C ARRAYS

-a collection of same type data, 1D, 2D-

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?

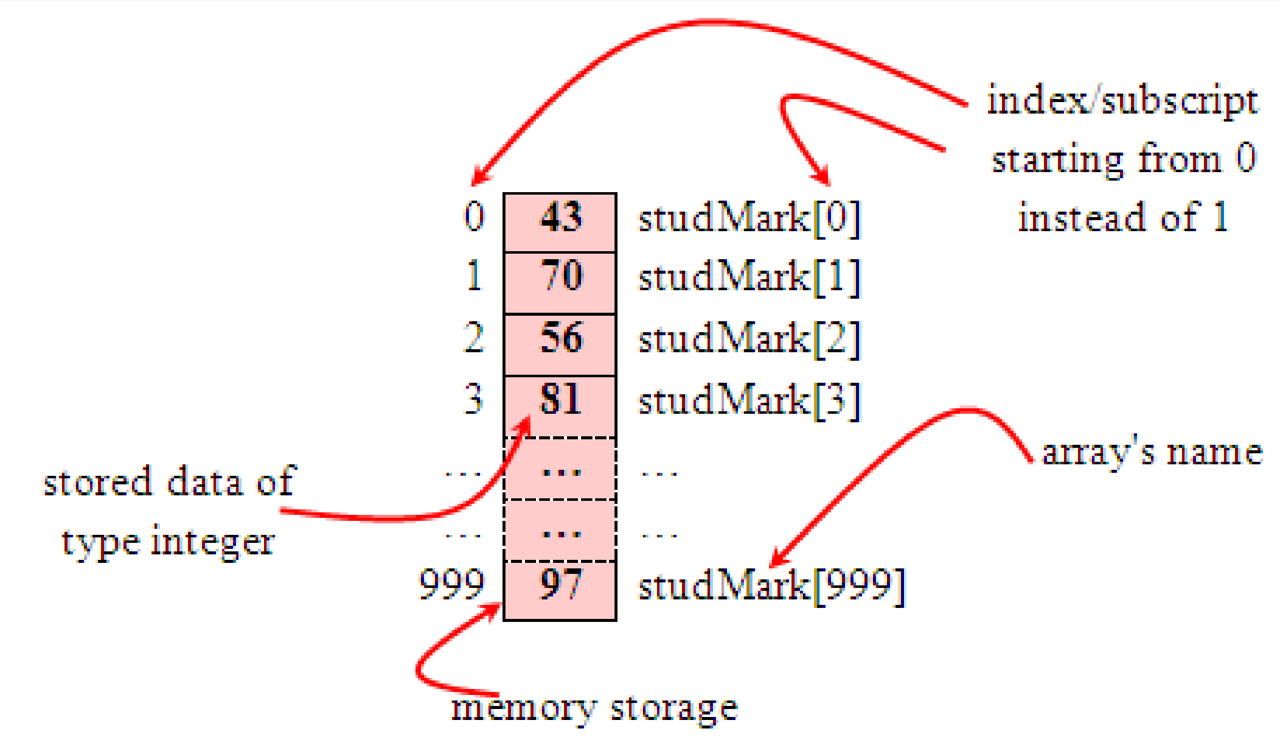
```
int main(void)
{
    int studMark1, studMark2, studMark3,
    studMark4, …, …, studMark998, stuMark999,
    studMark1000;
    …
    …
    return 0;
}
```

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

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];
```

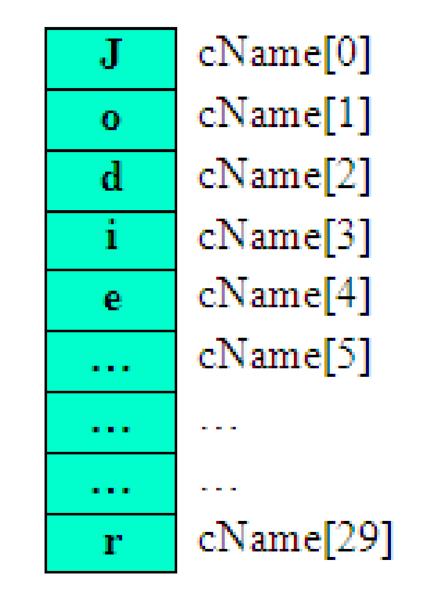
- Here, *array_element_data_type* define the base type of the array, which is the type of each element in the array.
- *array_name* is any valid identifier name that obeys the same rule for the identifier naming.
- *array_size* defines how many elements the array will hold.

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,



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

- 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.

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

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];
```

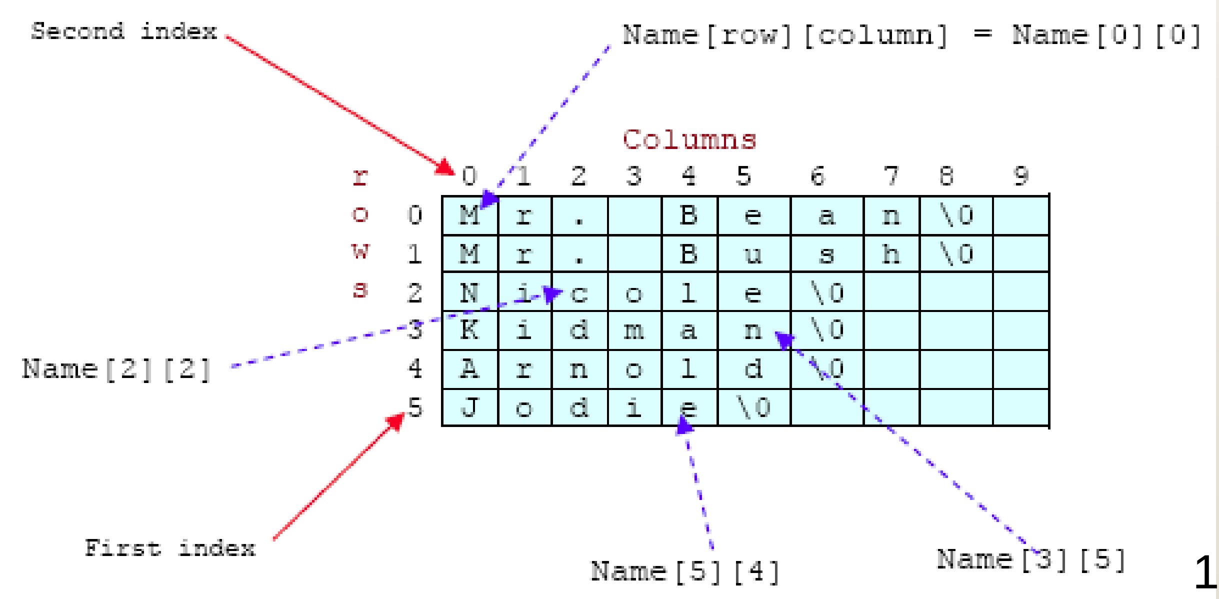
- The first line declares `xInteger` as an integer array with 3 rows and 4 columns.
- Second line declares a `matrixNum` as a floating-point array with 20 rows and 25 columns.

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.



	0	1	2	3	4	5	6	7	8	9
0	M	r	.		B	e	a	n	\0	
1	M	r	.		B	u	s	h	\0	
2	N	i	c	o	l	e	\0			
3	K	i	d	m	a	n	\0			
4	A	r	n	o	l	d	\0			
5	J	o	d	i	e	\0				

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 x 10 = 60 and equal to the number of the colored square. In general, for

```
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 dimensions.

End-of-C-arrays