Introduction

Introduction

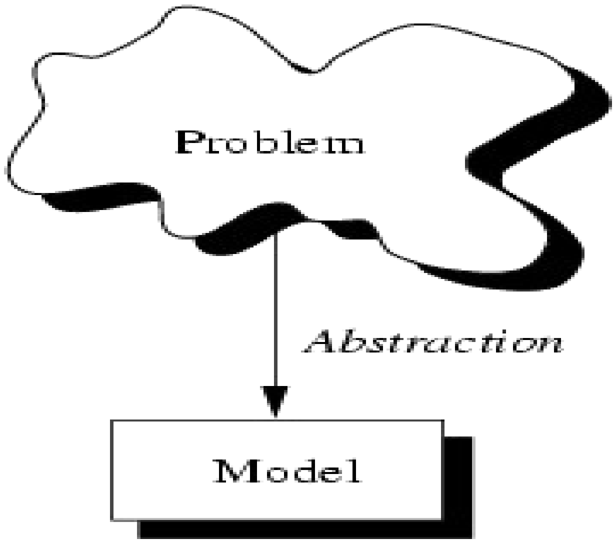
- Introduction to Data Structures
  - Abstract Data types
  - Abstraction
- Algorithms
  - Definition of Algorithm
  - Properties of Algorithms

- A program is written in order to solve a problem.
- A solution to a problem actually consists of two things:
  - A way to organize the data
  - Sequence of steps to solve the problem
- The way data are organized in a computers memory is said to be Data Structure and the sequence of computational steps to solve a problem is said to be **an algorithm**. Therefore,

a program is nothing but data structures plus algorithms.

Introduction to Data Structures

•Given a problem, the first step to solve the problem is obtaining ones own abstract view, or *model*, of the problem. This process of modeling is called *abstraction*.



- The model defines an abstract view to the problem. This implies that the model focuses only on problem related stuff and that a programmer tries to define the *properties* of the problem.
- These properties include
  - The *data* which are affected and
  - The *operations* that are involved in the problem.
- With abstraction you create a well-defined entity that can be properly handled. These entities define the *data structure* of the program.
- An entity with the properties just described is called an *abstract data type* (ADT).

- **Abstract Data Types**
  - An ADT consists of an abstract data structure and operations. Put in other terms, an ADT is an abstraction of a data structure.
  - The ADT specifies:
    1. What can be stored in the Abstract Data Type
    2. What operations can be done on/by the Abstract Data Type.
  - For example, if we are going to model employees of an organization:
  - This ADT stores employees with their relevant attributes and discarding irrelevant attributes.
  - This ADT supports hiring, firing, retiring, … operations.

- There are lots of formalized and standard Abstract data types such as Stacks, Queues, Trees, etc.

- Operations of Stack ADT
  - size(), this function is used to get number of elements present into the list
  - insert(x), this function is used to insert one element into the list
  - remove(x), this function is used to remove given element from the list
  - get(i), this function is used to get element at position i
  - replace(x, y), this function is used to replace x with y value

- Operations of Stack ADT
  - −isFull(), This is used to check whether stack is full or not
  - isEmpry(), This is used to check whether stack is empty or not
  - push(x), This is used to push x into the stack
  - pop(), This is used to delete one element from top of the stack
  - peek(), This is used to get the top most element of the stack
  - size(), this function is used to get number of elements present into the stack

- Operations of Queue ADT
  - isFull(), This is used to check whether queue is full or not
  - isEmpry(), This is used to check whether queue is empty or not
  - insert(x), This is used to add x into the queue at the rear end
  - delete(), This is used to delete one element from the front end of the queue
  - size(), this function is used to get number of elements present into the queue

Algorithms

- An algorithm is a well-defined computational procedure that takes some value or a set of values as input and produces some value or a set of values as output.
- Data structures model the static part of the world. They are unchanging while the world is changing.
- In order to model the dynamic part of the world we need to work with algorithms. Algorithms are the dynamic part of a program’s world model.

- An algorithm transforms data structures from one state to another state in two ways:
  - An algorithm may change the value held by a data structure
  - An algorithm may change the data structure itself

Properties of an algorithm

- **Finiteness**: Algorithm must complete after a finite number of steps.
- **Definiteness**: Each step must be clearly defined, having one and only one interpretation. At each point in computation, one should be able to tell exactly what happens next.
- **Sequence**: Each step must have a unique defined preceding and succeeding step. The first step (start step) and last step (halt step) must be clearly noted.
- **Feasibility**: It must be possible to perform each instruction.
- **Correctness**: It must compute correct answer all possible legal inputs.
- **Language Independence**: It must not depend on any one programming language.

- **Completeness**: It must solve the problem completely.
- **Effectiveness**: It must be possible to perform each step exactly and in a finite amount of time.
- **Efficiency**: It must solve with the least amount of computational resources such as time and space.
- **Generality**: Algorithm should be valid on all possible inputs.
- **Input/Output**: There must be a specified number of input values, and one or more result values.