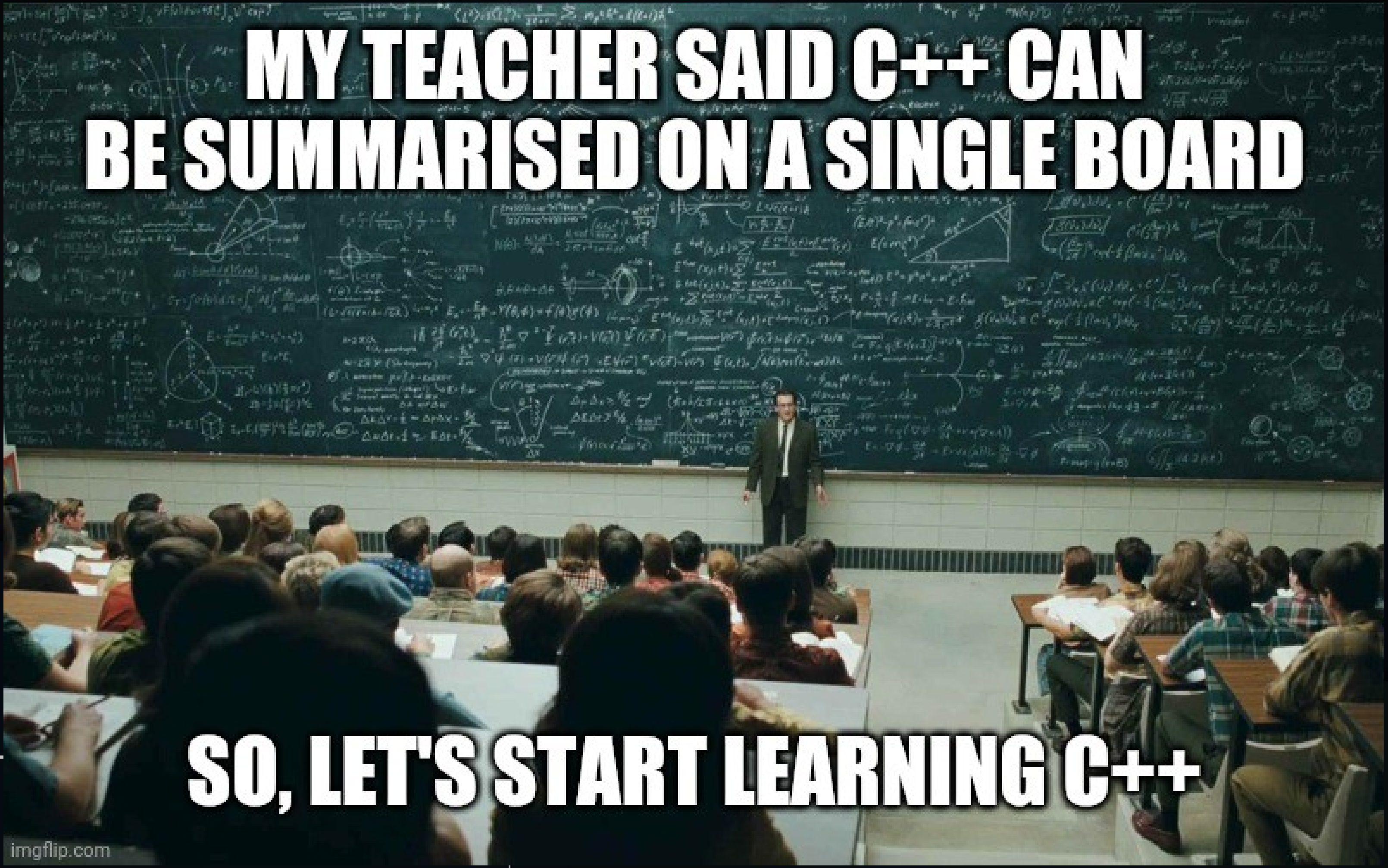

MY TEACHER SAID C++ CAN BE SUMMARISED ON A SINGLE BOARD
SO, LET'S START LEARNING C++
imgflip.com

5.1: What is a Linked List?

CS2308
Gentry Atkinson

Can a class point to itself?

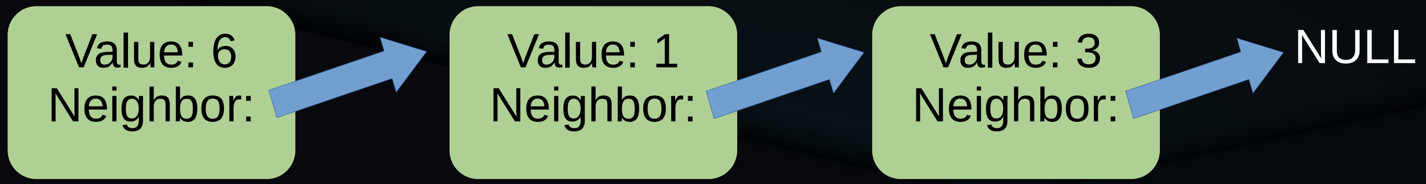
```cpp
class C{
    public:
        int value;
        C* neighbor;
};

int main() {
```

cks | Search results | Build log | Build messages | Debugger

Line	Message
	=== Build file: "no target" in "no project" (compiler: unknown) ===
	=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

Looks like yes, but why?

What if we make a chain?



- Each link in the chain is holding one integer.
- This chain could represent the list: [6, 1, 3]
- This chain could continue to grow indefinitely.

Linked Lists

- Chains for same-type classes are used to store a list with any number of values.
- We do not have to know the size of the list at creation.
- We have to keep track of the “head” of the list.
- List nodes are created using dynamic memory allocation.

Example 1

```
class ListNode{
    public:
        int value;
        ListNode* next;
};

int main(int argc, char** argv){
    ListNode* head;
    head = new ListNode;
    head→value = 7;
    head→next = NULL:
    return 0;
} //try to guess the output
```

<u>ListNode</u>
val: 7
next: NULL

Linked List Tasks:

1) Create an empty list
2) Create a new node
3) Add a new node to front of list (given newNode)
4) Traverse the list (and output)
5) Find the last node (of a non-empty list)
6) Find the node containing a certain value
7) Find a node AND it’s previous neighbor.
8) Append to the end of a non-empty list
9) Delete the first node
10) Delete an element, given previous and next
11) Insert a new element, given previous and next

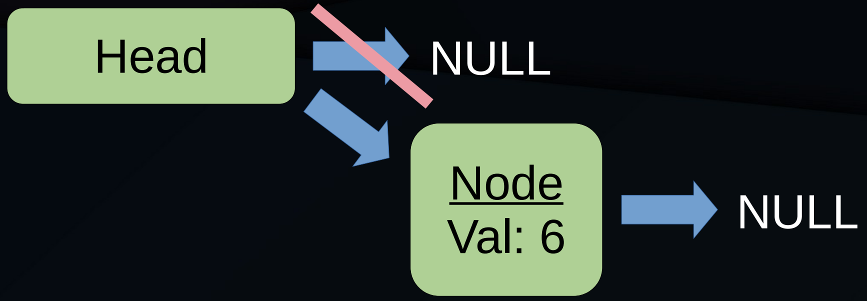
Task 1: Create an empty list

- The NULL pointer value is used to indicate the end of a list.
- Setting the **head** pointer to NULL shows that a list is empty.

List = []

Task 2: Create a new node

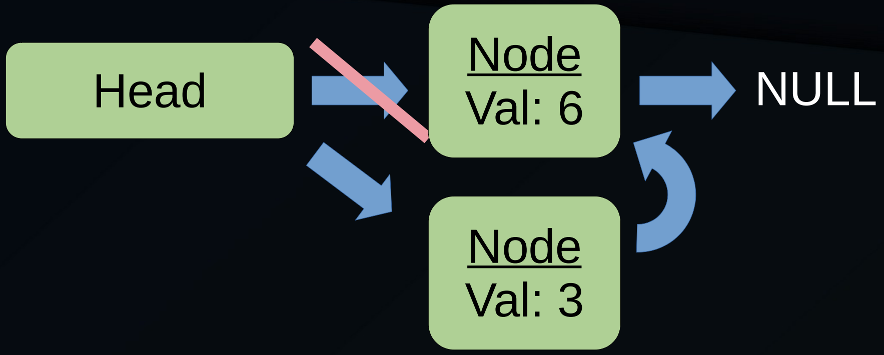
- The **head** pointer can be directed towards a dynamically allocated ListNode.



List = [6]

Task 3: Add a new node to front of list

- The **head** pointer is re-directed to the new node.
- The **next** pointer of the new node should be pointed to the old head.
- The **next** pointer of the old head still points to NULL.



List = [3, 6]

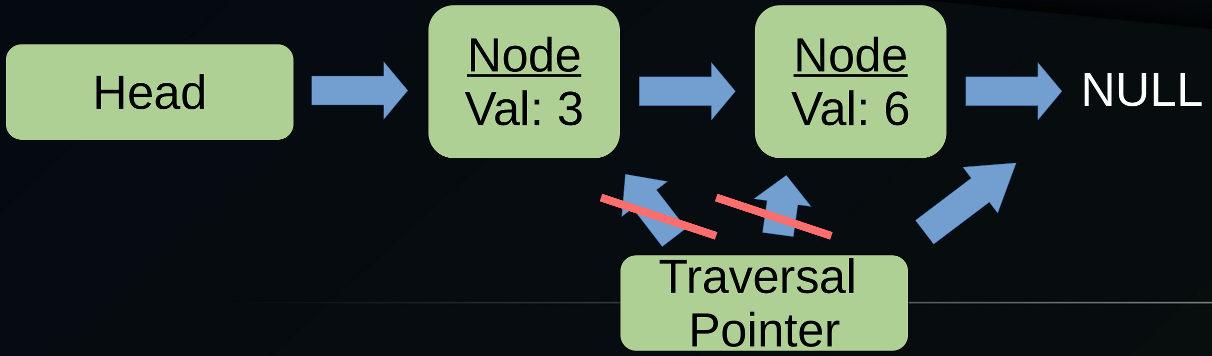
Example 2

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
int main(int argc, char** argv){
    Node* head = new Node(6);
    Node* newNode = new Node(3);
    newNode->next = head;
    head = newNode;
    return 0;
} //try to guess the output
```

Task 4: Traverse the list

- A **ListNode** pointer can be used to examine each element in the list, starting at the head.
- This could be done to print every value in a list.
- The list pointer advances until it finds NULL.



List = [3, 6]

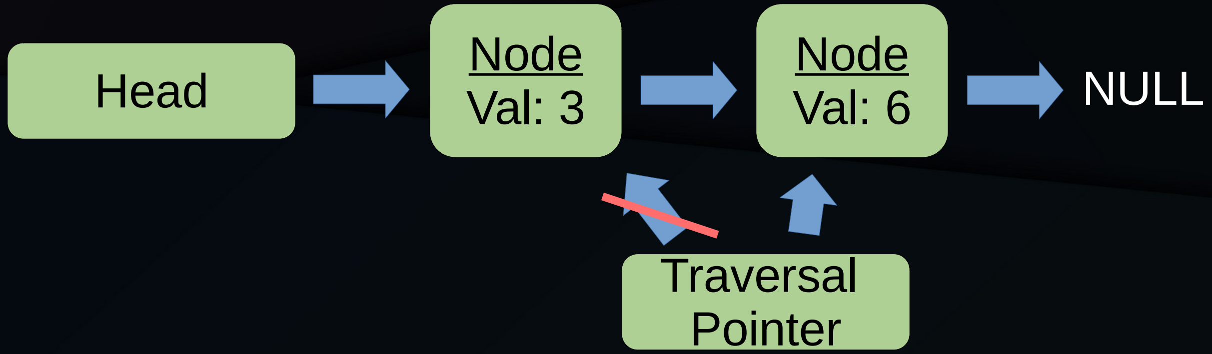
Example 3

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
void printList(Node* h){
    while(h!=NULL){
        cout << h->value << ' ';
        h = h->next;
    }
    cout << endl;
}
```

Task 5: Find the last node of a non-empty list

- Like traversal, but we should stop with the traversal pointer pointing to the node who's **next** pointer is NULL.

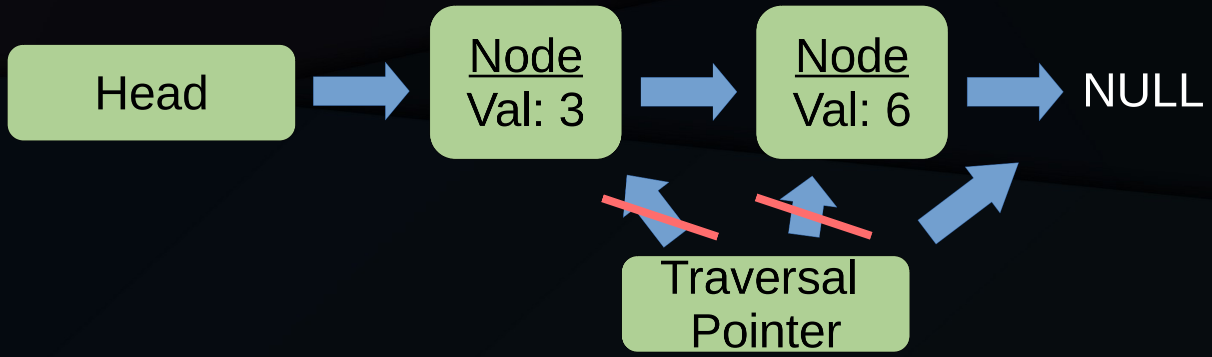


List = [3, 6]

Last Value = 6

Task 6: Find the node containing a value

- Like traversal, but stop if a node's value matches the target OR if the traversal pointer reaches NULL.

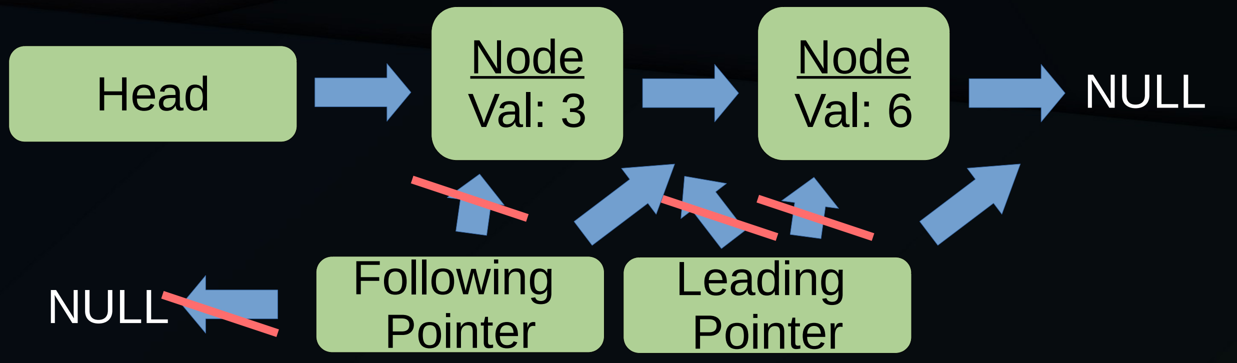


List = [3, 6]

Target Value = 4

Task 7: Find a node AND it's previous neighbor.

- Traverse with two pointers, one leading and one following.
- Halt on target value OR if the leading pointer is NULL.

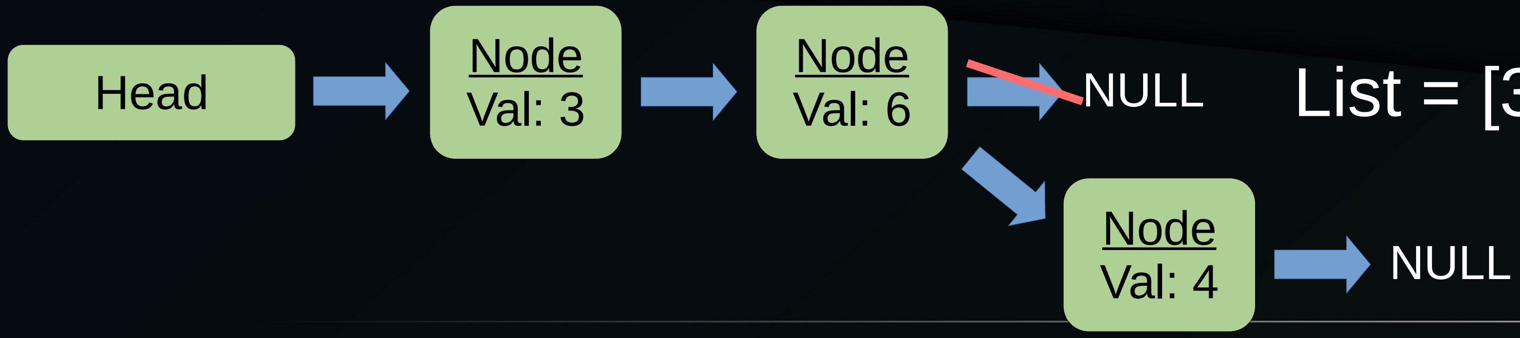


List = [3, 6]

Target Value = 4

Task 8: Append to the end of a non-empty list

- First, find the end of the list (task 5).
- Set the **next** pointer of the old tail to point to the new node.
- Set the **next** pointer of the new tail to NULL.



List = [3, 6, 4]

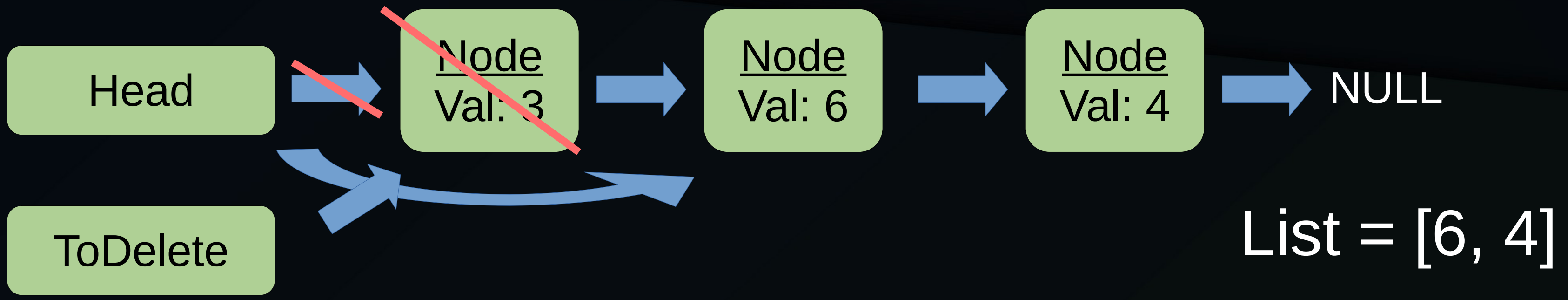
Example 4

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
int main(int argc, char** argv){
    Node* head = new Node(6);
    Node* newNode = new Node(3);
    newNode->next = head;
    head = newNode;
    Node* tail = head;
    while(tail->next != NULL)
        tail = tail->next;
    tail->next = new Node(4);
    return 0;
} //try to guess the output
```

Task 9: Delete the first node

- Linked lists are dynamically allocated memory, so it’s our job to clean up.
- Assign a pointer to the **head** node, reassign the **head** pointer to head→next, the the old head.



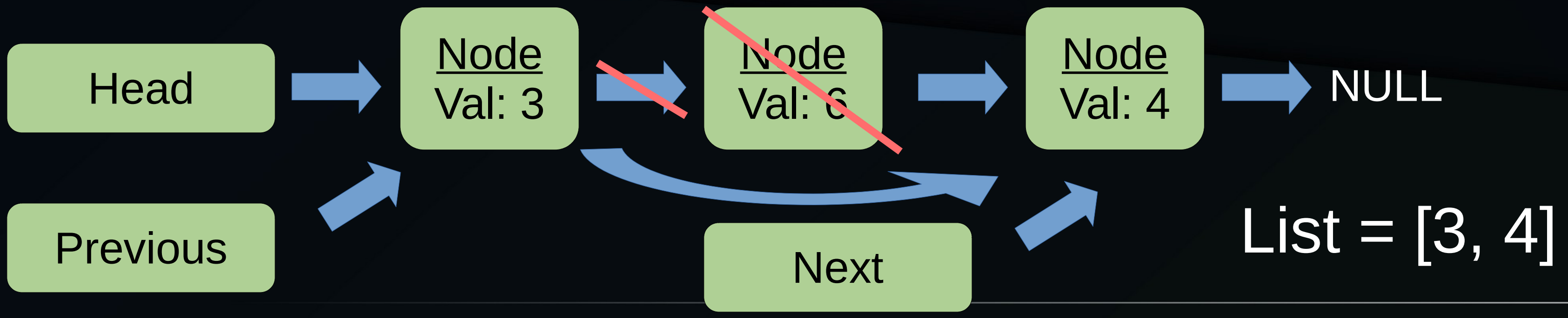
Example 5

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
int main(int argc, char** argv){
    Node* head = new Node(3);
    head->next = new Node(6);
    head->next->next = new Node(4);
    Node* oldHead = head;
    head = head->next;
    delete oldHead;
    return 0;
} //try to guess the output
```

Task 10: Delete an element, given previous and next

- Delete previous→next.
- Assign previous→next to **next.**



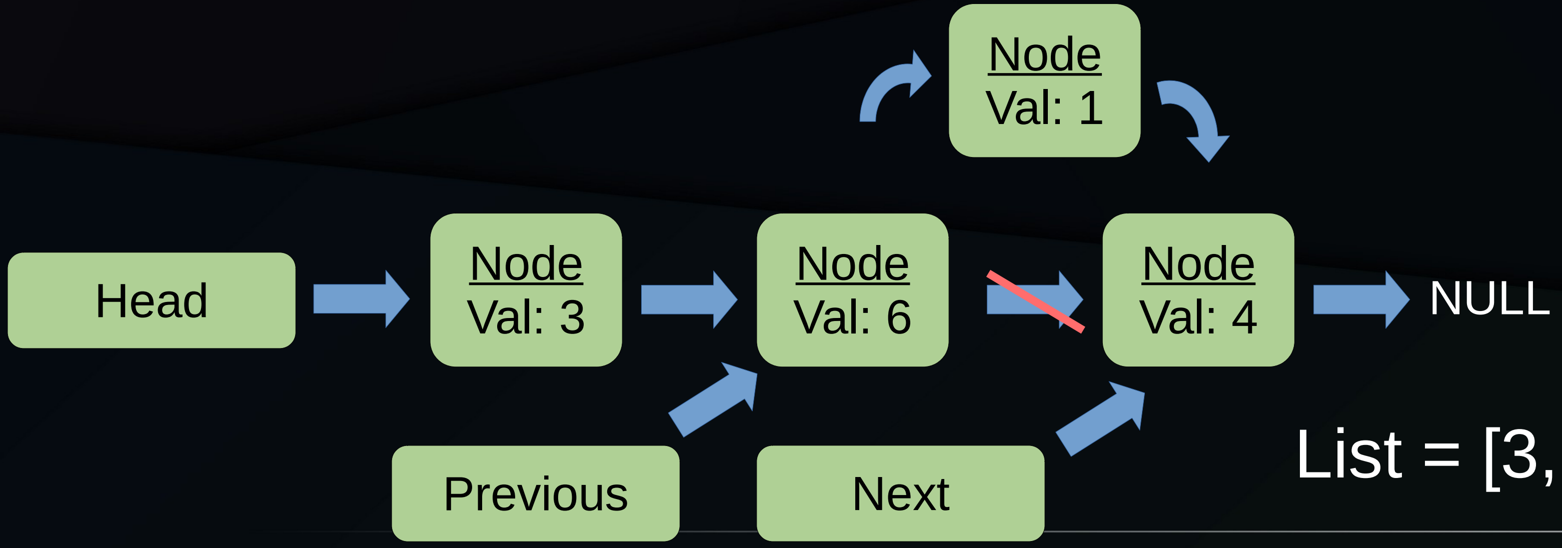
Example 6

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
void deleteNode(int target, Node* next){
    Node* prev = NULL;
    while(next!=NULL){
        if(next->value == target)
            break;
        prev = next;
        next = next->next;
    }
    prev->next = next->next;
    delete next;
}
```

Task 11: Insert a new element, given previous and next

- Set previous→next to point to the new node.
- Assign the new nodes **next** pointer to point to next.



List = [3, 6, 1, 4]

Example 7

```
class Node{
    public:
        int value;
        Node* next;
        Node(int v){
            value = v;
            next = NULL;
        }
};
```

```
void insetBetween(int newVal, Node* p, Node* n)
{
    p->next = newNode(newVal);
    p->next->next = n;
}
```

Linked Lists in Memory

- Consecutive elements of a list do <u>not</u> have to be stored in consecutive addresses.
- List nodes can be added until a program runs out of memory.

Questions or Comments?