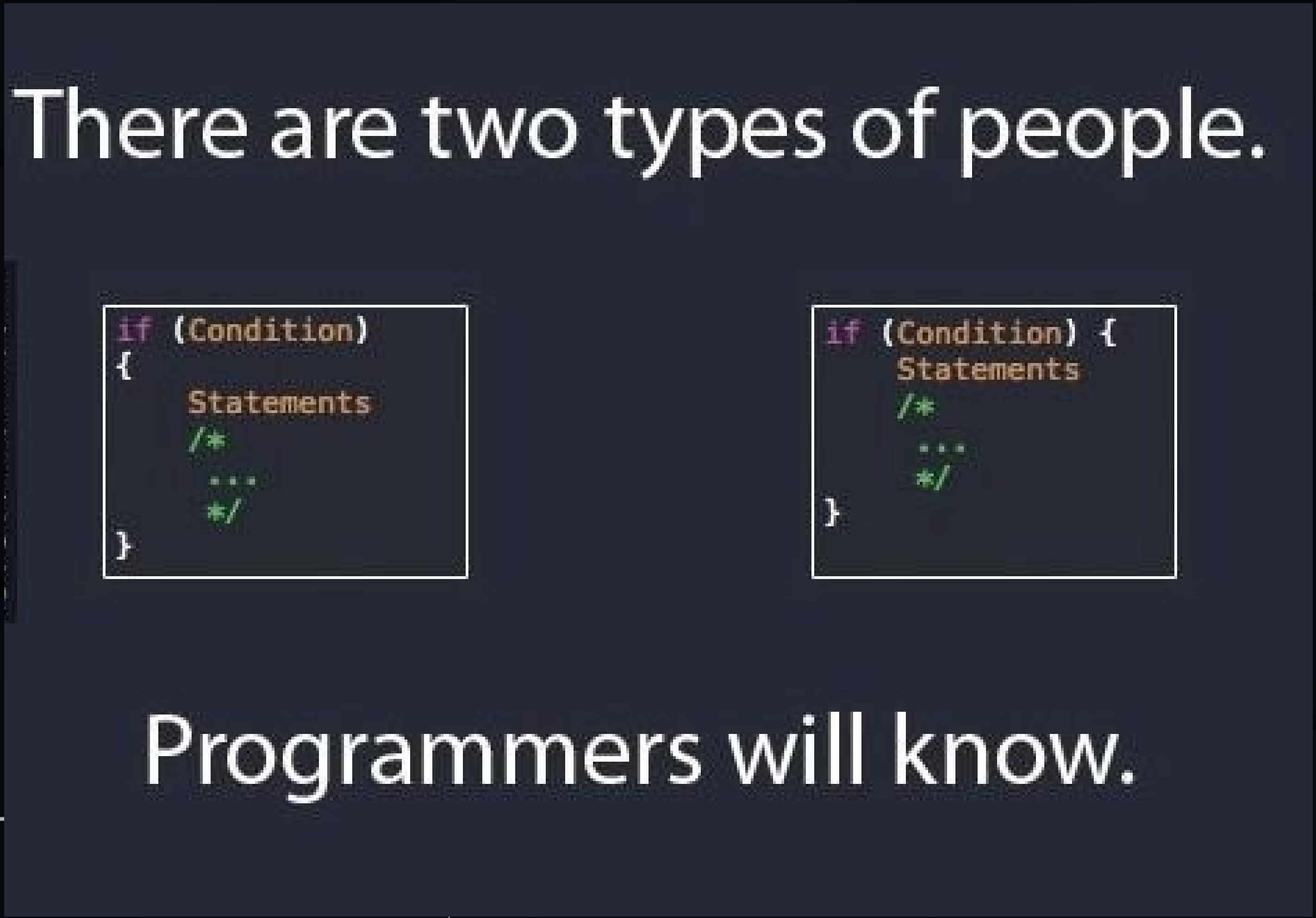

There are two types of people.
if (Condition)
{
Statements
/*
...
*/
}
if (Condition) {
Statements
/*
...
*/
}
Programmers will know.

5.2: Using Linked Lists

CS2308
Gentry Atkinson

When are Linked Lists Useful?

<u>Linked List</u>

- Insertion and deletion is cheap.
- Can hold *any* number of elements.
- Access is expensive.

<u>Array</u>

- Easy to create. Easy to delete.
- Access is easy.
- Re-sizing arrays is expensive.

Searching a Linked List

- This is an application of Task 4: Traverse the list.
- Move a pointer along list nodes until one contains the target value.
- Reaching a NULL next pointer indicates the end of the list.
- This is Sequential Search.

Sorting a Linked List

- Bubble sort and Selection sort work on linked lists.
- BUT, let’s think about Insertion sort.
  - Inserting into a linked list is much cheaper than inserting into an array.
  - Insertion sort (for every value, find the right position) is a reasonable choice for linked lists.

Example 1

```
void selectionSort(Node* head)
{
    //from: https://www.geeksforgeeks.org/iterative-selection-sort-for-linked-list/
    Node* temp = head;
    while (temp != NULL) {
        Node* min = temp;
        Node* r = temp->next;
        while (r) {
            if (min->val > r->val)
                min = r;
            r = r->next;
        }
        int x = temp->val;
        temp->val = min->val;
        min->val = x;
        temp = temp->next;
    }
} //This is a lazy solution
```

Types of linked lists

- Singly linked:
  - each node points to the next
- Doubly linked:
  - each node points to the next and the previous
- Circular:
  - the tail node points back to the head
- We use singly linked in this class.

Building a Linked List Class

- So far we have been building linked lists using raw access to Nodes.
- We could write a LinkedList class to control access to the nodes in the list.
- We could also offer our users a simpler interface to avoid hard-coding the 11 “tasks” every time we need them.

What Should a linked list do?

- Insert a value.
- Delete a value.
- Check if the list is empty.
- Print the list.
- Search for a value in the list.
- Delete the list.

Example 2

```
class LinkedList{
    private:
        class Node{
            public:
                int val;
                Node* next;
                Node(int v){val = v;next = NULL;}
        };
        Node* head;
        Node* tail;
    public:
        void insert(int v);
        void del(int v);
        bool isEmpty();
        void printList();
        bool searchList(int v);
        void delList();
        LinkedList(){head=NULL;tail=NULL;}
        ~LinkedList(){delList();}
};
```

Insert

- If head is NULL, we need to create a new Node at the head.
- Otherwise, we will create a new node following the tail node, and then shift the tail pointer to point at the new node.

Example 3

```
void LinkedList::insert(int v){
    if(head==NULL){
        head = new Node(v);
        tail = head;
    }
    else{
        tail->next = new Node(v);
        tail = tail->next;
    }
}
```

Delete

- Traverse the list looking for a Node with a matching value, with leading and following pointers.
- If we reach the end, do nothing.
- If the first Node needs to be deleted, set head to head→next.
- If the tail node needs to be deleted, set tail to the following pointer.

Example 4

```
void LinkedList::del(int v){
    Node* leading = head;
    Node* following = NULL;
    while(leading->val != v){
        following = leading;
        leading = leading->next;
        if(leading == NULL) return;
    }
    if(leading == head)
        head = head->next;
    else if(leading == tail){
        tail = following;
        tail->next = NULL;
    }
    else
        following->next = leading->next;
    delete leading;
}
```

Is the list empty?

- Return true if head is NULL.
- Return false otherwise.

Example 5

```
bool LinkedList::isEmpty(){
    return head==NULL;
}
```

Print the list

- Traverse the list.
- Print the val of every Node.
- Halt when you reach a NULL value when following next pointers.

Example 6

```
void LinkedList::printList(){
    Node* temp = head;
    while(temp != NULL){
        cout << temp->val << ' ';
        temp = temp->next;
    }
    cout << endl;
}
```

Search for a value

- Traverse the list.
- Return true if we find a Node with a matching value.
- Return false if we reach the end (NULL).

Example 7

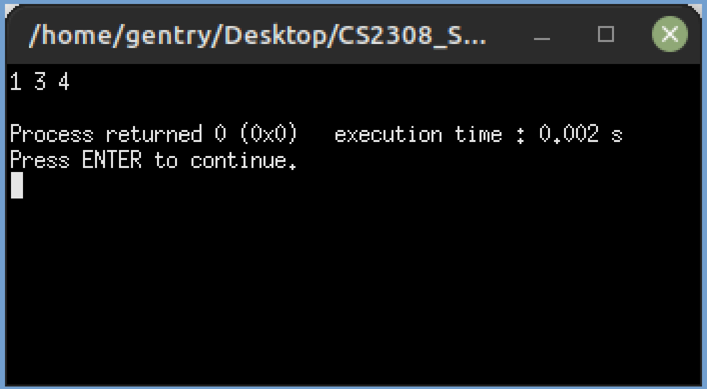
```
bool LinkedList::searchList(int v){
    Node* temp = head;
    while(temp != NULL){
        if(temp->val == v) return true;
        temp = temp->next;
    }
    return false;
}
```

Delete the list

- Every node in the list is dynamically allocated, so we need to delete all of them.
- Move the head pointer down the list with a following pointer behind it.
- Use the following pointer to delete each node.
- Set head and tail to NULL.
- Why can this function change head?

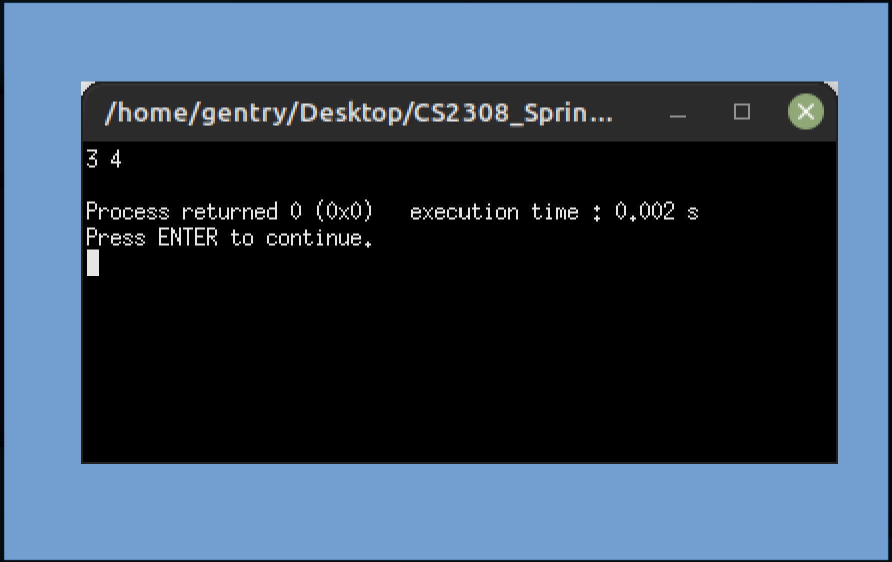
Test Run 1

```
int main(int argc, char** argv){
    LinkedList myList;
    myList.insert(1);
    myList.insert(2);
    myList.insert(3);
    myList.insert(4);
    myList.del(2);
    myList.printList();
} //try to guess the output
```



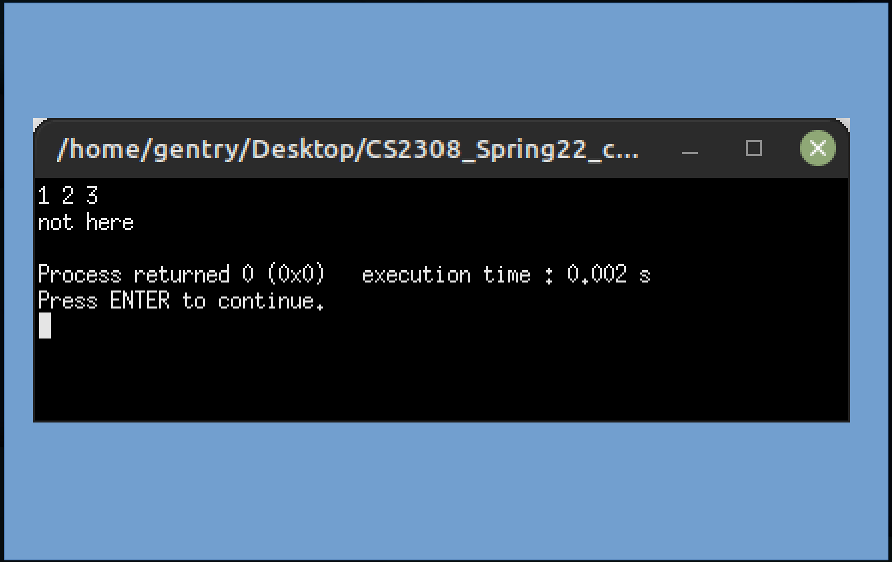
Test Run 2

```
int main(int argc, char** argv){
    LinkedList myList;
    myList.insert(1);
    myList.insert(2);
    myList.delList();
    myList.insert(3);
    myList.insert(4);
    myList.printList();
} //try to guess the output
```



Test Run 3

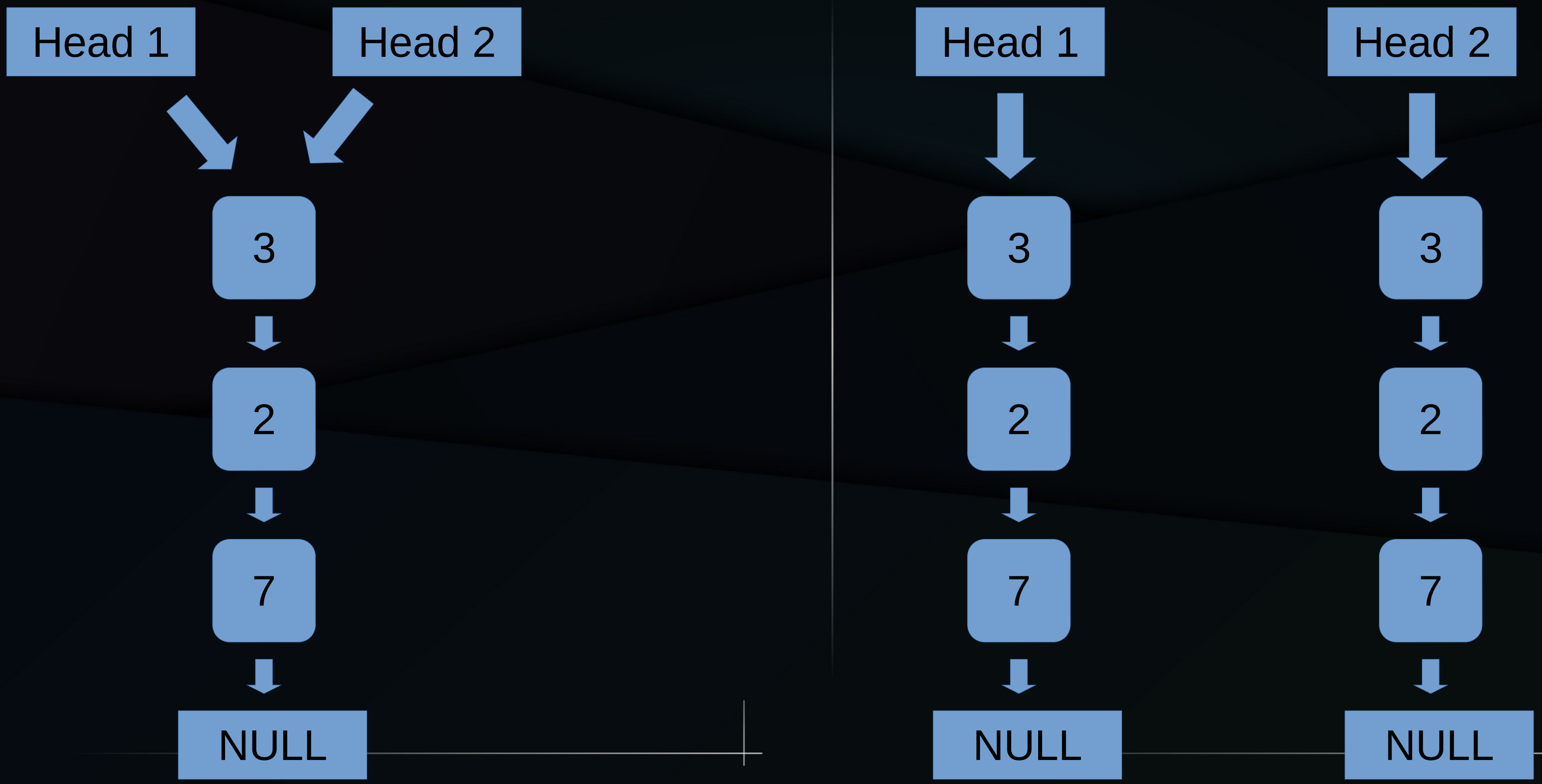
```
int main(int argc, char** argv){
    LinkedList myList;
    myList.insert(1);
    myList.insert(2);
    myList.insert(3);
    myList.printList();
    if (myList.searchList(5))
        cout << "found it" << endl;
    else
        cout << "not here" << endl;
} //try to guess the output
```



Shallow Copying vs. Deep Copying

- We can create a new pointer to point to the head of a list.
- If two pointers point to the same list, we call this a “shallow” copy. Traversing either list will print the same values, but changing one also changes the other.
- Deep copying means creating new Nodes with the same values.

Shallow Copying vs. Deep Copying



Questions or Comments?