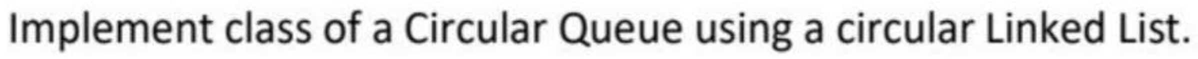
DSA LAB 03

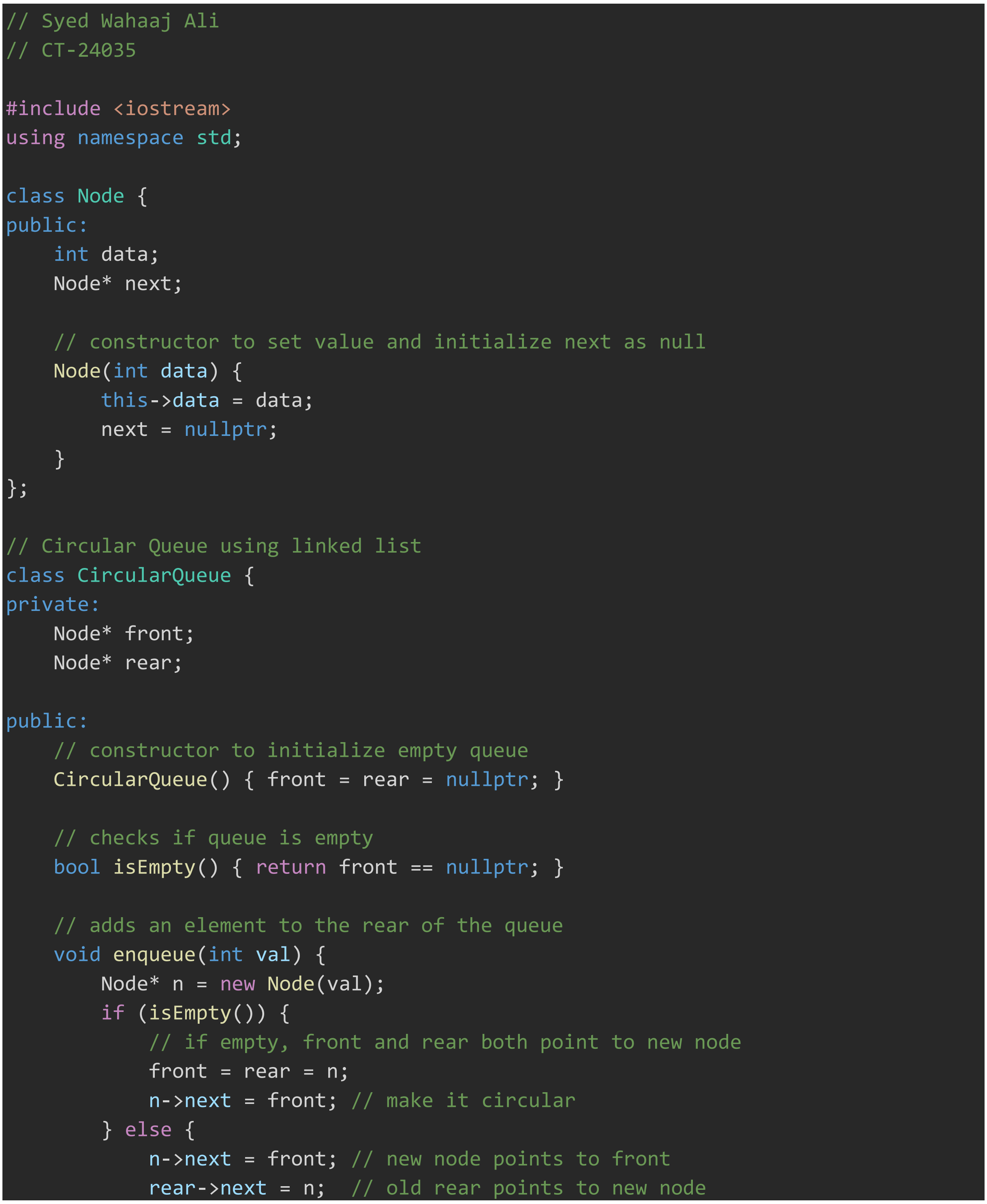
Name: *Syed Wahaaj Ali*
Roll No: *CT-24035*
Course Code: *CT-159*
CSIT
Section: *A*

Q1:

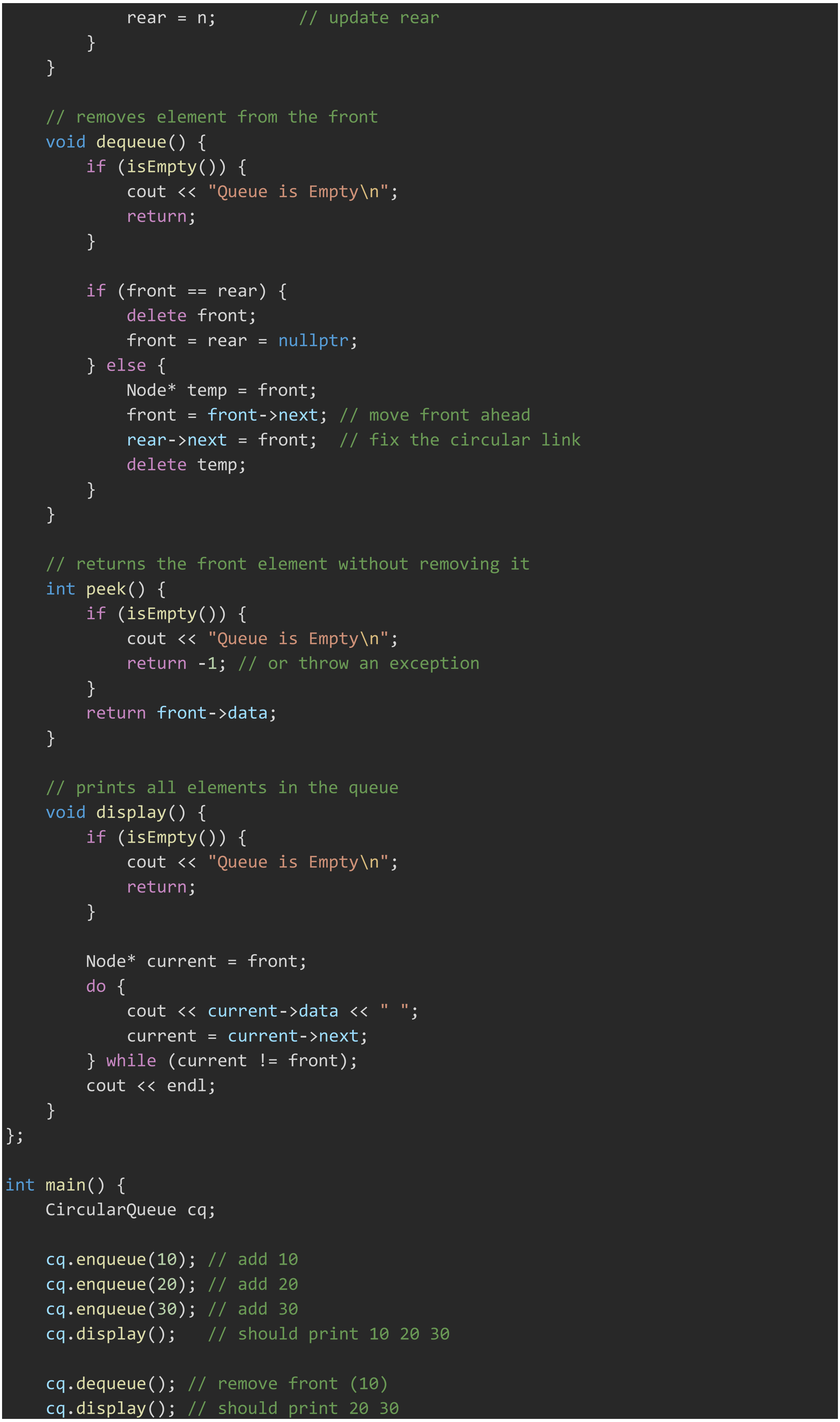


Implement class of a Circular Queue using a circular Linked List.

Code:

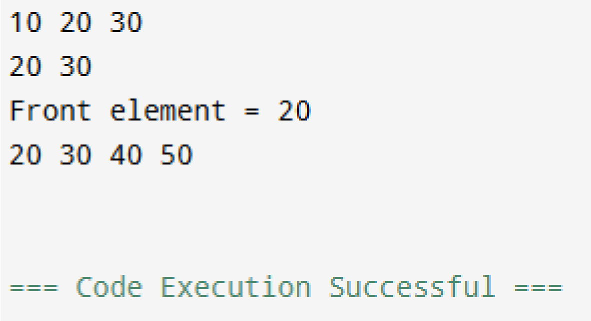


```
// Syed Wahaaj Ali
// CT-24035

#include <iostream>
using namespace std;

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

    // constructor to set value and initialize next as null
    Node(int data) {
        this->data = data;
        next = nullptr;
    }
};

// Circular Queue using linked list
class CircularQueue {
private:
    Node* front;
    Node* rear;

public:
    // constructor to initialize empty queue
    CircularQueue() { front = rear = nullptr; }

    // checks if queue is empty
    bool isEmpty() { return front == nullptr; }

    // adds an element to the rear of the queue
    void enqueue(int val) {
        Node* n = new Node(val);
        if (isEmpty()) {
            // if empty, front and rear both point to new node
            front = rear = n;
            n->next = front; // make it circular
        } else {
            n->next = front; // new node points to front
            rear->next = n;  // old rear points to new node
```



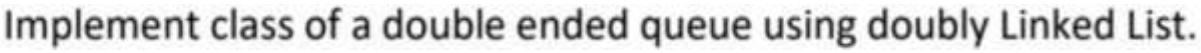
```
            rear = n;        // update rear
        }
    }

    // removes element from the front
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is Empty\n";
            return;
        }

        if (front == rear) {
            delete front;
            front = rear = nullptr;
        } else {
            Node* temp = front;
            front = front->next; // move front ahead
            rear->next = front;  // fix the circular link
            delete temp;
        }
    }

    // returns the front element without removing it
    int peek() {
        if (isEmpty()) {
            cout << "Queue is Empty\n";
            return -1; // or throw an exception
        }
        return front->data;
    }

    // prints all elements in the queue
    void display() {
        if (isEmpty()) {
            cout << "Queue is Empty\n";
            return;
        }

        Node* current = front;
        do {
            cout << current->data << " ";
            current = current->next;
        } while (current != front);
        cout << endl;
    }
};

int main() {
    CircularQueue cq;

    cq.enqueue(10); // add 10
    cq.enqueue(20); // add 20
    cq.enqueue(30); // add 30
    cq.display();   // should print 10 20 30

    cq.dequeue(); // remove front (10)
    cq.display(); // should print 20 30
    cout << "Front element = " << cq.peek() << endl;

    cq.enqueue(40); // add 40
    cq.enqueue(50); // add 50
    cq.display();   // should print 20 30 40 50

    return 0;
}
```

Output:



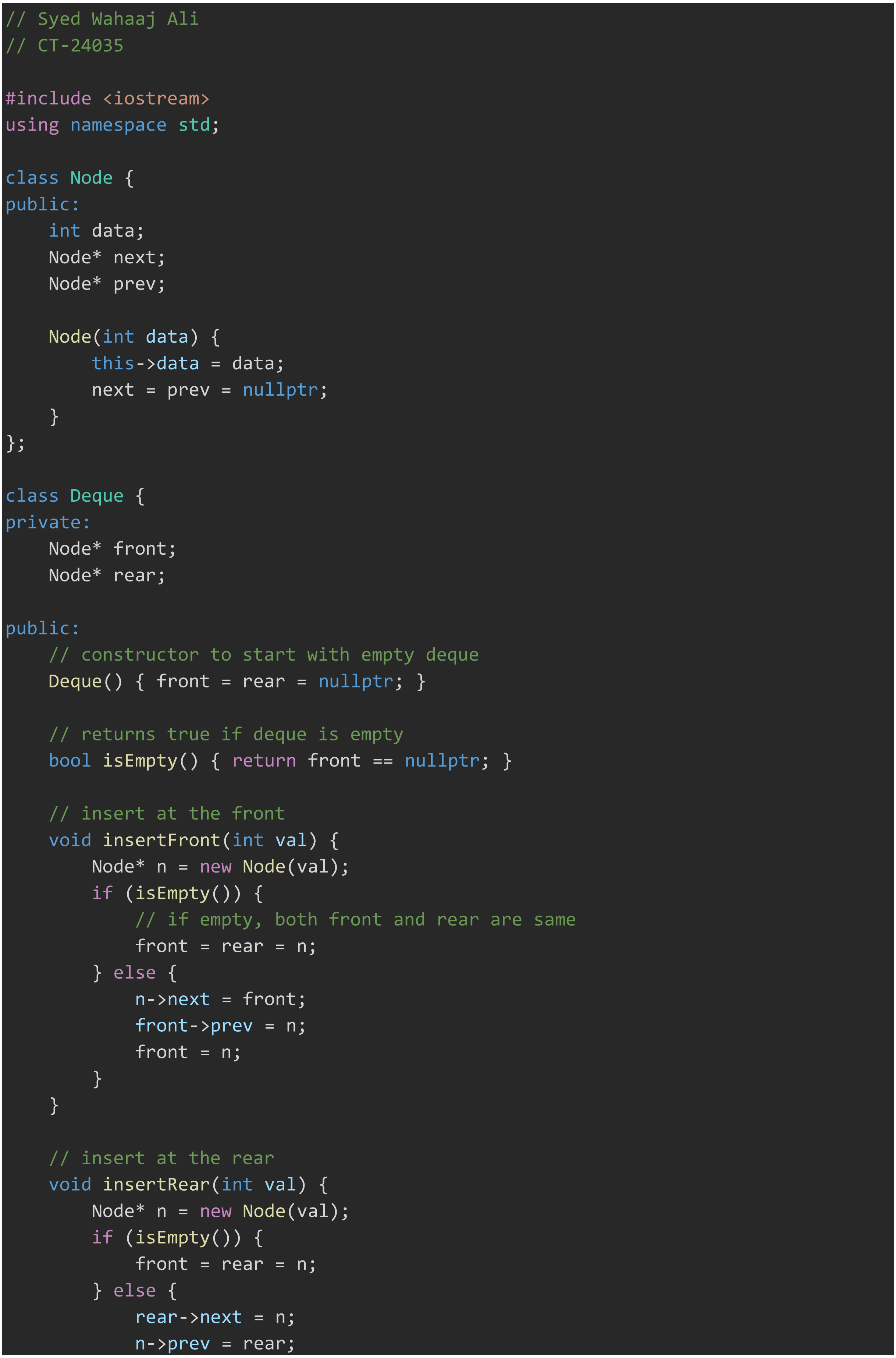
```
10 20 30
20 30
Front element = 20
20 30 40 50


=== Code Execution Successful ===
```

Q2:

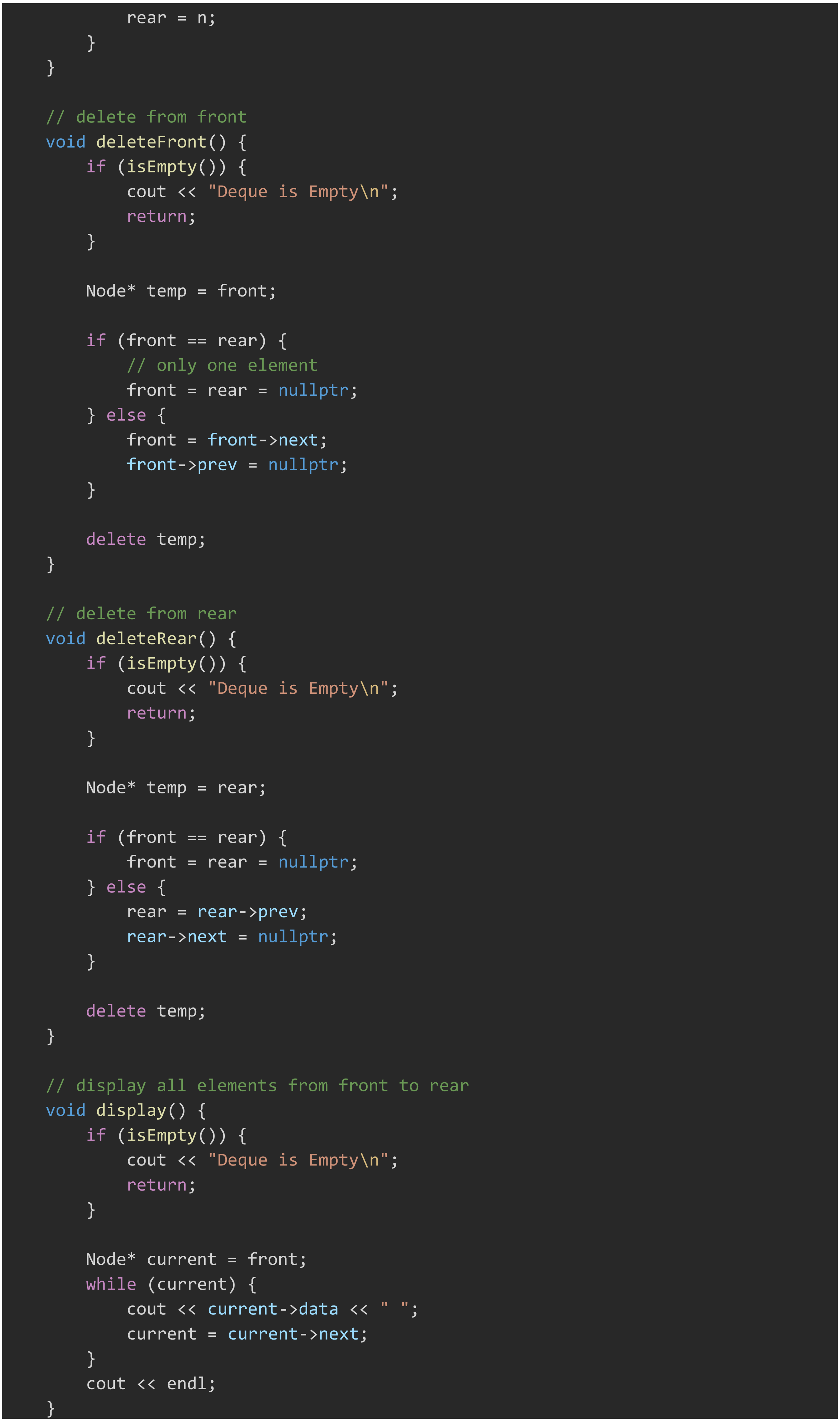


Implement class of a double ended queue using doubly Linked List.

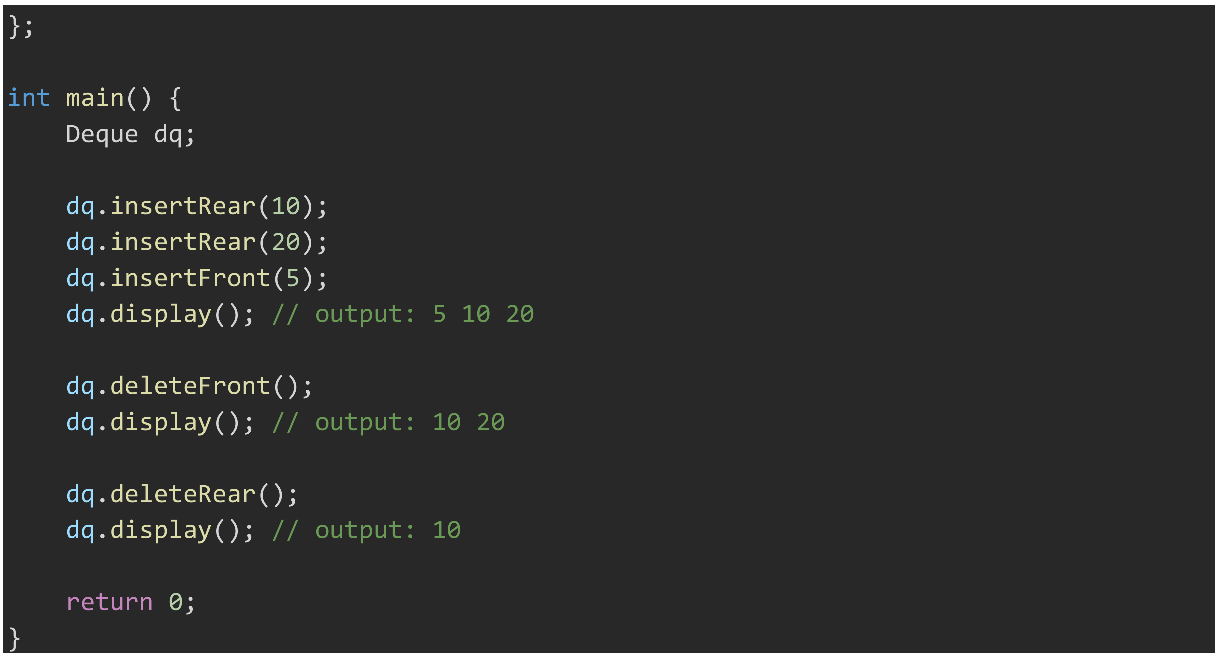
Code:



```
// Syed Wahaaj Ali
// CT-24035

#include <iostream>
using namespace std;

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

    Node(int data) {
        this->data = data;
        next = prev = nullptr;
    }
};

class Deque {
private:
    Node* front;
    Node* rear;

public:
    // constructor to start with empty deque
    Deque() { front = rear = nullptr; }

    // returns true if deque is empty
    bool isEmpty() { return front == nullptr; }

    // insert at the front
    void insertFront(int val) {
        Node* n = new Node(val);
        if (isEmpty()) {
            // if empty, both front and rear are same
            front = rear = n;
        } else {
            n->next = front;
            front->prev = n;
            front = n;
        }
    }

    // insert at the rear
    void insertRear(int val) {
        Node* n = new Node(val);
        if (isEmpty()) {
            front = rear = n;
        } else {
            rear->next = n;
            n->prev = rear;
```

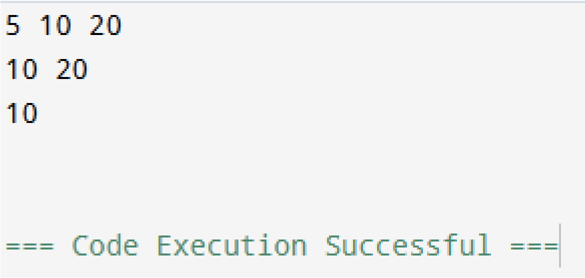


```
            rear = n;
        }
    }

    // delete from front
    void deleteFront() {
        if (isEmpty()) {
            cout << "Deque is Empty\n";
            return;
        }

        Node* temp = front;

        if (front == rear) {
            // only one element
            front = rear = nullptr;
        } else {
            front = front->next;
            front->prev = nullptr;
        }

        delete temp;
    }

    // delete from rear
    void deleteRear() {
        if (isEmpty()) {
            cout << "Deque is Empty\n";
            return;
        }

        Node* temp = rear;

        if (front == rear) {
            front = rear = nullptr;
        } else {
            rear = rear->prev;
            rear->next = nullptr;
        }

        delete temp;
    }

    // display all elements from front to rear
    void display() {
        if (isEmpty()) {
            cout << "Deque is Empty\n";
            return;
        }

        Node* current = front;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
```



```
};

int main() {
    Deque dq;

    dq.insertRear(10);
    dq.insertRear(20);
    dq.insertFront(5);
    dq.display(); // output: 5 10 20

    dq.deleteFront();
    dq.display(); // output: 10 20

    dq.deleteRear();
    dq.display(); // output: 10

    return 0;
}
```

Output:

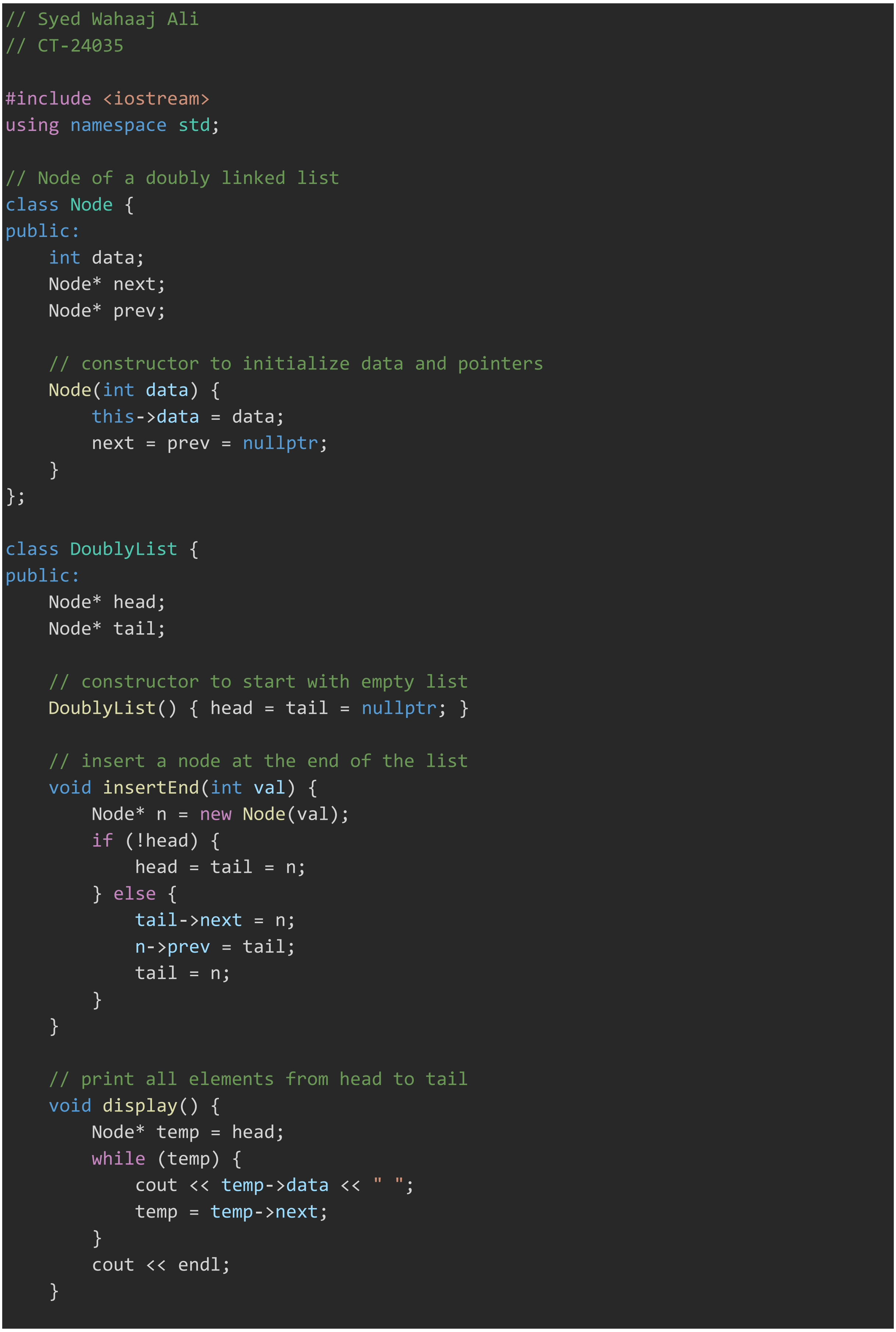


```
5 10 20
10 20
10


=== Code Execution Successful ===
```

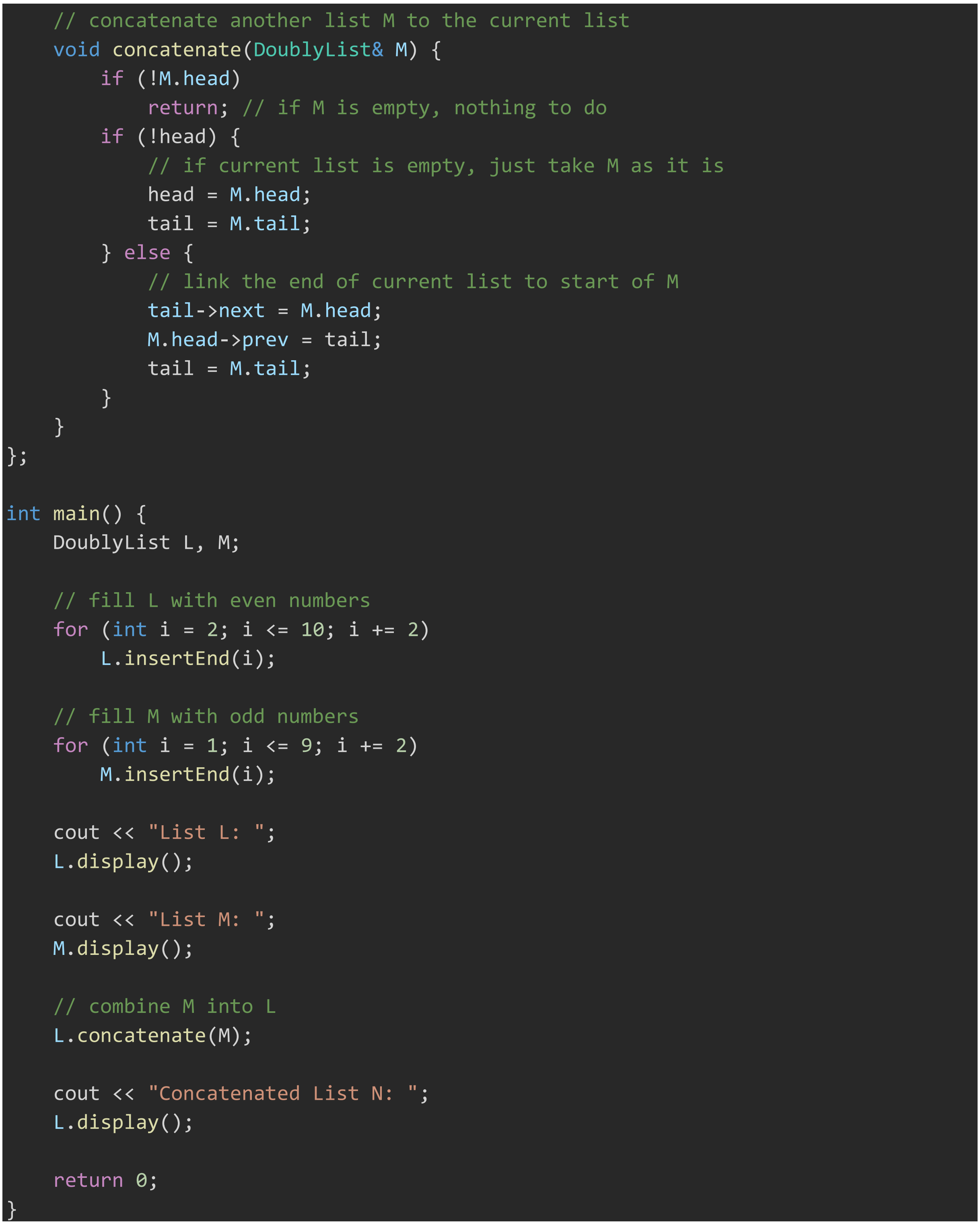
Q3:

Create two doubly link lists, say L and M . List L should be containing all even elements from 2 to 10 and list M should contain all odd elements from 1 to 9. Create a new list N by concatenating list L and M.

Code:

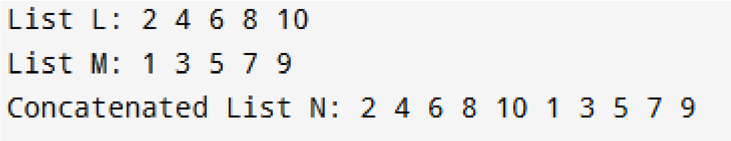


```
// Syed Wahaaj Ali
// CT-24035

#include <iostream>
using namespace std;

// Node of a doubly linked list
class Node {
public:
    int data;
    Node* next;
    Node* prev;

    // constructor to initialize data and pointers
    Node(int data) {
        this->data = data;
        next = prev = nullptr;
    }
};

class DoublyList {
public:
    Node* head;
    Node* tail;

    // constructor to start with empty list
    DoublyList() { head = tail = nullptr; }

    // insert a node at the end of the list
    void insertEnd(int val) {
        Node* n = new Node(val);
        if (!head) {
            head = tail = n;
        } else {
            tail->next = n;
            n->prev = tail;
            tail = n;
        }
    }

    // print all elements from head to tail
    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
```



```cpp
    // concatenate another list M to the current list
    void concatenate(DoublyList& M) {
        if (!M.head)
            return; // if M is empty, nothing to do
        if (!head) {
            // if current list is empty, just take M as it is
            head = M.head;
            tail = M.tail;
        } else {
            // link the end of current list to start of M
            tail->next = M.head;
            M.head->prev = tail;
            tail = M.tail;
        }
    }
};

int main() {
    DoublyList L, M;

    // fill L with even numbers
    for (int i = 2; i <= 10; i += 2)
        L.insertEnd(i);

    // fill M with odd numbers
    for (int i = 1; i <= 9; i += 2)
        M.insertEnd(i);

    cout << "List L: ";
    L.display();

    cout << "List M: ";
    M.display();

    // combine M into L
    L.concatenate(M);

    cout << "Concatenated List N: ";
    L.display();

    return 0;
}
```

Output:

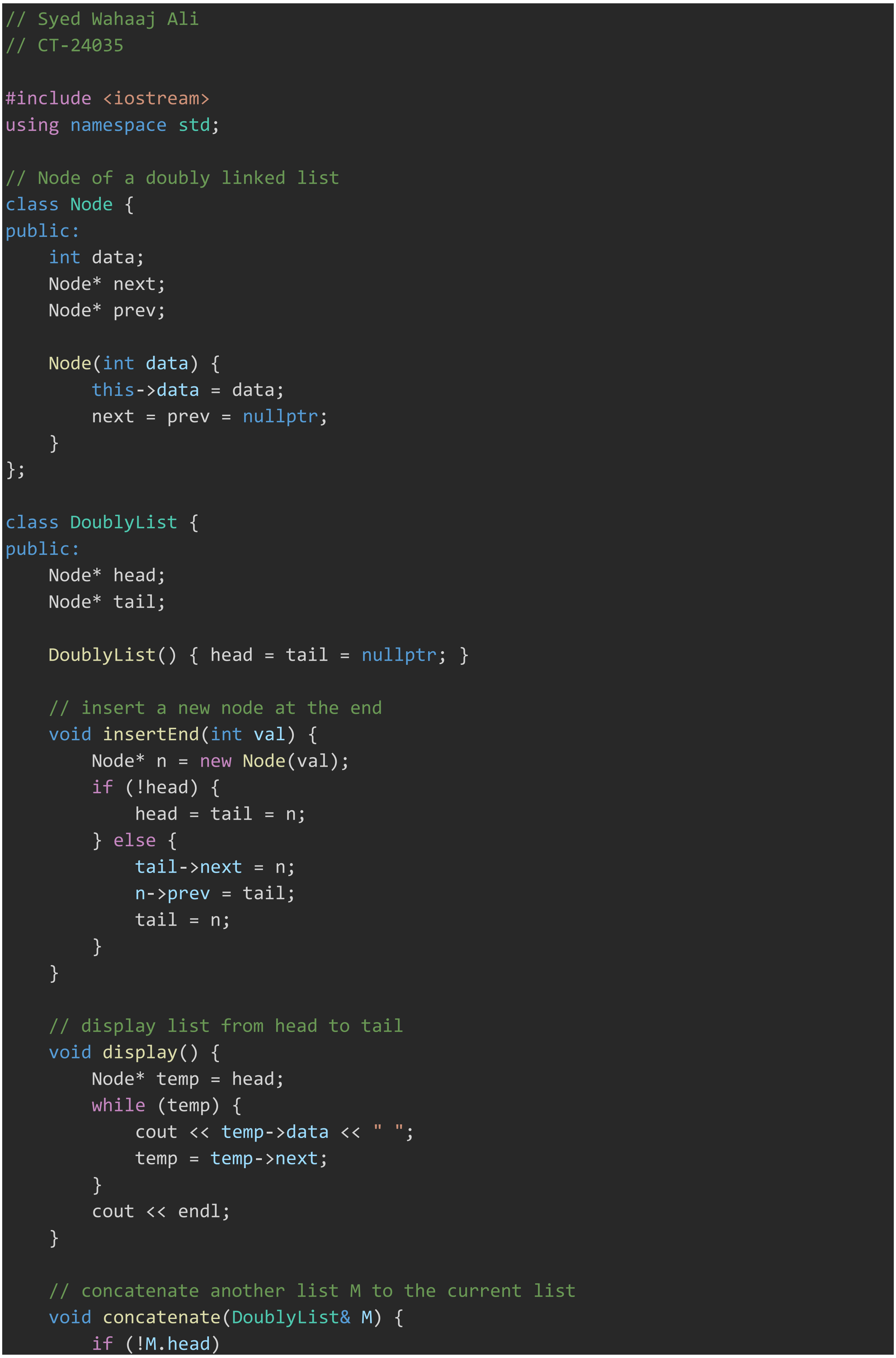


```
List L: 2 4 6 8 10
List M: 1 3 5 7 9
Concatenated List N: 2 4 6 8 10 1 3 5 7 9
```

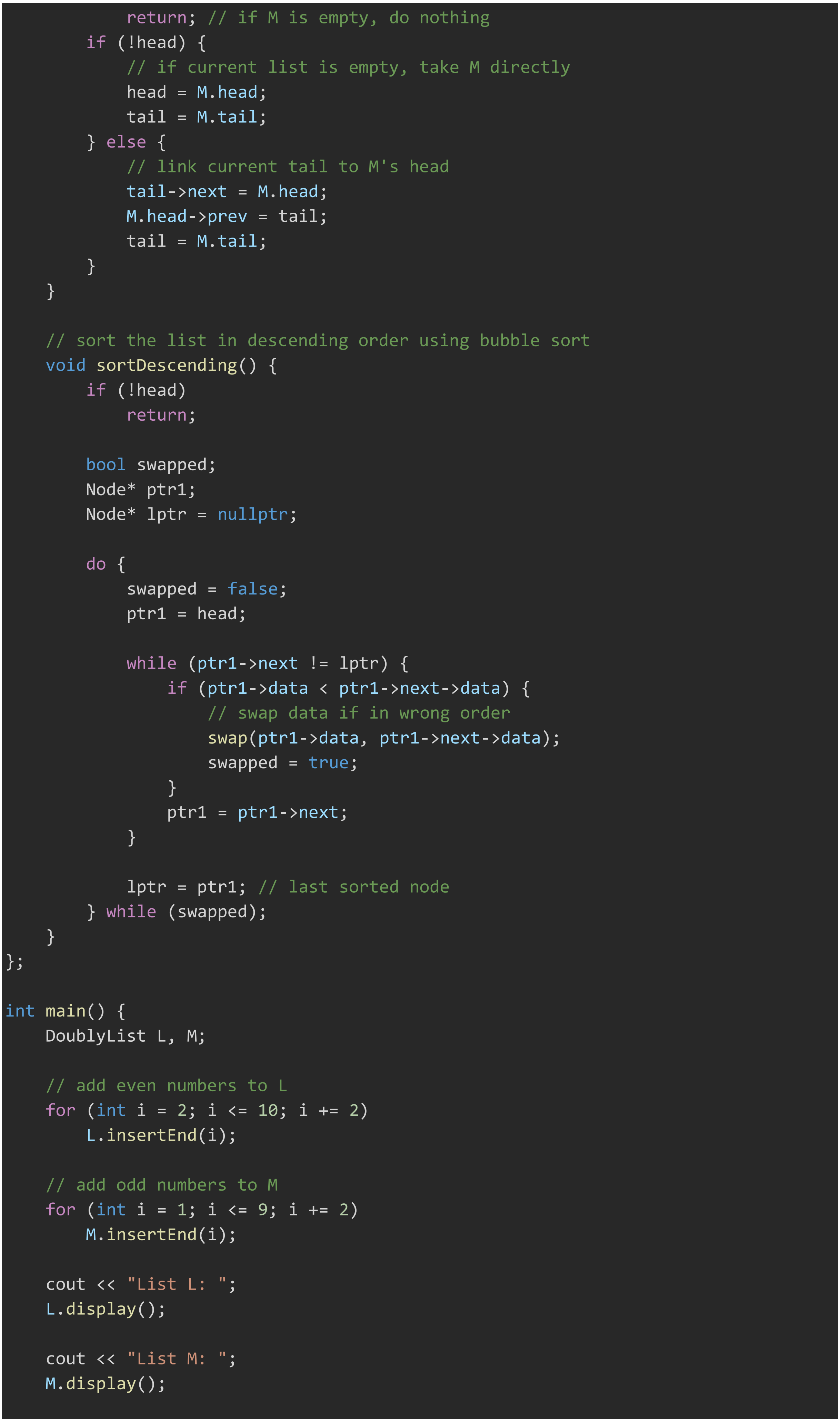
Q4:

Sort the contents of list N created in Q4 in descending order.

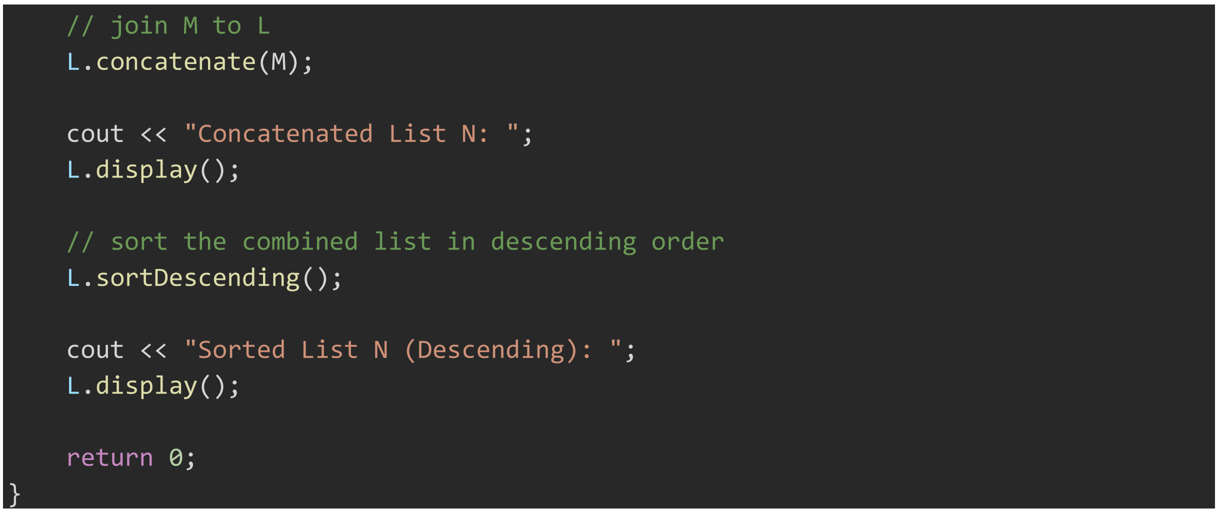
Code:



```
// Syed Wahaaj Ali
// CT-24035

#include <iostream>
using namespace std;

// Node of a doubly linked list
class Node {
public:
    int data;
    Node* next;
    Node* prev;

    Node(int data) {
        this->data = data;
        next = prev = nullptr;
    }
};

class DoublyList {
public:
    Node* head;
    Node* tail;

    DoublyList() { head = tail = nullptr; }

    // insert a new node at the end
    void insertEnd(int val) {
        Node* n = new Node(val);
        if (!head) {
            head = tail = n;
        } else {
            tail->next = n;
            n->prev = tail;
            tail = n;
        }
    }

    // display list from head to tail
    void display() {
        Node* temp = head;
        while (temp) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // concatenate another list M to the current list
    void concatenate(DoublyList& M) {
        if (!M.head)
```

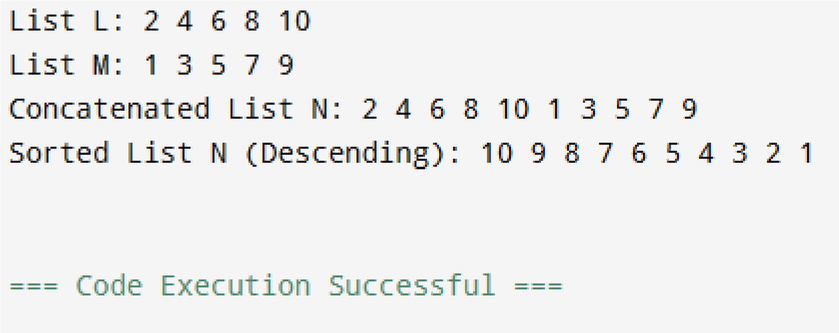


```
            return; // if M is empty, do nothing
        if (!head) {
            // if current list is empty, take M directly
            head = M.head;
            tail = M.tail;
        } else {
            // link current tail to M's head
            tail->next = M.head;
            M.head->prev = tail;
            tail = M.tail;
        }
    }

    // sort the list in descending order using bubble sort
    void sortDescending() {
        if (!head)
            return;

        bool swapped;
        Node* ptr1;
        Node* lptr = nullptr;

        do {
            swapped = false;
            ptr1 = head;

            while (ptr1->next != lptr) {
                if (ptr1->data < ptr1->next->data) {
                    // swap data if in wrong order
                    swap(ptr1->data, ptr1->next->data);
                    swapped = true;
                }
                ptr1 = ptr1->next;
            }

            lptr = ptr1; // last sorted node
        } while (swapped);
    }
};

int main() {
    DoublyList L, M;

    // add even numbers to L
    for (int i = 2; i <= 10; i += 2)
        L.insertEnd(i);

    // add odd numbers to M
    for (int i = 1; i <= 9; i += 2)
        M.insertEnd(i);

    cout << "List L: ";
    L.display();

    cout << "List M: ";
    M.display();
```



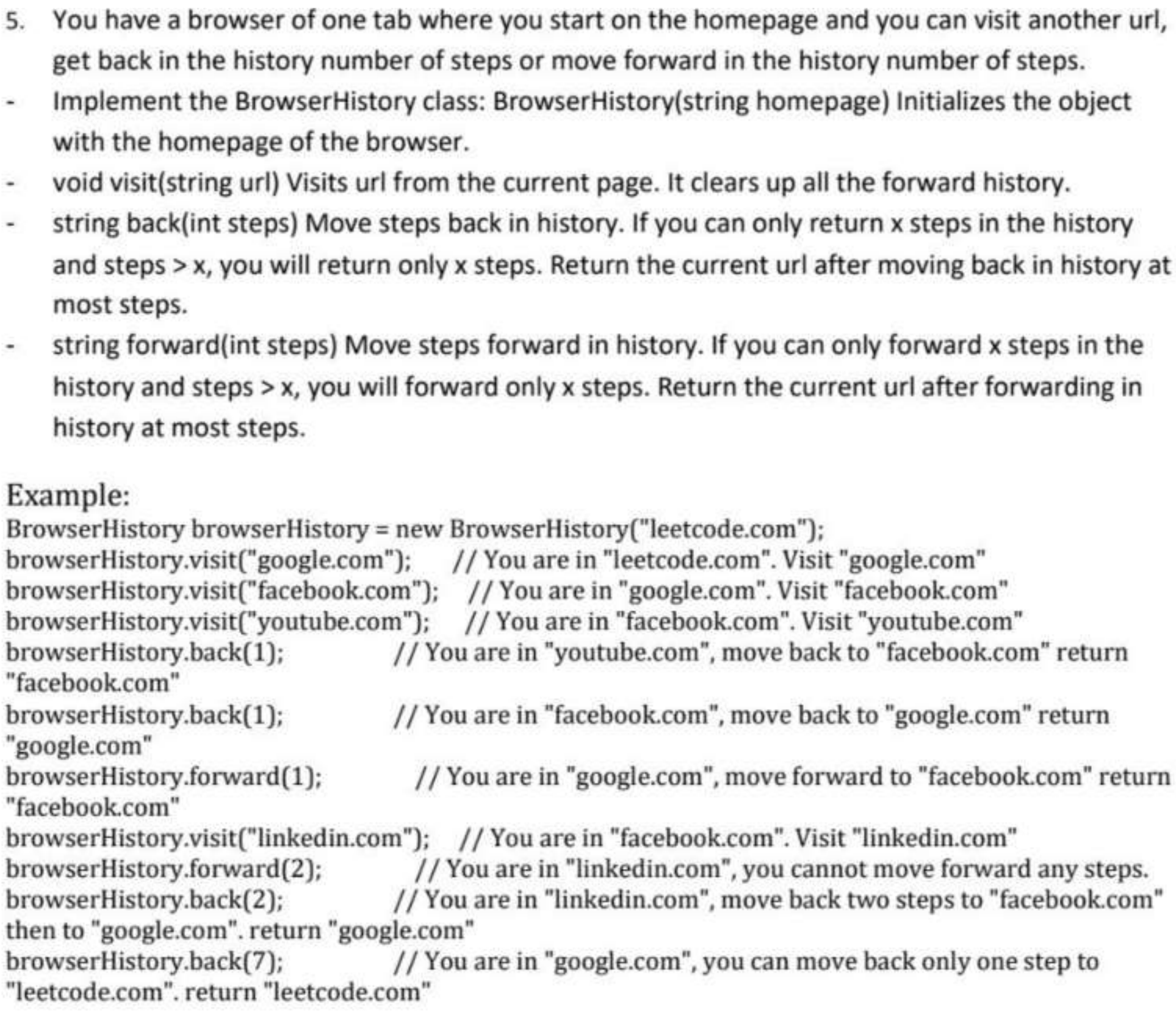
```cpp
    // join M to L
    L.concatenate(M);

    cout << "Concatenated List N: ";
    L.display();

    // sort the combined list in descending order
    L.sortDescending();

    cout << "Sorted List N (Descending): ";
    L.display();

    return 0;
}
```

Output:



```
List L: 2 4 6 8 10
List M: 1 3 5 7 9
Concatenated List N: 2 4 6 8 10 1 3 5 7 9
Sorted List N (Descending): 10 9 8 7 6 5 4 3 2 1



=== Code Execution Successful ===
```

Q5:



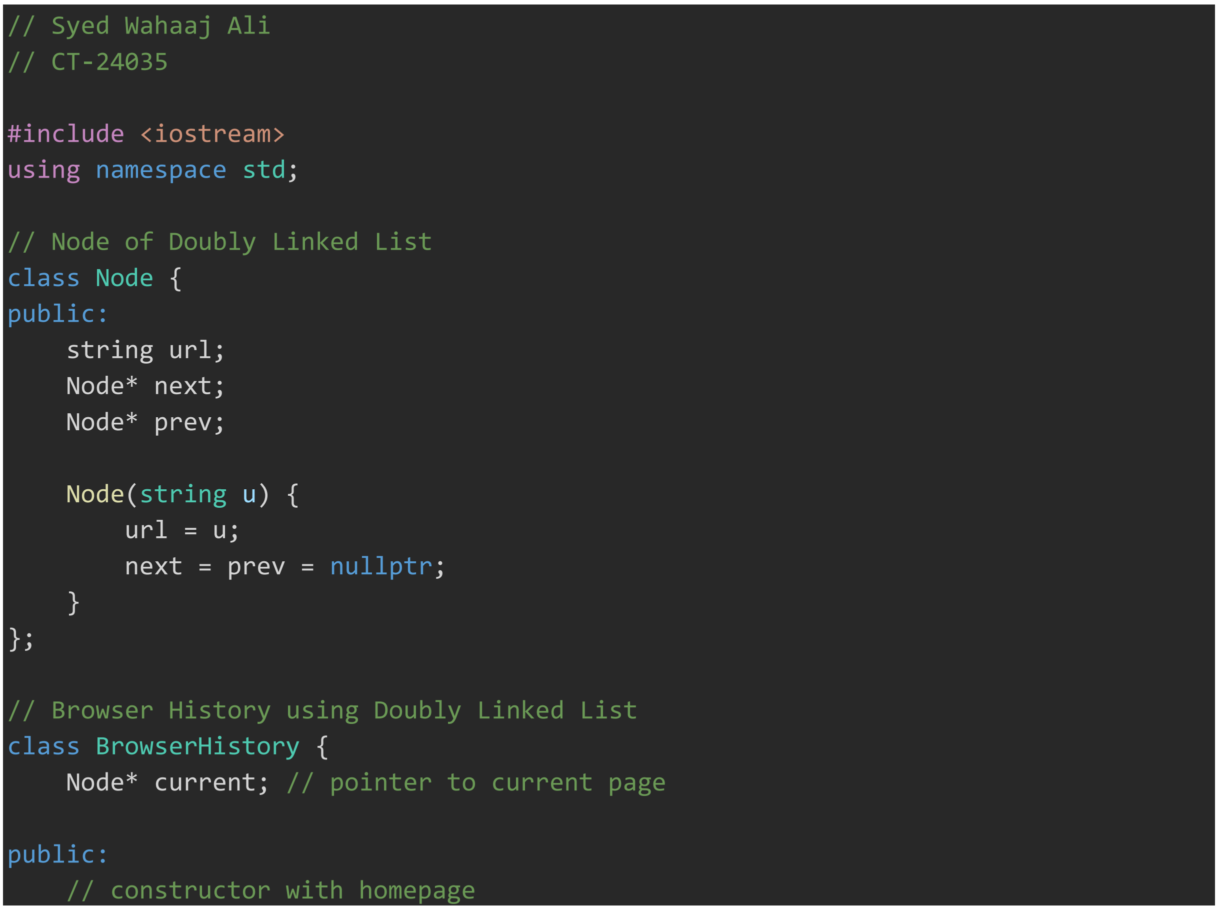
5. You have a browser of one tab where you start on the homepage and you can visit another url, get back in the history number of steps or move forward in the history number of steps.

- Implement the BrowserHistory class: BrowserHistory(string homepage) Initializes the object with the homepage of the browser.
- void visit(string url) Visits url from the current page. It clears up all the forward history.
- string back(int steps) Move steps back in history. If you can only return x steps in the history and steps > x, you will return only x steps. Return the current url after moving back in history at most steps.
- string forward(int steps) Move steps forward in history. If you can only forward x steps in the history and steps > x, you will forward only x steps. Return the current url after forwarding in history at most steps.

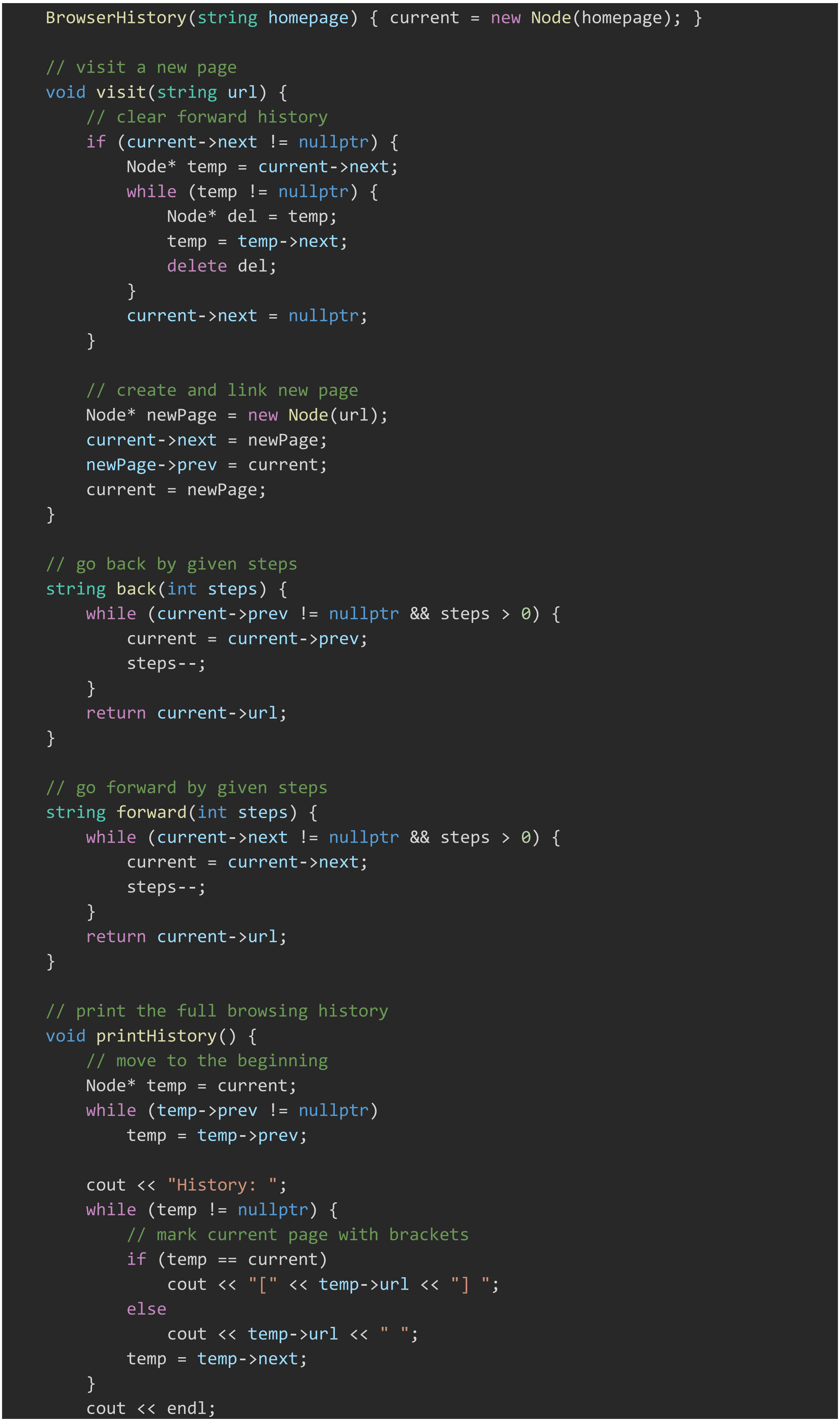
Example:

BrowserHistory browserHistory = new BrowserHistory("leetcode.com");
browserHistory.visit("google.com");   // You are in "leetcode.com". Visit "google.com"
browserHistory.visit("facebook.com");   // You are in "google.com". Visit "facebook.com"
browserHistory.visit("youtube.com");   // You are in "facebook.com". Visit "youtube.com"
browserHistory.back(1);   // You are in "youtube.com", move back to "facebook.com" return "facebook.com"
browserHistory.back(1);   // You are in "facebook.com", move back to "google.com" return "google.com"
browserHistory.forward(1);   // You are in "google.com", move forward to "facebook.com" return "facebook.com"
browserHistory.visit("linkedin.com");   // You are in "facebook.com". Visit "linkedin.com"
browserHistory.forward(2);   // You are in "linkedin.com", you cannot move forward any steps.
browserHistory.back(2);   // You are in "linkedin.com", move back two steps to "facebook.com" then to "google.com". return "google.com"
browserHistory.back(7);   // You are in "google.com", you can move back only one step to "leetcode.com". return "leetcode.com"

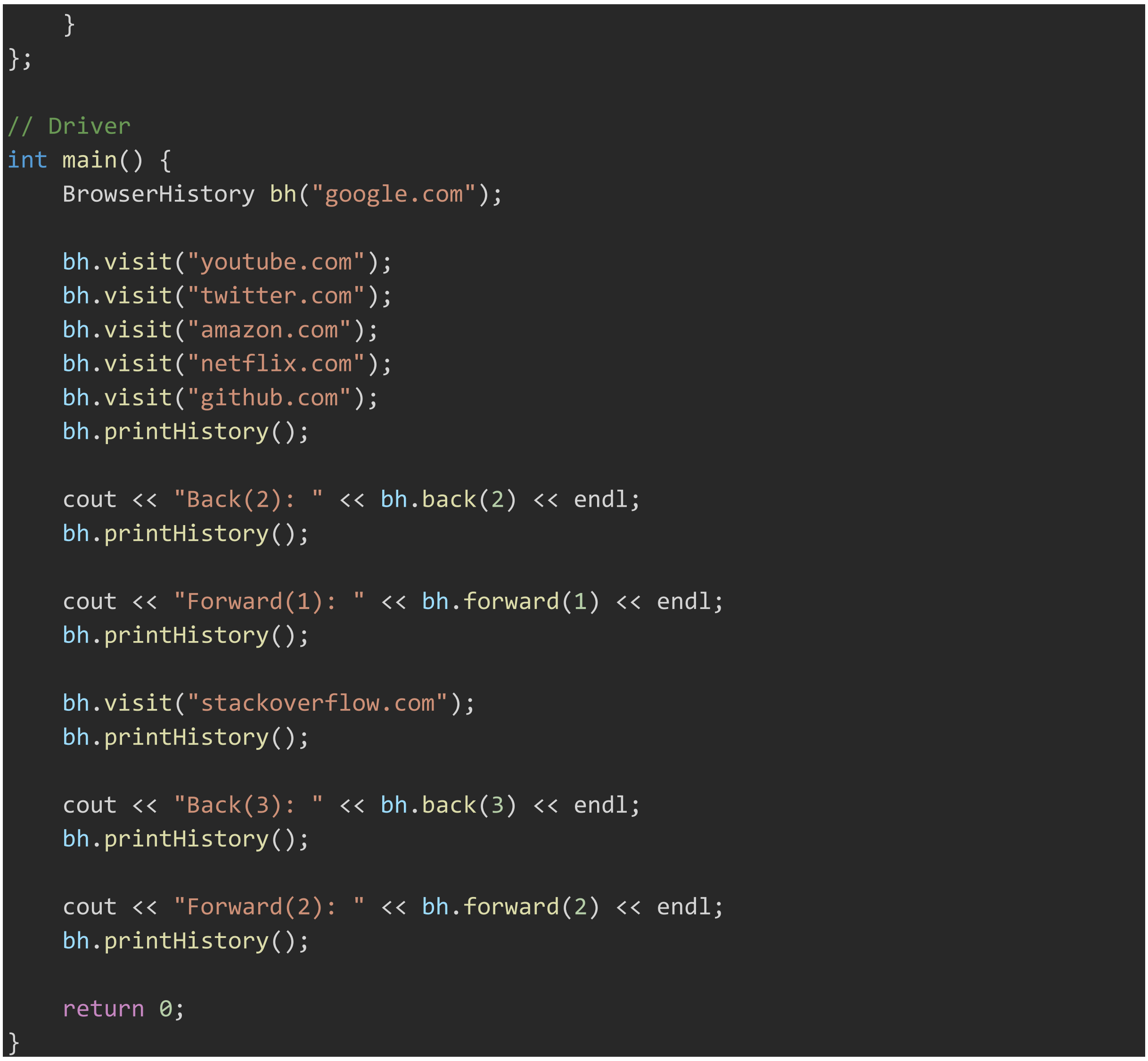
Code:



```
// Syed Wahaaj Ali
// CT-24035

#include <iostream>
using namespace std;

// Node of Doubly Linked List
class Node {
public:
    string url;
    Node* next;
    Node* prev;

    Node(string u) {
        url = u;
        next = prev = nullptr;
    }
};

// Browser History using Doubly Linked List
class BrowserHistory {
    Node* current; // pointer to current page

public:
    // constructor with homepage
```

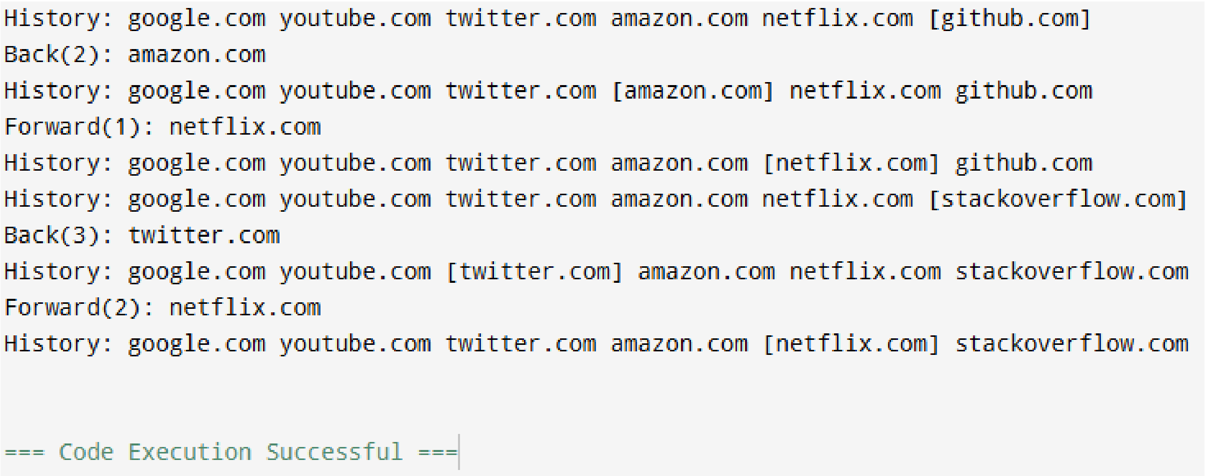


```
    BrowserHistory(string homepage) { current = new Node(homepage); }

    // visit a new page
    void visit(string url) {
        // clear forward history
        if (current->next != nullptr) {
            Node* temp = current->next;
            while (temp != nullptr) {
                Node* del = temp;
                temp = temp->next;
                delete del;
            }
            current->next = nullptr;
        }

        // create and link new page
        Node* newPage = new Node(url);
        current->next = newPage;
        newPage->prev = current;
        current = newPage;
    }

    // go back by given steps
    string back(int steps) {
        while (current->prev != nullptr && steps > 0) {
            current = current->prev;
            steps--;
        }
        return current->url;
    }

    // go forward by given steps
    string forward(int steps) {
        while (current->next != nullptr && steps > 0) {
            current = current->next;
            steps--;
        }
        return current->url;
    }

    // print the full browsing history
    void printHistory() {
        // move to the beginning
        Node* temp = current;
        while (temp->prev != nullptr)
            temp = temp->prev;

        cout << "History: ";
        while (temp != nullptr) {
            // mark current page with brackets
            if (temp == current)
                cout << "[" << temp->url << "] ";
            else
                cout << temp->url << " ";
            temp = temp->next;
        }
        cout << endl;
```



```cpp
    }
};

// Driver
int main() {
    BrowserHistory bh("google.com");

    bh.visit("youtube.com");
    bh.visit("twitter.com");
    bh.visit("amazon.com");
    bh.visit("netflix.com");
    bh.visit("github.com");
    bh.printHistory();

    cout << "Back(2): " << bh.back(2) << endl;
    bh.printHistory();

    cout << "Forward(1): " << bh.forward(1) << endl;
    bh.printHistory();

    bh.visit("stackoverflow.com");
    bh.printHistory();

    cout << "Back(3): " << bh.back(3) << endl;
    bh.printHistory();

    cout << "Forward(2): " << bh.forward(2) << endl;
    bh.printHistory();

    return 0;
}
```

Output:



```
History: google.com youtube.com twitter.com amazon.com netflix.com [github.com]
Back(2): amazon.com
History: google.com youtube.com twitter.com [amazon.com] netflix.com github.com
Forward(1): netflix.com
History: google.com youtube.com twitter.com amazon.com [netflix.com] github.com
History: google.com youtube.com twitter.com amazon.com netflix.com [stackoverflow.com]
Back(3): twitter.com
History: google.com youtube.com [twitter.com] amazon.com netflix.com stackoverflow.com
Forward(2): netflix.com
History: google.com youtube.com twitter.com amazon.com [netflix.com] stackoverflow.com


=== Code Execution Successful ===
```