Recursion

Calling functions from other functions

- Functions can be called from anywhere, as long as the parameters are valid
  - From main()
  - From other functions...

Calling functions from other functions

- Functions can be called from anywhere, as long as the parameters are valid
  - From main()
  - From other functions...
  - **AND FROM THEMSELVES**

Back up: Loop review

- Loops have three parts:
  - *Initialization* (setting a counter variable to 0)
  - *Update* (incrementing the counter)
  - *Terminating Condition* (how the loop actually stops)


```
for (i = 0; i < 5; i++) {
}
```

Recursion & Loops

- Recursion works in a similar manner

- The function calls itself, with *updated parameters*
  - This allows the function to behave differently when it is called multiple times

Recursion

- Counter program
  - Prints the numbers 0 - 4

```
for (i = 0; i < 5; i++) {
	printf(“%d\n”, i);
}
```

Recursion

- Counter program
  - Prints the numbers 0 - 4

```
void counter(int num); //Make a function
int main(void) {
    counter(1); //Call the function in main, with the initial condition
}
```

Recursion

- Counter program
  - Prints the numbers 0 - 4

```
void counter (int num) {
    if (num == 5){ //Terminating condition
        return;
    } printf("%d\n");
    return counter(num + 1); //returns a call to the counter function
again, with an update
}
```

Recursion - Behind the Scenes

- Recursion creates a *recursion stack*
- Each function call creates a new level in this stack
- Call 1
  - Call 2
    - Call 3
      - Call 4
        - Call 5
          - Etc...

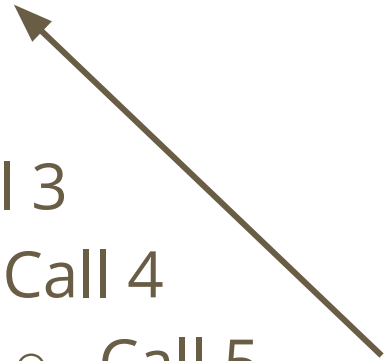
Recursion - Behind the Scenes

- Recursion creates a *recursion stack*
- Each function call creates a new level in this stack
- Call 1
  - Call 2
    - Call 3
      - Call 4
        - Call 5
          - Etc…

- This stack keeps building in the computer's memory…
- UNTIL a return statement which DOES NOT involve an update is called

Recursion - Behind the Scenes

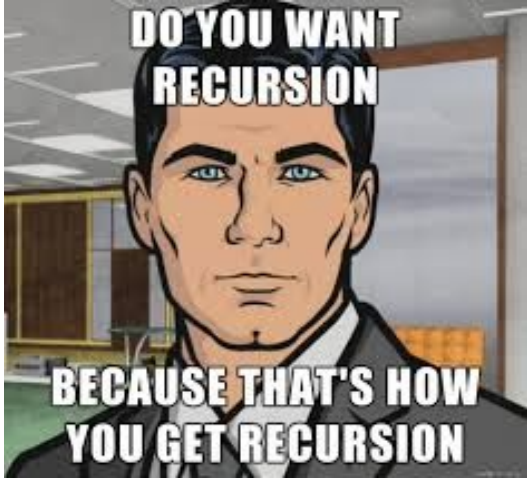
- Recursion creates a *recursion stack*
- Each function call creates a new level in this stack
- Call 1
  - Call 2
    - Call 3
      - Call 4
        - Call 5
          - Etc...



- At that point, the return statements propagate back up the recursion stack

Multiplication Exercise

```
int multiply(int num);
int main(void) {
    printf("%d\n", multiply(2) ); //starts with 2
}
int multiply(int num) {
    if (num > 1000) { //terminating condition
        return num;
    }
    return multiply(num*2); //ALWAYS multiplies num by 2
}
```



Recursion Coding 1

- Countdown from 10 - 0
  - Must use recursion!

  - *Hints: the if statement should stop at less than 0*

Recursion Coding 2: Introduction

- Factorials
  - 5! = 5 * 4 * 3 * 2 * 1
  - 2! = 2 * 1

  - Product of every integer between a number and 1
  - Used in many forms of statistics (and therefore computer science)

Recursion Coding 2: Introduction

- Scan in a number

- Write recursive code to find the factorial of that number
  - *Hint:* Return statement will involve more than just the function (should be a multiplication step)