SOLUTION TO EXERCISE 1 – SEQUENTIAL ANALYSIS PROBLEM

The following documentation provides one solution for Analysis Exercise 1 about Sequential Processes. Note: All italicized text below serves only to clarify the documentation for students, but would not actually appear in the real documentation.

PROBLEM STATEMENT

This should define the program's task. The "problem" is not to write the program! It is to do whatever task the problem requires for its solution. This statement must be clear and complete enough for the program to be written entirely from just it and the sample outputs that follow it.

Request and store three test scores typed at the keyboard (allowing for floating point data) and determine and display the average test score rounded to one decimal place, given that each test is of equal weight.

DATA DEFINITION

In the sample output below:

- The numerals on the left of the sample output below are there for the analyst’s reference only and will not appear on the screen.
- The characters <CR> indicate the presence of a "carriage return" control code, causing the start of a new line of output.
- Items shown in [brackets] indicate values entered by the user (as distinguished from output produced by the program), but the brackets should not actually appear on the screen.
- The values below are examples which are valid for only one of many possible runs of the program.

SAMPLE OUTPUT (Softcopy):

```plaintext
1 Test Score Calculating Program<CR>
2 by Randy Gibson - 1 January 2012<CR>
3 <CR>
4 This program will request and store three test scores<CR>
5 (allowing for floating point data) and determine and<CR>
6 display the average test score rounded to one decimal<CR>
7 place, given that each test is of equal weight.<CR>
8 <CR>
9 Please enter the following data (decimal values are OK): <CR>
10 Test 1 Score: [999.9]<CR>
11 Test 2 Score: [999.9]<CR>
12 Test 3 Score: [999.9]<CR>
13 <CR>
14 Average test score: 999.9<CR>
```
SYMBOLIC CONSTANT LIST:

Symbolic constants are fixed values that may be referenced using a label at many different locations within a program's instructions rather than repeating the value many times. This practice makes it easy to update the program if the value ever needs to be changed. The constant identified below is typical of those found in most programs.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DESCRIPTION</th>
<th>DATA TYPE</th>
<th>VALUE</th>
<th>USAGE</th>
<th>DESTINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESTS</td>
<td>Quantity of tests</td>
<td>Integer</td>
<td>3</td>
<td>for AVG</td>
<td>---</td>
</tr>
</tbody>
</table>

VARIABLE LIST:

Variables are storage locations that hold values that will be different each time a program is run. Each variable is given a label that can be used by programmers to identify the storage location without having to know its numeric address in computer memory.

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>DESCRIPTION</th>
<th>DATA TYPE</th>
<th>SOURCE</th>
<th>USAGE</th>
<th>DESTINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Score on Test 1</td>
<td>Floating Point</td>
<td>Keyboard</td>
<td>for TOT</td>
<td>---</td>
</tr>
<tr>
<td>S2</td>
<td>Score on Test 2</td>
<td>Floating Point</td>
<td>Keyboard</td>
<td>for TOT</td>
<td>---</td>
</tr>
<tr>
<td>S3</td>
<td>Score on Test 3</td>
<td>Floating Point</td>
<td>Keyboard</td>
<td>for TOT</td>
<td>---</td>
</tr>
<tr>
<td>TOT</td>
<td>Sum of test scores</td>
<td>Floating Point</td>
<td>Calculation</td>
<td>for AVG</td>
<td>---</td>
</tr>
<tr>
<td>AVG</td>
<td>Average of test scores</td>
<td>Floating Point</td>
<td>Calculation</td>
<td>---</td>
<td>Screen</td>
</tr>
</tbody>
</table>

Remember that the description column clearly identifies each piece of data that must be stored including units of measure. The DATA TYPE column indicates the data type of the variable. The last three columns are used to indicate: (SOURCE) where the data comes from, (USAGE) what happens to it while it is stored, and (DESTINATION) where it will end up (for example, a screen, a printer, or disk storage). Notice that each variable has an entry in the SOURCE column (all data comes from somewhere) and each variable has an entry in one of the other two columns as well. Some variables have an entry in all three columns.

ALGORITHM

A. Start.
B. Output Intro. & Instructions as shown in softcopy.
   B1. Program Title on Line 1.
   B2. Program Credits (Author and Date) on Line 2.
   B5. Blank Line.
C. Request and store data as shown in softcopy on lines 9 - 12.
   C1. Display input instructions.
   C2. Display prompt for S1 (w/o carriage return).
   C3. Read keyboard entry and store it in S1, then display car. return.
   C4. Display prompt for S2 (w/o carriage return).
   C5. Read keyboard entry and store it in S2, then display car. return.
   C6. Display prompt for S3 (w/o carriage return).
   C7. Read keyboard entry and store it in S3, then display car. return.
D. Calculate and store interim and final answers.
   D1. Store TOT as S1+S2+S3.
   D2. Store AVG as TOT divided by TESTS.
E. Display answers on the screen as shown in the sample softcopy on lines 13 & 14.
      E2a. "Average test score: " (note trailing space without carriage return).
      E2b. AVG (rounded to one decimal place) followed by a carriage return.
F. End.

DESK CHECK

To perform a desk check, simply make up some sample input data to use as you read through and perform the steps in your algorithm on paper. Whenever your algorithm says to output something, write that on a blank piece of paper simulating either the monitor screen or printed paper output. Whenever your algorithm says to store a value, record that in a tracing chart similar to the one below so that you can easily keep track of all of the variables while performing your test. When you reach the end of your algorithm, the simulated output (softcopy or hardcopy) should exactly match the ones that you wrote as sample goals at the beginning of the analysis.

DATA TRACING CHART USING SAMPLE DATA:

The Data Tracing Chart is used to document what would be happening in the computer's memory during the execution of the steps described in your algorithm. A column is provided for each variable in your analysis with an additional column (#1) to serve as a reference to steps in your algorithm. Note in the chart below that it only relates to steps in the algorithm that effect the memory (ie. steps C & D).

<table>
<thead>
<tr>
<th>Step</th>
<th>Input</th>
<th>Calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>C3</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>C7</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td></td>
<td>270</td>
</tr>
<tr>
<td>D2</td>
<td></td>
<td>90</td>
</tr>
</tbody>
</table>
TEST OUTPUT (Softcopy):

The Test Output is produced by manually reading through the steps in your algorithm and recording (on paper) any output (softcopy or hardcopy) that your steps would produce. For example, given the test data used in the chart above, the algorithm on the following page would produce the following softcopy.

```
1 Test Score Calculating Program<CR>
2 by Randy Gibson - 1 January 2012<CR>
3 <CR>
4 This program will request and store three test scores<CR>
5 (allowing for floating point data) and determine and<CR>
6 display the average test score rounded to one decimal<CR>
7 place, given that each test is of equal weight.<CR>
8 <CR>
9 Please enter the following data (decimal values are OK): <CR>
10 Test 1 Score: [90]<CR>
11 Test 2 Score: [100]<CR>
12 Test 3 Score: [80]<CR>
13 <CR>
14 Average test score: 90.0<CR>
```

SOURCE CODE IN C++

Note: Analysis assignments do not require source code. It is included here simply to provide a complete example of a program's development.

```
/***************************************
* Test Averaging Program - testavg.cpp *
* Written by: Randolph U. Gibson       *
* Date: January 22, 2015               *
***************************************
*/
#include <iostream> // header file defining cout and cin
using namespace std; // defines context for cin and cout
#include <iomanip>  // header file defining setprecision and fixed
#define TESTS 3      // Symbolic constant for quantity of tests

int main ()
{

    float S1, S2, S3; // Individual test scores to be entered by user
    float TOT;        // Sum of all test scores
    float AVG;        // Average of all test scores

    /* Intro. & Instructions */
    cout << "Test Score Calculating Program\n";
    cout << "by Randolph U. Gibson - 1 January 2012\n";
    cout << "This program will request and store three test scores\n";
    cout << "(allowing for floating point data) and determine and\n";
    cout << "display the average test score rounded to one decimal\n";
    cout << "place, given that each test is of equal weight.\n";
```
/* Data Input Section */
cout << "Please enter the following data (decimal values are OK):
";
cout << "Test 1 Score: ";
cin >> S1; // cin automatically displays \n when done
cout << "Test 2 Score: ";
cin >> S2; // cin automatically displays \n when done
cout << "Test 3 Score: ";
cin >> S3; // cin automatically displays \n when done

/* Calculation Section */
TOT = S1 + S2 + S3;
AVG = TOT / TESTS;

/* Output Section */
cout << endl;
cout << setprecision(1) << fixed;
cout << "Average test score: " << AVG << endl;

return 0; // Send a null error code to the parent process
}