Prelab 6: October 7th (prep for E1)

For this prelab you are to implement a List ADT, using linked lists, that supports the following functions:

```
/* This function returns an empty List object, i.e., this must
   be called before operations are performed on the list. The
   parameter is a reference to an error code. 0 signifies the
   operation was performed correctly, 1 means there was
   insufficient memory available to initialize the list. */
List * initList(int*)

/* This function inserts the object of the first parameter
   at the head of the list. The last parameter is an error
   code (0 implies success, 1 implies insufficient memory).
   Returns pointer to updated list if there is no error;
   otherwise returns the given list without change.   */
List * insertAtHead(void*, List*, int*)

/* This function returns the object at the index location
   (starting at 1 for the head) of the first parameter. */
void * getAtIndex(int, List*)

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

/* This function frees all memory allocated for a list and
   returns NULL. */
List * freeList(List*)
```

Here's the struct you will use for your linked list nodes:

```
typedef struct listStruct {
      void *object;
      struct listStruct *next;
} List;
```

Note that a void pointer in each node will contain the address of an object provided by the user. We don't need to know what the object is, e.g., its data type, because our ADT doesn't actually make use of the objects – *it just stores them in a list for the user*.

Also note that there won't be an online lab on Friday the 7th because that's when Exam 1 will be held in the regular lecture auditorium in Middlebush. This lab is intended to prepare you both for E1 and subsequent prelabs/labs involving linked data structures.

A user can initialize a list as follows:

```
List * empList;
int ec;

empList = initList(&ec);
```

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

```
Employee *emp, *head=NULL;

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

emp = freeList(head);
```

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

- JKU