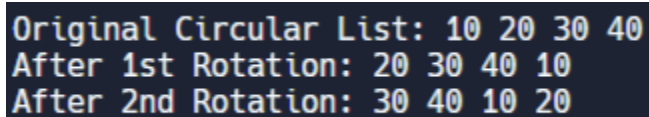```cpp
#1
#include <iostream>
using namespace std;
struct node {
    int data;
    node* next;
};
class CircularLinkedList {
    private:
        node* tail;
    public:
        CircularLinkedList() {
            tail = nullptr;
        }
        void Append(int value) {
            node* newNode = new node;
            newNode->data = value;
            if (tail == nullptr) {
                tail = newNode;
                tail->next = tail;
            } else {
                newNode->next = tail->next;
                tail->next = newNode;
                tail = newNode;
            }
        }
        void Display() {
            if (tail == nullptr) return;
                node* temp = tail->next;
            do {
                cout << temp->data << " ";
                temp = temp->next;
            } while (temp != tail->next);
            cout << endl;
        }
        void Rotate() {
        if (tail != nullptr) {
        tail = tail->next;
        }
        }
};
int main() {
    CircularLinkedList myList;
    myList.Append(10);
```

```cpp
    myList.Append(20);
    myList.Append(30);
    myList.Append(40);
    cout << "Original Circular List: ";
    myList.Display(); // Output: 10 20 30 40
    myList.Rotate();
    cout << "After 1st Rotation: ";
    myList.Display(); // Output: 20 30 40 10
    myList.Rotate();
    cout << "After 2nd Rotation: ";
    myList.Display(); // Output: 30 40 10 20
    return 0;
}
```

```
Original Circular List: 10 20 30 40
After 1st Rotation: 20 30 40 10
After 2nd Rotation: 30 40 10 20
```

#2

```cpp
#include <iostream>
using namespace std;
struct node {
    int data;
    node* next;
    node* prev;
};
class DLinkedList {
    private:
        node* head;
        node* tail;
    public:
    DLinkedList() {
        head = nullptr;
        tail = nullptr;
        cout << "head and tail nodes are initiated with nullptr" << endl;
    }
    void ListAppend(int elem) {
        node* newNode = new node;
        newNode->data = elem;
        newNode->next = nullptr;
        newNode->prev = tail;
        if (head == nullptr) { // List is empty
            head = newNode;
```

```cpp
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    void ListPrepend(int elem) {
        node* newNode = new node;
        newNode->data = elem;
        newNode->next = head;
        newNode->prev = nullptr;
        if (head == nullptr) { // List is empty
            head = newNode;
            tail = newNode;
        } else {
            head->prev = newNode;
            head = newNode;
        }
    }
    void InsertAfter(node* curNode, int elem) {
        if (curNode == nullptr) return;
            node* newNode = new node;
            newNode->data = elem;
            newNode->next = curNode->next;
            newNode->prev = curNode;
        if (curNode->next != nullptr) {
            curNode->next->prev = newNode;
        } else { // Insert after the tail
            tail = newNode;
        }
        curNode->next = newNode;
    }

    void RemoveAfter(node* curNode) {
        if (head == nullptr) {
            cout << "List is empty. Nothing to remove." << endl;
            return;
        }
            if (curNode != nullptr && curNode->next != nullptr) {
            node* sucNode = curNode->next;
            curNode->next = sucNode->next;
        if (sucNode->next != nullptr) {
            sucNode->next->prev = curNode;
        } else {
```

```cpp
        tail = curNode;
      }
      delete sucNode; // Free memory
    }
  }

  void Erase(node* curNode) {
    if (curNode == nullptr || head == nullptr) {
      cout << "No node to erase." << endl;
      return;
    }
    if (curNode == head) {
      head = curNode->next;
      if (head != nullptr) {
        head->prev = nullptr;
      } else { // List becomes empty
        tail = nullptr;
      }
    } else if (curNode == tail) {
      tail = curNode->prev;
      if (tail != nullptr) {
        tail->next = nullptr;
      }
    } else { // Removing in the middle
      curNode->prev->next = curNode->next;
      curNode->next->prev = curNode->prev;
    }
    delete curNode; // Free memory
  }

// Display the list
void ListDisplay() {
  node* tmp = head;
  while (tmp != nullptr) {
    cout << tmp->data << " ";
    tmp = tmp->next;
  }
    cout << endl;
  }

  // Find a node with a given value
node* GetNode(int value) {
  node* tmp = head;
  while (tmp != nullptr) {
```

```cpp
        if (tmp->data == value) {
        return tmp;
        }
        tmp = tmp->next;
    }
    return nullptr; // If the node is not found
}
};


int main() {
    DLinkedList numList;
    numList.ListAppend(30);
    numList.ListAppend(40);
    numList.ListPrepend(20);
    numList.ListPrepend(10);
    cout << "Original List: ";
    numList.ListDisplay();
    node* curNode = numList.GetNode(30);
    if (curNode != nullptr) {
        numList.RemoveAfter(curNode);
    }
    cout << "After Removing Node after 30: ";
    numList.ListDisplay();

    curNode = numList.GetNode(30);
    if (curNode != nullptr) {
        numList.Erase(curNode);
    }
    cout << "After Erasing Node with value 30:";
    numList.ListDisplay();
    return 0;
}
```

```
head and tail nodes are initiated with nullptr
Original List: 10 20 30 40
After Removing Node after 30: 10 20 30
After Erasing Node with value 30:10 20
```

#3
```cpp
#include <iostream>
using namespace std;
class MyClass {
  private:
    int value;
  public:
```

```cpp
    MyClass(int value) {
      this->value = value;
    }
    void display() {
    cout << "Value: " << this->value << endl;
    }
    MyClass& setValue(int value) {
    this->value = value;
    return *this;
    }
};
int main() {
  MyClass obj(10);
  obj.display();
  obj.setValue(20);
  obj.display();
  return 0;
}
```

```
Value: 10
Value: 20
```

#4

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(){
  vector<int> myIntVector;
  vector<int>::iterator it;
  for (int i = 1; i <= 5; i++) myIntVector.push_back(i);
  cout << "myvector contains: ";
  for (it = myIntVector.begin(); it != myIntVector.end(); it++) {
  cout << *it << " ";
  }
  cout << endl;
  return 0;
}
```

```
myvector contains: 1 2 3 4 5
```