**Basic guidelines for modularizing:**

1. Each function should do just 1 task.
   1. Communicating with the user is usually a task.
   2. Managing a set of tasks is often a task.
   3. Performing an algorithm is usually a task.
   4. If one task is so big that it contains several subtasks, then make 1 function to manage the task and others to do the subtasks.
   5. Don’t go crazy here. Not every line of code needs to be its own function.
2. If you copy and paste a block of statements, it should probably be a function. Stop and ask yourself why you are not making it one.
3. A function should fit on 1 screen. If it doesn’t, there had better be a good reason.
4. Learning how parameters and return types work is very important.
   1. The return type is the answer the function provides to the calling function.
   2. The parameter list is the data set that the function needs to receive from the calling function.
5. If you can’t copy the function and paste it into a different program, then its reusability is limited.
   1. Most user interaction code is usable only in the program where it’s written because user interaction is often very specific to the program.
   2. Most algorithmic solution code is transferable/reusable because solving an algorithm does not usually require user interaction and can therefore be made more generic.
6. Understanding variable scope is important for modularity.
   1. Information Hiding (a variable should only be visible to modules that need it) is the technical term for limiting the scope of variables so they can be more easily managed.
   2. Global variables are syntactically legal in C++, but not allowed because they violate the principle of Information Hiding.
   3. If only 1 function needs a variable, it should be local to that function only.
   4. Parameters are local variables, but they should only be used when a function needs external data.