**Prelab 5**

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 int 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(int, List*, int*)

/* This function returns the int object at the index location
   (starting at 1 for the head) of the first parameter. */
int 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 {
      int object;
      struct listStruct *next;
} List;
```

The above struct and interface functions are specialized to only store integer values. This is purely to make your first experience with linked lists as simple as possible. Everything can be easily generalized to store objects of any type by replacing the int object with a void pointer in each node to 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*. That's the sense in with List ADT is said to be a *container* ADT.

The following page shows an example of a how a user might apply the above List ADT interface functions.

A user can initialize a list as follows:

```
List * head;
int ec;
head = initList(&ec);
```

And the user can now put 100 integers into the list by doing something along the lines of the following:

```
for (i=0, i<100, i++) {
    head=insertAtHead(i, head, &ec); // Put int i at head of list
}

// Write a loop to print the values in the list

emp = freeList(head);
```

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