# Prelab 3

Programming isn't just about writing a program for one-time use to solve some problem. Large pieces of software (e.g., Photoshop) are constructed by creating special-purpose functions that can be used as building blocks to create even more powerful functions.

A programming language like C provides the bare minimum set of functions, e.g., input/output (I/O) functions like `scanf` and `printf` along with standard math and string functions. It doesn't need to provide much more than that because the real power of a general-purpose programming language is that it allows programmers to expand the language by creating libraries of new functions.

As I described in lecture, it's possible to create a set of functions to improve the way arrays can be used in C. Specifically, we can implement a function with the following prototype that can create an array of any datatype:

```
void * createArray(int, int)
```

where the first parameter is the length of the array (number of elements) and the second parameter is the size of each element, i.e., the datatype size. What makes this function an improvement over malloc is that it permits use of a function, getArraySize(void *), which returns the length of the array whenever needed. In other words, getArraySize, can be used with any array created using createArray. Here is its prototype:

```
int getArraySize(void *)
```

The reason for using void * is because we have no way of knowing the datatype of user's array, *and we have no need for knowing it*. Because void * represents a "generic" pointer (i.e., a pointer of any type can be assigned to it, and it can be assigned to any pointer variable of any type), the user can pass their array to it regardless of its datatype. As discussed in lecture, the magic of getArraySize comes from the fact that createArray secretly hides the size of the array in an integer space before the start of the array.

To free an array created by createArray, the user will have to call the following function:

```
void freeArray(void *)
```

In lecture I described the entire implementations of all three of the above functions, and in fact the code for them is in the slides. You're not expected to memorize something like that during lecture, but if you grasped the gist of the main concepts, you should be able to create your own implementations of them. Please try to implement and verify the three functions entirely on your own before looking back at the slides. If you really, truly, can't do it after giving it an honest effort, then read over the slides and give it another shot – *but don't try to memorize the code segments given in the slides*.