To:
Tenouk

C LAB WORKSHEET 10
2 Dimensional (2D) Array 1

1. The C & C++ 2D array data type.
2. 2D array declaration and initialization.
3. Flowcharts, swapping array elements etc.
4. Tutorial reference that should be used together with this worksheet are C & C++ array part 1 and C & C++ array part 2.

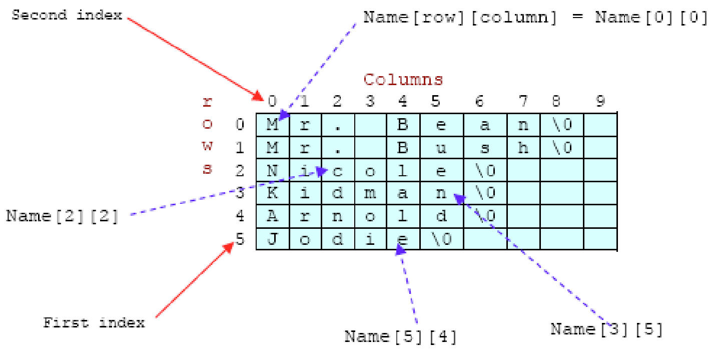
- In previous worksheet we were introduced with one dimensional array, 1D array as in the previous worksheet example, score[6]. A dimension is the size of the array. If we say that an array is 1D, it means the size of the array is one dimension. Array can has more than one dimension and in this worksheet we will deal with two dimension array, 2D array. Look at the following 2D array declaration:

  char Name[6][10];

- Here we have two indexes/subscripts. Normally the two indexes refer to the **rows** and **columns**, that is the [6] refers to rows and [10] refers to columns. If we assign initial string values for the 2D array it will look something like the following.

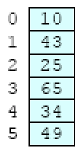
  char Name[6][10] = {"Mr. Bean", "Mr. Bush", "Nicole", "Kidman", "Arnold", "Jodie"};

- Here, we can initialize the array with 6 strings, each with maximum 9 characters long. If depicted in rows and columns it will look something like the following and can be considered as contiguous arrangement in the memory.



- And compare to the 1D array shown below.

  Score[6];



- Take note that for strings the null character (\0) still needed. From the shaded square area of the Figure we can determine the size of the array. For an array Name[6][10], the array size is 6 x 10 = 60 and equal to the number of the colored square. In general:

  array_name[x][y];

  The array size is = First index **x** second index = xy.

- This also true for other array dimension, for example three dimensional array:

  array_name[x][y][z] = First index **x** second index **x** third index = xyz.

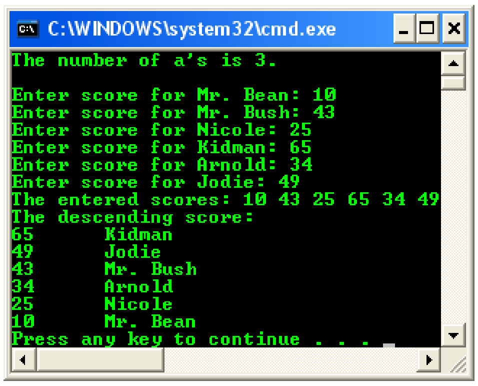
- For example:

  ThreeDimArray[2][4][7] = 2 x 4 x 7 = 56.

- And if you want to illustrate the 3D array, it could be a cube with wide, long and height. Can you imagine the 4D, 5D and higher dimension arrays?
- Let put the 2D array in the real C program. Create an empty Win32 console application project named **myextarray**, add source file named **myextarraysrc** and set your project c**ompiled as C code**, build and run the following program example.

```c
#include <stdio.h>
// for strcpy_s()
#include <string.h>

void main()
{
     // array that holds 6 strings, each max 9 characters long
     char Name[6][10] ={"Mr. Bean", "Mr. Bush", "Nicole", "Kidman", "Arnold", "Jodie"};
     // array that holds scores for 6 players of "name[ ][ ]"
     int Score[6],
    // a variable to count the number of a's.
    Count = 0,
    // index for the largest score while doing the sorting
    LargestIdx,
    // used in the for loop
    i, j,
    // used to temporarily store a score while doing the swapping
    Temp;
    // used to temporarily store a name while doing the swapping
    char TempStr[10];
    // loop and nested loop to count the number of a's in all the names
    // loop for the rows...
    for(i = 0; i <= 5; i = i + 1)
          // loop for the columns
          for(j = 0; Name[i][j] != '\0'; j = j + 1)
               // start counting the 'a' and 'A'
               if(Name[i][j] == 'a' || Name[i][j] == 'A')
                    Count = Count + 1;
    printf("The number of a's is %d.\n\n", Count);
    // loop to get the scores of all 6 players
    for(i = 0; i <= 5; i = i + 1)
    {
          printf("Enter score for %s: ", Name[i]);
          // scanf("%d", &Score[i]);
          scanf_s("%d", &Score[i], 1);
    }
    // nothing here, just to see the entered scores...
    printf("The entered scores: ");
    for(i = 0; i <= 5; i = i + 1)
          printf("%d ", Score[i]);
    // start the sorting, i is the number of the pass
    for(i = 0; i <= 4; i = i + 1)
    {
          // find index that has the largest number for this pass
          LargestIdx = i;
          for(j = i + 1; j <= 5; j = j + 1)
               if(Score[j] > Score[LargestIdx])
                    LargestIdx = j;
          // swap the first score and name for this pass with that of the largest score
          Temp = Score[i];
          Score[i] = Score[LargestIdx];
          Score[LargestIdx] = Temp;
          // strcpy(TempStr, Name[i]);
          strcpy_s(TempStr, 9, Name[i]);
          strcpy_s(Name[i], 9, Name[LargestIdx]);
          strcpy_s(Name[LargestIdx], 9, TempStr);
    }
    // print out the sorted scores and names
    printf("\nThe descending score:\n");
    for(i = 0; i <= 5; i = i + 1)
          printf("%d\t%s\n", Score[i], Name[i]);
}
```



```
The number of a's is 3.

Enter score for Mr. Bean: 10
Enter score for Mr. Bush: 43
Enter score for Nicole: 25
Enter score for Kidman: 65
Enter score for Arnold: 34
Enter score for Jodie: 49
The entered scores: 10 43 25 65 34 49
The descending score:
65      Kidman
49      Jodie
43      Mr. Bush
34      Arnold
25      Nicole
10      Mr. Bean
Press any key to continue . . .
```

- In the program example, two arrays, Name[6][10] is a 2D array and Score[6] is a 1D array. Name[ ][ ] is an array of characters and Score[ ] is an array of integers. To access a slot in Score[ ], only one index has to be specified but to access a slot in Name[ ], two indexes have to be specified. The first index is called **row** and the second one **column**.
- For the program example, during the execution, the values for Score[ ] will be read in and stored in the Score[ ] array. The Name[6][10] array has two subscript/indexes. The first subscript specifies that there are 6 players and the second specifies that there are up to 9 characters in the names of these players, not including the null character ('\0').
- Name[2][2] has the value of 'c'. That is in the slot where the row is 2 and the column is 2, the value is 'c'. Name[0][0] has the value of 'M'. Here the row is 0 and the column index is 0. The names of 6 game players are stored in this array. They are character strings so each one ends with a **null character**. When printing strings, all characters are printed in order until a null character is encountered, so Name[2] has the value of "Mr. Bean". The Name[ ][ ] array holds the names of players and Score[ ] array holds the scores of the corresponding players. For example, the game score of 25 is stored in Score[2] for "Nicole" which is stored in Name[2].

Accessing 2D Array Using Nested for Loop

- After the definition of variables, we see nested for loop 1, which counts the number of a's and A's. The first loop varies i from 0 to 5 in increments of 1. The nested loop varies j from 0 in increment of 1 until the end of the string is encountered.

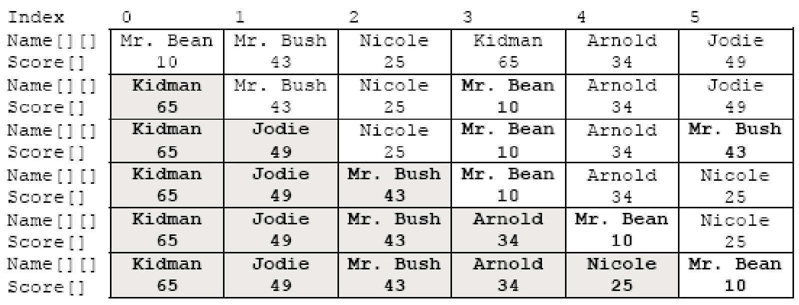
```c
// loop for the rows...
for(i = 0; i <= 5; i = i + 1)
```

```
// loop for the columns
for(j = 0; Name[i][j] != '\0'; j = j + 1)
     // start counting the 'a' and 'A'
     if(Name[i][j] == 'a' || Name[i][j] == 'A')
          Count = Count + 1;
 printf("The number of a's is %d.\n\n", Count);
```

- What happens here is that i stays fixed at 0, while in the inside loop, j varies until a null character is found. After that, i stays at 1 and j varies from 0 until a null character is found, and so on. While the loop make i and j vary so that all valid characters are accessed, the if statement counts the number of 'a' and 'A'.

Selection Sort

- Sorting of data is a common task that needs to be done when processing data. In C++ common routines could use Standard Template Library (STL). There are many sorting algorithms, but the ones that are easy to understand are called "natural sorts". Selection sort is a natural sort because it is one of the algorithms by which sorting is done by humans. Most text may use the "bubble sort", but it is not any faster nor is it a natural sort.
- From the program example we want to sort out the two arrays so that the scores are in descending order (high to low) and Name[][] is also sorted so that the names and scores stay together. In other words, "Kidman" with score 65 (both in slot number 3) will both be moved to slot number 0 after the sort is complete. Likewise, "Jodie" with a score of 49 will be moved from slot number 5 to slot number 1, since he has the second highest score and so on. While sorting, we need to keep the players' names with their respective scores.
- You can see how this sort process works in the following Figure. The shaded cells in the Figure show how much of the array is sorted during each pass. During the first pass, the person with the highest score, "Kidman" is moved to an index of 0. "Mr. Bean" with a score of 10 and who occupied this position, is moved down to the index of 3. A swap has occurred because "Kidman" needed to be in the first position and "Mr. Bean" needed a new slot/index.
- Now we find the player with the highest score in the array starting at index 1. "Jodie" is swapped into position 1 with "Mr. Bush" taking "Jodie"'s place in index 5. Next for the third pass "Mr. Bush" takes the position at index 2, swapping with "Nicole" and the process goes on until we have a descending sorted array.

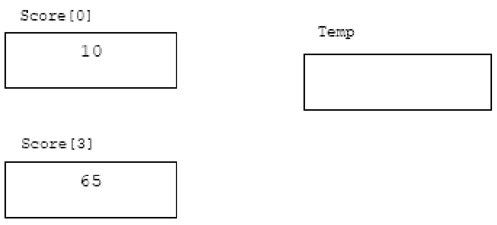


Index	0	1	2	3	4	5
Name[][]	Mr. Bean	Mr. Bush	Nicole	Kidman	Arnold	Jodie
Score[]	10	43	25	65	34	49
Name[][]	Kidman	Mr. Bush	Nicole	Mr. Bean	Arnold	Jodie
Score[]	65	43	25	10	34	49
Name[][]	Kidman	Jodie	Nicole	Mr. Bean	Arnold	Mr. Bush
Score[]	65	49	25	10	34	43
Name[][]	Kidman	Jodie	Mr. Bush	Mr. Bean	Arnold	Nicole
Score[]	65	49	43	10	34	25
Name[][]	Kidman	Jodie	Mr. Bush	Arnold	Mr. Bean	Nicole
Score[]	65	49	43	34	10	25
Name[][]	Kidman	Jodie	Mr. Bush	Arnold	Nicole	Mr. Bean
Score[]	65	49	43	34	25	10

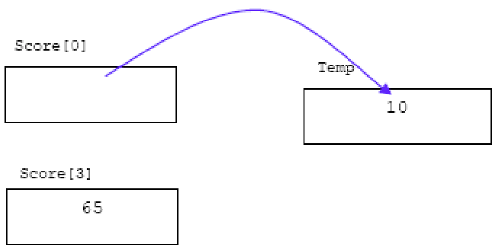
- The largest score has to be found many times as the sort progresses. During each pass, the next largest is found. Finding the largest score is not sufficient, but we need to know the index of the score with the largest value. We need to know where that largest score exists because a swap has to occur. For instance, for the first pass, when "Kidman" is found to have the highest score, we needed to know that he was stored at slot 3 so that "Kidman" and "Mr. Bean" and their scores could be swapped.
- The swapping is a three-step process and we need an empty slot for temporary storage. From the program example, we start at Temp = Score[i]. Here we need to swap the values stored at Score [i] and Score[LargestIdx]. When swapping "Kidman" and "Mr. Bean", i is 0 and LargestIdx is 3. We take 10 which is at Score[0] and place it in Temp. Then the 65 in Score[LargestIdx] is placed in Score[0], the 0th slot and the 10 that was in Temp is placed in Score[3], the slot from which the largest number was copied. The process is depicted in the following Figures.

```
// swap the first score and name for this pass with that of the largest score
Temp = Score[i];
Score[i] = Score[LargestIdx];
Score[LargestIdx] = Temp;
strcpy_s(TempStr, 9, Name[i]);
strcpy_s(Name[i], 9, Name[LargestIdx]);
strcpy_s(Name[LargestIdx], 9, TempStr);
```

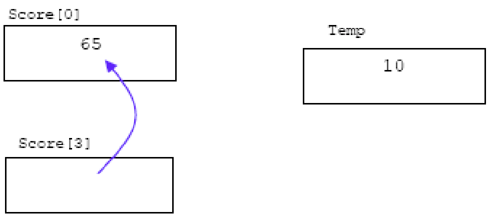
- Original position: Prepare a temporary storage, Temp.



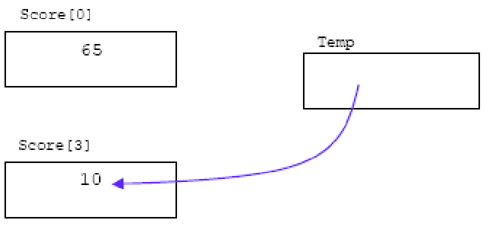
- Step 1: Store 10 into Temp.



- Step 2: Store 65 into Score[0]



- Step 3: Store 10 into Score[3]



- Te three-step swapping process is repeated for the players. Since the players are strings, the strcpy_s() function must be used rather than the assignment statements, which were used for the scores. Notice the Name[ ][ ], only one subscript is used because we are processing strings not the individual character. When we were counting the number of a's, we were processing individual characters that were in each slot. Then we needed to use both indexes.

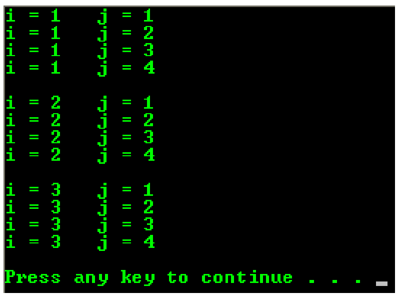
Experiment And Practice

1. Build, run and show the output for the following program then answer the questions. In this program, i (the row) is fixed, j (column) varies.

```
#include <stdio.h>

void main()
{
     int i, j;
     for(i = 1; i <= 3; i = i + 1)
     {
          for(j = 1; j <= 4; j = j + 1)
               printf("i = %d\tj = %d\n", i, j);
               printf("\n");
     }
}
```

   a. There are two loops above, the i loop and the j loop. Which loop is the outer loop?
   b. Which loop would you call the nested loop?
   c. Which loop is done only once? Which one is repeated?
   d. How many times is i set to 1? How many times is j set to 1?
   e. Which variable varies faster than the other?
   f. What is the effect of omitting the braces of the i loop?

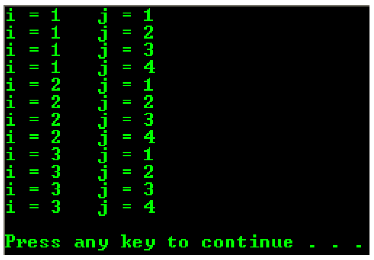


```
i = 1    j = 1
i = 1    j = 2
i = 1    j = 3
i = 1    j = 4

i = 2    j = 1
i = 2    j = 2
i = 2    j = 3
i = 2    j = 4

i = 3    j = 1
i = 3    j = 2
i = 3    j = 3
i = 3    j = 4
Press any key to continue . . .
```

a. The i loop is the outer loop.
b. j loop is the nested loop.
c. i loop was done once and j loop was repeated.
d. 4 times.
e. j variable.
f. By omitting the braces, only the last printf() will be effected. In this program only one line of code available and effect each of the for loop. The last printf() only print new line for every outer for loop iteration. A sample output is shown below when the outer for loop braces have been omitted.



```
i = 1    j = 1
i = 1    j = 2
i = 1    j = 3
i = 1    j = 4
i = 2    j = 1
i = 2    j = 2
i = 2    j = 3
i = 2    j = 4
i = 3    j = 1
i = 3    j = 2
i = 3    j = 3
i = 3    j = 4

Press any key to continue . . .
```

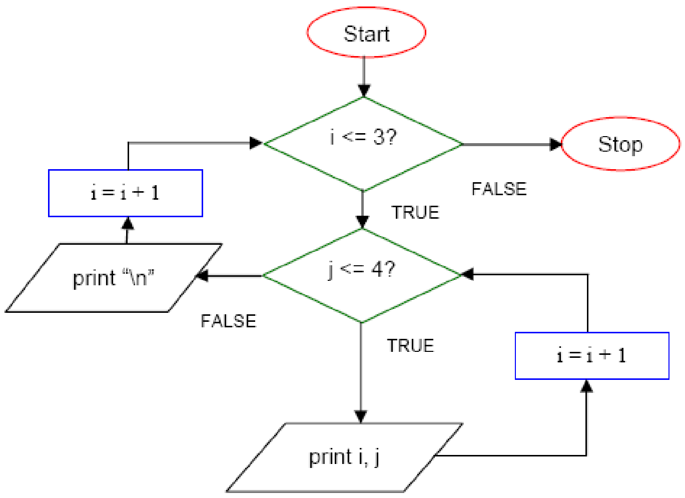
The Figure on the right is an illustration of the inner and outer for loops. The green area is the inner loop that will be executed while the outer loop depicted in blue is true.

```
#include <stdio.h>

void main()
{
    int i, j;

    for(i = 1; i <= 3; i = i + 1)
    {
        for(j = 1; j <= 4; j = j + 1)
            printf("i = %d\tj = %d\n", i, j);
            printf("\n");
    }
}
```

The flowchart for this program example is given on the right.

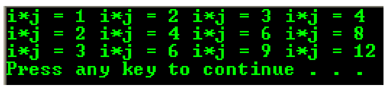


2. Let do some operation on the array. We do some modification in the first printf(). Show the output and answer the questions.

```
#include <stdio.h>

void main()
{
    int i, j;
    for(i = 1; i <= 3; i = i + 1)
    {
        for(j = 1; j <= 4; j = j + 1)
            printf("i*j = %d\t", i*j);
        printf("\n");
    }
}
```

   a. Firstly, i is 1 and j is 1. What is printed at that time?
   b. Then, i is 1 and j is 2. What is printed at that time? Why is it on the same line?
   c. Eventually, i is 1 and j is 4. What is printed at that time?
   d. Now that the j loop is complete for the first time, what happens before i becomes 2?
   e. When i is 2 and j is 3, what is printed?
   f. When i is 3 and j is 4, what is printed?



```
i*j = 1 i*j = 2 i*j = 3 i*j = 4
i*j = 2 i*j = 4 i*j = 6 i*j = 8
i*j = 3 i*j = 6 i*j = 9 i*j = 12
Press any key to continue . . .
```

a. 1 was printed because 1*1 = 1.
b. 2 was printed because 1*2 = 2.
c. 4 was printed because 1*4 = 4.
d. It goes to a new line because of the printf("\n"); statement.
e. 6 was printed because 2*3 = 6.
f. 12 was printed because 3*4 = 12.

3. Complete the for statement for the j loop so that the output is as shown below. When i is 1, we want j to go as far as 1. When i is 2, we want j to go as far as 2 and so on.

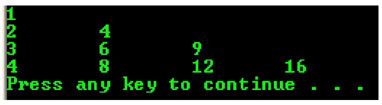
```
#include <stdio.h>

void main()
{
    int i, j;
    for(i = 1; i <= 4; i = i + 1)
    {
        for(j =_______; j <= ______; j = j + 1)
            printf("%d\t", i*j);
        printf("\n");
    }
}
```

```
1
2    4
3    6    9
4    8    12    16
```

```
#include <stdio.h>

void main()
{
   int i, j=0;
   for(i = 1; i <= 4; i = i + 1)
   {
      for(j = 1; j <= i; j = j + 1)
         printf("%d\t", i*j);
      printf("\n");
   }
}
```



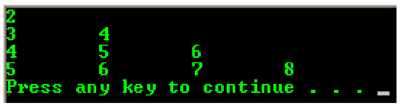
```
1
2       4
3       6       9
4       8       12      16
Press any key to continue . . .
```

4. Next, can you replace i * j with something else in the printf() to give the following output?

```
2
3   4
4   5   6
5   6   7   8
```

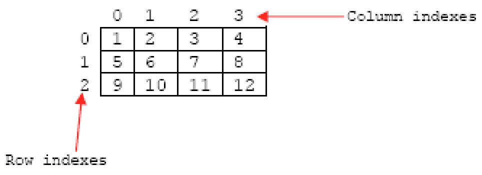
```
#include <stdio.h>

void main()
{
    int i, j=0;
    for(i = 1; i <= 4; i = i + 1)
    {
        for(j = 1; j <= i; j = j + 1)
            printf("%d\t", i+j);
        printf("\n");
    }
}
```



```
2
3       4
4       5       6
5       6       7       8
Press any key to continue . . .
```

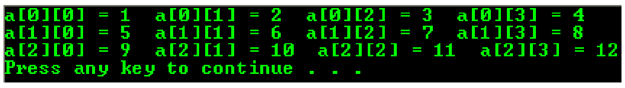
5. a[ ][ ] is a 2D array. The first index is always the row index and the second one is always the column. a[ ][ ] here has 3 rows and 4 columns. i, being the first index, is the index for the row. If j was the first index, then it would have been the row index. The array is initialized in memory as shown below. Show the output for the program example and answer the questions.



```
#include <stdio.h>

void main()
{
     int i, j, a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
     for(i = 0; i <= 2; i = i + 1)
     {
          for(j = 0; j <= 3; j = j + 1)
               printf("a[%d][%d] = %d  ", i, j, a[i][j]);
          printf("\n");
     }
}
```

a. When i is 0 and j is 0, the element in the first row and the first column is printed. When i is 0 and j is 1, the element in which slot is printed?
b. When i is 0 and j is 3, the element in which slot is printed?
c. When i is 2 and j is 1, the element in which slot is printed?



```
a[0][0] = 1  a[0][1] = 2  a[0][2] = 3  a[0][3] = 4
a[1][0] = 5  a[1][1] = 6  a[1][2] = 7  a[1][3] = 8
a[2][0] = 9  a[2][1] = 10  a[2][2] = 11  a[2][3] = 12
Press any key to continue . . .
```

a. Element in the first row and second column.
b. Element in the first row and fourth column.
c. Element in the third row and second column.

- We also can force the arrangement of the array content by using another curly brace as shown below. It is very convenient method in forcing the arrangement of the array element.

  a[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

- Here we have a very clear arrangement of the array element for 3 rows, each with 4 columns. Let drop one element, a[0][3] in the first row of the array, then we have:

  a[3][4] = {{1,2,3}, {5,6,7,8}, {9,10,11,12}};

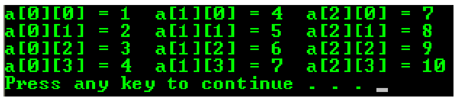
- In this case, the element a[0][3] will be filled with 0. You can try using this array initialization in other program examples.

6. In the following example, we swap i with j in the previous program example. Show the output and answer the questions.

```
#include <stdio.h>

void main()
{
     int i, j, a[4][3] = {1,2,3,4,5,6,7,8,9,10,11,12};
     for(i = 0; i <= 3; i = i + 1)
     {
          for(j = 0; j <= 2; j = j + 1)
               printf("a[%d][%d] = %d  ", j, i, a[j][i]);
          printf("\n");
     }
}
```

a. When i is 0 and j is 0, the element in which slot is printed?
b. When i is 0 and j is 1, the element in which slot is printed?
c. When i is 0 and j is 2, the element in which slot is printed?



```
a[0][0] = 1  a[1][0] = 4  a[2][0] = 7
a[0][1] = 2  a[1][1] = 5  a[2][1] = 8
a[0][2] = 3  a[1][2] = 6  a[2][2] = 9
a[0][3] = 4  a[1][3] = 7  a[2][3] = 10
Press any key to continue . . .
```

a. Element in first row and first column.
b. Element in first row and second column.

d. When i is 1 and j is 0, the element in which slot is printed?

c. Element in first row and third column.
d. Element in second row and first column.

www.tenouk.com

| Main |< C & C++ Array Data Type 4 | C & C++ 2D Array 2 >| Site Index | Download |

The C & C++ 2D Array Manipulation: Part 1 | Part 2 | Part 3