What is a Data Structure?

- “Data Structure is a way to store and organize data so that it can be used efficiently.”
- To store/organize a given data in the memory of computer so that each subsequent operations (query/update) can be performed efficiently.
- The term data structure is used to describe the way data is stored, and the term algorithm is used to describe the way data is processed.
- Choosing a data structure affects the kind of algorithm you might use, and choosing an algorithm affects the data structures we use.
- To develop a program of an algorithm we should select an appropriate data structure for that algorithm. Therefore, data structure is also represented as:

Algorithm + Data structure = Program

Characteristics of Data Structure

- It contains data items that can be elementary item, group item or another data structure.
- It has a set of operations that can be performed on data items. Such as searching, insertion etc.
- It describes the rules of how the data items are related to each other.
- Correctness − Data structure implementation should implement its interface correctly.
- Time Complexity − Running time or the execution time of operations of data structure must be as small as possible.
- Space Complexity − Memory usage of a data structure operation should be as little as possible.

Need of Data structures

As applications are getting complex and amount of data is increasing day by day, there may arise the following problems:

1. Processor speed: To handle very large amount of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

2. Data Search: Consider an inventory size of 60005 items in a store. If our application needs to search for a particular item, it needs to

traverse 60005 items every time, results in slowing down the search process.

3. Multiple requests: If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process.

4. Searching a number from millions of numbers is not possible in normal way but it can be done effectively through data structures by finding the best algorithm.

5. Data is organized to form a data structure in such a way that all items are not required to be searched and required data can be searched instantly.

Advantages of Data Structures

1. Efficiency: Efficiency of a program depends upon the choice of data structures. If our choice is proper then the program becomes effective in the terms of time and space. (Minimum time and space)

For example, we have some data and we need to perform the search for a particular record more frequently. In that case, if we organize our data in an array, we will have to search sequentially element by element. Hence, using array may not be very efficient here. There are better data structures which can make the search process efficient like ordered array, binary search tree or hash tables.

2. Reusability: Data structures are reusable, i.e. once we have implemented a particular data structure, we can use it at any other

place. Implementation of data structures can be compiled into libraries which can be used by different clients.

3. Abstraction: Data structure is specified by the ADT (Abstract Data Type) which provides a level of abstraction. The client program uses the data structure through interface only, without getting into the implementation details.

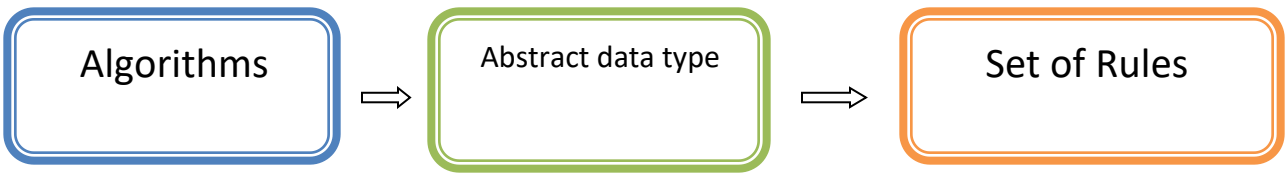
What is ADT (Abstract Data Type)?

ADTs are mathematical specifications of set of data and the set of operations that can be performed on the data without specifying the implementation details. The keyword "Abstract" is used as we can use these data types, we can perform different operations.

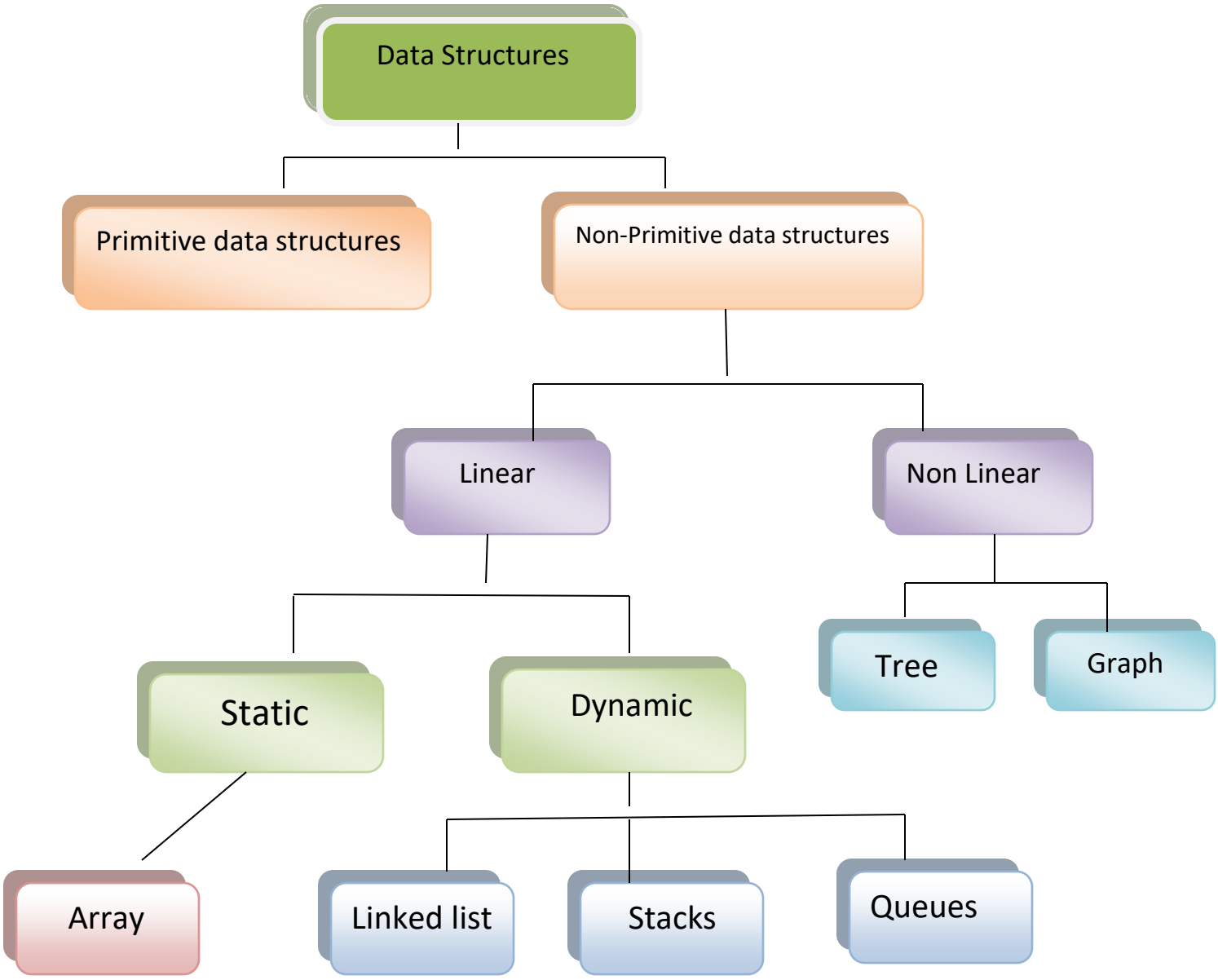
For example,

Stack is an ADT and we can perform following operations on it:

1. isFull( ): This is used to check whether the stack is full or not.
2. isEmpty( ): This is used to check whether the stack is empty or not.
3. push(x): inserting an element x into the stack.
4. pop( ): deleting an element
5. peek( ): this is used to get number of elements present in the stack



Data structures classification:



Data structures are divided into two categories.

1) Primitive data structures: The primitive data structures can be manipulated or operated by the machine instruction.

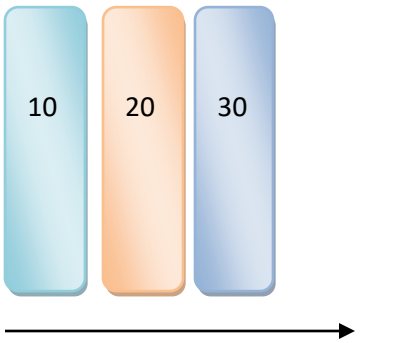
Example – int, float, char, etc are the some of the different data structures provided in c language.

2) Non-Primitive data structures: The data structures which cannot be directly operated by the machine instructions. These are derived from the primitive data structures.

Example – Arrays, Structures, Stack, Queues, Linked Lists, trees, graphs, etc.

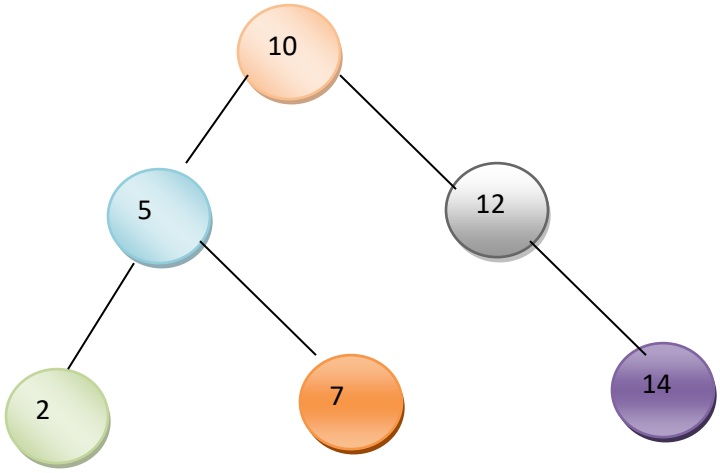
The non-primitive data structures are divided into two categories:

1. Linear Data structure
2. Non-Linear Data structure



Linear data structure

Non-Linear Data structure



1) Linear Data structure:

Collection of nodes which are logically (or physically) adjacent, i.e., logical adjacency is maintained by the pointers. – Linear relationship –sequential memory locations or links

Example – Arrays, Linked Lists, Stacks, Queues, etc.

In linear data structures, each element has the successors and predecessors except the first and last element.

In linear data structure, single level is involved. Therefore, we can traverse all the elements in single run only.

2) Non-Linear Data structure:

Non-Linear Data structure can be constructed as a collection of randomly distributed set of data items joined together by using a special pointer (tag).

In non-linear data structure the relationship of adjacency is not maintained.

Example : Tree, Graph, etc.

In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only.

Operations on data structures:

Inserting:

- To insert new element into data structure.

Deleting:

- To remove element from data structure.

Traversing:

- Visiting or accessing each element
- For better performance, visit only once.

Searching:

- Search an element is presented or not.
- return the location of item

Sorting:

- To arrange the elements in order

Merging:

- combining data items into a single list of items.