



C ARRAYS

-a collection of same type data-

ARRAYS

- An array is a collection of elements of the same type that are referenced by a common name.
- Compared to the basic data type (`int`, `float` & `char`) it is an aggregate or derived data type.
- All the elements of an array occupy a set of contiguous memory locations.
- Why need to use array type?
- Consider the following issue:

"We have a list of 1000 students' marks of an integer type. If using the basic data type (`int`), we will declare something like the following..."

```
int studMark0, studMark1, studMark2, ..., studMark999;
```

ARRAYS

- Can you imagine how long we have to write the declaration part by using normal variable declaration?

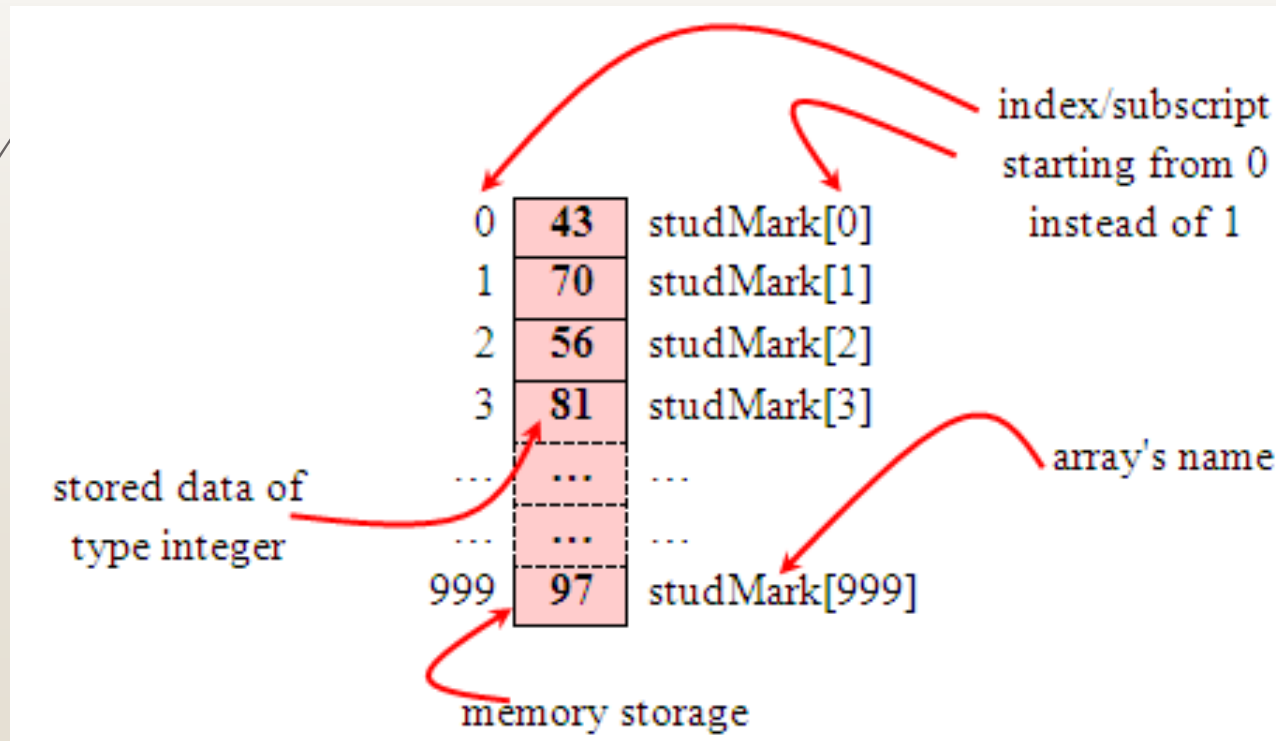
```
int main(void)
{
    int studMark1, studMark2, studMark3,
    studMark4, ..., ..., studMark998, stuMark999,
    studMark1000;
    ...
    ...
    return 0;
}
```

ARRAYS

- By using an array, we just declare like this,

```
int studMark[1000];
```

- This will reserve 1000 contiguous memory locations for storing the students' marks.
- Graphically, this can be depicted as in the following figure.



ARRAYS

- This absolutely has simplified our declaration of the variables.
- We can use index or subscript to identify each element or location in the memory.
- Hence, if we have an index of `jIndex`, `studMark[jIndex]` would refer to the *jIndex*th element in the array of `studMark`.
- For example, `studMark[0]` will refer to the first element of the array.
- Thus by changing the value of `jIndex`, we could refer to any element in the array.
- So, array has simplified our declaration and of course, manipulation of the data.

ARRAYS

One Dimensional Array: Declaration

- Dimension refers to the array's size, which is how big the array is.
- A single or one dimensional array declaration has the following form,

```
array_element_data_type array_name[array_size];
```

- Here, *array_element_data_type* define the base type of the array, which is the type of each element in the array.
- *array_name* is any valid C / C++ identifier name that obeys the same rule for the identifier naming.
- *array_size* defines how many elements the array will hold.

ARRAYS

- For example, to declare an array of 30 characters, that construct a people name, we could declare,

```
char    cName[ 30 ] ;
```

- Which can be depicted as follows,
- In this statement, the array character can store up to 30 characters with the first character occupying location `cName[0]` and the last character occupying `cName[29]`.
- Note that the index runs from 0 to 29. In C, an index always starts from 0 and ends with array's (size-1).
- So, take note the difference between the array size and subscript/index terms.

J	cName[0]
o	cName[1]
d	cName[2]
i	cName[3]
e	cName[4]
...	cName[5]
...	...
...	...
r	cName[29]

ARRAYS

- Examples of the one-dimensional array declarations,

```
int      xNum[20], yNum[50];  
float    fPrice[10], fYield;  
char     chLetter[70];
```

- The first example declares two arrays named `xNum` and `yNum` of type `int`. Array `xNum` can store up to 20 integer numbers while `yNum` can store up to 50 numbers.
- The second line declares the array `fPrice` of type `float`. It can store up to 10 floating-point values.
- `fYield` is basic variable which shows array type can be declared together with basic type provided the type is similar.
- The third line declares the array `chLetter` of type `char`. It can store a string up to 69 characters.
- Why 69 instead of 70? Remember, a string has a null terminating character (`\0`) at the end, so we must reserve for it.

ARRAYS

Array Initialization

- An array may be initialized at the time of declaration.
- Giving initial values to an array.
- Initialization of an array may take the following form,

```
type    array_name[size] = {a_list_of_value};
```

- For example:

```
int     idNum[7] = {1, 2, 3, 4, 5, 6, 7};  
float   fFloatNum[5] = {5.6, 5.7, 5.8, 5.9, 6.1};  
char    chVowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};
```

- The first line declares an integer array `idNum` and it immediately assigns the values 1, 2, 3, ..., 7 to `idNum[0]`, `idNum[1]`, `idNum[2]`, ..., `idNum[6]` respectively.
- The second line assigns the values 5.6 to `fFloatNum[0]`, 5.7 to `fFloatNum[1]`, and so on.
- Similarly the third line assigns the characters 'a' to `chVowel[0]`, 'e' to `chVowel[1]`, and so on. Note again, for characters we must use the single apostrophe/quote (') to enclose them.
- Also, the last character in `chVowel` is NULL character ('\0').

ARRAYS

- Initialization of an array of type char for holding strings may take the following form,

```
char    array_name[size] = "string_lateral_constant";
```

- For example, the array chVowel in the previous example could have been written more compactly as follows,

```
char    chVowel[6] = "aeiou";
```

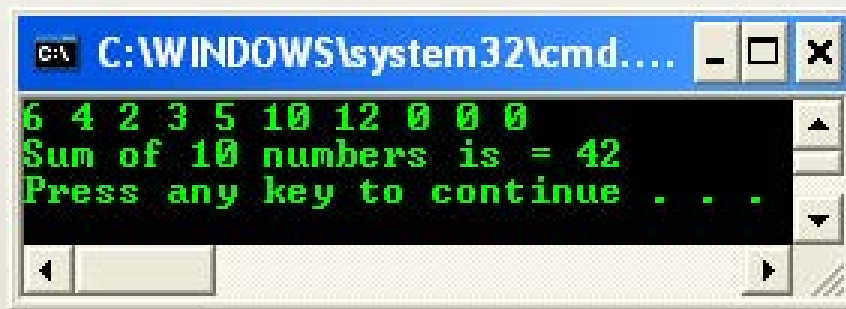
- When the value assigned to a character array is a string (which must be enclosed in double quotes), the compiler automatically supplies the NULL character but we still have to reserve one extra place for the NULL.
- For unsized array (variable sized), we can declare as follow,

```
char chName[ ] = "Mr. Dracula";
```

- C compiler automatically creates an array which is big enough to hold all the initializer.

ARRAYS

- Arrays allow programmers to group related items of the same data type in one variable.
- However, when referring to an array, one has to specify not only the array or variable name but also the index number of interest.
- Program example 1: Sum of array's element.
- Notice the array's element which is not initialized is set to 0 automatically.



```
C:\WINDOWS\system32\cmd....  
6 4 2 3 5 10 12 0 0 0  
Sum of 10 numbers is = 42  
Press any key to continue . . .
```

ARRAYS


- Program example 2: Searching the smallest value.
- Finding the smallest element in the array named `fSmallest`.
- First, it assumes that the smallest value is in `fSmallest[0]` and assigns it to the variable `nSmall`.
- Then it compares `nSmall` with the rest of the values in `fSmallest`, one at a time.
- When an element is smaller than the current value contained in `nSmall`, it is assigned to `nSmall`. The process finally places the smallest array element in `nSmall`.



```
C:\WINDOWS\system32\cmd.exe
12.00 41.50 -31.20 -45.00 33.00 -21.20 -24.10 0.70 3.20 0.50
Searching...
The smallest value in the given array is -45.00
Press any key to continue . . .
```

ARRAYS

- Program example 3: Searching the biggest value. By modifying the previous example we can search the biggest value.

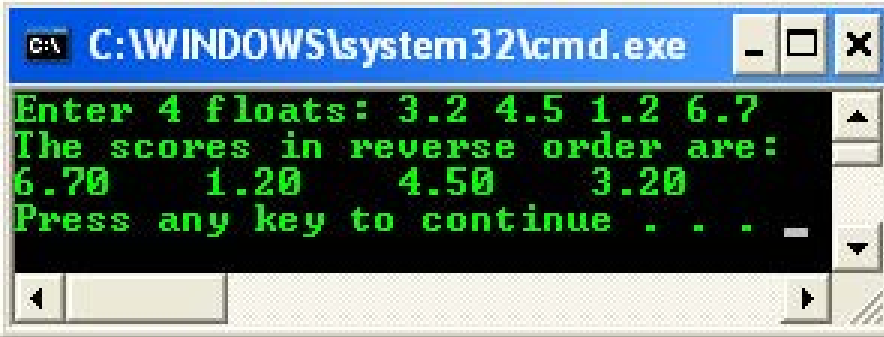


```
C:\WINDOWS\system32\cmd.exe
12.00 41.50 -31.20 -45.00 33.00 -21.20 -24.10 0.70 3.20 0.50
Searching...
The biggest value in the given array is 41.50
Press any key to continue . . .
```

The screenshot shows a Windows command prompt window with a blue title bar. The window title is "C:\WINDOWS\system32\cmd.exe". The command prompt displays a list of ten floating-point numbers in green text: 12.00, 41.50, -31.20, -45.00, 33.00, -21.20, -24.10, 0.70, 3.20, and 0.50. Below the numbers, the text "Searching..." is displayed. The next line shows the result: "The biggest value in the given array is 41.50". The final line of text is "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

ARRAYS

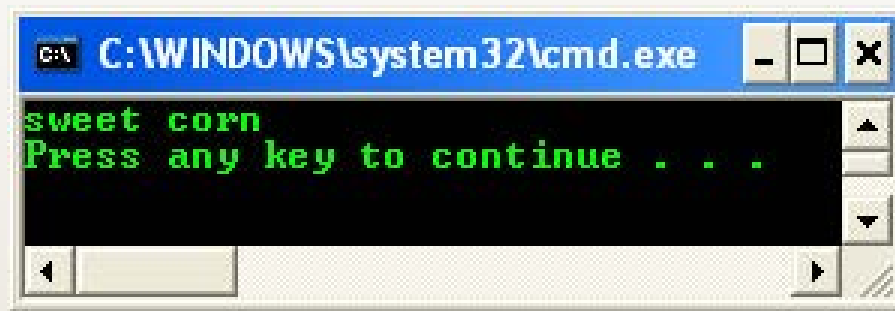
- Program example 4: Searching the location for the given value in an array



```
C:\WINDOWS\system32\cmd.exe
Enter 4 floats: 3.2 4.5 1.2 6.7
The scores in reverse order are:
6.70    1.20    4.50    3.20
Press any key to continue . . .
```

ARRAYS

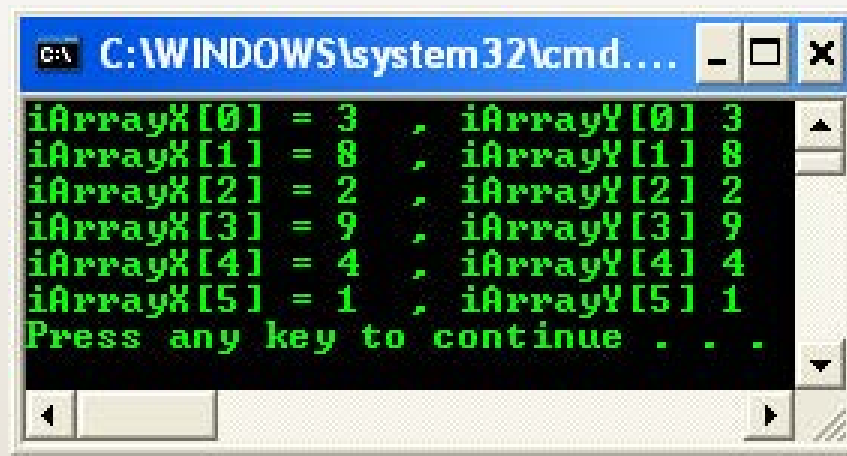
- Program example 5: Storing and reading a string




A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the text "sweet corn" on the first line and "Press any key to continue . . ." on the second line. The text is displayed in green on a black background. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

ARRAYS

- Program example 6: Storing and reading array content and its index

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd...". The window contains the following text in green on a black background:

```
iArrayX[0] = 3 , iArrayY[0] 3  
iArrayX[1] = 8 , iArrayY[1] 8  
iArrayX[2] = 2 , iArrayY[2] 2  
iArrayX[3] = 9 , iArrayY[3] 9  
iArrayX[4] = 4 , iArrayY[4] 4  
iArrayX[5] = 1 , iArrayY[5] 1  
Press any key to continue . . .
```

An orange arrow pointing to the right, located on the left side of the slide.

ARRAYS

Two Dimensional/2D Arrays

- A two dimensional array has two subscripts/indexes.
- The first subscript refers to the row, and the second, to the column.
- Its declaration has the following form,

```
data_type    array_name[1st dimension size][2nd dimension size];
```

- For examples,

```
int          xInteger[3][4];  
float        matrixNum[20][25];
```

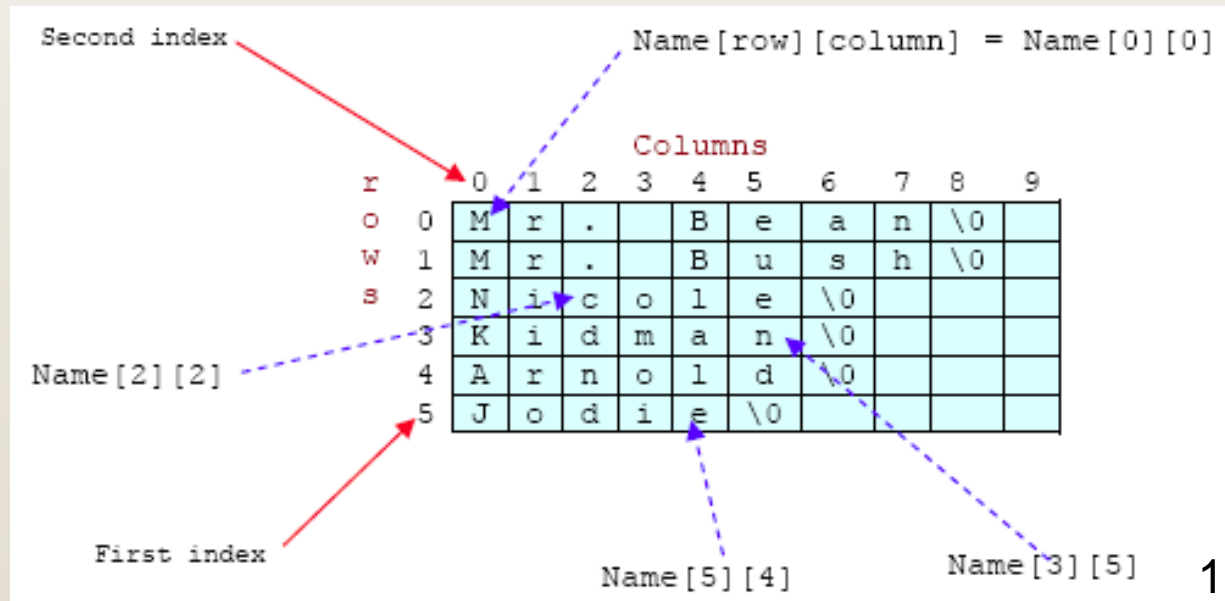
- The first line declares `xInteger` as an integer array with 3 rows and 4 columns.
- Second line declares a `matrixNum` as a floating-point array with 20 rows and 25 columns.

ARRAYS

- 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.



ARRAYS

- 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 $6 \times 10 = 60$ and equal to the number of the colored square. In general, for

```
array_name[x][y];
```

- The array size is = First index \times second index = xy .
- This also true for other array dimension, for example three dimensional array,

```
array_name[x][y][z]; => First index  $\times$  second index  $\times$  third index =  $xyz$ 
```

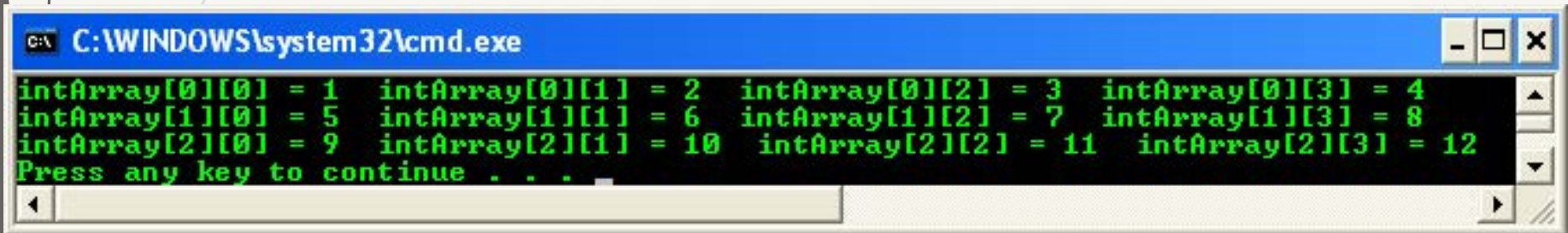
- For example,

```
ThreeDimArray[2][4][7] =  $2 \times 4 \times 7 = 56$ .
```

- And if you want to illustrate the 3D array, it could be a cube with wide, long and height dimensions.

ARRAYS

- Program example 7: Storing and reading array content and its index



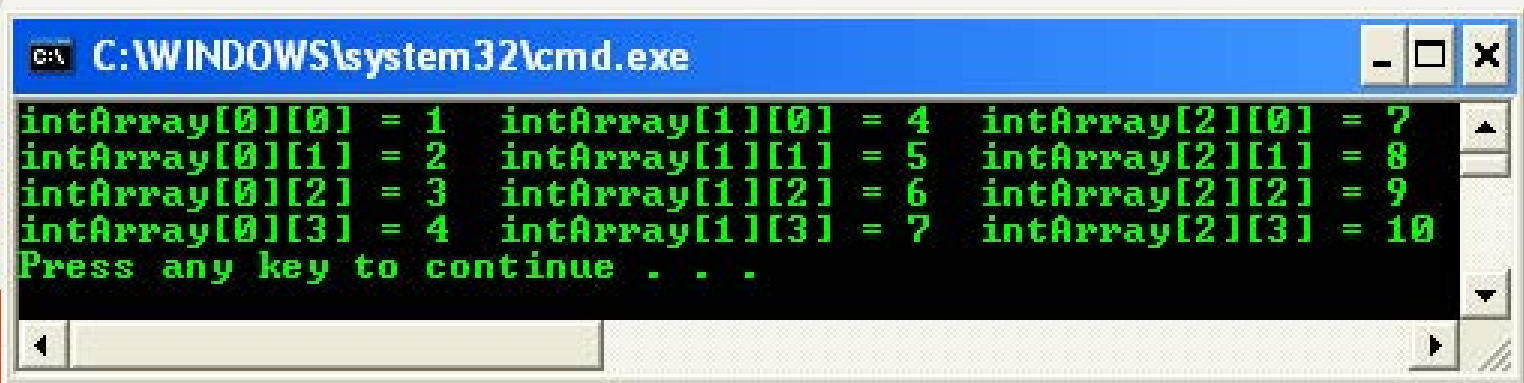
A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays the following text in green on a black background:

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

The window has a blue title bar and standard Windows window controls (minimize, maximize, close) in the top right corner. A scroll bar is visible at the bottom of the window.

ARRAYS

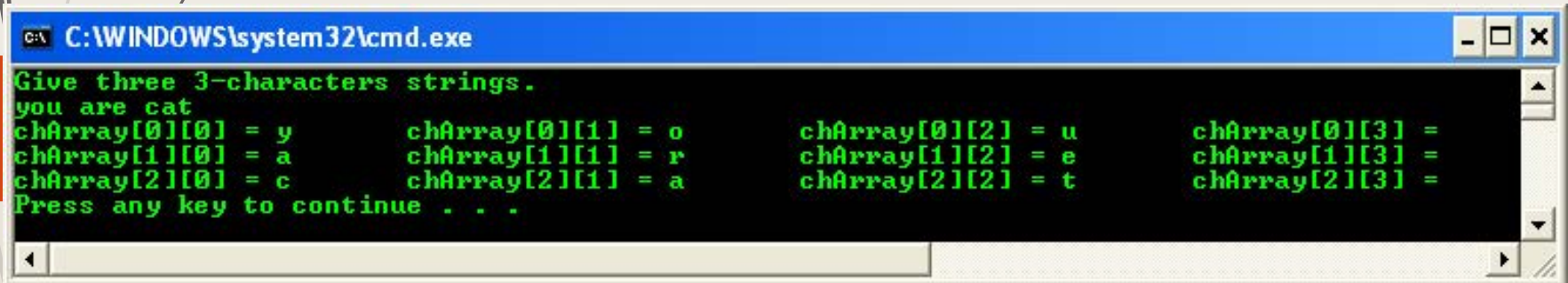
- Program example 8: Swapping iIndex (iRow) with jIndex (iColumn) in the previous program example



```
C:\WINDOWS\system32\cmd.exe
intArray[0][0] = 1   intArray[1][0] = 4   intArray[2][0] = 7
intArray[0][1] = 2   intArray[1][1] = 5   intArray[2][1] = 8
intArray[0][2] = 3   intArray[1][2] = 6   intArray[2][2] = 9
intArray[0][3] = 4   intArray[1][3] = 7   intArray[2][3] = 10
Press any key to continue . . .
```

ARRAYS

1. Program example 9: Strings are read in by the rows.
2. Each row will have one string. Enter the following data: “you”, “are”, “cat” for the following example.
3. Remember that after each string, a null character is added.
4. We are reading in *strings* but printing out only *characters*.



```
C:\WINDOWS\system32\cmd.exe
Give three 3-characters strings.
you are cat
chArray[0][0] = y      chArray[0][1] = o      chArray[0][2] = u      chArray[0][3] =
chArray[1][0] = a      chArray[1][1] = r      chArray[1][2] = e      chArray[1][3] =
chArray[2][0] = c      chArray[2][1] = a      chArray[2][2] = t      chArray[2][3] =
Press any key to continue . . .
```

ARRAYS

- The contents of the array in memory after the three strings are read in the array.

	0	1	2	3
0	y	o	u	\0
1	a	r	e	\0
2	c	a	t	\0

- Re-run the program, enter the following data: “you”, “my”. Illustrates the content as done previously.

	0	1	2	3
0	y	o	u	\0
1	m	y	\0	
2	l	u	v	\0

ARRAYS

1. Does your output agree?
2. How is the null character, '\0' printed?
3. Is there a garbage character in `a[1][3]`? If so, why?
 - a) The output matched, except the *garbage*.**
 - b) Just an empty space.**
 - c) Yes. This slot has been reserved but not filled so whatever the previous data that has been stored here would be displayed (if possible).**

End-of-C-arrays