DATA STRUCTURES

Data Structures and Algorithms

A ***data structure*** is an arrangement of data in a computer's memory or even disk storage. An example of several common data structures are arrays, linked lists, queues, stacks, trees,….etc.

Algorithms, on the other hand, are used to manipulate the data contained in these data structures as in searching and sorting.

Many algorithms apply directly to a specific data structures. When working with certain data structures you need to know how to:
insert new data.
search for a specified item.
deleting a specific item.
Sort the data items in the data structure.

Abstract Data Types

An ***Abstract Data Type* (ADT)** is more a way of looking at a data structure: focusing on what it does and ignoring how it does its job.

A stack or a queue is an example of an ADT. It is important to understand that both stacks and queues can be implemented using an array. It is also possible to implement stacks and queues using a linked list. This demonstrates the "abstract" nature of stacks and queues: how they can be considered separately from their implementation.

To best describe the term Abstract Data Type, it is best to break the term down into **"data type"** and then **"abstract".**

Data type

When we consider a primitive type we are actually referring to two things: a data item with certain characteristics and the permissible operations on that data. The data type's permissible operations are an inseparable part of its identity; understanding the type means understanding what operations can be performed on it.

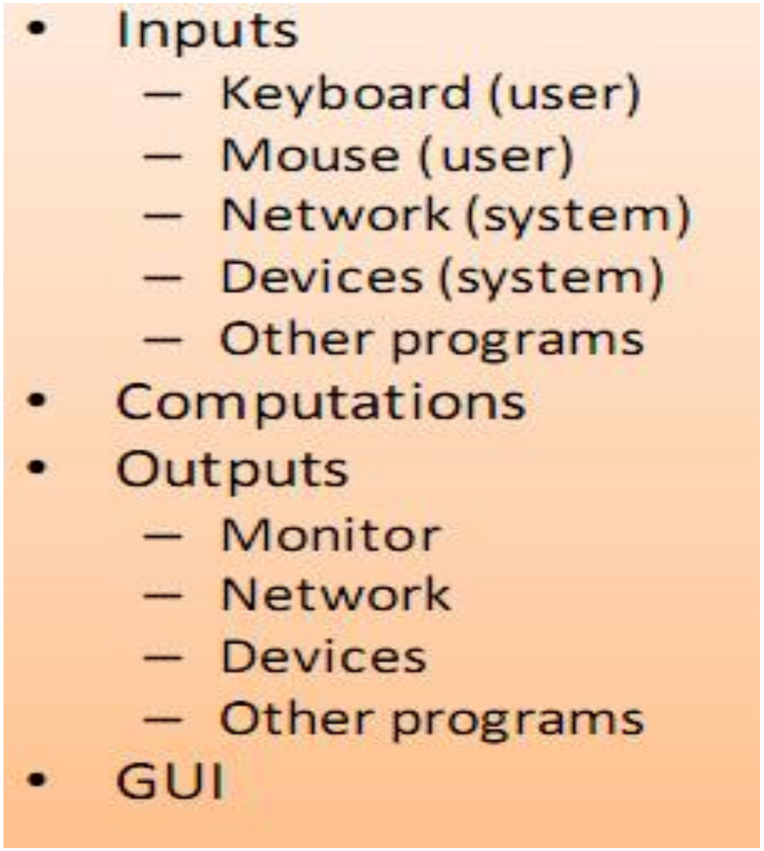
Abstract

Now lets look at the "abstract" portion of the phrase. The word abstract in our context stands for "considered apart from the detailed specifications or implementation".

Characteristics of Data Structures

Data Structure	Advantages	Disadvantages
Array	Quick inserts Fast access if index known	Slow search Slow deletes Fixed size
Ordered Array	Faster search than unsorted array	Slow inserts Slow deletes Fixed size
Stack	Last-in, first-out acces	Slow access to other items
Queue	First-in, first-out access	Slow access to other items
Linked List	Quick inserts Quick deletes	Slow search
Binary Tree	Quick search Quick inserts (If the tree remains balanced)	Deletion algorithm is complex
Graph	Best models real-world situations	Some algorithms are slow and very complex

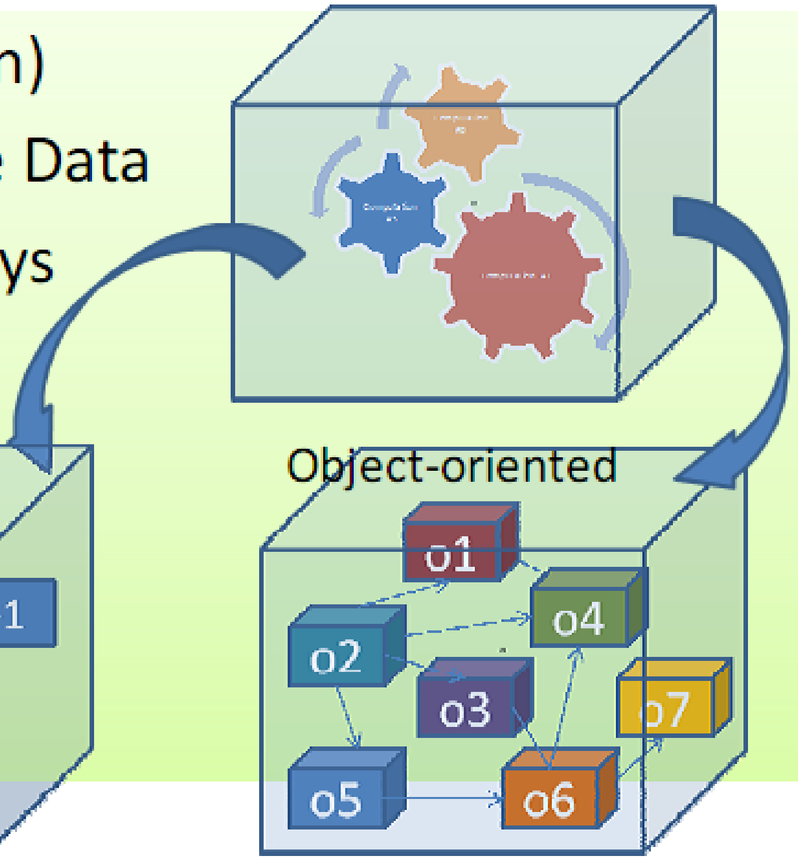
A Programming Solution



- Inputs
  - Keyboard (user)
  - Mouse (user)
  - Network (system)
  - Devices (system)
  - Other programs
- Computations
- Outputs
  - Monitor
  - Network
  - Devices
  - Other programs
- GUI

Computation

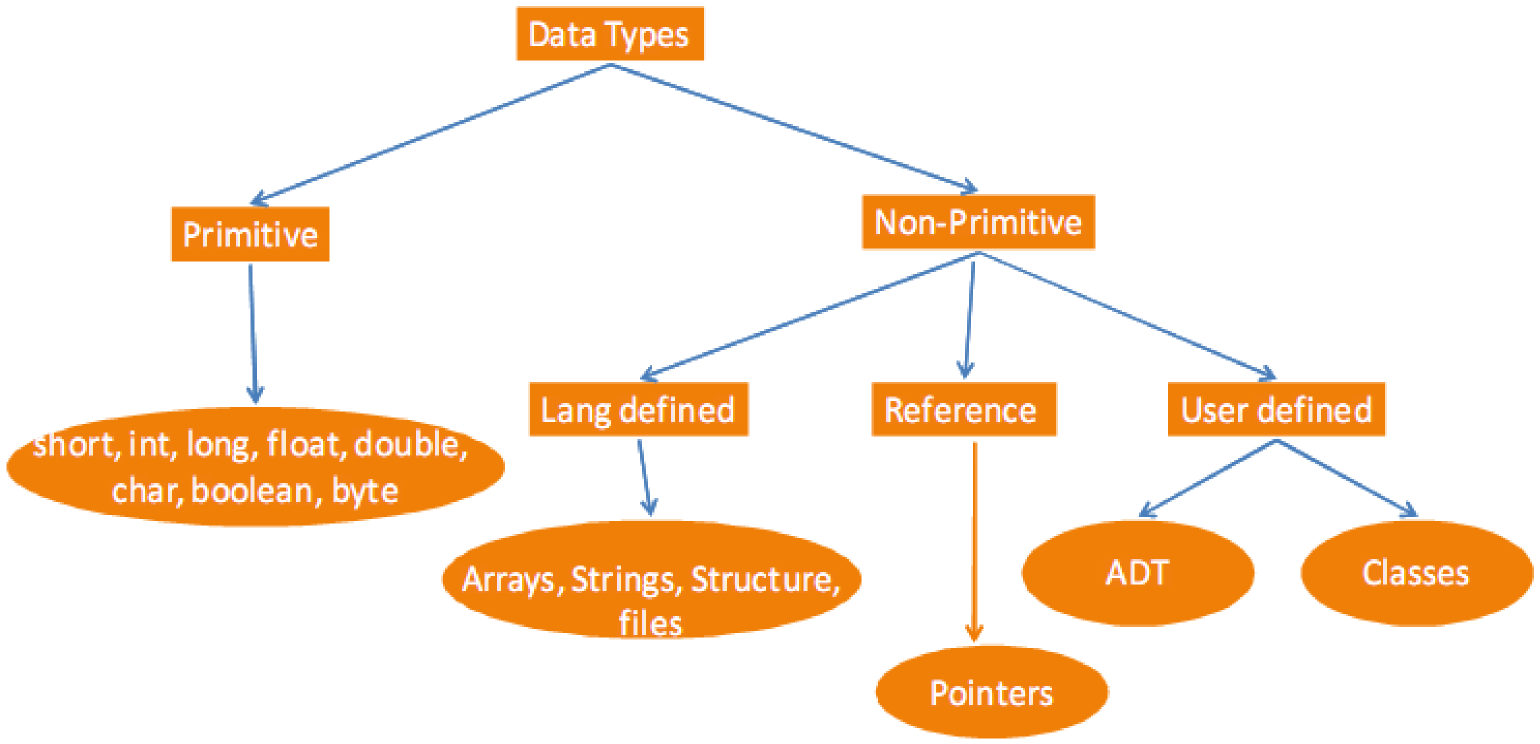
- Data (Information)
- Processing of the Data
- Two principle ways of Computation



Data Types

- Set of values
- Operations that can be performed on those values
  - Ex: **short int**
    - can take values (-32768 to 32767)
    - Operations are +, -, ×, /

Data Types


Data Types
Primitive
Non-Primitive
short, int, long, float, double, char, boolean, byte
Lang defined
Reference
User defined
Arrays, Strings, Structure, files
Pointers
ADT
Classes

Abstract Data Types

- Both an interface and an implementation
- Interface and implementation are independent
- Interface defines
  - the type of the data stored
  - operations that are performed on the data
  - parameters of each operation
- Implementation defines
  - data organization
  - developing efficient algorithm for each operation
- We will discuss both interface and implementation

Abstract Data Types: Benefits

- Encapsulation: Separation of concerns
- Module independence
  - Division of labor
  - Facilitates unit-testing
- Reuse
  - COTS
- Cheaper sub-contracts