Prelab: Week of April 26

Implement both an iterative and a recursive version of binary search in which you assume the existence of a function:

```
int compare(void *a, void *b)
```

which returns a negative number if object `a` is less than object `b`, a number greater than zero if `a` is greater than `b`, and zero if the two objects are equal. This function would of course have to be implemented by the user because only the user would know what the objects are and how to define "less than" or any other inequality involving those objects. *Remember, you've used this before…*

If you assume the existence of the above comparison function (provided by the user) then your binary search algorithm can use it instead of inequality operators, e.g., "<" or ">=", which are only defined for numeric values. By having the user provide a comparison function you can implement a general-purpose binary search algorithm that can work with pointers to any data type. (This is similar to the way general-purpose library functions for sorting and searching are implemented.)

As an example, suppose you've implemented a function that has the following statement involving the comparison of integer or float values:

```
if (query < array[i]) {
      Do stuff...
}
```

you could replace it with the following for comparing pointers to arbitrary objects:

```
if (compare(query, array[i]) < 0) {
      Do stuff...
}
```

You should remember how the user can implement the `compare()` function, but technically it doesn't really matter for purposes of this prelab because all you need to know is that the function exists. In other words, you can work out the logic of binary search for use with floats and then replace the inequality statements with the assumed comparison function. However, for testing purposes you could implement your own comparison function, e.g., to permit searching on Employee social security numbers, by doing something similar to the following:

```
int compare(void *a, void *b) {
      Employee *queryEmp, *emp;
      queryEmp = a; // Equivalent to queryEmp = (Employee *) a;
      emp = b;
      if (queryEmp->ssn < emp->ssn) return -1;
      if (queryEmp->ssn > emp->ssn) return 1;
      return 0;
}
```

Note that the user is the only one who can implement something like this because she's the only one who knows that the void pointers are actually pointers to Employee structs and that the comparison needs to be made using the ssn member. The binary search algorithm doesn't know nor care about any of that because it just works with void pointers to generic objects and relies on the assumed comparison function for determining if one object is less than another object.

- JKU