UNIT – 1 (INTRODUCTION TO DATA STRUCTURE)

INTRODUCTION

There are a number of ways to write programs: they must run correctly and efficiently; be easy to read and understand; be easy to debug and; be easy to modify.

Data is the basic fact or entity that is utilized in calculation or manipulation.

There are two different types of data such as **numerical data** and **alphanumeric data**.

These two data types specify the nature of data item that go through certain operations. Integers and floating-point numbers are of numerical data type.

Data may be a single value or it may be a set of values. Whether it is a single value or a group of values to be processed must be organized in a particular fashion. This organization leads to structuring of data.

The intimate relationship between data and programs can be traced to the beginning of computing. In any area of application, the input data, (internally stored data) and output data may each have a unique structure.

Definitions

Data structure is representation of the logical relationship existing between individual elements of data.

OR

In other words, a data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

OR

We can also define it as a mathematical or logical model of particular organization of data items.

Data structure mainly specifies the following four things:

1. Organization of data.
2. Accessing methods.
3. Degree of associativity.
4. Processing alternatives for information.

Data structures are the building blocks of a program. And hence the selection of a particular data structure stresses on the following two things.

1. The data structures must be rich enough in structure to reflect the relationship existing between the data.
2. And the structure should be simple so that we can process data effectively whenever required.

The identification of the inherent data structure is vital in nature. And the structure of input and output data can be use to derive the structure of a program. Data structure affects the design of both structural and functional aspects of a program.

Algorithm + Data structure = Program

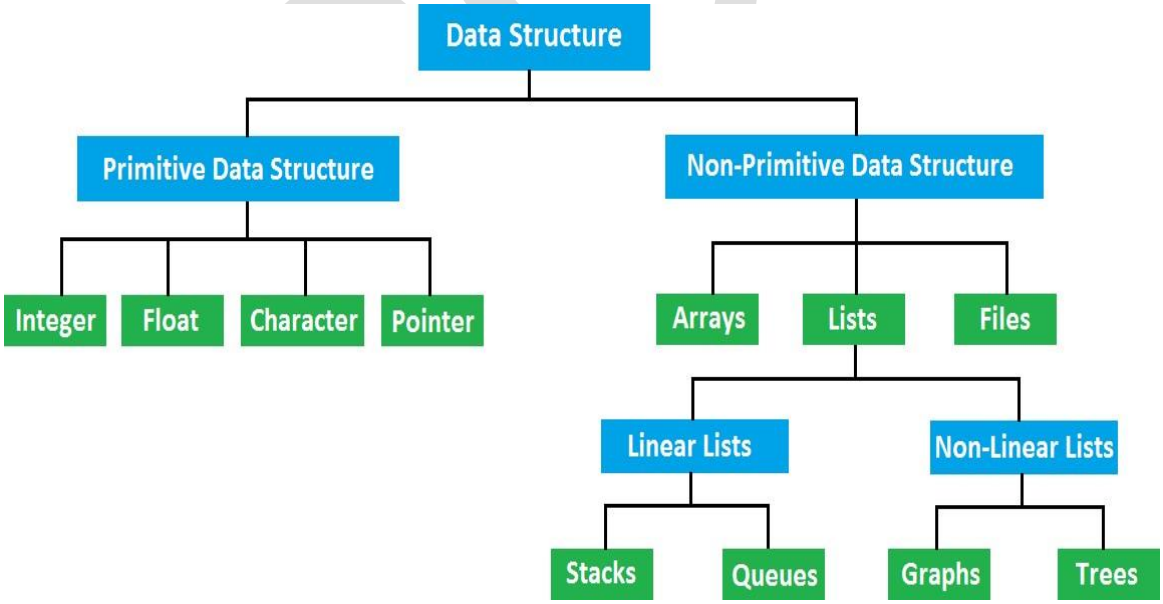
An algorithm is a step-by-step procedure to solve a particular function. That is, it is a set of instructions written to carry out certain tasks and the data structure is the way of organizing the data with their logical relationship retained.

To develop a program of an algorithm, we should select an appropriate data structure for that algorithm. Therefore, algorithm and its associated data structures form (creates) a program.

*CLASSIFICATION OF DATA STRUCTURE (FAMILY OF D. S.)

Data structures are normally divided into **two** broad categories:

1. Primitive data structures.
2. Non-primitive data structures.



Classification of Data Structures

Primitive Data Structure

The data structures that are directly operated by machine instruction are called primitive data structure.

i.e. Integer, floating-point numbers, character constants, string constants, pointers etc.

Non – Primitive Data Structure

The data structures that are not directly operated by machine instruction is called Non- Primitive Data Structure.

These data structures are derived from the primitive data structures. The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

i.e. Arrays, Lists and Files.

Operations on Data Structure

The most commonly used operations on Data Structure are:

1. CREATE
2. DESTROY or DELETE
3. SELECTION
4. UPDATION

1. **CREATE:** This operation results in reserving memory for the program elements (variables). This can be done by declaration statement. The creation of data structure may take place either during compile-time or during run-time.

2. **DESTROY:** This operation destroys the memory space allocated for the specified data structure. malloc( ) and free( ) function of C language are used for these two operations respectively (as far as dynamic memory allocation and de-allocation is concerned).

3. **SELECTION:** This operation deals with accessing a particular data within a data structure.

4. **UPDATE:** This operation updates or modifies the data in the data structure. Probably new data may be entered or previously data may be deleted.

Other operations performed on data structures include:

5. **SEARCHING:** This operation finds the presence of the desired data item in the list of data item. It may also find the locations of all elements that satisfy certain conditions.

6. **SORTING:** This is the process of arranging all data items in a data structure in a particular order say for example, either in ascending order or in descending order.

7. **MERGING:** This is a process of combining the data items of two different sorted lists into a single sorted list.

*APPLICATIONS OF DATA STRUCTURE

- **Applications of an Array**: Implementation of other data structures, Execution of matrices and vectors, Dynamic memory allocation, Pointer container, Control tables
- **Applications of Stack**: Evaluation of expressions, Backtracking, Runtime memory management, Arrangement of books in a library
- **Applications of Queue**: Disk scheduling, CPU scheduling, File IO, Data transmission
- **Applications of linked list**: web crawling, Representation of sparse matrices, Non-contiguous data storage, Implementation of non-binary tree or other data structures, Dynamic memory management, Equalizing parenthesis, Symbol tables
- **Applications of Graph**: Computer networking, Problem solutions involving 'Depth- First' search or 'Breadth-First' search algorithms, Representation of matrices, Study of molecular interactions in Chemistry
- **Applications of Tree**: Representation of data lists, Quickly accessible data storage, Representation of hierarchal data, Routing of algorithms
- **Applications of Files**: Implementation of computer programs, Data comparison, Storage of data having varying data types

*ADVANTAGES OF DATA STRUCTURE

1. Allows easier processing of data.
2. It allows information stored on disk very efficiently.
3. These are necessary for designing an efficient algorithm.
4. It provides management of databases like indexing with the help of hash tables and arrays.
5. We can access data anytime and anywhere.
6. It is secure way of storage of data.
7. Graphs models real life problems
8. It allows processing of data on software system

*DISADVANTAGES OF DATA STRUCTURE

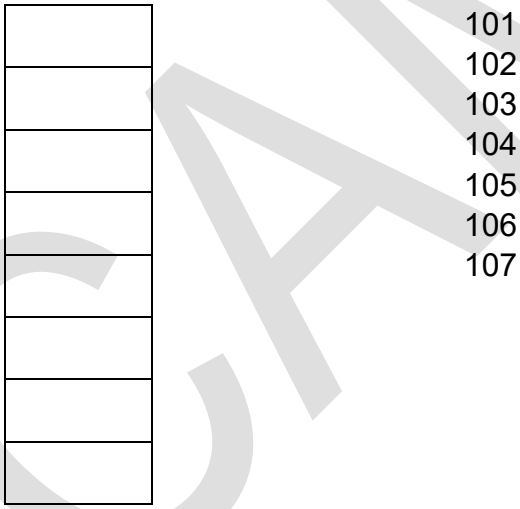
1. It is applicable only for advanced users.

2. If any issue occurs, it can be solved only by experts.
3. Slow access in case of some data types

*INTRODUCTION TO ARRAYS

If we want to store a group of data together in one place, then array is suitable data structure. This data structure enables us to arrange more than one element that's why it is known as composite data structure.

In this data structure, all the elements are stored in contiguous location of memory. Figure shows an array of data stored in a memory block starting at location 101.



Definition

An array is a finite, ordered and collection of homogeneous data elements.

- Array is a **finite** because it contains only limited number of elements.
- Array is **ordered** because all the elements are stored one by one in contiguous location of computer memory in a linear ordered fashion.
- Array is **collection of homogeneous (identical) elements** because all the elements of an array are of the same data type only.

An array is known as **linear data structure** because all elements of the array are stored in a linear order.

e.g. 1. an array of integer to store the age of all student in a class

2. an array of string of characters to store name of all villagers in village

Terminology

1. **Size:** Number of elements in an array is called the size of the array. It is also known as dimension or length.

2. **Type:** Type of an array represents the kind of data type for which it is meant (designed). E.g. array of integer, array of character.

3. **Base:** Base of an array is the address of memory location where the first element in the array is located.
   E.g. 101 is the base address of the array mentioned in Figure.

4. **Index:** All the elements in an array can be referred by a subscript like Ai or A[i], this subscript is known as index. An index is always an integer value.

5. **Range of indices:** Indices of an array element may change from lower bound (L) to an upper bound (U). These bounds are called the boundaries of an array.

If the rage of index varies from L...U then size of the array can be calculated as:
Size (A) = U – L + 1

***ONE DIMENSIONAL ARRAYS**

A one dimensional array is one in which only one subscript specification is needed to specify a particular element of the array.

Declaration of One dimensional array

One dimensional array can be declared as follows:

data_type var_name [expression];

- data_type is the type of elements to be stored in the array.
- var_name specifies the name of array. It may be given any name like other simple variables.
- expression or subscript specifies the number of values to be stored in the array.

e.g. **int n[5];** This array will store seven integer values, which is specified by n. It can be visualized as below.

Index	Data
n[0]	5
n[1]	48
N[2]	0
n[3]	80
n[4]	21

Initializing one dimensional array

Array variables can be initialized in declarations by constant initializers.

These initializing expressions must be constant values. Expression with identifiers or function calls may not be used in the identifiers. The initializers are specified within braces and separated by commas.

e.g. int n[10] = {2, 53, 19, 41, 46, 22}; char s[10] = {'a', 'd', 'f', 's', '\0'};

String initializers may be written as string constants instead of character constants within braces.

e.g. char str[ ] = "This is a message"; char s[20] = "Hello World";

In the case of str[ ], enough memory is allocated to accommodate the string plus a terminating NULL character and we do not need to specify the size of the array.

Accessing one dimensional array elements

Individual elements of the array can be accessed using the following syntax:
array_name [index or subscript];

Example:

1. To access third element from array we write: **n[2]**

   The subscript for third element is 2 because the lower bound of array in C is 0.

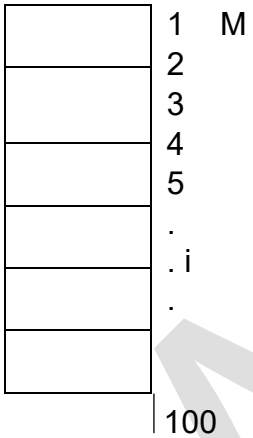
2. To assign a value to second location of the array, we write: **n[1] = 20;**

3. To read a particular value we write: **scanf ("%d", &n[2]) ;**

This statement reads a value from the keyboard and assigns it to third location of the array.

*MEMORY ALLOCATION AND ADDRESS CALCULATION OF ELEMENTS OF ONE DIMENSIONAL ARRAYS

Suppose an array A[100] is to be stored in a memory as in Figure. Let the memory location where the first element can be stored is M.



An array can be written as A[L..U] where L denotes lower bound and U denotes the upper bound for index.

TO FIND ADDRESS OF AN ELEMENT IN ONE DIMENSIONAL ARRAY WITH LOWER BOUND = 1

- **If an element in one dimensional array occupy 1 word (byte):**

If each element requires one word (byte) then the location for any element say A[i] in the array can be obtained as:

Address (A[i]) = M + (i – 1) **[Note:** Lower bound of array is assumed to be 1.]

M	Base address of array
i	Subscript of an element whose address is required to be calculated

- **If an element in one dimensional array occupy w word (byte):**

Address (A[i]) = M + (i – 1) * w [Note: Lower bound of array is assumed to be 1]

M	Base address of array
i	Subscript of an element whose address is required to be calculated
c	Size of an element

TO FIND ADDRESS OF AN ELEMENT IN ONE DIMENSIONAL ARRAY FOR ANY VALUE OF LOWER BOUND

If array is stored starting from memory location M and for each element it requires w number of words then the address for A[i] will be

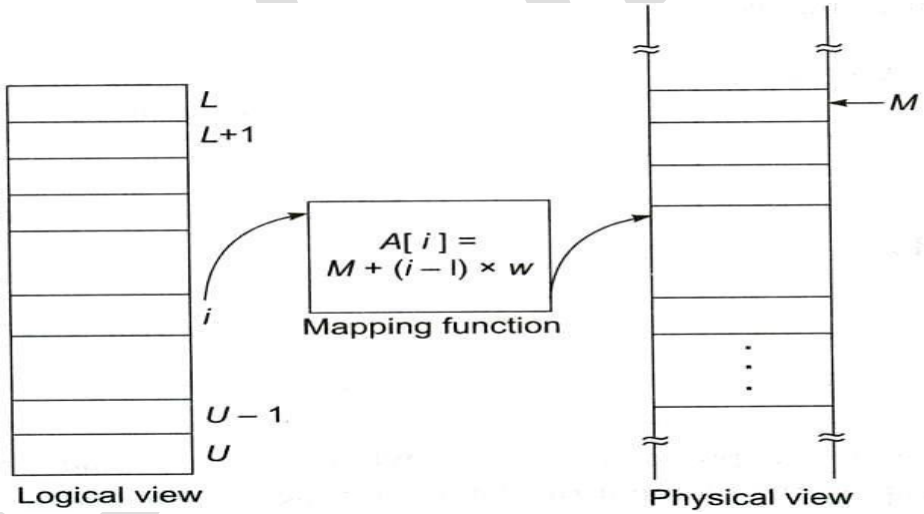
Address (A[i]) = M + (i – L) * w

Here **Lower bound** of array can be any arbitrary value denoted by **L**.

M	Base address of array
i	Subscript of an element whose address is required to be calculated
L	Lower bound of array
w	Size of an element

Above formula is known as **indexing formula**, which is used to map the logical presentation of an array to physical presentation.

By knowing the starting address of array M, the location of the i^{th} element can be calculated instead of moving towards i from M.



Example

Suppose that each element requires 2 word (byte), the base address of the array a[10] is 100 and the lower bound of the array is 1. Find out the address of following elements:

1. Find address of a[4].
2. Find address of a[7].

Answer

We are given that: Base address of the array is 100. So M=100
Lower bound of the array is 1. So L=1. Each element requires 2 word (byte). So w=2.

Formula: **Address (A[i]) = M + (i – L) * w**

1. To find address of a[4]
   Address (a[4]) = 100 + (4 – 1) * 2
   = 106

2. To find address of a[7]
   Address (a[7]) = 100 + (7 – 1) * 2
   = 112

***MULTIDIMENSIONAL / TWO DIMENSIONAL ARRAY**

Definition:

A **Two-Dimensional array** (commonly called a **matrix)** consists of elements of the same type arranged in rows and columns.

The rows and columns of a matrix are numbered starting from 0. Thus, for an array of size rows x cols, the row subscripts are in the range 0 to rows - 1 and the column subscripts are in the range 0 to cols - 1.

An m X n matrix where m denotes the number of rows and n denotes number of columns is as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \cdots & a_{mn} \end{bmatrix}$$ m x n

The subscripts of any arbitrary element say Aij represent i^{th} row and j^{th} column.

Declaration of two dimensional array

The declaration of a two-dimensional array takes the following form:

data_type var_name [row_exp] [col_exp] ;

where var_name is the name of the array being declared, each element of which is of type data_type.

The expressions rows and cols enclosed in square brackets are constant integral expressions (or integral constants) that specify the number of rows and columns in the array.

When a compiler comes across the declaration of an array, it reserves memory for the specified number of array elements. For a two-dimensional array, the memory required is given as rows* cols* sizeof(arr_type).

Initializing two dimensional array

A matrix can be initialized using a syntax similar to that used for the initialization of vectors. Thus, the declaration

int mat [3] [4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

C language allows the initialization list for each row to be enclosed within a pair of braces. Thus, the declaration of array mat given above can alternatively be written as follows:

int mat[3] [4] = {{l, 2, 3, 4), {5, 6, 7, 8), {9, 10, 11, 12}};

Accessing two dimensional array elements

An element of a matrix can be accessed by writing the array name followed by row and column subscript expressions within separate array subscript operators as shown below.

var_ name [row_expr] [col_expr]

where row_ expr and col_ expr are integral expressions that specify the row and column positions, respectively, of an element in the array.

As mentioned earlier, for an element access to be valid, the values of row_ expr should be in the range 0 to rows - 1 and the values of col_ expr should be in the range 0 to cols – 1.

Example

1. a[0][0]=10; //Assigns 10 to element at first row and first column
2. a[0][1]=3; //Assign 3 element at first row and second column

*MEMORY REPRESENTATION OF A MATRIX

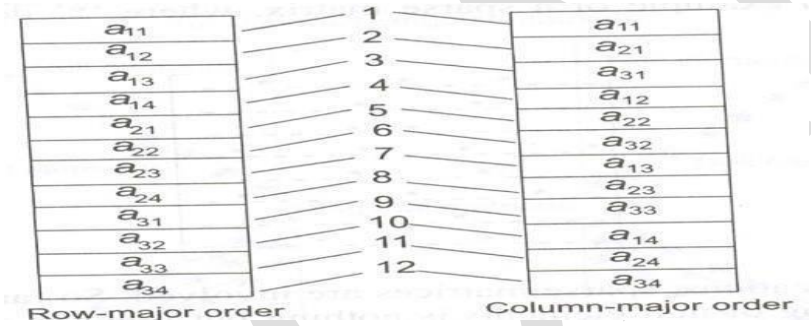
Like one dimensional array, matrices are also stored in continuous memory location. There are two conventions of storing any matrix in memory:

1. Row major order
2. Column major order

In row major order, elements of matrix are stored on a row by row basis. i.e. all the elements of the first row, then all the elements of second row and so on.

In column major order, elements of matrix are stored column by column. i.e. all the elements of the first column are stored in their order of rows, then in second column and so on.

E.g Consider the representation of a Matrix of 3 x 4



Reference of elements in a matrix

Logically a matrix appears as two dimensional but physically it is stored in a linear fashion. So in order to map from logical view to physical structure, we need indexing formula. Obviously, the indexing formula for different order will be different

ROW MAJOR ORDER – ADDRESS CALCULATION

Assume that the base address is 1 (the first location of the memory). So the address of Aij will be obtained as -

Storing all the elements in first (i -1)th rows + Number of elements in the ith row upto the jth column" i.e. Address (Aij) = (i – 1) * n + j

So for the matrix A3X4, the location of A32 will be calculated as 10.

TO FIND ADDRESS OF AN ELEMENT IN TWO DIMENSIONAL ARRAY WITH LOWER BOUND = 1 FOR ROW AND COLUMN

Note: This formula assumes that lower bound for i (row) and j (column) will be 1.

Address (Aij) = M + ((i – 1) * n + j – 1) * w where n is total number of columns

M	Base address of array
i	Row subscript of an element whose address is required to be calculated.
n	Number of columns
j	Column subscript of an element whose address is required to be calculated.
w	Size of an element

TO FIND ADDRESS OF AN ELEMENT IN TWO DIMENSIONAL ARRAY FOR ANY VALUE OF LOWER BOUND FOR ROW AND COLUMN

Address (Aij) = M + ((i – L1) * n + j – L2) * w where n is no. of columns

M	Base address of array
i	Row subscript of an element whose address is required to be calculated.
L1	Lower bound of i (row)
n	Number of columns
j	Column subscript of an element whose address is required to be calculated.
L2	Lower bound of j (column).
w	Size of an element

COLUMN MAJOR ORDER – ADDRESS CALCULATION

Assume that the base address is 1 (the first location of the memory). So the address of Aij will be obtained as-

Storing all the elements in first $(j - 1)^{th}$ columns + Number of elements in the j^{th} column upto the i^{th} row i.e. Address (Aij) = (j – 1) * m + i
So for the matrix A3X4, the location of A32 will be calculated as 6.

TO FIND ADDRESS OF AN ELEMENT IN TWO DIMENSIONAL ARRAY WITH LOWER BOUND = 1 FOR ROW AND COLUMN

Note: This formula assumes that lower bound for i (row) and j (column) will be 1.

Address (Aij) = M + ((j – 1) * m + i – 1) * w where m is total number of rows

M	Base address of array
j	Column subscript of an element whose address is required to be calculated.
m	Number of rows
i	Row subscript of an element whose address is required to be calculated.
w	Size of an element

TO FIND ADDRESS OF AN ELEMENT IN TWO DIMENSIONAL ARRAY FOR ANY VALUE OF LOWER BOUND FOR ROW AND COLUMN

Address (Aij) = M + ((j – L2) * m + i – L1) * w where m is no. of rows

M	Base address of array
j	Column subscript of an element whose address is required to be calculated.
L2	Lower bound of j (column).
m	Number of rows
i	Row subscript of an element whose address is required to be calculated.
L1	Lower bound of i (row)
w	Size of an element

Example

Assume that the base address of the two dimensional array A[3][3] is 100, each element requires 2 byte (word). Find the address of the element A[3][2] using

1. Row major order
2. Column major order

Solution

We are given that: Base address of the array is 100. So M=100 Each element requires 2 word (byte). So w=2.
Number of rows are 3. So m=3.
Number of columns are 3. So n=3.
Lower bound of i (row) L1 = 1
Lower bound of j (column) L2 = 1

1. Row major

To find address of the A[3][2] element of the array. So i = 3 and j = 2
Address (A[3][2]) = M + ((i – L1) * n + j – L2) * w
= 100 + ((3 – 1) * 3 + 2 – 1) * 2
= 114

2. Column major

To find address of the A[3][2] element of the array. So i = 3 and j = 2

Address (A[3][2]) = M + ((j – L2) * m + i – L1) * w
= 100 + ((2 – 1) * 3 + 3 – 1) * 2
= 110

*APPLICATION OF ARRAYS

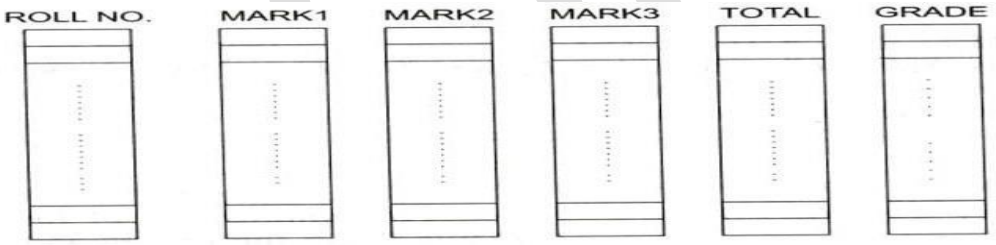
1. Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables. Many databases, small and large, consist of (or include) one-dimensional arrays whose elements are records.

2. Arrays are used to implement other data structures, such as heaps, hash tables, deques, queues, stacks, strings, and VLists.

3. There are wide applications of arrays in computation. That is why almost every programming language includes this data type as a built in data type.
4. Suppose you want to store records of all students in a class. The record structure is given by

STUDENT

Roll No.	Mark1	Mark2	Mark3	Total	Grade
Alpha Numeric	Numeric	Numeric	Numeric	Numeric	Character



If sequential storage of record is not objection, then we can store the records by maintaining 6 array whose size is specified by the total number of students in the class as in Figure.

5. Arrays can be used to represent polynomials so that mathematical operations can be performed in an efficient manner.

Note:

1. One Dimensional array declaration, initialization and accessing array element (integer, float and character array)
2. Two Dimensional array declaration, initialization and accessing array element (integer, float and character array)

Refer following material for detailed notes on above two topics.

****ONE-DIMENSIONAL INTEGER & FLOAT ARRAY**

An **array** is a group of related data items that share a common name. For example, we can define an array name salary to represent a set of salaries of a group of employees.

A list of items can be given one variable name using only one subscript and such a variable is called a **single subscripted variable** or **one-dimensional array.** Arrays can be any variable type.

The complete set of values is referred to as an array. Individual values / data item are called **elements**.

A particular element of an array is indicated by writing a number called **index number** or **subscript** in brackets after the array name. For e.g. salary[10] represents the salary of the 11^{th} employee because in C language subscript starts with 0.

The subscript of an array can be integer constants, integer variables like i, or expressions that produce (yield) integers.

***Declaration of One Dimensional integer and float Arrays**

Arrays must be declared before they are used. The general form of array declaration is

type variable-name[size];

The type specifies the type of element that will be contained in the array, such as int, float, or char.

And the size indicates the maximum number of elements that can be stored inside the array. The size should be either a numeric constant or a symbolic constant. E.g.

int a[30]; Declares **a** to be an array containing 30 integer elements. Any subscripts from 0 to 29 are valid.

float Declares **height** to be an array containing 50 float (real) elements. **height[50];** Any subscripts from 0 to 49 are valid.

***Initialization of One Dimensional integer and float Arrays**

After an array is declared, its elements must be initialized. Otherwise, they will contain “garbage”.

An array can be initialized at either of the following stages:

1) At compile time (at time of declaration of an array)
2) At run time (input form user and with assignment statement)

1) Compile time initialization (at time of declaration of an array)

We can initialize the elements of an array when they are declared. The general form of initialization of arrays is:

type array-name[size] = { list of values };

The values in the list are separated by commas.

int number[3] = { 1, 5, 7 }; This will declare number as an array of size 3 and will assign values 1, 5 and 7 to element 0, 1 and 2 respectively.

float height[3] = { 5.8, 6.2, 5 } This will declare height as an array of size 3 and will assign values 5.8, 6.2 and 5.0 to element 0, 1 and 2 respectively.

2) Run time initialization (input form user and with assignment statement)

An array can be explicitly initialized at run time.

Run time initialization can be done by

(1) Taking input from user or

(2) By assignment statement

(1) Taking input from user

We can also initialize array elements by taking input from user. This concept is known as run time initialization. E.g.

Input 1-D integer array

```
int marks[10]; for
(i=0; i<5;
i++)
    {  printf (“Enter mark %d :
        “,i+1); scanf (“%d”,
        &marks[i]);
    }
```

Input 1-D float array

```
float
height[10]; for
(i=0; i<5; i++)
   { printf (“Enter height %d :
        “,i+1); scanf (“%f”,
        &height[i]);
    }
```

(2) By assignment statement

Elements of one dimensional array can be initialized individually also. E.g. int number[5]; number[0] = 5;
number[1] = a *
5;
number[2]=number[0] + number[1]; etc.
This approach is time consuming and it will increase the length of program.

***Accessing One Dimensional Integer and Float Array Elements**

Individual elements of the array can be accessed using the following syntax:
array_name [index or subscript];

Example:

1. To access third element from array we write: **temp = marks[2];**

   The subscript for third element is 2 because the lower bound of array in C is 0.

2. To print a particular element from array, we write

printf("%d",mark[3]); **ONE-DIMENSIONAL CHARACTER ARRAY**

(STRING)

A **string** is a sequence of characters which is treated as a single data item. Any group of characters
(except double quote sign) defined between double quotations marks is a constant string.
Example: "Man is obviously made to think."

***Declaration of One Dimensional character array (string)**

C does not support string as a data type. The C language treats character strings simply as arrays of characters. So, a string variable in C is any valid C variable name and it is always declared as an array of characters.

The general form of declaration of a string variable is:

char string_name [size] ;

The size represents the maximum number of characters in the string_name.

char name[10]; Declares the **name** as a character array (string) variable that can hold a maximum of 10 characters (including null character).

When compiler assigns a character string to a character array, it automatically appends a null character ('\0') at the end of the string. So, the size should be equal to maximum number of characters in the string plus one.

***Initialization of One Dimensional character Array (string)** An

array can be initialized at either of the following stages:

1) At compile time (at time of declaration of an array)
2) At run time (input form user and with assignment statement)

1) Compile time initialization of One Dimensional character array char

name[6] = { 'H', 'E', 'L', 'L', 'O', '\0' }; **OR** char name[6] = "HELLO";

Above statements declares name to be an array of 6 characters and initialize it with string "HELLO" ending with null character. Name string has to be 6 characters long because it needs 5 characters to store string HELLO and one character to store null terminator.

2) Run time initialization (input form user and copy with strcpy function)

An array can be explicitly initialized at run time. This approach is usually applied for initializing string as per user input.

Run time initialization can be done by

(1) Taking input from user or
(2) Character-by-character copy / with strcpy function

(1) Taking input from user

Using scanf()

```
char name[10];
printf ("Enter name :
"); scanf ("%s",
name);
```

Using gets()

```
char name[10];
printf ("Enter name :
"); gets(name);
```

(2) copy with strcpy function

C does not allow assigning one string directly to another string with assignment statement. We may assign one string to another sting with the help of strcpy function or we can copy character by character.

```
char name1[10] = "HELLO", name2[10] ; strcpy
(name2, name1);

Note: Here name1's content will be copied in name2.
```

***Accessing One Dimensional Character Array Elements**

One dimensional charcter array can be accessed using **array_name.**:

Example:

1. To access an array we write: **strcpy(destination_string, source_string);**

   source_string will be copied into destination_string.

2. To print a character array, we write **printf("%s",string1);**

TWO-DIMENSIONAL INTEGER

& FLOAT ARRAYS

Array can be used to store a table of values.

*Declaration of Two Dimensional integer and float Array:

Like any other variable, arrays must be declared before they are used. The general form of array declaration is **type array-name**

[row_size][column_size];

The type specifies the type of element that will be contained in the array, such as int, float, or char. The row_size indicates the maximum number of rows and column_size indicates the maximum number of columns that can be stored inside the array.

The size should be either a numeric constant or a symbolic constant. E.g.

int exam[4][3]; Declares **exam** to be an array containing 12 integer elements. Any subscripts from 0 to 3 are valid for row. Any subscripts from 0 to 2 are valid for columns.

float Declares **temp** to be an array containing 9 float (real) elements. Any **temp[3][3];** subscripts from 0 to 2 are valid for row. Any subscripts from 0 to 2 are valid for columns.

*Initializing two-dimensional arrays

After an array is declared, its elements must be initialized. Otherwise, they will contain

"garbage". An array can be initialized at either of the following stages: **1)**
At compile time (at time of declaration of an array)
2) At run time (input from user and with assignment statement)

1) Compile time initialization (at time of declaration of an array)

Like the one-dimensional arrays, two-dimensional arrays may be initialized by following their declaration with a list of initial values enclosed in braces. The general form of initialization of arrays is:

type array-name[size] = { list of values };

The values in the list are separated by commas.

int table[2][3] = {0,0,0,1,1,1}; OR float height[2][2] = { 5.8, 6.2, 5, 5.3
} int table[2][3] = { {0,0,0}, {1,1,1}
};

2) Run time initialization (input from user and with assignment statement)

An array can be explicitly initialized at run time.

Run time initialization can be done by

(1) Taking input from user or
(2) By assignment statement

(1) Taking input from user

We can also initialize array elements by taking input from user. This concept is known as run time initialization. This approach is usually applied for initializing large arrays.

Looping can be used to input array elements. As two dimensional array deals with two subscripts, we need to use two for loops to control row and column subscript respectively.
For E.g.

Input 2-D integer array	Input 2-D float array
int exam[4][3]; for (i=0; i<4; i++) { for (j=0; j<3; i++) { printf (“Enter mark [%d][%d] : “,i+1,j+1); scanf (“%d”, &exam[i][j]); } }	float temp[3][3]; for (i=0; i<4; i++) { for (j=0; j<3; i++) { printf (“Enter temperature [%d][%d] : “,i+1,j+1); scanf (“%f”, &temp[i]); } }

(2) By assignment statement

Elements of two dimensional array can be initialized individually also. E.g.

int exam[4][3]; exam[0][2] = 15; exam[1][1]
= a * 5; exam[2][2] = exam[2][0] +
exam[2][1]; etc.
This approach is time consuming and it will increase the length of program.

***Accessing Two Dimensional Integer and Float Array Elements**

Individual elements of the array can be accessed using the following syntax:
array_name [row_ subscript] [column_subscript];

Example:

1. To access third element from array we write: **temp = exam[1][2];**

2. To print a particular element from array, we write **printf("%d",exam[2][3]);**

TWO-DIMENSIONAL CHARACTER ARRAY (TABLE OF STRINGS)

We often use lists of character strings, such as a list of names of students in a class, list of names of employees in an organization, list of places, etc.

***Declaration of Two Dimensional character array**

The general form of declaration of a 2-D character array is:

char string_name [row_size][column_size] ;

The row_size represents the maximum number of rows (total number of strings) and column_size represents number of characters in each row.

char Declares the **city** as a 2-D character array variable that can **city[5][15];**
hold a maximum of 5 city name each with15 characters (including null character).

***Initialization of One Dimensional character Array (string)**

An array can be initialized at either of the following stages:

1) At compile time (at time of declaration of an array)
2) At run time (input from user)

1) Compile time initialization of One Dimensional character array

char city[5][15] = { { 'C', 'h', 'a', 'n', 'd', 'i', 'g', 'a', 'r', 'h', '\0' } , { 'M', 'a', 'd', 'r', 'a', 's', '\0' } };

OR char city[5][15] = { "Chandigarh", "Madras" };

2) Run time initialization (input form user) char

city[5][10]; int

```
i;
for (i=0; i<5; i++)
    {    printf (“Enter city name :
         “); gets(city[i]);
    }
```

***Accessing Two Dimensional Character Array Elements**

Two dimensional charcter array can be accessed using **array_name [row_subscript]**

Example:
To print a two dimensional character array, we write a loop: char
city[5][10]; int
i;

```
for (i=0; i<5; i++)
    {    puts(city[i]);
    }
```