Prelab 6: Week of March 8th

For this prelab you are to implement an array-based List ADT that supports the following functions:

```
/* This function returns an empty List object. The
   parameter specifies the maximum length of the list. */
List * initList(int)

/* This function inserts the object of the first parameter
   at the head of the list and returns an error code. */
int insertAtHead(void*, List*)

/* This function returns the object at the index location
   given by the first parameter. */
void * getAtIndex(int, List*)

/* This function returns the size limit of the array, i.e.,
   the value that was specified when the List was created. */
int getListSizeLimit(List*)

/* This function returns the number of objects in the list. */
int getListLength(List*)

/* This function clears the list (makes it empty) just like
   it was after initList was called. */
void clearList(List*)

/* This function frees all memory allocated for a list and so
   is not the same as clearList. */
void freeList(List*)
```

The implementation will *not* use linked lists. It will implement the List ADT using an array. That's why the initList function requires an integer specifying the maximum size/length of the list. All that initList does is fill in the members of a List struct, which includes an array of void pointers. What else will you need to store in that struct? You could have a member that contains the number of objects in the list, which would initially be zero, but you'll definitely need another integer member. Consider this:

```
typedef struct arrayListStruct {
      void **array;
      int arraySize, listSize;
} List;
```

What is this, you ask? Well, the first member is an array of pointers to user objects. It will have to be allocated based on the parameter to initList. We need to store the length of the array (max size of list), and that's what arraySize is for. The third member, listSize, stores the number of items currently in the list, so of course it should be initialized to zero in initList.

You might want to ignore the above suggested struct and try coming up with something like it entirely on your own. (*It's never a good idea to use something blindly without completely understanding it!*). For example, if you feel like you're not sure why those members are included and defined the way they are, then try creating your own "better" alternative. The process will make everything very clear.

A user can initialize a list as follows:

```
List * empList;
empList = initList(100);
```

And the user can now read 100 employee records and insert them into the list by doing something along the lines of the following:

```
int ec;
Employee *emp;

for (i=0, i<100, i++) {
    emp = malloc(sizeof(Employee));
    emp = readEmployeeFromFile(fp); // Read the employee struct
    insertAtHead(emp, empList, &ec); // put it at head of list
}
```

From these examples you can hopefully figure out how to implement and use the required functions.

- JKU