Supplement: How to Test Your Machine Problem (MP) Solution

サルバドール・フロランテ

1. INTRODUCTION

This document describes a technique for testing your MP solution. It is recommended that input redirection be used instead of typing via keyboard the data values corresponding to **scanf()**. This will allow testing to be automated which in turn will lessen time-consuming and possibly error-prone manual input.

2. EXAMPLE PROBLEM

Write a C program that will process data stored in the text file **STUDENTS.TXT**. The data should be read via **fscanf()** and stored in an array of structures. The file contains several rows of data with each row segregated into four columns pertaining to a student record. The 1st column shows the a student's ID number, the 2nd and 3rd columns show the student's last name, and first name respectively, and the 4th column shows the student's grade (between 0.0 to 4.0). Assume that all names are coded in CAPITAL letters. Open the file **STUDENTS.TXT** and study and examine its contents.

Implement a function that will compute and return a value that answers the following parametrized question:

Parametrized Question: What is the grade of **<param_lastname> <param_firstname>**?

where **<param_lastname>** and **<param_firstname>** are values supplied as function parameters.

Actual Question #1: What is the grade of **CRUZ MARIA**?
Actual Question #2: What is the grade of **WAYNE BRUCE**?

The answer to Question #1 is 2.5 since a student with the parameters CRUZ MARIA match the name of one of the students. The required function should thus return a value of 2.5.

For Question #2, however, there is no student matching the name WAYNE BRUCE. In such a case, the function should return a -1.0 which is the numeric indicator that the search did not produce a match.

3. EXAMPLE SOLUTION

An example solution is coded in **SHAZAM.c**. Please study and understand the code. Note that it includes a header file named **datastructure.h**. Open that file as well to see the **typedef** and **struct** declarations.

The function that answers the Question: What is the grade of **<param_lastname> <param_firstname>**? is **`Get_Grade()`**. Its function prototype is

```
float
Get_Grade(struct studentTag List[], int nstudents,
          const char *param_lastname, const char *param_firstname);
```

The **`const`** keyword is not really necessary but it is better to have it. An alternative function prototype using char [] data type that also works is shown below. It works because **`char *`** is synonymous with **`char []`** as we have already learned before.

```
float
Get_Grade(struct studentTag List[], int nstudents,
          char param_lastname[],char param_firstname[]);
```

Study and understand the implementation of the **`Get_Grade()`** function. Several CCPROG2 concepts are involved here, specifically, arrays, strings, structures and files.

4. HOW TO TEST THE Get_Grade() FUNCTION

The **Get_Grade()** function can be tested by making actual function calls in the **main()**. For example:

```
float answer1, answer2;

answer1 = Get_Grade(List, nstudents, "CRUZ", "MARIA");   // Actual Question #1 above
answer2 = Get_Grade(List, nstudents, "WAYNE", "BRUCE");  // Actual Question #2 above
```

The values of **answer1** and **answer2** can then be printed via **printf()** to verify if **Get_Grade()** computed and returned the expected correct answers.

The problem with this way of testing is that the test values are static. If there's a need to change the parameter values, new function will need to be added to the source code, which needs to be re-compiled and re-run to see the effects.

The recommended way of testing is the one shown in the **main()** function, in particular, in the while loop. The relevant codes are copy/pasted below:

```
while ( scanf("%s %s", param_lastname, param_firstname) == 2) {
      answer = Get_Grade(List, nstudents, param_lastname, param_firstname);
      printf("%s %s %.1f\n", param_lastname, param_firstname, answer);
}
```

The function **Get_Grade()** is tested with function parameters that were input via **scanf()**. As long as the two parameters are supplied, the while loop will keep on executing.

To eliminate the manual time-consuming and error-prone keyboard input, it is recommended that the exe file be run in the command line with input redirection. More specifically,

```
D:\CCPROG2> SHAZAM < TEST-DATA.TXT
```

where **TEST-DATA.TXT** contains the values that will be read via **scanf()** stored to variables **param_lastname** and **param_firstname.**

Make sure that you try to do this in your computer. Can you make sense of what happened? Do you understand the results printed by the program?

Note that **scanf()** actually returns a value -- which means that is not a void function! The return value is a whole number that indicates how many values were actually read from the text file via input redirection. Since we are reading two values, then it must be compared to 2.

What's good about this approach? It avoids repetitive manual input via keyboard. More importantly, we don't need to add new source codes that will call **Get_Grades()** with different test values for **param_lastname** and **param_firstname.**

In other words, this is a better way of testing!

5. APPLICATION

Let's apply what you learned by accomplishing two exercises described below.

EXERCISE #1:

- Open the **TEST-DATA.TXT** file. Add a new line by typing a lastname and firstname that does not have any match in the data stored in **STUDENTS.TXT** file. Save the file. What result would you expect to see when **SHAZAM.exe** is executed again with input redirection as described above?

- Open the **TEST-DATA.TXT** file again, and add another new line by typing a lastname and firstname that matches one of the student records in **STUDENTS.TXT** file. Save the file. What result would you expect to see when **SHAZAM.exe** is executed again with input redirection as described above?

If you successfully accomplished this exercise, you just learned how to test both the positive and negative case. By positive, we mean a search that produces a hit, and by negative, we mean a search that does not produce a hit (or a match). These are only two possible results in the search algorithm used in **Get_Grades()** function.

EXERCISE #2:

- Copy **SHAZAM.c** into a new C source code file. You can use any filename that you want. Remove the function definition for **Get_Grade()** and the corresponding call inside **main()**.

- Implement a new function that answers the following question:

  Parametrized Question: How many students got a grade of **<param-grade>**?

  Example actual question: How many students got a grade of 4.0? Examine **STUDENTS.TXT** file manually and give your answer.

- Test your new function in the same way as **`Get_Grade()`**. You'll need to create a new text file that will contain the parameter values that you would like to use as test data.

- Compile, and the run your exe file with input redirection. Verify if the results of your function are the same with expected answers (obtained by visually examining the contents of **STUDENTS.TXT** file).

6. FINAL NOTE

It is imperative that you learn how to perform this kind of testing because this is the same way that we're going to test most of the functions that you will implement in your Machine Problem.

終