Robbie Gibson-Practice Solutions **Name:** ____________________
Quiz 0

1. Angela loves Catalan numbers and their cool properties in combinatorics. Write a function to find the nth Catalan number. The Catalan numbers are expressed as:

$$\frac{(2n)!}{(n+1)!n!}$$

```
int factorial(int n);

int catalan(int n)
{
    return factorial(2 * n) / (factorial(n + 1) * factorial(n));
}

int factorial(int n)
{
    if (n < 1)
    {
        return 1;
    }

    return n * factorial(n - 1);
}
```

2. Below is a C program that sets a lot of variables. In the table, write the value of each variables after the given line has been executed. You should use &x and the like when writing the value of the pointers.

```c
int main(int argc, char** argv)
{
    int x = 2, y = 3, z = 4;
    int* a = &x;
    int* b = &y;
    int* c = &z;

    y += *a;
    z = y + *b;
    *a = x + *b;
    c = b;
    *b = *c;
    y = z + *c;

    return 0;
}
```

Line	x	y	z	a	b	c
6	2	3	4	&x	&y	&z
8	2	5	4	&x	&y	&z
9	2	5	10	&x	&y	&z
10	7	5	10	&x	&y	&z
11	7	5	10	&x	&y	&y
12	7	5	10	&x	&y	&y
13	7	15	10	&x	&y	&y

3. True or False:

    1. Malloc allocates memory on the stack.
    2. A string in C is an array.
    3. It's possible to sort an array in $O(n \log n)$ time.
    4. Angela creates and declares an `int` array, but doesn't initialize it. It's filled with 0's.
    5. Typing `next` in gdb will step into a function called on that line.
    6. The size of an `int` is 4 bytes.
    7. 4 bytes is 36 bits.

---

1. False. Malloc allocates its memory on the heap.
2. True. Strings can be thought of as arrays of `chars`. Pointer dereferencing and array indexing can be used equivalently.
3. True. Merge sort, for instance, will always run in $O(n \log n)$ time.

4. False. If an array isn't initialized, the values there are unknown; they could be anything. They are whatever values were there in memory before.
5. False. `next` will step over any function call. `step` will enter functions.
6. True. On a 32-bit architecture (like the CS50 appliance), `int`s are 4 bytes.
7. False. 1 byte is 8 bits, so 4 bytes is $4 * 8 = 32$ bits.

4. Angela writes some code to determine if a number is prime. Unfortunately, it's buggy and that makes her sad.

   How can you fix this implementation?

```c
#include <stdio.h>
#include <cs50.h>

int main(int argc, char** argv)
{
    printf("Enter value of N : ");
    int n = GetInt();

    int flag = 0;
    int i;
    for (i = 2; i <= (n / 2); )
    {
        if (n % i == 0) /* If true n is divisible by i */
        {
            flag = 0;
            break;
        }
    }

    if (flag)
    {
        printf("%d is prime\n", n);
    }
    else
    {
        printf("%d has %d as a factor\n", n, i);
    }
    return 0;
}
```

   There are two main problems here. First, in the `for` loop, `i` is never incremented. The loop will never end! The first thing to change is to add an `i++` in the update section of the `for` loop.

   The second thing is that flag never changes it is always 0. What you want is for the flag to start off at one value, and then if you find a divisor, change the flag and stop the loop. To fix this, initialize `flag` to 1, so the loop will continue until `flag` becomes 0.

5. How many different, unique values can you represent with 4 bits?

   Each bit can be one of 2 values, so 4 bits can represent $2^4 = 16$ different values.

6. Angela's arguing with Robbie about how to best sort an array. Angela thinks MergeSort is faster, but Robbie keeps saying Bubblesort is faster. Who's correct, what are the best-case and worst-case runtimes, and why are the two runtimes different?

   In general, Angela is correct, although not always. Merge sort has a best case and worst case runtime of $O(n \log n)$. Bubble sort a worst case runtime of $O(n^2)$ and a best case runtime of $O(n)$, so in the best case, it'll be faster than merge sort. However, this best case is fairly unlikely. It's that the array is already sorted! So merge sort is usually better.

   Bubble sort is slower because it has to iterate through almost the whole array many times. Merge sort uses a divide and conquer strategy similar to binary search, allowing it to do the whole thing faster.

7. Angela says a pointer is a variable whose value is the direct address of a memory location.

   1. Is there anything wrong with this definition? What's wrong with it?
   2. Declare a pointer.
   3. Why are pointers useful?

   1. No, there's nothing wrong.
   2. To declare a pointer, add a $\star$ character to the declaration, like: .

8. Write your own string compare function. The real `strcmp()` has complicated return values, but yours should `return 1` if the strings are equal and `return 0` otherwise. Use pointer arithmetic to check the string equality and not array indexing. Also do not use any other functions. For instance, you cannot use `strlen`.

```
int strcmp(char* str1, char* str2)
{
    char* test1 = str1;
    char* test2 = str2;

    for ( ;
         (*test1 != '\0') || (*test2 != '\0');
         test1++, test2++)
    {
        if (*test1 != *test2)
        {
            return 0;
        }
    }
    return 1;
}
```