MODULE 1

INTRODUCTION TO DATA STRUCTURE

DATA STRUCTURE AND ALGORITHMS

FACULTY OF COMPUTING
UNIVERSITI TEKNOLOGI MALAYSIA

MODULE 1: INTRODUCTION TO DATA STRUCTURE

OBJECTIVES FOR STUDENTS

1. Understand and able to describe data structure and algorithm concept.
2. Know programming development paradigm and able to apply in problem solving.
3. Know the key programming principle and able to apply in program development and implementation.

KEY CONCEPT

1.0 SOFTWARE ENGINEERING AND PROBLEM SOLVING

1.1. **Software engineering** - Provides techniques to facilitate the development of computer program. A systematic approach using engineering principle to develop, implement and maintain software.

1.2. **Problem solving** :
- The entire process of taking the statement of a problem and developing a computer program that solves that problem
- Requires to pass many phases, from understanding the problem, design solution and implement the solution.

1.3. A **solution** to a problem is computer program written in C++ and consist of modules :
- A single, stand-alone function
- A method of a class
- A class
- Several functions or classes working closely together
- Other blocks of code

1.4. **Challenges** to create a good **solution**:
i. Create a good set of modules that
   - must store, move, and alter data
   - use algorithms to communicate with one another

ii. Organize your data collection to facilitate operations on the data in the manner that an algorithm requires.

iii. Functions and methods implement algorithms.

2.0 ALGORITHM

2.1. **Algorithm** - A **step-by-step** recipe for performing a task within a finite period of time. A **sequence of instructions**, often used for calculation and data processing.

2.2. Algorithms often operate on a collection of data, which is stored in a structured way in the computer memory.

2.3. It is formally a type of effective method in which a list of well-defined instructions for completing a task will:

- when given an initial state, (INPUT)
- proceed through a well-defined series of successive states, (PROCESS)
- eventually terminating in an end-state (OUTPUT)

2.4. 3 types of **algorithm** basic **control structure** :

i. Sequential
ii. Selection
iii. Repeatition (Looping)

2.5. Simple algorithm to withdraw money at ATM machine

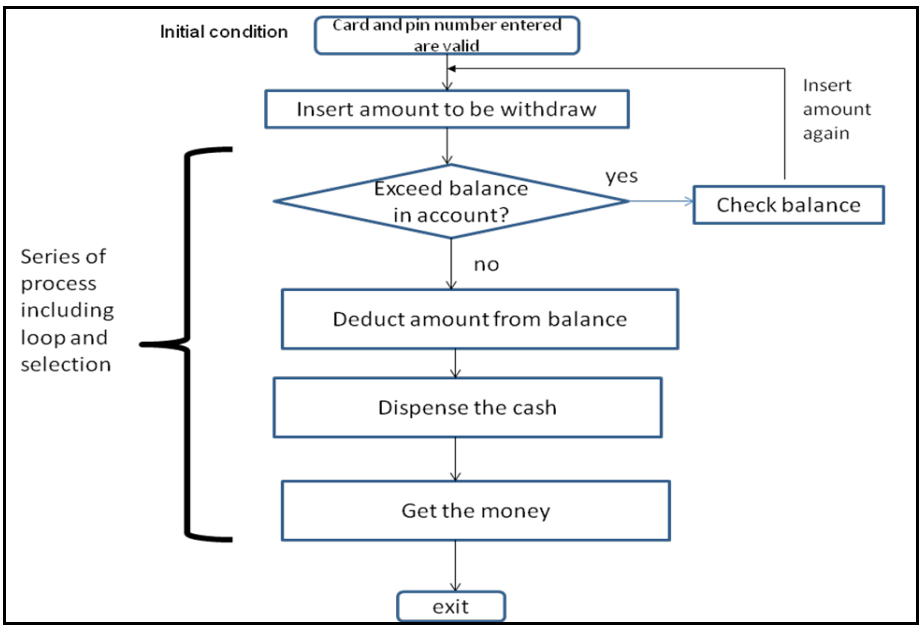

Figure 1.1 – Flowchart to withdraw money from ATM machine

2.6. Basic **algorithm characteristics**

- Finite solution
- Clear instructions

- Has input to start the execution
- Has output as the result of the execution
- Operate effectively

2.7. Algorithm **creation techniques**
- Flowchart, pseudo code, structure chart, language etc.

2.8. Factors for measuring good algorithm
- Running time and total memory usage

3.0 DATA STRUCTURE

3.1. **Data Structure** - a way of storing data in a computer so that it can be used efficiently.
- Carefully chosen data structure will allow the most efficient algorithm to be used.
- A well-designed data structure allows a variety of critical operations to be performed, using as few resources, both execution time and memory space, as possible.

3.2. **Operations on the data structure**
- Traversing- access and process every data in data structure at least once
- Searching – search for a location of data
- Insertion – insert item in the list of data
- Deletion - delete item from a set of data
- Sorting – sort data in certain order
- Merging – merge multiple group of data

3.3. **Data types :** basic data types and structured data types
Basic Data Types (C++) – store only a single data
- Integral
- Boolean – bool
- Enumeration – enum
- Character - char
- Integer – short, int, long
- Floating point – float, double

Structured Data Types – shown in Figure 1.2.
Storage Structure
- Array – can contain multiple data with the same types
- Struct – can contain multiple data with different type

```
typedef struct {
   int age;
   char *name;
   enum {male, female} gender;
```

```
} Person;
```

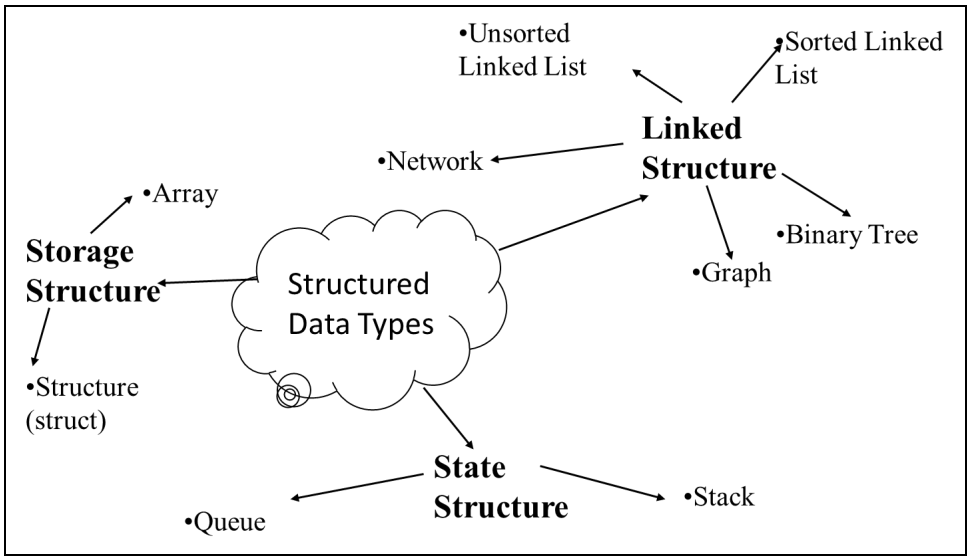

Figure 1.2 : Structured Data Types

Linked Data Structure

- Linear Data Structure with restriction (state structure)
  - Queue and Stack
- Linear Data Structure with no restriction
  - Unsorted linked list
  - Sorted linked list
- Non-linear Data Structure
  - Binary Tree
  - Graph

3.4. Queue

- First-In-First-Out (FIFO) data structure
- The first element added to the queue will be the first one to be removed, example: a queue at the post office, at the bank or at the supermarket.

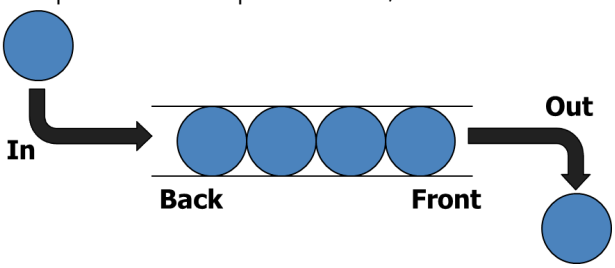

Figure 1.3 : Queue

3.5. Stack

- Based on the principle of *Last In First Out (LIFO)*.
- Stacks are used extensively at every level of a modern computer system, example: compiler.

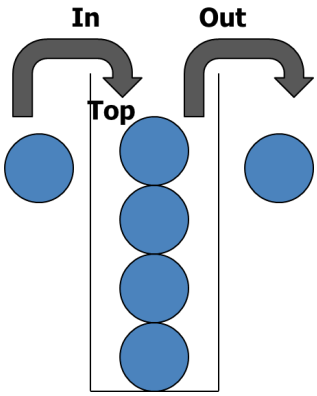

Figure 1.4 : Stack

3.6. Linked list

- Consists of a sequence of nodes, each containing arbitrary data fields and one or two references (links) pointing to the next and/or previous nodes
- Singly linked list – each node contains a value and a link to the next node – Figure 1.5.

Figure 1.5 Singly linked list

- Circularly linked list - each node contains a value and a link to the next node. The link in the last node point to the first node in the list – Figure 1.6.

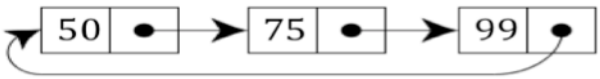

Figure 1.6 Circularly linked list

- Doubly linked list – the node in doubly linked list contains a value and two link that point to the previous node and point to the next node – Figure 1.7.

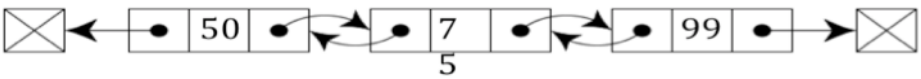

Figure 1.7 Doubly circular linked list

3.7. Sorted linked list

- Data stored in ascending or descending order with no duplicates.
- Insertion at front, middle or rear of the list.
- Deletion will not affect the ascending / descending order of the list.

3.8. **Unsorted linked list** - A linked list with no ordering.

3.9. Binary Tree

- Non-linear data structure.
- A **tree structure** is a way of representing the hierarchical nature of a structure in a graphical form
- A **binary tree** is a tree data structure in which each node has at most two children
- Used for searching big amount of data efficiently.

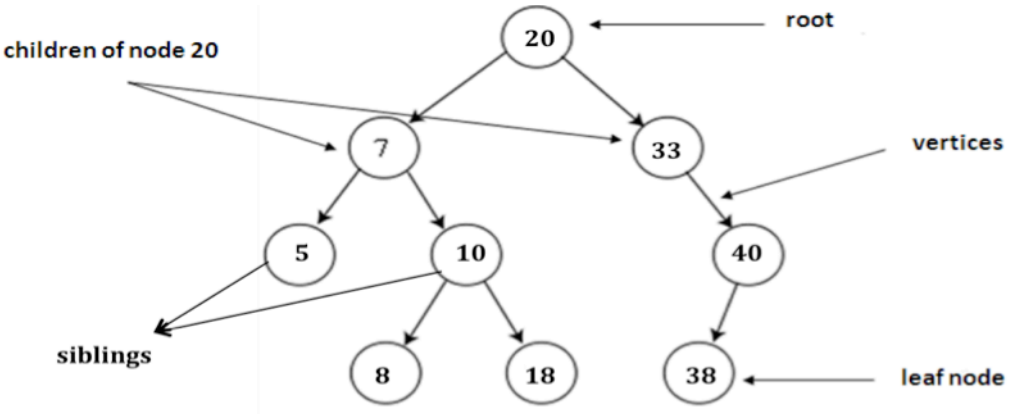

Figure 1.8 Binary Tree

3.10. Graph

- A graph consists of a set of vertices, and a set of edges, such that each edge in is a connection between a pair of vertices.
- Some applications require visiting every vertex in the graph exactly once.
- The application may require that vertices be visited in some special order based on graph topology.
- Examples:
  - Artificial Intelligence Search (Breadth-first search, depth first search)
  - Shortest paths problems
  - Web sites containing a link to and from other websites.
  - Graph that represent courses and the pre-requisites.

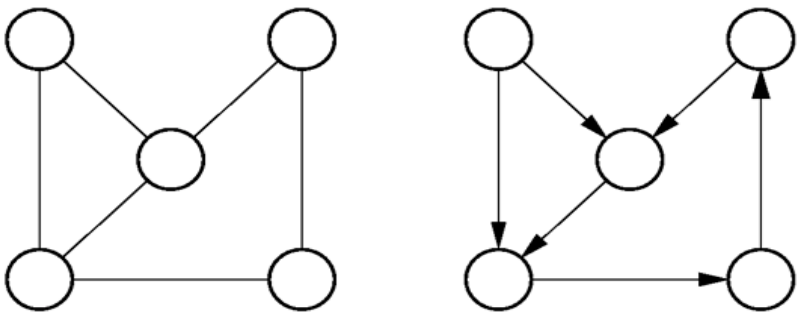

Figure 1.9 : Undirected Graph and Directed Graph

3.11. Network

- Network is a directed graph.
- Can be used to represent a route.
- Example :
  - A route for an airline.
  - A route for delivery vehicles.
- Figure 1.9 is a weighted network that represents a route for a delivery truck. The route shows all cities in Johor for the truck to deliver items and the time taken for a journey from one city to another.

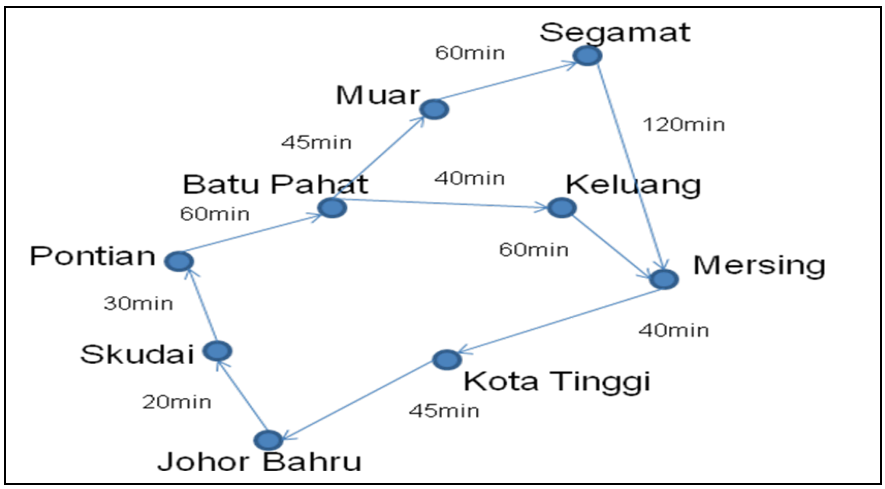

Figure 1.9 : Weighted Graph

4.0 PROGRAMMING PARADIGM

4.1. **Phases of Software Development** – provide a very systematic and organized approach for developing software.

i. **System Requirements** – Generally, this phase provide planning for the project. Identify objectives, job scope, resources such as cost, people and equipments for the project and provide schedule for the work plan.

ii. Analysis – Conduct preliminary investigation, study the current system, determine the user requirements and provide solution to the problem.

iii. **Design** – Develop details of the system by dividing into modules. Prepare algorithms for each module.

iv. **Coding** – Transfer from design to computer source codes.

v. **System Testing** – To ensure that the system can work properly and free from errors, either syntax or logic errors. 3 types of testing: system testing, integration testing and acceptance test.

vi. **Maintenance** – Monitor system performance; identify errors not detected during system testing, and enhancement to the system.

4.2. Algorithm is the steps to solve problems by breaking the analyzed problems.

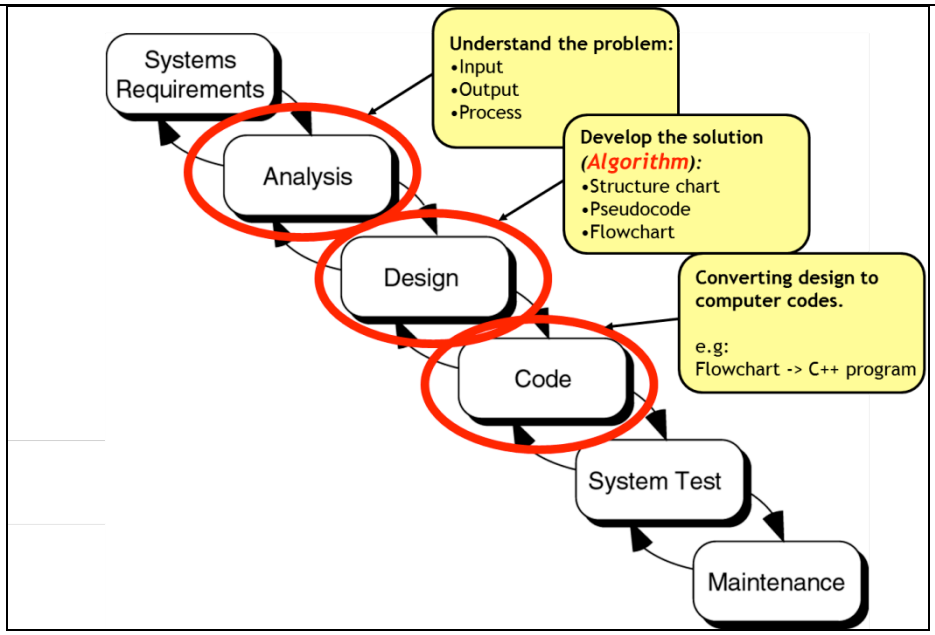

Figure 1.11 System Development Process

5.0 PROGRAMMING PRINCIPLE

5.1. **Seven Key Isuues in Programming** are as follows :

1. Modularity
2. Style
3. Modifiability
4. Ease of Use
5. Fail-safe programming
6. Debugging
7. Testing

5.2. **Modularity** has a favorable impact on :

1. Constructing programs – solve the problem by dividing into modules, functions or classes. The example is shown in Figure 1.12.
2. Debugging programs – task of debugging large programis reduced to small modular program.
3. Reading programs- easier to understand compared to large program
4. Modifying programs – reduce large modification by concentrating on modules
5. Eliminating redundant code – by calling the modules will avoid the same code to be written multiple times

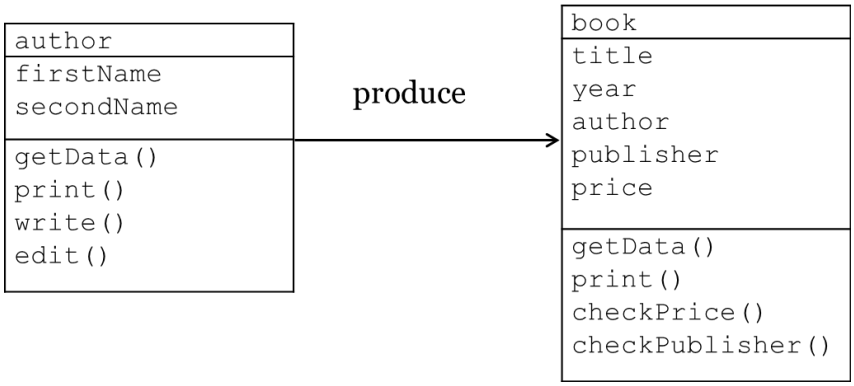

Figure 1.12 Modularity example with two classes

5.3. Style

1. Use of private data members – hide data members from modules – information hiding.
2. Proper uses of reference arguments – pass by value / pass by reference.
3. Proper use of methods to reduce coupling.
4. Avoidance of global variables in modules – thru encapsulation.
5. Error handling – invalid input : action to handle.
6. Readability – code easy to follow.
7. Documentation – well documented

5.4. **Modifiability** - Program need to change after each iteration. Requires program to be written in a way that is easy to modify. Some example of modifiability through the use of:

1. Named constants

```
const int number = 200;
int scores[number];
```

2. The **typedef** statement

```
typedef float cpaStudent;
typedef long double cpaStudent;
```

5.5. Ease of Use

- In an interactive environment, the program should prompt the user for input in a clear manner.
- A program should always echo its input.
- The output should be well labeled and easy to read.

5.6. Fail-Safe Programming

- Fail-safe programs will perform reasonably no matter how anyone uses it.
- Test for invalid input data and program logic errors.
- Enforce preconditions.
- Check argument values.

5.7. Debugging

- An activity where programmer systematically check a program's logic to find where an error occurs
- Tools to use while debugging:
  - Single-stepping
  - Watches
  - Breakpoints
  - **cout** statements
  - Dump functions

5.8. Testing

Levels of testing

- Unit testing: Test methods, then classes
- Integration testing: Test interactions among modules
- System testing: Test entire program
- Acceptance testing: Show that system complies with requirements

Types of testing

- Open-box (white-box or glass-box) testing
  - Test knowing the implementation
  - Test all lines of code (decision branches, etc.)
- Closed-box (black-box or functional) testing
  - Test knowing only the specifications