# Grade 7/8 Math Circles

## March 6/7/8/9, 2023

## Recursion and Stack ADTs - Problem Set

1. What are the two cases you need to consider when thinking about recursion? Briefly describe each case.

2. Given the following code, identify any functions, arguments, and variables. What is the final outcome of the function call `check(main(1, 2))` on line 14?

```
1  main(arg1, arg2) {
2      return "Hello World."
3  }
4
5  check(phrase) {
6      if(phrase="Hello World!") {
7          return 1
8      } else if(phrase="Hello World.") {
9          return 2
10     } else {
11         return 3
12     }
13 }
14
15 check(main(1, 2))
```

3. Suppose you are given a numerical grade called `grade`. A student has an A if their numerical grade is between 90-100, a B if their numerical grade is between 80-89, a C if their numerical grade is between 70-79, a D if their numerical grade is between 60-69, an E if their numerical grade is between 50-59, and F if their numerical grade is under 50. Numerical grades are only integers. Write an `if` statement that returns the letter grade.

   Hint: You can say `-1<=x<=1` to say that `x` is between -1 and 1 (inclusive).

4. Suppose we started with an empty stack named `stack` that can hold a maximum of **6 items**. Given the following code, draw a picture of the stack at the points (A), (B), and (C). When-

ever `top()`, `is_empty()`, or `is_full()` is called, write the return value with its line number (ex: Write 'Line 5: "Circles"' if `stack.top()` returns "Circles").

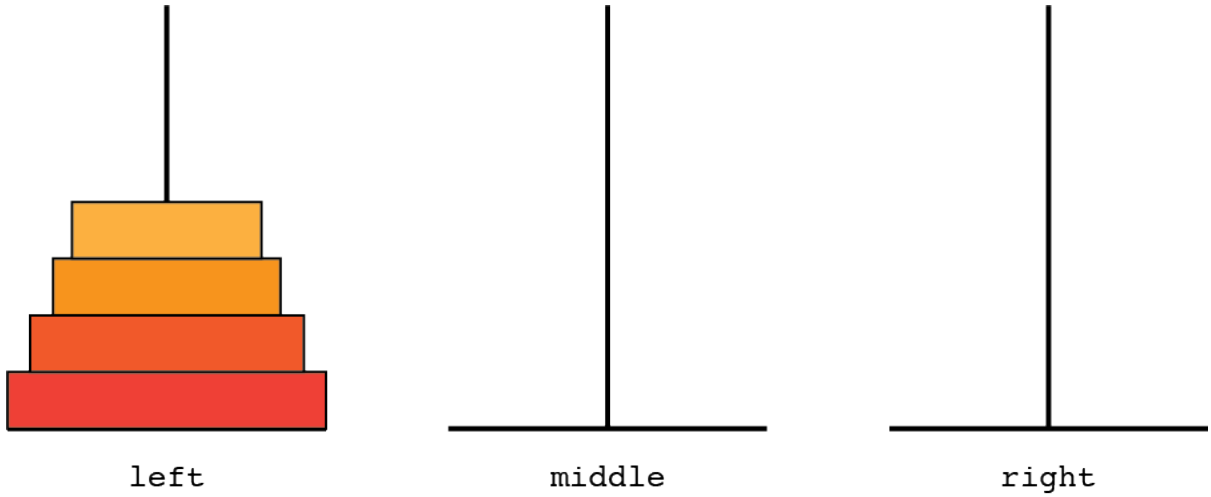
*Note that the `not` in line 18 turns `true` into `false` and `false` into `true`. In other words, it negates the value of `stack.is_full()`.*

```
1  stack.is_empty()
2  stack.pop()
3  stack.push("Math")
4  stack.push("Circles")
5  stack.top()
6  * (A) *
7  stack.push("Grade 7 and 8")
8  stack.push("Recursion")
9  stack.push("and")
10 stack.push("Stack")
11 stack.push("ADTs")
12 stack.top()
13 * (B) *
14 if(stack.is_full()) {
15     stack.pop()
16     stack.pop()
17     stack.push("I love Stacks!")
18 } else if(not stack.is_full()) {
19     stack.push("I hate Stacks!")
20 } else {
21     stack.pop()
22 }
23 * (C) *
```

5. The Tower of Hanoi is a classical problem where a stack of disks is stacked biggest at the bottom to smallest at the top. There are various versions of the problem with differing amounts of disks. The goal of the problem is to move all the disks from the left stack to the right stack with the same order of disks in the end. However, one restriction is that you cannot place larger disks on top of smaller disks. Otherwise, your tower would fall! Use stack operations to solve this problem.

We will call our stacks `left`, `middle`, and `right` respectively.

If you would like to visualize it, you can do so here. Make sure you change the number of disks to 4. You can also try the problem with more disks if you would like. If you want to learn more about the Tower of Hanoi, you can do so here.




6. Suppose you are given a `stack` that you don't know the size of. Write a function called `size` that takes a `stack` and a variable `count` as arguments and returns the size of the stack. Note that `count` will always start at 0, so we will always initially call the function like this: `size(stack, 0)`.

### Challenge Problem

8. Suppose you are given a `stack` and want to find an `item` and put it at the top of the stack. Suppose in addition that you are given a function called `push_to_stack` as defined below that takes two stacks, `stack` and `temp`, and pushes all the items on `temp` to `stack`. Write a function called `put_at_top` that takes two stacks, `stack` and `temp`, and a variable `item` and places `item` at the top of the stack without changing the order of the other items in the stack. If the `item` is not in the stack, then return "Error". `temp` is an empty stack, and `stack` is the stack that may or may not hold `item`.

```
1 push_to_stack(stack, temp) {
2     if(temp.is_empty()) {
3         return    * Return nothing to stop recursion *
4     } else {
5         stack.push(temp.pop())
6         push_to_stack(stack, temp)
7     }
8 }
```