Building the string object: phase4

Lab details:

We will continue working with our string object. This lab assumes that you completed all of the previous phases to build the string object

Open your header file (string.h) and add the following declarations to your header file:

```
//Precondition: hString is the handle to a valid String_Ptr 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* string_at(String_Ptr hString, int index);

//Precondition: hString is the handle to a valid String_Ptr 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* string_c_str(String_Ptr hString);
```

Follow the example vector code we have been doing in class to code the implementation of the first function. Basically, the **string_at** function returns a pointer to the char at the given index location in the string. The function should return a **NULL Pointer** if the **index > size of the string.**

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 **string_c_str** function. The **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.

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.

What you will need to submit:

- string.c file holding the implementation of the above functions
- string.h file holder the functions declaration
- test.c holding the test program you created to test your code
- A Makefile to use to compiler your code
- A valgrind report showing that you do not have any memory leaks
- Combine all of the above files into a single compressed file. Name the compressed file: lab4<first-initals><LastName>
- Submit the compressed file using the submit command to your lab TA.