Unit - 5

Introduction to Arrays

Array

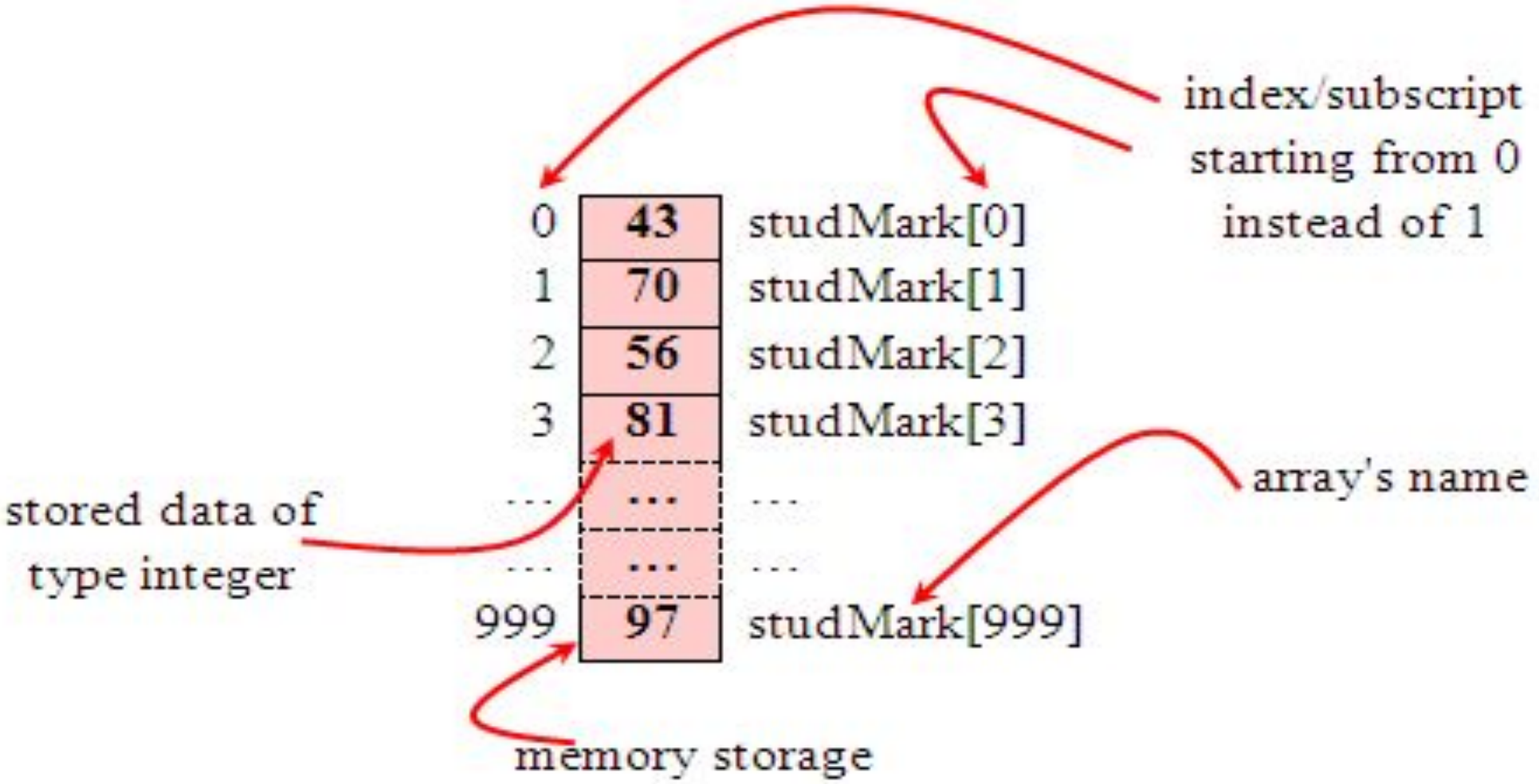
- **An array is a variable that can store multiple values.**
- For example, if you want to store 100 integers, you can create an array for it.
- An array is a collection of elements of the same type that are referenced by a common name.
- Compared to the basic data type (int, float & char) it is a derived data type.
- 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**

Array

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

- By using an array, we just declare like this,
  - int studMark[1000];
- This will reserve 1000 contiguous memory locations for storing the students' marks.

Array


index/subscript
starting from 0
instead of 1
0
1
2
3
...
...
999
43
70
56
81
...
...
97
studMark[0]
studMark[1]
studMark[2]
studMark[3]
...
...
studMark[999]
stored data of
type integer
array's name
memory storage

Array

- We can use index or subscript to identify each element or location in the memory.
- For example, studMark[0] will refer to the first element of the array.
- Thus by changing the value of index, we could refer to any element in the array.
- So, array has simplified our declaration and manipulation of the data.

Array Dimension

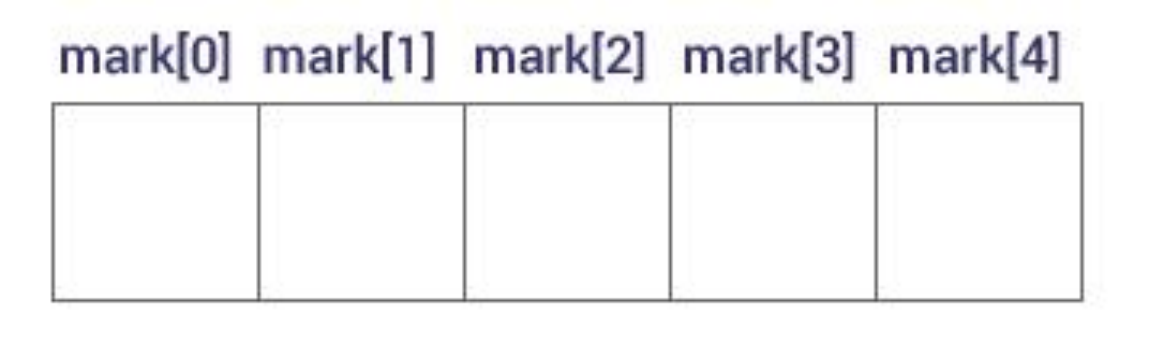
- Dimension refers to the array's size, which is how big the array is.
- A single or one dimensional array declaration has the following form,

  - Syntax : 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 identifier name that obeys the same rule for the identifier naming.
- *array_size* defines how many elements the array will hold.
- **Note:** the size and type of an array cannot be changed once it is declared.

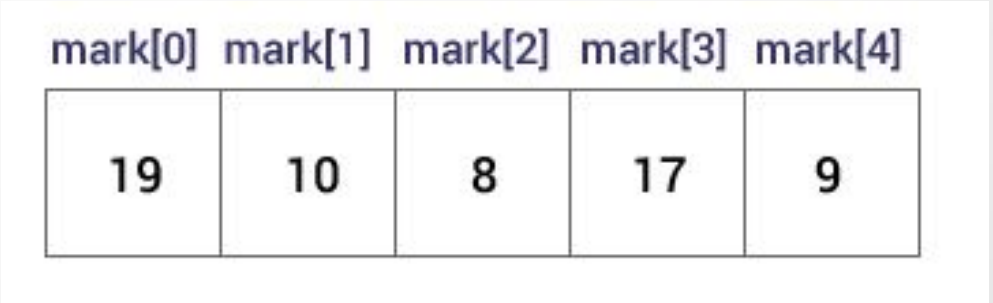
Array: Access Array Elements

- **Access Array Elements**
  - **float mark[5];**
  - The first element is mark[0], the second element is mark[1] and so on.
  - Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
  - If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]



Array: initialize

- Initialize an array
- int mark[5] = {19, 10, 8, 17, 9};
- int mark[] = {19, 10, 8, 17, 9};
- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.
  - mark[0] is equal to 19
  - mark[1] is equal to 10
  - mark[2] is equal to 8
  - mark[3] is equal to 17
  - mark[4] is equal to 9



Array Dimension

- For example, to declare an array of 30 characters, that construct a people name, we could declare,
  - char cName[30];
- 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.

Two Dimensional Array

- A two dimensional array has two subscripts/indexes. It is also called Multi dimensional array.
- **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 example,
  - int number[3][4];
  - float number_1[20][25];
- The first line declares variable number as an integer array with 3 rows and 4 columns.
- Second line declares a variable number_1 as a floating-point array with 20 rows and 25 columns.

Two Dimensional Array

- float x[3][4];
- Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]