AERE 361 Lab 4

Jordan Reese

January 2018

1 Integer Types

- 1. 1,854 should be stored as an int.
- 2. 23 should be stored as a char.
- 3. -34 should be stored as a char.
- 4. 10^6 should be stored as an int.
- 5. 2^{100} should be stored as a float.
- 6. The character 's' should be stored as a char.

2 Float Types

- 2.34 should be stored as a float
- 3.65728383 should be stored as a double
- 1.5E+23 should be stored as a float

3 Type Selection

The highest your sum can be in 200. The lowest your sum can be is 0. So unsigned char is the best data type to store the sum.

4 Type Errors

In the first sample the compiler does not report any errors. There is an error, unsigned char cannot store 690. You can change this by changing the unsigned char before the sum to an unsigned int. The output was 178, which is the remainder of 690/256.
In the second sample the compiler again doesn't report any errors. There is an error. Since C cannot math two different data types together, it has to

promote one to the highest storing data type. In this situation, float is way bigger than int, so the int get promoted to a float. Also in the division line, it was originally unsigned int div = a/b. But a/b is a decimal. The int command will just truncate the decimal and read zero instead of .1. to fix this just make the second unsigned int a float.

In sample 3 the compiler does not report errors unless you mess with the code. The code works properly, but at the end of our printed number we have what appears to be random trash. To fix this error we simply forget about those number and tell the printf state to just print 15 decimal points.

5 Big-O

Since the array is completely random number, we have to check every single value of the array in order to determine if one of them is 500. O(n) represents linear complexity, so we may need to run the loop "n" number of times.

6 Adding

The two adding algorithms are very different in how they compute the answer. They both calculate the correct answer, but the Gauss adding algorithm is much, much faster. This is because unlike the naive adding algorithm, the Gauss adding algorithm does not have to loop for every single value in the sequence. This isn't really noticeable for smaller numbers, but as you get up into the millions and billions your time to compute will double and triple. The Big-O complexity of the Gauss algorithm is 1, whereas the naive algorithm is linear. It will scale with the amount of inputs given to it.

7 Hello World

We needed to get the command line to print "Hello, World!". This was just a simple print statement with the declaration inside. I have done many of these programs, and didn't learn anything I didn't know already.

8 Functions!

Design a function to be called to print "Hello, World!"Creating functions is absolutely necessary when using C. We create a function to keep ourselves from constants copy and pasting, and for error diagnostics. All I did was create a function above main. Then I called that function verbatim in my main code. This will tell C to run whatever is in the function. I learned how to create basics function and how to call them correctly.

9 Is it???

This section made us figure out if our input was an integer or not. This exercise stumped me quite a bit. You need to use a function that had inputs passed into it and that had multiple end conditions. The function I create looped over the length of an input string and used isdigit to determine whether or not each term was a digit. I could then print error message on whether or not this test failed at all, and if it didn't the string passed and was deemed an integer. This was a linear complexity problem, since it only looped as many times as the string was long. I still don't know entirely what a pointer is and how it works, but I had to use one because the string I passed in was an array.

10 Looping Integers

This section was very easy if you created a good function in the previous exercise. All this problem had to do was loop 5 times, each time checking to see if the new input was a integer or not. I didn't learn much from the exercise, because the previous one covered most of the work need to complete this one. The complexity is roughly the same, but we have to factor in the fact that we can have 5 different inputs. So our complexity is 5 times greater the just the check on its on.

11 Character versus Integers

This code required us to input an integer and then get asked to input that number of characters. This code made us use the our previous function and a knowledge of data types. To understand what the program wanted as a character, one must know what can be stored as a character. Short answer is any single entity can be a character like "l", or ";" or "5". So we needed to make sure the string length of or inputs was 1 or we already failed the test. So in order to to this exercise correctly we had to have an adaptable loop that tested conditions of every input. This leads to very large complexity if the user enters large numbers, but we remain linear.