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:	**1**
Student Number:	**2020-03670**	Date:	**07/15/2021**

- 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 switch statements in c allows you to test a value from a variable or an expression for equality against a list of values, called cases, inside the statement. The expression in the switch statement cannot be a float or a string since it only accepts a valid expression that yields an integer value. That being said, switch statements can have a character as an expression since all characters are eventually converted to an integer before any operation <https://overiq.com/c-programming-101/the-switch-statement-in-c/> (as a character variable contains only one value). Constant expressions on cases can be of character type or integer type. We also do not allow multiple constant expressions in a single case statement. Cases' constant expressions should also be unique from one another to prevent unnecessary errors. A default case is also unnecessary for a switch statement - meaning, we can freely run a switch statement in our code without even setting up a default case for it. A break statement is used at the end of each case to break out of the switch statement. Note that switch statements can also be rewritten using if-else statements. The main reason we use switch statement to compare expressions of less complexity is because it sometimes executes faster than its if-else counterpart. Furthermore, the syntax of a switch statement is much easier to read and write.

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

1. In setting up the windows subsytem for Linux so I can compile and run my codes in C language, the Linux distribution called Ubuntu (From the Microsoft Store) repeatedly encountered failures during several attempts of installation.

2. Changing the current working directory from the default terminal directory [/home/teioh] to the location I'm storing my C programs [/mnt/c/Users/ASUS/Desktop/CMSC 21 Codes] every time I choose to create a new c file proved to be a bit of a hassle for me.

3. This is a minor problem but I think it'll be good to include it anyway:
I constantly forget to put a semicolon - ';' at the end of some statements in my c program.

4. Using scanf function proves to be problematic for me in getting inputs from the user as most of the time that I plan to set restrictions in fetching the said inputs I continuously encounter never-ending chains of errors.
Here is an example:
I've learned from this YouTube video <https://youtu.be/feUTrLz7l8k> that scanf function returns an integer value of how many input/s was/were successfully passed to their designated variable. This integer value can then be used to check whether the scanf function was successful in fetching a valid input.
For example, if you enter 12, then the scanf function will return an integer value of 1 (succesful in reading and storing a valid input). On the other hand, if I try to enter values like 'a', 'b' or 'c', the scanf function returns an integer value of 0 as scanf function was not successful in passing the input data to its designated variable. Thus, enabling myself to restrict the scanf function in reading inputs of integers only. Unfortunately, bigger problems occur when I try to enter something like an array of characters ("abcd") or a line of integers (123 456 789) with whitespaces. When these kinds of inputs were to be processed by the scanf function, overflow in the input stream would occur. In the first scenario (where an array of characters is entered), the scanf function reads the characters from this input one by one as scanf distinguished each of them as character data-typed and not as a string or an array. On the second scenario, the whitespaces in between the sets of integers separate and distinguish each of them as one integer input value. Meaning, the first set of integers (123) would fill in one slot in our input stream, (456) would fill in another slot, likewise the set of integers (789) would take one slot for it in our input stream. This overflow in our input stream would make our scanf function iterates a number of times. As the terminal outputs unnecessary sequence of statements because of this, our code would appear unpresentable to the user/s or to our client/s.

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

1. Upon browsing for almost an hour in the internet, I came across this forum in GitHub <https://github.com/MicrosoftDocs/WSL/issues/436> where people discuss the same issue I've encountered.
There, I discovered that Virtualization technology in the computer's BIOS Menu should be enabled. A disabled Virtualization would restrict a single processor to act as if it was multiple individual CPUs. Thus, enabling this, would allow the processor to run a guest operating system (Linux) simultaneously with the host operating system (Windows). After I enabled the said option in the BIOS, I was then able to successfully installed my windows subsystem for Linux.

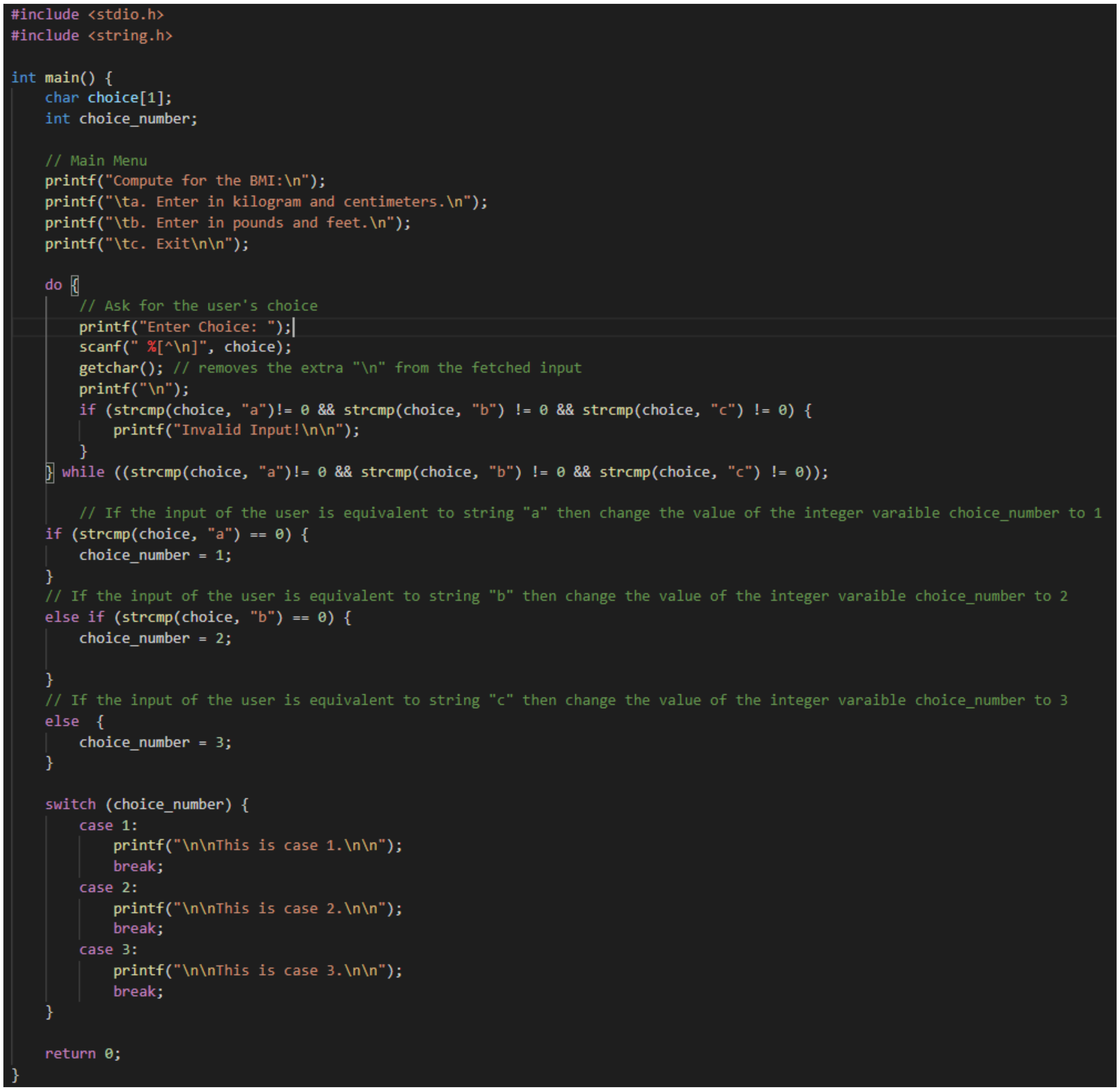
2. After spending some time understanding how the wsl2 extension I've installed for Visual Studio Code works with the help of this site <https://code.visualstudio.com/blogs/2019/09/03/wsl2>, I've discovered that by simply opening a folder first in VS Code Home Window before creating new files/C programs would automatically change the default working directory to the current location of the opened folder. Doing this saves me a lot of time working with my programs.

3. To teach myself some discipline about putting a semicolon symbol – ‘ ; ‘ at the end of every statement in my code, I am now putting a semicolon right after I call a certain function (e. g. printf(), scanf(), etc.) before even filling in the parameter/s of the said function. This technique proves to be effective for me as now, I'm only encountering one, two or no error at all due to missing semicolons in my code whenever I try to compile it.

4. I found two solutions for the problem mentioned above (about scanf functions). The first solution is to play with the string.h preprocessor directive to manipulate arrays of characters. Here, instead of declaring character variables or integer or float variables, we use arrays to store the scanf fetched inputs. By doing this, we would be able to restrict the input values the scanf function accepts more efficiently than just by restricting them with the technique I've learned from the video above. Here, we can also avoid overflow in our input stream since the scanf function would read an entire line of input as one array (regardless of the appearances of whitespaces and various data types). We can then compare the fetched inputs to the strings that contain the valid data the scanf function should only take using the strcmp function from the string.h header <https://stackoverflow.com/questions/8222495/how-to-compare-strings-in-an-if-statement>. If the input string match the valid data string, we can then transform the valid data string to a valid data of different type by declaring a new variable (that is not an array) and storing the same exact data from the valid string to that declared new variable.

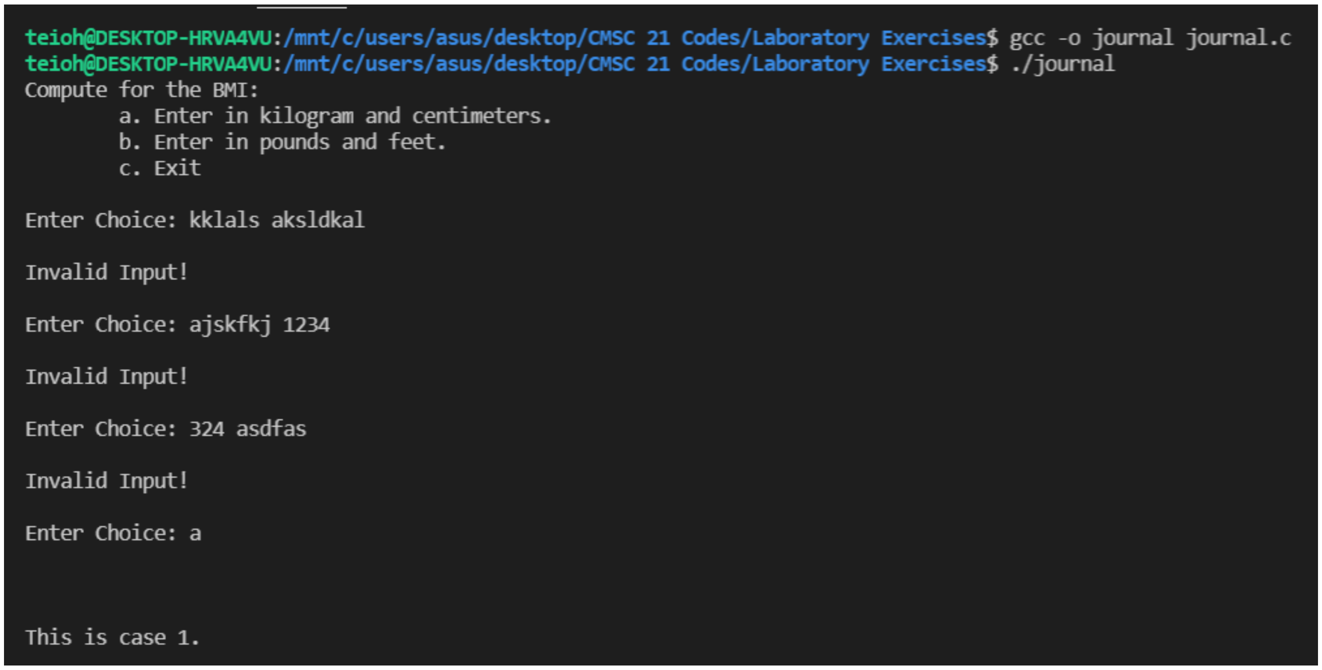
Simple use of the said solution above:

Code:



```c
#include <stdio.h>
#include <string.h>

int main() {
    char choice[1];
    int choice_number;

    // Main Menu
    printf("Compute for the BMI:\n");
    printf("\ta. Enter in kilogram and centimeters.\n");
    printf("\tb. Enter in pounds and feet.\n");
    printf("\tc. Exit\n\n");

    do {
        // Ask for the user's choice
        printf("Enter Choice: ");
        scanf(" %[^\n]", choice);
        getchar(); // removes the extra "\n" from the fetched input
        printf("\n");
        if (strcmp(choice, "a")!= 0 && strcmp(choice, "b") != 0 && strcmp(choice, "c") != 0) {
            printf("Invalid Input!\n\n");
        }
    } while ((strcmp(choice, "a")!= 0 && strcmp(choice, "b") != 0 && strcmp(choice, "c") != 0));

        // If the input of the user is equivalent to string "a" then change the value of the integer varaible choice_number to 1
    if (strcmp(choice, "a") == 0) {
        choice_number = 1;
    }
    // If the input of the user is equivalent to string "b" then change the value of the integer varaible choice_number to 2
    else if (strcmp(choice, "b") == 0) {
        choice_number = 2;

    }
    // If the input of the user is equivalent to string "c" then change the value of the integer varaible choice_number to 3
    else  {
        choice_number = 3;
    }

    switch (choice_number) {
        case 1:
            printf("\n\nThis is case 1.\n\n");
            break;
        case 2:
            printf("\n\nThis is case 2.\n\n");
            break;
        case 3:
            printf("\n\nThis is case 3.\n\n");
            break;
    }

    return 0;
}
```

Output:



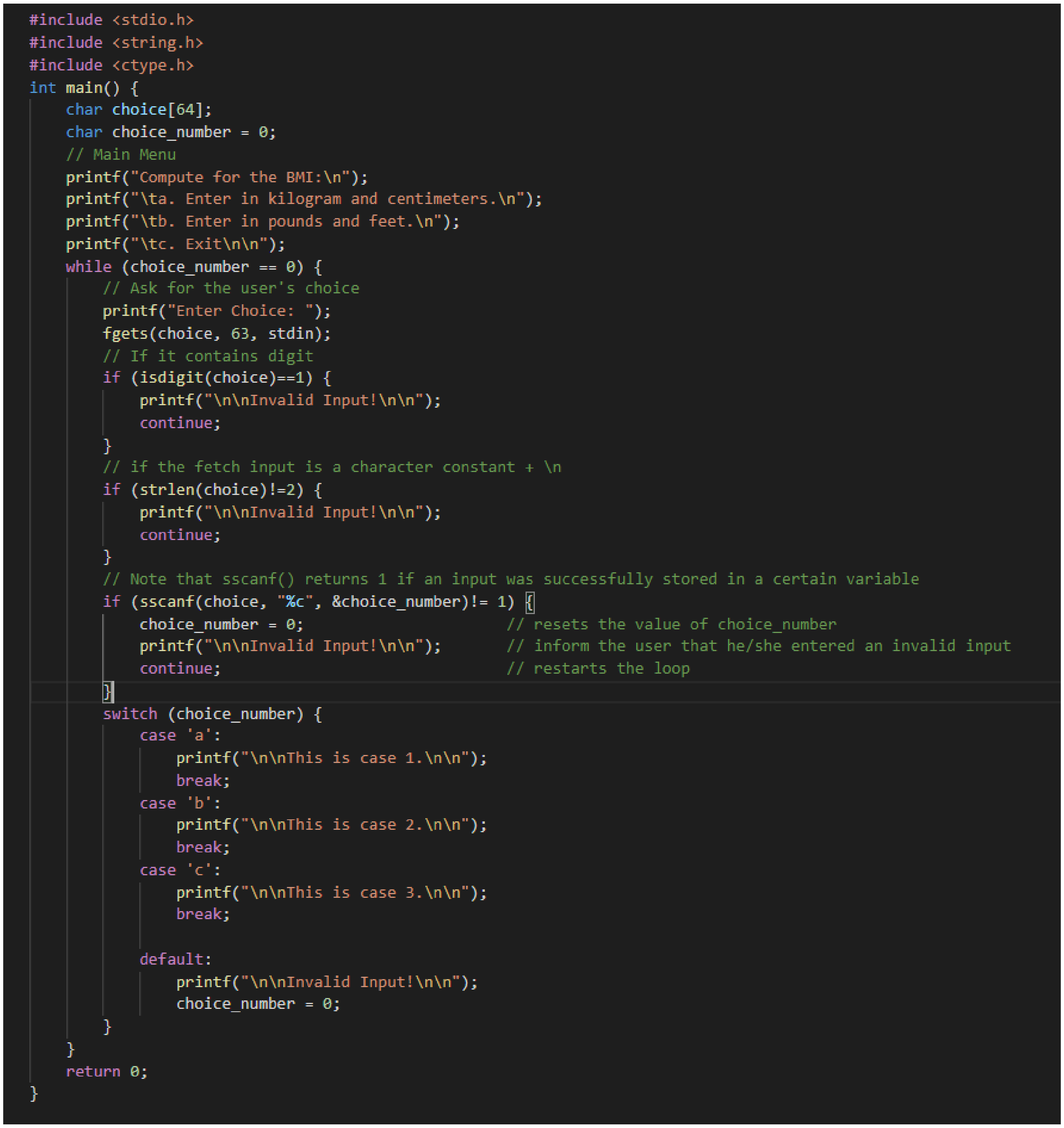
```
teioh@DESKTOP-HRVA4VU:/mnt/c/users/asus/desktop/CMSC 21 Codes/Laboratory Exercises$ gcc -o journal journal.c
teioh@DESKTOP-HRVA4VU:/mnt/c/users/asus/desktop/CMSC 21 Codes/Laboratory Exercises$ ./journal
Compute for the BMI:
        a. Enter in kilogram and centimeters.
        b. Enter in pounds and feet.
        c. Exit

Enter Choice: kklals aksldkal

Invalid Input!

Enter Choice: ajskfkj 1234

Invalid Input!

Enter Choice: 324 asdfas

Invalid Input!

Enter Choice: a



This is case 1.
```

This is one of the solutions I've found for this problem in scanf functions.

However, this also proves so tedious to use as you would need setup lots of restrictions if you want the scanf function to only accept integer/float values or characters in the alphabet. Thus, I searched for another way that is more efficient than our first solution. Upon browsing through the internet, I've come across this YouTube video <https://youtu.be/zZPkgW9VPKw> where I discovered that fgets and sscanf functions provide much easier way to read and store input data. The fgets function (just like the first solution) accepts inputs as an array. The sscanf function would then convert the data (from the fgets function) to a certain variable declared in our program that is not data-typed string. We can then normally use if statements, loop statements or switch statements to restrict input values for these functions.

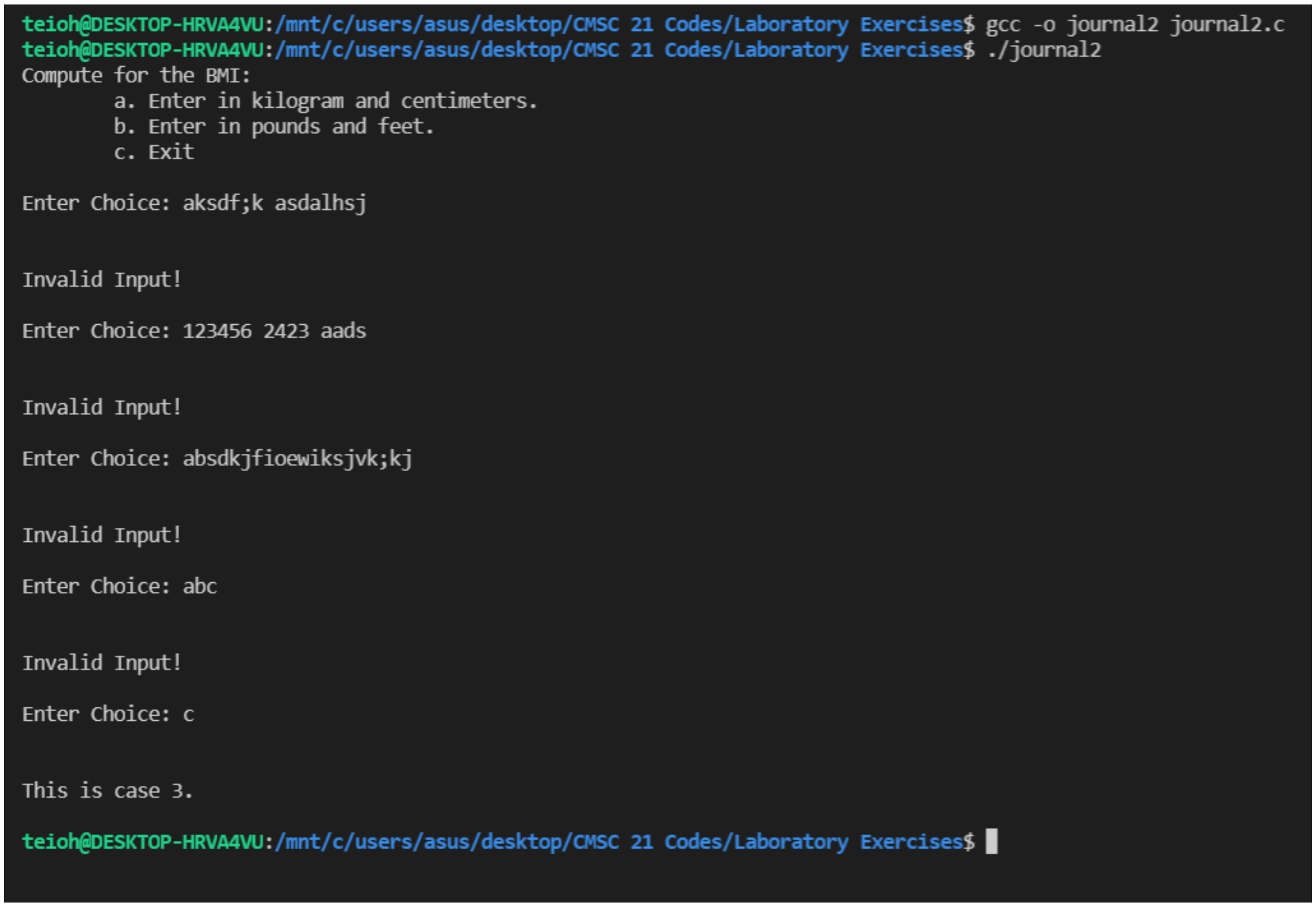
Simple use of the said solution above:

Code:



```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
    char choice[64];
    char choice_number = 0;
    // Main Menu
    printf("Compute for the BMI:\n");
    printf("\ta. Enter in kilogram and centimeters.\n");
    printf("\tb. Enter in pounds and feet.\n");
    printf("\tc. Exit\n\n");
    while (choice_number == 0) {
        // Ask for the user's choice
        printf("Enter Choice: ");
        fgets(choice, 63, stdin);
        // If it contains digit
        if (isdigit(choice)==1) {
            printf("\n\nInvalid Input!\n\n");
            continue;
        }
        // if the fetch input is a character constant + \n
        if (strlen(choice)!=2) {
            printf("\n\nInvalid Input!\n\n");
            continue;
        }
        // Note that sscanf() returns 1 if an input was successfully stored in a certain variable
        if (sscanf(choice, "%c", &choice_number)!= 1) {
            choice_number = 0;                      // resets the value of choice_number
            printf("\n\nInvalid Input!\n\n");       // inform the user that he/she entered an invalid input
            continue;                               // restarts the loop
        }
        switch (choice_number) {
            case 'a':
                printf("\n\nThis is case 1.\n\n");
                break;
            case 'b':
                printf("\n\nThis is case 2.\n\n");
                break;
            case 'c':
                printf("\n\nThis is case 3.\n\n");
                break;

            default:
                printf("\n\nInvalid Input!\n\n");
                choice_number = 0;
        }
    }
    return 0;
}
```

Output:



```
teioh@DESKTOP-HRVA4VU:/mnt/c/users/asus/desktop/CMSC 21 Codes/Laboratory Exercises$ gcc -o journal2 journal2.c
teioh@DESKTOP-HRVA4VU:/mnt/c/users/asus/desktop/CMSC 21 Codes/Laboratory Exercises$ ./journal2
Compute for the BMI:
        a. Enter in kilogram and centimeters.
        b. Enter in pounds and feet.
        c. Exit

Enter Choice: aksdf;k asdalhsj


Invalid Input!

Enter Choice: 123456 2423 aads


Invalid Input!

Enter Choice: absdkjfioewiksjvk;kj


Invalid Input!

Enter Choice: abc


Invalid Input!

Enter Choice: c


This is case 3.

teioh@DESKTOP-HRVA4VU:/mnt/c/users/asus/desktop/CMSC 21 Codes/Laboratory Exercises$
```

And these are the ways I've found to deal with the scanf problems in c.

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