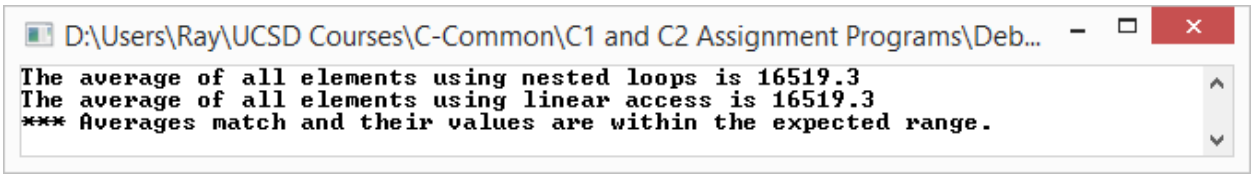
Exercise 1 *(8 points – C++ Program)*

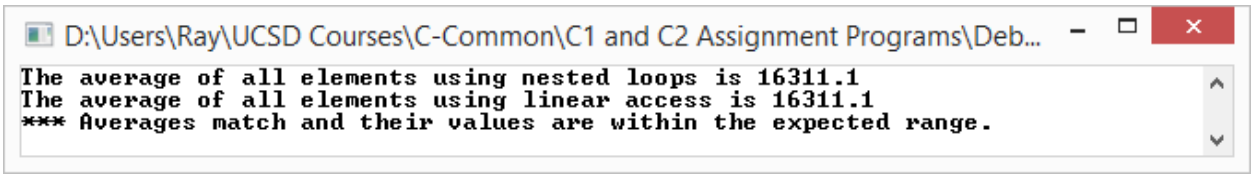
```
    ************************************* FILE C2A4E1_ArraySize.h *************************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains definitions of dimension sizes for a 4D array.
//

#ifndef C2A4E1_ARRAYSIZE_H
#define C2A4E1_ARRAYSIZE_H

// The dimension sizes of a 4D array.
const int DIM0 = 10, DIM1 = 7, DIM2 = 6, DIM3 = 8;
// The total number of elements in the 4D.
const int ELEMENTS = DIM0 * DIM1 * DIM2 * DIM3;

#endif


  ********************************** FILE C2A4E1_WorkerFunction.cpp **********************************
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains function WorkerFunction, which declares a 4D
// array and calls two other functions to process data in that array.
//

#include "C2A4E1_ArraySize.h"

void RandomizeArray(float (*dAp)[DIM1][DIM2][DIM3]);
void ComputeAverages(float (*dAp)[DIM1][DIM2][DIM3],
   float *nestedAvg, float *linearAvg);

//
// 1. Declare a 4D array of floats.
// 2. Store random values in all the array's elements.
// 3. Compute the average of the array elements using
//    both nested loop and linear array access and
//    store in the appropriate function parameters.
//
void WorkerFunction(float *nestedAvg, float *linearAvg)
{
   float testArray[DIM0][DIM1][DIM2][DIM3];
   RandomizeArray(testArray);
   ComputeAverages(testArray, nestedAvg, linearAvg);
}


  ---------------------------------------- EXERCISE CONTINUES ON NEXT PAGE ----------------------------------------
```

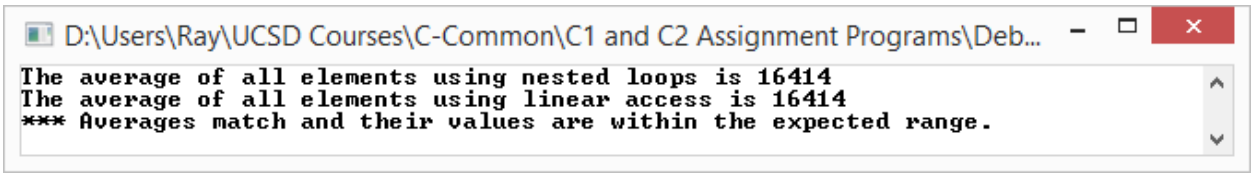
```
1
2     ********** FILE C2A4E1_RandomizeArray.cpp **********
3  //
4  // ...the usual title block Student/Course/Assignment/Compiler information goes here...
5  //
6  // This file contains function RandomizeArray, which stores random values into
7  // all elements of the 4D array represented by its parameter.
8  //
9
10 #include <cstdlib>
11 #include <ctime>
12 #include "C2A4E1_ArraySize.h"
13
14 //
15 // 1. Seed the random number generator with the real time clock.
16 // 2. Store random numbers in all array elements.
17 //
18 void RandomizeArray(float (*dAp)[DIM1][DIM2][DIM3])
19 {
20    // Seed random number generator.
21    std::srand((unsigned)std::time(0));
22    // Store random numbers in all array elements.
23    for (float *dp = (float *)dAp; dp < (float *)(dAp + DIM0);)
24       *dp++ = (float)std::rand();
25 }
26
27
28    ********** FILE C2A4E1_ComputeAverages.cpp **********
29
30 //
31 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
32 //
33 // This file contains function ComputeAverages, which computes the average of
34 // all values in the 4D array represented by its first parameter.
35 //
36
37 #include "C2A4E1_ArraySize.h"
38
39 //
40 // Calculate and display the average value of the elements in the 4D array
41 // in <dAP>.  This is done first by accessing the elements using indexing
42 // and nested for loops, then a second time by accessing the elements
43 // linearly.  If done correctly the average value will be exactly the same
44 // for each method of access.  The results of the two averaging techniques
45 // are stored in the addresses specified by parameters <nestedAvg> and
46 // <linearAvg>.
47 //
48 void ComputeAverages(float (*dAp)[DIM1][DIM2][DIM3],
49    float *nestedAvg, float *linearAvg)
50 {
51    // Nested loops method: Proceed from first to last element in order.
52    // The outermost loop controls the leftmost index and each successive
53    // inner loop controls the next index to the right.
54    *nestedAvg = 0;
55    for (int idx0 = 0; idx0 < DIM0; ++idx0)
56       for (int idx1 = 0; idx1 < DIM1; ++idx1)
57          for (int idx2 = 0; idx2 < DIM2; ++idx2)
58             for (int idx3 = 0; idx3 < DIM3; ++idx3)
```

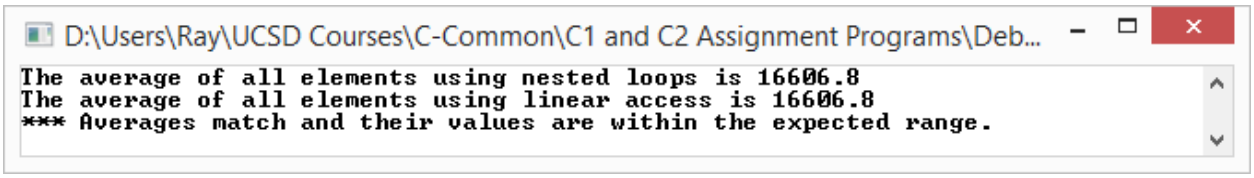
```
                *nestedAvg += dAp[idx0][idx1][idx2][idx3];
    *nestedAvg /= ELEMENTS;

    // Linear method: Proceed from first to last element in order.
    *linearAvg = 0;
    for (float *dp = (float *)dAp, *end = (float *)(dAp + DIM0); dp < end;)
       *linearAvg += *dp++;
    *linearAvg /= ELEMENTS;
}
```

C2A4E1 Screen Shots





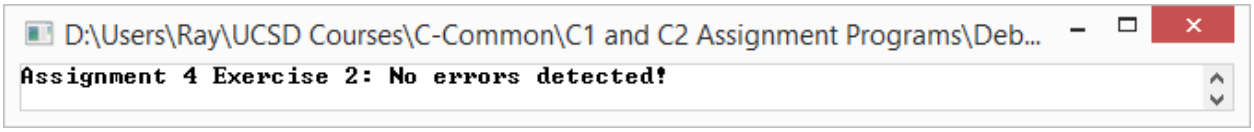




Exercise 2 *(6 points – C Program)*

```
/*
 * ...the usual title block Student/Course/Assignment/Compiler information goes here...
 *
 * This file contains macro StorageMap5D, which implements a 5-dimensional
 * storage map equation.
 */

#ifndef C2A4E2_STORAGEMAP5D_H
#define C2A4E2_STORAGEMAP5D_H

/*
 * Definition of a macro that implements a 5D storage map equation. It
 * will access the elements of a block of memory in the same way a true
 * 5-dimensional array accesses the elements of the block of memory it
 * represents.  The macro will work for any data type and any values of
 * the dimensions specified by <dim1> through <dim4>.  <dim0> need not
 * be specified.
 */

#define StorageMap5D(ptr,\
   idx0, idx1, idx2, idx3, idx4, dim1, dim2, dim3, dim4) \
   (*((ptr) + \
   (idx0) * (dim1) * (dim2) * (dim3) * (dim4) + \
   (idx1) * (dim2) * (dim3) * (dim4) + \
   (idx2) * (dim3) * (dim4) + \
   (idx3) * (dim4) + \
   (idx4)  ))

#endif
```

C2A4E2 Screen Shot



Exercise 3 *(6 points – C++ Program)*

```
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains code that creates a 4-dimensional pointer array of
// floats by declaring and initializing all necessary separate arrays.
//

//
// The following declarations and initializations create a pointer array
// named pointerArray4D.  This permits the elements of that array to be
// referenced using the same multidimensional syntax used when accessing
// a true 4-dimensional array having the same dimension sizes.
//
// Notes concerning declaration and testing:
//    * Since arrays cannot contain automatic initializers, all arrays
//      except pointerArray4D must be declared static (or must be external)
//      since they are used in the initialization lists of other arrays.
//    * Since there is no easy way to thoroughly test the 4D pointer array
//      a "reasonable test" might consist of:
//         1. using nested for loops to first write sequential values
//            into all elements, then,
//         2. using nested for loops again to go through all elements,
//            verifying those values.
//         3. This test does not test for "out of bounds" accesses!
//
extern const int DIM0 = 2, DIM1 = 3, DIM2 = 4, DIM3 = 5;

static float wa[DIM3], wb[DIM3], wc[DIM3], wd[DIM3], we[DIM3], wf[DIM3];
static float wg[DIM3], wh[DIM3], wi[DIM3], wj[DIM3], wk[DIM3], wl[DIM3];
static float wm[DIM3], wn[DIM3], wo[DIM3], wp[DIM3], wq[DIM3], wr[DIM3];
static float wss[DIM3], wt[DIM3], wu[DIM3], wv[DIM3], ww[DIM3], wx[DIM3];

static float *xa[DIM2] = {wa, wb, wc, wd}, *xb[DIM2] = {we, wf, wg, wh};
static float *xc[DIM2] = {wi, wj, wk, wl}, *xd[DIM2] = {wm, wn, wo, wp};
static float *xe[DIM2] = {wq, wr, wss, wt}, *xf[DIM2] = {wu, wv, ww, wx};

static float **yr[DIM1] = {xa, xb, xc}, **ys[DIM1] = {xd, xe, xf};

// pointerArray4D[ix0][ix1][ix2][ix3] generates:
//    *(*(*(*(pointerArray4D + ix0) + ix1) + ix2) + ix3)
float ***pointerArray4D[DIM0] = {yr, ys};
```

C2A4E3 Screen Shot