1 Introduction

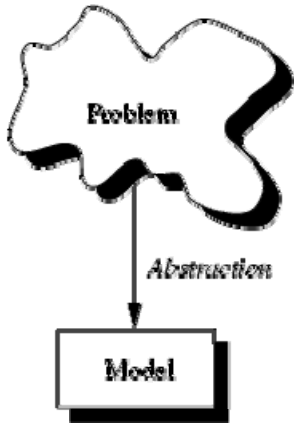
A program is written in order to solve a problem. A solution to a problem actually consists of two things:

- 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.

1.1 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).

1.1.1 Abstract Data Types

An ADT consists of an abstract data structure and operations. In other terms, an ADT is an abstraction of a data structure. The ADT specifies:

- What can be stored in the Abstract Data Type
- 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, and the like operations.

A data structure is a language construct that the programmer has defined in order to implement an abstract data type. There are lots of formalized and standard Abstract data types such as Stacks, Queues, Trees, etc.

Do all characteristics need to be modeled? Not at all

- It depends on the scope of the model
- It depends on the reason for developing the model

1.1.2 Abstraction

Abstraction is a process of classifying characteristics as relevant and irrelevant for the particular purpose at hand and ignoring the irrelevant ones. Applying abstraction correctly is the essence of successful programming. How do data structures model the world or some part of the world?

- The value held by a data structure represents some specific characteristic of the world
- The characteristic being modeled restricts the possible values held by a data structure
- The characteristic being modeled restricts the possible operations to be performed on the data structure.

Note: - Notice the relation between characteristic, value, and data structures

1.2 Algorithm

1.2.1 Algorithm Defined

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

The quality of a data structure is related to its ability to successfully model the characteristics of the world. Similarly, the quality of an algorithm is related to its ability to successfully simulate the changes in the world.

However, independent of any particular world model, the quality of data structure and algorithms is determined by their ability to work together well. Generally speaking, correct data structures lead to simple and efficient algorithms and correct algorithms lead to accurate and efficient data structures.

1.2.2 Properties of 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.

1.3 Mathematics Review (Student Work)

- **Exponents**
- **Logarithms**
- **Series**

1.4 Proofing statements

We can proof statements in data structure analysis either by:

1.4.1 Proof by induction

A proof by induction has two standard parts. The first part is to proof the base case, which is establishing that the theorem is true for some small value(s). Next an inductive hypothesis is assumed.
Generally this means that the theorem is true for all case up to some limit k, using this assumption, the theorem is then shown to be true for the next value, which is typically k + 1.

1.4.2 Proof by contradiction

Proof by contradiction proceeds by assuming that the theorem is false showing that this assumption implies that some property is false, and hence the original assumption was erroneous.