CSI 201
Functions - Making a Neat Main - Call by Reference

1. It'd be nice to have a main that is even more human readable than our normal code. Something like:

```
int main() {
  int a, b, c;
  getInputs(a,b);
  c = computeAverage(a,b);
  printResults(a,b,c);
}
```

2. This main is practically readable to any laymen and it is our use of proper function names and arrangements that allow this to work. We can write the last two functions right now, without a problem. However, the first function `getInputs` requires some more knowledge.

3. To do this, we need a new mechanism: call by reference. Aside from vectors, we have always passed parameters as "call by value." This means that the value of a parameter is passed. When we call `sqrt(42)` we pass the value 42 to the sqrt function. If we have a double variable called b and we call `sqrt(b)`. Well we're not really passing b to sqrt. But rather, we're passing the value of b.

4. To actually pass an object we must pass a reference to the object. One way to do that is with pointers (which we'll see later in the course), but C++ also has a direct mechanism: call by reference.

5. When writing our own functions, to make a parameter call by reference, we simply add an ampersand (&) after the type. Let's start with `getInputs` from above. The prototype is:

```
void getInputs(int &, int &);
```

6. The full definition:

```
void getInputs(int &f, int &g) {
   cin >> f >> g;
}
```

7. This function is different than usual. Because, normally when we pass a and b from main to this function, the function makes new buckets f and g which hold the values from the a and b from main. This is called scope. a and b have the local scope of main. f and g have the local scope of getInputs. In call by value, changing f and g (say by doing a cin) would have no affect on a and b. However, with call by reference when we pass a to the function as the first parameter, then f becomes a reference to a. This means f and a are really the same integer object. Changing f changes a. This can be powerful (and many programming languages actually do this implicitly).

8. For the most part, call by reference is generally used with a const like we saw in the last worksheet with vectors. There are bigger things at play there (avoiding copies of big objects). However, if you pass something as a const, you can't actually change it. So this would not allow us to write our getInputs function.

9. From this author's opinion, it is generally considered bad programming practice to write input functions like this with call by reference. But until we learn more about pointers, this is the only mechanism we have to do so. In style guides in big bodies of software, places like Google will forbid this kind of function in favor of the other referencing mechanism. Still, call by constant reference (like we saw with vectors) is perfectly accepted and used often.

10. Let's put some of this in practice: Write the functions that allow this main to work:

```
int main() {
   int a, b, c, d;
   getInputs(a,b,c); //get the values of a b and c from the user.
   d = findMax(a,b,c); //find the biggest value of a b and c
   prettyOutput(a,b,c,d); // Give some verbose useful output
}
```