Evil Hangman

This lab assumes that you have completed all steps involved in labs 1 through 5.

Switch to your HANGMAN directory: cd /Spring2020/COMP1020/HANGMAN

Clean up your directory by typing make clean

Add a new .h file to your directory called generic.h with the following definition in it.

```
#ifndef GENERIC_H
#define GENERIC_H

typedef void* Item;

#endif
```

Modify your my_string.h file to include generic.h and then change the declaration for your destroy function so that it matches the following:

```
void my_string_destroy(Item* pItem);
```

Change the definition in my_string.c to match the declaration and modify the code so that it still accomplishes the job. Your unit tests should still function though they will start giving a lot of warnings because the types don't match on the calls anymore. You can fix this by casting the address of the string handle you are passing in to the destroy function as an Item*. Fix this in all of your unit tests so that there are no warnings.

TA CHECKPOINT 1: Demonstrate to your TA that you can compile your unit tests with the new changes and that you have no warnings even with the –Wall flag.

The purpose of changing your destroy function is so that you can interface with the generic vector that we are finishing up in class. You of course will need one more function to make that interface work and that is your assignment function that will allow the vector to initialize copies of your string objects and do assignment with deep copies.

Add the following declaration to your my_string.h file and write the definition in your my_string.c file.

```
//Precondition: pLeft is the address of a MY_STRING handle
//   containing a valid MY_STRING object address OR NULL.
//   The value of Right must be the handle of a valid MY_STRING object
//Postcondition:  On Success pLeft will contain the address of a handle
//   to a valid MY_STRING object that is a deep copy of the object indicated
//   by Right.  If the value of the handle at the address indicated by
//   pLeft is originally NULL then the function will attempt to initialize
//   a new object that is a deep copy of the object indicated by Right,
//   otherwise the object indicated by the handle at the address pLeft will
//   attempt to resize to hold the data in Right.  On failure pLeft will be
//   left as NULL and any memory that may have been used by a potential
//   object indicated by pLeft will be returned to the freestore.
void my_string_assignment(Item* pLeft, Item Right);
```

Once your implementation is complete write a test program that follows the following pseudocode.

Create an array of MY_STRING handles with 100 elements.
Initialize each element of the array to NULL.
Use your init_c_string function to initialize the first element of the array to the string “COPY ME!”.
Write a for loop that uses your assignment function to copy the first string into every other element of the array.
Destroy every element of the array with a for loop calling destroy on each element but use string_insertion to print each element to the screen just before deleting it.

TA CHECKPOINT 2: Demonstrate to your TA that your program has no memory leaks using valgrind. Demonstrate that by commenting out the for loop that destroys your strings you can create a memory leak big enough to hold 100 copies of the “COPY ME!” string.