1 INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

1.1 CONTEXT OF COMPUTING

Any computing application can be viewed as processing some input data to produce some outputs as illustrated in Figure 1.1 below. The processing box (black box) at the middle represent the computing application that we usually implement by writing a program in some programming language like C/C++ or Java.

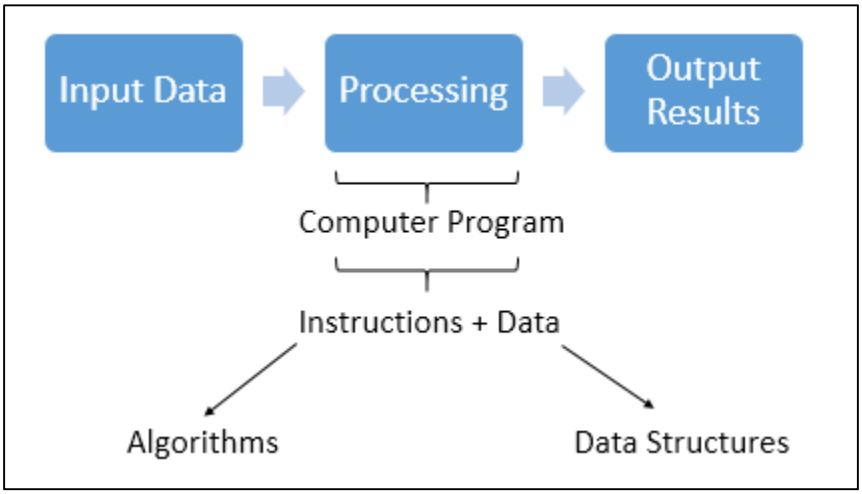


Figure 1-1: Context of Computing

If we have a close look at a computer program, regardless of what tool or language we used to implement it, one can always visualize data and set of instructions, specifying how to manipulate the data to derive the desired outputs. In other words, we can state that a computer programme essentially consists of instructions and data. At the machine code level one can visualize the existence of these two components in the op-code and operands parts of a machine code instruction. In a high-level programming language the two components are supported by, control structures, operators and variables of available datatypes.

The focus of this subject is to study the common approaches available to structure the instructions and data within a computer program to tackle complex computing problems.

1.2 WHAT ARE ALGORITHMS?

Organization of instructions to carry out a given task can be defined as an algorithm. In other words an algorithm is a set of instructions or steps intend to solve a given problem.

- An algorithm describes the method of solving a problem.
- Algorithm is an organization of set of steps to be carried out to achieve a specific task.

Algorithms are implemented using the 3 fundamental constructs available in any programming language; Sequencing, Selection, and Iteration. Algorithms are evaluated based on the CPU time and memory they consume to solve the given problem.

Example:

Problem - Finding the maximum from a set of values

Algorithm - *max = data[0]*

For (i=1; i<n; i++)

if(data[i]>max) max=data[i];

Analysis - Both running time and space/memory requirement are proportional to the size of the problem *n*.

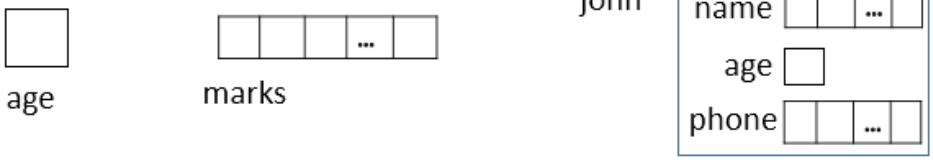
Discussion - Can we stop the search in case the maximum is in the first element?

1.3 WHAT ARE DATA STRUCTURES?

A program need to store or hold the input, intermediate and output data items in its operation. Programming languages support data storage via Variables, Arrays and some form of Records such as Structure in C. These elementary data structures are usually constructed based on the primitive data types available in a programming language such as *int*, *char* or *float* in C. More complex data structures such as Linked Lists, Trees or Graphs, those required to tackle complex problems, can be implemented using the basic data types and other language elements available in a programming environment.

Examples:

- `int age;`
- `float marks[100];`
- ```
  struct person { char name[20];
                  int age;
                  char phone[10]; } john;
  ```


  ```

1.4 ABSTRACT DATA TYPES (ADT)

Abstract data type or an ADT is a data structure together with a set of standard access functions to access the data stored. One does not need to know the data structures inside the ADT box or how access functions are implemented to use the ADT. Provision of primitive data types in a programming language like C are also examples for ADTs. For instance, one can use floating point numbers in C, without knowing how exactly they are kept in the RAM – one simply uses the arithmetic operators as the access functions in this scenario.

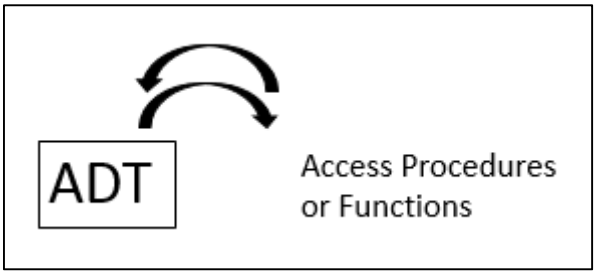
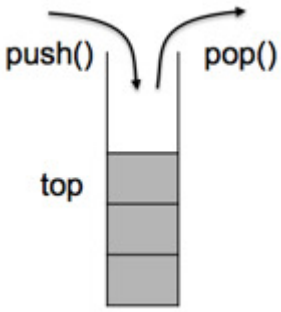


Figure 1-2: Concept of an Abstract Data Type

Further Examples:

Stack ADT

Data stored in the stack ADT can be accessed only using the two access functions push() and pop(). The users of the stack ADT only need to know the two access functions, how they can be called, what arguments are required and what values they return. How they have been implemented or the data is stored in the stack is transparent to the users.



The same explanation holds to the Queue ADT as well. Users only need to know how dequeue() and enqueue() can be used, and the queue is accessed only using the two access functions.

Queue ADT

1.5 ORGANIZATION OF THE TOPICS

In Chapter 2, we discuss how algorithms and their performance can be evaluated and Asymptotic Analysis is introduced as an effective measure for algorithm analysis. Chapter 3 focuses on the common algorithms available for searching and sorting – two most common problems in computing. Chapter 4 introduces common algorithm design approaches and recursion. Chapter 5 introduces Stack and Queue ADTs and their static implementation using the C Language. Chapter 6 introduces Linked List as a dynamic data

structure and discusses dynamic implementation of the Stack and Queue ADTs in C Language. Chapter 7 introduces Trees and related concepts. Implementation of Binary Search Tree (BST) is discussed at length allowing the readers to familiarize with efficient implementation of the Tree concepts using recursion and Features of the C Programming Language.