Introduction

In this project, you will implement the mechanism of Tombstones to catch and avoid dangling references. We will be implementing a version of the typical int data type using the tombstone method.

Your Task

You will implement the functions declared in the skeletal code given in TombInt.cpp and TombInt.h to provide the necessary functionality to do the following:

A.) Declare an IntTomb pointer

Much like a pointer in C/C++, when these are declared, they do not point to anything. What they point to needs to be allocated somewhere else. A declaration will look like a typical C/C++ declaration.

    TombInt aTomb;

B.) Tombstone creation and memory allocation:

There needs to be a way to allocate the memory of the variable and a tombstone which points to it. This function will create a tombstone and allocate some integer memory for the tombstone to point to. The form of a call for memory allocation will look like the following.

    MemAlloc(aTomb);

C.) Memory deletion:

The point of using the tombstone mechanism is that memory can be deleted without leaving other variables that point to the same place dangling. When the delete function is called, the tombstone that the variable points to is not deleted. The memory that the tombstone points to is deleted and the link between the variable and the tombstone is broken. The actual tombstone is not deleted so other
references are not broken. Deletion will take the following form.

   DeleteTombInt(aTomb);

D.) Various Operators

Assignment (=):

Assigning an integer value the variable

   aTombInt = 4;

Making one variable point to the same location as another

   oneTombInt = anotherTombInt;

NOTE: Only assignment by reference is necessary. Assignment by value is not required.

Addition (+):

Adding a TombInt and an integer

   aTombInt + 4;

Adding two TombInts together

   OneTombInt + anotherTombInt;

NOTE: The return type should be a normal integer, not a TombInt.

Comparison: Less Than (<) Only

Comparison between TombInt/int and TombInt/TombInt should both be implemented. The operator should return a bool;

E.) Error checking and handling

When memory is deleted, the memory pointed to by the tombstone is de-allocated, but the tombstone remains. Other variables that pointed to it are not left dangling. Therefore, there is some error checking required when accessing tombstones to make sure that they point to allocated memory. This point is
better illustrated in the following code example, which is illustrated in Figure 7.16 of the textbook:

```c
TombInt my_ptr, ptr2; //variable declaration

//Create a tombstone and Allocate the memory it points to
MemAlloc(my_ptr);

//Now let second point to the same tombstone as first
ptr2 = my_ptr;

//Now delete the memory block pointed to by the tombstone
//of second using our delete command
DeleteTombInt(my_ptr);
```

At this point, since my_ptr was deleted, there is no memory allocated for ptr2. Therefore, the following call should cause an error.

```c
someptr = ptr2 + 4;
```

However, if no memory is allocated for ptr2 and a value is assigned to it, the error is recoverable. New memory can be allocated for the value and no error message required.

```c
ptr2 = 4;
```

For un-recoverable errors there is supplied error handling function called ReportError(). This function takes a string in as an argument. The function outputs an error based on the passed in string, then halts the program. An example call is as follows:

```c
ReportError("found Null pointer");
```

**Notes**

This project is not intended to be coding intensive. Each function, other than main, should not be more than 10 or so lines of code. Possibly, you may want to add a couple convenience functions to make certain things simpler, but those also should be relatively small. You should not find yourself doing a lot of intensive coding.

**Files**

The 3 required files (TombInt.cpp, TombInt.h, pp5seed.cpp) along with some addition examples are
located in /comp/80/files/pp5

All work should be done in TombInt.cpp and TombInt.h. A file very similar to pp5seed.cpp will be used for grading. The file will only include TombInt.h and be compiled only with TombInt.cpp. No other files will be accepted.

Submitting

Assuming your code is in the current directory on one of the local Sun machines, you should submit by typing:
provide comp80 pp5 TombInt.h TombInt.cpp