SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

DATA STRUCTURES – TYPES AND ADT

sushmakadge@somaiya.edu
swatimali@somaiya.edu

Somaiya TRUST

Classification of Data Structure

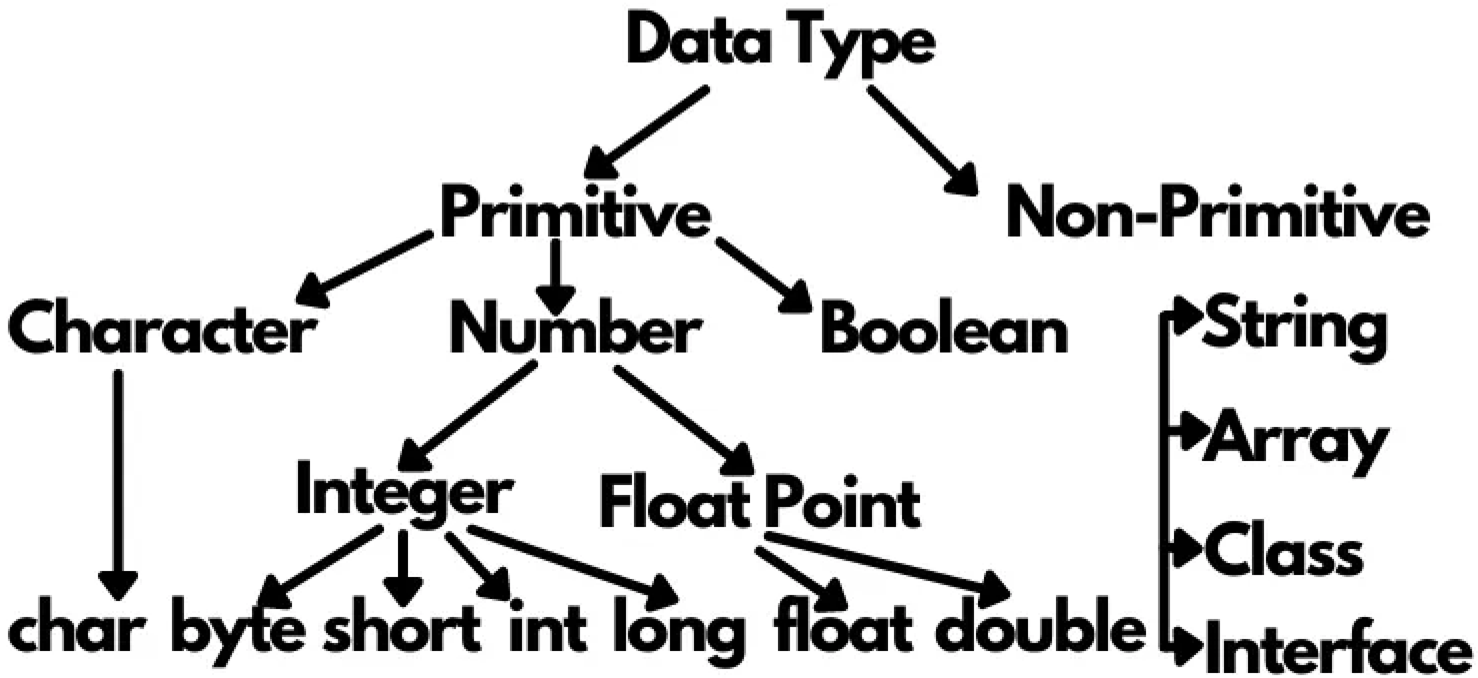
- Primitive Data Structure

  - are the basic DS that directly operate upon the machine instructions.
  - can store the value of only one data type. example, a char data structure can store only characters.

- Non-Primitive Data Structure
  - are more complicated DS
  - are derived from primitive DS.
  - they emphasize on grouping same or different data items with relationship between each data item. example, arrays. Lists and files come under this category

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Classification of Data

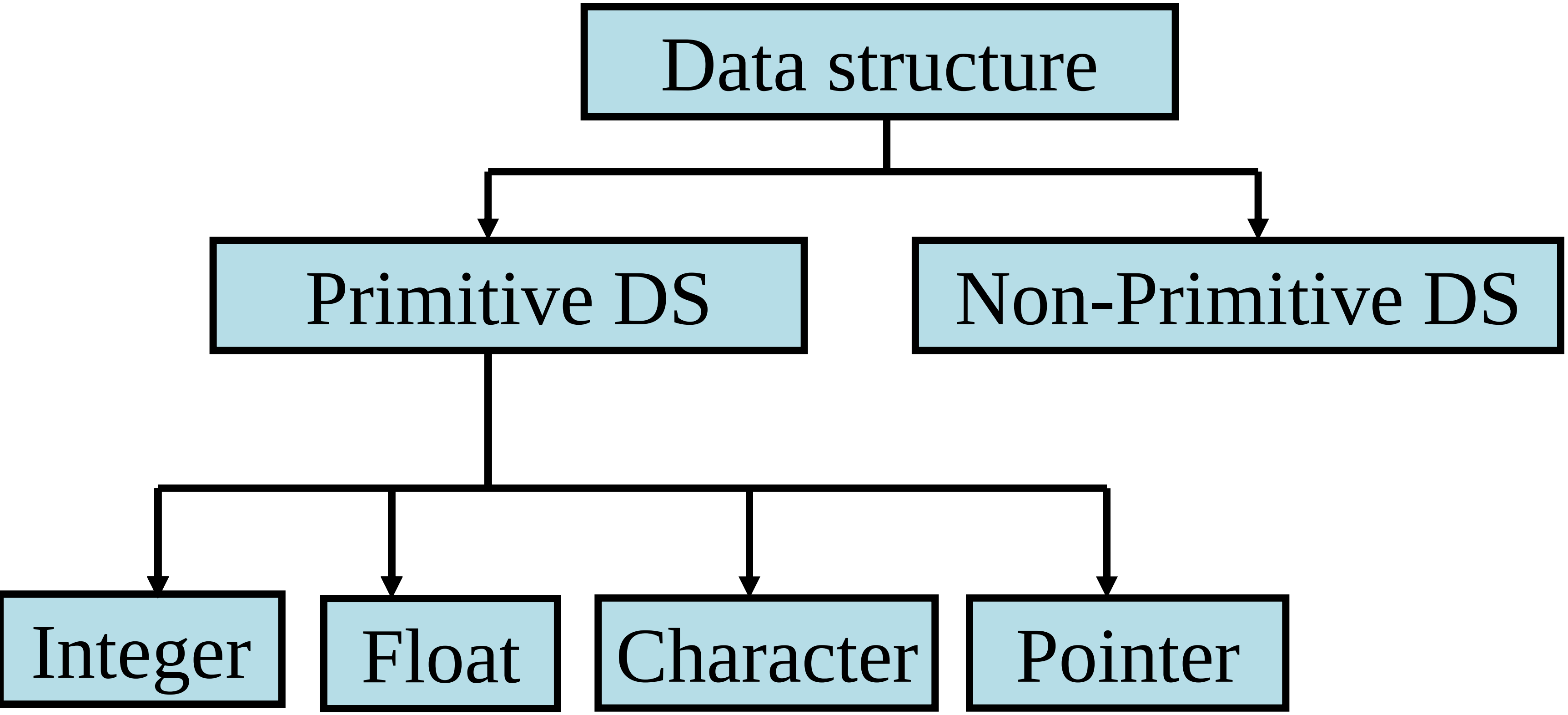


Somaiya TRUST

Primitive data structures

- are the basic DS that directly operate upon the machine instructions.
- can store the value of only one data type.
- example, a char data structure can store only characters.

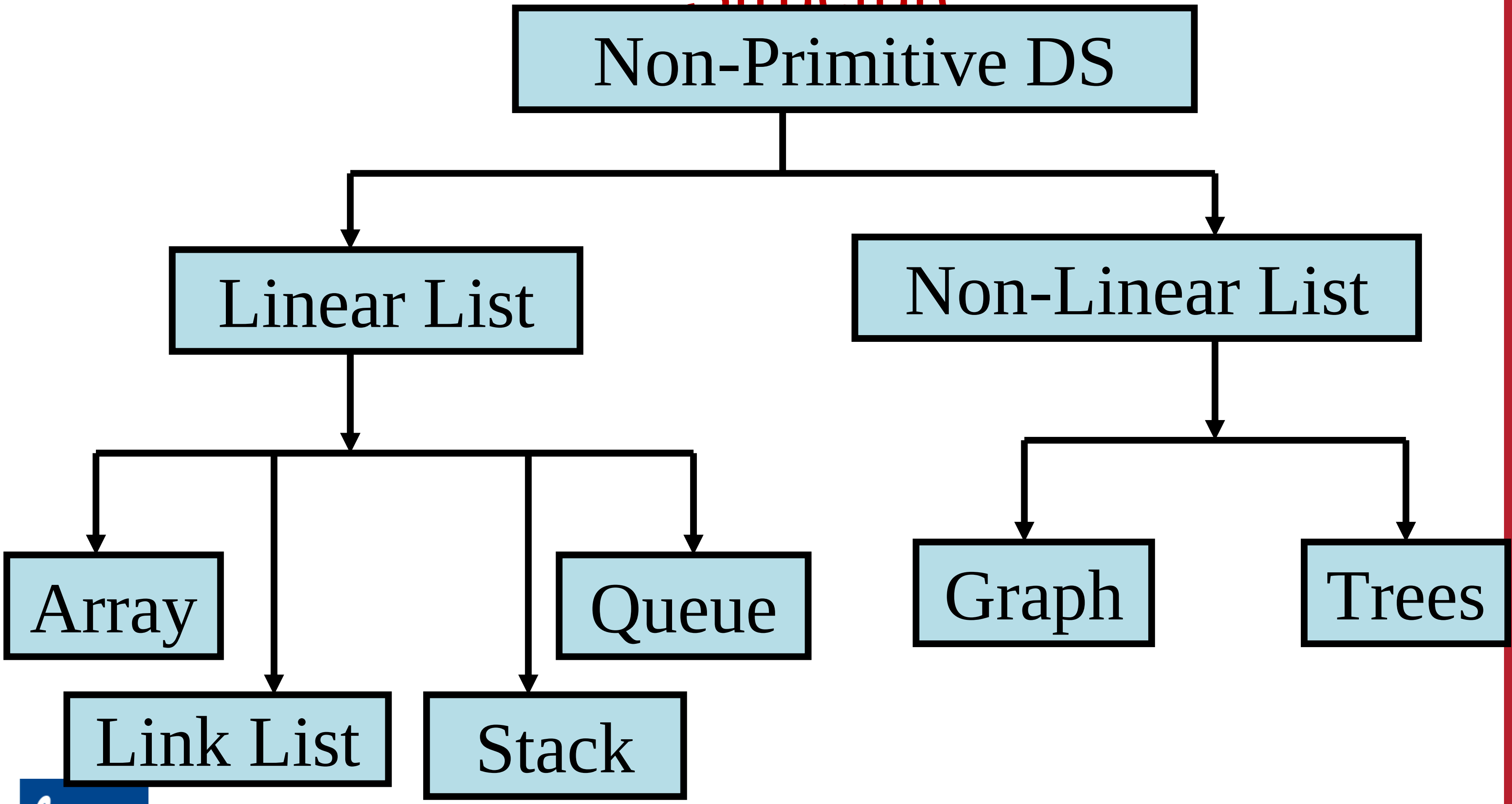
Classification of Data Structure



Non-Primitive data structures

- are more complicated DS
- are derived from primitive DS.
- they emphasize on grouping same or different data items with relationship between each data item.
- example, arrays. Lists and files come under this category

Classification of Data Structure



Linear data structures

- The data structure where data items are organized sequentially or linearily one after another is called Linear data structures.
- Examples : Stack and Queue

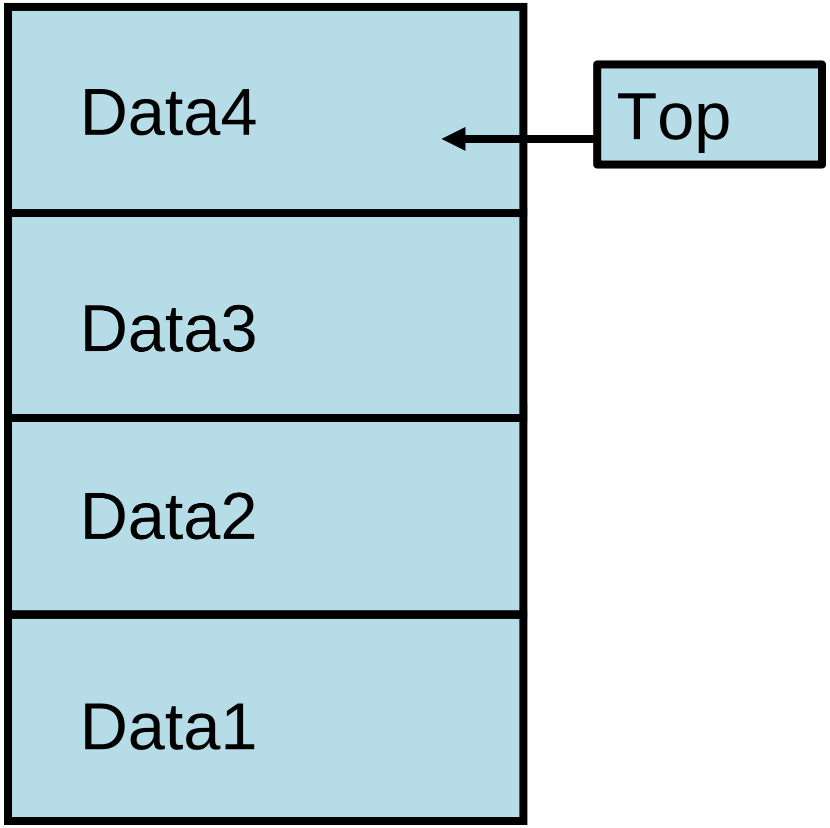
Data structures and their representations

Stack

- Stack is a DS in which addition and deletion of element is allowed at the same end called as TOP of the stack.

- A Stack is LIFO(Last In First Out) DS where element that added last will be retrieved first

Stack

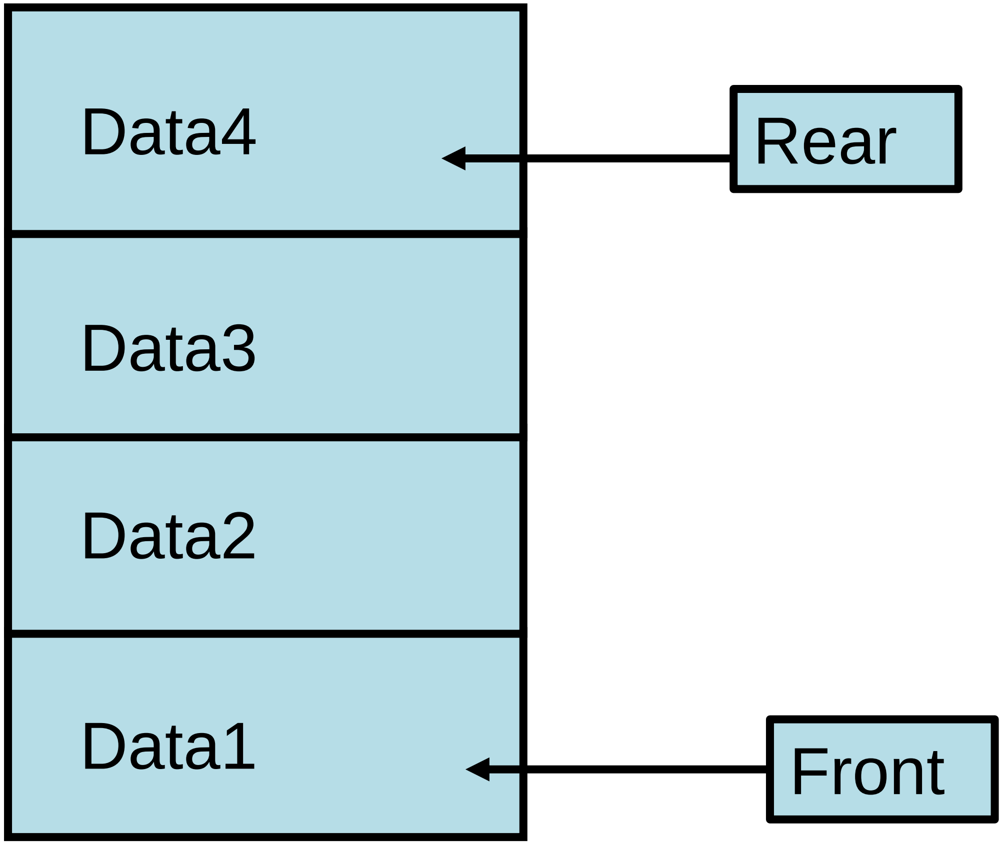


Queue

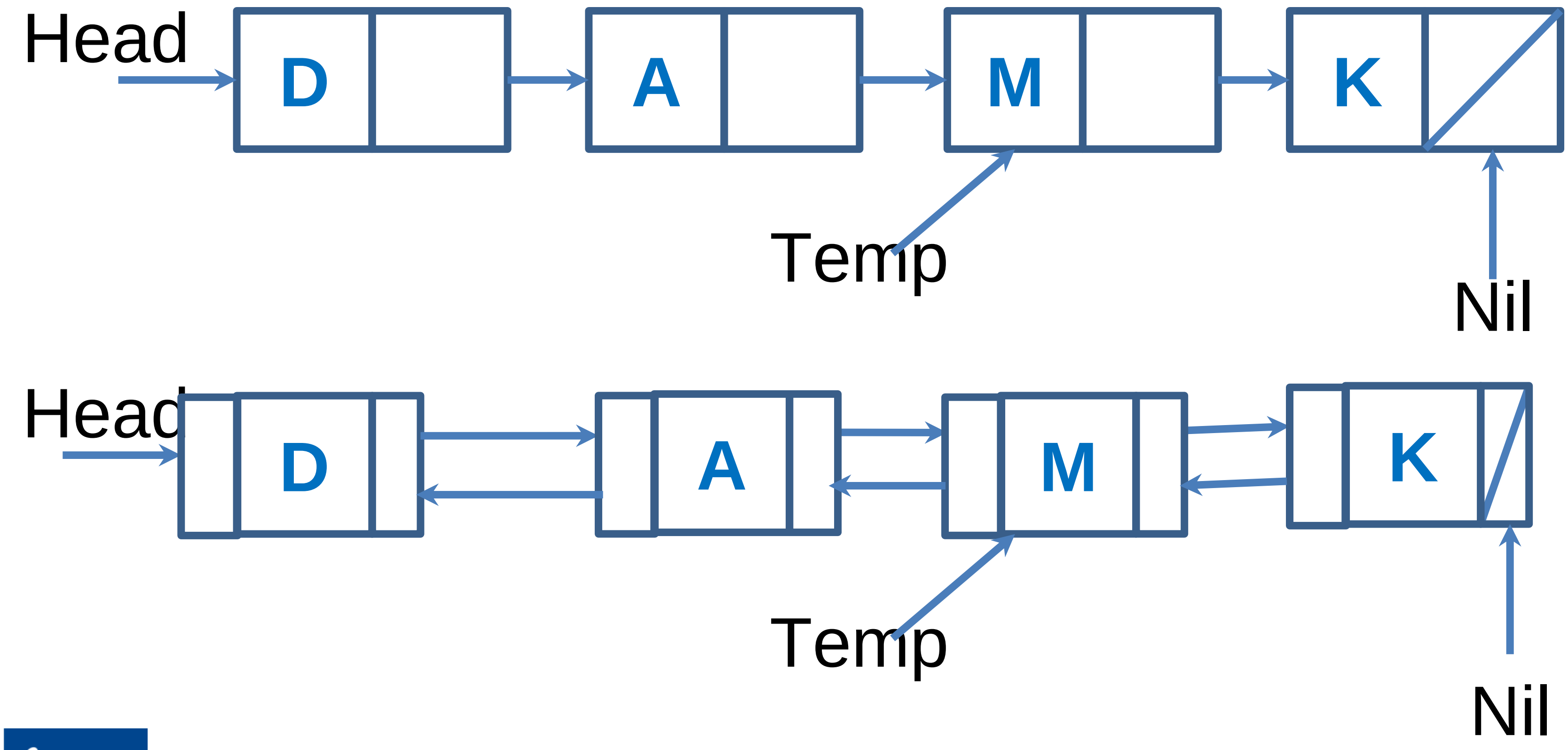
- A Queue is a DS in which addition of element is allowed at the one end called as REAR and deletion is allowed at another end called as FRONT.

- A Queue is FIFO(First In First Out) DS where element that added first will be retrieved first.

Queue



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



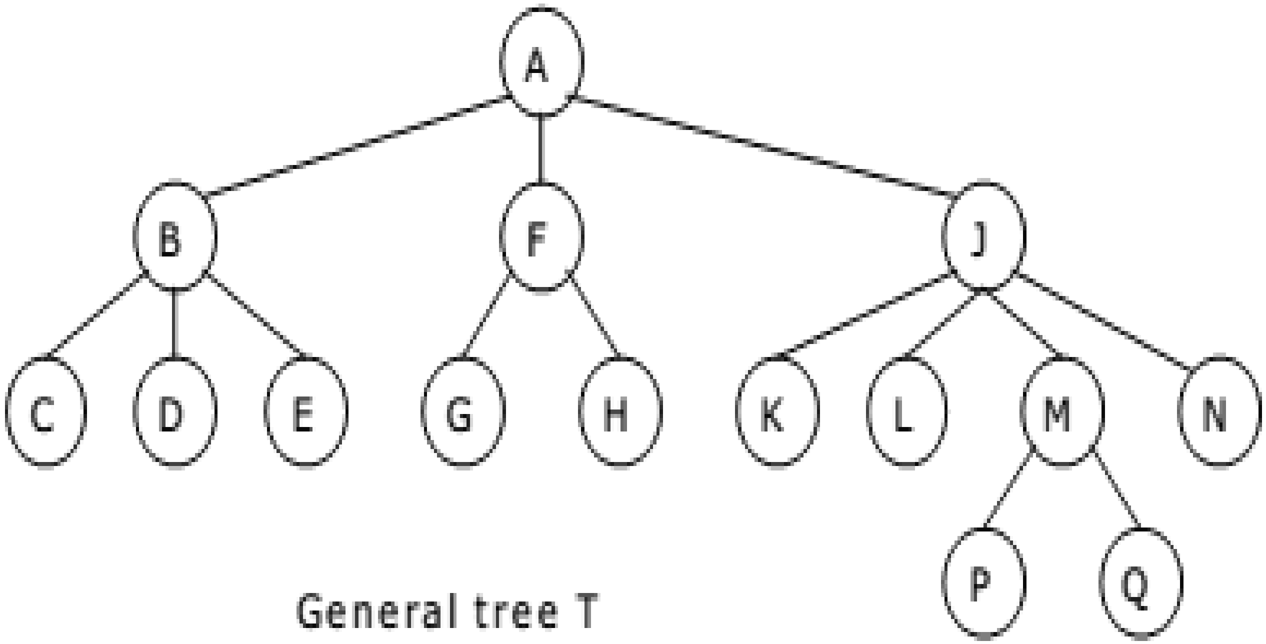
Non Linear data structures

- The data structure in which the data items are not organized sequentially or in linear fashion is called Non Linear data structures.
- Examples : Tree and Graph

Tree

- Tree is collection of nodes where these nodes are arranged hierarchically and form a parent child relationship

Tree



General tree T

Image courtesy: ExamRadar.com

Graph

- A Graph is a collection of a finite number of vertices and edges which connect these vertices.
- Edges represent relationships among vertices that stores data elements.

Binary Tree, Binary search tree and Heaps

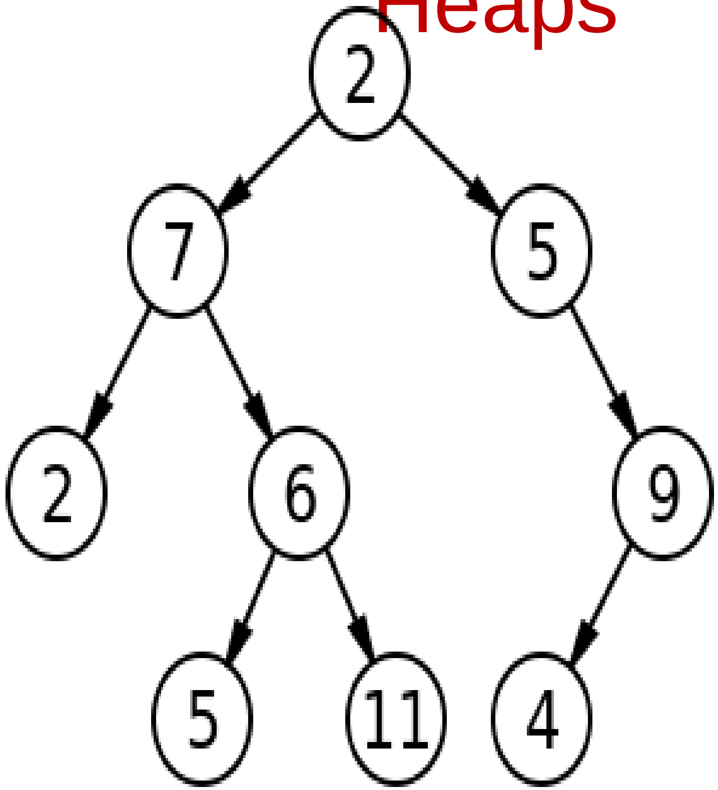


Image courtesy: ExamRadar.com

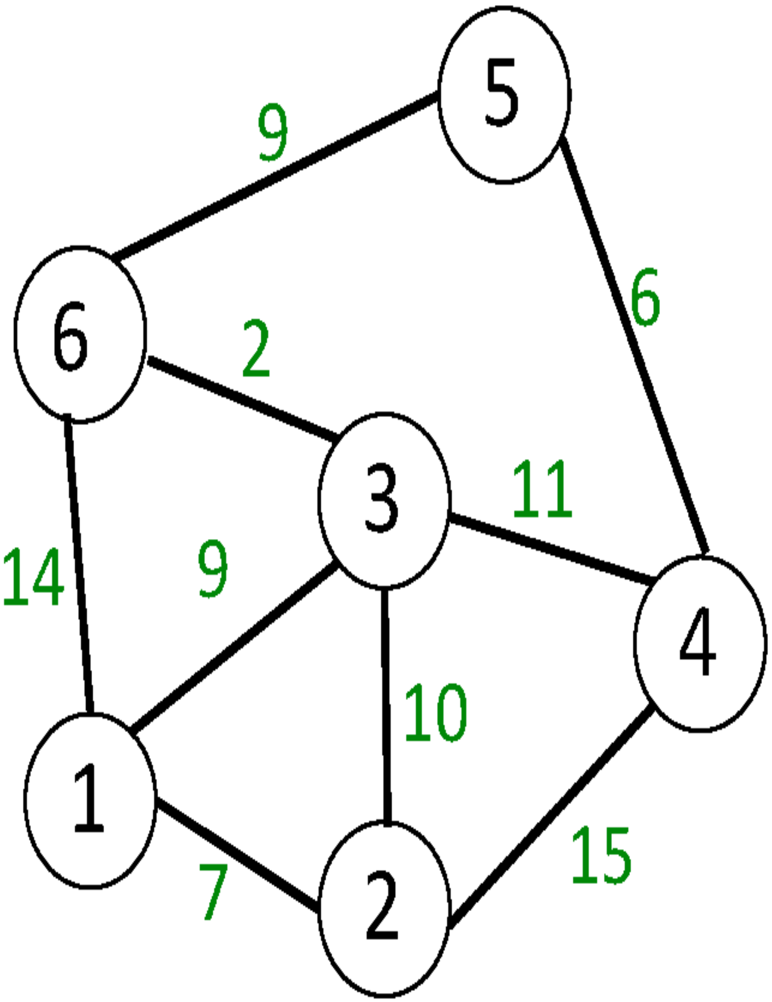


Image courtesy: Medium.com

Difference Linear and Non-linear Data Structures:

S.NO	Linear Data Structure	Non-linear Data Structure
1.	In a linear data structure, data elements are arranged in a linear order where each and every element is attached to its previous and next adjacent.	In a non-linear data structure, data elements are attached in hierarchically manner.
2.	In linear data structure, single level is involved.	Whereas in non-linear data structure, multiple levels are involved.
3.	Its implementation is easy in comparison to non-linear data structure.	While its implementation is complex in comparison to linear data structure.
4.	In linear data structure, data elements can be traversed in a single run only.	While in non-linear data structure, data elements can't be traversed in a single run only.
5.	In a linear data structure, memory is not utilized in an efficient way.	While in a non-linear data structure, memory is utilized in an efficient way.
6.	Its examples are: array, stack, queue, linked list, etc.	While its examples are: trees and graphs.
7.	Applications of linear data structures are mainly in application software development.	Applications of non-linear data structures are in Artificial Intelligence and image processing.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Abstract Data Type and Data Structure

- Definition:-
  - *Abstract Data Types (ADTs)* stores data and allow various operations on the data to access and change it.
  - A mathematical model, together with various operations defined on the model
  - An ADT is a collection of data and associated operations for manipulating that data

Somaiya TRUST

Abstract Data Type

- ADTs support *abstraction*, *encapsulation*, and *information hiding*.
- *Abstraction* is the structuring of a problem into well-defined entities by defining their data and operations.
- The principle of hiding the used data structure and to only provide a well-defined interface is known as *encapsulation.*

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

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:

Abstract Data Structure

- Logical Definition
- Mathematical definition

- ADTs represent concepts
- Free from hardware or software dependency
- Operation name is assumed as the return variable name

Abstraction

- The process of isolating implementation details and extracting only essential property from an entity
- Hence, abstractions in a program:
  - Data abstraction :What operations are needed by the data
  - Functional abstraction : What is the purpose of a function (algorithm)

Program = data + algorithms

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

ADTs

- Abstract Data Type (ADT):
  - End result of data abstraction
  - A collection of data together with a set of operations on that data
  - ADT = Data + Operations
- ADT is a language independent concept
  - Different language supports ADT in different ways
  - In C++, the class construct is the best match

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

IS ADT a function?

Important Properties of ADT

- Specification: The supported operations of the ADT
- Implementation: Data structures and actual coding to meet the specification

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

ADT : Specification and Implementation

- Specification and implementation are disjointed:
  - One specification
  - One or more implementations
    - Using different data structure
    - Using different algorithm
- Users of ADT:
  - Aware of the specification only
    - Usage only base on the specified operations
  - Do not care / need not know about the actual implementation
    - i.e. Different implementation do not affect the user

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

Example ADT : String

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

Example ADT : String

- Value Definition

Abstract Typedef StringType<<Chars>>

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

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)

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.

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)

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];

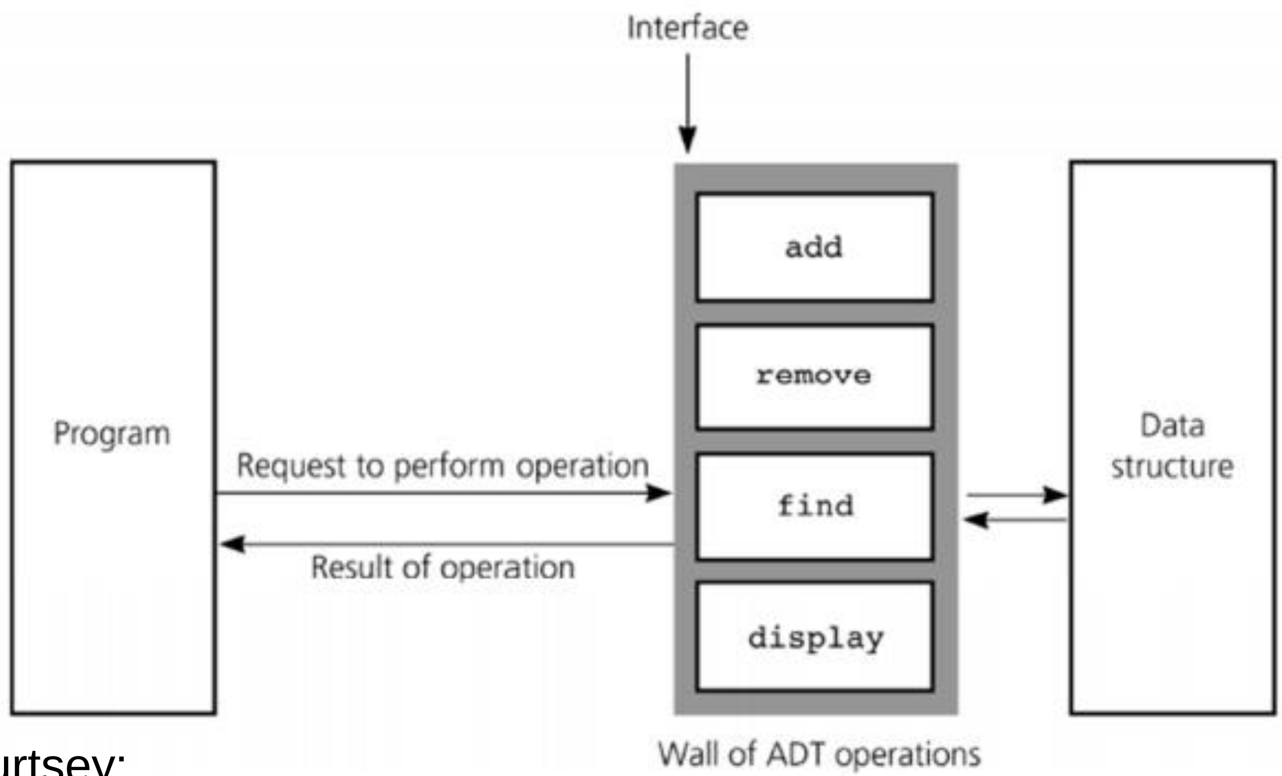
Abstract Data Types: Advantages

- 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

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

A wall of ADT operations

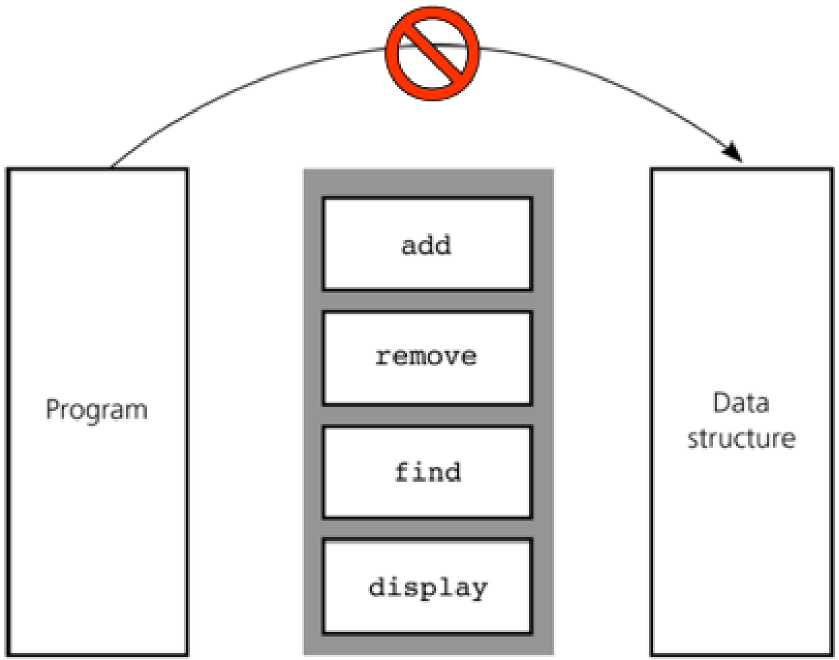
- ADT operations provides:
  - Interface to data structure
  - Secure access



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



Wall of ADT operations

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

ADT Implementation

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

- Abstract TypeDef StackType
- Condition:

Thank you