Recursion and Trees

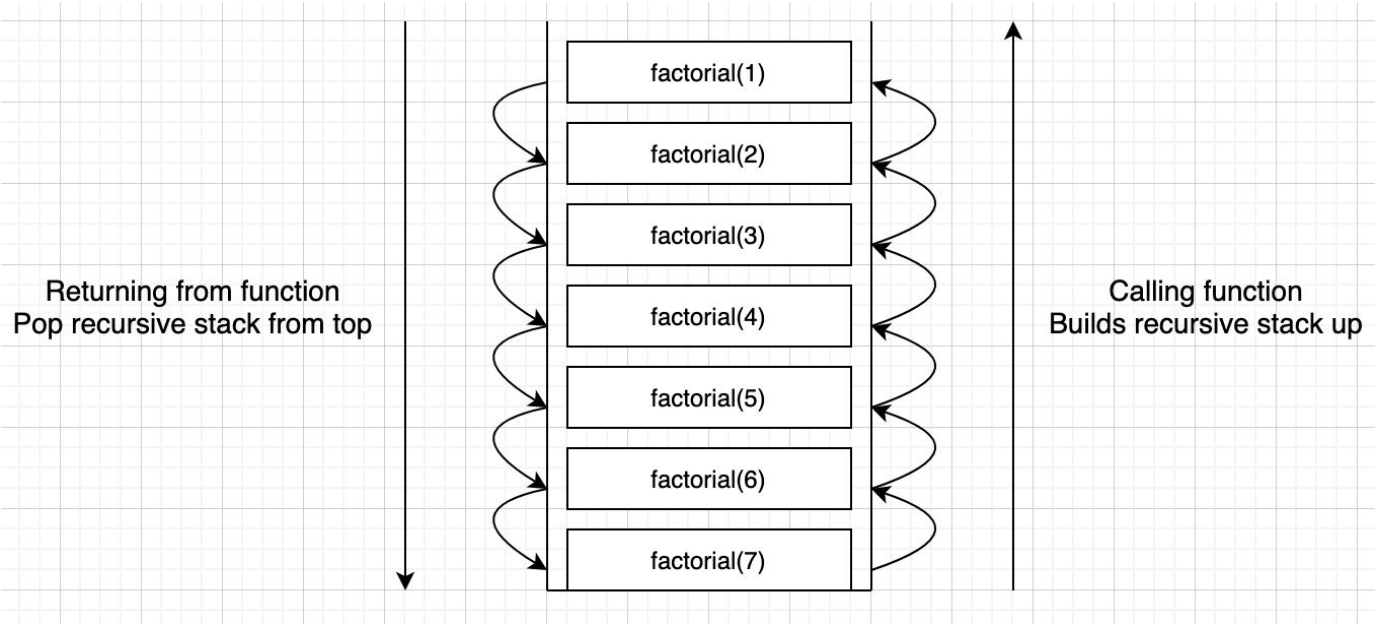
CSCI 2275

Recursion

- Recursion is when a function calls itself repeatedly to **break a large problem into a series of smaller problems**.
- It uses a **base case** (typically an if statement at the beginning of the function), to know when to stop
- You **break the problem into smaller problems** by changing the parameters on each recursive call
- Recursion can be awkward to think about for the first few problems but is a really important concept to get a grasp on

Recursion Example

- Consider finding the factorial of a number n -> n!
- First: Can you break it into smaller problems?
  - Yes, n! = n * (n-1)!
  - eg: 7! = 7 * 6!
- Second: Is there a basecase that all problems will lead to?
  - Yes, the smallest factorial possible is 0! which equals 1.
  - if(n == 0) return 1;


Returning from function
Pop recursive stack from top
factorial(1)
factorial(2)
factorial(3)
factorial(4)
factorial(5)
factorial(6)
factorial(7)
Calling function
Builds recursive stack up

How might this look in c++?

How might this look in c++?

Base case (line 15)

Recursive call (line 17)

- Notice the change in parameters

```
14  int factorial(int n){
15      if(n == 0) return 1;
16
17      return (n * factorial(n-1));
18  }
19
20  int main(int argc, char *argv[]){
21      cout << factorial(10) << endl;
22  }
```

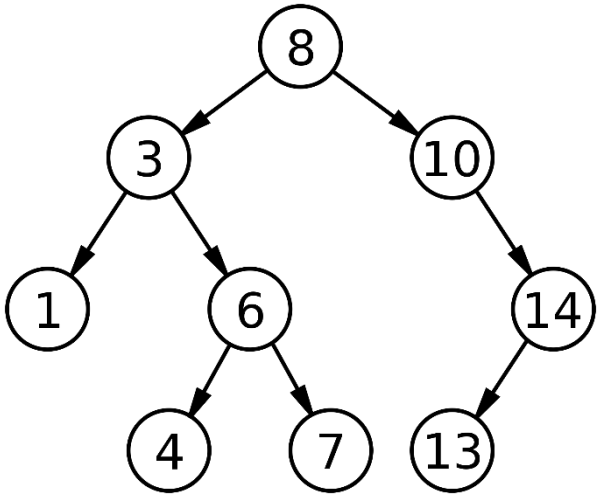
And that brings us to... Trees!

Trees

- Trees are a way you can store data
- Binary search trees are a way you can store data in a specific order to be able to access it much faster with a time complexity of O(log *n*)
- All non-empty trees will have a root node
- All subsets of Trees – Child nodes of trees can be their own smaller trees
- Leaf nodes are the ones which have no children
- All nodes will have at most 1 node pointing to it (parent node). Root will not have a parent

Binary Search Tree Example

1. Each node has exactly one key and the keys in the tree are distinct.
2. The keys in the left subtree are smaller than the key in the root.
3. The keys in the right subtree are larger than the key in the root.
4. The left and right subtrees are also binary search trees.



Insert - Coding example

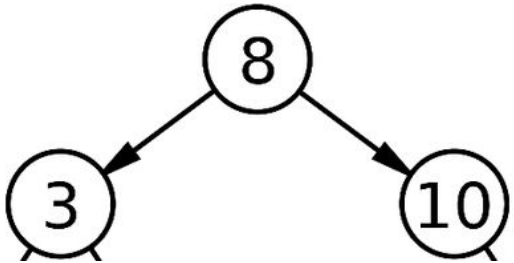
Search - Coding example

Different Types of Traversal

1. In order traversal – Left – Root – Right
2. Pre order traversal – Root – Left – Right
3. Post order traversal – Left – Right- Root

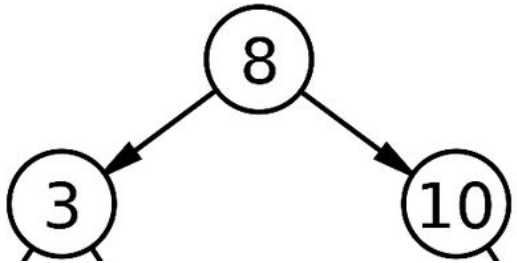
Consider this small example to remember:

- In-order would print: ??
- Pre-order would print: ??
- Post-order would print: ??



Different Types of Traversal

1. In order traversal – Left – Root – Right
2. Pre order traversal – Root – Left – Right
3. Post order traversal – Left – Right- Root



Consider this small example to remember:

- In order would print: 3,8,10
- Pre-order would print: 8,3,10
  - The root is visited *before* visiting its children
- Post-order would print: 3,10, 8
  - The root is visited *after* visiting its children

Exercise

Implement the (recursive) insert, search and In order traversal functions for a Binary search tree.

Perform

1. Insert 5
2. Insert 2
3. Insert 7
4. Insert 4
5. Insert 6
6. Search 2
7. Insert 1
8. Print the in order traversal