Building the string object: phase3

Learning outcomes:
- Opaque objects
- Functions

Lab details:

We will continue working with our string object. So far, we have been dealing with all the functions and **struct string** definition inside the header file. In this lab, you need to separate the definition of the struct from the definitions of the functions and save the **struct string** inside the implementation file. Basically, your header file will be the following:

```
#ifndef STRING_H_
#define STRING_H_

#include <stdio.h>
#include "status.h"

// A handle to the struct string
// The struct string definition is located in
// the implementation file
typedef struct string* String_Ptr;

//Precondition: None
//Postcndition: returns a handle to a valid string object
String_Ptr string_init_default();

//Precondition: c_str is a c string to use to initialize the string object
//Postcondition: a handle to the string object
String_Ptr string_init_c_string(char * c_str);

//Precondition: phString is a pointer to a handle to the string object
//Postcondition: Returns SUCCESS if we successfully freed
//the memory, FAILURE otherwise
Status string_destroy(String_Ptr* phString);

//Precondition: hString1 and hString2 are handles to two different
//string objects to compare between them.
//Postcondition: 0 both strings are equal
// < 0: the first character that does not match has a lower value in str1 than in str2
// >0: the first character that does not match has a higher value in str1 than in str2
int string_compare(String_Ptr hString1, String_Ptr  hString2);

//Precondition: hString is a handle to a string object
//Postcondition: the number of characters currently held by the string object
int string_get_size(String_Ptr hString);

//Precondition: hString is a handle to a string object
//Postcondition: the maximum number of characters the string object can hold
int string_get_capacity(String_Ptr hString);

//Precondition: hString is a handle to a string object, fp is a pointer
// to a file to read a string from
//Postcondition: SUCCESS if we successfully read a single string from the file,
// FAILURE if reached the EOF and string is still size zero
```

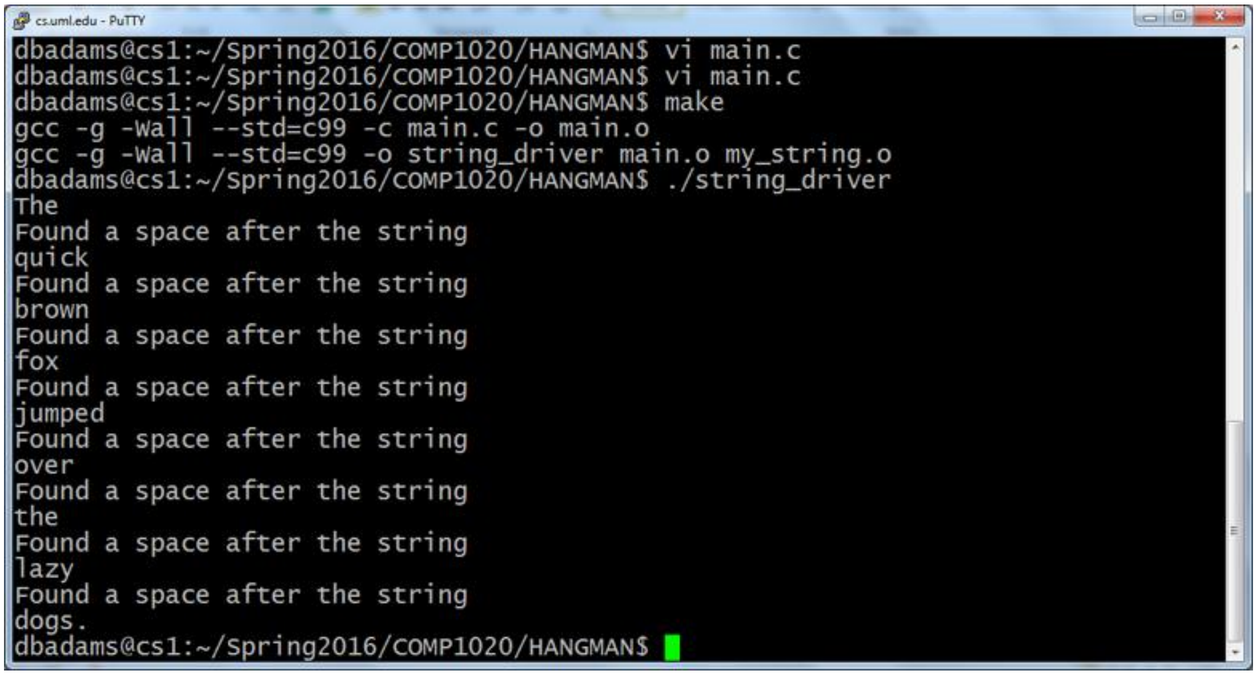
```
Status string_extraction(String_Ptr hString, FILE * fp);

//Precondition: hString is a handle to a string object, fp is a pointer to
// a file to write a tring to
//Postcondition: SUCCES if it successfully wrote into the FILE* and
// FAILURE if it failed to do so
Status string_insertion(String_Ptr hString, FILE * fp);



#endif  // STRING_H_
```

After you are done with reorganizing your header and implementation files, test that your code is working correctly by running the following driver program:

```
#include <stdio.h>
#include <stdlib.h>
#include "string.h"

//Simple.txt file includes the following sentence:
//The quick brown fox jumped over the lazy dogs.
//No spaces or newlines after that period
int main(int argc, char* argv[])
{
  String_Ptr hString = NULL;
  FILE* fp;

  hString = string_init_default();
  fp = fopen("simple.txt", "r");

  while(string_extraction(hString, fp))
    {
      string_insertion(hString, stdout);
      printf("\n");

      if(fgetc(fp)== ' ')
        {
          printf("Found a space after the string\n");
        }
    }

  string_destroy(&hString);
  fclose(fp);

  return 0;
}
```

Your output should look like the following:

TA CHECKPOINT 1: Demonstrate to your TA that your function behaves as above and have



```
dbadams@cs1:~/Spring2016/COMP1020/HANGMAN$ vi main.c
dbadams@cs1:~/Spring2016/COMP1020/HANGMAN$ vi main.c
dbadams@cs1:~/Spring2016/COMP1020/HANGMAN$ make
gcc -g -Wall --std=c99 -c main.c -o main.o
gcc -g -Wall --std=c99 -o string_driver main.o my_string.o
dbadams@cs1:~/Spring2016/COMP1020/HANGMAN$ ./string_driver
The
Found a space after the string
quick
Found a space after the string
brown
Found a space after the string
fox
Found a space after the string
jumped
Found a space after the string
over
Found a space after the string
the
Found a space after the string
lazy
Found a space after the string
dogs.
dbadams@cs1:~/Spring2016/COMP1020/HANGMAN$
```

them verify that there is only ONE space between your words in the file.

After you are done with the above, now it is time to add more functions to your string object. Add the following to functions to the above header file:

```
//Precondition: hString is a handle to a valid string object,
//item is a character to add to the end of the string
//Postcondition:
// return SUCCESS if the item is successfully
// added to the end of the string.
// The end of the string refers to the current size of the string.
// If the string reached its maximum capacity, i.e., size == capacity,
// then the string needed to be resized to be able to add the new item
// Resize the string by doubling its capacity
// Return FAILURE if the string could not be resized
Status string_push_back(String_Ptr hString, char item);

//Precondition: hString is a handle to a valid string object
//Postcondition: Removes the last character in the string.
// Returns SUCCESS if the last character was successfully removed
// and FAILURE if the string was empty
Status string_pop_back(String_Ptr hString);
```

Afterwards, test your code using the following driver program:

```
#include <stdlib.h>
#include <stdio.h>
#include "string.h"

int main(int argc, char* argv[])
{
  String_Ptr hString = NULL;

  hString = string_init_c_string("Hello World!");
  string_extraction(hString, stdout);
  printf("\n");
  printf("String size: %d \n\n", string_get_size(hString));


  string_push_back(hString, ' ');
  string_push_back(hString, 'N');
  string_push_back(hString, 'e');
  string_push_back(hString, 'W');
  string_extraction(hString, stdout);
  printf("\n");
  printf("String size: %d \n\n", string_get_size(hString));


  string_pop_back(hString);
  string_extraction(hString, stdout);
  printf("\n");
  printf("String size: %d \n\n", string_get_size(hString));


  string_pop_back(hString);
  string_pop_back(hString);
  string_pop_back(hString);
  string_extraction(hString, stdout);
  printf("\n");
  printf("String size: %d \n", string_get_size(hString));

  string_destroy(&hString);

  return 0;
}
```

Your output should be the following:

```
Hello World!
String size: 12

Hello World! NeW
String size: 16

Hello World! Ne
String size: 15

Hello World!
String size: 12
```

TA CHECKPOINT2: Demonstrate to your TA that your function behaves as above.

<u>What you will need to submit:</u>

- string.c file holding the implementation of the above functions
- string.h file holder the functions declaration

- The two driver programs: string_driver, string_driver2
- A Makefile to use to compiler your code
- A valgrind report showing that you do not have any memory leaks
- Combine all of he 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.