

### Problem 1: Singly-linked list

```
class LinkedList
{
public:
    LinkedList();
    ~LinkedList();
    void addToList(int value); // add to the head of the linked list
    void reverse(); // Reverse the linked list
private:
    struct Node
    {
        int num;
        Node *next;
    };
    Node *m_head;
};

LinkedList::LinkedList()
: _____
{}

LinkedList::~~LinkedList()
{
    Node *temp;
    while(m_head != nullptr) {

    }
}

void LinkedList::addToList(int value)
{

}

//What about adding to the tail of the linked list? Try it out yourself

void LinkedList::reverse()
{
    Node *nextNode = _____, *prevNode = _____, *current = _____;
    while(current)
    {

    }
}
```

(More lines on the back)

}

//What about removing a node from the linked list?

### Problem 2 Doubly-linked list (Not circular, no dummy node)

```
class LinkedList
```

{

public:

```
LinkedList();
```

```
~LinkedList();
```

```
void addToList(int value); // add to the head of the linked list
```

```
void remove(Node *node)
```

private:

```
struct Node
```

 $\{$ 

```
int num;
```

```
Node *next;
```

Node \*prev

 $\}:$ 

```
Node *m_head;
```

```
Node *m_tail;
```

$$\};$$

```
void LinkedList::addToList(int value)
```

{

}

```
void LinkedList::remove(Node *node)
```

{

$$\}$$

### Problem 3 circular doubly-linked list with a dummy node

```
class LinkedList
{
public:
    LinkedList();
    ~LinkedList();
    void addToList(int value); // add to the head of the linked list
    void remove(Node *node)
private:
    struct Node
    {
        int num;
        Node *next;
        Node *prev
    };
    Node *m_head;
};

LinkedList::LinkedList()
{

}

LinkedList::~~LinkedList()
{

}

void LinkedList::addToList(int value)
{

}

bool LinkedList::remove(Node *node)
{

}
```