Evil Hangman

This lab assumes that you have completed all steps involved in labs 1 through 3.

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

Clean up your directory by typing make clean

Add the following declarations to your string.h header file.

```
//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: If successful, places the character item at the end of the
//   string and returns SUCCESS.  If the string does not have enough room and
//   cannot resize to accomodate the new character then the operation fails
//   and FAILURE is returned.  The resize operation will attempt to amortize
//   the cost of a resize by making the string capacity somewhat larger than
//   it was before (up to 2 times bigger).
Status my_string_push_back(MY_STRING hMy_string, char item);

//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: Removes the last character of a string in constant time.
//  Guaranteed to not cause a resize operation of the internal data.  Returns
//  SUCCESS on success and FAILURE if the string is empty.
Status my_string_pop_back(MY_STRING hMy_string);

//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition:  Returns the address of the character located at the given
//   index if the index is in bounds but otherwise returns NULL.  This address
//   is not usable as a c-string since the data is not NULL terminated and is
//   intended to just provide access to the element at that index.
char* my_string_at(MY_STRING hMy_string, int index);

//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: Returns the address of the first element of the string object
//   for use as a c-string.  The resulting c-string is guaranteed to be NULL
//   terminated and the memory is still maintained by the string object though
//   the NULL is not actually counted as part of the string (in size).
char* my_string_c_str(MY_STRING hMy_string);
```

Follow the example vector code we have been doing in class to code the implementation of the first three functions. The last function will turn out to be a helpful function for us but may need a bit more explanation. The idea is that many of the things we will want to do with our strings require c-style strings (c-strings) for their input values. Take opening a file as an example. We could have a file name sitting in one of our string objects but how do we pass that information to fopen to open the file? The answer is the my_string_c_str function. The my_string_c_str function will basically add a null terminator to the memory representation of the string and then return the address of the first character in the string. This essentially allows the internal character array to behave as a c-string. Please note that the existence of this function does not mean that we always have to store a NULL terminator. In fact, we typically do not store it at all, but if the user needs this feature then we can offer it to them by making sure that the trailing character of the string is NULL. This function will resize the string if needed to make room for the NULL terminator but will typically not double the strings size but rather just add enough room for the NULL. As stated in the post-condition though the NULL character is not considered to be part of the string object and actually will exist outside of the current size of the string. For example, if the string is "the" then the size will still be 3 but this function will make sure that the character after the 'e' is NULL in the internal character array.

Now add the following function declarations to your string.h file and your string object will be (nearly) complete.

```
//Precondition: hResult and hAppend are handles to valid My_string objects.
//Postcondition:  hResult is the handle of a string that contains the original
//   hResult object followed by the hAppend object concatenated together. This
//   function should guarantee no change to the hAppend object and return
//   SUCCESS if they operation is successful and FAILURE if the hResult object
//   is unable to accomodate the characters in the hAppend string perhaps
//   because of a failed resize operation.  On FAILURE, no change to either
//   string should be made.
Status my_string_concat(MY_STRING hResult, MY_STRING hAppend);

//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: Returns an enumerated type with value TRUE if the string
//   is empty and FALSE otherwise.
Boolean my_string_empty(MY_STRING hMy_string);
```

Complete the implementation of the above functions. You should have been testing as you were coding but now you need a test file to demonstrate that each of these functions is working as intended to your TA.

TA CHECKPOINT 1: Demonstrate to your TA that your functions behave as described. There is no sample main program this time. Test well and show that you do not have any memory leaks.

In next week's lab we will begin a procedure to formally test our code against a list of expected behaviors. Before you can begin that lab you need to have all of these functions in working order.