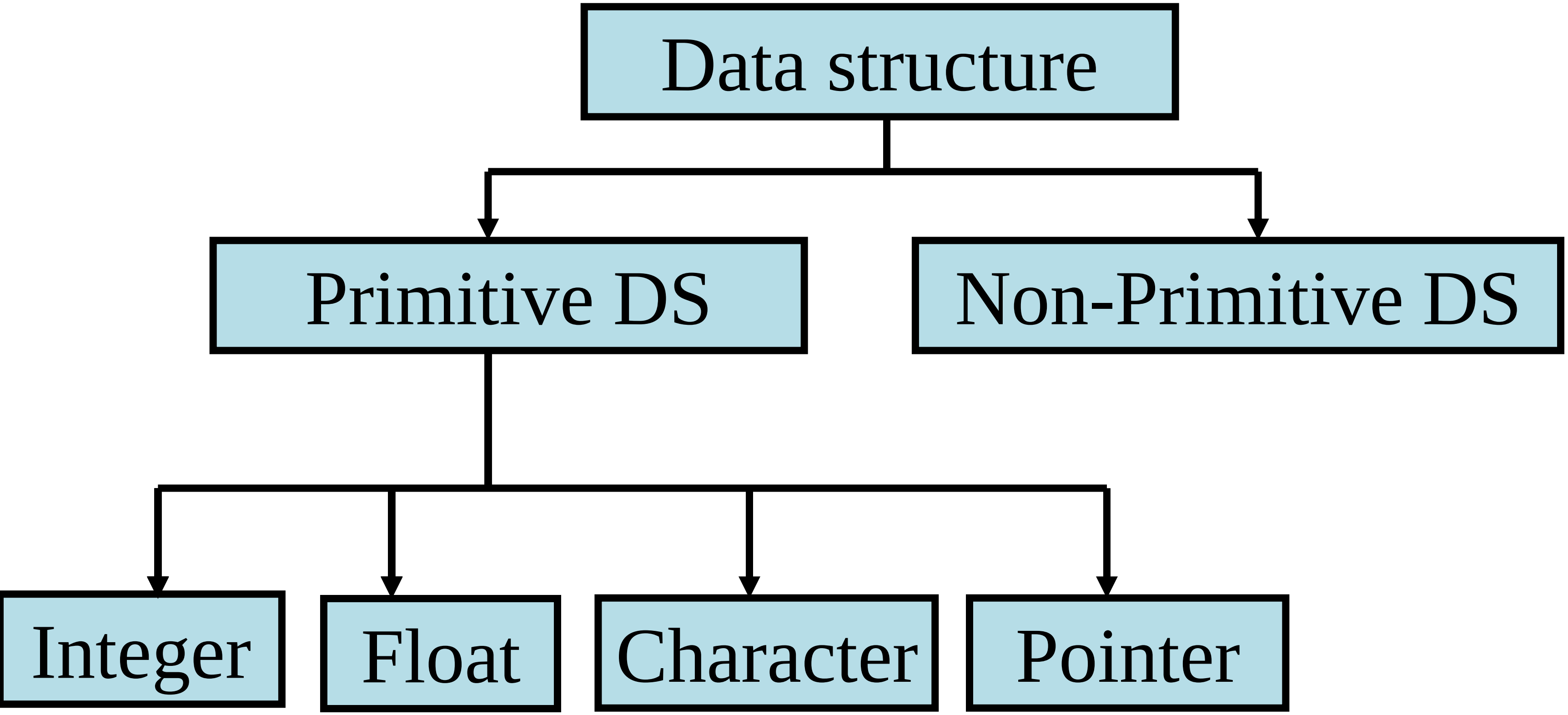
DATA STRUCTURES – TYPES AND ADT

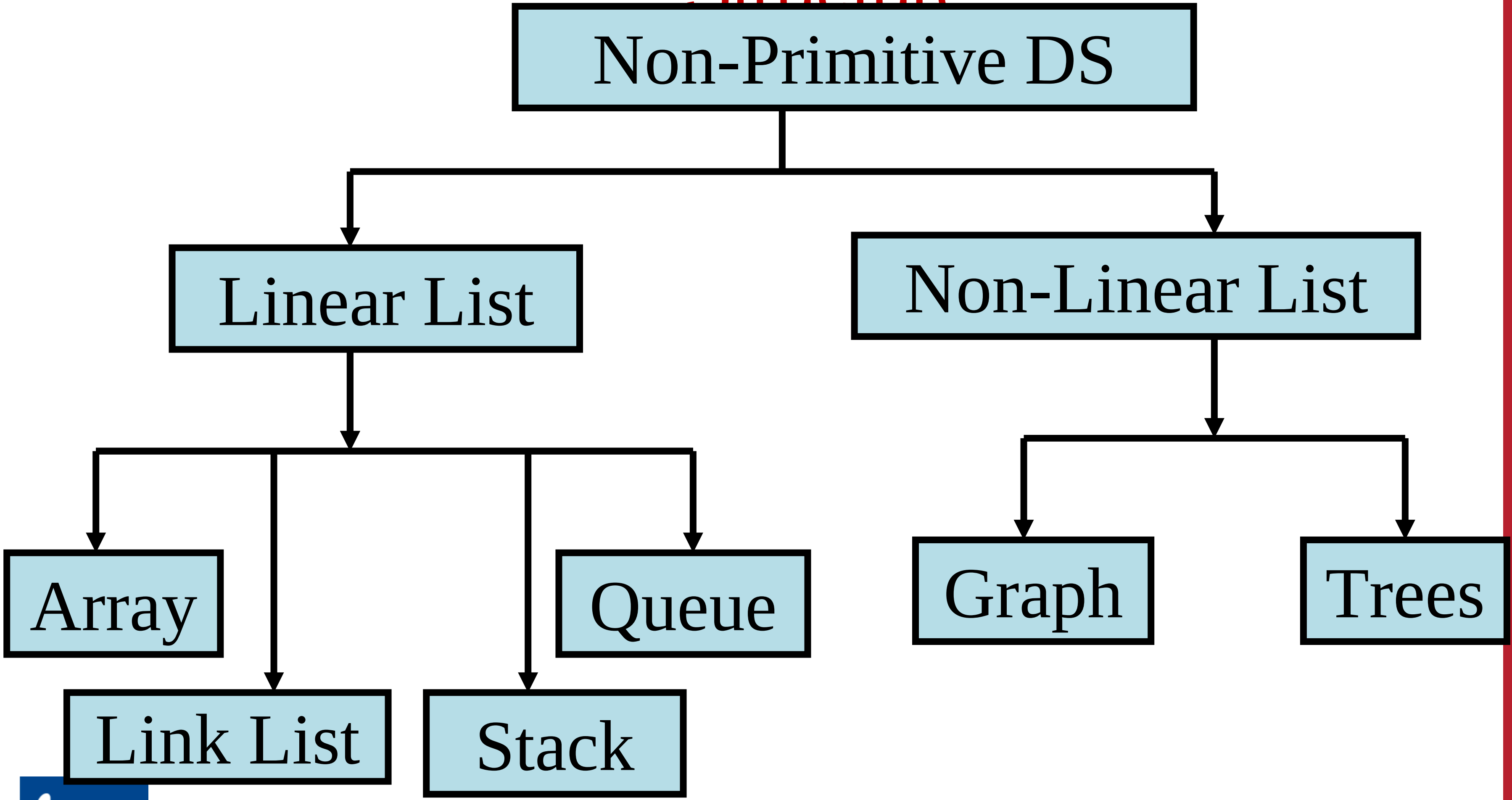
Classification of Data Structure

- Primitive Data Structure
- Non-Primitive Data Structure

Classification of Data Structure



Classification of Data Structure



Non-Primitive DS

Linear List

Non-Linear List

Array

Link List

Stack

Queue

Graph

Trees

Primitive data structures

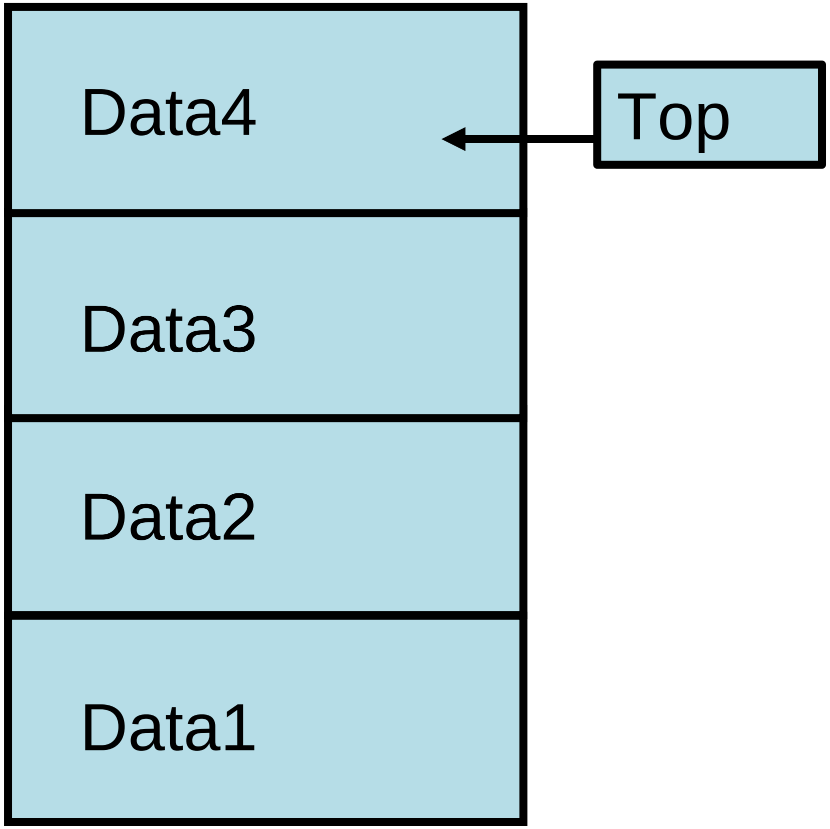
- Basic structures that are directly operated upon by the machine instructions.
- Usually built into the language, such as an integer, a float.

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.

Data structures and their representations

Stack



Data4

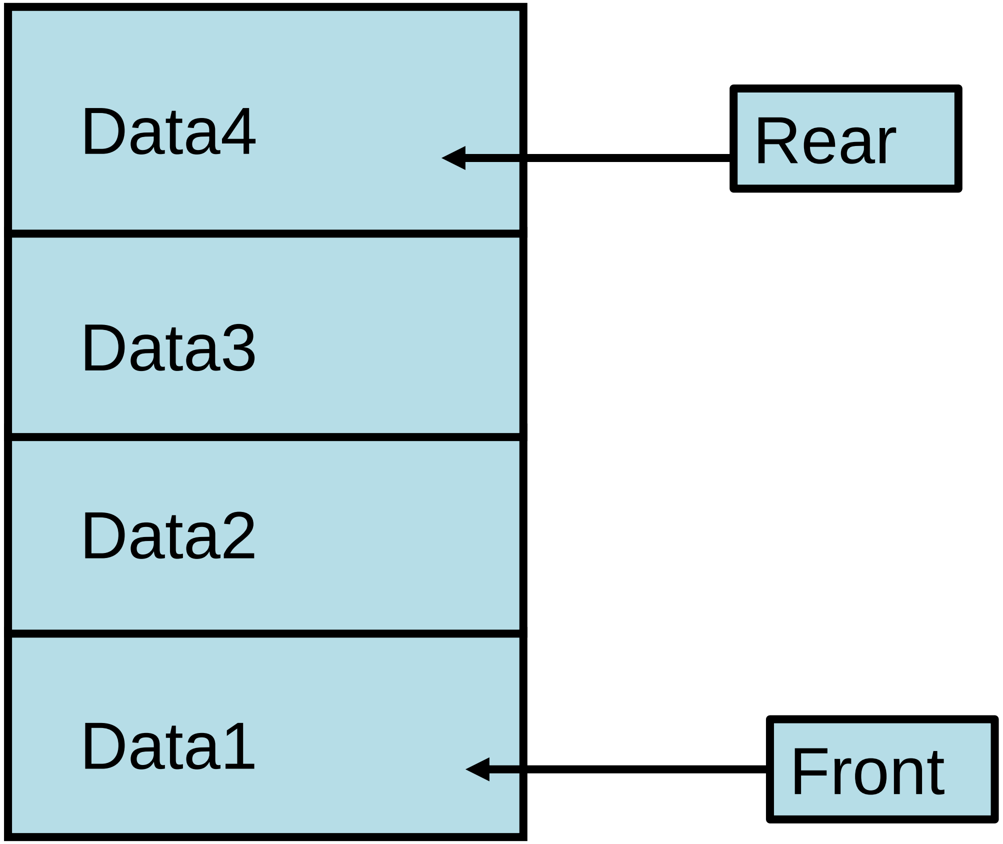
Top

Data3

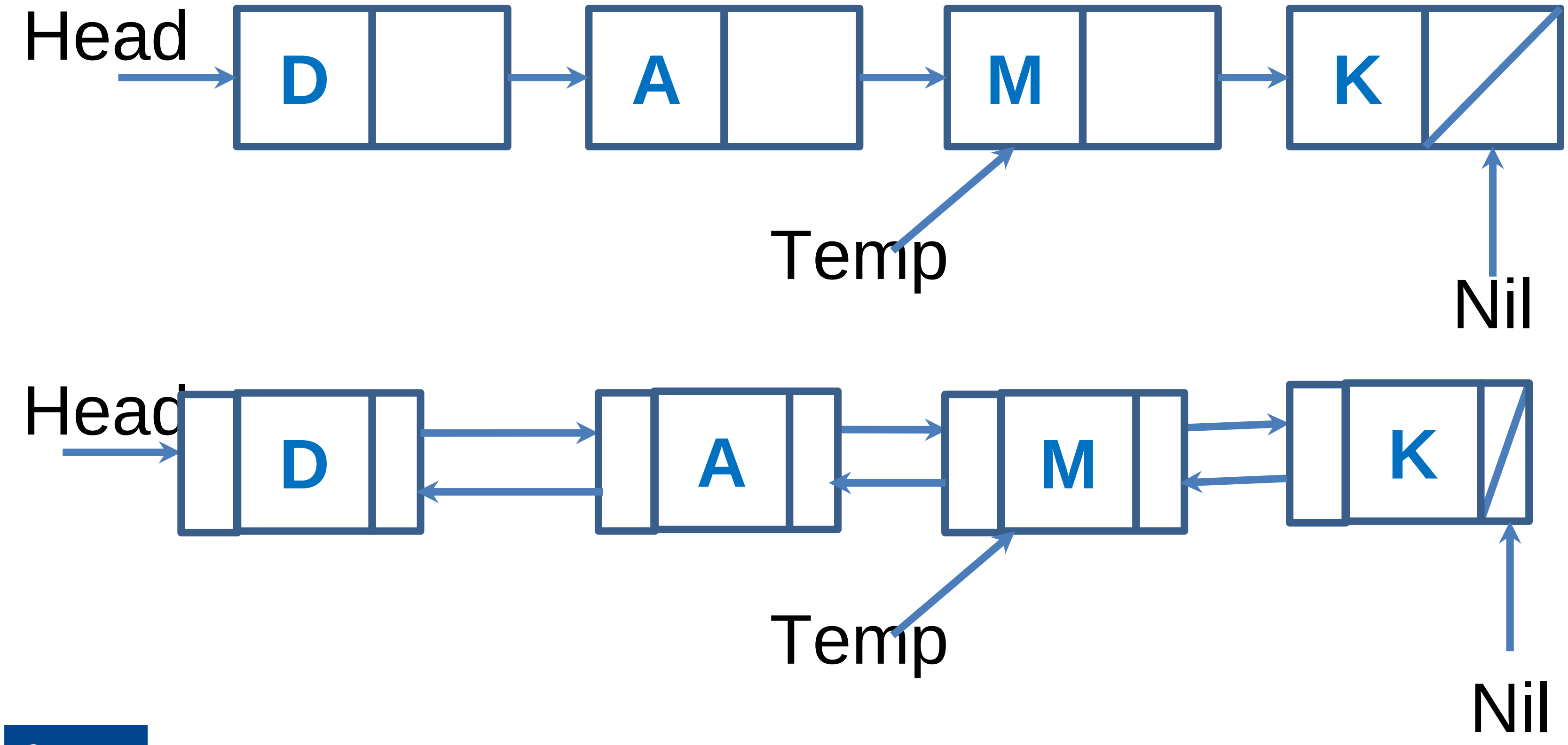
Data2

Data1

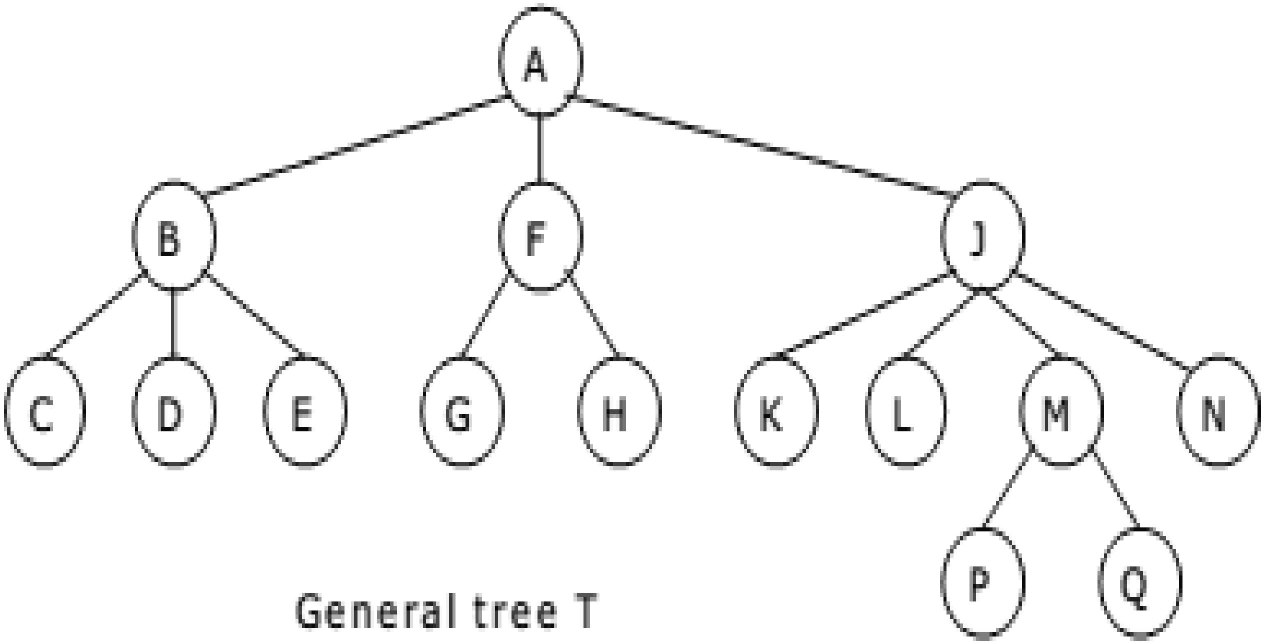
Queue



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

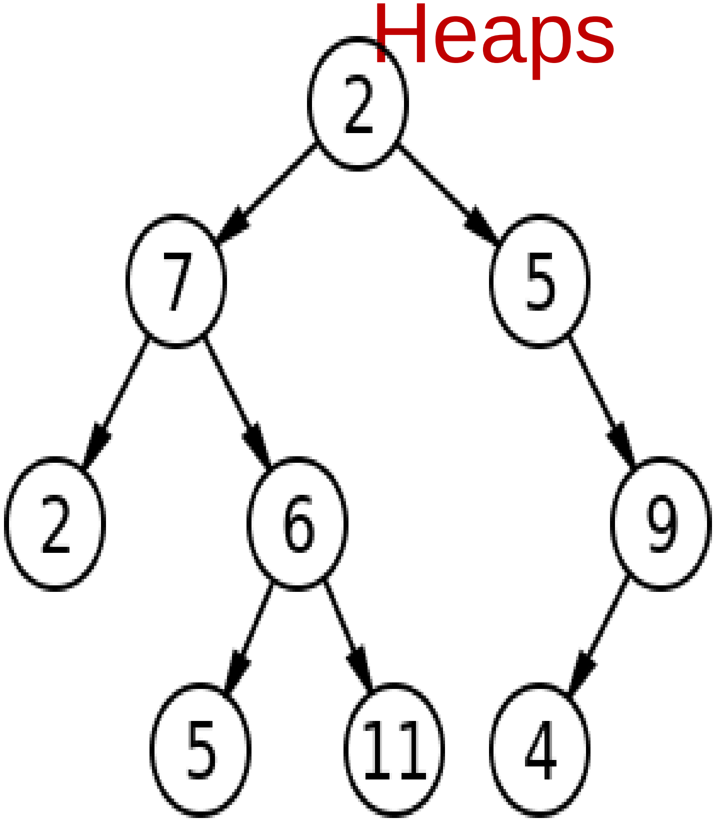


Image courtesy: ExamRadar.com

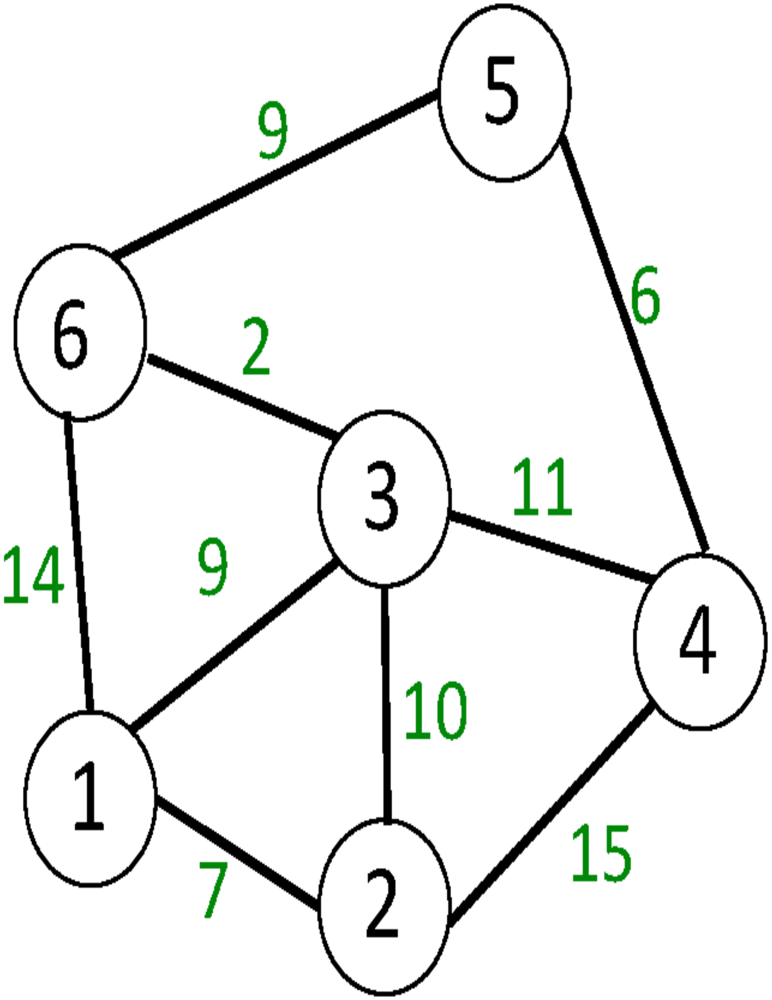


Image courtesy: Medium.com

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

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

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

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

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

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;

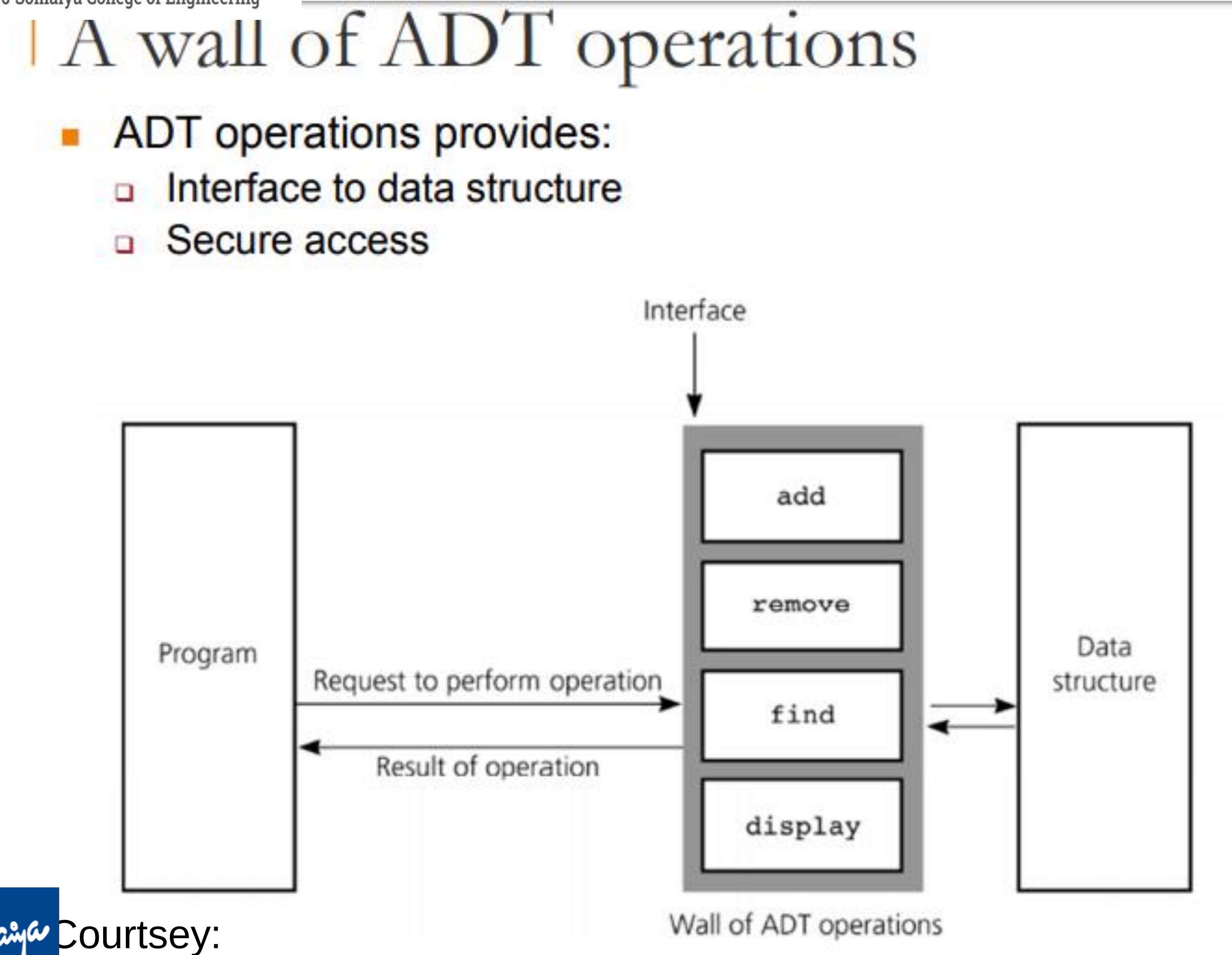
Precondition: none

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
  - So 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



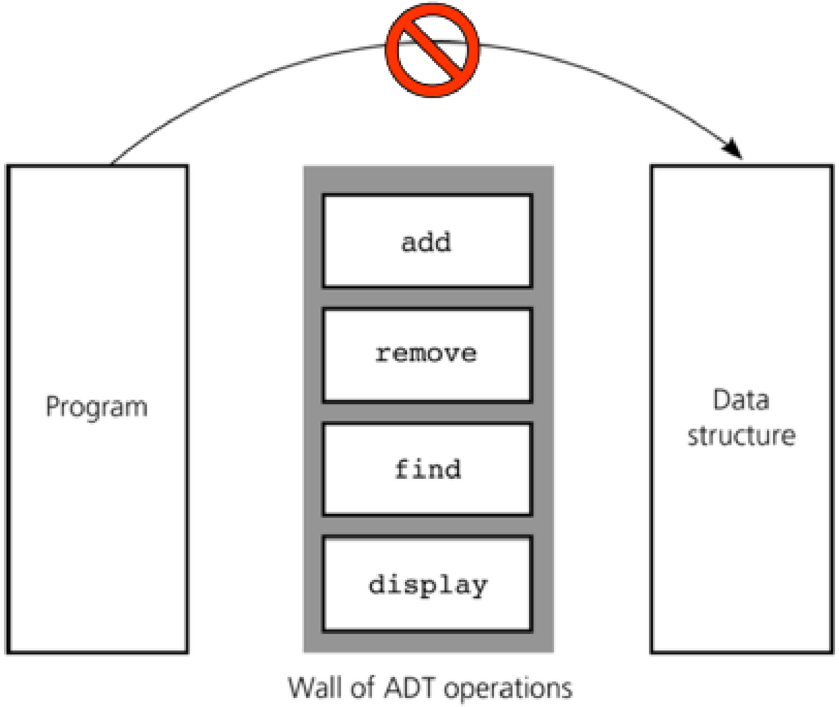
A wall of ADT operations

- 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



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