Abstract Data Types and Related Terms

**abstract, verb.** *to draw out the essence of a matter or to separate the fundamentals from irrelevant materials which surround them.*

Abstraction with respect to a data type means that we focus on what the data type can do, and not on how it is done. In other words, we postpone, for the time being, the discussion about actual implementation of the data type.

- View screen images as objects such as people, trees and mountains rather than as individual dots of color.
- Think in terms of sea beaches rather than grains of sand.
- Look at houses rather than bricks.

An **abstract data type**, or ADT, consists of a set of data values and a set of operations associated with the data values. While operations are precisely specified, they are independent of any particular implementation. Thus, data abstraction allows users and implementers to take different views of a specification.

The data values and related operations are defined with mathematical precision. Therefore, we may reason about effects of the operations regardless of whether a program actually implements the data type or not. Note that an ADT may be viewed as a black box where data values and associated functions are prominently specified and internal details are hidden. Here are certain related technical terms:

(i) The set of operations associated with an ADT is called the *interface* of the ADT. Operations of the interface are the only mechanism to access or change the data values.

(ii) The design decisions with respect to actual implementation may change over the lifetime of an ADT, and this process is called *information hiding*.

(iii) The principle of providing a well-defined interface and hiding the actual implementation is called *encapsulation*. Thus, encapsulation refers to bundling of data with the methods that operate on that data.
Hiding information isolates clients from requiring knowledge of the design to use a data type and from the effects of changing those decisions. Whereas information hiding is a design principle, encapsulation is a language facility.

Closely associated with abstraction is the concept of modularity, which is a technique of subdividing a solution into certain units (called modules), where each unit focuses on one task at a time, without other distractions. Modularity keeps the complexity of a large program manageable by systematically controlling the interaction of its components.

Modularity and abstraction complement each other. While modularity breaks a solution into manageable units, abstraction clearly specifies each module prior to actual implementation in a programming language.

Each module in a solution begins as a box that states what it does but not how it does it. For example, if one part of a solution is to sort some data, one of the boxes will be a sorting module, as shown in the figure below.

The other boxes will know that the sorting box does the actual sorting, but they will not know how the sorting is actually performed (quicksort or heapsort or mergesort). In this way, various components of a solution are kept isolated from each other.