Institute of Computer Science

CMSC 21: Fundamentals of Programming

Module JOURNAL

Instructions: Accomplish this every week so we can monitor your progress and improve everyone's learning experience. Completely and honestly answer the following questions in relation to this module's lesson/topic.

Submission:
Filename: "<surname><FirstnameInitial><MiddleInitial>_module0#_journal.pdf"
Example: "delaCruzJC_week01_journal.pdf"
Submit to Google Classroom on Module 01

****Accomplish this journal when you are done (or almost done) with the requirements for the module.*

Name:	**Keith Ginoel S. Gabinete**	Module No:	**5**
Student Number:	**2020-03670**	Date:	**08/12/2021**

- Choose at least one of the things discussed that you understood the most. Imagine explaining it to a classmate. **Explain it in your own words**.

I learned that programmers in C programming language can somehow create their own data type that is unique to basic data types in C with the special way of storing multiple data variables called structures. Structures are said to group multiple variables of different data types and distinguish them as a group under one name. In C, this structure name is often referred to as the structure tag. Structures useful in in storing records temporarily that can then be eventually be stored in files. Defining structures in your programs in C also proves to be a good practice as dealing with block of codes with much length and complexity could drastically decreases your codes' organization and readability; thus, grouping data in your programs such as variables of different data types could lessen the difficulty of understanding your work, much more helps you to work faster and efficiently.

There are three ways of defining a structure in C.

- The first one is by using the classic structure tags.

```
Struct <structure_tag> {
        Members
} where structure_tag, of course serves as the name of the structure.
```

- Second one is by using type-definition

```
Typedef struct {
        // members
} <synonym>;
```

The most noticeable difference between the first two aforementioned methods of defining a structure is the position where the structure's name is written. Unlike in the first one, where the structure tag is defined before the body/members of the structure, the synonym (still the structure's name) was defined after the structure's body/members.

- The third method of defining a structure is just the combination of the two:

```
Typedef struct <tag> {
        //members
} <synonym>
```

Where the structure can have two names and each can be used in the way you may intend to.

Be reminded that one must define a structure outside functions to prevent errors such as a segmentation fault as structures work best if its definition is placed just after the hear files/preprocessor directives defined in your program.

The variables declared inside your structure are called members. Members of the same structure definition mustn't have the same variable name. Moreover, a structure can not have an instance of itself as one of its members. Variables in structures can have data types like integers, floats, characters, doubles, arrays, and pointers. Although it cannot have an instance of itself inside it, instance of another structure definition can still be one of the members along with a pointer to the structure itself. Note that just like functions, if you're going to use an instance of another structure definition inside one of your structure definitions as well, then you must declare the former first before intending to use it inside the latter.

Defining structures do not actually declare any variable; thus, no memory allocated has been used to the structure. Since it is more like a data type of your own, you have to declare a variable with that data type to actually use it inside your program. The structure's name (structure_tag or synonym) can be used to declare the variables of the new data type you created.

There are two ways known to declare a structure variable:

- The first one is by using the keyword struct with format struct <structure_tag> <variable_name>;
- the second one is by using the typedef struct keyword: here instead of writing the word "struct" before the structure's name you'll directly write the structure's synonym then the variable name: <synonym> <variable_name>;

Please, also be reminded that you cannot use the synonym way of declaring pointers to an instance of the structure itself since synonyms are always written after its structure's members/body; for if you do so, you might encounter errors compiling your program telling you that a certain something is undeclared.

We can also initialize structures like normal variables upon their declaration. It's just that one must always follow the same order of values as the members of the structure:
That is - you cannot skip one member of the structure upon its declaration; otherwise, errors would occur. However, you can still leave other variables inside structure undeclared (original states) as long as you do not skip even one of the first variables. The remaining members left to still be declared would be initialized to 0 or NULL (if it is a pointer). Furthermore, an existing structure variable can still be assigned to a new one as well as existing variables to the individual fields. Unfortunately, you cannot do it with variables that are strings, mainly because strings, by default, do not have the capabilities. Still, we can use the strcpy function or the scanf function to put values inside a string member.

Although, structures are deemed to have somehow a data type of its own, the usual arithmetic, relational and bitwise operations cannot be used on structure variables. Moreover, logical operators can't be used to compare them as they are not always stored in a consecutive slot in the memory.

Still, lucky enough, we can still use the mentioned operators to the normal variable members of inside the structure as well as apply the things we usually do with arrays, strings, and etc. to use or access structure variables. With the information stated above, there are still 4 basic operators for structure variables. The first one is the dot operator (.) that is normally used to easily access the members of a given structure. Additionally, if the structure given is a nested one, we can just add another dot (.) to our existing access statement until we reach the value of that one certain member inside the nested structure.

We can use (=) symbol for our second operator where we just assign a structure variable to another structure variable. This means that, by using this kind of operator, we're just copying the contents/ members of one particular structure variable to another structure variable.

In the third available operator called the sizeof operator, we're just getting the amount of space needed by the structure variable when it is successfully declared.

Finally, in the fourth operation, we're allowed to take the address of a structure variable using an ampersand (&) symbol and might as well store it in a pointer. This operation is typically used to assign structure variables to pointers or pass-by-reference them to a function. Additionally, to access the members of a pointer with data type structure, one needs to use the structure pointer (->) – also known as the arrow operator.

And when passing a structure as parameters to functions by value, we just past either its individual members or the entire structure itself. On the other hand, in passing structures to functions by reference, we just pass the address of the structure or a pointer to the structure itself.

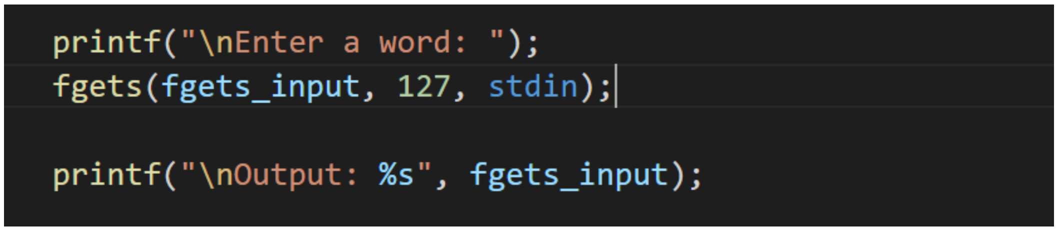
- What problem/confusion did you encounter about the module lessons? **Explain the problem.**

1. The idea of using functions to ask for an input from the user is not already new to me. Especially when working with a program that deals with structures and files, functions for inputs that can be used repeatedly sure is helpful in improving efficiency and readability of the block of codes as a whole. However, there are still times that I found myself succumbed into thinking deeply of what did I do wrong to not arrive at the desired output I had in mind of the program I thought to accomplish in no time. Specifically, in the individual exercise for module 5, where a student, such as I, need to make a music playlist with the use of structures and files, an idea that a music title can be an array of multiple words/strings came into my mind. For example, there are songs like "Fly Me to the Moon" by Frank Sinatra and "Old Town Road" by Lil Nas X whose titles are comprised of words more than 2. As I myself had tried already, the usual fgets function I use to get inputs from the user proves to have a little bit of trouble in storing data as each word (separated by a whitespace character) present in a one-line input stream were being read by the program as one string each. Therefore, if I try to enter "Fly Me to the Moon" as an input, the only set of characters that would be stored in the intended array would be the first word "Fly" as this is the only one fetched by the fgets function I set up. Thus, the need to seek for the better solution must be done nonetheless.

2. When I practice using string functions in C, I noticed that the strlen function behaves differently from the len function I learned in Python. Thus, this brought a little confusion to me while going over the topics discussed in Module 4 (this is more of a realization than an actual probrlem).

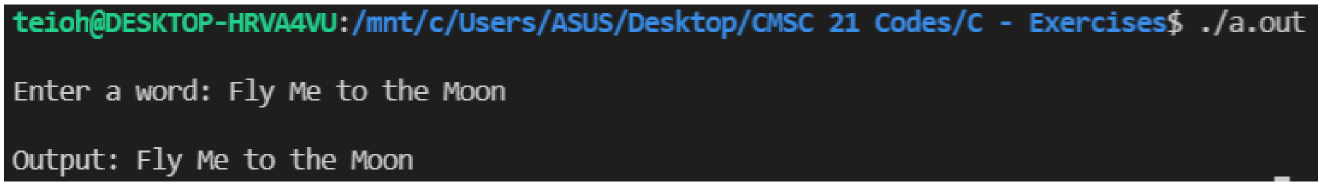
- How did you solve it? **Explain the fix or solution found.**

1. Having a generalization that thinking deeply for more than an hour would get me nowhere near solving this problem I have regarding input strings, I then proceeded to browse online – hoping to stumble upon something that offers a more adept explanation for this kind of concept. Luckily, I found this forum in stackoverflow <https://stackoverflow.com/questions/314401/how-to-read-a-line-from-the-console-in-c>, where it gives me the idea to just stick with my method of using the fgets function to fetch inputs from the user (as it is clearly being hailed as a top method for asking inputs) added with a set of new techniques in reading data in one-line input stream. Here, I learned that the fgets function reads input until it reaches '\n' – a new line character (fgets stopped when enter key is pressed); thus, correlating this idea in my problem, I discovered that by not passing the data fetched by the fgets function to the sscanf function (sscanf is used to restrict inputs) I would be able to get input data from the whole one-line input stream (including whitespaces).
Sample Code:



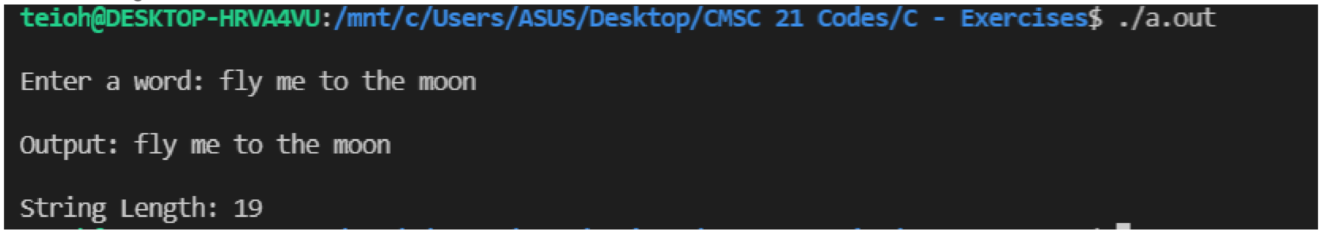
```
printf("\nEnter a word: ");
fgets(fgets_input, 127, stdin);

printf("\nOutput: %s", fgets_input);
```

Output:



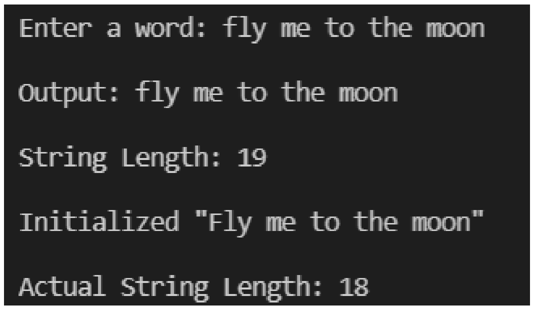
```
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/C - Exercises$ ./a.out

Enter a word: Fly Me to the Moon

Output: Fly Me to the Moon
```

However, new set of problems arises.
a. When using strlen function to measure the length of the fetched input, the returned integer is always greater than its actual length.



```
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/C - Exercises$ ./a.out

Enter a word: fly me to the moon

Output: fly me to the moon

String Length: 19
```

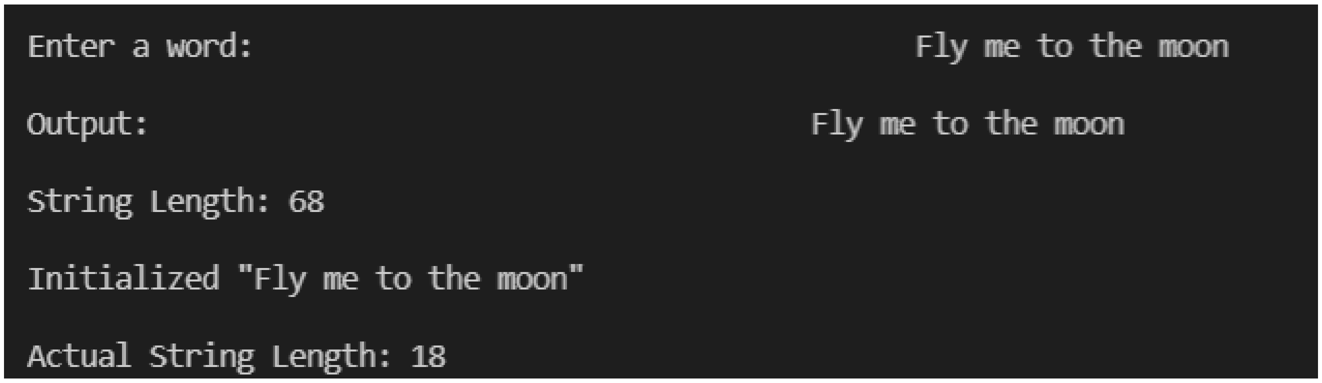
- Actual length of the array of characters "fly me to the moon" is 18.



```
Enter a word: fly me to the moon

Output: fly me to the moon

String Length: 19

Initialized "Fly me to the moon"

Actual String Length: 18
```

This is mainly because, again, the fgets() function stops reading an input when it reaches ‘\n’ character; thus its last element is ‘\n’ instead of ‘\0’, and ‘\n’ character still counts as an element in the strlen function since the strlen function only stops working when it reaches the ‘\0’ – null character.

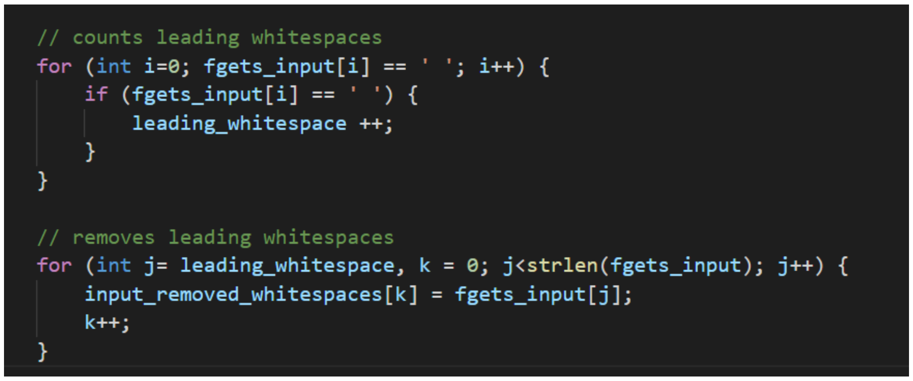
b. Leading and trailing whitespaces are also fetched and is always part of input data.
Example:
Input: “   Fly me to the moon   “



```
Enter a word:                     Fly me to the moon

Output:                       Fly me to the moon

String Length: 68

Initialized "Fly me to the moon"

Actual String Length: 18
```

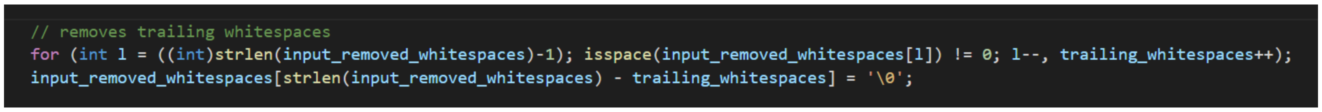
This happens, of course because the fgets function reads all characters (whitespaces included) present in the whole one-line input stream.

Luckily, by spending another time thinking, come up with these solutions below on my own.
Solutions:
Solving the issues about whitespaces:
- By counting the whitespaces (by for-loop) present in fetched data before an actual character(letter), then declaring a new empty array of strings to store (by for-loop) a new one-line string data derived from the original fetched data, I was able to remove the leading whitespaces.



```c
// counts leading whitespaces
for (int i=0; fgets_input[i] == ' '; i++) {
    if (fgets_input[i] == ' ') {
        leading_whitespace ++;
    }
}

// removes leading whitespaces
for (int j= leading_whitespace, k = 0; j<strlen(fgets_input); j++) {
    input_removed_whitespaces[k] = fgets_input[j];
    k++;
}
```

- By using the isspace() function [The isspace() function checks whether a character is a white-space character or not. If an argument (character) passed to the isspace() function is a white-space character, it returns non-zero integer. If not, it returns 0.] the input data(without leading whitespaces) <https://www.programiz.com/c-programming/library-function/ctype.h/isspace> to do the same thing (of removing whitespaces) but in a more efficient way, I was able remove the trailing whitespaces.

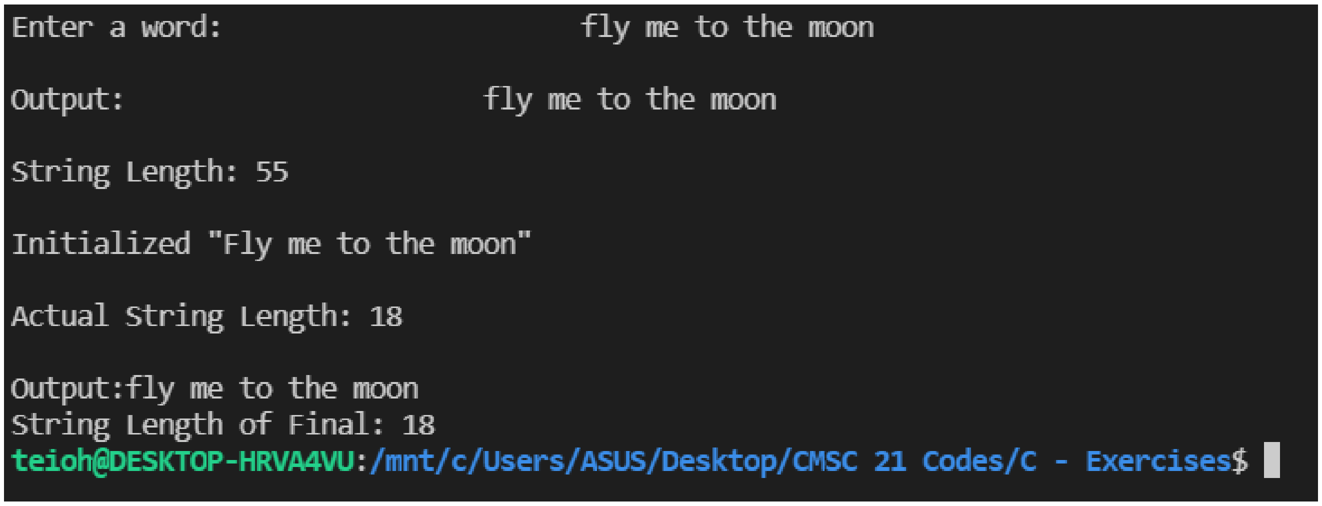


```c
// removes trailing whitespaces
for (int l = ((int)strlen(input_removed_whitespaces)-1); isspace(input_removed_whitespaces[l]) != 0; l--, trailing_whitespaces++);
input_removed_whitespaces[strlen(input_removed_whitespaces) - trailing_whitespaces] = '\0';
```

[the character right after the last valid character in the input data would be replaced by '\0']

Fortunately enough, the '\n' character was actually read by the isspace() function as a whitespace; thus, the need to solve the issue in incorrect string length has also been solved. I truly hit two birds with one stone in this one accidentally.

Final Output:



```
Enter a word:                      fly me to the moon

Output:                      fly me to the moon

String Length: 55

Initialized "Fly me to the moon"

Actual String Length: 18

Output:fly me to the moon
String Length of Final: 18
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/C - Exercises$
```

Notes:

- Using the strcpy function to store this input data to a string variable member of a structure would also work just fine.
- By also using the fgets function, restricting inputs also prove to be much easier (I restrict inputs by using other string.h and ctype.h functions)

- Comments/Suggestions (Optional but we will appreciate it if you tell us one you have in mind)