SENG1000 – C/C++ PROGRAMMING

FOCUSED ASSIGNMENT 6 - ARRAYS AND FUNCTIONS

OVERVIEW

Write a program to give you practice at using pointer variables to modify parameters.

GENERAL COURSE OBJECTIVES ADDRESSED IN THIS ASSIGNMENT

- Pass arrays as parameters.
- Use dereferencing and address-of operators.
- Use pointers to obtain multiple values from a called function.
- Use arithmetic operators.
- Following requirements for function interface definitions.

ACADEMIC INTEGRITY AND LATE PENALTIES

- Link to Academic Integrity Information
- Link to Late Policy

EVALUATION

- The evaluation of this assignment will be done as detailed in the Marking lecture from Week 2.

PREPARATION

- View Week 9 videos.

REQUIREMENTS

getDouble():

- Create a function that corresponds with the following function comment:

```
/*

Function: getDouble()

Parameter: double *pNumber: pointer to a variable that is
filled in by the user input, if valid
```

```
Return Value: int: 1 if the user entered a valid floating-
point number, 0 otherwise

Description: This function gets a floating-point number
from the user. If the user enters a valid floating-point
number, the value is put into the variable pointed to by
the parameter and 1 is returned.  If the user-entered value
is not valid, 0 is returned.

*/
```

- It would likely be easiest if you change the getNum() function to create this function.
- If you use sscanf(), you should be aware that the appropriate format string for getting a double from the user is "%lf" (that's a lowercase L) and not "%f" (unlike printf).

calculateStats():

- Create a function that corresponds with the following function comment:

```
/*

Function: calculateStats()

Parameters:    double d1, d2, d3, d4: four floating-point
numbers

               double *pAverage: pointer to a variable that
is filled in by this function with the average of d1, d2,
d3, and d4

               double *pSum: pointer to a variable that is
filled in by this function with the sum of d1, d2, d3, and
d4

Return Value: none

Description: This function takes four floating-point
numbers passed as doubles and calculates the average and
sum of the numbers.  Once calculated, the average gets put
into the variable pointed to by the pAverage parameter and
the sum gets put into the variable pointed to by the pSum
parameter.

*/
```

- This function should only have two lines in the body. pSum should be used in both lines (i.e. don't add the three numbers together twice).

---

calculateArrayStats():

- Create a function that corresponds with the following function comment:

```
/*

Function: calculateArrayStats()

Parameters:    double values[]: floating-point numbers

               int numArray: number of array elements

               double *pSum: pointer to a variable that is
filled in by this function with the sum of all elements in
the array

               double *pAverage: pointer to a variable that
is filled in by this function with the average of all
elements in the array

Return Value: none

Description: This function takes an array of floating-point
(double) numbers, given the number of elements in the array
as a parameter, and calculates the average and sum of the
numbers.  Once calculated, the average gets put into the
variable pointed to by the pAverage parameter and the sum
gets put into the variable pointed to by the pSum
parameter.

*/
```

   - This function is more flexible than calculateStats().

---

fillArray():

1. Create a function that corresponds with the following function comment:

```
/*

Function: fillArray()

Parameters:    double values[]: floating-point numbers
```

```
                int numArray: number of array elements

                double fillValue: value to put into array
elements

Return Value: none

Description: This function takes an array of floating-point
(double) numbers, given the number of elements in the array
as a parameter, and puts the fillValue into each element of
the array.

*/
```

main():

1. In main:
   a. Declare four double variables.
   b. Prompt the user and get those four floating-point numbers using the getDouble() function (one number per input line). If any of the floating-point numbers are invalid, quit the program (do not use exit()).
   c. Declare a double variable for the average and another double variable for the sum.
   d. Call calculateStats(), passing the six variables as appropriate.
   e. Display the average and the sum, preceded by “The average and sum of the variables:”.
   f. Declare an array of seven doubles.
   g. Prompt the user and fill in the array values using the getDouble() function (one number per input line). If any of the floating-point numbers are invalid, quit the program (do not use exit()).
   h. Call calculateArrayStats(), passing the array, average, and sum variables as appropriate.
   i. Display the average and the sum, preceded by “The average and sum of the array elements:” and with a comma between the average and sum.
   j. Call fillArray() to change all of the array element values to 40.
   k. On a single line, display all elements of the array, separated by commas. Make sure that the final number does not have a comma after it.
   l. End with return 0.

Make sure to prototype appropriately. Do not use global variables (**automatic mark of 0**).

Other Requirements:

- In calculateArrayStats() and fillArray(), do **not** use a fixed size for the array that is passed as a parameter. Both functions should work with any size of array.

- As usual, prompt the user for input (I shouldn't have to say this any more, though, since you should do this in all such cases anyway).
- Use best practices with respect to Magic Numbers.
- When passing an argument by reference, do **not** create a pointer variable solely to use as the argument. Pass the address of the variable that you want to pass by reference instead.
- Not using or creating all of the required functions will result in a **cap of 40** on your mark. The entire point of this mini-assignment is to work with the parameters of the functions. Having any deviations from the parameters for the functions will result in a **cap of 80** on your mark.
- Make sure that you use a header comment as described in the SET Submission Standards. Also, copy and paste the above function comments before the respective functions. No other comments are necessary.
- Do not clear the screen.
- Do not obtain user input or display output other than as explicitly required in these requirements.

CHECKLIST REQUIREMENTS

- Create a requirements checklist. This should contain the specific requirements from this assignment as well as any relevant requirements that have been covered in lecture or that are found in the SET Coding Standards or SET Submission Standards. Do it in whatever form you wish. Hand in your completed checklist in PDF form as checklist.pdf. Not having this checklist will result in a cap of 80 on your mark.

FILE NAMING REQUIREMENTS

- You must call your source file f6.cpp.
- You must call your checklist checklist.pdf.

SUBMISSION REQUIREMENTS

- Do not hand in any other source files besides those mentioned in the File Naming Requirements.
- Follow the instructions in the SET Submission Standards and the lecture on Submitting Assignments to submit your program. Submit both files to the correct Assignment folder.
- Once you have submitted your files, make sure that you've received the eConestoga e-mail confirming your submission. Do not submit that e-mail (simply keep it for your own records until you get your mark).