Basics of Data Structures

Introduction to Data Structures

Data Structure is an arrangement of data in a computer's memory (or sometimes on a disk). Data structures include arrays, linked lists, stacks, binary trees, and hash tables, among others. Algorithms manipulate the data in these structures in various ways, such as searching for a particular data item and sorting the data.

Overview of Data Structures

Another way to look at data structures is to focus on their strengths and weaknesses. Table 1.1 shows the advantages and disadvantages of the various data structures.

TABLE 1.1 Characteristics of Data Structures

Data Structure	Advantages	Disadvantages
Array	Quick insertion, very fast access if index known.	Slow search, slow deletion, fixed size.
Ordered array	Quicker search than unsorted array	Slow insertion and deletion, fixed size.
Stack	Provides last-in, first-out access.	Slow access to other items.
Queue	Provides first-in, first-out access.	Slow access to other items.
Linked list	Quick insertion, quick deletion.	Slow search.
Binary tree	Quick search, insertion, deletion (if tree remains balanced).	Deletion algorithm is complex.
Red-black tree	Quick search, insertion, deletion. Tree always balanced.	Complex.
2-3-4 tree	Quick search, insertion, deletion. Tree always balanced. Similar trees good for disk storage.	Complex.
Hash	tableVery fast access if key known. Fast insertion	Slow deletion, access slow if key not known, inefficient memory usage.
Heap	Fast insertion, deletion, access to largest item.	Slow access to other items.
Graph	Models real-world situations.	Some algorithms are slow and complex.

The data structures shown in Table 1.1, except the arrays, can be thought of as Abstract Data Types, or ADTs.

A data type consists of

- a domain (= a set of values)
- A set of operations.

Example 1: *Boolean* or *logical* data type provided by most programming languages.

- Two values: true, false.
- Many operations, including: AND, OR, NOT, etc.

Abstract Data Type (**ADT**): The basic idea behind an abstract data type is the **separation of the use of the data type from its implementation** (i.e., what an abstract data type does can be specified separately from how it is done through its implementation)
The advantages of using the **ADT** approach are:

1. The implementation of **ADT** can change without affecting those method that use the **ADT**
2. The complexity of the implementation are hidden

For example, an **abstract stack data structure** could be defined by two operations: **push**, that inserts some data item into the structure, **and pop**, that extracts an item from it.

Advantages of data structures

The major advantages of data structures are:
• It gives different level of organizing data.
• It tells how data can be stored and accessed in its elementary level

Operation on Data Structures

The operations that can be performed on data structures are:
Creation: This operation creates a data structure. The declaration statement causes space to be created for data upon entering at execution time.

Destroy: This operation destroys the data structure and aids in the efficient use of memory.

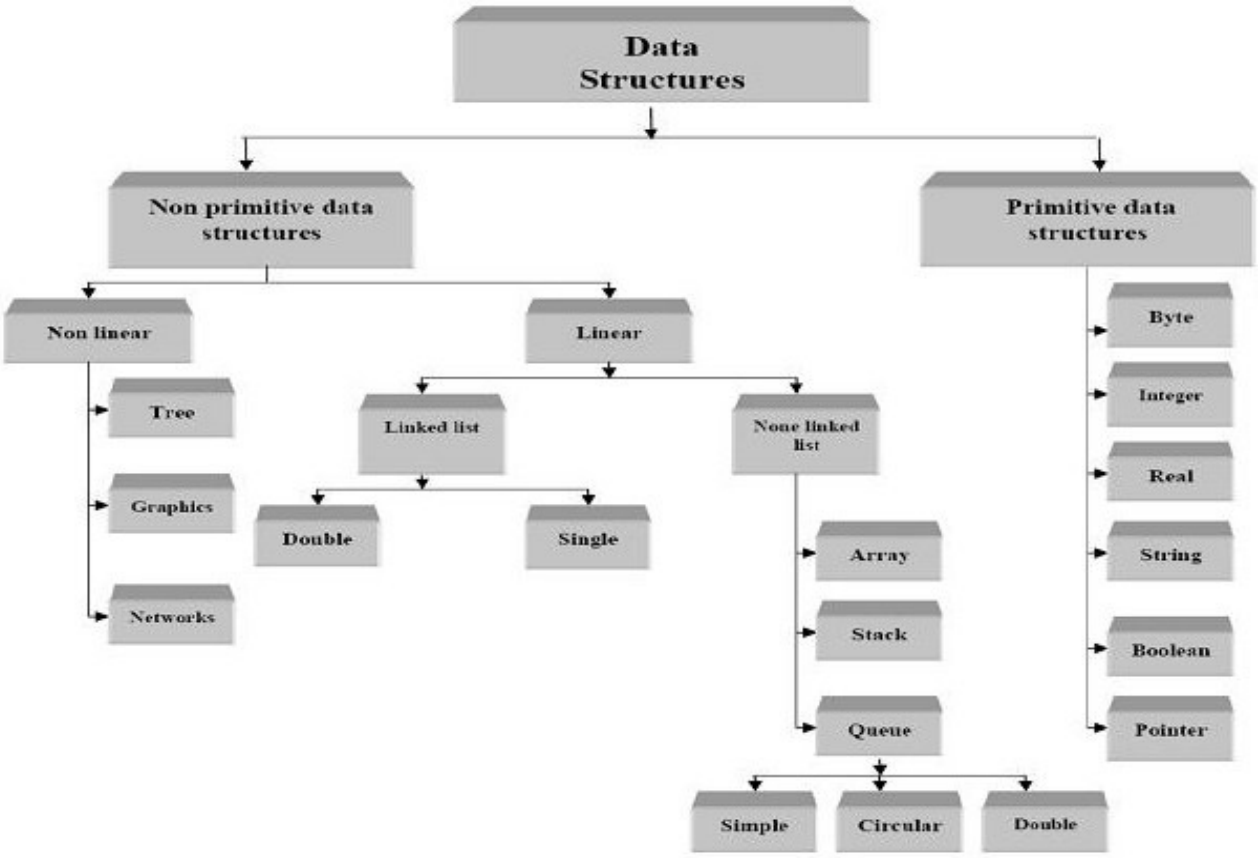
Selection: This operation is used to access data within a data structure. The form of selection depends on the type of data structure being accessed.

Update: This operation changes or modifies the data in the data structure and it is an important property in selection operation.

Types of Data Structures

Linear Data Structure: Stacks, Queues, Linked Lists, etc.

Non-linear Data Structure: Trees, Graphs, etc .
As illustrated in block diagram below.



Block diagram of Data Structures

Difference between Linear and Nonlinear Data Structures

Main difference between linear and nonlinear data structures lie in the way they organize data elements. In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. Non-Linear data structure is that if one element can be connected to more than two adjacent element then it is known as non-linear data structure.

Overview of Algorithms

Many of the algorithms we'll discuss apply directly to specific data structures. For most data structures, you need to know how to:

- Insert a new data item.
- Search for a specified item.
- Delete a specified item.

The concept of **recursion** is important in designing certain algorithms. Recursion involves a method **calling itself**.

Problems with Procedural Languages

OOP was invented because procedural languages, such as C, Pascal, and early versions of BASIC, were found to be inadequate for large and complex programs. **Why was this**?

There were **two kinds of problems**. **One** was the lack of correspondence between the program and the real world, **and the other** was the internal organization of the program.

1. Poor Modeling of the Real World

Conceptualizing a real-world problem using procedural languages is difficult. Methods carry out a task, while data stores information, but most real-world objects do both of these things.

For large programs, which might contain hundreds of entities like thermostats, this procedural approach made things chaotic, error-prone, and sometimes impossible to implement at all. What was needed was a better match between things in the program and things in the outside world.

2. Crude Organizational Units

A more subtle, but related, problem had to do with a program's internal organization. Procedural programs were organized by dividing the code into methods. One difficulty with this kind of method-based organization was that it focused on methods at the expense of data. There weren't many options when it came to data.
To simplify slightly, data could be local to a particular method, or it could be global—accessible to all methods. There was no way (at least not a flexible way) to specify that some methods could access a variable and others couldn't.

This **inflexibility** caused problems when several methods needed to access the same data. To be available to more than one method, such variables needed to be global, but global data could be accessed inadvertently by any method in the program. This lead to frequent programming errors. What was needed was a way to fine-tune data accessibility, allowing data to be available to methods with a need to access it, but hiding it from other methods.

Arrays

The array is the most commonly used data storage structure; it's built into most programming languages.
We can define an array, as numbered collection of variables all of the same type. Each variable, or cell, in an array has an index, which uniquely refers to the value stored in that cell. The cells of an array a are numbered 0, 1,2, and so on.

Exercise
Write Java prgram to insert ten item in array