CHAPTER 1: INTRODUCTION TO DATA STRUCTURE AND ALGORITHM

BIBHA STHAPIT
ASST. PROFESSOR
IoE, PULCHOWK CAMPUS

SYLLABUS OUTLINE

- 1. Concept of Data Structure
- 2. The stack and queue
- 3. List
- 4. Linked List
- 5. Recursion
- 6. Trees
- 7. Sorting
- 8. Searching
- 9. Growth functions
- 10. Graphs

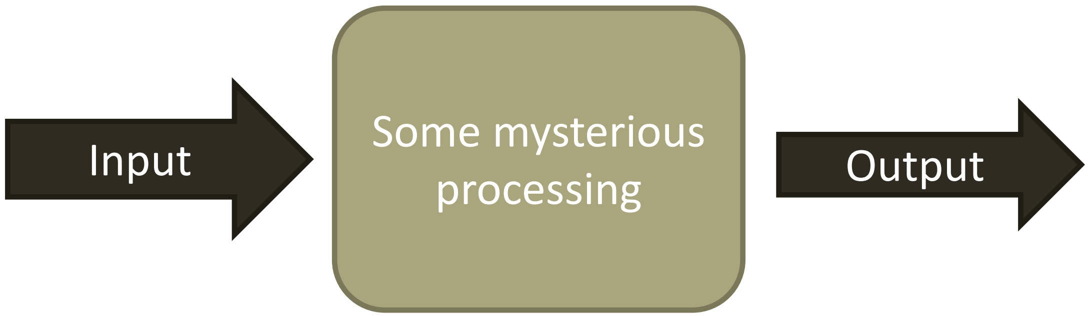
- Text Book: Data Structures using C and C++ -> Langsam, Augenstein and Tanenbaum

Introduction to Data Structure

- NEED??
- To apply the best practices for developing efficient solutions using computer programming.
- To understand the relevant concepts and decisions to apply the best fit data structure for the particular requirements.
- It deals with architecting, designing, developing and optimizing the applications using computer programming.
- Data structure helps the programmers to develop effective and reliable solutions.

Data Structure

- To know about the data structure, we'll have to know about the computer programming

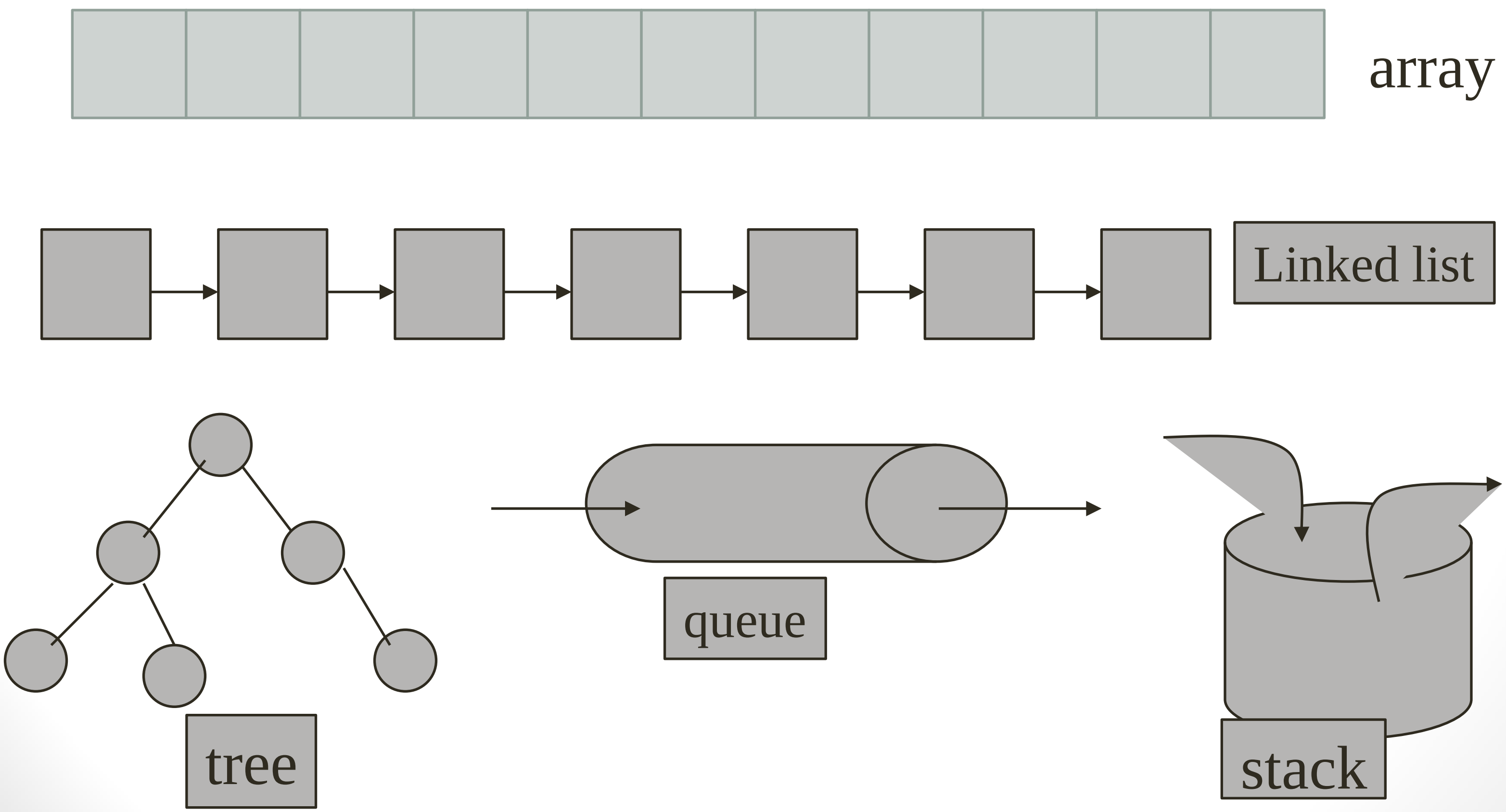


Example :

- Data structure for storing data of students:-
  - Arrays , Linked Lists
- Issues
  - Space needed
  - Operations efficiency (Time required to complete operations)
    - Retrieval, Insertion, Deletion

Data Structure

- Data structures let the input and output be represented in a way that can be handled efficiently and effectively.



Data Structure

- DEFINITION:
  - Particular way of storing and organizing data in a computer so that it can be used efficiently

- Data :
  - A piece of information
  - Taken as input, processed within a computer, and provided as output as information
  - Can be “atomic” or “composite”
  - Atomic data takes single value like integer 123
  - Composite data which can be further broken down to subfields like student record consists of roll_no, name, faculty, year etc.

Data Structure

- Data type:
  - Kind of data that a variable can store.
  - Built-in data types are system defined data types like int, float, char.
  - User-defined data type are created according to the need of the program like structure, union, class.
- Hence in brief, a data structure :
  - Depicts the logical representation of data in computer memory
  - Represents the logical relationship between various data elements
  - Helps in efficient manipulation of stored data elements
  - Allows programs to process data in efficient manner

Types of Data Structure

- Static and dynamic data structure
    - Static can store data upto fix number/size. E.g. Array
    - Dynamic allows to change its size during program execution. E.g. Linked-list
- Linear and non-linear data structure
    - In linear, data is stored in consecutive memory i.e. every element has unique predecessor and unique successor. E.g. Array, linked list, stack, queue
    - In non-linear, data is stored in non-consecutive memory. Eg. Tree, graph

Types of Data Structure

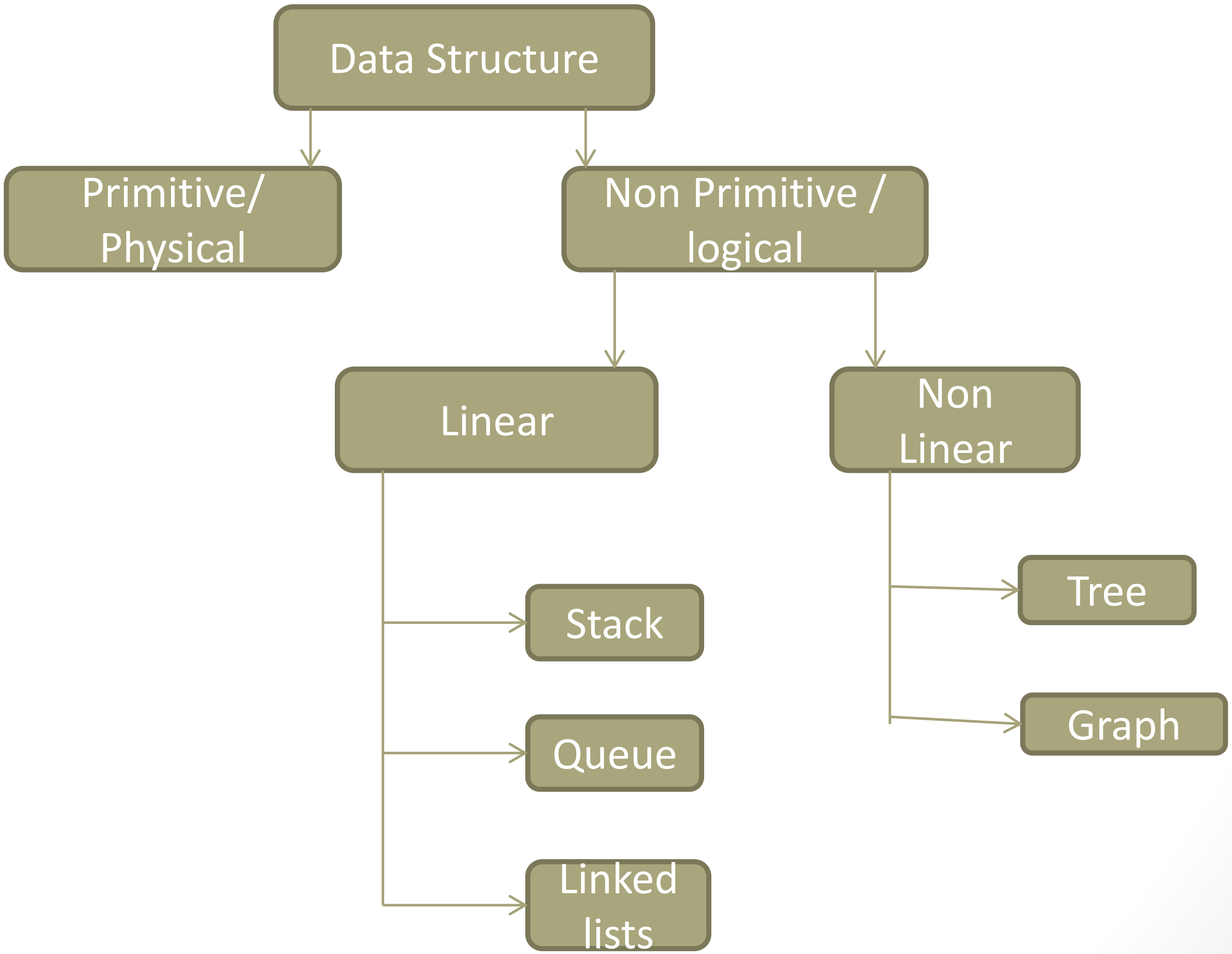
- Primitive and non-primitive data structure
  - Primitive data structures are generally primary or built-in types and are directly operated upon by machine instructions. E.g. integers, floating points, characters
  - Non-primitive emphasizes on structuring of homogeneous or heterogeneous data. E.g. Array, lists, linked list, tree

- Homogeneous and non-homogeneous data structure
  - Homogeneous consist same type of data. E.g. Array
  - Heterogeneous consists data of different types. E.g.. Structures, class

Types of Data Structure

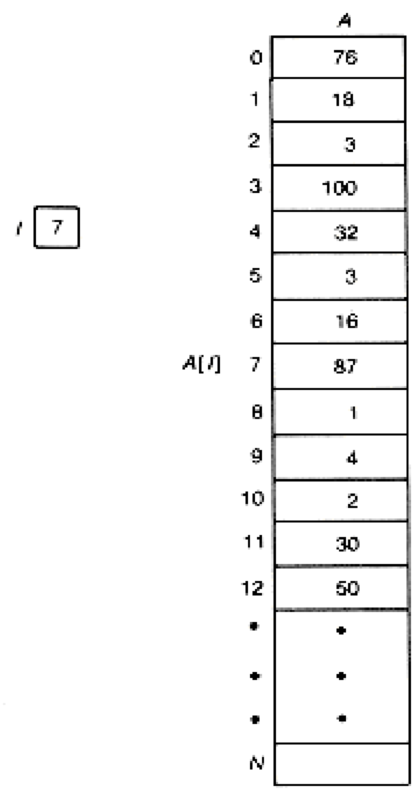
- Sequential and direct/random access data structure
  - In sequential, we must access preceding (n-1) data to access n^{th} data. E.g. Linked list
  - In direct access, we can directly access n^{th} element. E.g. array

Types of Data Structure



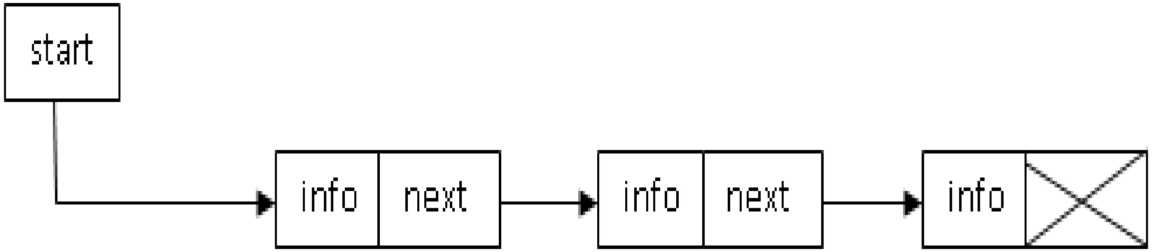
Overview of different Data Structures

- Arrays:
  - Collection of similar data elements
  - Elements are stored in consecutive memory.
  - Have random access to data
  - Application:
    - Used in program that require storing large collection of similar data



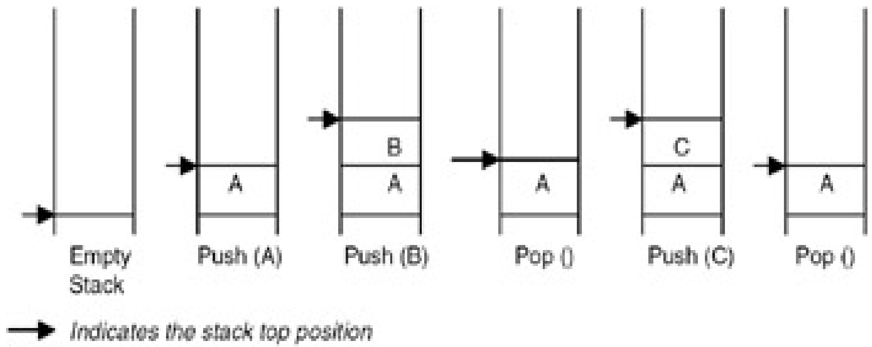
Overview of different Data Structures

- Linked list
  - It provides a flexible storage system by dynamically assigning the required memory.
  - Linked lists are special list of some data elements linked to one another.
  - Application:
    - Used wherever dynamic memory allocation is needed



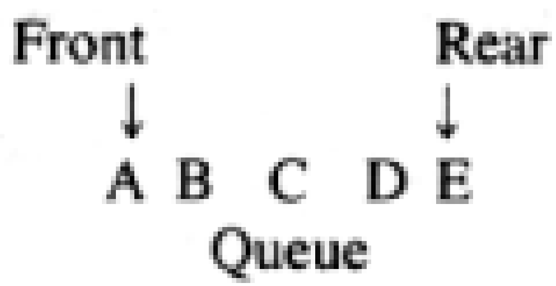
Overview of different Data Structures

- Stack:
  - List of elements with insertions and deletions permitted at one end only – called stack top
  - A stack data structure exhibits the LIFO (last in first out) property
  - Application:
    - System processes such as compilation and program control



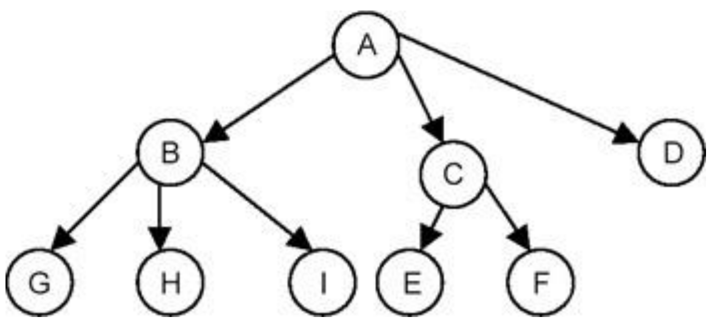
Overview of different Data Structures

- Queue
  - A list of elements with insertions permitted at one end—called the rear, and deletions permitted from the other end—called the front
  - a queue data structure exhibits the *FIFO* (*first in first out*) property.
  - Application:
    - Used in CPU scheduling, resource sharing



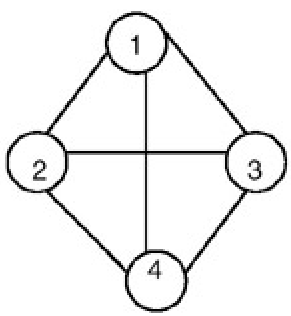
Overview of different Data Structures

- Trees:
  - Non- linear data structure
  - Arranges its **nodes** in the form of hierarchal tree structure
  - Application:
    - Used for storing hierarchical data, implementing search trees, maintaining sorted data.

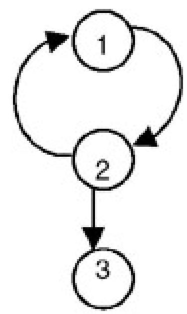


Overview of different Data Structures

- Graphs:
  - A graph is a structure made of two components: a set of vertices V, and a set of edges E
  - Application:
    - Modelling of networking systems, costing of network paths



Undirected Graph G_1



Directed Graph G_2

Operations on Data Structures

- Traversing
  - Accessing each record of data structure exactly once
- Searching
  - Identifying location of record that consists specific key value
- Inserting
  - Adding new record
- Deleting
  - Removing existing record
- Sorting
  - Arranging data in specific order
- Merging
  - Combining two different sorted records to produce single sorted data set.

Algorithm

- Once the data structure of particular application is chosen, an algorithm must be developed to manipulate related data items stored in it.
- It is a precise plan for performing sequence of actions
- Definition:
  - It is step-by-step finite set of instructions to solve a well-defined computational problem.
    - Sum of Two Numbers:
      - step 1 – START ADD
      - step 2 – get values of a & b
      - step 3 – c $\leftarrow$ a + b
      - step 4 – display c
      - step 5 – STOP

Algorithm

- Every algorithm must satisfy following criteria –
- *input:* there are zero or more quantities which are externally supplied;
- *output:* at least one quantity is produced;
- *definiteness:* each instruction must be clear and unambiguous;
- *finiteness:* if we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps;
- *effectiveness:* every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be definite as in (iii), but it must also be feasible.

Algorithm design techniques

- Top down approach

- Bottom up approach
  - Brute Force
    - goes through all the possible solutions one after another until you find the optimum solution.
    - Linear Search algorithm

  - Divide and conquer
    - This algorithm solves a problem by dividing the original problem into smaller chunks which are similar sub-problems.
    - The solutions of these smaller sub problems are later combined to get the solution of the given original problem.
    - Quick sort and merge sort

Algorithm design techniques

- Greedy
  - works by taking a decision that appears the **best at the moment**, without thinking about the future.
  - Djikstra's algorithm, Prim's algorithm

- Branch and bound
  - The first solution is remembered by the algorithm and set as a benchmark. It returns the original or the first solution if no solution is found after the algorithm completes the whole search space.
  - Travelling salesman problem

- Simple Recursive
  - Process in which a problem is defined in terms of itself.
  - The problem is solved by repeatedly breaking it into smaller problems, which are similar in nature to the original problems.
  - Fibonacci series, Tower of Hanoi

Algorithm design techniques

- Randomized
  - Here, random numbers are used to make some decisions. These randomized algorithms are approximated using a **pseudo – random number generator**.
  - Quick sort using random number for pivot element, Monte-Carlo algorithm

- Backtracking
  - Based on depth-first recursive search
  - Depth-first recursive search in a tree

Analysis of algorithm

- The two factors of Algorithm Complexity are:
  - Time Factor: Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
  - Space Factor: Space is measured by counting the maximum memory space required by the algorithm.

- Therefore the complexity of an algorithm can be divided into two types:
  - Space Complexity:
    - Space complexity of an algorithm refers to the amount of memory that this algorithm requires to execute and get the result. This can be for inputs, temporary operations, or outputs.

Analysis of algorithm

- Time Complexity:
  - Time complexity of an algorithm refers to the amount of time that this algorithm requires to execute and get the result. This can be for normal operations, conditional if-else statements, loop statements, etc.

  - Worst case running time
    - It assures that the algorithm will never go beyond this limit
  - Average case running time
    - It assumes that all inputs of given size are equally likely
  - Best case running time
    - It is used to analyse an algorithm under optimal condition

Abstract Data Type (ADT)

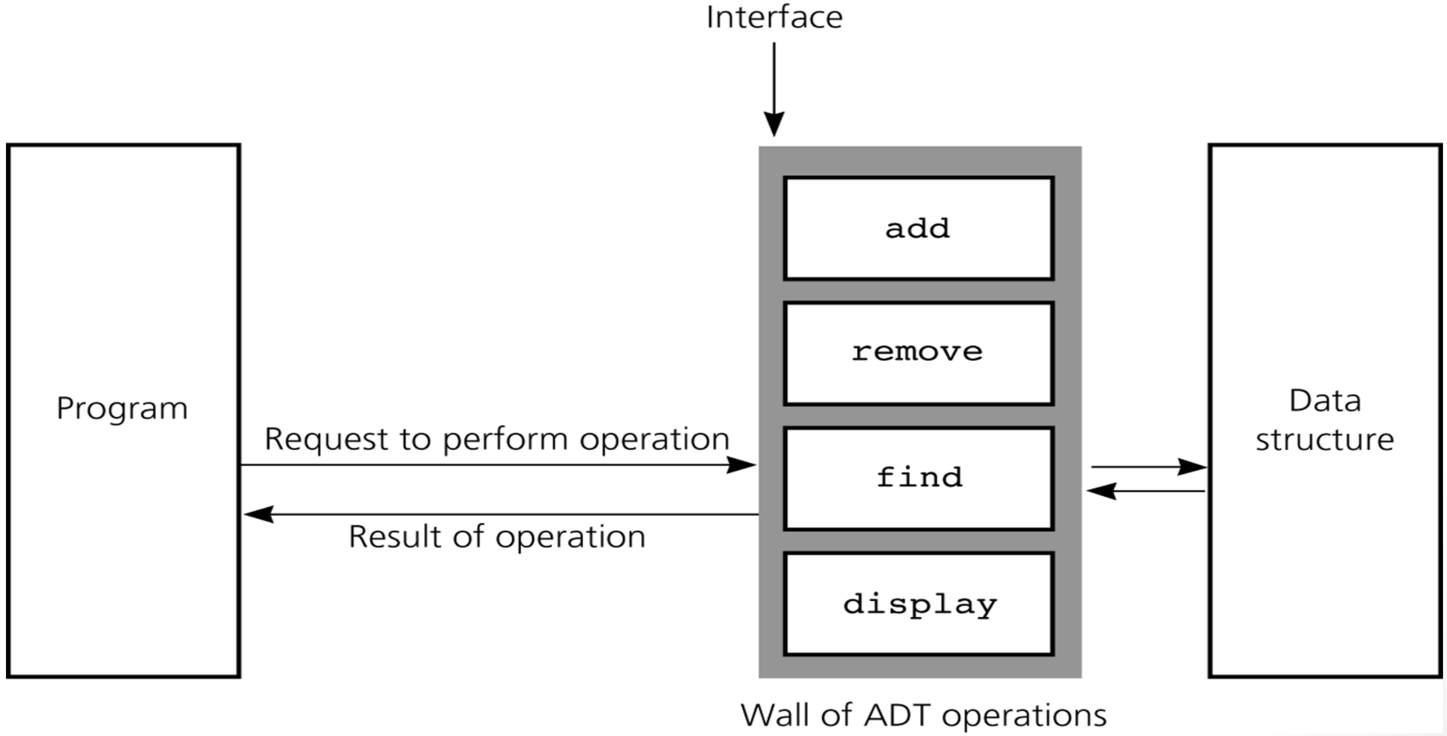
- Data abstraction
  - Asks you to think *what* you can do to a collection of data independently of *how* you do it
  - Allows you to develop each data structure in relative isolation from the rest of the solution

- An abstract data type (ADT) is composed of
  - A collection of data
  - A set of operations on that data

- Specifications of an ADT indicate what the ADT operations do (but not how to implement them)
- Implementation of an ADT includes choosing a particular data structure

Abstract Data Type (ADT)

- An ADT consists of
  - Declaration of data
  - Declaration of operations
  - Encapsulation of data and operations : data is hidden from user and can be manipulated only by means of operations`



Abstract Data Type (ADT)

- To implement an ADT, you need to choose:
  - A data representation (VALUE DEFINITION)
    - must be able to represent all necessary values of the ADT
    - should be private
  - An algorithm for each of the necessary operation (OPERATOR DEFINITION):
    - must be consistent with the chosen representation
    - all auxiliary (helper) operations that are not in the contract should be private

Abstract Data Type (ADT)

```
/*value definition*/
abstract typedef<int, int> RATIONAL;
condition RATIONAL[1] != 0;

/*operator definition*/
abstract equal(a,b)                          /* written a == b*/
RATIONAL a,b;
postcondition equal == (a[0]*b[1] == b[0]*a[1]);

abstract RATIONAL makerational(a,b)     /* written [a,b]*/
int a,b;
precondition  b != 0;
postcondition makerational[0]*b == a*makerational[1]

abstract RATIONAL add(a,b)   /* written a + b */
RATIONAL a,b;
postcondition add == [a[0] * b[1] + b[0] * a[1], a[1]*b[1]]

abstract RATIONAL mult(a,b) /* written a * b */
RATIONAL a,b;
postcondition mult == [a[0] * b[0], a[1] * b[1] ]
```

THANK YOU!