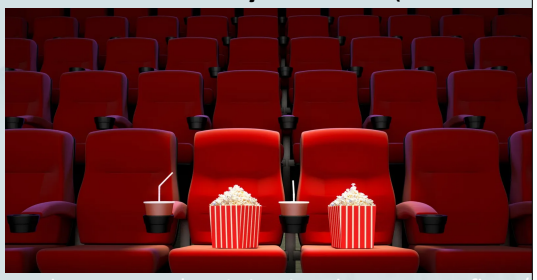
## Reminders/Announcements

- MC-06 due tomorrow!
- Exam-02 tomorrow–same time, same place
- No Recursive Backtracking on Exam-02

## DISCUSSION SESSION
## WEEK 9

# RECURSION

**How would you explain recursion to a 4-year-old?** (Interview question on Quora.com)



Someone in a movie theater asks you what row you're sitting in. There are no floor/seat numbers or labels. You don't want to count, so you ask the person in front of you what row they are sitting in, knowing that you will respond one greater than their answer. The person in front will ask the person in front of them. This will keep happening until word reaches the front row, and it is easy to respond: "I'm in row 1!" From there, the correct message (incremented by one each row) will eventually make its way back to the person who asked.

Recursion is an amazing problem-solving technique in which we solve a task by reducing it to smaller tasks (of the same kind).

- Recursion helps to reduce the number of lines of code and make it easier to read and write.
- Recursion may be direct or indirect.
  - ✓ Direct: func( ) calls func( )
  - ✓ Indirect: func( ) calls foo( ), then foo( ) calls func( )
- Recursion may unnecessarily perform repetitive steps.
- Anything that can be done iteratively can be done recursively, and vice versa.

**Note:** Iterative algorithms are generally more efficient than recursive algorithms.

**Recursive functions**

A recursive function is a function that calls itself at least once.

- A recursive function must generally have a base case (a limiting condition) to terminate the recursive process, otherwise you will have an infinite repetition that will eventually cause your code to crash with *segmentation fault*.
- In a recursive algorithm, each recursive function call must make progress towards the base case.

Fibonacci Series

**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …..**

Base Cases:

fib(0) = 0 //if n == 0, then return 0

fib(1) = 1 // if n == 1, then return 1

Recursive Case:

fib(n) = fib(n-1) + fib(n-2)

// if n > 1, return fib(n-1) + fib(n-2)

Fibonacci Series

**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …..**

```
int fib(int n) {
    if (n == 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}
```

**Sum of Digits Using Recursion**

```
int sumDigits(int n) {
    if(n == 0) {
        return 0;
    }
    return n % 10 + sumDigits(n / 10);
}
```

**Exercise (5 minutes): Sum of Digits in Integer**

1. Write a function **countDigits** that counts the digits of a number using recursion. For example, countDigits(7563) should return 4.

2. Draw the recursive call tree for an initial call: countDigits(568)

**Exercise (5 minutes): Sum of Elements in Array**

```
int arr[ ] = {2, 4, 3, 6, 1};
```

1. Given the above array in main, write a function **sumArray** that sums up the elements of the array using recursion. *(Hint: think about how pointer arithmetic will help in the recursive call)*

2. Draw the recursive call tree for sumArray({2, 4, 3, 6, 1}).

Consider a pair of integers, (a, b). The following operations can be performed on (a, b) in any order, zero or more times.

- (a, b) ---> (a + b, b)
- (a, b) ---> (a, b + a)

Return a string ("Yes" or "No") that denotes whether or not (a, b) can be converted to (c, d) by performing the operation zero or more times.

```
std::string func(int a, int b, int c, int d);
```

**Example**

(a, b) = (1, 1)  (c, d) = (5, 2)

Perform the operation (1, 1 + 1) to get (1, 2), perform the operation (1 + 2, 2) to get (3, 2), and perform the operation (3 + 2, 2) to get (5, 2). Alternatively, the first operation could be (1 + 1, 1) to get (2, 1) and so on.