Assignment 4
C/C++ Programming II

C2A4 General Information

--- No General Information for This Assignment---

Get a Consolidated Assignment 4 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment, send an empty email to the assignment checker with the subject line **C2A4_ID**, where **ID** is your 9-character UCSD student ID. Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

1 <mark>C2A4E1 *(8 points – C++ Program)*</mark>

2 Exclude any existing source code files that may already be in your IDE project and add four new ones,
3 naming them **C2A4E1_ArraySize.h**, **C2A4E1_WorkerFunction.cpp**, **C2A4E1_RandomizeArray.cpp**, and
4 **C2A4E1_ComputeAverages.cpp**.   Also   add   instructor-supplied   source   code   file
5 **C2A4E1_main-Driver.cpp**. <u>Do not write a `main` function!</u> `main` already exists in the instructor-supplied
6 file and it will use the code you write.

7

8 File **C2A4E1_ArraySize.h** must contain the definitions of four **const int** variables that provide a
9 4-dimensional array's dimension sizes and a 5th **const int** variable that provides its total element count.
10 The dimension sizes, left-to-right, must be 10, 7, 6, and 8. You must include this file and use these
11 variables in any files that need this information. Do not calculate the total element count anywhere else
12 in your code.

13

14 File **C2A4E1_WorkerFunction.cpp** must contain a function named `WorkerFunction`.
15 `WorkerFunction` syntax:
16   **void** WorkerFunction(**float** *nestedAvg, **float** *linearAvg);
17 Parameters:
18   `nestedAvg` – a pointer to a type **float** object in the instructor-supplied driver file
19   `linearAvg` – a pointer to another type **float** object in the instructor-supplied driver file
20 Synopsis:
21   Does only the following three things, in order:
22   1. declares a local, automatic, type **float**, 4-dimensional array named `testArray` whose
23   dimension sizes are specified by the variables in file **C2A4E1_ArraySize.h**
24   2. makes the exact function call `RandomizeArray(testArray)`
25   3. makes the exact function call `ComputeAverages(testArray, nestedAvg, linearAvg)`
26 Return:
27   **void**

28

29 File **C2A4E1_RandomizeArray.cpp** must contain a function named `RandomizeArray`.
30 `RandomizeArray` syntax:
31   **void** RandomizeArray(*--One parameter; see the parameter description below--*)
32 Parameter:
33   You must design the appropriate parameter based upon the call to `RandomizeArray` shown in the
34   synopsis of `WorkerFunction` above. The parameter may <u>not</u> be a C++ "reference" type (note 5.9).
35 Synopsis:
36   1. seeds the random number generator with the value of the real time clock (RTC) using the
37   standard library **srand** and **time** functions;
38   2. initializes each element of the 4D array represented by the function's parameter with the
39   unaltered values returned from repeated calls to the library **rand** function; do not normalize or
40   restrict the range of those values in any way;
41 Return:
42   **void**

43

44 File **C2A4E1_ComputeAverages.cpp** must contain a function named `ComputeAverages`.
45 `ComputeAverages` syntax:
46   **void** ComputeAverages(*--Three parameters; see the parameter description below--*)
47 Parameters:
48   You must design the appropriate parameters based upon the call to `ComputeAverages` shown in
49   the synopsis of `WorkerFunction` above. The parameters may <u>not</u> be C++ "reference" types (note
50   5.9).
51 Synopsis:
52   1. computes the average of all elements in the 4D array represented by the function's 1st
53   parameter, accessing them in order using nested "for" loops and 4D indexing; the result is stored

in the address specified by the function's 2nd parameter. Ignore the potential for overflow or loss of precision when adding the element values.

2. computes the average of all elements in the 4D array represented by the function's 1st parameter, accessing them in order linearly using compact pointer operations; the result is stored in the address specified by the function's 3rd parameter. Ignore the potential for overflow or loss of precision when adding the element values.

Return:

void

Do <u>not</u> declare/create any arrays other than **testArray**.

Manually re-run your program several times; the average value should be different every time. If they're not something is wrong. Look up the description of the **rand** function and note the range of values it returns. Your average values should be approximately midway in this range.

Submitting your solution

Send all five source code files to the Assignment Checker with the subject line **C2A4E1_ID**, where **ID** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hints:

Use the decayed Right-Left rule to make sure you know the data type of the first parameter being passed to **RandomizeArray** and **ComputeAverages**. Beware of the pitfall discussed in note 13.19. If you place the call to the **srand** function inside the same loop that calls the **rand** function your results will be meaningless (Do you know why?).

1 **C2A4E2** *(6 points – C Program)*

2 Exclude any existing source code files that may already be in your IDE project and add a new one,
3 naming it **C2A4E2_StorageMap5D.h**. Also add instructor-supplied source code file
4 **C2A4E2_main-Driver.c**. <u>Do not write a **main** function!</u> **main** already exists in the instructor-supplied file
5 and it will use the code you write.

6

7 File **C2A4E2_StorageMap5D.h** must contain a macro named **StorageMap5D**.
8 **StorageMap5D** syntax (*never actually prototype a macro*):
9   **StorageMap5D(ptr, idx0, idx1, idx2, idx3, idx4, dim1, dim2, dim3, dim4)**
10 Parameters:
11   **ptr** – a pointer to the first element of a block of memory to be used as a 5-dimensional array
12   **idx0**, **idx1**, **idx2**, **idx3**, **idx4** – the indices of the desired element of the array
13   **dim1**, **dim2**, **dim3**, **dim4** – the rightmost 4 dimensions of the array (**dim0** isn't needed)
14 Synopsis:
15   implements the storage map equation for a 5-dimensional array of arbitrary type having arbitrary
16   dimension values. It may be used to access the elements of any existing 5-dimensional array of any
17   type and sufficient size, or in the general case may be used to access any arbitrary block of
18   memory of sufficient size as if it were a 5-dimensional array.
19 Value:
20   the element specified by the macro's 2nd through 6th arguments.

21

22

23 Example:
24   For an arbitrary array of any **type** originally declared as
25   **type** test[SZ_A][SZ_B][SZ_C][SZ_D][SZ_E];

26

27   or any block of dynamic memory allocated by
28   **type** *test =
29   (**type** *)malloc((SZ_A * SZ_B * SZ_C * SZ_D * SZ_E) * **sizeof**(**type**))

30

31   the expression
32   **StorageMap5D((type** *)test, 68, 73, 22, 58, 49, SZ_B, SZ_C, SZ_D, SZ_E)

33

34   would access the following element of the array or the dynamically-allocated memory block.
35   [68][73][22][58][49]

36

37

38 If you get any Assignment Checker errors/warnings regarding instructor-supplied file
39 **C2A4E2_main-Driver.c** it is because there is a problem in your macro.

40

41

42 **Submitting your solution**

43 Send both source code files to the Assignment Checker with the subject line **C2A4E2_ID**, where *ID* is your
44 9-character UCSD student ID.

45 *See the course document titled "Preparing and Submitting Your Assignments" for additional exercise*
46 *formatting, submission, and Assignment Checker requirements.*

47

48

49 **Hints:**
50 A typical procedure for testing any storage map equation is to first declare a standard array having the
51 same type and dimension sizes as the storage map equation and store a different value in each
52 element, then point the storage map equation to that array and verify that it accesses the expected
53 values. That is what the instructor-supplied source code file for this exercise does.

1   **C2A4E3** *(6 points – C++ Program)*

2   Exclude any existing source code files that may already be in your IDE project and add a new one,
3   naming it **C2A4E3_pointerArray4D.cpp**.   Also add instructor-supplied source code file
4   **C2A4E3_main-Driver.cpp**. <u>Do not write a **main** function!</u> **main** already exists in the instructor-supplied
5   file and it will use the code you write.

6
7   File **C2A4E3_pointerArray4D.cpp** must contain a 4-dimensional pointer array of type **float** named
8   **pointerArray4D** that you create using the method illustrated in notes 13.26 and 13.27. Specify the
9   dimension sizes by defining identifiers **DIM0**, **DIM1**, **DIM2**, and **DIM3** (left-to-right), which must have
10   values of **2**, **3**, **4**, and **5**, respectively. **pointerArray4D** must be accessible to any other file (e.g., the
11   instructor-supplied source code file) while the names of any other arrays must only be accessible within
12   file **C2A4E3_pointerArray4D.cpp** (note 5.14). Do <u>not</u> write any functions or macros or create any files
13   other than **C2A4E3_pointerArray4D.cpp**.

14
15
16   ## Submitting your solution

17   Send both source code files to the Assignment Checker with the subject line **C2A4E3_*ID***, where *ID* is your
18   9-character UCSD student ID.

19   *See the course document titled "Preparing and Submitting Your Assignments" for additional exercise*
20   *formatting, submission, and Assignment Checker requirements.*

21
22
23   **Hints:**
24   There is no quick and easy way to fully test a pointer array. As a minimal test a different value can be
25   stored into each element then read back to verify that no memory violations occur and that each
26   element contains its original value. However, this does not guarantee that more memory than
27   necessary hasn't been used or that an out-of-bounds access hasn't occurred.