Why do we organize our pantry?

Why do we arrange the washed clothes inside the cupboard?

Why do we stack the books in the shelf?

Why do we wait in a queue to buy something and did not crowd?

All the above asked questions can be answered with only these words – "**FOR EASY RETRIEVAL".**

At times when you are in a hurry and want to obtain something, things can be retrieved easily only when they are arranged properly. Arranging things not only saves your time but also increases your searching efficiency. The same can be applied to computers wherein it will be difficult to fetch the required data if the data is scattered. This is where data structures come into play. Data Structures helps in easy access and retrieval of data by organizing the data in a particular way.

WHAT IS DATA STRUCTURE?

Data structure is defined as the special way of storing and organizing the data in the computer memory so that it can be used efficiently.

Need for data structures

As the amount of data increases day by day, need for data structure increases to solve several computer – related problems

Speed

The speed with which data is processed plays a vital role in fastest retrieval and modification of records.

Handling multiple requests

Multiple requests can be processed effectively with the help of data structures.

Consider two different persons are performing two different operations simultaneously at the same time such as fetching a record and updating the data.

Data search

Searching for a particular item from a repository of huge number of items can be done effectively with data structures.

Consider an inventory size of about 1000 items and if you want to search for a particular item, you need to traverse through all the 1000 items which slows down the searching process.

With data structures, multiple users can handle and modify the data simultaneously.

Advantages

The following are some of the advantages of using data structures

Increased efficiency

Why do we need to organize the data? It is to increase efficiency. Efficiency of a system also depends on the choice of data structure.

For example, if we want to search for an element, with array data structure it takes more time as the elements are stored sequentially whereas it takes less time in case of other data structures such as binary search trees or hash tables.

Reusability

The data structures can be reused. Once the data structure has been implemented, it can be reused later.

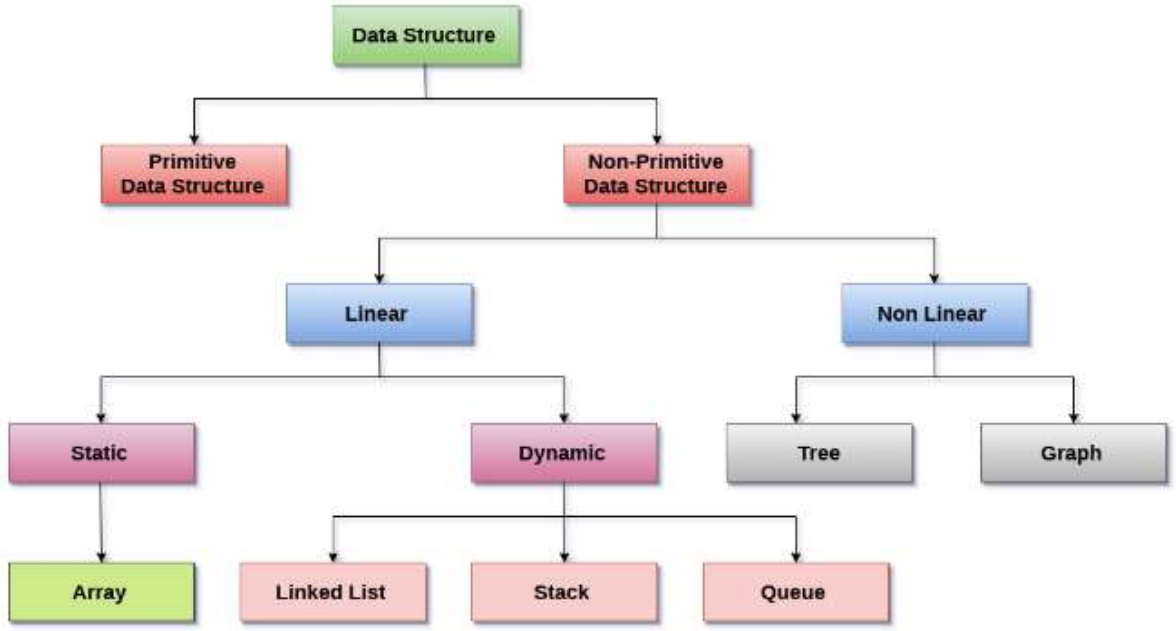
Data abstraction

The implementation details of the data structure are hidden from the user. The user can enjoy what the data structure provides without knowing the implementation details.

Data structure classification& types

Data structure can be classified as Primitive and Non – Primitive data structures. Non-Primitive data structures are further subdivided into two broad categories namely

- Linear data structure
- Non – Linear data structure



Linear data structure

When all the elements are arranged in a sequential order, the data structure is called as linear data structure. Since all the elements are stored in a non – hierarchical way, each element has

a successor and predecessor to it. The linear data structure can be represented in two ways inside a memory

- *Arrays* – representing the elements using linear memory location
- *Linked list* – representing the elements using pointers or links

The various types of linear data structures are

- Array
- Linked list
- Stack
- Queue

Array

Array is a collection of similar data items where each item represents the element of an array. Each element in the array has a different index number called as the subscript.

Linked list

Linked list is a linear data structure that consists of collection of nodes that are stored at non – contiguous memory location. Each node consists of data and a pointer field. The pointer consists of items that points to the next node.

Stack

Stack is a linear data structure which is similar to stack of books. Here, both insertion and deletion can be done only at the top.

Queue

Queue is a linear data structure which is similar to the queue in a ticket counter. Insertion is done at the rear end whereas deletion is done at the front end.

Non-Linear data structure

Here, the elements are not stored in a sequential order. The elements in non-linear data structure are connected with each other in a non-linear arrangement. There are two types of non-linear data structure namely

- Trees
- Graphs

ARRAY

An array is a linear data structure where elements of similar data type are stored in a linear fashion i.e. in a contiguous memory location.

Array can be viewed as a staircase where objects are placed in each step. Each step corresponds to the index and each object corresponds to the element. You can identify the objects by the steps on which they were on. Similarly the data elements in an array can be located based on its index (position) value.

Consider if the staircase has only 10 steps. Is it possible to place an 11th object? Absolutely NO. Similarly, an array is of fixed size and only fixed number of elements can be stored in the given array. Also it is not possible to shrink or expand array size as similar to that of staircase.

Array can be of two types

- One-dimensional array
- Multi-dimensional array

One-dimensional array

One-dimensional array can be viewed as a row of elements where each element is stored one after the other.

Multi-dimensional array

Multi-dimensional array is further classified as

- Two-dimensional array
- Three-dimensional array

Various operations that can be performed in an array

Random access to the elements is possible in an array. This makes it possible to access the elements in an array based on its position. Thus, accessing an element, insertion and searching can be done faster in an array.

- *Max()* - finds the maximum element in the array

```
array[0]=Max;
for(i=1;i<=size-1;i++)
{
If(array[i]>Max)
{
Max=array[i];
}
}
return max;
```

- *Min()* – finds the minimum element in the array

```
array[0]=Min;
for(i=1;i<=size-1;i++)
{
If(array[i]<Min)
{
Min=array[i];
}}
return Min;
```

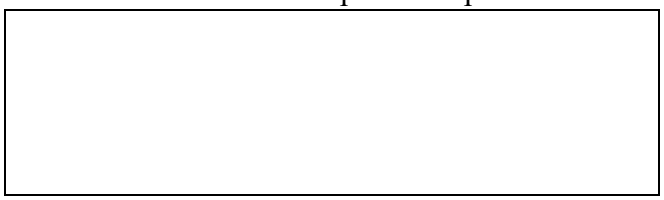
- *Search(x)* – search for the given element and returns the index in which it is found

```
for(i = 0; i < sizeofarray; i++)
{
 if(array[i] == x)
    {
      return i;
    }
 }
```

- *Insert(x,pos)* – inserts the element x at the given index

```
for(i=size-1;i>=pos;i--)
{
//shift elements towards the front
array[i]=array[i-1];
}
//inserts the element x at position pos
array[pos-1]=x;
```

- *Delete(pos)* – deletes the element in the position specified



- *Print() / traverse()* – traverse through the array and prints all the elements of the array

```
for(i=0;i<size_of_array;i++)
{
printf("%d",array[i]);
}
```

Advantage

Disadvantages

QUIZ
How do we change the size of an array?
ANSWER
Dynamic arrays