead;
}
```



```cpp
void displayList(Node *head)
{
    Node *curr = head;
    while (curr != nullptr)
    {
        cout << curr->data << " ";
        curr = curr->next;
    }
    cout << endl;
}
int main()
{
    Node *list1 = nullptr;
    insert(&list1, 1);
    insert(&list1, 2);
    insert(&list1, 3);
    insert(&list1, 4);
    insert(&list1, 5);
    cout << "Original List: ";
    displayList(list1);
    Node *list2 = copyList(list1);
    cout << "Copied List: ";
    displayList(list2);
    return 0;
}
```



```
Original List: 1 2 3 4 5
Copied List: 1 2 3 4 5
```

9) Write a program in C++ to implement the Doubly linked list:

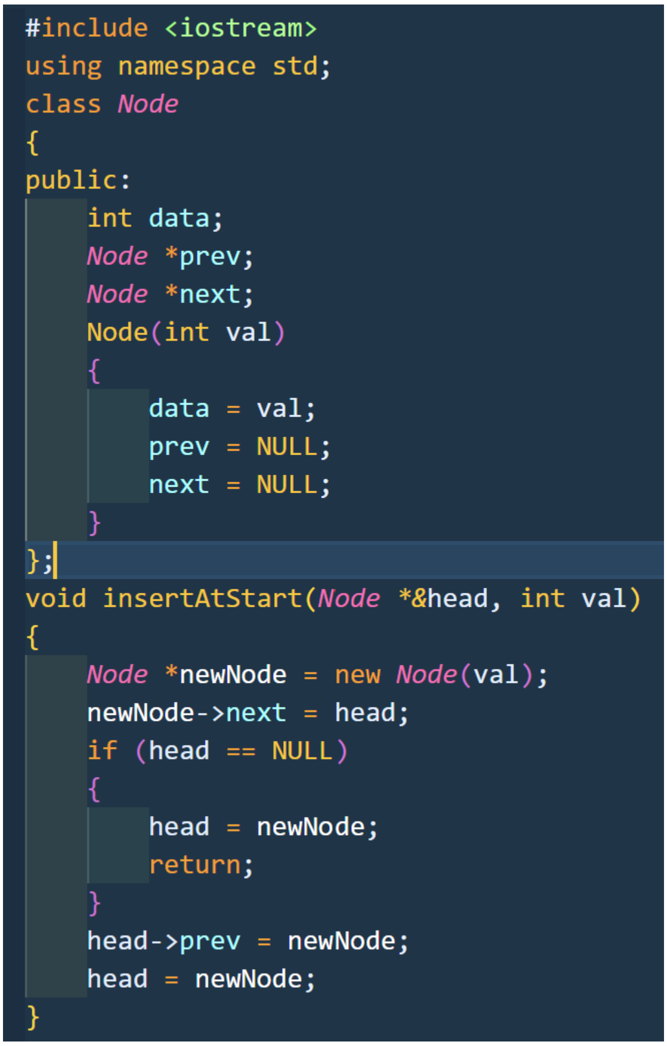
i. Insertion of node at start

ii. Insertion of node at end

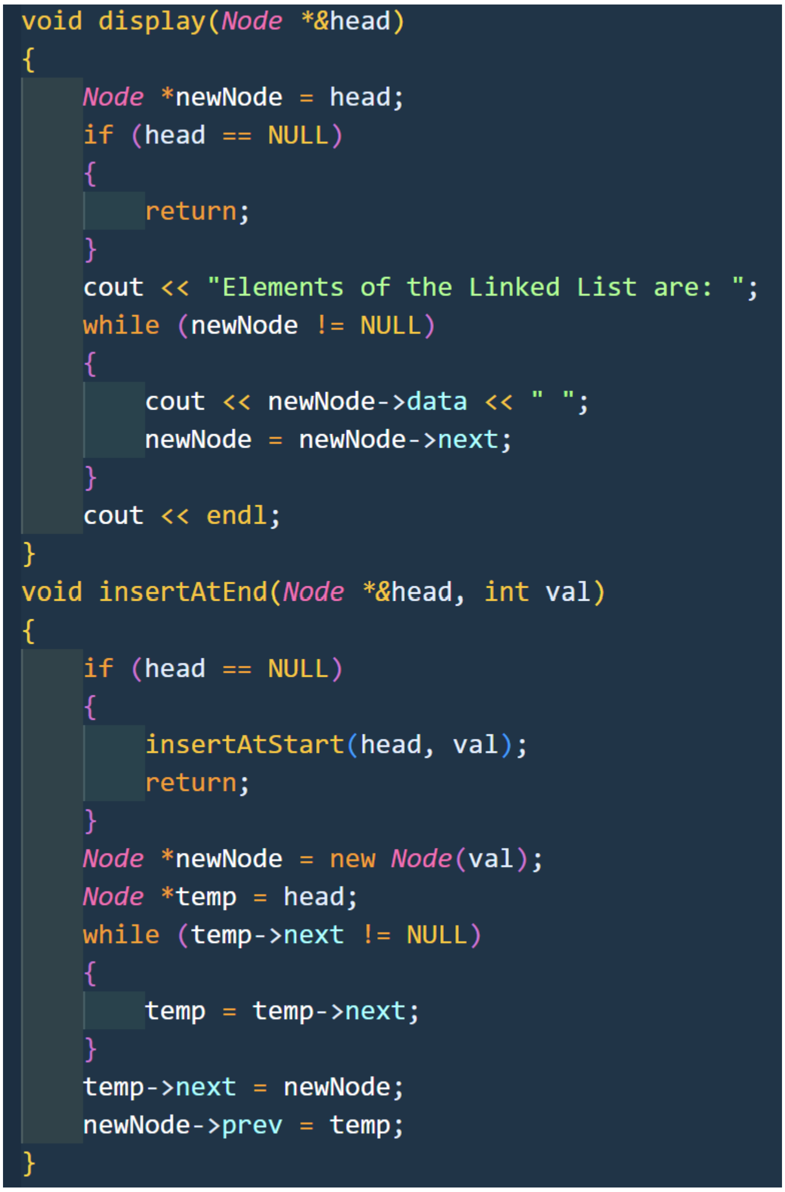
iii. Insertion of node at any position

iv. Deletion of start, end and nth position

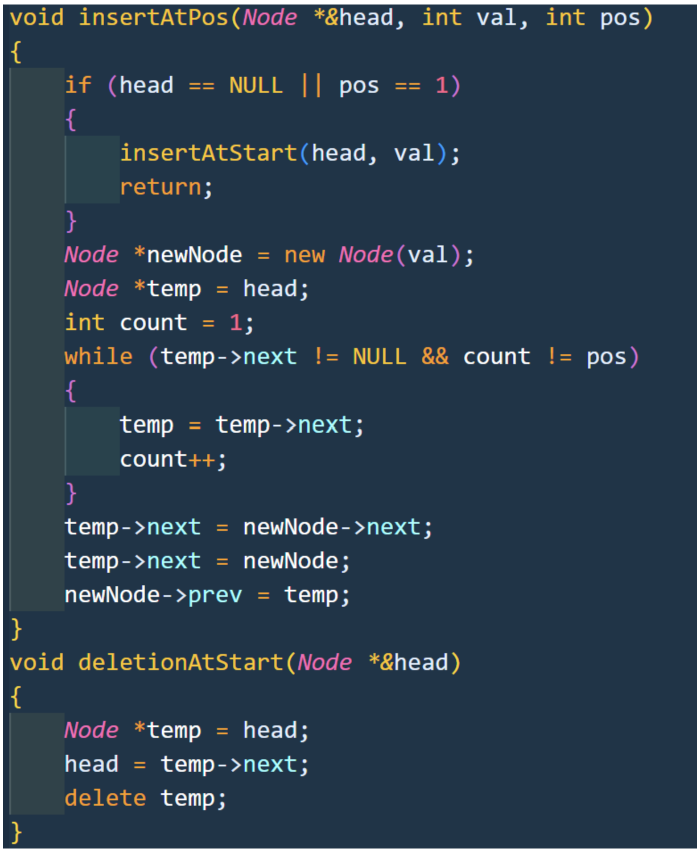
v. Traversal of Doubly linked list



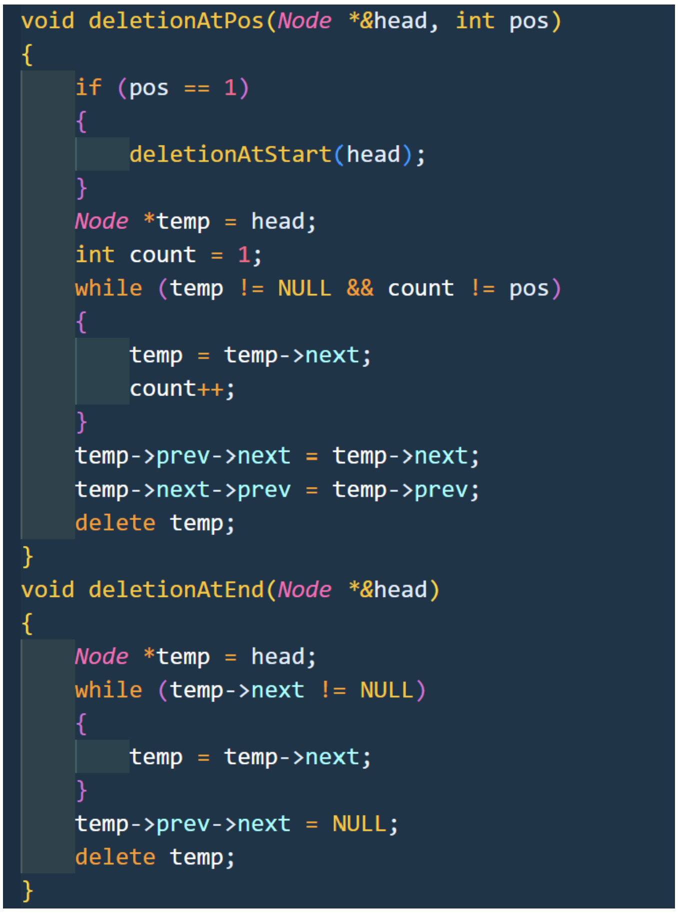
```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *prev;
    Node *next;
    Node(int val)
    {
        data = val;
        prev = NULL;
        next = NULL;
    }
};
void insertAtStart(Node *&head, int val)
{
    Node *newNode = new Node(val);
    newNode->next = head;
    if (head == NULL)
    {
        head = newNode;
        return;
    }
    head->prev = newNode;
    head = newNode;
}
```



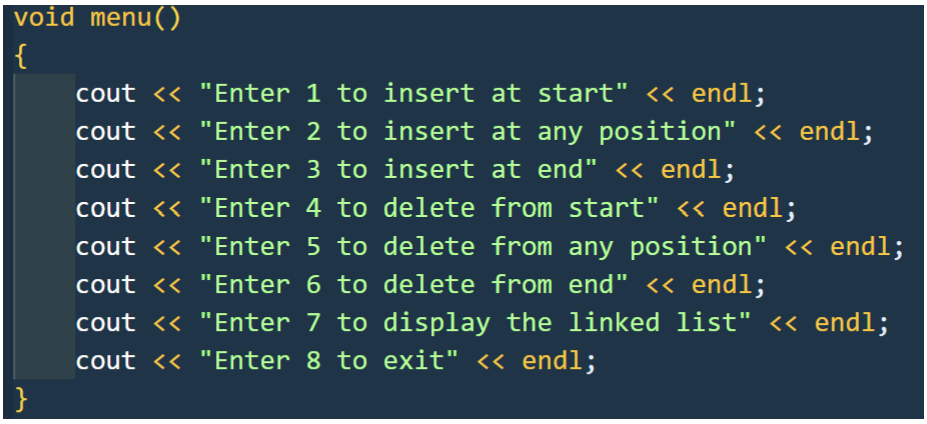
```
void display(Node *&head)
{
    Node *newNode = head;
    if (head == NULL)
    {
        return;
    }
    cout << "Elements of the Linked List are: ";
    while (newNode != NULL)
    {
        cout << newNode->data << " ";
        newNode = newNode->next;
    }
    cout << endl;
}
void insertAtEnd(Node *&head, int val)
{
    if (head == NULL)
    {
        insertAtStart(head, val);
        return;
    }
    Node *newNode = new Node(val);
    Node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
```



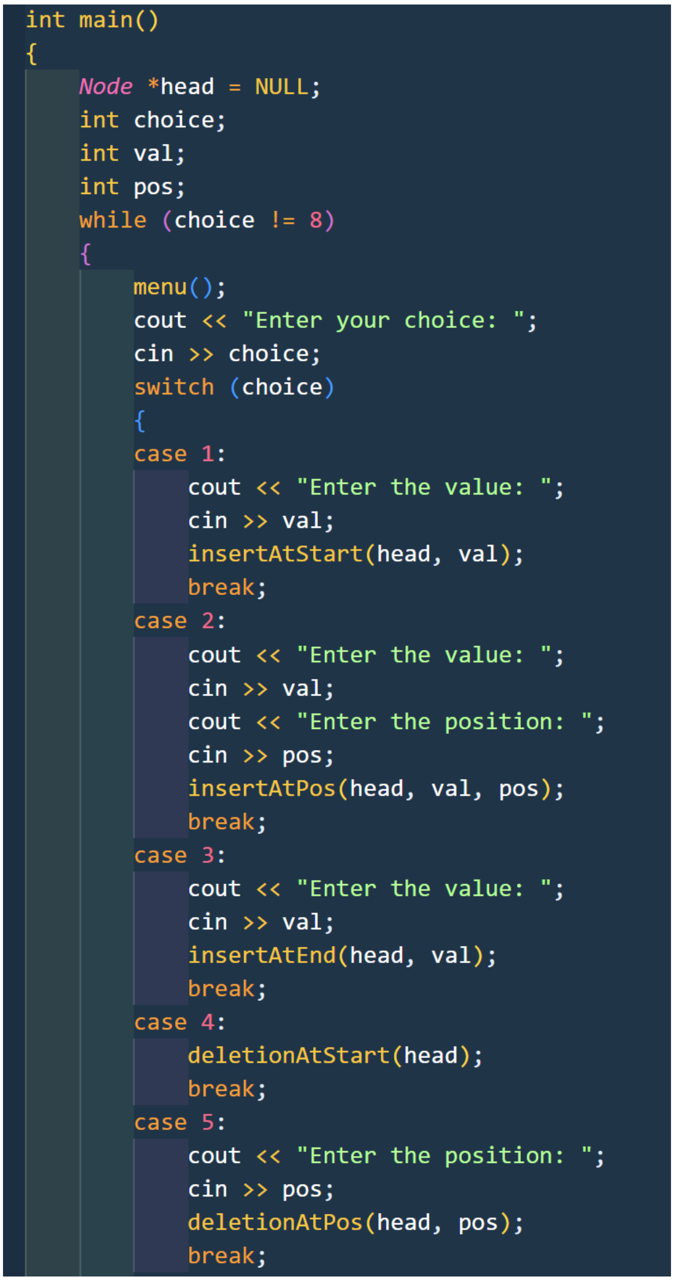
```cpp
void insertAtPos(Node *&head, int val, int pos)
{
    if (head == NULL || pos == 1)
    {
        insertAtStart(head, val);
        return;
    }
    Node *newNode = new Node(val);
    Node *temp = head;
    int count = 1;
    while (temp->next != NULL && count != pos)
    {
        temp = temp->next;
        count++;
    }
    temp->next = newNode->next;
    temp->next = newNode;
    newNode->prev = temp;
}
void deletionAtStart(Node *&head)
{
    Node *temp = head;
    head = temp->next;
    delete temp;
}
```



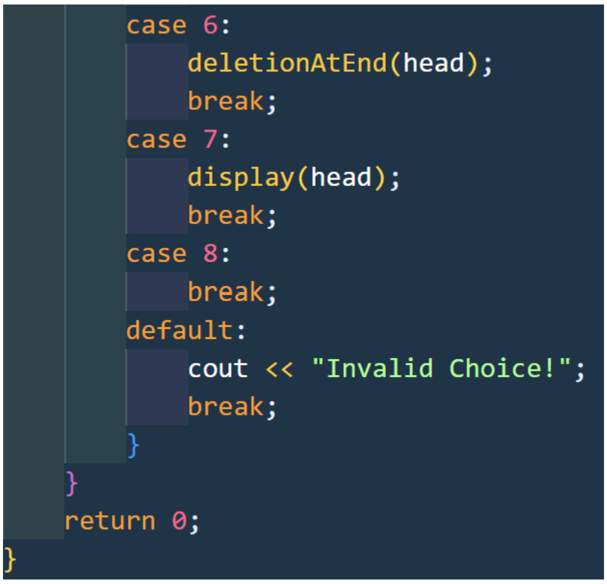
```cpp
void deletionAtPos(Node *&head, int pos)
{
    if (pos == 1)
    {
        deletionAtStart(head);
    }
    Node *temp = head;
    int count = 1;
    while (temp != NULL && count != pos)
    {
        temp = temp->next;
        count++;
    }
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    delete temp;
}
void deletionAtEnd(Node *&head)
{
    Node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->prev->next = NULL;
    delete temp;
}
```



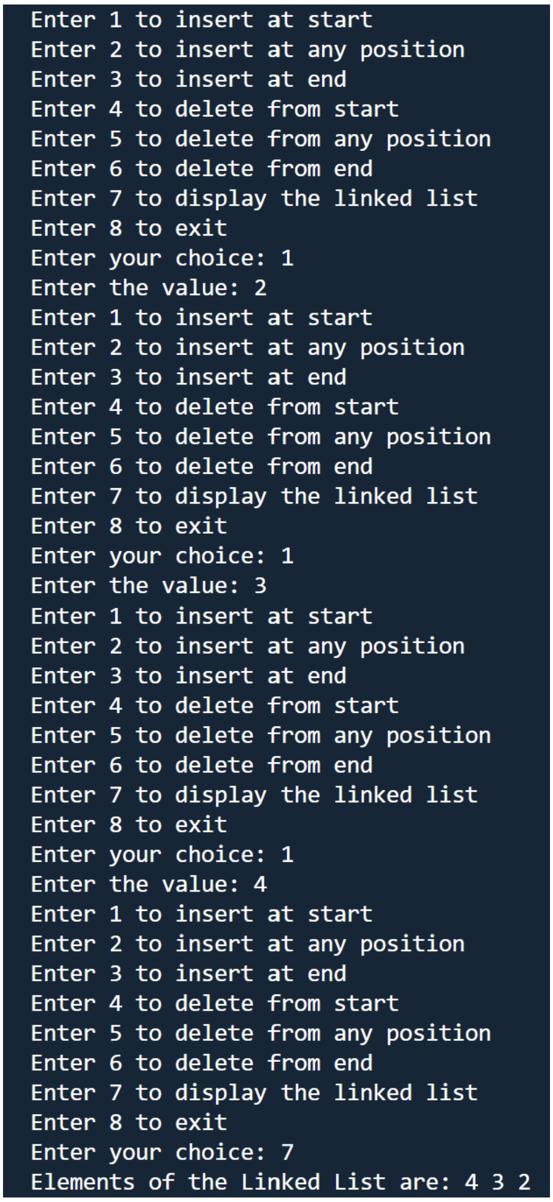
```cpp
void menu()
{
    cout << "Enter 1 to insert at start" << endl;
    cout << "Enter 2 to insert at any position" << endl;
    cout << "Enter 3 to insert at end" << endl;
    cout << "Enter 4 to delete from start" << endl;
    cout << "Enter 5 to delete from any position" << endl;
    cout << "Enter 6 to delete from end" << endl;
    cout << "Enter 7 to display the linked list" << endl;
    cout << "Enter 8 to exit" << endl;
}
```



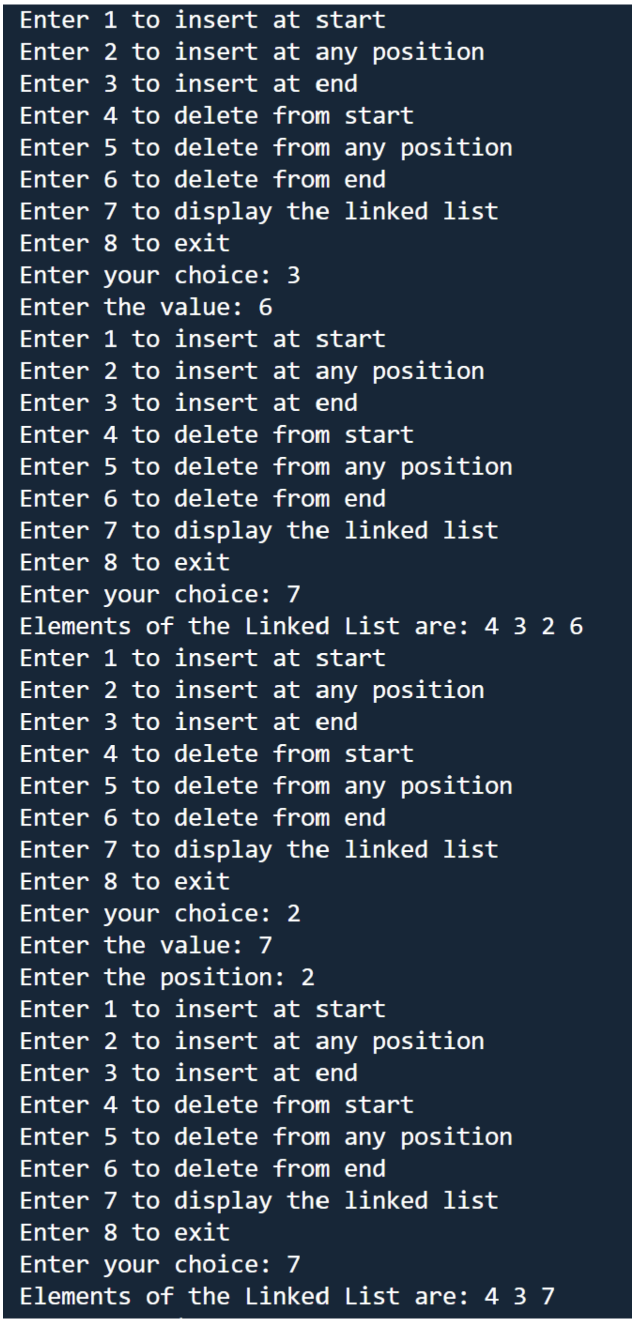
```cpp
int main()
{
    Node *head = NULL;
    int choice;
    int val;
    int pos;
    while (choice != 8)
    {
        menu();
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "Enter the value: ";
            cin >> val;
            insertAtStart(head, val);
            break;
        case 2:
            cout << "Enter the value: ";
            cin >> val;
            cout << "Enter the position: ";
            cin >> pos;
            insertAtPos(head, val, pos);
            break;
        case 3:
            cout << "Enter the value: ";
            cin >> val;
            insertAtEnd(head, val);
            break;
        case 4:
            deletionAtStart(head);
            break;
        case 5:
            cout << "Enter the position: ";
            cin >> pos;
            deletionAtPos(head, pos);
            break;
```



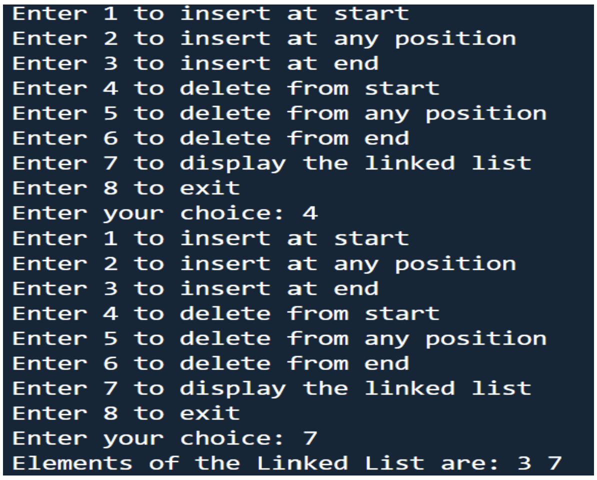
```
        case 6:
            deletionAtEnd(head);
            break;
        case 7:
            display(head);
            break;
        case 8:
            break;
        default:
            cout << "Invalid Choice!";
            break;
        }
    }
    return 0;
}
```



```
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 1
Enter the value: 2
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 1
Enter the value: 3
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 1
Enter the value: 4
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 7
Elements of the Linked List are: 4 3 2
```



```
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 3
Enter the value: 6
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 7
Elements of the Linked List are: 4 3 2 6
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 2
Enter the value: 7
Enter the position: 2
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 7
Elements of the Linked List are: 4 3 7
```



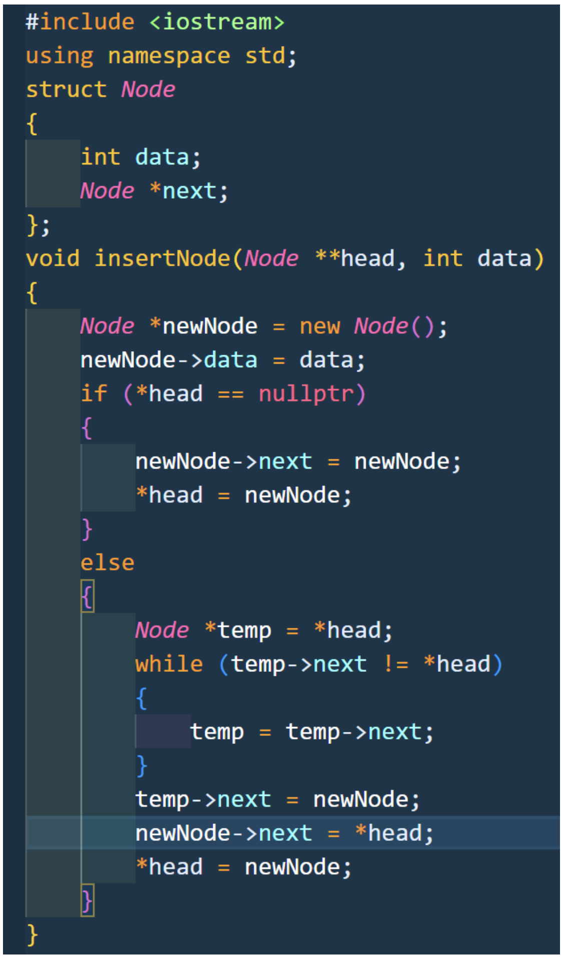
```
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 4
Enter 1 to insert at start
Enter 2 to insert at any position
Enter 3 to insert at end
Enter 4 to delete from start
Enter 5 to delete from any position
Enter 6 to delete from end
Enter 7 to display the linked list
Enter 8 to exit
Enter your choice: 7
Elements of the Linked List are: 3 7
```

10) Write a program in C++ to implement circular linked list and to perform:

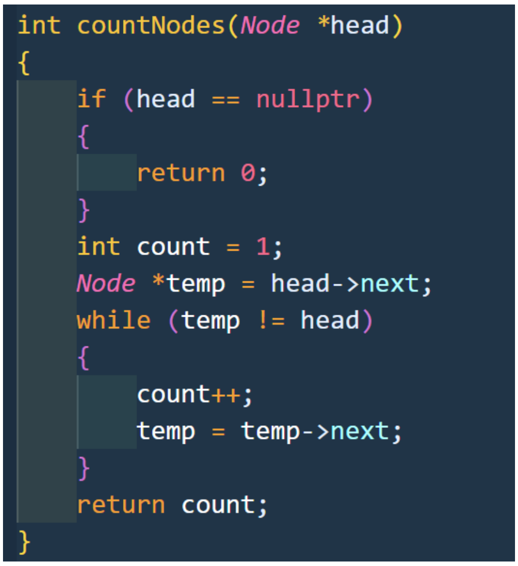
i. Insertion of node

ii. Counting number of nodes

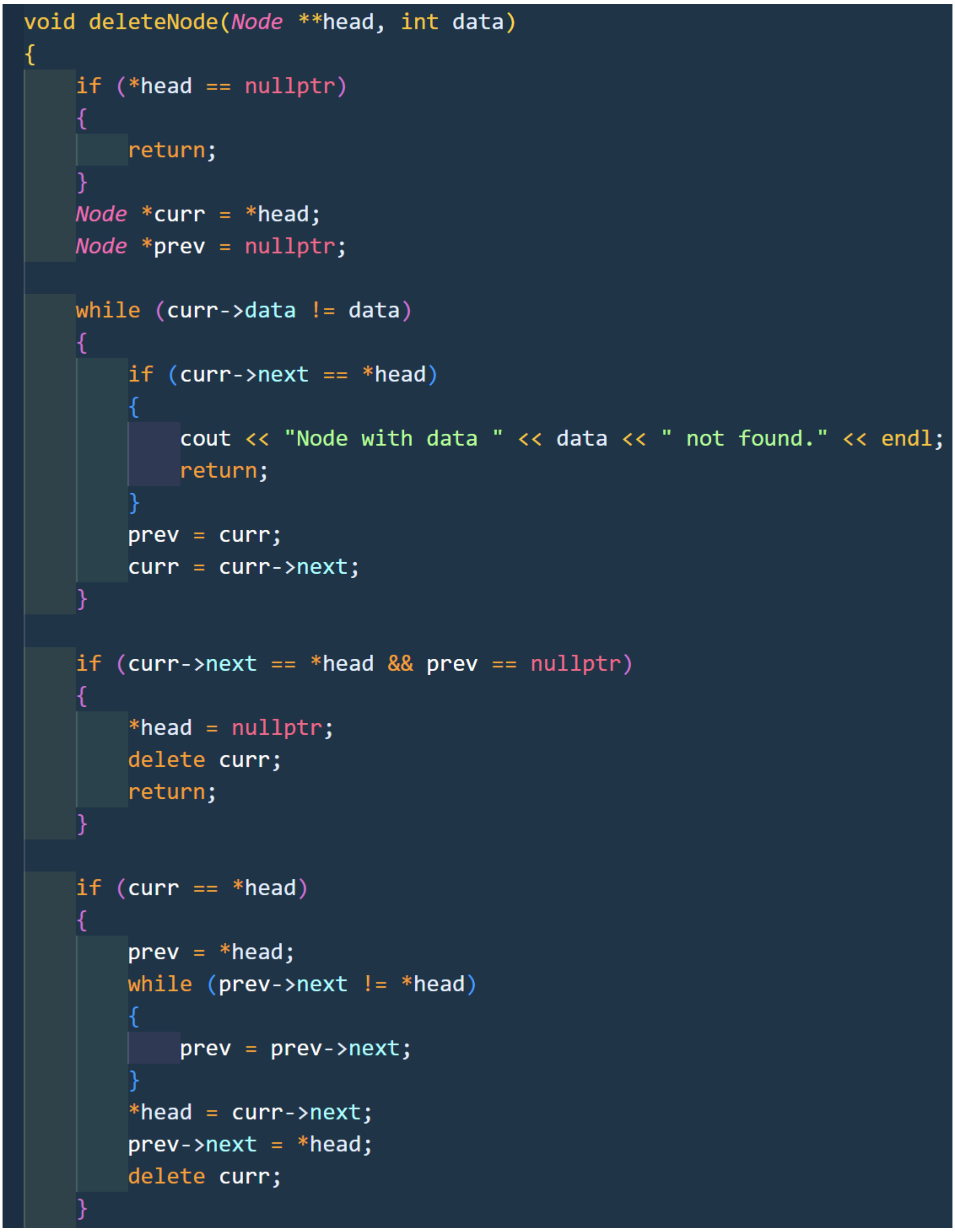
iii. Deletion of node



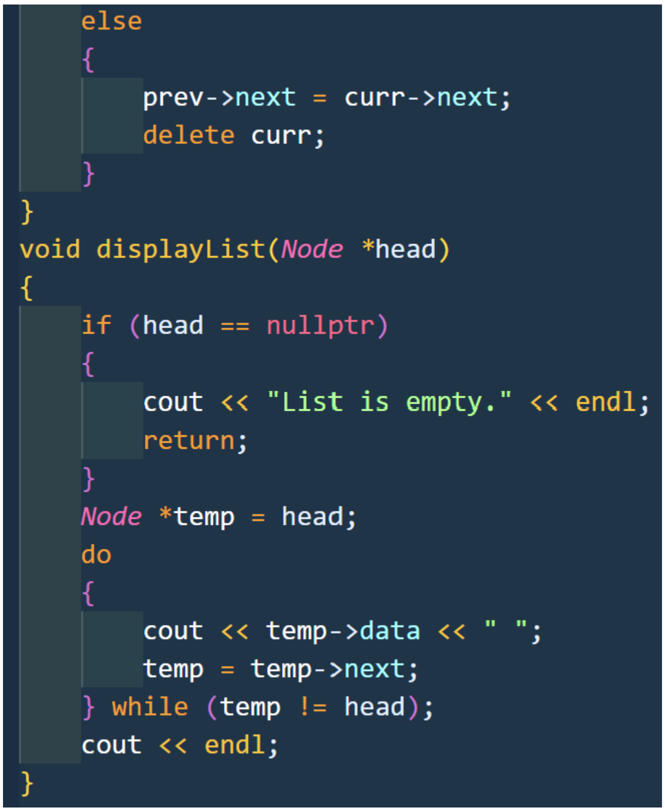
```cpp
#include <iostream>
using namespace std;
struct Node
{
    int data;
    Node *next;
};
void insertNode(Node **head, int data)
{
    Node *newNode = new Node();
    newNode->data = data;
    if (*head == nullptr)
    {
        newNode->next = newNode;
        *head = newNode;
    }
    else
    {
        Node *temp = *head;
        while (temp->next != *head)
        {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;
        *head = newNode;
    }
}
```



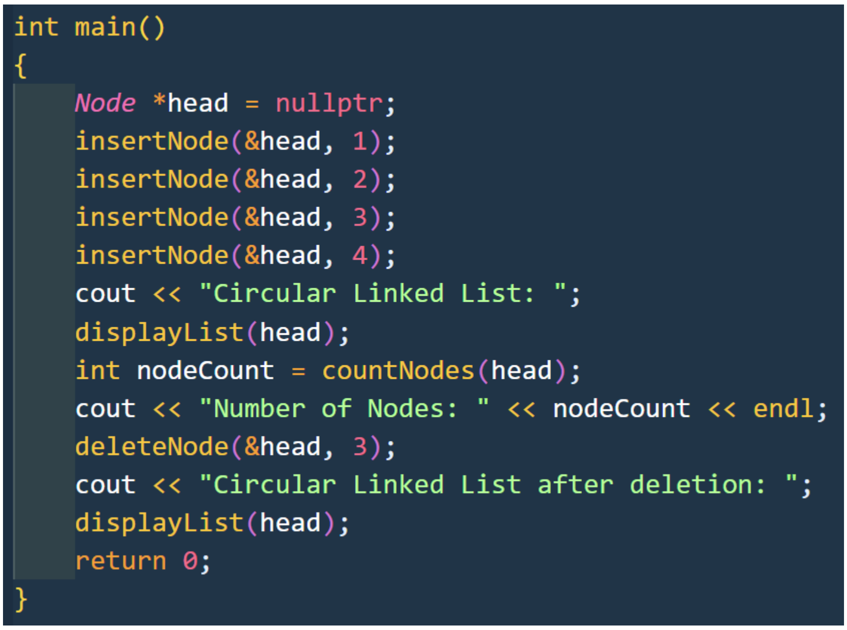
```cpp
int countNodes(Node *head)
{
    if (head == nullptr)
    {
        return 0;
    }
    int count = 1;
    Node *temp = head->next;
    while (temp != head)
    {
        count++;
        temp = temp->next;
    }
    return count;
}
```



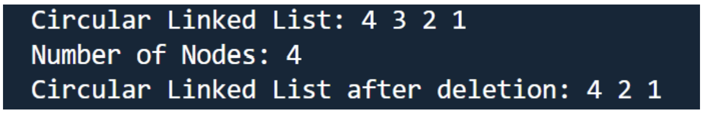
```
void deleteNode(Node **head, int data)
{
    if (*head == nullptr)
    {
        return;
    }
    Node *curr = *head;
    Node *prev = nullptr;

    while (curr->data != data)
    {
        if (curr->next == *head)
        {
            cout << "Node with data " << data << " not found." << endl;
            return;
        }
        prev = curr;
        curr = curr->next;
    }

    if (curr->next == *head && prev == nullptr)
    {
        *head = nullptr;
        delete curr;
        return;
    }

    if (curr == *head)
    {
        prev = *head;
        while (prev->next != *head)
        {
            prev = prev->next;
        }
        *head = curr->next;
        prev->next = *head;
        delete curr;
    }
```



```cpp
    else
    {
        prev->next = curr->next;
        delete curr;
    }
}
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;
}
```

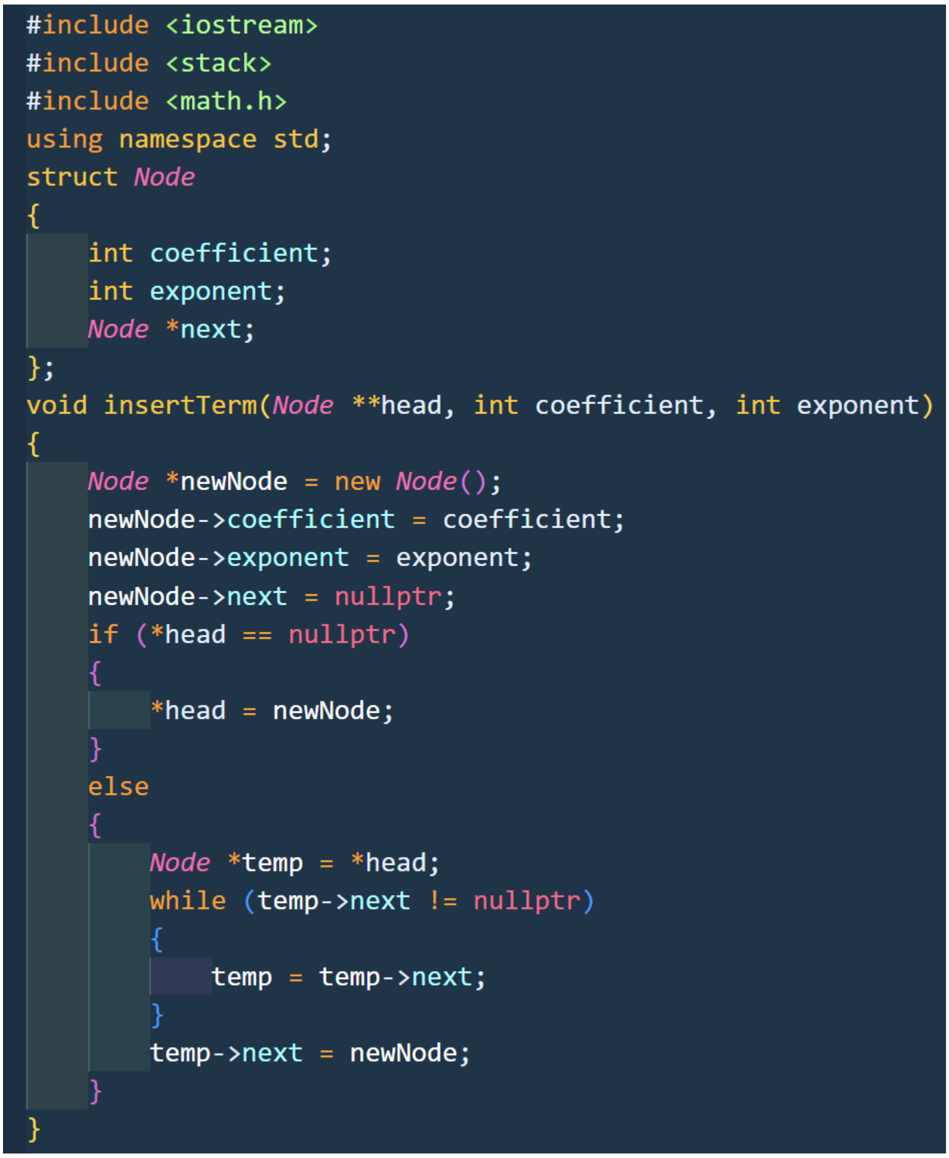


```cpp
int main()
{
    Node *head = nullptr;
    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);
    insertNode(&head, 4);
    cout << "Circular Linked List: ";
    displayList(head);
    int nodeCount = countNodes(head);
    cout << "Number of Nodes: " << nodeCount << endl;
    deleteNode(&head, 3);
    cout << "Circular Linked List after deletion: ";
    displayList(head);
    return 0;
}
```

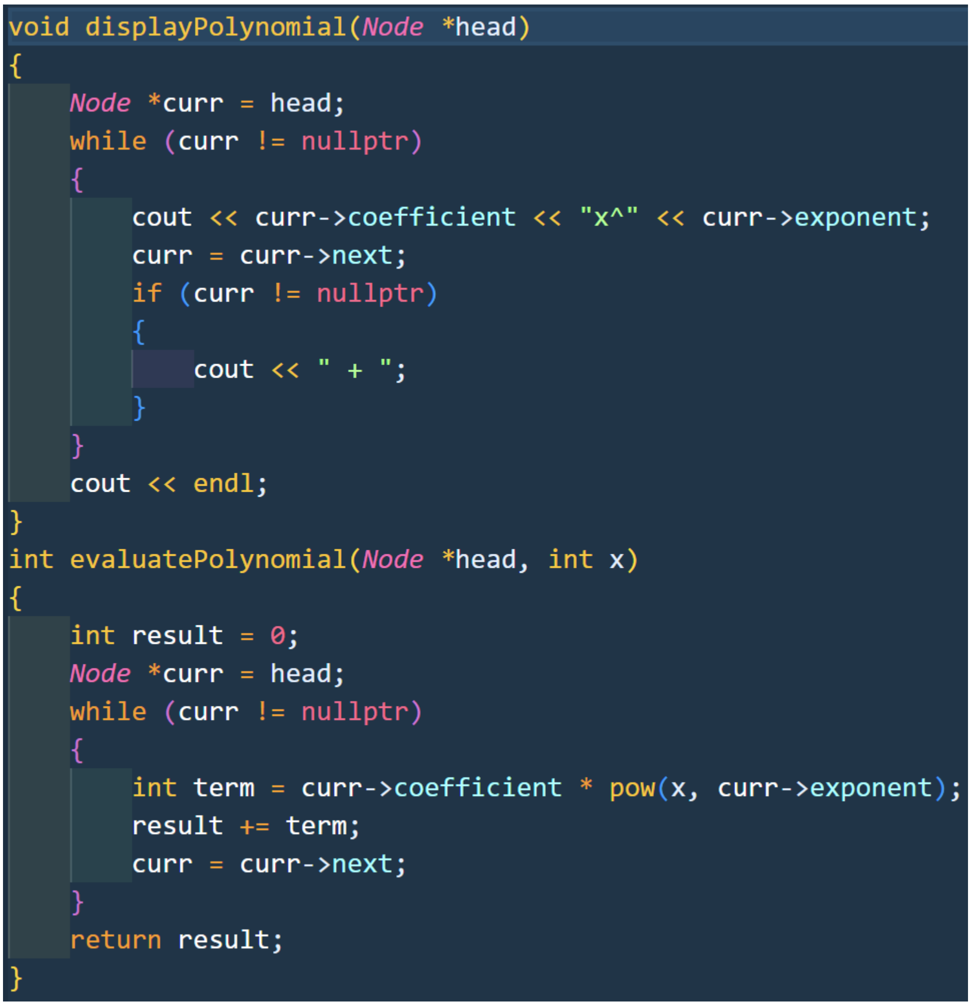


```
Circular Linked List: 4 3 2 1
Number of Nodes: 4
Circular Linked List after deletion: 4 2 1
```

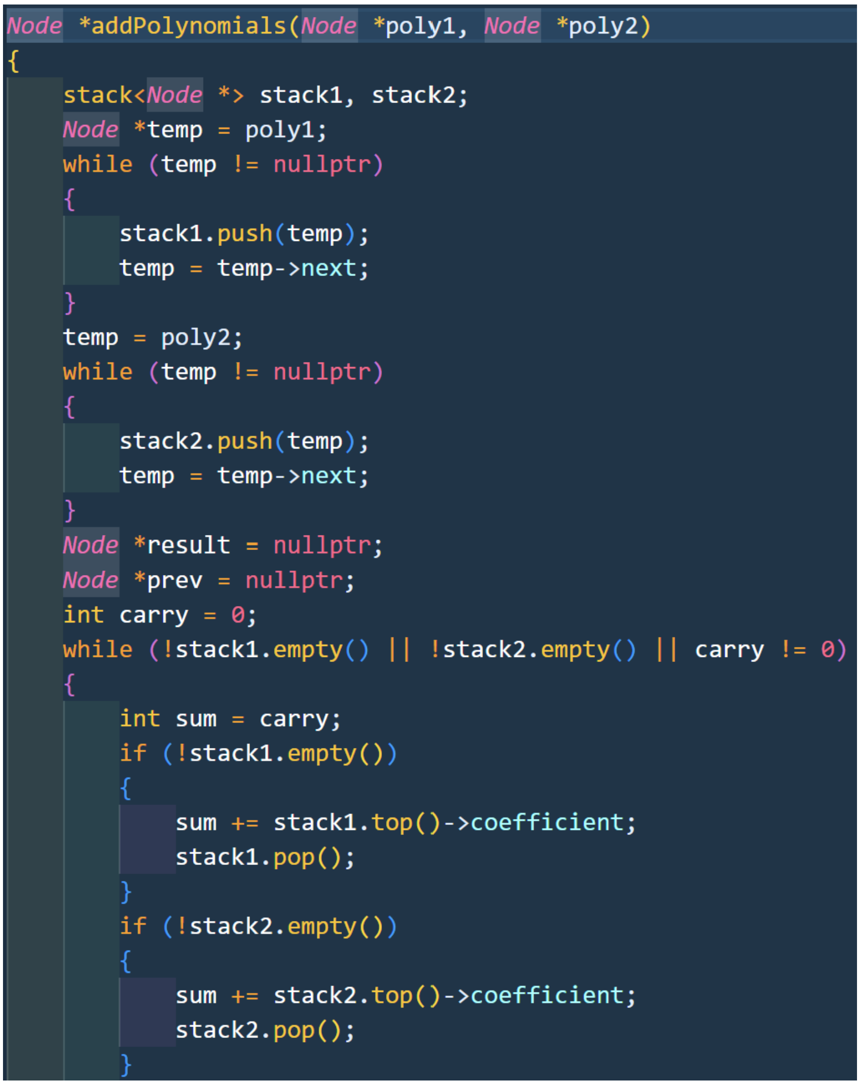
11) Write a program in C++ to implement the polynomials using stack and linked list.



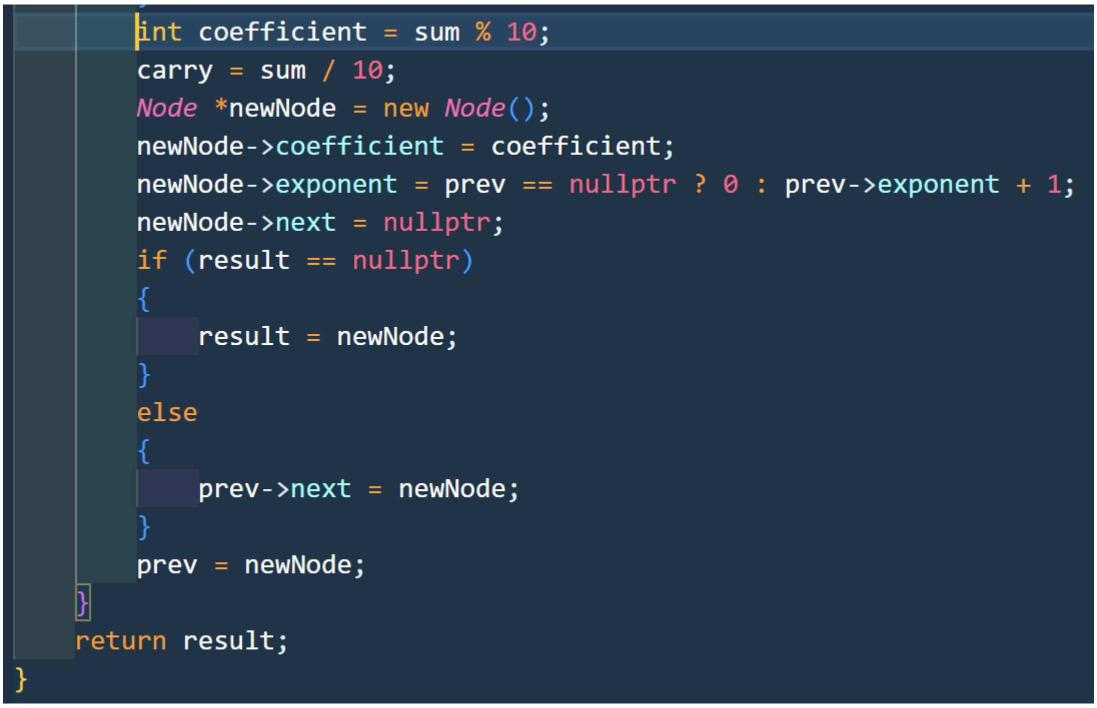
```
#include <iostream>
#include <stack>
#include <math.h>
using namespace std;
struct Node
{
    int coefficient;
    int exponent;
    Node *next;
};
void insertTerm(Node **head, int coefficient, int exponent)
{
    Node *newNode = new Node();
    newNode->coefficient = coefficient;
    newNode->exponent = exponent;
    newNode->next = nullptr;
    if (*head == nullptr)
    {
        *head = newNode;
    }
    else
    {
        Node *temp = *head;
        while (temp->next != nullptr)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```



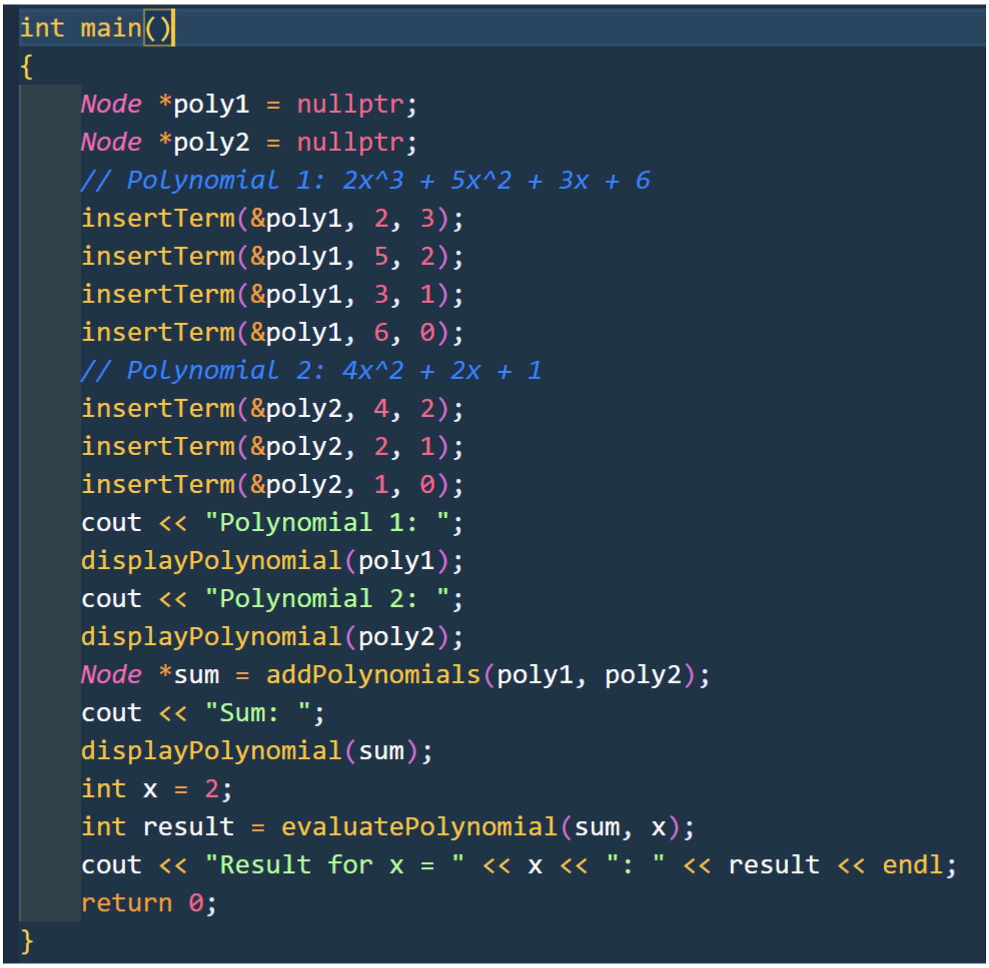
```cpp
void displayPolynomial(Node *head)
{
    Node *curr = head;
    while (curr != nullptr)
    {
        cout << curr->coefficient << "x^" << curr->exponent;
        curr = curr->next;
        if (curr != nullptr)
        {
            cout << " + ";
        }
    }
    cout << endl;
}
int evaluatePolynomial(Node *head, int x)
{
    int result = 0;
    Node *curr = head;
    while (curr != nullptr)
    {
        int term = curr->coefficient * pow(x, curr->exponent);
        result += term;
        curr = curr->next;
    }
    return result;
}
```



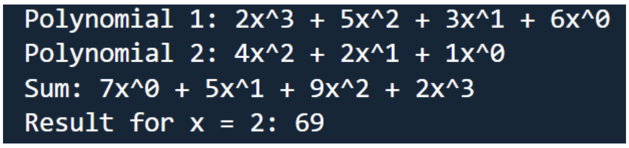
```cpp
Node *addPolynomials(Node *poly1, Node *poly2)
{
    stack<Node *> stack1, stack2;
    Node *temp = poly1;
    while (temp != nullptr)
    {
        stack1.push(temp);
        temp = temp->next;
    }
    temp = poly2;
    while (temp != nullptr)
    {
        stack2.push(temp);
        temp = temp->next;
    }
    Node *result = nullptr;
    Node *prev = nullptr;
    int carry = 0;
    while (!stack1.empty() || !stack2.empty() || carry != 0)
    {
        int sum = carry;
        if (!stack1.empty())
        {
            sum += stack1.top()->coefficient;
            stack1.pop();
        }
        if (!stack2.empty())
        {
            sum += stack2.top()->coefficient;
            stack2.pop();
        }
```



```cpp
        int coefficient = sum % 10;
        carry = sum / 10;
        Node *newNode = new Node();
        newNode->coefficient = coefficient;
        newNode->exponent = prev == nullptr ? 0 : prev->exponent + 1;
        newNode->next = nullptr;
        if (result == nullptr)
        {
            result = newNode;
        }
        else
        {
            prev->next = newNode;
        }
        prev = newNode;
    }
    return result;
}
```



```cpp
int main()
{
    Node *poly1 = nullptr;
    Node *poly2 = nullptr;
    // Polynomial 1: 2x^3 + 5x^2 + 3x + 6
    insertTerm(&poly1, 2, 3);
    insertTerm(&poly1, 5, 2);
    insertTerm(&poly1, 3, 1);
    insertTerm(&poly1, 6, 0);
    // Polynomial 2: 4x^2 + 2x + 1
    insertTerm(&poly2, 4, 2);
    insertTerm(&poly2, 2, 1);
    insertTerm(&poly2, 1, 0);
    cout << "Polynomial 1: ";
    displayPolynomial(poly1);
    cout << "Polynomial 2: ";
    displayPolynomial(poly2);
    Node *sum = addPolynomials(poly1, poly2);
    cout << "Sum: ";
    displayPolynomial(sum);
    int x = 2;
    int result = evaluatePolynomial(sum, x);
    cout << "Result for x = " << x << ": " << result << endl;
    return 0;
}
```



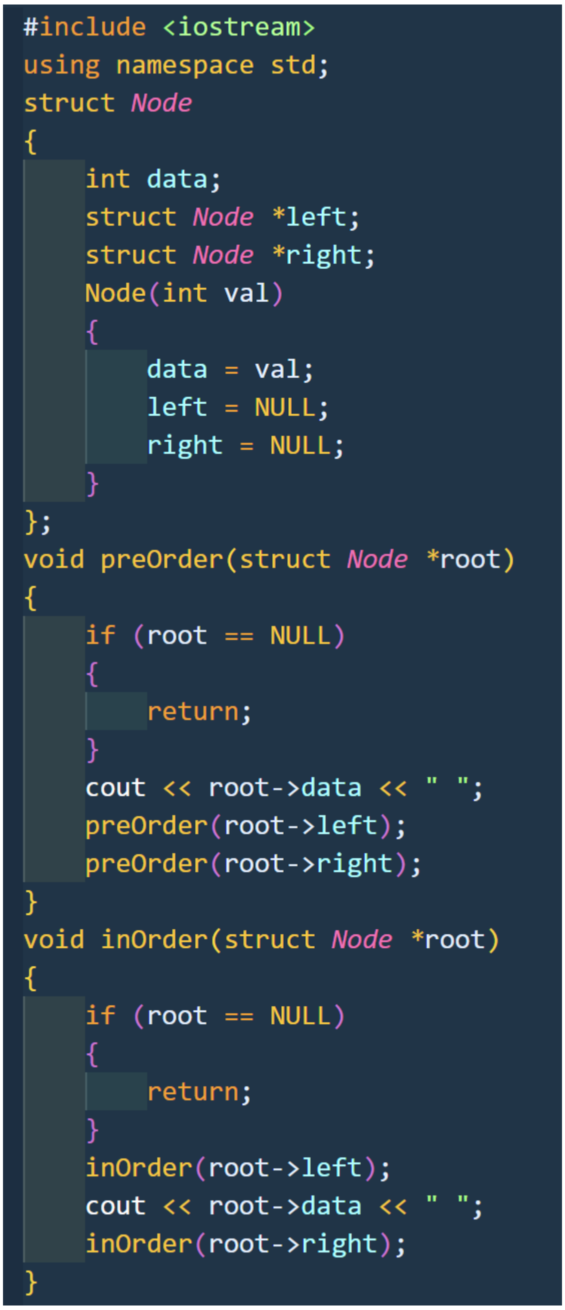
```
Polynomial 1: 2x^3 + 5x^2 + 3x^1 + 6x^0
Polynomial 2: 4x^2 + 2x^1 + 1x^0
Sum: 7x^0 + 5x^1 + 9x^2 + 2x^3
Result for x = 2: 69
```

12) Write a program in C++ to create a binary search tree and implement the following traversal algorithms:

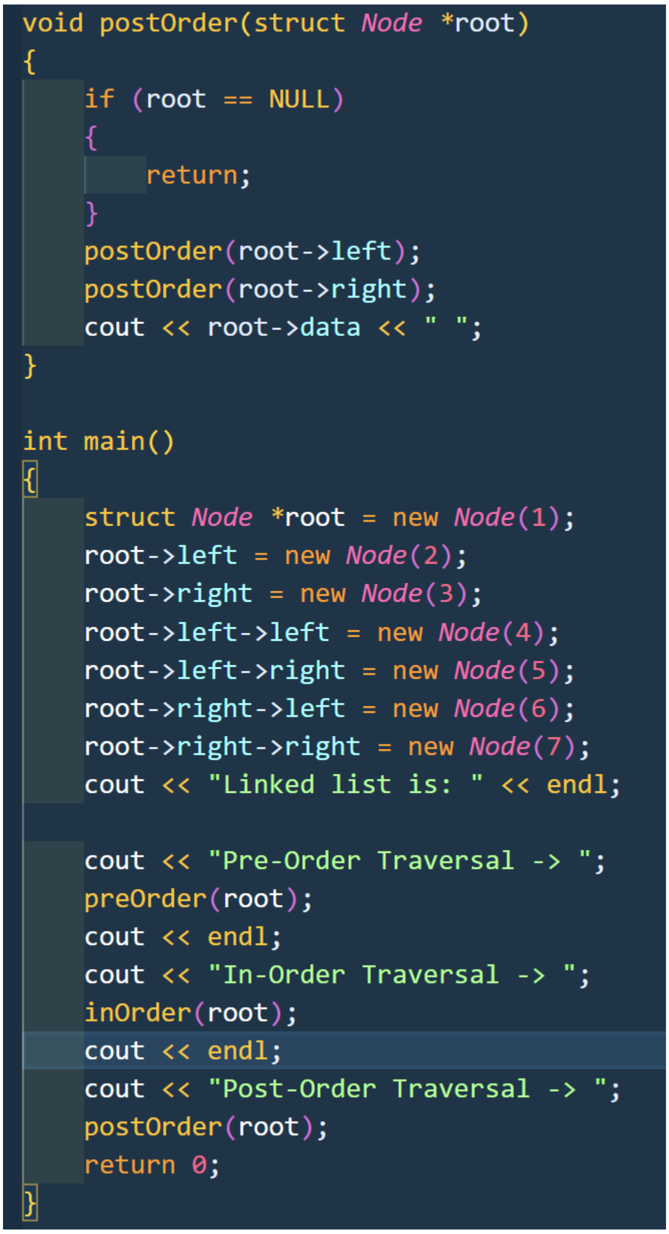
i. Inorder Traversal

ii. Preorder Traversal

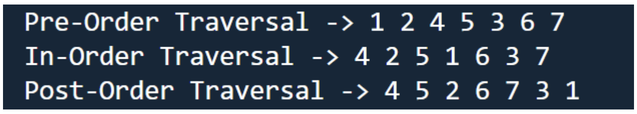
iii. Postorder Traversal



```cpp
#include <iostream>
using namespace std;
struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
    Node(int val)
    {
        data = val;
        left = NULL;
        right = NULL;
    }
};
void preOrder(struct Node *root)
{
    if (root == NULL)
    {
        return;
    }
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}
void inOrder(struct Node *root)
{
    if (root == NULL)
    {
        return;
    }
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}
```



```cpp
void postOrder(struct Node *root)
{
    if (root == NULL)
    {
        return;
    }
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}

int main()
{
    struct Node *root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);
    cout << "Linked list is: " << endl;

    cout << "Pre-Order Traversal -> ";
    preOrder(root);
    cout << endl;
    cout << "In-Order Traversal -> ";
    inOrder(root);
    cout << endl;
    cout << "Post-Order Traversal -> ";
    postOrder(root);
    return 0;
}
```



```
Pre-Order Traversal -> 1 2 4 5 3 6 7
In-Order Traversal -> 4 2 5 1 6 3 7
Post-Order Traversal -> 4 5 2 6 7 3 1
```

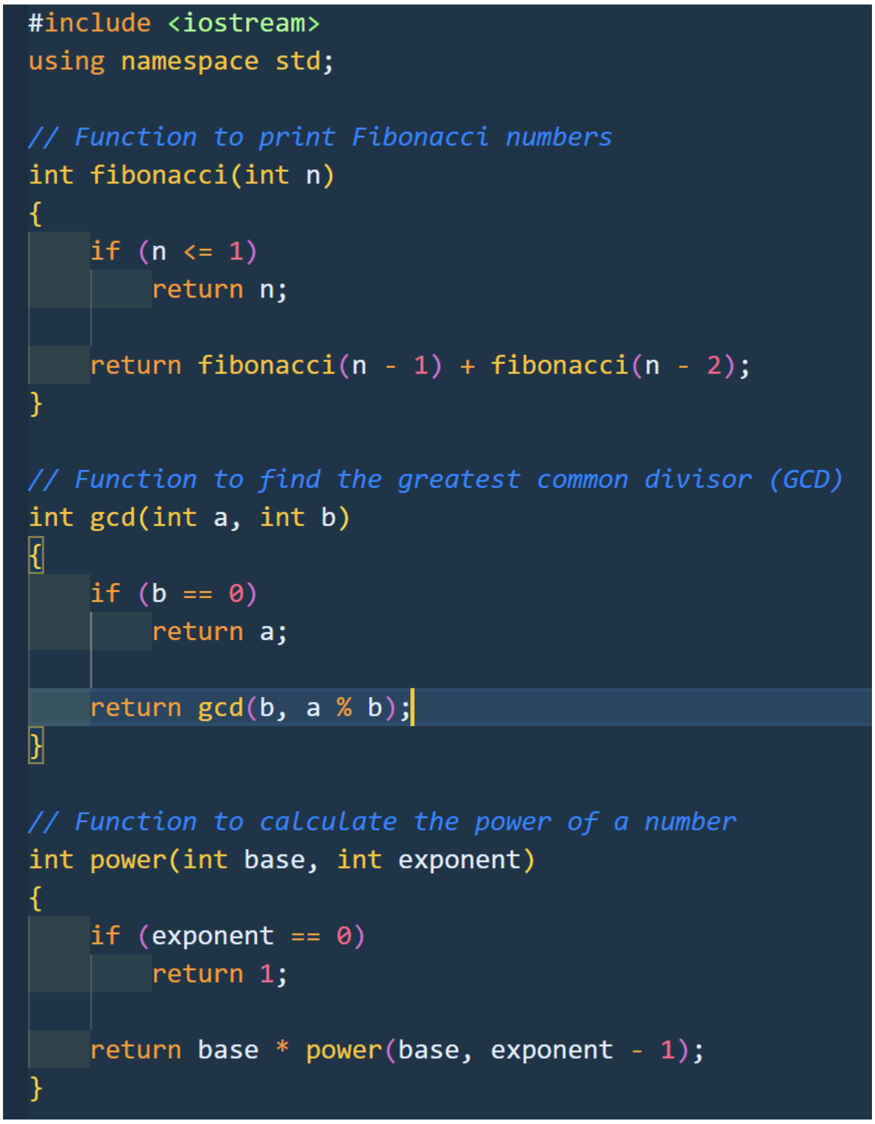
13) Write a program in C++ to implement the following recursion:

i. Print Fibonacci numbers

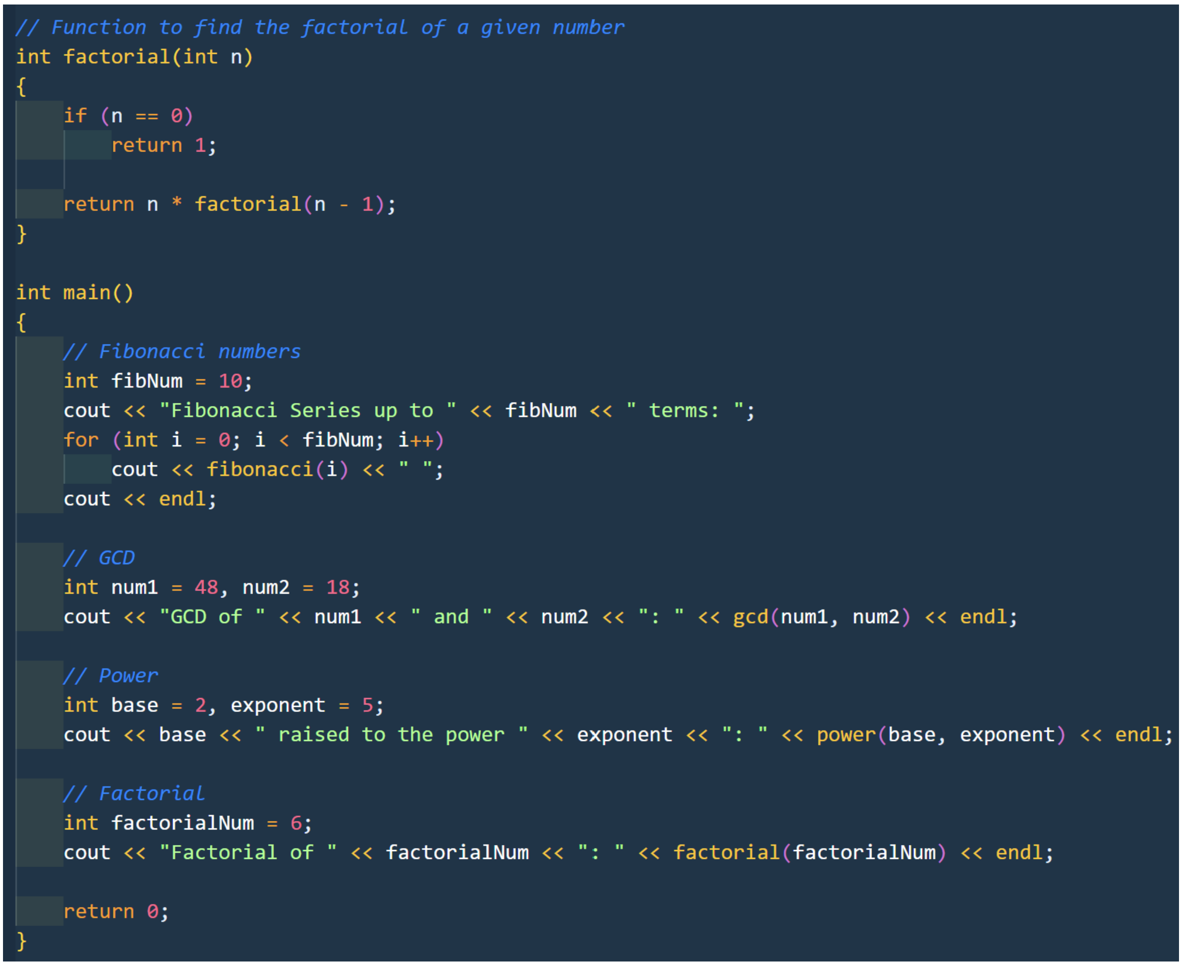
ii. To find the greatest common divisor

iii. To calculate the power of a number

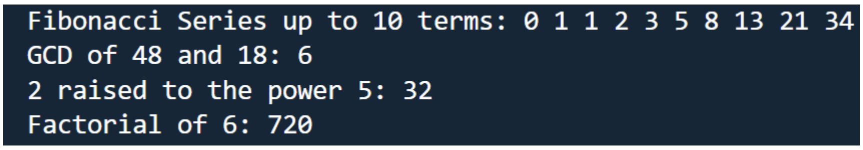
iv. To find out factorial of a given number



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

// Function to print Fibonacci numbers
int fibonacci(int n)
{
    if (n <= 1)
        return n;

    return fibonacci(n - 1) + fibonacci(n - 2);
}

// Function to find the greatest common divisor (GCD)
int gcd(int a, int b)
{
    if (b == 0)
        return a;

    return gcd(b, a % b);
}

// Function to calculate the power of a number
int power(int base, int exponent)
{
    if (exponent == 0)
        return 1;

    return base * power(base, exponent - 1);
}
```



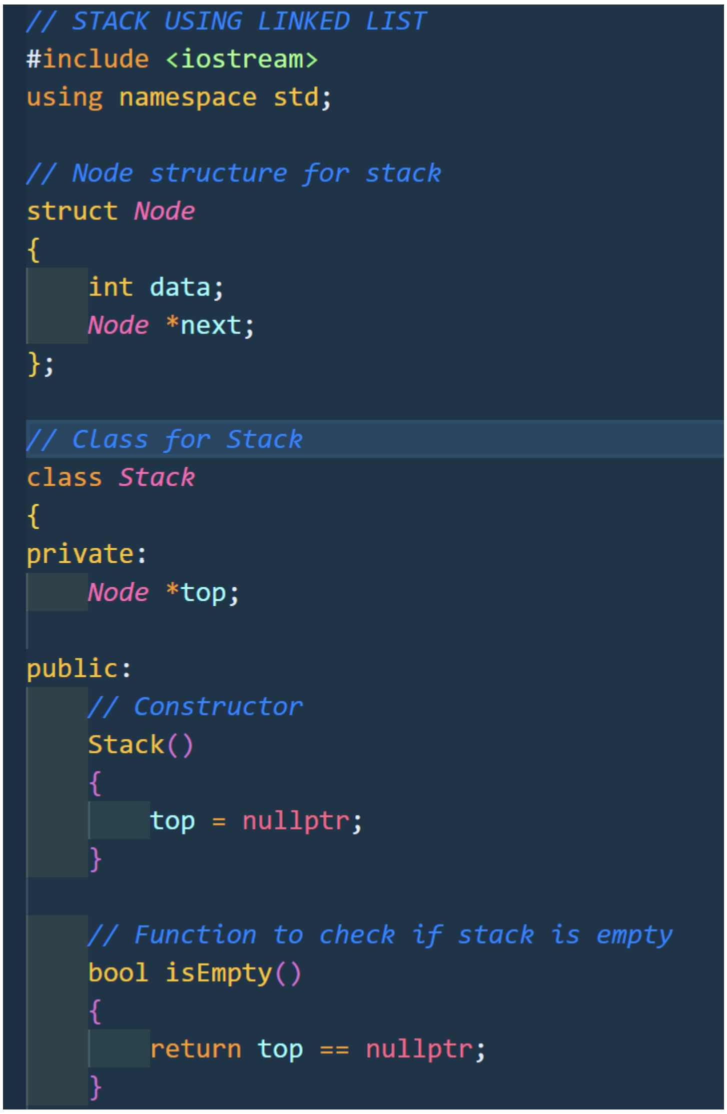
```cpp
// Function to find the factorial of a given number
int factorial(int n)
{
    if (n == 0)
        return 1;

    return n * factorial(n - 1);
}

int main()
{
    // Fibonacci numbers
    int fibNum = 10;
    cout << "Fibonacci Series up to " << fibNum << " terms: ";
    for (int i = 0; i < fibNum; i++)
        cout << fibonacci(i) << " ";
    cout << endl;

    // GCD
    int num1 = 48, num2 = 18;
    cout << "GCD of " << num1 << " and " << num2 << ": " << gcd(num1, num2) << endl;

    // Power
    int base = 2, exponent = 5;
    cout << base << " raised to the power " << exponent << ": " << power(base, exponent) << endl;

    // Factorial
    int factorialNum = 6;
    cout << "Factorial of " << factorialNum << ": " << factorial(factorialNum) << endl;

    return 0;
}
```



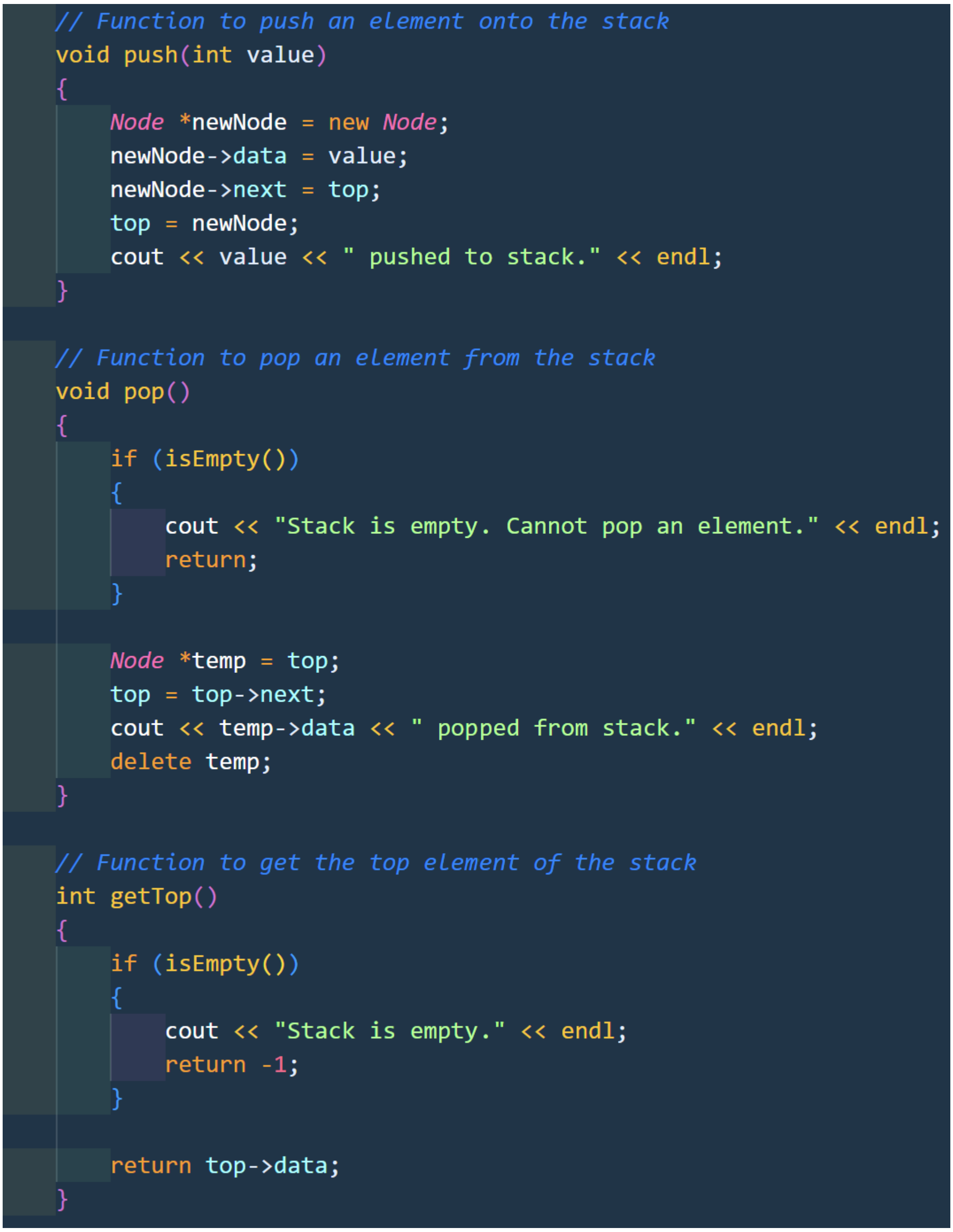
```
Fibonacci Series up to 10 terms: 0 1 1 2 3 5 8 13 21 34
GCD of 48 and 18: 6
2 raised to the power 5: 32
Factorial of 6: 720
```

14) Write a program in C++ to implement queue and stack using linked list.

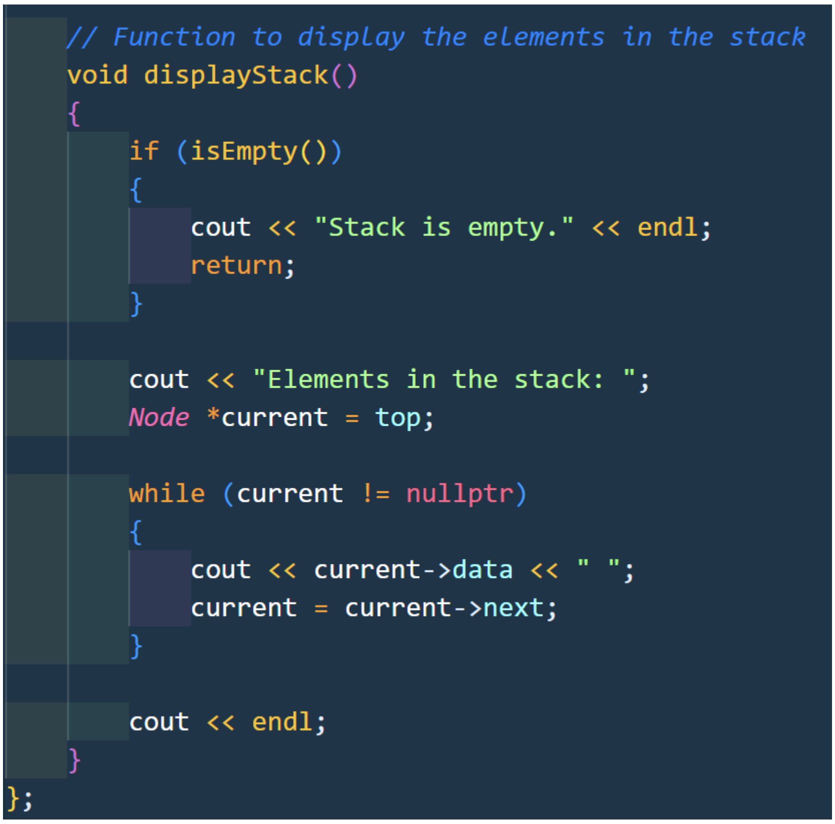
→**Stack:**



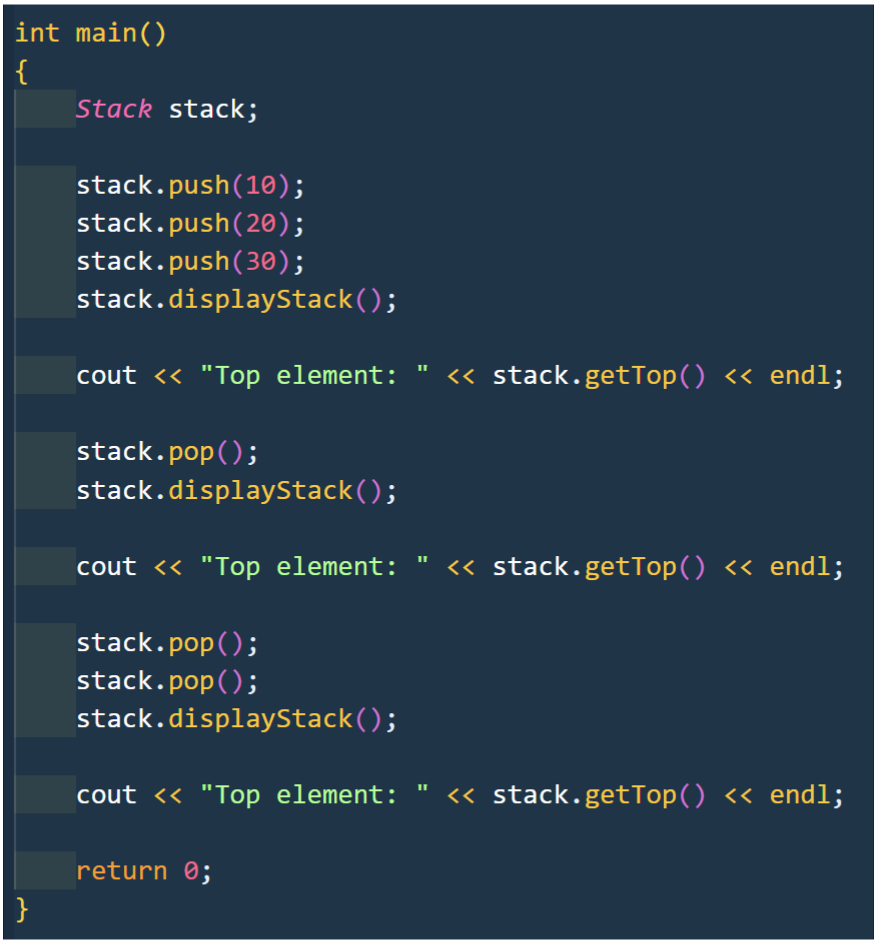
```
// STACK USING LINKED LIST
#include <iostream>
using namespace std;

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

// Class for Stack
class Stack
{
private:
    Node *top;

public:
    // Constructor
    Stack()
    {
        top = nullptr;
    }

    // Function to check if stack is empty
    bool isEmpty()
    {
        return top == nullptr;
    }
```



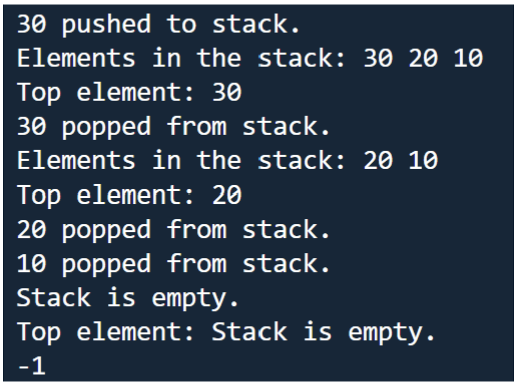
```
// Function to push an element onto the stack
void push(int value)
{
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    cout << value << " pushed to stack." << endl;
}

// Function to pop an element from the stack
void pop()
{
    if (isEmpty())
    {
        cout << "Stack is empty. Cannot pop an element." << endl;
        return;
    }

    Node *temp = top;
    top = top->next;
    cout << temp->data << " popped from stack." << endl;
    delete temp;
}

// Function to get the top element of the stack
int getTop()
{
    if (isEmpty())
    {
        cout << "Stack is empty." << endl;
        return -1;
    }

    return top->data;
}
```



```
    // Function to display the elements in the stack
    void displayStack()
    {
        if (isEmpty())
        {
            cout << "Stack is empty." << endl;
            return;
        }

        cout << "Elements in the stack: ";
        Node *current = top;

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

        cout << endl;
    }
};
```

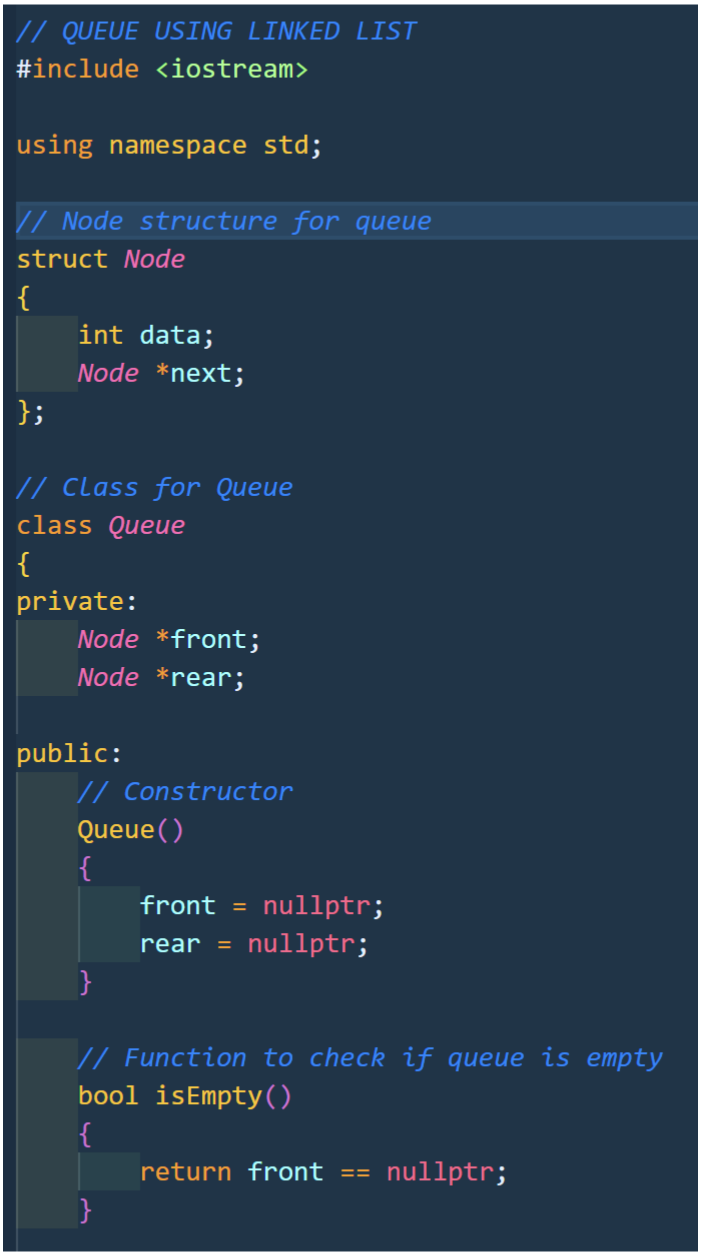


```
int main()
{
    Stack stack;

    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.displayStack();

    cout << "Top element: " << stack.getTop() << endl;

    stack.pop();
    stack.displayStack();

    cout << "Top element: " << stack.getTop() << endl;

    stack.pop();
    stack.pop();
    stack.displayStack();

    cout << "Top element: " << stack.getTop() << endl;

    return 0;
}
```

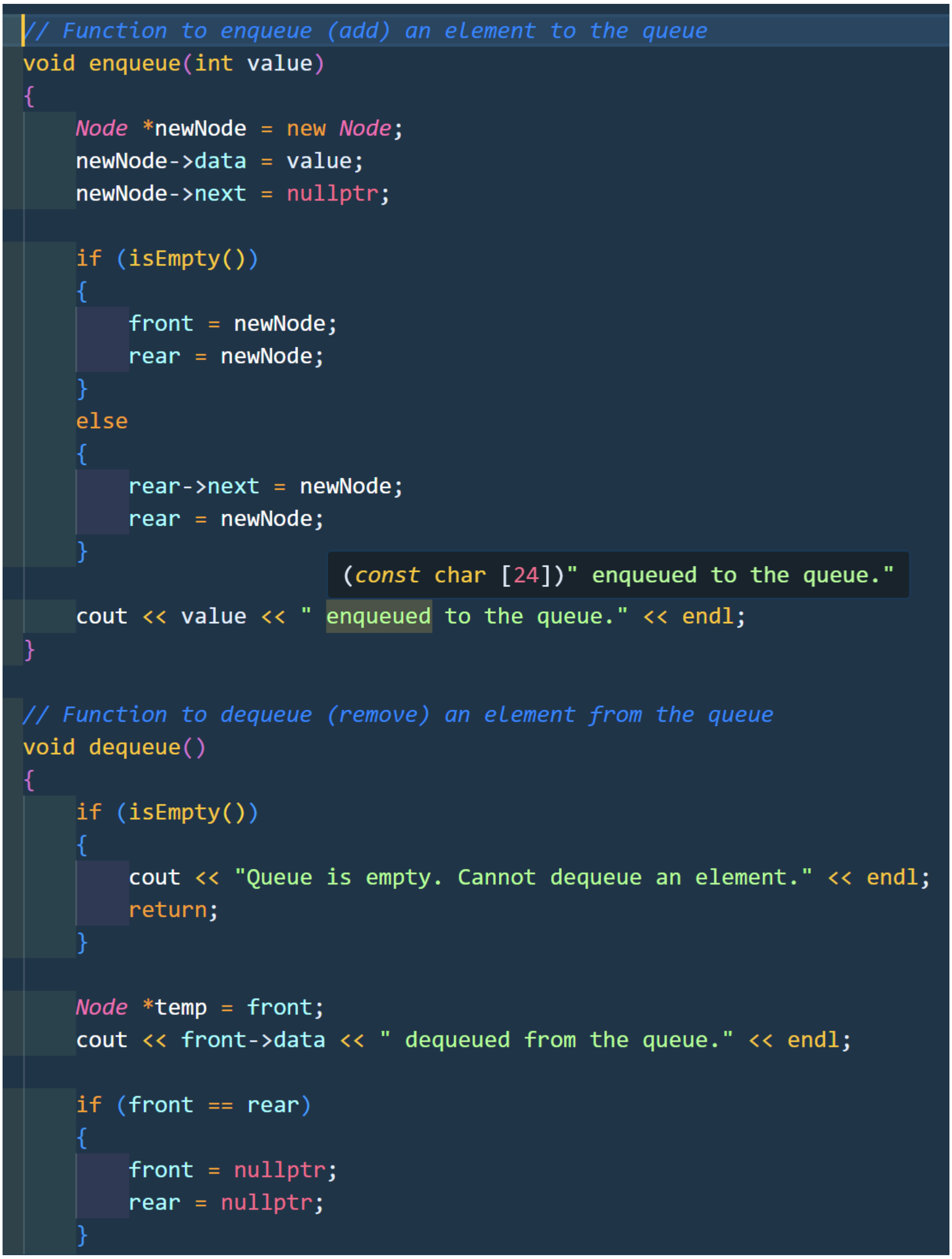


```
30 pushed to stack.
Elements in the stack: 30 20 10
Top element: 30
30 popped from stack.
Elements in the stack: 20 10
Top element: 20
20 popped from stack.
10 popped from stack.
Stack is empty.
Top element: Stack is empty.
-1
```

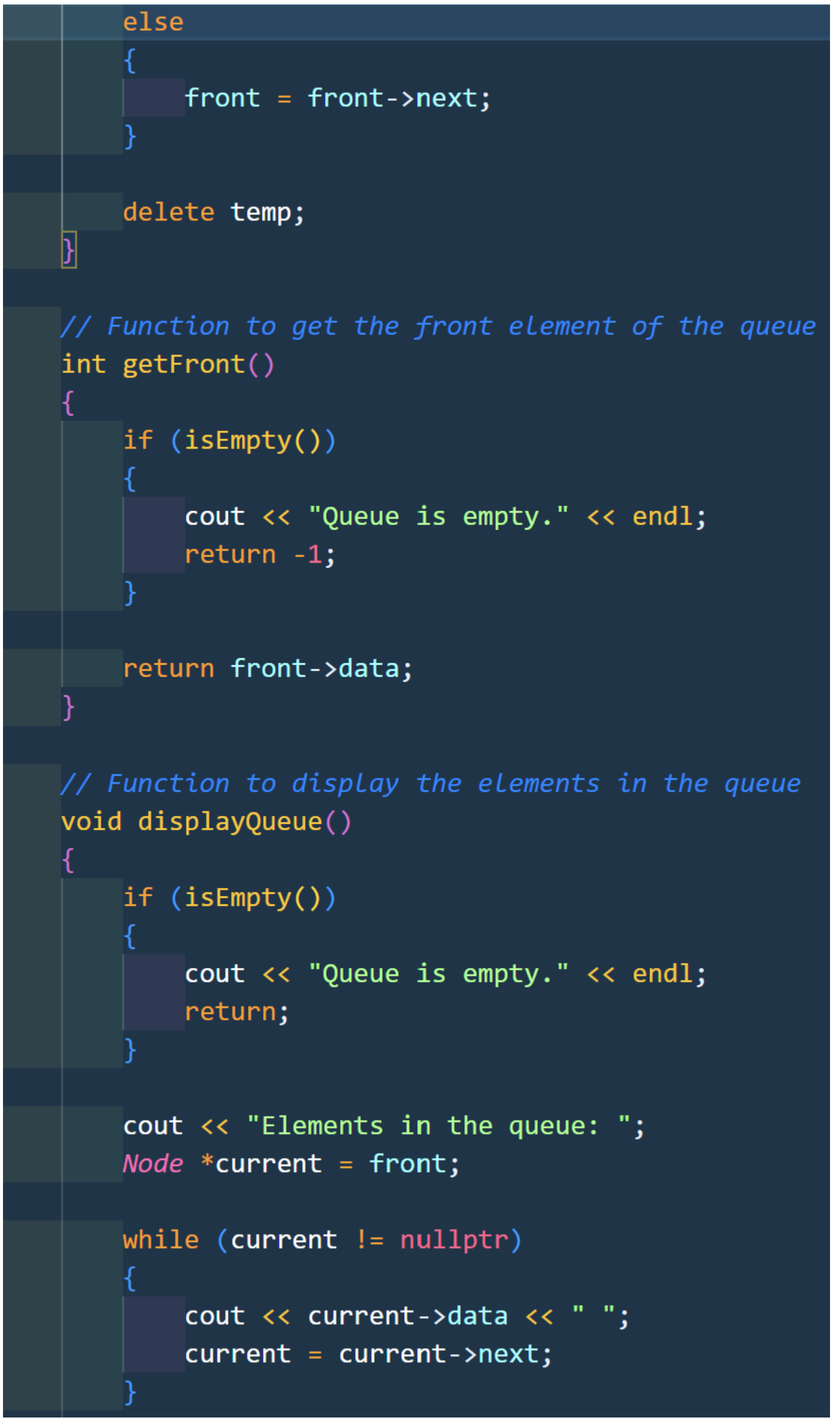
→Queue:



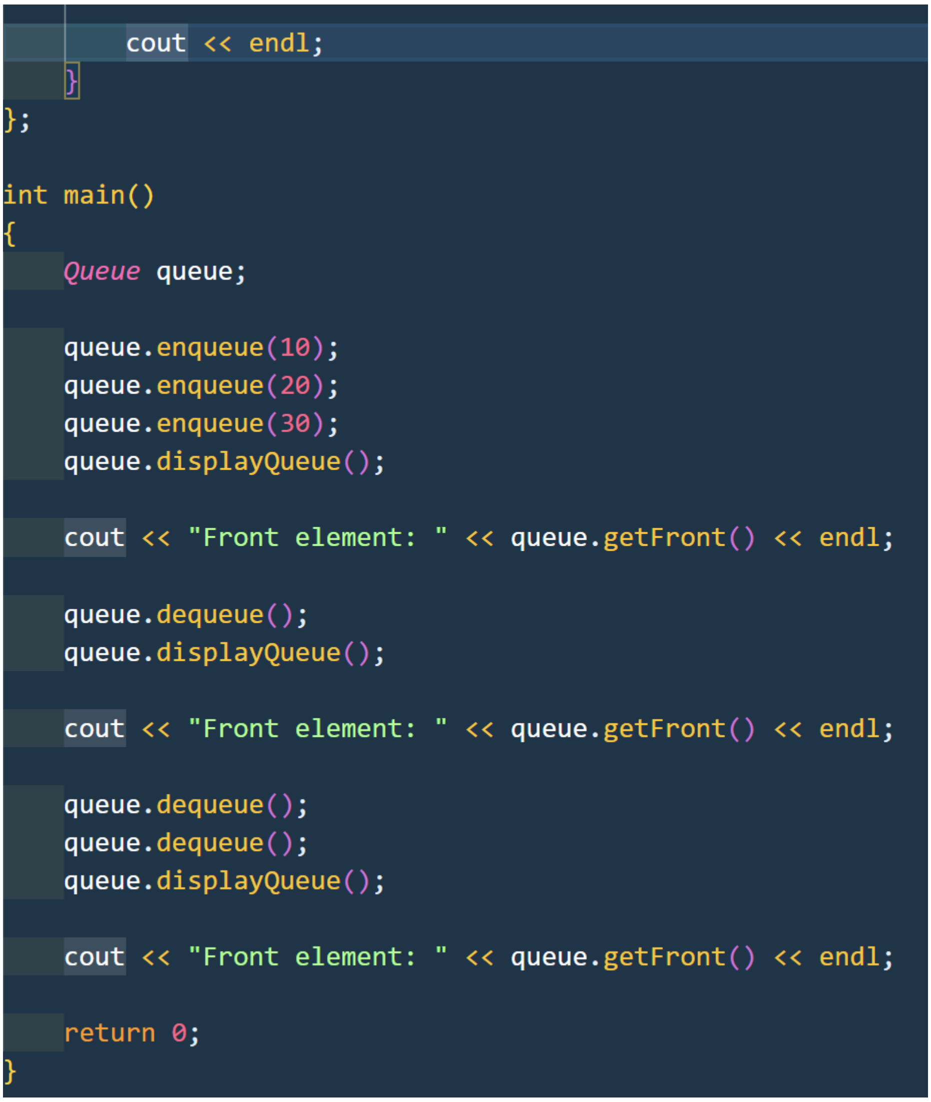
```
// QUEUE USING LINKED LIST
#include <iostream>

using namespace std;

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

// Class for Queue
class Queue
{
private:
    Node *front;
    Node *rear;

public:
    // Constructor
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }

    // Function to check if queue is empty
    bool isEmpty()
    {
        return front == nullptr;
    }
```



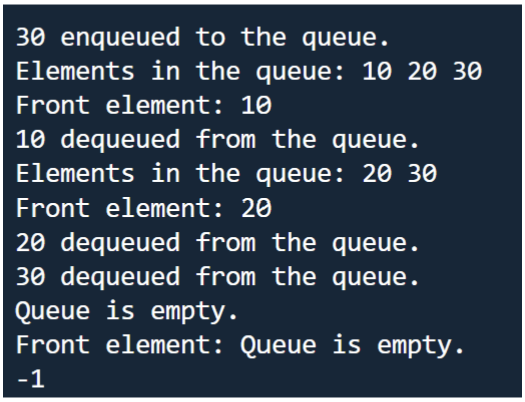
```
// Function to enqueue (add) an element to the queue
void enqueue(int value)
{
    Node *newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (isEmpty())
    {
        front = newNode;
        rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
                       (const char [24])" enqueued to the queue."
    cout << value << " enqueued to the queue." << endl;
}

// Function to dequeue (remove) an element from the queue
void dequeue()
{
    if (isEmpty())
    {
        cout << "Queue is empty. Cannot dequeue an element." << endl;
        return;
    }

    Node *temp = front;
    cout << front->data << " dequeued from the queue." << endl;

    if (front == rear)
    {
        front = nullptr;
        rear = nullptr;
    }
```



```cpp
        else
        {
            front = front->next;
        }

        delete temp;
    }

    // Function to get the front element of the queue
    int getFront()
    {
        if (isEmpty())
        {
            cout << "Queue is empty." << endl;
            return -1;
        }

        return front->data;
    }

    // Function to display the elements in the queue
    void displayQueue()
    {
        if (isEmpty())
        {
            cout << "Queue is empty." << endl;
            return;
        }

        cout << "Elements in the queue: ";
        Node *current = front;

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



```
        cout << endl;
    }
};

int main()
{
    Queue queue;

    queue.enqueue(10);
    queue.enqueue(20);
    queue.enqueue(30);
    queue.displayQueue();

    cout << "Front element: " << queue.getFront() << endl;

    queue.dequeue();
    queue.displayQueue();

    cout << "Front element: " << queue.getFront() << endl;

    queue.dequeue();
    queue.dequeue();
    queue.displayQueue();

    cout << "Front element: " << queue.getFront() << endl;

    return 0;
}
```



```
30 enqueued to the queue.
Elements in the queue: 10 20 30
Front element: 10
10 dequeued from the queue.
Elements in the queue: 20 30
Front element: 20
20 dequeued from the queue.
30 dequeued from the queue.
Queue is empty.
Front element: Queue is empty.
-1
```