What is an Array of Strings?

A string is a 1-D array of characters, so an array of strings is a 2-D array of characters. Just like we can create a 2-D array of `int`, `float` etc; we can also create a 2-D array of character or array of strings. Here is how we can declare a 2-D array of characters.

```
char ch_arr[3][10] = {
                        {'s', 'p', 'i', 'k', 'e', '\0'},
                       {'t', 'o', 'm','\0'},
                        {'j', 'e', 'r', 'r', 'y','\0'}
                    };
```

It is important to end each 1-D array by the null character, otherwise, it will be just an array of characters. We can't use them as strings.

Initializing a String: A string can be initialized in different ways. We will explain this with the help of an example. Below is an example to declare a string with name as str and initialize it with “HelloGcitians”.

```
1. char str[] = “HelloGcitians”;

2. char str[50] = “HelloGcitians”;

3. char str[] = {'H','e','l','l','o','G','c','i','t','i','a','n','s','\0'};

4. char str[14] = {'H','e','l','l','o','G','c','i','t','i','a','n','s','\0'};
```

Declaring an array of strings this way is rather tedious, that's why C provides an alternative syntax to achieve the same thing. This above initialization is equivalent to:

```
char ch_arr[3][10] = {
                        "spike",
                        "tom",
                        "jerry"
                    };
```

The first subscript of the array i.e `3` denotes the number of strings in the array and the second subscript denotes the maximum length of the string. Recall the that in C, each character occupies `1` byte of data, so when the compiler sees the above statement it allocates `30` bytes (`3*10`) of memory.

We already know that the name of an array is a pointer to the 0th element of the array. Can you guess the type of `ch_arr`?

The `ch_arr` is a pointer to an array of `10` characters or `int(*)[10]`.

Therefore, if `ch_arr` points to address `1000` then `ch_arr + 1` will point to address `1010`.

From this, we can conclude that:

`ch_arr + 0` points to the 0th string or 0th 1-D array.
`ch_arr + 1` points to the 1st string or 1st 1-D array.
`ch_arr + 2` points to the 2nd string or 2nd 1-D array.

In general, `ch_arr + i` points to the ith string or ith 1-D array.

We know that when we dereference a pointer to an array, we get the base address of the array. So, on dereferencing `ch_arr + i` we get the base address of the 0th 1-D array.

From this we can conclude that:

`*(ch_arr + 0) + 0` points to the 0th character of 0th 1-D array (i.e `s`)
`*(ch_arr + 0) + 1` points to the 1st character of 0th 1-D array (i.e `p`)
`*(ch_arr + 1) + 2` points to the 2nd character of 1st 1-D array (i.e `m`)

In general, we can say that: `*(ch_arr + i) + j` points to the jth character of ith 1-D array.

Note that the base type of `*(ch_arr + i) + j` is a pointer to `char` or `(char*)`, while the base type of `ch_arr + i` is array of 10 characters or `int(*)[10]`.

To get the element at jth position of ith 1-D array just dereference the whole expression`*(ch_arr + i) + j`.

```
*(*(ch_arr + i) + j)
```

We have learned in chapter Pointers and 2-D arrays that in a 2-D array the pointer notation is equivalent to subscript notation. So the above expression can be written as follows:

```
ch_arr[i][j]
```

The following program demonstrates how to print an array of strings.

```
#include<stdio.h>

int main()
{
   int i;

   char ch_arr[3][10] = {
                                 "spike",
                                 "tom",
                                 "jerry"
                          };

   printf("1st way \n\n");

   for(i = 0; i < 3; i++)
   {
      printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);
   }

   // signal to operating system program ran fine
   return 0;
}
```

Expected Output:

```
1string = spike address = 2686736
2string = tom address = 2686746
```

3string = jerry address = 2686756