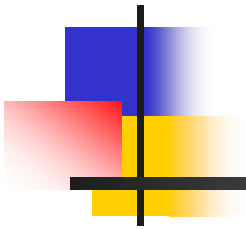
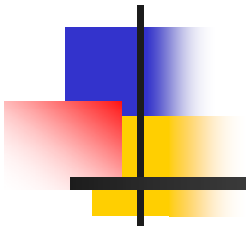


Introduction to Data Structures & Algorithm

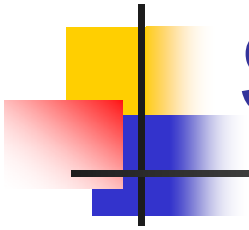
Objectives:



By the end of the class, students are expected to understand the following:

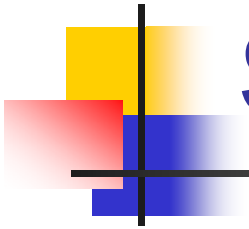
- data structure and algorithm concept
- programming development paradigm
- key programming principle

Algorithm & Data Structure



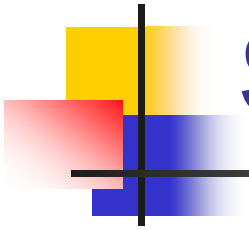
- Software engineering
  - Provides techniques to facilitate the development of computer program
- Problem solving
  - The entire process of taking the statement of a problem and developing a computer program that solves that problem
  - Requires to pass many phases, from understanding the problem, design solution and implement the solution.

Algorithm & Data Structure



- 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

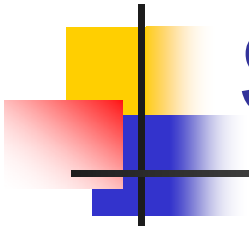
Algorithm & Data Structure



Challenges to create a good solution

- Create a good set of modules that
  - must store, move, and alter data
  - use algorithms to communicate with one another
- Organize your data collection to facilitate operations on the data in the manner that an algorithm requires

Algorithm & Data Structure



- Functions and methods implement algorithms
  - **Algorithm**: a step-by-step recipe for performing a task within a finite period of time
  - **Algorithms** often operate on a collection of data, which is stored in a structured way in the computer memory **(Data Structure)**
  - **Algorithms**: Problem solving using logic

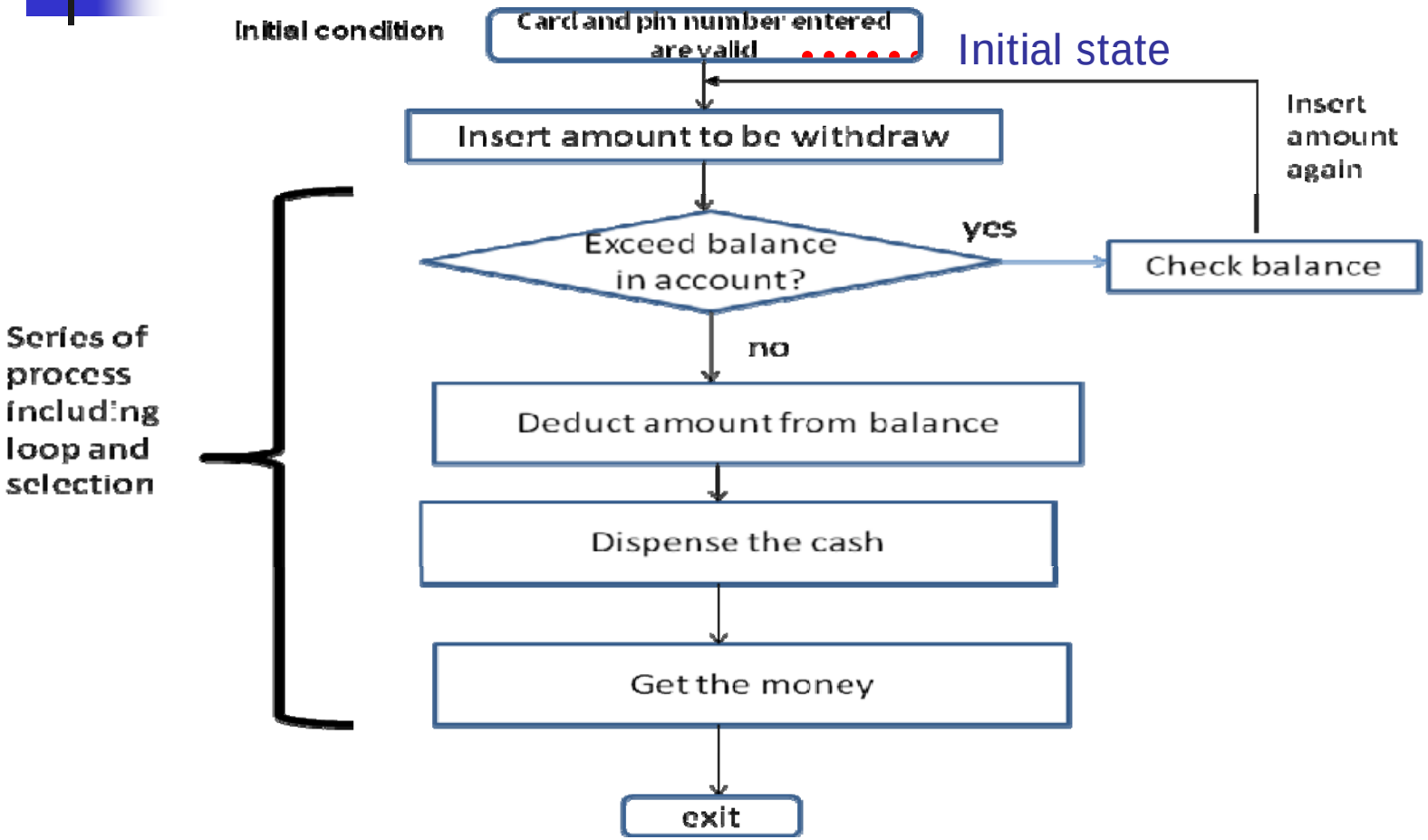
Algorithm & Data Structure

Algorithm

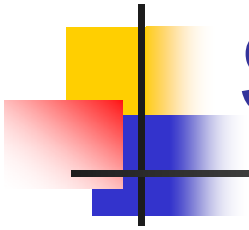
- a sequence of instructions, often used for calculation and data processing
- It is formally a type of effective method in which a list of well-defined instructions for completing a task will:
  - when given an initial state, (INPUT)
  - proceed through a well-defined series of successive states, (PROCESS)
  - eventually terminating in an end-state (OUTPUT)

Algorithm & Data Structure

Flowchart to withdraw money from ATM machine



Algorithm & Data Structure

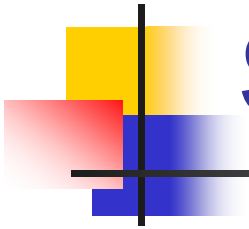


- **3 types of algorithm basic control structure**
  - Sequential
  - Selection
  - Repeatition (Looping)

Algorithm & Data Structure

- **Basic algorithm characteristics**
  - Finite solution (*ada penamat*)
  - Clear instructions (*jelas*)
  - Has input to start the execution
  - Has output as the result of the execution
  - Operate effectively (dilaksana dengan berkesan)
- Algorithm creation techniques
  - Flowchart, pseudo code,structure chart, language etc
- Factors for measuring good algorithm
  - Running time
  - Total memory usage

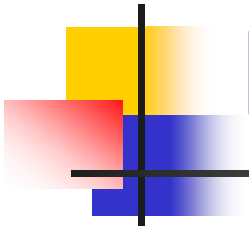
Algorithm & Data Structure



- **Data Structure**
  - a way of storing data in a computer so that it can be used efficiently – (this class – RAM only)
  - 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

Algorithm & Data Structure

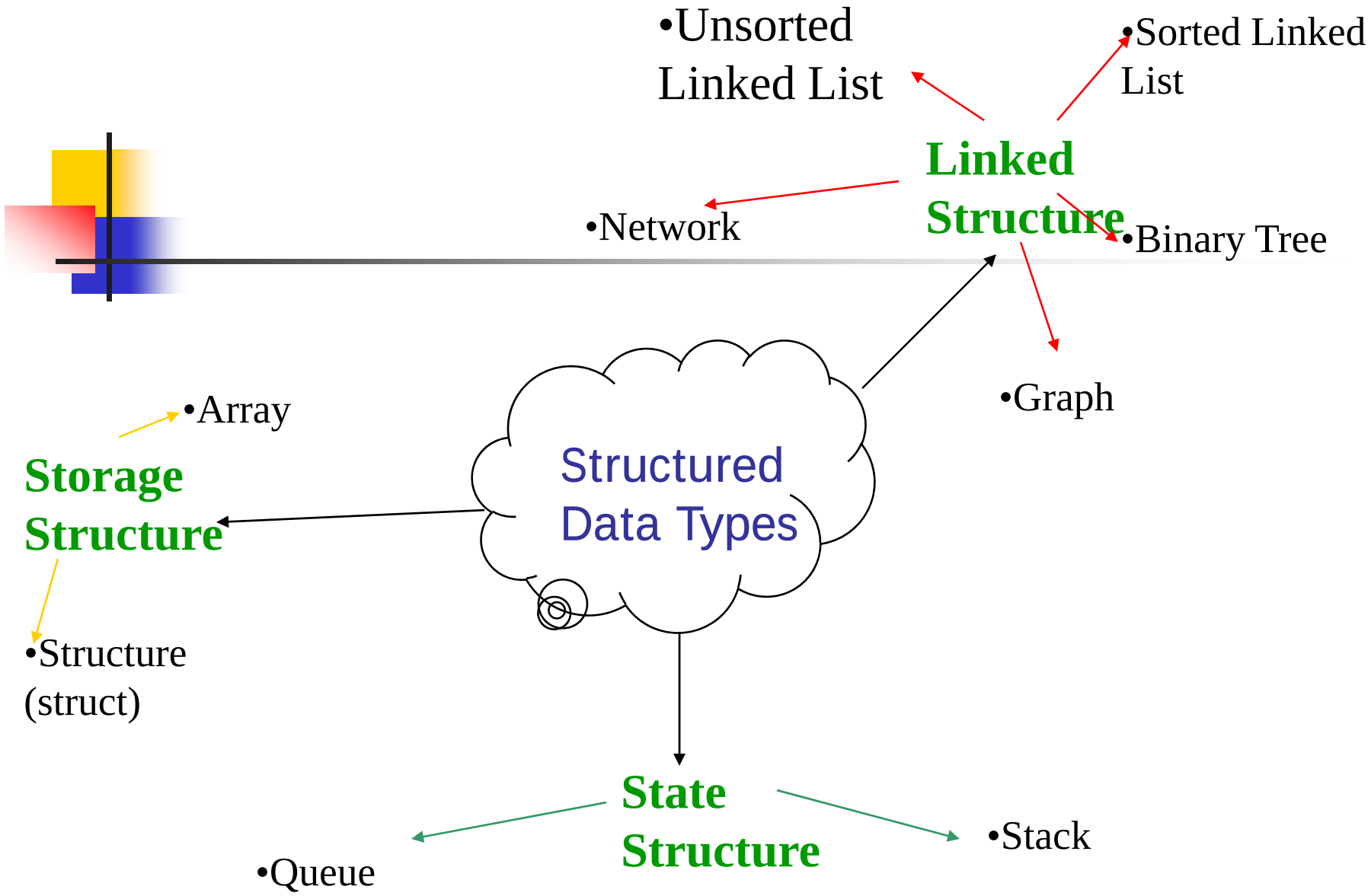
- **Operations to the Data Structure**
  - Traversing- access and process every data in data structure at least once
  - Searching – search for a location of data
  - Insertion – insert item in the list of data
  - Deletion - delete item from a set of data
  - Sorting – sort data in certain order
  -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

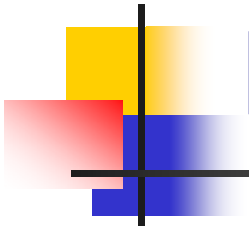


Data Types

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

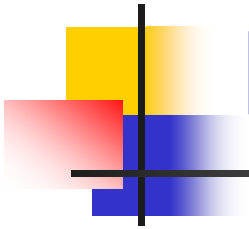

•Unsorted Linked List
•Sorted Linked List
Linked Structure
•Network
•Binary Tree
•Graph
•Array
Storage Structure
Structured Data Types
•Structure (struct)
State Structure
•Queue
•Stack



Data Types

- Structured Data Types
  - 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;
```

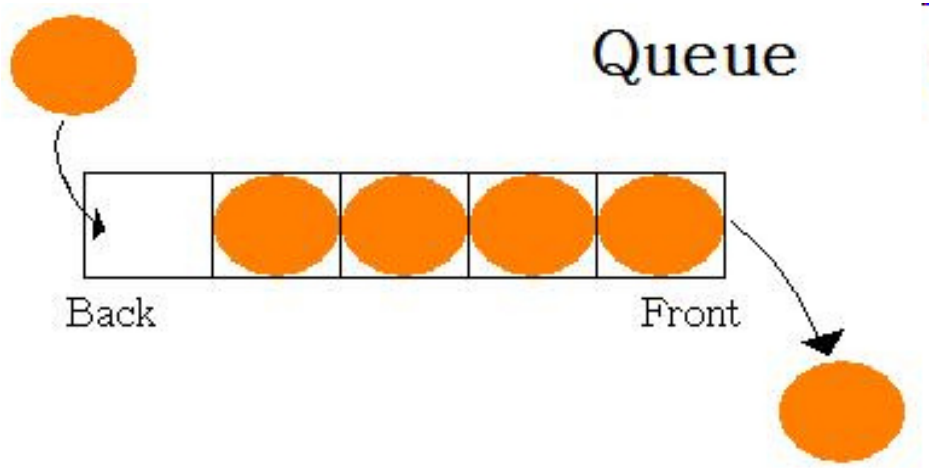


Data Types

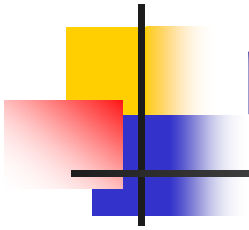
- Linked Data Structure
  - Linear Data Structure with restriction
    - Queue & Stack
  - Linear Data Structure with no restriction
    - Unsorted linked list
    - Sorted linked list
  - Non-linear Data Structure
    - Binary Tree
    - Graph

Linear Data Structure with restriction

- Queue
  - First-In-First-Out (FIFO) data structure
  - the first element added to the queue will be the first one to be removed (post office, bank etc)

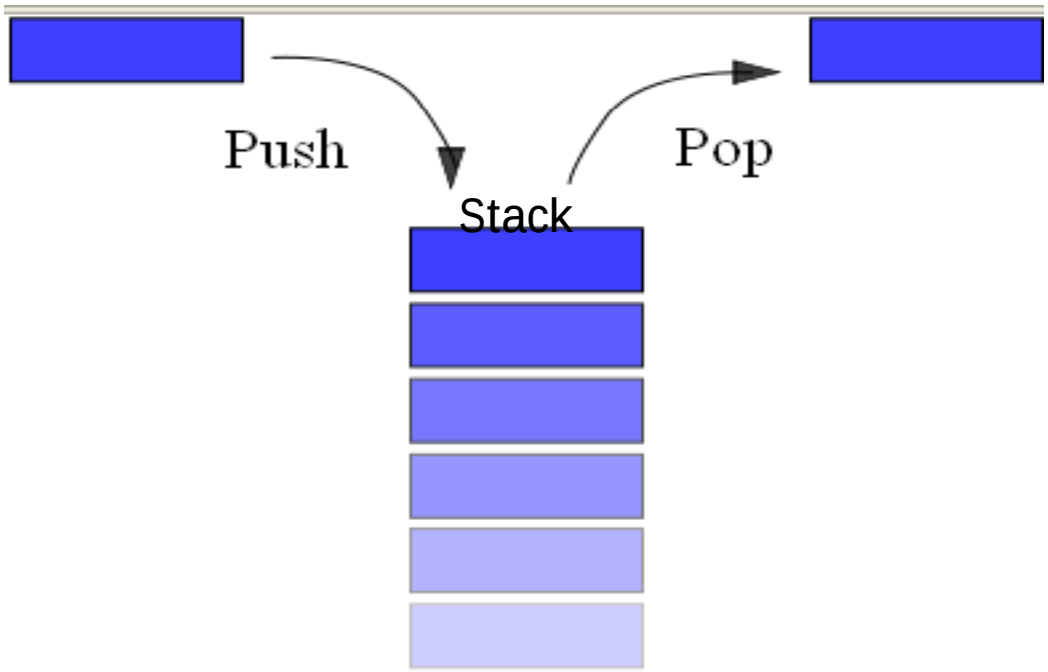


Linear Data Structure with restriction

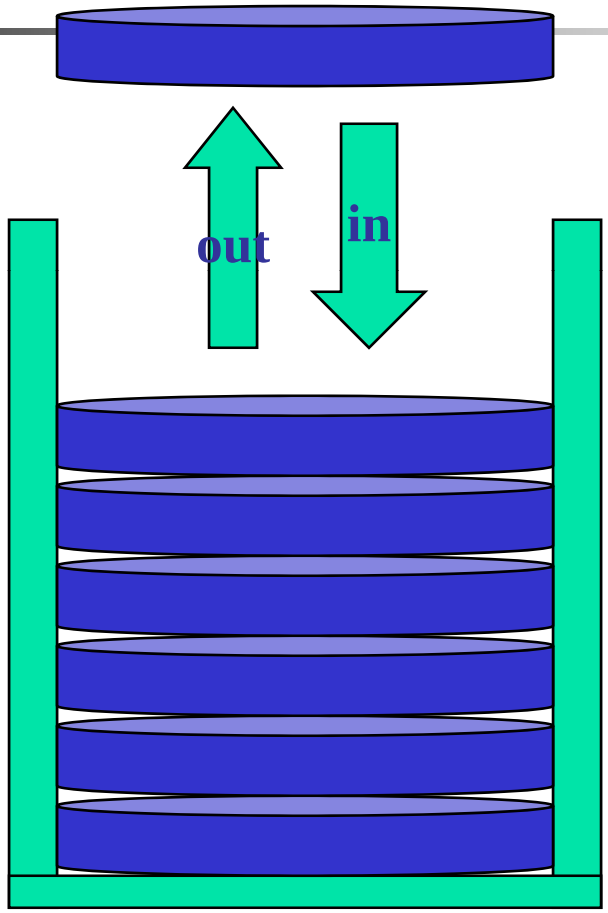


- Stack
  - Based on the principle of *Last In First Out (LIFO)*
  - Stacks are used extensively at every level of a modern computer system (compiler etc.)

Stack

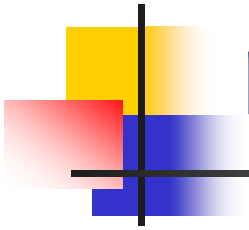


Stack



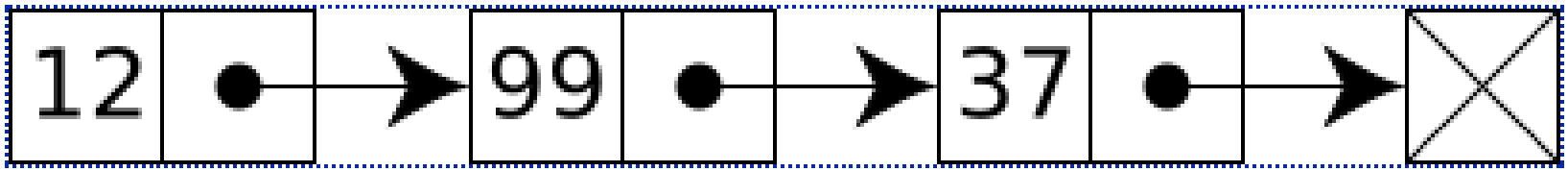
Last In First Out

Linear Data Structure with no restriction

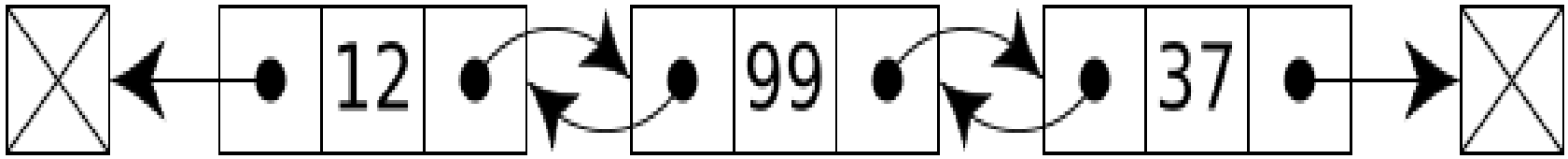


- Linked list
  - consists of a sequence of nodes, each containing arbitrary data fields and one or two references ("links") pointing to the next and/or previous nodes

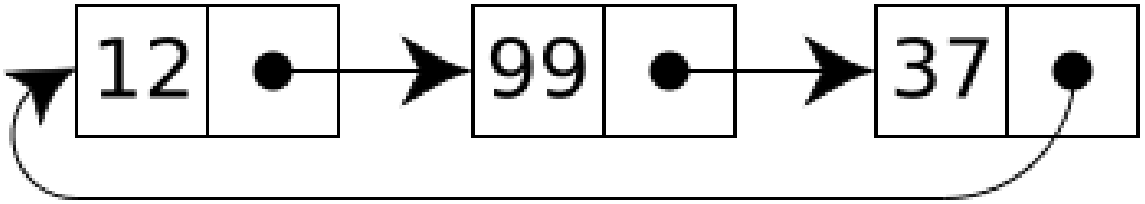
Linear Data Structure with no restriction



A singly-linked list containing two values: the value of the current node and a link to the next node

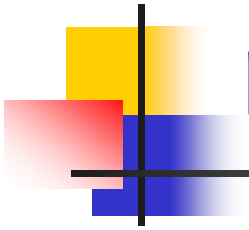


A doubly-linked list containing three integer values: the value, the link forward to the next node, and the link backward to the previous node

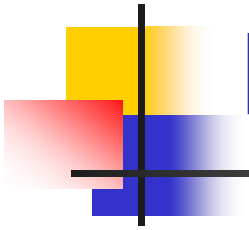


A circularly-linked list containing three integer values

Linear Data Structure with no restriction



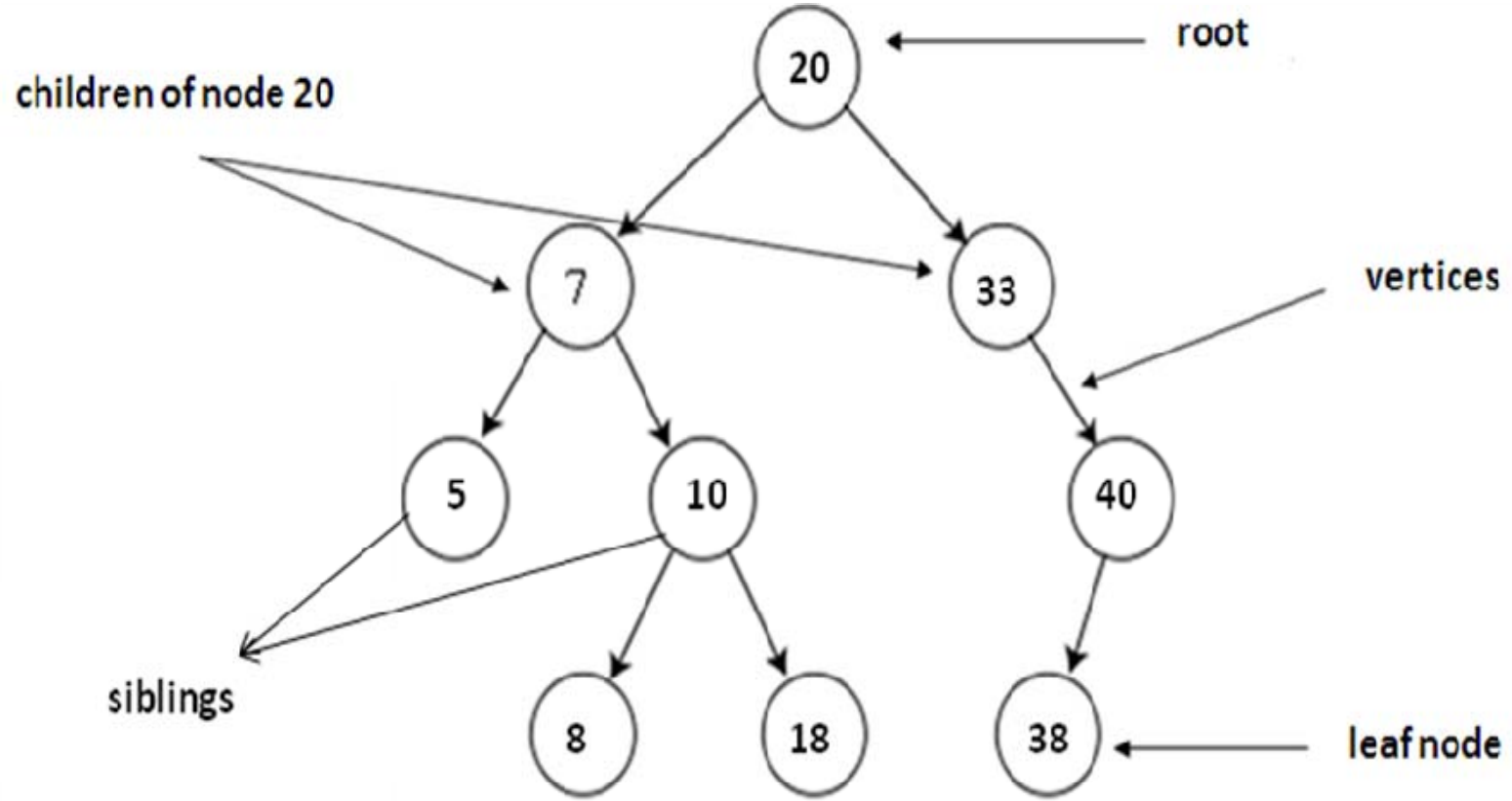
- Sorted linked list
  - Data stored in ascending or descending order with no duplicates
  - Insertion at front, middle or rear of the list
  - Deletion will not affect the ascending / descending order of the list
- Unsorted linked list
  - A linked list with no ordering

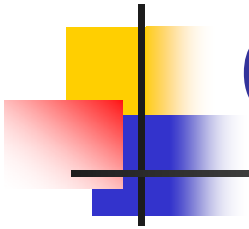


Non-linear Data Structure

- Binary Tree
  - A data structure based on a tree 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

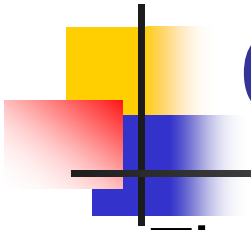
Tree


root
20
children of node 20
7
33
vertices
5
10
40
siblings
8
18
38
leaf node



Graph

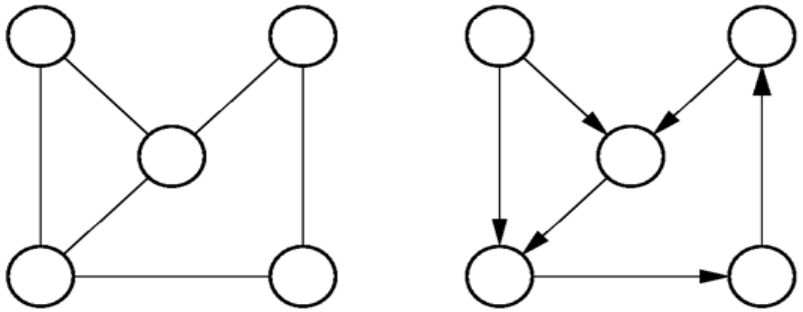
- A graph consists of a set of vertices, and a set of edges, such that each edge in is a connection between a pair of vertices.
- Some applications require visiting every vertex in the graph exactly once.



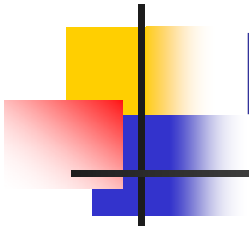
Graph

- 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.

Graph



Directed and undirected graph

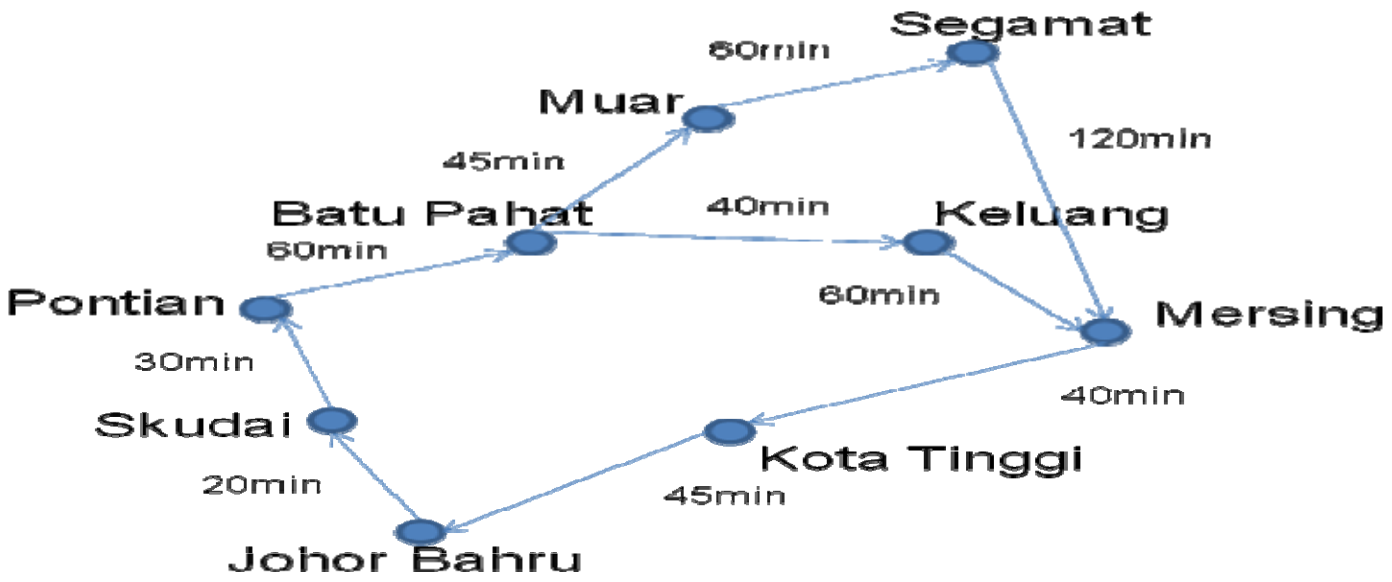


Network

- Network is a directed graph.
- Can be used to represent a route.
- Example :
  - A route for an airline.
  - A route for delivery vehicles.

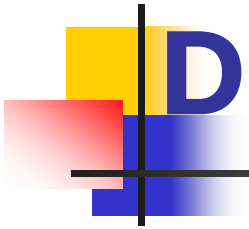
Network

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.

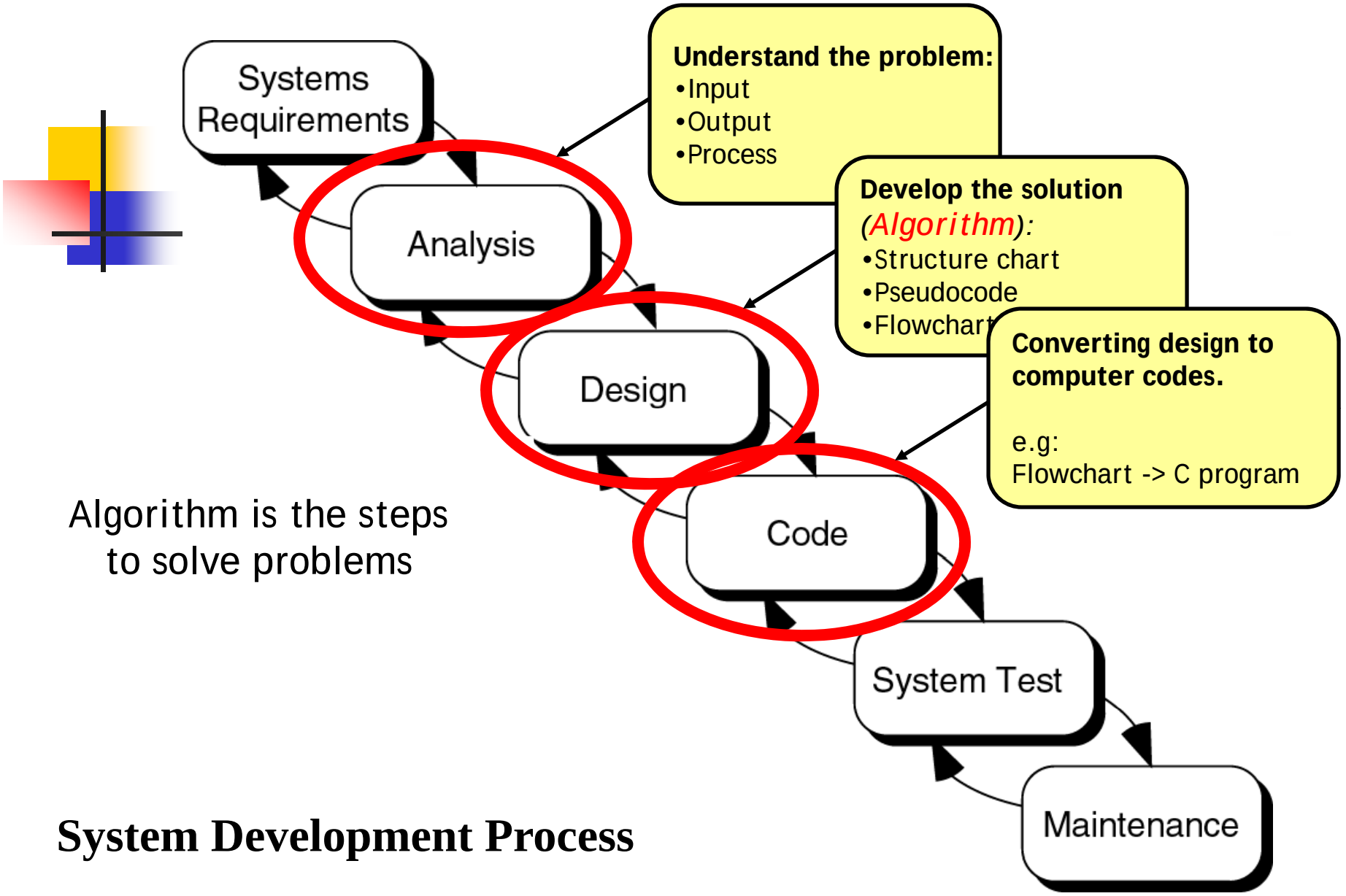


Programming Paradigm

Phases of Software Development



- Provide a very systematic and organized approach for developing software.



Algorithm is the steps to solve problems

System Development Process

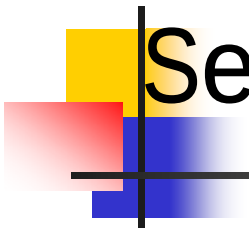
Phases of Software Development

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

Phases of Software Development

- **Design** – Develop details of the system by dividing into modules. Prepare algorithms for each modules.
- **Coding** – Transfer from design (algorithms) to computer source codes.
- **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.
- **Maintenance** – Monitor system performance; identify errors not detected during system testing, and enhancement to the system.

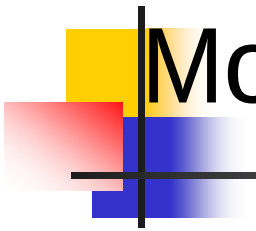
Programming Principle



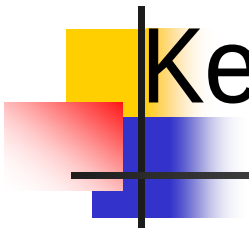
Seven Key Issues in Programming

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

Key Issues in Programming: Modularity



- Modularity has a favorable impact on
  - Constructing programs – small/large modules
  - Debugging programs – task of debugging large programis reduced to small modular program.
  - Reading programs- easier to understand compared to large program
  - Modifying programs – reduce large modification by concentrating on modules
  - Eliminating redundant code – by calling the modules will avoid the same code to be written multiple times



Key Issues in Programming: Style

1. Use of private data members –
   hide data members from modules – information hiding
2. Proper use 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

Key Issues in Programming: Modifiability

Program need to change after each iteration. Requires program to be written in a way that is easy to modify.

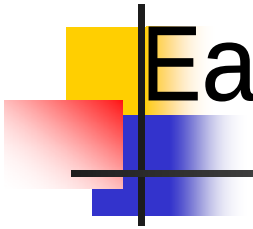
- Modifiability is easier through the use of
  - Named constants

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

  - The **`typedef`** statement

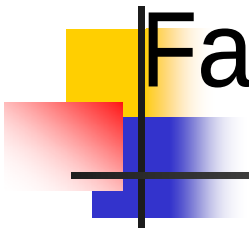
```
typedef float cpaStudent;
typedef long double cpaStudent;
```

Key Issues in Programming: Ease of Use



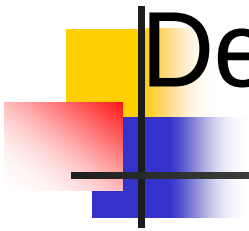
- In an interactive environment, the program should prompt the user for input in a clear manner
- A program should always echo its input
- The output should be well labeled and easy to read.

Key Issues in Programming: 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

Key Issues in Programming: Debugging

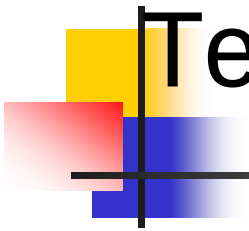


- Programmer must systematically check a program's logic to find where an error occurs
- Tools to use while debugging:
  - Single-stepping
  - Watches
  - Breakpoints
  - `cout` statements
  - Dump functions

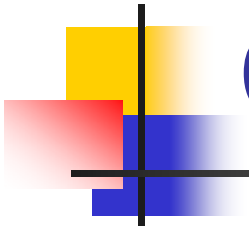
Key Issues in Programming: Testing

- Levels
  - 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

Key Issues in Programming: Testing



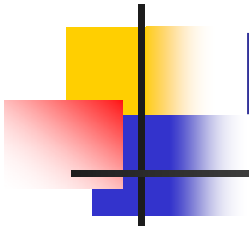
- Types
  - Open-box (white-box or glass-box) testing
    - Test knowing the implementation
    - Test all lines of code (decision branches, etc.)
  - Closed-box (black-box or functional) testing
    - Test knowing only the specifications



Conclusion

In this class you have learned about:

- Data structure and types of data stuructures
- Algorithm and its characteristics
- Programing principle
- System development process
  - The knowledge given is to ensure that you are able to provide good solution to problem solving



References

- Frank M. Carano, *Data Abstraction and problem solving with C++*.
- Nor Bahiah et al. *Struktur data & algoritma menggunakan C++. 2005, Penerbit UTM*