Week 7 Part 2

Kyle Dewey

Overview

- `NULL`
- Recursion
- Exam #2 Review

NULL

Recall...

- When a file can’t be opened with `fopen`, it returns `NULL`
- `NULL` is a special value in C

NULL

- Is zero at the binary representation
- Is a pointer
- Can be assigned to any pointer type

```
char* string = NULL;
int* arr = NULL;
```

NULL

- Can be used as a sentinel value
- Often used to show that an operation couldn't be performed
  - Return NULL if we can't open a file, etc.

Example

```
#define LENGTH 4
int array[] = { 1, 2, 3, 4 };
int* subarray( int start ) {
  if ( start >= 0 &&
       start < LENGTH ) {
    return &(array[ start ]);
  } else {
    return NULL;
  }
}
```

Caveats

- If we try to dereference a `NULL` pointer, the program will crash, usually with a **segmentation fault**
    - This means it tried to access memory that it didn’t have permission to access

```
char* string = NULL;
string[ 2 ] = ‘f’; // crash
printf( “%s\n”, string ); // crash
```

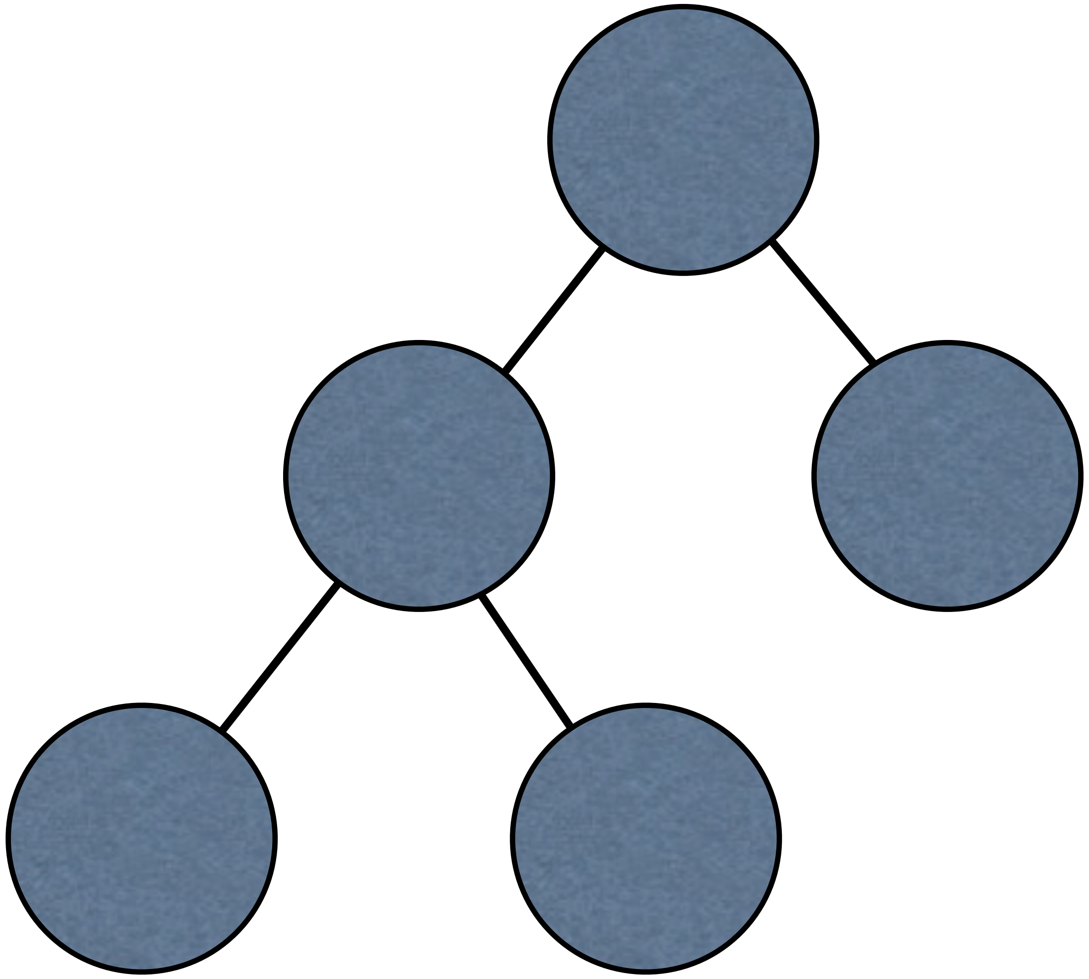
Caveat

- It can also make code trickier
  - Is `NULL` a possible value?
- In real code, there are places where `NULL` is impossible but people check anyway (and vice-versa!)

Recursion

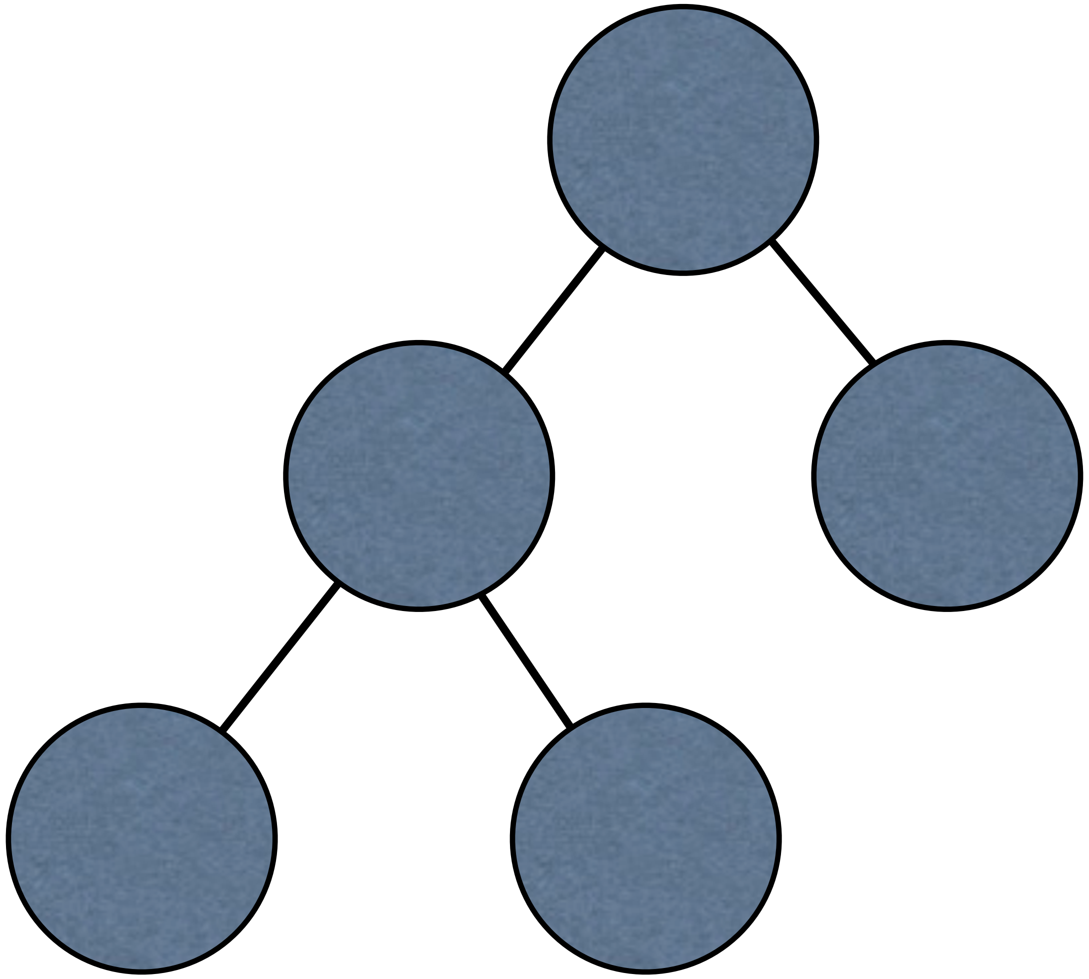
Binary Tree

- Mathematical concept
- A special kind of graph



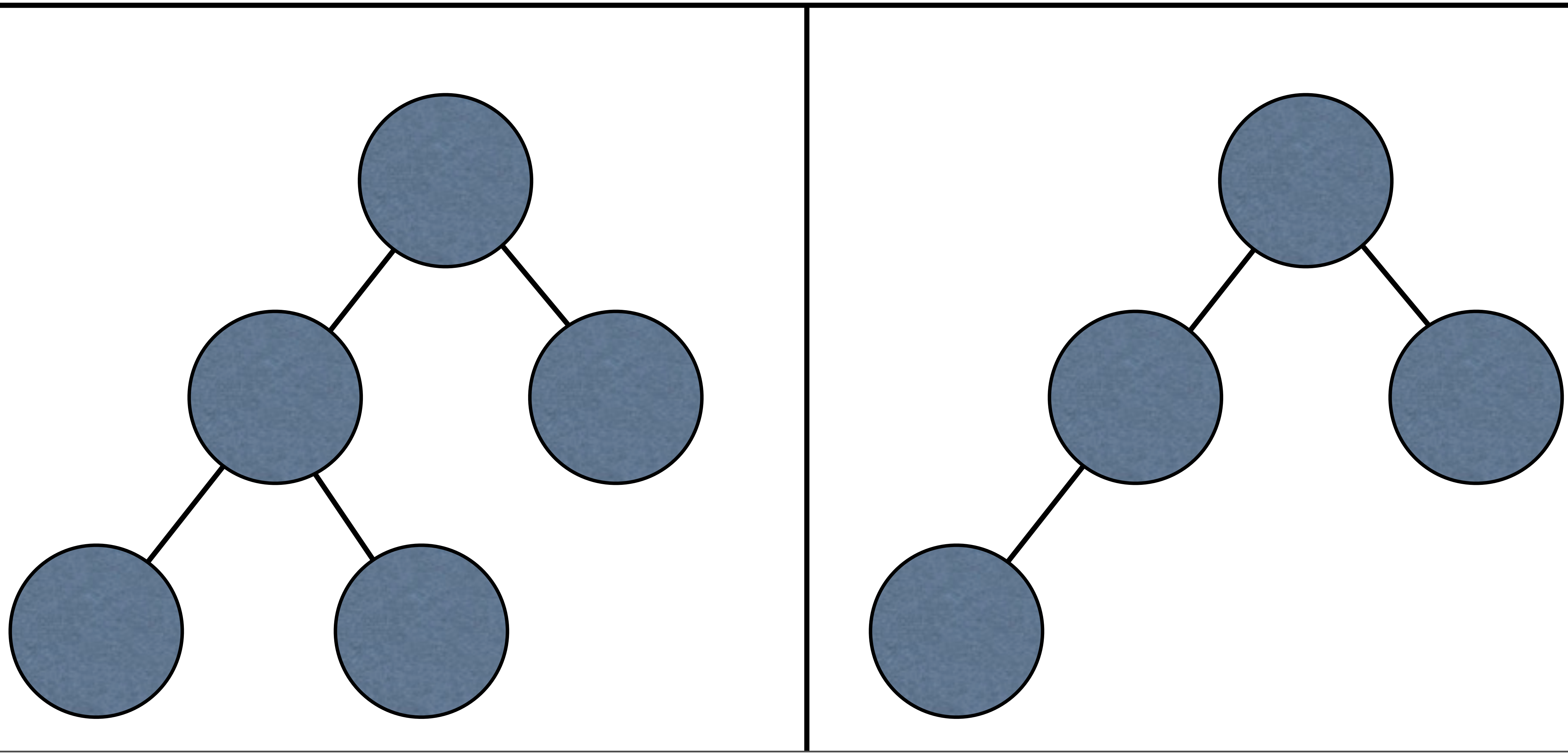
Counting

- Basic operation: how many nodes are in the tree?



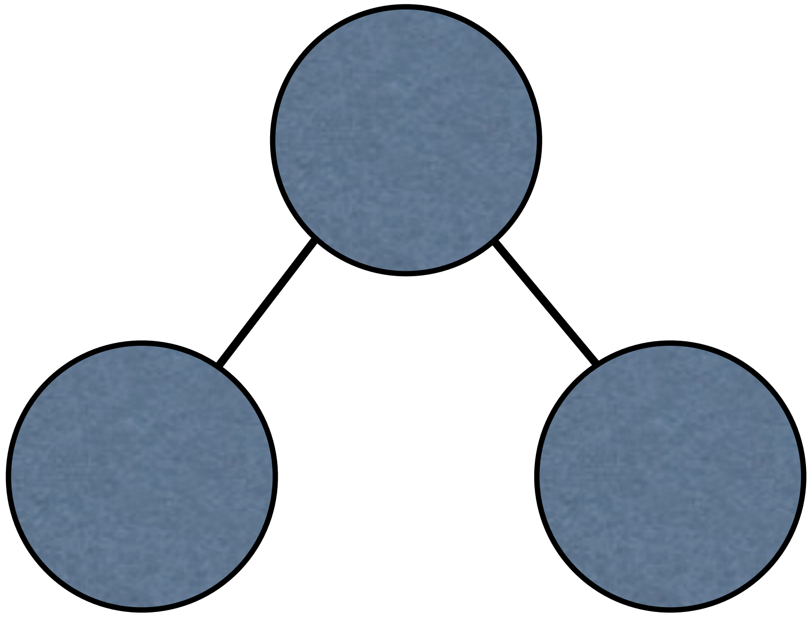
Depth

- Basic operation: how deep is the tree?



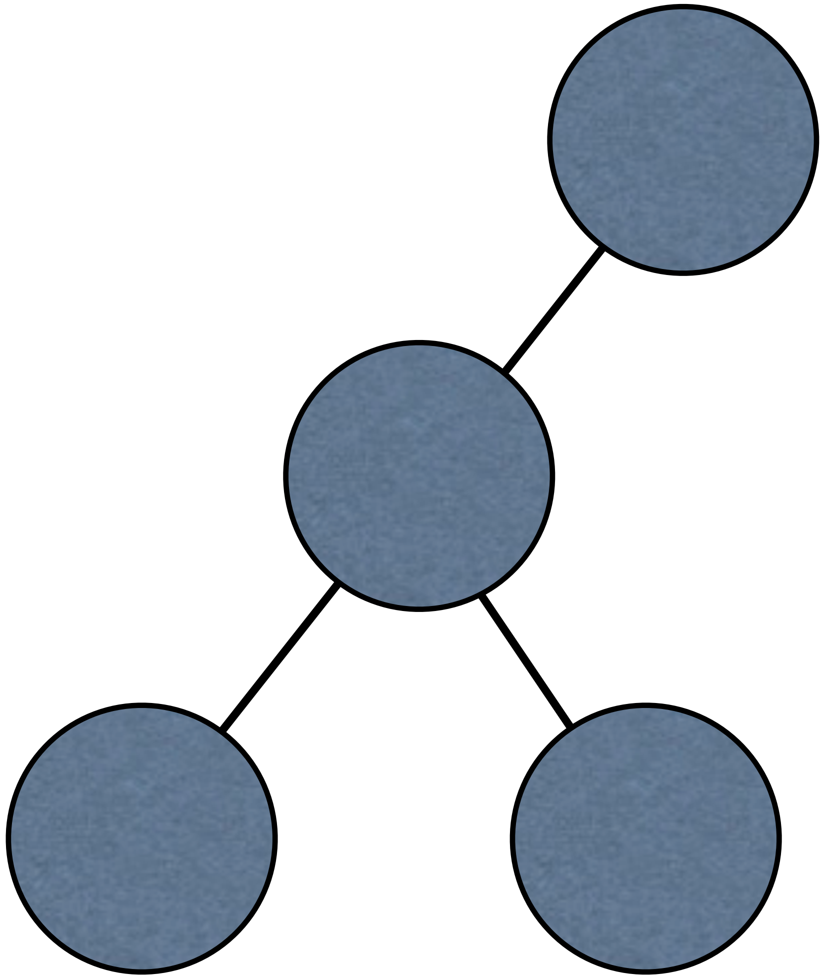
Representation

- Nodes: the circles
- Edges: things that connect the circles
- Nodes in a binary tree have at most two edges connecting to other nodes
- No cycles



Representation

- Each node has two edges
- An edge is either connected or it's not



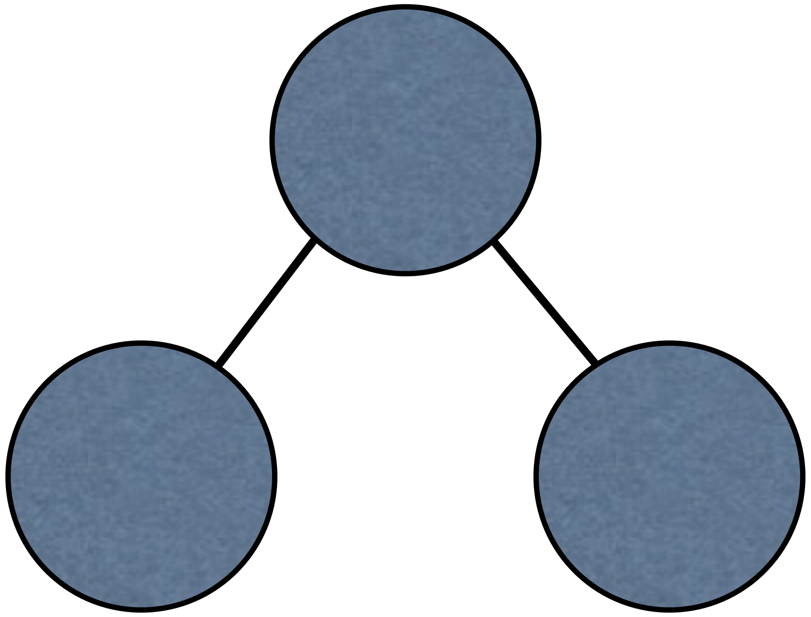
Code Representation

- Hint: nodes should be represented as `struct`s
- What would this definition look like?

Code Representation

- Hint: nodes should be represented as `struct`s
- What would this definition look like?

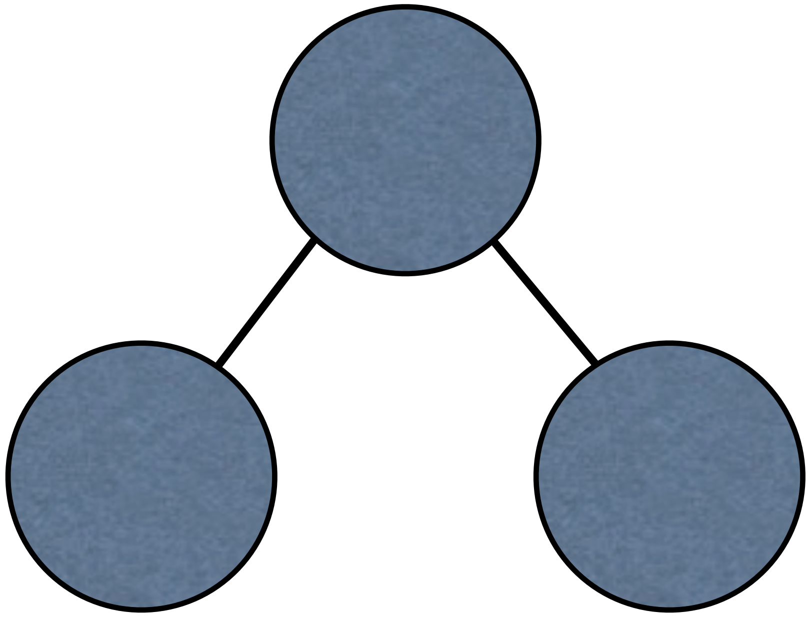
```
struct Node {
  struct Node* left;
  struct Node* right;
};
```



Code Representation

- Represent nodes as `struct Node`s
- If there is not a connection, use `NULL`

```
struct Node {
   struct Node* left;
   struct Node* right;
};
```



Recursion

- A `struct Node` holds pointers to other `struct Node`s
- A `struct Node` is defined in terms of itself!

Recursion

- In general, this means there is something defined in terms of itself
  - Can be data structures (like `struct`s)
  - Can be functions (a little later)
- Broken up into recursive cases and base cases

Base Case

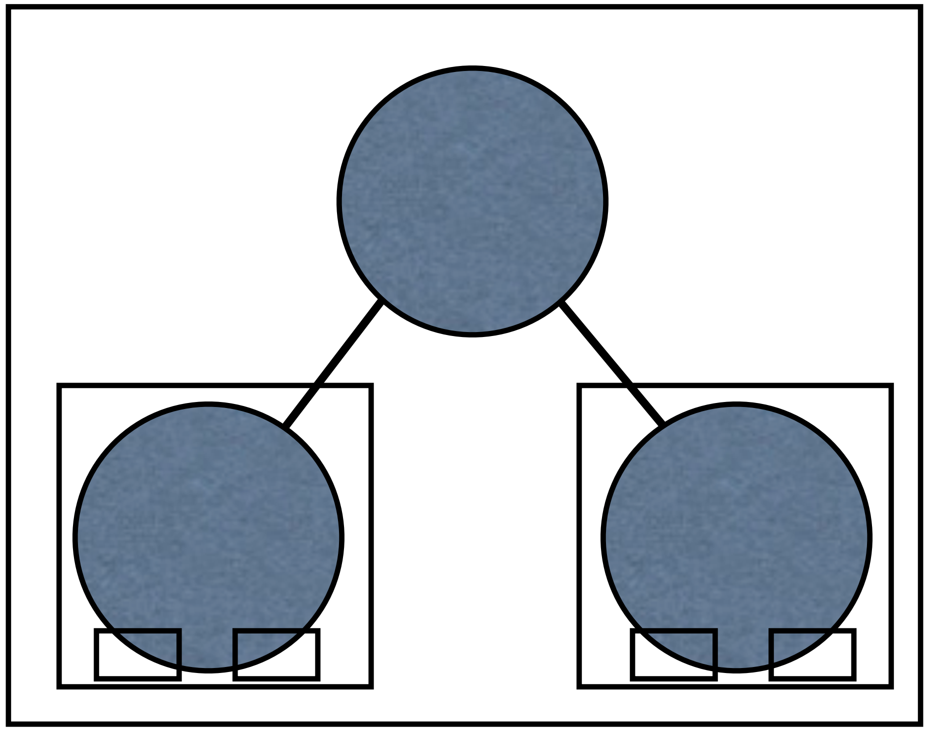
- Something not defined in terms of itself
- Act to end the recursion
- Can be multiple base cases
- For a `struct Node`, this means `NULL`

Recursive Case

- Case that is defined in terms of itself
- This is a `struct Node` that connects to another `struct Node`

Tree as a Whole

- How to represent this?
- Interesting note: there are subtrees



Tree as a Whole

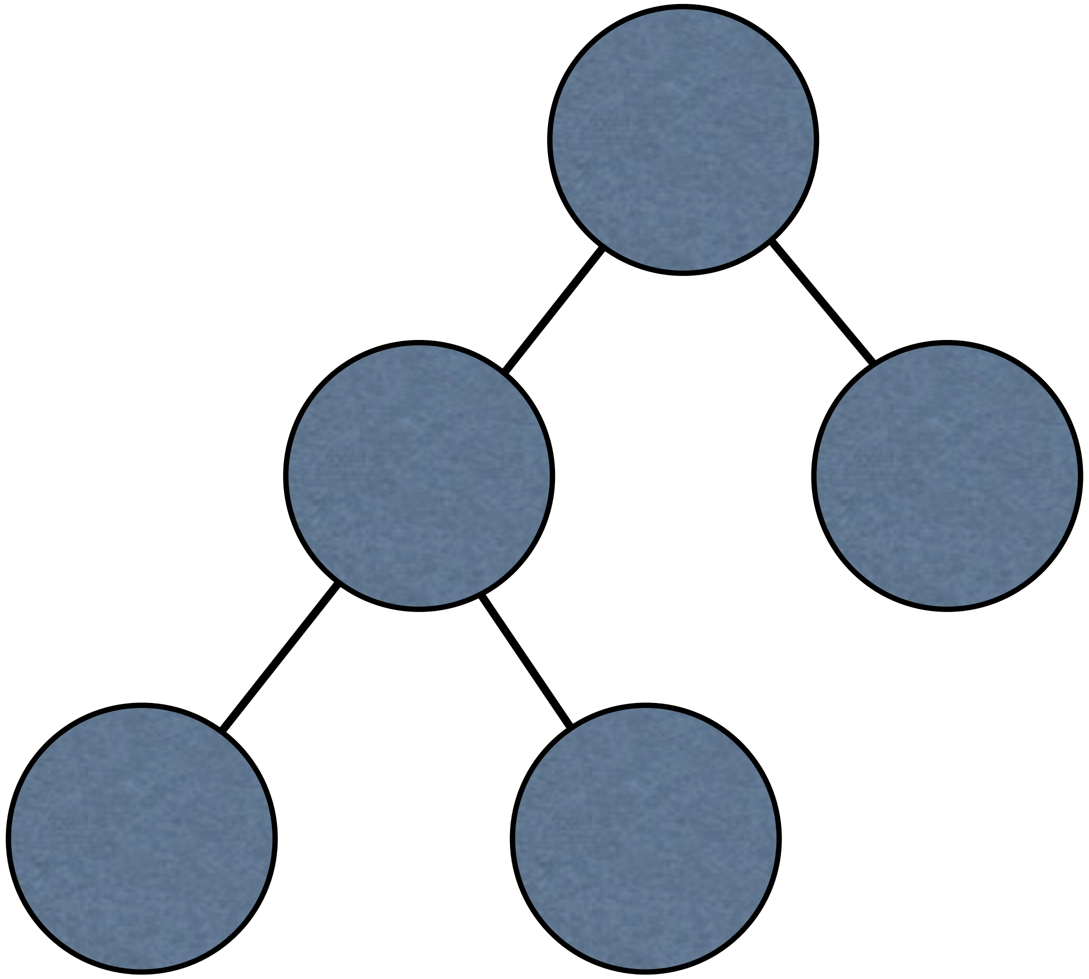
- Can simply use a `struct Node` without anything else
- This is a very flexible representation

Operations

- So keeping this representation in mind...

Counting

- Basic operation: how many nodes are in the tree?



Base Case

- A tree that is not there (i.e. `NULL`) has no nodes (i.e. `0` nodes)

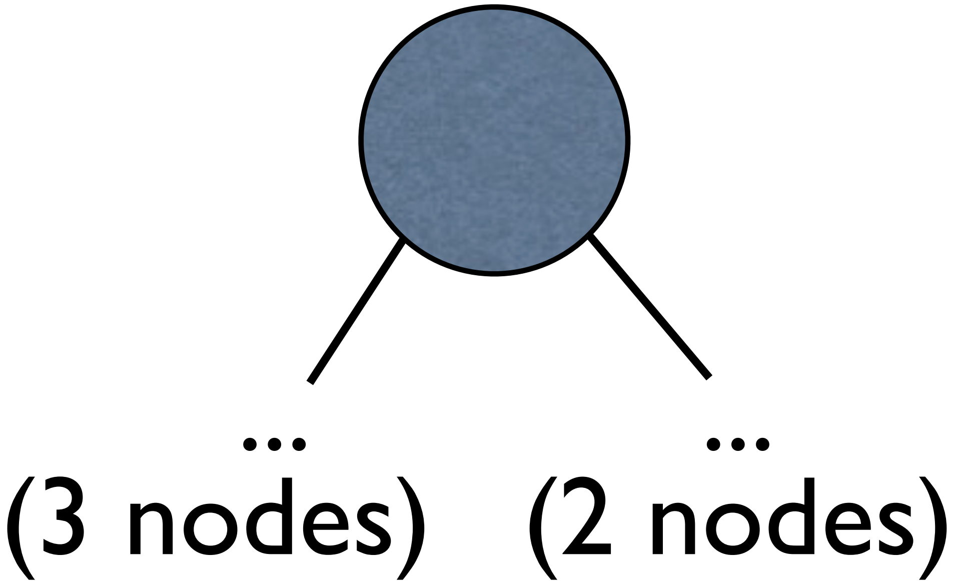
Base Case

- A tree that is not there (i.e. `NULL`) has no nodes (i.e. `0` nodes)

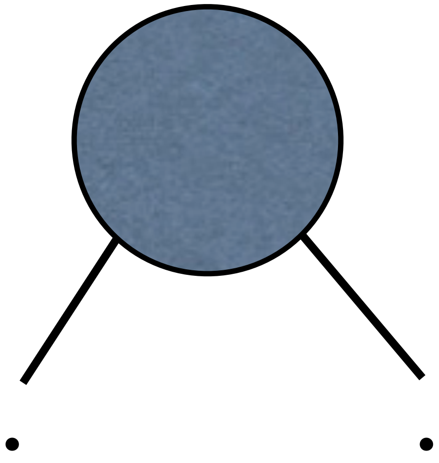
```
if ( node == NULL ) {
   return 0;
}
```

Recursive Case

- Given:
  - The number of nodes on the left
  - The number of nodes on the right
  - How many nodes are here?



Recursive Case



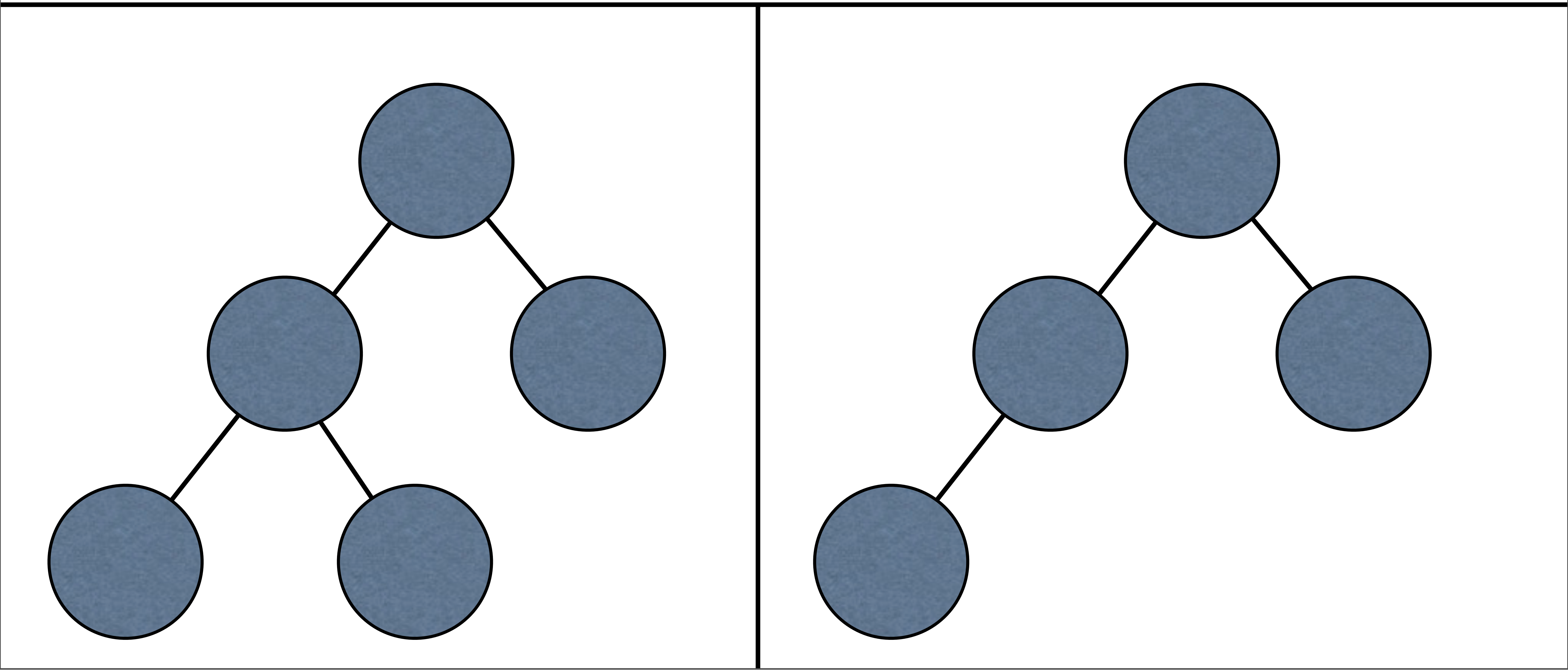
(3) (2) (current node)

```
nodesLeft + nodesRight + 1;
```

Full Code

Depth

- Basic operation: how deep is the tree?



Base Case

- A tree that is not there (i.e. `NULL`) has no depth (i.e. `0`)

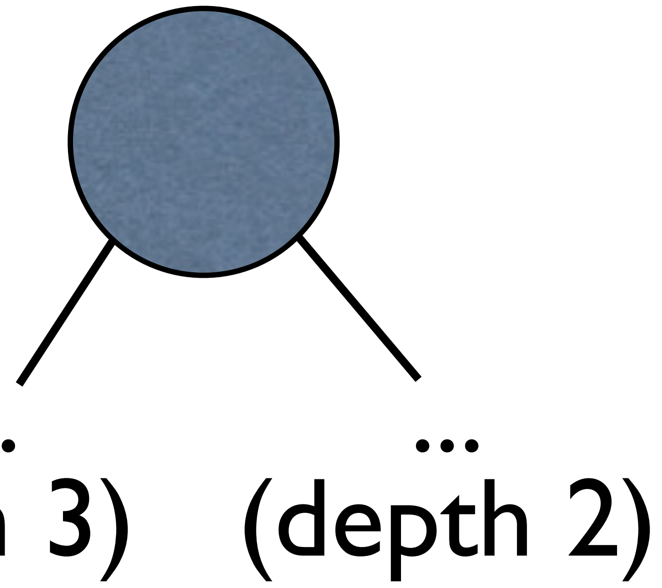
Base Case

- A tree that is not there (i.e. `NULL`) has no depth (i.e. `0`)

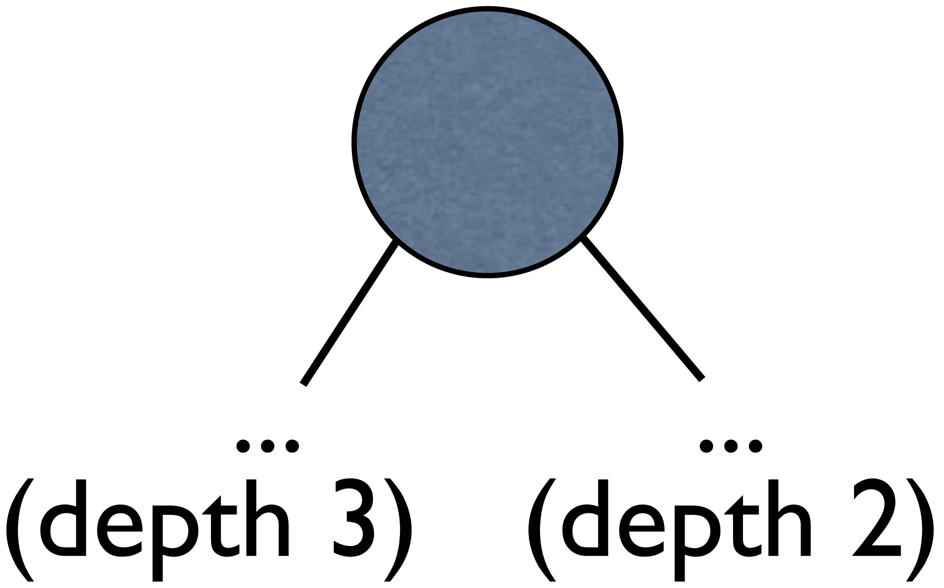
```
if ( node == NULL ) {
  return 0;
}
```

Recursive Case

- Given:
  - The depth of the tree on the left
  - The depth of the tree on the right
  - How deep is the tree?



Recursive Case



```
         (3)           (2)            (current node)
max( depthLeft, depthRight ) + 1
```

Full Code

Exam #2

Exam #2

- Exam is unintentionally cumulative
  - Still need to know how to use `if`, assignment, etc.
  - Will not focus on that material

Focus

- Functions
    - Prototype
    - Definition
    - Calls
- For all of these, what it is and how to do it

Focus

- Loops (`while`, `do/while`, `for`)
  - How to read them
  - How to write them
  - Be able to say what code does (i.e. the variable `x` is `5` after this code runs)

Focus

- Arrays
  - Initialize them
  - Index into them to get / set values
- “Given an array of length 10, find the first element that...”

Focus

- File I/O
  - Opening / closing
  - Reading / writing

Focus

- Types
  - Be able to identify the type of an expression
  - Just like last time, **except** now pointers are fully in the mix

Focus

- `Struct`s
  - You **will not** have to trace crazy pointer logic
  - You **will** need to know how to access them and set fields in them
  - Know what `->` does

Don’t Worry About

- Recursion
- `typedef`