



SOMAIYA
VIDYAVIHAR UNIVERSITY

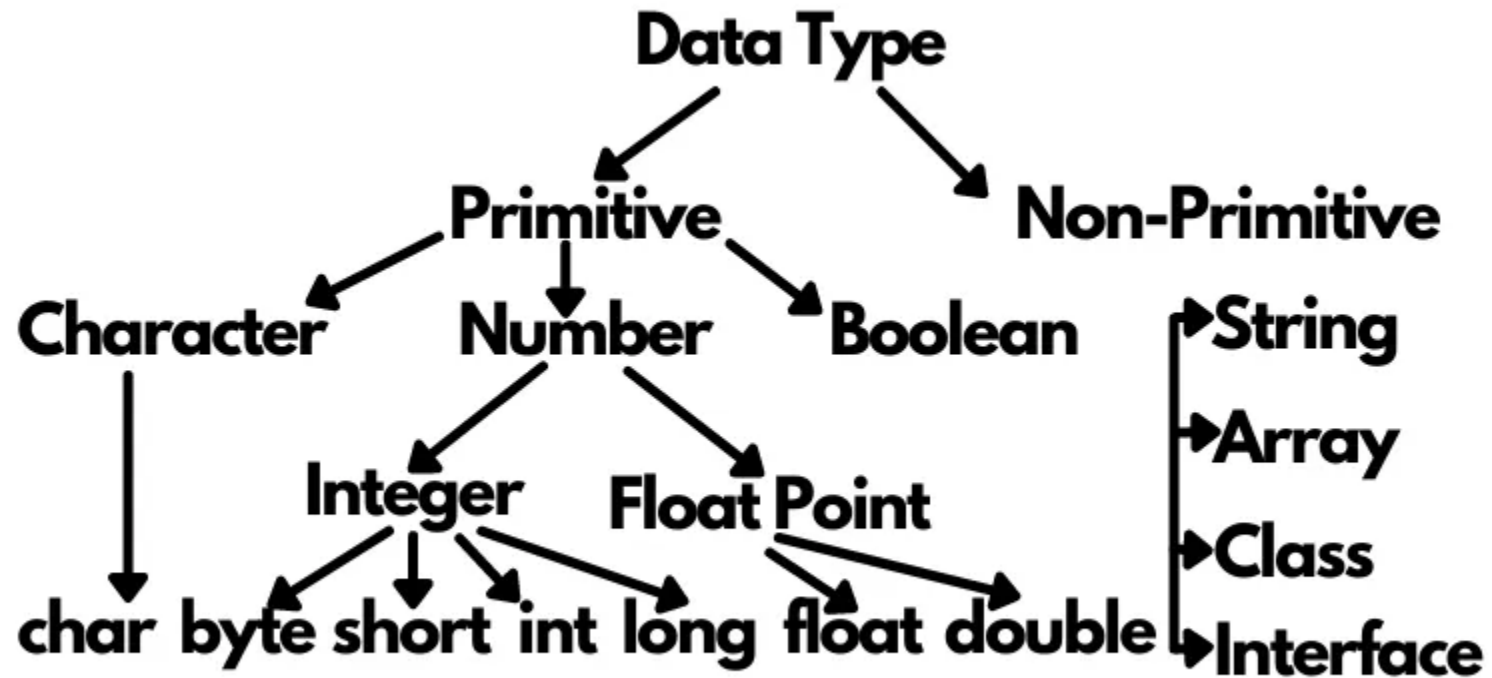
K J Somaiya College of Engineering

DATA STRUCTURES – TYPES AND ADT

sushmakadge@somaiya.edu

swatimali@somaiya.edu

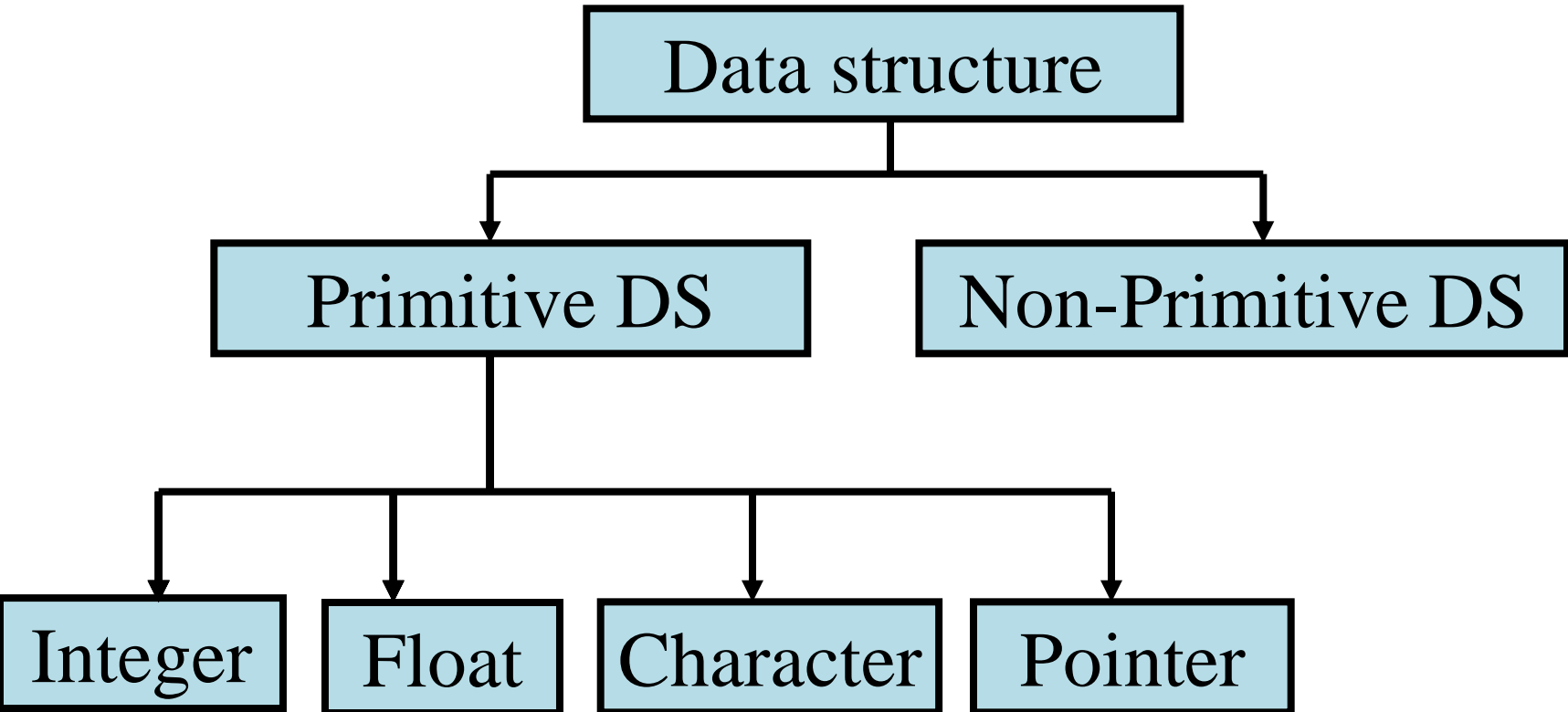
Classification of Data



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.

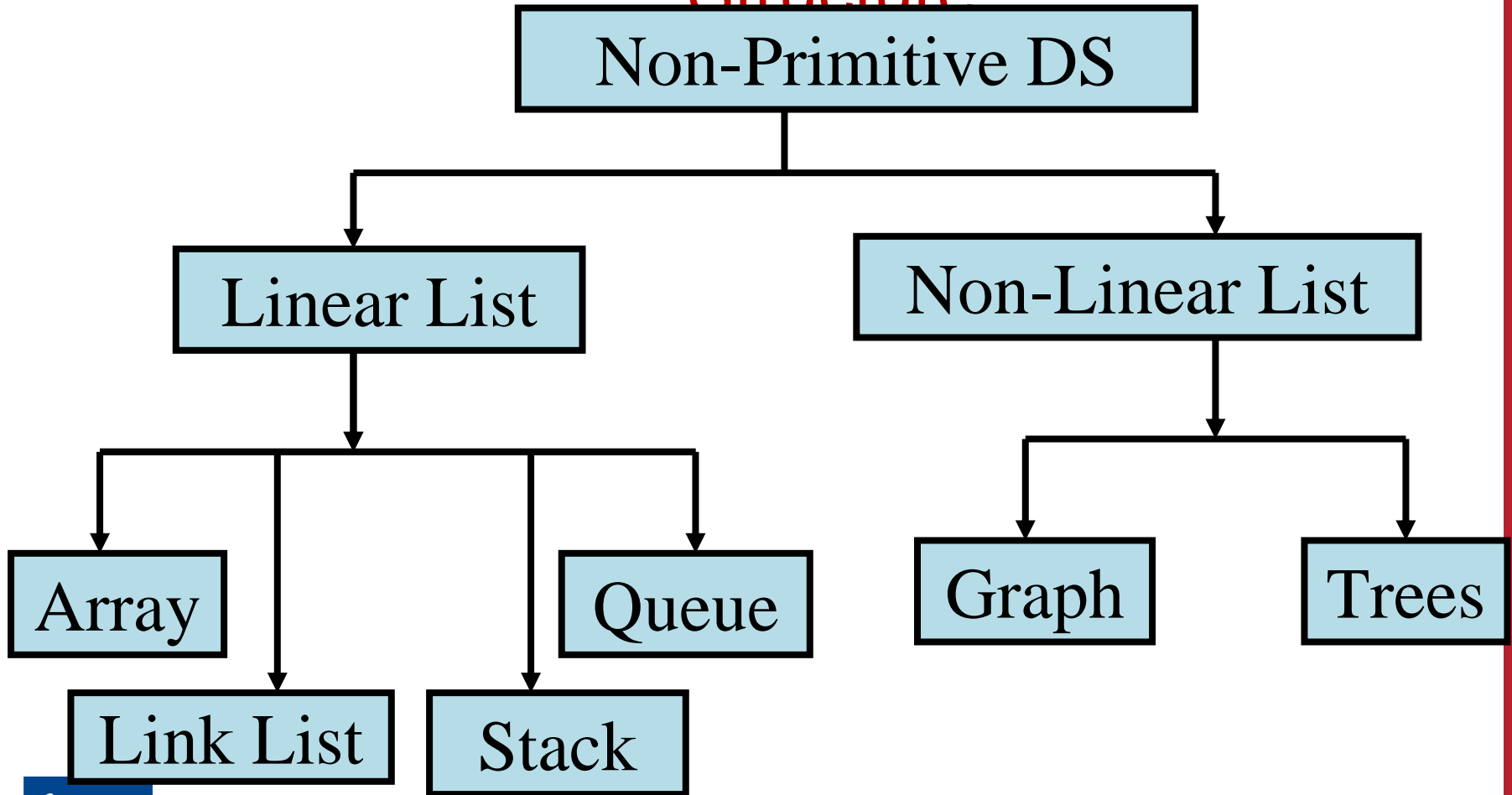
Classification of Data Structure



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 linearly one after another is called **Linear data structures**.
- **Examples : Stack and Queue**



SOMAIYA

VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering

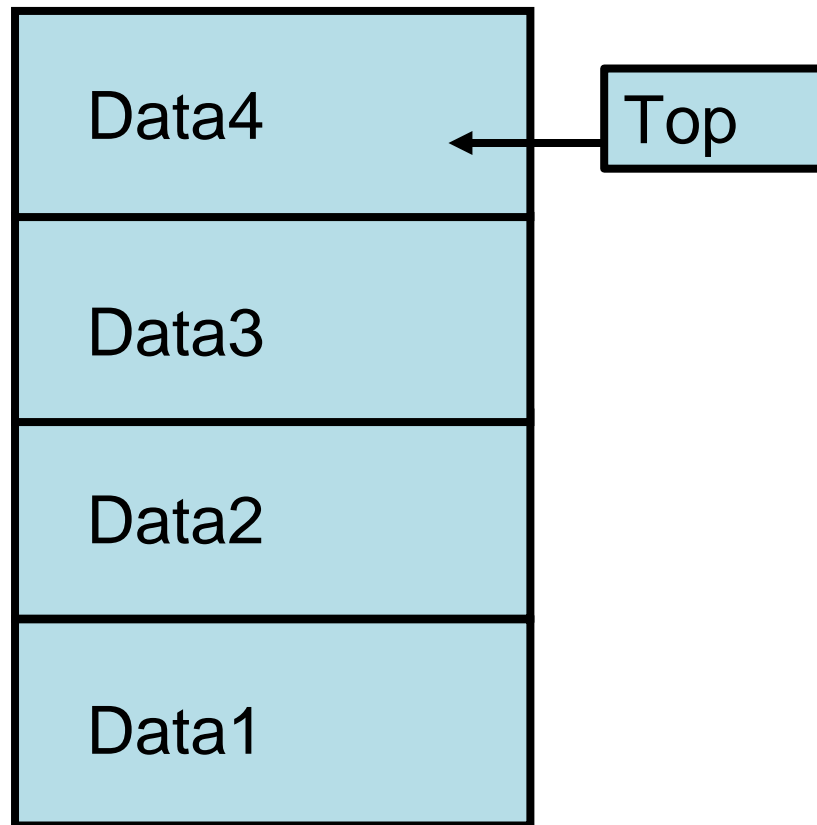
Data structures and their representations

Stack

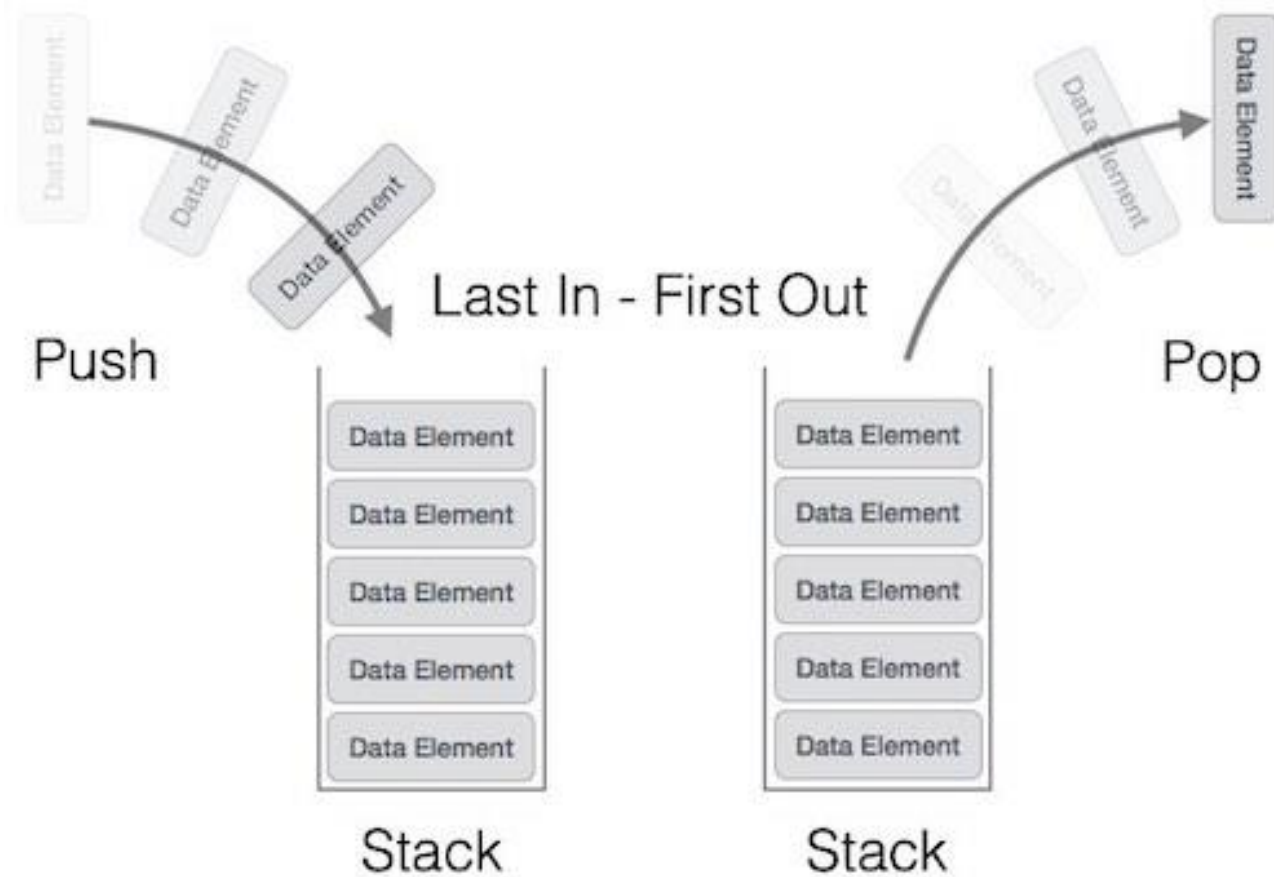
- Stack is a DS in which addition and deletion of element is allowed at the same end called as **TOP** of the stack.
- A Stack is **LIFO**(Last In First Out) DS where element that added last will be retrieved first.



Stack



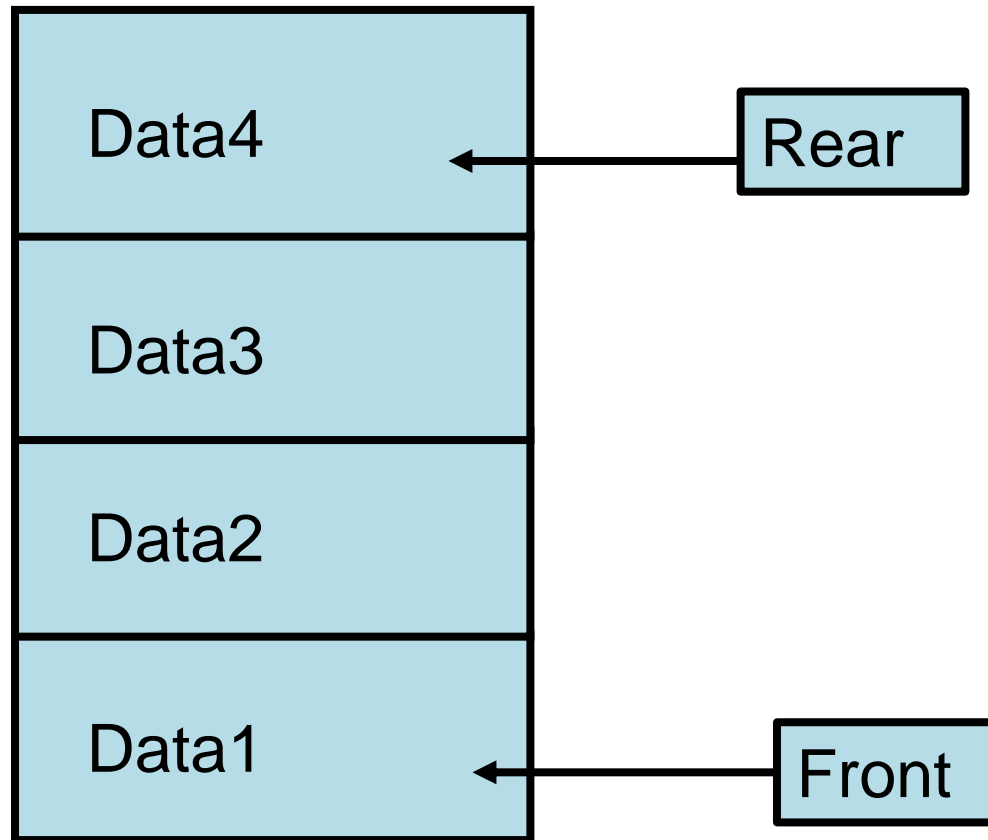
Stack



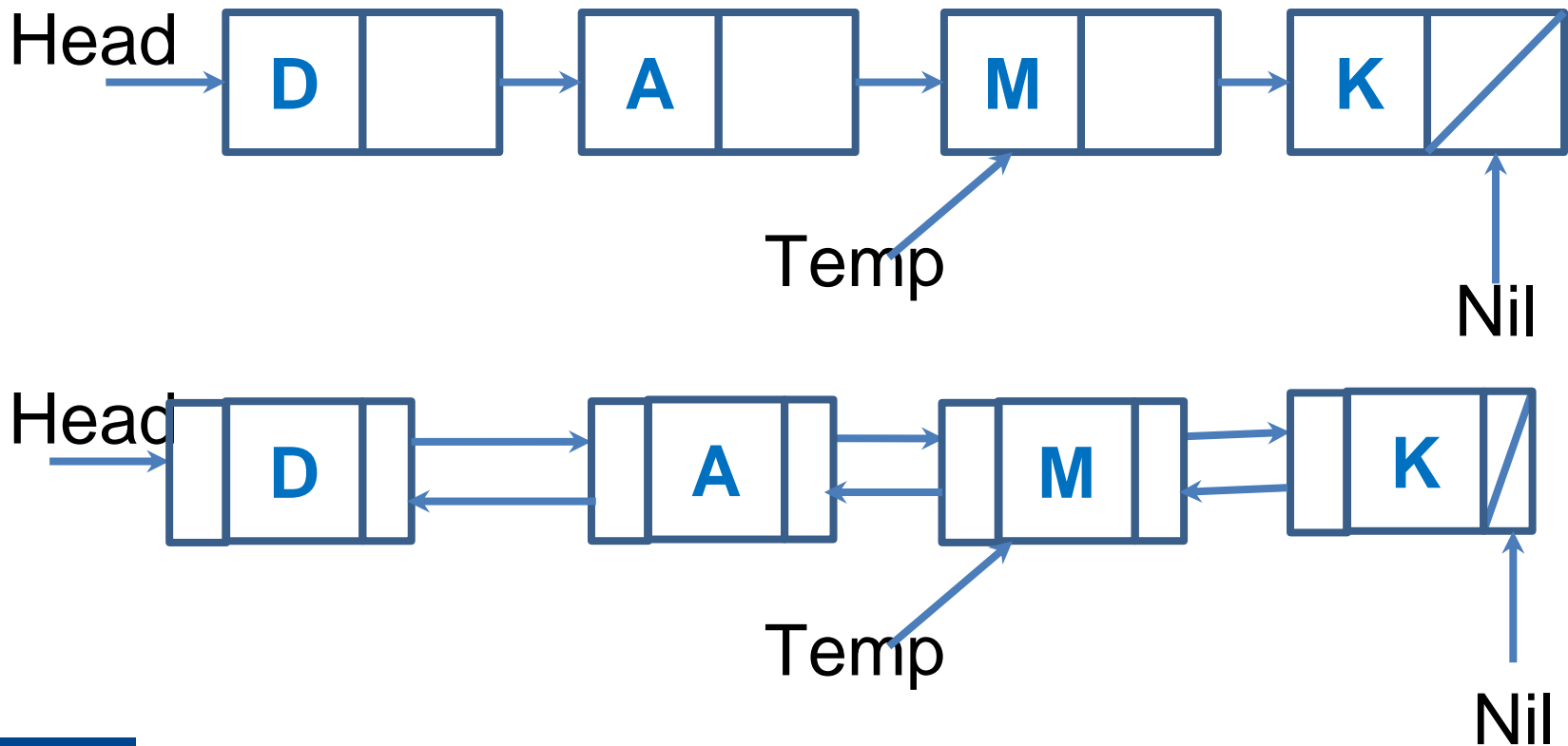
Queue

- 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



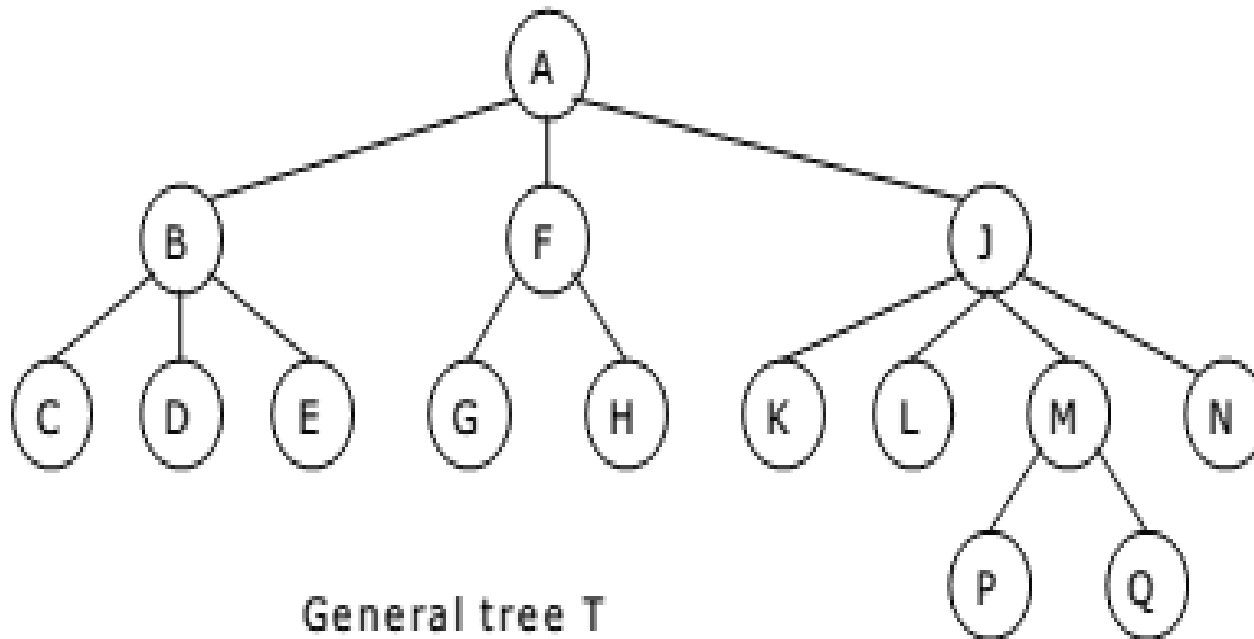
Non Linear data structures

- 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

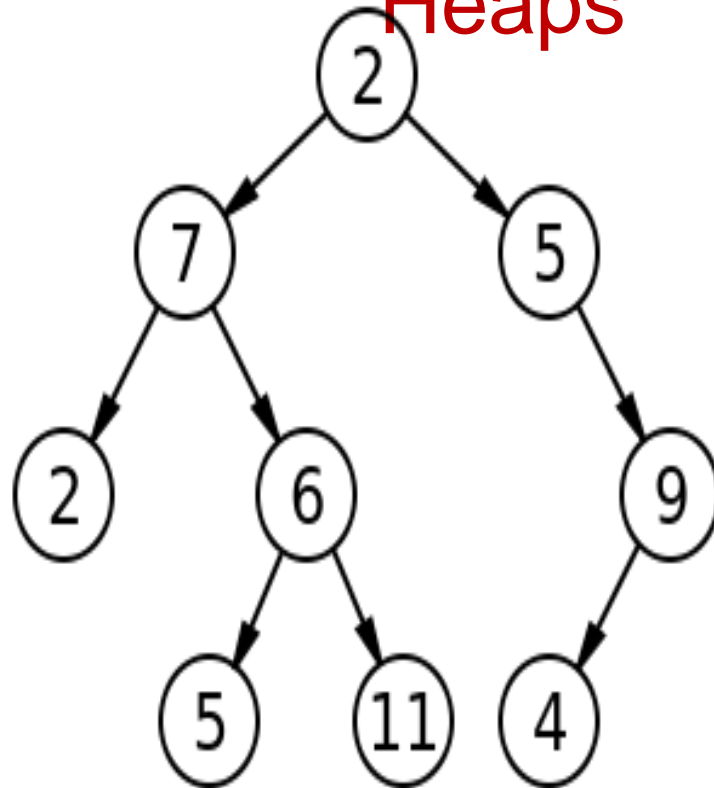
Tree



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



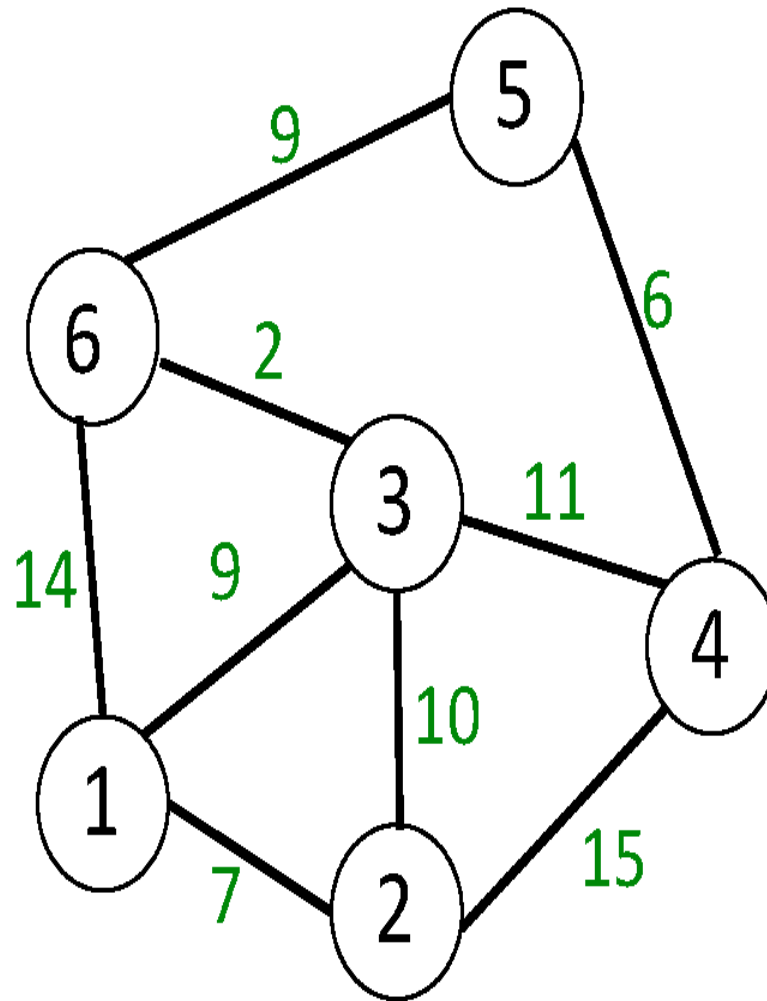


Image courtesy: Medium.com

Difference Linear and Non-linear Data Structures:

S.NO 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.
2. In linear data structure, single level is involved.
3. Its implementation is easy in comparison to non-linear data structure.
4. In linear data structure, data elements can be traversed in a single run only.
5. In a linear data structure, memory is not utilized in an efficient way.
6. Its examples are: array, stack, queue, linked list, etc.
7. Applications of linear data structures are mainly in application software development.

Non-linear Data Structure

- In a non-linear data structure, data elements are attached in hierarchically manner.
- Whereas in non-linear data structure, multiple levels are involved.
- While its implementation is complex in comparison to linear data structure.
- While in non-linear data structure, data elements can't be traversed in a single run only.
- While in a non-linear data structure, memory is utilized in an efficient way.
- While its examples are: trees and graphs.
- Applications of non-linear data structures are in Artificial Intelligence and image processing.

Abstract Data Type and Data Structure

- An abstract data type (ADT) is the way we look at a data structure, focusing on **what it does and ignoring how it does its job**.
- For example, stacks and queues are perfect examples of an ADT.
- We can implement both these ADTs using an array or a linked list. This demonstrates the 'abstract' nature of stacks and queues.

Abstract Data Type and Data Structure

- To further understand the meaning of an abstract data type, we will break the term into 'data type' and 'abstract', and then discuss their meanings.
- Data type of a variable is the **set of values that the variable can take.**
- The basic data types in C include int, char, float, and double.

Abstract Data Type and Data Structure

- When we talk about a primitive type (built-in data type), we actually consider two things: a data item with **certain characteristics** and the permissible **operations on that data**.
- For ex, an int variable can contain any whole-number value from **-32768 to 32767** and can be operated with the operators **+, -, *, and /**.
- In other words, the operations that can be performed on a data type are an **inseparable part** of its identity.
- Therefore, when we declare a variable of an abstract data type (e.g., stack or a queue), we also need to specify the operations that can be performed on it.

Abstract Data Type and Data Structure

- ❖ In C, an abstract data type can be a structure considered without regard to its implementation.
- ❖ It can be thought of as a 'description' of the data in the structure with a list of operations that can be performed on the data within that structure.
- ❖ The end-user is not concerned about the details of how the methods carry out their tasks.
- ❖ They are only aware of the methods that are available to them and are only concerned about calling those methods and getting the results.
- ❖ They are not concerned about how they work

Abstract Data Type and Data Structure

- For ex, when we use a stack or a queue, the user is concerned only with the **type of data** and the **operations** that can be performed on it.
- Therefore, the fundamentals of how the data is stored should be invisible to the user.
- They should not be concerned with how the methods work or what structures are being used to store the data.
- They should just **know that to work with stacks**, they have push() and pop() functions available to them. Using these functions, they can manipulate the data (insertion or deletion) stored in the stack.

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

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 \langle *ParameterType* *Parameter1*,
ParameterType *Parameter2*....., *ParameterType*
ParameterN \rangle 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

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

Important Properties of ADT

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

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

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 $n \geq 0$)

Example ADT : String Operator Definition

1. **abstract Integer** StringLength (StringType
String)

Precondition: None (A string may contain n
characters where $n \geq 0$)

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()
 - MultiplyRational()
 - 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: $b \neq 0$;
 postcondition :
 makerational [0] =a;
 makerational [1] =b;

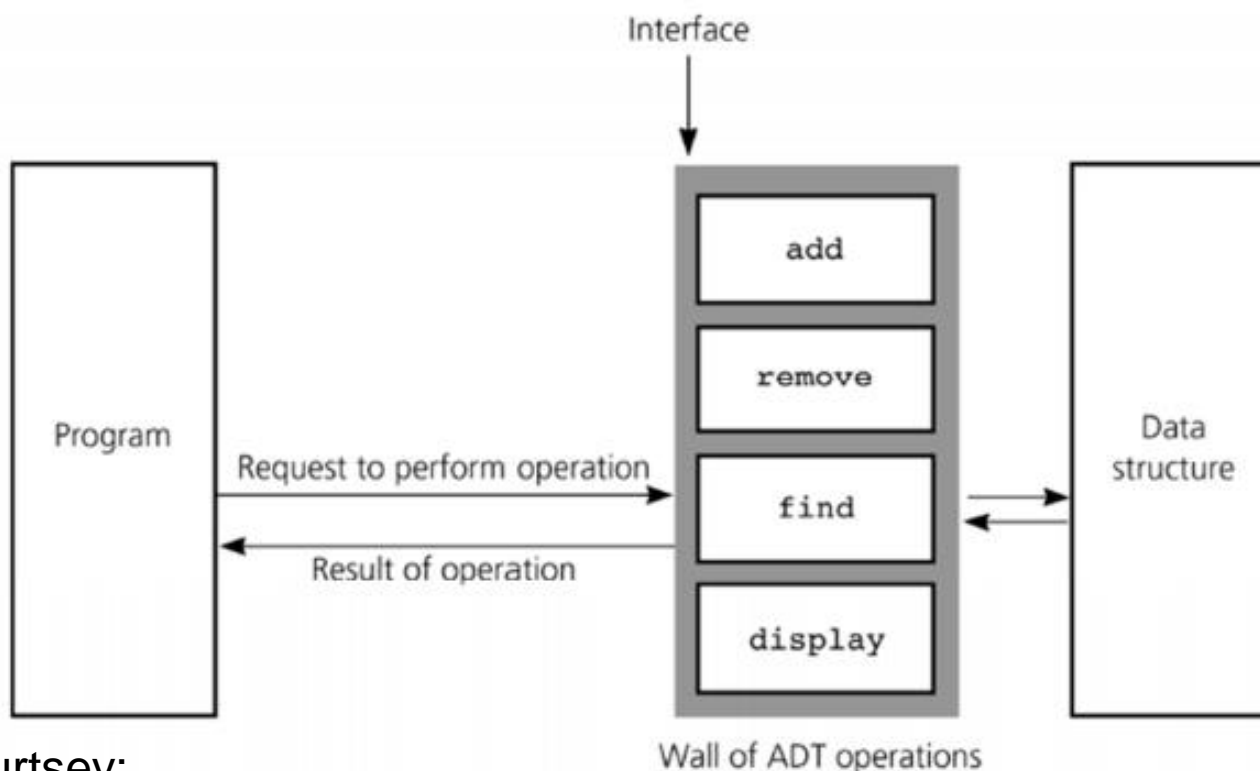
- abstract
 RATIONALtype
 add<a,b>
 RATIONALType a,b;
 Precondition: none
 postcondition :
 add[0] =
 $a[0]*b[1]+b[0]*a[1]$
 add[1] = $a[1] * b[1]$

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

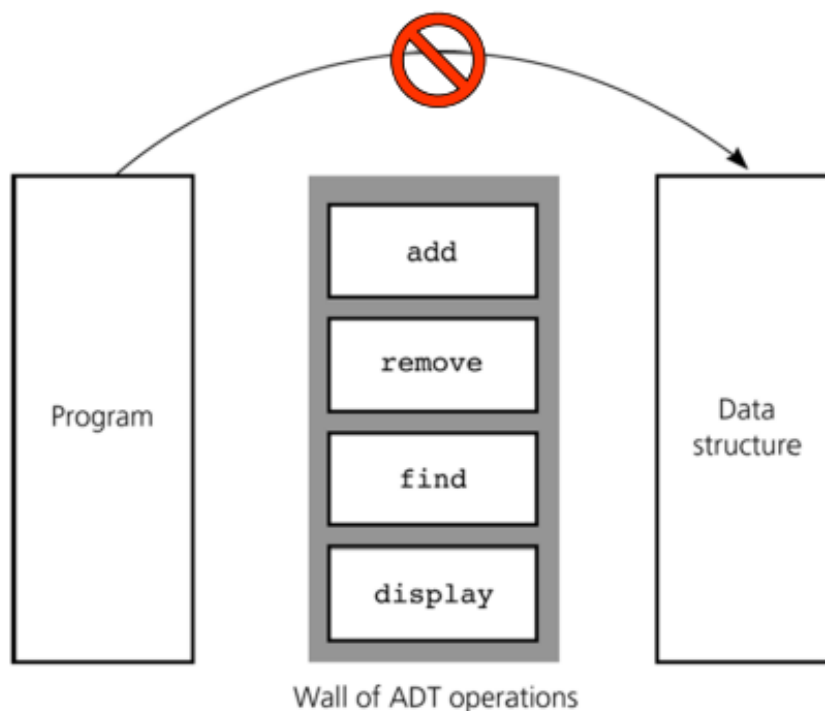
A wall of ADT operations

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



Violating the Abstraction

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



ADT Implementation

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



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya College of Engineering

Thank you

