Data structures and Algorithms – Introduction

Need for Data Structures :

More complex problems demand more computation, making the need for efficient programs even greater.

In the most general sense, a data structure is any data representation and its associated operations. Even an integer or floating point number stored on the computer can be viewed as a simple data structure. More typically, a data structure is meant to be an organization or structuring for a collection of data items. A sorted list of integers stored in an array is an example of such a structuring.

A solution is said to be efficient if it solves the problem within the required resource constraints. Examples of resource constraints include the total space available to store the data — possibly divided into separate main memory and disk space constraints — and the time allowed to perform each sub-task.

When selecting a data structure to solve a problem, you should follow these steps.

1. Analyze your problem to determine the basic operations that must be supported. Examples of basic operations include inserting a data item into the data structure, deleting a data item from the data structure, and finding a specified data item.
2. Quantify the resource constraints for each operation.
3. Select the data structure that best meets these requirements

A data type is a type together with a collection of operations to manipulate the type. For example, an integer variable is a member of the integer data type.
Addition is an example of an operation on the integer data type.

An abstract data type (ADT) is the realization of a data type as a software component. The interface of the ADT is defined in terms of a type and a set of operations on that type. The behavior of each operation is determined by its inputs and outputs. An ADT does not specify how the data type is implemented. These implementation details are hidden from the user of the ADT and protected from outside access, a concept referred to as encapsulation.

Example) An ADT for a list of integers might specify the following operations:

- Insert a new integer at a particular position in the list.
- Return true if the list is empty.
- Reinitialize the list.
- Return the number of integers currently in the list.
- Delete the integer at a particular position in the list.

From this description, the input and output of each operation should be clear, but the implementation for lists has not been specified.

Example When operating a car, the primary activities are steering, accelerating, and braking. On nearly all passenger cars, you steer by turning the steering wheel, accelerate by pushing the gas pedal, and brake by pushing the brake pedal. This design for cars can be viewed as an ADT with operations “steer,” “accelerate,” and “brake.” Two cars might implement these operations in radically different ways, say with different types of engine, or front- versus rear-wheel drive. Yet, most drivers can operate many different cars because the ADT presents a uniform method of operation that does not require the driver to understand the specifics of any particular engine or drive design. These differences are deliberately hidden.

Example We apply the label “hard drive” to a collection of hardware that manipulates data on a particular type of storage device, and we apply the label “CPU” to the hardware that controls execution of computer instructions. These and other labels are gathered together under the label “computer.” Because even small home computers have millions of components, some form of abstraction is necessary to comprehend how a computer operates.

Data types have both a logical and a physical form. The definition of the data type in terms of an ADT is its logical form. The implementation of the data type as a data structure is its physical form.