

Evil Hangman

In order to begin to build a Hangman game we need a dictionary of words. Last week you began to build a string opaque object that will help with this task. This week we will continue adding features to the string object to allow us to read and write words from a dictionary. This lab assumes that you have completed all steps involved in labs 1 and 2.

Switch to your HANGMAN directory: `cd /Spring2019/COMP1020/HANGMAN`

Clean up your directory by typing `make clean`

Add the following declarations to your `string.h` header file and remember to include `stdio.h` near the top of the file.

```
//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: hMy_string will be the handle of a string object that contains
// the next string from the file stream fp according to the following rules.
// 1) Leading whitespace will be ignored.
// 2) All characters (after the first non-whitespace character is obtained
// and included) will be added to the string until a stopping condition
// is met. The capacity of the string will continue to grow as needed
// until all characters are stored.
// 3) A stopping condition is met if we read a whitespace character after
// we have read at least one non-whitespace character or if we reach
// the end of the file.
// Function will return SUCCESS if a non-empty string is read successfully.
// and failure otherwise. Remember that the incoming string may already
// contain some data and this function should replace the data but not
// necessarily resize the array unless needed.
Status my_string_extraction(MY_STRING hMy_string, FILE* fp);

//Precondition: hMy_string is the handle to a valid My_string object.
//Postcondition: Writes the characters contained in the string object indicated
// by the handle hMy_string to the file stream fp.
// Function will return SUCCESS if it successfully writes the string and
// FAILURE otherwise.
Status my_string_insertion(MY_STRING hMy_string, FILE* fp);
```

In addition to following the pre and post-conditions above you need to follow one additional rule for the extraction function. If you read a string from a file stream and then stop on a whitespace then that

whitespace character that you stopped on should be readable by the next file operation. As an example, suppose that the file “simple.txt” contains the sentence: “The quick brown fox jumped over the lazy dogs.” with no spaces or newlines after the period. Consider the following driver program:

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

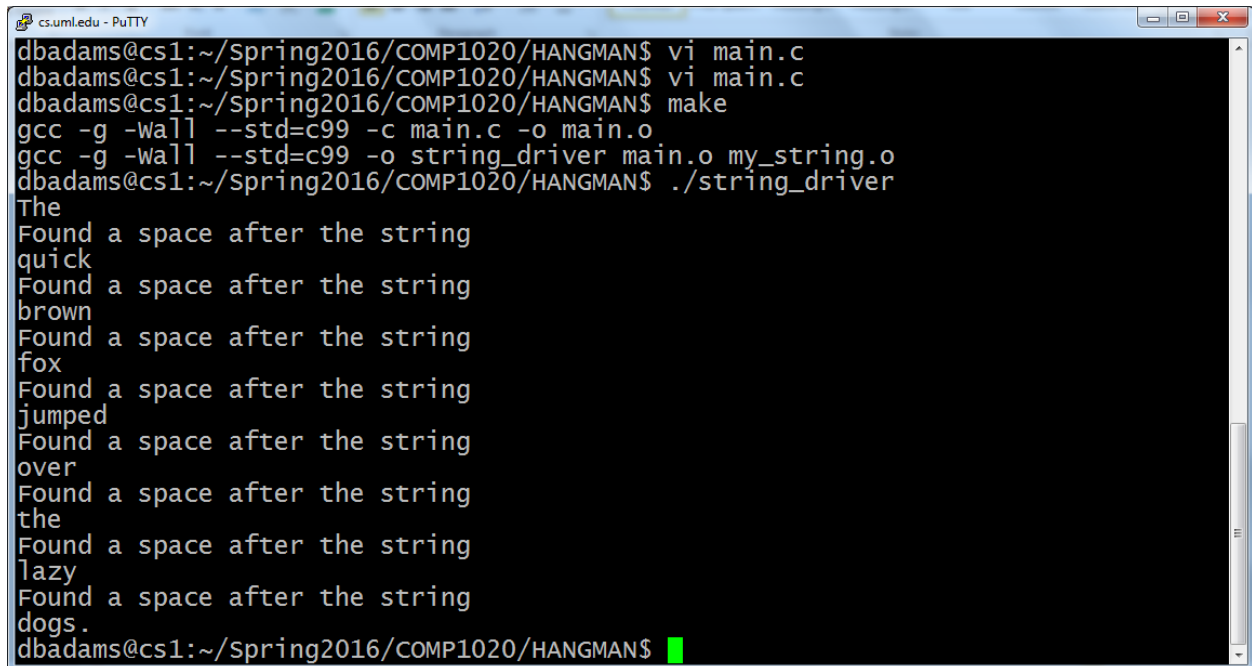
int main(int argc, char* argv[])
{
    MY_STRING hMy_string = NULL;
    FILE* fp;

    hMy_string = my_string_init_default();
    fp = fopen("simple.txt", "r");

    while(my_string_extraction(hMy_string, fp))
    {
        my_string_insertion(hMy_string, stdout);
        printf("\n");
        if(fgetc(fp) == ' ')
        {
            printf("Found a space after the string\n");
        }
    }

    my_string_destroy(&hMy_string);
    fclose(fp);
    return 0;
}
```

Your output should look like the following:



```
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$
```

TA CHECKPOINT 1: Demonstrate to your TA that your function behaves as above and have them verify that there is only ONE space between your words in the file.

Copy the file dictionary.txt from my directory to yours by using the command:

```
cp /usr/cs/fac1/dbadams/Public/dictionary.txt ./
```

Make a copy of your main.c file and call it lab3_checkpoint1.c.

Change your main.c program driver so that it reads all of the words in the dictionary.txt file using my_string_extraction and prints all of the words that have only 5 characters in them on the screen using your my_string_insertion function.

TA CHECKPOINT 2: Demonstrate your new driver to your TA and then while your TA is watching, change the driver so that it only prints the words that are 29 characters long. Explain the meaning of each word in this list to your TA and use them in a sentence to explain your feelings toward this part of this assignment. Have your TA verify that, outside of opening and closing the file and initializing your string, the only functions you call in your code are your my_string_extraction, my_string_insertion, my_string_get_size, and printf for new lines. Finally, demonstrate that your code is running without any memory leaks using valgrind.