CS 2123 Bootcamp
Linked Lists
Dr. Mark Robinson
Inserting into an Ordered Array

```c
int arr[10] = {10, 20, 40, 50, 60, 70, 80};
```

How to insert 30???
1. Move Larger Elements

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

All larger elements move 1 index to the right (use a loop)

Result:

| 10 | 20 | 40 | 40 | 50 | 60 | 70 | 80 | 0  | 0  |
2. Insert the new Value

arr

```
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 0 | 0 |
```

Done.
### What’s the Problem?

This! We make 6 assignments just to insert 1 new value

| 10 | 20 | 40 | 50 | 60 | 70 | 80 | 0   | 0   | 0   |

What if this array were 1000s of elements?

Because array elements are **contiguously stored** (i.e., physically next to each other), We have to **physically make room** to insert new values in the middle of an array.
A Different Approach

- What if instead, each element just pointed to the next value in the sequence?

- Forget the array for a second
Traversing a Linked List

1. Start a front of list
2. Do something with that node (e.g., print it)
3. Move to its next neighbor (or node)
4. Repeat until there is no next neighbor
Traversing a Linked List

Output

```
10
```
Traversing a Linked List

list

Output

10 20
Traversing a Linked List

Output

10 20 40
Traversing a Linked List

Output

```
10 20 40 50
```
Traversing a Linked List

Output

10 20 40 50 60
Traversing a Linked List

Output

10 20 40 50 60 70
Traversing a Linked List

Output

10 20 40 50 60 70 80

80 does not have a next node so we are done
Now, how to Insert?

How can we insert 30???
1. Connect **new node** to next node

30 points to the node that 20 is pointing to (i.e., 40)
2. Connect previous node to new node

20 now points to 30. Only 2 assignments necessary!

Traverse it and see if the nodes are ordered
Searching a Linked List

1. Start at the front of the list
2. Is that the node you are looking for?
3. If not, move to the next node using the next field
4. Go back to Step 2 until you:
   - Either find what you are looking for
   - Or, the next field does not point to anything
Search Return Values

- Search should return
  - Index of node in list array if using array for list
  - Pointer to node if using pointers
- Search can also return index or pointer of node where insertion could happen
- This makes insertion function a LOT easier
Search Example

Is this our search value?

int found = search(list, 30);
Search Example

Is this our search value?

```java
int found = search(list, 30);
```
Search Example

```
int found = search(list, 30);
```

Is this our search value?

Yes, so return a reference to node (e.g., index 99999)
Another Search Example

Is this our search value?

int found = search(list, 35, &insertIndex);
Another Search Example

```
int found = search(list, 35, &insertIndex);
```
Another Search Example

```
int found = search(list, 35, &insertIndex);
```
Another Search Example

```c
int found = search(list, 35, &insertIndex);
```
Another Search Example

int found = search(list, 35, &insertIndex);
Another Search Example

In this instance, search would return:
1. an indicator that it did not find 35 (e.g., -1)
2. insertIndex stores a reference to the node just before where we could insert 35 (handy!)

int found = search(list, 35, &insertIndex);
Linked List Search Problem

- Always have to linear search to access ANY node
- Big problem if the list is huge
Linked List Insertion

1. Use search function to find out where you need to insert the new node

2. Put the new node in the list (e.g., end of the list)

3. If new node is new front of list the new node points to previous front and front now points to new node

4. Otherwise, new node points to whatever insertion node is pointing to, and insertion node points to new node
Insertion Example

Search tells us where we need to insert

int found = insert(list, 35);
Insertion Example

```java
int found = insert(list, 35);
```

Add node to end of list
Insertion Example

New node points to insert node’s next

int found = insert(list, 35);
Insertion Example

```java
int found = insert(list, 35);
```

Insert node’s next points to new node
Exercise 3

- Use my linked list library to build a linked list containing the integers: 10, 20, 30
- Also search for 20 and 25
- Lines that start with "***" are in the main()
Exercise 3 continued

#define MY_NODE_DEBUG 1
#define MY_NODE_TRUE 1
#define MY_NODE_FALSE 0

typedef struct MyNode {
    int iVal;
    struct MyNode * next;
} MyNode;

//prototypes
void printList(MyNode * head);
MyNode * allocateNode(MyNode * node);
void freeList(MyNode * head);

MyNode * searchLL(MyNode *pHead, int matchInt, MyNode **ppPrecedes);
MyNode * insertLL(MyNode **ppHead, MyNode * node);
Exercise 4

- Implement your own searchLL function
  - Use my_node_skeleton.c
- Start by compiling with the skeleton code and drawing out what is happening
- Next, iteratively design, code, and test the function
- And add a few more test cases to better cover the code