COP 3223: C Programming
Spring 2009

Arrays In C – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3223/spr2009/section1

School of Electrical Engineering and Computer Science
University of Central Florida

Multi-dimensional Arrays In C

- So far the arrays that we have seen have all been one-dimensional arrays. In other words they contained only a single value for the size of the array and thus every element in the array was referenced using a single index (or subscript) value.
- C allows for arrays to have multiple dimensions, i.e., multiple subscripts.
- Depending on the application and the type of data that needs to be represented, arrays of any dimension are possible, although 2-dimensional and 3-dimensional arrays are the most common variants.
- From a visual perspective, a 1-dimensional array appears like a list of values, a 2-dimensional array appears like a table of values, and a 3-dimensional array appears like a cube of values.

Multi-dimensional Arrays In C

1-dimensional array

```
int a[7];
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

2-dimensional array

```
int a[7][6];
```

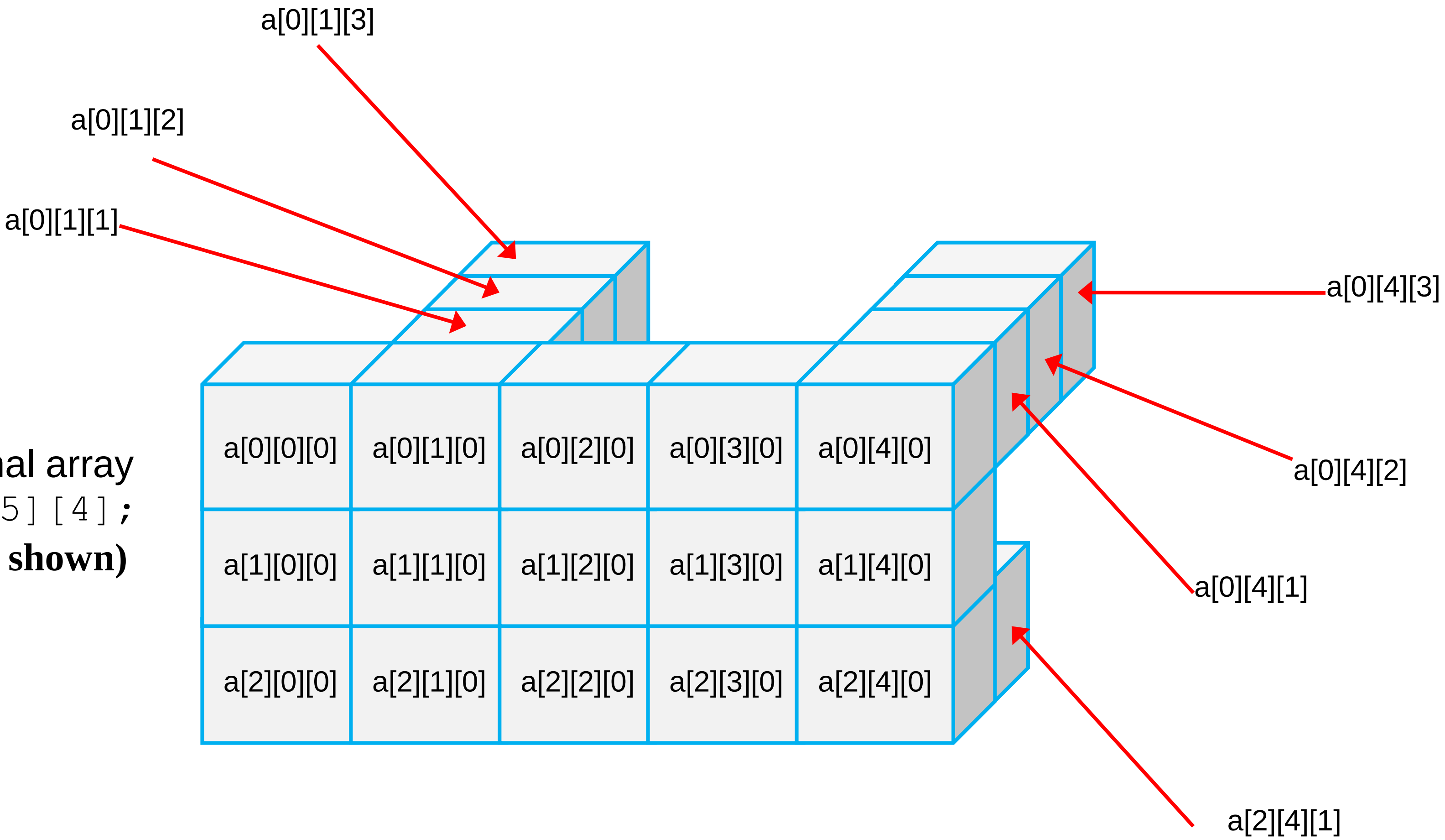
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]	a[3][5]
a[4][0]	a[4][1]	a[4][2]	a[4][3]	a[4][4]	a[4][5]
a[5][0]	a[5][1]	a[5][2]	a[5][3]	a[5][4]	a[5][5]
a[6][0]	a[6][1]	a[6][2]	a[6][3]	a[6][4]	a[6][5]

Multi-dimensional Arrays In C

3-dimensional array

`int a[3][5][4];`

(not all cells shown)



a[0][1][3]

a[0][1][2]

a[0][1][1]

a[0][4][3]

a[0][4][2]

a[0][4][1]

a[2][4][1]

a[0][0][0]	a[0][1][0]	a[0][2][0]	a[0][3][0]	a[0][4][0]
a[1][0][0]	a[1][1][0]	a[1][2][0]	a[1][3][0]	a[1][4][0]
a[2][0][0]	a[2][1][0]	a[2][2][0]	a[2][3][0]	a[2][4][0]

Declaring Multi-dimensional Arrays In C

- A 2-dimensional array is declared in C with 2 size (i.e., index or subscript) values each placed inside separate square brackets.

<u>Examples</u>

```
int a[4][4];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]
a[3][0]	a[3][1]	a[3][2]	a[3][3]

```
int matrix[2][3];
```

matrix[0][0]	matrix[0][1]	matrix[0][2]
matrix[1][0]	matrix[1][1]	matrix[1][2]

Declaring Multi-dimensional Arrays In C

COMMON PROGRAMMING MISTAKE

It is a very common mistake to reference a multiple-subscripted array using common mathematical notation, which is to separate the indices or subscripts with commas.

Thus, if we declare the following array in C: `int myArray[4][8];`

A reference to this array must look like `myArray[x][y]`
where $0 <= x < 4$ and $0 <= y < 8$

To reference this array as `myArray[x,y]` would be a syntax error.

Initializing Multi-dimensional Arrays In C

- C uses what is known as a row-major representation for a 2-d array. This basically means that the first dimension in an array definition refers to the number of rows in the array and the second dimension in the definition refers to the number of columns in the array.
- Thus, the definition `int anArray[2][4];` defines an array with 2 rows and 4 columns.
- Another way to think of this is that this definition defines two 1-dimensional arrays each with 4 locations (cells, indices, or subscripts). It does NOT allocate four 1-dimensional arrays each with 2 locations.
- Arrays are allocated contiguous memory space in row order.

Initializing Multi-dimensional Arrays In C

- The implications of the way C “views” multi-dimensional arrays can be utilized by programmers.

- For example, if we declare `int anArray[2][4];` then we can refer `anArray[1]`, which represents the entire second row of `anArray`, which is itself an array. Thus, `anArray[0]` and `anArray[1]` are defined.

- However, since C stores multi-dimensional arrays in row-major order, this means that there is no way to refer to an entire column of a 2-d array, since the values in the columns are not stored contiguously. Thus, `anArray[3]` in the above example, would be undefined since technically it would refer to a row that does not exist. In other words, it does not represent the third column of `anArray`.

Initializing Multi-dimensional Arrays In C

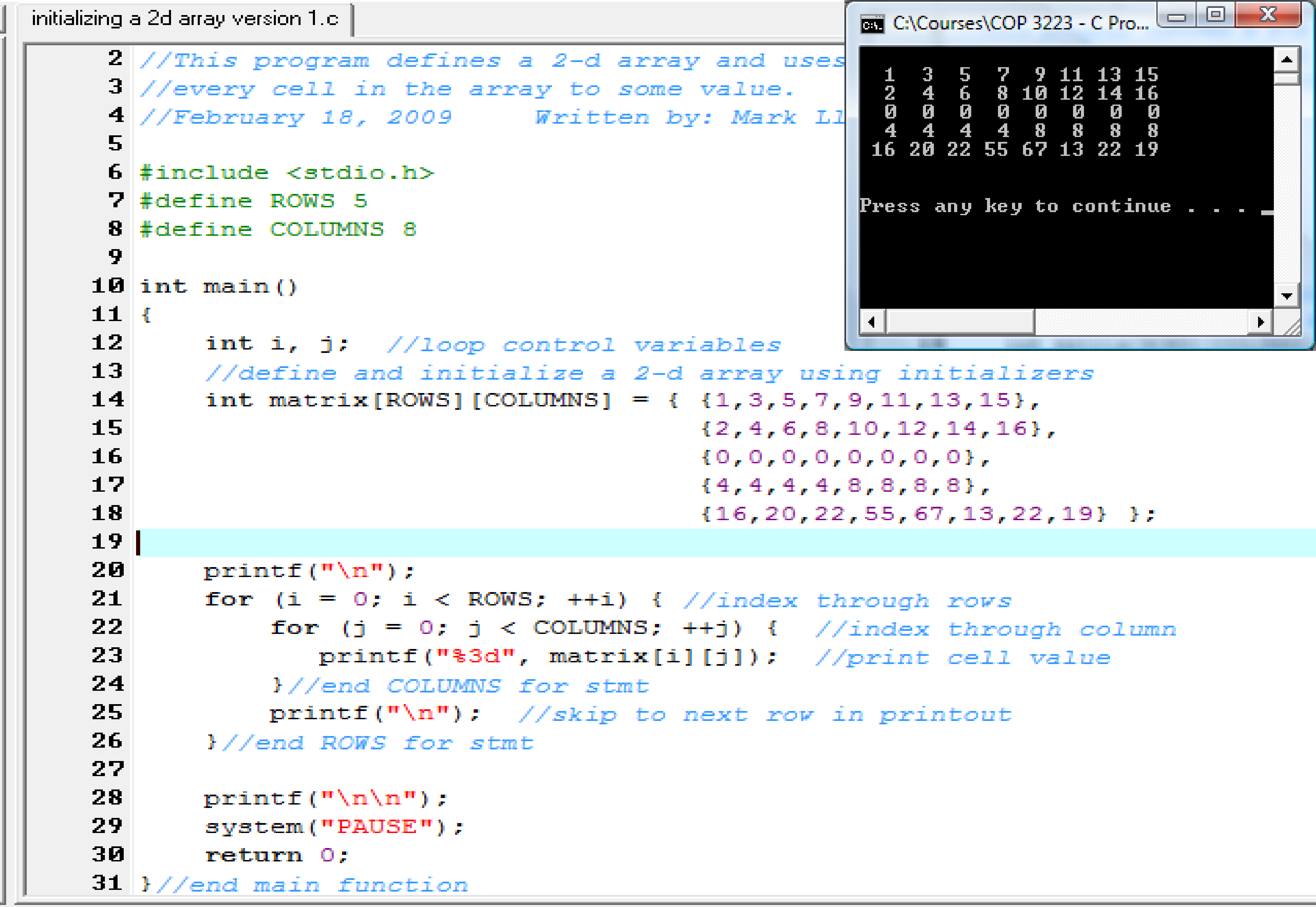
- As with 1-d arrays, there are techniques in C for initializing multi-dimensional arrays using initializers as well as loops.
- Using initializers with multi-dimensional arrays is very similar to the way it was done for 1-d arrays, however, more flexibility is available in the multi-dimensional case.
- Basically, using initializers for a multi-dimensional array means that you nest one-dimensional initializers.

<u>Example:</u>

```
int anArray[5] = {0,1,2,3,4};  //1-d case
int aMatrix[2][3] = { {1, 2, 3 },
                      {4, 5, 6 } }; //2-d case
```

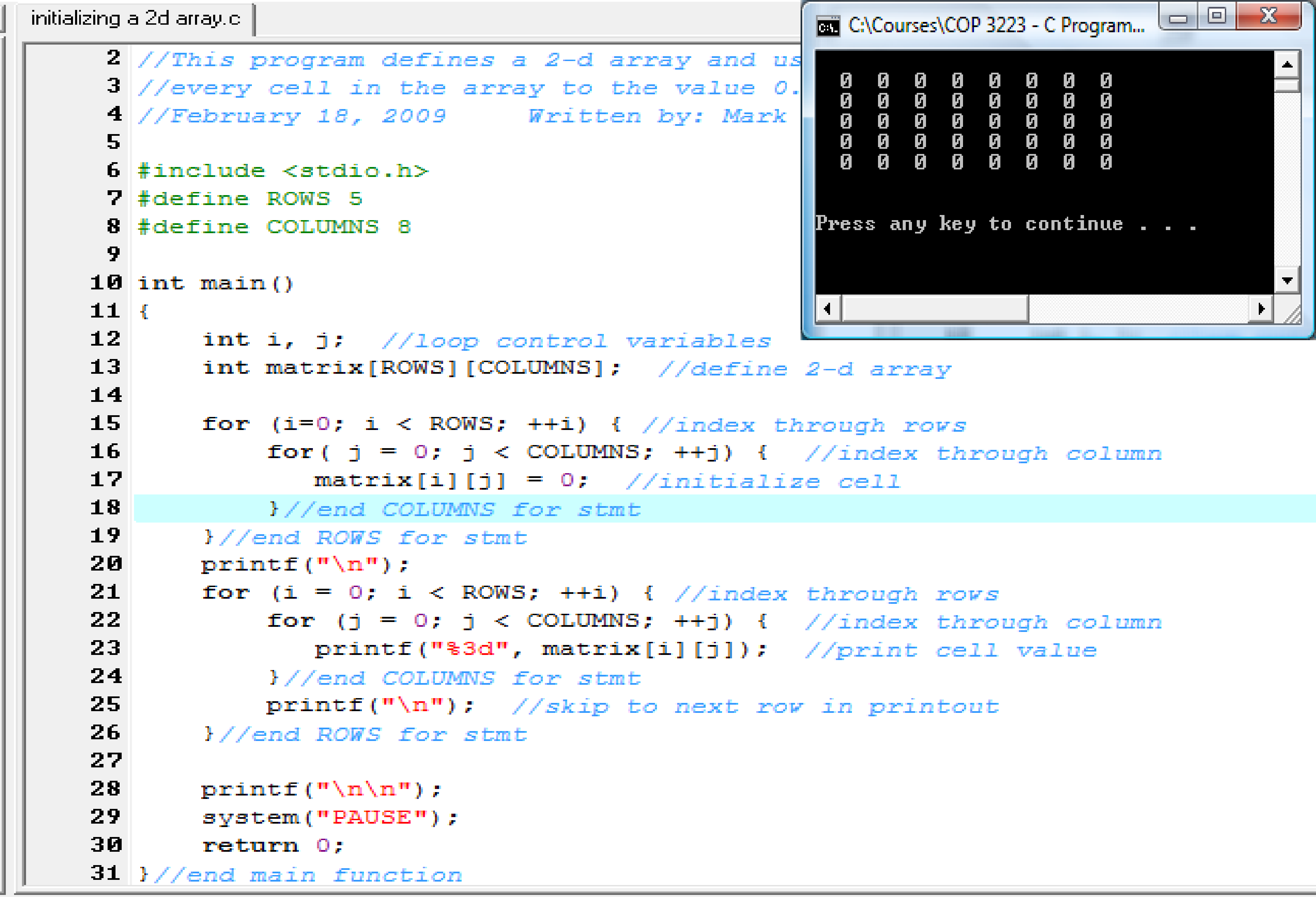
Initializing Multi-dimensional Arrays In C

- C provides a variety of ways to abbreviate initializers for multi-dimensional arrays. Generally speaking, it is easy to get yourself into trouble using some of these short-hand techniques, because missing braces or values, can produced unintended results. Therefore, we will not look at any of these techniques.
- The only other technique we will look at for initializing multi-dimensional arrays is the technique using loops similar to that we used for 1-d arrays. This is also the most common way to initialize multi-dimensional arrays.
- A single for loop is a common way to initialize a 1-d array in C. Nested loops go hand-in-hand with multi-dimensional arrays, so you want to get very familiar with how nested loops operate if this is still a little confusing to you.
- The next two pages are programs that illustrate both techniques for initializing a 2-d array.


initializing a 2d array version 1.c
C:\Courses\COP 3223 - C Pro...

```c
//This program defines a 2-d array and uses
//every cell in the array to some value.
//February 18, 2009     Written by: Mark Ll

#include <stdio.h>
#define ROWS 5
#define COLUMNS 8

int main()
{
    int i, j;  //loop control variables
    //define and initialize a 2-d array using initializers
    int matrix[ROWS][COLUMNS] = { {1,3,5,7,9,11,13,15},
                                  {2,4,6,8,10,12,14,16},
                                  {0,0,0,0,0,0,0,0},
                                  {4,4,4,4,8,8,8,8},
                                  {16,20,22,55,67,13,22,19} };

    printf("\n");
    for (i = 0; i < ROWS; ++i) { //index through rows
        for (j = 0; j < COLUMNS; ++j) {  //index through column
           printf("%3d", matrix[i][j]);  //print cell value
        }//end COLUMNS for stmt
        printf("\n");  //skip to next row in printout
    }//end ROWS for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

```

  1  3  5  7  9 11 13 15
  2  4  6  8 10 12 14 16
  0  0  0  0  0  0  0  0
  4  4  4  4  8  8  8  8
 16 20 22 55 67 13 22 19


Press any key to continue . . .
```

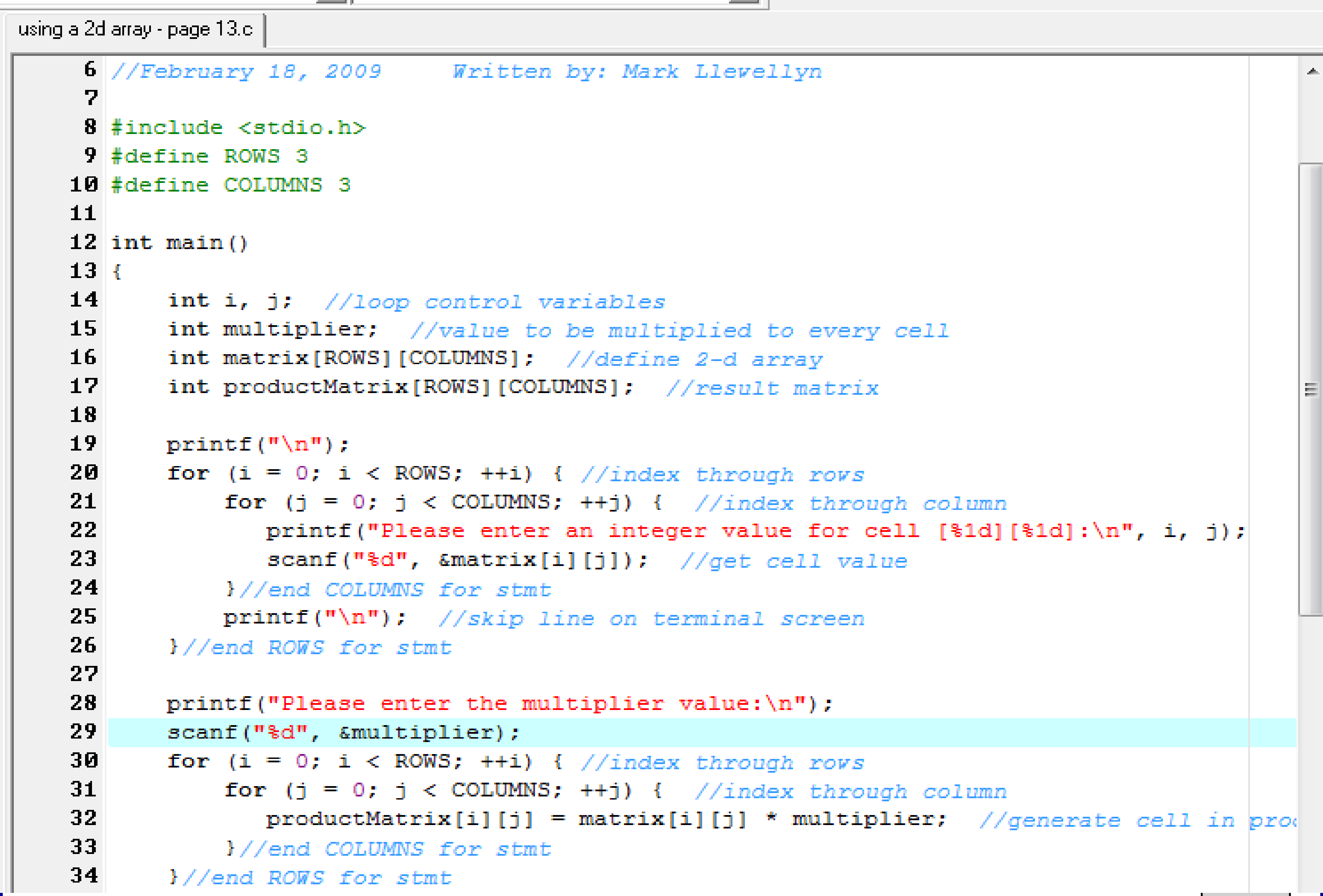


```c
//This program defines a 2-d array and us
//every cell in the array to the value 0.
//February 18, 2009     Written by: Mark

#include <stdio.h>
#define ROWS 5
#define COLUMNS 8

int main()
{
    int i, j;  //loop control variables
    int matrix[ROWS][COLUMNS];  //define 2-d array

    for (i=0; i < ROWS; ++i) { //index through rows
        for( j = 0; j < COLUMNS; ++j) {  //index through column
           matrix[i][j] = 0;  //initialize cell
        }//end COLUMNS for stmt
    }//end ROWS for stmt
    printf("\n");
    for (i = 0; i < ROWS; ++i) { //index through rows
        for (j = 0; j < COLUMNS; ++j) {  //index through column
           printf("%3d", matrix[i][j]);  //print cell value
        }//end COLUMNS for stmt
        printf("\n");  //skip to next row in printout
    }//end ROWS for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

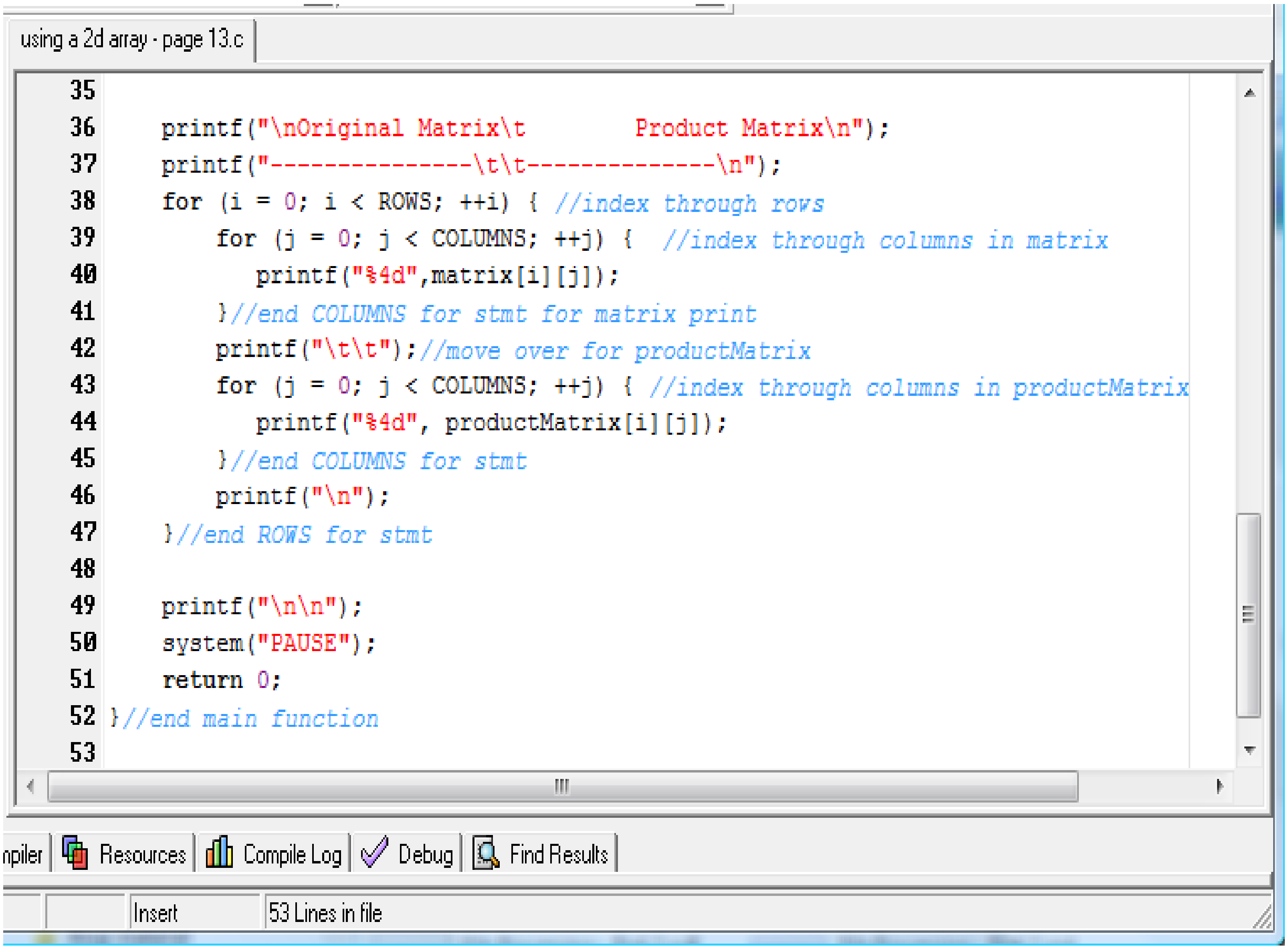
```

  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0


Press any key to continue . . .
```

Using Multi-dimensional Arrays In C

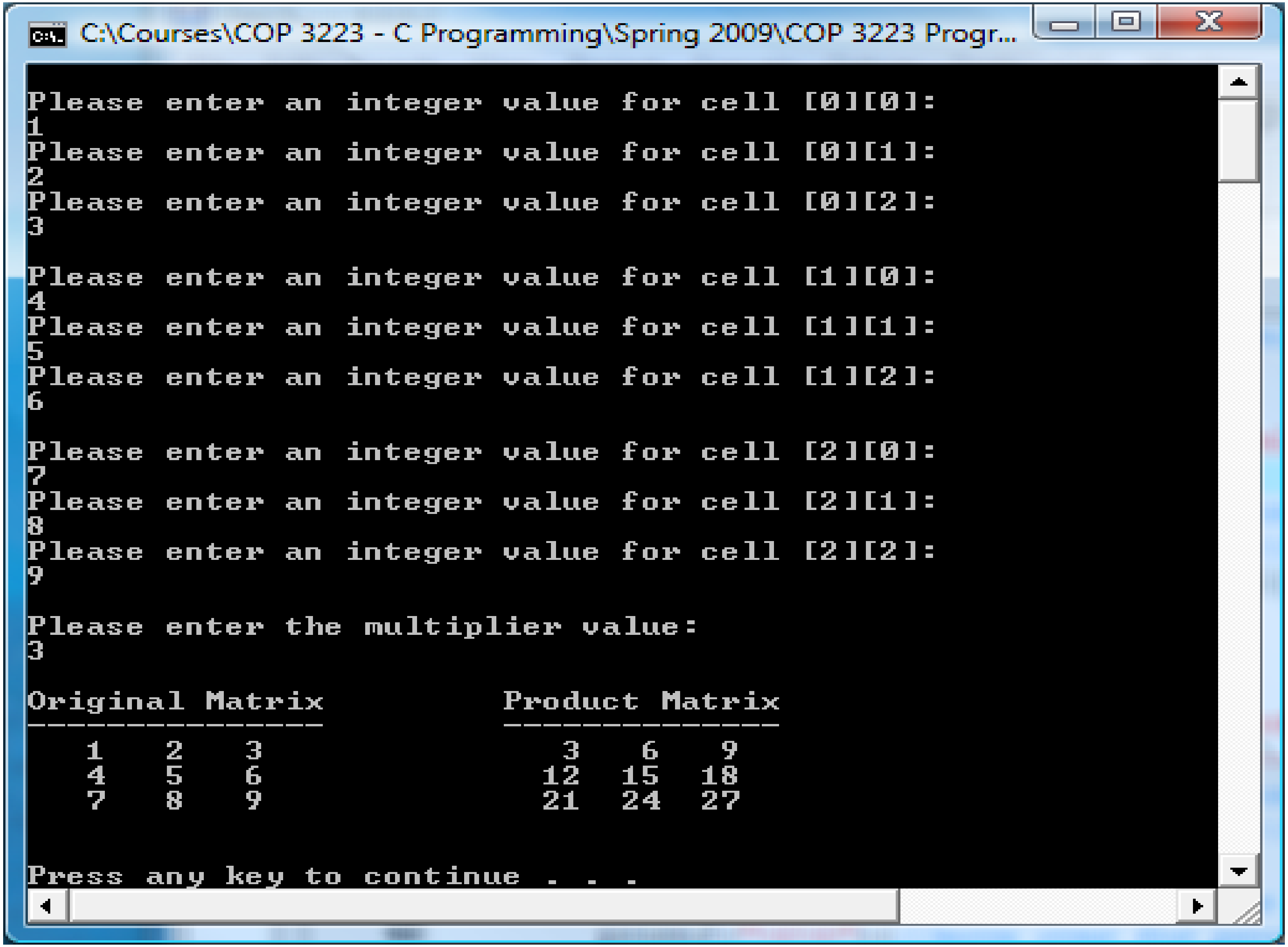
- Let's look at a series of examples that illustrate using multi-dimensional arrays in C.
- The next example uses a 2-d array and has the user enter the values into the array in a row by row fashion. Then once the values are entered into the array, the program will ask the user to enter a single integer value. Then every cell in the array will be multiplied by this value and saved in the array (this is scalar matrix multiplication). Finally, the original array and the "multiplied" array will be printed side by side.
- We'll assume the array will be a 3x3 two-dimensional array.
- Notice the way that I nested two `for` statements independently inside another `for` statement so that I could print out the two matrices side-by-side.



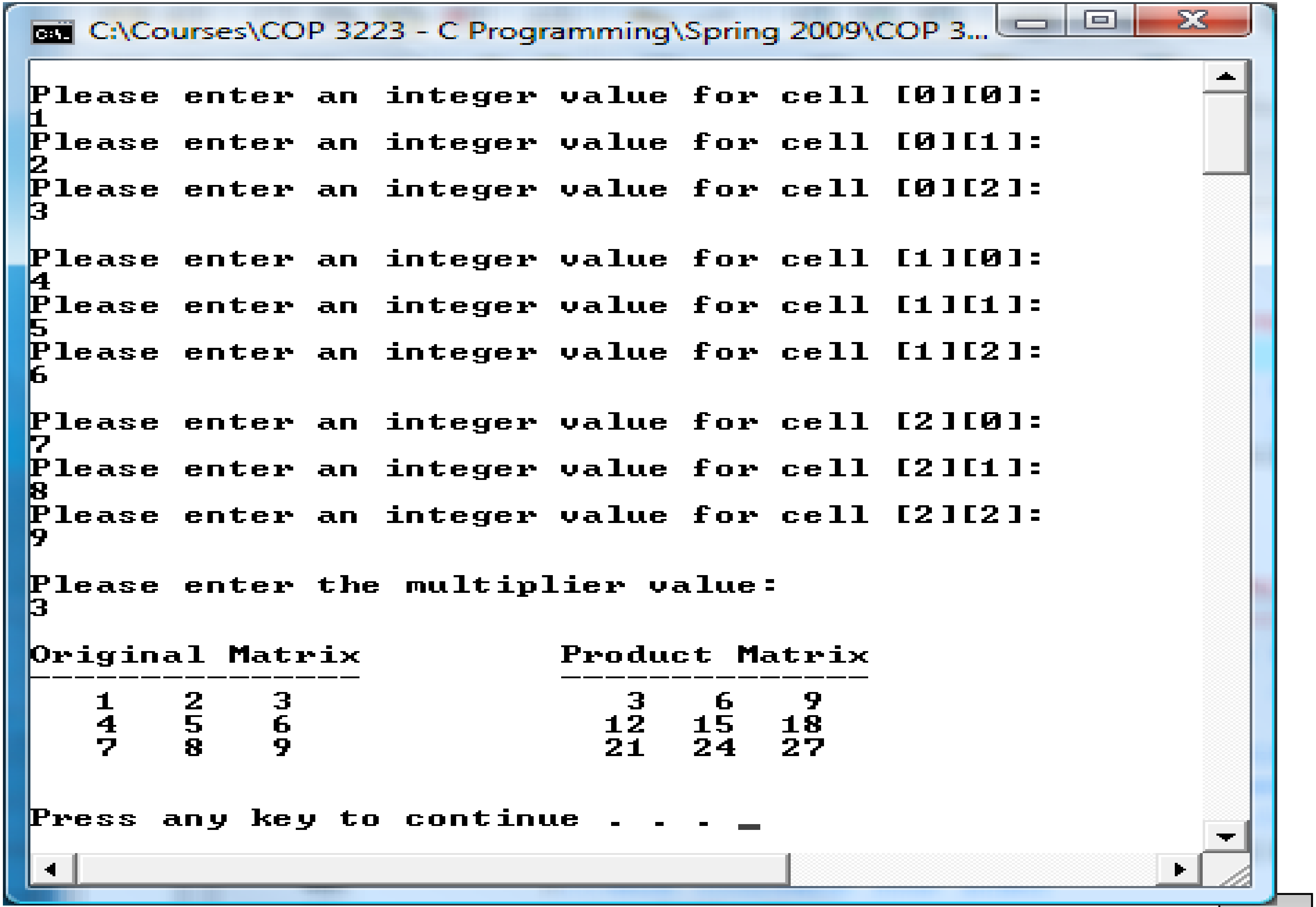
```c
//February 18, 2009     Written by: Mark Llewellyn

#include <stdio.h>
#define ROWS 3
#define COLUMNS 3

int main()
{
    int i, j;  //loop control variables
    int multiplier;  //value to be multiplied to every cell
    int matrix[ROWS][COLUMNS];  //define 2-d array
    int productMatrix[ROWS][COLUMNS];  //result matrix

    printf("\n");
    for (i = 0; i < ROWS; ++i) { //index through rows
        for (j = 0; j < COLUMNS; ++j) {  //index through column
           printf("Please enter an integer value for cell [%1d][%1d]:\n", i, j);
           scanf("%d", &matrix[i][j]);  //get cell value
        }//end COLUMNS for stmt
        printf("\n");  //skip line on terminal screen
    }//end ROWS for stmt

    printf("Please enter the multiplier value:\n");
    scanf("%d", &multiplier);
    for (i = 0; i < ROWS; ++i) { //index through rows
        for (j = 0; j < COLUMNS; ++j) {  //index through column
           productMatrix[i][j] = matrix[i][j] * multiplier;  //generate cell in pro
        }//end COLUMNS for stmt
    }//end ROWS for stmt
```



```c

    printf("\nOriginal Matrix\t        Product Matrix\n");
    printf("---------------\t\t--------------\n");
    for (i = 0; i < ROWS; ++i) { //index through rows
        for (j = 0; j < COLUMNS; ++j) {  //index through columns in matrix
           printf("%4d",matrix[i][j]);
        }//end COLUMNS for stmt for matrix print
        printf("\t\t");//move over for productMatrix
        for (j = 0; j < COLUMNS; ++j) { //index through columns in productMatrix
           printf("%4d", productMatrix[i][j]);
        }//end COLUMNS for stmt
        printf("\n");
    }//end ROWS for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function

```


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Progr...

```
Please enter an integer value for cell [0][0]:
1
Please enter an integer value for cell [0][1]:
2
Please enter an integer value for cell [0][2]:
3

Please enter an integer value for cell [1][0]:
4
Please enter an integer value for cell [1][1]:
5
Please enter an integer value for cell [1][2]:
6

Please enter an integer value for cell [2][0]:
7
Please enter an integer value for cell [2][1]:
8
Please enter an integer value for cell [2][2]:
9

Please enter the multiplier value:
3

Original Matrix         Product Matrix
---------------         --------------
   1   2   3               3   6   9
   4   5   6              12  15  18
   7   8   9              21  24  27


Press any key to continue . . .
```



```
Please enter an integer value for cell [0][0]:
1
Please enter an integer value for cell [0][1]:
2
Please enter an integer value for cell [0][2]:
3

Please enter an integer value for cell [1][0]:
4
Please enter an integer value for cell [1][1]:
5
Please enter an integer value for cell [1][2]:
6

Please enter an integer value for cell [2][0]:
7
Please enter an integer value for cell [2][1]:
8
Please enter an integer value for cell [2][2]:
9

Please enter the multiplier value:
3

Original Matrix         Product Matrix
---------------         --------------
   1   2   3               3   6   9
   4   5   6              12  15  18
   7   8   9              21  24  27


Press any key to continue . . . _
```

Using Multi-dimensional Arrays In C

- For our next example, let’s look at the problem of adding two matrices (2-d arrays) together.

- There are many different matrix operations and they have an extremely wide variety of applications. Not every matrix operation is defined for all matrices. For example, if A and B are two matrices, then matrix addition is only defined if the number of columns in matrix A is equal to the number of rows in matrix B. The resulting matrix C will then have the number of rows of A and the number of columns of B.

  – Thus, if we have A[3][2] and B[2][4], then matrix addition of A and B is defined and the result is C[3][4].

  – If we have A[4][3] and B[4][2], then matrix addition is undefined for A and B.

Using Multi-dimensional Arrays In C

Matrix Addition Example:

generic case

$$\begin{bmatrix} a\,b\,c \\ d\;e\;f \end{bmatrix} + \begin{bmatrix} g\;h \\ i\;j \\ k\;l \end{bmatrix} = \begin{bmatrix} (a+g)+(b+i)+(c+k) & (a+h)+(b+j)+(c+l) \\ (d+g)+(e+i)+(f+k) & (d+h)+(e+j)+(f+l) \end{bmatrix}$$

specific case

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} (1+7)+(2+9)+(3+11) & (1+8)+(2+10)+(3+12) \\ (4+7)+(5+9)+(6+11) & (4+8)+(5+10)+(6+12) \end{bmatrix} = \begin{bmatrix} 33 & 36 \\ 42 & 45 \end{bmatrix}$$

Using Multi-dimensional Arrays In C

Matrix Addition Example:

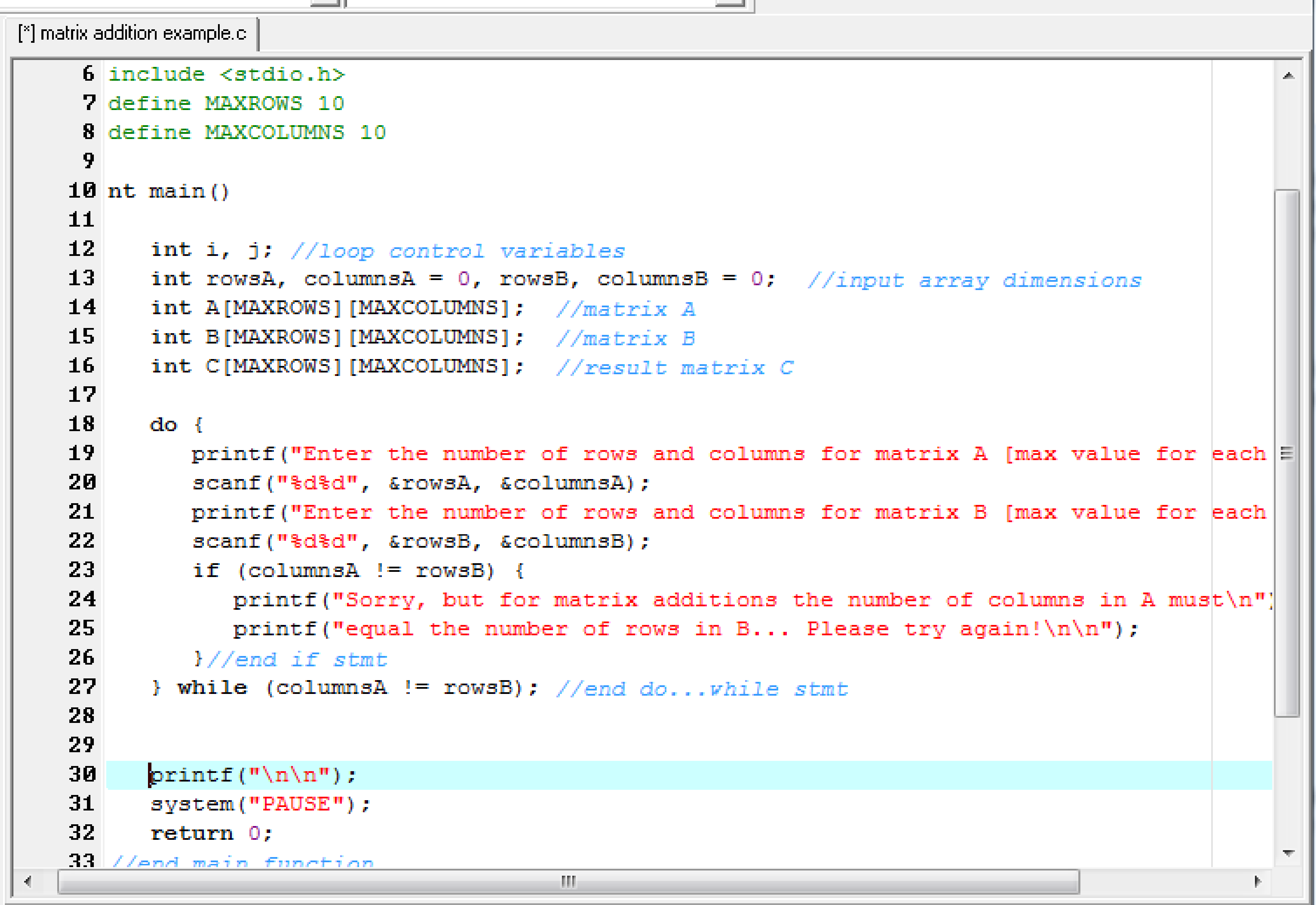
Most probable outcome of rolling two dice

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 9 & 10 & 11 & 12 \end{bmatrix}$$

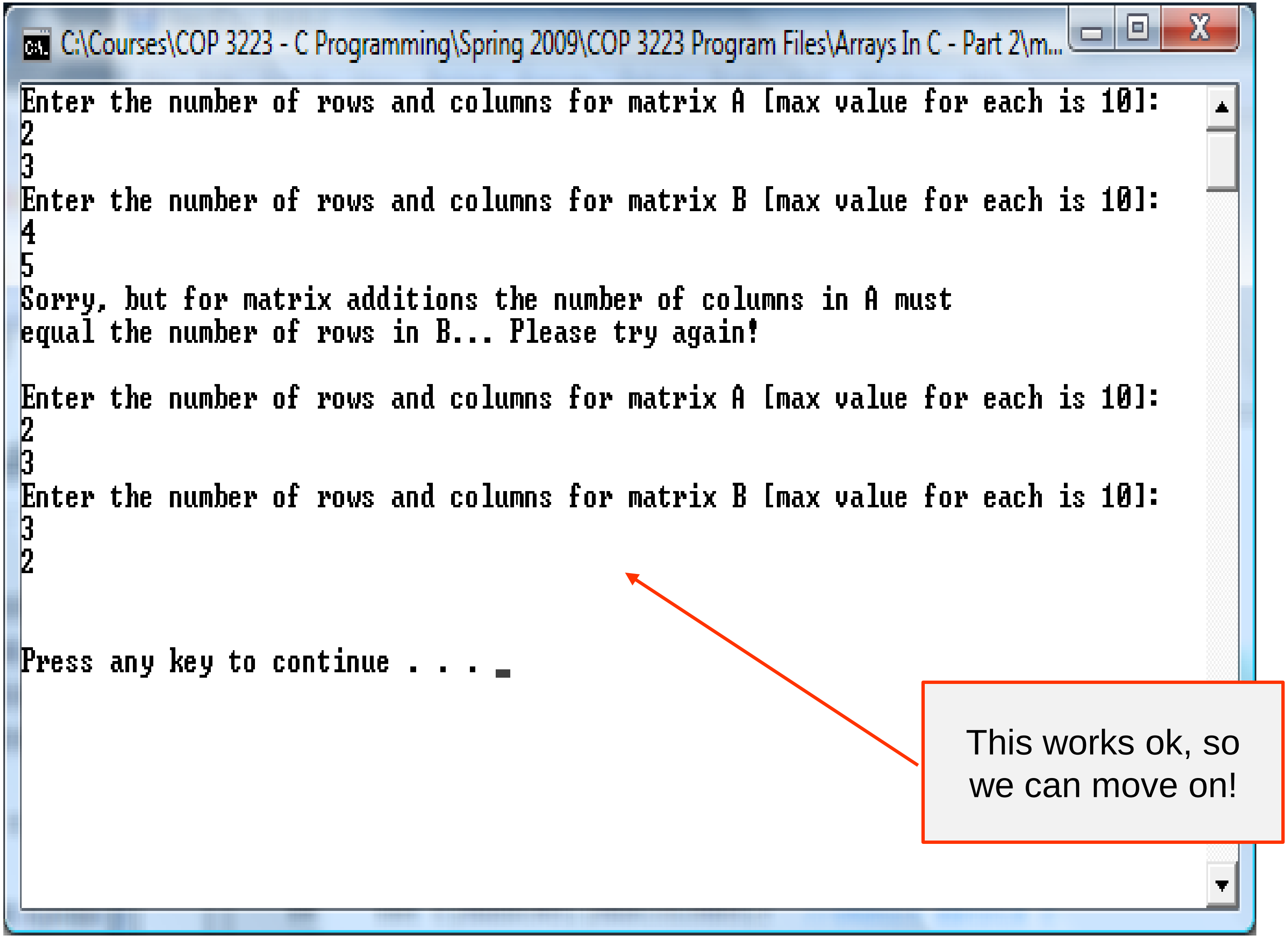
Notice that there are more 7's than any other number so you have the highest probability of rolling a total of 7, the next highest probability is rolling a total of either 6 or 8, and so on.

Using Multi-dimensional Arrays In C

- I want to do something else with this example that I think many of you are missing or jumping over when you work on your programming assignments.
- This is the concept of step-wise or modular development and refinement. More simply put, work on one task at a time…get it done right…then move on to the next task.
- In this example, let's first write the portion of the program that will have the user enter the sizes of the two arrays and determine if the arrays are compatible for matrix addition.
- This will help us in several ways: (1) we can get this part of the program working correctly before we ever even consider how to add the two matrices together, (2) there isn't any point in reading in the values for the two arrays if they aren't compatible for matrix addition!, (3) it will help you begin to think in terms of functions.



```
include <stdio.h>
define MAXROWS 10
define MAXCOLUMNS 10

nt main()

   int i, j; //loop control variables
   int rowsA, columnsA = 0, rowsB, columnsB = 0;  //input array dimensions
   int A[MAXROWS][MAXCOLUMNS];  //matrix A
   int B[MAXROWS][MAXCOLUMNS];  //matrix B
   int C[MAXROWS][MAXCOLUMNS];  //result matrix C

   do {
      printf("Enter the number of rows and columns for matrix A [max value for each
      scanf("%d%d", &rowsA, &columnsA);
      printf("Enter the number of rows and columns for matrix B [max value for each
      scanf("%d%d", &rowsB, &columnsB);
      if (columnsA != rowsB) {
         printf("Sorry, but for matrix additions the number of columns in A must\n"
         printf("equal the number of rows in B... Please try again!\n\n");
      }//end if stmt
   } while (columnsA != rowsB); //end do...while stmt


   printf("\n\n");
   system("PAUSE");
   return 0;
//end main function
```



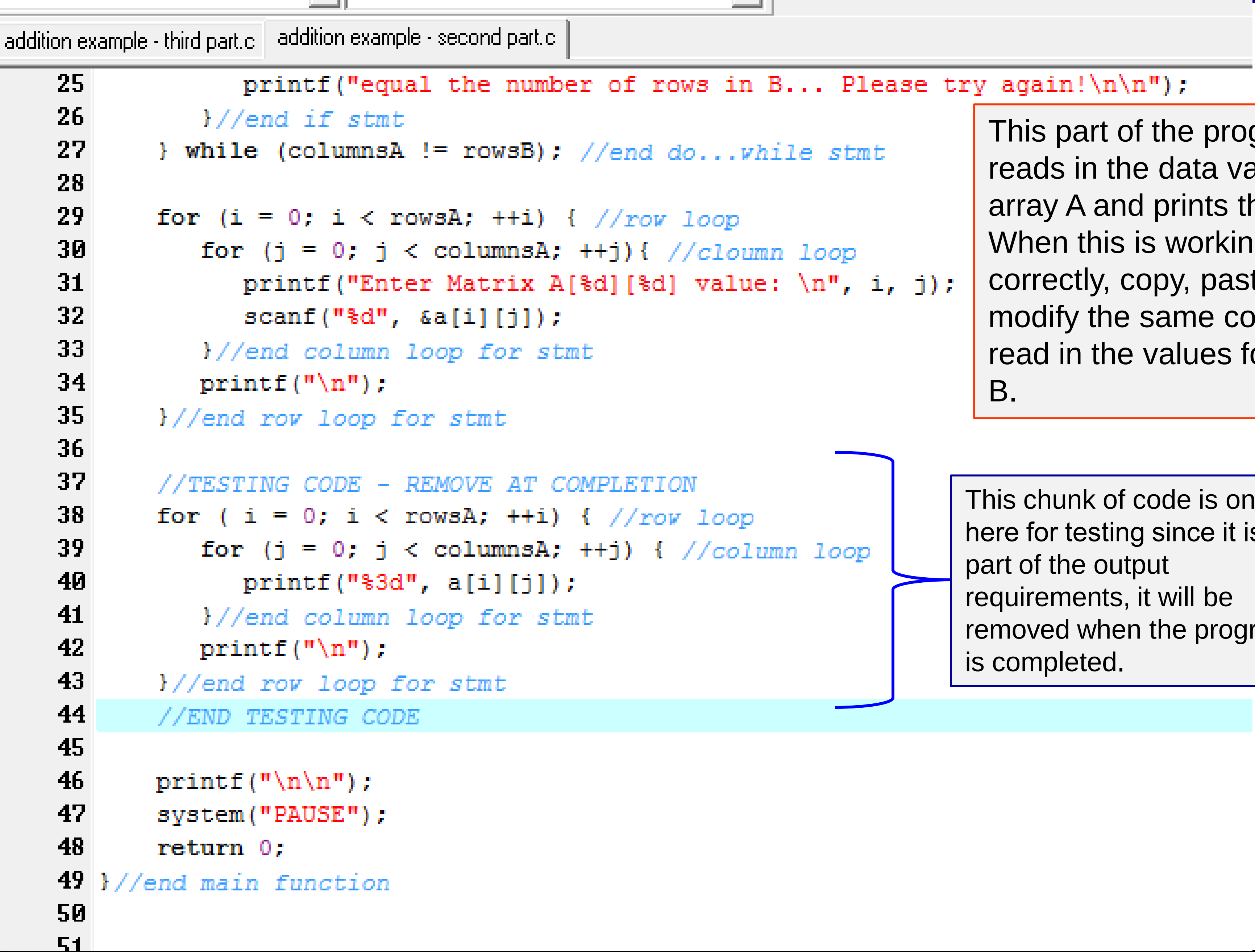
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Arrays In C - Part 2\m...

```
Enter the number of rows and columns for matrix A [max value for each is 10]:
2
3
Enter the number of rows and columns for matrix B [max value for each is 10]:
4
5
Sorry, but for matrix additions the number of columns in A must
equal the number of rows in B... Please try again!

Enter the number of rows and columns for matrix A [max value for each is 10]:
2
3
Enter the number of rows and columns for matrix B [max value for each is 10]:
3
2



Press any key to continue . . . _
```

This works ok, so we can move on!

Using Multi-dimensional Arrays In C

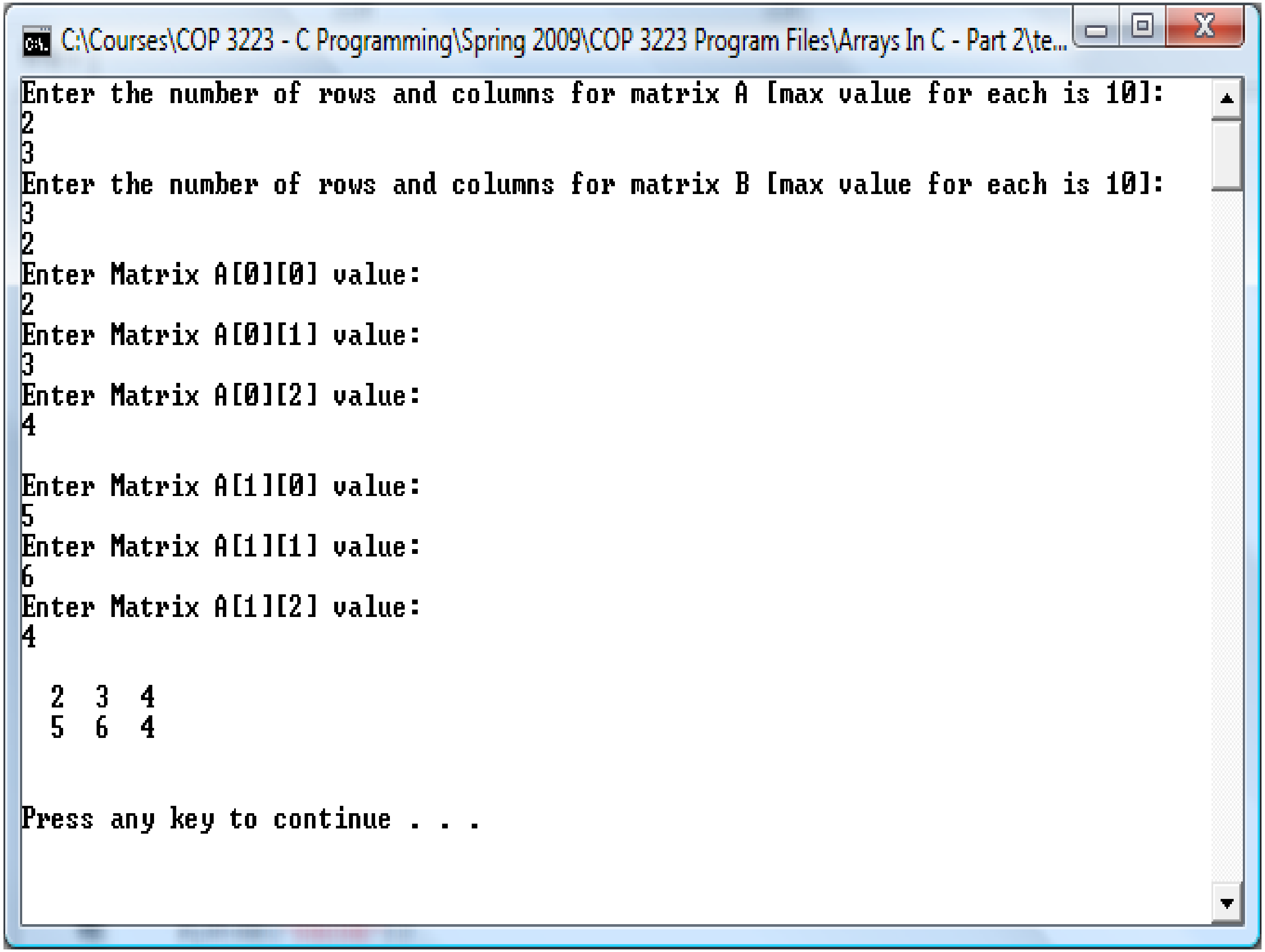
- Now that we have getting the array dimensions entered and verified as compatible for matrix addition we can move on to handling the input of the data for the two matrices.
- For this type of operation, when developing your code, I would suggest printing out the arrays after they have been input, even though it is not part of the problem, just so that you can verify that the data went into the arrays in the correct locations.
- Once you've done this, simply comment out the code used to print the arrays. I wouldn't erase it, you might want to use it again somewhere else in the program. When everything is done and tested, if it is no longer needed, then remove it.
- Write the code to enter the data for one array first and get this working correctly. Only when this is working correctly do you add the code to read the data into the second array. Then it is a simple copy, paste, and modify operation.


addition example - third part.c
addition example - second part.c

```
          printf("equal the number of rows in B... Please try again!\n\n");
       }//end if stmt
    } while (columnsA != rowsB); //end do...while stmt

    for (i = 0; i < rowsA; ++i) { //row loop
       for (j = 0; j < columnsA; ++j){ //cloumn loop
          printf("Enter Matrix A[%d][%d] value: \n", i, j);
          scanf("%d", &a[i][j]);
       }//end column loop for stmt
       printf("\n");
    }//end row loop for stmt

    //TESTING CODE - REMOVE AT COMPLETION
    for ( i = 0; i < rowsA; ++i) { //row loop
       for (j = 0; j < columnsA; ++j) { //column loop
          printf("%3d", a[i][j]);
       }//end column loop for stmt
       printf("\n");
    }//end row loop for stmt
    //END TESTING CODE

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

This part of the program reads in the data values for array A and prints them out. When this is working correctly, copy, paste, and modify the same code to read in the values for array B.

This chunk of code is only here for testing since it isn't part of the output requirements, it will be removed when the program is completed.

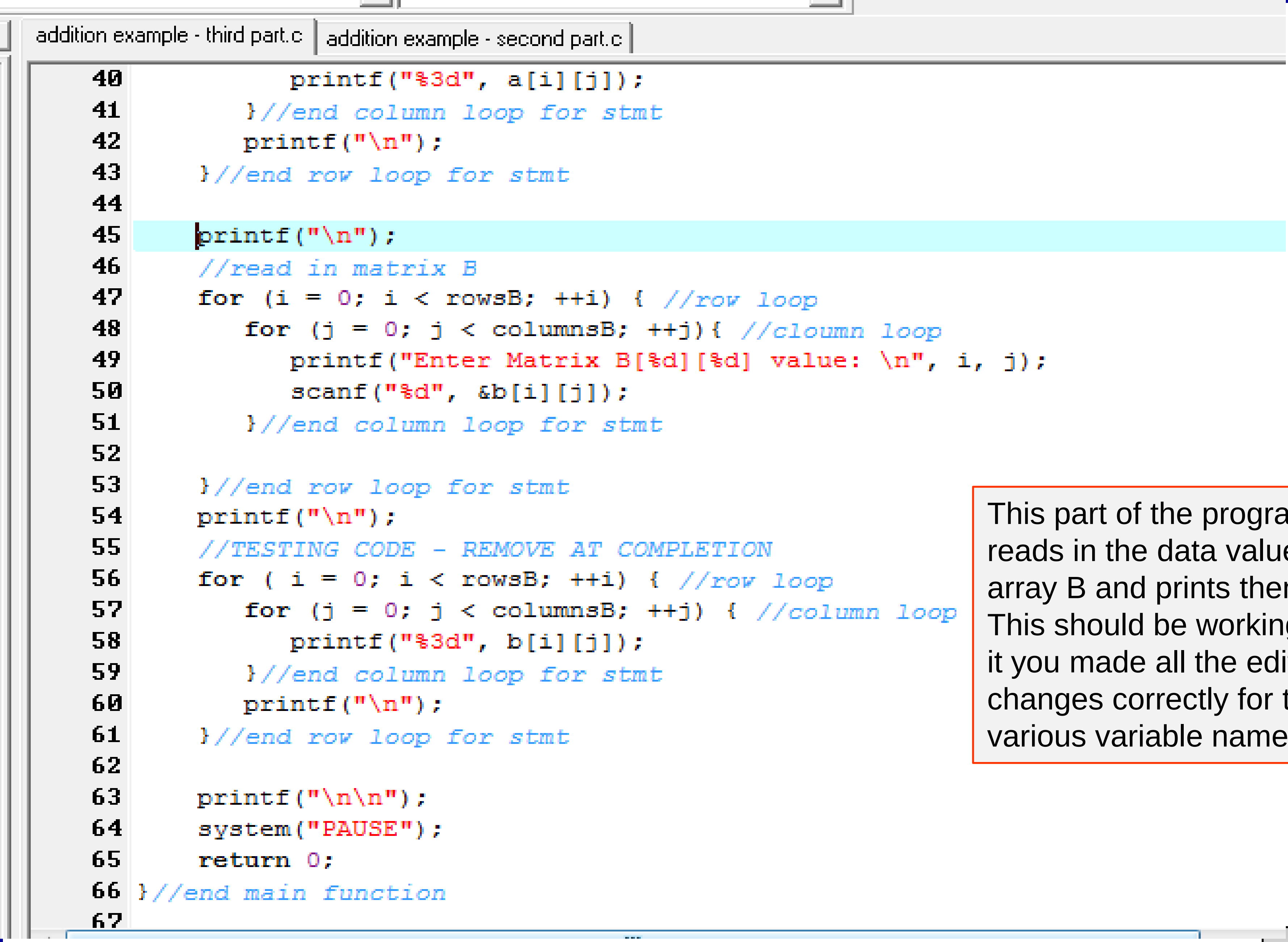


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Arrays In C - Part 2\te...

```
Enter the number of rows and columns for matrix A [max value for each is 10]:
2
3
Enter the number of rows and columns for matrix B [max value for each is 10]:
3
2
Enter Matrix A[0][0] value:
2
Enter Matrix A[0][1] value:
3
Enter Matrix A[0][2] value:
4

Enter Matrix A[1][0] value:
5
Enter Matrix A[1][1] value:
6
Enter Matrix A[1][2] value:
4

  2  3  4
  5  6  4


Press any key to continue . . .
```

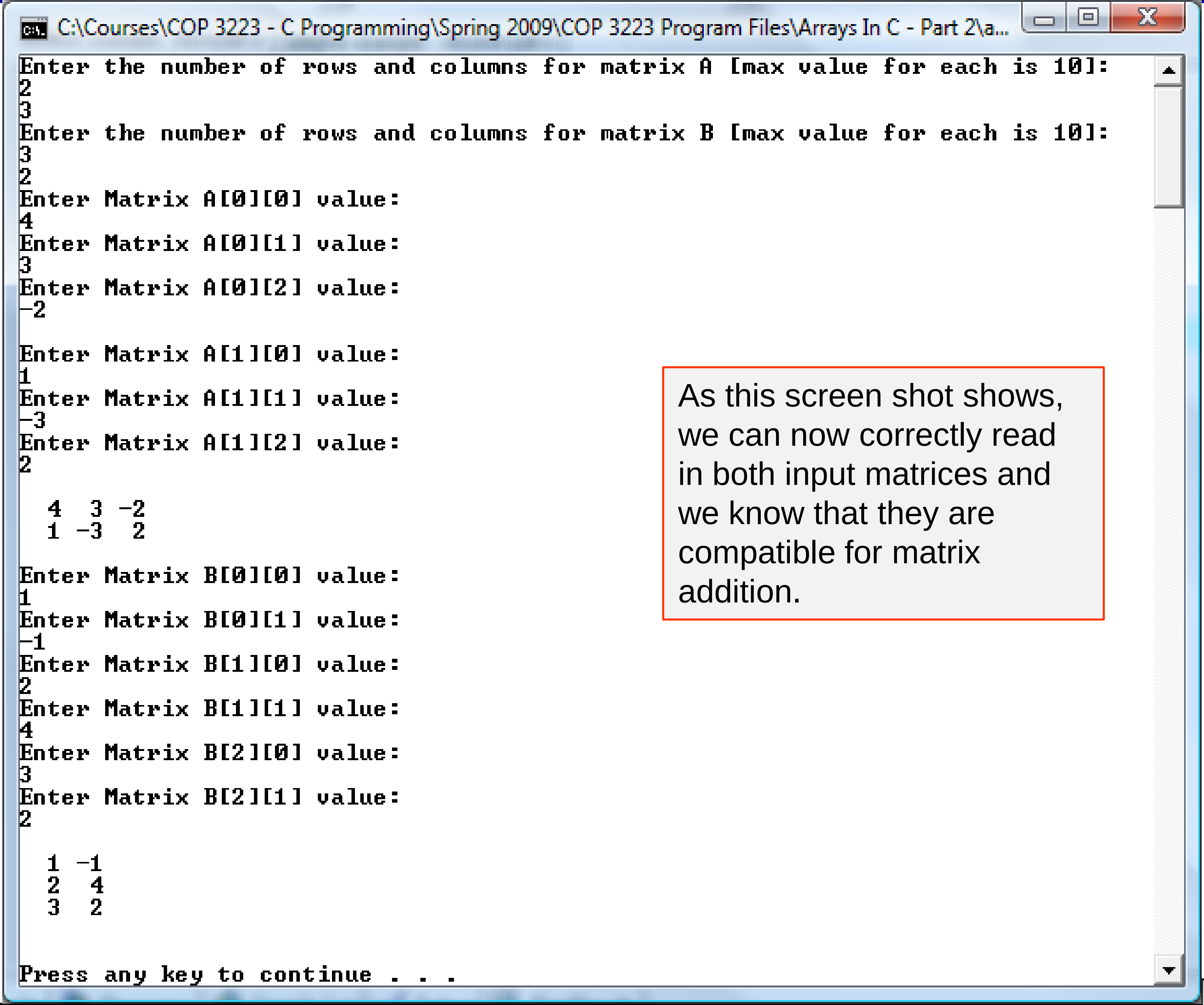
Using Multi-dimensional Arrays In C

- Once you know that you can enter the first array’s values and place them correctly into the array (which you’ve verified by printing out the contents of the array), simply copy the code and change the names of the variables to correspond to the second input array.
- For testing purposes, while you’re developing the code, just leave the lines of code the print out the array in the code. As mentioned earlier, we might want to use the code again and then we won’t need to rewrite it.
- This is shown in the next slide.


addition example - third part.c
addition example - second part.c

```
          printf("%3d", a[i][j]);
       }//end column loop for stmt
       printf("\n");
    }//end row loop for stmt

    printf("\n");
    //read in matrix B
    for (i = 0; i < rowsB; ++i) { //row loop
       for (j = 0; j < columnsB; ++j){ //cloumn loop
          printf("Enter Matrix B[%d][%d] value: \n", i, j);
          scanf("%d", &b[i][j]);
       }//end column loop for stmt

    }//end row loop for stmt
    printf("\n");
    //TESTING CODE - REMOVE AT COMPLETION
    for ( i = 0; i < rowsB; ++i) { //row loop
       for (j = 0; j < columnsB; ++j) { //column loop
          printf("%3d", b[i][j]);
       }//end column loop for stmt
       printf("\n");
    }//end row loop for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function
```

This part of the program reads in the data values for array B and prints them out. This should be working fine, it you made all the editing changes correctly for the various variable names.



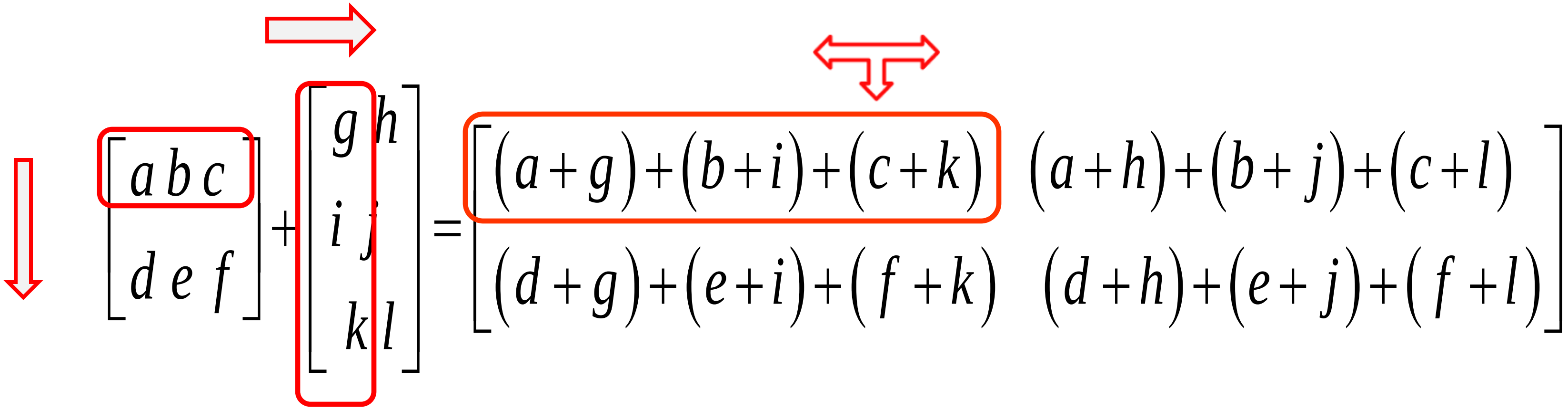
C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Arrays In C - Part 2\a...

```
Enter the number of rows and columns for matrix A [max value for each is 10]:
2
3
Enter the number of rows and columns for matrix B [max value for each is 10]:
3
2
Enter Matrix A[0][0] value:
4
Enter Matrix A[0][1] value:
3
Enter Matrix A[0][2] value:
-2

Enter Matrix A[1][0] value:
1
Enter Matrix A[1][1] value:
-3
Enter Matrix A[1][2] value:
2

   4  3 -2
   1 -3  2

Enter Matrix B[0][0] value:
1
Enter Matrix B[0][1] value:
-1
Enter Matrix B[1][0] value:
2
Enter Matrix B[1][1] value:
4
Enter Matrix B[2][0] value:
3
Enter Matrix B[2][1] value:
2

   1 -1
   2  4
   3  2


Press any key to continue . . .
```

As this screen shot shows, we can now correctly read in both input matrices and we know that they are compatible for matrix addition.

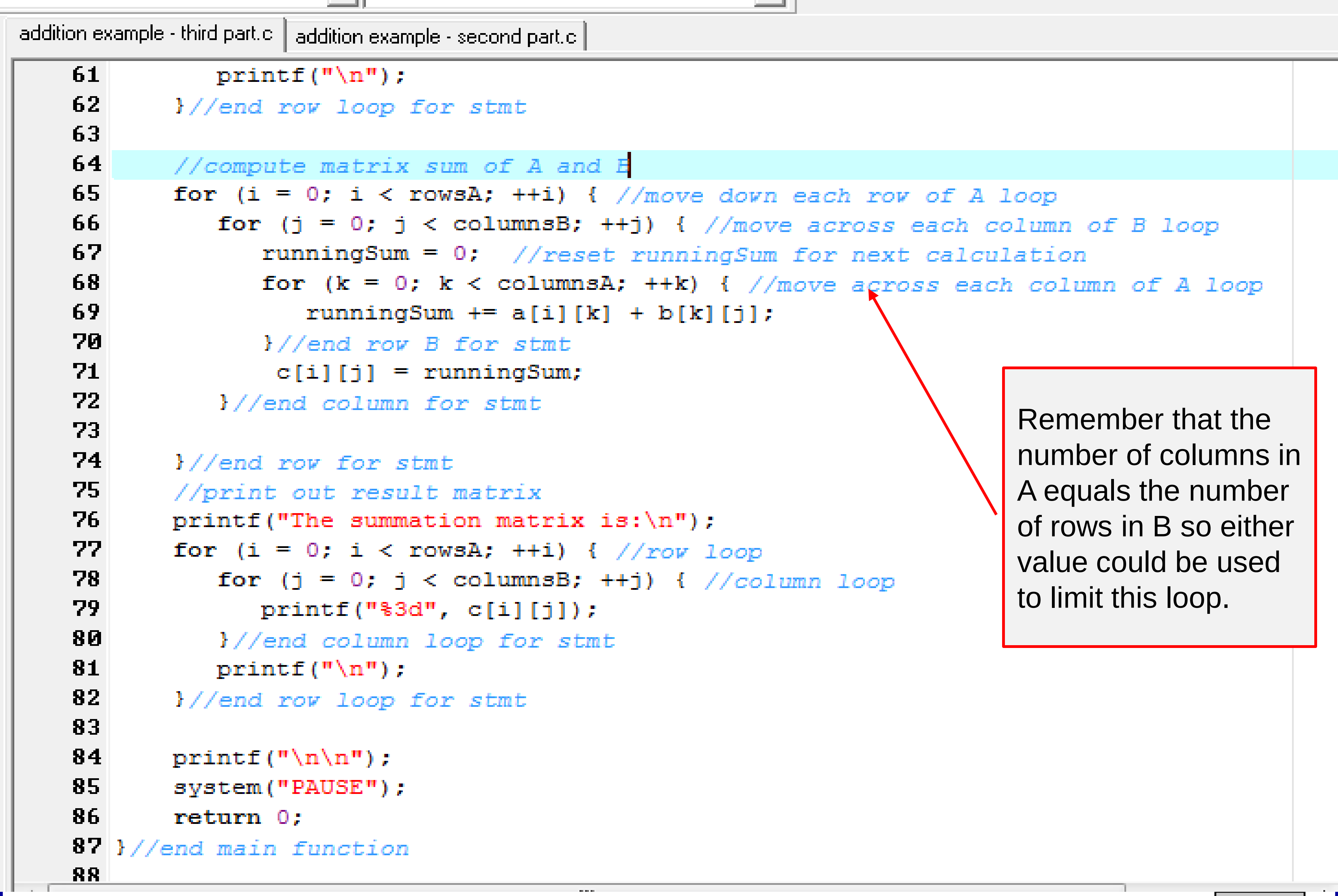
Using Multi-dimensional Arrays In C

- Now that we know we can enter two matrix addition compatible 2-d arrays, we move on to the real core of this problem.
- We now need to write the code that will add the two matrices together according to the generic expression shown on page 19.
- Consider how to code this into nested loops.



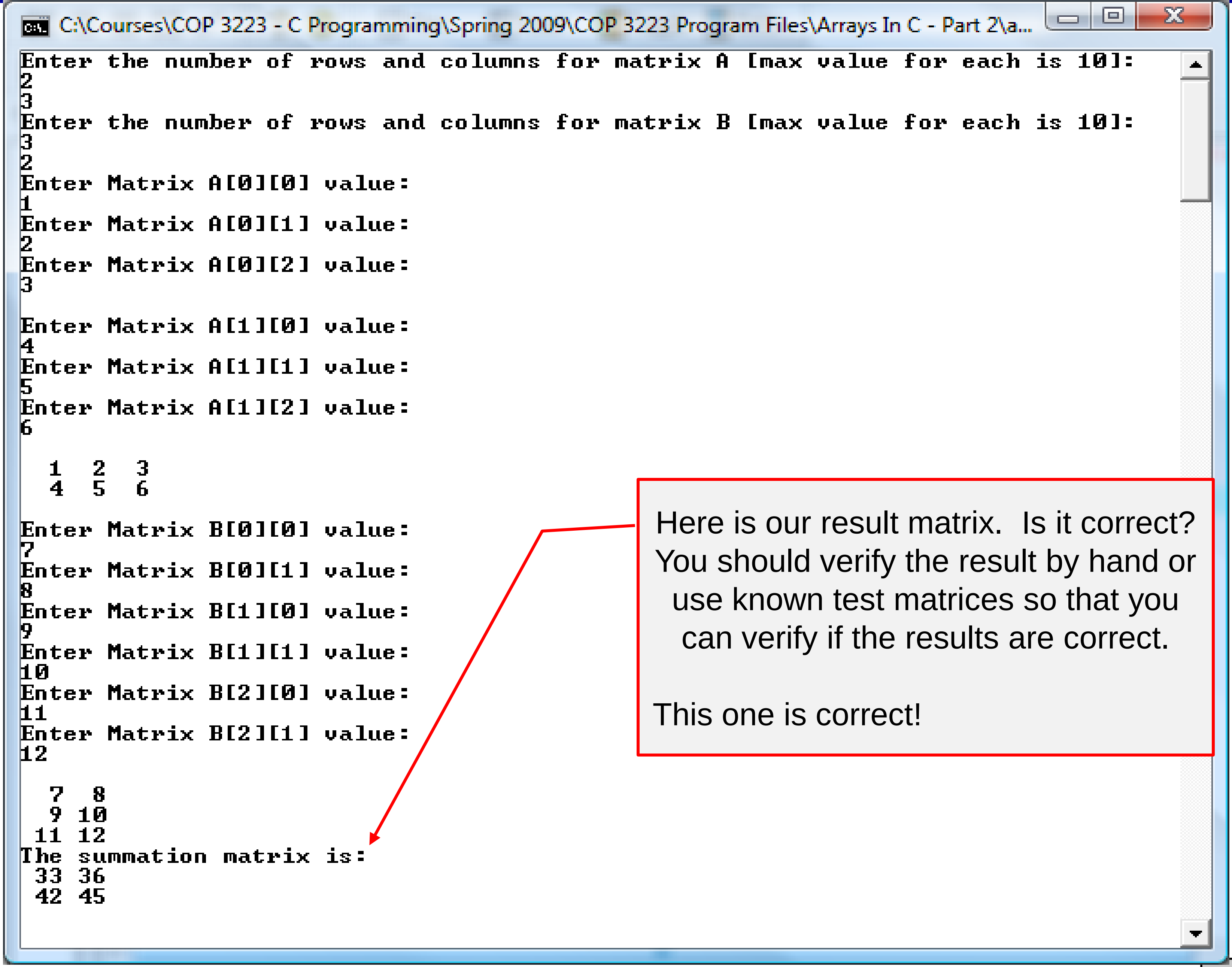
$$\begin{bmatrix} \boxed{a\ b\ c} \\ d\ e\ f \end{bmatrix} + \begin{bmatrix} g & h \\ i & j \\ k & l \end{bmatrix} = \begin{bmatrix} \boxed{(a+g)+(b+i)+(c+k)} & (a+h)+(b+j)+(c+l) \\ (d+g)+(e+i)+(f+k) & (d+h)+(e+j)+(f+l) \end{bmatrix}$$

We need an out loop to run across all the rows of A; for each row in A we need to move across all the columns of B; each column "movement" in B generates a new cell in C.


addition example - third part.c
addition example - second part.c

```c
       printf("\n");
    }//end row loop for stmt

    //compute matrix sum of A and B
    for (i = 0; i < rowsA; ++i) { //move down each row of A loop
       for (j = 0; j < columnsB; ++j) { //move across each column of B loop
          runningSum = 0;  //reset runningSum for next calculation
          for (k = 0; k < columnsA; ++k) { //move across each column of A loop
             runningSum += a[i][k] + b[k][j];
          }//end row B for stmt
           c[i][j] = runningSum;
       }//end column for stmt

    }//end row for stmt
    //print out result matrix
    printf("The summation matrix is:\n");
    for (i = 0; i < rowsA; ++i) { //row loop
       for (j = 0; j < columnsB; ++j) { //column loop
          printf("%3d", c[i][j]);
       }//end column loop for stmt
       printf("\n");
    }//end row loop for stmt

    printf("\n\n");
    system("PAUSE");
    return 0;
}//end main function

```

Remember that the number of columns in A equals the number of rows in B so either value could be used to limit this loop.


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Arrays In C - Part 2\a...

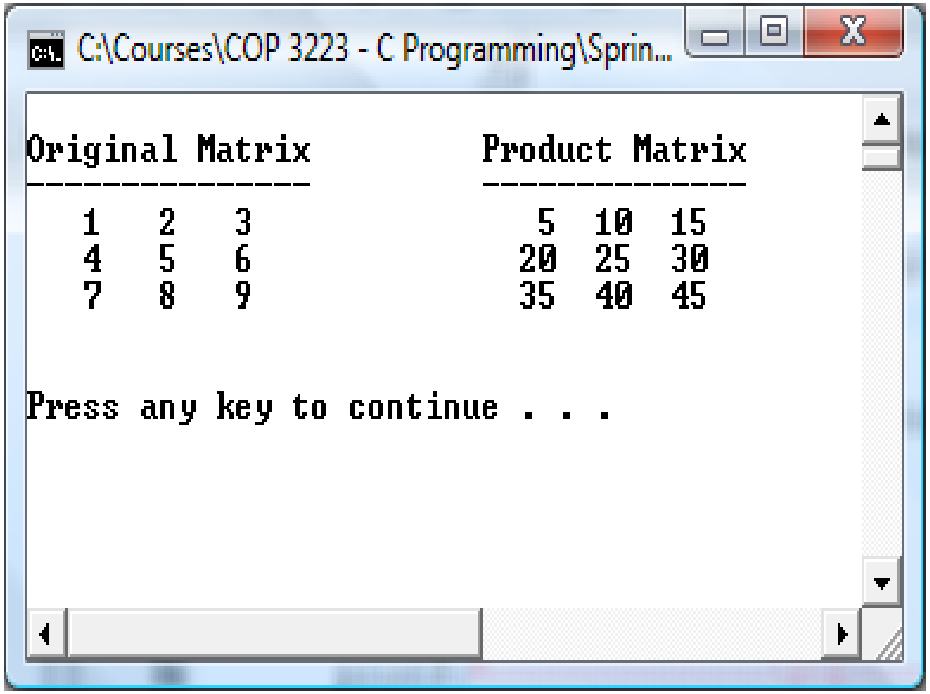
```
Enter the number of rows and columns for matrix A [max value for each is 10]:
2
3
Enter the number of rows and columns for matrix B [max value for each is 10]:
3
2
Enter Matrix A[0][0] value:
1
Enter Matrix A[0][1] value:
2
Enter Matrix A[0][2] value:
3

Enter Matrix A[1][0] value:
4
Enter Matrix A[1][1] value:
5
Enter Matrix A[1][2] value:
6

  1  2  3
  4  5  6

Enter Matrix B[0][0] value:
7
Enter Matrix B[0][1] value:
8
Enter Matrix B[1][0] value:
9
Enter Matrix B[1][1] value:
10
Enter Matrix B[2][0] value:
11
Enter Matrix B[2][1] value:
12

  7  8
  9 10
 11 12
The summation matrix is:
 33 36
 42 45
```

Here is our result matrix. Is it correct? You should verify the result by hand or use known test matrices so that you can verify if the results are correct.

This one is correct!

Practice Problems

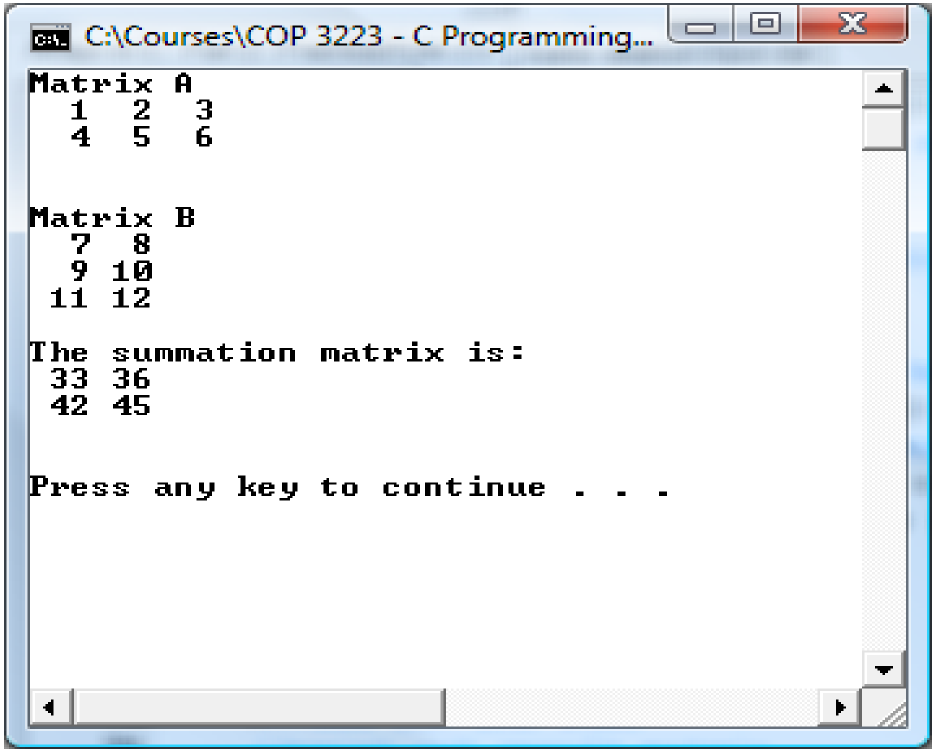
1. Modify the example on page 14 & 15 so that the values to be entered into the array and the multiplier value are read from an input file named “`arrayinput.dat`”.



```

Original Matrix          Product Matrix
---------------          --------------
   1   2   3                5  10  15
   4   5   6               20  25  30
   7   8   9               35  40  45


Press any key to continue . . .
```

Practice Problems

2. Modify the matrix addition problem so that all user entered data is read from an input file named “`matrix addition input.dat`”. Note that for file input, we will assume that the file contains valid input, since there is no way to prompt a file to reenter invalid data. So all the initial prompting to make sure that the column dimension of A and row dimension of B are equal can be removed from the program code.



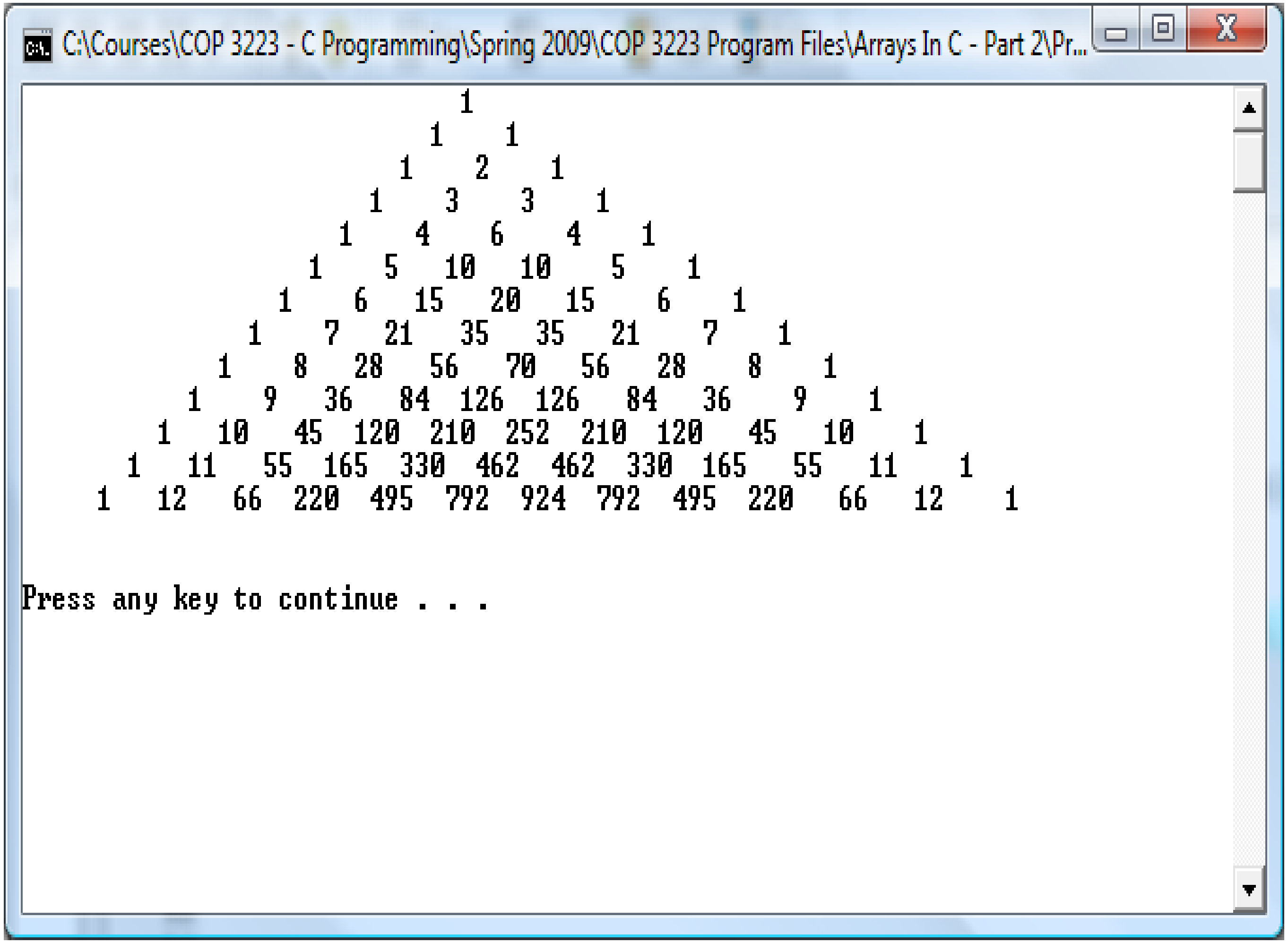
```
Matrix A
   1   2   3
   4   5   6


Matrix B
   7   8
   9  10
  11  12

The summation matrix is:
  33  36
  42  45


Press any key to continue . . .
```

Practice Problems

3. Write a C program that uses a 2-d array to store the values in Pascal's triangle. Recall from your mathematics background that Pascal's triangle is a triangle of values that is formed by creating each element in subsequent rows as the sum of the two elements in the previous row. Here are the first few rows of Pascal's triangle. Have your program generate the first 12 rows.

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

Practice Problems


C:\Courses\COP 3223 - C Programming\Spring 2009\COP 3223 Program Files\Arrays In C - Part 2\Pr...

```
                             1
                           1    1
                         1    2    1
                       1    3    3    1
                     1    4    6    4    1
                   1    5   10   10    5    1
                 1    6   15   20   15    6    1
               1    7   21   35   35   21    7    1
             1    8   28   56   70   56   28    8    1
           1    9   36   84  126  126   84   36    9    1
         1   10   45  120  210  252  210  120   45   10    1
       1   11   55  165  330  462  462  330  165   55   11    1
     1   12   66  220  495  792  924  792  495  220   66   12    1


Press any key to continue . . .
```