

## Position Statement

The comments here are not necessarily a strong "position" that I hold, but rather are some observations I have made in teaching Computer Science 2 every semester for the past 8 years and an issue that I think we need to keep in mind when considering the first year of instruction. These comments are based in a CS program that has moved from a procedural programming approach in Pascal and C to a hybrid procedural/object-oriented programming approach in C++. There are many other issues that may be more important, but this is one that I have been thinking about/struggling with recently.

In the traditional procedural programming approach, functions (somewhat) naturally fall out of the problem decomposition. This gives a natural progression from smaller to larger problems, less to more functions, less to more complexity throughout CS 1 and 2. Of course, the art of designing a solution/decomposing the problem is not performed well by beginning students. But even when the program design is provided, in my experience students are initially reluctant (some continue to avoid it as long as possible) to use functions due mainly to the syntax of parameter passing. I also think part of the problem for students with using functions is the added abstraction. They no longer see all of the code. The discomfort of students with using code that is not of their own creation is evident in all levels of students, from CS 1 to a junior/senior level computer architecture course. They tend not to "trust" this code and problems with their programs are often first attributed to this other code.

Adding objects to this mix adds another level of complexity not only to the function calls themselves, an object has to invoke a function, but also to the level of abstract thinking required of the student, there is more "stuff" hidden from the programmer. Parameters are no longer the only way to get information to a function, the functions magically have access to the member variables. Of course, these issues have more to do with mechanics of the language than the even deeper issue of the shift in thinking required to program using the object oriented paradigm.

So, given the above brief observations, an issue to face during the first year of instruction is how to balance the student's use of predefined functions/objects and the student's implementation of functions/objects. For the latter, I am assuming that students are poor at problem decomposition/object design, so substantial portions of the design, whether procedural or OO, are given to the student. If the balance between using and implementing moves too far in either direction there are consequences. In the extreme of only implementing objects, the student mindset of not "trusting" code they have not created is perpetuated and they want to create everything from scratch. In the extreme of only using objects, the student runs the risk of suffering from what I call the *Wizard of Oz* phenomenon. Objects are lifted up as a mysterious entity that can solve many of our problems if we can just find the right object to use. At some point the student needs to have the curtain pulled back to remove the mystery and see that objects are just using techniques that the student already knows how to do.