What is Data Structure?

The data structure name indicates itself that organizing the data in memory. There are many ways of organizing the data in the memory as we have already seen one of the data structures, i.e., array in C language. Array is a collection of memory elements in which data is stored sequentially, i.e., one after another. In other words, we can say that array stores the elements in a continuous manner.

### Types of Data Structures

There are two types of data structures:

* Primitive data structure
* Non-primitive data structure

**Primitive Data structure**

The primitive data structures are primitive data types. The int, char, float, double, and pointer are the primitive data structures that can hold a single value.

**Non-Primitive Data structure**

The non-primitive data structure is divided into two types:

* Linear data structure
* Non-linear data structure

**Linear Data Structure**

The arrangement of data in a sequential manner is known as a linear data structure. The data structures used for this purpose are Arrays, Linked list, Stacks, and Queues. In these data structures, one element is connected to only one another element in a linear form.

**When one element is connected to the 'n' number of elements known as a non-linear data structure. The best example is trees and graphs. In this case, the elements are arranged in a random manner.**

We will discuss the above data structures in brief in the coming topics. Now, we will see the common operations that we can perform on these data structures.

**Data structures can also be classified as:**

* **Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.
* **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.

### Major Operations

The major or the common operations that can be performed on the data structures are:

* **Searching:** We can search for any element in a data structure.
* **Sorting:** We can sort the elements of a data structure either in an ascending or descending order.
* **Insertion:** We can also insert the new element in a data structure.
* **Updation:** We can also update the element, i.e., we can replace the element with another element.
* **Deletion:** We can also perform the delete operation to remove the element from the data structure.

An ADT tells **what** is to be done and data structure tells **how** it is to be done. In other words, we can say that ADT gives us the blueprint while data structure provides the implementation part.

As the different data structures can be implemented in a particular ADT, but the different implementations are compared for time and space.

Advantages of Data structures

**The following are the advantages of a data structure:**

* **Efficiency:** If the choice of a data structure for implementing a particular ADT is proper, it makes the program very efficient in terms of time and space.
* **Reusability:** The data structure provides reusability means that multiple client programs can use the data structure.
* **Abstraction:** The data structure specified by an ADT also provides the level of abstraction.

**Data structures can also be classified as:**

* **Static data structure:** It is a type of data structure where the size is allocated at the compile time. Therefore, the maximum size is fixed.
* **Dynamic data structure:** It is a type of data structure where the size is allocated at the run time. Therefore, the maximum size is flexible.

### Data Structure Classification



# **Pointer**

Pointer is used to points the address of the value stored anywhere in the computer memory. To obtain the value stored at the location is known as dereferencing the pointer. Pointer improves the performance for repetitive process such as:

* Traversing String
* Lookup Tables
* Control Tables
* Tree Structures

## **Pointer Details**

* **Pointer arithmetic:** There are four arithmetic operators that can be used in pointers: ++, --, +, -
* **Array of pointers:** You can define arrays to hold a number of pointers.
* **Pointer to pointer:** C allows you to have pointer on a pointer and so on.
* **Passing pointers to functions in C:** Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
* **Return pointer from functions in C:** C allows a function to return a pointer to the local variable, static variable and dynamically allocated memory as well.



### Program

#### **Pointer**

1. #include <stdio.h>
3. **int** main( )
4. {
5. **int** a = 5;
6. **int** \*b;
7. b = &a;
9. printf ("value of a = %d\n", a);
10. printf ("value of a = %d\n", \*(&a));
11. printf ("value of a = %d\n", \*b);
12. printf ("address of a = %u\n", &a);
13. printf ("address of a = %d\n", b);
14. printf ("address of b = %u\n", &b);
15. printf ("value of b = address of a = %u", b);
16. **return** 0;
17. }

#### **Output**

1. value of a = 5
2. value of a = 5
3. address of a = 3010494292
4. address of a = -1284473004
5. address of b = 3010494296
6. value of b = address of a = 3010494292

### Program

#### **Pointer to Pointer**

1. #include <stdio.h>
3. **int** main( )
4. {
5. **int** a = 5;
6. **int** \*b;
7. **int** \*\*c;
8. b = &a;
9. c = &b;
10. printf ("value of a = %d\n", a);
11. printf ("value of a = %d\n", \*(&a));
12. printf ("value of a = %d\n", \*b);
13. printf ("value of a = %d\n", \*\*c);
14. printf ("value of b = address of a = %u\n", b);
15. printf ("value of c = address of b = %u\n", c);
16. printf ("address of a = %u\n", &a);
17. printf ("address of a = %u\n", b);
18. printf ("address of a = %u\n", \*c);
19. printf ("address of b = %u\n", &b);
20. printf ("address of b = %u\n", c);
21. printf ("address of c = %u\n", &c);
22. **return** 0;
23. }

#### **Pointer to Pointer**

1. value of a = 5
2. value of a = 5
3. value of a = 5
4. value of a = 5
5. value of b = address of a = 2831685116
6. value of c = address of b = 2831685120
7. address of a = 2831685116
8. address of a = 2831685116
9. address of a = 2831685116
10. address of b = 2831685120
11. address of b = 2831685120
12. address of c = 2831685128

# **Structure**

A structure is a composite data type that defines a grouped list of variables that are to be placed under one name in a block of memory. It allows different variables to be accessed by using a single pointer to the structure.

**Syntax**

1. struct structure\_name
2. {
3. data\_type member1;
4. data\_type member2;
5. .
6. .
7. data\_type memeber;
8. };

### Advantages

* It can hold variables of different data types.
* We can create objects containing different types of attributes.
* It allows us to re-use the data layout across programs.
* It is used to implement other data structures like linked lists, stacks, queues, trees, graphs etc.

**Program**

1. #include<stdio.h>
2. #include<conio.h>
3. **void** main( )
4. {
5. struct employee
6. {
7. **int** id ;
8. **float** salary ;
9. **int** mobile ;
10. } ;
11. struct employee e1,e2,e3 ;
12. clrscr();
13. printf ("\nEnter ids, salary & mobile no. of 3 employee\n"
14. scanf ("%d %f %d", &e1.id, &e1.salary, &e1.mobile);
15. scanf ("%d%f %d", &e2.id, &e2.salary, &e2.mobile);
16. scanf ("%d %f %d", &e3.id, &e3.salary, &e3.mobile);
17. printf ("\n Entered Result ");
18. printf ("\n%d %f %d", e1.id, e1.salary, e1.mobile);
19. printf ("\n%d%f %d", e2.id, e2.salary, e2.mobile);
20. printf ("\n%d %f %d", e3.id, e3.salary, e3.mobile);
21. getch();
22. }

## **C Structures (structs)**

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.

Unlike an [array](https://www.w3schools.com/c/c_arrays.php), a structure can contain many different data types (int, float, char, etc.).

## **Create a Structure**

## **Create a Structure**

You can create a structure by using the struct keyword and declare each of its members inside curly braces:

struct MyStructure {   // Structure declaration  
  int myNum;           // Member (int variable)  
  char myLetter;       // Member (char variable)  
}; // End the structure with a semicolon

To access the structure, you must create a variable of it.

Use the struct keyword inside the main() method, followed by the name of the structure and then the name of the structure variable:

Create a struct variable with the name "s1":

struct myStructure {  
  int myNum;  
  char myLetter;  
};  
  
int main() {  
**struct myStructure s1;**  
  return 0;  
}

//////////////////////////////////////

1. #include<stdio.h>
2. #include<conio.h>
3. **void** main( )
4. {
5. struct employee
6. {
7. **int** id ;
8. **float** salary ;
9. **int** mobile ;
10. } ;
11. struct employee e1,e2,e3 ;
12. clrscr();
13. printf ("\nEnter ids, salary & mobile no. of 3 employee\n"
14. scanf ("%d %f %d", &e1.id, &e1.salary, &e1.mobile);
15. scanf ("%d%f %d", &e2.id, &e2.salary, &e2.mobile);
16. scanf ("%d %f %d", &e3.id, &e3.salary, &e3.mobile);
17. printf ("\n Entered Result ");
18. printf ("\n%d %f %d", e1.id, e1.salary, e1.mobile);
19. printf ("\n%d%f %d", e2.id, e2.salary, e2.mobile);
20. printf ("\n%d %f %d", e3.id, e3.salary, e3.mobile);
21. getch();
22. }

///////////////////////

# **Array in Data Structure**

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is one of the simplest data structures where each data element can be randomly accessed by using its index number.

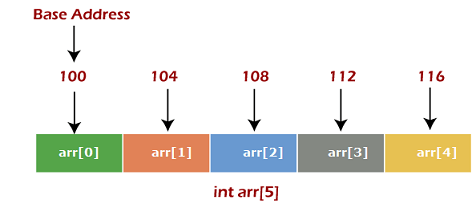
Properties of array

There are some of the properties of an array that are listed as follows -

* Each element in an array is of the same data type and carries the same size that is 4 bytes.
* Elements in the array are stored at contiguous memory locations from which the first element is stored at the smallest memory location.
* Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Representation of an array

We can represent an array in various ways in different programming languages. As an illustration, let's see the declaration of array in C language -



As per the above illustration, there are some of the following important points -

* Index starts with 0.
* The array's length is 10, which means we can store 10 elements.
* Each element in the array can be accessed via its index.

Why are arrays required?

Arrays are useful because -

* Sorting and searching a value in an array is easier.
* Arrays are best to process multiple values quickly and easily.
* **Arrays are good for storing multiple values in a single variable -**