Programming Abstractions

CS106B

Cynthia Bailey Lee

Julie Zelenski

Topics:

- **Review: Pointers**
- **Today: Link Nodes**
  - LinkNode struct
  - Chains of link nodes
  - LinkNode operations

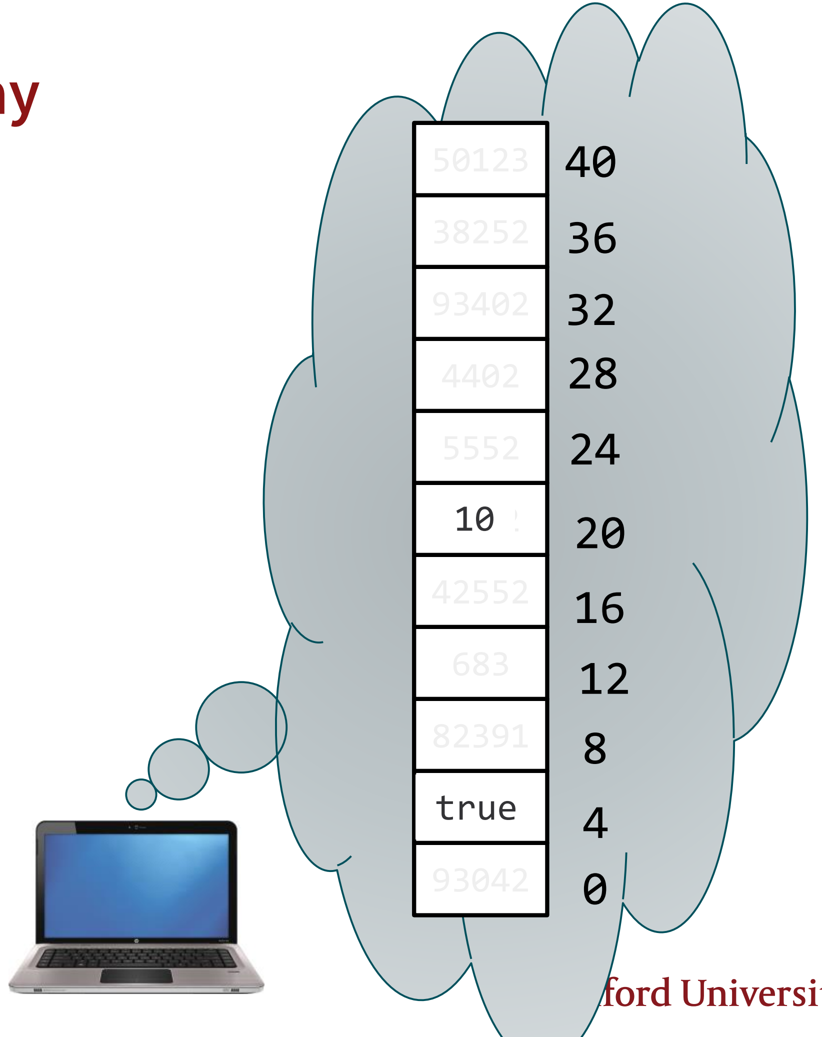
Pointers

TAKING A DEEPER LOOK AT THE SYNTAX OF THAT ARRAY ON THE HEAP

Memory is a giant array

```
bool kitkat = true;
int candies = 10;
```

- Whenever you declare a variable, you allocate a bucket (or more) of memory for the value of that variable
- Each bucket of memory has a **unique address**



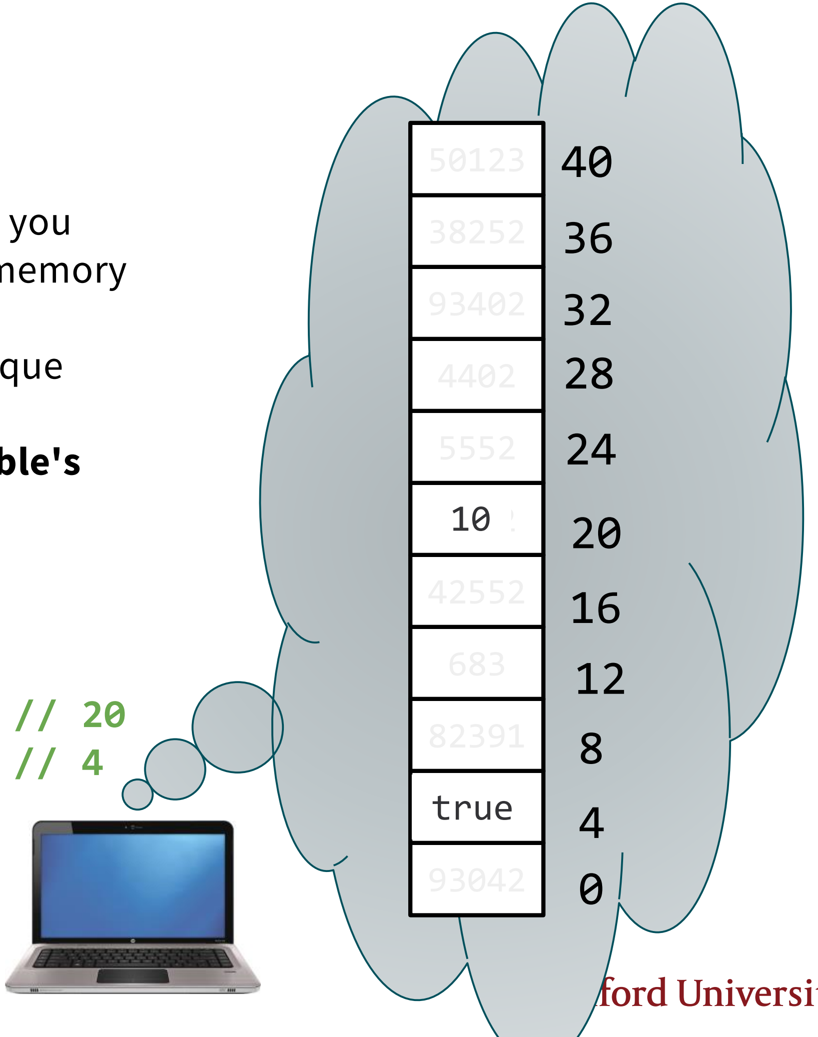
Address-of operator &

Whenever you declare a variable, you allocate a bucket (or more) of memory for the value of that variable

Each bucket of memory has a unique address

You can get the value of a variable's address using the & operator.

```
int candies = 10;
bool kitkat = true;
cout << &candies << endl;    // 20
cout << &kitkat << endl;     // 4
```



Address-of operator &

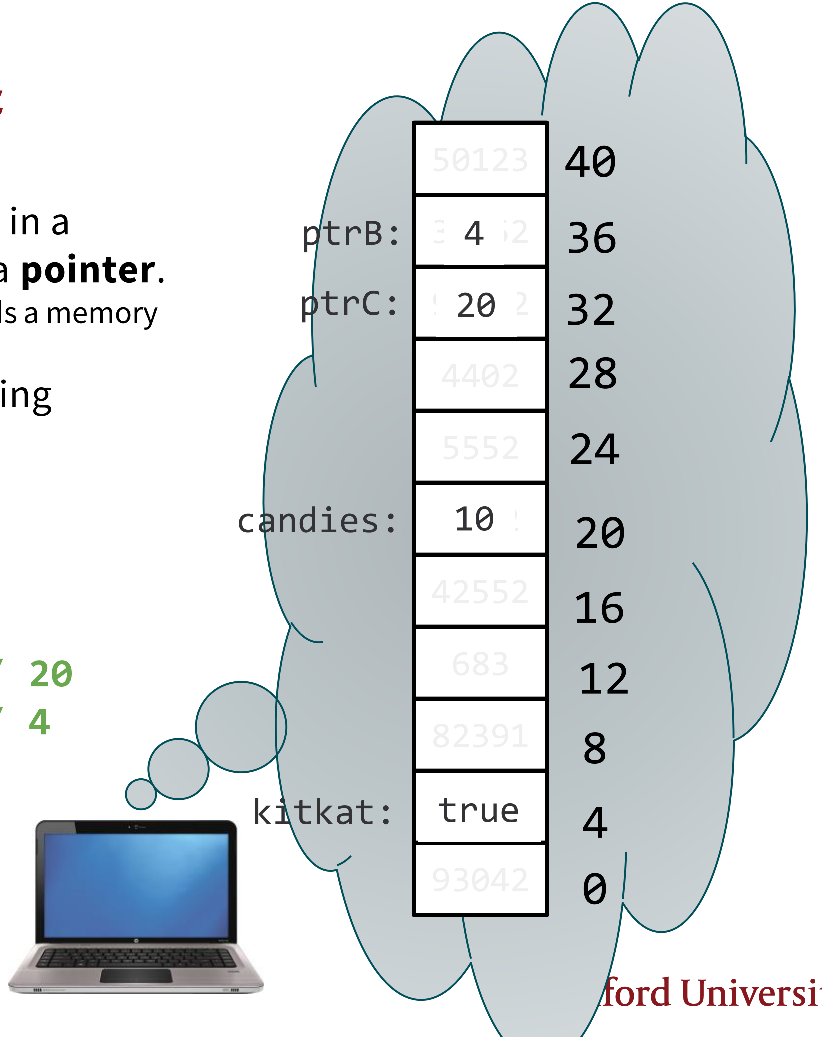
You can store memory addresses in a special type of variable called a **pointer**.

- i.e. A pointer is a variable that holds a memory address.

You can declare a pointer by writing *(The type of data it points at)**

- e.g. `int*`, `string*`

```
int candies = 10;
bool kitkat = true;
cout << &candies << endl;    // 20
cout << &kitkat << endl;     // 4
int* ptrC = &candies;
bool* ptrB = &kitkat;
```



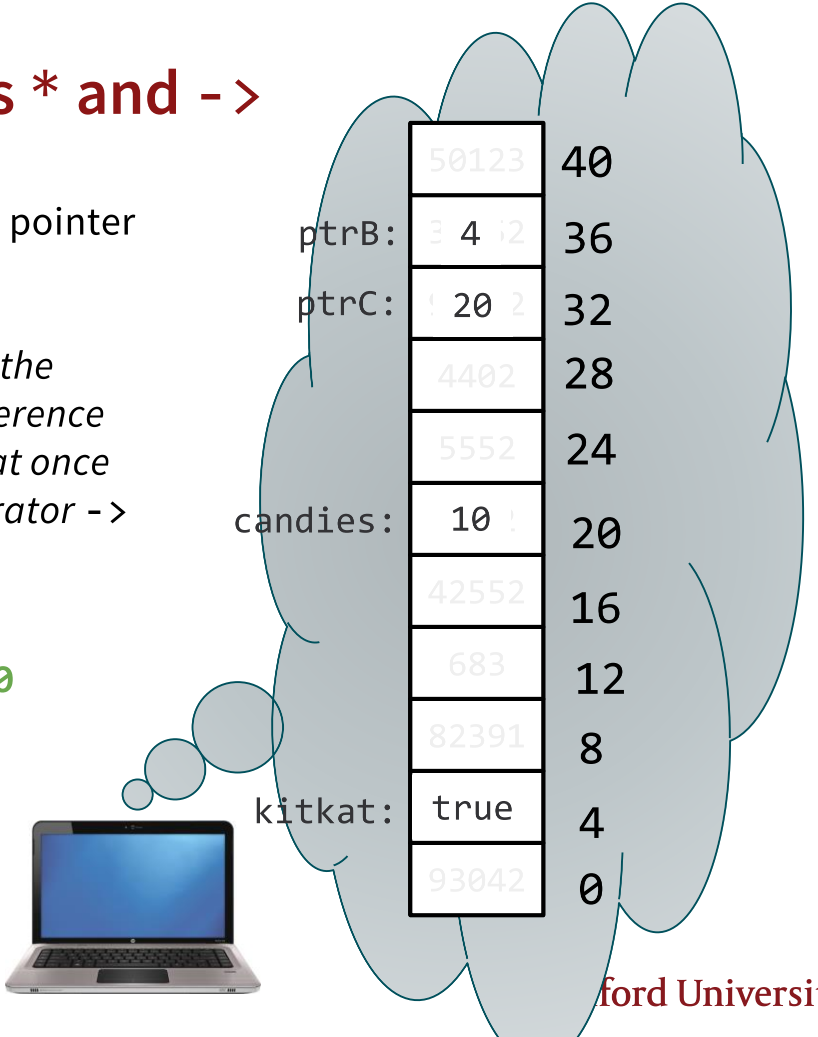
Dereference operators * and ->

You can follow ("**dereference**") a pointer by writing
variable_name

Remember that if what we find at the destination is a struct, we dereference AND access a field of the struct at once with the struct dereference operator ->

```
int candies = 10;
bool kitkat = true;
cout << &candies << endl;    // 20
cout << &kitkat << endl;     // 4
int* ptrC = &candies;
bool* ptrB = &kitkat;

cout<< ptrC << endl; // 20
cout<< *ptrC << endl; // 10
```



Null Pointer

A SPECIAL POINTER VALUE

Null Pointer

- When we want a variable with a pointer type to be "blank," we set it to be a "null pointer"
- This means it doesn't point to any valid memory address
- This turns out to be useful if you want a pointer to be shown as in a "waiting" state (waiting to be set to a real pointer value/memory address)

- Example:

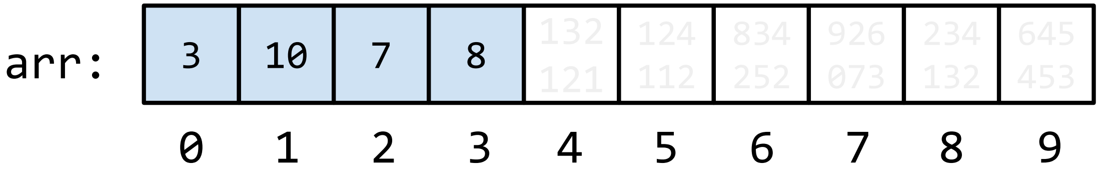
```
int* myptr = nullptr;
…
if (input > 0) {
    myptr = new int[input];
}
…
if (myptr == nullptr) {
    cout << "haven't assigned a value to myptr yet!" << endl;
}
```

Array Performance

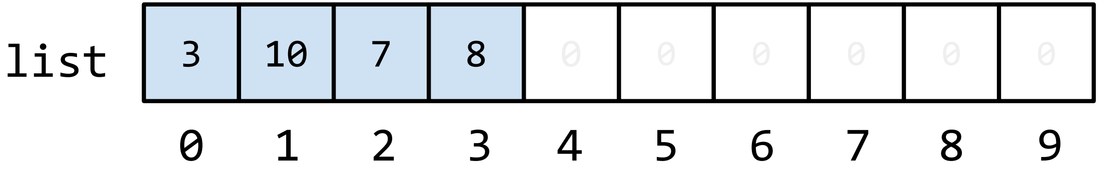
LIMITATIONS OF THE ARRAY,
AND A MORE FLEXIBLE
ALTERNATIVE

Arrays

What are arrays good at? What are arrays bad at?



Array Performance



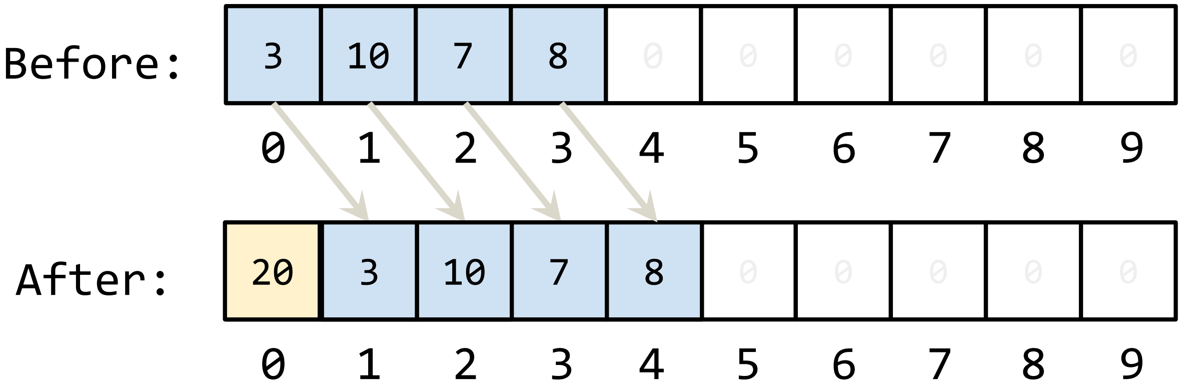
What are the most annoying operations on a tightly packed row of theater seats, or a tightly packed book shelf, etc?

Insertion - **O(n)**
Deletion - **O(n)**
Lookup (given index/memory address) - **O(1)**

Let's brainstorm ways to improve insertion and deletion....

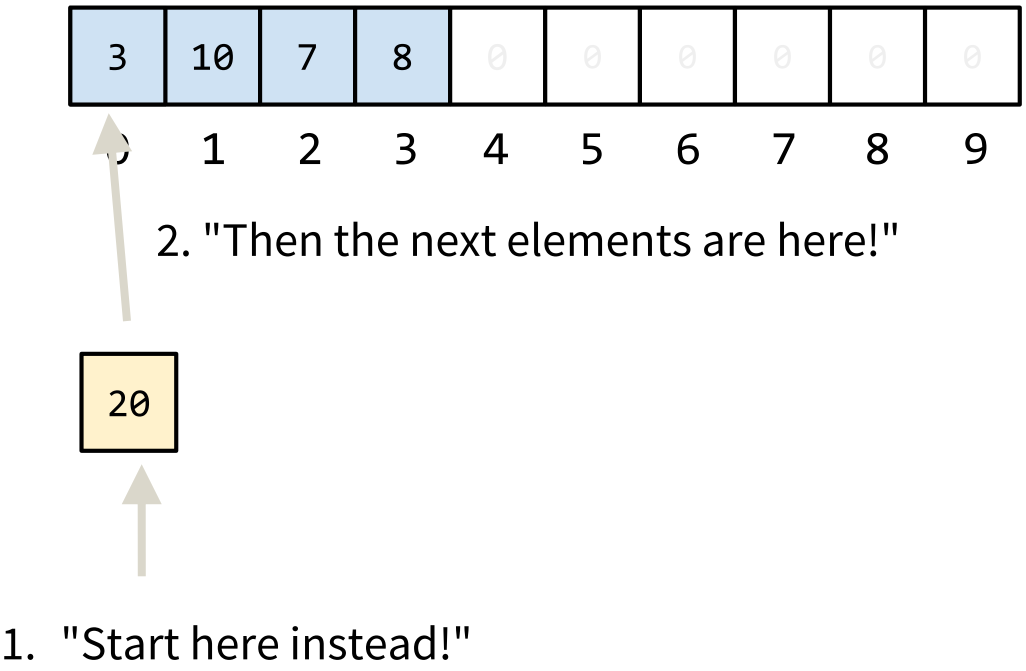
Add to front

What if we were trying to add an element "20" at index 0?



Add to front

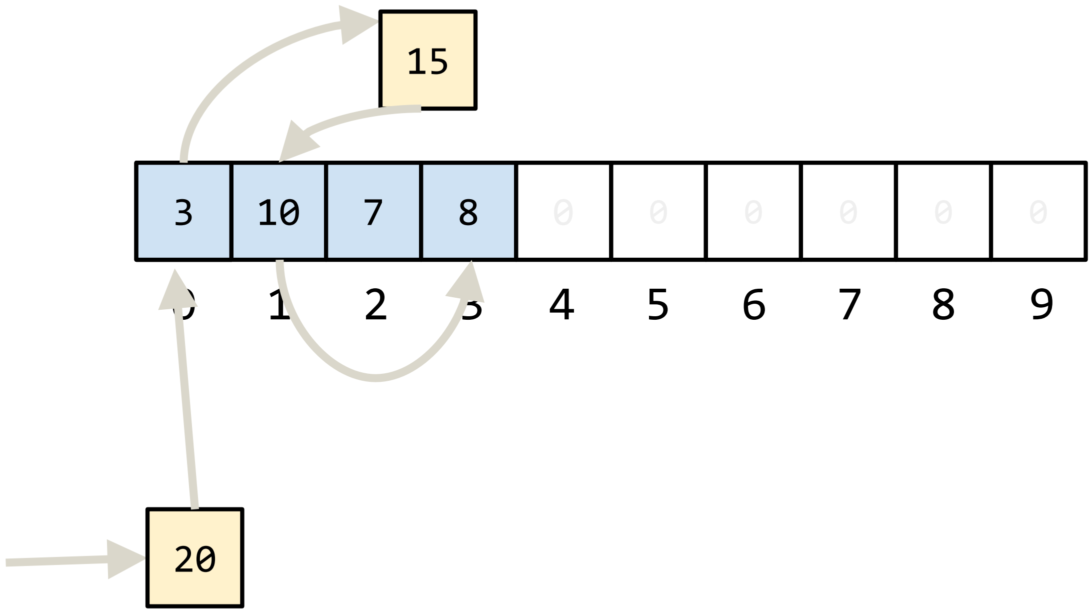
Wouldn't it be nice if we could just do something like:



More operations

Now we add 15 as a new 3rd element, and remove the 7:

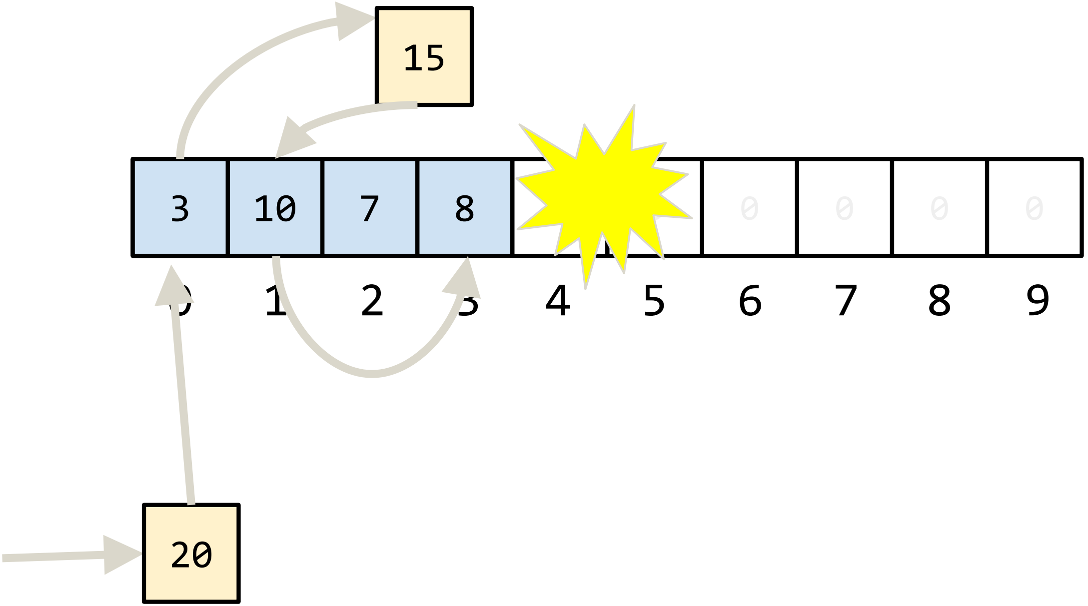
Arrows everywhere! (but no scooting over in those array buckets/seats, at least…)

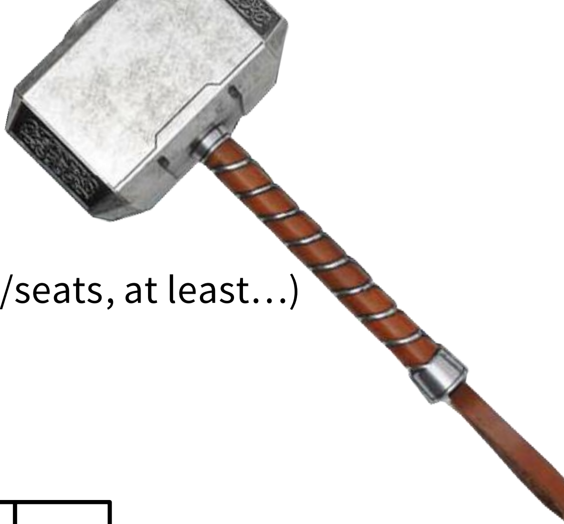


More operations

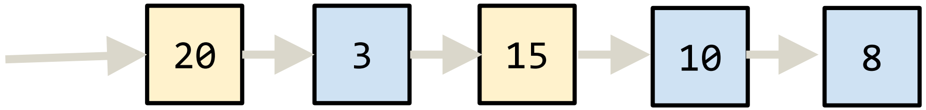
Now we add 15 as a new 3rd element, and remove the 7:

Arrows everywhere! (but no scooting over in those array buckets/seats, at least…)





This is a list of linked nodes!



- A list of linked nodes (or a linked list) is composed of interchangeable nodes
- Each element is stored separately from the others (vs contiguously in arrays)
- Elements are chained together to form a one-way sequence using pointers
- Edits are easier than an array in that no “scooting over” is needed!

Linked Nodes

A GREAT WAY TO EXERCISE YOUR POINTER UNDERSTANDING

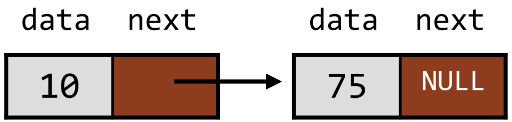
Linked Node Struct

- To enable each bucket of the more flexible array alternative to both hold a value *and* tell you where to look for the next value, we need a `struct` with two fields:

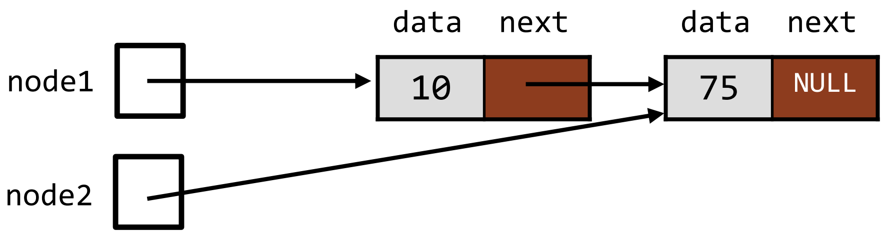
```
struct LinkNode {
   int data;
   LinkNode* next;
};
```

  - `data`: the data being stored (what would be in the array)
  - `next`: a pointer to the next node `struct` in the sequence (or `nullptr` if this is the end of the sequence)

- The result is a chain that looks like this:

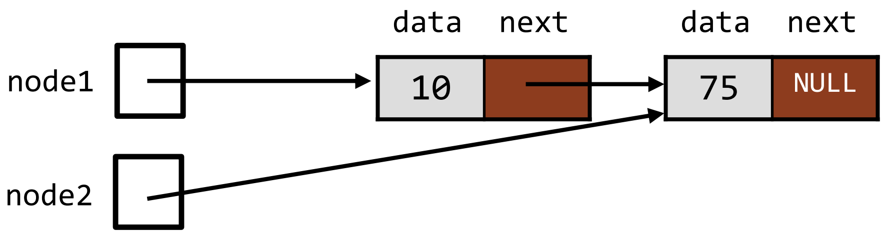


Your Turn: finish the code to match the picture




```
LinkNode* node1 = new LinkNode;
node1->data = 10;
LinkNode* node2 = new LinkNode;
node2->data = 75; // YOUR TURN: complete the code to make picture
```

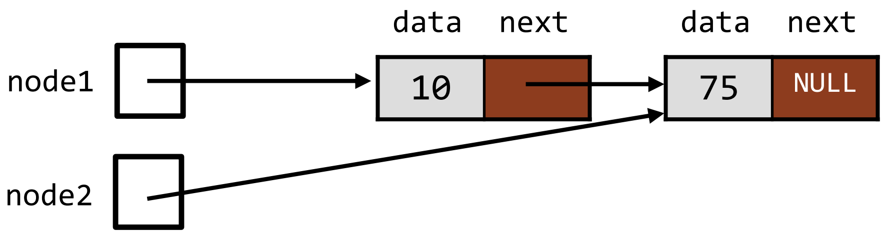
Your Turn: finish the code to match the picture




```
LinkNode* node1 = new LinkNode;
node1->data = 10;
LinkNode* node2 = new LinkNode;
node2->data = 75; // YOUR TURN: complete the code to make picture

node1->next = node2; // correct answer
```

Your Turn: finish the code to match the picture

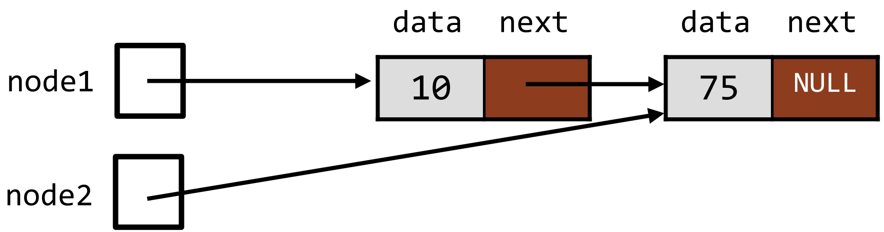



```
LinkNode* node1 = new LinkNode;
node1->data = 10;
LinkNode* node2 = new LinkNode;
node2->data = 75; // YOUR TURN: complete the code to make picture

node1->next = node2; // correct answer
```

IMPORTANT: ASSIGNMENT OPERATOR WITH POINTERS
When assigning one pointer to another, we are making the two pointers *point to the same destination*. We are *not* making the one on the right point to the one on the left as its destination.

Your Turn: finish the code to match the picture

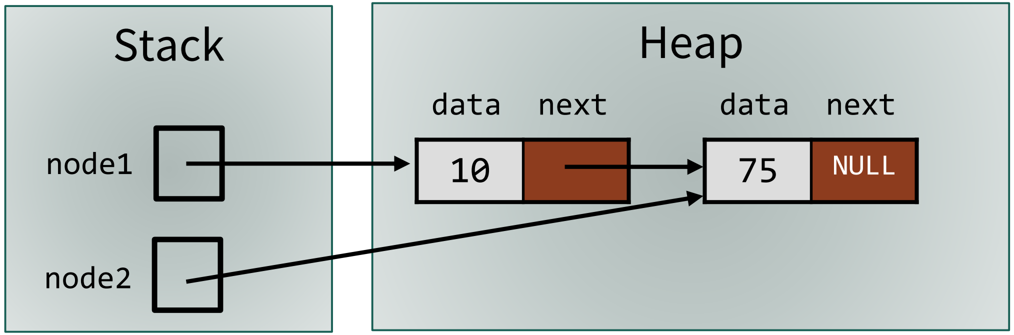



```
LinkNode* node1 = new LinkNode;
node1->data = 10;
LinkNode* node2 = new LinkNode;
node2->data = 75; // YOUR TURN: complete the code to make picture

node1->next = node2; // correct answer
```

Note: After this point, we don't really need the pointer variable named `node2` anymore. The node it points to may be reached via the variable `node1`.

Your Turn: finish the code to match the picture




```
LinkNode* node1 = new LinkNode;
node1->data = 10;
LinkNode* node2 = new LinkNode;
node2->data = 75; // YOUR TURN: complete the code

node1->next = node2; // correct answer
```

Review/Reminder: the variables node1 and node2 are local variables, so they'll be stored in the **stack** part of memory. The nodes themselves will be stored in the **heap** part of memory, since we got them from new.

FIRST RULE OF LINKED NODE/LISTS CLUB:

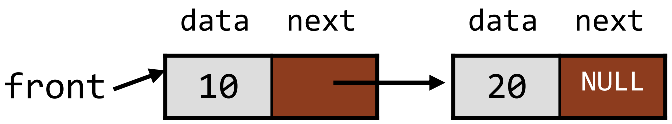
DRAW A PICTURE OF LINKED LISTS

Do no attempt to code linked nodes/lists without pictures!

List code example: Draw a picture!

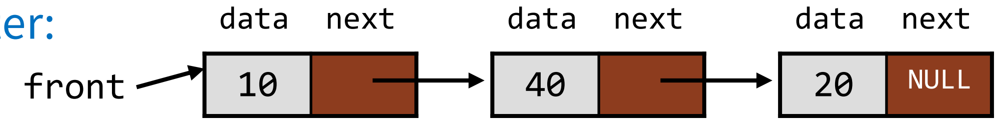
```
struct LinkNode {
  int data;
  LinkNode* next;
};
```

Before:

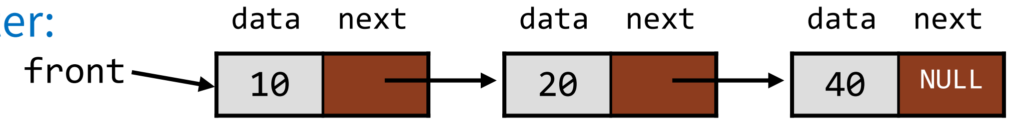



```
front->next->next = new LinkNode;
front->next->next->data = 40;
```

A. After:



B. After:



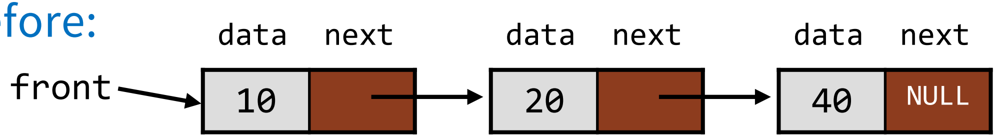
C. Using `next` that is `nullptr` gives an error

D. Other/none/more than one

List code example: Draw a picture!

```
struct LinkNode {
  int data;
  LinkNode* next;
};
```

Before:



Write code that will put these in the reverse order: