

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: "deCruzJC\_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: **2**

Student Number: **2020-03670**

Date: **07/22/2021**

- 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 a function in C programming language, just like in Python, is a useful block of code that can be used repeatedly just as how many times you want to use it to either improve the readability and cleanliness of your code or to just simply save you a bit of time performing certain tasks throughout your code with much convenience and simplicity.

The general form of a function in C is:

```
<return_type> <function_name> ([formal_parameter_list]) {  
    <function_body>  
}
```

return\_type pertains to the data type of the value the function returns. Note that some functions can perform tasks without returning a value; thus, the return\_type is void. <function\_name>, of course, is the actual name of your function. While <formal\_parameter\_list> is the list of arguments your function expects. Altogether, the first three parts of a function in C: return\_type, function\_name, and parameter\_list makes up the function signature. Lastly, the <function\_body> comprises sets of commands/blocks of codes that will be executed once the function was called.

Now, what is a function call? Function is the way to use the function you defined. You can just simply invoke a function without assigning a variable to it (especially if the return\_type of the function is just void) or with assigning certain variable to store its returned value. When a function is being called it takes arguments of data type similar to what you defined on its formal\_parameter\_list. This list of arguments/parameters you are now passing to the function is called the actual\_parameter\_list (actual - means - actual use of function). You can then display the returned value of the function you called simply by using built-in functions like printf.

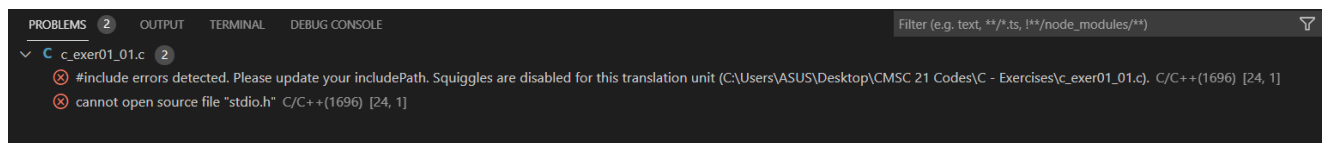
Just like in Python, functions in C should be defined before a function call. However, there is a way in C programming language where you can define a function after its function call as long as you declare it first. Declaring a function in C without its function\_body is called Function Prototyping. A function prototype only needs the signature of the function. Moreover, in its formal\_parameter\_list, you can just actually declare only the parameter/s data type/s first and declare a name for it/them completely later at the actual definition of the function. Take note that you cannot initialize a value for the parameter/s of your function. You're only allowed to declare its/their data type/s and variable

name/s. Also remember that return statements halt the execution of the defined function and that local variables in a function must be declared at the beginning part of the function to avoid unnecessary errors.

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

1. Typing a long line of command in the bash or windows terminal every time I wanted to use the gcc compiler to compile my c program and run it proves to be a tedious and inefficient work for me (especially when working with a lot of codes at the same time).

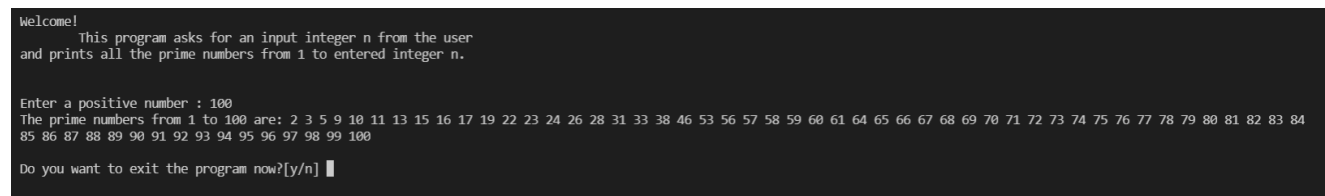
2. While working on the individual exercises in visual studio code, the mentioned source-code editor suddenly encountered errors that restrict my c program to be compiled and run. The following error messages implies that the problem occurred within the compiler itself.



```
PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE
Filter (e.g. text, **/*.ts, !**/node_modules/**)
c_exer01_01.c 2
#include errors detected. Please update your includePath. Squiggles are disabled for this translation unit (C:\Users\ASUS\Desktop\CMSC 21 Codes\C - Exercises\c_exer01_01.c). C/C++(1696) [24, 1]
cannot open source file "stdio.h" C/C++(1696) [24, 1]
```

3. Creating a recursive function surely was not an easy task. There would occur errors that would really frustrate you as it makes you clueless with the available solutions you could try. I, too, experienced this firsthand while I was working with the individual exercise for module 2 where the program should output all the prime numbers in range 1 to a given positive integer input n. Having planned and analyzed the whole recursive function I made, I was more than certain that my program would output the desired correct outputs. Unfortunately, occurrences of unexpected outputs displayed in the terminal made my expectations flopped. What's more annoying about it is no matter how much I tweak that recursive function, the said problem continues to exist.

Program output:



```
welcome! This program asks for an input integer n from the user
and prints all the prime numbers from 1 to entered integer n.

Enter a positive number : 100
The prime numbers from 1 to 100 are: 2 3 5 9 10 11 13 15 16 17 19 22 23 24 26 28 31 33 38 46 53 56 57 58 59 60 61 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Do you want to exit the program now?[y/n]
```

Expected Output:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

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

1. While thinking of various ways to compile and run my c programs efficiently, I suddenly remembered that there was a certain part on the lab handouts for module 1 that discusses this exact problem for programmers. As I reread the lab handouts for the "Introduction to C" topic, I discovered that by creating makefiles, you would not need to write the gcc - o command to the terminal every time you want to compile and run your programs. Rather, you can just store the same command for compiling a c program to a makefile once, and assign some names and target files for it, then simply invoke the makefile in the terminal by just typing the word 'make'. Also, with makefiles, if you want compile another program you're working with, you can just modify the exact same makefile using the same editor you write your programs on, change the names of some target files and invoke it again in the terminal. This does not only save me time but allows me to conserve some energy as well.

Also, thanks to this YouTube video right here < <https://www.youtube.com/watch?v=zfuOcvYrhOs>>. I was able to understand more clearly how makefiles work. Now I use makefiles to compile and run my program. Additionally, I discovered that you can also use arrow keys in Visual Studio code terminal to reuse recent-called commands.

Inside a makefile:

```
1  c_exer01_02: c_exer01_02.c
2  gcc -o c_exer01_02 c_exer01_02.c
```

The makefile is in a directory where the target file is located, that's why the lines inside this makefile are these only short.

How it works in the terminal:

```
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/C - Exercises$ make
gcc -o c_exer01_02 c_exer01_02.c
teioh@DESKTOP-HRVA4VU:/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes/C - Exercises$ ./c_exer01_02
Enter a month number (1-12): █
```

2. Upon watching some YouTube videos to solve this issue of mine (like this one <<https://www.youtube.com/watch?v=e1oQbEzAMUA>>), I discovered that my Visual Studio Code has wrongly located my gcc compiler. The reason that made this problem occur is still unclear to me. As what I'm trying to say earlier, this problem pops out of nowhere; thus, I spend hours trying to fix this problem. Anyway, upon watching the said video, I've found out that I need to configure the extension in visual studio code for the C language and manually change the directory of the compiler itself to fix the problem. However, I cannot locate my installed compiler in my PC for the first few times I tried to resolve the problem. Arriving at a realization, that the current process I'm doing would go nowhere near success, I then searched again in the web for possible solutions. I then found out from this another GitHub forum <<https://github.com/microsoft/vcpkg/issues/11816>> that having vcpkg in my PC would let me install new stable libraries and compilers for my C programs. Thanks to the clear instructions in this site <<https://vcpkg.io/en/getting-started.html>>, I was able to add vcpkg to my windows laptop along with my new set of c libraries and compilers. Thankfully, trying again the same method earlier about changing the path of the gcc compiler in the settings of certain C extension in Visual Studio Code, I now luckily was able to locate my new gcc compiler and save the configurations for the extension without problem occurrences. Error messages about compiling after that disappeared. Having the problem fixed, I then began to work again with my codes.

Unconfigured setting:

```
1  {
2    "configurations": [
3      {
4        "name": "Win32",
5        "includePath": [
6          "${workspaceFolder}/**"
7        ],
8        "defines": [
9          "_DEBUG",
10         "UNICODE",
11         "_UNICODE"
12       ],
13       "cStandard": "c17",
14       "cppStandard": "c++17",
15       "intelliSenseMode": "windows-msvc-x64"
16     }
17   ],
18   "version": 4
19 }
```

Configured setting:

```
1  {
2    "configurations": [
3      {
4        "name": "Win32",
5        "includePath": [
6          "${workspaceFolder}/**"
7        ],
8        "defines": [
9          "_DEBUG",
10         "UNICODE",
11         "_UNICODE"
12       ],
13       "cStandard": "c17",
14       "cppStandard": "c++17",
15       "intelliSenseMode": "clang-x64",
16       "compilerPath": "c:\\Program Files\\Msys64\\mingw64\\bin\\gcc.exe"
17     }
18   ],
19   "version": 4
20 }
```

3. Upon spending some time browsing in the internet - finding a possible solution for this problem of mine, I stumbled upon this forum in stackoverflow <<https://stackoverflow.com/questions/26807173/c-recursive-function-for-prime-number-with-just-one-parameter/47714823>> where some user mentioned the use of a global variable for some functions (including recursive functions). Having a hope that declaring a certain variable inside my program globally (instead of declaring it locally inside the recursive function) would fix my problem, I then decided to give it a try. Shocked on what has happened, a simple declaration of one variable outside the recursive function actually fixes the issue. When I try to compile and run my program, the before unexpected outputs are now gone while the desired correct outputs were still there. I then realized that by declaring that certain variable locally inside the recursive function would actually resets the value of that variable every time the function recurs; thus, producing unexpected outputs. Though the solution to the problem was very plain and simple, I'm really grateful that I was able to solve it without rewriting my recursive function from scratch.

With a local variable:

```
// This recursive function returns 1 if the given test_number is a prime number; if not, it returns 0
int isPrime (int test_number, int divisor) {
    // Note: integer 2 will always be passed to the divisor variable in the formal parameters above whenever we call this recursive function

    // this stores the number of divisors a number currently has
    int divisor_count;

    // base cases (smallest base cases)
    if (test_number <=2) {
        // if the test_number is 1, automatically distinguishes it as a non-prime number
        if (test_number == 1) {
            return 0;
        }
    }
}
```

Output:

```
Welcome!
    This program asks for an input integer n from the user
    and prints all the prime numbers from 1 to entered integer n.

Enter a positive number : 100
The prime numbers from 1 to 100 are: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Do you want to exit the program now?[y/n] █
```

With a global variable:

```
// This is a global integer variable that will be used inside a recursive function
// this stores the number of divisors a number currently has
int divisor_count;

// This recursive function returns 1 if the given test_number is a prime number; if not, it returns 0
int isPrime (int test_number, int divisor) {
    // Note: integer 2 will always be passed to the divisor variable in the formal parameters above whenever we call this recursive function

    // base cases (smallest base cases)
    if (test_number <=2) {
        // if the test_number is 1, automatically distinguishes it as a non-prime number
        if (test_number == 1) {
            return 0;
        } else if (test_number == 2) { // again 2 is our default value (whenever calling this function) for prime
            return 1;
        }
    }
}
```

Output:

```
Welcome!
    This program asks for an input integer n from the user
    and prints all the prime numbers from 1 to entered integer n.

Enter a positive number : 99
The prime numbers from 1 to 99 are: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Do you want to exit the program now?[y/n] █
```

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