-Arrays-

Programming Fundamentals

Background

- So far we have used only the fundamental data types (int, char, float, double)
- They are constrained by the fact that a variable of these types can store **only one value at any given time**. Therefore, they can be used only to handle limited amounts of data.
- In many applications, however, we need to handle a large volume of data in terms of reading, processing and printing.

Array- Definition

- **An array is a sequenced collection of elements of the same data type (homogeneous;** that share a common name)**.**
- It is simply a grouping of like-type(same datatype) data.
- In its simplest form, an array can be used to represent a list of numbers, or a list of names for e.g.
- Some examples where the concept of an array can be used:

1. List of temperatures recorded every hour in a day, or a month, or a year.
2. List of employees in an organization.
3. List of products and their cost sold by a store.
4. Test scores of a class of students.

- Arrays and structures are referred to as **structured datatype** because they can be used to represent data values that have a structure of some sort.
- Structured data types provide an **organizational scheme** that shows the relationships among the individual elements and facilitate efficient data manipulations. (0,1,2…..)
- Array has the ability to **use a single name to represent a collection of items (by using array-name and its index)** and to refer to an item by specifying the item number allows us to write efficient and concise programs.

For e.g.: int student[75];

Here, students is an array of 75 items each of which is an int.

- Size of student array: 4 x 75 =300B

ONE-DIMENSIONAL ARRAYS

- **One-dimensional array/Single-scripted variable**: A list of items can be given one variable name using only one subscript and such a variable is called **Single-scripted variable**.
- The subscripted variable x_i (in prog: x[i]) refers to the ith element of single-subscripted variable i can be expressed as x[1], x[2], x[3],.........x[n]
- The subscript can begin with number 0, i.e. x[0]

- DECLARATION OF ONE-DIMENSIONAL ARRAYS:

Like any other variable, arrays must be declared before they are used so that the compiler can allocate space for them in memory. The general form of array declaration is:

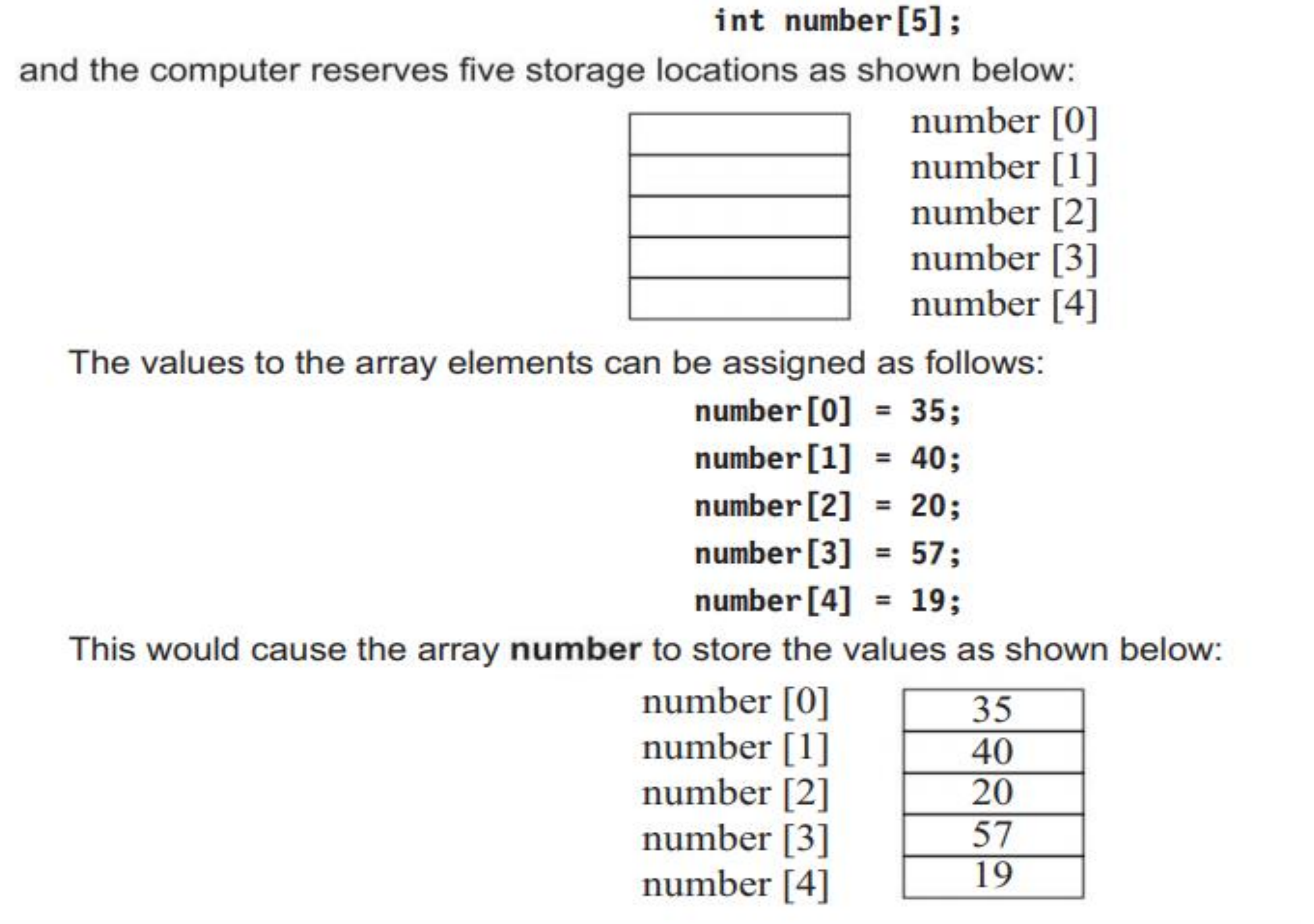
data-type variable-name[size];

data-type: specifies datatype of elements inside an array (char,float,int..)

variable-name: name of an array (using valid identifier)

size: indicates max no. of elements inside an array.

- Example: Continuous Memory Organization of an array



```
int number[5];
```

and the computer reserves five storage locations as shown below:

	number [0]
	number [1]
	number [2]
	number [3]
	number [4]

The values to the array elements can be assigned as follows:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

This would cause the array **number** to store the values as shown below:

number [0]	35
number [1]	40
number [2]	20
number [3]	57
number [4]	19

- Some more examples:

1. float totalMarks[50];
2. int rollNumbers[10];
3. short subjects[5];
4. char names[4] ={‘a’,’b’,’c’,’\0’};

Character Array:

C language treats string as character array where each character in string is treated as an element of array.

e.g. “abc” → arr[0]=‘a’, arr[1]=‘b’, arr[2]=‘c’, **arr[3]=‘\0’ (null character)**

When the compiler sees a character string, it terminates it with an additional null character. Thus, the element name[10] holds the null character ‘\0’.

Note: Whenever we declare size of a character array, provide one extra space for \0 (null character).

INITIALIZATION OF ONE-DIMENSIONAL ARRAYS

- Once array is declared, it is initialized. It can be done:

1. At compile time
2. At run time

Compile Time Initialization:

- Done in the same way as the ordinary variables are declared.
- Format: **type array-name[size] = { list of values };**
- The values in the list are separated by commas.
- E.g.:

a. int number[3] = { 0,0,0 }; //declares an array named number that can hold 3 elements and each elements is of int datatype. It also initializes all elements with value 0.

b. float total[5] = {0.0,15.75,–10}; //declares an array named total that can hold 5 elements and each elements is of float datatype. It also initializes all elements with 0.0,15.75,–10 respectively.

c. char name[ ] = {‘J’,‘o’, ‘h’, ‘n’, ‘\0’}; //declares a character array named total where each elements is of char datatype. It also initializes all elements with ‘J’,‘o’, ‘h’, ‘n’, ‘\0’ respectively.

Note:

1. When compile-time initialization is done, we can skip writing size. (example c)
2. If we have more initializers than the declared size, the compiler will produce an error. int number [3] = {10, 20, 30, 40};

It is illegal in C

3. Shortcut for initializing array with default value: (both are same)

int group [10] = {0,0,0,0,0,0,0,0,0,0};

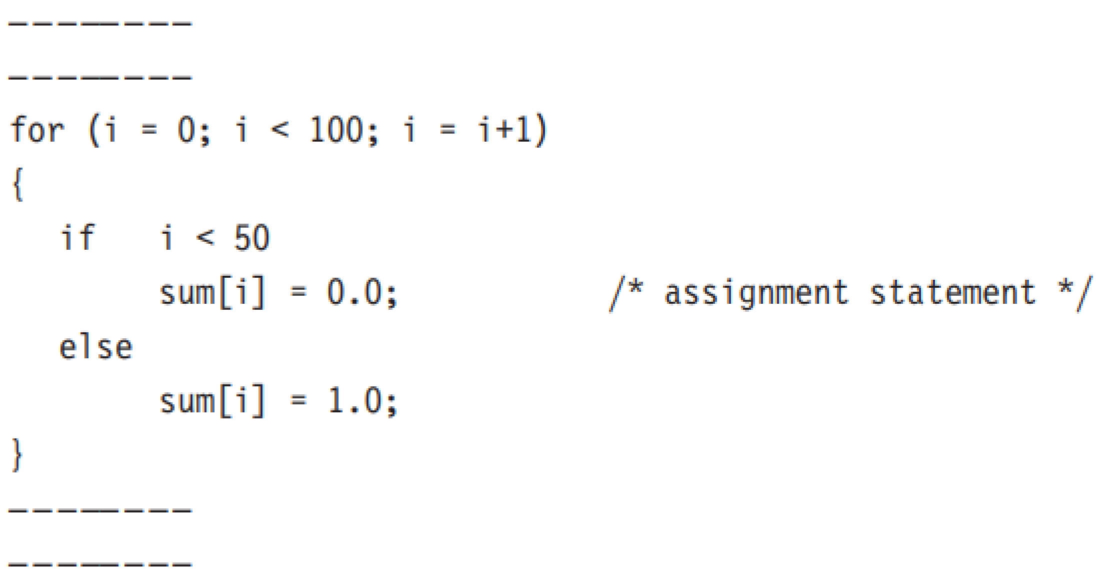
int group [10] = {0};

- Run Time Initialization:

An array can be explicitly initialized at run time.

This approach is usually applied for initializing large sized array.

Way-1.: Using iteration/loop



```
--------
--------
for (i = 0; i < 100; i = i+1)
{
   if    i < 50
         sum[i] = 0.0;            /* assignment statement */
   else
         sum[i] = 1.0;
}
--------
--------
```

- Way-2: Using scanf

Can use a read function such as scanf to initialize an array.

For example,

int x [3];

scanf(“%d%d%d”, &x[0], &[1], &x[2]);

will initialize array elements with the values entered through the keyboard.

Basic operations on Arrays

Basic Operations:

- **Traverse** − reach out (print) all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

- Searching and sorting are the two most frequent operations performed

on arrays.

- Several data structures for searching and sorting techniques are devised.
- Sorting: process of arranging elements in the list according to their values, in ascending or descending order. (sorted list)
- Sorted lists are especially important in list searching because they facilitate rapid search operations.
- Examples of sorting techniques:

1. Bubble Sort
2. Selection sort
3. Insertion sort
4. Shell sort
5. Merge sort
6. Quick sort

- Searching: finding the position of specified element (**search key**) in a given list.
- Successful search- if process of searching finds a match of the search key within an list of elements.
- Unsuccessful search- if search key is not found in the provided list.
- Example:

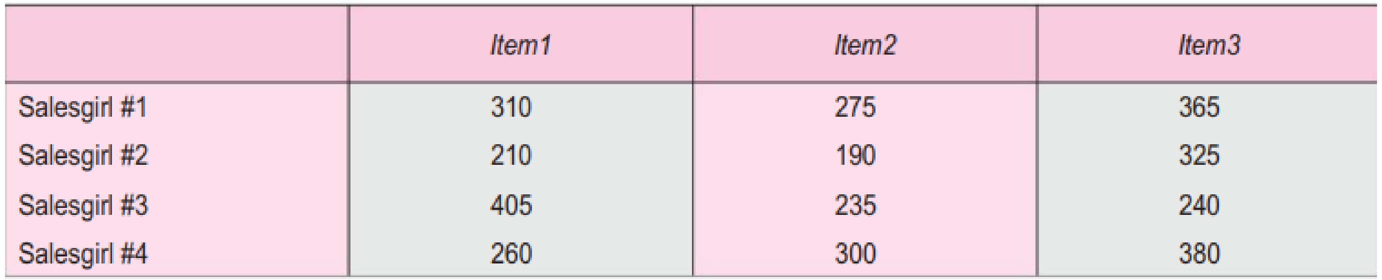
1. Sequential/Linear search (0th,1st,2nd, …… last index)
2. Binary Search (Pre-requisite: list should be sorted-asc/desc)

1,3,0,4 (not sorted)

2 4 5 6 8 (sorted)

TWO-DIMENSIONAL ARRAYS

- Used to store table of values. Also called **matrix**.
- E.g.: Matrix for sales of three items by four sales girls:



	Item1	*Item2*	*Item3*
Salesgirl #1	310	275	365
Salesgirl #2	210	190	325
Salesgirl #3	405	235	240
Salesgirl #4	260	300	380

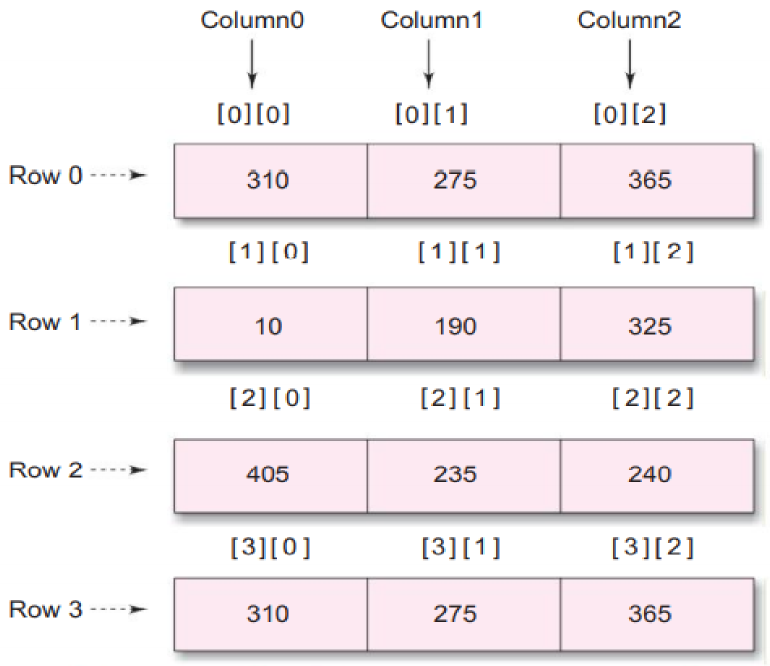
- In mathematics, we represent a particular value in a matrix by using two subscripts such as $sales_{ij}$

  sales: denotes the entire matrix (as well as name)
- $sales_{ij}$: refers to the value in the i-th row and j-th column.

- Syntax for declaration:

```
type array_name [row_size][column_size];
```

- Total size(#elements) in 2Darray= row_size x column_size
- Memory representation of 2-D array: (contiguous)
- Consider same sales example for memory representation



	Column0	Column1	Column2
Row 0	[0][0] 310	[0][1] 275	[0][2] 365
Row 1	[1][0] 10	[1][1] 190	[1][2] 325
Row 2	[2][0] 405	[2][1] 235	[2][2] 240
Row 3	[3][0] 310	[3][1] 275	[3][2] 365

INITIALIZING TWO-DIMENSIONAL ARRAYS

1) Way 1: Same as 1-D array (one after the other for each row)

e.g. int table[2][3] = { 0,0,0,1,1,1};

2) Way-2: Row by row

e.g.: int table[2][3] = {{0,0,0}, {1,1,1}};

Note:

1. When the array is completely initialized with all values, explicitly, we need not specify the size of the first dimension.

e.g.: int table[][3] = {{0,0,0}, {1,1,1}}; //no need to specify row/first dimension

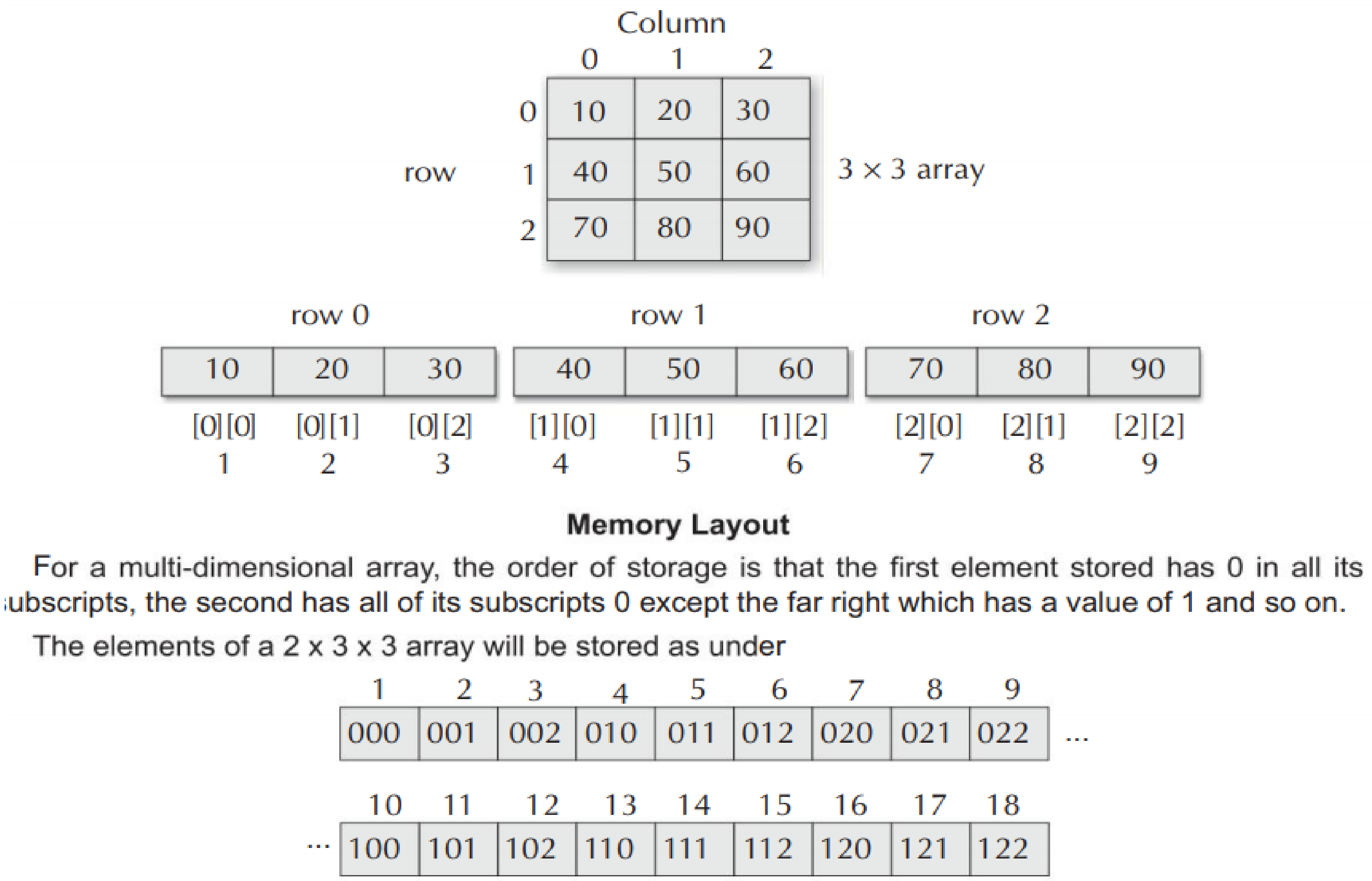
2. Commas are required after each bracket that closes off a row.

- When all the elements are to be initialized to zero, the following short-cut methods (two) may be used.

Way-1: int m[3][5] = { {0}, {0}, {0}};

Way-2: int m [3] [5] = { 0, 0};

- Memory Layout:



		Column 0	 1	 2
	0	10	20	30
row	1	40	50	60
	2	70	80	90

3 × 3 array

row 0			row 1			row 2		
10	20	30	40	50	60	70	80	90
[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]
1	2	3	4	5	6	7	8	9

Memory Layout

For a multi-dimensional array, the order of storage is that the first element stored has 0 in all its subscripts, the second has all of its subscripts 0 except the far right which has a value of 1 and so on.

The elements of a 2 x 3 x 3 array will be stored as under

1	2	3	4	5	6	7	8	9	
000	001	002	010	011	012	020	021	022	...

	10	11	12	13	14	15	16	17	18
...	100	101	102	110	111	112	120	121	122

MULTI-DIMENSIONAL ARRAYS

- C allows arrays of three or more dimensions. The exact limit is determined by the compiler.
- Format: **array_name[s1][s2][s3]....[sm];**

where si: size of th dimension.

E.g.:

1. int survey[3][5][12]; // 3-D array with size of 180(3x5x12) integer type elements.

2. float table[5][4][5][3]; // 4-D array with size of 300(5x4x5x3) integer type elements.