CSCI 24000 – Fall 2017
Assignment #2 – Pointer Land
Due: 9/25/2017

This second assignment will build upon our knowledge of Pointers and how they allow us, the programmer, to directly access and manipulate memory in both C and C++. We will also explore the syntax of the **C++** language and how to create and use a **Makefile** for compilation. This will give you a taste of C++ in its most basic form – later this semester we will explore the full power of this Object-Oriented programing language and its many important concepts.

For this assignment, we are going to explore how we can use pointers, functions, and conditionals in C++ to create a simulation of the popular children's game Candy Land. In our "game" two players (you and a friend) will attempt to be the first to successfully reach King Kandy at the Candy Castle through a series of random chance movements. The goal is for you to be the first to reach the Castle and win the game. More on the game Candy Land can be found here: https://en.wikipedia.org/wiki/Candy_Land

For simplicity, let us assume that the following are the rules to our version of the game:

- The game is made up of 50 spaces – space #50 is the Candy Castle and thus the winning space.
- Each turn you will "draw" a card that will instruct you to either move forward one (1) space, two (2) spaces, or encounter a Special Spot (these spots can be both good and bad).
    - Special Spot Movements:
        - Mountain Pass: Move forward three (3) spaces.
        - Rainbow Trail: Move forward five (5) spaces.
        - Cherry Pitfalls: Move backward three (3) spaces.
        - Molasses Swamp: Move backward five (5) spaces.
- You and your friend **cannot** occupy the same space – if this occurs **you** must move backward one (1) space.
- The player that reaches the 50th space first is declared the winner.

There is one catch – your friend tends to get very lucky when drawing Special Spot cards from the deck. Despite this "advantage" you are determined to beat your friend to Candy Castle – can you do it!?

We will use the following table to outline the probability of "drawing" a respective card:

	Movement Type	**Probability**	**Action**
You	Move Forward 1	40%	Move Forward 1 Space
	Move Forward 2	20%	Move Forward 2 Spaces
	Mountain Pass	10%	Move Forward 3 Spaces
	Rainbow Trail	10%	Move Forward 5 Spaces
	Cherry Pitfalls	10%	Move Backward 3 Spaces
	Molasses Swamp	10%	Move Backward 5 Spaces
Your Friend	Move Forward 1	30%	Move Forward 1 Space
	Move Forward 2	10%	Move Forward 2 Spaces
	Mountain Pass	20%	Move Forward 3 Spaces
	Rainbow Trail	10%	Move Forward 5 Spaces
	Cherry Pitfalls	20%	Move Backward 3 Spaces
	Molasses Swamp	10%	Move Backward 5 Spaces

Your goal is to create a program to simulate this game. You will be using C++ to accomplish this task, and capitalize on the languages implementation of *pointers*, *control structures*, *functions*, and *libraries*. I will provide the skeleton (bare bones) structure of the program in the form of the file (**Candy.cpp**) on Canvas. You are to edit this file to accomplish the goal of the assignment. You **may not** add any additional methods and your program **must** make use of *pointers* and *pass-by-reference* to simulate the game.

Programming Notes:

- We will assume that you will both start at Space #0.
- You or your friend cannot move past the finish line (e.g., if you are on space 49 and draw a card that moves you forward 3 spaces you should not move to space 52 – instead you should just move to space 50).
    - Likewise, you may not move behind the start line (e.g., if you are on space 1 and draw a card that moves you backward 3 spaces you should not move to space -2 – instead you should just move back to the start space of 0).
- The first player that reaches Space #50 should trigger a message that notifies the players on who the winner is.
- You will always begin the game first.
- You will need to find some way of generating random numbers to determine the probability.
    - Guidance will be provided in lecture and lab as to how to achieve this.

Development Process:

For this assignment, all development must take place on the **master branch** in a **private** GitHub repository. It is strongly recommended that you commit and push often! You are also required to include the Honor Pledge & Digital Signature in your Candy.cpp file – this **MUST** be included at the top of your file.

Your output **MUST** be formatted in the same fashion as shown below:

```
Welcome to CSCI 240's Candy Land Game!

B   A                                                |
     B  A                                            |
   B  A                                              |
         BA                                          |
             AB                                      |
            B    A                                   |
               A B                                   |
                    A B                              |
                  A        B                        |
                      A  B                           |
                          A   B                      |
                        A    B                       |
                     A            B                  |
                   A              B                  |
                       A                B           |
                    A                       B        |
                 A                        B          |
                     A                          B    |
                         A                        B|

Your friend reached Candy Castle 1st! Better luck next time.
```

As with the previous assignment you will be submitting an algorithm. You should **always** write your algorithm before you begin actual coding. This is a good practice and demonstrates the importance of software design. Your algorithm **must** match your code/solution.

Submission:

All assignments must be submitted on IU GitHub (github.iu.edu). The name of your IU GitHub repository must be as follows: **csci24000_fall2017_A2**. You should be submitting the **Candy.cpp** file, a **Makefile** that we can then use to compile and subsequently run your program, as well as a file (e.g. **algorithm.txt**) outlining your algorithm for accomplishing this assignment. The names of these files **MUST** match that shown here.

There are NO Blackbelt challenges for this assignment.