CSE 101
Introduction to Data Structures and Algorithms
More on ADTs in C

Suppose you wish to implement an ADT in C. The particular ADT is unimportant, so let's just call it a "Blah". You should create the following files at minimum: Blah.c, Blah.h, BlahTest.c. Blah.c represents the 'inside' of the black box, Blah.h represents the interface, and BlahTest.c is a kind of dummy client module used to wring the bugs out of the Blah ADT.

File Blah.h will contain prototypes for all exported ADT operations. It will also contain the line.

```
typedef struct BlahObj* Blah;
```

This defines `Blah` to be a pointer to some struct called `BlahObj`. Blah.c will `#include` Blah.h, and will contain definitions of all exported functions, as well as definitions of private functions and structs as well. Blah.c will also contain the following typedef statement.

```
typedef struct BlahObj{
   // code that defines fields for the Blah ADT
} BlahObj;
```

A client module can then `#include` Blah.h giving it the ability to declare variables of type `Blah`, as well as functions that take `Blah` parameters. However, the client cannot dereference a `Blah`, since the object it points to is not defined in Blah.h. The ADT operations take `Blah` arguments, so the client does not need to (and is in fact unable to) directly access the struct these references point to. Therefore the client can interact with a Blah object only through the exported ADT operations. This is how information hiding is accomplished in C.

Blah.c also contains a constructor:

```
Blah newBlah(...){
   Blah B;
   // code that initializes B
   return(B);
}
```

and a destructor

```
void freeBlah(Blah* pB){
   if(pB!=NULL && *pB!=NULL){
      // free all heap memory associated with *pB
      free(*pB);
      *pB = NULL;
   }
}
```

Notice that the destructor is defined in a strange way. It's argument is not a `Blah`, but a pointer to a `Blah` (i.e. a pointer to a pointer to `BlahObj`.) Therefore the destructor is called by passing the address of a `Blah` reference. Each Blah object is explicitly created and destroyed as follows.

```
Blah B = newBlah(...);
// do something with B
freeBlah(&B);
```

Function `freeBlah()` must be defined in this way (i.e. taking a pointer to a `Blah` reference rather than a simple `Blah` reference) since it is the one ADT operation that changes the `Blah` reference itself (in addition to the `BlahObj` it points to) by setting it to NULL.

Recall that all ADT operations must check their own preconditions and exit with a useful error message when one of them is violated. The error message should state the module in which the error occurred (i.e. Blah), the operation in which it occurred, and exactly which precondition was violated. The purpose of this message is to provide diagnostic assistance to whoever is programming a client of the Blah ADT. In this course that person is of course you the student, but in a the real world, it may well be another programmer, so you must make the error message as helpful as possible.

In the C language however, each ADT operation has at least one precondition that should be checked before all others, namely that its main reference argument is not NULL. This check must come first since any attempt to dereference a NULL pointer will result in a segmentation fault.

```
void some_op(Blah B){
   if(B==NULL){
      printf("Blah Error: calling some_op on NULL Blah reference");
      exit(EXIT_FAILURE);
   }
   // check other preconditions and do stuff
}
```

Finally a word about our naming conventions in C. In some other programming classes you may have used names like `BlahHndl`, `BlahRef` or `BlahPtr` instead of just `Blah`. We have chosen this convention in order to parallel the Java language as closely as possible. Obviously one name is not intrinsically better than another, but for the sake of consistency, you are required to adhere to the naming conventions outlined in this document and the previous ADT handout. In particular the file names Blah.c, Blah.h, BlahTest.c; the function names `newBlah()`, `freeBlah()`, `printBlah()`; and the type names `BlahObj`, and `Blah` are not open to modification.