Chapter 1: Concept of data Structures

Introduction:

Data Types:

- A data type in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it.
- A data type is the collection of values and a set of operations on the values.
- For example:
    - A string is a data type that is used to classify text.
    - An integer is a data type that is used to classify whole numbers.

Data types	Examples
Integer	1,3,15,67…
String	Hello world, ram, sita…
Float	3.14, 6.78…

Data Structure:

- A data structure is a systematic way of organizing data in a computer so that it can be used effectively.
- Data structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.
- Data structure is about rendering data elements in terms of some relationship, for better organization and storage.
- Data structure is classified into the following two categories
    - **Primitive Data Structure:** They are basic structure and are directly operated by machine instructions.
    - **Non-Primitive Data Structure:** These are more sophisticated data structure and are derived from primitive data structure. The non-primitive data structure emphasis on structuring of group of homogeneous and heterogeneous data items.

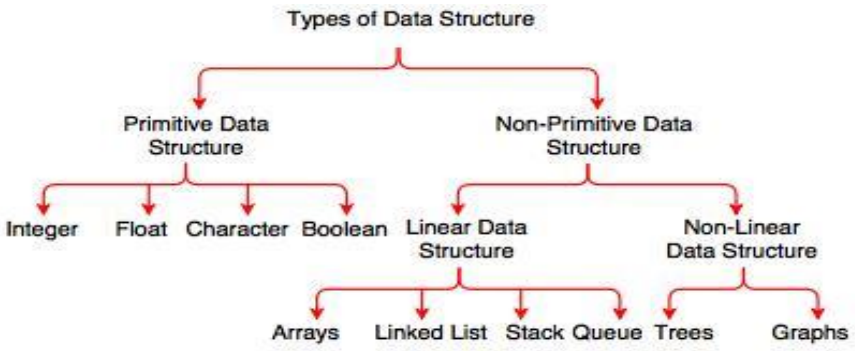


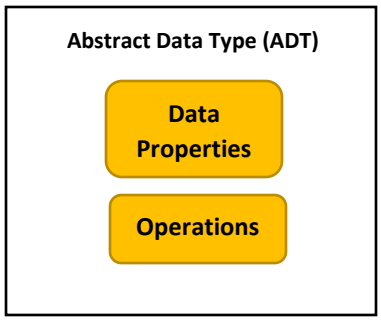
Fig. Types of Data Structure

Chapter 1: Concept of data Structures

- Data Structure = Organized Data + Allowed Operations
- Commonly used operations are Searching, Inserting, Deleting, Sorting etc.
- Example
    - An array is a data structure for storing more than one data item that has a similar data type. The items of an array are allocated at adjacent memory locations.
    - Following are the operations supported by an array.
        - Traverse: Print all the array elements one by an array.
        - Insertion: Adds an element at the given index.
        - Deletion: Deletes an element at the given index.
        - Search: Searches an element using the given index or by the value.
        - Update: Updates an element at the given index.

Abstract Data Type (ADT)

- A useful tool for specifying the logical properties of a datatype is called Abstract Data Type (ADT).
- ADT's specification describes what data can be stored (the characteristics of ADT) and how it can be used (the operations) but not how it is implemented or represented in the program.



- ADT is a type for objects whose behavior is defined by a set of values and a set of operations.
- An abstract data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects.
- It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

Specification of ADT:

For illustration, consider rational number in the form –

$$\left\{ \frac{p}{q} / \; p, q \in Z \text{ and } q \neq 0 \right\}$$

The above equation indicates that any rational number will be in the form of p divided by q, where p and q are integers (the set Z) and the denominator q is not equal to zero.

In this case, the sum of two rational numbers (a1/a2) and (b1/b2) would be –

$$\frac{a_1}{a_2} + \frac{b_1}{b_2} = \frac{a_1 * b_2 + b_1 * a_2}{a_2 * b_2}$$

Specification of any ADT consists of two parts:

1. **value definition:**
   - Here, we specify the set of possible values taken by the ADT along with some conditions or constraints with which the ADT bounds.
   - It contains definition clause and condition clause.
2. **operator definition:**
   - Here, various operations which are imposed on ADT are defined. This part contains 3 sections viz. a header, the preconditions (which is optional) and the postconditions. The term 'abstract' in the header indicates that this is not a C function; rather it is an ADT operator definition. This term also indicates that the role of an ADT is a purely logical definition of a new data type.

The following listing gives the ADT for rational numbers. The value definition here indicates the constraints on rational number. The operator definition parts contain the definition for various operations like creation of rational number, addition and multiplication of rational numbers and for checking the equality of two rational numbers.

```
// value definition
abstract typedef<integer, integer> RATIONAL;
condition RATIONAL[1] != 0;

//Operator definition
abstract RATIONAL createrational (a, b)
int a,b;
precondition b!=0;
postcondition   createrational [0] = = a;
                        createrational [1] = = b;

abstract RATIONAL add(a,b)
RATIONAL a, b;
postcondition add[0] = =  a[0]*b[1] + b[0]*a[1];
                      add[1] = = a[1]*b[1];
abstract RATIONAL mul(a,b)
RATIONAL a, b;
postcondition mul[0] = = a[0]*b[0];
                       mul[1] = = a[1]*b[1];

abstract equal(a, b)
RATIONAL a,b;
postcondition
              equal = = (a[0]*b[1] = = b[0]*a[1]);
```

Chapter 1: Concept of data Structures

Algorithms

An algorithm is a set of rules for carrying out calculations either by hand or machine. An algorithm is a sequence of computational steps that transform the input into the output. An algorithm performed on data that have to be organized in data structure. An algorithm is an abstraction of a program to execution on a physical machine.

Algorithm Design:

An algorithm design is a specific method to create a mathematical process in solving problem.

Steps for developing algorithm:

1. Problem definition
2. Specification of algorithm
3. Design an algorithm
4. Checking the correctness of algorithm
5. Analysis of algorithm
6. Implementation of algorithm
7. Problem Testing
8. Documentation

Algorithm Efficiency:

How fast is algorithm?

How much money does it cost?