Prelab 4: Week of February 15th

The objective of this prelab is to create a set of functions that will allow a user to create an array of *any* data type. This can be achieved using a prototype like the following:

```
void * createArray(int n, int dataTypeSize); // ignore return type for now
```

where the user is expected to pass the number of elements in the array *and the size of each of those elements*. We need to know both pieces of information so that we can determine the amount of memory to request from malloc. A general-purpose function won't know the data type in advance and so must require the user to provide the data-type size as a parameter like the following, which creates an array of 1000 doubles:

```
double *array;
array = createArray(1000, sizeof(double));
```

Note that the pointer returned by CreateArray is being assigned to a double* pointer without a need for casting. Does that mean that the return type of CreateArray must be a pointer to double? No, the return type of `void`* in the prototype represents a generic pointer that can be assigned to a pointer of any type (and the reverse is also true). This is exactly what we need to create general-purpose (generic) functions that can return pointers objects of any desired data type. Just to really nail things down, a user could allocate an array of 1000 floats like the following:

```
float *array;
array = createArray(1000, sizeof(float));
```

This is just like the previous example except that the user wants an array of floats instead of an array of doubles, so she passed sizeof(float) instead of sizeof(double). A different way to think about all of this is that instead of asking the user for a single integer giving the total size of the requested array – e.g., as malloc requires – we're asking the user to provide the number of elements and the size of each element as separate parameters. We can then multiply them together (and add sizeof(int) to the result) and pass that size to malloc.

What you'll find is that the generic createArray, getArraySize, and freeArray functions will be almost as simple as the special-purpose functions associated with createIntArray and createDoubleArray and yet can do everything that those functions do. In other words, by thinking big – i.e., about the most general case – we can design a set of general-purpose functions instead of separate sets of special-purpose functions for every possible data type. In fact, we'll see later that createArray will even work for new data types that users may define in the future.

After completing this prelab you should have a strong understanding of pointers, pointer arithmetic, and memory allocation. More generally, you'll hopefully also get a feel for what that understanding can allow you to achieve.

- JKU