Programming Abstractions

CS106B

Cynthia Lee

Topics:

- **Pointers**
  - new/delete dynamic memory allocation
  - What is a pointer?
- **Link Nodes**
  - LinkNode struct
  - Chains of link nodes
  - LinkNode operations

Pointers

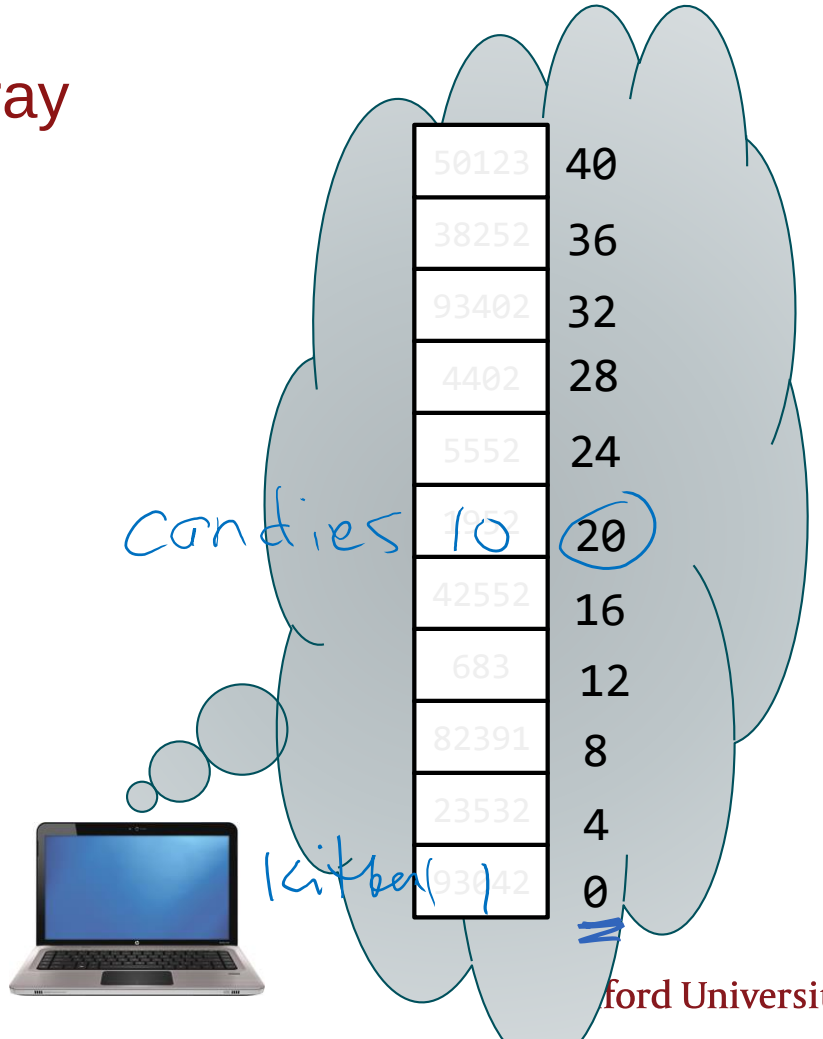
Taking a deeper look at the syntax of that array on the heap

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**



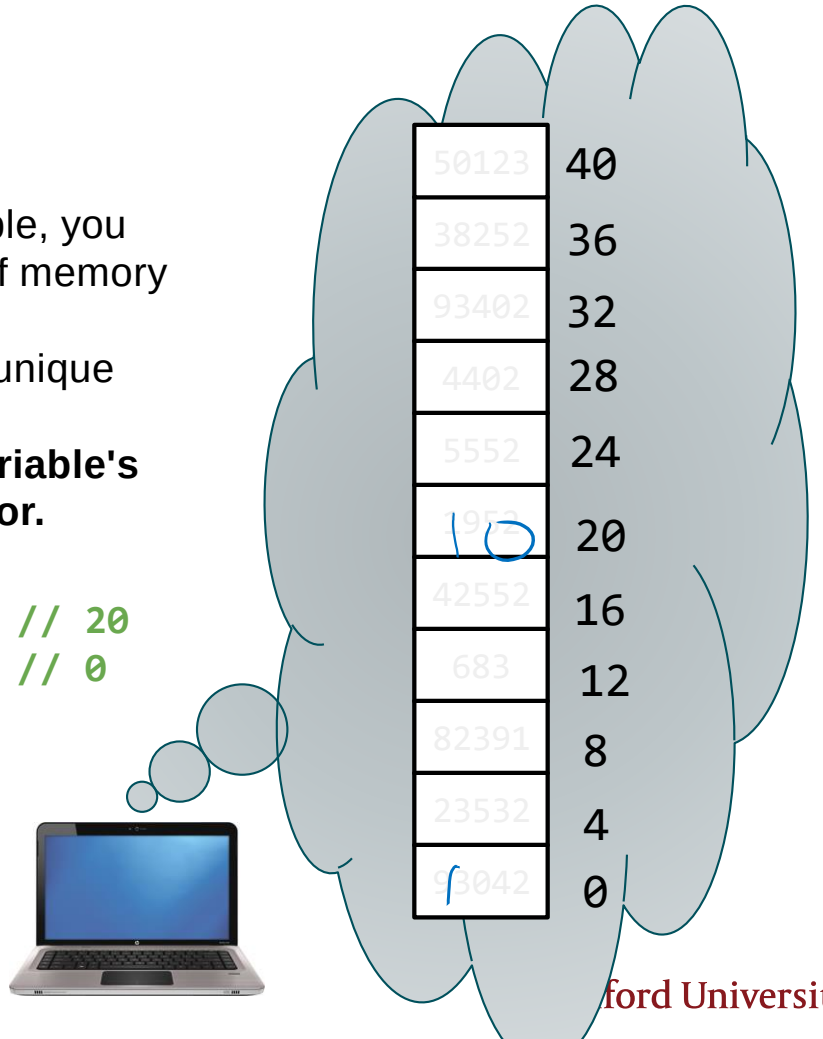
Memory addresses

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.

```
cout << &candies << endl;    // 20
cout << &kitkat << endl;     // 0
```



Memory addresses

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

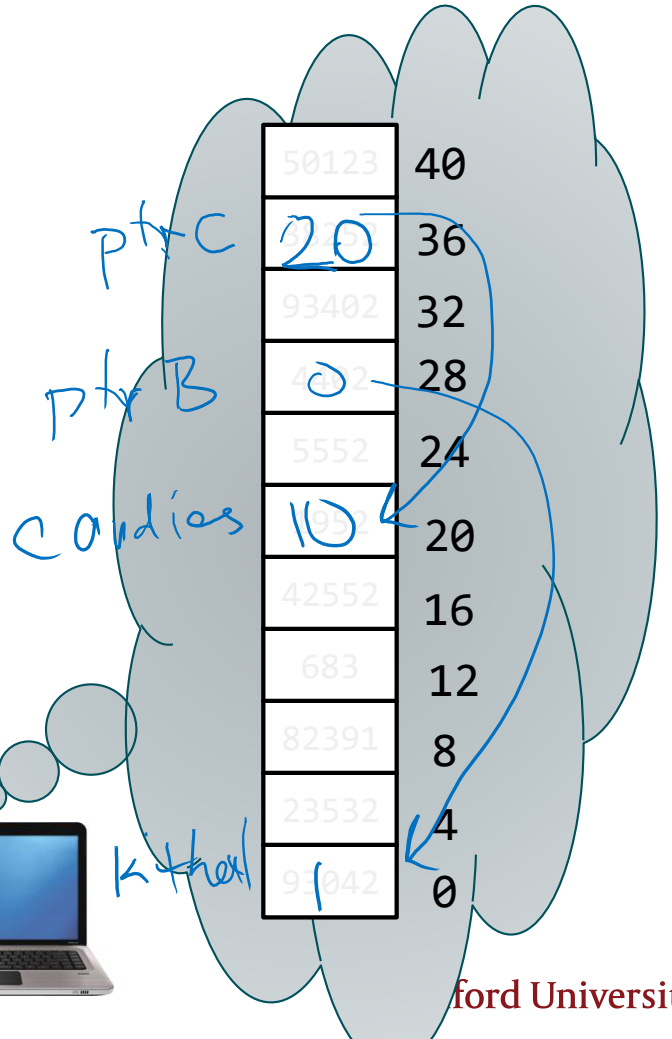
- i.e. A pointer is a variable that holds a memory address.

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

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

```
cout << &candies << endl;    // 20
cout << &kitkat << endl;     // 0
int* ptrC = &candies;    // 20
bool* ptrB = &kitkat;     // 0
```

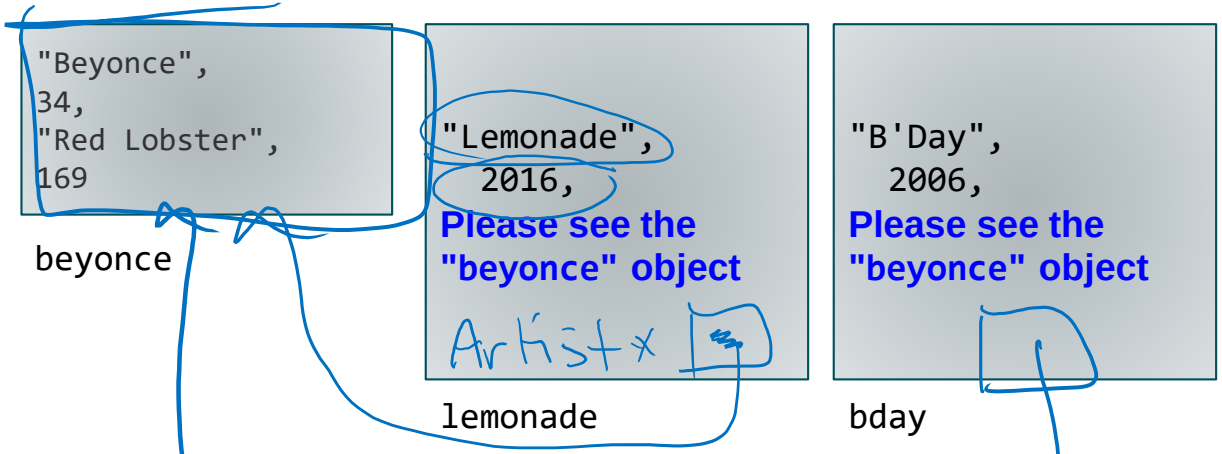
cout << &ptrC << endl;



Pointers

Picking up where we left off: remember the Album struct, and our new Artist struct….

What do we really want?



The album's artist field should **point to** the “beyonce" data structure instead of storing it.

How do we do this in C++?

…pointers!

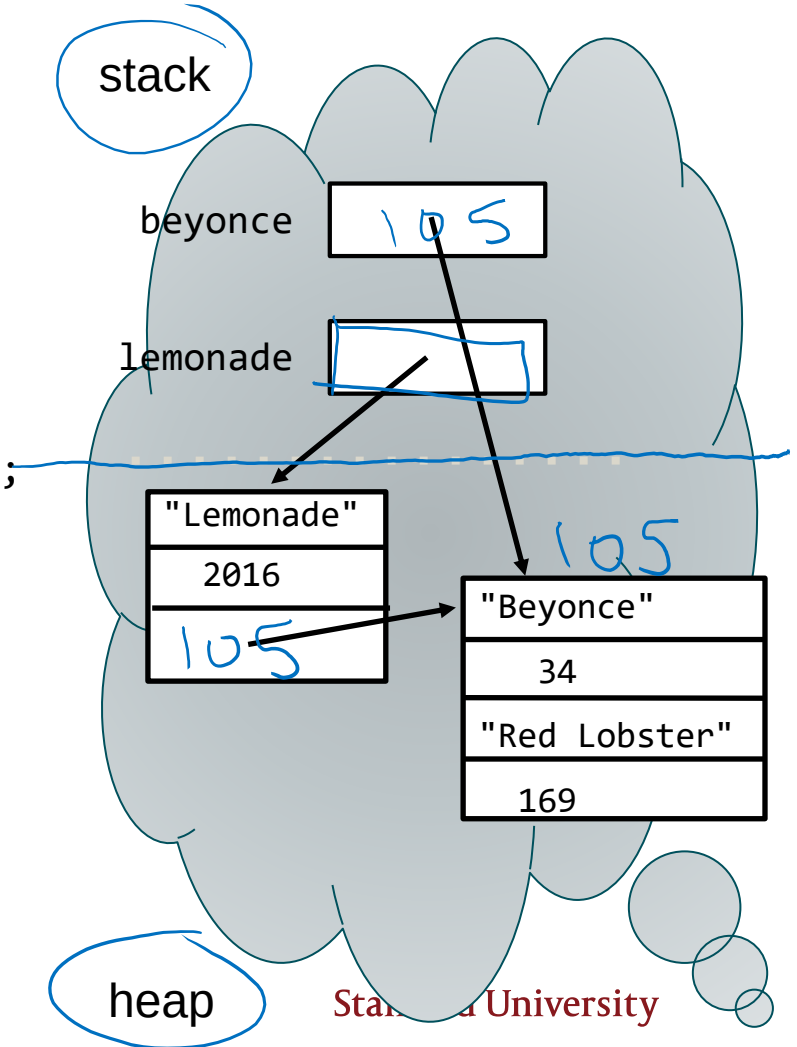
New and delete with structs!

Example:

```
Artist* beyonce = new Artist;
beyonce->name = "Beyonce";
beyonce->age = 34;
beyonce->favorite_food = "Red Lobster";
beyonce->height = 169;

Album* lemonade = new Album;
lemonade->title = "Lemonade";
lemonade->year = 2016;
lemonade->artist = beyonce;

delete beyonce;
delete lemonade;
```

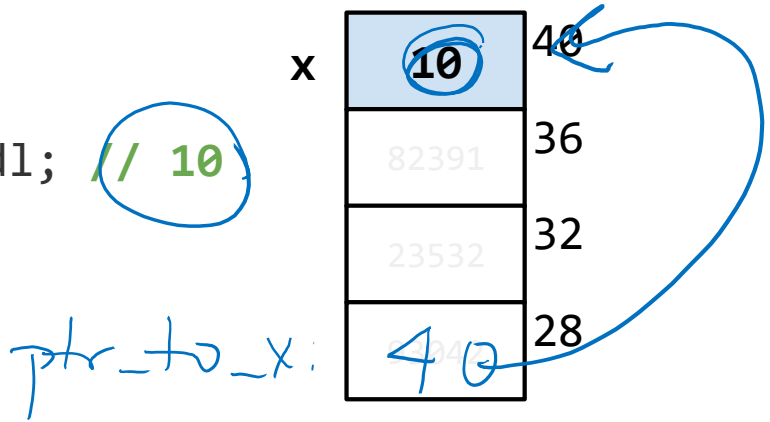


Next steps with pointers and structs/classes/objects

"Dereferencing" a pointer

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

```
int x = 10;
int* ptr_to_x = &x;
cout << *ptr_to_x << endl; // 10
```



Fixing the Album/Artist example with pointers

```
struct Artist {
  string name;
  int age;
  string favorite_food;
  int height; // in cm
};
```

```
struct Album {
  string title;
  int year;
  Artist* artist;
};
```

```
Artist* britney = new Artist;
// TODO: now we need to set the fields of britney
(*britney).name = "Britney Spears"; // this works but really clunky

Album blackout = { "Blackout", 2007, britney };
Album circus = { "Circus", 2008, britney };
```

-> operator: Dereferencing and accessing a member

```
struct Artist {
  string name;
  int age;
  string favorite_food;
  int height; // in cm
};
```

```
struct Album {
  string title;
  int year;
  Artist* artist;
};
```

```
Artist* britney = new Artist;
// TODO: now we need to set the fields of britney
britney->name = "Britney Spears"; // ptr->member is the exact same as (*ptr).member

Album blackout = { "Blackout", 2007, britney };
Album circus = { "Circus", 2008, britney };
```

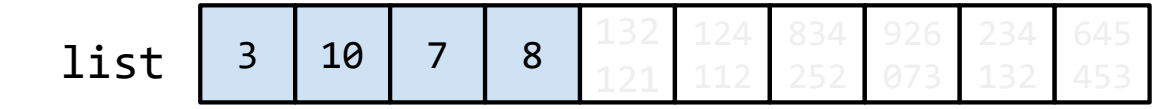
Linked Nodes

Another important application of pointers

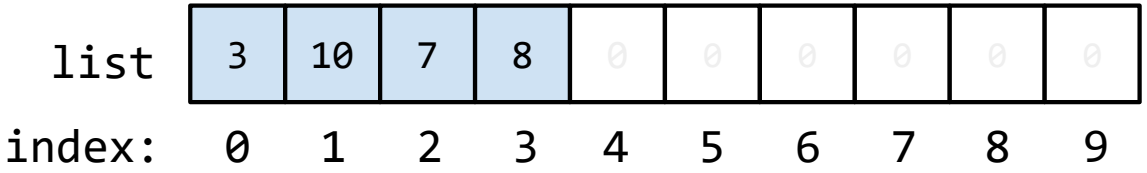
We'll start by looking at a limitation of the array

Arrays

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



Memory is a giant array...



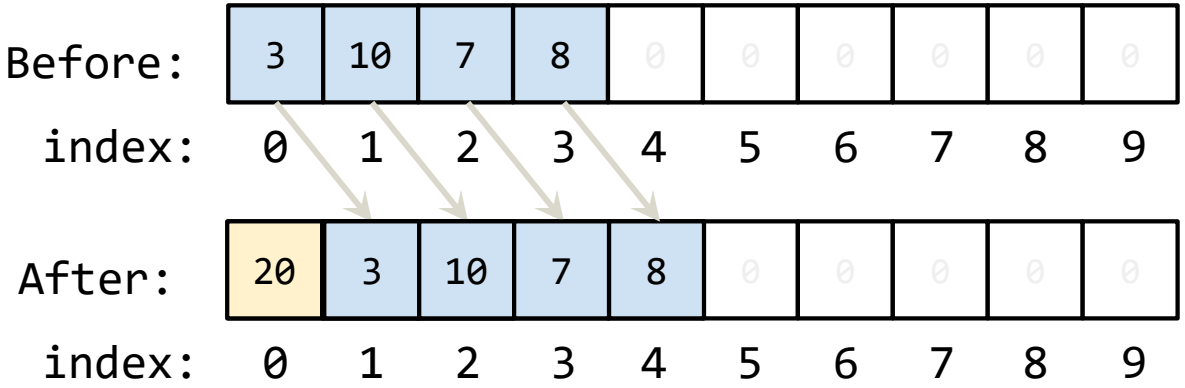
What are the most annoying operations on a tightly packed book shelf, liquor cabinet, shoe closet, etc?

Insertion - **O(n)**
Deletion - **O(n)**
Lookup - **O(1)**

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

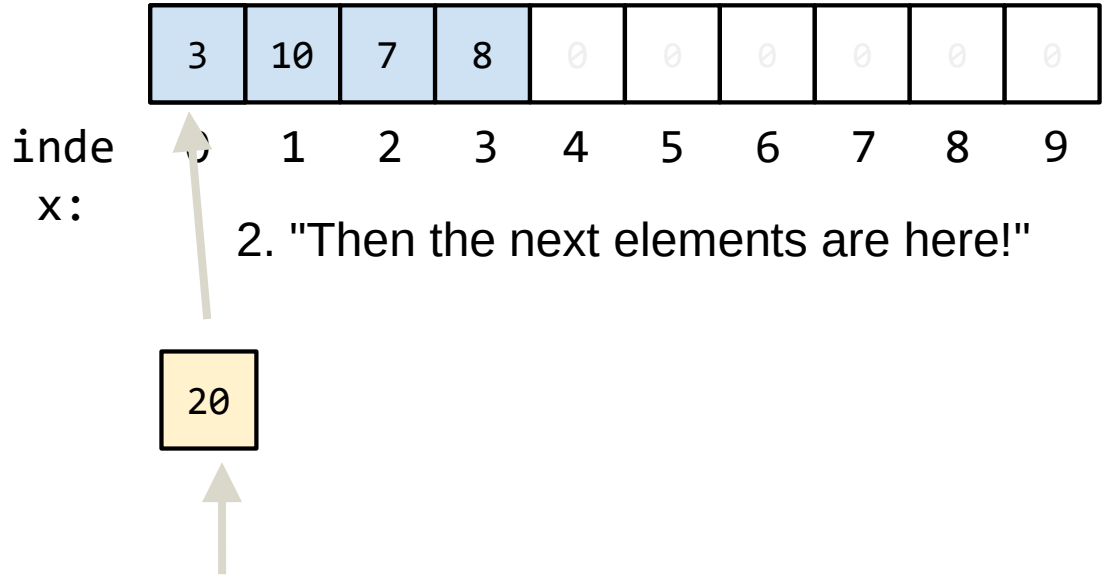
Add to front

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

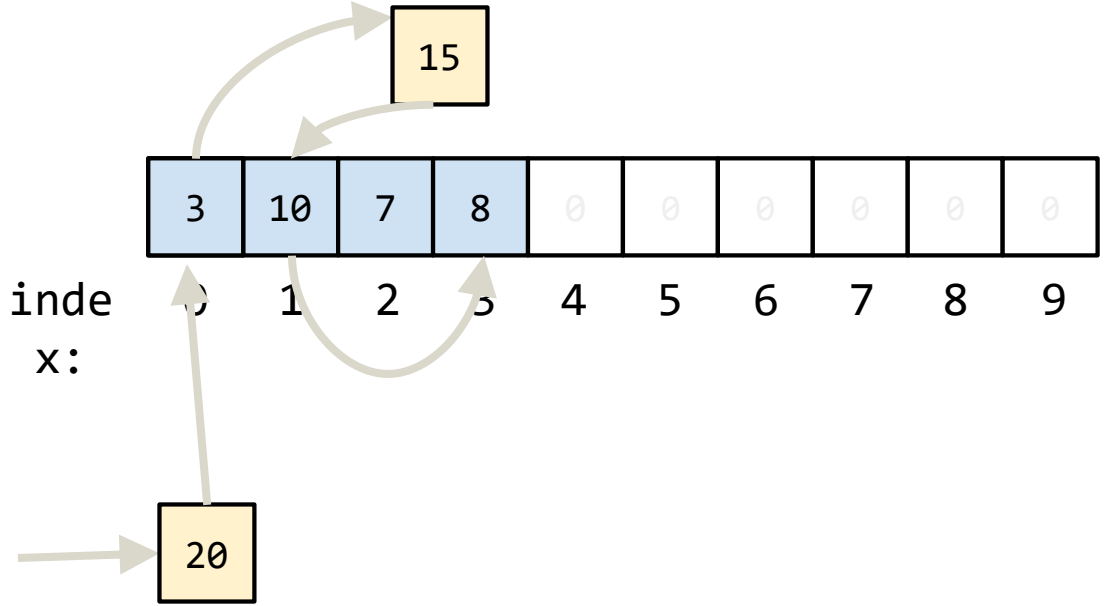


Add to front

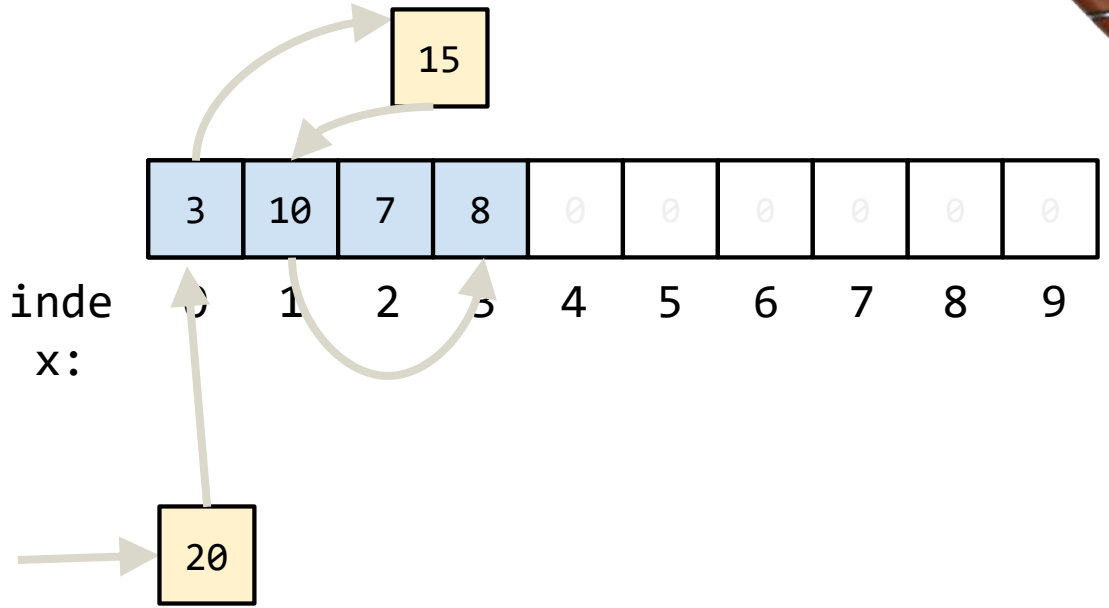
Wouldn't it be nice if we could just do something like:



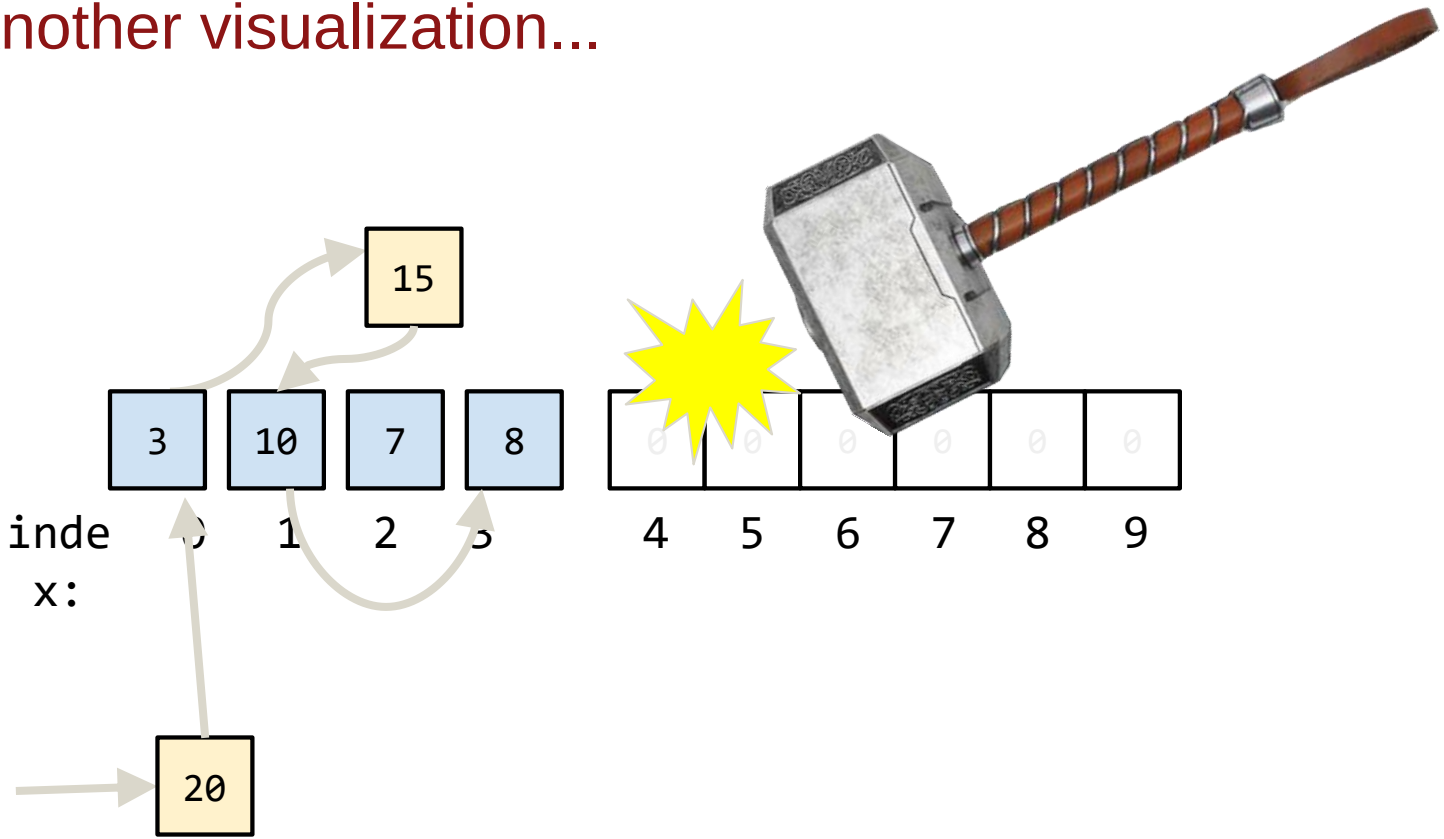
Now we add to the front again: Arrows everywhere!



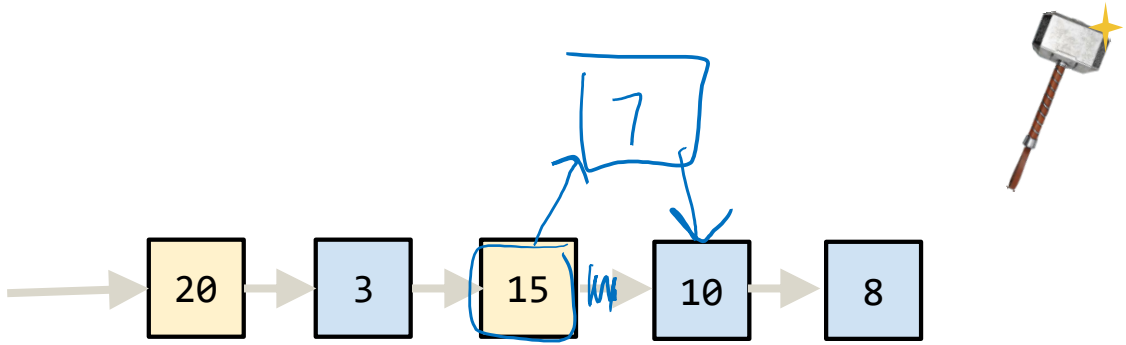
Another visualization...



Another visualization...



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

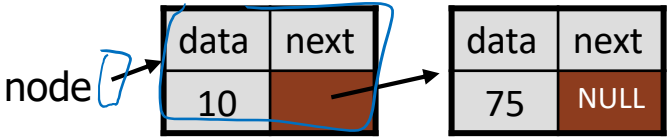
Linked Nodes

A great way to exercise your pointer understanding

Linked Node

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

- We can chain these together in memory:




```
LinkNode *node1 = new LinkNode;
node1->data = 10;
node1->next = NULL;
LinkNode *node = new LinkNode;
node->data = 10;
node->next = node1;
```

```
// complete the code to make picture
```

node->next->data
=75;

FIRST RULE OF LINKED NODE/LISTS CLUB:

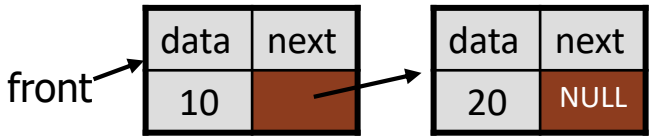
DRAW A PICTURE OF LINKED LISTS

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

List code example: Draw a picture!

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

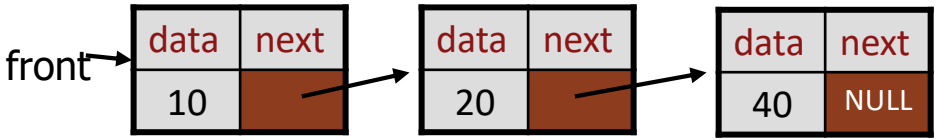
Before:




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

A. After:

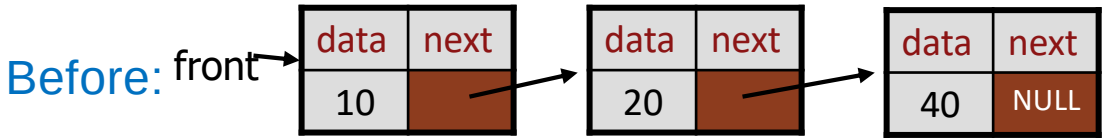
B. After:



C. Using “next” that is NULL gives error

D. Other/none/more than one

List code example: Draw a picture!




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

Write code that will put these in the reverse order.