Lecture 01: An Introduction to Abstract Data Type (ADT)
The 3 Important Techniques

- **Algorithm**: Outline, the essence of a computational procedure, step by step instructions.
- **Program**: An implementation of an algorithm in some programming languages.
- **Data structure**: Organization of data needed to solve the problem.
Important factors of Algorithm Design

- Correctness
- Efficiency
Algorithm Design

- Uses a high level description of the algorithm, instead of testing one of its implementations
- Take into account all possible inputs
- Allows one to evaluate the efficiency of any algorithm in a way that is independent of the hardware and software environment
Pseudo-Code

A mixture of natural language and high level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.

Eg: Algorithm arrayMax(A, n):
    Input: An array A storing n integers.
    Output: The maximum element in A.
    currentMax ← A[0]
    for i ← 1 to n-1 do
        if currentMax < A[i] then currentMax ← A[i]
    return currentMax
What is a Data type?

**Data type:** a data type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data.
What is a Primitive Data type?

- **Primitive data type:** a primitive data type is the most basic data type available within a programming language. There are 8 primitive data types in Java: boolean, byte, char, short, int, long, float and double. These types serve as the building blocks of data manipulation in Java.
What is a Reference Data type?

**Reference data type:** A reference data type is associated with Class and Objects. The possible values of a class are the objects. The data representation is a reference (pointer), stored in the stack, to a block of storage in the heap
- The structure of this block is defined by the fields (both inherited and immediate) of the class.
- The operations are the methods
- Objects are often mutable or immutable. (Example?)
What is an Abstract Data Type (ADT)?

**Abstract data type:** ADT is mathematically specified entity that defines a set of its instances with:
- a specific interface - a collection of signatures of operations that can be invoked on an interface.
- a set of axioms (preconditions and postconditions) that define the semantics of the operations (i.e., what the operations do to the instances of ADT but not how)
ADTs - What are the types of operations?

- Constructor
- Access functions
- Manipulation procedures
Why do we need to talk about ADT in the algorithm course?

- They serve as the specifications of requirements for the building blocks of solutions to algorithmic problems
- Provide a language to talk on a higher level of abstraction
- ADTs encapsulate data structures and algorithms that implement them
- Separate the issues of correctness and efficiency
Let us look at a simple example of ADT

- We will deal with ADTs, instances of which are sets of some type of elements.
  - Operations are provided that change the set
- We will call such class of ADTs dynamic sets
Dynamic Sets

- **Methods:**
  - `New()`: ADT
  - `Insert(S:ADT, v:element):ADT`
  - `Delete(S:ADT, v:element):ADT`
  - `IsIn(S:ADT, v:element):boolean`

- **New - Constructor**
- **Insert and Delete - Manipulation procedures**
- **IsIn - Access functions**
Dynamic Sets

Axioms: Axioms that define the methods:
- $\text{IsIn}(\text{New}(), v) = \text{false}$
- $\text{IsIn}(\text{Insert}(S, v), v) = \text{true}$
- $\text{IsIn}(\text{Insert}(S, u), v) = \text{IsIn}(S, v)$ if $u \neq v$
- $\text{IsIn}(\text{Delete}(S, v), v) = \text{false}$
- $\text{IsIn}(\text{Delete}(S, u), v) = \text{IsIn}(S, v)$ if $u \neq v$
Other Example of ADTS

- Stacks
- Queue
Questions