Vincent Lee
Assignment 3 Reflection
4/18/2015

The end product of assignment three was a number guessing game between two players. Player 1 enters the number to be guessed, and player 2 enters guesses until they guess correctly. The program then prints how many guesses player 2 entered.

Using a top down approach, the mechanics of the program can be broken in to two categories. The first is reading in input from a player and the second is evaluating that input. I intended for my final program to be as streamlined as possible, and read as close to natural language as possible.

My understanding of the problem was clear from the start – anytime a programmer is asked to evaluate a test condition an unknown number of times, the tool to use is a standard while loop. Before the while loop, the program reads in the number to be guessed and the other player's first guess using standard stream objects to store those values in integer variables. I also needed to include a counter variable that increments after every guess. The program then uses a while loop to continue asking player 2 for guesses if their guess is incorrect. Once player 2 guesses correctly, the while loop test condition becomes false and the body is skipped. Execution then picks up with a final print statement, logging how many guesses player 2 entered to the console.

My final program design includes the function:

```
int promptPlayerForGuess(int & numberOfGuesses);
```

This function prompts player 2 for a guess, increments the guess counter, and returns the value of the guess to the current execution point. I decided to make a function because it made my program more readable and because my initial program had the same exact lines of code in two places in the body of the program. It is poor programming style to allow duplicate code to exist in your programs. If duplicate code does exist, it usually signals an opportunity to create a sub-task in the form of a function.

To test that my logic and program were correct, I decided to use typical corner cases that can break a program. I felt that testing negative input, 0, 1, and repeated numbers would cover the potential issues we could see in this program. I also tested guessing correctly on the first guess to ensure the counter variable correctly took this in to account. Although it is a manual process, I also rely heavily on print debugging to keep track of variable values to verify the user-entered data is flowing correctly through the program.

Overall, the strategy of step-wise refinement and creating sub-tasks is a powerful problem solving tool. I have read several programming books outside of CS 161, and worked my way through the free online courses that Stanford University offers. That background work has exposed me to different problem solving templates and the way programmers approach a problem.