DATA STRUCTURES – TYPES AND ADT

SWATIMALI@SOMAIYA.EDU

Data structures Implementation

Variables, arrays and Pointers

- Variable
- Pointer variable
- Memory allocation
- Array allocations
- Dynamic memory allocation

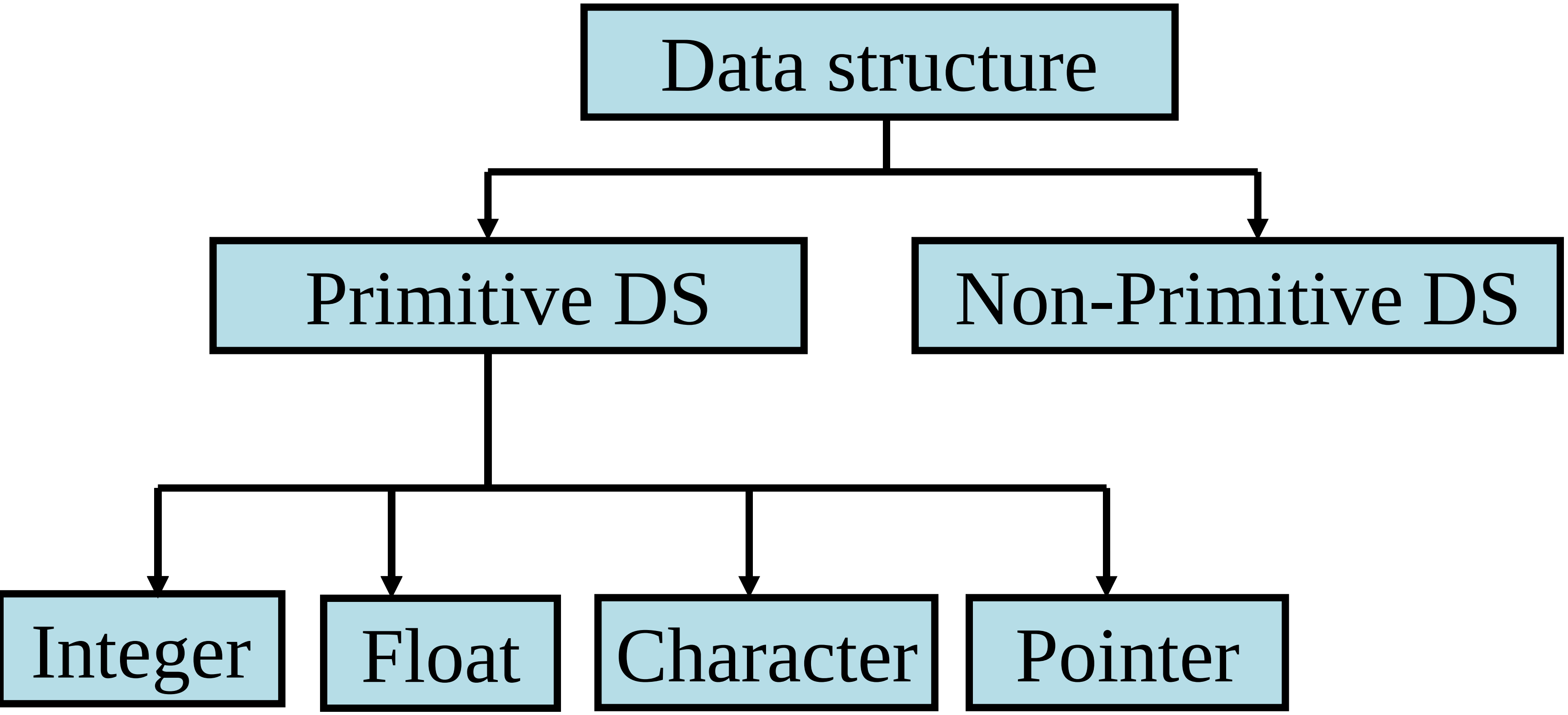
Memory

- Memory – Main memory, Secondary memory
- How do you differentiate : RAM, main memory, Primary memory, secondary memory, HDD???
- Main memory and program execution
- Can a user have access to entire main memory space?
- Can a program be larger than main memory?

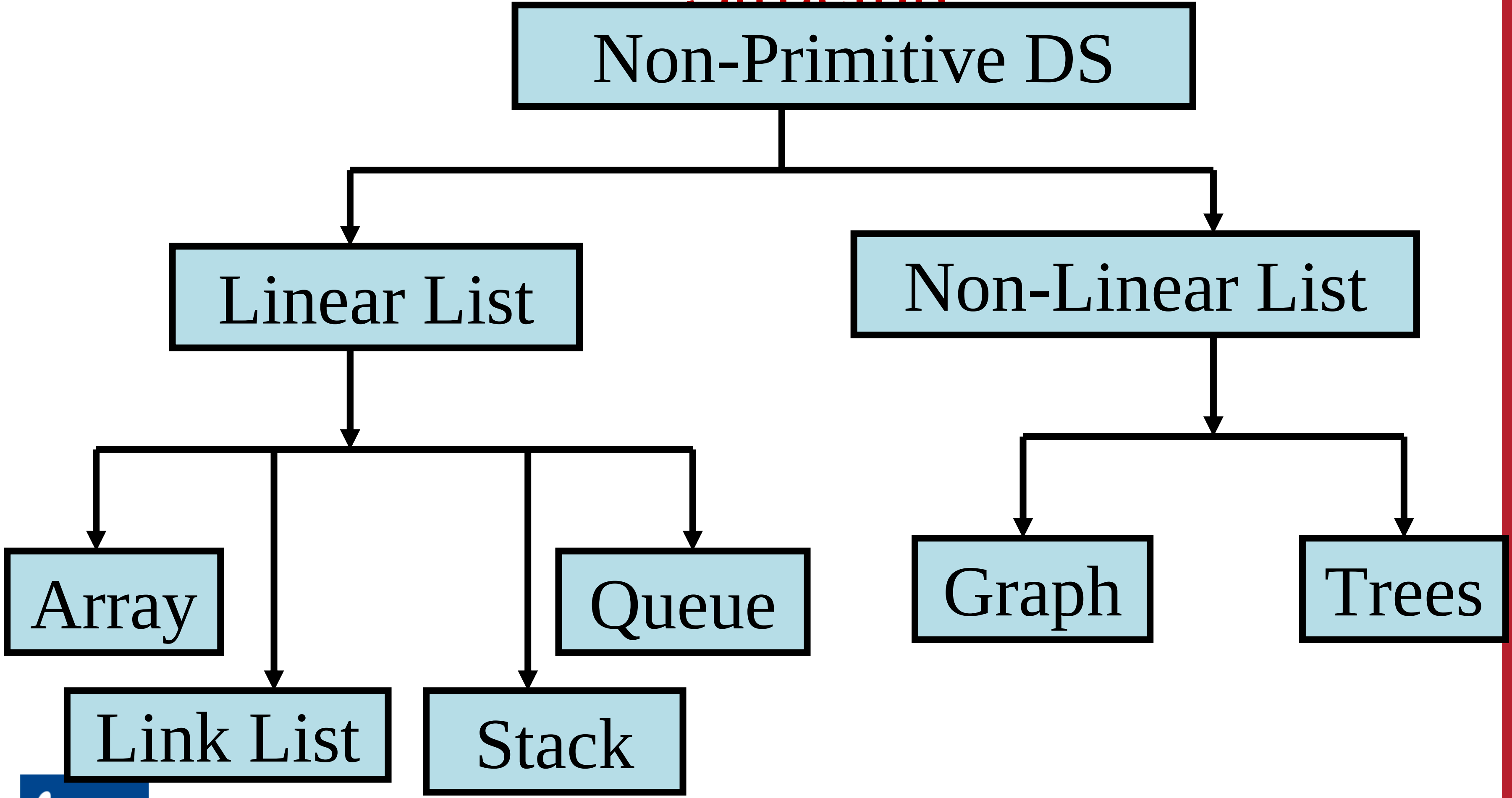
Classification of Data Structure

- Primitive Data Structure
- Non-Primitive Data Structure

Classification of Data Structure



Classification of Data Structure



Primitive data structures

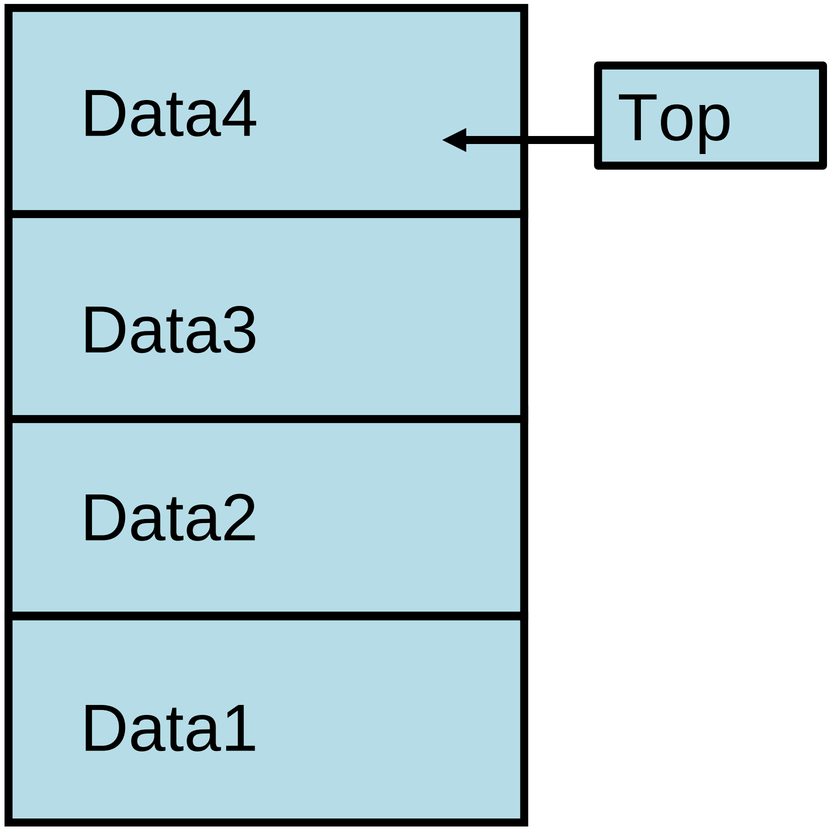
- Basic structures that are directly operated upon by the machine instructions.
- Usually built into the language, such as an integer, a float.

Non-Primitive data structures

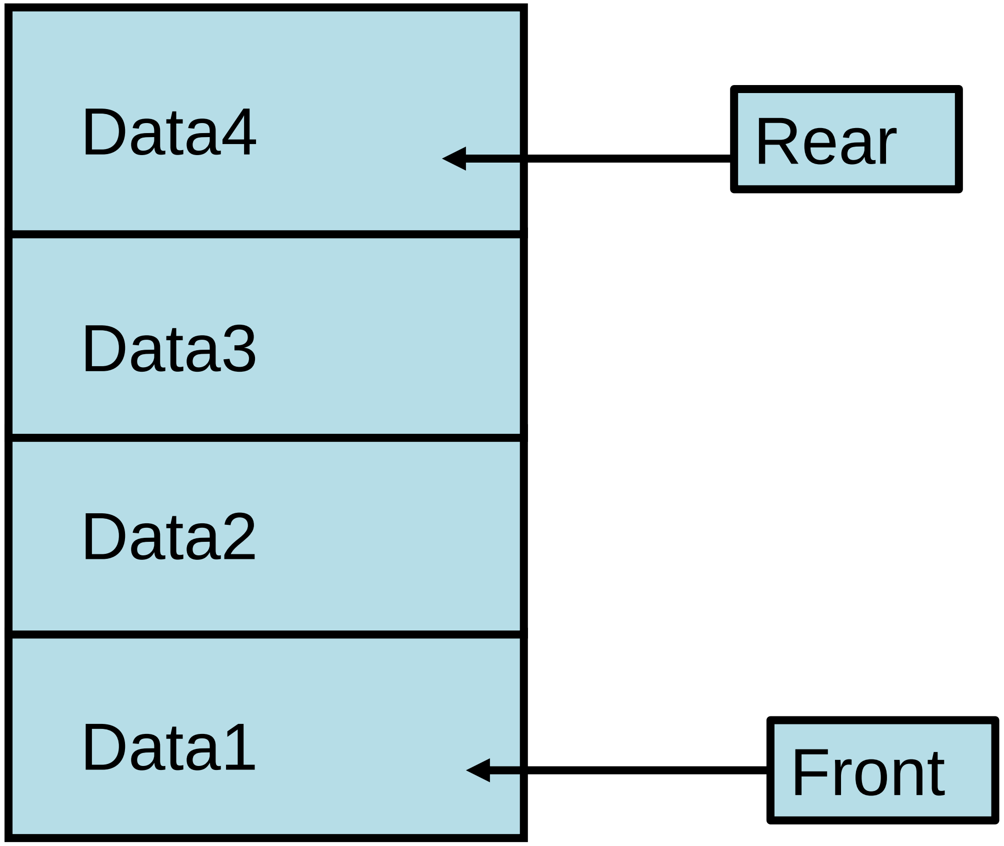
- More sophisticated data structures.
- Derived from the primitive data structures.
- Emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items.

Data structures and their representations

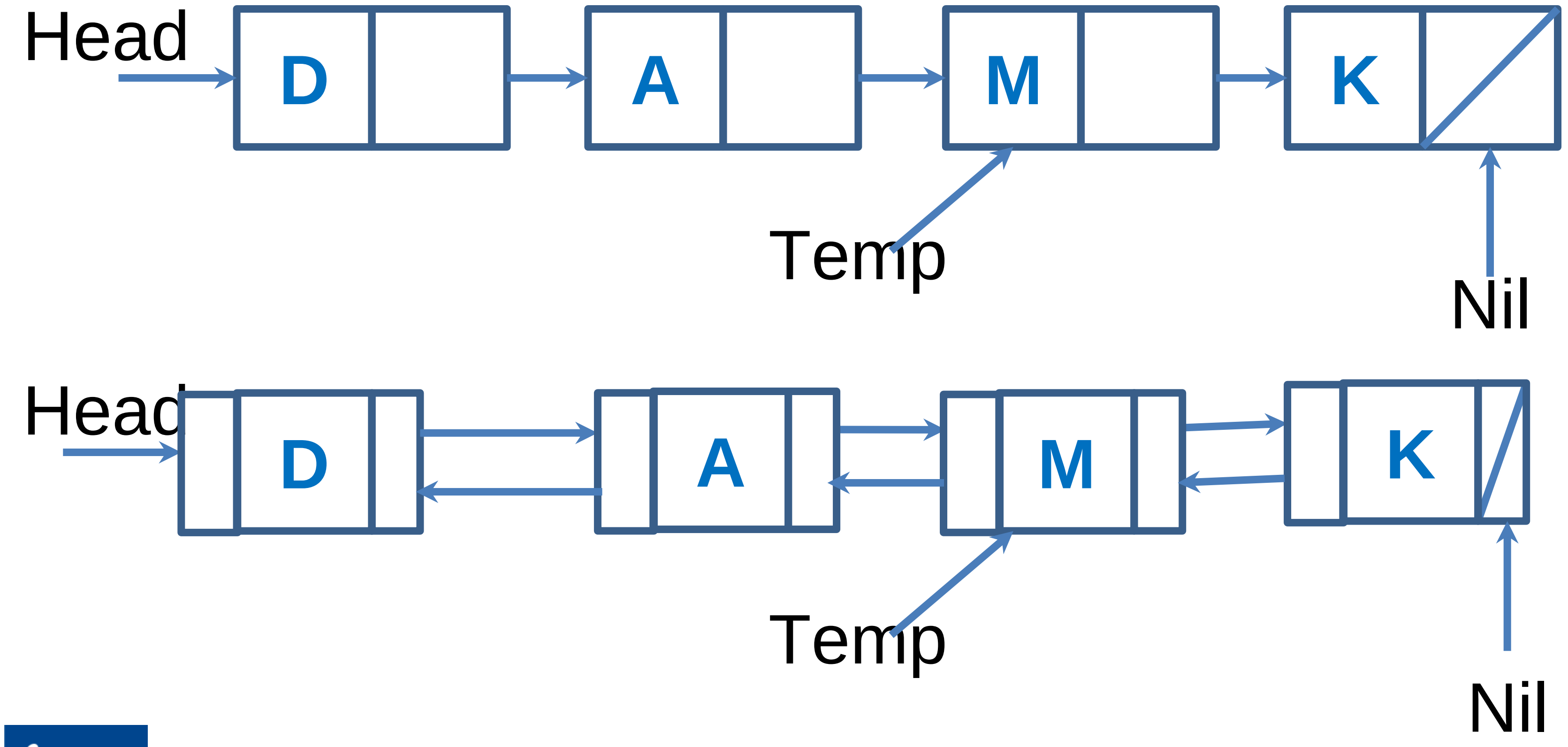
Stack



Queue



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



Tree

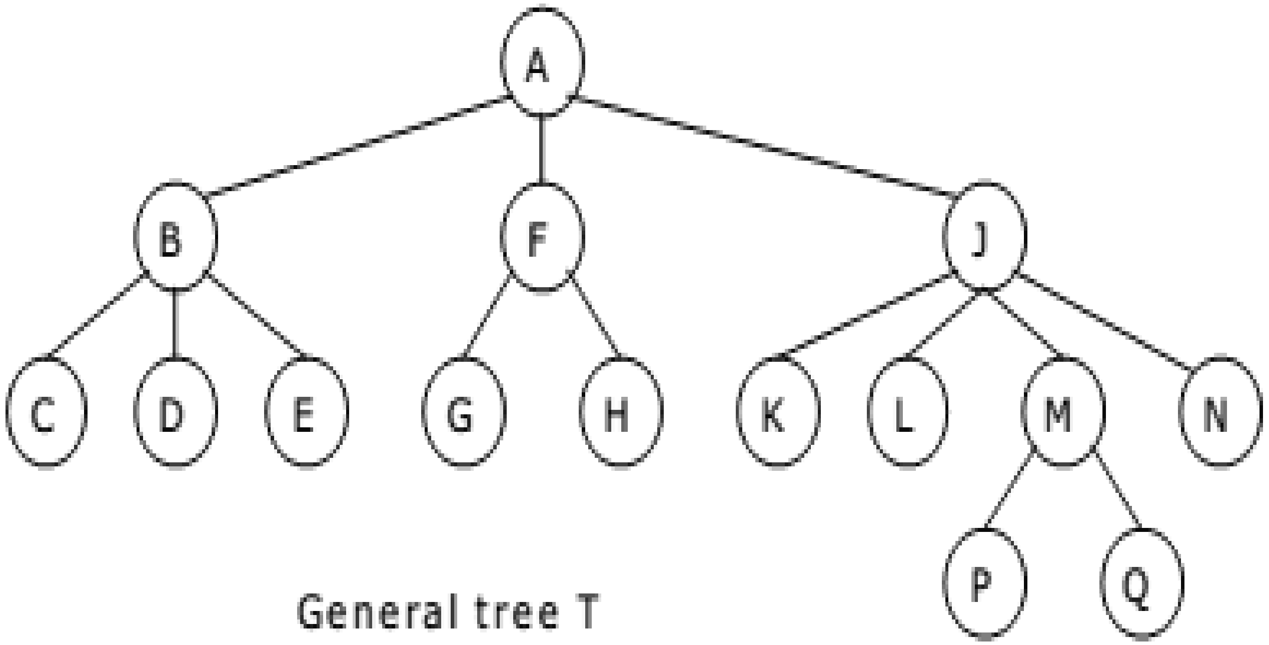


Image courtesy: ExamRadar.com

Binary Tree, Binary search tree and Heaps

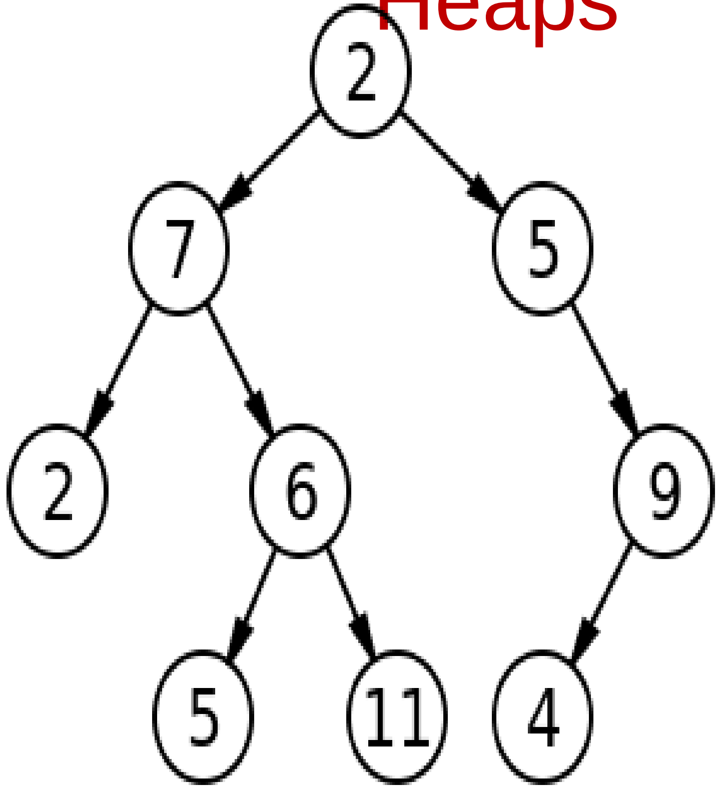


Image courtesy: ExamRadar.com

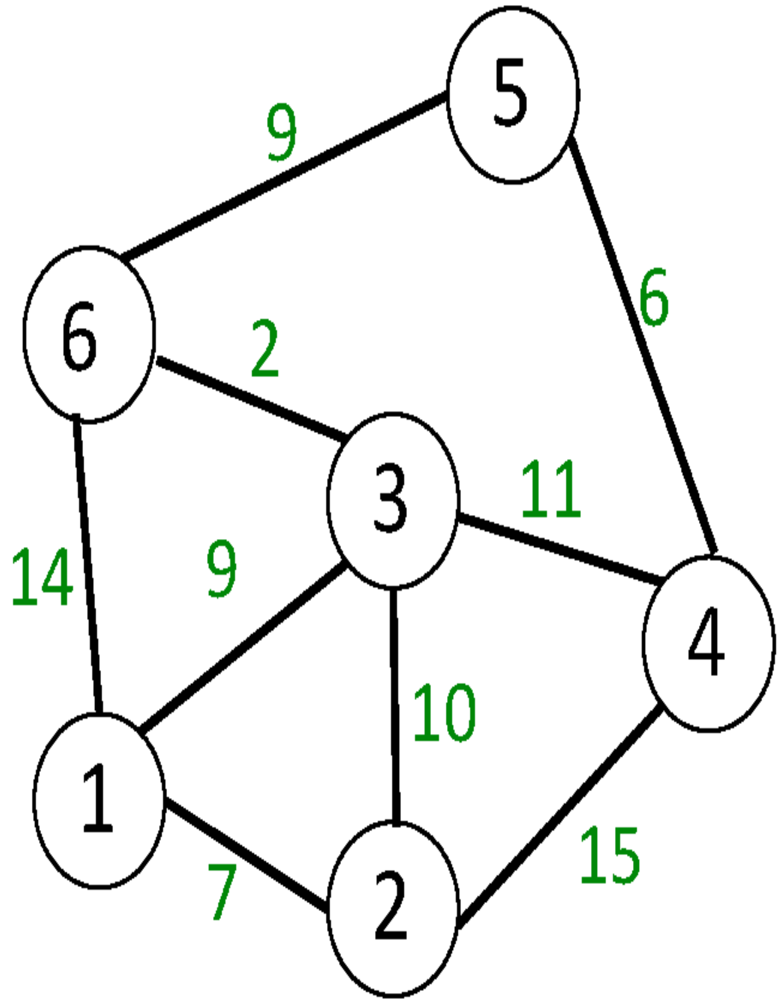


Image courtesy: Medium.com

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

Communicating your ideas in CSE

- Variable naming
- Comments
- Function declarations
- Algorithm writing
- ADT writing

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

What ADT doesn’t specify?

- Algorithm
- Platform
  - h/w
  - s/w
- Programming Language

What ADT does specify?

- Allowed values which follow the constraint(s) in definition
- Permitted operations on the data type being created
- Operations must have
  - Meaningful name
  - Return type
  - List of input parameters
  - Data types of all input parameters

Specify return types and inputs

- Data type: integer
- Operations:
  - Addition
  - Subtraction
  - Compare
  - Sum of digits
  - Log_{10}
- Specify name, no of inputs, their data types, data type of operation result

Specify return types and inputs

- Data type: integer array
- Specify name, no of inputs, their data types, data type of operation result
- Operation:
  - Sum of elements
  - Add/delete an element into/from array
  - Compare two arrays
  - Head or tail value of array

ADT and Operations

How do you perform following operations?
String a,b ; a+b, a/b, x : ? y
Integer x,y ; x+y, x/y, x :? y
Real m,n ; m+n, m/n, m : ? y

- Same operations on different data type might have different processes
- All data types might not follow same operations

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];

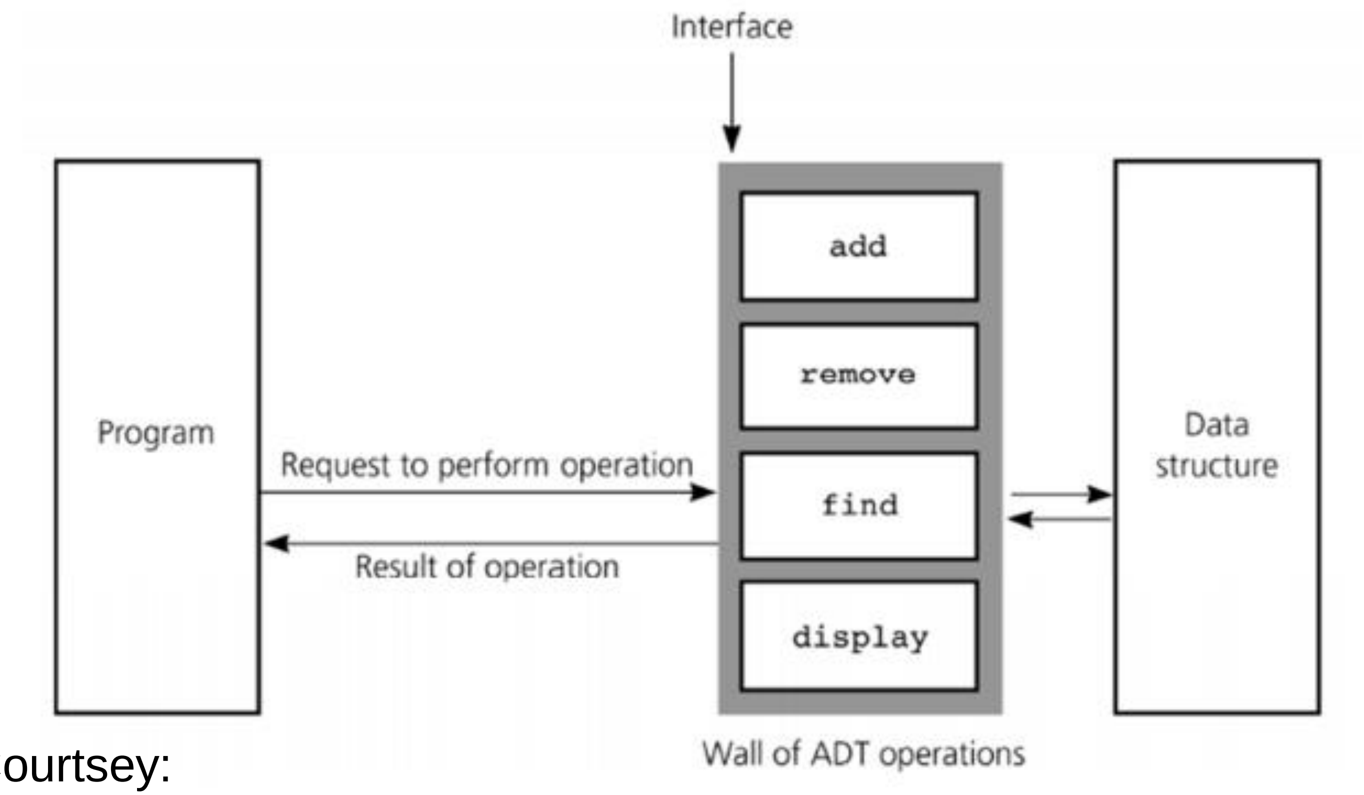
Abstract Data Types: Advantages

- 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

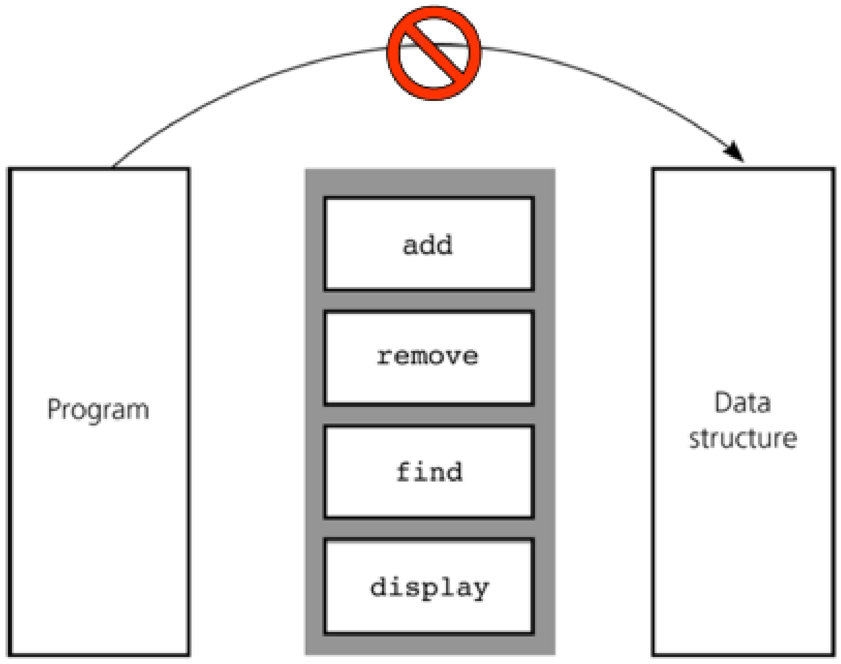


Courtsey: https://www.comp.nus.edu.sg/~stevenha/cs1020e/lectures/L5%20-%20ADT.pdf

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

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.

Write Array as ADT

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

Thank you