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.

© 1992-2018 Ray Mitchell

C2A4E1 *(8 points – C++ Program)*

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

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

File **C2A4E1_WorkerFunction.cpp** must contain a function named WorkerFunction.
WorkerFunction syntax:

```
void WorkerFunction(float *nestedAvg, float *linearAvg);
```

Parameters:

nestedAvg – a pointer to a type **float** object in the instructor-supplied driver file
linearAvg – a pointer to another type **float** object in the instructor-supplied driver file

Synopsis:

Does only the following three things, in order:

1. declares a local, automatic, type **float**, 4-dimensional array named testArray whose dimension sizes are specified by the variables in file **C2A4E1_ArraySize.h**
2. makes the exact function call RandomizeArray(testArray)
3. makes the exact function call ComputeAverages(testArray, nestedAvg, linearAvg)

Return:

void

File **C2A4E1_RandomizeArray.cpp** must contain a function named RandomizeArray.
RandomizeArray syntax:

```
void RandomizeArray(--One parameter; see the parameter description below--)
```

Parameter:

You must design the appropriate parameter based upon the call to RandomizeArray shown in the synopsis of WorkerFunction above. The parameter may not be a C++ "reference" type (note 5.9).

Synopsis:

1. seeds the random number generator with the value of the real time clock (RTC) using the standard library srand and time functions;
2. initializes each element of the 4D array represented by the function's parameter with the unaltered values returned from repeated calls to the library rand function; do not normalize or restrict the range of those values in any way;

Return:

void

File **C2A4E1_ComputeAverages.cpp** must contain a function named ComputeAverages.
ComputeAverages syntax:

```
void ComputeAverages(--Three parameters; see the parameter description below--)
```

Parameters:

You must design the appropriate parameters based upon the call to ComputeAverages shown in the synopsis of WorkerFunction above. The parameters may not be C++ "reference" types (note 5.9).

Synopsis:

1. computes the average of all elements in the 4D array represented by the function's 1st parameter, accessing them in order using nested "for" loops and 4D indexing; the result is stored

© 1992-2018 Ray Mitchell

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 not 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?).

© 1992-2018 Ray Mitchell

C2A4E2 *(6 points – C Program)*

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

File **C2A4E2_StorageMap5D.h** must contain a macro named **`StorageMap5D`**.
`StorageMap5D` syntax (*never actually prototype a macro*):

```
StorageMap5D(ptr, idx0, idx1, idx2, idx3, idx4, dim1, dim2, dim3, dim4)
```

Parameters:

`ptr` – a pointer to the first element of a block of memory to be used as a 5-dimensional array
`idx0`, `idx1`, `idx2`, `idx3`, `idx4` – the indices of the desired element of the array
`dim1`, `dim2`, `dim3`, `dim4` – the rightmost 4 dimensions of the array (`dim0` isn't needed)

Synopsis:

implements the storage map equation for a 5-dimensional array of arbitrary type having arbitrary dimension values. It may be used to access the elements of any existing 5-dimensional array of any type and sufficient size, or in the general case may be used to access any arbitrary block of memory of sufficient size as if it were a 5-dimensional array.

Value:

the element specified by the macro's 2nd through 6th arguments.

Example:

For an arbitrary array of any **type** originally declared as

```
type test[SZ_A][SZ_B][SZ_C][SZ_D][SZ_E];
```

or any block of dynamic memory allocated by

```
type *test =
   (type *)malloc((SZ_A * SZ_B * SZ_C * SZ_D * SZ_E) * sizeof(type))
```

the expression

```
StorageMap5D((type *)test, 68, 73, 22, 58, 49, SZ_B, SZ_C, SZ_D, SZ_E)
```

would access the following element of the array or the dynamically-allocated memory block.

```
[68][73][22][58][49]
```

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

Submitting your solution

Send both source code files to the Assignment Checker with the subject line **C2A4E2_*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:

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

© 1992-2018 Ray Mitchell

C2A4E3 *(6 points – C++ Program)*

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

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

Submitting your solution

Send both source code files to the Assignment Checker with the subject line **C2A4E3_*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:

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

© 1992-2018 Ray Mitchell