Building the string object: phase2

Learning outcomes:
- Functions
- Struct
- Pointers
- Dynamic memory allocation
- Building an opaque object

Lab details:

In this lab, we will continue working with the string object. We will add more functions that will allow better access to the string. The functions to add are the following:
- string_get_size
- string_get_capacity
- string_extraction
- string_insertion

You will also need to create an **enum status** that contains two values: **SUCCESS** and **FAILURE.**

1. string_get_size:
   This function is responsible for returning the size of the current string. It takes a struct string * as pointer and return its size as output. Basically,
   - Takes **struct string*** as input
   - Returns the current size of the string (the size resides inside the struct string).

2. string_get_capacity:
   This function returns the current capacity of the **char_arr** array inside the string struct. The capacity is a member variable inside the string struct. Basically,
   - Takes **struct string*** as input
   - Returns the capacity

3. string_extraction:
   Since the scanf (fscanf) function is an unsafe function and could lead to a buffer overflow, we want to create our own string extraction function to extract a string from a file or from the command line. The function also allows your **char_arr** (inside the struct string) to expand as long as there are more characters to read. In more details:
   - The function takes two inputs: **FILE *** and **string struct *. FILE ***: the file to extract the string from. The file will be opened in the calling function (main for example) and the **string_extraction** function does have to check if the file was correctly opened or not. **string struct *** to save the string in.
   - The function ignores any leading whitespaces till it finds the first character. To do that, you could use the **fscanf(“ %c”, &c): the space before %c tells the scanf to ignore any leading whitespaces. fscanf returns EOF (-1 on our machines) if it could not find any characters and only white spaces.**

- After the first character of the string is found, keep on reading till either reaching a white space (tab, ..etc) or EOF. You will read one character at a time. You could use the **fgetc** to read one character. To check for white spaces, you can use the **isspace** function located in the ctype library. Don't forget to increment the **size** inside the **struct string** (it represents the current size of your string).
- An example for the text in the file would be: "**Test string extraction**"
- To read only the word "**Test**" in the above statement, you will need to stop once you reach the white space. Now, to know that you reached a white space, you would have already read it from the file and holding it inside a char variable. To make your **string_extraction** work similar to the **scanf**, you will need to **ungetc** the **white space character** (use the ungetc function to return the character back to the file stream).
- If the **size == capacity,** this indicates that you have reached the end of your **char_arr** and the **char_arr** needs to expand. To make your **char_arr** expand, you will do the following:
  - Allocate a new character array with double the capacity of **char_arr**
  - Copy **char_arr** into the new character array
  - Destroy the memory held by **char_arr**
  - Make **char_arr** hold the address of the new character array
  - The **string_extraction** needs to return **SUCCESS** if it successfully read the string from the file and **FAILURE** if it failed to do so. You could identify failure if you reach end of file (EOF) and your string is still size zero. This can happen if the whole file was white spaces and contained no visible characters for example. Please note that you should only put back a white space character if you successfully read a non-empty string.

4. string_insertion:
   This function inserts a string into a file or the command line. In more details:
- The function takes two input: **FILE *** and **struct string***
- Reads the characters inside the **char_arr** one by one and write then into the **FILE*. fputc** function could be used to write one character at a time to the **FILE *.** The **fputc** function also returns **EOF** if there was an error in writing to the **FILE ***
- **String_insertion** needs to return **SUCCESS** if it successfully wrote into the **FILE *** or **FAILURE** if it failed to do so.

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

- string.c file holding the implementation of the above functions
- main.c file testing your functions: showing that you can successfully create, extract a string from a file, and write a string into a file.
- A Makefile to use to compile your code
- A valgrind report showing that you do not have any memory leaks.
- Combined all of the above files into a single compressed file. Name the compressed file: lab2-<first-initials><Lastname>
- Submit the compressed file using the submit command to your lab TA.