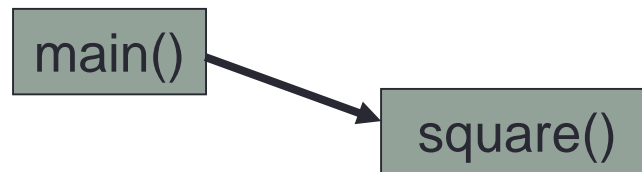


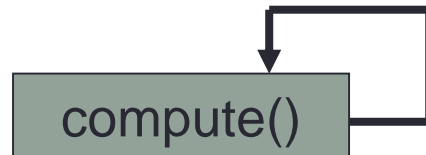
RECURSION

Recursive Functions

- All function calls that we have seen so far have been made by other functions



- A recursive function is a function that calls itself



Recursion

- Some problems are more easily solved using recursion
- Tree functionality is more easily solved by recursion than by iteration

First Recursive Problem

```
void count (int index)
{
    printf ("%d", index);
    if (index < 2)
    {
        count (index + 1);
    }
}

int main ()
{
    count (0);
    return 0;
}
```

What is the output?

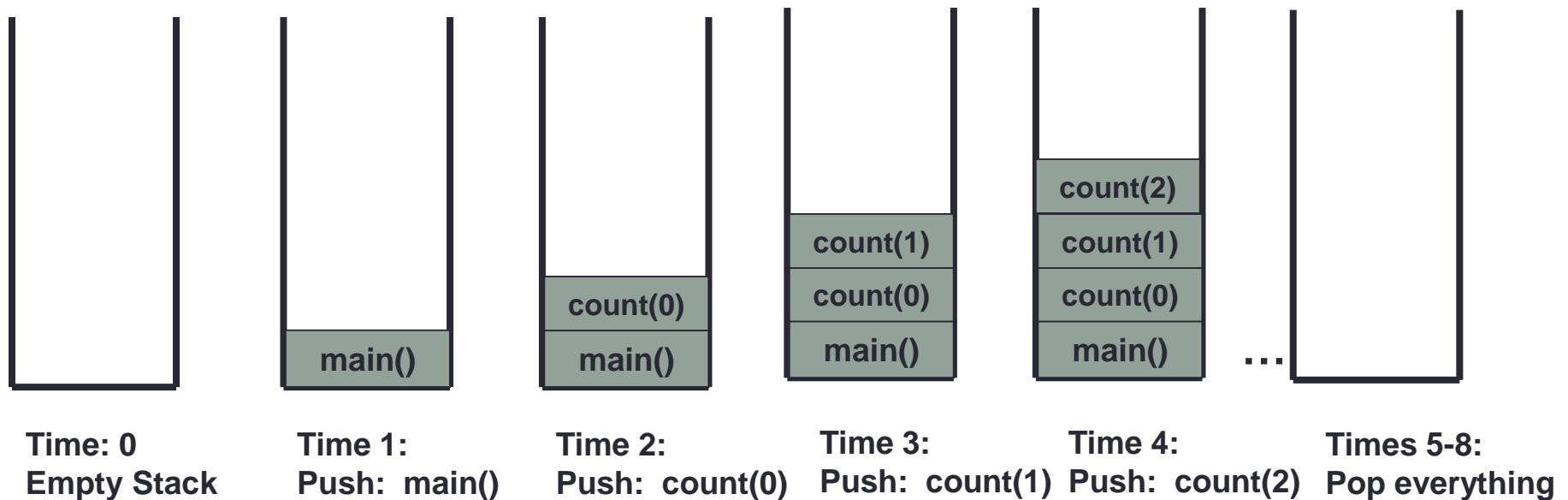
Visualizing Recursion

- To understand how recursion works, it helps to visualize what is going on
- We will do this using *Activation Records (stack frames)* and the *Call Stack (runtime stack)*
- Each time a function is called, an activation record is created and pushed on to the top of the stack
- When the function returns, the activation record is popped off the stack

Recursion and the Call Stack

- When a method calls itself recursively, you just push another copy of the function on to the top of the stack

Recursion and Call Stacks



Yes, `printf ()` generates an AR each time `count ()` is called

What is the Output?

```
void count (int index)
{
    printf ("%d", index);
    if (index < 2)
    {
        count (index + 1);
    }
}
```

```
int main ()
{
    count (3);
    return 0;
}
```


Recursion and Factorials

- Computing factorials are a classic problem for examining recursion.

- A factorial is defined as follows:

$$n! = n * (n-1) * (n-2) \dots * 1;$$

- For example:

$$1! = 1 \text{ (Base Case)}$$

$$2! = 2 * 1 = 2$$

$$3! = 3 * 2 * 1 = 6$$

$$4! = 4 * 3 * 2 * 1 = 24$$

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

Recursion and Factorials

- First step is to frame the problem in terms of itself. You do this by finding a pattern
- Once you see the pattern, you can apply this pattern to create a recursive solution to the problem
- Divide a problem up into:
 - What it can do (usually a base case)
 - What it cannot do
 - What it cannot do resembles original problem
 - The function launches a new copy of itself (recursion step) to solve what it cannot do

Recursive Factorial Solution

```
int main ()
{
    int i;

    for (i = 1; i <= 10; ++i)
    {
        printf ("%d!: %d\n", i, factorial (i));
    }
    return 0;
}
```

Recursive Binary Search

- In class we wrote an iterative binary search. Can you make that recursive?