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:	**3**
Student Number:	**2020-03670**	Date:	**07/29/2021**

- 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 programs in C also use the computer's memory (along with all the applications and processes running in our machine) to store vital information. Every time a C program runs, each variable declared in it is assigned to a unique location in computer's memory. We can view memory as a sequence of storage locations in which each location is capable of storing one byte of data and is labeled by numbers (usually in hexadecimal notation) starting from zero. In C, we can access the values of the variables stored in the memory using pointer variables. A pointer is a variable that stores the memory address of another variable. To declare a pointer variable, we follow the default syntax:

<data_type>*<variable_name>

where <data_type> is any valid data type in C that corresponds to the data type of the variable the pointer is pointing to and <variable_name> is any valid identifier you want to use to name your pointer. Note that whitespaces are normally ignored in a C workspace (although whitespaces inside quotation marks ' ', " " are exceptions), therefore even if we put the star symbol * closer to <data_type>, or closer to <variable_name> or somewhere in between <data_type> and <variable_name>, we can still successfully declare our pointer.

In dealing with pointers, we use two operators. The first one is called the Address Operator (&) and the second one is called the Indirection Operator (*). The former is used to obtain the memory address of a certain variable in C while the latter is used to access the content the pointer is pointing to. We must remember that a pointer can only point to the data type we specified in its declaration and that it can have "no value" if we assign a value of NULL to it. Moreover, without the address operation, the indirection operation will be semantically incorrect since it does not point to any variable - or to simply say, its content has a NULL value. This semantic error will not be shown during the program's compilation but might result into a segmentation fault during the program's execution. This error is also commonly known as uninitialized pointer error.

Pointers can also point to another pointer. To do this, we just need to add one asterisk greater than the number of asterisks present in the pointer we want to point our current pointer to. To easily understand this, we have: <data_type>** <variable_name> [= <assignment>] which can be used to a pointer with one asterisk (e.g. *p, *k, *abc). Note that if we want to get the address of a certain pointer that has n number of asterisks, we just need to declare a pointer variable with n+1 number of asterisks.

There are many applications/uses of pointers in C. Example of which is in creating dynamic variables. Dynamic variables are variables created during the runtime of a program. Creating some of these is efficient when you do not know if certain variables are needed in certain tasks or not. A pointer is used in holding the address of the dynamically allocated variable returned by the malloc function (defined in the stdlib.h library). The dynamically allocated variable can then be accessed using pointer operations. After using that particular dynamic variable, we can then free its allocated space using the free() function (also defined in the stdlib.h library) so that other processes will be able to use that space in the memory. Another application of pointers can be observed in parameter reference passing. By passing a parameter by reference, we are passing the address of a variable or the value of a pointer to a certain function. This technique is useful especially if we need to return more than one value using a function. Pass by reference can also be used to swap values of certain variables for convenience.

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

1. Since I'm now using fgets() function in fetching inputs to accurately restrict or validate my users' input data, arrays are now also always present in my programs. Upon declaring a pointer to an array using the standard method of putting an asterisk before a valid identifier/variable name (which I thought would work), I then encounter floods of errors. I even thought at first that the cause of the errors is not my singe declaration of that pointer pointing to an array since the errors are pointing to other lines different from that declaration of pointer so I even tried to resolve the program. Unfortunately, errors continuously appear during compilation. When I decided to temporarily comment out that declaration of pointer to an array of mine, to my surprise, my program for the individual exercise has then successfully compiled itself. That's when I realized the normal method of declaring a pointer would not work when declaring a pointer that'd be pointing to an array.

2. As I'm working with functions that can be used generally for various variables, an idea of displaying the name of the variable that is being passed to the function during runtime has crossed my mind so that the user would not get confused as to why he/she needs to put another input after already entering a first valid input (since the display message is uniform every time the function is being called; especially if the program requires 2 or more inputs). I thought that by just typing the variable name inside the function using a printf() function would solve this concern of mine, but then I realize that since the function is being shared between other variables with different names as well then every time I call that function with these different variables as parameters, the displayed variable name included in that printf() function would be uniform to all calling/execution of the said function.

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

1. I browsed online for solutions about this problem of mine regarding declarations of a pointer to an array. I then stumbled at this site <https://www.geeksforgeeks.org/pointer-array-array-pointer/>, where I learned the correct way of declaring a pointer variable that can point to an array. Here, instead of just adding an asterisk before a valid identifier/variable name, you must enclose that *<variable_name> with parentheses then add bracket symbols [ ] at the right side of the closing parenthesis then specify the size of an array (we're gonna point into) inside those brackets. Here is an example,

```c
#include <stdio.h>

int main() {
    // array
    char sample[4] = "four";

    // pointer
    char (*sample_ptr)[4];

    sample_ptr = &sample;

    printf("%s\n", *sample_ptr);

    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

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

2. The easiest solution I first came up with is to define other functions for each input variable but then again, the very concept of creating functions in my program would be pointless since the very purpose of functions is to reuse blocks of code so the programmer could work efficiently; thus, I moved on to the internet to find other solutions. I then found out in this site <https://www.geeksforgeeks.org/how-to-print-a-variable-name-in-c/> that it is actually possible to get the name of a certain variable using the #directive. According to this site, a #directive converts its argument in a string. Here is an example,

```c
#include <stdio.h>
#define getName(var)  #var

int main() {
    int myVar;
    printf("\n%s\n\n", getName(myVar));
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

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

myVar

teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/Laboratory Exercises$
```

After learning about this technique, I then went back to continue working on my code. It worked... at least. A problem arises again when I try to use this technique in a pointer variable. Instead of returning the variable name the pointer is pointing to using the Indirection operator *, what the technique returned was the name of the pointer itself with the symbol * attached before it. So, for the nth time, I browsed again in the internet for possible solutions. Finally, I reached this forum in stack overflow <https://stackoverflow.com/questions/7205764/is-it-possible-to-find-out-the-variable-name-the-pointer-pointing-to/7206174> where I found the shocking truth - that is, by default, there is no normal way to get the name of the variable/pointer another pointer is pointing to unless you know some advance stuffs in C programming language to comprehend the presented solutions in this forum. Unfortunately, at this point, I still do not have that knowledge to understand clearly the methods they discussed in that forum that suffice my aforementioned desire. Pondering with myself for about half an hour to find for other ways to still somehow resolve my problem, I then came up with the thought of adding another parameter to my function that takes an array/character variable I can then be used in a printf() function inside the said function to just easily specify the variable being passed to the function during runtime. Here is a simple demonstration,

```c
#include <stdio.h>

void general (char *variable_letter, char var_name[64]) {
    printf("Value: %c \tVariable name: %s\n\n", *variable_letter, var_name);
}

int main() {
    char x = '1', y = '2', z = '3';

    general(&x, "x variable");
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/Laboratory Exercises$ gcc test.c
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/Laboratory Exercises$ ./a.out
Value: 1        Variable name: x variable

teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/Laboratory Exercises$
```

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