Data Structure Introduction

Data structure is a branch of computer science. The study of data structure helps you to understand how data is organized and how data flow is managed to increase efficiency of
any process or program. Data structure is the structural representation of logical relationship between data elements. This means that a data structure organizes data items
based on the relationship between the data elements.

Example:

A house can be identified by the house name, location, number of floors and so on. These structured set of variables depend on each other to identify the exact house. Similarly, data structure is a structured set of variables that are linked to each other, which forms the basic component of a system

- Data structures are the building blocks of any program or the software. Choosing the
- appropriate data structure for a program is the most difficult task for a programmer.
-

- Following terminology is used as far as data structures are concerned
- Data: Data can be defined as an elementary value or the collection of values, for
- example, student's name and its id are the data about the student.
- Group Items: Data items which have subordinate data items are called Group item, for
- example, name of a student can have first name and the last name

- Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

- File: A File is a collection of various records of one type of entity, for example, if there are 60 employees in the class, then there will be 20 records in the related file where each record contains the data about each employee.
- Attribute and Entity: An entity represents the class of certain objects. it contains various attributes. Each attribute represents the particular property of that entity.
-

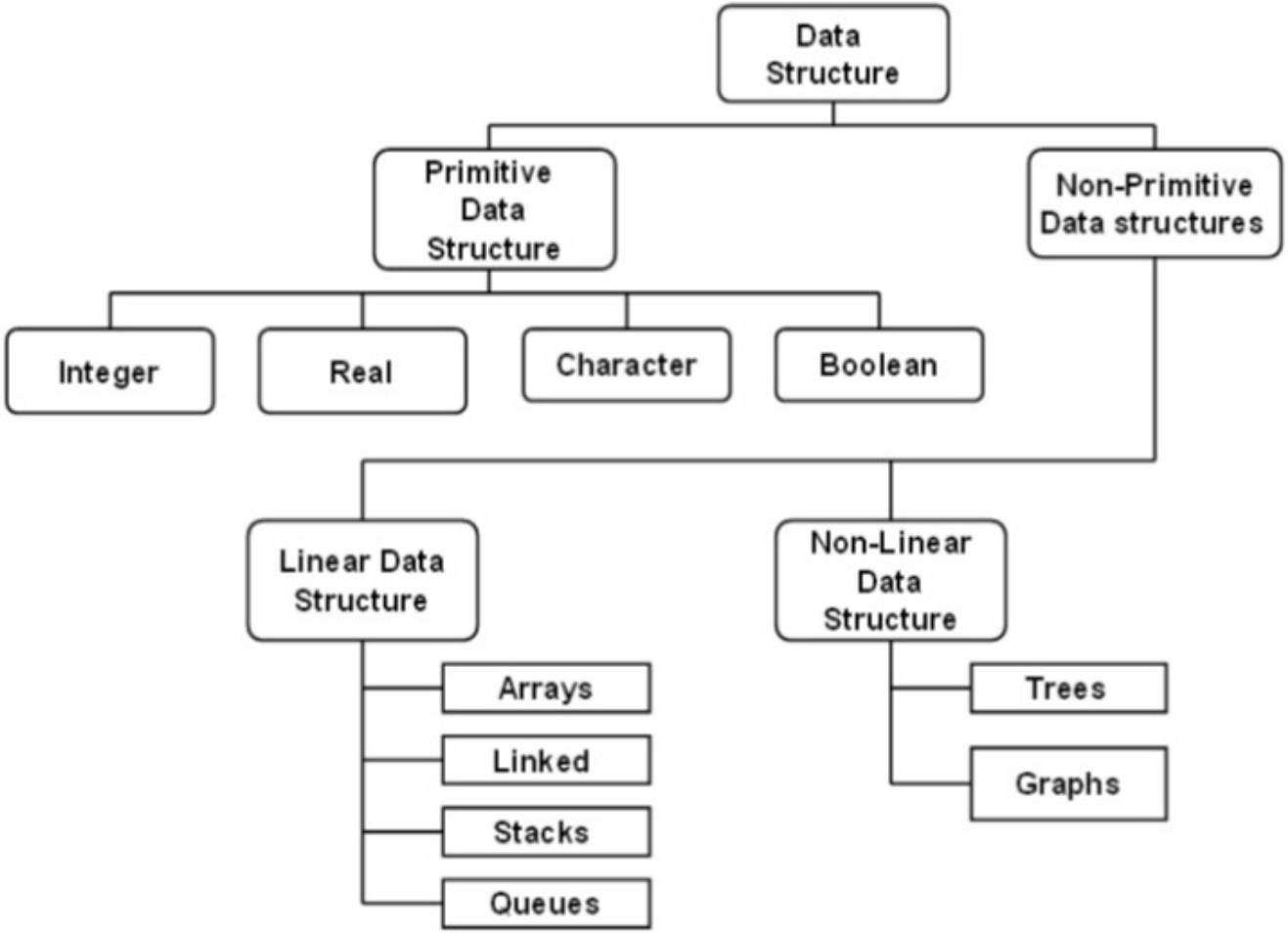
- Field: Field is a single elementary unit of information representing the attribute of an entity

- Need for 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.

- Goals of Data Structure
- Data structure basically implements two complementary goals.
- Correctness: Data structure is designed such that it operates correctly for all kinds of input, which is based on the domain of interest. In other words, correctness forms the primary goal of data structure, which always depends on the specific problems that the data structure is intended to solve.
-

- Efficiency: Data structure also needs to be efficient. It should process the data at high speed without utilizing much of the computer resources such as memory space. In a real time state, the efficiency of a data structure is an important factor that determines the success and failure of the process.
-
-

- A data structure provides a structured set of variables that are associated with each other in different ways. It forms a basis of programming tool that represents the relationship between data elements and helps programmers to process the data easily.
- Data structure can be classified into two categories:
- Primitive data structure
- Non-primitive data structure


Data Structure
Primitive Data Structure
Non-Primitive Data structures
Integer
Real
Character
Boolean
Linear Data Structure
Non-Linear Data Structure
Arrays
Linked
Stacks
Queues
Trees
Graphs

STATIC DATA STRUCTURE VS DYNAMIC DATA STRUCTURE

What is a Static Data structure?
In Static data structure the size of the structure is fixed. The content of t
data structure can be modified but without changing the memory space
located to it.
Example of Static Data Structures: Array

What is Dynamic Data Structure?
In Dynamic data structure the size of the structure in not fixed and can b
modified during the operations performed on it. Dynamic data structures
are designed to facilitate change of data structures in the run time.

OPERATIONS ON DATA STRUCTURES

- Traversing It means to access each data item exactly once so that it can be processed. For example, to print the names of all the students in a class.
- Searching It is used to find the location of one or more data items that satisfy the given constraint. Such a data item may or may not be present in the given collection of data items. For example, to find the names of all the students who secured 100 marks in mathematics.

- Inserting It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.
- Deleting It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.
-

- Sorting Data items can be arranged in some order like ascending order or descending order depending on the type of application. For example, arranging the names of students in a class in an alphabetical order, or calculating the top three winners by arranging the participants' scores in descending order and then extracting the top three.
- Merging Lists of two sorted data items can be combined to form a single list of sorted data items

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language

- From the data structure point of view, following are some important categories of algorithms −
- • Search − Algorithm to search an item in a data structure.
- • Sort − Algorithm to sort items in a certain order.
- • Insert − Algorithm to insert item in a data structure.
- • Update − Algorithm to update an existing item in a data structure.
- • Delete − Algorithm to delete an existing item from a data structure.

- Clear and Unambiguous: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- • Well-Defined Inputs: If an algorithm says to take inputs, it should be welldefined inputs.
- • Well-Defined Outputs: The algorithm must clearly define what output will be yielded and it should be well-defined as well.
- • Finite-ness: The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
- • Feasible: The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.
- • Language Independent: The Algorithm designed must be languageindependent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same,

- Different approach to design an algorithm
- Top-Down Approach: A top-down approach starts with identifying major components of system or program decomposing them into their lower level components & iterating until desired level of module complexity is achieved . In this we start with topmost module & incrementally add modules that is calls.
- 2. Bottom-Up Approach: A bottom-up approach starts with designing most basic or primitive component & proceeds to higher level components. Starting from very bottom , operations that provide layer of abstraction are implemented

- There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.
- As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm