SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

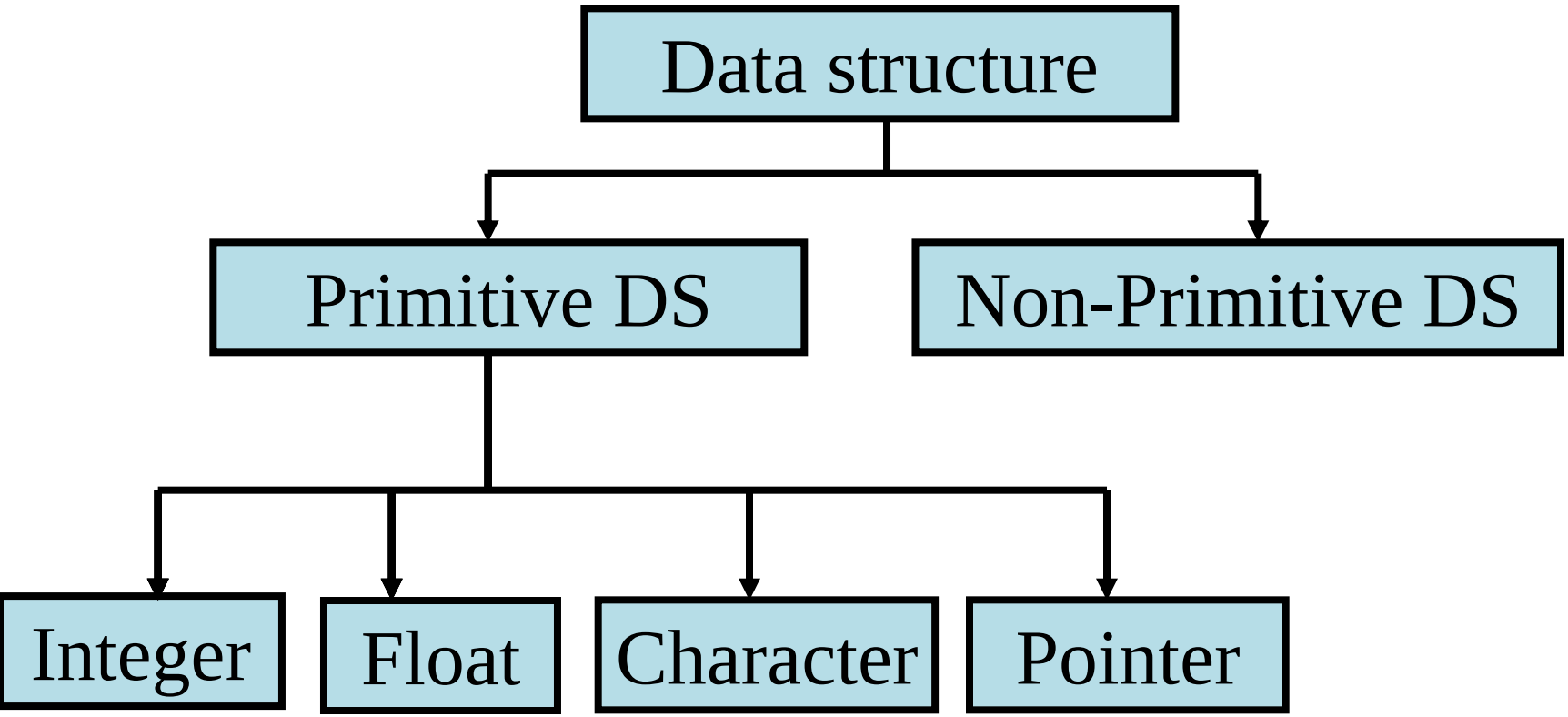
DATA STRUCTURES – TYPES AND ADT

Somaiya TRUST

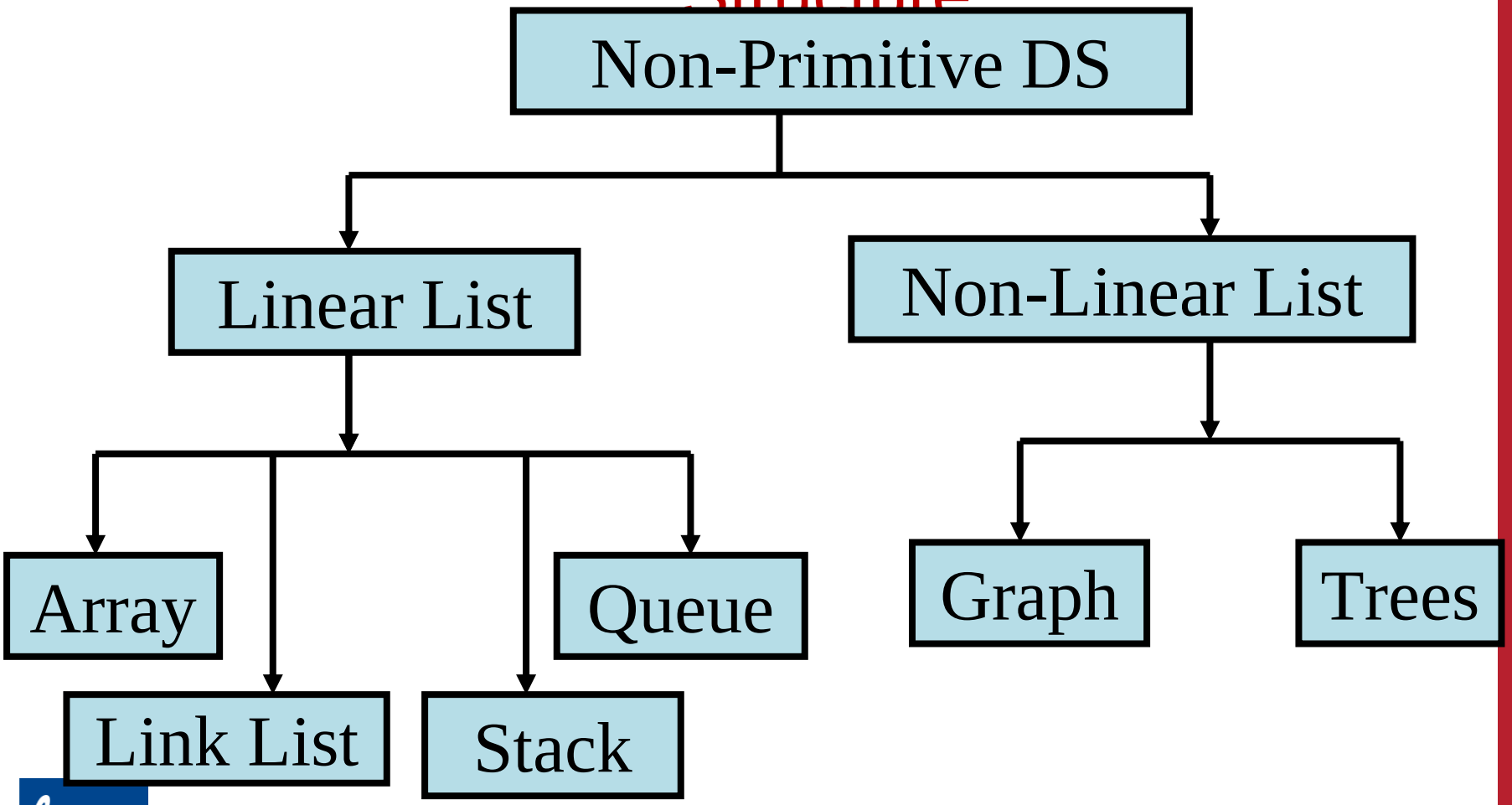
Classification of Data Structure

- Primitive Data Structure
- Non-Primitive Data Structure

Classification of Data Structure



Classification of Data Structure



Primitive data structures

- Basic structures that are directly operated upon by the machine instructions.
- Usually built into the language, such as an integer, a float.

Non-Primitive data structures

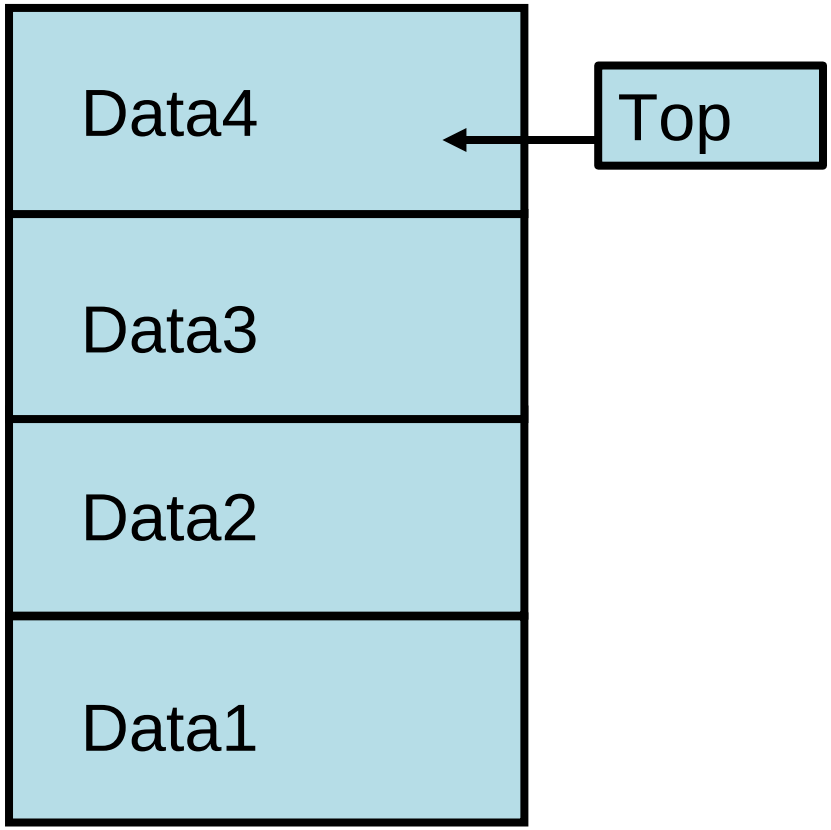
- More sophisticated data structures.
- Derived from the primitive data structures.
- Emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

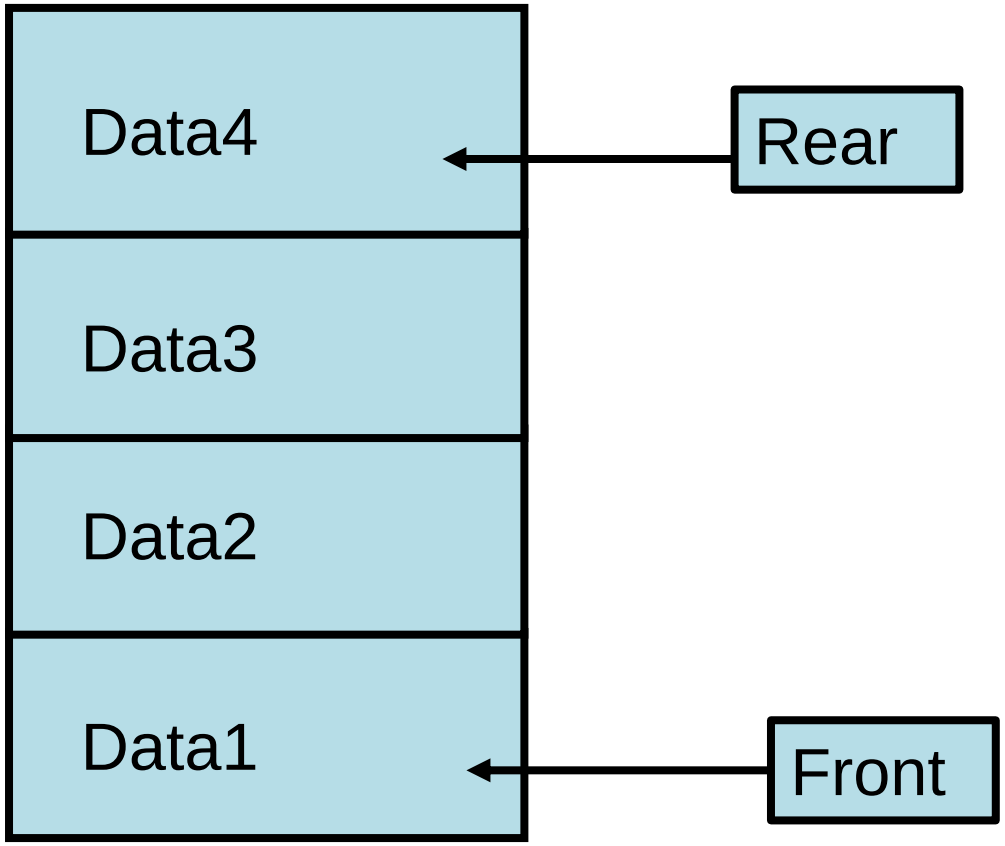
Data structures and their representations

Somaiya TRUST

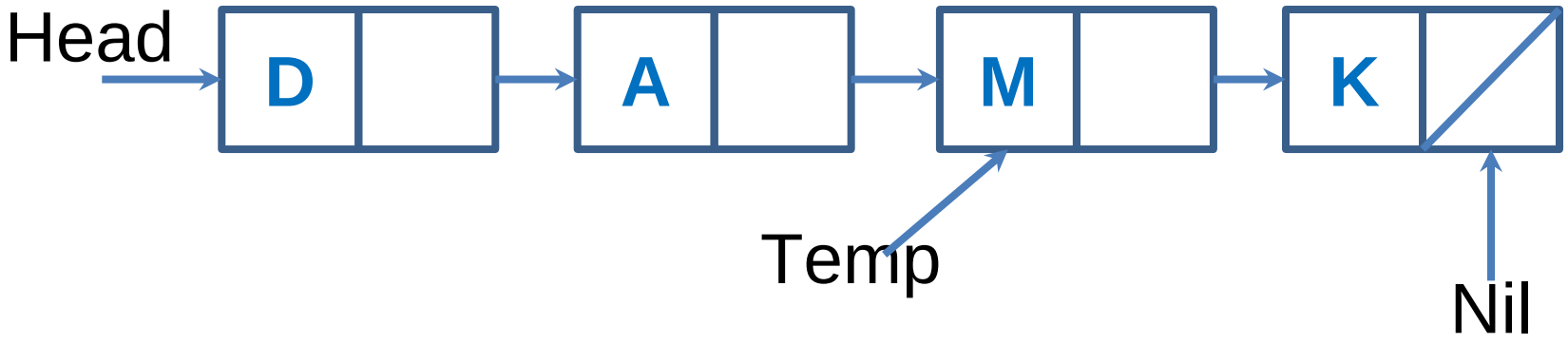
Stack

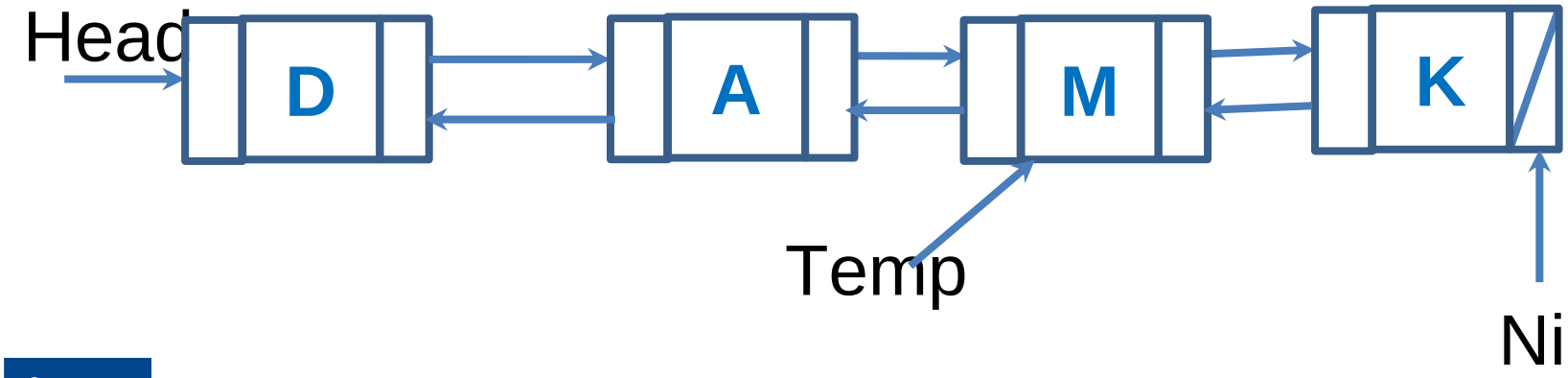


Queue

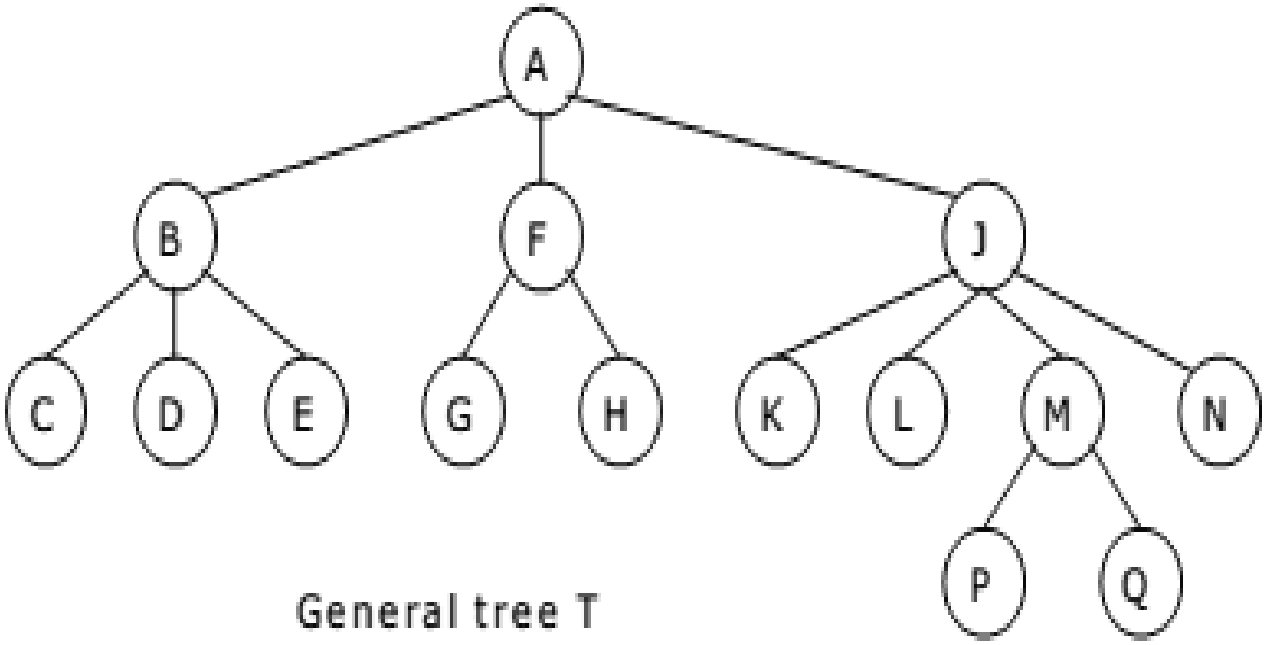


List- A ***Flexible*** structure that can grow and shrink on demand





Tree



General tree T

Image courtesy: ExamRadar.com

Binary Tree, Binary search tree and Heaps

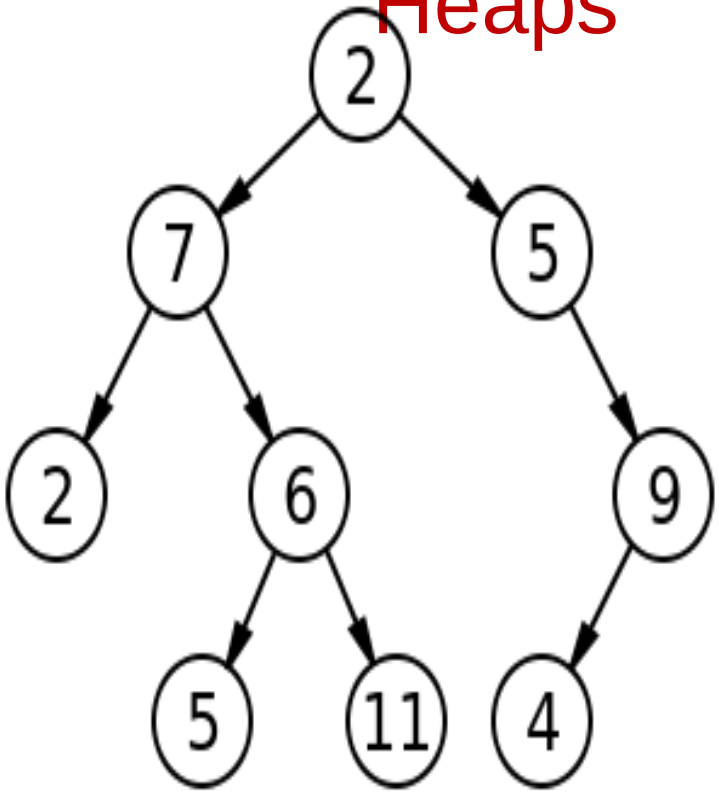


Image courtesy: ExamRadar.com

Graph

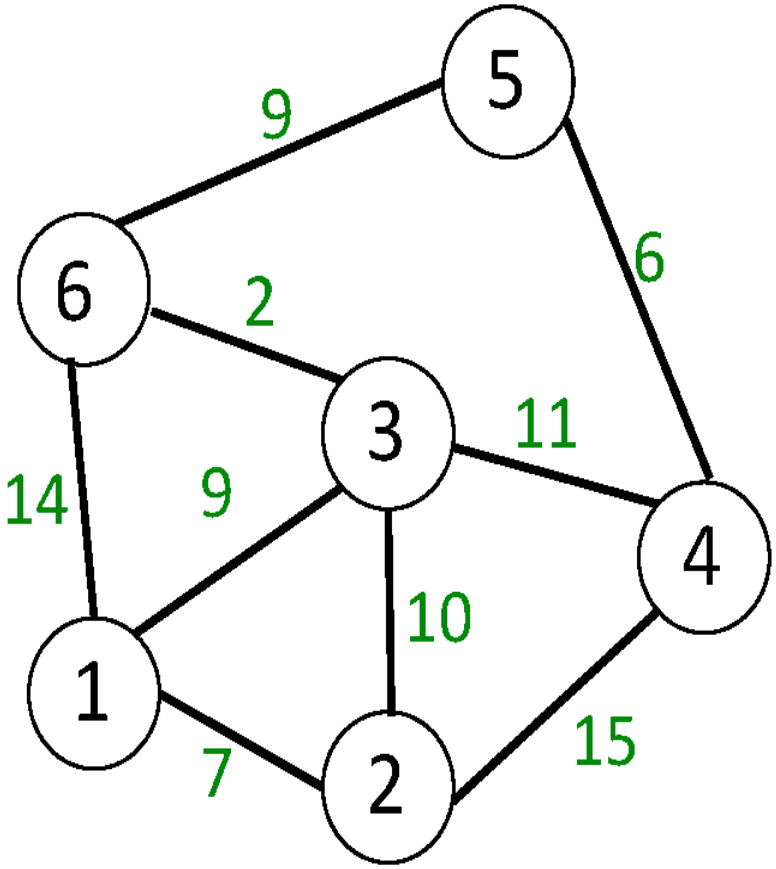


Image courtesy: Medium.com

Abstract Data Type and Data structures

- Abstract Data Type is the abstraction of data structures which provides the interface which the data structure should adhere too

- The interface doesn't specify details about how something has to implement and in which programming language

Abstract Data Type and Data Structure definition

- Definition:-
  - An ADT is a collection of data and associated operations for manipulating that data
  - A mathematical model, together with various operations defined on the model

ADT Operations

Every Collection ADT should provide a way to:

- Create data structure
- add an item
- remove an item
- find, retrieve, or access an item

No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them

Example ADT : String

- Definition: String is a sequence of characters
- Operations:
  - StringLength
  - StringCompare
  - StringConcat
  - StringCopy

ADT Syntax : Value Definition

Abstract typedef < *ParameterType Parameter1, ParameterType Parameter2……, ParameterType ParameterN* > ADTType

condition:

ADT Syntax : Operator definition

Abstract ReturnType OperationName (ParameterType Parameter1, ParameterType Parameter2……, ParameterType ParameterN)

Precondition:

Postcondition:

OR

Abstract ReturnType OperationName (Parameter1, Parameter2……, ParameterN)

ParameterType Parameter1, ParameterType Parameter2……, ParameterType ParameterN

Precondition:

Postcondition:

Example ADT : String

- Value Definition

Abstract Typedef StringType<<Chars>>

Condition: None (A string may contain n characters where n=>0)

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Example ADT : String Operator Definition

1. abstract Integer StringLength (StringType String)

Precondition: None (A string may contain n characters where n=>0)

Postcondition: Stringlength= NumberOfCharacters(String)

Somaiya TRUST

Example ADT : String Operator Definition

2. abstract StringType StringConcat(StringType String1, StringType String2)

Precondition: None

Postcondition: StringConcat= String1+String2 / All the characters in Strings1 immediately followed by all the characters in String2 are returned as result.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Example ADT : String Operator Definition

3. abstract Boolean StringCompare(StringType String1, StringType String2)

Precondition: None

Postcondition: StringCompare= True if strings are equal, StringCompare= False if they are unequal . (Function returns 1 if strings are same, otherwise zero)

Somaiya TRUST

Example ADT : String Operator Definition

4. abstract StringType StringCopy(StringType String1, StringType String2)

Precondition: None

Postcondition: StringCopy: String1= String2 / All the characters in Strings2 are copied/overwritten into String1.

Example ADT : Rational Number

- Definition: expressed as the quotient or fraction of two integers,
- Operations:
  - IsEqualRational()
  - MultiplyRationa()
  - AddRational()

Example ADT : Rational Number

- Value Definition

abstract TypeDef<integer, integer> RATIONALType;

Condition: RATIONALType [1]!=0;

Example ADT : Rational Number Operator Definition

- abstract RATIONALType makerational<a,b>

integer a,b;

Preconditon: b!=0;

postcondition :

makerational [0] =a;
makerational [1] =b;

- abstract RATIONALtype add<a,b>

RATIONALType a,b;

Precondition: none

postcondition :

add[0] = a[0]*b[1]+b[0]*a[1]

add[1] = a[1] * b[1]

Example ADT : Rational Number Operator Definition

- abstract RATIONALType mult<a, b>

RATIONALType a,b;

Precondition: none

postcondition

mult[0] = = a[0]*b[0]

mult[1] = = a[1]*b[1]

- abstract RetunType? Equal<a,b>

RATIONALType a,b;

Precondition: none

postcondition equal = |a[0] * b[1] = = b[0] * a[1];

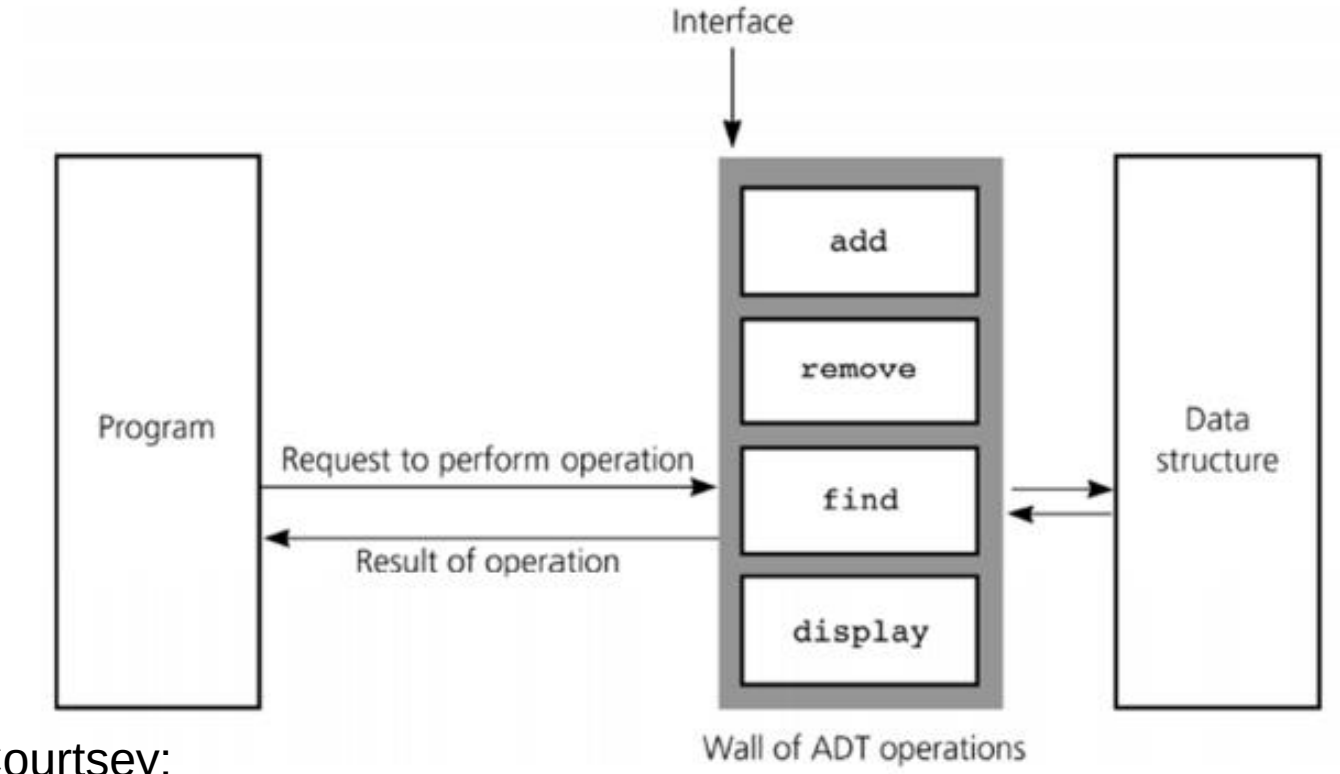
Abstract Data Types: Advantages

- Hide the unnecessary details by building walls around the data and operations
  - o that changes in either will not affect other program components that use them
- Functionalities are less likely to change
- Localize rather than globalize changes
- Help manage software complexity
- Easier software maintenance

Courtesy: https://www.comp.nus.edu.sg/~stevenha/cs1020e/lectures/L5%20-%20ADT.pdf

A wall of ADT operations

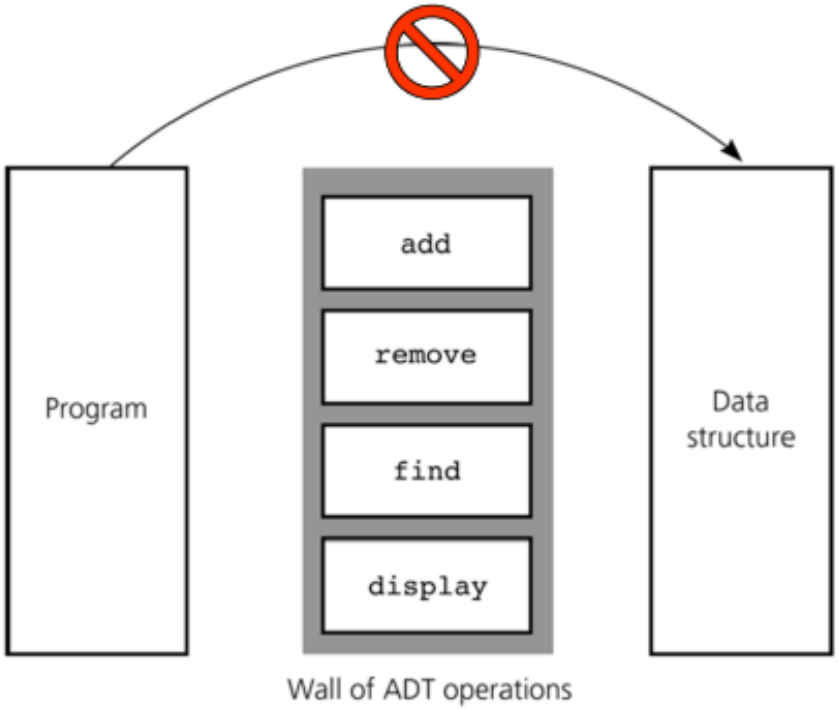
- ADT operations provides:
  - Interface to data structure
  - Secure access



Courtsey: https://www.comp.nus.edu.sg/~stevenha/cs1020e/lectures/L5%20-%20ADT.pdf

Violating the Abstraction

- User programs **should not**:
  - Use the underlying data structure directly
  - Depend on implementation details



Courtsey:
https://www.comp.nus.edu.sg/~stevenha/cs1020e/lectures/L5%20-%20ADT.pdf

Abstract Data Types: **Advantages**

- Hide the unnecessary details by building walls around the data and operations
  - So that changes in either will not affect other program components that use them

- Functionalities are less likely to change
- Localise rather than globalise changes
- Help manage software complexity
- Easier software maintenance

ADT Implementation

- Computer languages do not provide complex ADT packages.
- To create a complex ADT, it is first implemented and kept in a library.

Write Array as ADT

- Operations:
  - Create
  - Add
  - Delete
  - Sum
  - Search
  - SizeOfArray

SOMAIYA VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering

Thank you

Somaiya TRUST