CMPS 101
Algorithms and Abstract Data Types
Summer 2006

Some Additional Remarks on ADTs and Modules in ANSI 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, BlahClient.c. The file Blah.h will contain prototypes for all exported functions which represent ADT operations. It will also contain the line.

```
typedef struct Blah*  BlahRef;
```

This defines `BlahRef` to be a pointer to some struct called `Blah`. The file Blah.c will #include Blah.h, and will contain definitions for all exported functions, and perhaps definitions for some private functions and structs. Blah.c will also contain the following typedef statement.

```
typedef struct Blah{
   /* code which defines fields for the Blah ADT */
} Blah;
```

A client module can then #include Blah.h giving it the ability to declare variables of type BlahRef, as well as functions which take BlahRef parameters. However, the client cannot dereference this pointer since the object it points to is not defined in Blah.h. The ADT operations take BlahRef arguments, so the client does not need to (and is in fact unable to) directly access the struct which 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.

File Blah.c also contains a constructor:

```
BlahRef newBlah(...){
   BlahRef B;
   /* code which initializes B */
   return(B);
}
```

and a destructor:

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

The destructor is called with the address of a BlahRef as follows:

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

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

As in java, all ADT operations must check their own preconditions and exit with a useful error message when one of them is violated. What should the message say? It 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 class that is of course you the student, but in a the real world that may well be another programmer, so you must make the error message as helpful as possible.

In C however there is one more item to check. Each ADT operation should check that the reference which is it's main argument is not NULL. This check should come before the checks of preconditions since any attempt to dereference a NULL reference will result in a segmentation fault.

```
void some_op(BlahRef B){
   if(B==NULL){
      printf("Blah Error: calling some_op on NULL BlahRef");
      exit(1);
   }
   /* check preconditions and do stuff */
}
```

The reason this was not necessary in java was because calling an instance method on a null reference variable automatically causes a NullPointerException to be thrown, which provides some error tracking to the programmer.

Finally a word about our naming conventions. In some other programming classes you may have used names like 'BlahHndl' or 'BlahPtr' instead of 'BlahRef'. Is one name better than the other? Of course not. However, 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, BlahClient.c; the function names newBlah, freeBlah, printBlah; and the type names BlahRef, and Blah are not open to modification. This is not unlike the situation programmers face in industry. Each company has some set of notational conventions and procedures which everyone follows so that everybody can easily read and maintain each other's work.