Data Structures using C

Unit I (INTRODUCTION)

DATA STRUCTURES:

In computer science, a data structure is a particular way of storing and organizing data in a computer's memory so that it can be used efficiently. **Data may be organized in many different ways the logical or mathematical model of a particular organization of data is called a data structure**. The choice of a particular data model depends on the two considerations first; it must be rich enough in structure to mirror the actual relationships of the data in the real world. On the other hand, the structure should be simple enough that one can effectively process the data whenever necessary.

Need of data structure:

- It gives different level of organization data.
- It tells how data can be stored and accessed in its elementary level.
- Provide operation on group of data, such as adding an item, looking up highest priority item.
- Provide a means to manage huge amount of data efficiently.
- Provide fast searching and sorting of data.

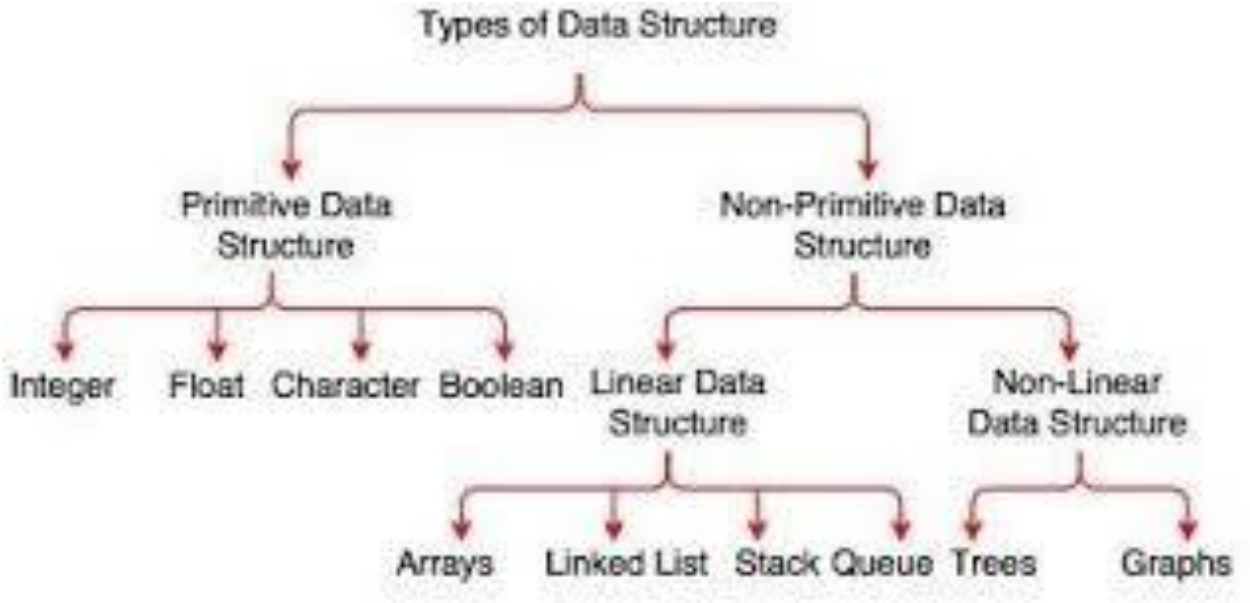
CLASSIFICATION OF DATA STRUCTURES:

The data structures are classified into two categories:

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

Basic data types such as integer, real, character and Boolean are known as primitive data structures.

The simplest example of non-primitive data structure is the processing of complex numbers. Very few computers are capable of doing arithmetic on complex numbers. Linked-list, Stacks, Queues, Trees and Graphs are the examples of non-primitive data structures.



Linear Data Structure:

A data structure is said to be linear if its elements form any sequence. There are basically two ways of representing such linear structure in memory.

a) One way is to have the linear relationships between the elements represented by means of sequential memory location. These linear structures are called arrays.
b) The other way is to have the linear relationship between the elements represented by means of pointers or links. These linear structures are called linked lists.

The common examples of linear data structure are arrays, queues, stacks and linked lists.

Non-linear Data Structure:

This structure is mainly used to represent data containing a hierarchical relationship between elements.

E.g. Graphs, Trees

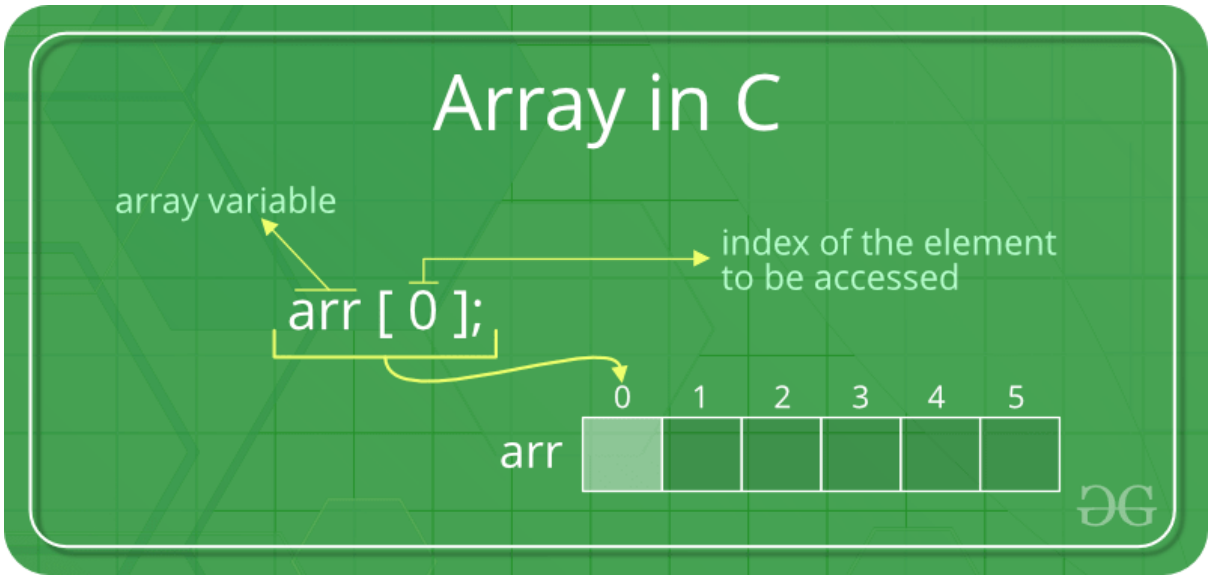
DATA STRUCTURES OPERATIONS:

The data appearing in our data structures are processed by means of certain operations. In fact, the particular data structure that one chooses for a given situation depends largely in the frequency with which specific operations are performed. The following four operations play a major role in this text:

- **Traversing:** accessing each record/node exactly once so that certain items in the record may be processed. (This accessing and processing is sometimes called “visiting” the record.)
- **Searching:** Finding the location of the desired node with a given key value, or finding the locations of all such nodes which satisfy one or more conditions.
- **Inserting:** Adding a new node/record to the structure.
- **Deleting:** Removing a node/record from the structure.

Array

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.



Declaration of an array:

We know that all the variables are declared before they are used in the program. Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified. The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

Syntax:

data_type array_name[n];

where, n is the number of data items (or) index(or) dimension. 0 to (n-1) is the range of array.

Ex: int a[5]; float x[10];

Initialization of Arrays:

The different types of initializing arrays:

1. At Compile time

(i) Initializing all specified memory locations.

(ii) Partial array initialization

(iii) Initialization without size.

(iv) String initialization.

2. At Run Time

1.Compile Time Initialization:

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is

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

(i). Initializing all specified memory locations:

Arrays can be initialized at the time of declaration when their initial values are known in advance. Array elements can be initialized with data items of type int, char etc.

Ex:- int a[5]={10,15,1,3,20};

During compilation, 5 contiguous memory locations are reserved by the compiler for the variable a and all these locations are initialized as shown in figure.

A[0]	A[1]	A[2]	A[3]	A[4]
10	15	1	3	20
1001	1002	1003	1004	1005

Fig: Initialization of int Arrays

Ex:- int a[3]={9,2,4,5,6}; //error: no. of initial vales are more than the size of array

(ii). Partial array initialization:

Partial array initialization is possible in c language. If the number of values to be initialized is less than the size of the array, then the elements will be initialized to zero automatically.

Ex:- int a[5]={10,15};

Eventhough compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

A[0]	A[1]	A[2]	A[3]	A[4]
10	15	0	0	0
1001	1002	1003	1004	1005

Fig: Partial Array Initialization

Initialization with all zeros:-

Ex:- int a[5]={0};

A[0]	A[1]	A[2]	A[3]	A[4]
0	0	0	0	0
1001	1002	1003	1004	1005

(iii). Initialization without size:

Consider the declaration along with the initialization.

Ex:- char b[]={'C','O','M','P','U','T','E','R'};

In this declaration, even though we have not specified exact number of elements to be used in array b, the array size will be set of the total number of initial values specified. So, the array size will be set to 8 automatically. The array b is initialized as shown in figure.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
C	O	M	P	U	T	E	R
1001	1002	1003	1004	1005	1006	1007	1008

Fig: Initialization without size

(iv). Array initialization with a string: -Consider the declaration with string initialization.

Ex:- char b[]="COMPUTER";

The array b is initialized as shown in figure

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
C	O	M	P	U	T	E	R	\0
1001	1002	1003	1004	1005	1006	1007	1008	1009

Fig: Array Initialized with a String

Even though the string "COMPUTER" contains 8 characters, because it is a string, it always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER"; // correct

char b[8]="COMPUTER"; // wrong

2. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:- scanf can be used to initialize an array.

int x[3];

scanf("%d%d%d",&x[0],&x[1],&x[2]);

The above statements will initialize array elements with the values entered through the key board.

TYPES OF ARRAY

The array may be categorized into

- One dimensional array
- Two dimensional array
- Multidimensional array

TWO-DIMENSIONAL ARRAY

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array

is divided into rows and columns. These are well suited to handle a table of data. In 2-D array we can declare an array as,

Declaration:

Syntax: data_type array_name[row_size][column_size];

Ex: int arr[3][3];

where first index value shows the number of the rows and second index value shows the number of the columns in the array.

Initializing two-dimensional arrays:

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

Ex: int a[2][3]={0,0,0,1,1,1};

Initializes the elements of the first row to zero and the second row to one. The initialization is done row by row.

The above statement can also be written as,

int a[2][3] = {{ 0,0,0},{1,1,1}};

by surrounding the elements of each row by braces. We can also initialize a two-dimensional array in the form of a matrix as shown below,

int a[2][3]={ {0,0,0}, {1,1,1} };

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

Ex:

int a[][3]={ {0,2,3}, {2,1,2} };

If the values are missing in an initializer, they are automatically set to zero.

Ex:

int a[2][3]={ {1,1}, {2} };

Will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

MULTIDIMENSIONAL ARRAYS

Multidimensional arrays are often known as array of the arrays. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional arrays we will be discussing mainly about two dimensional arrays and a bit about three dimensional arrays.

Syntax: data_type array_name[size1][size2][size3]------[sizeN];

In 2-D array we can declare an array as :

```
int arr[3][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

where first index value shows the number of the rows and second index value shows the number of the columns in the array. To access the various elements in 2-D array we can use:

```
printf("%d", a[2][3]);
```

In 3-D we can declare the array in the following manner :

```
int arr[3][3][3] = { 1, 2, 3,
                     4, 5, 6,
                     7, 8, 9,
                10, 11, 12,
                13, 14, 15,
                16, 17, 18,
                     19, 20, 21,
                     22, 23, 24,
                     25, 26, 27 };
```

/* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns */

If we want to access the element the in 3-D array we can do it as follows :

```
printf("%d",a[2][2][2]);
```

/* its output will be 26, as a[2][2][2] means first value in [] corresponds to the grid no. i.e. 3 and the second value in [] means third row in the corresponding grid and last [] means third column */

Ex: int arr[3][5][12];

float table[5][4][5][3];

STRUCTURES

A structure can be considered as a template used for defining a collection of variables under a single name. Structures help programmers to group elements of different data types into a single logical unit (Unlike arrays which permit a programmer to group only elements of same data type).

Suppose we want to store a date inside a C program. Then, we can define a structure called date with three elements day, month and year. The syntax of this structure is as follows:

```
struct date
        {
                int day;
                 int month;
                int year;
        };
```

A structure type is usually defined at the beginning of a program. This usually occurs just after the main() statement in a file. Then a variable of this structure type is declared and used in the program.

Eg: struct date order_date;

C Structure Initialization

1. When we declare a structure, memory is not allocated for un-initialized variable.

2. Let us discuss very familiar example of structure student, we can initialize structure variable in different ways.

Way 1 : Declare and Initialize

```
struct student
{
        char name[20];
        int roll;
        float marks;
}
std1 = { "Pritesh",67,78.3};
```

In the above code snippet, we have seen that structure is declared and as soon as after declaration we have initialized the structure variable.

std1 = { "Pritesh",67,78.3 }

Way 2 : Declaring and Initializing Multiple Variables

```
struct student
{
        char name[20];
        int roll;
        float marks;
}
std1 = {"Pritesh",67,78.3};
std2 = {"Don",62,71.3};
```

In this example, we have declared two structure variables in above code. After declaration of variable we have initialized two variable.

std1 = {"Pritesh",67,78.3}; std2 = {"Don",62,71.3};

Way 3 : Initializing Single member

```
struct student
{
int mark1;
int mark2;
int mark3;
}
sub1={67};
```

Though there are three members of structure, only one is initialized, then remaining two members are initialized with Zero. If there are variables of other data type then their initial values will be,

Data Type	Default value if not initialized
Integer	0
Float	0.00
Char	NULL

Way 4 : Initializing inside main

```
struct student
 {
int mark1;
```

```
int mark2;
int mark3;
};
void main() { struct student s1 = {89,54,65}; - - - - -- - - - - -- - - - - -- };
```

UNION

Unions are quite similar to the structures in C. Union is also a derived type as structure. Union can be defined in same manner as structures just the keyword used in defining union in union where keyword used in defining structure was struct.

```
union car
{
        char name[50];
        int price;
};
```

Union variables can be created in similar manner as structure variable.

```
union car
{
        char name[50];
        int price;
}
c1, c2, *c3;
```

OR

```
union car
{
        char name[50];
        int price;
};
-------Inside Function----------
union car c1, c2, *c3;
```

In both cases, union variables c1, c2 and union pointer variable c3 of type union car is created.

Accessing members of an union:

The member of unions can be accessed in similar manner as that structure. Suppose, we you want to access price for union variable c1 in above example, it can be accessed as c1.price. If you want to access price for union pointer variable c3, it can be accessed as (*c3). price or as c3->price.

Difference between union and structure:

Though unions are similar to structure in so many ways, the difference between them is crucial to understand. This can be demonstrated by this example,

```
#include <stdio.h>
union job
{ //defining a union
        char name[32];
        float salary;
        int worker_no;
}u;
struct job1
{
        char name[32];
        float salary;
        int worker_no;
        }s;
int main()
{
        printf("size of union = %d",sizeof(u));
        printf("\nsize of structure = %d", sizeof(s)); return 0;
}
```

Output

size of union = 32

size of structure = 40

Structure	Union
Struct keyword is used to define a structure.	Union keyword is used to define a union.
Members do not share memory in a structure.	Members share the memory space in a union.
Any member can be retrieved at any time in a structure.	Only one member can be accessed at a time in a union.
Several members of a structure can be initialized at once.	Only the first member can be initialized.
Size of the structure is equal to the sum of size of the each member.	Size of the union is equal to the size of the largest member.
Altering value of one member will not affect the value of another.	Change in value of one member will affect other member values.
Stores different values for all the members.	Stores same value for all the members.