



**Crack**  
GATE/IES/PSUs & OTHER COMPETITIVE EXAMS *with*  
**arihant's**  
**Handbook Series**

# Computer Science & IT

— Highly focuses on —

Key Terms and their Definitions

Well Illustrated Formulae

Theoretical Analysis with Suitable Diagrams

Coverage of Key Points for Additional Info

*The only book to hold all the coverage in your hand*

**arihant's**  
**Handbook Series**

# **Computer Science & Information Technology**

**Highly focuses on**

Key Terms and their Definitions

Well Illustrated Formulae

Theoretical Analysis with Suitable Diagrams

Coverage of Key Points for Additional Info

**Surabhi Mishra**



ARIHANT PUBLICATIONS (INDIA) LIMITED



## Arihant Publications (India) Ltd.

All Rights Reserved

### ¤ © PUBLISHER

No part of this publication may be re-produced, stored in a retrieval system or distributed in any form or by any means, electronic, mechanical, photocopying, recording, scanning, web or otherwise without the written permission of the publisher. Arihant has obtained all the information in this book from the sources believed to be reliable and true. However, Arihant or its editors or authors or illustrators don't take any responsibility for the absolute accuracy of any information published and the damages or loss suffered there upon.

All disputes subject to Meerut (UP) jurisdiction only.

### ¤ ADMINISTRATIVE & PRODUCTION OFFICES

#### *Regd. Office*

'Ramchhaya' 4577/15, Agarwal Road, Darya Ganj, New Delhi -110002  
Tele: 011- 47630600, 43518550

#### *Head Office*

Kalindi, TP Nagar, Meerut (UP) - 250002  
Tele: 0121-7156203, 7156204

### ¤ SALES & SUPPORT OFFICES

Agra, Ahmedabad, Bengaluru, Bareilly, Chennai, Delhi, Guwahati,  
Hyderabad, Jaipur, Jhansi, Kolkata, Lucknow, Nagpur & Pune

### ¤ ISBN : 978-93-5094-430-1

Published by Arihant Publications (India) Ltd.

For further information about the books published by Arihant  
log on to [www.arihantbooks.com](http://www.arihantbooks.com) or email to [info@arihantbooks.com](mailto:info@arihantbooks.com)

Follow us on



# Preface

GATE-Gate Aptitude Test in Engineering offers a good opportunity to the students who are willing to seek admission to postgraduate courses and want to avail government scholarship for a further academic carrier in Engineering. In this era of competition, the engineering have gone to peak position. The engineers have a different innovation to do something new. So, there is lot of craze, an aspirant have to crack GATE and PSUs through many competition exams.

An aspirant would like to learn lot of things in short duration but in less volume with the help of single book, this quality which our handbook consists. This handbook is meant for an exhaustive and precise collection of all subjects that come under Computer Science and IT.

## ***They key features of this handbook are***

- Each topic is summarized in an exhaustive manner in the form of key points and notes.
- Every topic is taken up separately along with key points and notes.
- Focused material in entirety to prevent ambiguity in concepts.

I am thankful to **Arihant Publications (India) Limited** for giving me this opportunity to write such a book which covers almost 100% syllabus of GATE and PSUs and thus enlightens the path to your success.

I would like to thank **Er. Akash Shukla** (Project Coordinator) for giving me full support during this project. My good wishes to all the readers.

***Surabhi Mishra***

# Contents

<b>Theory of Computation</b>	<b>1-29</b>
• Basics of Theory of Computation	1      • Context-Free Language
	13     • Pushdown Automata
• Finite Automaton	3      • Parsing
	23     • Turing Machine
• Grammar	7      25
<b>Data Structure with Programming in C</b>	<b>30-74</b>
• Programming Concept in C Language	30     • Strings
	49     • Structures
• C Flow Control Statement	34     • Data Structures
	52     • Stack
• C Variable Types	39     • Queue
	53     • Linked List
• Storage Classes in C	40     • Tree
	55     • Graph
• Functions	43
• Pointers	46
• Array	47
<b>Design and Analysis of Algorithm</b>	<b>75-116</b>
• Introduction to Analysis of Algorithm	75     • Hashing
	96     • Collision
• Searching	80     • Tree
	99     100
• Sorting	84
<b>Database Management Systems</b>	<b>117-181</b>
• Database Management Systems	117     • SQL
	143     • Joins
• Schema	118     • Transaction Management
	148     151
• Relational Database Management Systems	126     • Concurrency Control
	153     • Schedule
• E-R Modeling	128     • File Structure
	154     • File Organisation
• Relational Algebra and Relational Calculus	133     170
	174
• Normal Forms/Normalization	139

<b>Operation System</b>		<b>182-217</b>	
• Operating System	182	• Deadlock	199
• Threads	186	• Memory Management	205
• Scheduling Algorithm	189	• File Management	213
<b>Computer Network</b>		<b>218-272</b>	
• Computer Network	218	• Switching	248
• OSI Model	227	• Routing and Network Security	253
• TCP/IP Protocol Suite	233	• Domain Name System(DNS)	264
• Flow Control	240	• Security Services	266
• Medium Access Control Sublayer	243	• Cryptography	269
		• RSA Algorithm	271
<b>Compiler Design</b>		<b>273-299</b>	
• Compiler	273	• Activation Tree	295
• Syntax Directed Translation	277	• Control Stack	296
• Symbol Tables	290	• Variable Scope	296
		• Displays	298
<b>Software Engineering and Information System</b>		<b>300-327</b>	
• Software Engineering	300	• Software Design	310
• Software Development Life Cycle (SDLC)	301	• Testing	314
• Software Requirements	305	• Software Measurement	318
• Software Risk Management	308	• COCOMO Model	323
		• Cyclomatic Complexity	324
		• Software Maintenance	327
<b>Web Technology</b>		<b>328-352</b>	
• HTML	328	• Client Server Model of Computing	345
• XML	341	• Scripting Language	347
• HTML Form	337		

## **Switching Theory and Computer Architecture**

**353-404**

• Number System	353	• Multiplexer	378
• Binary Codes	360	• Flip Flops	380
• Fixed Point Representation	363	• Counters	384
• Floating Point Representation	364	• Machine Instructions	386
• Boolean Algebra	365	• Addressing Modes	384
• Karnaugh Map	368	• Parallel Processing	397
• Combinational Circuit	370	• Pipelining	399
		• Memory Hierarchy	401

## **Appendix**

**405-416**

# 1

# Theory of Computation

---

## Basics of Theory of Computation

Computation is defined as any type of calculation. It is also defined as use of computer technology information processing. The theory of computation is the branch that deals with whether and how efficiently problems can be solved on a model of computation, using an algorithm.

### Basic Definitions

**Symbol** A symbol is an abstract entity *i.e.*, letters and digits.

**String** A string is a finite sequence of symbols.

**Alphabet** An alphabet is a finite set of symbols, usually denoted by  $\Sigma$ .

**Language** A formal language is a set of strings of symbols from some alphabet.

### Key Points

- ♦ Generally  $a, b, c, \dots$  used to denote symbols. Alphabets will represent the observable events of an automata.
- ♦ Generally  $w, x, y, z$  used to denote words. A word will represent the behaviour of an automation.

## Kleene Closure

If  $\Sigma$  is the set of alphabets, then there is a language in which any string of letters from  $\Sigma$  is a word, even the null string. We call this language **closure** of the alphabet. It is denoted by \* (asterisk) after the name of the alphabet is  $\Sigma^*$ . This notation is also known as the **Kleene Star**.

e.g.,                  If  $\Sigma = \{a\}$ , then

$$\Sigma^* = \{\lambda, a, aa, aaa, \dots\}$$

where,  $\lambda$  represents null string.

If                  If  $\Sigma = \{a, b\}$ , then

$$\Sigma^* = \{\lambda, a, b, aa, ab, bb, \dots\}$$

### Key Points

- ♦ By using Kleene Star operation, we can make an infinite language of strings of letters out of an alphabet.
- ♦ The words in increasing order of length called **lexicographic order**.

## Positive Closure

The '+' (plus operation) is sometimes called **positive closure**. A + (Plus) closure never contain null value.

If                  If  $\Sigma = \{a\}$ , then  $\Sigma^+ = \{a, aa, aaa, \dots\}$

**Note** If  $S$  is a set of strings, then  $S^+$  is the language  $S^*$  without the word  $\lambda$ .

## Operations Over Words in $\Sigma^*$

- **Concatenation** If  $x, y \in \Sigma^*$ , then  $x$  concatenated with  $y$  is the word formed by the symbols of  $x$  followed by the symbols of  $y$ . This is denoted by  $xy$ .
- **Substring** A string  $v$  is a substring of a string  $\omega$  if and only if there are strings  $x$  and  $y$  such that  $\omega = xvy$ .
- **Suffix** If  $\omega = xv$  for some string  $x$ , then  $v$  is suffix of  $\omega$ .
- **Prefix** If  $\omega = vy$  for some string  $y$ , then  $v$  is a prefix of  $\omega$ .
- **Reversal** Given a string  $\omega$ , its reversal denoted by  $\omega^R$  is the string spelled backwards.

e.g.,                   $(abcd)^R = dcba$

### Alphabet $\Sigma^*$

$\Sigma^*$  It is the set of all words for a given alphabet  $\Sigma$ . This can be described inductively in at least two different ways

- **Basic case** The empty word  $\lambda$  is in  $\Sigma^*$  (notation :  $\lambda \in \Sigma^*$ )

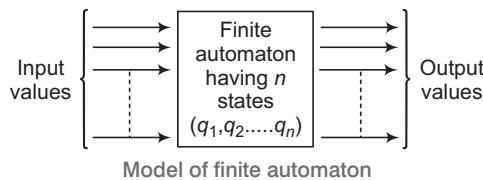
**Inductive step** If  $a \in \Sigma$  and  $x \in \Sigma^*$ , then  $ax \in \Sigma^*$

and also  $xa \in \Sigma^*$

- **Null set** The language that has no words and can be represented by  $\emptyset$ .

## Finite Automaton

A **Finite State Machine** (FSM) or finite state automaton is an abstract machine used in the study of computation and language that has only a finite, constant amount of memory.



### Description of Finite Automaton

This finite automaton is also known as **Deterministic Finite Automaton** (DFA). But when the transition function maps  $Q \times \Sigma \rightarrow 2^Q$ , ( $2^Q$  is called **powerset function**, is used then that automaton is known as **Non-deterministic Finite Automaton** (NDFA or NFA).

A finite automaton is defined as 5-tuples  $(Q, \Sigma, \delta, q_0, F)$

where,  $Q$  is finite non-empty set of states.

$\Sigma$  is finite non-empty set of inputs called alphabets.

$\delta$  is transition function which maps  $Q \times \Sigma$  into  $Q$ .

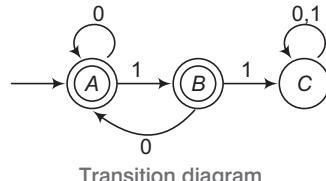
$q_0$  is initial state and  $q_0 \in Q$ .

$F$  is set of final states and  $F \subseteq Q$ .

## Transition Diagrams

A transition diagram is a finite directed labelled graph in which each vertex represents a state and directed edges indicate the transition from one state to another. Edges are labelled with input/output. In this, the initial state is represented by a circle with an arrow towards it, the final state is represented by two concentric circles and intermediate states are represented by just a circle.

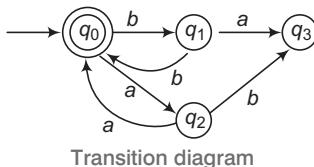
e.g., In the given transition diagram, vertex A is the initial state of the finite automata and vertices A and B are the final states and all the edges of this transition diagram are labelled with the inputs.



## Transition Table

Transition table is the tabular representation of transition system. In this representation, initial (start) state is represented by an arrow towards it and a final state is represented by a circle.

**Transition Table**



States	Inputs	
	a	b
$q_0$	$q_2$	$q_1$
$q_1$	$q_3$	$q_0$
$q_2$	$q_0$	$q_3$

Transition table

In this transition table, we have mentioned only three states  $q_0$ ,  $q_1$  and  $q_2$ ,  $q_3$  is not mentioned because  $q_3$  state is not reachable to any other node.

## Key Points

- Transition function is usually represented by a transition table or a transition diagram.
- Finite automata are finite collections of states with transition rules that take input to move from one state to another.
- The machine starts in the start state (initial state) and reads in a string of symbols from its alphabet. It uses the transition function  $\delta$  to determine the next state using the current state and the symbol just read. If, when it has finished reading, it is in an accepting state, it is said to accept the string, otherwise it is said to reject the string.

## Acceptability of a String by Finite Automata

A string  $\omega$  is accepted by a finite automaton,  $M = (Q, \Sigma, \delta, q_0, F)$ , if  $\delta(q_0, \omega) = F$ , where  $q_0$  is initial (start) state and  $F$  is the final state.

**Note** FA (Finite Automata) must accept those strings which are in the given language but should not accept those strings which are not in the language of finite automata.

**Transition Function** It takes two arguments i.e., a state and an input symbol.  $\delta(q, a)$  is the transition function for the DFA (Deterministic Finite Automata) which is at the state  $q$  and when it receives the input  $a$ , DFA will move to the next state.

**Extended Transition Function** It takes two arguments a state and an input string.

$$\delta^*(q, \omega) = \delta(\delta(q, x), a)$$

where,  $\omega$  is a string i.e.,  $\omega = xa$ , in which  $a$  is a single word and  $x$  is remaining string except the last symbol.

## Key Points

- ♦ The finite automata has only string pattern recognising power.
- ♦ Generally, all FA having only one reading head on the input tape, but a FA having more than one reading head has equal power as FA with one reading head.
- ♦ A language  $L \in E^*$  is regular if and only if there is a FA that recognises  $L$ .

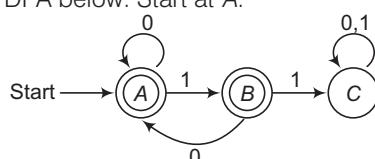
## Language of a DFA

An automata of all kinds defined languages. If  $A$  is an automaton,  $L(A)$  is its languages.

For a DFA  $A$ ,  $L(A)$  is the set of strings labelling paths from the start state to a final state. Formally,  $L(A)$  is the set of strings  $\omega$  such that  $\delta(q_0, \omega)$  is in  $F$  (final state or set of final states).

e.g., String 101 is in the language of the DFA below. Start at  $A$ .

Follow arc labelled 1, then arc labelled 0 from current state  $B$ . Finally, arc labelled 1 from current state  $A$ . Result in an accepting state, so 101 is in the language.



Transition diagram of a string 101

The language of our example DFA is

$$L = \{ \omega \mid \omega \text{ is in } \{0, 1\}^* \text{ and } \omega \text{ does not have two consecutive 1's} \}$$

- Read a set former as ‘the set of string  $\omega$ , such that these conditions about  $\omega$  are true

### Equivalence of DFA and NDFA

In contrast to the NFA (NDFA), the Deterministic Finite Automata (DFA) has

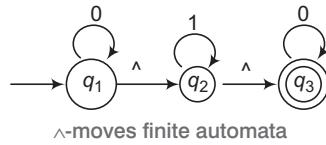
- No  $\lambda$ -transition (null transition).
- For every  $(q, a)$  with  $q \in Q$  and  $a \in \Sigma$  at most one successor state.
- A deterministic finite automata can simulate the behaviour of NFA by increasing the number of states.

### Key Points

- In Deterministic Finite Automata (DFA), for each state, there is at most one transition for each possible input.
- In non-deterministic finite automata, there can be more than one transition from a given state for a given possible input.
- If a language  $L$  is accepted by an NFA, then there is a DFA that accepts  $L$ .
- All DFA are NDFA, but not all NFA are DFA.
- All NDFA and DFA have the same power.
- Processing of an input string is more time consuming when NFA is used and less time consuming when DFA is used.

### Finite Automata having $\wedge$ -moves

The model NFA include those transitions by which, without giving any input FA can move to the next state.



### Finite Automata with Output Capabilities

The finite automata was explained in above sections have binary output i.e., either they accept the string or they do not accept the string.

*Under finite automata, there are two other machines*

### Moore Machine

It is a finite automata in which output is associated with each state. Each state of Moore machine has a fix output.

Moore machine is defined as 6-tuples  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ , where,

- $Q$  is finite non-empty set of states.
- $\Sigma$  is set of input symbols.
- $\Delta$  is output alphabet.
- $\delta$  is transition function which maps  $\delta(\Sigma \times Q) \rightarrow Q$ .
- $\lambda$  is output function which maps  $Q$  into  $\Delta$ .
- $q_0$  is initial state.

### Mealy Machine

It is a finite automata in which the output depends upon the present input and present state, both.

Mealy machine is also a 6-tuple  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ , where all symbols except  $\lambda$  have the same meaning as in Moore machine.  $\lambda$  is the output function mapping  $\Sigma \times Q$  into  $\Delta$ .

### Key Points

- ♦ We can construct equivalent Mealy machine for a Moore machine and vice-versa.
- ♦ If  $M_1$  and  $M_2$  are equivalent Moore and Mealy machines, respectively then two outputs  $T_1(\omega)$  and  $T_2(\omega)$  are produced by the Moore and Mealy machines  $M_1$  and  $M_2$  respectively, for input string  $\omega$ . Then, the length of  $T_1(\omega)$  is one greater than the length of  $T_2(\omega)$  i.e.,

$$|T_1(\omega)| = |T_2(\omega)| + 1$$

## Grammar

In 1956, Noam Chomsky gave a model of grammar. A grammar is defined as 4-tuples  $(V_N, \Sigma, P, S)$ ,

where,

- $V_N$  is finite non-empty set of non-terminals.
- $\Sigma$  is finite non-empty set of input terminals.
- $P$  is finite set of production rules.
- $S$  is the start symbol.

### Chomsky Hierarchy

The Chomsky hierarchy is a containment hierarchy of classes of formal grammars that generate formal languages. The formal grammars consist of a finite set of terminal symbols, a finite set of non-terminal symbols, a set of production rules with a left and a right hand side consisting of a word of these symbols and a start symbol.

The Chomsky hierarchy consists of following four levels

#### Type 0 Grammar (Unrestricted Grammar)

These are unrestricted grammars which include all formal grammars. These grammars generate exactly all languages that can be recognized by a Turing machine.

Rules are of the form  $\alpha \rightarrow \beta$ ,

where,  $\alpha$  and  $\beta$  are arbitrary strings over a vocabulary  $V$  and  $\alpha \neq \lambda$  (null).

#### Type 1 Grammar (Context Sensitive Grammar)

Languages defined by type-1 grammars are accepted by linear bounded automata.

Rules are of the form  $\alpha A\beta \rightarrow \alpha B\beta$

where,

$$A \in V_N$$

$$\alpha, \beta, B \in (V_N \cup \Sigma)^*$$

$$B \neq \lambda$$

#### Type 2 Grammar (Context-free Grammar)

Languages defined by type-2 grammars are accepted by push-down automata.

Rules are of the form  $A \rightarrow \alpha$ ,

where,

$$A \in V_N$$

$$\alpha \in (V_N \cup \Sigma)^*$$

#### Type 3 Grammar (Regular Grammar)

Languages defined by type-3 grammars are accepted by finite state automata.

Rules are of the form  $A \rightarrow \lambda$

$$A \rightarrow \alpha$$

$$A \rightarrow \alpha B$$

where,  $A, B \in V_N$  and  $\alpha \in \Sigma$ .

#### Comparison between Different Types of Classes

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing recognizable)	Turing machine
Type-1	Context sensitive	Context sensitive	Linear bounded
Type-2	Context-free	Context-free	Push down
Type-3	Regular	Regular	Finite

## Regular Expression

Regular expressions mean to represent certain sets of strings in some algebraic fashion.

*A regular expression over the alphabet E is defined as follows*

- $\emptyset$  is a regular expression corresponding to the empty language  $\emptyset$ .
- $\wedge$  is a regular expression corresponding to the language  $\{\wedge\}$ .
- For each symbol  $a \in E$ ,  $a$  is a regular expression corresponding to the language  $\{a\}$ .

## Regular Language

The languages accepted by FA are regular languages and these languages are easily described by simple expressions called regular expressions.

For any regular expression  $r$  and  $s$  over  $E$ , corresponding to the languages  $L_r$  and  $L_s$  respectively, each of the following is a regular expression corresponding to the language indicated.

- $(r s)$  corresponding to the language  $L_r L_s$ .
- $(r + s)$  corresponding to the lanugage  $L_r \cup L_s$ .
- $r^*$  corresponding to the language  $L_r$ .

*Some examples of regular expression are*

- |  |   |
|--|---|
| (i) $L(01) = \{0, 1\}$   | (ii) $L(01 + 0) = \{01, 0\}$                    |
| (iii) $L(0(1 + 0)) = \{01, 00\}$   | (iv) $L(0^*) = \{\epsilon, 0, 00, 000, \dots\}$ |
| (v) $L((0 + 10)^*(\wedge + 1)) = \text{all strings of } 0\text{'s and } 1\text{'s without two consecutive } 1\text{'s.}$ |   |

## Key Points

---

- ♦ If  $L_1$  and  $L_2$  are regular languages in  $E^*$ , then  $L_1 \cup L_2$ ,  $L_1 \cap L_2$ ,  $L_1 - L_2$  and  $L_1'$  (complement of  $L_1$ ), are all regular languages.
  - ♦ Pumping lemma is a useful tool to prove that a certain language is not regular.
  - ♦ The **Myhill-Nerode theorem** tells that the given language might be regular or not regular.
-

## Regular Set

A set represented by a regular expression is called **regular set**.  
e.g., If  $\Sigma = \{a, b\}$  is an alphabet, then

### Different Regular Sets and their Expressions

Sets	Regular Expression
{ }	$\phi$
{a}	$a$
{a, b}	$a + b$
{ $\epsilon$ , a, aa, aaa, ...}	$a^*$
{a, aa, aaa, ...}	$aa^* = a^+$

### Identities for Regular Expressions

The following points are the some identities for regular expressions.

- $\phi + R = R + \phi = R$
- $\wedge R = R \wedge = R$
- $R + R = R$ , where R is the regular expression.
- $(R^*)^* = R^*$
- $(P + Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$ , where P and Q are regular expressions.
- $R(P + Q) = RP + RQ$  and  $(P + Q)R = PR + QR$
- $P(QP)^* = (PQ)^*P$
- $\phi R = R\phi = \phi$
- $\wedge^* = \wedge$  and  $\phi^* = \epsilon$
- $RR^* = R^*R = R^+$
- $R^*R^* = R^*$

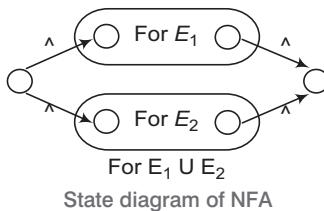
## Regular Expression and Finite Automata

NFA with  $\wedge$ -moves Let  $r_1$  and  $r_2$  be the two regular expressions over  $\Sigma_1, \Sigma_2$  and  $E_1$  and  $E_2$  are two NFA for  $r_1$  and  $r_2$ , respectively.

### Regular Expression to $\wedge$ -NFA (Union)

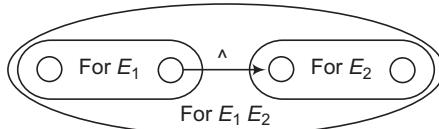
For

$$E_1 \cup E_2$$



### Regular Expression to $\wedge$ -NFA (Concatenation)

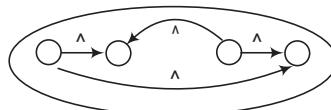
For

 $E_1 E_2$ 

State diagram of NFA

### Regular Expression to $\wedge$ -NFA (Closure)

For

 $E^*$ 

State diagram of NFA

### Construction of Regular Expression from DFA

If a DFA recognises the regular language  $L$ , then there exists regular expression, which describes  $L$ .

#### Arden's Theorem

- If  $P$  and  $Q$  are two expressions over an alphabet  $\Sigma$  such that  $P$  does not contain  $\wedge$ , then the following equation  $R = Q + RP$ .
- The above equation has a unique solution i.e.,  $R = QP^*$ . Arden's theorem is used to determine the regular expression represented by a transition diagram.

*The following points are assumed regarding transition diagrams*

- The transition system does not have any  $\wedge$ -move.
- It has only one initial (starting) state.

### Properties of Regular Language

*Regular languages are closed under following properties*

- |                      |                      |
|----------------------|----------------------|
| (i) Union            | (ii) Concatenation   |
| (iii) Kleene closure | (iv) Complementation |
| (v) Transpose        | (vi) Intersection    |
- If  $A$  is polynomial time reducible to  $B$  (i.e.,  $A \leq_p B$ ) and  $B$  is in  $P$ , then  $A$  is also in  $P$ .

- If A is NP-complete problem, then it is a member of P if and only if  $P = NP$ .
- The complexity class NP-complete is the intersection of NP and NP-hard classes.

### Key Points

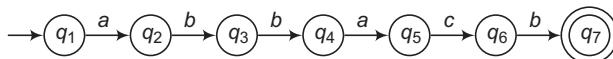
- All NP problems can be solved, if an NP problem can be reduced to an NP-hard problem before solving it in polynomial time.
- If A is NP-complete and there is a polynomial time reduction of A to B, then B is NP-complete.
- If there is a language L such that  $L \in P$ , then complement of L is also in P.
- P complexity class is closed under union, intersection, complementation, Kleene closure and concatenation.
- NP complexity class is closed under union, intersection, concatenation and Kleene closure.

## Design of Automaton

The design can be illustrated by under following three categories

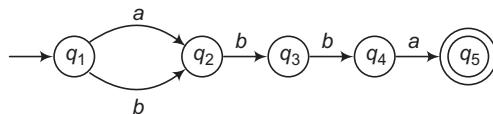
### Sequence of string is fixed

If the sequence of string is fixed that suppose an automaton is designed to accept a string with *abbacbc*.



### The value of string is changed by changing input value

Suppose, an automaton is designed to accept the string *abba* or *bbba*.



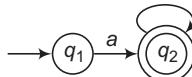
**Transition Table**

State	Input	
	a	b
$q_1$	$q_2$	$q_2$
$q_2$	—	$q_3$
$q_3$	—	$q_4$
$q_4$	$q_5$	—
$q_5$	—	—

### When input values are repeated

Suppose an automaton is to be designed for accepting all string generated by input  $a$ . That means it has to accept  $a, aa, aaa \dots$

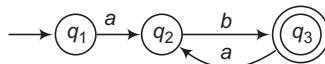
*This will be designed as*



**The Transition Table**

State	Input
	$a$
$q_1$	$q_2$
$q_2$	$q_2$

Now, if we have to design a machine which accepts all inputs containing string  $ab$



## Context-Free Language

In formal language theory, a context-free language is a language generated by some context-free grammar. The set of all context-free languages is identical to the set of languages accepted by push-down automata.

Finite automata accept all regular languages. Many simple languages are non-regular and there is no finite automata that accepts them.

e.g., The non-regular languages are like

$$\begin{aligned} &\{a^n b^n : n = 0, 1, 2, \dots\} \\ &\{w w^R : w \in \{a, b\}^*\} \end{aligned}$$

Context-free languages are a larger class of languages that encompasses all regular languages and many others, including the two above. But reverse is not true i.e., every context-free language is not necessarily regular.

## Context-Free Grammar (CFG)

A context-free grammar is defined as 4-tuples  $(\Sigma, V_N, P, S)$ , where,  $\Sigma$  is an alphabet (each character is  $\Sigma$  is called terminal).  $V_N$  is a set of non-terminals.

$P$  is the set of production rules, where every production has the form

$$A \rightarrow \alpha ; \text{ where, } A \in V_N, \alpha \in (V_N \cup \Sigma)^*$$

e.g., Here is a formal CFG for  $\{0^n 1^n : n \geq 1\}$

$$\text{Terminals} = \{0, 1\}$$

$$\text{Non-terminales} = \{S\}$$

$$\text{Start symbol} = S$$

$$\text{Productions} = S \rightarrow 0 1; S \rightarrow 0 S 1$$

**Note** The languages which are generated by context-free grammars are called Context-Free Language (CFL)

## Derivations

A derivation of a string is a sequence of rule applications. The language defined by a context-free grammar is the set of strings derivable from the start symbol  $S$  (for sentence).

## Definition

If  $v$  is one-step derivable from  $u$ , written  $u \Rightarrow v$ . If  $v$  is derivable from  $u$ , written  $u \xrightarrow{*} v$  if there is a chain of one derivations of the form.

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow v$$

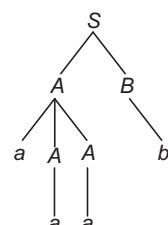
e.g., Consider the context free grammar

$$G = (\{s\}, \{0, 1\}, P, S)$$

- Productions
- (i)  $S \rightarrow 0 S 1$  or just only  $S \rightarrow 0 S 1 / \lambda$
  - (ii)  $S \rightarrow \lambda$

Derivations are

- |   |                             |
|---|-----------------------------|
| (a) $S \Rightarrow 0 S 1$ using (i)<br>$\Rightarrow 0 1$ using (ii)   | $S \Rightarrow \lambda$ (2) |
| (b) $S \Rightarrow 0 S 1$ using (i)<br>$\Rightarrow 0 0 S 1 1$ using (i)<br>$\Rightarrow 0 0 0 S 1 1 1$ using (i)<br>$\Rightarrow 0 0 0 1 1 1$ using (ii) |                             |



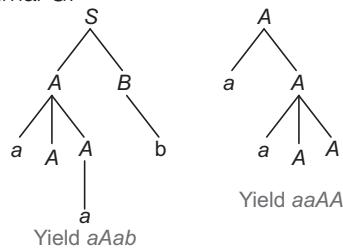
## Derivation Tree

A derivation tree (or parse tree) can be defined with any non-terminal as the root, internal nodes are non-terminals and leaf nodes are terminals. Every derivation corresponds to one derivation tree.

If a vertex  $A$  has  $k$  children with labels  $A_1, A_2, A_3, \dots, A_k$ , then  $A \rightarrow A_1 A_2 A_3 \dots A_k$  will be a production in context-free grammar  $G$ .

e.g.,

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAA \\ A &\rightarrow aA \\ B &\rightarrow bB \\ B &\rightarrow b \end{aligned}$$



### Derivation of a String

- Every derivation corresponds to one derivation tree.

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAAB \\ &\Rightarrow aaAB \\ &\Rightarrow aaaB \\ &\Rightarrow aaab \end{aligned}$$

- Every derivation tree corresponds to one or more derivations.

$$\begin{aligned} S &\Rightarrow AB \quad S \Rightarrow AB \quad S \Rightarrow AB \\ &\Rightarrow aAAB \Rightarrow Ab \quad \Rightarrow AB \\ &\Rightarrow aaAB \Rightarrow aAAb \Rightarrow aAAb \\ &\Rightarrow aaaB \Rightarrow aAab \Rightarrow aaAb \\ &\Rightarrow aaab \Rightarrow aaab \Rightarrow aaab \end{aligned}$$

## Left Most Derivation and Right Most Derivation

A derivation is left most (right most), if at each step in the derivation a production is applied to the left most (right most) non-terminal in the sentential form. In above three derivations, the first one is left most derivation and the second one is right most derivation, the third is neither.

## Ambiguous Grammar

A context-free grammar  $G$  is ambiguous if there is atleast one string in  $L(G)$  having two or more distinct derivation trees (or equivalently, two or more distinct left most derivations or two or more distinct right most derivations).

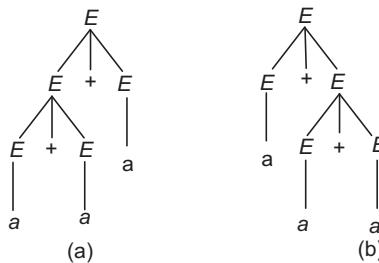
e.g., consider the context-free grammar  $G$  having productions  $E \rightarrow E + E / a$ . The string  $a + a + a$  has two left most derivations.

Let's see the derivations

$$E \Rightarrow E + E \Rightarrow E + E + E \Rightarrow a + E + E \Rightarrow a + a + E \Rightarrow a + a + a$$

$$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E + E \Rightarrow a + a + E \Rightarrow a + a + a$$

and the derivation trees are



### Potential Algorithmic Problems for Context-Free Grammars

- Is  $L(G)$  empty?
- Is  $L(G)$  finite?
- Is  $L(G)$  infinite?
- Is  $L(G_1) = L(G_2)$ ?
- Is  $G$  ambiguous?

## CFG Simplification

The four main steps will be followed in CFG simplification

- Eliminate ambiguity.
- Eliminate useless symbols productions.
- Eliminate  $\lambda$  productions :  $A \rightarrow \lambda$
- Eliminate unit productions :  $A \rightarrow B$

### Eliminate the Ambiguity

We can remove the ambiguity by removing the left recursing and left factoring.

### Left Recursion

A production of the context free grammar  $G = (V_N, E, P, S)$  is said to be left recursive if it is of the form

$$A \rightarrow A\alpha$$

Where,  $A$  is a non-terminal and

$$\alpha \in (V_N \cup E)^*$$

### Removal of Left Recursion

Let the variable A has left recursive productions as follows

$$(i) \quad A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 | \dots | A\alpha_n | \beta_1 | \beta_2 | \beta_3 | \dots | \beta_n$$

Where  $\beta_1, \beta_2, \dots, \beta_n$  do not begin with A. Then we replace A production in the form of

$$(ii) \quad A \rightarrow \beta_1 A^1 | \beta_2 A^1 | \dots | \beta_m A^1 \text{ where}$$

$$A^1 \rightarrow \alpha_1 A^1 | \alpha_2 A^1 | \alpha_3 A^1 | \dots | \alpha_n A^1 \wedge$$

### Left Factoring

Two or more productions of a variable A of the grammar  $G = (V_N, E, S, P)$  are said to have left factoring, if the productions are of the form

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n \text{ where } \beta_1, \dots, \beta_n (V_N \cup \Sigma)$$

### Removal of Left Factoring

Let the variable A has left factoring productions as follows

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n | y_1 | y_2 | y_3 | \dots | y_m$$

where,  $\beta_1, \beta_2, \dots, \beta_n$  have a common factor

$\alpha$  and  $y_1, y_2, \dots, y_m$  does not contain  $\alpha$  as a prefix, then we replace the production into the form as follows

$$A \rightarrow \alpha A^1 | Y_1 Y_2 | \dots | Y_M, \text{ where}$$

$$A^1 \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$$

### Eliminate the Useless Productions/Symbols

The symbols that cannot be used in any productions due to their unavailability in the productions or inability in deriving the terminals, are known as **useless symbols**.

e.g., consider the grammar G with the following production rules

$$S \rightarrow aS \mid A \mid C$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow ab$$

**Step 1** Generate the list of variables those produce terminal symbols

$$U = \{A, B, S\}$$

Because C does not produce terminal symbols so this production will be deleted . Now the modified productions are

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

**Step 2** Identity the variables dependency graph

$$S \rightarrow A B$$

In this graph,  $B$  variable is not reachable from  $S$  so it will be deleted also. Now the productions are  $S \rightarrow aS \mid A$

$$A \rightarrow a$$

### Eliminate Null Productions

If any variable goes to  $\lambda$  then that is called as **nullable variable**.

e.g.,  $A \rightarrow \lambda$  , then variable A is said to be nullable variable

**Step 1** Scan the nullable variables in the given production list.

**Step 2** Find all productions which does not include null productions.

$$\hat{p} = p \text{ (Null productions)}$$

e.g., consider the CFG has following productions-

$$S \rightarrow A B a C$$

$$A \rightarrow B C$$

$$B \rightarrow b \mid \lambda$$

$$C \rightarrow D \mid \lambda$$

$$D \rightarrow d$$

solve step find the nulable variables firstly the set is empty

$$N = \{\}$$

$$N = \{B, C\}$$

$$N = \{A, B, C\}$$

Due to  $B,C$  variables,

$A$  will also be a nullable variable.

**Step 3**  $\hat{p} = p \{ \text{Null productions} \}$

$$S \rightarrow B a C \mid A a C \mid A B a \mid a C \mid B a \mid A a \mid a$$

$$A \rightarrow B \mid C$$

$$B \rightarrow b$$

$$C \rightarrow D$$

$$D \rightarrow d$$

The above grammar is the every possible combination except  $\lambda$

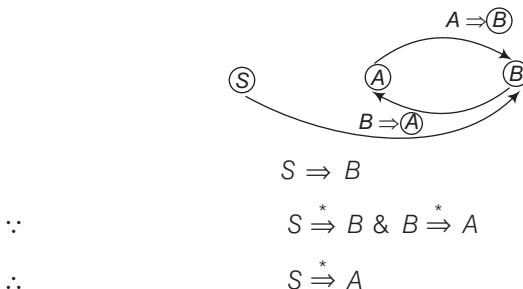
Now put this new grammar with original grammar with null.

$$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a$$

### **Eliminate the Unit-Productions**

A production of the type  $A \rightarrow B$ , where  $A, B$  are variables is called **unit productions**.

**Step 1** Using productions, we create dependency graph



**Step 2** Now the production without unit productions

$$S \rightarrow Aa \qquad \qquad S \rightarrow bb \mid a \mid bc$$

$$B \rightarrow bb \qquad + \qquad A \rightarrow bb$$

$$A \rightarrow a \mid bc \qquad \qquad B \rightarrow a \mid bc$$

Now the final grammar is

$$S \rightarrow Aa \mid bb \mid a \mid bc$$

$$B \rightarrow bb \mid a \mid bc$$

$$A \rightarrow a \mid bc \mid bb$$

### **Normal Forms of CFGs**

Ambiguity is the undesirable property of a context-free grammar that we might wish to eliminate. To convert a context-free grammar into normal form, we start by trying to eliminate null productions of the form  $A \rightarrow \lambda$  and the unit productions of the form  $B \rightarrow C$ .

*There are two normal forms*

1. Chomsky Normal Form (CNF)
2. Greibach Normal Form (GNF)

## Chomsky Normal Form (CNF)

A context-free grammar G is said to be in Chomsky Normal Form, if every production is of the form either  $A \rightarrow a$ , (exactly a single terminal in the right hand side of the production) or  $A \rightarrow BC$  (exactly two variables in the right hand side of the production).

e.g., the context-free grammar G with productions  $S \rightarrow AB, A \rightarrow a, B \rightarrow b$  is in Chomsky normal form.

### Chomsky Normal Form Properties

- The number of steps in derivation of any string  $\omega$  of length  $n$  is  $2n - 1$ , where the grammar should be in CNF.
- The minimum height of derivation tree of any  $\omega$  of length  $n$  is  $\lceil \log_2 n \rceil + 1$ .
- The maximum height of derivation tree of any  $\omega$  of length  $n = n$ .

## Greibach Normal Form (GNF)

A context-free grammar is said to be in Greibach Normal Form, if every production is of the form

$$A \longrightarrow a\alpha$$

where,  $a \in \Sigma, A \in V_N$  and  $\alpha \in V_N^*$

## Decision Algorithms for CFLs

**Theorem** There are algorithms to determine following for a CFL  $L$

**Empty** Is there any word in  $L$ ?

**Finite** Is  $L$  finite?

**Infinite** Is  $L$  infinite?

**Membership** For a particular string  $\omega$ , is  $\omega \in L$ ?

## Pumping Lemma for CFLs

The pumping lemma is used to prove that certain languages are not CFL.

## Closure Properties of CFLs

CFLs are closed under following properties

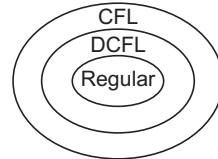
- |                  |                        |
|------------------|------------------------|
| • Union          | • Concatenation        |
| • Kleene closure | • Substitution         |
| • Homomorphism   | • Reverse homomorphism |

CFLs are not closed under following properties

- |                |                   |
|----------------|-------------------|
| • Intersection | • Complementation |
|----------------|-------------------|

## Deterministic Context-Free Language (DCFL)

The set of deterministic context-free languages is a proper subset of the set of context-free languages that possess an unambiguous context-free grammar.



CFL has subset of DCFL which has subset of regular

### Key Points

- ♦ The problem of whether a given context-free language is deterministic is undividable.
- ♦ Deterministic context-free languages can be recognized by a deterministic turning machine in polynomial time and  $O(\log^2 n)$  space.
- ♦ The language of this class have great practical importance in computer science as they can be passed much more efficiently then non deterministic context-free languages.

## Pushdown Automata (PDA)

A Pushdown Automata (PDA) is essentially an NFA with a **stack**. A PDA is inherently non-deterministic. To handle a language like  $\{a^n b^n \mid n \geq 0\}$ , the machine needs to remember the number of a's and b's. To do this, we use a stack. So, a PDA is a finite automaton with a stack. A stack is a data structure that can contain an number of elements but for which only the top element may be accessed.

### Definition of PDA

A Pushdown Automaton (PDA) is defined as 7-tuple.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$$

where,  $Q$  is a finite set of states

$\Sigma$  is the input alphabet

$\Gamma$  is the stack alphabet

$\delta$  is the transition function which maps

$$(Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})) = (Q \times (\Gamma \cup \{\epsilon\}))$$

$q_0$  is the start state and  $\epsilon$  denotes the empty string.

$q_0 \in Q$  is start state

$Z \in \Gamma$  is the intial stack symbol

$F \subseteq Q$  is the set of final or accepting states.

## Acceptance of PDA

Tape is divided into finitely many cells. Each cell contains a symbol in an alphabet  $\Sigma$ . The stack head always scans the top symbol of the stack as shown in figure.

*It performs two basic operations*

**Push** add a new symbol at the top.

**Pop** read and remove the top symbol.

$$\delta(q, a, v) = (p, u)$$

It means that if the tape head reads input  $a$ , the stack head read  $v$  and the finite control is in state  $q$ , then one of the possible moves is that the next state is  $p$ ,  $v$  is replaced by  $u$  at stack and the tape head moves one cell to the right.

$$\delta(q, \lambda, v) = (p, u)$$

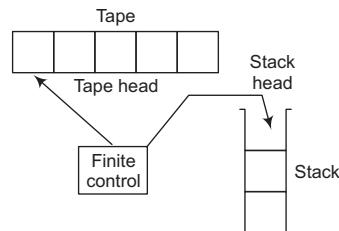
It means that this is a  $\lambda$ -move (null move)

$$\delta(q, a, \lambda) = (p, u)$$

It means that a push operation performs on stack.

$$\delta(q, a, v) = (p, \lambda)$$

It means that a pop operation performs on stack.



Push and pop of PDA

## Key Points

- ♦ In computer science, a pushdown automata is a type of automata that employs a stack.
- ♦ The PDA is used in theories about what can be computed by machine.
- ♦ PDA is more capable than a finite-state machine but less capable than a turing machine.

## Non-deterministic PDA

Like NFA, Non-deterministic PDA (NPDA) has number of choices for its inputs. An NPDA accepts an input, if sequence of choices leads to some final state or causes PDA to empty its stack.

## Deterministic PDA

Deterministic PDA (DPDA) is a pushdown automata whose action is a situation is fully determined rather than facing a choice between multiple alternative actions. DPDA cannot handle languages or grammars with ambiguity. A deterministic context-free language is a language recognised by some deterministic pushdown automata.

*Following languages are DCFLs*

- $L = \{a^n b^n : n \geq 0\}$
- $L = \{\omega c \omega^R : \omega \in (a+b)^*\} \text{ but not } L = \{\omega \omega^R : (a+b)^*\}$

## Key Points

- ♦ For any context-free language  $L$  there exist an NPDA in such form  $L = L(m)$
- ♦ If  $L = L(m)$  for some NPDA  $m$  then  $L$  is a context-free language.
- ♦ The family of context-free language is closed under union concatenation and star closure ( $L^*$ ). Means for once input  $b$  is arrived there should be state change to ensure that. Only input  $b$  is acceptable.
- ♦ To ensure that  $a$  is equal to  $b$  we will push all  $a$  to stack and after reading  $b$  we will be pop one  $a$  with one  $b$ .

# Parsing

Parsing is a technique to construct a parse (derivation) tree or to check whether there is some left most derivation or right most derivation.

*Parsing may be classified as*

## Top-down Parsing

In this parsing, start symbol is root, terminals are leaves (inputs symbols) and other nodes are variables. We start from the root and replacing the intermediate nodes one by one from left to right reach the leaves. This approach is also known as **recursive descent parsing** or **predictive descent parsing**.

LL parser is a top-down parser for a subset of the context-free grammars. It parses the input from left to right and constructs a left most derivation of the sentence. LL parser is called an LL( $k$ ) parser, if it constructs a left most derivation by looking  $k$  symbols ahead.

LL parser consists of

- An input buffer, holding the input string.
- A stack on which to store the terminals and non-terminals from the grammar yet to be parsed.
- A parsing table which tells it what grammar rule to apply given the symbols on top of its stack and the next input token.

## Bottom-up Parsing

Bottom-up parser reads the input from left and uses right most derivation in reverse order. Another name of bottom-up parser is **shift reduce parser**.

## LR Grammar

For grammar to be LR, it is sufficient that a left to right shift reduce parser should able to recognize handles when they appear on top of the stack when LR parser is implemented by using a stack. A grammar that can be parsed by an LR parser examining upto ' $k$ ' input symbols ( $k$  look ahead symbols) on each move is called an LR ( $k$ ) grammar.

### Points to be Remembered

- For every regular set, there exists a CFG  $G$  such that  $L = L(G)$ .
- Every regular language is a CFL.
- Let  $G_1$  and  $G_2$  be context-free grammars. Then,  $G_1$  and  $G_2$  are equivalent if and only if  $L(G_1) = L(G_2)$ .
- The intersection of a context-free language and a regular language is a context-free language.
- The reverse of a context-free language is context-free.
- A DFA can remember only a finite amount of information whereas a PDA can remember an infinite amount of information.
- For every PDA, there is a context-free grammar and for every context-free grammar, there is a PDA.
- If  $L_1$  is a DCFL and  $L_2$  is regular then,  $L_1 \cup L_2$  is also DCFL.
- If  $L_1$  is a DCFL and  $L_2$  is a regular language, then  $L_1 \cap L_2$  is also DCFL.
- Every regular language is DCFL.
- The power of non-deterministic pushdown automata and deterministic pushdown automata is not same. But the power of non-deterministic pushdown automata and deterministic pushdown is same.
- A FSM (Finite State Machine) with one stack is more powerful than FSM without stack.
- If left recursion or left factoring is present, it is not sure that the grammar is ambiguous but there may be chance of ambiguity.



## Difference between LL and LR

There is a significant difference between LL and LR grammars. For a grammar to be LR ( $k$ ), we must be able to recognise the occurrence of the right side of production having seen all of what is derived from right side with ' $k$ ' input symbols of look ahead. This requirement for LL ( $k$ ) grammars is quite different, where we must be able to recognise the use of a production looking only the first ' $k$ ' symbols of what its right side derives.

### LR ( $k$ )

L stands for left to right scanning.  
R stands for right most derivation.

### LL ( $k$ )

L stands for left to right scanning.  
L stands for left most derivation.

# Turing Machine

The Turing Machine (TM) was invented by Alan Turing in 1936. Turing machines are ultimate model for computers and have output capabilities. The languages accepted by Turing machine are said to be recursively enumerable. A Turing Machine (TM) is a device with a finite amount of read only hard memory (states) and an unbounded amount of read/write tape memory. There is no separate input. Rather, the input is assumed to reside on the tape at the time when the TM starts running.

- Recursive languages are closed under complementation, union, intersection, concatenation and Kleene closure.
- A Turing machine is said to be partially decide a problem, *if the following two conditions are satisfied*
  - (i) The problem is a decision problem.
  - (ii) The Turing machine accepts as given input if and only if the problem has an answer ‘yes’ for the input that is the Turing machine accepts the language  $L$ .
- A Turing machine is said to be decide a problem, if it partially decides the problem and all its computations are halting computations.
- Deterministic algorithms are treated as a special case of non-deterministic ones, so we can conclude that  $P \subseteq NP$ .
- The most famous unsolvable problem in computer theory is whether
$$P = NP \text{ or } P \neq NP$$
- A language  $L$  is called NP-hard language, if  $L_1 \leq_p L$  for every  $L_1 \in NP$ , where the notation  $\leq_p$  denotes the polynomial time reducibility relation.
- A language  $L$  is said to be NP-complete, if  $L \in NP$  and  $L$  is NP-hard.

## Universal Turing Machine (UTM)

- A UTM is a specified Turing machine that can simulate the behaviour of any TM.
- A UTM is capable of running any algorithm.
- For simulating even a simple behaviour, a Universal Turing Machine must have a large number of states. If we modify our basic model by increasing the number of read/write heads, the number of dimensions of input tape and adding a special purpose memory, then we can design a Universal Turing Machine.

## Definition of Turing Machine

A Turing Machine (TM) is defined as 7-tuples  $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$ , where,  $Q$  is a finite non-empty set of states.

$\Sigma$  is a non-empty set of input symbols (alphabets) which is a subset of  $\Gamma$  and  $b \notin \Sigma$ .

$\Gamma$  is a finite non-empty set of tape symbols.

$\delta$  is the transition function which maps  $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$

i.e., mapping from the present state of automata and tape symbol to next state, tape symbol and movements of head in left or right direction along the tape.

$q_0$  is the initial state and  $q_0 \in Q$ .

$b$  is the blank and  $b \in \Gamma$ .

$F$  is the set of final states and  $F \subseteq Q$ .

### Transition Function of a Turing Machine

The transition function  $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  states that if a turing machine is in some state (from set  $Q$ ), by taking a tape symbol (from set  $\Gamma$ ), it goes to some next state (from set  $Q$ ) by overwriting (replacing) the current symbol by another or same symbol and the read/write head moves one cell either left ( $L$ ) or right ( $R$ ) along the tape.

## Behaviour of Turing Machine

Depending upon the number of moves in transition, a TM may be deterministic or non-deterministic. If TM has at most one move in a transition, then it is called **Deterministic TM** (DTM), if one or more than one move, then **Non-deterministic TM** (NTM or NDTM).

- A non-deterministic TM is equivalent to a deterministic TM.
- Some single tape TM simulates every 2 PDA (a PDA with two stacks).
- The read only TM may be considered as a Finite Automata (FA) with additional property of being able to move its head in both directions (left and right).

## Language Recognition by Turing Machine

TM can be used as a language recogniser. TM recognises all languages, regular language, CFL, CSL, Type-0.

*There are several ways an input string might fail to be accepted by a Turing machine*

- It can lead to some non-halting configuration from which the Turing machine cannot move.

- At some point in the processing of the string, the tape head is scanning the first cell and the next move specifies moving the head left off the end of the tape.
- In either of these cases, we say that the Turing machine crashes.

## Variation of TM with other Automata

- **Multitape Turing Machine** A Turing machine with several tapes is said to be a multitape Turing machine. In a multitape Turing machine, each tape is controlled by its own independent read/write head.
- Turing machine with multiple tape is no more powerful than one tape Turing machine.
- **Multi-dimensional Turing Machine** A Turing machine is said to be multi-dimensional Turing machine, if its tape can be viewed as extending infinitely in more than one dimension.
- **Multihead Turing Machine** A multihead Turing machine can be viewed as a Turing machine with a single tape and a single finite state control but with multiple independent read/write heads.
- In one move, the read/write heads may take move independently left, right or remain stationary.
- **Offline Turing Machine** An offline Turing machine is a multitape Turing machine whose input tape is read only (writing is not allowed).
- An offline Turing machine can simulate any Turing machine A by using one more tape than Turing machine A. The reason of using an extra tape is that the offline Turing machine makes a copy of its own input into the extra tape and it then simulates Turing machine A as if the extra tape were A's input.

## Halting Problem of Turing Machine

A class of problems with two output (true/false) is called **solvable** (or decidable) problem, if there exists some definite algorithm which always halts (also called terminates), else the class of problem is called **unsolvable** (or undecidable).

## Linear Bounded Automata (LBA)

It is possible to limit the tape by restricting the way in which the tape can be used. The way of limiting the tape use is to allow the machine to use only that part of the tape occupied by the input. This way, more space is available for long input strings than for short strings, generating another class of machines called **Linear Bounded Automata (LBA)**.

## Recursive and Recursively Enumerable Languages

A language  $L$  is said to be recursively enumerable, if there exists a Turing machine that accepts it. A language is recursive if and only if there exists a membership algorithm for it. Therefore, a language  $L$  on  $\Sigma$  is said to be recursive, if there exists a Turing machine that accepts the language  $L$  and it halts on every  $\omega \in \Sigma^*$ . Recursively enumerable languages are closed under union, intersection, concatenation and Kleene closure and these languages are not closed under complementation.

### Key Points

- ♦ The complement of a recursive language is recursive.
- ♦ The union of two recursive languages is recursive.
- ♦ There are some recursively enumerable languages which are not recursive.
- ♦ If  $L$  is recursive then,  $L'$  is also recursive and consequently both languages are recursively enumerable.
- ♦ Every context sensitive language is recursive.
- ♦ The family of recursively enumerable languages is closed under union.
- ♦ If a language is not recursively enumerable, then its complements cannot be recursive.
- ♦ If a language  $L$  is recursive, then it is recursively enumerable language but vice-versa is not true.

## Decidable Problems

A problem is decidable, if there is an algorithm that answer either yes or no or we can say that there is a Turing machine that decides the problem.

## Undecidable Problems

A decision problem is undecidable, if a Turing machine may run forever without producing an answer.

## Halting Problems

A halting problem is undecidable problem. Alan Turing proved in 1936 that there is no general method or algorithm which can solve the halting problem for all possible inputs.

## Computability and Complexity

- What can be decided algorithmically is known as **computability**.
- What resources (time, memory, communication) are needed is known as **complexity**.

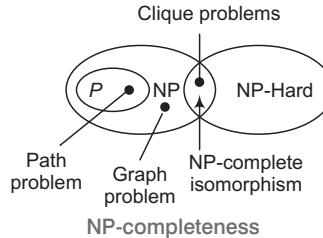
## Complexity Classes

- **P-Class** P is also known as PTIME or DTIME complexity. It contains all decision problems which can be solved by a deterministic Turing machine using polynomial amount of computation time.
- **NP-Class** NP refers to **non-deterministic polynomial time**. The complexity class NP, contains many problems that people would like to solve efficiently, but for which no efficient algorithm is known. P is subset of NP.
- **NP-Hard** A problem A is said to be NP-hard, if for every decision problem  $L$  in NP, there is an Oracle machine with an Oracle for A.

An **Oracle machine** is an abstract machine used to study decision problems. It can be visualised as a Turing machine with a black box, called an Oracle, which is able to decide decision problems in a single operation.

**Polynomial Time Reducibility** If  $L_1$  and  $L_2$  are two languages over  $\Sigma_1$  and  $\Sigma_2$  respectively, then  $L_1$  is said to be polynomial time reducible to  $L_2$  ( $L_1 \leq_p L_2$ ), if there is a function  $f : \Sigma_1^* \rightarrow \Sigma_2^*$  such that for any string  $x \in \Sigma_1^*$ ,  $x \in L_1$  if and only if  $f(x) \in L_2$  and  $f$  can be computed in polynomial time.

- **NP-Completeness** An NP-complete problem is one for which the corresponding language is NP-complete. A language  $L$  is NP-complete, if  $L \in NP$  and  $L$  is NP-hard.



## Some NP-Complete Problems

*Satisfiability problems are given below*

- The vertex cover problem.
- The travelling salesman problem.
- The simple max cut problem.
- The Hamiltonian cycle problem.
- The colorability problem.

## Key Points

- ♦ The set of all Turing machines, although infinite is countable.
- ♦ The halting problem of Turing machine is undecidable.
- ♦ Let  $M$  be any Turing machine, then the question of whether or not  $L(M)$  is undecidable.
- ♦ The post correspondence problem is undecidable.

# 2

# Data Structure with Programming in C

---

## Programming Concepts in C Language

C is a high level language. It is general as well as specific purpose language. It was developed at **Bell laboratory**, USA (now AT and T) in 1972, by Dennis Ritchie and Brian Kernighan.

To learn C language we must first know what alphabets, numbers and special symbols are used in C, then how to use them, contents, variables and keywords are constructed and finally, how are these combined to form an instruction.

### Character Set

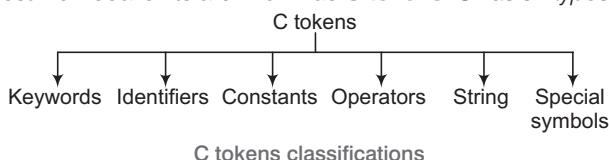
The characters that can be used to form words, numbers and expressions depend upon the computer on which the program runs.

*The characters in C are grouped into the following categories*

- |                          |                   |
|--------------------------|-------------------|
| (i) Letters              | (ii) Digits       |
| (iii) Special characters | (iv) White spaces |

## C Tokens

The smallest individual units are known as **C tokens**. C has six types of tokens.



## Keywords

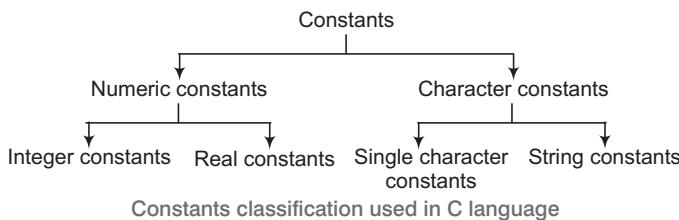
All keywords (i.e., reserved words) are basically the sequences of characters that have one or more fixed meanings. All C keywords must be written in lowercase letters. e.g., break, char, int, continue, default, do etc.

## Identifiers

Names given to the program elements such as variables, arrays and functions. Identifiers are sequences of alphabets and digits e.g., main, amount, emp\_id etc.

## Constants

Fixed values that do not change during the execution of a C program.



Backslash character constants are used in output functions. e.g., '\b' used for backspace and '\n' used for new line etc.

## Operator

It is symbol that tells computer to perform certain mathematical or logical manipulations. e.g., Arithmetic operators (+, -, \*, /) etc.

## String

- A string is nothing but an array of characters (printable ASCII characters).
- Special Symbols e.g., [ ], { , } etc.

## Key Points

- An integer and real constant must have at least one digit.
- No commas or blanks are allowed within an integer constant and real constants.

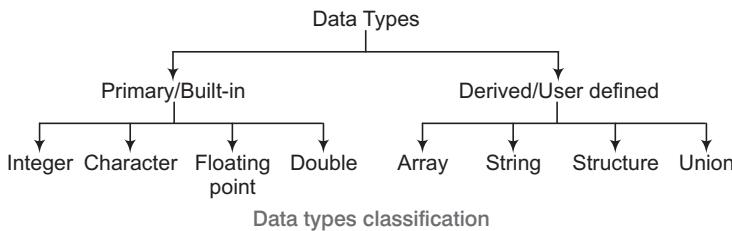
## Variable

A variable is used to store data value. A variable is a data name that may take different values at different times during execution.

### Comparison between Valid Variables and Invalid Variables

S.No.	Valid Variables	Invalid Variables
1.	Marks	6 pk
2.	Total_mark	Total marks
3.	Area_of_cube	Area--of--cube
4.	Num [10]	Ram's-Age
5.	Population_2006	Gross-salary-2009

### Data Types



### Key Points

- An operation between a real and real always yields a real result.
- An operation between an integer and real always yields a real result. In this, the integer is first promoted to a real and then the operation is performed.

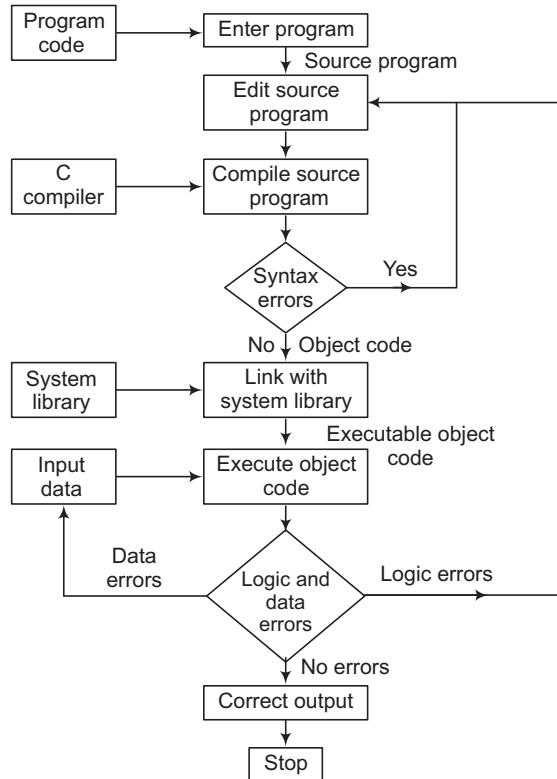
### Delimiters/Separators

These are used to separate constants, variables and statements e.g., comma, semicolon, apostrophes, double quotes and blank space etc.

### Different Types of Modifier with their Range

S.No.	Types of Modifier	Size (bytes)	Range of Values
1.	int	2	- 32768 to + 32767
2.	signed int	2	- 32768 to + 32767
3.	unsigned int	2	0 to 65535
4.	short int	2	- 32768 to + 32767
5.	long int	4	- 2147483648 to + 2147483647
6.	float	4	-(3.4 E + 48) to +(3.4 E + 48)
7.	double	8	-(1.7 E + 308) to (1.7E + 308)
8.	char	1	- 128 to 127
9.	unsigned char	1	0 to 255

### Flow Chart of Compiling and Running a C Program



### Key Points

- ♦ Syntax errors (*i.e.*, violation of grammar).
- ♦ Logical errors (*i.e.*, errors occur during coding process).
- ♦ Run-time errors (*i.e.*, errors occurs when we attempt to run the ambiguous instructions).

# C Flow Control Statements

Control statement is one of the instructions, statements or group of statement in a programming language which determines the sequence of execution of other instructions or statements.

C provides two styles of flow controls

1. Branching (deciding what action to take)
2. Looping (deciding how many times to take a certain action)

## Decision Making and Branching

C has three major decision making instructions as given below

### If Statement

It takes an expression in parenthesis and a statement or block of statements. Expressions will be assumed to be true, if evaluated values are non-zero.

- (i) Simple if statement

Syntax

```
if (expression)
{
    statements;
}
```

- (ii) If else statement

Syntax

```
if (test expression)
{
    block of statements;
}
else
{
    block of statements;
}
```

- (iii) If else-if ladder statement

Syntax

```
if (expression)
{
    statements;
}
```

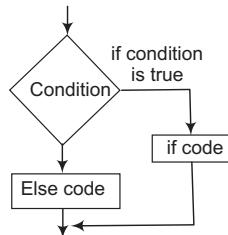
```

else if (expression)
{
    statements;
}
else
{
    statements;
}

```

**Note** C programming language assumes any non zero and non-null values as true and if it is either zero or null then it is assumed as false value.

### Flow Diagram of if-else Statement



### The switch Statement

The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.

```

switch (expression)
{
    case value 1:
        statement 1;
        break;
    case value 2:
        statement 2;
        break;
    :
    default:
        statement;
}

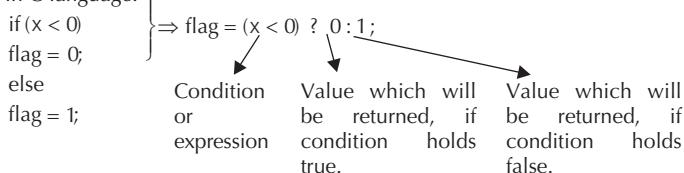
```

**Note** The break keyword transfers the control out of the switch statement.

### The Conditional Operators (? , :)

The ?, : operators are just like an if-else statement except that because it is an operator we can use it within expressions.

? : are a ternary operators in that it takes three values. They are the only ternary operator in C language.



## Loop Control Structure

Loops provide a way to repeat commands and control. This involves repeating some portion of the program either a specified numbers of times until a particular condition is satisfied.

*There are three types of loop*

### while Loop

initialize loop counter;

```

while (test loop counter using a condition/expression)
{
    do this;
    and this;
    decrement/increment loop counter;
}
  
```

### for Loop

```

for      (initialize      counter;      test      counter;
         increment/decrement counter)
{
    do this;
    and this;
    and this;
}
  
```

**do while Loop**

```
initialize loop counter;  
do  
{  
    this;  
    and this;  
}  
while (this condition is true);
```

**The break Statement**

The **break** statement is used to jump out of a loop instantly, without waiting to get back to the conditional test. A break is usually associated with an 'if'. When break is encountered inside any loop, control automatically passes to the first statement after the loop.

```
# include <stdio.h>  
void main ( )  
{  
    int num, i;  
    printf ("Enter a number");  
    scanf ("%d", & num);  
    i = 2;  
    while (i <= num - 1 )  
    {  
        if (num% i == 0)  
        {  
            printf ("Not a prime number");  
            break;  
        }  
        i++;  
    }  
    if (i == num)  
        printf("Prime number");  
}
```

**Note** When num% i turns out to be zero (i.e., num is exactly divisible by i) the message "Not a prime number" is printed and the control breaks out of the while loop.

## The continue Statement

The ‘continue’ statement is used to take the control to the beginning of the loop, by passing the statement inside the loop, which have not yet been executed.

```
# include <stdio.h>
void main ( )
{
    int i, j;
    for (i= 1; i<= 2; i++)
    {
        for (j = 1; j <= 2; j++)
        {
            if (i== j)
                continue;
            printf("\n %d%d", i, j);
        }
    }
}
Output 1 2
      2 1
```

*When the value of i equals to j, the ‘continue’ statement takes the control to the for loop (inner) by passing the rest of the statements pending for executions in the ‘for’ loop.*

## goto Statement

C supports an unconditional control statement, goto, to transfer the control from one point to another in a C program. The goto is a branching statement and requires a label.

```
# include (Stdio. h)
void main ()
{
    int i, j, k;
    for (i= 1; i<= 3; i++)
    {
        for (j= 1; j <= 3; j++)
        {
            for (k= 1; k <= 3; k++)
            {
                if (i== 2&&j == 2 &&k == 2)
```

```

        goto out;
    else
        printf ('%d%d%d \n' , i, j, k);
    }
}
}
ou t:
printf ('out of the loop at last !')
}

```

**Note** The usage of the goto keyword should be avoided as, it usually violates the normal flow of execution.

## C Variable Types

A variable is just a named area of storage that can hold a single value.  
There are two main variable types

1. Local variable
2. Global variable

### Local Variable

Scope of a local variable is confined within the block or function, where it is defined.

### Global Variable

Global variable is defined at the top of the program file and it can be visible and modified by any function that may reference it. Global variables are initialized automatically by the system when we define them. If same variable name is being used for global and local variables, then local variable takes preference in its scope.

```

#include<stdio.h>
#include<conio.h>
int i = 4; /*Global definition*/
main ( )
{
    i+= ; // This is global variable and will be incremented to 5
    func ( );
    printf ("value of i=% d... main function \n", i);
}
func ( )
{

```

```
int i = 10; /* Local definition */
i += 1; // This is local variable here
printf ("value of i=%d ... func () function \n", i);
}
```

This will produce following result

Value of i = 11 ... func () function

Value of i = 5 ... main function

## Storage Classes in C

A variable name identifies some physical location within the computer, where the string of bits representing the variable's value, is stored.

*There are basically two kinds of locations in a computer, where such a value may be kept*

- (i) Memory
- (ii) CPU registers

It is the variable's storage class that determines in which of the above two types of locations, the value should be stored.

We have four types of storage classes in C

- 1. Auto
- 2. Register
- 3. Static
- 4. Extern

### Auto Storage Class

*Features of this class are given below*

- Storage Location Memory
- Default Initial Value Garbage value
- Scope Local to the block in which the variable is defined.
- Life Till the control remains within the block in which variable is defined.

Auto is the default storage class for all local variables.

```
void main ( )
{
    auto int i = 3;
    auto int j;
}
```

## Register Storage Class

Register is used to define local variables that should be stored in a register instead of RAM. Register should only be used for variables that require quick access such as counters. *Features of register storage class are given below*

- **Storage Location** CPU register
- **Default Initial Value** Garbage value
- **Scope** Local to the block in which variable is defined.
- **Life** Till the control remains within the block in which the variable is defined.

```
void main ( )
{
    register int i;
}
```

## Static Storage Class

Static is the default storage class for global variables.

*Features of static storage class are given below*

- **Storage Location** Memory
- **Default Initial Value** Zero
- **Scope** Local to the block in which the variable is defined. In case of global variable, the scope will be through out the program.
- **Life** Value of variable persists between different function calls.

<pre># include&lt;stdio.h&gt; void increment ( ); void main ( ) {     increment ( );     increment ( ); } void increment ( ) {     auto int = 1;     printf ("%d \n", i);     i = i + 1; }</pre>	<pre>#include&lt;stdio.h&gt; void increment ( ); void main ( ) {     increment ( );     increment ( ); } void increment ( ) {     static int i=1;     printf ("%d \n", i);     i = i + 1; }</pre>
<b>Output 1</b> 2	<b>Output 1</b> 2

## Extern Storage Class

Extern is used of give a reference of a global variable that is variable to all the program files. When we use extern, the variable can't be initialized as all it does, is to point the variable name at a storage location that has been previously defined.

- Storage Location Memory
- Default Initial Value Zero
- Scope Global
- Life As long as the program's execution does not come to an end.

```
extern int y; * declaration*/  
int y = 3;    * definition*/
```

*Features of extern storage class are given below*

### File 1 main.c

```
int count = 5;  
main ( )  
{  
    write_extern ( );
```

```
}
```

### File 2 write.c

```
void write_extern (void);  
extern int count;  
void write_extern (void)  
{  
    printf ("count is %d \n", count);  
}
```

# Functions

A function is a self-contained block of statements that perform a coherent task of some kind. Making function is a way of isolating one block of code from other independent blocks of code.

```
# include< stdio.h>
void message ( );      /* Function prototype declaration*/
void main ( )
{
    message ( );      /* Function call*/
    printf ("\n After calling function");
}
void message ( )      /* Function definition */
{
    printf ("\n I am called function");
}
```

**Output** I am called function

After calling function, a function can take a number of arguments or parameters and a function can be called any number of times. A function can call itself such a process is called recursion.

*Functions can be of two types*

- (i) Library functions
- (ii) User-defined functions

## Key Points

- ♦ Avoids rewriting the same code again and again i.e., ensures reusability.
- ♦ Separating the code into modular functions also makes the program easier to design and understand.
- ♦ The value of each of the actual arguments in the calling function is copied into corresponding formal argument of the called function. With this method, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

## Call by Value

If we pass values of variables to the function as parameters, such kind of function calling is known as **call by value**.

```
#include <stdio.h>
void swapv (int x, int y);
void main ( )           // calling function
```

```

{
    int a = 10, b = 20;
    // values of arguments in calling function will be
    passed to the      called function.
    swapv (a, b);
    printf ("\n a = %d b = %d", a, b);
}
void swapv (int x, int y)    // called function
                                           Formal arguments in the called function
{
    int t;
    t = x;
    x = y;
    y = t;
    printf ("\n x = %d y = %d", x, y);
}
Output x = 20 y = 10
        a = 10 b = 20

```

**Note** The values of a and b remain unchanged even after exchanging or swapping the values of x and y.

## Call by Reference

Variables are stored somewhere in memory. So, instead of passing the value of a variable, if we pass the location number/address of the variable to the function, then it would become 'a call by reference'.

### Key Points

- ♦ In call by reference method, the address of actual arguments in the calling function are copied into the formal arguments of the called function.
- ♦ This means that, using these addresses, we would have an access to the actual arguments and hence we would be able to manipulate them.

```
#include<stdio.h>
void swapr (int *, int *);
void main ( )
{
    int a = 10, b = 20;
    swapr (&a, &b);
    printf ("\n a = %d b = %d, a,b);
}
void swapr (int * x, int *y)
{
    int t;
    t = *x; // Hear, we are assigning value at x to
the variable t
    * x = * y;
    * y = t;
}
```

## Previous to Swapping

a	b
10	20
100	200

In swapr (&a, &b); we are passing addresses of a and b i.e, 100 and 200. Which are being taken by the pointer variables x and y. Now, we are swapping the value at x to the value at y. That's why when we are getting the output it will be as follows

Output a = 20 b = 10

## After Swapping

a	b
20	10
100	200

This program manages to exchange the values of a and b using their addresses stored in x and y.

# Pointers

A pointer is a variable that stores memory address. Like all other variables, it also has a name, has to be declared and occupies some spaces in memory. It is called **pointer** because it points to a particular location.

Consider the declaration

int i = 9; this tells the C compiler to

- Reserve space in memory to hold the integer value.
- Associate the name i with memory location.
- Store the value i at this location.

i → Location name

[9] → Value at location

12354 → Location number or address

```
# include <stdio.h>
void main ( )
{
    int i = 9;
    printf ("\n Address of i=%u", &i);
    printf ("\n Value of i=%d", *(&i));
}
```

'&' = Address of operator

'\*' = Value at address operator or 'indirection' operator

&i returns the address of the variable i.

\*(&i) return the value stored at a particular address printing the value of  
\*(&i) is same as printing the value of i.

## Key Points

- If we want to assign some address to some variable, then we must ensure that the variable which is going to store some address must be of pointer type.
- If we want to assign &i to j i.e. j = &i; then we must declare j as int \*j;
- This declaration tells the compiler that j will be used to store the address of an integer value or j points to an integer.  
int \* j; means the value at the address contained in j is an integer.

Pointers are variables that contain addresses and since addresses are always whole numbers. Pointers would always contain whole numbers.

```

int i = 3; i      j
int *j; [3] [65524]
j = &i; 65524 65522
i's value is 3 and j's value is i's address.
printf ("\n Address of i = %u", &i);
printf ("\n Address of i = %u", j);
Output   Address of i = 65524
            Address of i = 65524
            printf ('/n Address of j = %u'', &j);
            printf ('/nvalue of j = %u'', j);
Output Address of j = 65522
            Value of j = 65524
            printf ("\n Value of i = %d", *(&i));
            printf ("\n Value of i = %d", *j);
Output   Value of i = 3
            Value of i = 3

```

## Array

It is a collection of similar elements (having same data type). For an array of 100 elements, the first element's index is zero '0' and the last index will be 99. This indexed access makes it very convenient to loop through each element of array. Array elements occupy contiguous memory locations.

0	1	2	3	4	5	6	7	Index
a	b	c	d	e	f	g	h	Values
100	101	102	103	104	105	106	107	Memory location

A [0] = a  
A [1] = b  
:: ::  
A [7] = h

## Multi Dimensional Array

In C language, one can have arrays of any dimensions. Let us consider a  $3 \times 3$  matrix.

	0	1	2	Column number [j]
Row number [i]	0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
	1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$
	2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$

$3 \times 3$  matrix for multi dimensional array

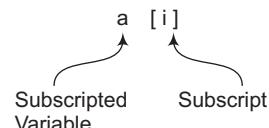
To access the particular element from the array, we have to use two subscripts; one for row number and other for column number. The notation is of the form  $a[i][j]$ , where  $i$  stands for row subscripts and  $j$  stands for column subscripts.

We can also define and initialize the array as follows

```
int values [3] [4] = {
    {1, 2, 3, 4,}
    {5, 6, 7, 8}
    {9, 10, 11, 12}
};
```

OR

```
int values [3] [4]
= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```



## A simple program using array

```
# include <stdio.h>
void main ( )
{
    int ave, sum = 0;
    int i;
    int marks [10];      // array declaration
    for (i = 0; i <= 9; i + +)
    {
        printf ("\n Enter marks");
        scanf ("%d", &marks [i]); // store data in array
    }
    for (i = 0; i <= 9; i + +)
        sum = sum + marks [i]; avg = sum/30;
    printf ("\n avg-marks = %d", avg);
}
```

# Strings

In C language, strings are stored in an array of character (char) type along with the null terminating character “\0” at the end.

```
char name [ ] = { 'K', 'R', 'I', 'S', 'H', 'N', 'A', '\0';
                  '\0' = Null character whose ASCII value is 0.
                  '0' = ASCII value is 48.
```

In the above declaration “0” is not necessary. C inserts the null character automatically.

```
# include <stdio.h>
void main ( )
{
    char name [ ] = "RAM";
    printf ("%s", name);
}
% s = It is used in printf ( ) as a format specification for
printing out a string.
```

*All the following notations refer to the same element*

```
name [i]
*(name + i)
* (i + name)
q [name]
# include <stdio.h>
void main ( )
{
    char name [ ] = "shyam";
    char * ptr;

    ptr = name; /*store base address of string */
    while (*ptr != '\0')
    {
        printf ("%c", *ptr);
        ptr++;
    }
}
```

**Note** The above program is used to print all the characters of a string using pointer.

## **Length of String**

Use strlen( ) function to get the length of a string minus the null terminating character.

**Syntax** int strlen (string);

## **Concatenation of String**

The strcat( ) function appends one string to another.

**Syntax** char \* strcat (string 1, string 2);

## **Copy String**

To copy one string to another string variable, we use strcpy( ) function.

**Syntax** strcpy (string 1, string 2);

# **Structures**

Structures in C are used to encapsulate or group together different data into one object. *We can define a structure as given below*

struct object

```
{  
    char id [20];  
    int rollno;  
    int marks;  
};
```

The variables we declare inside the structure are called data member.

## **Initializing a Structure**

Structure members can be initialized when we declare a variable of our structure.

## **Sample Code**

```
struct object student1 = {"Ram", 1, 95};
```

The above declaration will create a struct object called student1 with an id equal to "Ram", rollno equal to 1 and marks equal to 95. To access the members of a structure, we use the "." (dot operator) i.e., scope resolution operator.

```
struct object student1;
student1. id = "Ram"
student1. roll no = 1;
student1. marks = 95;
```

## Key Points

- ♦ The values of a structure variable can be assigned to another structure variable of the same type using assignment operator.
- ♦ One structure can be nested within another structure. Using this facility, complex data types can be created.  
e.g., struct emp
 

```
{
        char name [100];
        struct Address a;
      };
```
- ♦ A structure variable can also be passed to a function. We may either pass individual structure elements or the entire structure variable at one go .
- ♦ We can have a pointer pointing to a struct. Such pointers are known as structure pointers.

## Unions

A union is a collection of heterogeneous elements that is, it is a group of elements which are having different data types. Each member within a structure is assigned its own memory location. But the union members, all share a common memory location. Thus, unions are used to save memory.

```
union person
{
  char name [20];
  int age;
  float height;
};
```



Like structures, unions are also defined and declared.

```
union person Ram; // Union declaration
```

# Data Structure

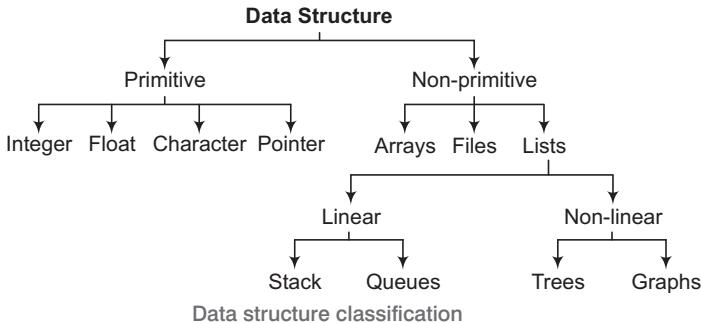
A data structure is a specialised way for organising and storing data in memory, so that one can perform operations on it.

*Data structure is all about*

- How to represent data element(s).
- What relationship data elements have among themselves.
- How to access data elements i.e., access methods

$$\text{Data Structures} + \text{Algorithms} = \text{Programs}$$

## Types of Data Structure



## Operations on Data Structures

*The operations involve in data structure are as follows*

**Create** Used to allocate/reserve memory for the data element(s).

**Destroy** This operation deallocate/destroy the memory space assigned to the specified data structure.

**Selection** Accessing a particular data within a data structure.

**Update** For updation (insertion or deletion) of data in the data structure.

**Searching** Used to find out the presence of the specified data item in the list of data item.

**Sorting** Process of arranging all data items either in ascending or in descending order.

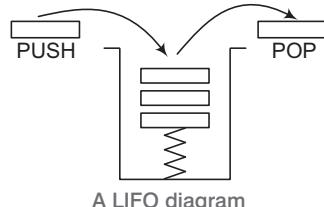
**Merging** Process of combining data items of two different sorted lists of data items into a single list.

# Stack

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end, called the TOP of the stack.

It is a LIFO (Last In First Out) kind of data structure.

e.g., If a person wants to put a plate on the pile of plates, he has to place it on the top (this is how stack grows) and if he wants to take a plate from the pile of plates he has to take it from the top (this is how stack shrinks).



## Operations on Stack

There are two operations including in stack as given below

- PUSH (when an item is added to stack)  
PUSH ( $s, i$ ); Adds the item  $i$  to the top of stack.
- POP (when an item is removed from stack)  
POP ( $s$ ); Removes the top element and returns it as a function value.

## Key Points

- ♦ Because of the PUSH operation which adds elements to a stack is sometimes called a **pushdown list**.
- ♦ If a stack contains a single item and the stack is popped, then the resulting stack contains no items and is called **empty stack**.
- ♦ The result of an illegal attempt to POP or access an item from empty stack is called **underflow**.
- ♦ Underflow can be avoided by ensuring that the stack is not empty.

## Implementation of Stack

A stack can be implemented using two ways

1. Array
2. Linked list

But since array size is defined at compile time, it can't grow dynamically. Therefore, an attempt to insert/push an element into stack (which is implemented through array) can cause a stack overflow situation, if it is already full.

So, to avoid the above mentioned problem we need to use linked list to implement a stack, because linked list can grow dynamically and shrink at runtime. We need to have an additional pointer (TOP) that will always point to the first node in the stack or the top of the stack (in case of linked list implementation of stack).

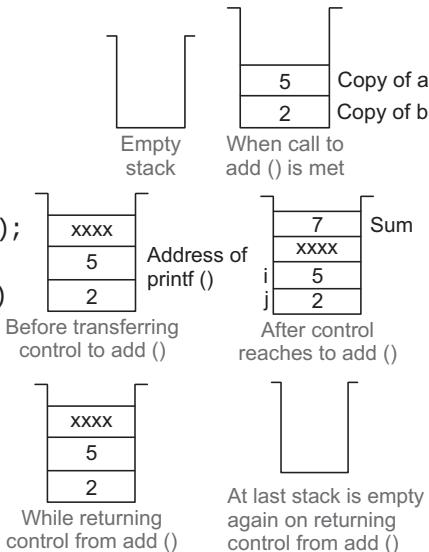
## Applications of Stack

*There are many applications of stack some of the important applications are given below*

### Function Calls

Different ways of organising the data are known as data structures. The compiler uses one such data structure called stack for implementing normal as well as recursive function calls.

```
# include <stdio.h>
int add (int, int);
void main ( )
{
    int a = 5, b = 2, c;
    c = add (a, b);
    printf (''sum=%d'', c);
}
int add (int i, int j)
{
    int sum,
    sum = i + j;
    return sum;
}
```



### Expression Evaluation

How a stack can be used for checking on syntax of an expression.

- **Infix expression** It is the one, where the binary operator comes between the operands e.g.,  $A + B * C$ .
- **Postfix expression** Here, the binary operator comes after the operands. e.g.,  $ABC * +$
- **Prefix expression** Here, the binary operator precedes the operands.

e.g.,  $+ A * BC$

This prefix expression is equivalent to  $A + (B * C)$  infix expression.

Prefix notation is also known as Polish notation.

Postfix notation is also known as suffix or Reverse Polish notation.

### Operator precedence

Exponential operator  $\wedge$  has highest precedence.

Multiplication/Division  $*, /$  operators have next precedence.

Addition/Subtraction  $+, -$  operators have least precedence.

## Reversing a List

First push all the elements of string in stack and then pop elements.

# Queue

It is a non-primitive, linear data structure in which elements are added/inserted at one end (called the REAR) and elements are removed/deleted from the other end (called the FRONT). A queue is logically a FIFO (First In First Out) type of list.

e.g., Consider a line of persons at a railway reservation counter. Whenever a person enters the queue, he stands at the end of the queue (addition of the nodes to the queue).

**Note** Every time the person at the front of the queue deposits the fee for ticket, he leaves the queue (deleting nodes from a queue).

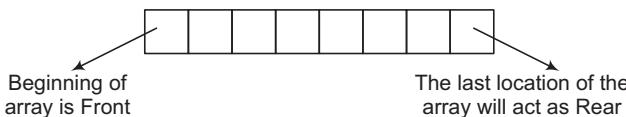
## Queue Implementation

*Queue can be implemented in two ways*

- Static implementation (using arrays)
- Dynamic implementation (using pointers)

## Key Points

- ♦ In static implementation, we have to declare the size of the array at design time or before the processing starts.
- ♦ Implementing queues using pointers, the main disadvantage is that a node in a linked representation occupies more memory space than a corresponding element in array representation.



Total number of elements present in the queue =  $(\text{Front} - \text{Rear} + 1)$

## Circular Queue

In a circular queue, the first element comes just after the last element or a circular queue is one in which the insertion of a new element is done at the very first location of the queue, if the last location of queue is full and the first location is empty.

**Note** A circular queue overcomes the problem of unutilised space in linear queues implemented as arrays.

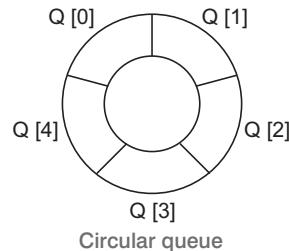
We can make following assumptions for circular queue

- Front will always be pointing to the first element (as in linear queue).
- If Front = Rear, the queue will be empty.
- Each time a new element is inserted into the queue, the Rear is incremented by 1.

$$\text{Rear} = \text{Rear} + 1$$

- Each time, an element is deleted from the queue, the value of Front is incremented by one.

$$\text{Front} = \text{Front} + 1$$

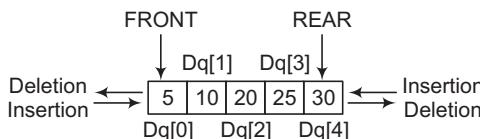


## Double Ended Queue (DEQUE)

It is a list of elements in which insertion and deletion operations are performed from both the ends. That is why it is called **double-ended queue** or DEQUE.

*There are two types of DEQUE, because of the restrictions put to perform either the insertion or deletion only at one end.*

- Input restricted DEQUE
- Output restricted DEQUE



### Operations on DEQUE

*There are following operations on DEQUE*

- Insertion of an element at the REAR end of the queue.
- Deletion of an element from the FRONT end of the queue.
- Insertion of an element at the FRONT end of the queue.
- Deletion of an element from the REAR end of the queue.

### Key Points

- ♦ For an input restricted DEQUE all of the above four operations are valid.
- ♦ For output restricted DEQUE only 1,2 and 3rd operations are valid.

## Priority Queues

This type of queue enables us to retrieve data items on the basis of priority associated with them. *Below are the two basic priority queue choices*

**Sorted Array or List** It is very efficient to find and delete the smallest element. Maintaining sortedness make the insertion of new elements slow.

**Binary Heaps** Here, the minimum key (in min Heap) is always at the root of the heap. Binary heaps are the right choice whenever we know an upperbound on the number of items in our priority queue. Since, we must specify the array size at compile time.

## Applications of Priority Queues

- Round Robin (RR) technique for processor scheduling is implemented using queues.
- All types of customer service (e.g., railway reservation) center softwares are designed using queues.
- Printer server routines are designed using queues.

# Linked List

It is a special data structure in which data elements are linked to one another. *Here, each element is called a node which has two parts*

- Info part which stores the information.
- Address or pointer part which holds the address of next element of same type.

Linked list is also known as **self referential structure**.

```
struct linked_list_node
{
    int info;
    struct linked_list_node *next;
};
```

Syntax of declaring a node which contains two fields in it one is for storing information and another is for storing address of other node, so that one can traverse the list.



**Note** Last node of a linked list contains NULL value in its address field.

## Advantages of Linked List

- Linked lists are dynamic data structure as they can grow and shrink during the execution time.
- Efficient memory utilisation because here memory is not pre-allocated.
- Insertions and deletions can be done very easily at the desired position.

## Disadvantages of Linked List

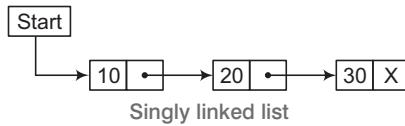
- More memory is required, if the number of fields are more.
- Access to an arbitrary data item is time consuming.

## Types of Linked List

These are different types of linked list as given below

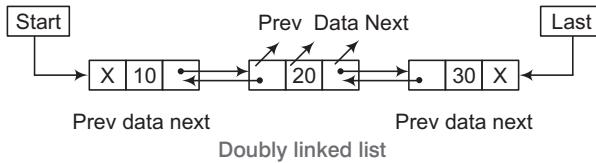
### Singly Linked List

In this type of linked list, each node has only one address field which points to the next node. So, the main disadvantage of this type of list is that we can't access the predecessor of node from the current node.



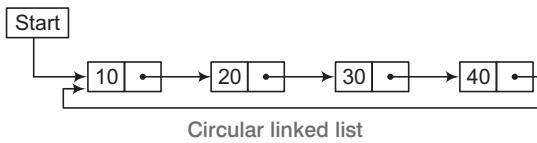
### Doubly Linked List

Each node of linked list is having two address fields (or links) which help in accessing both the successor node (next node) and predecessor node (previous node).



### Circular Linked List

It has address of first node in the link (or address) field of last node.



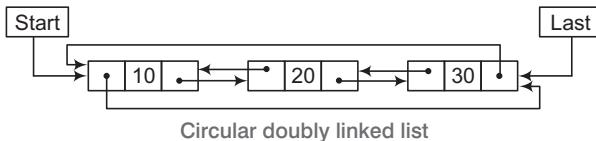
## Key Points

- ♦ **Null Pointer** Address field of the last node contains NULL value to indicate the end of list.
- ♦ **External Pointer (Start Node)** Pointer to the very first node of a linked list, it enables us to access the entire linked list.
- ♦ **Empty List** If the nodes are not present in the list, it is empty list or NULL list. The value of the external pointer/start pointer will be NULL for an empty list.

Start = NULL;

### Circular Doubly Linked List

It has both the previous and next pointer in circular manner.



## Operations on Linked Lists

The following operations involve in linked list are as given below

### Creation

Used to create a linked list.

### Insertion

Used to insert a new node in linked list at the specified position. A new node may be inserted

- At the beginning of a linked list
- At the end of a linked list
- At the specified position in a linked list
- In case of empty list, a new node is inserted as a first node.

### Deletion

This operation is basically used to delete an item (a node). A node may be deleted from the

- Beginning of a linked list.
- End of a linked list.
- Specified position in the list.

### Traversing

It is a process of going through (accessing) all the nodes of a linked list from one end to the other end.

## Memory Allocation: Garbage Collection in Linked List

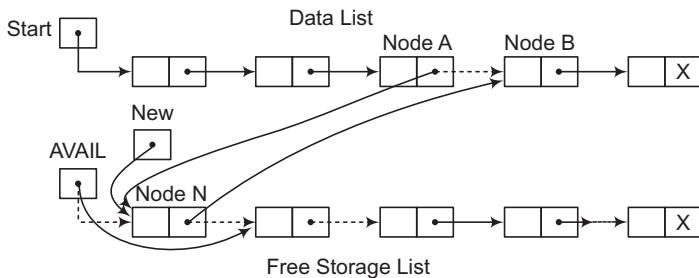
*There should be some mechanisms*

- Which provides unused memory space for new nodes.
- Which makes available the memory space of deleted nodes for future use.

## Situation of Overflow and Underflow

Suppose, someone wants to insert new data into a data structure but there is no available space *i.e.*, the free storage list is empty. This situation is usually called **OVERFLOW**. The situation, where one wants to delete data from a data structure that is empty is referred as **UNDERFLOW**.

If  $\text{START} = \text{NULL}$  and someone still wants to delete data, then UNDERFLOW situation occurs.



### Key Points

- Together with the linked lists in memory, a special list is maintained which consists of unused memory cells.
- The special list has its own pointer and is called the "List of Available Space" or the "Free Storage List" or the "Free Pool".

### Insertion of a Node at Specific Position in a Linked List

The next pointer field of Node A now points to the new Node N, to which AVAIL previously pointed. AVAIL now points to the second node in the free pool, to which Node N previously pointed.

The next pointer field of Node N now points to Node B to which Node A previously pointed. Suppose, we are given the value of LOC, where either LOC is the location of a Node A in a linked list or LOC = NULL.

$N$  = New node whose location is NEW

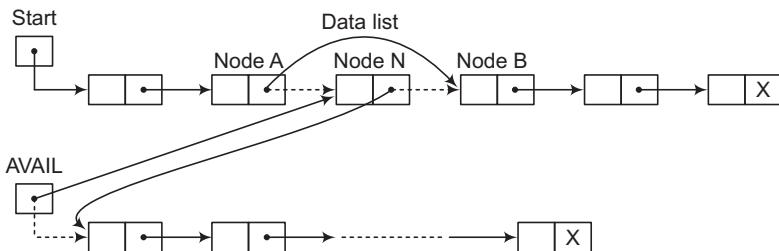
We need to insert ITEM into list, so that ITEM follows Node A or when LOC = NULL, so that ITEM is first node. If LOC is not NULL, then we let Node N point to Node B (which originally followed Node A) by the assignment.

$\text{LINK [NEW]} = \text{LINK [LOC]}$

and we let Node A point to the new Node N by the assignment.

$\text{LINK [LOC]} = \text{NEW}$

### Deletion from a Linked List



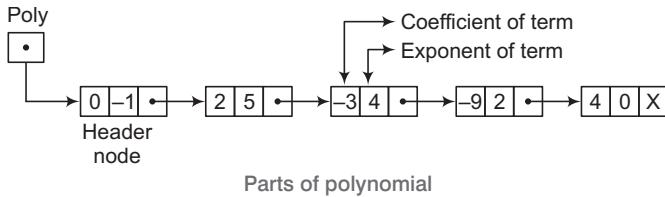
In the above diagram, the three pointer fields are changed as follows

- The next pointer field of Node A now points to Node B, where Node N previously pointed.
- The next pointer field of N now points to the original first node in free pool, where AVAIL previously pointed.
- AVAIL new points to the deleted Node N.

### Polynomials (An Application of Linked Lists)

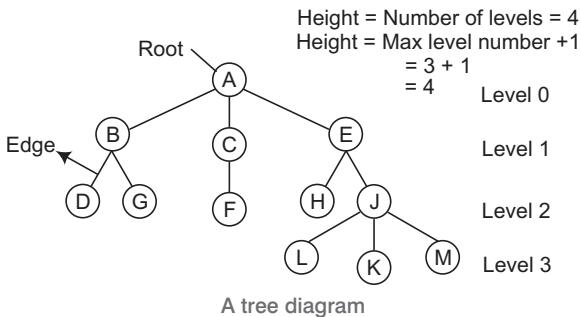
A polynomial  $p(x) = 2x^5 - 3x^4 - 9x^2 + 4$ , may be represented by list, where each node corresponds to a non-zero term of  $p(x)$ .

The information part of the node is divided into two fields representing, the coefficient and the exponent of the corresponding term, respectively. All nodes are linked according to decreasing degree.



# Tree (Non-linear Data Structures)

Trees are used to represent data containing a hierarchical relationship between elements e.g., records, family trees and table contents.



For above tree

Number of level = 4

Max level number = 3

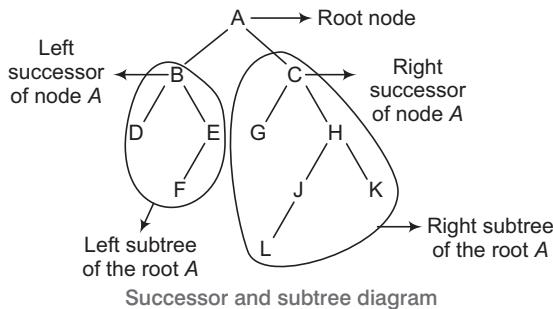
- **Node** Each data item in a tree.
- **Root** First or top data item in hierarchical arrangement.
- **Degree of a Node** Number of subtrees of a given node.  
e.g., Degree of A = 3, Degree of E = 2
- **Degree of a Tree** Maximum degree of a node in a tree.  
e.g., Degree of above tree = 3
- **Depth or Height** Maximum level number of a node + 1(i.e., level number of farthest leaf node of a tree + 1).  
e.g., Depth of above tree =  $3 + 1 = 4$
- **Non-terminal Node** Any node except root node whose degree is not zero.
- **Terminal Node or Leaf Nodes** having degree = 0
- **Forest** Set of disjoint trees.
- **Siblings** D and G are siblings of parent Node B.
- **Path** Sequence of consecutive edges from the source node to the destination node.  
e.g., Path between A and K comprises (A, E), (E, J) and (J, K) node pairs.

## Binary Tree

In this kind of tree, the maximum degree of any node is at most 2.

A *binary tree T* is defined as a finite set of elements such that

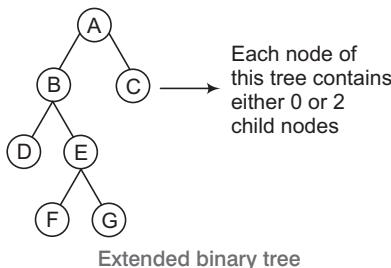
- $T$  is empty (called NULL tree or empty tree).
- $T$  contains a distinguished Node  $R$  called the root of  $T$  and the remaining nodes of  $T$  form an ordered pair of disjoint binary trees  $T_1$  and  $T_2$ .



Any node  $N$  in a binary tree  $T$  has either 0, 1 or 2 successors. Level  $r$  of a binary tree  $T$  can have at most  $2^r$  nodes.

## Extended Binary Trees : 2-Trees or Strictly Binary Trees

If every non-terminal node in a binary tree consist of non-empty left subtree and right subtree. In other words, if any node of a binary tree has either 0 or 2 child nodes, then such tree is known as **strictly binary tree** or **extended binary tree** or **2-tree**.

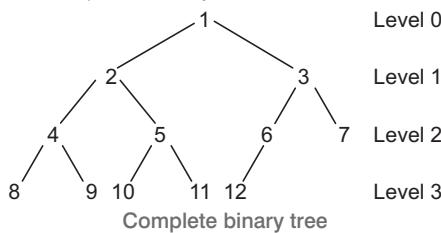


## Complete Binary Tree

A binary tree is one which have the following properties

- Which can have 0, 1 or 2 nodes as a child node.
- In which first, we need to fill left node, then right node in a level.
- In which, we can start putting data item in next level only when the previous level is completely filled.

If all the above three conditions are fulfilled by any binary tree, then we can say that as a complete binary tree.



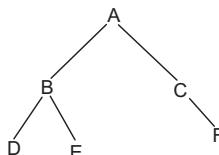
Now, if we want to add new item in above tree, then we need to add it as right child of node 6. We can't add it anywhere else. (According to condition 2 and 3).

## Tree Traversal

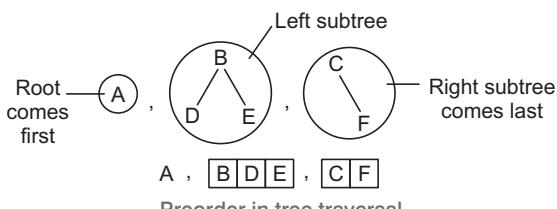
Three types of tree traversal are given below

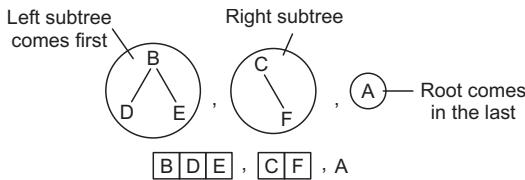
1. Preorder (Root, Left subtree, Right subtree)
2. Postorder (Left subtree, Right subtree, Root)
3. Inorder (Left subtree, Root, Right subtree)

The detailed description of different types of tree traversal are given below

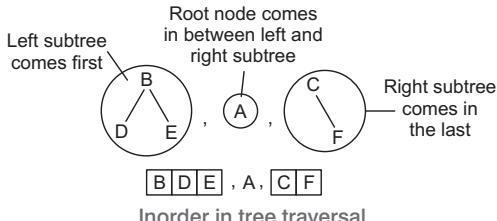


Preorder



**Postorder**

Postorder in tree traversal

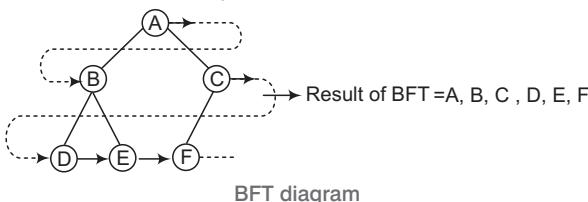
**Inorder**

Inorder in tree traversal

**Note** In all of the above three techniques, we need to decompose left and right subtree, according to respective rules only.

**Breadth First Traversal (BFT)**

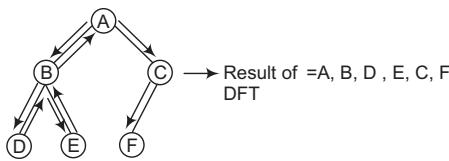
The BFT of a tree visits all the nodes in the order of their depth in the tree. BFT first visits all the nodes at depth zero (*i.e.*, root), then all the nodes at depth 1 and so on. At each depth, the nodes are visited from left to right.



BFT diagram

**Depth First Traversal (DFT)**

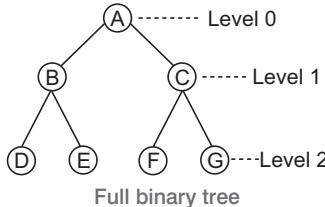
In DFT, one starts from root and explores as far as possible along each branch before backtracking.



A DFT diagram

## Perfect Binary Tree or Full Binary Tree

A binary tree in which all leaves are at the same level or at the same depth and in which every parent has 2 children.



Here, all leaves ( $D, E, F, G$ ) are at depth 3 or level 2 and every parent is having exactly 2 children.

### Key Points

- ♦ BFT uses queue for traversing.
- ♦ DFT uses stack for traversing.

## Some Important Formulas of Binary Tree

*Number of nodes 'n' in a perfect binary tree can be found using the following formula*

$$n = 2^h - 1$$

Where,  $h$  = Height of perfect binary tree i.e., number of levels in tree or (maximum level number + 1). The number of nodes ' $n$ ' in a complete binary tree is atleast  $(2^{h-1})$  and at most  $(2^h - 1)$ .

*In a complete binary tree, if we want to find out children and parent of any node  $k$ , use following formula*

Left child of the node  $k = 2 * k$

Right child of the node  $k = (2 * k) + 1$

Parent of the node  $k = \lfloor k/2 \rfloor$

Left child of node  $2 = 2 * 2 = 4$

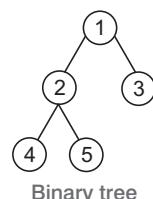
Right child of node  $2 = (2 * 2) + 1 = 5$ , here,  $k = 2$

Parent of node  $2 = \lfloor 2/2 \rfloor = 1$

The depth  $d_n$  of the complete binary tree  $T_n$  with  $n$  nodes is given by

$$d_n = \lfloor \log_2 n + 1 \rfloor$$

Suppose, any complete tree  $T_n$  has 8 nodes, then



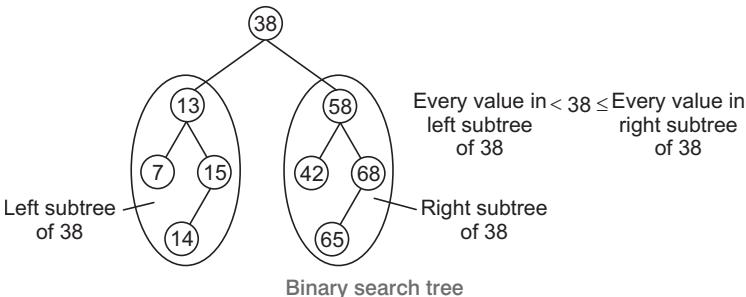
$$d_n = \lfloor \log_2 8 + 1 \rfloor = \lfloor \log_2 2^3 + 1 \rfloor = \lfloor 3 \log_2 2 + 1 \rfloor$$

$$d_n = \lfloor 3 + 1 \rfloor = 4$$

The number of nodes in a perfect binary tree can also be found using this formula  $n = 2L - 1$ , where  $L$  is the number of leaf nodes in the tree. The number of null links in a binary tree of ' $n$ ' nodes is equal to  $(n + 1)$ .

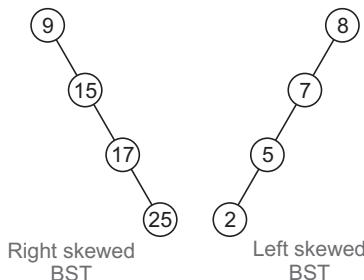
## Binary Search Tree (BST)

A binary tree  $T$ , is called binary search tree (or binary sorted tree), if each node  $N$  of  $T$  has the following property. The value at  $N$  is greater than every value in the left subtree of  $N$  and is less than or equal to every value in the right subtree of  $N$ .



## AVL-Tree

The disadvantage of a BST is that if every item which is inserted to be next is greater than the previous item, then we will get a **right skewed BST** or if every item which is to be inserted is less than to the previous item, then we will get a **left skewed BST**.



So, to overcome the skewness problem in BST, the concept of AVL-tree or height balanced tree came into existence.

A non-empty binary tree  $T$  is an AVL-tree if and only if

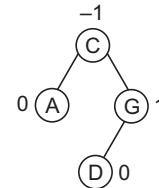
$$|h(T_L) - h(T_R)| \leq 1$$

where,  $h(T_L)$  = Height of left subtree  $T_L$  of tree  $T$

$h(T_R)$  = Height of right subtree  $T_R$  of tree  $T$

$h(T_L) - h(T_R)$  is also known as Balance Factor (BF). For an AVL (or height balanced tree), the balance factor can be either 0, 1 or  $-1$ . An AVL search tree is binary search tree which is an AVL-tree.

$$\begin{aligned} h(T_L \text{ of } C) &= 1 \\ h(T_R \text{ of } C) &= 2 \\ \text{BF of } C &= h(T_L \text{ of } C) - h(T_R \text{ of } C) \\ &= 1 - 2 \\ &= -1 \end{aligned}$$



### Key Points

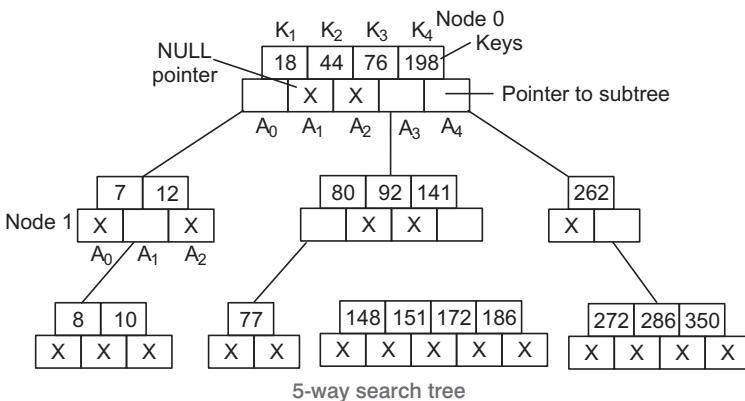
- ♦ In the AVL-tree, every node should have the value of BF either 0 or 1 or  $-1$ .
- ♦ If the value of BF is not falling in this criteria, then that tree is not an AVL-tree.

### $m$ -way Search Tree

To favour retrieval and manipulation of data stored in external memory viz. storage devices such as disks etc., there is a need for some special data structures. e.g., such data structures as  $m$ -way search trees. B-Trees and  $B^+$ -trees. An  $m$ -way search tree  $T$  may be an empty tree, if  $T$  is non-empty, it satisfies the following properties

- For some integer  $m$ , known as order of the tree, each node is of degree which can reach a maximum of  $m$ , in other words, each node has at most  $m$  child nodes.
- A node may be represented as  $A_0, (k_1, A_1), (k_2, A_2), \dots (k_{m-1}, A_{m-1})$ , where  $k_i, 1 \leq i \leq (m-1)$  are the keys and  $A_i, 0 \leq i \leq (m-1)$  are the pointers to subtrees of  $T$ .
- If a node can have  $k$  child nodes, where  $k \leq m$ , then the node can have only  $(k-1)$  keys  $k_1, k_2, \dots k_{k-1}$ , contained in the node such that  $k_i < k_{i+1}$  and each of the keys partitions all the keys in the subtrees into  $k$  subsets.

- For a node  $A_0, (k_1, A_1), (k_2, A_2), \dots, (k_{m-1}, A_{m-1})$ , all key values in the subtree pointed to by  $A_i$  are less than the key  $k_{i+1}$ ,  $0 \leq i \leq m-2$  and all key values in the subtree pointed to by  $A_{m-1}$  are greater than  $K_{m-1}$ .
- Each of the subtrees  $A_i$ ,  $0 \leq i \leq m-1$  are also  $m$ -way search trees.
- Below is an example of a 5-way search tree. Here,  $m = 5$  which means each node can have at most  $m = 5$  child nodes and therefore has atmost  $(m - 1) = 4$  keys contained in it.



In the above diagram, we can see that  $k_1 < k_2 < k_3 < k_4$  for each node.

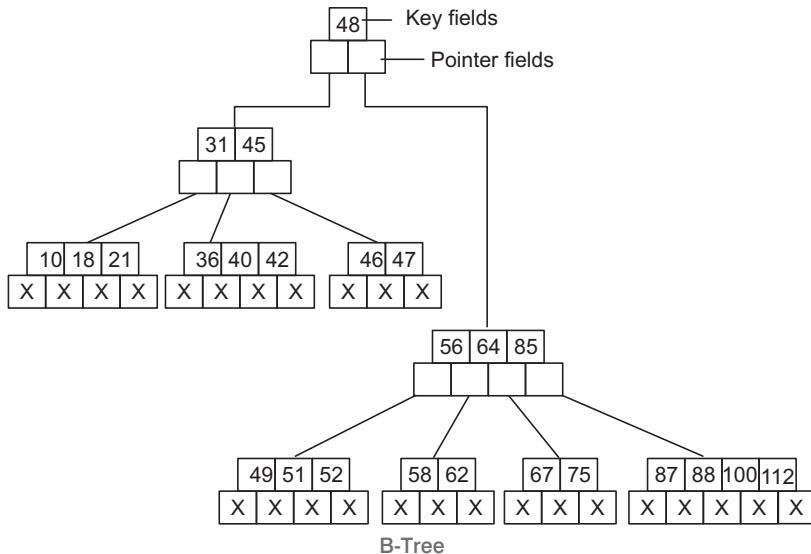
## B-Tree

$m$ -way search trees have the advantage of minimising file accesses due to their restricted height. But still, we need to keep the height of  $m$ -way search tree as low as possible and therefore the need to maintain balanced  $m$ -way search trees arises.

So, a B-tree is nothing but a balance  $m$ -way search tree. A B-tree of order  $m$ , if non-empty, is an  $m$ -way search tree in which

- The root has atleast 2 child nodes and at most  $m$  child nodes.
- The internal nodes except the root have atleast  $\left\lceil \frac{m}{2} \right\rceil$  child nodes and at most  $m$  child nodes.
- The number of keys in each internal node is one less than the number of child nodes and these keys partition the keys in the subtrees of the node in a manner similar to that of  $m$ -way search trees.
- All leaf nodes are on the same level.

A B-tree of order 5 is referred to as 4 - 5 tree, since the internal nodes are of degree 2 or 3 only.



### B<sup>+</sup> Trees

- It is a variant of B-tree in which all records are stored in the leaves and all leaves are linked sequentially.
- A B<sup>+</sup>tree consists of one or more blocks of data, called nodes linked together by pointers.
- Internal nodes point to other nodes in the tree and leaf nodes point to data in database using data pointers.
- In a B<sup>+</sup>-tree, in contrast to a B-tree, all records are stored at the lowest level of the tree.

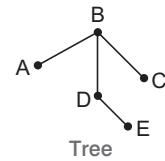
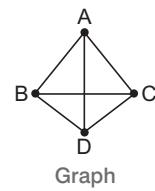
# Graph

A graph is a set of vertices and edges which connects them. A *graph G* consists of two things

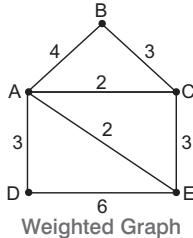
1. A set of vertices  $V$ .
  2. A set of edges  $E$  such that edge  $e$  in  $E$  is identified with a unique pair  $u, v$  of vertices in  $V$ , denoted by  $e = \{u, v\}$
- **Path** A path  $P$  of length  $n$  from a node  $u$  to a node  $v$  is defined as a sequence of  $(n + 1)$  nodes.

$$P = (v_0, v_1, v_2, \dots, v_n)$$

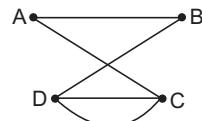
- **Adjacent Nodes or Neighbours** Suppose  $e = [u, v]$ , then the nodes  $u$  and  $v$  are called the endpoints of  $e$  and  $u$  and  $v$  are said to be adjacent nodes or neighbours.
- **Degree of a Node** The number of edges containing  $u$ , is the degree of a node  $\deg(u)$ . If  $\deg(u) = 0$ , this means  $u$  doesn't belong to any edge, then  $u$  is called as isolated node.
- **Connected Graph** A graph  $G$  is said to be connected, if there is a path between any two of its nodes.
- **Complete Graph** A graph  $G$  is said to be complete, if every node  $u$  in  $G$  is adjacent to every other node  $v$  in  $G$ .  
$$\left\{ \text{Number of edges with } n \text{ nodes in a complete graph } G = \frac{n(n-1)}{2} \right\}$$
- **Tree Graph or Free Tree/Tree** A connected graph  $T$  without any cycles is known as tree graph or free tree or simply a tree.  
If  $T$  is a finite tree with  $k$  nodes, then  $T$  will have  $(k - 1)$  edges.
- **Labelled and Weighted Graph** A graph  $G$  is said to be labelled, if its edges are assigned data.  $G$  is said to be weighted, if each edge  $e$  in  $G$  is assigned a non-negative numerical value.



$w(e)$  = Weight or length of edge  $e$



**Multigraph** If a graph is having multiple edges and/or loops, then it will be called a **multigraph**.



**Multiple Edges** Distinct edges  $e$  and  $e'$  are called **multiple edges**, if they connect the same endpoints i.e., if  $e [u, v]$  and  $e' [u, v]$ .

**Loops** An edge  $e$  is called a **loop**, if it has identical start point and end point i.e., if  $e = [u, u]$

#### Directed Graph (Digraph)

Each edge in graph  $G$  is assigned a direction or each edge  $e$  is identified with an ordered pair  $(u, v)$  of nodes  $G$  rather than an unordered pair  $[u, v]$ .

Origin or initial point of  $e \leftarrow u \xrightarrow{e} v \rightarrow$  Destination or terminal point of  $e$

$u$  is predecessor of  $v$  and  $v$  is a successor of  $u$  or neighbour of  $u$ .

$u$  is adjacent to  $v$  and  $v$  is adjacent to  $u$ .

- Outdegree of a node  $u$  in  $G$

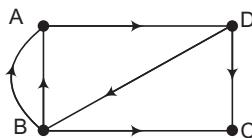
$\text{outdeg}(u) = \text{Number of edges beginning at } u$

- Indegree of a node  $u$  in  $G$

$\text{indeg}(u) = \text{Number of edges ending at } u$

- A node  $u$  is called source, if it has positive outdegree but zero indegree.
- A node  $u$  is called sink, if it has a zero outdegree but a positive indegree.

e.g.,



$$\text{indeg}(c) = 2; \quad \text{outdeg}(c) = 0$$

So, C is a sink and there is no source node of this graph.

### Adjacency Matrix

Suppose  $G$  is a simple directed graph with  $m$  nodes and suppose the nodes of  $G$  have been ordered are called  $v_1, v_2, \dots, v_m$ .

Then, the adjacency matrix  $(A) = (a_{ij})$  of the graph  $G$  is the  $m \times m$  matrix defined as follows

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j, \text{ i.e., if there is an edge } (v_i, v_j) \\ 0 & \text{Otherwise} \end{cases}$$

Such a matrix which contains entries of 0 by 0 and 1, is called a bit matrix or a Boolean matrix.

The adjacency matrix  $A$  of the graph  $G$  does depend on the ordering of the nodes of  $G$ ; i.e., a different ordering of the nodes may result in a different adjacency matrix.

# 3

# Design and Analysis of Algorithms

---

## Introduction to Analysis of Algorithms

An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. Informally, an algorithm is any well defined computational procedure that takes some value or set of values as input and produces some value or set of values as output. Thus, an algorithm is a sequence of computational steps that transform the input into the output.

*There are some terms related to the algorithms are given as follows*

### **Computational Problem**

A computational problem is a specification of the desired input-output relationship. It focuses on classifying computational problems according to their inherent difficulty and relating those classes to each other.

### **Instance**

An instance of a problem is all the inputs needed to compute a solution to the problem. A computational problem can be viewed as an infinite collection of instances together with a solution for every instance.

### **Algorithm**

An algorithm is a well defined computational procedure that transforms inputs into outputs, achieving the desired input-output relationship.

## Correct Algorithm

A correct algorithm halts with the correct output for every input instance. We can say that the algorithm solves the problem.

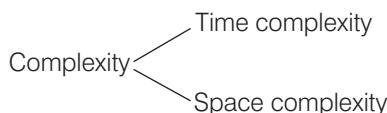
## Analysis of Algorithms

Analysis of algorithm depends upon various factors such as memory, communication bandwidth or computer hardware. The most often used for analysing of algorithms is the computational time that an algorithm requires for completing the given task. By counting the number of steps, the runtime of an algorithm is measured.

## Complexity

Algorithm can be classified by the amount of time they need to complete compared to their input size.

The analysis of an algorithm focuses on the complexity of algorithm.



### Time Complexity

The time complexity is a function that gives the amount of time required by an algorithm to run to completion. The time complexity quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input.

### Space Complexity

The space complexity is a function that gives the amount of space required by an algorithm to run to completion.

### Key Points

- ♦ We usually refer the time complexity in three terms
  - (i) Best case time complexity
  - (ii) Average case time complexity
  - (iii) Worst case time complexity
- ♦ **Worst case time complexity** It is the function defined by the maximum amount of time needed by an algorithm for an input of size  $n$ .
- ♦ **Average case time complexity** It is the execution of an algorithm having typical input data of size  $n$ .
- ♦ **Best case time complexity** It is the minimum amount of time that an algorithm requires for an input of size  $n$ .

## Asymptotic Analysis

This can be done by considering the following analysis

- It describes the behaviour of function in the limit.
- There are multiple asymptotic notations those compare the sizes of functions.

$O$  (Big Oh)  $\approx \leq$  {Asymptotic upper bound}

$\Omega$  (Big Omega)  $\approx \geq$  {Asymptotic lower bound}

$\theta$  (Big Theta)  $\approx =$  {Asymptotic tight bound}

$o$  (Small Oh)  $\approx <$  {Tight upper bound}

$\omega$  (Small Omega)  $\approx >$  {Tight lower bound}

## Asymptotic Notations

The notations we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers  $N = \{0, 1, 2, \dots\}$

*The asymptotic notations consist of the following useful notations*

### Big Oh ( $O$ )

If we write  $f(n) = O(g(n))$ , then there exists a function  $f(n)$  such that

$$f(n) \leq cg(n)$$

with any constant  $c$ .

Or we can say  $g(n)$  is an asymptotic upper bound for  $f(n)$ .

e.g.,  $2n^2 = O(n^3)$  with constant  $c = 1$ .

### Big Omega ( $\Omega$ )

If we write  $f(n) = \Omega(g(n))$ , then there exists a function  $f(n)$  such that

$$f(n) \geq cg(n)$$

with any constant  $c$ .

Function  $g(n)$  is an asymptotic lower bound for  $f(n)$ .

e.g.,  $\sqrt{n} = \Omega(\log_2 n)$  with constant  $c = 1$ .

### Big Theta ( $\theta$ )

If we write  $f(n) = \theta(g(n))$ , then there exists a function  $f(n)$  such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

with any positive constants  $c_1$  and  $c_2$ .

Function  $g(n)$  is an asymptotically tight bound for  $f(n)$ .

**Theorem**  $f(n) = \Theta(g(n))$  if and only if  $f = O(g(n))$  and  $f(n) = \Omega(g(n))$ .

### Small Oh ( $o$ )

**Notation** If we write  $f(n) = o(g(n))$ , then there exists a function such that  $f(n) < cg(n)$ , with any positive constant  $c$ .

$g(n)$  is an asymptotically tight upper bound of  $f(n)$ .

e.g., 1.  $n^{1.99999} = o(n^2)$       2.  $\frac{n^2}{\log_2 n} = o(n^2)$

### Small Omega ( $\omega$ )

**Notation** If we write  $f(n) = \omega(g(n))$ , then there exists a function such that  $f(n) > cg(n)$ , with any positive constant  $c$ .

$g(n)$  is asymptotically tight lower bound of  $f(n)$ .

e.g.,  $n^{2.00001} = \omega(n^2)$   
and  $n^2 \neq \omega(n^2)$

## Comparisons of a Symptotic Notations Based on the Relational Properties

Assume,  $f(n)$  and  $g(n)$  are as asymptotically positive.

*The relational properties are given as under*

### Transitivity

$$\begin{aligned} f(n) &= \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \\ \Rightarrow f(n) &= \Theta(h(n)) \end{aligned}$$

This property is same for  $O$ ,  $\Omega$ ,  $o$  and  $\omega$ .

### Reflexivity

$$f(n) = \Theta(f(n))$$

This property is same for  $O$  and  $\Omega$ .

### Symmetry

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n))$$

### Transpose Symmetry

$$\begin{aligned} f(n) &= O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)) \\ f(n) &= O(g(n)) \text{ if and only if } g(n) = \omega(f(n)) \end{aligned}$$

## Recurrences

A recurrence is a function, defined in terms of

- One or more base cases
- Itself with smaller arguments

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(n - 1) + 1 & \text{if } n > 1 \end{cases}$$

$$T(n) = n$$

In algorithm analysis, we usually express both the recurrence and its solution using asymptotic notation.

## Methods to Solve the Recurrence Relation

There are two methods to solve the recurrence relation given as

1. Substitution method
2. Master method

### Substitution Method

The substitution method is a condensed way of proving an asymptotic bound on a recurrence by induction. In the substitution method, instead of trying to find an exact closed form solution, we only try to find a closed form bound on the recurrence.

There are two steps in this method

- Guess the solution
- Use induction to find the constants and show that the solution works.

### Key Points

- ♦ The substitution method is powerful approach that is able to prove upper bounds for almost all recurrences.
- ♦ The substitution method requires the use of induction.

### Master Method

The master method gives us a quick way to find solutions to recurrence relations of the form  $T(n) = aT(n/b) + f(n)$ .

where,  $a$  and  $b$  are constants,  $a \geq 1$  and  $b > 1$ . Here,  $a$  represents how many recursive calls are made,  $b$  represents the factor by which the work is reduced in each recursive call.  $f(n)$  represents how much work is done by each call.

### Solution of Recurrence Relation

- The solution of the recurrence relation will be in the form of

$$T(n) = n^{\log_b a} [U(n)]; \quad h(n) = \frac{f(n)}{n^{\log_b a}}$$

- Now, the values of  $U(n)$  and  $h(n)$  are related in three cases as follows

**Case 1** If  $h(n) = n^r$ ,  $r < 0$ , then  $U(n) = O(1)$

**Case 2** If  $h(n) = n^r$ ,  $r > 0$ , then  $U(n) = O(n^r)$

**Case 3** If  $h(n) = (\log_2 n)^i$ ,  $i \geq 0$ , then  $U(n) = \frac{O(\log_2 n)^{i+1}}{i+1}$

$$T(n) = O(n \log n)$$

## Searching

In computer science, search operation is used for finding an item with specified properties among a collection of items.

Searching can be done in two ways

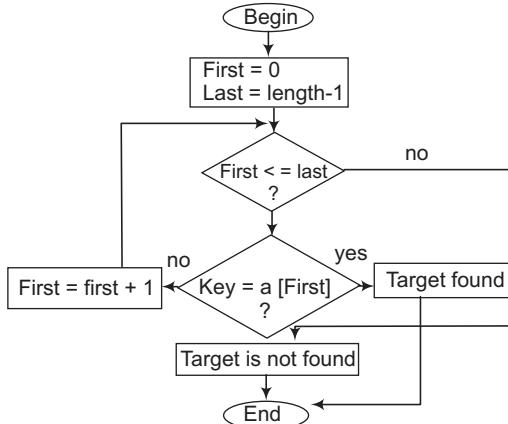
### Sequential Search (Linear Search)

This is the most common algorithm which starts at the beginning and walk to the end, testing for a match at each item. This searching has the benefit of simplicity.

Pseudo code for sequential searching

```
int linearsearch (int a [ ], int first , int last, int key)
{
    for (int i = first; i <= last; i++)
    {
        if (key == a [i])
        {
            return i;           // successfully found the
        }           // key and return location
    }
    return - 1;           // failed to find key element
}
```

### Flow Chart for Linear Search



### Analysis of Sequential Search

*The time complexity in sequential search in all three cases is given below*

**Best case** When we find the key on the first location of array, then the complexity is  $O(1)$ .

**Worst case** When the key is not found in the array and we have to scan the complete array, then the complexity is  $O(n)$ .

**Average case** When the key could appear anywhere in the array, then a successful search will take total  $1 + 2 + 3 + \dots + n$  comparisons.

$$\therefore 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2} \text{ comparisons}$$

$$\begin{aligned} \therefore \text{Average number of comparisons} &= n \left( \frac{n+1}{2} \right) \times \frac{1}{n} \\ &\quad \left( \because \frac{\text{Total number of comparisons}}{\text{number of items}} \right) \\ &= \frac{n+1}{2} \end{aligned}$$

So, the time complexity =  $O(n)$

## Binary Search

In computer science, a binary search (half interval search) algorithm finds the position of a specified value with a sorted array. *In each step, this algorithm divides the array into three sections*

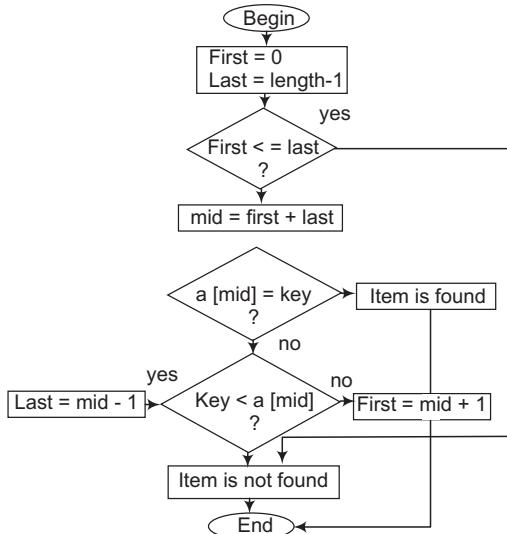
- Middle element
- Elements on left side of the middle element
- Elements on right side of the middle element

Then, check that if the middle element is the correct value, the searching stops. Otherwise go to the left side of the array if searching item is less than the middle element or go to the right side of the array if searching item is greater than middle element and this will go on until either the algorithm found the element or there are no elements to examine. This algorithm always look at the center value. Each time you get to discard half of the remaining list.

### Pseudo Code of Binary Search

```
int binarysearch (int a[ ], int n, int key)
{
    int first = 0, last = n - 1, middle;
    while (first <= last)
    {
        middle = (first + last)/2; /*
        calculate middle*/
        if (a [middle]==value) /*
        if value is found at mid */
        {
            return middle;
        }
        else if (a [middle] > value) /*
        if value is at left half */
        {
            last = middle - 1;
        }
        else
            first = middle + 1; /*
            if value is in right half */
    }
    return -1;
}
```

### Flow Chart for Binary Search



### Analysis of Binary Search

The time complexity of binary search in all three cases is given below

**Best case** The best case complexity is  $O(1)$  i.e., if the element to search is the middle element of the complete array. So, there is only one comparison required.

**Average case** This search cut the range in half every time.

$$T(n) = T(n/2) + 1$$

$$T(n) = T(n/2^2) + 2$$

$$\vdots \qquad \vdots \qquad \vdots$$

Repeat upto  $K$  times, we get

$$T(n) = T(n/2^K) + K$$

$$\text{Let } n = 2^K; T(n) = T(2^K/2^K) + K$$

$$T(n) = T(1) + K$$

$$T(n) = 1 + K // \text{when array has one element, then it takes } O(1) \text{ time}$$

$$T(n) = O(K);$$

$$\text{If } n = 2^K \Rightarrow K = \log_2 n; T(n) = O(\log_2 n)$$

**Worst case** In worst case, the complexity of binary search is  $O(\log_2 n)$ , because if element is not found but we have to run the algorithm, until there are no element in the array.

# **Sorting**

In Computer Science, sorting operation is used to put the elements of list in a certain order, i.e., either in non-decreasing or decreasing order.

*Sorting can be of two types*

- In-place sorting
- Stable Sorting

## **In-place Sorting**

The in-place sorting algorithm does not use extra storage to sort the elements of a list. An algorithm is called in-place as long as it overwrites its input with its output. Those algorithms, which are not in-place, are out of place algorithms.

## **Stable Sorting**

Stable sorting algorithm maintain the relative order of records with equal values during sorting and after sorting operation or we can say that a stable sorting algorithm is called stable, if it keeps elements with equal keys in the same relative order as they were in the input.

### **Divide and Conquer Approach**

The divide and conquer approach is based on the recursion or it works by recursively breaking down a problem into two or more subproblems, until these becomes simple enough to be solved directly. The solutions of the subproblems are then combined to give a solution to the original problem.

## **Methods of Sorting**

*There are different sorting techniques as given below*

### **Bubble Sorting**

Bubble sorting is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them, if they are in random or unorganised order. This sorting technique go through multiple passes over the array and each pass moves the largest (or smallest) element to the end of the array.

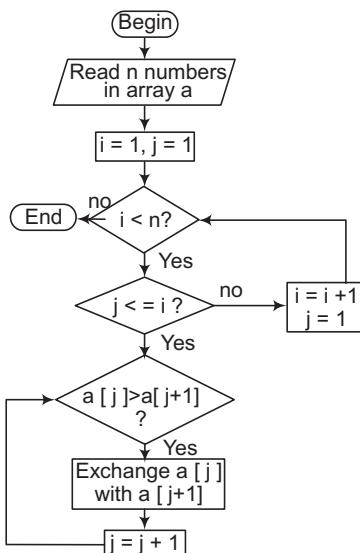
**Pseudo Code for Bubble Sorting**

```

void BubbleSort (int a[ ], int n)
{
    int i, temp,j;
    for (i = 1; i <= n ; i++)
    {
        for (j = 1; j <= i; j++)
        {
            if (a[j] > a [j + 1])
            {
                temp = a [j],
                a [j] = a [j + 1];
                a [j + 1] = temp;
            }
        }
    }
}

```

The internal loop is used to compare adjacent elements and swap elements, if they are not in order. After the internal loop has finished the execution, the largest element in the array will be moved at the top. The external loop is used to control the number of passes.

**Flow Chart for Bubble Sorting**

### **Analysis of Bubble Sort**

Generally, the number of comparisons between elements in bubble sort can be stated as follows

$$(n - 1) + (n - 2) + \dots + 2 + 1 = n(n - 1)/2 = O(n^2)$$

In any case (worst case, best case or average case) to sort the list in ascending order, the number of comparisons between elements would be same. But the time complexity is different.

*The time complexity of bubble sort in all three cases is given below*

- Best case  $O(n)$
- Average case  $O(n^2)$
- Worst case  $O(n^2)$

### **Insertion Sorting**

The insertion sort only passes through the array once. Therefore, it is very fast and efficient sorting algorithm with small arrays.

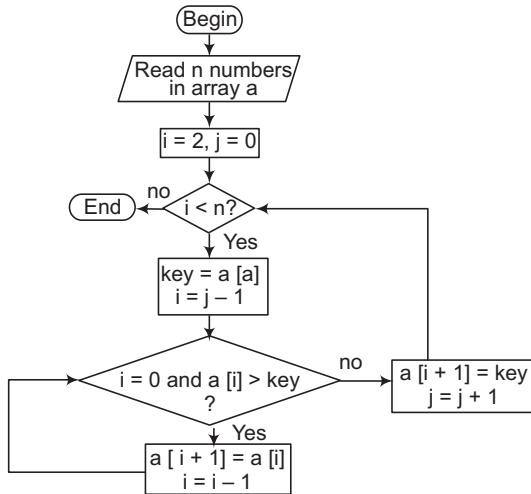
#### **Key Points**

- ◆ This algorithm works the way you might sort a hand of playing cards
- ◆ We start with an empty left hand [sorted array] and the cards face down on the table [unsorted array].
- ◆ Then, remove one card [key] at a time from the table [unsorted array] and insert it into the correct position in the left hand [sorted array].
- ◆ To find the correct position for the card, we compare it with each of the cards already in the hand, from right to left.

#### **Pseudo Code of Insertion Sort**

```
int insertionsort (int a[], int n)
{
    int i, j, key;
    for (j = 2; j <= n; j++)
    {
        key = a[j];
        i = j - 1;
        while (i > 0 && a[i] > key)
        {
            a[i + 1] = a[i];
            i = i - 1;
        }
        a[i + 1] = key;
    }
}
```

## Flow Chart for Insertion Sorting



### Analysis of Insertion Sort

The running time depends on the input. An already sorted sequence is easier to sort.

*The time complexity of insertion sort in all these cases is given below*

**Worst case** Input is in reverse sorted order

$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2)$$

**Average case** All permutations equally likely

$$T(n) = \sum_{j=2}^n \Theta(j/2) = \Theta(n^2)$$

**Best case** If array is already sorted

$$T(n) = O(n)$$

## Merge Sort

Merge sort is a comparison based sorting algorithm. Merge sort is a stable sort, which means that the implementation preserves the input order of equal elements in the sorted output array. Merge sort is a divide conquer algorithm.

The performance is independent of the initial order of the array items.

**Pseudo Code of Merge Sort**

```
void mergeSort (int a [ ], int first, int last)
{
    if (first < last)
    {
        int mid = (first + last)/2;
        mergeSort (a, first, mid);
        mergeSort (a, mid + 1, last);
                //Divide the array into pieces
        merge (a, first, mid, last);
                //Small pieces are merged
    }
}

void merge (int a [ ],
            int first, int mid, int last)
{
    int c[100]; // temporary array
    int i, j, k, m;
    i = first ;
    j = mid;
    k = mid +1;
    m = first;
    while (first <= last)
    {
        while (i <= j && k <= last)
        {
            if (a [i] < a [k])
            {
                c[m] = a [i];
                i++;
                m++;
            }
            else
            {
                c[m] = a [k];
                m++;
                k++;
            }
        }
        while (i <= j)
        {
```

```

        c [m] = a [i];
        i++;
        m++;
    }
    while (k <= last)
    {
        c [m] = a [k];
        m++;
        k++;
    }
    for (i = 0; i <= last; i++)
    {
        a [i] = c [i];
    }
}
}

```

## Key Points

*There are three main steps in merge sort algorithm*

- ♦ Divide An array into halves, until only one piece left.
- ♦ Sort Each half portion.
- ♦ Merge The sorted halves into one sorted array.

## Analysis of Merge Sort

The merge sort algorithm always divides the array into two balanced lists.

So, the recurrence relation for merge sort is

$$T(n) = \begin{cases} 1 & \text{if } n < = 1 \\ 2T(n/2) + 4n & \text{otherwise} \end{cases}$$

Using master method,

$$T(n) = 2T(n/2) + 4n$$

$$a = 2, b = 2, f(n) = 4n$$

$$h(n) = \frac{f(n)}{n^{\log_2 b}} = \frac{4n}{n^{\log_2 2}} = \frac{4n}{n} = 4(\log_2 n)^0$$

So,  $U(n) = \frac{(\log_2 n)^1}{1} = O(\log_2 n)$  // 4 is constant. So, ignore this

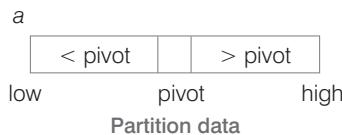
$$\begin{aligned} T(n) &= n^{\log_b a} \cdot U(n) = n^{\log_2 2} \cdot O(\log_2 n) \\ &= O(n \log_2 n) \end{aligned}$$

The time complexity of merge sort in all three case (best, average and worst) is given below

- Best case  $O(n \log_2 n)$
- Average case  $O(n \log_2 n)$
- Worst case  $O(n \log_2 n)$

## Quick Sort

It is in-place sorting, since it uses only a small auxiliary stack. It is also known as partition exchange sort. Quick sort first divides a large list into smaller sublists. Quick sort can then recursively sort the sublists.



## Key Points

- ♦ It is a divide and conquer algorithm.
- ♦ In quick sort algorithm, pivot element is selected and arrange all the items in the lower part are less than the pivot and all those in the upper part greater than it.

The pseudo code of quick sort

```

int quicksort (int a [ ], int low, int high)
{
    int pivot;
    if (low < high)
    {
        pivot = partition (a, low, high);
        quicksort (a, low, pivot - 1);
        quicksort (a, pivot + 1, high);
    }
}
int partition (int a [ ], int low, int high)
{
    int temp;
    int i = low;
```

```

int p = low;
int q = last;
while (i < j)
{
    while (a [p] <= a [i] && i < last)
    {
        i++;
    }
    while (a[q] > a [i])
    {
        j--;
    }
    if (p < q)
    {
        temp = a [p];
        a [p]= a [q];
        a [q]= temp;
    }
}
temp = a [i];
a [i] = a [q];
a [q] = temp;
return q;
}

```

### **Analysis of Quick Sort**

The time complexity calculation of quick sort in all three cases is given below

**Worst case  $O(n^2)$**  This happens when the pivot is the smallest (or the largest) element. Then, one of the partitions is empty and we repeat recursively the procedure of  $n - 1$  elements.

The recurrence relation for worst case

$$T(n) = T(n - 1) + cn, n > 1$$

$$T(n) = T(n - 2) + c(n - 1) + cn$$

$$T(n) = T(n - 3) + c(n - 2) + c(n - 1) + cn$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

Repeat upto  $K - 1$  times, we get

$$\begin{aligned} T(n) &= T(1) + c(2 + 3 + \dots + n) \\ &= 1 + c\left(\frac{n(n+1)}{2} - 1\right) \\ T(n) &= O(n^2) \end{aligned}$$

**Best case**  $O(n \log_2 n)$  The pivot is in the middle and the subarrays divide into balanced partition every time. So, the recurrence relation for this situation

$$T(n) = 2T(n/2) + cn$$

Solve this, using master method

$$\begin{aligned} a &= 2, b = 2, f(n) = cn \\ h(n) &= \frac{cn}{n^{\log_b a}} = \frac{cn}{n^{\log_2 2}} = c (\log_2 n)^0 \\ U(n) &= O(\log_2 n) \\ \text{Then, } T(n) &= n^{\log_b a} U(n) \\ &= n \cdot O(\log_2 n) \\ T(n) &= O(n \log_2 n) \end{aligned}$$

**Average case**  $O(n \log_2 n)$  Quick sort's average running time is much closer to the best case than to the worst case.

Size of Left Partition	Size of Right Partition
1	$n - 1$
2	$n - 2$
3	$n - 3$
$\vdots$	$\vdots$
$n - 3$	3
$n - 2$	2
$n - 1$	1

The average value of  $T(i)$  is  $\frac{1}{N}$  times, the sum of  $T(0)$  through  $T(n - 1)$

$$\Rightarrow \frac{1}{n} \sum T(j), j = 0 \text{ to } n - 1$$

$$\Rightarrow T(n) = \frac{2}{n} (\sum T(j)) + cn$$

(Here we multiply with 2 because there are two partitions)

On multiply by  $n$ , we get

$$n T(n) = 2(\sum T(j)) + cn * n$$

To remove the summation, we rewrite the equation for  $n - 1$ .

$$(n - 1) T(n - 1) = 2 (\sum T(j)) + c(n - 1)^2, j = 0 \text{ to } n - 1 \text{ and subtract}$$

$$nT(n) - (n - 1) T(n - 1) = 2T(n - 1) + 2cn - c$$

Rearrange the terms, drop the insignificant  $c$

$$\Rightarrow nT(n) = (n + 1) T(n - 1) + 2cn$$

On dividing by  $n(n + 1)$ , we get

$$\begin{aligned} \frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{2c}{n+1} \\ \Rightarrow \frac{T(n-1)}{n} &= \frac{T(n-2)}{(n-1)} + \frac{2c}{n} \\ \Rightarrow \frac{T(n-2)}{n-1} &= \frac{T(n-3)}{(n-2)} + \frac{2c}{n-1} \\ &\vdots && \vdots && \vdots \\ \Rightarrow \frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{2c}{3} \end{aligned}$$

Add the equations and cross equal terms

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c \sum \left( \frac{1}{j} \right), j = 3 \text{ to } n + 1$$

$$T(n) = (n + 1) (1/2 + 2c \sum (1/j))$$

The sum  $\sum (1/j), j = 3 \text{ to } (n - 1)$  is about  $\log_2 n$

$$T(n) = (n + 1) (\log_2 n)$$

$$T(n) = O(n \log_2 n)$$

### Advantages of Quick Sort Method

- One of the fastest algorithms on average.
- It does not need additional memory, so this is called in-place sorting.

## Heap Sort

Heap sort is simple to implement and is a comparison based sorting. It is in-place sorting but not a stable sort.

**Note** Heap sorting uses a heap data structure.

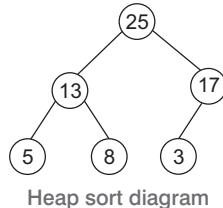
**Heap Data Structure** Heap A is a nearly complete binary tree.

Height of node = Height of edges on a longest simple path from the node down to a leaf

$$\begin{aligned}\text{Height of heap} &= \text{Height of root} \\ &= \theta(\log n)\end{aligned}$$

e.g., We represent heaps in level order, going from left to right. The array corresponding to the heap at the right side is [25, 13, 17, 5, 8, 3].

The root of the tree  $A[1]$  and given index  $i$  of a node, the indices of its parent, left child and right child can be computed.



Heap sort diagram

```

Parent (i)
    return floor (i/2)
Left (i)
    return 2i
Right (i)
    return 2i + 1
  
```

### Heap Property

The heap properties based on the comparison of value of node and its parent are as given below

**Max heap** In a heap, for every node  $i$ , other than the root, the value of a node is less than or equal (atmost) to the value of its parent.

$$A[\text{parent}(i)] \geq A[i]$$

**Min heap** In a heap, for every node  $i$ , other than the root, the value of a node is greater than or equal to the value of its parent.

$$A[\text{parent}(i)] \leq A[i]$$

### Maintaining the Heap Property

Max heapify is important for manipulating max heap. It is used to maintain the max heap property.

The following conditions will be considered for maintaining the heap property are given as

- Before max heapify,  $A[i]$  may be smaller than its children.
- Assume left and right subtree of  $i$  are max heaps.
- After max heapify, subtree rooted at  $i$  is a max heap.

### Pseudo Code for Max Heapify

MaxHeapify (int a [ ], i, n)

```

{
    int largest, temp;
    int l = 2*i;
    int r = 2*i + 1;
    if (l ≤ n && a [l] > a [i])
    {
        largest = l;
    }
    else
        largest = i;
    if (r ≤ n && a [r] > a [largest])
    {
        largest = r;
    }
    if (largest != i)
    {
        temp = a [largest];
        a [largest] = a [i];
        a [i] = temp;
    }
    MaxHeapify (a, largest, n);
}

```

Basically, three basic operations performed on heap are given as

- Heapify, which runs in  $O(\log_2 n)$  time.
- Build heap, which runs in linear time  $O(1)$ .
- Heap sort, which runs in  $O(n \log_2 n)$  time.

We have already discussed the first operation of heap sort. Next we will discuss how to build a heap.

## **Building a Heap**

The following procedure given an unordered array, will produce a max heap

```

BuildMaxHeap (int a [ ], int n)
{
    int i;
    for (i = (n/2); i > 1; i --)
    {
        Maxheapify (a, i, n);
    }
}

```

## Heap Sort Algorithm

The heap sort algorithm combines the best of both merge and insertion sort. Like merge sort, the worst case time of heap sort is  $O(n \log_2 n)$  and like insertion sort, heap sort is in-place sort.

### Pseudo code for heap sort procedure

```

Heapsort (int a [ ], int n)
{
    int i, temp;
    BuildMaxHeap (a, n);
    for (i = n; i >= 2; i - -)
    {
        temp = a[1];
        a[1] = a [i];
        a [i] = temp;
        heapsize [a] = heapsize [a] - 1;
    }
    Heapify (a, 1);
}

```

## Analysis of Heap Sort

The total time for heap sort is  $O(n \log n)$  in all three cases  
(i.e., best case, average case and worst case)

# Hashing

Hashing is a common method of accessing data records. Hashing is the process of mapping large amount of data item to a smaller table with the help of hashing function.

### Linear Search *versus* Hashing

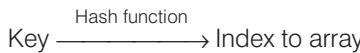
- A linear search looks down a list, one item at a time, without jumping. In complexity terms, this is an  $O(n)$  search, the time taken to search the list.
- If we use hashing to search, data need not be sorted. Without collision and overflow, search only takes  $O(1)$  time.

## Hash Table

A hash system that stores records in an array, called a hash table. A hash table uses a hash function to compute an index into an array of buckets, from which the correct value can be found.

## Hash Function

- Hash function primarily is responsible to map between the original data items and the smaller table or in other words, we can say that the fixed process, to convert a key into a hash key is known as a hash function.
- A hash function is used whenever access to the table is needed.



- A good hash function satisfies (approx) the assumption of simple uniform hashing. Each key is equally likely to hash to any of the  $m$  slots, independently of where any other key has hashed to.

*There are many hash functions approaches as follows*

### Division Method

Mapping a key  $K$  into one of  $m$  slots by taking the remainder of  $K$  divided by  $m$ .

$$h(K) = K \bmod m$$

e.g., Let's say, there are three numbers or keys that we want to map into the hash table of 10 slots

1 2 3 4 5 6,    1 2 3 4 6 7,    1 2 3 4 5 0

key 1              key 2              key 3

0	1	2	3	4	5	6	7	8	9

- 123456% 10  $\Rightarrow$  6

The remainder is 6, so the key would be placed at 6th index.

- 123467% 10  $\Rightarrow$  7

The remainder is 7, so the key would be placed at 7th index.

- 123450% 10  $\Rightarrow$  0

The remainder is 0, so the key would be placed at 0th index.

### Mid-Square Method

Mapping a key  $K$  into one of  $m$  slots, by getting the some middle digits from value  $K^2$ .

$$h(k) = K^2 \text{ and get middle } (\log_{10} m) \text{ digits}$$

### Folding Method

Divide the key  $K$  into some sections, besides the last section, have same length. Then, add these sections together.

- Shift folding
- Folding at the boundaries

$$h(K) = \Sigma (\text{section divided from } K) \text{ by } a \text{ or } b$$

e.g., Key 123 20 03 241112 20, section length = 3

	<table border="1"><tr><td>1 2 3</td></tr></table>	1 2 3	<table border="1"><tr><td>2 0 3</td></tr></table>	2 0 3	<table border="1"><tr><td>2 4 1</td></tr></table>	2 4 1	<table border="1"><tr><td>1 1 2</td></tr></table>	1 1 2	<table border="1"><tr><td>2 0</td></tr></table>	2 0
1 2 3										
2 0 3										
2 4 1										
1 1 2										
2 0										
Sections	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$					

(a) If we use shift folding method, then add all the sections

$S_1$	123
$S_2$	203
$S_3$	241
$S_4$	112
$S_5$	20
	<u>879</u>

(b) If we use folding at the boundaries method, then

$S_1$	123
$S_2$	302
$S_3$	241
$S_4$	211
$S_5$	20
	<u>879</u>

# Collision

No matter what the hash function, there is the possibility that two different keys could resolve to the same hash address. This situation is known as a collision.

When this situation occurs, *there are two simple methods to this problem solve.*

## Chaining

A chain is simply a linked list of all the elements with the same hash key. The hash table slots will no longer hold a table element (key). They will now hold the address of a table element

$$h(K) = \text{key \% table slots}$$

e.g., The following keys to be hashed

36, 18, 72, 43, 6, 10, 5, 15

There are 8 slots in the hash table. So, first of all we have to calculate the hash index of each key.

36 % 8 = 4	0	→	72	☒
18 % 8 = 2	1			
72 % 8 = 0	2	→	18	☒
43 % 8 = 3	3	→	43	☒
6 % 8 = 6	4	→	36	☒
10 % 8 = 2	5	→	5	☒
5 % 8 = 5	6	→	6	☒
15 % 8 = 7	7	→	15	☒

Hash table for the keys

## Hashing with Linear Probing

When using a linear probing method the item will be stored in the next available slot in the table, assuming that the table is not already full. This is implemented via a linear searching for an empty slot, from the point of collision. If the end of table is reached during the linear search, the search will wrap around to the beginning of the table and continue from there.

### Key Points

- If an empty slot is not found before reaching the point of collision, the table is full.
- In chaining, we put all the elements, which go to the same slot, in a linked list.

e.g., Suppose the keys to be inserted or hashed are as follows

36, 18, 72, 43, 6, 10, 5, 15

and there are 8 slots in hash table.

Now, we have to find out the hash address of keys.

$36 \% 8 = 4 \leftarrow$  inserted

$18 \% 8 = 2 \leftarrow$  inserted

$72 \% 8 = 0 \leftarrow$  inserted

$43 \% 8 = 3 \leftarrow$  inserted

$6 \% 8 = 6 \leftarrow$  inserted

$10 \% 8 = 2 \leftarrow$  problem is here because 2nd place is already full

$5 \% 8 = 5 \leftarrow$

$15 \% 8 = 7 \leftarrow$

0	72
1	15
2	18
3	43
4	36
5	10
6	6
7	5

Hash table for the keys

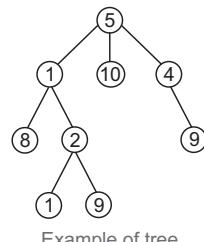
So, when the hash index, what we get through the hash function is already full, then search the next empty slot. So, for key 10, 2nd index is already full, then search the 3rd, it is also full, then search the 4th, it is also full, the search 5th index it is empty, then fill with key 10.

## Tree

A tree is the data structure that are based on hierarchical tree structure with set of nodes. It is a acyclic connected graph with one or more children nodes and at most one root node.

In a tree, there are two types of node

- **Internal nodes** All nodes those have children nodes are called as internal nodes.
- **Leaf nodes** Those nodes, which have no child, are called leaf nodes.



Example of tree

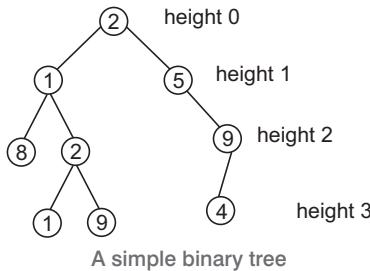
## Tree terminology

The depth of a node is the number of edges from the root to the node.

The height of a node is the number of edges from the node to the deepest leaf. The height of a tree is the height of the root.

## Binary Tree

A binary tree is a tree like structure that is rooted and in which each node has at most two children and each child of a node is designated as its left or right child.

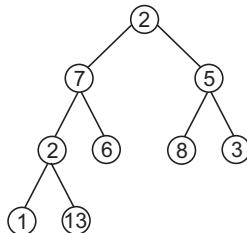


## Types of Binary Trees

The types of binary trees depending on the trees structure are as given below

### Full Binary Tree

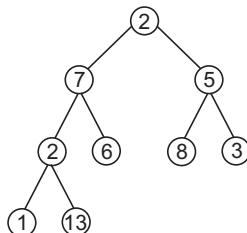
A full binary tree is a tree in which every node other than the leaves has two children or every node in a full binary tree has exactly 0 or 2 children.



Full binary tree

### Complete Binary Tree

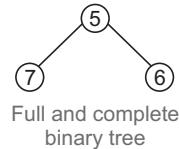
A complete binary tree is a tree in which every level, except possibly the last, is completely filled.



Complete binary tree

The number  $n$  of nodes in a binary tree of height  $h$  is atleast  $n = h + 1$  and atmost

$$n = 2^{h+1} - 1, \text{ where } h \text{ is the depth of the tree}$$



## Tree Traversal

There are three standard ways of traversing a binary tree  $T$  with root  $R$ . These three ways are given below

### Preorder

- Process the root  $R$ .
- Traverse the left subtree of  $R$  in preorder.
- Traverse the right subtree of  $R$  in preorder.

### Inorder

- Traverse the left subtree of  $R$  in inorder.
- Process the root  $R$ .
- Traverse the right subtree of  $R$  in inorder.

### Postorder

- Traverse the left subtree of  $R$  in postorder.
- Traverse the right subtree of  $R$  in postorder.
- Process the root  $R$ .

## Breadth First Traversal (BFT)

The breadth first traversal of a tree visits the nodes in the order of their depth in the tree. Breadth first traversal first visits all the nodes at depth zero (i.e., root), then all the nodes at depth one and so on.

## Depth First Traversal (DFT)

Depth first traversal is an algorithm for traversing or searching a tree, tree structure or graph. One starts at the root and explores as far as possible along each branch before backtracking.

### Key Points

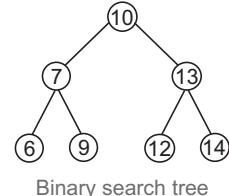
- ♦ Queue data structure is used for this traversal.
- ♦ Stack data structure is used for traversal

## Binary Search Tree

A binary search tree is a binary tree, which may also be called an ordered or sorted binary tree.

*A binary search tree has the following properties*

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left subtrees and right subtrees must also be binary search trees.



### Pseudo Code for Binary Search Tree (BST)

The pseudo code for BST is same as binary tree because it also has node with data, pointer to left child and a pointer to right child.

```

struct node
{
    int data;
    struct node * left;
    struct node * right;
}
// A function that allocates a new node with the given
// data and NULL // left and right pointers
struct node* newNode (int data)
{
    struct node* newNode = (struct node*) malloc (sizeof
    (struct node));
    newNode → data = data;
    newNode → left = NULL;
    newNode → right = NULL;
    return (newNode);
}
    
```

## Insert a Node into BST

/\* Given a binary search tree and a number, inserts a new node with the given number in the correct place in the tree\*/

```

struct node * insert (struct node* node, int data)
{
    /* If the tree is empty, return a new, single node */
    if (node == NULL)
        return (newNode (data));
    
```

```

        else
    {
        /* Otherwise, insert a newnode at the
        appropriate place, maintaining the property
        of BST */
        if (data < node → data)
            node → left = insert (node → left, data);
        else
            node → right = insert (node→ right, data);
        return node;
    }
}

```

In a BST, insertion is always at the leaf level, traverse the BST, comparing the new value to existing ones, until you find the right point, then add a new leaf node holding that value.

## Deletion From a Binary Search Tree

Deletion is the most complex operation on a BST, because the algorithm must result in a BST. The question is “what value should replace the one deleted?”

Basically, to remove an element, we *have to follow two steps*

- Search for a node to remove.
- If the node is found, run remove algorithm.

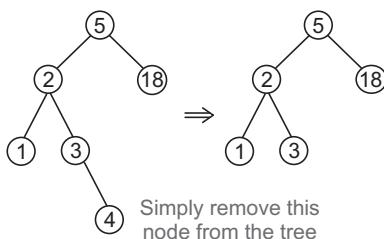
## Procedure for Deletion of a Node From BST

*There are three cases to consider*

### Deleting a Leaf Node (Node with No Children)

Deleting a leaf node is easy, as we can simply remove it from the tree.

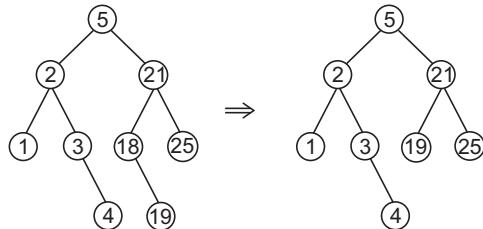
Remove 4 from a BST given below



## Deleting a Node with One Child

Remove the node and replace it with its child.

*Remove 18 from a BST given below*



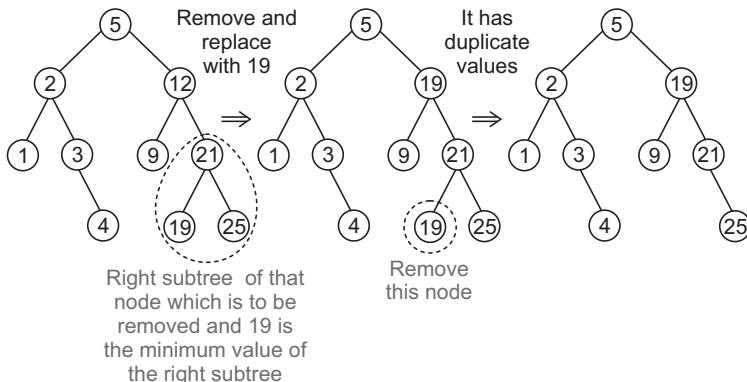
- In this case, node is cut from the tree and node's subtree directly links to the parent of the removed node

## Deleting a Node with Two Children

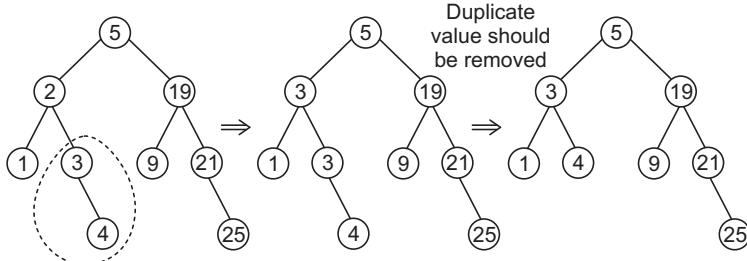
This is the most complex case to solve it, we use the approach as given below

- Find a minimum value in the right subtree.
- Replace value of the node to be removed with found minimum. Now, right subtree contains a duplicate.
- Apply remove to the right subtree to remove a duplicate.

e.g., Remove 12 from the given BST



Now, if we delete 2 from this BST, then



In the right subtree 3 is minimum, so 2 should be replaced with the value 3

## Implementation of BST Deletion

```
void deleteNode (striuct node * treePtr, int value)
{
    struct node rightPtr, leftPtr;
    struct node current = * treePtr, * parentPtr, * temp;
    if (! current)
    {
        printf ("The tree is empty or has not this node");
        return;
    }
    if (current → data == value)
    {
        /* It's a leaf node */
        if ( ! current → rightPtr && ! current → leftPtr)
        {
            * treePtr = NULL;
            free (current);
        }
        /* the right child is NULL and left child
        is not null t */
        else if ( ! current → rightPtr)
        {
            * treePtr = current → leftPtr;
            free (current);
        }
    }
}
```

```
        /* the node has both children */
        else
    }
    temp = current → rightPtr;
    /* the rightPtr with left child */
    if (! temp → leftPtr)
        temp → leftPtr = current → leftPtr;
    /* the rightptr have left child */
    else
    {
        /* find the smallest node in right subtree */
        while (temp → leftPtr)
        {
            /* record the parent node of temp */
            parentPtr = temp;
            temp = temp → leftPtr;
        }
        parentPtr → leftPtr = temp → rightPtr;
        temp → leftPtr = current → leftPtr;
        temp → rightPtr = current → rightPtr;
    }
    * treePtr = temp;
    tree (current);
}
}
else if (value > current → data)
{
    /* search the node */
    deleteNode (& (current → rightPtr), value);
}
else if (value < current → data)
{
    deleteNode (& (current → leftPtr),
value);
}
```

## Average Case Performance of Binary Search Tree Operations

Internal Path Length (IPL) of a binary tree is the sum of the depths of its nodes. So, the average internal path length  $T(n)$  of binary search trees with  $n$  nodes is  $O(n \log n)$ .

The average complexity to find or insert operations is  $T(n) = O(\log n)$ .

### Improving the Efficiency of Binary Search Trees

- Keeping the tree balanced.
- Reducing the time for key comparison.

## Balanced Trees

Balancing ensures that the internal path lengths are close to the optimal  $n \log n$ . A balanced tree will have the lowest possible overall height.

*There are many balanced trees*

1. AVL Trees
2. B-Trees

### AVL Trees

An AVL (Adelson-Velskii and Landis) is a binary tree with the following additional balance property.

- For any node in the tree, the height of the left and right subtrees can differ by atmost.
- The height of an empty subtree is  $-1$ .

The method to keep the tree balanced is called node rotation. AVL trees are binary trees those use node rotation upon inserting a new node.

*An AVL is a binary search tree which has the following properties*

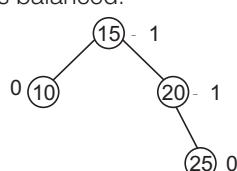
- The subtrees of every node differ in height by atmost one.
- Every subtree is an AVL tree.

Every node of an AVL tree is associated with a balance factor.

Balance factor of a node = Height of left subtree – Height of right subtree

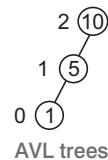
A node with balance factor  $-1, 0$  or  $1$  is considered as balanced.

An AVL tree is shown here



This AVL tree is balanced and a BST.

It is a BST but not AVL tree because it is not balanced. At node 10, the tree is unbalanced. So, we have to rotate this tree.

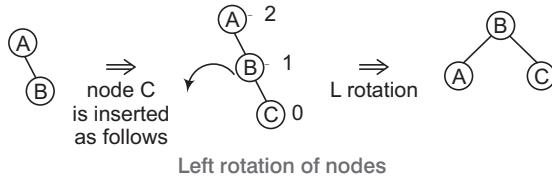


## Rotations

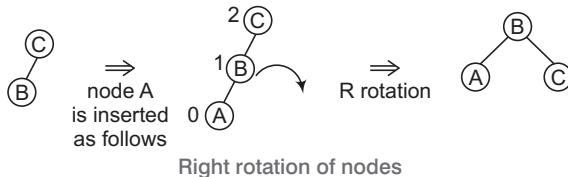
A tree rotation is required when we have inserted or deleted a node which leaves the tree in an unbalanced form.

*There are four types of rotations as given below*

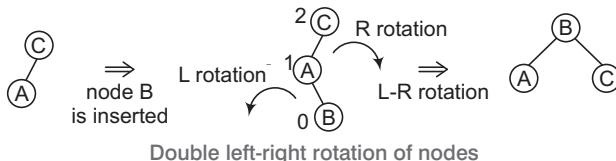
**Left rotation (L-rotation)** Suppose we have AVL tree



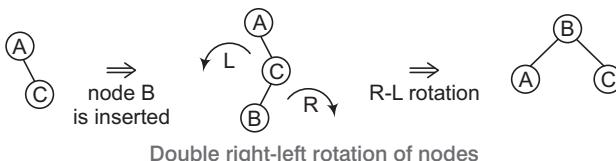
**Right rotation (R-rotation)** Suppose, we have an AVL tree



**Double left-right rotation (L-R rotation)** Suppose, we have an AVL tree



**Double right-left rotation (R-L rotation)** Suppose, we have an AVL tree.



## **Greedy Algorithms**

Greedy algorithms are simple and straight forward. They are short sighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect of this decisions in the future. *They are as follows*

- Easy to implement.
- Easy to invent and require minimal amount of resources.
- Quite efficient.

**Note** Greedy approaches are used to solve optimization problems.

## **Feasibility**

A feasible set is promising, if it can be extended to produce not merely as solution, but an optimal solution to the problem. Unlike dynamic programming which solves the subproblems bottom-up strategy, a Greedy strategy usually progress in a top-down fashion, making one Greedy choice after another, reducing each problem to a smaller one.

## **Optimization Problems**

An optimization problem is one in which you want to find not just a solution, but the best solution. A Greedy algorithm sometimes works well for optimization problems.

## **Examples of Greedy Algorithms**

*The examples based on Greedy algorithms are given below*

### **Activity Selection Problem**

If there are  $n$  activities,  $S = \{1, 2, \dots, n\}$ , where  $S$  is the set of activities. Activities are to be scheduled to use some resources, where each activity  $i$  has its starting time  $S_i$  and finish time  $f_i$  with  $S_i \leq f_i$ , i.e., the two activities  $i$  and  $j$  are non-interfering, if their start-finish intervals do not overlap. So,  $(S_i, f_i) \cap (S_j, f_j) = \emptyset$

The activities  $i$  and  $j$  are compatible, if their time periods are disjoint. In this algorithm, the activities are firstly sorted in an increasing order, according to their finish time. With the Greedy strategy, we first select the activity with smallest interval ( $F_i - S_i$ ) and schedule it, then skip all activities, that are not compatible with the current selected activity and repeats this process until all the activities are considered.

### Algorithm for Activity Selection Problem

```

Activity_Selection_Problem (S, f, i, j)
{
    A ← i
    m ← i + 1
    while m < j and Sm < fi
        do m ← m + 1
    if m < j
        then return A = A ∪ Activity_Selection_Problem (S, f, m, j)
    else
        return φ
}

```

The running time of this algorithm is  $\Theta(n)$  assuming that the activities are already sorted.

## Fractional Knapsack Problem

There are  $n$  items;  $i$ th item is worth  $v_i$  dollars and weights  $w_i$  pounds, where  $v_i$  and  $w_i$  are integers. Select items to put in Knapsack with total weight  $\leq W$ , so that total value is maximised. In fractional Knapsack problem, fractions of items are allowed.

### Key Points

- ♦ To solve the fractional problem, we have to compute the value per weight for each item.
- ♦ According to Greedy approach, item with the largest value per weight will be taken first.
- ♦ This process will go on until we can't carry any more.

### Algorithm of Fractional Knapsack

```

Greedy_Fractional_Knapsack (G, n, w)
{
    /* G is already sorted by the value per weight of items
       order */
    Rw = W, i = 1
    while ((Rw > 0) and (i ≤ n)) do
        if (Rw ≥ wi) then fi = 1
        else fi = Rw/wi
        Rw = Rw - fi * wi
}

```

```

    i ++
end while
    return F
}

```

where  $F$  is the filling pattern i.e.,  $F = \{f_1, f_2, \dots, f_n\}$  and  $0 \leq f_i \leq 1$  denotes the fraction of the  $i$ th item.

### Analysis

If the items are already sorted into the decreasing order of value per weight, then the algorithm takes  $O(n)$  times.

## Task Scheduling Problem

There is a set of  $n$  jobs,  $S = \{1, 2, \dots, n\}$ , each job  $i$  has a deadline  $d_i \geq 0$  and a profit  $p_i \geq 0$ . We need one unit of time to process each job. We can earn the profit  $p_i$ , if job  $i$  is completed by its deadline. To solve the problem of task scheduling sort the profit of jobs ( $p_i$ ) into non-decreasing order.

After sorting, we will take the array and maximum deadline will be the size of array. Add the job  $i$  into the array, if job  $i$  can be completed by its deadline. Assign  $i$  to the array slot of  $[r - 1, r]$ , where  $r$  is the largest such that  $1 \leq r \leq d_i$ . This process will be repeated, until all jobs are examined.

**Note** The time complexity of task scheduling algorithm is  $O(n^2)$ .

## Minimum Spanning Tree Problem

**Spanning Tree** A spanning tree of a graph is any tree that includes every vertex in the graph. More formally, a spanning tree of a graph  $G$  is a subgraph of  $G$  that is a tree and contains all the vertices of  $G$ . Any two vertices in a spanning tree are connected by a unique path. The number of spanning trees in the complete graph  $K_n$  is  $n^{n-2}$ .

**Minimum Spanning Tree** A Minimum Spanning Tree (MST) of a weighted graph  $G$  is a spanning tree of  $G$  whose edges sum is minimum weight.

*There are two algorithms to find the minimum spanning tree of an undirected weighted graph.*

## Kruskal's Algorithm

Kruskal's algorithm is a Greedy algorithm. In this algorithm, starts with each vertex being its component. Repeatedly merges two components into one by choosing the light edge that connects them, that's why, this algorithm is edge based algorithm. According to this algorithm

Let  $G = (V, E)$  is a connected, undirected, weighted graph.

Scans the set of edges in increasing order by weight. *The edge is selected such that*

- Acyclicity should be maintained.
- It should be minimum weight.
- When tree  $T$  contains  $n - 1$  edges, also must terminate.

Uses a disjoint set of data structure to determine whether an edge connects vertices in different components.

**Data structures** Before formalizing the above idea, we must review the disjoint set of data structure.

**MAKE\_SET ( $v$ )** Create a new set whose only member is pointed to by  $v$ . Note that for this operation,  $v$  must already be in a set.

**FIND\_SET ( $v$ )** Returns a pointer to the set containing  $v$ .

**UNION ( $u, v$ )** Unites the dynamic sets that contain  $u$  and  $v$  into a new set that is union of these two sets.

## Algorithm for Kruskal

Start with an empty set  $A$  and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in the graph.

```

KRUSKAL ( $V, E, w$ )
{
    A  $\leftarrow \emptyset$ 
    for each vertex  $v \in V$ 
        do MAKE_SET ( $v$ )
    Sort  $E$  into non-decreasing order by weight  $w$ 
    for each  $(u, v)$  taken from the sorted list
        do if FIND_SET ( $u$ )  $\neq$  FIND_SET ( $v$ )
    then  $A \leftarrow A \cup \{(u, v)\}$ 
        UNION ( $u, v$ )
    return A
}

```

## Analysis of Kruskal's Algorithm

The total time taken by this algorithm to find the minimum spanning tree is  $O(E \log_2 E)$  (if edges are already sorted).

But the time complexity, if edges are not sorted is  $O(E \log_2 V)$ .

## Prim's Algorithm

Like Kruskal's algorithm, Prim's algorithm is based on a generic minimum spanning tree algorithm. The idea of Prim's algorithm is to find the shortest path in a given graph. The Prim's algorithm has the property that the edges in the set  $A$  always form a single connected tree.

In this, we begin with some vertex  $v$  in a given graph  $G = (V, E)$ , defining the initial set of vertices  $A$ . Then, in each iteration, we choose a minimum weight edge  $(u, v)$ , connecting a vertex  $v$  in the set  $A$  to the vertex  $u$  outside of set  $A$ . Then, vertex  $u$  is brought into  $A$ . This tree is repeated, until a spanning tree is formed.

*This algorithm uses priority queue  $Q$ .*

- Each object is a vertex in  $V - V_A$ .
- Key of  $v$  is minimum weight of any edge  $(u, v)$ , where  $u \in V_A$ .
- Then, the vertex returned by EXTRACT\_MIN is  $v$  such that there exists  $u \in V_A$  and  $(u, v)$  is a light edge crossing  $(V_A, V - V_A)$ .

## Algorithm for Prim

```

PRIM (V, E, w, r)
{
    Q ← φ
    for each u ∈ V
        do key [u] ← ∞
           Π [u] ← NIL
           INSERT (Q, u)
           DECREASE_KEY (Q, r, o)
    while Q ≠ φ
        do u ← EXTRACT_MIN (Q)
           for each v ∈ Adj [u]
               do if v ∈ Q and w (u, v) < key [v]
                  then Π [v] ← u
                  DECREASE_KEY (Q, v, w (u, v))
}

```

## Analysis of Prim's Algorithm

The time complexity for Prim's algorithm is  $O (E \log_2 V)$ .

## Dynamic Programming Algorithms

Dynamic programming approach for the algorithm design solves the problems by combining the solutions to subproblems, as we do in divide and conquer approach. As compared to divide and conquer, dynamic programming is more powerful.

Dynamic programming is a stage-wise search method suitable for optimization problems whose solutions may be viewed as the result of a sequence of decisions.

### Greedy versus Dynamic Programming

- For many optimization problems, a solution using dynamic programming can be unnecessarily costly. The Greedy algorithm is simple in which each step chooses the locally best. The drawback of Greedy method is that the computed global solution may not always be optimal.
- Both techniques are optimization techniques and both build solution from a collection of choices of individual elements. Dynamic programming is a powerful technique but it often leads to algorithms with higher running times.
- Greedy method typically leads to simpler and faster algorithms.
- The Greedy method computes its solution by making its choices in a serial forward, never looking back or revising previous choices.
- Dynamic programming computes its solution bottomup by synthesising them from smaller subsolutions and by trying many possibilities and choices before it arrives the optimal set of choices.

## Approaches of Dynamic Programming

*The dynamic programming consists of following sections*

### 0-1 Knapsack

The idea is same as fractional Knapsack but the items may not be broken into smaller pieces, so we may decide either to take an item or to leave it, but we may not take a fraction of item. Only dynamic programming algorithm exists.

### Single Source Shortest Paths

In general, the shortest path problem is to determine one or more shortest path between a source vertex and a target vertex, where a set of edges are given. We are given a directed graph  $G = (V, E)$  with non-negative edge weight and a distinguished source vertex,  $s \in V$ . The problem is to determine the distance from the source vertex to every other vertex in the graph.

## Dijkstra's Algorithm

Dijkstra's algorithm solves the single source shortest path problem on a weighted directed graph. It is Greedy algorithm. Dijkstra's algorithm starts at the source vertex, it grows a tree  $T$ , that spans all vertices reachable from  $S$ .

### Algorithm for Dijkstra

```

DIJKSTRA(V, E, W, S)
{
    INITIALIZE_SINGLE_SOURCE(V, S)
    S ←  $\emptyset$ 
    Q ← V # insert all vertices into Q
    while Q ≠  $\emptyset$ 
        do u ← EXTRACT_MIN(Q)
        S ← S ∪ {u}
        for each vertex v ∈ Adj [u]
            do RELAX(u, v, w)
    }
    INITIALIZE_SINGLE_SOURCE(G, S)
    {
        for each vertex v ∈ V[G]
            do d[v] ←  $\infty$ 
            π[v] ← NIL
            d[S] ← 0
    }
    RELAX(u, v, w)
    {
        if d[v] > d[u] + w (u, v)
            then d[v] ← d[u] + w (u, v)
    } π[v] ← u
        have two sets of vertices

```

$S$  = Vertices whose final shortest path weights are determined  
 $Q$  = Priority queue =  $V - S$ .

# 4

# Database Management System

---

## Database Management System

A database management system is a collection of interrelated data and a set of programs to access those data. It manages new large amount of data and supports efficient access to new large amounts of data.

### Data

Known facts that can be stored or recorded is called data.

*It can be of two types*

**Persistent Data** It continues to exist even, when the system is not active.

**Transient Data** It created while an application is running and not needed, when the application has terminated. It must be stored in secondary memory.

### Database

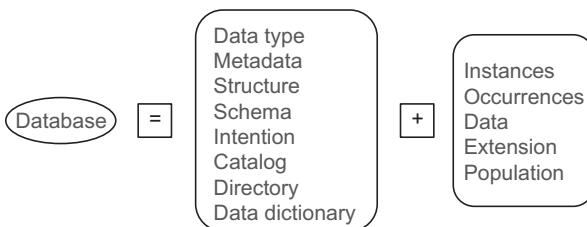
*Database is a collection of related data with*

- A logically coherent structure (can be characterised as a whole).
- Some inherent meaning (represents some partial view of a portion of the real world).
- Some specific purpose and for an intended group of users and applications.
- A largely varying size (from a personal phone book directory to the phone book directory of a city or state).

- A scope or content of varying breadth (from a personal list of addresses to a multimedia encyclopedia).
- A physical organisation of varying complexity (from a manual personal list, managed with simple files, to huge multiuser databases with geographically distributed data and access).

### **Formal Definition of Data**

Logically coordinated objectives, data is defined once for a community of users and accessed by various applications with specific needs.



*The information contained in a database is represented on two levels*

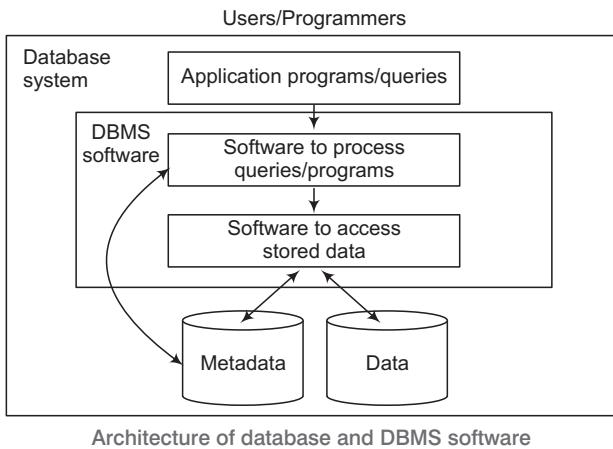
- (i) Data (which is large and is being frequently modified)
- (ii) Structure of data (which is small and stable in time)

### **Basic Concept of DBMS**

It is a collection of general purpose, application independent programs providing services to

- Define the structure of a database, i.e., data types and constraints that the data will have to satisfy.
- Manage the storage of data, safely for long periods of time, on some storage medium controlled by the DBMS.
- Manipulate a database, with efficient user interfaces to query the database to retrieve specific data, update the database to reflect changes in the world, generate reports from the data.
- Manage database usage for users with their access rights, performance optimisation, sharing of data among several users, security from accidents or unauthorised use.
- Monitor and analyse database usage.

Database Management System (DBMS) provides efficient, reliable, convenient and safe multiuser storage of and access to massive amounts of persistent data.



Architecture of database and DBMS software

## Key People Involved in a DBMS

*The followings are the key people involved in a DBMS as*

### DBMS Implementer

Person who builds system.

### Database Designer

Person responsible for preparing external schemas for applications, identifying and integrating user needs into a conceptual (or community or enterprise) schema.

### Database Application Developer

Person responsible for implementing database application programs that facilitate data access for end users.

### Database Administrator

Person responsible for defining the internal schema, sub-schemas (with database designers) and specifying mappings between schemas, monitoring database usage and supervising DBMS functionality (e.g., access control, performance optimisation, backup and recovery policies, conflict management).

### End Users

*There are two categories of end users*

- **Casual users** Occasional unanticipated access to database. e.g., tourists, managers.

- **Parametric users** Users who query and update the database through fixed programs (invoked by non-programmer users) e.g., banking.

### **Important Functions on a Database**

- **Structure Definition** Declare files or relations (i.e.,tables + data types). e.g., Employee (Emp\_Name, Emp\_No, Salary, Dept\_ID)
- **Dept** (Dept\_Name,Dept\_ID)
- **Population** Input data about specific employee, department.
- **Querying** List name of employees who are getting salary more than 3 lakh in department marketing.
- **Reporting** We can prepare a report having employee names with their department name using join on employee table and dept table.
- **Modification and Update of Population** We can create a new department name for the table department and also we can update salary of a particular employee.
- **Modification of Structure and Schema** We can create a new relation (i.e.,table) for '(Emp\_No,Skills).' We can add address attribute to relation employee.

## **Instances and Schemas and Data Model**

**Data Model** It provides mechanisms (languages) for defining data structures and operations for retrieval and modification of data.

**Schema/Intension** The overall design of the database or description of data in terms of a data model. Schemas do not change frequently. In addition to data structures, the schema also comprises the definition of domains for data elements (attributes) and the specification of constraints, to define the data structure part of the schema. While intension is a constant value that gives the name, structure of table and the constraints laid on it.

### **Instance/Extension**

Databases change over time because records are inserted and deleted frequently. The collection of information stored in the database at a particular moment is called an instance of the database. While extension means the number of tuples present in a table at any instance. This is dependent on time.

### **Data Abstraction**

Hiding the complexity from users through several levels of abstraction, to simplify users' interactions with the system, as many database systems users are not computer trained.

## Levels of Data Abstraction

*There are three levels of data abstraction as given below*

### **Physical Level**

It is lowest level of abstraction and describes how the data are actually stored and complex low level data structures in detail. At the physical level, an employee record can be described as a block of consecutive storage locations. e.g., words or bytes.

### **Logical Level**

It is the next higher level of abstraction and describes what data are stored and what relationships exist among those data. At the logical level, each such record is described by a type definition and the interrelationship of these record types is defined as well. Database administrators usually work at this level of abstraction.

### **View Level**

It is the highest level of abstraction and describes only part of the entire database and hides the details of the logical level.

- At the view level, several views of the database are defined and database users see these views.
- The views also provide a security mechanism to prevent users from accessing certain parts of the database. e.g., tellers in a bank can see information on customer accounts but they cannot access information about salaries of employees.

### **Key Points**

- ◆ Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
- ◆ Many users of the database system do not need all stored information.
- ◆ Users classified need to access only a part of the database.
- ◆ The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

# Schema

A schema is also known as database schema. It is a logical design of the database and a database instance is a snapshot of the data in the database at a given instant of time. A **relational schema** consists of a list of attributes and their corresponding domains.

## Types of Schemas

*It can be classified into three parts, according to the levels of abstraction*

**Physical/Internal Schema** Describes the database design at the physical level.

**Logical/Conceptual Schema/Community User View** Describes the database design at the logical level.

**Sub-schemas/View/External Schema** Describes different views of the database, views may be queried, combined in queries with base relations, used to define other views in general, not updated freely.

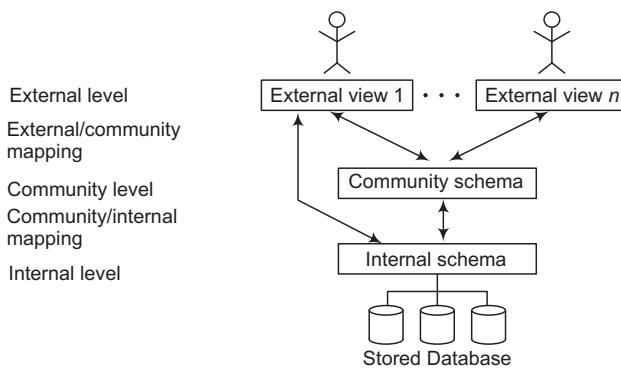
## Schema Architecture

### Data Independence

Possibility to change the schema at one level without having to change it at the next higher level (nor having to change programs that access it at that higher level).

*There are two parts of data independence*

**Logical Data Independence** Refers to the immunity of the external schema to changes in the conceptual schema (i.e., community schema). e.g., add new record or field.



**Physical Data Independence** Refers to the immunity of the conceptual schema to changes in the internal schema. e.g., addition of an index should not affect existing one.

## Database Functions and Application Functions

### Application Program Functions

To be programmed in application programs.

### Database Functions or DBMS Functions

Supplied by the DBMS and invoked in application programs.



## Transaction

One execution of a user program (executing the same programs several times corresponds to several transactions). Basic unit of change as seen by the DBMS.

OLTP (On-Line Transaction Processing) applications (e.g., banking and airline systems) with multiple simultaneous users.

## Functionality of DBMS

- 1. Concurrency control
- 2. Backup and recovery
- 3. Redundancy management
- 4. Access control
- 5. Performance optimisation
- 6. Metadata management
- 7. Active features (rules, triggers)

## Concurrency Control

It is responsible for ensuring correctness of competing accesses to same data. One or more Structure Query Language (SQL) statements altogether treated as one single unit.

Correctness of data requires four desirable properties (ACID properties) e.g., concurrency control means there should not be two simultaneous withdrawals from the same bank account or there should not be multiple reservations of the same airplane seat.

## **Backup and Recovery**

- Facilities for recovering from hardware and software failures.
- If the computer system fails during a complex update program, the database must be restored to its state before the program started, or the program must be resumed, where it was interrupted so that its full effect is recorded in the database.
- In a multiuser environment, it is more complex and important.

## **Redundancy Management**

Redundancy means storing several copies of the same data. Redundant entries are frequent in traditional file processing; a goal of the database approach was to control redundancy as much as possible.

Problems with redundancy includes waste of storage space, duplication of effort to perform a single conceptual update, danger of introducing inconsistency, if multiple updates are not coordinated.

## **Access Control**

Responsible for enforcing security and authorisation (e.g., who can create new bank accounts) and data (e.g., which bank accounts can I see).

It is all about who accesses what data, to do what, when, from where etc.

*Some examples of access privileges are as follows*

- To create a database
- To authorise (grant) additional users to access the database, access some relations, create new relations and update the database.
- To revoke privileges.

## **Key Points**

---

- ♦ In a multiuser database, access control is mandatory e.g., for confidentiality.
  - ♦ Various ways to access data (e.g., read only, read and update).
  - ♦ The data dictionary holds information about users and their access privileges (e.g., name and password).
- 

## **Performance Optimisation**

Performing physical reorganisations to enhance performance e.g., adding index, dropping index, sorting file.

Performance optimisation is made possible by physical data independence and high level data models with user programs which can be optimised through DBMS software.

## Metadata Management

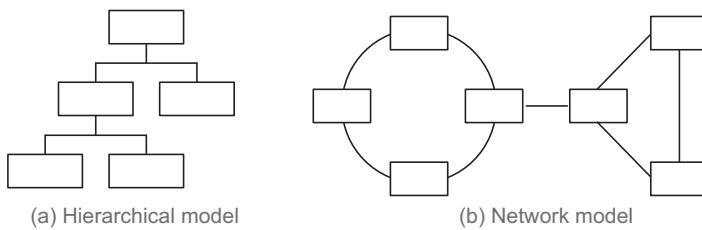
Metadata means data about data. Metadata is maintained in a special database, which we can call system catalog or data dictionary. It involves storing of information about other information. With different types of media being used, references to the location of the data can allow management of diverse repositories.

## Active Features

Data objects, database statistics, physical structures and access paths, user access privileges etc, are active features of DBMS.

## Types of Database Model

*The database model can be of three types as given below*



Attributes/Column

Name	Roll	Class
Ram	001	CS
Shyam	002	IT

Value (c) Relational model

Database models

# **Relational Database Management System (RDBMS)**

A system in which users access data with use of relation (i.e., data must be available in tabular form i.e., as a collection of tables, where each table consisting a set of rows and columns). That provides a variety of relational operator so that we can manipulate the data in tabular form.

## **Features of RDBMS**

*There are many features of RDBMS*

- We can create multiple relations (tables) and feed data into them.
- Provides an interactive query language.
- We can retrieve information from more than one table using join concept.
- Provides a catalog or dictionary which consists of system tables.

### **RDBMS Vocabulary**

**Relation** A table.

**Attribute** A column in a table.

**Degree** Number of attributes in a relation.

**Cardinality** Number of tuples in a table.

**Domain or Type** A pool of values from which specific attributes draw their values.

**NULL** Special value for ‘unknown’ or ‘undefined’.

**Relational Data Model** It provides mechanisms (languages) for defining data structures, operations for retrieval and modification of data and integrity constraints.

## **Keys**

An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a **key** for that entity set.

*Different types of keys are as follows*

### **Super Key**

It is a set of one or more attributes that allow us to identify uniquely an entity in the entity set. e.g., the Customer\_Id attribute of the entity set, customer is sufficient to distinguish one customer entity from another. Thus, Customer\_Id is a super key. Similarly, the combination of Customer\_Name and Customer\_Id is a super key for the entity set customer. The Customer\_Name attribute of customer is not a super key, because several people might have the same name.

## Candidate Key

Since, as we saw a super key may contain extraneous attributes (i.e., Customer\_Name here in the above case). If  $K$  is a super key, then candidate key is any superset of  $K$ . We are often interested in super keys for which no proper subset is a super key. Such minimal super keys are called candidate keys. Suppose that a combination of Customer\_Name and Customer\_Street is sufficient to distinguish among members of the customer entity set. Then, both {Customer\_Id} and {Customer\_Name, Customer\_Street} are candidate keys. Although, the attributes Customer\_Id and Customer\_Name together can distinguish customer entities, their combination does not form a candidate key, since the attribute Customer\_Id alone is a candidate key.

## Primary Key

A candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate and super) is a property of the entity set rather than of the individual entities.

## Alternate Key

A table may have one or more choices for the primary key. Collectively these are known as candidate keys as discuss earlier. One is selected as the primary key. Those not selected are known as secondary keys or alternative keys.

## Secondary Key

An attribute or set of attributes that may not be a candidate key but that classifies the entity set on a particular characteristics. e.g., suppose entity set employee having the attribute department whose value identifies all instances employee who belong to a given department.

## Foreign Key

A foreign key is generally a primary key from one table that appears as a field in another, where the first table has a relationship to the second. In other words, if we had a table TBL1 with a primary key A that linked to a table TBL2, where A was a field in TBL2, then A would be a foreign key in TBL2.

## Simple and Composite Key

Any key consisting of a single attribute is called a simple key, while that consisting of a combination of attributes is called a composite key.

# **E-R Modeling**

Entity–Relationship model (ER model) in software engineering is an abstract way to describe a database. Describing a database usually starts with a relational database, which stores data in tables. Some of the data in these tables point to data in other tables for instance, your entry in the database could point to several entries for each of the phone numbers that are yours. The ER model would say that you are an entity, and each phone number is an entity, and the relationship between you and the phone numbers is ‘has a phone number’.

## **Relationship**

An association among entities. e.g., There is a relationship between employees and department, which can be named as `Works_in`.

### **Relationship Set**

An association of entity sets e.g., `Employee_Department`

- A relationship instance is an association of entity instances e.g., `Shyam_Sales`.
- Same entity set could participate in different relationship sets.
- An  $n$ -array relationship set  $R$  relates  $n$  entity sets.
- A relationship set involving three entity sets, is known as a ternary relationship.

## **Entity**

Anything that exists and can be distinguished/ real world object which can be distinguished from other objects. e.g., student.

### **Entity Set**

A group of similar entities, e.g., all students.

- All entities in an entity set have the same set of attributes.
- Each entity set has a key.
- Can be mapped to a relation easily.

## **Weak Entity Set and Strong Entity Set**

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. An entity set that has a primary key is termed a strong entity set. For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated with an identifying entity i.e., the weak entity set is said to be existence dependent on the identifying entity set.

## Attribute

Properties that describe an entity or we can say attributes are descriptive properties possessed by each member of an entity set and each attribute has a domain.

*There are many types of attributes*

### Composite Attribute

Attributes which can have component attribute. e.g., a composite attribute name, with component attributes First\_Name, Middle\_Name and Last\_Name.

### Derived Attribute

The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the customer entity set has an attribute Loans\_Held, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of loan entities associated with that customer.

### Descriptive Attribute

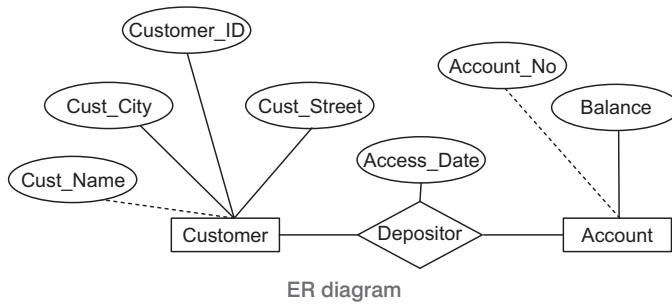
If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. e.g., consider a relationship set depositor with entity sets customer and account. We could associate the attribute Access\_Date to that relationship to specify the most recent date on which a customer accessed an account.

### Single Valued Attribute

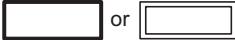
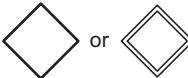
Attribute which has only one value, e.g., the Employee\_Number attribute for a specific Employee\_entity refers to only one employee number.

### Multi Valued Attribute

Attributes which can have 0, 1 or more than 1 values. An employee entity set with the attribute Phone\_Number. An employee may have zero, one or several phone numbers and different employees may have different numbers of phones.



**Notations/Shapes in ER Modeling**

Rectangle represents entity type.	
Double/Bold rectangle represent weak entity type.	
Diamond represents relationship type.	
Double/Bold diamond represents weak relationship type.	
Ellipse represents attribute type.	
Double ellipse represents multivalued attribute.	
Dashed ellipse denotes derived attribute.	
Line a link attribute to entity sets and entity sets to relationship sets	
Double lines which indicate total participation of an entity in a relationship set i.e., each entity in the entity set occurs in atleast one relationship in that relationship set.	

**Mapping Cardinalities/Cardinality Ratio/Types of Relationship**

Expresses the number of entities to which another entity can be associated *via* a relationship set. For a binary relationship set  $R$  between entity sets  $A$  and  $B$ , the mapping cardinality must be one of the following

**One to One**

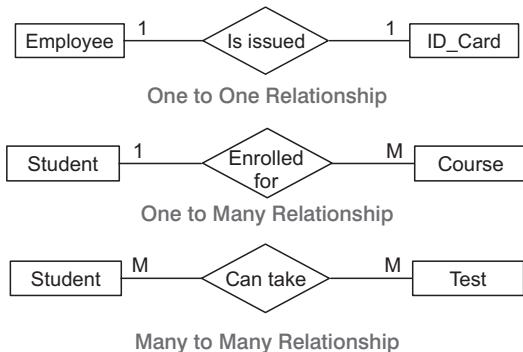
An entity in  $A$  is associated with at most one entity in  $B$  and an entity in  $B$  is associated with at most one entity in  $A$ .

**One to Many**

An entity in  $A$  is associated with any number (zero or more) of entities in  $B$ . An entity in  $B$ , however, can be associated with at most one entity in  $A$ .

## Many to Many

An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.



## Extended E-R Features

*The extended E-R features are given below*

### Specialization

Consider an entity set person with attributes name, street and city. A person may be further classified as one of the following

- (i) Customer
- (ii) Employee

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes. The process of designating subgroupings within an entity set is called specialization.

The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicitly.

### Generalization

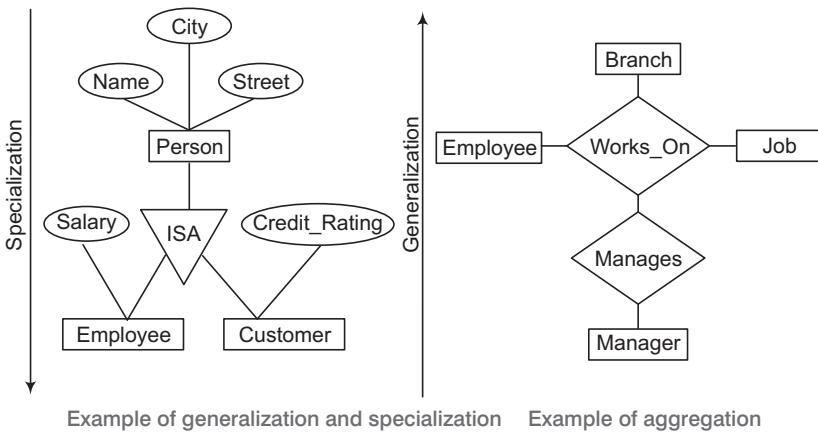
Basically generalization is a simple inversion of specialization. Some common attributes of multiple entity sets are chosen to create higher level entity set. If the customer entity set and the employee entity set are having several attributes in common, then this commonality can be expressed by generalization.

Here, person is the higher level entity set and customer and employee are lower level entity sets. Higher and lower level entity sets also may be designated by the terms super class and subclass, respectively. The person entity set is the super class of the customer and employee subclasses.

### **Aggregation**

Aggregation is used when we have to model a relationship involving entity set and a relationship set. Aggregation is an abstraction through which relationships are treated as higher level entities.

Suppose the relationship set Works\_On (relating the entity sets employee, branch and job) as a higher\_level entity set called Works\_On. Such an entity set is treated in the same manner as is any other entity set. We can create a binary relationship manages between Works\_On and manager to represent who manages what tasks. Aggregation is meant to represent a relationship between a whole object and its component parts.



### **Integrity Constraints**

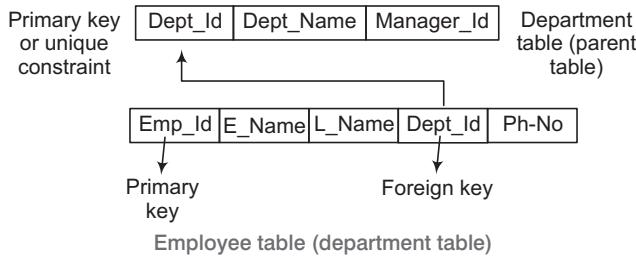
Necessary conditions to be satisfied by the data values in the relational instances so that the set of data values constitute a meaningful database.  
*There are four types of integrity constraints*

**Domain Constraint** The value of attribute must be within the domain.

**Key Constraint** Every relation must have a primary key.

**Entity Integrity Constraint** Primary key of a relation should not contain NULL values.

**Referential Integrity Constraint** In relational model, two relations are related to each other over the basis of attributes. Every value of referencing attributes must be NULL or be available in the referenced attribute.



## Relational Algebra and Relational Calculus

Relational model is completely based on relational algebra. It consists of a collection of operators that operate on relations.

Its main objective is data retrieval. It is more operational and very much useful to represent execution plans, while relational calculus is non-operational and declarative. Here, declarative means user define queries in terms of what they want, not in terms of how compute it.

### Relational Query Languages

Used for data manipulation and data retrieval. Relational model support simple yet powerful query languages. To understand SQL, we need good understanding of two relational query language (*i.e.*, relational algebra and relational calculus).

## Basic Operation in Relational Algebra

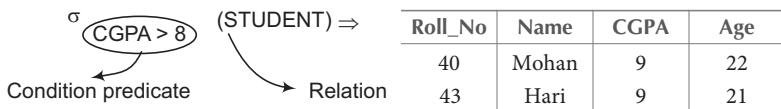
*The operations in relational algebra are classified as follows*

### Selection

The select operation selects tuples/rows that satisfy a given predicate or condition. We use  $(\sigma)$  to denote selection. The predicate/condition appears as a subscript to  $\sigma$ .

e.g., Consider the below relation STUDENT

Roll_No	Name	CGPA	Age
30	Ram	7	20
35	Shyam	8	21
40	Mohan	9	22
43	Hari	9	21



### Projection

It selects only required/specify columns/attributes from a given relation/table. Projection operator eliminates duplicates (i.e., duplicate rows from the result relation) e.g., consider the STUDENT relation.

Age
20
21
22

$\Pi_{Age}(\text{STUDENT}) \Rightarrow$

Name	CGPA
Ram	7
Shyam	8
Mohan	9
Hari	9

$\Pi_{Name, CGPA}(\text{STUDENT}) \Rightarrow$

Combining selection and projection

$$\Pi_{Name, CGPA}(\sigma_{CGPA > 8}(\text{STUDENT}))$$

↓

Name	CGPA
Mohan	9
Hari	9

## Union

It forms a relation from rows/tuples which are appearing in either or both of the specified relations. For a union operation  $R \cup S$  to be valid, *below two conditions must be satisfied.*

- The relations  $R$  and  $S$  must be of the same entity. i.e., they must have the same number of attributes.
- The domains of the  $i$  th attribute of  $R$  and  $i$  th attribute of  $S$  must be the same, for all  $i$ .

## Intersection

It forms a relation of rows/ tuples which are present in both the relations  $R$  and  $S$ . As with the union operation, we must ensure that both relations are compatible.

	Name	Age		Name	Age
	Ram	20		Ram	20
	Mohan	21		Shyam	22
<i>R</i>					
$R \cup S \Rightarrow$ (Union operation)	Name	Age	$R \cap S \Rightarrow$ (Intersection operation)	Name	Age
	Ram	20		Ram	20
	Shyam	22			
	Mohan	21			

## Set Difference

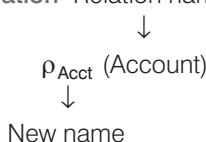
It allows us to find tuples that are in one relation but are not in another. The expression  $R - S$  produces a relation containing those tuples in  $R$  but not in  $S$ .

$$R - S \Rightarrow \begin{array}{|c|c|} \hline \text{Name} & \text{Age} \\ \hline \text{Mohan} & 21 \\ \hline \end{array} (\Pi_{\text{Name}, \text{Age}}(R) - \Pi_{\text{Name}, \text{Age}}(S))$$

## Cross Product/Cartesian Product

Assume that we have  $n_1$  tuples in  $R$  and  $n_2$  tuples in  $S$ . Then, there are  $n_1 * n_2$  ways of choosing a pair of tuples; one tuple from each relation. So, there will be  $(n_1 * n_2)$  tuples in result relation  $P$  if  $P = R \times S$ .

Rename operation Relation name



$p_x(E_1)$ , where  $x$  is the new name for the result of  $E_1$  and  $E_1$  may be an expression.

**Schema Refinement/Normalization** Decomposition of complex records into simple records. Normalization reduces redundancy using non-loss decomposition principle.

**Shortcomings of Redundancy** Data redundancy causes three types of anomalies insert, update and delete. Storage problem (*i.e.*, wastage of storage).

**Decomposition** Splitting a relation  $R$  into two or more subrelation  $R_1$  and  $R_2$ . A fully normalized relation must have a primary key and a set of attributes.

### **Database Design Goal (Schema Refinement)**

There are many goals for the design of a database. *Here are some of them listed*

**The Database is Compressive** It includes all the needed data and connections.

**The Database is Understandable** There is a clear structure which leads to easy, flexible and fast reading and updating of the data.

**The Database is Expandable** It is possible to change the structure of the database with a minimum change to the existing software.

- 0% redundancy

*Decomposition should satisfy*

- |                   |                             |
|-------------------|-----------------------------|
| (i) Lossless join | (ii) Dependency preservence |
|-------------------|-----------------------------|

### **Lossless Join Decomposition**

Join between the subrelations should not create any additional tuples or there should not be a case such that more number of tuples in  $R_1$  than  $R_2$

$$R \subset R_1 \bowtie R_2 \Rightarrow (\text{Lossy})$$

$$R \equiv R_1 \bowtie R_2 \Rightarrow (\text{Lossless})$$

### **Dependency Preservence**

Because of decomposition, there must not be loss of any single dependency.

## Functional Dependency (FD)

Dependency between the attribute is known as functional dependency. Let  $R$  be the relational schema and  $X, Y$  be the non-empty sets of attributes and  $t_1, t_2, \dots, t_n$  are the tuples of relation  $R$ .

$X \rightarrow Y$  {values for  $X$  functionally determine values for  $Y$ }

If the above condition holds, it means if

$$t_1 \cdot X = t_2 \cdot X, \text{ then } t_1 \cdot Y = t_2 \cdot Y$$

tup Les	X	Y
$t_1$	a	b
$t_2$	a	b

Relation  $R_1$ , which holds  $X \rightarrow Y$

tup Les	X	Y
$t_3$	a	b
$t_4$	a	c

Relation  $R_2$ , which does not hold  $X \rightarrow Y$

### Trivial Functional Dependency

If  $X \supseteq Y$ , then  $X \rightarrow Y$  will be trivial FD.

Here,  $X$  and  $Y$  are set of attributes of a relation  $R$ .

In trivial FD, there must be a common attribute at both the sides of ' $\rightarrow$ ' arrow.

$$\left. \begin{array}{l} S_{id} S_{name} \rightarrow S_{name} \\ S_{id} S_{name} \rightarrow S_{id} \end{array} \right\} \text{trivial FD}$$

### Non-Trivial Functional Dependency

If  $X \cap Y = \emptyset$  (no common attributes) and  $X \rightarrow Y$  satisfies FD, then it will be a non-trivial FD.

$$\left. \begin{array}{l} C_{id} \rightarrow C_{name} \\ S_{id} \rightarrow S_{name} \end{array} \right\} \text{non-trivial FD}$$

(no common attribute at either side of ' $\rightarrow$ ' arrow)

Case of semi-trivial FD

$$S_{id} \rightarrow S_{id} S_{name} \text{ (semi-trivial)}$$

Because on decomposition, we will get

$$S_{id} \rightarrow S_{id} \text{ (trivial FD) and}$$

$$S_{id} \rightarrow S_{name} \text{ (non-trivial FD)}$$

## **Properties of Functional Dependence (FD)**

- **Reflexivity** If  $X \supseteq Y$ , then  $X \rightarrow Y$  (trivial)
- **Transitivity** If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$
- **Augmentation** If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$
- **Splitting or Decomposition** If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$
- **Union** If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$

### **Attribute Closure**

Suppose  $R(X, Y, Z)$  be a relation having set of attributes i.e.,  $(X, Y, Z)$ , then  $(X^+)$  will be an attribute closure which functionally determines other attributes of the relation (if not all then atleast itself).

### **DBMS versus RDBMS**

DBMS	RDBMS
• In DBMS, relationship between two tables or files are maintained programmatically, morganatically	In RDBMS, relationship between two tables or files can be specified at the time of table creation
• DBMS does not support client/server architecture	Most of the RDBMS supports client/server architecture
• DBMS does not support distributed database	Most of the RDBMS support distributed databases
• In DBMS, there is no security of data	In RDBMS, <i>there are multiple levels of security</i>
	(i) Cogging in at o/s Level
	(ii) Comand Level
	(iii) Object level
• Each table is given an extension in DBMS	Many tables are grouped in one database in RDBMS

# Normal Forms/Normalization

In relational database design, the normalization is the process for organizing data to minimize redundancy. Normalization usually involves dividing a database into two or more tables and defining relationship between the tables.

The normal forms define the status of the relation about the individuated attributes. *There are five types of normal forms*

## First Normal Form (1NF)

Relation should not contain any multivalued attributes or relation should contain atomic attributes.

Anomalies in Database

Stud_Id	Stud_Name	Course_Name
S <sub>1</sub>	A	C/C++
S <sub>2</sub>	B	C++
S <sub>3</sub>	B	C++/Java

(STUDENT) Relation

The above relation is not in 1 NF



Stud_Id	Stud_Name	Course_Name
S <sub>1</sub>	A	C
S <sub>1</sub>	A	C++
S <sub>2</sub>	B	C++
S <sub>3</sub>	B	C++
S <sub>3</sub>	B	Java

The above relation is in 1NF but now Stud\_Id is no more a primary key.

Now in the above case, we need to modify our primary key which is (Stud\_Id, Course-Name)

**Note** The main disadvantage of 1NF is high redundancy.

## **Second Normal Form (2NF)**

Relation  $R$  is in 2NF if and only if

- $R$  should be in 1NF.
- $R$  should not contain any partial dependency.

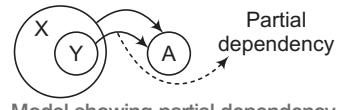
### **Partial Dependency**

Let  $R$  be the relational schema having  $X, Y, A$ , which are non-empty set of attributes, where

$X$  = Any candidate key of the relation.

$Y$  = Proper subset of any candidate key

$A$  = Non-prime attribute (*i.e.*,  $A$  doesn't belong to any candidate key)



Model showing partial dependency

In the above example,  $X \rightarrow A$  already exists and if  $Y \rightarrow A$  will exist, then it will become a partial dependency, if and only if

- $Y$  is a proper subset of candidate key.
- $A$  should be non-prime attribute.

If any of the above two conditions fail, then  $Y \rightarrow A$  will also become fully functional dependency.

### **Full Functional Dependency**

A functional dependency  $P \rightarrow Q$  is said to be fully functional dependency, if removal of any attribute  $S$  from  $P$  means that the dependency doesn't hold any more.

(*Student\_Name, College\_Name*  $\rightarrow$  *College\_Address*)

Suppose, the above functional dependency is a full functional dependency, then we must ensure that there are no FDs as below.

(*Student\_Name*  $\rightarrow$  *College\_Address*)

or      (*College\_Name*  $\rightarrow$  *Collage\_Address*)

## **Third Normal Form (3NF)**

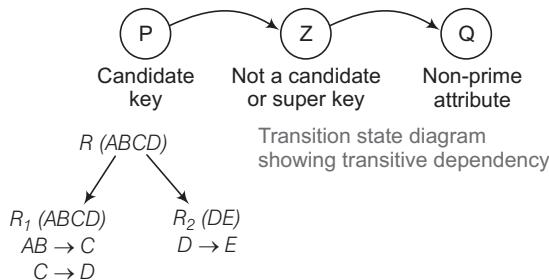
Let  $R$  be a relational schema, then any non-trivial FD  $X \rightarrow Y$  over  $R$  is in 3NF, if

- $X$  should be a candidate key or super key.  
or
- $Y$  should be a prime attribute.
- Either both of the above conditions should be true or one of them should be true.
- $R$  should not contain any transitive dependency.
- For a relation schema  $R$  to be a 3NF, it is necessary to be in 2NF.

## Transitive Dependency

A FD,  $P \rightarrow Q$  in a relation schema  $R$  is a transitive if

- There is a set of attributes  $Z$  that is not a subset of any key of  $R$ .
- Both  $X \rightarrow Z$  and  $Z \rightarrow Y$  hold



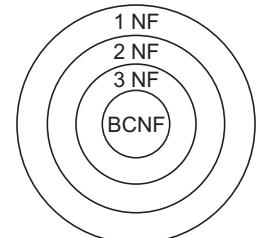
- The above relation is in 2NF.
- In relation  $R_1$ ,  $C$  is not a candidate key and  $D$  is non-prime attribute. Due to this,  $R_1$  fails to satisfy 3NF condition. Transitive dependency is present here.

$AB \rightarrow C$  and  $C \rightarrow D$ , then  $AB \rightarrow D$  will be transitive.

### Boyce Codd Normal Form (BCNF)

Boyce Codd Normal Form (BCNF) Let  $R$  be the relation schema and  $X \rightarrow Y$  be the any non-trivial FD over  $R$  is in BCNF if and only if  $X$  is the candidate key or super key.

$X \rightarrow Y$       } If  $R$  satisfies this dependency, then of course it satisfies 2NF and 3NF.  
candidate / super key }



Hierarchy of normal forms

**Summary of 1 NF, 2 NF and 3 NF**

Normal Form	Test	Remedy (Normalization)
1 NF	Relation should have no non-atomic attributes or nested relations.	Form name relation for each non-atomic attribute or nested relation.
2 NF	For relations where primary key contains multiple attributes, no non-key attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attributes. Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
3 NF	Relation should not have a non-key attribute functionally determined by another non-key attribute (or by a set of non-key attributes), <i>i.e.</i> , there should be no transitive dependency of a non-key attribute on the primary key.	Decompose and setup a relation that includes the non-key attribute(s) that functionally determine(s) other non-key attribute(s)

**Fourth Normal Form (4NF)**

4NF is mainly concerned with multivalued dependency. A relation is in 4NF if and only if for every one of its non-trivial multivalued dependencies  $X \rightarrow\!\!\rightarrow Y$ , X is a super key (*i.e.*, X is either a candidate key or a superset).

**Fifth Normal Form (5NF)**

It is also known as Project Join Normal Form (PJ/NF). 5NF reduces redundancy in relational database recording multivalued facts by isolating semantically related multiple relationships.

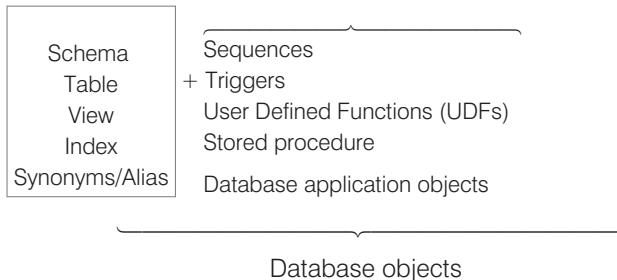
A table or relation is said to be in the 5NF, if and only if every join dependency in it, is implied by the candidate keys.

**Key Points**

- ♦ The Normal Forms (NF) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies
- ♦ Databases intended for Online Transaction Processing (OLTP) are typically more normalized than databases intended for Online Analytical Processing (OLAP)
- ♦ OLTP applications are characterized by a high volume of small transactions such as updating a sales record at a supermarket checkout counter.
- ♦ The expectation is that each transaction will leave the database in a consistent state

# SQL

Structured Query Language (SQL) is a language that provides an interface to relation database systems. SQL was developed by IBM in the 1970, for use in system R and is a defacto standard, as well as an ISO and ANSI standard.



- To deal with the above database objects, we need a programming language and that programming language is known as SQL.

*Three subordinate languages of SQL are*

## **Data Definition Language (DDL)**

*It includes the commands as*

- **CREATE** To create tables in the database.
- **ALTER** To modify the existing table structure.
- **DROP** To drop the table with table structure.
- **Data Manipulation Language (DML)** It is used to insert, delete, update data and perform queries on these tables. *Some of the DML commands are given below.*
- **INSERT** To insert data into the table.
- **SELECT** To retrieve data from the table.
- **UPDATE** To update existing data in the table.
- **DELETE** To delete data from the table.

## **Data Control Language (DCL)**

*It is used to control user's access to the database objects. Some of the DCL commands are*

- **GRANT** Used to grant select/insert/delete access.
- **REVOKE** Used to revoke the provided access.

**Transaction Control Language (TCL)**

It is used to manage changes affecting the data.

- **COMMIT** To save the work done, such as inserting or updating or deleting data to/from the table.
- **ROLLBACK** To restore database to the original state, since last commit.
- **SQL Data Types** SQL data types specify the type, size and format of data/information that can be stored in columns and variables.

**Key Points**

- SQL is a special-purpose programming language designed for managing data held in a Relational Database Management Systems (RDBMS).
- SQL based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language.

**Various Data Types in SQL**

- Data Time time-stamp
- Char Big int Integer
- Decimal Small int Double

*There are so many other data types also.*

**Database Constraints**

These are user defined that let us restrict the behaviours of column. We can create constraints when we define a table with a SQL CREATE statement.

**Inline Constraint**

A constraint defined on the same line as its column.

**Out of Line Constraint**

A constraint defined on it's own line in a CREATE statement. This type of constraint must reference the column that they constrain.

### Constraint Types with Description

Name of constraint	Table level/column level/Row level/External level	Description
NOT NULL	Column level	Restricts a column by making it mandatory to have some value.
Unique	Table level	Checks whether a column value will be unique among all rows in a table.
Primary key	Column level and Table level	Checks whether a column value will be unique among all rows in a table and disallows NULL values.
Check constraint	Column level Row level External level	Restrict a column value to a set of values defined by the constraint.
Foreign key constraint	Column level External level	Restrict the values that are acceptable in a column or group of columns of a table to those values found in a listing of the column/group of columns used to define the primary key in other table.

### Key Points

- ♦ The most common operation in SQL is the query, which is performed with the declarative SELECT statement.
- ♦ SELECT retrieves data from one or more tables, or expressions.
- ♦ Standard SELECT statements have no persistent effects on the database.
- ♦ Queries allow the user to describe desired data, leaving the Database Management System (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

### Default Constraint

It is used to insert a default value into a column, if no other value is specified at the time of insertion.

#### Syntax

```
CREATE TABLE Employee
{
    Emp_id int NOT NULL,
    Last_Name varchar (250),
    City varchar (50)DEFAULT 'BANGALURU'
}
```

### DDL Commands

1. CREATE TABLE < Table\_Name >
 

```

      {
          Column_name 1 < data_type >,
          Column_name 2 < data_type >
      }
```

2. ALTER TABLE < Table\_Name >  
ALTER Column < Column\_Name > SET NOT NULL
3. RENAME < object\_type >object\_name >to <new\_name >
4. DROP TABLE <Table\_Name>

### **DML Commands**

SELECT             $A_1, A_2, A_3, \dots, A_n$  what to return  
 FROM             $R_1, R_2, R_3, \dots, R_m$  relations or table

WHERE condition filter condition i.e., on what basis, we want to restrict the outcome/result.

If we want to write the above SQL script in the form of relational calculus, we use the following syntax

$$\{\Pi_{A_1 \dots A_n} (\sigma_{\text{condition}} (R_1 \times R_2 \times \dots \times R_m))\}$$

Comparison operators which we can use in filter condition are  
 $(=, >, <, > =, < =, <>)$  ' $<>$ ' means not equal to.

### **INSERT Statement**

Used to add row (s) to the tables in a database.

INSERT INTO Employee (F\_Name, L\_Name)VALUES ('Atal', 'Bihari')

### **Key Points**

- ♦ Data values can be added either in every column in a row or in same columns in a row by specifying the columns and their data.
- ♦ All the columns that are not listed in the column list in INSERT statement, will receive NULL.

### **UPDATE Statement**

It is used to modify/update or change existing data in single row, group of rows or all the rows in a table.

e.g.,

UPDATE Employee

SET City = 'LUCKNOW'

WHERE Emp\_Id BETWEEN

9 AND 15;

An example of selective update which will update some rows in a table.

UPDATE Employee SET City  
 = 'LUCKNOW' ;

Example of global update which will update city column for all the rows.

## **DELETE Statement**

This is used to delete rows from a table.

e.g.,

```
DELETE Employee WHERE Example of selective delete
Emp_Id = 7;
```

```
DELETE Employee
```

This command will delete all the rows from Employee table.

## **ORDER BY Clause**

This clause is used to sort the result of a query in a specific order (ascending or descending).

By default sorting order is ascending.

```
SELECT Emp_Id, Emp_Name, City FROM Employee
WHERE City = 'LUCKNOW'
ORDER BY Emp_Id DESC;
```

## **GROUP BY Clause**

It is used to divide the result set into groups. Grouping can be done by a column name or by the results of computed columns when using numeric data types.

- The HAVING clause can be used to set conditions for the GROUP BY clause.
- HAVING clause is similar to the WHERE clause, but having puts conditions on groups.
- WHERE clause places conditions on rows.
- WHERE clause can't include aggregate function, while HAVING conditions can do so.

e.g.,

```
SELECT Emp_Id, AVG (Salary)
FROM Employee
GROUP BY Emp_Id
HAVING AVG (Salary) > 25000;
```

## **Aggregate Functions**

Function	Description
SUM ()	It returns total sum of the values in a column.
AVG ()	It returns average of the values in a column.
COUNT ()	Provides number of non-null values in a column.
MIN () and MAX ()	Provides lowest and highest value respectively in a column.
COUNT (*)	Counts total number of rows in a table.

# Joins

Joins are needed to retrieve data from two tables' related rows on the basis of some condition which satisfies both the tables. Mandatory condition to join is that atleast one set of column (s) should be taking values from same domain in each table.

## Types of Joins

The Two types of joins are given below

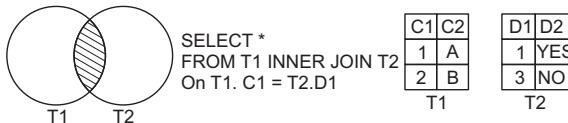
### Inner Join

Inner join is the most common join operation used in applications and can be regarded as the default join-type. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. These may be further divided into three parts.

- (i) Equi Join (satisfies equality condition)
- (ii) Non-Equi Join (satisfies non-equality condition)
- (iii) Self Join (one or more column assumes the same domain of values)

Considers only pairs that satisfy the joining condition

Result is the intersection of the two tables.



Inner join in set  $T_1$  and set  $T_2$

Result set of  $T_1$  and  $T_2$

$T_1 \cdot C_1$	$C_2$	$T_2 \cdot D_1$	$D_2$
1	A	1	YES

## Key Points

- ♦ The cross join does not apply any predicate to filter records from the joined table. Programmers can further filter the results of a cross join by using a WHERE clause.
- ♦ An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate
- ♦ A special case, a table (base table, view, or joined table) can JOIN to itself in a self-join.

## Outer Join

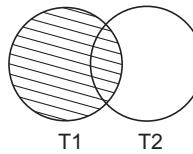
An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists.

Considers also the rows from table (s) even if they don't satisfy the joining condition

- (i) Right outer join
- (ii) Left outer join
- (iii) Full outer join

### Left Outer Join

The result of a left outer join for table A and B always contains all records of the left table (A), even if the join condition does not find any matching record in the right table (B).



```
SELECT * FROM T1
Left Outer Join T2 ON T1.
ID = T2 . ID
```

ID	Name
1	Ram
2	Shyam
T1	

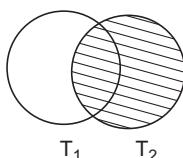
ID	Branch
1	IT
3	CS
Left Outer Join	

Result set of T1 and T2

T1.ID	Name	T2.ID	Branch
1	Ram	1	IT
2	Shyam	NULL	NULL

### Right Outer Join

A right outer closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the right table will appear in the joined table at least once. If no matching with left table exists, NULL will appear.



```
SELECT * FROM T1 RIGHT
OUTER JOIN T2 ON T1 . ID
= T2 . ID
```

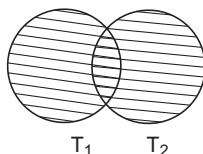
ID	Name	ID	Branch
1	Ram	1	IT
2	Shyam	NULL	NULL
T1		T2	
Right outer Join			

Result set of T1 and T2

T1.ID	Name	T2.ID	Branch
1	Ram	1	IT
NULL	NULL	3	CS

### **Full Outer Join**

A full outer join combines the effect of applying both left and right outer joins. where records in the FULL OUTER JOIN table do not match, the result set will have NULL values for every column of the table that lacks a matching row. for those records that do match, as single row will be produced in the result set.



```
SELECT * FROM T1 FULL OUTER
JOIN T2 ON T1 . ID=T2.ID
```

**Result set of T1 and T2 (Using tables of previous example)**

T1 · ID	Name	T2 · ID	Branch
1	Ram	1	IT
2	Shyam	NULL	NULL
NULL	NULL	3	CS

### **Cross Join (Cartesian Product)**

Cross join returns the cartesian product of rows form tables in the join. It will produce rows which combine each row from the first table with each row from the second table.

Select \* FROM T1, T2

Number of rows in result set = (Number of rows in table 1 × Number of rows in table 2)

**Result set of T1 and T2 (Using previous tables T1 and T2)**

ID	Name	ID	Branch
1	Ram	1	IT
2	Shyam	1	IT
1	Ram	3	CS
2	Shyam	3	CS

### **Key Points**

- ♦ A programmer writes a JOIN predicate to identify the records for joining. If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table.
- ♦ A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).
- ♦ A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table.

# **Transaction Management**

A sequence of many actions which are considered to be one atomic unit of work. A transaction is a collection of operations involving data items in a database. There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures.

## **Atomicity**

Atomicity requires that each transaction is all or nothing. If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.

## **Consistency**

If each transaction is consistent and the data base starts on as consistent, it ends up as consistent.

## **Isolation**

Execution of one transaction is isolated from that of another transactions. It ensures that concurrent execution of transaction results in a system state that would be obtained if transaction were executed serially, i.e., one after the other.

## **Durability**

Durability means that once a transaction has been committed, it will remain even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently.

## **Key Points**

---

- ♦ In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably.
  - ♦ In the context of databases, a single logical operation on the data is called a transaction.
  - ♦ A transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.
- 

If a transaction commits, its effects persist.

- A transaction starts with any SQL statement and ends with a COMMIT or ROLLBACK.

- COMMIT statement makes changes permanent to the database.
- ROLLBACK statement reverses changes.
- A transaction includes one or more database access operations. These can include insertion, deletion, modification or retrieval operations.

### **Database Access Operations**

*The basic database access operations are*

- **Read-item (X)** Reads a database item named X into a program variable or  $R(X)$ .
- **Write-item (X)** Writes the value of program variable X into the database item named X or  $W(X)$ .

## **Classification of a Database System According to the Number of Users**

### **Single User**

A DBMS is single user, if at most one user at a time can use the system.

### **Multiuser**

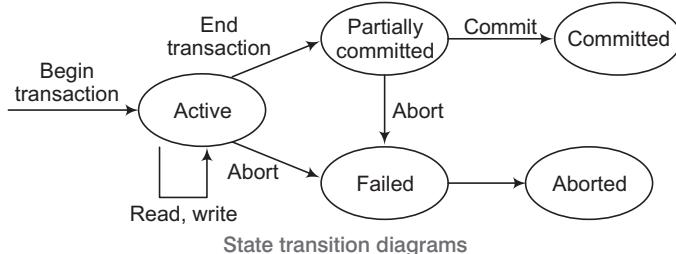
A DBMS is multiuser, if many users can use the system and hence access the database concurrently.

e.g., An airline reservation system is used by hundreds of travel agents and reservation clerks submit transactions concurrently to the system.

### **Transaction States**

*The following are the different states in transaction processing in a database system*

1. Active
2. Partially committed
3. Failed
4. Aborted



State transition diagrams

**Active** This is the initial state. The transaction stays in this state while it is executing.

**Partially Committed** This is the state after statement of the transaction is executed.

**Failed** After the discovery that normal execution can no longer proceed.

**Aborted** The state after the transaction has been rolled back and the database has been resorted to its state prior to the start of the transaction.

## Isolation Levels

If every transaction does not make its updates visible to other transaction until it is committed, so isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks.

*There have been attempts to define the level of isolation of a transaction.*

**Level-0 Isolation** A transaction is said to have level-0 isolation, if it does not overwrite the dirty reads of higher-level transactions.

**Level-1 Isolation** Level-1 isolation has no lost updates.

**Level-2 Isolation** Level-2 isolation has no lost updates and no dirty reads.

**Level-3 Isolation** Level-3 isolation (also true isolation) has repeatable reads.

# Concurrency Control

Process of managing simultaneous execution of transactions in a shared database, is known as **concurrency control**. Basically, concurrency control ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

## Need of Concurrency Control

Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.

### Lost Update

This problem occurs when two transactions that access the same database items, have their operations interleaved in a way that makes the value of some database items incorrect.

### Dirty Read Problems

This problem occurs when one transaction reads changes the value while the other reads the value before committing or rolling back by the first transaction.

## Inconsistent Retrievals

This problem occurs when a transaction accesses data before and after another transaction(s) finish working with such data.

*We need concurrence control, when*

- The amount of data is sufficiently great that at any time only fraction of the data can be in primary memory and rest should be swapped from secondary memory as needed.
- Even if the entire database can be present in primary memory, there may be multiple processes.

## Key Points

- ♦ A failure in concurrency control can result in data corruption from torn read or write operations.
- ♦ DBMS need to deal also with concurrency control issues not typical just to database transactions but rather to operating systems in general.
- ♦ Concurrency control is an essential element for correctness in any system where two database transactions or more, executed with time overlap, can access the same data, e.g., virtually in any general purpose database system.

## Schedule

A schedule (or history) is a model to describe execution of transactions running in the system. When multiple transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from the various transactions is known as a schedule or we can say that a schedule is a sequence of read, write, abort and commit operations from a set of transactions.

*The following is an example of a schedule*

$T_1$	$T_2$	$T_3$
$R(X)$		
$W(X)$		
Commit		
	$R(Y)$	
	$W(Y)$	
	Commit	
		$R(Z)$
		$W(Z)$
		Commit

The above schedule consists of three transactions  $T_1, T_2$  and  $T_3$ . The first transaction  $T_1$  reads and writes to object  $X$  and then commits. Then,  $T_2$  reads and writes to object  $Y$  and commits and finally  $T_3$  reads and writes to object  $Z$  and commits.

## Classification of Schedules Based on Serializability

*The schedules can be classified as*

### Serial Schedule

A schedule in which the different transactions are not interleaved (i.e., transactions are executed from start to finish one-by-one).

Serial Schedule		Serial Schedule	
$T_1$	$T_2$	$T_1$	$T_2$
	$R(A)$ $W(A)$	$R(B)$ $W(B)$	
$R(B)$ $W(B)$			$R(A)$ $W(A)$

### Complete Schedule

A schedule that contains either a commit or an abort action for each transaction.

**Note** Consequently, a complete schedule will not contain any active transaction at the end of the schedule.

Complete Schedule		Complete Schedule		Complete Schedule	
$T_1$	$T_2$	$T_1$	$T_2$	$T_1$	$T_2$
$R(A)$		$R(A)$		$R(A)$	
	$R(B)$	$W(A)$		$W(A)$	
$W(A)$			$R(B)$	Commit	
	$W(B)$	Commit			$R(B)$
Commit			$W(B)$		$W(B)$
	Abort		Abort		Abort

## **Non-serial Schedules**

Non-serial schedules are interleaved schedules and these schedules improve performance of system (i.e., throughput and response time). But concurrency or interleaving operations in schedules, might lead the database to an inconsistent state.

We need to seek to identify schedules that are

- (i) As fast as interleaved schedules.
- (ii) As consistent as serial schedules.

**Conflicting Operations** When two or more transactions in a non-serial schedule execute concurrently, then there may be some conflicting operations.

Two operations are said to be conflicting, if they satisfy all of the following conditions

- The operations belong to different transactions.
- Atleast one of the operations is a write operation.
- The operations access the same object or item.

The following set of operations is conflicting

$T_1$	$T_2$	$T_3$
$R(X)$	$W(X)$	$W(X)$

While the following sets of operations are not conflicting

$T_1$	$T_2$	$T_3$
$R(X)$	$R(X)$	$R(X)$

//No write on same object

$T_1$	$T_2$	$T_3$
$R(X)$	$W(Y)$	$R(X)$

//No write on same object

## **Key Points**

- ♦ If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, hence the schedules are not conflict equivalent.
- ♦ In view equivalence respective transactions in the two schedules read and write the same data values while in conflict equivalence, respective transactions in two schedules have the same order of conflicting operations.

## Serializable Schedule

A schedule  $S$  of  $n$  transactions is serializable, if it is equivalent to some serial schedule of the same  $n$  transactions.

A non-serial schedule  $S$  is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule.

*There are two types of serializable schedule*

- (i) Conflict serializable schedule
- (ii) View serializable schedule

### Conflict Serializable Schedule

When the schedule ( $S$ ) is conflict equivalent to some serial schedule ( $S'$ ), then that schedule is called as conflict serializable schedule. In such a case, we can reorder the non-conflicting operations in  $S$  until we form the equivalent serial schedule  $S'$ .

Serial schedule	Serializable schedule A	Serializable schedule B
$T_1$	$T_1$	$T_1$
$T_2$	$T_2$	$T_2$
$R(A)$	$R(A)$	$R(A)$
$W(A)$	$W(A)$	$W(A)$
$R(B)$	$R(B)$	$R(B)$
$W(B)$	$W(B)$	$W(B)$
$R(A)$		
$W(A)$		
$R(B)$		
$W(B)$		

### Conflict Equivalence

The schedules  $S_1$  and  $S_2$  are said to be conflict equivalent, if the following conditions are satisfied

- Both schedules  $S_1$  and  $S_2$  involve the same set of transactions (including ordering of operations within each transaction).
- The order of each pair of conflicting actions in  $S_1$  and  $S_2$  are the same.

### Testing for Conflict Serializability of a Schedule

There is a simple algorithm that can be used to test a schedule for conflict serializability. This algorithm constructs a precedence graph (or serialization graph), which is a directed graph.

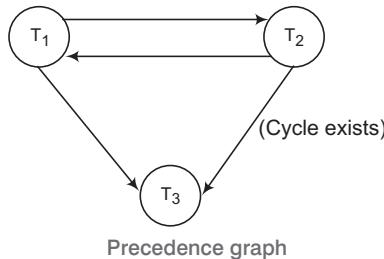
*A precedence graph for a schedule  $S$  contains*

- (i) A node for each committed transaction in  $S$ .
- (ii) An edge from  $T_i$  to  $T_j$ , if an action of  $T_i$  precedes and conflicts with one of  $T_j$ 's operations.

$T_1$	$T_2$	$T_3$
$R(A)$		
$W(A)$	$W(A)$	$W(A)$

Conflict serializability

A schedule  $S$  is conflict serializable if and only if its precedence graphs is acyclic.



**Note** The above example of schedule is not conflict serializable schedule. However, in general, several serial schedules can be equivalent to any serializable schedule  $S$ , if the precedence graph for  $S$  has no cycle and if the precedence graph has a cycle, it is easy to show that we cannot create any equivalent serial schedule, so  $S$  is not serializable.

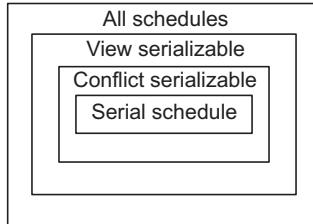
## Key Points

- ♦ Serial schedule is slower but guarantees consistency (correctness).
- ♦ Every serial schedule is a serializable schedule but not every serializable schedule is serial schedule.
- ♦ Being serializable implies that  
*The schedule is a correct schedule, if*  
It will leave the database in a consistent state and the interleaving is appropriate and will result in a state as if the transaction were serially executed.
- ♦ A schedule is an efficient (interleaved) schedule.

## View Serializable Schedule

A schedule is view serializable, if it is view equivalent to some serial schedule.

Conflict serializable  $\Rightarrow$  View serializable, but not vice-versa.



## View Equivalence

Two schedules  $S_1$  and  $S_2$  are view equivalent,

- (i)  $T_i$  reads initial value of database object  $A$  in schedule  $S_1$ , then  $T_i$  also reads initial value of database object  $A$  in schedule  $S_2$ .
- (ii)  $T_i$  reads value of  $A$  written by  $T_j$  in schedule  $S_1$ , then  $T_i$  also reads value of  $A$  written by  $T_j$  in schedule  $S_2$ .
- (iii)  $T_i$  writes final value of  $A$  in schedule  $S_1$ , then  $T_i$  also writes final value of  $A$  in  $S_2$ .

<b>Schedule <math>S_1</math></b>			<b>Schedule <math>S_2</math></b>		
$T_1$	$T_2$	$T_3$	$T_1$	$T_2$	$T_3$
$R(A)$			$R(A)$		
	$W(A)$			$W(A)$	$W(A)$

In the above example, both schedules  $S_1$  and  $S_2$  are view equivalent. So, they are view serializable schedules. But  $S_1$  schedule is not conflict serializable schedule and  $S_2$  is not conflict serializable schedule because cycle is not formed.

## Classification of Schedules Based on Recoverability

- In recoverability, there is need to address the effect of transaction failures on concurrently running transactions.
- Once a transaction  $T$  is committed, it should never be necessary to rollback  $T$ .
- The schedules those meet this criterion are called recoverable schedules and those do not, are called **non-recoverable**.

### Recoverable Schedule

Once a transaction  $T$  is committed, it should never be necessary to rollback  $T$ . The schedules under this criteria are called recoverable schedules and those do not, are called non-recoverable.

A schedule  $S$  is recoverable, if no transaction  $T$  in  $S$  commits until all transactions  $T'$ , that have written an item that  $T$  reads, have committed.

**Initial Recoverable Schedule**

$T_1$	$T_2$
$R(X)$ $W(X)$	<i>Dirty read</i>
Abort	$R(X)$ $W(X)$ Commit

(This schedule is not recoverable because  $T_2$  made a dirty read and committed before  $T_1$ ,  $T_1$  should have committed first.)

The above given schedule is non-recoverable because when the recovery manager rolls back  $T_1$  transaction, then  $X$  gets its initial value before updation. But  $T_2$  has already utilised the wrong value of  $X$  that was updated by  $T_1$  and  $T_2$  committed. Now, database is consequently in an inconsistent state.

### Composition of Recoverable

**Initial Non-recoverable Schedule**

$T_1$	$T_2$
$R(X)$ $W(X)$	<i>Dirty read</i>
Abort	$R(X)$ $W(X)$ Commit

**Recoverable Schedule (A)**

$T_1$	$T_1$
$R(X)$	
$W(X)$	
Abort	$R(X)$ $W(X)$ Commit

(Commit after parent of dirty read)  
(Recoverable)

**Recoverable Schedule (B)**

$T_1$	$T_2$
$R(X)$	
$W(X)$	$R(X)$
Abort	$W(X)$ Commit

Remove dirty read (Recoverable)  
not serializable schedule (Order of conflicting actions has changed)

**Cascadeless Schedule**

These schedules avoid cascading rollbacks. Even, if a schedule is recoverable to recover correctly from failure of transaction  $T_i$ , we may have to rollback several transactions. This phenomenon in which a single transaction failure which leads to a series of transaction rollbacks, is called **cascading rollbacks**.

Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those, where cascading rollback cannot occur. Such schedules are called **cascadeless schedules**.

**Cascadeless Schedule**

$T_1$	$T_2$
$R(X)$	
$W(X)$	
$R(Y)$	
$W(Y)$	
(a) Rollback Abort	(b) Now $T_2$ reads the value of $X$ that existed before $T_1$ started. $R(X)$ $W(X)$ Commit

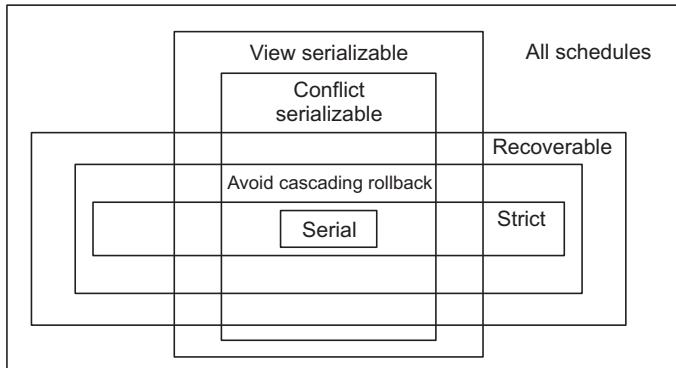
**Strict Schedule**

A *schedule is strict*,

- If overriding of uncommitted data is not allowed.
- Formally, if it satisfies the following conditions

- (i)  $T_j$  reads a data item  $X$  after  $T_i$  has terminated (aborted or committed).
- (ii)  $T_j$  writes a data item  $X$  after  $T_i$  has terminated (aborted or committed).

### Characterizing Schedules through Venn Diagram



## Concurrency Control with Locking

- Previously, we have characterised transaction schedules based on serializability and recoverability.
- If concurrency control with locking technique is used, then locks prevent multiple transactions from accessing the items concurrently.
- Access on data only, if TA 'has lock' on data.
- Transactions request and release locks.
- Schedule allows or differs operations based on lock table.

## Lock

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. (e.g., read lock, write lock). Locks are used as means of synchronizing the access by concurrent transactions to the database items. There is one lock per database item.

### Problems Arising with Locks

*There are two problems which are arised when using locks to control the concurrency among transactions*

**Deadlock** Two or more competing transactions are waiting for each other to complete to obtain a missing lock.

**Starvation** A transaction is continually denying the access to a given data item.

### Purpose of Concurrency Control with Locking

- To enforce isolation among conflicting transactions.
- To preserve database consistency.
- To resolve read-write, write-read and write-write conflicts.

*Locking is an operation that secures*

- Permission to read
- Permission to write a data item

**Lock (X)** Data item X is locked on behalf of the requesting transaction.  
Unlocking is an operation which removes these permissions from the data item.

**Unlock (X)** Data item X is made available to all other transactions.

### Types of Locks

*The two types of Locks are given below*

#### Binary Locks

*A binary lock has two states*

Locked/Unlocked

If a database item X is locked, then X cannot be accessed by any other database operation.

#### Shared/ Exclusive Locks (Read/Write Locks)

*There are three states*

Read locked/Write locked/Unlocked

Several transactions can access the same item X for reading (shared lock), however if any of the transactions want to write the item X, the transaction must acquire an exclusive lock on the item.

**Note** Using binary locks or read/write locks in transactions, does not guarantee serializability of schedules on its own.

### Key Points

- ♦ Lock and unlock are atomic operations (all or nothing).
- ♦ If every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable.

## Guaranteeing Serializability by Two-phase Locking

A transaction is said to follow the **two-phase locking protocol**, if all locking operations precede the first unlock operation in the transaction.

*Such a transaction can be divided into two phases*

**Expanding or Growing Phase** During which new locks on items can be acquired but none can be released.

**Shrinking Phase** During which existing locks can be released but no new locks can be acquired.

## Variations of Two-phase Locking (2PL) Technique

**Basic 2PL (2 Phase Locking)** Transaction locks data items incrementally. This may cause the problem of deadlock.

**Conservative 2PL (Static 2PL)** Prevents deadlock by locking all desired data items before transaction begins execution. However, it is difficult to use in practice because of the need to predeclare the read-set and write-set, which is not possible in most situations.

**Strict 2PL** A more stricter version of basic algorithm, where unlocking of write lock is performed after a transaction terminates (commits or aborts and rolled back). Hence, no other transaction can read or write an item that is written by transaction.

**Rigorous 2PL** Strict 2PL is not deadlock free. A more restrictive variation of strict 2PL, is rigorous 2PL, which also guarantees strict schedule. In this variation, a transaction does not release any of its locks (exclusive or shared) until after it commits or aborts.

## Deadlock

A deadlock is a condition in which two or more transaction are waiting for each other deadlock ( $T_1$  and  $T_2$ ). An example is given below

$T_1$	$T_2$
Read-lock (Y) $R(Y)$	Read-lock (X) $R(X)$
Write-Lock (X) (Waits for X)	Write-Lock (Y) ( Waits for Y)

## Deadlock Prevention

A transaction locks all data items, it refers to before it begins execution. This way of locking prevents deadlock, since a transaction never waits for a data item. The conservative two-phase locking uses this approach.

## Deadlock Detection and Resolution

In this approach, deadlocks are allowed to happen. The scheduler maintains a wait-for-graph for detecting cycle. If a cycle exists, then one transaction involved in the cycle is selected (victim) and rolled back.

### Key Points

- ♦ A wait-for-graph is created using the lock table. As soon as a transaction is blocked, it is added to the graph.
- ♦ When a chain like  $T_i$  waits for  $T_j$ ,  $T_j$  waits for  $T_k$  and  $T_k$  waits for  $T_i$  or  $T_j$  occurs, then this creates a cycle. One of the transaction of the cycle is selected and rolled back.

## Deadlock Avoidance

There are many variations of two-phase locking algorithm. Some avoid deadlock by not letting the cycle to complete. This is as soon as the algorithm discovers that blocking a transaction is likely to create a cycle, it rolls back the transaction.

*Following schemes use transaction times-tamps for the sake of deadlock avoidance*

### Wait-die scheme (Non-preemptive)

Older transaction may wait for younger one to release data item. Younger transactions never wait for older ones; they are rolled back instead. A transaction may die several times before acquiring the needed data item.

### Wound-wait scheme (Preemptive)

Older transaction bounds (forces rollback) of younger transaction instead of waiting for it. Younger transactions may wait for older ones. May be fewer rollbacks than wait-die scheme.

## Starvation

Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further. In a deadlock resolution scheme, it is possible that the same transaction may consistently be selected as victim and rolled back.

## **Time-stamp Based Concurrency Control Algorithm**

This is a different approach that guarantees serializability involves using transaction time-stamps to order transaction execution for an equivalent serial schedule.

### **Time-stamp**

A time-stamp is a unique identifier created by the DBMS to identify a transaction.

#### **Key Points**

- ♦ Time-stamp is a monotonically increasing variable (integer) indicating the age of an operation or a transaction.
- ♦ A larger time-stamp value indicates a more recent event or operation.

### **Starvation versus Deadlock**

Starvation	Deadlock
Starvation happens if same transaction is always chosen as victim.	A deadlock is a condition in which two or more transaction are waiting for each other.
It occurs if the waiting scheme for locked items is unfair, giving priority to some transactions over others.	A situation where two or more transactions are unable to proceed because each is waiting for one of the other to do something.
Starvation is also known as lived lock.	Deadlock is also known as circular waiting.
<b>Avoidance</b>	<b>Avoidance</b>
Switch priorities so that every thread has a chance to have high priority.	Acquire locks are predefined order.
Use FIFO order among competing requests.	Acquire locks at once before starting.
It means that transaction goes in a state where transaction never progresses.	It is a situation where transactions are waiting for each other.

The algorithm associates with each database item  $X$  with two Time Stamp (TS) values

#### **Read\_TS ( $X$ )**

The read time stamp of item  $X$ ; this is the largest time-stamp among all the time-stamps of transactions that have successfully read item  $X$ .

### **Write\_TS (X)**

The write time-stamp of item X ; this is the largest time-stamp among all the time-stamps of transactions that have successfully written item X.

*There are two Time-Stamp based Concurrency Control Algorithm*

### **Basic Time-stamp Ordering (TO)**

Whenever some transaction  $T$  tries to issue a  $R(X)$  or a  $W(X)$  operation, the basic time-stamp ordering algorithm compares the time-stamp of  $T$  with  $\text{read\_TS}(X)$  and  $\text{write\_TS}(X)$  to ensure that the time-stamp order of transaction execution is not violated. If this order is violated, then transaction  $T$  is aborted. If  $T$  is aborted and rolled back, any transaction  $T_1$  that may have used a value written by  $T$  must also be rolled back. Similarly, any transaction  $T_2$  that may have used a value written by  $T_1$  must also be rolled back and so on. This effect is known as cascading rollback.

*The concurrency control algorithm must check whether conflicting operations violate the time-stamp ordering in the following two cases*

#### **Transaction T issues a $W(X)$ operation**

If  $\text{read\_TS}(X) > \text{TS}(T)$  or if  $\text{write\_TS}(X) > \text{TS}(T)$ , then an younger transaction has already read the data item, so abort and roll back  $T$  and reject the operation.

If the condition in above part does not exist, then execute  $W(X)$  of  $T$  and set  $\text{write\_TS}(X)$  to  $\text{TS}(T)$ .

#### **Transaction T issues a $R(X)$ operation**

If  $\text{write\_TS}(X) > \text{TS}(T)$ , then an younger transaction has already written to the data item, so abort and rollback  $T$  and reject the operation.

If  $\text{write\_TS}(X) \leq \text{TS}(T)$ , then execute  $R(X)$  of  $T$  and set  $\text{read\_TS}(T)$  to the larger of  $\text{TS}(T)$  and the current  $\text{read\_TS}(X)$ .

### **Strict Time-stamp Ordering (TO)**

A variation of basic time-stamp ordering called strict time-stamp ordering, ensures that the schedules are both strict (for easy recoverability) and serializable (conflict).

**Transaction T issues a  $W(X)$  operation**   If  $\text{TS}(T) > \text{read\_TS}(X)$ , then delay  $T$  until the transaction  $T'$  that wrote or read  $X$  has terminated (committed or aborted).

**Transaction T issues a  $R(X)$  operation**   If  $\text{TS}(T) > \text{write\_TS}(X)$ , then delay  $T$  until the transaction  $T'$  that wrote or read  $X$  has terminated (committed or aborted).

### Thomas's Write Rule

- If  $\text{read\_TS}(X) > \text{TS}(T)$ , then abort and rollback  $T$  and reject the operation.
  - If  $\text{write\_TS}(X) > \text{TS}(T)$ , then just ignore the write operation and continue execution.  
This is because the most recent writes counts in case of two consecutive writes.
- If the conditions given in (i) and (ii) above do not occur, then execute  $W(X)$  of  $T$  and set  $\text{write\_TS}(X)$  to  $\text{TS}(T)$ .

## Multiversion Concurrency Control Techniques

This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus, unlike other mechanisms a read operation in this mechanism is never rejected.

**Side effect** Significantly more storage is required to maintain multiple versions.

## Validation Based Concurrency Control Protocol

*In this, execution of transaction  $T_i$  is done in three phases.*

**Read and Execution Phase** Transaction  $T_i$  writes only to temporary local variables.

**Validation Phase** Transaction  $T_i$  performs a validation test to determine, if local variables can be written without violating serializability.

**Write Phase** If  $T_i$  is validated, the updates are applied to the database, otherwise  $T_i$  is rolled back.

**Note** The above three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.

*Each transaction  $T_i$  has three time-stamps*

**Start ( $T_i$ )** The time when  $T_i$  started its execution.

**Validation ( $T_i$ )** The time when  $T_i$  entered its validation phase.

**Finish ( $T_i$ )** The time when  $T_i$  finished its write phase.

Serializability order is determined by time-stamp given at validation time to increase concurrency. Thus,  $\text{TS}(T_i)$  is given the value of validation ( $T_i$ ).

For all  $T_j$  with  $\text{TS}(T_i) < \text{TS}(T_j)$  either one of the following conditions holds

- (i)  $\text{Finish}(T_i) < \text{Start}(T_j)$
- (ii)  $\text{Start}(T_j) < \text{Finish}(T_i) < \text{Validation}(T_j)$

and the set of data items written by  $T_i$  does not intersect with the set of data items read by  $T_j$ . Then, validation succeeds and  $T_j$  can be committed. Otherwise, validation fails and  $T_j$  is aborted.

## Multiple Granularity

Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones. It can be represented graphically as a tree. When a transaction locks a node in the tree explicitly, it implicitly locks all the nodes descendants in the same node.

**Coarse Granularity** The larger the data item size, the lower the degree of concurrency.

**Fine Granularity** The smaller the data item size, the more locks to be managed and stored and the more lock/unlock operations needed.

## Key Points

---

- ◆ Multiple users can access databases and use computer system simultaneously because of the concept of multiprogramming, which allows the computer to execute multiple programs or processes at the same time.
  - ◆ If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible.
  - ◆ While one transaction is waiting for a page to be read from disk, the CPU can process another transaction. This may increase system throughput (the average number of transactions completed in a given time).
  - ◆ Interleaved execution of a short transaction with a long transaction allows the short transaction to complete first. This gives improved response time (average time taken to complete a transaction).
  - ◆ A transaction may be incomplete because the (database) system crashes or because it is aborted by either the system or the user (or application).
  - ◆ Complete transactions are committed.
  - ◆ Consistency and isolation are primarily the responsibility of a scheduler. Atomicity and durability are primarily responsibility of recovery manager.
- 

## Interleaving

Multiprogramming operating systems execute some commands from one process, then suspend that process and execute some commands from the next process and so on. A process is resumed at the point, where it was suspended, whenever it gets its turn to use the CPU again. Hence, this process is actually known as **interleaving**.

# **File Structures**

File structure or organisation refers to the relationship of the key of the record to the physical location of that record in the computer file.

## **Disk Storage**

Databases must be stored physically as files of records, which are typically stored on some computer storage medium. The DBMS software can then retrieve, update and process this data as needed.

*Several aspects of storage media must be taken into account*

- (i) Speed with which data can be accessed.
  - (ii) Cost per unit of data
  - (iii) Reliability
- Data loss on power failure or system crash.
  - Physical failure of the storage device.
  - So, we can differentiate storage into

### **Volatile Storage**

Losses contents when power is switched off.

### **Non-volatile Storage**

Contents persist even when power is switched off.

### **Category of Computer Storage Media**

*Computer storage media form a storage hierarchy that includes two main categories.*

- **Primary Storage** This category includes storage media that can be operated on directly by the computer Central Processing Unit (CPU), such as the computer main memory and smaller but faster cache memories. Primary storage usually provides fast access to data but is of limited storage capacity.
- **Secondary Storage** This category includes magnetic disks, optical disks and tapes. These devices usually have a larger capacity, cost less and provides slower access to data than do primary storage devices. Data in secondary storage cannot be processed directly by the CPU.

## **Characteristics of Secondary Storage Devices**

- (i) Random access versus sequential access
- (ii) Read-write, write-once, read-only
- (iii) Character versus block data access

## Storage of Databases

Need data to be stored permanently or persistently over long periods of time. Cost of storage per unit of data is an order of magnitude less for disk secondary storage than for primary storage.

## Magnetic Hard Disk Mechanism

Magnetic disks are used for storing large amounts of data. The most basic unit of data on the disk is a single bit of information. All disks are made of magnetic material shaped as a thin circular disk and protected by a plastic or acrylic cover. A disk is single-sided, if it stores information on one of its surfaces only and double-sided, if both surfaces are used.

### Key Points

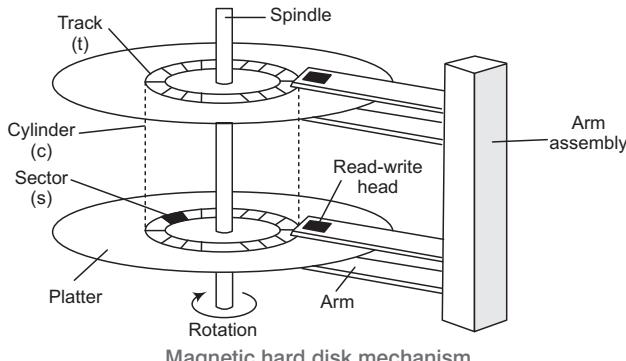
- ♦ To increase storage capacity, disks are assembled into a disk pack, which may include many disks.
- ♦ Information is stored on disk surface in concentric circles, each circle is called a track.
- ♦ The disk surfaces are called platter surfaces.

### Read-write Head

- Positioned very close to the platter surface (touching it).
- Reads or writes magnetically encoded information.

### Each Track is Divided into Sectors

- A sector is the smallest unit of data that can be read or written.
- Sector size typically 512 bytes.
- Typical sectors per track : 500 (on inner tracks) to 1000 (on outer tracks).



## Performance Measures on Disks

**Access Time** The time it takes from when a read or write request is issued to when data transfer begins. *It consists of*

- **Seek time** Time it takes to reposition the arm over the correct track. Average seek time is 1/2 of the worst case seek time.
- **Rotational latency** Time it takes for the sector to be accessed to appear under the head. Average latency is 1/2 of the worst case latency.

**Data Transfer Rate** The rate at which the data can be retrieved from or stored to the disk.

**Mean Time To Failure** (MTTF) The average time, the disk is expected to run continuously without any failure.

## Optimization of Disk-Block Access

**Block** A contiguous sequence of sectors from a single track. Data is transferred between disk and main memory in blocks. Sizes range from 512 bytes to several kilobytes. *Blocks are of two types*

- **Smaller blocks** more transfers from disk.
- **Larger blocks** more space wasted due to partially filled blocks.

**Disk-arm Scheduling** This algorithms order pending accesses to tracks so that disk arm movement is minimized.

**Elevator Algorithm** In this algorithm, move disk arm in one direction (from outer to inner tracks or vice-versa), processing next request in that direction, till no more request in that direction, then reverse direction and repeat.

## RAID (Redundant Arrays of Independent Disks)

The choice of disk structure is very important in databases. *Important factors, besides price are*

- (i) Capacity                    (ii) Speed                    (iii) Reliability

It is a disk organisations technique that manage a large numbers of disks, *providing a view of a single disk of*

- High capacity and high speed by using multiple disks in parallel and
- High reliability by storing data redundantly so that data can be recovered even if a disk fails.

## Storage Access in Databases

A database file is partitioned into fixed length storage units called blocks. Blocks are units of both storage allocation and data transfer. Database system seeks to minimize the number of block transfers between the disk and memory. The number of disk accesses can be reduced by keeping as many blocks as possible in main memory.

### Buffer

It is a portion of main memory that is available to store copies of disk blocks when several blocks need to be transferred from disk to main memory and all the block addresses are known, several buffers can be reserved in main memory to speed up the transfer while one buffer is being read or written, the CPU can process data in the other buffer.

### Buffer Manager

It is a subsystem which is responsible for allocating buffer space in main memory.

Programs call on the buffer manager when they need a block from disk.

- (i) If the block is already in the buffer, buffer manager returns the address of the block in main memory.
- (ii) If the block is not in the buffer, the buffer manager

#### Space in the Buffer

Allocates space in the buffer for the block.

- Replacing (throwing out) some other block, if required to make space for the new block.
- Replaced the block written back to disk only if it was modified, since the most recent time that it was written to/fetched from the disk.

Reads the block from the disk to the buffer and returns the address of the block in main memory to requester.

**Buffer Replacement Policies** Most operating systems replace the block that is Least Recently Used (LRU strategy).

**Most Recently Used (MRU) Strategy** System must pin the blocks currently being processed. After the final tuple of that block has been processed, the block is unpinned and it becomes the most recently used blocks.

**Pinned Block** Memory block that is not allowed to be written back to disk.

# File Organisation

The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields. Data is usually stored in the form of records. Records usually describe entities and their attributes. e.g., an employee record represents an employee entity and each field value in the record specifies some attributes of that employee, such as Name, Birth-date, Salary or Supervisor.

If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed length records. If different records in the file have different sizes, the file is said to be made up of variable length records.

## Spanned versus Unspanned Records

- The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory. When the block size is larger than the record size, each block will contain numerous records, although some of the files may have unusually large records that cannot fit in one block.
- Suppose that block size is  $B$  bytes. For a file of fixed length records of size  $R$  bytes, with  $B \geq R$ , then we can fit  $bfr = \left\lfloor \frac{B}{R} \right\rfloor$  records per block. The value of  $bfr$  is called the blocking factor.
- In general,  $R$  may not divide  $B$  exactly, so we have some unused space in each block equal to  $B - (bfr * R)$  bytes.
- To utilize this unused space, we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record. This organisation is called spanned.
- If records are not allowed to cross block boundaries, the organisation is called unspanned.

## Allocating File Blocks on Disk

*There are several standard techniques for allocating the blocks of a file on disk*

**Contiguous Allocation** The file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast.

**Linked Allocation** In this, each file contains a pointer to the next file block.

**Indexed Allocation** Where one or more index blocks contain pointers to the actual file blocks.

## File Headers

A file header or file descriptor contains information about a file that is needed by the system programs that access the file records.

## Operations on Files

*Operations on files are given below*

**Retrieval Operations** These do not change any data in the files but only locate certain records so that their field values can be examined and processed.

**Update Operations** These change the file by insertion or deletion of records or by modification of field values.

**Open** Prepares the file for reading or writing. Set the file pointer to the beginning of the file.

**Reset** Sets the file pointer of an open file to the beginning of the file.

**Find** (or **Locate**) Searches for the first record that satisfies a search condition. Transfers the block containing that record into a main memory buffer (if it is not already there).

**Read** (or less **Get**) Copies the current record from the buffer to a program variable in the user program.

**FindNext** Searches for the next record in the file that satisfies the search condition.

**Delete** Deletes the current record and updates the file on disk to reflect the deletion.

**Modify** Modifies some field values for the current record and updates the file on disk to reflect the modification.

**Insert** Insert a new record in the file by locating the block, where the record is to be inserted, transferring that block into the main memory buffer, writing the record into the buffer and writing the buffer to disk to reflect the insertion.

**Close** Completes the file access by releasing the buffers and performing any other needed cleanup operations.

## Files of Unordered Records (Heap Files)

In the simplest type of organization records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organisation is called a heap or pile file.

This organisation is often used with additional access paths, such as the secondary indexes.

In this type of organisation, inserting a new record is very efficient. Linear search is used to search a record.

### **Files of Ordered Records (Sorted Files)**

- We can physically order the records of a file on disk based on the values of one of their fields called the ordering field. This leads to an ordered or sequential file.
- If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record, then the field is called the ordering key for the file. Binary searching is used to search a record.

## **Hashing Techniques**

Another type of primary file organisation is based on hashing which provides very fast access to records under certain search conditions. This organisation is usually called a hash file. A hash file has also been called a direct file.

### **Key Points**

- The search condition must be an equality condition on a single field, called the hash field.
- $$\boxed{\text{Hash field} = \text{Key field}} \Rightarrow \boxed{\text{Hash key}}$$
- Hash function is used to determine page address for storing records. This is chosen to provide most even distribution of records and tends to minimum collisions.
- 

*There are various techniques for hashing*

### **Internal Hashing**

For internal files, hashing is typically implemented as a hash table through the use of an array of records and the array index range is from 0 to  $M - 1$ , then we have  $M$  slots in an array.

One common hash function is

$$h(K) = K \bmod M$$

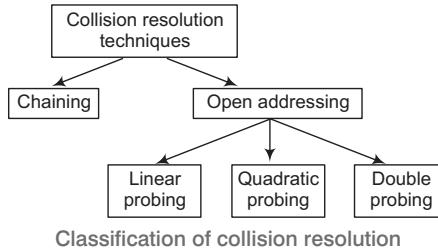
Which returns the remainder of an integer hash field value  $K$  after division by  $M$ .

### **Collision**

Hash function does not calculate unique address for two or more records. Here, a collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In

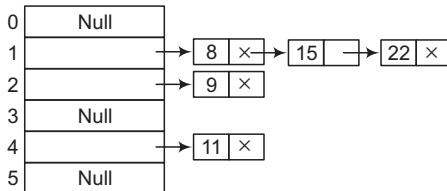
this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called **collision resolutions**.

*There are number of methods for collision resolution*



## Chaining

In this method, all the elements, where keys hash to the same hash table slot are put in linked list manner.



## Collision Resolution by Open Addressing

In open addressing, the keys to be hashed is to put in the separate location of the hash table. Each location contains some key or the some other character to indicate that the particular location is free.

In this method to insert key into the table, we simply hash the key using the hash function. If the space is available, then insert the key into the hash table location, otherwise search the location in the forward direction of the table to find the slot in a systematic manner. The process of finding the slot in the hash table is called **probing**.

**Linear probing** *The linear probing use the following hash function*

$$h(k, i) = [h'(k) + i] \bmod m \text{ for } i = 0, 1, 2, \dots, m - 1,$$

where  $m$  is the size of the hash table,  $h'(k) = k \bmod m$ , the basic function, and  $i$  = The probe number.

**Quadratic probing** *The quadratic probing uses the following hash function*

$$h(k, i) = [h'(k) + c_1 i + c_2 i^2] \bmod m \text{ for } i = 0, 1, \dots, m - 1$$

where,       $m$  = size of hash table

$h'(k) = k \bmod m$ , the basic hash function

$c_1$  and  $c_2 \neq 0$  are auxiliary and  $i$  = the probe number.

**Double hashing** In double hashing, second hashing function  $H'$  is used for resolving a collision. Suppose a record  $R$  with key  $K$  has the hash address

$$H(K) = h \text{ and } H'(K) = h' \neq m$$

Then, we linearly search the location with addresses

$$h, h + h', h + 2h', h + 3h', \dots$$

If  $m$  is prime number, then the above sequence will access all the locations in the table  $T$ .

## External Hashing for Disk Files

Hashing for disk files is called external hashing. To suit the characteristics of disk storage, the target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hashing function maps a key into a relative bucket number, rather than assigning an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address. The collisions problem is less severe with buckets.

## Indexing Structures for Files

Indexing mechanism are used to optimize certain accesses to data (records) managed in files. e.g., the author catalog in a library is a type of index. Search key (definition) attribute or combination of attributes used to look-up records in a file.

An index file consists of records (called index entries) of the form.

Search key value	Pointer of block in data file
------------------	-------------------------------

Indexing structures for files

Index files are typically much smaller than the original file because only the values for search key and pointer are stored. The most prevalent types of indexes are based on ordered files (single-level indexes) and tree data structures (multilevel indexes).

## Types of Single Level Ordered Indexes

In an ordered index file, index entries are stored sorted by the search key value. *There are several types of ordered Indexes*

## Primary Index

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field called the primary key of the data file and the second field is a pointer to a disk block (a block address).

### Key Points

- ♦ There is one index entry in the index file for each block in the data file.
- ♦ Indexes can also be characterised as dense or sparse.
- ♦ **Dense index** A dense index has an index entry for every search key value in the data file.
- ♦ **Sparse index** A sparse index (non-dense), on the other hand has index entries for only some of the search values.
- ♦ A primary index is a non-dense (sparse) index, since it includes an entry for each disk block of the data file rather than for every search value.

## Clustering Index

If file records are physically ordered on a non-key field which does not have a distinct value for each record that field is called the clustering field. We can create a different type of index, called a clustering index, to speed up retrieval of records that have the same value for the clustering field.

### Key Points

- ♦ A clustering index is also an ordered file with two fields. The first field is of the same type as the clustering field of the data file.
- ♦ The record field in the clustering index is a block pointer.
- ♦ A clustering index is another example of a non-dense index.

## Secondary Index

A secondary index provides a secondary means of accessing a file for which some primary access already exists. The secondary index may be on a field which is a candidate key and has a unique value in every record or a non-key with duplicate values. The index is an ordered file with two fields. The first field is of the same data type as some non-ordering field of the data file that is an indexing field. The second field is either a block pointer or a record pointer.

A secondary index usually needs more storage space and longer search time than does a primary index.

### **Multilevel Indexes**

The idea behind a multilevel index is to reduce the part of the index. A multilevel index considers the index file, which will be referred now as the first (or base) level of a multilevel index. Therefore, we can create a primary index for the first level; this index to the first level is called the second level of the multilevel index and so on.

### **Dynamic Multilevel Indexes Using B-Trees and B<sup>+</sup>-Trees**

*There are two multilevel indexes*

#### **B-Trees**

- When data volume is large and does not fit in memory, an extension of the binary search tree to disk based environment is the B-tree.
- In fact, since the B-tree is always balanced (all leaf nodes appear at the same level), it is an extension of the balanced binary search tree.
- The problem which the B-tree aims to solve is given a large collection of objects, each having a key and a value, design a disk based index structure which efficiently supports query and update.
- A B-tree of order  $p$ , when used as an access structure on a key field to search for records in a data file, can be defined as follows

- (i) Each internal node in the B-tree is of the form

$$< P_1, < K_1, P_{r_1} >, P_2, < K_2, P_{r_2} >, \dots < K_{q-1}, P_{r_{q-1}} >, P_q >$$

where,  $q \leq p$ .

Each  $P_i$  is a tree pointer to another node in the B-tree.

Each  $P_{r_j}$  is a data pointer to the record whose search key field value is equal to  $K_j$ .

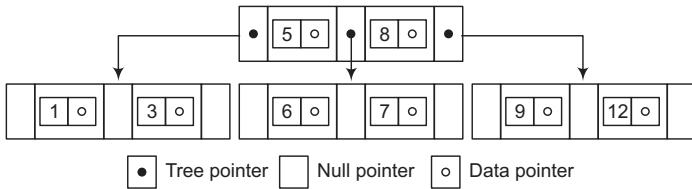
- (ii) Within each node,  $K_1 < K_2 < \dots < K_{q-1}$ .

- (iii) Each node has at most  $p$  tree pointers.

- (iv) Each node, except the root and leaf nodes, has atleast  $\lceil (p/2) \rceil$  tree pointers.

- (v) A node within  $q$  tree pointers  $q \leq p$ , has  $q - 1$  search key field values (and hence has  $q - 1$  data pointers).

e.g., A B-tree of order  $p = 3$ . The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.



### B<sup>+</sup> Trees

- It is the variation of the B-tree data structure.
- In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer. In a B<sup>+</sup>-tree, data pointers are stored only at the leaf nodes of the tree. Hence, the structure of the leaf nodes differs from the structure of internal nodes.
- The pointers in the internal nodes are tree pointers to blocks that are tree nodes whereas the pointers in leaf nodes are data pointers.

#### B<sup>+</sup> Tree's Structure

The structure of the B<sup>+</sup> -tree of order  $p$  is as follows

- Each internal node is of the form  $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$   
Where,  $q \leq p$  and each  $P_i$  is a tree pointer.
- Within each internal node,  $K_1 < K_2 < K_3 \dots < K_{q-1}$ .
- Each internal node has at most  $p$  tree pointers and except root, has atleast  $\lceil (p/2) \rceil$  tree pointers. The root node has atleast two tree pointers, if it is an internal node.
- Each leaf node is of the form.

$\langle\langle K_1, P_{f_1} \rangle, \langle K_2, P_{f_2} \rangle, \dots, \langle K_{q-1}, P_{f_{q-1}} \rangle, P_{\text{next}} \rangle$

Where,  $q \leq p$ , each  $P_{f_i}$  is a data pointer and  $P_{\text{next}}$  points to the next leaf node of the B<sup>+</sup> -trees.

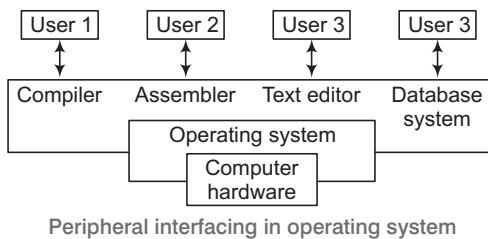
# 5

# Operating System

---

## Operating System

An operating system acts as an intermediary between the user of a computer and the computer hardware. An Operating System (OS) is a software that manages the computer hardware.



## Components of a Computer System

**Hardware** It provides the basic computing resources for the system. It consists of CPU, memory and the input/output (I/O) devices.

**Application Programs** Define the ways in which these resources are used to solve user's computing problems. e.g., word processors, spreadsheets, compilers and web browsers.

## Functions of Operating System

Operating system provides many functions for ensuring the efficient operation of the system itself. *Some functions are listed below*

**Resource Allocation** Allocation of resources to the various processes is managed by operating system.

**Accounting** Operating system may also be used to keep track of the various computer resources and how much and which users are using these resources.

**Protection** Protection ensures that all access to the system resources is controlled. When several processes execute concurrently, it should not be possible for one process to interface with the other or with the operating system itself.

## Operating System Services

Many services are provided by OS to the user's programs.

*Some of the OS services are listed below*

**Program Execution** The operating system helps to load a program into memory and run it.

**I/O Operations** Each running program may request for I/O operation and for efficiency and protection the users cannot control I/O devices directly. Thus, the operating system must provide some means to do I/O operations.

**File System Manipulation** Files are the most important part which is needed by programs to read and write the files and files may also be created and deleted by names or by the programs. The operating system is responsible for the file management.

**Communications** Many times, one process needs to exchange information with another process, this exchange of information can takes place between the processes executing on the same computer or the exchange of information may occur between the process executing on the different computer systems, tied together by a computer network. All these things are taken care by operating system.

**Error Detection** It is necessary that the operating system must be aware of possible errors and should take the appropriate action to ensure correct and consistent computing.

## **Batch Systems**

Early computers were the large machines that run from a console. The users of batch systems did not interact directly with the computer systems. Rather, the user prepared a job which consists of programs, data and control information and then submit it to the computer operator. The job prepared would be in the form of punch cards. After sometime perhaps minutes, hours or days, the output is prepared.

### **Key Points**

- ♦ The main drawback of batch system is the lack of interaction between the user and the job while it was executing.
- ♦ Multiprogramming creates logical parallelism.

## **Multiprogramming**

It is the technique of running several programs at a time using time sharing. It allows a computer to do several things at the same time.

The concept of multiprogramming is that the OS keeps several jobs in memory simultaneously. The operating system selects a job from the job pool and starts executing a job, when that job needs to wait for any input/output operations, the CPU is switched to another job. So, the main idea here is that the CPU is never idle.

## **Multitasking**

Multitasking is the logical extension of multiprogramming. The concept of multitasking is quite similar to multiprogramming but difference is that the switching between jobs occurs so frequently that the users can interact with each program while it is running.

This concept is also known as **time sharing system**. A time shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of time shared system.

## **Real Time Systems**

A real time operating system is used in environments, where a large number of events, mostly external to the computer system must be accepted and processed in short time or within certain deadlines.

Real time systems are used when there are rigid time requirements on the flow of data or the operation of processor and therefore these are used as a control device in a dedicated application.

## Distributed Systems

In distributed systems, the computation is distributed among several processors. Each processor in distributed system has its own local memory and do not share memory or a clock.

A distributed operating system governs the operation of a distributed computer system and provides a virtual machine abstraction to its users.

### Key Points

---

- ♦ A pool of job on disk allows the OS to select which job to run next, to increase CPU utilization
  - ♦ If jobs come directly on cards (or on magnetic tape), they run sequentially on FCFS basis but when several jobs are on direct access devices like disk, job scheduling is possible.
  - ♦ The most important aspect of job scheduling is the ability to multiprogramming.
  - ♦ If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them, making this decision is job scheduling.
  - ♦ When the OS selects a job from job pool, it loads that job into memory. This cause the residence of several programs in memory and calls for memory management scheme.
  - ♦ If several jobs are ready to run at same time, the system must choose among them. This decision is called **CPU scheduling**.
-

# **Threads**

A thread is a basic unit of CPU utilisation. A thread comprises a 1-dimensional program counter , a register set and a stack. It shares with other threads belonging to the same process its code section, data section and other system resources such as open files and signals . A traditional process has a single thread of control. If a process has multiple thread of control, it can perform more than one task at a time.

## **Multithreading**

An application typically is implemented as a separate process with several threads of control. In some situations, a single application may be required to perform several similar tasks. e.g., a web server accepts client request for web pages, images, sound and so on. A busy web server may have several of clients concurrently accessing it. If the web server run as a traditional single threaded process, it would be able to service only one client at a time.

The amount of time that a client might have to wait for its request to be serviced could be enormous. So, it is efficient to have one process that contains multiple threads to serve the same purpose.

This approach would multithreaded the web server process, the server would create a separate thread that would listen for client requests, when a request was made rather than creating another process, it would create another thread to service the request.

## **Multithreading Model**

*There are two types of threads*

- (i) User threads
- (ii) Kernel threads

### **Kernel Threads**

Kernel threads are supported and managed directly by the operating system.

### **User Threads**

They are above the kernel and they are managed without kernel support.  
*There are three common ways of establishing relationship between user threads and kernel threads*

- (a) Many-to-many model
- (b) One-to-one model
- (c) Many-to-one model

- **One-to-one model** maps each user thread to corresponding kernel threads.
- **Many-to-many model** multiplexes many user threads to a smaller or equal number of kernel threads.
- **Many-to-one model** maps many user threads to single kernel threads.

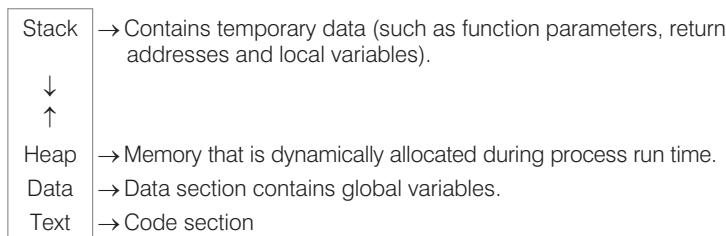
### Key Points

- ♦ User level threads are threads that are visible to the programmer and unknown to the Kernel.
- ♦ User level threads are faster to create and manage than that of Kernel threads.

## Process

A process is a program in **execution**. A process is more than the program code *i.e.*, text section. It also includes the current activity as represented by the value of the program counter and the contents of the processor's register.

Max



A simple process block

## Process in Memory

Each process is represented in the OS by a **Process Control Block** (PCB) also called a **task control block**.

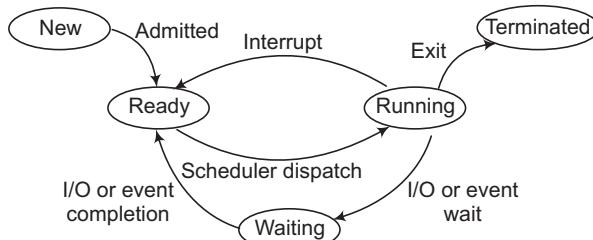


Diagram of a process state

As processes enter the system, they are put into a job queue, which consists of all processes in the system.

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.

Process state
Process number
Program counter
Registers
Memory limits
List of open files
:

Process control block

## Key Points

- ♦ The list of processes waiting for a particular I/O device is called a **device queue**. Each device has its own device queue.
- ♦ I/O bound process is one that spends more of its time doing I/O rather than it spends doing computations.
- ♦ A CPU bound process is one which uses more of its time doing computations rather than it spends doing I/O activities.

## Schedulers

A process migrates among various scheduling queues throughout its lifetime. The OS must select for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.

### Long Term and Short Term Schedulers

- A long term scheduler or job scheduler selects processes from job pool (mass storage device, where processes are kept for later execution) and loads them into memory for execution.
- A short term scheduler or CPU scheduler, selects from the main memory among the processes that are ready to execute and allocates the CPU to one of them.
- The long term scheduler controls the degree of multiprogramming (the number of processes in memory).

**Note** Mid-term scheduler uses the concept of swapping.

### Dispatcher

It is the module that gives control of the CPU to the process selected by the short term scheduler. *This function involves the following*

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program.

# Scheduling Algorithm

All of the processes which are ready to execute and are placed in main memory then selection of one of those processes is known as **scheduling**, and after selection that process gets the control of CPU.

## Scheduling Criteria

*The criteria for comparing CPU scheduling algorithms include the following*

**CPU Utilization** Means keeping the CPU as busy as possible.

**Throughput** It is nothing but the measure of work i.e., the number of processes that are completed per time unit.

**Turnaround Time** The interval from the time of submission of a process to the time of completion. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

**Waiting Time** The sum of the periods spent waiting in the ready queue.

**Response Time** The time from the submission of a request until the first response is produced.

## Key Points

- ♦ The amount of time, a process takes to start responding not the time it takes to output the response is called response time.
- ♦ Response time is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time and response time.

*There are many CPU scheduling algorithms as given below*

## First Come First Served (FCFS) Scheduling

With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with FIFO queue. When a process enters the ready queue, its PCB (Process Control Block) is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

Gantt chart			Process Table		
$P_1$	$P_2$	$P_3$	Process	Burst Time (in milliseconds)	
0	24	27	$P_1$	24	
Waiting time for $P_1 = 0$ ms			$P_2$	3	
Waiting time for $P_2 = 24$ ms			$P_3$	3	
Waiting time for $P_3 = 27$ ms				Turn around time for $P_1 = 24 - 0 = 24$ ms	
Average waiting time				Turn around time for $P_2 = 27 - 24 = 3$ ms	
				Turn around time for $P_3 = 30 - 27 = 3$ ms	

## Shortest Job First (SJF) Scheduling

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the two processes have the same length or amount of next CPU burst, FCFS scheduling is used to break the tie.

Process Table		Gantt Chart			
Process	Burst Time (in milliseconds)	$P_4$	$P_1$	$P_3$	$P_2$
$P_1$	6	0	3	9	16
$P_2$	8				24
$P_3$	7				
$P_4$	3				

waiting time for  $P_1 = 3$

waiting time for  $P_2 = 16$

waiting time for  $P_3 = 9$

waiting time for  $P_4 = 0$

$$\text{Average waiting time} = \frac{3 + 16 + 9 + 0}{4} = 7 \text{ ms}$$

A more appropriate term for this scheduling method would be the shortest next CPU burst algorithm because scheduling depends on the length of the next CPU burst of a process rather than its total length.

## Preemptive and Non-preemptive Algorithm

The SJF algorithm can either be preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is still executing. The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process.

A preemptive SJF algorithm will preempt the currently executing process, whereas a non-preemptive algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called **shortest-remaining-time-first-scheduling**.

**Process Table**

Process	Arrival Time	Burst Time
$P_1$	0	8
$P_2$	1	4
$P_3$	2	9
$P_4$	3	5

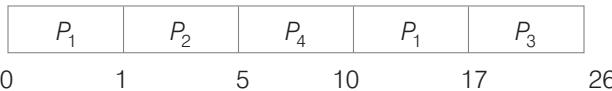
$$\text{Waiting time for } P_1 = (10 - 1) = 9$$

$$\text{Waiting time for } P_2 = (1 - 1) = 0$$

$$\text{Waiting time for } P_3 = 17 - 2 = 15$$

$$\text{Waiting time for } P_4 = 5 - 3 = 2$$

**Gantt Chart**



$$\text{Average waiting time} = \frac{9 + 0 + 15 + 2}{4} = 6.5 \text{ ms}$$

## Priority Scheduling

A priority is associated with each process and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order.

We can be provided that low numbers represent high priority or low numbers represent low priority. According to the question, we need to assume any one of the above.

### Key Points

- ♦ Priority scheduling can be either preemptive or non-preemptive.
- ♦ A preemptive priority scheduling algorithm will preempt the CPU, if the priority of the newly arrived process is higher than the priority of the currently running process.
- ♦ A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

A major problem with priority scheduling algorithm is indefinite blocking or starvation.

**Process Table**

Process	Burst Time	Priority
$P_1$	10	3
$P_2$	1	1
$P_3$	2	4
$P_4$	1	5
$P_5$	5	2

Waiting time for  $P_1 = 6$

Waiting time for  $P_2 = 0$

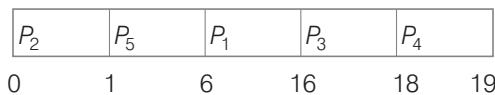
Waiting time for  $P_3 = 16$

Waiting time for  $P_4 = 18$

Waiting time for  $P_5 = 1$

$$\text{Average waiting time} = \frac{6 + 0 + 16 + 18 + 1}{5} = 8.2 \text{ ms}$$

**Gantt Chart**



## Round Robin (RR) Scheduling

The RR scheduling algorithm is designed especially for time sharing systems. It is similar to FCFS scheduling but preemption is added to switch between processes. A small unit of time called a **time quantum** or **time slice** is defined.

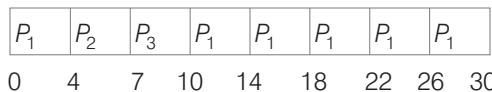
The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval of up to 1 time quantum. The process may have a CPU burst of less than 1 time quantum, in this case the process itself will release the CPU voluntarily. The scheduler will then proceed to next process in the ready queue.

Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the time will go off and will cause an interrupt to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue. The CPU scheduler will then select the next process in the ready queue.

### Process Table

Process	Burst Time (in milliseconds)
$P_1$	24
$P_2$	3
$P_3$	3

Let's take time quantum = 4 ms. Then the resulting RR schedule is as follows

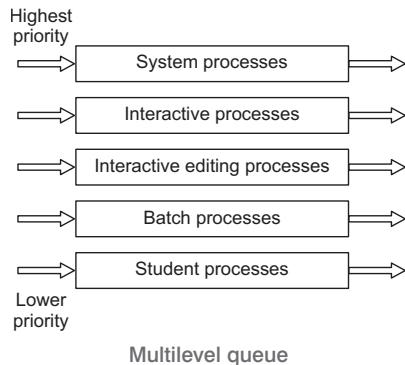


$P_1$  waits for the 6 ms ( $10 - 4$ ),  $P_2$  waits for 4 ms and  $P_3$  waits for 7 ms. Thus,

$$\text{Average waiting time} = \frac{6 + 4 + 7}{3} = 5.66 \text{ ms}$$

### Multilevel Queue Scheduling

This scheduling algorithm has been created for situations in which processes are easily classified into different groups. e.g., a division between foreground (interactive) processes and background (batch) processes. A multilevel queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority or process type.



### Multilevel Feedback Queue Scheduling

This scheduling algorithm allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU bursts.

If a process uses too much CPU time, it will be moved to a lower priority queue. Similarly, a process that waits too long in a lower priority queue may be moved to a higher priority queue. This form of aging prevents starvation.

## Synchronization

When several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called **race condition**.

e.g., Suppose we have two variables  $A$  and  $B$ . The operation on  $A$  and  $B$  are as follows

**Operation 1**    $A = \text{Result}$       **Operation 2**    $B = \text{Result}$

$A = A + 1$                            $B = B - 1$

    Result =  $A$                           Result =  $B$

Now, initially if value of result = 4 and sequence is operation 1, then operation 2. Then,

$$A = 4$$

$$A = 4 + 1 = 5$$

$$\text{Result} = A = 5$$

$$B = \text{Result} = 5$$

$$B = 5 - 1 = 4$$

$$\text{Result} = B = 4$$

Now, if the sequence of operation gets changed as is operation 2, then operation 1. Then,

$$B = \text{Result} = 4$$

$$B = B - 1 = 4 - 1 = 3$$

$$\text{Result} = B = 3$$

$$A = \text{Result} = 3$$

$$A = A + 1 = 4$$

$$\text{Result} = A = 4$$

For the race condition above, we need to ensure that only one process at a time can be manipulating the variable result. To make such a guarantee, we require that the processes be synchronized in some way.

## Inter-Process Communication (IPC)

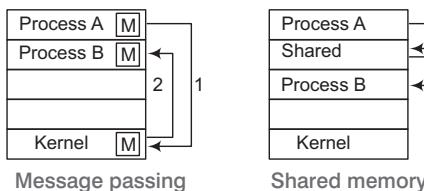
Processes executing concurrently in the operating system may be either independent or cooperating processes. A process is independent, if it can't affect or be affected by the other processes executing in the system. Any process that shares data with other processes is a cooperating process. Advantages of process cooperations are information sharing, computation speed up, modularity and convenience to work on many task at the same time. Cooperating processes require an Inter-Process Communication (IPC) mechanism that will allow them to exchange data and information.

*There are two fundamental models of IPC*

- (i) Shared memory
- (ii) Message passing

In the shared memory model, a region of memory that is shared by cooperating process is established. Process can then exchange information by reading and writing data to the shared region.

In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.



## The Critical Section Problem

Consider a system containing of  $n$  processes  $\{P_0, P_1, \dots, P_{n-1}\}$ . Each process has a segment of code, called a **critical section**, in which the process may be changing common variables, updating a table, writing a file and so on.

The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.

- The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section.
- The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.

General structure of a typical process  $P_i$  is given below

```
do {
    Entry Section
    Critical Section
    Exit Section
    Remainder Section
} while (True);
```

A solution to critical section problem must satisfy the following three requirements

### **Mutual Exclusion**

If a process  $P_i$  is executing in its critical section, then no other processes can be executing in their critical sections.

### **Progress**

If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next and this selection can't be postponed indefinitely.

### **Bounded Waiting**

There exists a bound or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

## **Semaphores**

Semaphore is nothing but a synchronization tool. A semaphore  $S$  is an integer variable that apart from initialization, is accessed only through two standard atomic operations wait ( ) and signal ( ). The wait ( ) operation does testing of the integer value of  $S$  ( $S \leq 0$ ) as well as its possible modification ( $S--$ ).

```
wait (S)
{
    while S < = 0
        i; // no operation
        S --;
}
```

The signal ( ) operation does increment to the integer value of  $S$  ( $S++$ ).

```
Signal (S)
{
    S++;
}
```

All modifications to the integer value of semaphore in the wait ( ) and signal ( ) operations must be executed indivisibly.

## Usage of Semaphores

- We can use binary semaphores to deal with the critical section problem for multiple processes. The  $n$  processes share a semaphore, mutex, initialized to 1.
- The value of binary semaphore can range only between 0 and 1.
- Binary semaphores are known as **mutex locks** as they are locks that provide mutual exclusion.

Mutual exclusion implementation with semaphores

```
do
{
    wait (mutex);
    // Critical section
    signal (mutex);
    // Remainder section
} while (TRUE);
```

## Key Points

- ♦ Counting semaphores can be used to control access to a given resource consisting of a finite number of instances.
- ♦ The value of a counting semaphore can range over an **unrestricted domain**.

## Classical Problem of Process Synchronization

1. Producer-consumer problem
2. Readers-writer problem
3. Dining-philosopher problem

### Producer-Consumer Problem

*In this problem, there are two processes*

(i) producer process                      (ii) consumer process.

A shared buffer (global) is defined which is accessible by producer and consumer.

- Producer process produces items and appends these in buffer.
- Consumer process consumes items from buffer.

### The Readers-Writers Problem

Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the databases (we can say these as readers). Whereas other processes may want to update (that is, to read and write) the database (we can say these processes as writers). Obviously, if two readers access the shared data simultaneously, no adverse effects will result.

- However, if a write and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.
- To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to database.
- This synchronization problem is referred to as the readers-writers problem.

### Key Points

- ♦ Both of the processes (*i.e.*, producer and consumer) cannot access buffer simultaneously.
- ♦ Mutual exclusion should be here.
- ♦ Producer cannot append item in full buffer (finite buffer) and consumer can not take item from empty buffer.

### The Dining-Philosophers Problem

Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by 5 chairs, each belonging to one philosopher. In the center of the table is a bowl of rice and the table is laid with 5 chopsticks. When a philosopher thinks, he doesn't interact with his colleagues.

From time to time, a philosopher gets hungry and tries to pick up the 2 chopsticks that are closest to him (the chopsticks that are between him and his left and right neighbours). A philosopher may pick up only one chopstick at a time. Obviously, he can't pick up a chopstick that is already in the hand of a neighbour. When a hungry philosopher has both his Chopsticks at the same time. He eats without releasing his chopsticks. When he is finished, he puts down both of his chopsticks and starts thinking again.

This problem is a simple representation of the need to allocate several resources among several processes in a deadlock free and starvation free manner.

# Deadlock

In a multiprogramming environment, a situation when permanent blocking of a set of processes that either compete for system resources or communicate with each other happens, we can call this as deadlock situation. This deadlock problem involves conflicting needs for resources by two or more processes.

## Necessary Conditions for Deadlock

A deadlock situation can arise, if the following four conditions hold simultaneously in a system.

### Mutual Exclusion

Resources must be allocated to processes at any time in an exclusive manner and not on a shared basis for a deadlock to be possible. If another process requests that resource, the requesting process must be delayed until the resource has been released.

### Hold and Wait Condition

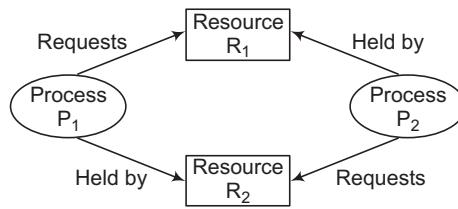
Even if a process holds certain resources at any moment, it should be possible for it to request for new ones. It should not give up (release) the already held resources to be able to request for new ones. If it is not true, a deadlock can never take place.

### No Preemption Condition

Resources can't be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed its task.

### Circular Wait Condition

There must exist a set =  $\{P_0, P_1, P_2, \dots, P_n\}$  of waiting processes such that  $P_0$  is waiting for a resource that is held by  $P_1$ ,  $P_1$  is waiting for a resource that is held by  $P_2, \dots, P_{n-1}$  is waiting for a resource that is held by  $P_n$  and  $P_n$  is waiting for a resource that is held by  $P_0$ .



State diagram of circular wait condition

## Resource Allocation Graph

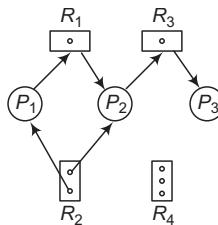
The resource allocation graph consists of a set of vertices  $V$  and a set of edges  $E$ .

*Set of vertices  $V$  is partitioned into two types*

- (i)  $P = \{P_1, P_2, \dots, P_n\}$ , the set consisting of all the processes in the system.
- (ii)  $R = \{R_1, R_2, \dots, R_m\}$ , the set consisting of all resource types in the system.
- Directed Edge  $P_i \rightarrow R_j$  is known as **request edge**.
- Directed Edge  $R_j \rightarrow P_i$  is known as **assignment edge**.

### Resource Instance

- One instance of resource type  $R_1$ .
- Two instances of resource type  $R_2$ .
- One instance of resource type  $R_3$ .
- Three instances of resource type  $R_4$ .



Example of resource allocation graph

## Process States

- Process  $P_1$  is holding an instance of resource type  $R_2$  and is waiting for an instance of resource type  $R_1$ .
- Process  $P_2$  is holding an instance of  $R_1$  and  $R_2$  is waiting for an instance of resource type  $R_3$ .
- Process  $P_3$  is holding an instance of  $R_3$ .
- *Basic facts related to resource allocation graphs are given below*

**Note** If graph consists no cycle it means there is no deadlock in the system.

If graph contains cycle

- (i) If only one instance per resource type, then deadlock.
- (ii) If several instances per resource type, then there may or may not be deadlock.

## Deadlock Handling Strategies

1. Deadlock prevention
2. Deadlock avoidance
3. Deadlock detection

### Deadlock Prevention

Deadlock prevention is a set of methods for ensuring that atleast one of the necessary conditions can't hold.

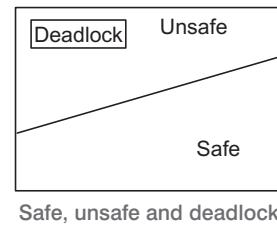
### Deadlock Avoidance

A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

#### Safe State

A state is safe, if the system can allocate resources to each process and still avoid a deadlock.

A system is in safe state, if there exists a safe sequence of all processes. A deadlock state is an unsafe state. Not all unsafe states cause deadlocks.



### Banker's Algorithm

Data structures for Banker's algorithm available. Vector of length  $m$ . If  $\text{available}[j] = k$ , there are  $k$  instances of resource type  $R_j$  available.

- **Max**  $n \times m$  matrix. If  $\text{max}[i, j] = k$ , then process  $P_i$  may request at most  $k$  instances of resource type  $R_j$ .
- **Allocation**  $n \times m$  matrix. If  $\text{allocation}[i, j] = k$ , then  $P_i$  is currently allocated  $k$  instances of  $R_j$ .
- **Need**  $n \times m$  matrix. If  $\text{need}[i, j] = k$ , then  $P_i$  may need  $k$  more instances of  $R_j$  to complete its task.

$$\text{Need}[i, j] = \text{max}[i, j] - \text{allocation}[i, j]$$

- **Safety Algorithm**

**Step 1** Let work and finish be vectors of length  $m$  and  $n$ , respectively.  
Initialize

$$\text{Work} = \text{Available}$$

$$\text{Finish}[i] = \text{False} \quad (\text{for } i = 1, 2, \dots, n)$$

**Step 2** Find  $i$  such that both

(a)  $\text{Finish}[i] = \text{False}$

(b)  $\text{Need} \leq \text{Work}$

If no such  $i$  exists, go to step 4.

**Step 3**  $\text{Work} = \text{Work} + \text{Allocation}$

$\text{Finish}[i] = \text{True}$

Go to step 2.

**Step 4**  $\text{Finish}[i] = \text{True}$  (for all  $i$ )

Then, the system is in a safe system.

### Example Banker's Algorithm

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
$P_0$	0	1	0	7	5	3	3	3	2
$P_1$	2	0	0	3	2	2			
$P_2$	3	0	2	9	0	2			
$P_3$	2	1	1	2	2	2			
$P_4$	0	0	2	4	3	3			

$\text{Need} = \text{Max} - \text{Allocation}$

Thus, we have

	Allocation			Max			Need			
	A	B	C	A	B	C	A	B	C	
$P_0$	7	4	3				$P_0$	7	4	3
	$(7 - 0)$	$(5 - 1)$	$(3 - 0)$				$P_1$	1	2	2
$P_1$	1	2	2				$P_2$	6	0	0
	$(3 - 2)$	$(2 - 0)$	$(2 - 0)$				$P_3$	0	1	1
$P_2$	6	0	0				$P_4$	4	3	1
	$(9 - 3)$	$(0 - 0)$	$(2 - 2)$							
$P_3$	0	1	1							
	$(2 - 2)$	$(2 - 1)$	$(2 - 1)$							
$P_4$	4	3	1							
	$(4 - 0)$	$(3 - 0)$	$(3 - 2)$							

$\Rightarrow$

As available resources are (3 3 2). The process  $P_1$  with need (1 2 2) can be executed.

Available resource = Available + Allocated resource of  $P_1$

$$\begin{array}{r} 3 \quad 3 \quad 2 \\ 2 \quad 0 \quad 0 \\ \hline 5 \quad 3 \quad 2 \end{array}$$

Now, available resources are (5 3 2).

The next process that can be executed after assigning available resource is  $P_3$ . Thus,  $P_3$  will execute next.

Now, available resource = Available resource + Resource of  $P_3$

$$\begin{array}{r} 5 \quad 3 \quad 2 \\ 2 \quad 1 \quad 1 \\ \hline 7 \quad 4 \quad 3 \end{array}$$

Now, available resources are 7 4 3. Next process will be  $P_4$ .

$$\begin{array}{r} A \quad B \quad C \\ 7 \quad 4 \quad 3 \\ 0 \quad 0 \quad 2 \\ \hline 7 \quad 4 \quad 5 \end{array}$$

Next process that will be executed is  $P_0$ .

Available resource

$$\begin{array}{r} 7 \quad 4 \quad 5 \\ 0 \quad 1 \quad 0 \\ \hline 7 \quad 5 \quad 5 \end{array}$$

## Key Points

- ♦ The sequence  $\langle P_1, P_3, P_4, P_0, P_2 \rangle$  ensures that the deadlock will never occur.
- ♦ If a safe sequence can be found that means the processes can run concurrently and will never cause a deadlock.
- ♦ First in first out scheduling based on queuing.
- ♦ Shortest seek time scheduling is designed for maximum throughput in most scenarios
- ♦ RR scheduling involves extensive overhead, especially with a small time unit.
- ♦ Kernel is defined as a program running all the time on the computer. It is the part of operating system that loads first. In simple words, kernel provides the communication between system software and hardware.

### Deadlock Detection

Allow system to enter deadlock state and then there is

- (i) Detection algorithm
- (ii) Recovery scheme

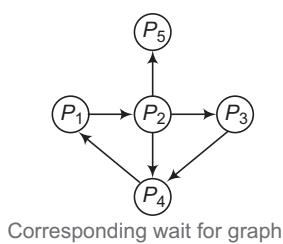
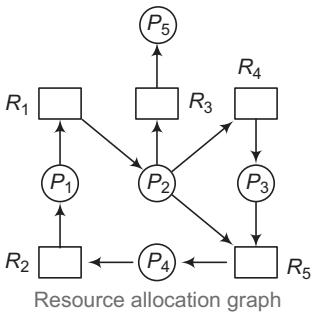
Single instance of each resource type

- (i) Maintain wait for graph.

(a) Nodes are processes.    (b)  $P_i \rightarrow P_j$ , if  $P_i$  is waiting for  $P_j$ .

- (ii) Periodically invoke an algorithm that searches for a cycle in the graph.

- (iii) An algorithm to detect a cycle in a graph requires an order of  $n^2$  operations, where  $n$  is the number of vertices in the graph.



### Recovery Scheme

- Resource preemption
- Process termination
  - (i) Abort all deadlock processes
  - (ii) Abort one process at a time until the deadlock cycle eliminated.

# Memory Management

Memory management techniques allow several processes to share memory. When several processes are in memory they can share the CPU, thus increasing CPU utilisation.

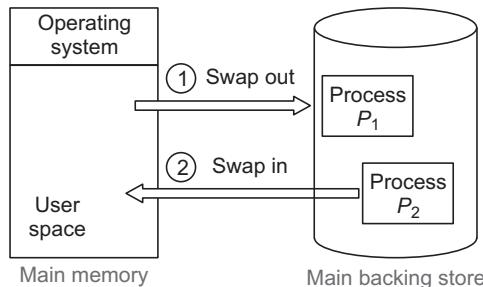
## Overlays

This techniques allow to keep in memory only those instructions and data, which are required at given time.

The other instruction and data is loaded into the memory space occupied by the previous ones when they are needed.

## Swapping

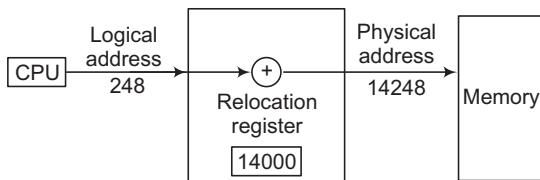
Consider an environment which supports multiprogramming using say Round Robin (RR) CPU scheduling algorithm. Then, when one process has finished executing for one time quantum, it is swapped out of memory to a backing store.



The memory manager then picks up another process from the backing store and loads it into the memory occupied by the previous process. Then, the scheduler picks up another process and allocates the CPU to it.

### Logical *versus* Physical Address Space

An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit is commonly referred to as physical address.



Schematic view of logical *versus* physical address

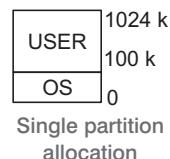
## Memory Management Techniques

The main memory must accommodate both the operating system and the various user processes. The parts of the main memory must be allocated in the most efficient way possible.

*There are two ways for memory allocation as given below*

### Single Partition Allocation

The memory is divided into two parts. One to be used by OS and the other one is for user programs. The OS code and data is protected from being modified by user programs using a base register.



### Multiple Partition Allocation

*The multiple partition allocation may be further classified as*

#### Fixed Partition Scheme

Memory is divided into a number of fixed size partitions. Then, each partition holds one process. This scheme supports multiprogramming as a number of processes may now be brought into memory and the CPU can be switched from one process to another.

When a process arrives for execution, it is put into the input queue of the smallest partition, which is large enough to hold it.

## Variable Partition Scheme

A block of available memory is designated as a hole at any time, a set of holes exists, which consists of holes of various sizes scattered throughout the memory.

When a process arrives and needs memory, this set of holes is searched for a hole which is large enough to hold the process. If the hole is too large, it is split into two parts. The unused part is added to the set of holes. All holes which are adjacent to each other are merged.

### Searching for Hole in the Set

*There are several algorithm which are used to search for the hole in the set.*

- **First fit** This allocates the first hole in the set, which is big enough to hold the process.
- **Next fit** This works like the first fit algorithm except that it keeps track of the position of the hole found in the set. The next time it is called it starts searching from that position.
- **Best fit** This allocates the smallest hole, which is large enough to hold the process.
- **Worst fit** This algorithm simply allocates the largest hole.

## Disadvantages of Memory Management Techniques

*The above schemes cause external and internal fragmentations of the memory as given below*

**External Fragmentation** When there is enough total memory in the system to satisfy the requirements of a process but the memory is not contiguous.

**Internal Fragmentation** The memory wasted inside the allocated blocks of memory called internal fragmentation.

e.g., consider a process requiring 150 k, thus if a hold of size 170 k is allocated to it the remaining 20 k is wasted.

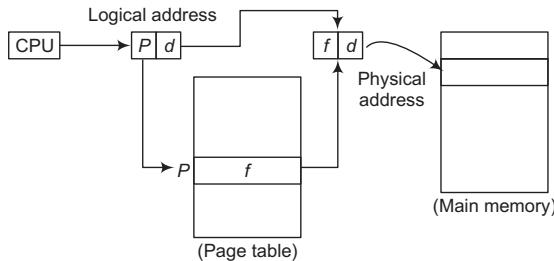
### Compaction

This is strategy to solve the problem of external fragmentation. All free memory is placed together by moving processes to new locations.

## Paging

It is a memory management technique, which allows the memory to be allocated to the process wherever it is available. Physical memory is divided into fixed size blocks called **frames**. Logical memory is broken into blocks of same size called **pages**. The backing store is also divided into same size blocks.

When a process is to be executed its pages are loaded into available page frames. A frame is a collection of contiguous pages. Every logical address generated by the CPU is divided into two parts. The page number ( $P$ ) and the page offset ( $d$ ). The page number is used as an index into a page table.



A paging block diagram

Each entry in the page table contains the base address of the page in physical memory ( $f$ ). The base address of the  $P$ th entry is then combined with the offset ( $d$ ) to give the actual address in memory.

## Key Points

- The size of a page is typically a power of 2. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset, particularly easy.
- If the size of logical address space is  $2^m$  and a page size is  $2^n$  addressing units (bytes or words), then the high order ( $m-n$ ) bits of a logical address designates the page number and the  $n$  low-order bits designates the page offset.

## Advantages of Paging

*The following are the advantages of paging*

It allows the memory of a process to be non-contiguous. This also solves the problem of fitting or storing memory blocks of varying sizes in secondary memory (such as backing store).

Paging also allows sharing of common code. Two or more processes are allowed to execute the same code at the same time.

## Virtual Memory

Separation of user logical memory from physical memory. It is a technique to run process size more than main memory. Virtual memory is a memory management scheme which allows the execution of a partially loaded process.

## Advantages of Virtual Memory

The advantages of virtual memory can be given as

- Logical address space can therefore be much larger than physical address space.
- Allows address spaces to be shared by several processes.
- Less I/O is required to load or swap a process in memory, so each user can run faster.

## Segmentation

- Logical address is divided into blocks called segment i.e., logical address space is a collection of segments. Each segment has a name and length.
- Logical address consists of two things  
 $\langle \text{segment\_number}, \text{offset} \rangle$
- Segmentation is a memory-management scheme that supports the following user view of memory. All the location within a segment are placed in contiguous location in primary storage.

## Demand Paging

Virtual memory can be implemented by using either of the below.

- (i) Demand paging    (ii) Demand segmentation

Demand paging is combination of swapping and paging. With demand paged virtual memory, pages are only loaded when they are demanded during program execution. As long as we have no page faults, the effective access time is equal to the memory access time. If however, a page fault occurs, we must first read the relevant page from disk and then access the desired word.

### Effective Access Time

$$\text{Effective access time} = [(1 - p) \times \text{ma} + (p \times \text{page fault time})]$$

Here, ma = Memory access time

$p$  = The probability of page fault ( $0 \leq p \leq 1$ )

If  $p = 0$ , it means no page faults.

If  $p = 1$ , every reference is a fault.

We expect  $p$  to be close to zero that is there will be only few page faults.

## Page Replacement

In a multiprogramming environment, the following scenario often results. While execution of a process, page fault occurs and there are no free frames on the free frame list. This is called over allocation of memory and results due to increase in the degree of multiprogramming. Page replacement is a technique to solve this problem.

### Key Points

- ♦ Paging and segmentation implement non-contiguous allocation of memory.
- ♦ Logical memory is divided into blocks called pages physical memory is divided into fixed sized blocks known as frames.

### Concept

If no free frame is available, a frame is found which is not in use. The contents of the frame are written onto a backing store and the page tables are modified to indicate that the page is no longer in memory. Thus, the frame can be used to hold the page for which the page fault was generated.

One bit is associated with each frame. If the bit is 1, this implies that the contents of the page was modified this is called the dirty bit. If the dirty bit is 0, the content of the frame need not be written to the backing store.

## Page Replacement Algorithms

In a computer operating system that uses paging for virtual memory management, page replacement algorithms decide which memory pages to page out when a page of memory needs to be allocated.

### First In First Out (FIFO)

A FIFO replacement algorithm associates with each page, the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

- It is not strictly necessary to record the time, when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory we insert it at the tail of the queue.

e.g., Consider the reference string

7, 0, 1, 2, 0, 3, 0 4, 2, 3, 0 3, 2, 1, 2, 0, 1, 7, 0, 1

Initially our 3 frames (frame size = 3) are empty. The first 3 references (7, 0, 1) cause page faults and are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since, 0 is the next reference and 0 is already in memory, we have no fault for this reference. The first reference to 3 results in replacement of page 0, since it is now first in line. Because of this replacement, the next reference to 0, will fault. Page 1 is then replaced by page 0. This process continues as shown in below figure.

Page faults occurs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2		2	2	4	4	4	0		0	0			7	7	7	
Page frames		0	0	0		3	3	3	2	2	2		1	1			1	0	0	
		1	1		1	0	0	0	3	3		3	2			2	2	1		

Total number of page faults in the above example = 15

### Optimal Page Replacement

It has the lowest page fault rate of all algorithms and will never suffer from Belady's anomaly. This algorithm replaces the page that will not be used for the longest period of time. This algorithm gives a lowest page fault rate. It is very difficult to implement because, if requires future knowledge about the usage of the page.

e.g., Using the previous reference string, the optimal page replacement algorithm would field 9 page faults. The first 3 references cause faults that fill the 3 empty frames. The reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page 0 will be used at 5 and page 1 at 14. The reference page 3 replaces page 1, as page 1 will be the last of the 3 pages in memory to be referenced again.

Page faults occurs	✓	✓	✓	✓	✓	✓	✓	✓	✓											
Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	2		2	2		2		2		2		2		7			
Page frames		0	0	0		0	4		0		0		0		0		0			
		1	1		3	3		3		3		1		1		1				

### Key Points

- ♦ Beladys' Anomaly For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.
- ♦ This phenomenon known as Belady's anomaly.

### Least Recently Used (LRU) Page Replacement

In this, we use the recent past as an approximation of the near future, then we can replace the page that has not been user for the longest period of time.

e.g., We are considering the previous reference string (used in optimal page replacement algorithm). In LRU algorithm, the first 5 faults are the same as those for optimal replacement. When the reference to page 4 occurs, however LRU replacement sees that of the 3 frames in memory, page 2 was used least recently, Thus, the LRU algorithm replaces page 2, not knowing that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3, since it is how the least recently used of the three pages in memory. Consider frame size = 3 for example given below.

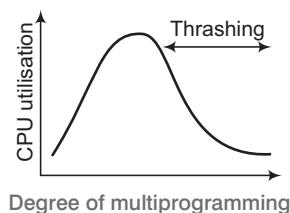
Page faults occurs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2
	7	7	7	2		2		4	4	4	0		1
Page frames	0	0	0		0		0	0	3	3		3	0
		1	1	3		3	2	2	2		2	2	7

### Frame Allocation

The maximum number of frames that can be allocated to a process depend on the size of the physical memory (i.e., total number of frames available). The minimum number of frames which can be allocated to a process depend on the instruction set architecture.

### Thrashing

- A process is said to be thrashing when it is spending more time in paging (i.e., it is generating a lot of page faults) than executing.
- Thrashing causes low CPU utilisation. The processes which are thrashing queue up for the paging device.
- The CPU schedule sees the empty ready queue and tries to increase the degree of multiprogramming by introducing more processes in the system. These new processes cause more page faults and increase the length of the queue for the paging device.



# **File Management**

Logically related data items on the secondary storage are usually organised into named collections called files. A file may contain a report, an executable program as a set of commands to the operating system. A file often appears to the users as a linear array of characters or record structures.

*The file system consists of two parts*

- (i) A collection of files
- (ii) A directory structure

The file management system can be implemented as one or more layers of the operating system.

*The common responsibilities of the file management system includes the following*

- Mapping of access requests from logical to physical file address space.
- Transmission of file elements between main and secondary storage.
- Management of the secondary storage such as keeping track of the status, allocation and deallocation of space.
- Support for protection and sharing of files and the recovery and possible restoration of the files after system crashes.

## **File Attributes**

Each file is referred to by its name. The file is named for the convenience of the users and when a file is named, it becomes independent of the user and the process. *Below are file attributes*

- Name
- Location
- Protection
- Type
- Size
- Time and date

## **Disk Scheduling**

One of the responsibilities of the OS is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.

*Access time has two major components*

- Seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.

- The rotational latency is the additional time for the disk to rotate the desired sector to the disk head. It is not fixed, so we can take average value.

$$\text{Rotational latency} = \frac{\text{One complete revolution time}}{2}$$

Disk bandwidth is the total number of bytes transferred, divided by the total time between the first for service and the completion of last transfer.

### File Operations

The OS provides system call to create, read , write, delete and truncate files.

*The common file operations are as follows*

- File Creation** For a file to be created firstly, a space must be found for a file, in the file system and secondarily a new entry must be made in the directory for the new file.
- Writing a File** In order to write a file, a system call is made, which specify the file name and the information to be written in file.
- Reading a File** To read from a file, we use a system call that specifies the name of the file and where in the memory. The next block of the file should be put and again for the associated directory entry.
- Repositioning within a File** Repositioning within a file doesn't need to invoke any actual I/O. This is also known as file seek.
- File Deletion** The directory for the named file is searched in order to delete a file and when the associated directory entry is found, all the file space is released and the directory entry is removed.
- Truncating a File** Truncating a file is used when the user wants the attributes of the file to remain the same but wants to erase the contents of the file. There is no use of deleting a file and recreating it rather this function allow all attributes to remain unchanged but for the file to reset the length zero.

### FCFS Scheduling

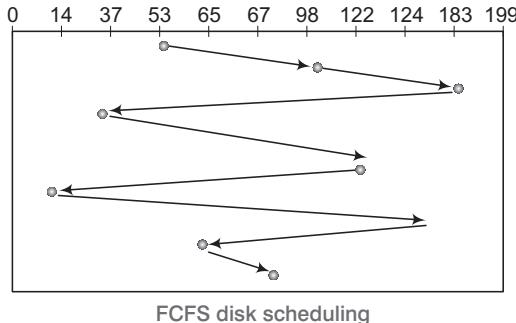
This is also known as First In First Out (FIFO) simply queues processes in the order that they arrive in the ready queue.

*The following features which FIFO scheduling have.*

- First come first served scheduling.
- Processes request sequentially.
- Fair to all processes, but it generally does not provide the fastest service.
- Consider a disk queue with requests for I/O to blocks on cylinder i.e.,

$$\text{Queue} = 98, 183, 37, 122, 14, 124, 65, 67$$

Head starts at 53



Total head movement

$$\begin{aligned}
 &= [(98 - 53) + (183 - 98) + (183 - 37) + (122 - 37) + \\
 &\quad (14 - 122) + (65 - 124) + (67 - 65)] \\
 &= 604
 \end{aligned}$$

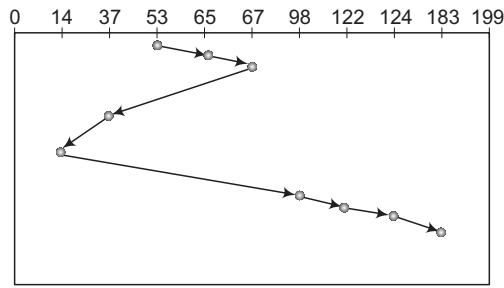
The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinder 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially and performance could be thereby improved.

### **Shortest Seek Time First (SSTF) Scheduling**

It selects the request with the minimum seek time from the current head position. SSTF scheduling is a form of SJF scheduling may cause starvation of some requests. It is not an optimal algorithm but its improvement over FCFS.

Queue = 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53.



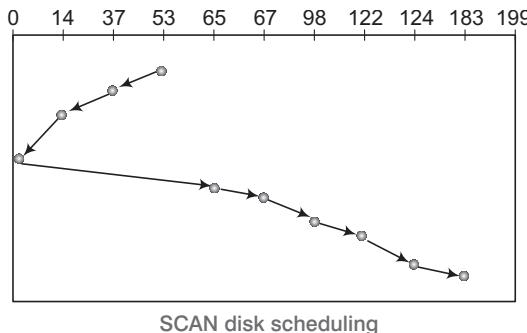
- This scheduling method results in a total head movement of only 236 cylinders little more than one-third of the distance needed for FCFS scheduling of this request time. Clearly this algorithm gives a substantial improvement in performance.

### **SCAN Scheduling**

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder until it gets to the other end of the disk. At the other end, the direction of head movement is reversed and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the request going up and then reversing to service requests the other way.

Queue = 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53.

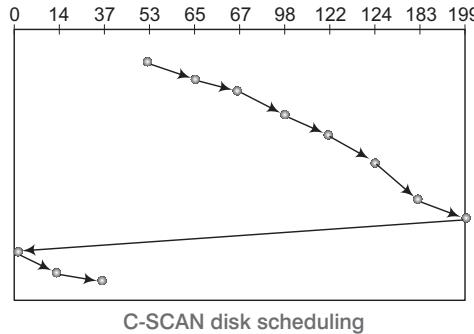


### **C-SCAN Scheduling**

Circular SCAN is a variant of SCAN, which is designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however it immediately returns to the beginning of the disk without servicing any requests on the return trip. The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

Queue = 98, 183, 37, 122, 14, 124, 65, 67

Head starts at 53.



## Key Points

- ♦ **Bootstrap** For a computer to start running for instances when it is powered up or rebooted, it needs to have an initial program to run. This initial program is known as bootstrap program.
- ♦ Bootstrap program is stored in ROM or EEPROM which is known as a firmware.
- ♦ **System Calls** System calls provide an interface to the service made available by operating system.
- ♦ **Single Processor System** In a single processor system, there is one main CPU capable of executing a general purpose instruction set, including instructions from user processes.
- ♦ **Multi Processor System** It is also known as a parallel system or tightly coupled system.
- ♦ **Cluster System** In Cluster system, computers share common storage and are closely linked via a LAN.

## Different Types of Scheduling Algorithm

Scheduling Algorithm	CPU Overhead	Throughput	Turn Around Time	Response Time
First In First Out	Low	Low	High	Low
Shortest job first	Medium	High	Medium	Medium
Priority based Scheduling	Medium	Low	High	High
Priority Based Scheduling	High	Medium	Medium	High
Multilevel Queue Scheduling	High	High	Medium	Medium

# 6

# Computer Network

---

## Computer Network

A collection of computers (or computer like devices) that are able to communicate with each other through some medium, using hardware and software. Two computers (or computer like devices) are said to be connected, if they are able to exchange information or able to communicate.

*Every network includes elements to enable data transfer or sharing are given below.*

- Atleast two computers (or computer like devices)
- Network interfaces
- A connection medium
- Operating system, strategies, algorithms and protocols

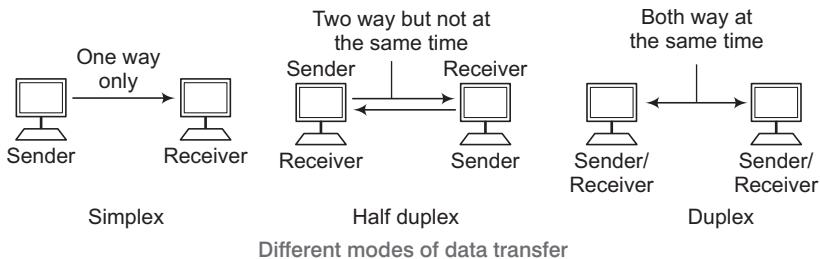
## Data Transfer Modes

*There are mainly three modes of data transfer*

**Simplex** Data transfer only in one direction e.g., radio broadcasting.

**Half Duplex** Data transfer in both direction, but not simultaneously i.e., in one direction at a time e.g., talk back radio, CB radio (citizen band).

**Full Duplex or Duplex** Data transfer in both directions, simultaneously e.g., telephone.



## Elements used in Computer Networks for Communication

Some basic elements which are using in communication systems of a computer network are given below

- **Data Source** Provides the data to transmit.
- **Sender (Transmitter)** Converts data to signals for transmission.
- **Data Transmission System** Transmits the data i.e., converted in signals.
- **Receiver** Converts received signals to data.
- **Destination** Receives and uses incoming data.
- **Node** A device with independent communication ability and unique network address.
- **Protocol** A formal description, comprising rules and conventions defines the method of communication between networking devices.

## Methods of Message Delivery (i.e., Casting)

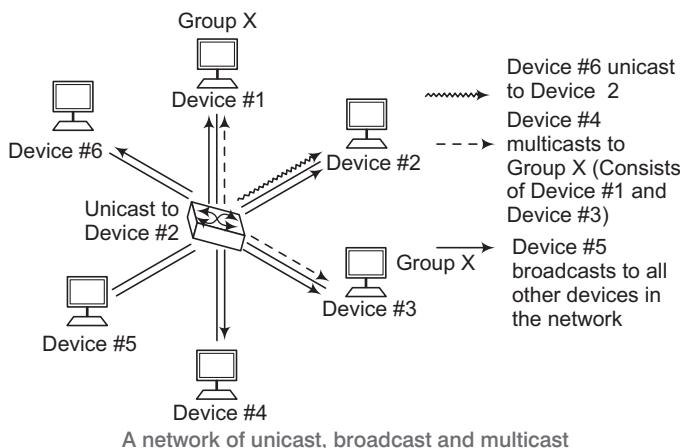
A message can be delivered in the following ways

**Unicast** One device sends message to the other to its address.

**Broadcast** One device sends message to all other devices on the network.

The message is sent to an address reserved for this goal.

**Multicast** One device sends message to a certain group of devices on the network.



## Types of Networks

*The types of network based on their coverage areas are as given below*

### LAN (Local Area Network)

LAN is privately owned network within a single building or campus. LANs can be small, linking as few as three computers, but often link hundreds of computers used by thousands of people (like in some IT office, etc.)

An arbitration mechanism is installed in LAN to decide, which machine will use the access, when more than one machines are requesting for communication. LAN are used for connecting personal computers, workstations, routers and other devices.

*Their main characteristics are given below*

- **Topology** The geometrical arrangement of the computers or nodes.
- **Protocols** How they communicate.
- **Medium** Through which medium.

### MAN (Metropolitan Area Network)

A MAN covers a city. An example of MAN is cable television network in city.

### WAN (Wide Area Network)

A wide area network or WAN spans a large geographical area often a country.

**Internet** It is also known as network of networks. The Internet is a system of linked networks that are world wide in scope and facilitate data communication services such as remote login, file transfer, electronic mail, world wide web and newsgroups etc.

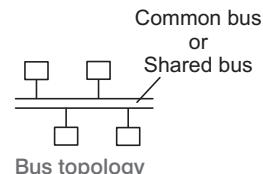
## Network Topology

Network topology is the arrangement of the various elements of a computer or biological network. Essentially it is the topological structure of a network, and may be depicted physically or logically. Physical topology refers to the placement of the network's various components, inducing device location and cable installation, while logical topology shows how data flows within a network, regardless of its physical design.

*The common network topologies include the following sections*

### Bus Topology

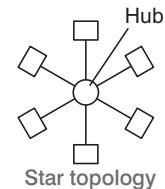
In bus topology, each node (computer server, other computer like devices) is directly connected to a common cable.



**Note** The drawback of this topology is that if the network cable breaks, the entire network will be down.

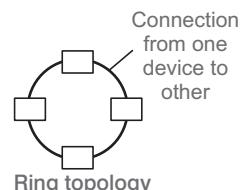
### Star Topology

In this topology, each node has a dedicated set of wires connecting it to a central network hub. Since, all traffic passes through the hub, it becomes a central point for isolating network problems and gathering network statistics.



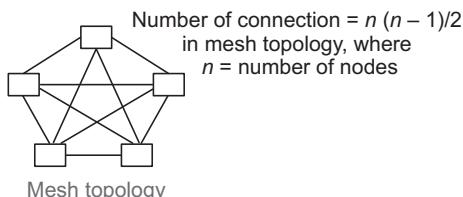
### Ring Topology

A ring topology features a logically closed loop. Data packets travel in a single direction around the ring from one network device to the next. Each network device acts as a repeater to keep the signal strong enough as it travels.



### Mesh Topology

In mesh topology, each system is connected to all other systems in the network.

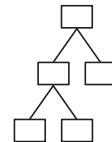


## Key Points

- ♦ In bus topology at the first, the message will go through the bus then one user can communicate with other.
- ♦ In star topology, first the message will go to the hub then that message will go to other user.
- ♦ In ring topology, user can communicate as randomly.
- ♦ In mesh topology, any user can directly communicate with other users.

## Tree Topology

In this type of network topology, in which a central root is connected to two or more nodes that are one level lower in hierarchy.



Tree topology

## Hardware/Networking Devices

Networking hardware may also be known as network equipment. computer networking devices.

*Some important networking devices used in the medium of communication are given below*

### Network Interface Card (NIC)

NIC provides a physical connection between the networking cable and the computer's internal bus.

NICs comes in three basic varieties 8 bit, 16 bit and 32 bit. The larger number of bits that can be transferred to NIC, the faster the NIC can transfer data to network cable.

### Repeater

Repeaters are used to connect together two Ethernet segments of any media type. In larger designs, signal quality begins to deteriorate as segments exceed their maximum length. We also know that signal transmission is always attached with energy loss. So, a periodic refreshing of the signals is required.

### Hubs

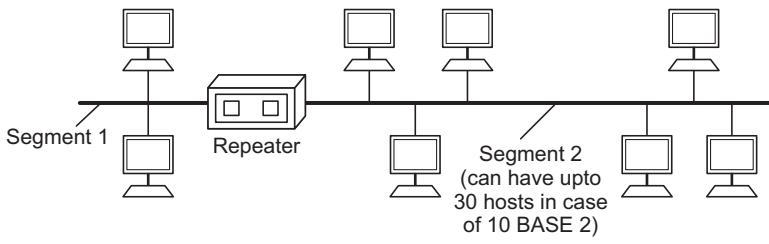
Hubs are actually a multiport repeaters. A hub takes any incoming signal and repeats it out all ports.

## Key Points

- Repeaters provide the signal amplification required to allow a segment (cable) to be extended a greater distance.
- A typical repeater has just 2 ports, a hub generally has from 4 to 24 ports.

**Different Types of Ethernet Network Table**

Ethernet Network Type	Maximum Nodes/Segment	Maximum Distance/Segment (Cable)
10 BASE-T	2	100 m
10 BASE 2	30	185 m
10 BASE 5	100	500 m
10 BASE-FL	2	2000 m



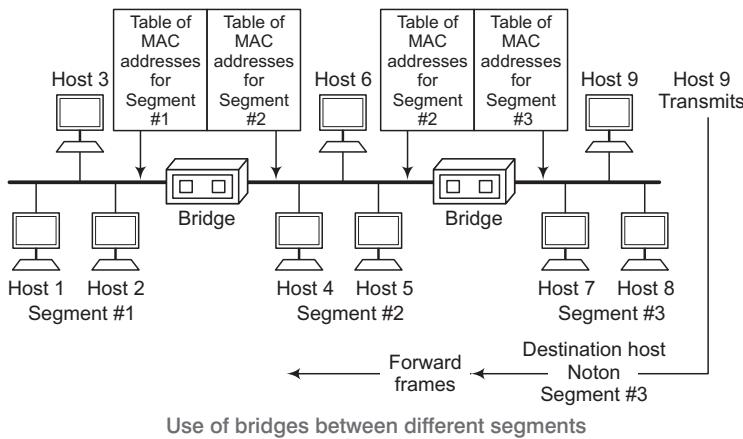
Before repeater

Digital signal after repeater

Use of repeaters between segments

## Bridges

When the size of the LAN is difficult to manage, it is necessary to breakup the network. The function of the bridge is to connect separate networks together. Bridges do not forward bad or misaligned packets.



## Key Points

- ♦ Bridges map the addresses of the nodes residing on each network segment and allow only necessary traffic to pass through the bridge.
- ♦ When a packet is received by the bridge, the bridge determines the destination and source segments.
- ♦ If the segments are the same, the packet is filtered, if the segments are different, then the packet is forwarded to the correct segments.

## Switch

Switches are an expansion of the concept of bridging. LAN switches can link 4,6,10 or more networks together.

Cut through switches examine the packet destination address, only before forwarding it onto its destination segment, while a store and-forward switch accepts and analyzes the entire packet before forwarding it to its destination. It takes more time to examine the entire packet, but it allows to catch certain packet errors and keep them from propagating through the network.

## Routers

Router forwards packets from one LAN (or WAN) network to another. It is also used at the edges of the networks to connect to the Internet.

## Gateway

Gateway acts like an entrance between two different networks. Gateway in organisations, is the computer that routes the traffic from a work station to the outside network that is serving web pages.

ISP (Internet Service Provider) is the gateway for Internet service at homes.

### Some IEEE Standards for Networking

Standard	Related to/Used for
IEEE 802.2	Logical Link Control
IEEE 802.3	Ethernet
IEEE 802.5	Token Ring
IEEE 802.11	Wireless LAN
IEEE 802.15	Bluetooth
IEEE 802.16	Wireless MAN

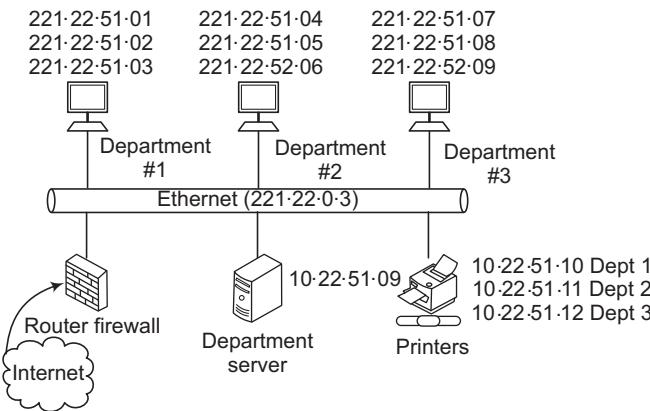
### Key Points

- ♦ Switches have 2 basic architectures: cut through and store-and-forward.
- ♦ Switching is a technology that alleviates congestion in Ethernet LANs by reducing traffic and increasing bandwidth.
- ♦ It operates in the third layer and forwards packets based on network addresses using routing tables and protocols.
- ♦ Gateway's main concern is routing traffic from a work station to the outside network.

## Ethernet

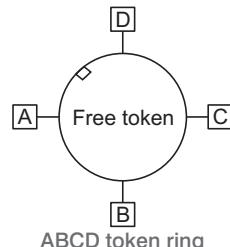
It is basically a LAN technology which strikes a good balance between speed, cost and easy of installation. The Institute for Electrical and Electronics Engineering (IEEE) defines the Ethernet Standard as IEEE Standard 802.3. This standard defines rule for configuring an Ethernet network as well as specifying how elements in an Ethernet network interact with each other.

Ethernet uses Carrier Sense Multiple Access/Collision Detect (CSMA/CD) technology, broadcasting each frame onto the physical medium (wire fibre and so on). All stations attached to the Ethernet listen to the line for traffic and the station with the matching destination MAC address accepts the frame.



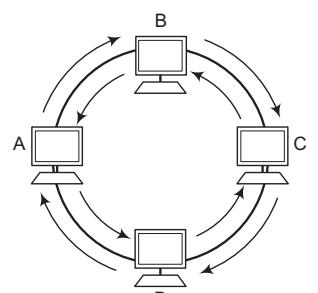
### Token Ring

- It is another form of network configuration, which differs from Ethernet in that all messages are transferred in a unidirectional manner along the ring at all times.
- Medium Access Control (MAC) is provided by a small frame, the token that circulates around the ring when all stations are idle. Only the station (node) possessing the token is allowed to transmit at any given time. Sender looks for free token and changes free token to busy token and appends data.
- This technology can connect upto 255 nodes in a physical star or ring connection that can sustain 4 or 16 Mbps.



### FDDI (Fibre Distributed Data Interface)

This is a form of network configuration, which uses a ring topology of multimedia or single mode optical fibre transmission links operating at 100 Mbps to span upto 200 km and permits upto 500 stations. It employs dual counter rotating rings. Here, 16 and 48 bit addresses are allowed. In FDDI, token is absorbed by station and released as soon as it completes the frame transmission.



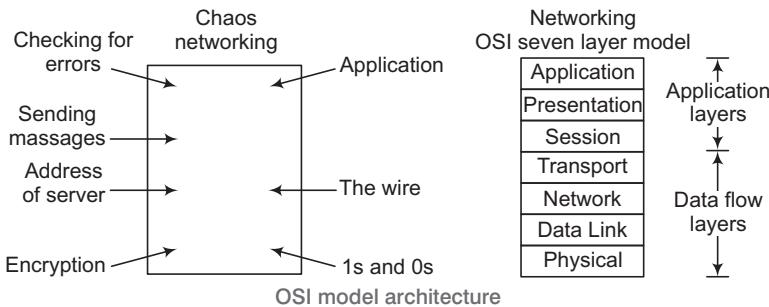
FDDI between users A,B,C and D

# OSI Model

The Open System Interconnection (OSI) model is a reference tool for understanding data communication between any two networked system. It divides the communication processes into 7 layers. Each layer performs specific functions to support the layers above it and uses services of the layers below it. Each layer represents a different level of abstraction and layers boundaries are well defined.

## Benefits of OSI Model

It helps users understand the big picture of networking. It helps users understand how hardware and software elements function together. OSI model makes troubleshooting easier by separating networks into manageable pieces. The OSI model provides a common language to explain components and their functionality.



**Note** Without the OSI model, networks would be very difficult to understand and implement.

## Physical Layer

The physical layer coordinates the functions required to transmit a bit stream over a physical medium. It deals with the mechanical and electrical specifications of interface and transmission medium. It also defines the procedures and functions that physical devices and interfaces have to perform for transmission to occur.

## Functions of Physical Layer

*There are several functions physical layer*

- Physical layer defines characteristics of the interface between the devices and the transmission medium.
- It defines the type of transmission medium.
- It defines the transmission rate *i.e.*, the number of bits sent each second.
- It performs synchronisation of sender and receiver clocks.
- It is concerned with the connection of devices to the medium.

(i) **Point-to-point configuration** Two devices are connected together through dedicated link.

(ii) **Multipoint configuration** A link is shared between several devices

- It is concerned with the physical topology.
- It defines the direction of transmission *i.e.*, transmission mode (simplex, half duplex or duplex).
- It transmits bit stream over the communication channel.

**Hardware Used** Repeater and Hub.

**Data Unit** Bit stream

## Data Link Layer

The data link layer transforms the physical layer, a raw transmission facility, to a reliable link and is responsible for Node-to-Node delivery. It makes the physical layer appear error free to the upper layer (*i.e.*, network layer).

### Functions of Data Link Layer

*Data link layer is responsible for*

**Framing** (*i.e.*, division of stream of bits received from network layer into manageable data units called frames/or segmentation of upper layer datagrams (also called packets) into frames).

**Flow Control** (*i.e.*, to manage communication between a high speed transmitter with the low speed receiver).

**Error Control** (*i.e.*, adding mechanism to detect and retransmit damaged or lost frames and to prevent duplication of frames. To achieve error control, a trailer is added at the end of a frame).

**Access Control** (*i.e.*, to determine which device has control over the link at any given time, if two or more devices are connected to the same link).

**Physical Addressing** (*i.e.*, adding a header to the frame to define the physical address of the sender (source address) and/or receiver (destination address) of the frame.)

**Hardware Used** Bridges and switches.

**Data Unit** Frames

**Protocol Used** Simplex protocol, stop and wait protocol, sliding window, HDLC (High Level Data Link Control), SDLC, NDP, ISDN, ARP, PSL, OSPF, NDP.

## **Network Layer**

Network layer is responsible for source to destination delivery of a packet possibly across multiple networks (links). If the two systems are connected to the same link, there is usually no need for a network layer. However, if the two systems are attached to different networks (links) with connecting devices between networks, there is often a need of the network layer to accomplish source to destination delivery.

### **Functions of the Network Layer**

*Network layer responsibilities include*

- **Logical Addressing** The physical addressing implemented by the data link layer handles the addressing problem locally (*i.e.*, if the devices are in the same network). If packet passes the network boundary, we need another addressing system to distinguish the source and destination systems.
- **Routing** Independent networks or links are connected together with the help of routers or gateways. Routers route the packets to their final destination. Network layer is responsible for providing routing mechanism.
- **Hardware Used** Routers
- **Data Units** Packets
- **Protocols Used** IP (Internet Protocol), NAT (Network Address Translation), ARP (Address Resolution Protocol), ICMP (Internet control Message Protocol), BGP(Border Gateway Protocol), RARP (Reverse Address Resolution Protocol), DHCP (Dynamic Host Configuration Protocol), BOOTP, OSPF.

## **Transport Layer**

The transport layer is responsible for source to destination (end-to-end) delivery of the entire message. Network layer does not recognise any relationship between the packets delivered. Network layer treats each packet independently, as though each packet belonged to a separate message, whether or not it does. The transport layer ensures that the whole message arrives intact and in order.

## Key Points

- The network layer adds a header to the packet coming from the upper layer that among other things, includes the logical addresses of the sender and receiver.
- The network layer gets each packet to the correct computer, the transport layer gets the entire message to the correct process on that computer.
- These sequence numbers are used to reassemble the message correctly and to identify and replace packets that were lost in transmission.

## Functions of Transport Layer

*Responsibilities of transport layer includes*

- **Service Point Addressing** The transport layer header must include a type of address called service point address (or part address).
- **Segmentation and Reassembly** A message is divided into transmittable segments, each segment containing a sequence number.
- **Flow Control** Flow control at this layer is performed end to end rather than across a single link.
- **Error Control** This layer performs an end to end error control by ensuring that the entire message at the receiving transport layer without error (damage, loss or duplication). Error correction is usually achieved through retransmission.
- **Connection Control** Transport layer can deliver the segments using either connection oriented or connectionless approach.

**Hardware Used** Transport Gateway

**Data Unit Segments**

**Protocol Used** TCP (Transmission Control Protocol) for connection oriented approach and UDP (User Datagram Protocol) for connectionless approach.

## Session Layer

The session layer is the network dialog controller. It establishes, maintains and synchronises the interaction between communicating systems. It also plays important role in keeping applications data separate.

## Functions of Session Layer

*Specific responsibilities of the session layer include the following*

**Dialog Control** Session layer allows the communication between two processes to take place either in half duplex or full duplex. It allows applications functioning on devices to establish, manage and terminate a dialog through a network.

**Synchronization** The session layer allows a process to add check points (synchronization points) into a stream of data. For example, if a system is sending a file of 2000 pages, we can insert check points after every 100 pages to ensure that each 100 page unit is received and acknowledged independently. In this case, if a crash happens during the transmission of page 523, retransmission begins at page 501, pages 1 to 500 need not to be retransmitted.

### Unit Used in Session Layer

- **Data Unit** Data
- **Protocol Used** ADSP, ASP, ISO-SP, L2TP, F2F, PAP, PPTP, RPC, SMPP, SDP, ZIP, RTCP.

## Presentation Layer

This layer is responsible for how an application formats data to be sent out onto the network. This layer basically allows an application to read (or understand) the message.

### Functions of Presentation Layer

*Specific responsibilities of this layer include the following*

**Translation** Different systems use different encoding system, so the presentation layer provides interoperability between these different encoding methods. This layer at the sender end changes the information from sender dependent format into a common format. The presentation layer at receiver end changes the common format into its receiver dependent format.

**Encryption and Decryption** This layer provides encryption and decryption mechanism to assure privacy to carry sensitive information. Encryption means sender transforms the original information to another form and at the receiver end, decryption mechanism reverses the new form of data into its original form.

**Compression** This layer uses compression mechanism to reduce the number of bits to be transmitted. Data compression becomes important in the transmission of multimedia such as text, audio and video.

### Units used in Presentation Layer

- **Data Unit** Data
- **Protocol Used** AFP, ASCII, EBCDIC, ICA, LPP, NCP, NDR, XDR, X.25 PAP.

## Application Layer

This layer enables the user, whether human or software, to access the network. It provides user interfaces and support for services such as electronic mail, remote file access and transfer shared database management and other types of distributed information services.

### Functions of Application Layer

*Specific services provided by the application layer include the following*

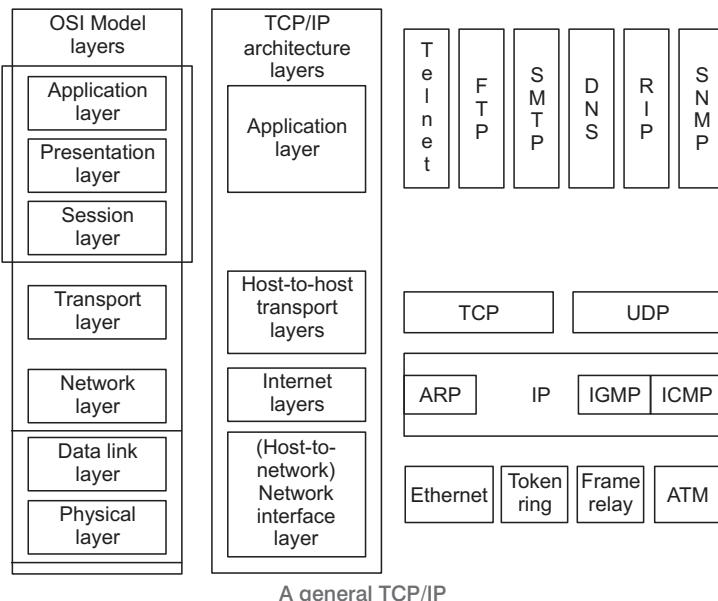
- **Network Virtual Terminal** A network virtual terminal is a software version of a physical terminal and allows a user to logon to a remote host. To do so, the application creates a software emulation of a terminal at the remote host.
- **File Transfer, Access and Management** This application allows a user to access files, retrieve files, manage files or control files in a remote computer.
- **Mail Services** Electronic messaging (*i.e.*, e-mail storage and forwarding) is provided by this application.
- **Directory Services** This application provides distributed database sources and access for global information about various objects and services.

### Units used in Application Layer

- **Hardware Used** Application Gateway
- **Protocol Used** HTTP, SMTP, POP3, FTP, Telnet etc.
- **Data Unit Data**

# TCP/IP Protocol Suite

The TCP/IP protocol suite used in the Internet, was developed prior to the OSI model. Therefore , the layers in the TCP/IP protocol suite do not exactly match those in the OSI model. The original TCP/IP protocol suite was defined as having four layer.



As we can see in the above diagram TCP/IP (Transmission Control Protocol /Internetworking Protocol) model contains four layers. The first three layers of TCP/IP model (Network Interface layer, Internet layer and Transport layer) provide physical standards, network interface, Internetworking and transport functions that corresponds to first four layers of the OSI model.

The three top most layer in the OSI model, however are represented in TCP /IP by a single layer called the **Application layer**.

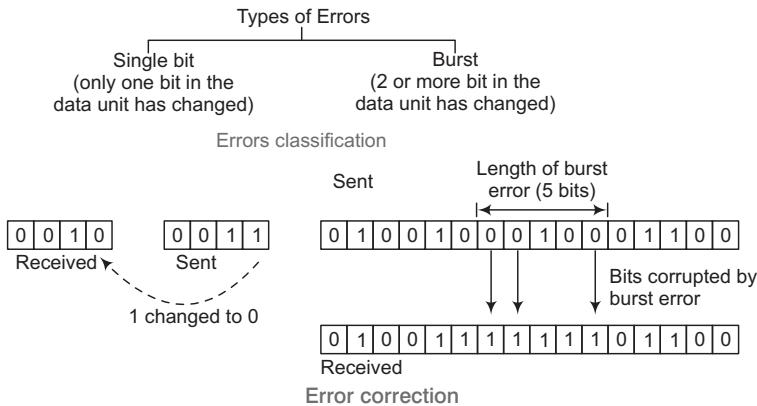
**Transport layer** is designed to allow peer entities on the source and destination, host to carry on a conversation.

**Internet layer** permits the host to inject packets into any network and let them travel independently to the destination.

**The data link layer** is the networking scope of the local network connection to which host is attached.

## Error Control (Detection and Correction)

Many factors including line noise can alter or wipe out one or more bits of a given data unit.

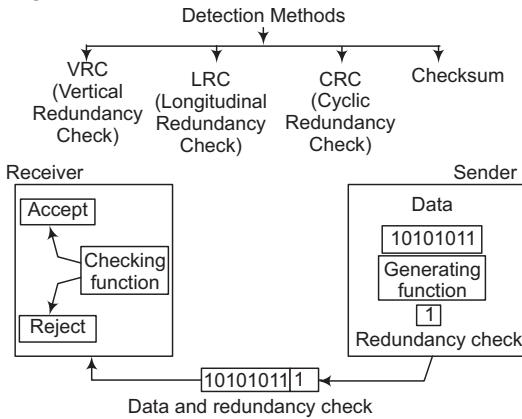


## Key Points

- Reliable systems must have mechanism for detecting and correcting such errors.
- Error detection and correction are implemented either at the data link layer or the transport layer of the OSI model.

## Error Detection

Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

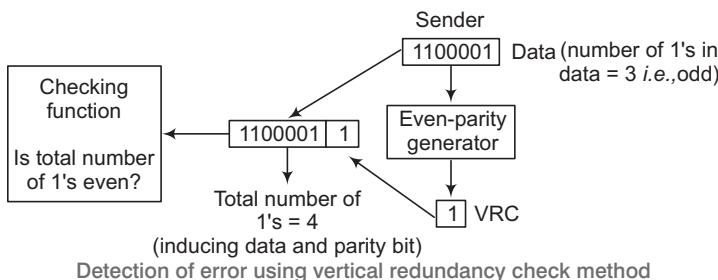


How error detection works with redundancy concept

**Note** Checking function performs the action that the received bit stream passes the checking criteria, the data portion of the data unit is accepted else rejected.

### Vertical Redundancy Check (VRC)

In this technique, a redundant bit, called parity bit, is appended to every data unit, so that the total number of 1's in the unit (including the parity bit) becomes even. If number of 1's are already even in data, then parity bit will be 0.



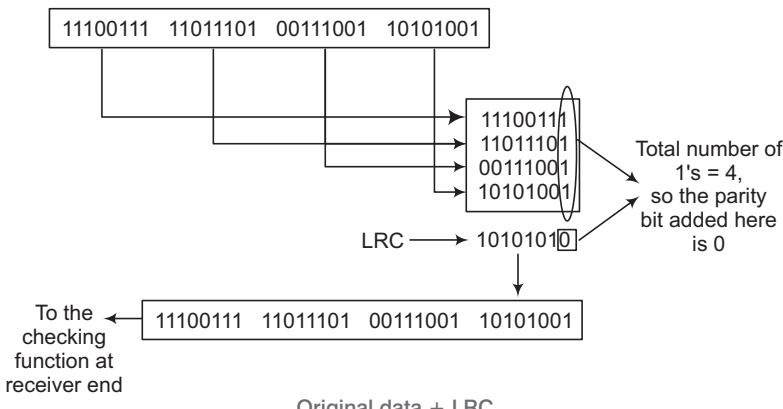
Detection of error using vertical redundancy check method

Some systems may use odd parity checking, where the number of 1's should be odd. The principle is the same, the calculation is different.

### Longitudinal Redundancy Check (LRC)

In this technique, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Original data at sender end



Original data + LRC

**Note** The first parity bit in the 5th row is calculated based on all first bits. The second parity bit is calculated based on all second bits and so on.

## Checksum

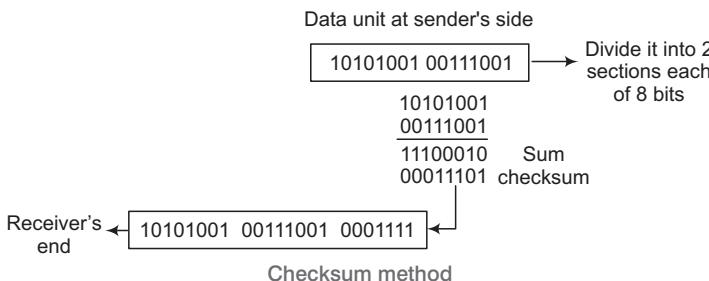
There are two algorithms involved in this process, checksum generator at sender end and checksum checker at receiver end.

*The sender follows these steps*

- The data unit is divided into  $k$  sections each of  $n$  bits.
- All sections are added together using 1's complement to get the sum.
- The sum is complemented and becomes the checksum.
- The checksum is sent with the data.

*The receiver follows these steps*

- The received unit is divided into  $k$  sections each of  $n$  bits.
- All sections are added together using 1's complement to get the sum.
- The sum is complemented.
- If the result is zero, the data are accepted, otherwise they are rejected.



- At receiver end, we break this data stream into three sections each of 8 bits and perform steps according to algorithm at receiver's end.

10101001	← 1st 8 bits
00111001	← 2nd 8 bits
11100010	← 3rd 8 bits
00011101	Sum
11111111	Complement
00000000	

- All bits 0 (zero) indicates number error in data.

## Cyclic Redundancy Check (CRC)

CRC is based on binary division. A sequence of redundant bits called CRC or the CRC remainder is appended to the end of a data unit, so that the resulting data unit becomes exactly divisible by a second, predetermined binary number. At its destination, the incoming data unit is divided by the

same number. If at this step there is no remainder, the data unit is assumed to be intact and therefore is accepted.

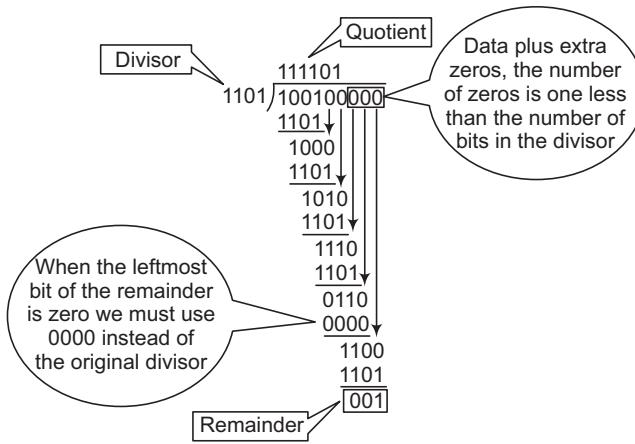
e.g., Generate CRC code for a frame 100100, using the generator.

$$G(x) = x^3 + x^2 + 1$$

$$G(x) = x^3 + x^2 + 0x^1 + x^0$$

3	2	1	0
1	1	0	1

We do not have  $x^1$  term that's why we put a 0 for that.



Method of error detection using CRC method

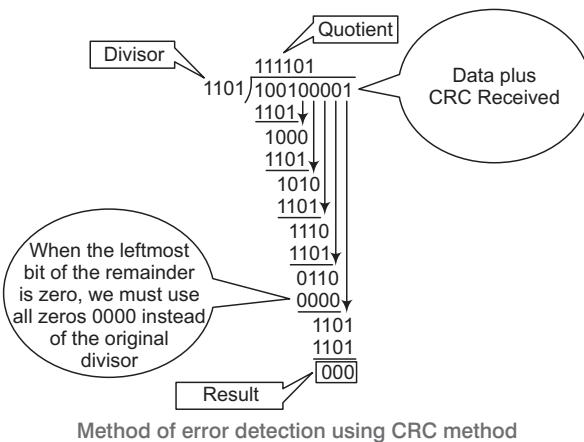
**Note** Each bit of the divisor is subtracted from the corresponding bit of the dividend without distributing the next higher bit.

In the above example, the divisor 1101, is subtracted from the first 4 bits of the dividend, 1001, yielding 001 (the leading 0 of the remainder is dropped off).

So, here we get input data stream for receiver that is original data plus remainder i.e., CRC received.

Input stream = 100100001

Now, we perform CRC checker process. Here, we again perform the same modulo-2 division. If the remainder is all zeros, the CRC is dropped and data accepted. Otherwise, the received stream of bits is discarded and data is resent.



Method of error detection using CRC method

We received remainder as all zeros, so the CRC will be dropped off and data (100100) will be accepted at receiver end. It shows that there is not error in data.

### Performance of CRC

CRC is very effective error detection method, if the divisor is chosen according to the mentioned rule i.e., *A polynomial should be selected to have atleast the following properties to be a divisor.*

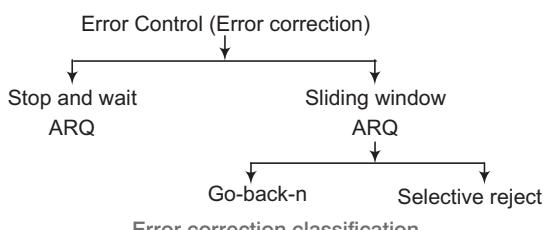
- It should not be divisible by  $x$
- It should be divisible by  $(x + 1)$

The first condition guarantees that all burst errors of a length equal to the degree of the polynomial are detected. The second condition guarantees that all burst errors affecting an odd number of bits are detected.

## Error Correction

Error correction in data link layer is implemented simply anytime, an error is detected in an exchange, a negative acknowledgement NAK is returned and the specified frames are retransmitted. This process is called Automatic Repeat Request (ARQ). *Retransmission of data happens in three cases*

Damaged frame,  
Lost frame and  
Lost acknowledgement.



Error correction classification

## Stop and Wait ARQ

Include retransmission of data in case of lost or damaged frame. For retransmission to work, four features are added to the basic flow control mechanism.

### Key Points

- ◆ The sending device keeps a copy of the last frame transmitted, until it receives an acknowledgement for that frame.
  - ◆ For identification purpose both data frames and ACK are numbered alternately 0 and 1.
  - ◆ A data 0 frame is acknowledged by an ACK 1 frame, indicating that the receiver has gotten data 0 and now expecting data 1.
- 
- If an error is discovered in a data frame, indicating that it has been corrupted in transit, a NAK frame is returned. NAK frames, which are not numbered, tell the sender to retransmit the last frame sent.
  - The sender device is equipped with a timer. If an expected acknowledgement is not received within an allotted time period, the sender assumes that the last data frame was lost in transmit and sends it again.

## Sliding Window ARQ

To cover retransmission of lost or damaged frames, *three features are added to the basic flow control mechanism of sliding window*.

- The sending device keeps copies of all transmitted frames, until they have been acknowledged.
- In addition to ACK frames, the receiver has the option of returning a NAK frame, if the data have been received damaged. NAK frame tells the sender to retransmit a damaged frame. Here, both ACK and NAK frames must be numbered for identification. ACK frames carry the number of next frame expected. NAK frames on the other hand, carry the number of the damaged frame itself. If the last ACK was numbered 3, an ACK 6 acknowledges the receipt of frames 3,4 and 5 as well. If data frames 4 and 5 are received damaged, both NAK 4 and NAK 5 must be returned.
- Like stop and wait ARQ, the sending device in sliding window ARQ is equipped with a timer to enable it to handle lost acknowledgements.

## Key Points

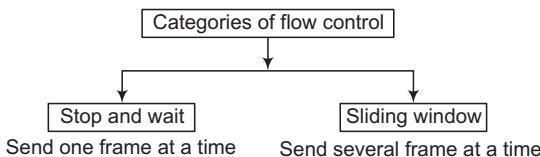
- ♦ In sliding window ARQ,  $(n - 1)$  frames (the size of the window) may be sent before an acknowledgement must be received.
- ♦ If  $(n - 1)$  frames are awaiting acknowledgement, the sender starts a timer and waits before sending any more.
- ♦ If allotted time has runout, sender assumes that frames were not received and retransmit one or all frames depending on the protocol.

**Go-back-n ARQ** In this method, if one frame is lost or damaged all frames sent, since the last frame acknowledged are retransmitted.

**Selective Reject ARQ** In this method, only specific damaged or lost frame is retransmitted. If a frame is corrupted in transmit, a NAK is returned and the frame is resent out of sequence. The receiving device must be able to sort the frames it has and insert the retransmitted frame into its proper place in the sequence.

## Flow Control

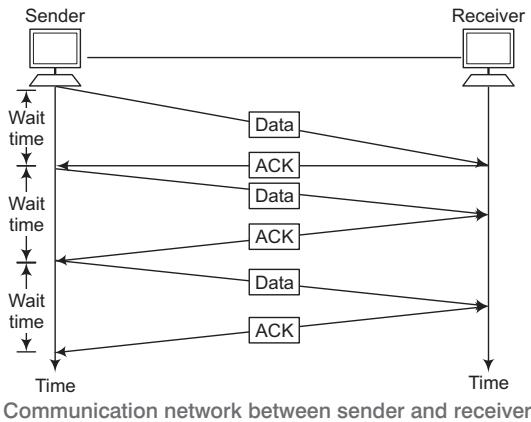
One important aspect of data link layer is flow control. Flow control refers to a set of procedures used to restrict the amount of data the sender can send before waiting for acknowledgement.



### Stop and Wait

In this method, the sender waits for an acknowledgement after every frame it sends. Only when a acknowledgment has been received is the next frame sent. This process continues until the sender transmits an End of Transmission (EOT) frame.

- We can have two ways to manage data transmission, when a fast sender wants to transmit data to a low speed receiver.
- The receiver sends information back to sender giving it permission to send more data i.e., feedback or acknowledgment based flow control.
- Limit the rate at which senders may transmit data without using feedback from receiver i.e., Rate based flow control.



### Advantages of Stop and Wait

It's simple and each frame is checked and acknowledged well.

### Disadvantages of Stop and Wait

- It is inefficient, if the distance between devices is long.
- The time spent for waiting ACKs between each frame can add significant amount to the total transmission time.

### Sliding Window

In this method, the sender can transmit several frames before needing an acknowledgement. The sliding window refers to imaginary boxes at both the sender and the receiver. This window can hold frames at either end and provides the upper limit on the number of frames that can be transmitted before requiring an acknowledgement.

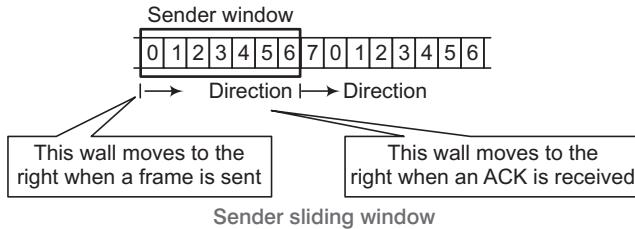
### Key Points

- The frames in the window are numbered modulo- $n$ , which means they are numbered from 0 to  $n - 1$ . For example, if  $n = 8$ , the frames are numbered 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1... so on. The size of the window is  $(n - 1)$  in this case size of window = 7.
- In other words, the window can't cover the whole module (8 frames) it covers one frame less than that is 7.

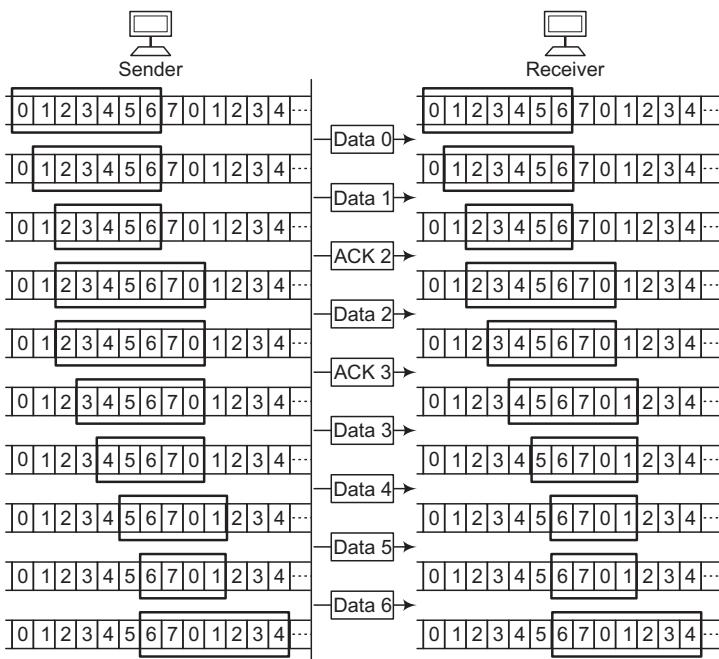
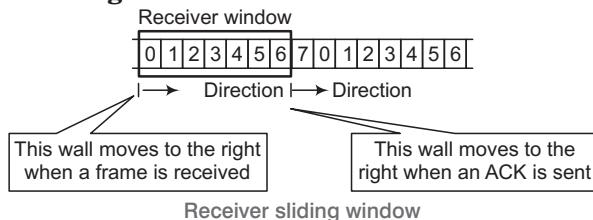
When the receiver sends an ACK, it includes the number of the next frame it expects to receive. When the receiver sends an ACK containing the number 5, it means all frames up to number 4 have been received.

### Sender Sliding Window

Size of sender window = 7, it means we are using modulo 8.



### Receiver Sliding Window



Examples of sliding window and methods of receiver sliding window

## Key Points

- ♦ As each ACK is sent out, the receiving window expands to include as many new placeholders as newly acknowledged frames. The window expands to include.
- ♦ A number of new frame spaces = The number of most recently acknowledged frame – The number of previously acknowledged frame e.g., In a seven frame window.
- ♦ If the prior ACK was for frame 2 and the current ACK is for frame 5. The window expands by three (5 – 2). If the prior ACK was for frame 3 and the current ACK is for frame 1, the window expands by six(1 + 8 – 3).

## Medium Access Control Sublayer

The protocols used to determine, who goes next on a multiaccess channel belong to a sublayer of the data link layer. This sublayer is called **medium access control sublayer**.

### Channel Allocation

**Static Channel Allocation** FDM (Frequency Division Multiplexing) and TDM (Time Division Multiplexing) is used for this purpose. Let us focus on FDM, the mean time delay  $T$  for a channel having capacity  $C$  bit/sec, with an arrival rate of  $\lambda$  frame/sec and each frame having a length drawn from an exponential probability density function with mean  $\mu$  bits/frame. Then,

$$T = \frac{1}{\mu C - \lambda}$$

If the channel is used by  $N$  independent subchannels.

Then,  $T_{FDM} = \frac{1}{\mu \left(\frac{C}{N}\right) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT$

### Dynamic Channel Allocation

In these methods, the channel is allocated to a particular system dynamically i.e., no predetermined order of senders for accessing the channel in order to send data.

## Multiple Access Protocols

*These protocols can be divided into two categories*

### ALOHA

- Pure ALOHA
- Slotted ALOHA

## CSMA

- 1-persistent CSMA
- Non-persistent CSMA
- $p$ -persistent CSMA

## ALOHA Protocols

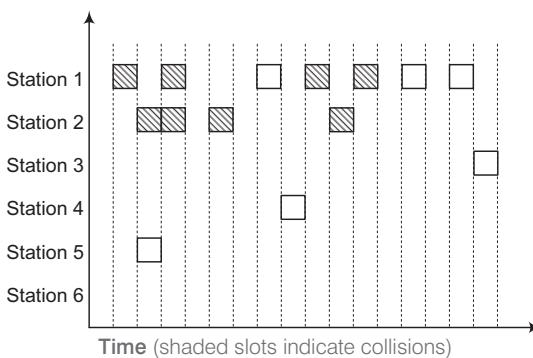
ALOHA net, known as the ALOHA system, or simply ALOHA, was a pioneering computer networking system developed at the university of Huwai. It was designed for a radio wireless LAN, but it can be used on any shared medium.

*The ALOHA protocols can be of following types*

**Pure ALOHA** In this approach, a node, which wants to transmit will go ahead and send the packet on its broadcast channel with no consideration whatsoever as to anybody else is transmitting or not. One serious drawback here is that, we are not sure about whether the data has been received properly at the receiver end. To resolve this, the pure ALOHA, when one node finishes speaking, it expects an acknowledgement in a finite amount of time otherwise, it simply retransmits the data.

## Key Points

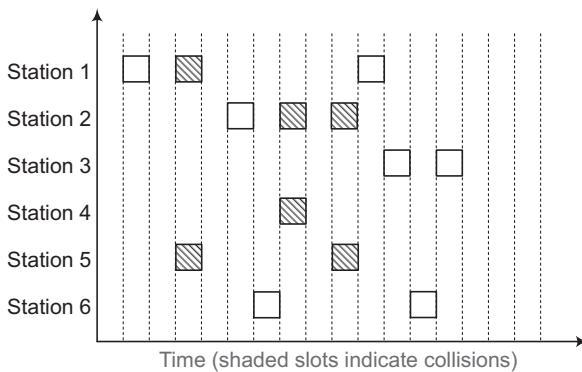
- ♦ Pure ALOHA This scheme works well in small networks where the load is not high.
- ♦ In large load intensive networks where many nodes may want to transmit at the same time, the Pure ALOHA scheme fails miserably.



**Slotted ALOHA** This is quite similar to pure ALOHA, differing only in the way transmissions take place. Instead of transmitting right at demand time, the sender waits for some time. This delay is specified as follows the timeline is divided into equal slots and then it is required that transmission should take place only at slot boundaries.

To be more precise, the slotted ALOHA makes the following assumption

- All frames consists of exactly  $L$  bits.
- Time is divided into discrete frame time slots i.e., a slot equals the time to transmit one frame.
- Nodes start to transmit frames only at the beginning of slots.
- The nodes are synchronized, so that each nodes knows when the slot begin.
- If two or more frames collide in a slot, then all the node detect collision event before the slot ends.

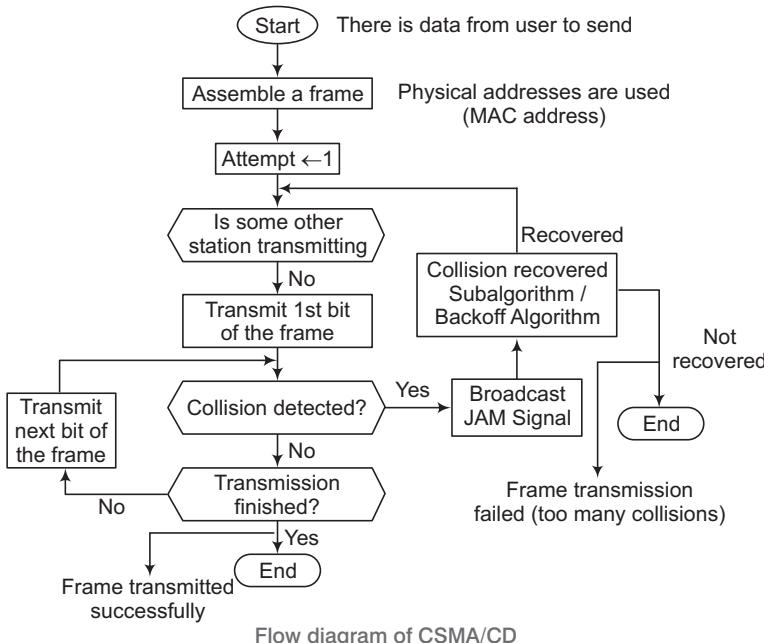


## Key Points

- ♦ Whenever multiple users have unregulated access to a single line, there is a danger of signals overlapping and destroying each other. Such overlaps, which turn the signals into unusable noise are called collisions.
- ♦ A LAN therefore needs a mechanism to coordinate traffic, minimize the number of collisions that occur and maximize the number of frames that are delivered successfully. The access mechanism used in an Ethernet is called CSMA/CD, standardized in IEEE 802.3.
- ♦ Switches are hardware and/or software devices capable of creating temporary connections between 2 or more devices linked to the switch but not to each other.

## Carrier Sense Multiple Access/Collision Detection (CSMA/CD)

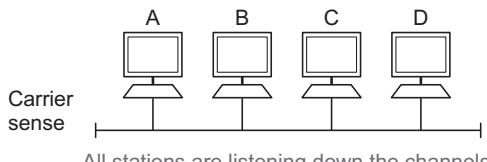
To minimize the chance of collision and, therefore increase the performance, the CSMA/CD was developed. The idea with carrier sense multiple access is to listen down the channels before a transmission takes place.



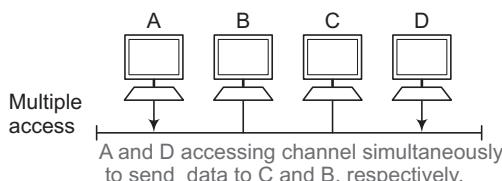
Flow diagram of CSMA/CD

## Persistent CSMA

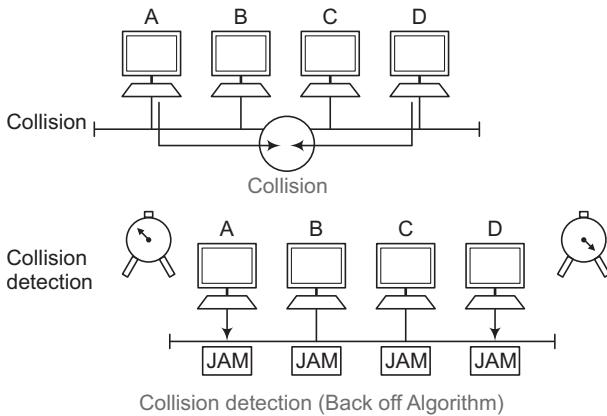
In this scheme, transmission proceeds immediately, if the carrier is idle. However, if the carrier is busy, then sender continues to sense the carrier until it becomes idle. The main problem here is that, if more than one transmitters are ready to send, a collision is guaranteed. In case of collision, stations wait for a random period of time before retransmission.



All stations are listening down the channels



A and D accessing channel simultaneously to send data to C and B, respectively.



There are three types of CSMA approaches as given following

### **Non-persistent CSMA**

In this scheme, the broadcast channel is not monitored continuously. The sender listens it at a random time intervals and transmits whenever the carrier is idle. This decreases the probability of collisions. But it is not efficient in a low load situation, where the number of collisions are anyway small. *The problems it entails are*

- If back-off time is too long, the idle time of carrier is wasted in some sense.
- It may results in long access delays.

### **$p$ -Persistent CSMA**

Even if a sender finds the carrier to be idle, it uses a probabilistic distribution to determine whether to transmit or not. Put simpley "Toss a coin to decide". If carrier is idle, then transmission takes place with a probability  $p$  otherwise the sender waits with a probability ( $q = 1 - p$ ).

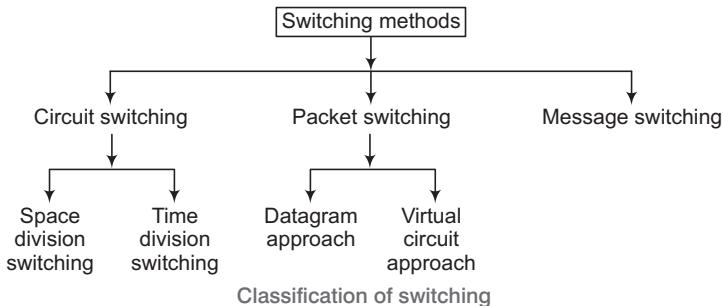
This scheme is good trade-off between the non-persistent and 1-persistent schemes. So, for low load situations,  $p$  is high (e.g., 1-persistent) and for high load situations,  $p$  may be lower.

# Switching

Whenever, we have multiple devices we have a problem of how to connect them to make one-to-one communication possible. *Two solutions could be like as given below*

- Install a point-to-point connection between each pair of devices (Impractical and wasteful approach when applied to very large network).
- For large network, we can go for switching. A switched network consists of a series of interlinked nodes, called switches.

In a switched network, some of these nodes are connected to communicating devices. Others are used only for routing.



## Circuit Switching

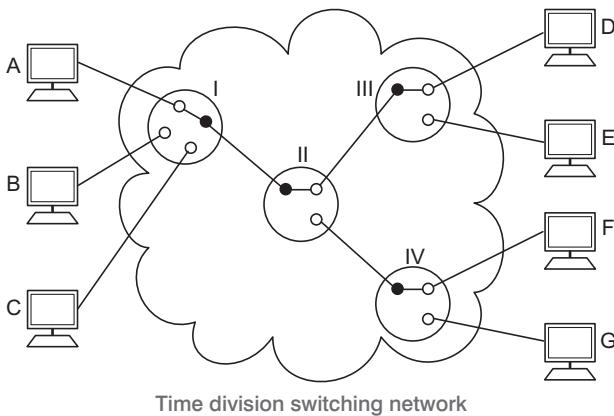
It creates a direct physical connection between two devices such as phones or computers. In the given diagram, instead of point-to-point connections between the 3 computers on the left (A, B, C) to the 4 computers on the right (D,E, F, G) requiring 12 links. We can use 4 switches to reduce the number and the total length of links. So in figure computer, A is connected through switches I, II and III to computer D. A circuit switch is a device with  $n$  inputs and  $m$  outputs that creates a temporary connection between an input link and output link.

### Space division switching

Separates the path in the circuit from each other spatially.

### Time division switching

Uses time division multiplexing to achieve switching.



Circuit switching was designed for voice communication. In a telephone conversation, e.g., Once a circuit is established, it remains connected for the duration of the session.

## Disadvantages of Circuit Switching

- Less well suited to data and other non-voice transmissions. Non-voice transmissions tend to be bursty, meaning that data come in spurts with idle gaps between them. When circuit switched links are used for data transmission, therefore the line is often idle and its facilities wasted.
- A circuit switched link creates the equivalent of a single cable between two devices and thereby assumes a single data rate for both devices. This assumption limits the flexibility and usefulness of a circuit switched connection.
- Once a circuit has been established, that circuit is the path taken by all parts of the transmission, whether or not it remains the most efficient or available.
- Circuit switching sees all transmissions as equal. Any request is granted to whatever link is available. But often with data transmission, we want to be able to prioritise.

## Packet Switching

To overcome the disadvantages of circuit switch. Packet switching concept came into the picture.

In a packet switched network, data are transmitted in discrete units of potentially variable length blocks called packets. Each packet contains not

only data but also a header with control information (such as priority codes and source and destination address). The packets are sent over the network node to node. At each node, the packet is stored briefly, then routed according to the information in its header.

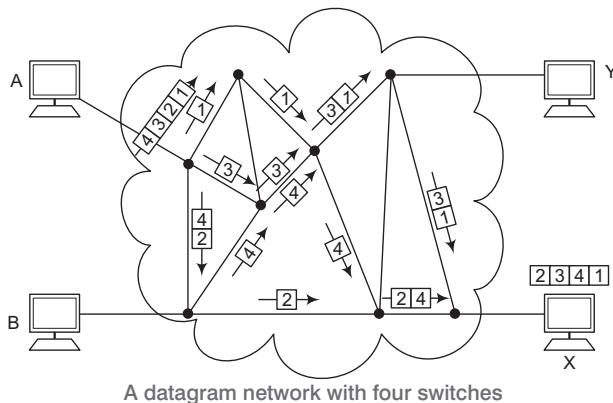
*There are two popular approaches to packet switching.*

- (i) Datagram
- (ii) Virtual circuit

### Datagram Approach

Each packet is treated independently from all others. Even when one packet represents just a piece of a multipacket transmission, the network (and network layer functions) treats it as though it existed alone.

Packets in datagram approach *technology* are referred to as *datagrams*.



### Virtual Circuit Approach

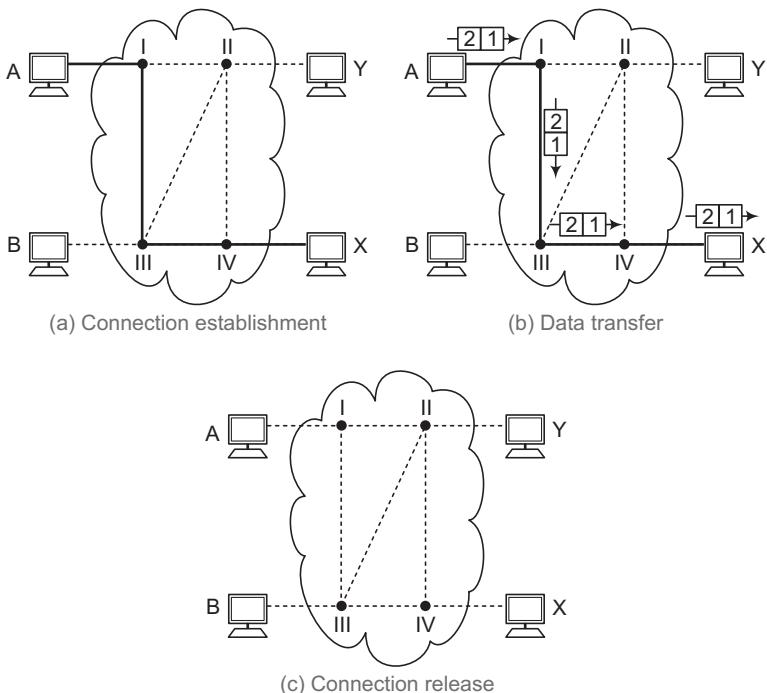
The relationship between all packets belonging to a message or session is preserved. A single route is chosen between sender and receiver at the beginning of the session. When the data are sent, all packets of the transmission travel one after another along that route.

We can implement it into two formats

- Switched Virtual Circuit (SVC)
- Permanent Virtual Circuit (PVC)

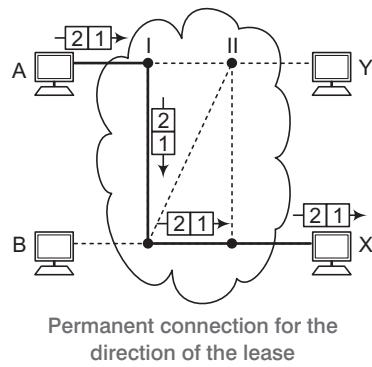
#### SVC (Switched Virtual Circuit)

This SVC format is comparable conceptually to dial-up lines in circuit switching. In this method, a virtual circuit is created whenever, it is needed and exists only for the duration of the specific exchange.



### PVC (Permanent Virtual Circuit)

The PVC format is comparable to leased lines in circuit switching. In this method, the same virtual circuit is provided between two users on a continuous basis. The circuit is dedicated to the specific users. No one else can use it and because it is always in place, it can be used without connection establishment and connection termination.



### Key Points

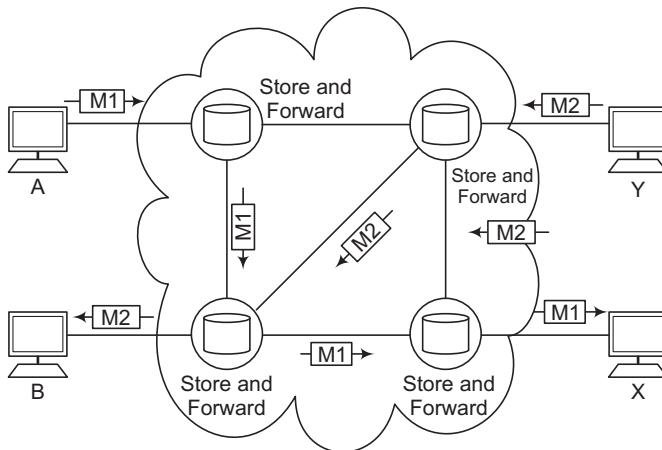
- Whereas, two SVC users may get a different route every time, they request a connection.
- Two PVC users always get the same route.

## Message Switching

It is also known as **store and forward**. In this mechanism, a node receives a message, stores it, until the appropriate route is free, then sends it along.

Store and forward is considered a switching technique because there is no direct link between the sender and receiver of a transmission. A message is delivered to the node along one path, then rerouted along another to its destination.

In message switching, the messages are stored and relayed from secondary storage (disk), while in packet switching the packets are stored and forwarded from primary storage (RAM).



# **Routing and Network Security**

Routing is the process of selecting paths in a network along which to send network traffic. Routing is performed for many kinds of network, including the telephone network, electronic data networks and transportation networks.

*Routing can be grouped into two categories*

1. Non-adaptive routing
2. Adaptive routing

## **Non-adaptive Routing**

Once the pathway to destination has been selected. The router sends all packets for that destination along that one route. In other words, the routing decisions are not made based on the condition or topology of the network.

## **Adaptive Routing**

A router may select a new route for each packet (even packets belonging to the same transmission) in response to changes in condition and topology of the networks.

*For this purpose two common methods are used to calculate the shortest path between two routers*

1. Distance Vector Routing
2. Link State Routing

# **Routing Algorithms**

*There are many routing algorithms*

1. Distance vector Routing
2. Link State Routing
3. Flooding
4. Flow based Routing

## **Distance Vector Routing**

In this routing scheme, each router periodically shares its knowledge about the entire network with its neighbours. Each router has a table with information about network (ID, cost and the router to access the particular network). These tables are updated by exchanging information with the immediate neighbours.

## **Link State Routing**

In link state routing, each router shares its knowledge of its neighbourhood with all routers in the network.

When a router floods the network with information about its neighbourhood, it is said to be advertising. The basis of this advertising is a short packed called a Link State Packet (LSP). An LSP usually contains 4 fields: the ID of the advertiser, the ID of the destination network, the cost and the ID of the neighbour router.

Every router receives LSP and puts the information into a link state database. Because every router receives the same LSPs, every router builds the same database. But the shortest path trees and the routing tables are different for each router. Each router finds out its own shortest paths to the other routers by using **Dijkstra's algorithm**.

### **Flooding Algorithm**

It is a non-adaptive algorithm or static algorithm. When a router receives a packet, it sends a copy of the packet out on each line (except the one on which it arrived).

To prevent form looping forever, each router decrements a hop count contained in the packet header. As soon as the hop count decrements to zero, the router discards the packet.

### **Flow Based Routing Algorithm**

It is a non-adaptive routing algorithm. It takes into account both the topology and the load. In this routing algorithm, we can estimate the flow between all pairs of routers.

Given the line capacity and the flow, we can determine the delay. It needs to use the formula for delay time.

$$T = \frac{1}{\mu c - \lambda}$$

where,  $\frac{1}{\mu}$  = The mean packet size in the bits

$\lambda$  = Mean number of arrivals in packet/sec

and  $c$  = Line capacity (bits/s)

### **The Optimality Principle**

This simple states that if router  $J$  is on the optimal path from router  $I$  to router  $K$ , then the optimal path from  $J$  to  $K$  also falls along this same path.

## Congestion Control

When one part of the subnet (e.g., one or more routers in an area) becomes overloaded, congestions results. Because routers are receiving packet faster than they can forward them, one of the two must happen

The subnet must prevent additional packets from entering the congested region, until those already present can be processed.

The congested routers can discard queued packets to make room for those that are arriving.

### Factors that Cause Congestion

- Packet arrival rate exceeds the outgoing link capacity
- Insufficient memory to store arriving packets
- Bursty traffic
- Slow processor

## Congestion Control Techniques

Several techniques can be employed for congestion control. These include

1. Warning bit
  2. Choke packets
  3. Load shedding
  4. Random early discard
  5. Traffic shaping
- These three deal with congestion detection and recovery
- These two deal with congestion avoidance

### Warning Bit

A special bit in the packet header is set by the router to warn the source when congestion is detected. The bit is copied and piggy-backed on the ACK and sent to sender.

The sender mentions the number of ACK (acknowledgment) packets, it receives with the warning bit set and adjusts its transmission rate accordingly.

### Choke Packets

A choke packet is control packet generated at congested node and transmitted to restrict traffic flow.

The source, one receiving the choke packet must reduce its transmission rate by a certain percentage.

### Load Shedding

When buffers become full routers simply discard packets. Which packet is chosen to be the victim depends on the application and on the error strategy used in data link layer.

For a file transfer for e.g., we can't discard older packets, since this will cause a gap in the received data. For real time voice or video, it is probably better to throw away old data and keep new packets.

### Random Early Discarded (RED)

This is a proactive approach in which the router discards one or more packets before the buffer becomes completely full. Each time a packet arrives, the RED algorithm computes the average queue length, say AVG.

### Key Points

- ♦ If  $\text{AVG} <$  some lower threshold, congestion is assumed to be minimal or non-existent and packet is queued.
- ♦ If  $\text{AVG}$  is greater than some upper threshold, congestion is assumed to be serious and the packet is discarded.
- ♦ If  $\text{AVG}$  is between the two thresholds, this might indicate the onset of congestion. The probability of congestions is then calculated.

### Traffic Shaping

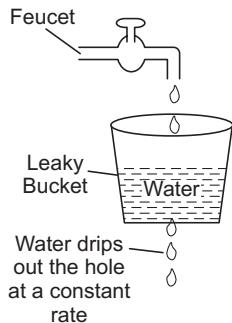
- Another method to congestion control is to **shape** the traffic before it enters the network.
- It controls the rate at which packets are sent (not just how many). Used in ATM and integrated services networks.
- At connections setup time, the sender and carrier negotiate a traffic pattern (shape).

*Two traffic shaping algorithms are as follows*

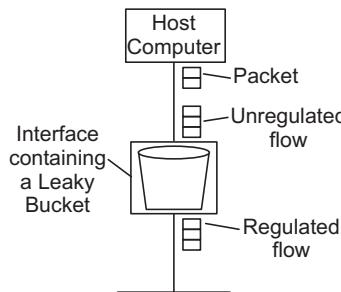
- (i) Leaky Bucket                  (ii) Token Bucket

### The Leaky Bucket (LB) Algorithm

The Leaky Bucket algorithm used to control rate in the network. It is implemented as single-server queue with constant service time. If the buffer (bucket) overflows, then packets are discarded.



(a) A leaky bucket with water



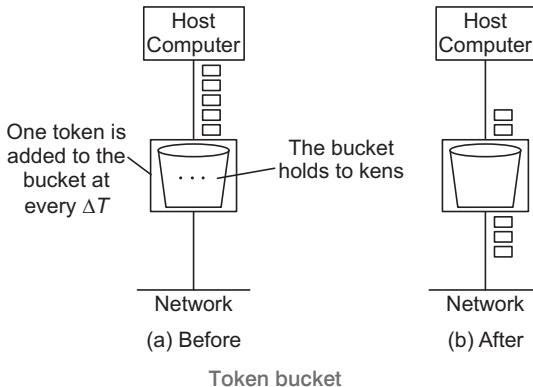
(b) A leaky bucket with packets

The leaky bucket enforces a constant output rate (average rate) regardless of the burstiness of the input. Does nothing when input is idle.

When packets are of the same size (as in ATM cells), the host should inject one packet per clock tick onto the network. But for variable length packets, it is better to allow a fixed number of bytes per tick.

### Token Bucket (TB) Algorithm

In contrast to LB, the token bucket algorithm, allows the output rate to vary depending on the size of the burst.



Token bucket

### Key Points

- ♦ In the TB algorithm, the bucket holds tokens.
- ♦ To transmit a packet, the host must capture and destroy one token.
- ♦ Tokens are generated by a clock at the rate of one token every  $\Delta t$  sec.
- ♦ Idle hosts can capture and save up tokens (upto the maximise size of the bucket) in order to send larger bursts later.

## **Protocols used in Network Layer**

*Some of the important protocols used in network layer with their functions are given below*

### **Address Resolution Protocol (ARP)**

ARP is used to find the physical address of the node when its Internet address is known. Anytime, a host or a router needs to find the physical address of another host on its network, it formats an ARP query packet that includes that IP address and broadcasts it over the network. Every host on the network receives and processes the ARP packet, but the intended recipient recognises its Internet address and sends back its physical address.

### **Reverse Address Resolution Protocol (RARP)**

This protocol allows a host to discover its Internet address when it knows only its physical address. RARP works much like ARP. The host wishing to retrieve its Internet address broadcasts an RARP query packet that contains its physical address to every host of its physical network. A server on the network recognises the RARP packet and returns the host's Internet address.

### **Internet Control Message Protocol (ICMP)**

The ICMP is a mechanism used by hosts and routers to send notifications of datagram problems back to the sender. IP is essentially an unreliable and connectionless protocol. ICMP allows IP (Internet Protocol) to inform a sender, if a datagram is undeliverable.

ICMP uses each test/reply to test whether a destination is reachable and responding. It also handles both control and error messages but its sole function is to report problems not correct them.

### **Internet Group Message Protocol (IGMP)**

The IP can be involved in two types of communication unicasting and multicasting. The IGMP protocol has been designed to help a multicast router to identify the hosts in a LAN that are members of a multicast group.

## Addressing at Network Layer

In addition to the physical addresses that identify individual devices, the Internet requires an additional addressing connection an address that identifies the connection of a host of its network. Every host and router on the Internet has an IP address which encodes its network number and host number. The combination is unique in principle, no 2 machines on the Internet have the same IP address.

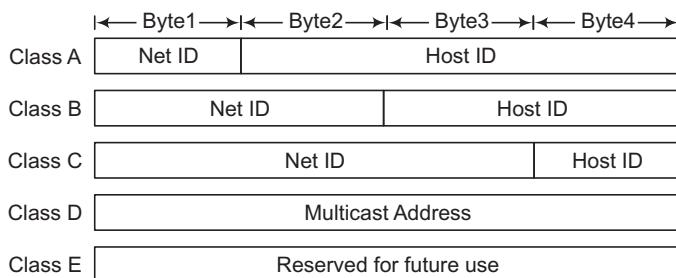
### Classes and Subnetting

*There are currently five different field length pattern in use, each defining a class of address.*

An IP address is 32 bit long. One portion of the address indicates a network (Net ID) and the other portion indicates the host (or router) on the network (i.e., Host ID).

To reach a host on the Internet, we must first reach the network, using the first portion of the address (Net ID). Then, we must reach the host itself, using the 2nd portion (Host ID).

The further division a network into smaller networks called **subnetworks**.



**For Class A** First bit of Net ID should be 0 like in following pattern  
01111011·10001111·11111100·11001111

**For Class B** First 2 bits of Net ID should be 1 and 0 respective, as in below pattern  
10011101·10001111·11111100·11001111

**For Class C** First 3 bits Net ID should be 1,1 and 0 respectively, as follows  
11011101·10001111·11111100·11001111

**For Class D** First 4 bits should be 1110 respectively, as in pattern  
11101011·10001111·11111100·11001111

**For Class E** First 4 bits should be 1111 respectively, like

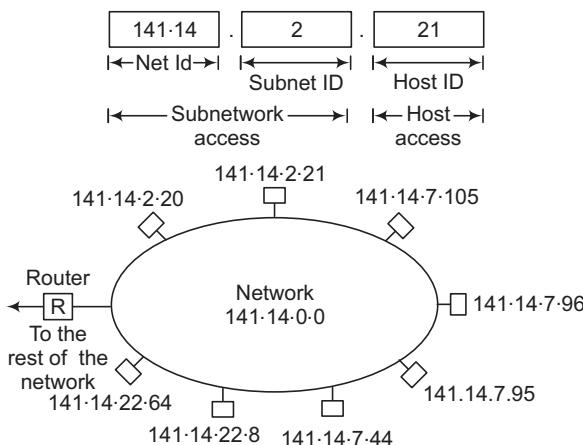
11110101·10001111·11111100·11001111

### Class Ranges of Internet Address in Dotted Decimal Format

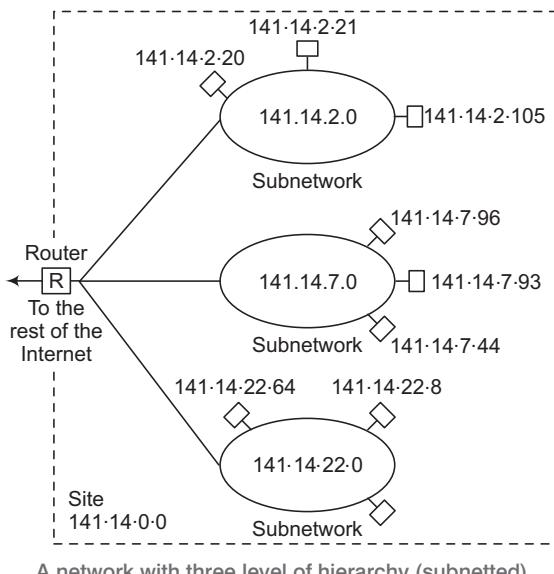
	From	
Class A	0·0·0·0	127·255·255·255
	←→ ← Host ID →	←→ ← Host ID →
	Net ID	Net Id
Class B	128·0·0·0	191·255 255·255
	←→ ← Host ID →	←→ ← Host ID →
	Net ID	Net Id
Class C	192·0·0·0	223·255 255·255
	←→ ← Host ID →	←→ ← Host ID →
	Net ID	Net ID      Host ID
Class D	224·0·0·0	239·0·0·0
	←→ Group Address →	←→ Group Address →
Class E	240·0·0·0	255·255 255·255
	←→ Undefined →	←→ Undefined →

### Three Levels of Hierarchy

Adding subnetworks creates an intermediate level of hierarchy in the IP addressing system. Now, we have *three levels*: net ID; subnet ID and host ID. e.g.,



A network with two level of hierarchy (not subnetted)



## Masking

Masking is process that extracts the address of the physical network form an IP address. Masking can be done whether we have subnetting or not. If we have not subnetted the network, masking extracts the network address form an IP address. If we have subnetted, masking extracts the subnetwork address form an IP address.

### Masks without Subnetting

To be compatible, routers use mask even, if there is no subnetting.



### Masks with Subnetting

When there is subnetting, the masks can vary



### Masks for Unsubnetted Networks

Class	Mask	Address (Example)	Network Address
A	255 · 0 · 0 · 0	15 · 32 · 56 · 7	15 · 0 · 0 · 0
B	255 · 255 · 0 · 0	135 · 67 · 13 · 9	135 · 67 · 0 · 0
C	255 · 255 · 255 · 0	201 · 34 · 12 · 72	201 · 34 · 12 · 0
D	N/A	N/A	N/A
E	N/A	N/A	N/A

### Masks for Subnetted Networks

Class	Mask	Address (Example)	Subnetwork Address
A	255 · 255 · 0 · 0	15 · 32 · 567	15 · 32 · 0 · 0
B	255 · 255 · 255 · 0	135 · 67 · 13 · 9	135 · 67 · 13 · 0
C	255 · 255 · 255 · 192	201 · 34 · 12 · 72	201 · 34 · 12 · 0
D	N/A	N/A	N/A
E	N/A	N/A	N/A

## Types of Masking

There are two types of masking as given below

### Boundary Level Masking

If the masking is at the boundary level (the mask numbers are either 255 or 0), finding the subnetwork address is very easy. *Follow these 2 rules*

- The bytes in IP address that correspond to 255 in the mask will be repeated in the subnetwork address.
- The bytes in IP address that correspond to 0 in the mask will change to 0 in the subnetwork address.

e.g.,      **IP address**    45 · 23 · 21 · 8  
**Mask**                255 · 255 · 0 · 0  
**Subnetwork**    45 · 23 · 0 · 0  
**address**

### Non-boundary Level Masking

If the masking is not at the boundary level (the mask numbers are not just 255 or 0), finding subnetwork address involves using the bit-wise AND operator. *follow these 3 rules*

- The bytes in IP address that correspond to 255 in the mask will be repeated in the subnetwork address.

- The bytes in the IP address that correspond to 0 in the mask will be change to 0 in the subnetwork address.
- For other bytes, use the bit-wise AND operator

e.g.,      **IP address**      213 · 23 · 47 · 37  
**Mask**                  255 · 255 · 255 · 240  
**Subnetwork**       $\underline{213 \cdot 23 \cdot 47 \cdot 32}$   
**address**

As we can see, 3 bytes are easy to determine. However, the 4th bytes needs the bit-wise AND operation.

37	0 0 1 0 0 1 0 1
240	<hr/>
32	1 1 1 1 0 0 0 0

## Transport Layer Protocols

There are two transport layer protocols as given below

### UDP (User Datagram Protocol)

UDP is a connectionless protocol. UDP provides a way for application to send encapsulate IP datagram and send them without having to establish a connection.

UDP transmitted segments consisting of an 8 byte header followed by the payload. The two parts serve to identify the end points within the source and destinations machine. When UDP packets arrives, its payload is handed to the process attached to the destination ports.

Source Port Address (16Bits)	Destination Port Address (16 Bits)
Total length of the User Datagram (16 Bits)	Checksum (used for error detection) (16 Bits)

UDP datagram format

### TCP (Transmission Control Protocol)

TCP provides full transport layer services to applications. TCP is reliable stream transport port-to-port protocol. The term stream, in this context, means connection-oriented, a connection must be established between both ends of transmission before either may transmit data. By creating this

connection, TCP generates a virtual circuit between sender and receiver that is active for the duration of transmission.

Source port address 16 Bits				Destination port address 16 Bits							
Sequence number (32 Bits)											
Acknowledgement Number (32 Bits)											
HLEN 4 Bits	Reserved 6 Bits	U R G	A C K	P S H	R S T	S Y N	F I N				
Checksum 16 Bits				Window size 16 Bits							
Options and Padding				Urgent Pointer 16 Bits							

TCP segment format

Each machine supporting TCP has a TCP transport entity either a library procedure, a user process or part of kernel. In all cases, it manages TCP streams and interfaces to the IP layer. A TCP entities accepts the user data stream from local processes, breaks them up into pieces not exceeding 64 K bytes and sends each piece as separate IP datagrams.

### Key Points

- When a datagram containing TCP data arrive at a machine, they are given to the TCP entity, which reconstruct the original byte stream.
- All TCP connections are full duplex and point-to-point and TCP connection is a byte stream not message steam.

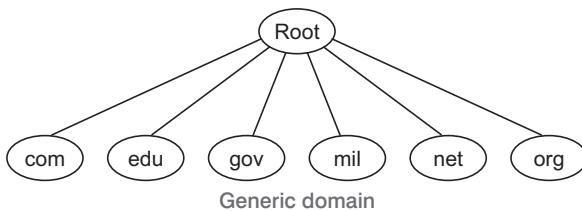
## Domain Name System (DNS)

To identify an entity, TCP/IP protocol uses the IP address which uniquely identifies the connection of a host to the Internet. However, people refer to use names instead of address. Therefore, we need a system that can map a name to an address and conversely an address to name. In TCP/IP, this is the domain name system.

### DNS in the Internet

DNS is protocol that can be used in different platforms. *Domian name space is divided into three categories.*

- Generic Domain** The generic domain defines registered hosts according to their generic behavior. Each node in the tree defines a domain which is an index to the domain name space database.



- **Country Domain** The country domain section follows the same format as the generic domain but uses 2 characters country abbreviations (e.g., US for United States) in place of 3 characters.
- **Inverse Domain** The inverse domain is used to map an address to a name.

## Application Layer Protocols

*There are various application layer protocols as given below*

### SMTP (Simple Mail Transfer Protocol)

One of the most popular network service is electronic mail (e-mail). The TCP/IP protocol that supports electronic mail on the Internet is called Simple Mail Transfer Protocol (SMTP).

SMTP is system for sending messages to other computer. Users based on e-mail addresses. SMTP provides services for mail exchange between users on the same or different computers.

### TELNET (Terminal Network)

TELNET is client-server application that allows a user to log onto remote machine and lets the user to access any application program on a remote computer.

TELNET uses the NVT (Network Virtual Terminal) system to encode characters on the local system. On the server (remote) machine, NVT decodes the characters to a form acceptable to the remote machine.

### FTP (File Transfer Protocol)

FTP is the standard mechanism provided by TCP/IP for copying a file from one host to another.

FTP differs from other client-server applications because it establishes 2 connections between hosts. One connection is used for data transfer, the other for control information (commands and responses).

## **Multipurpose Internet Mail Extensions (MIME)**

It is an extension of SMTP that allows the transfer of multimedia messages.

## **POP (Post Office Protocol)**

This is a protocol used by a mail server in conjunction with SMTP to receive and holds mail for hosts.

**HTTP** (Hypertext Transfer Protocol) This is a protocol used mainly to access data on the World Wide Web (www) , a repository of information spread all over the world and linked together.

The HTTP protocol transfer data in the form of plain text, hyper text, audio, video and so on.

### **Important Terms Related to Computer Security**

- **Computer Security** It is a generic name of the collection of tools designed to protect data and thwart hackers.
- **Network Security** It measures to protect data during their transmission.
- **Internet Security** It measures to protect data during their transmission over a collection of interconnected networks.
- **Information Security** We consider 3 aspects of information security
  - (a) Security attack
  - (b) Security mechanism
  - (c) Security service

## **Security Services**

*These are two main categories of security services*

**RFC 2828** It defines as a processing or communication service provided by a system to give a specific kind of protection to system resources.

**X.800** It defines as a service provided by a protocol layer of communicating open systems, which ensures adequate security of the systems or of data transfers.

### **X.800 Classification**

*X. 800 defines it in five major categories*

- **Authentication** Assurance that the communicating is the one claimed.
- **Access Control** Prevention of the unauthorised use of resources.
- **Data Confidentiality** Protection of data from unauthorised disclosure.
- **Data Integrity** Assurance that data received is as sent by an authorised entity.
- **Non-Repudiation** Protection against denial by one of the parties in a communication.

## Security Mechanism

Feature designed to detect, prevent or recover from security attack.

*The two categories of security mechanism are as follows*

### Specific Security Mechanism

Encipherment, digital signature, access control, data integrity, authentication exchange, traffic padding, routing control, notarisation.

### Pervasive Security Mechanism

Treated functionality, security labels, event detection, security audit trails, security recovery.

## Security Attacks

It is an action that compromises the security of information owned by an organisation.

*These are two types of security attacks*

### Passive Attacks

Eavesdropping on or monitoring of transmission to

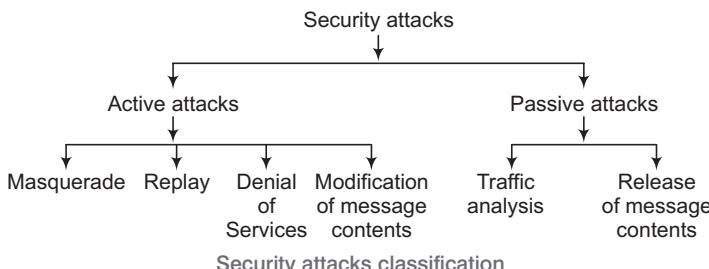
- Obtain message contents              • Monitor traffic flows

### Active Attacks

In computer and computer networks an attack is any attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset.

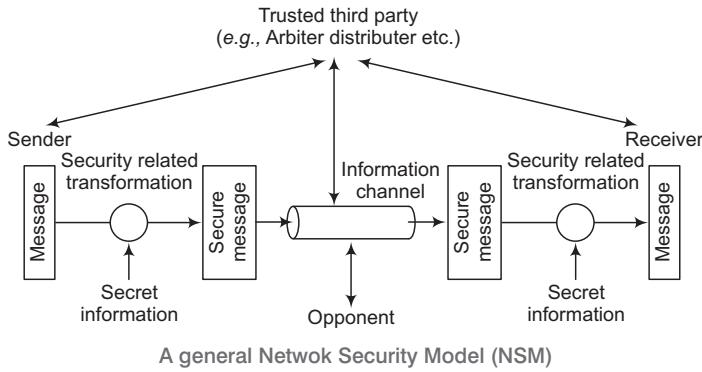
*Modification of data stream to*

- Masquerade of one entity as some other
- Replay previous messages
- Modify messages in transit
- Denial of service



## Model for Network Security

Network security starts with authenticating, commonly with a username and password. since, this requires just one detail authenticating the username i.e., the password this is some times teamed one factor authentication.



Using this model require us to

- Design a suitable algorithm for the security transformation.
- Generate the secret in formations (keys) used by the algorithm.
- Develop methods to distribute and share the secret information.
- Specify a protocol enabling the principles to use the transformation and secret information for security service.

# Cryptography

It is a science of converting a stream of text into coded form in such a way that only the sender and receiver of the coded text can decode the text.

Now a days, computer use requires automated tools to protect files and other stored information. Use of network and communication links require measures to protect data during transmission.

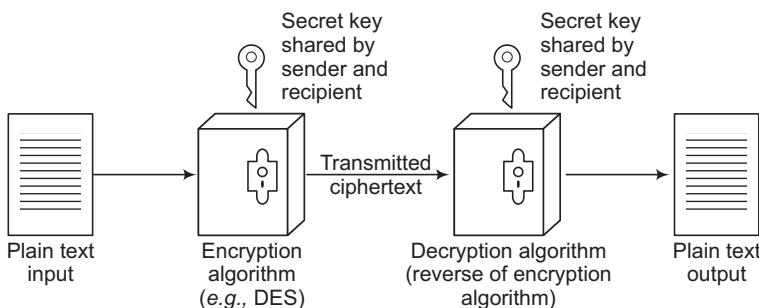
## Symmetric/Private Key

### Cryptography (Conventional/ Private key/Single key)

Symmetric key algorithms are a class of algorithms fo cryptography that use the same cryptographic key for both encryption of plaintext and decryption of ciphertext. The may be identical or there may be a simple transformation to go between the two keys.

*In symmetric private key cryptography the following key features are involved*

- Sender and recipient share a common key.
- It was only prior to invention of public key in 1970.
- If this shared key is disclosed to opponent, communications are compromised.
- Hence, does not protect sender form receiver forging a message and claiming is sent by user.



**Symmetric cipher model** (used in symmetric encryption)

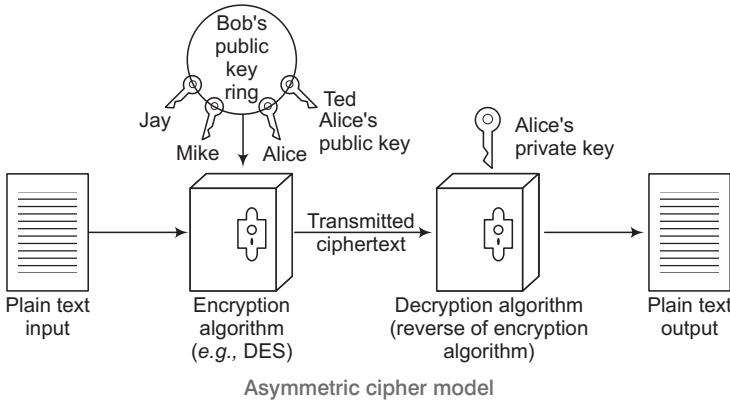
## Asymmetric/Public Key Cryptography

A public key cryptography refers to a cyptographic system requiring two separate keys, one of which is secrete/private and one of which is public although different, the two parts of the key pair are mathematically linked.

- Public Key** A public key, which may be known by anybody and can be used to encrypt messages and verify signatures.
- Private Key** A private key, known only to the recipient, used to decrypt messages and sign (create) signatures.

It is symmetric because those who encrypt messages or verify signature cannot decrypt messages or create signatures. It is computationally infeasible to find decryption key knowing only algorithm and encryption key.

Either of the two related keys can be used for encryption, with the other used for decryption (in some schemes).



In the above public key cryptography mode

- Bob encrypts a plaintext message using Alice's public key using encryption algorithm and sends it over communication channel.
- On the receiving end side, only Alice can decrypt this text as she only is having Alice's private key.

## Message Authentication Codes (MAC)

In cryptography, a Message Authentication Code (MAC) is a short piece of information used to authenticate a message and to provide integrity and authenticity assurance on the message. Integrity assurance detects accidental and intentional message changes, while authenticity assurance affirms the message's origin.

A keyed function of a message sender of a message  $m$  computes MAC ( $m$ ) and appends it to the message.

**Verification** The receiver also computes MAC ( $m$ ) and compares it to the received value.

**Security of MAC** An attacker should not be able to generate a valid  $(m, \text{MAC}(m))$ , even after seeing many valid messages MAC pairs, possible of his choice.

## MAC from a Block Cipher

MAC from a block cipher can be obtained by using the following suggestions

- Divide a message into blocks.
- Compute a checksum by adding (or xoring) them.
- Encrypt the checksum.

### Key Points

- ♦ MAC keys are symmetric. Hence, does not provide non-repudiation (unlike digital signatures).
- ♦ MAC function does not need to be invertible.
- ♦ A MACed message is not necessarily encrypted.

## RSA Algorithm

RSA is an algorithm for public key cryptography RSA (Rivest Shamir Adleman) algorithm was publicly described in 1977.

### Mathematical Background of RSA Algorithm

**Extended Euclidian algorithm** Given  $x$ , find  $y$ , such that  $x \cdot y = 1 \pmod{m}$ . The extended Euclidian algorithm can efficiently find the solution to this problem.

**Euler's Theorem** For any number, a relatively prime to

$$n = pq, a^{(p-1)(q-1)} = 1 \pmod{pq}$$

(i) Why this is very useful?

(ii) Let  $Z = k(p-1)(q-1) + r$ , we have

$$a^Z = a^{k(p-1)(q-1)} \times a^r \dots = a^r \pmod{pq}$$

(iii) In other words, If  $z = r \pmod{(p-1)(q-1)}$ , then  $a^z = a^r \pmod{pq}$

**Special case** If  $z = 1 \pmod{(p-1)(q-1)}$ , then  $a^z = a \pmod{pq}$

We can use Euler's theorem to simplify  $a^z \pmod{pq}$

## RSA Algorithm

(i) Let  $n = pq$ , where  $p$  and  $q$  are 2 large primes.

(ii) Public key  $(e, n)$ , where  $e$  is relative prime to

$$(p - 1)(q - 1)$$

(iii) Private key  $(d, n)$ , such that

$$ed \equiv 1 \pmod{(p - 1)(q - 1)}$$

$d$  can calculated using extended Euclidian Algorithm

**Encryption**  $c = m^e \pmod{n}$

**Decryption**  $c^d = (m^e)^d = m^{ed} \pmod{n}$

**Security of RSA** dependes on the hardness of factoring.

factoring  $n = p \times q$  is hard when  $n$  is large.

### DES (Data Encryption Standard)

The data encryption standard was developed in IBM. DES is a symmetric key crypto system which has a 56 bit key and encrypts 64 bit plaintext to 64 bit cipher texts . To improve DES, the concept of 3 DES was introduced.

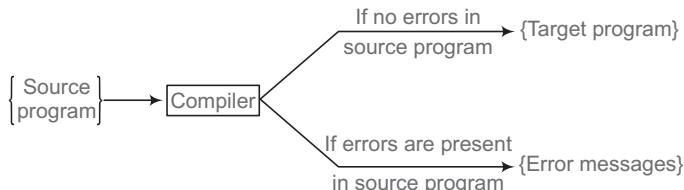
# 7

# Compiler Design

## Compiler

A compiler is a program written in one language (i.e., source language) and translate it into an equivalent program in a target language.

This translation process could also report the errors in the source system, if any.



Diagrammatic view of working of a compiler

### Major Parts of a Compiler

*There are two major parts of a compiler*

- 1. Analysis      2. Synthesis

- In analysis phase, an intermediate representation is created from the given source program. Lexical analyzer, syntax analyzer and semantic analyzer are phases in this part.
- In synthesis phase, the equivalent target program is created from this intermediate representation. Intermediate code generator, code generator and code optimizer are phases in this part.

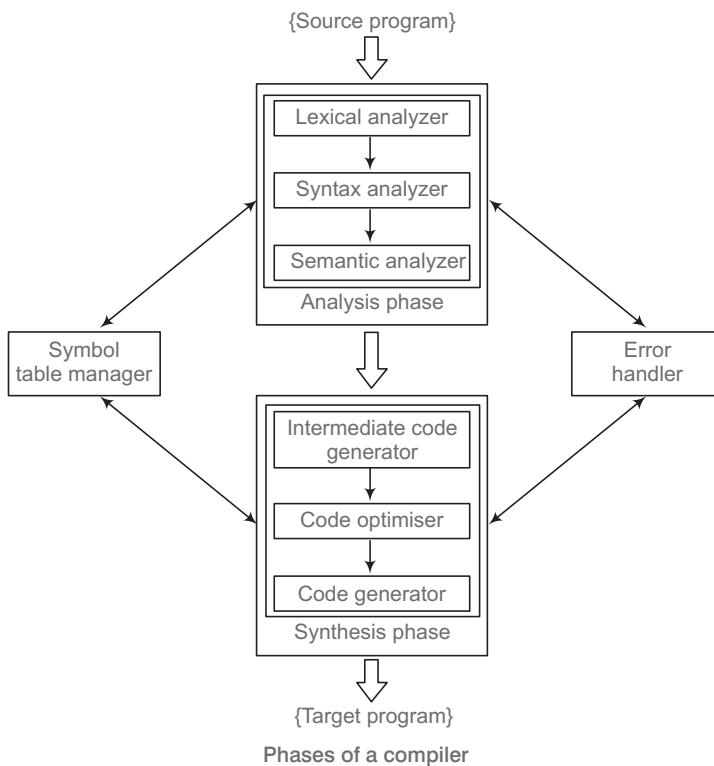
## Phases of a Compiler

Each phase shown in the figure (phases of a compiler) transform from one representation into another representation. They communicate with error handlers and the symbol table.

### Lexical Analyzer

Lexical analyzer reads the source program character by character and returns the tokens of the source program. It puts information about identifiers into the symbol table.

Some typical task of lexical analyzer includes recognising reserved keywords, ignoring comments, binding integer and floating point constants. Counting the number of lines, finding identifiers (variables), finding string and character constants, reporting error messages and treating white spaces appropriately in the form of blank, tab and new line characters.



## Tokens, Lexemes and Patterns

- A token describes a pattern of characters having same meaning in the source program such as identifiers, operators, keywords, numbers, delimiters and so on. A token may have a single attribute which holds the required information for that token. For identifiers, this attribute is a pointer to the symbol table and the symbol table holds the actual attributes for that token.
- Token type and its attribute uniquely identify a lexeme.
- Regular expressions are widely used to specify pattern.

**Note** A token can represent more than one lexeme, additional information (*i.e.*, attribute of the token) should be held for that specific lexeme.

## Specification of Tokens

Regular expressions are an important notation for specifying patterns. Each pattern matches a set of strings, so regular expressions will serve as names for sets of string.

## Recognition of Tokens

Lexical analyzer are based on a simple computational model called as the **finite state automata**.

Transition diagram depicts the actions that take place when a lexical analyzer is called by the **parser** to get the next token.

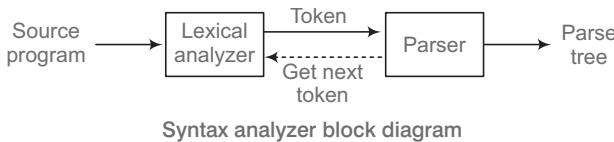
## Syntax Analyzer (Parser)

Syntax analyzer creates the syntactic structure of the given source program. This syntactic structure is mostly a parse tree. The syntax of a programming is described by a **Context-Free Grammar** (CFG). We will use BNF (Backus-Naur Form) notation in the description of CFGs.

The syntax analyzer (parser) checks whether a given source program satisfies the rules implied by a context-free grammar or not. If it satisfies, the parser creates the parse tree of that program. Otherwise the parser gives the error messages.

## Key Points

- A context-free grammar gives a precise syntactic specification of a programming language.
- The design of the grammar is an initial phase of the design of a compiler.
- A grammar can be directly converted into a parser by some tools which works on stream of tokens and the smallest item.
- Parser sends get next token command to lexeme to identify next token.



Syntax analyzer block diagram

We categorise the parser into two groups

- (i) Top-down parser (starts from the root).
- (ii) Bottom-up parser (starts from the leaf).
- Both top-down and bottom-up parsers scan the input from left-to-right (one symbol at a time).
- Efficient top-down and bottom-up parsers can be implemented only for subclasses of context-free grammars.
  - (i) LL for top-down parsing
  - (ii) LR for bottom-up parsing

### Context-Free Grammars

Inherently recursive structures of a programming language are defined by a CFG. In a CFG, we have A start symbol (one of the non-terminals). A finite set of terminals (in our case, this will be the set of tokens). A set of non-terminals (syntactic variables).

A finite set of production rules in the following form

$A \rightarrow \alpha$ , where  $A$  is non-terminal and  $\alpha$  is a string of terminals (including the empty string).

### Derivations

It refers to replacing an instance of a non-terminal in a given string's non terminal by the right hand side of production rule, whose left hand side contains the non-terminal to be replaced. Derivation produces a new string from a given string.

If the derived string contained only terminal symbols, then no further derivation is possible.

$E \Rightarrow -E$ , is read as “ $E$  derives  $-E$ ”.

Each derivation step needs to choose a non-terminal to rewrite and a production rule to apply. A left most derivation always chooses the left most non-terminal to rewrite.

e.g.,  $E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + id$

A right most derivation always chooses the right most non-terminal to rewrite.

e.g.,  $E \Rightarrow E + E \Rightarrow E + id \Rightarrow id + id$

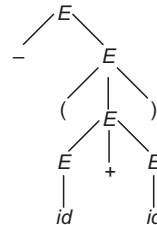
## Parse Trees

Graphical representation for a derivation that filters out the order of choosing non-terminals to avoid rewriting. The root node represents the start symbol, inner nodes of a parse tree are non-terminal symbol.

The leaves represent terminal symbols

$$\begin{aligned}
 E &\Rightarrow -E \\
 &\Rightarrow - (E) \\
 &\Rightarrow - (E + E) \\
 &\Rightarrow - (id + E) \\
 &\Rightarrow - (id + id)
 \end{aligned}$$

Left most derivation



Parse tree diagram

## Ambiguity

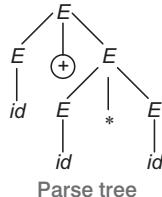
A grammar produces more than one parse tree for a sentence is called as **ambiguous grammar**. **Unambiguous grammar** refers unique selection of the parse tree for a sentence.

We should eliminate the ambiguity in the grammar during the design phase of the compiler. Ambiguous grammars can be disambiguated according to the precedence and associativity rules.

e.g., Consider the grammar

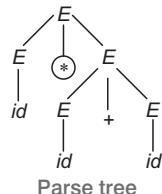
$$E \Rightarrow E + E \mid E * E \mid id$$

If we start with the  $E \Rightarrow E + E$ , then the parse tree will be created as follows



Parse tree

If we start with the  $E \Rightarrow E * E$ , then the parse tree will be created as follows



Parse tree

**Note** The above grammar is ambiguous as we are getting more than one parse tree for it.

## Removing Ambiguity

Consider an example of ambiguous grammar as given below

$$E \rightarrow E + E \mid E * E \mid E \wedge E \mid \text{id} \mid (E)$$

We can use precedence of operators as follows

$\wedge$  (right to left)

$*$  (left to right)

$+$  (left to right)

Using the above operator precedence and associativity, we get the following unambiguous grammar

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow G \wedge F \mid G$$

$$G \rightarrow \text{id} \mid (E)$$

## Left Recursion

A grammar is left recursive, if it has a non-terminal A such that there is a derivation.

$$A \Rightarrow A \alpha \text{ for some string } \alpha$$

Top-down parsing technique can't handle left recursive grammars. So, we have to convert our left recursive grammar into an equivalent grammar which is not left recursive.

The left-recursion may appear in a single step of the derivation (immediate left recursion) or may appear in more than one step of the derivation.

## Left Factoring

A predictive parser (a top-down parser without backtracking) insists that the grammar must be left factored.

grammar  $\rightarrow$  a new equivalent grammar suitable for predictive parsing.

stmt  $\rightarrow$  if expr then stmt else stmt | if expr then stmt

When we see, if we can't know which production rule is to be chosen then rewrite stmt in the derivation.

In general,

$$A \rightarrow \alpha \beta_1 \mid \alpha \beta_2,$$

where  $\alpha$  is not empty and the first symbols of  $\beta_1$  and  $\beta_2$  (if they have one) are different.

When processing  $\alpha$ , we can't know whether expand

$A \rightarrow \alpha \beta_1$  or

$A \rightarrow \alpha \beta_2$

But, if we rewrite the grammar as follows

$A \rightarrow \alpha A^1$

$A^1 \rightarrow \beta_1 \mid \beta_2$ , so we can immediately expand  $A$  to  $\alpha A^1$ .

## YACC

YACC generates C code for a syntax analyzer or parser, YACC uses grammar rules that allow it to analyze token from LEX and create a syntax tree. A syntax tree imposes a hierarchical structure tokens. e.g., operator precedence and associativity are apparent in the syntax tree.

YACC takes a default action when there is a conflict. For shift reduce conflicts, YACC will shift. For shift reduce conflict, it will use the first rule in the listing. It also issues a warning message whenever a conflict exists.

## Key Points

- ♦ The warning may be suppressed by making the grammar unambiguous.
- ♦ The definition S section consists of token declarations and C code bracketed by "% {" and "% }".
- ♦ The BNF grammar is placed in the rules section and the user subroutines are added in the subroutines section.
- ♦ Input to YACC is divided into three sections
  - ... definitions ...
  - %%
  - ... rules ...
  - %%
  - ... subroutines ...

## Top-down Parsing

There are two main techniques to achieve top-down parse tree

- (i) Recursive descent parsing
- (ii) Predictive parsing
- Recursive predictive parsing
- Non-recursive predictive parsing

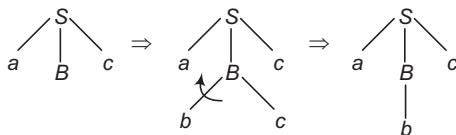
### Recursive Descent Parsing (Uses Backtracking)

Backtracking is needed (if a choice of a production rule doesn't work, we backtrack to try other alternatives). It tries to find the left most derivation. It is not efficient.

e.g., If the grammar is

$$S \rightarrow aBc$$

$B \rightarrow bc \mid b$  and the input is abc



In case failure, it backtracks to generate the new one

### Predictive Parser

Grammar  $\xrightarrow{\text{Eliminate left recursion}}$   $\xrightarrow{\text{Left factor}}$  A grammar suitable for predictive parsing a LL(1) grammar (no 100% guarantee).

When re-writing a non-terminal in a derivation step, a predictive parser can uniquely choose a production rule by just looking the current symbol in input string.

e.g.,  $\text{stmt} \rightarrow \text{if } \dots \mid \text{while } \dots \mid \text{begin } \dots \mid \text{for } \dots$

- When we are trying to write the non-terminal stmt, we have to choose first production rule.
- When we are trying to write the non-terminal stmt, we can uniquely choose the production rule by just looking the current token.
- We eliminate the left recursion in the grammar and left factor it. But it may not be suitable for predictive parsing (not LL (1) grammar).

**Recursive Predictive Parsing** Each non-terminal corresponds to a procedure.

e.g.,  $A \rightarrow aBb \mid bAB$

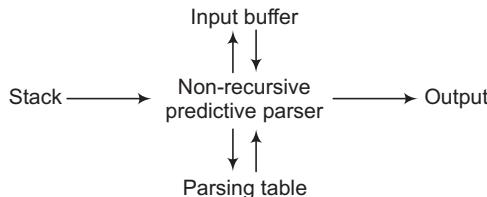
```

Proc A
{
    case of the current token
    {
        'a' - match the current token with a and
        move to the next token;
        - call B;
        - match the current token with b and move to
        the next token;
        'b'-match the current token with b and move
        to the next token;
        - call 'A';
        - call 'B';
    }
}

```

#### Non-recursive Predictive Parsing (LL (1) Parser)

- Non-recursive predictive parsing is a table driven parser.
- It is a top-down parser.
  - It is also known as LL (1) parser.



#### Processing of non-recursive predictive parsing

**Input buffer** Our string to be parsed. We will assume that its end is marked with a special symbol \$.

**Output** A production rule representing a step of the derivation sequence (left most derivation) of the string in the input buffer.

**Stack** Contains the grammar symbol. At the bottom of the stack, there is a special end marker symbol \$. Initially the stack contains only the symbol \$ and the starting symbol S ( $\$S \leftarrow$  initial stack). When the stack is emptied (i.e., only \$ left in the stack), the parsing is completed.

## Parsing table

- A two-dimensional array  $M [A, a]$ .
- Each row is a non-terminal symbol.
- Each column is a terminal symbol or the special symbol \$.
- Each entry holds a production rule.

## Parser Actions

The symbol at the top of the stack (say  $X$ ) and the current symbol in the input string (say  $a$ ) determine the parser action.

*There are four possible parser actions*

- (i) If  $X$  and  $a$  are  $\$$  → parser halts (successful completion).
- (ii) If  $X$  and  $a$  are the same terminal symbol (different from  $\$$ ).

Parser pops  $X$  from the stack and moves the next symbol in the input buffer.

- (iii) If  $X$  is a non-terminal.

Parser looks at the parsing table entry  $M[X, a]$ . If  $M[X, a]$  holds a production rule  $X \rightarrow Y_1, Y_2, \dots, Y_k$ , it pops  $X$  from the stack and pushes  $Y_k, Y_{k-1}, \dots, Y_1$  into the stack. The parser also outputs the production rule  $X \rightarrow Y_1, Y_2, \dots, Y_k$  to represent a step of the derivation.

- (iv) None of the above → error

All empty entries in the parsing table are errors.

If  $X$  is a terminal symbol different from  $a$ , this is also an error case.

## Functions used in Constructing LL (1) Parsing Tables

- Two functions are used in the construction of LL (1) parsing tables -FIRST and FOLLOW.
- FIRST ( $\alpha$ ) is a set of the terminal symbols which occur as first symbols in strings derived from  $\alpha$ , where  $\alpha$  is any string of grammar symbols. If  $\alpha$  derives to  $\epsilon$ , then  $\epsilon$  is also in FIRST ( $\alpha$ ).
- FOLLOW (A) is the set of the terminals which occur immediately after (FOLLOW) the non-terminal A in the strings derived from the starting symbol.
- A terminal  $a$  is in FOLLOW (A), if  $S \Rightarrow \alpha A a \beta$
- $\$$  is in FOLLOW (A), if  $S \Rightarrow \alpha A$

## To Compute FIRST of any String X

- If  $X$  is a terminal symbol → FIRST ( $X$ ) =  $\{X\}$
- If  $X$  is a non-terminal symbol and  $X \rightarrow \epsilon$  is a production rule →  $\epsilon$  is in FIRST ( $X$ ).
- If  $X$  is a non-terminal symbol and  $X \rightarrow Y_1, Y_2, \dots, Y_n$  is a production rule.  
If a terminal  $a$  in FIRST ( $Y_i$ ) and  $\epsilon$  is in all FIRST ( $Y_j$ ) for  $j = 1, \dots, i-1$ , then  $a$  is in FIRST ( $X$ ).  
If  $\epsilon$  is in all FIRST ( $Y_j$ ) for  $j = 1, \dots, n$ , then  $\epsilon$  is in FIRST ( $X$ ).
- If  $X$  is  $\epsilon$ , then FIRST ( $X$ ) =  $\{\epsilon\}$
- If  $X$  is  $Y_1, Y_2, \dots, Y_n$

If a terminal  $a$  in FIRST ( $Y_j$ ) and  $\epsilon$  is in all FIRST ( $Y_j$ ) for  $j = 1, \dots, i - 1$ , then  $a$  is in FIRST ( $X$ ).

If  $\epsilon$  is in all FIRST ( $Y_j$ ) for  $j = 1, \dots, n$ , then  $\epsilon$  is in FIRST ( $X$ ).

To compute FOLLOW (for Non-terminals)

If  $S$  is the start symbol,  $\$$  is in FOLLOW ( $S$ ).

- If  $A \rightarrow \alpha B \beta$  is a production rule, then everything in FIRST ( $\beta$ ) is FOLLOW ( $B$ ) except  $\epsilon$ .
- If  $(A \rightarrow \alpha B$  is a production rule) or  $(A \rightarrow \alpha B \beta$  is a production rule and  $\epsilon$  is in FIRST ( $\beta$ )) then everything in FOLLOW ( $A$ ) is in FOLLOW ( $B$ ).
- Apply these rules until nothing more can be added to any FOLLOW set.

## Bottom-up Parsing Techniques

A bottom-up parser creates the parse tree of the given input string from leaves towards the root. A bottom-up parser tries to find the right most derivation of the given input in the reverse order.

- (i)  $S \Rightarrow \dots \Rightarrow \omega$  (the right most derivation of  $\omega$ ).
- (ii)  $\leftarrow$  (the bottom-up parser finds the right most derivation in the reverse order).

Bottom-up parsing is also known as shift reduce parsing.

## Shift Reduce Parsing

A shift reduce parser tries to reduce the given input string into the starting symbol. At each reduction step, a substring of the input matching to the right side of a production rule is replaced by the non-terminal at the left side of that production rule.

**Note** If the substring is chosen correctly, the right most derivation of that string is created in the reverse order.

## Handle

Informally, a handle of a string is a substring that matches the right side of a production rule, but not every substring that matches the right side of a production rule is handle. A handle of a right sentential form  $\gamma$  ( $\equiv \alpha \beta \omega$ ) is a production rule  $A \rightarrow \beta$  and a position of  $\gamma$ , where the string  $\beta$  may be found and replaced by  $A$  to produce the previous right sentential form in a right most derivation of  $\gamma$ .

$$S \Rightarrow \alpha A \omega \Rightarrow \alpha \beta \omega \text{, where } \omega \text{ is a string of terminals.}$$

If the grammar is unambiguous, then every right sentential form of the grammar has exactly one handle. A right most derivation in reverse can be obtained by Handle-Pruning.

### Stack Implementation

*There are four possible actions of a shift reduce parser*

- Shift The next input symbol is shifted onto the top of the stack.
- Reduce Replace the handle on the top of the stack by the non-terminal.
- Accept Successful completion of parsing.
- Error Parser discovers a syntax error and calls an error recovery routine.
- Initial stack just contains only the end-marker \$.
- The end of the input string is marked by the end-marker \$.

### Conflicts During Shift Reduce Parsing

There are CFGs for which shift reduce parser can't be used. Stack contents and the next input symbol may not decide action.

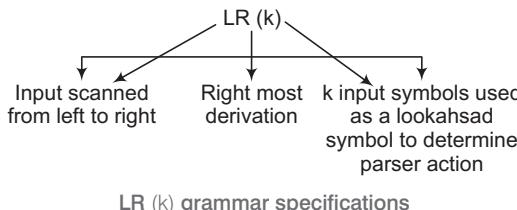
#### Shift/Reduce Conflict

Whether make a shift operation or a reduction.

#### Reduce/Reduce Conflict

The parser can't decide which of several reductions to make.

- If a shift reduce parser can't be used for a grammar, that grammar is called as non-LR ( $k$ ) grammar.
- An ambiguous grammar can never be a LR grammar.



LR (k) grammar specifications

### Types of Shift Reduce Parsing

*There are two main categories of shift reduce parsers*

#### Operator Precedence Parser

Simple, but supports only a small class of grammars.

#### LR Parser

Covers wide range of grammars

- SLR Simple LR parser
- CLR Most general LR parser (canonical LR)
- LALR Intermediate LR parser (look-ahead LR)

SLR, CLR and LALR work in same way, but their parsing tables are different.

## Operator Precedence Parsing

Operator grammar small but an important class of grammars. We may have an efficient operator precedence parser (a shift reduce parser) for an operator grammar.

In an operator grammar, no production rule can have  $\epsilon$  at the right side and two adjacent non-terminals at the right side.

### Precedence Relations

In operator precedence parsing, we define three disjoint precedence relations between certain pair of terminals.

- $a < b$ ,  $b$  has higher precedence than  $a$ .
- $a = b$ ,  $b$  has same precedence as  $a$ .
- $a > b$ ,  $b$  has lower precedence than  $a$ .

The determination of correct precedence relation between terminals are based on the traditional notions of associativity and precedence of operators.

- Scan the string from left end until the first  $>$  is encountered.
- Then, scan the backwards (to the left) ove.
- Using precedence relations to find handle any '=' until a ' $<$ ' is encountered.
- The handle contains everything to the first ' $>$ ' and to the right of the ' $<$ ' is encountered.

The handles thus obtained can be used to shift reduce a given string.

### Handling Unary Minus

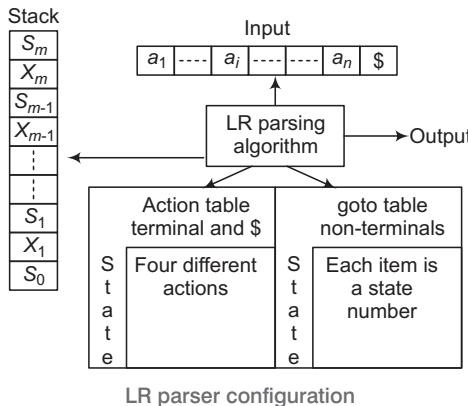
- Operator precedence parsing can't handle the unary minus, when we also use the binary minus in our grammar.
- The best approach to solve this problem is to let the lexical analyzer handle this problem, as the lexical analyzer will return two different operators for the unary minus and the binary minus.
- Lexical analyzer will need a look-ahead to distinguish the binary minus from the unary minus.

## LR parsing

LR parsing is most general non-back tracking shift reduce parsing. The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with predictive parsers.

LL (1) grammars ( $\subset$  LR (1) grammars

An LR parser can detect a syntactic error as soon as it is possible.



LR parser configuration

A configuration of a LR parsing is

$$(S_0 \ X_1 S_1 \dots X_m S_m, a_i a_{i+1} \dots a_n \$)$$

Stack                      Rest of input

- $S_m$  and  $a_i$  decides the parser action by consulting the parsing action table (initial stack contains just  $S_0$ ).
- A configuration of a LR parsing represents the right sentential form

$$X_1 \dots X_m \ a_i \ a_{i+1} \dots a_n \$$$

## LR Parser Actions

**Shift S** Shift the next input symbol and the state S onto the stack

$$(S_0 \ X_1 S_1 \dots X_m S_m, a_i a_{i+1} \dots a_n \$) \rightarrow$$

$$(S_0 \ X_1 S_1 \dots X_m S_m, a_i s, a_{i+1} \dots a_n \$)$$

**Reduce A  $\rightarrow \beta$**  (or  $m$ , where  $n$  is a production number)

Pop  $2|\beta| (=r)$  items from the stack; let us assume that  $\beta = Y_1, Y_2, \dots, Y_r$

Then, push A and S, where  $S = \text{goto } [S_{m-r}, A]$

$$(S_0 \ X_1 S_1 \dots X_m S_m, a_i a_{i+1} \dots a_n \$) \rightarrow$$

$$(S_0 \ X_1 S_1 \dots X_{m-r} S_{m-r} As, a_i \dots a_n \$)$$

**Accept** Parsing successfully completed.

**Error** Parser detected an error (an empty entry in the action table).

# Syntax Directed Translation

Grammar symbols are associated with attributes to associate information with the programming language constructs that they represent. Values of these attributes are evaluated by the semantic rules associated with the production rules.

*Evaluation of the semantic rules are as follows*

- (i) May generate intermediate codes
- (ii) May put information into the symbol table
- (iii) May perform type checking
- (iv) May issue error messages
- (v) May perform some other activities
- An attribute may hold a string, a number, a memory location, a complex record etc.
- Evaluation of a semantic rule defines the value of an attribute, but a semantic rule may also have some side effects such as printing a value.

e.g.,

Production	Semantic Rule	Program Fragment
$L \rightarrow E \text{ return}$	Print (E.val)	Print (val [top - 1])
$E \rightarrow E' + T$	$E.\text{val} = E'.\text{val} + T.\text{val}$	$\text{val [n top]} = \text{val [top - 2]}$ $+ \text{val [top]}$
$E \rightarrow T$	$E.\text{val} = T.\text{val}$	
$T \rightarrow T'^* F$	$T.\text{val} = T'.\text{val} * F.\text{val}$	$\text{val [n top]} = \text{val [top - 2]}$ $* \text{val [top]}$
$T \rightarrow F$	$T.\text{val} = F.\text{val}$	
$F \rightarrow (E)$	$F.\text{val} = E.\text{val}$	$\text{val [n top]} = \text{val [top - 1]}$
$F \rightarrow \text{digit}$	$F.\text{val} = \text{digit.lexval}$	$\text{val [top]} = \text{digit.lexval}$

Syntax directed translation table

- Symbols  $E$ ,  $T$  and  $F$  are associated with an attribute value.
- The token digit has an attribute lexval (it is assumed that it is evaluated by the lexical analyzer).
- The program fragment above represents the implementation of the semantic rule for a bottom-up parser.
- At each shift of digit, we also push digit.lexval into val\_stack.

- At all other shifts, we do not put anything into val\_stack because other terminals donot have attributes (but we increment the stack pointer for val\_stack).
- The above model is suited for a desk calculator, where the purpose is to evaluate and to generate code.

### Intermediate Code Generation

Intermediate codes are machine independent codes, but they are close to machine instructions. The given program in a source language is converted to an equivalent program in an intermediate language by the intermediate code generator.

The designer of the compiler decides the intermediate language.

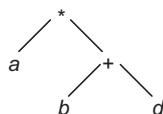
- Syntax trees can be used as an intermediate language.
- Postfix notations, three address code (quadruples) can be used as an intermediate language.

### Syntax Tree

Syntax tree is a variant of the parse tree, where each leaf represents an operand and each interior node represent an operator.

Production	Semantic Rule
$E \rightarrow E \text{ op } E$	$E.\text{val} = \text{NODE}(\text{op}, E1.\text{val}, E2.\text{val})$
$E \rightarrow (E)$	$E.\text{val} = E1.\text{val}$
$E \rightarrow -E$	$E.\text{val} = \text{Unary}(-, E1.\text{val})$
$E \rightarrow \text{id}$	$E.\text{val} = (\text{LEAF}(\text{id}))$

A sentence  $a * (b + d)$  would have the following syntax tree



Example of a syntax tree

Production	Semantic Rule	Program Fragment
$E \rightarrow E1 \text{ op } E2$	$E.\text{code} = E1.\text{code}    E2.\text{code} \mid \text{op}$	Print op
$E \rightarrow (E)$	$E.\text{code} = E1.\text{code}$	
$E \rightarrow \text{id}$	$E.\text{code} = \text{id}$	Print id

## Three-Address Code

When each statement contains three addresses (two for operands and one for result), Most general kind of three-address code is

$$x = y \text{ op } z$$

Where  $x$ ,  $y$  and  $z$  are names, constants or compiler generated temporaries and  $\text{op}$  is any operator.

But we can also use the following notation for quadruples (much better notation because it looks like a machine code instruction)

$$\text{op } y, z, x$$

Apply operator  $\text{op}$  to  $y$  and  $z$  and store the result in  $x$ .

## Representation of Three-Address Codes

Three-address code can be represented in various forms i.e., quadruples, triples and indirect triples. These forms are demonstrated by way of examples below.

e.g.,  $A = -B * (C + D)$

*Three address code is as follow*

$$T1 = -B$$

$$T2 = C + D$$

$$T3 = T1 * T2$$

$$A = T3$$

### Quadruples

	Operator	Operand 1	Operand 2	Result
(1)	-	B		T1
(2)	+	C	D	T2
(3)	*	T1	T2	T3
(4)	=	A	T3	

### Triples

	Operator	Operand 1	Operand 2
(1)	-	B	
(2)	+	C	D
(3)	*	(1)	(2)
(4)	=	A	(3)

Indirect triple	
	Statement
(0)	(56)
(1)	(57)
(2)	(58)
(3)	(59)

	Operator	Operand 1	Operand 2
(56)	-	B	
(57)	+	C	D
(58)	*	(56)	(57)
(59)	=	A	(58)

## Symbol Tables

Symbol table is a data structure meant to collect information about names appearing in the source program. It keeps track about the scope/binding information about names. Each entry in the symbol table has a pair of the form (name and information).

- Information consists of attributes (e.g., type, location) depending on the language.
- Whenever a name is encountered, it is checked in the symbol table to see, if already occurs. If not, a new entry is created.

In some cases, the symbol table record is created by the lexical analyzer as soon as the name is encountered in the input and the attribute of the name are entered when the declarations are processed.

If same name can be used to denote different program elements in the same block, the symbol table record is created only when the name's semantic role is discovered.

### Operations on Symbol Table

- Determine whether a given name is in the table.
- Add a new name to the table.
- Access information associated to a given name.
- Add a new information for a given name.
- Delete a name (or a group of names) from the table.

## Implementation of Symbol Table

- Each entry in a symbol table can be implemented as a record that consists of several field.
- The entries in symbol table records are not uniform and depend on the program element identified by the name.
- Some information about the name may be kept outside of the symbol table record and/or some fields of the record may be left vacant for the reason of uniformity. A pointer to this information may be stored in the record.
- The name may be stored in symbol table record it self or it can be stored in a separate array of characters and a pointer to it in the symbol table.
- The information about runtime storage location, to be used at the time of code generation, is to kept in the symbol table.
- Various approaches to symbol table organisation e.g., linear list, search tree and hash table.

### Linear List

- It is the simplest approach in symbol table organisation.
- The new names are added to the table in order they arrive.
- A name is searched for its existence linearly.

### Key Points

- ◆ The average number of comparisons required in a linear list are proportional to  $0.5 * (n + 1)$  where  $n$  = number of entries in the table.
- ◆ It takes less space but more access time.
- ◆ The time for adding/searching a name in search tree is proportional to  $(m + n) \log_2 n$ .
- ◆ The hash function maps the name into an integer value between 0 and  $k-1$  and uses it as an index in the hash table to search the list of the table records that are built on that hash index.

### Search Tree

- It is more efficient than linear lists.
- We provide two links left and right, which point to record in the search tree.
- A new name is added at a proper location in the tree such that it can be accessed alphabetically.
- For any node A1 in the tree, all nodes accessible by the following left link precede node A1 alphabetically.
- Similarly, for any node A1 in the tree, all names accessible by the following right link succeed A1 alphabetically.

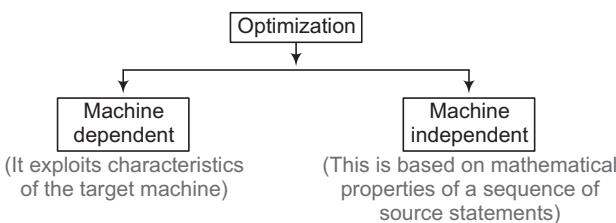
### Hash Table

- A hash table is a table of  $k$ -pointers from 0 to  $k-1$  that point to the symbol table and record within the symbol table.
- To search a value, we find out the hash value of the name by applying suitable hash function.
- To add a non-existent name, we create a record for that name and insert it at the head of the list.

### Code Optimization

It refers to obtain a more efficient code. While performing optimization, there are two points, which we need to focus on.

- We must ensure that the transformed program is semantically equivalent to the original program.
- The improvement of the program efficiency must be achieved without changing the algorithms which are used in the program.



## Techniques used in Optimization

The following are the techniques used in optimization as

### Common Sub-expression Elimination

An expression need not be evaluated, if it was previously computed and values of variables in this expression have not changed, since the earlier computations.

$$\begin{aligned} \text{e.g., } &a = b * c; \\ &d = b * c + x - y; \end{aligned}$$

We can eliminate the 2nd evaluation of  $b * c$  from this code if none of intervening statements has changed its value. So, we can rewrite the above code as

$$\begin{aligned} T1 &= b * c; \\ a &= T1; \\ d &= T1 + x - y; \end{aligned}$$

## Compile Time Evaluation

We can improve the execution efficiency of a program by shifting execution time actions to compile time.

e.g.,  $A = 2 * (22.0 / 7.0) * r$

Here, we can perform the computation  $2 * (22.0 / 7.0)$  at compile time itself. This is known as folding.

## Using Constant Propagation

$$\begin{array}{l} x = 12.4 \\ y = x/2.3 \end{array} \xrightarrow[\text{code}]{\text{Optimised}} y = 12.4 / 2.3$$

In the above example y can directly be computed at compile time.

If a variable is assigned a constant value and is used in an expression without being assigned other value to it, we can evaluate some portion of the expression using the constant value.

## Using Variable Propagation

If a variable is assigned to another variable, we use one in place of another. This will be useful to carry out other optimisation that were otherwise not possible.

e.g.,

$$\boxed{\begin{array}{l} c = a * b; \\ x = a; \end{array}} \longrightarrow \boxed{d = x * b}$$

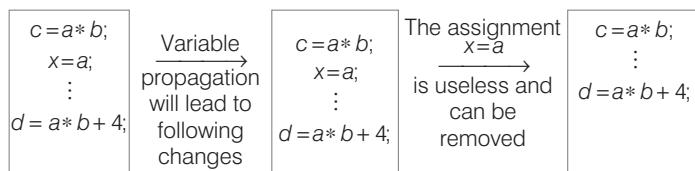
(Variable propagation technique)

In the above example, if we replace x by a then,  $a * b$  and  $x * b$  will be identified as common sub-expressions.

## Using Dead Code Elimination

If the value contained in a variable at that point is not used anywhere in the program subsequently, the variable is said to be dead at that place. Variable propagation often leads to making assignment statement into dead code.

e.g.,



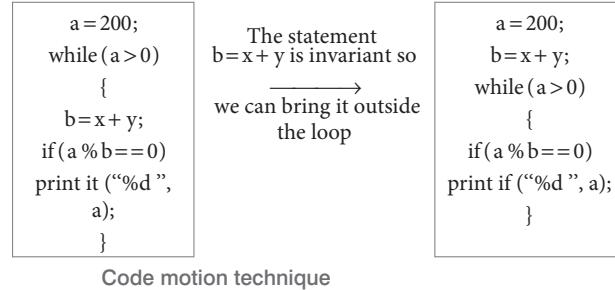
(Dead code elimination technique)

### Using Code Motion

Evaluation of expression is moved from one part of the program to another in such a way that it is evaluated lesser frequently.

We can bring the loop-invariant statement out of the loops.

e.g.,

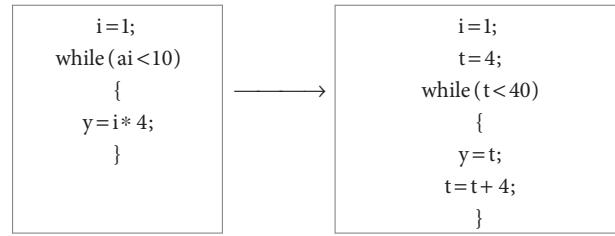


### Using Induction Variable and Strength Reduction

An induction variable may be defined as an integer scalar variable which is used in loop for the following kind of assignments i.e.,  $i = i + \text{constant}$ .

Strength reduction refers to the replacing the high strength operator by a low strength operator. Strength reduction used on induction variables to achieve a more efficient code.

e.g.,



### Use of Algebraic Identities

Certain computations that look different to the compiler and are not identified as common sub-expressions are actually same.

An expression  $BopC$  will usually be treated as being different to  $CopB$ .

But for certain operations (like addition and multiplication), they will produce the same result.

We can achieve further optimization by treating them as common sub-expressions for such operations.

## Run-time Administration

It refers how do we allocate the space for the generated target code and the data object of our source programs? The places of the data objects that can be determined to compile time will be allocated statically. But the places for some of data objects will be allocated at run-time.

The allocation and deallocation of the data objects is managed by the run-time support package. Run-time support package is loaded together with the generated target code. The structure of the run-time support package depends on the semantics of the programming language (especially the semantics of procedures in that language).

## Procedure Activation

Each activation of a procedure is called as activation of that procedure. An execution of a procedure starts at the beginning of the procedure body. When the procedure is completed, it returns the control to the point immediately after the place, where that procedure is called. Each execution of the procedure is called as its activation.

- Lifetime of an activation of that procedure (including the other procedures called by that procedure).
- If  $a$  and  $b$  are procedure activations, then their lifetimes are either non-overlapping or are nested.
- If a procedure is recursive, a new activation can begin before an earlier activation of the same procedure has ended.

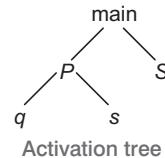
## Activation Tree

We can create a tree (known as activation tree) to show the way control enters and leaves activations. In an activation tree

- Each node represents an activation of a procedure.
- The root represents the activation of the main program.
- The node  $a$  is a parent of the node  $b$  if and only if the control flows from  $a$  to  $b$ .
- The node  $a$  is left to the node  $b$  if the lifetime of  $a$  occurs before the lifetime of  $b$ .

e.g.,

Program main;	enter main
Procedure s;	enter p
begin....end;	enter q
Procedure p;	exit q
Procedure q;	enter s
begin .... end;	exit s
begin q; s; end;	exit p
begin p; s; end;	enter s
	exit s
	exit main



## Control Stack

The flow of the control in a program corresponds to a depth first traversal of the activation tree that

- (i) Starts at the root.
- (ii) Visits a node before its children.
- (iii) Recursively visits children at each node and a left-to-right order.
- A stack called control stack can be used to keep track of live procedure activations.
  - (i) An activation record is pushed onto the control stack as the activation starts.
  - (ii) That activation record is popped when that activation ends.
- When node  $n$  is at the top of the control stack, the stack contains the nodes along the path from  $n$  to the root.

## Variable Scope

The scope rules of the language determine, which declaration of a name applies when the name appears in the program.

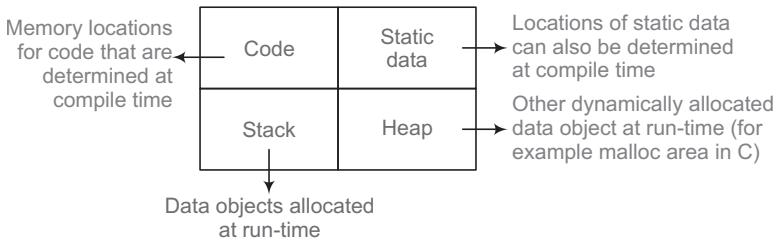
An occurrence of a variable is local, if that occurrence is in the same procedure in which that name is declared and the variable is non-local, if it is declared outside of that procedure.

e.g.,

```
procedure q;
var a: real;
procedure r;
var b: integer;
begin b = 1;a=2; end;
begin ... end;
```

⇒ Variable b is local  
to procedure r  
and variable a is  
non-local to  
procedure r.

## Storage Organisation

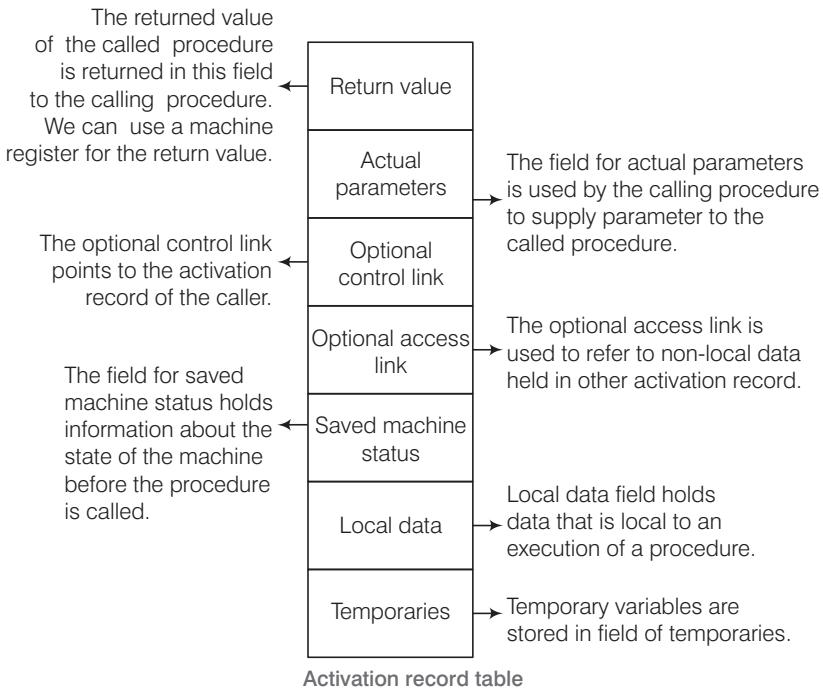


## Activation Record

Information needed by a single execution of a procedure is managed using a contiguous block of storage called activation record. When a procedure is entered, an activation record is allocated and it is deallocated when that procedure exits. Size of each field can be determined at compile time, although actual location of the activation record is determined at run-time.

## Key Points

- ♦ If a procedure has a local variable and its size depends on a parameter, its size is determined at run-time.
- ♦ Some part of the activation record of a procedure is created by that procedure immediately after that procedure is entered and some part is created by the caller of that procedure before that procedure is entered.



## Displays

The array of pointer which is used to access activation records is known as displays.

*For each level, there will be an array entry as given below*

- Current activation record at level 1.
- Current activation record at level 2.
- Current activation record at level 3.

## Error Detection and Recovery

- The parser should be able to give an error free message which should be meaningful.
- The parser should also be capable of recovering from the errors and it should be able enough to continue the parsing with the rest of the input.

## Error Recovery Techniques

### Panic Mode Error Recovery

Skipping the input symbols until a synchronising token is found.

### Phrase Level Error Recovery

Each empty entry in the parsing table is filled with a pointer to a specific error routine to take care that error case.

## Error Productions

If we have a good idea of the common errors that might be encountered, we can augment the grammar with productions that generate erroneous constructs. When an error production is used by the parser, we can generate appropriate error diagnostics. Since, it is almost impossible to know all the errors that can be made by the programmers, this method is not practical.

## Global Correction

Ideally, we would like a compiler to make as few changes as possible in processing incorrect inputs. We have to globally analyze the input to find the error. This is an expensive method and it is not in practice.

# 8

# Software Engineering and Information System

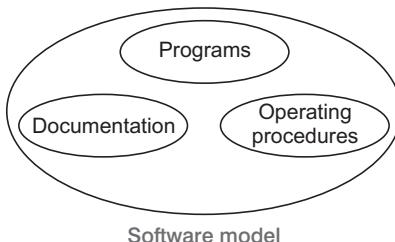
---

## Software Engineering

Software engineering is defined as a discipline whose aim is the production of quality software, software that is delivered on time within budget and that satisfies its requirements.

## Software

A software consists of programs, documentation of any fact of the program and the procedures used to setup and operate the software system. Basically, program is a combination of source code and objects code. Operating procedures consists of instructions to setup and use the software system and instructions so that they can react to the system failure.



$$\text{Software} = \text{Programs} + \text{Documentation} + \text{Operating procedures}$$

### Product and Process

- What is delivered to the customer is called a **product**. It may include source code, specification document manuals, documentation, etc.
- Process is the way in which we produce software. It is the collection of activities that leads to a part of a product.

## Measures, Metrics and Measurement

- A measure provides a quantitative indication of the extent, dimension, size, capacity, efficiency, productivity or reliability of some attributes of a product or process.
- Measurement is the act of evaluating a measure.
- A metric is a quantitative measure of the degree to which a system component or process possesses a given attribute.

## Software Development Life Cycle (SDLC)

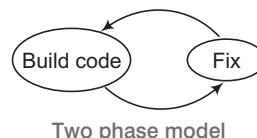
A software life cycle is often called as a **software development life cycle** and it is a particular abstraction that represents a software life cycle. The period of time that starts when a software product is conceived and ends when the product is no longer available for use.

*The software development life cycle typically includes following phases*

1. Requirement phase
2. Design phase
3. Implementation phase
4. Test phase
5. Installation and check out phase
6. Operation and maintenance phase

## Build and Fix Model

Sometimes a product is constructed without specification. Basically, this is an **adhoc approach** and not well defined. It is a simple two phase model. The first phase is to write code and the next phase is to fix it. Fixing in this context may be error correction or addition of further functionality.

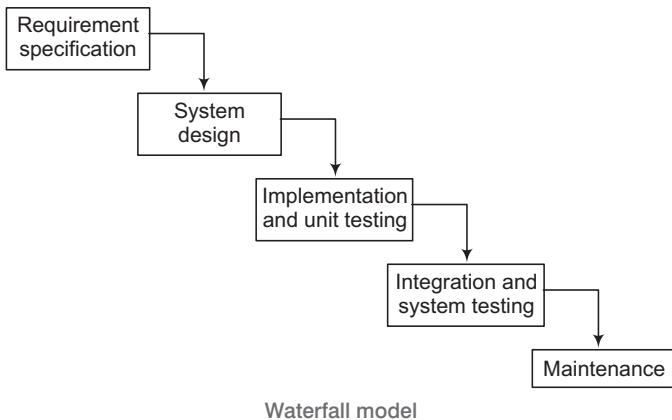


Two phase model

## Waterfall Model

The waterfall model is a sequential software development model in which development is seen as following steadily downwards like a waterfall through several phases. This model maintains that one should move to next phase only when its preceding phase is complete and perfect.

Phases of development in the waterfall model are thus discrete and there is no jumping back and forth or overlapping between them.



## **Problems of Waterfall Model**

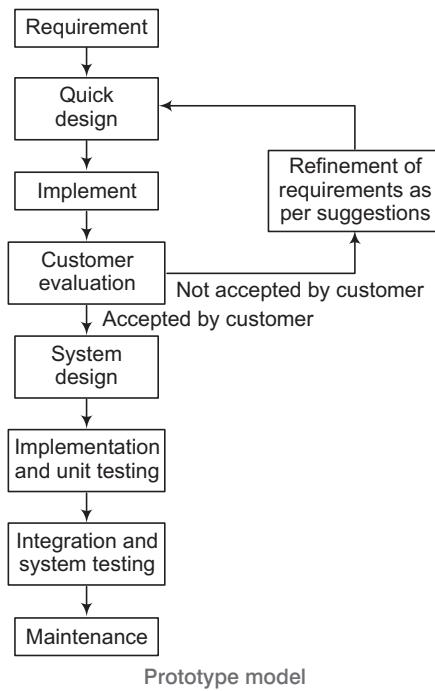
- It is difficult to define all requirements at the beginning of a project.
- This model is not suitable for accomodating any change.
- A working version of the system is not seen until late in the project's life, thus delaying the discovery of serious errors.

## **Prototype Model**

Here, we first develop a working prototype of the software instead of developing the actual software.

## **Advantages of Prototyping Model**

- Users are actively involved in the development.
- It provides better system to users, as users have natural tendency to change their mind in specifying requirements.
- Since, in this methodology, a working model of the system is provided to the users so that they can get a better understanding of the system being developed.
- Errors can be detected much earlier as the system is made side by side.
- Quick user feedback is available leading to better solution.



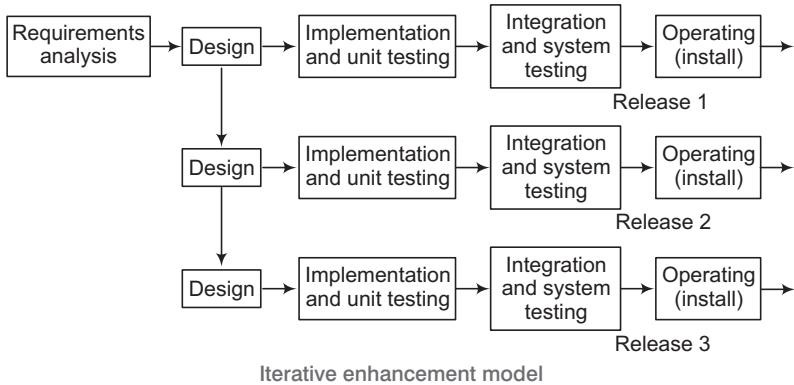
## Disadvantages of Prototyping Model

- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- This model leads to implementing and then repairing way of building system.

## Iterative Enhancement Life Cycle Model

This model counters the limitation of the waterfall model and combines the benefits of both prototyping and the waterfall models. The basic idea is that the software should be developed in increments, where each increment adds some functional capability to the system until the full system is implemented. At each step extensions and design modifications can be made.

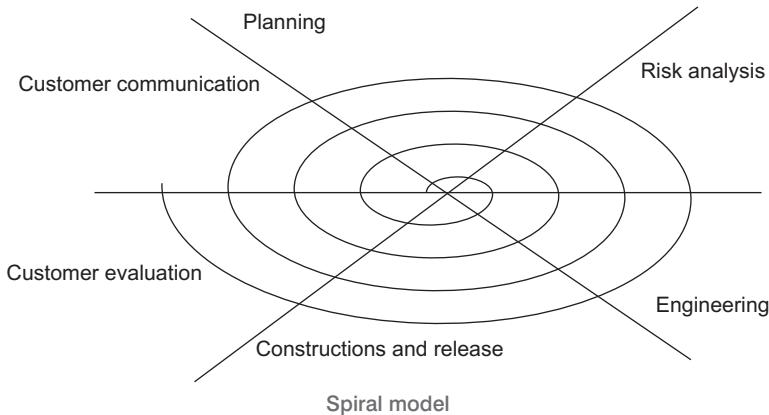
An advantage of this approach is that it can result in better testing, since testing each increment is likely to be easier than testing entire system like in the waterfall model.



## **Spiral Model**

This is the recent model that has been proposed by Barry Boehm. The spiral model has many cycles. The radial dimension represents the cumulative cost incurred in accomplishing the steps done so far and the angular dimension represents the progress made in completing each cycle of spiral. The spiral model is divided into a number of framework activities, also called task regions.

*Typically, there are between three to six task regions as shown in figure.*



- **Customer Communication** Task required to establish effective communication between developer and customer.

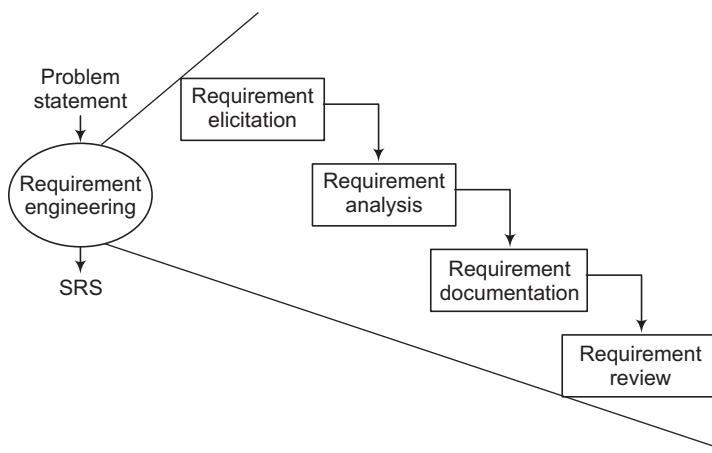
- **Planning** Task required to define resources, timeliness and other project related information.
- **Risk Analysis** Task required to access both technical and management risks.
- **Engineering** Task required to build one or more representations of the application.
- **Construction and Release** Task required to construct, test, install and provide user support (e.g., documentation and training).
- **Customer Evaluation** Task required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

## Software Requirements

Software requirement is a process to understand the exact requirements of the customer and to document them properly. The hardest part of building a software system is deciding precisely what to build.

## Analysis and Specifications

- Requirements describe the 'what' of a system not the 'how'.
- Requirements engineering produces one large document, contains a description of what the system will do.



Various steps of analysis and specifications

## **Requirement Elicitation**

This is also known as gathering of requirements. Here, requirements are identified with the help of customer and existing system processes, if available.

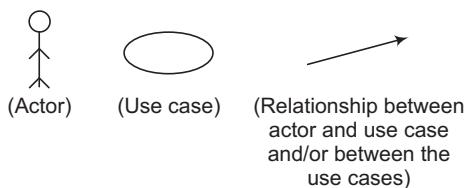
*There are following methods that can be used in requirement elicitation*

- **Interviews** First step to understand the problem statement of customer i.e., meeting with customer.
- **Brainstorming Sessions** It is a kind of group discussion which may lead to new ideas quickly and help to promote creative thinking.
- **Facilitated Application Specification Technique (FAST)** The objective of FAST approach is to bridge the expectation gap, a difference between what developers think they are supposed to build and what customers think they are going to get.
- **The Use Case Approach** This approach uses a combination of text and pictures in order to improve the understanding of requirements.

Use case diagrams are graphical representations that may be decomposed into further levels of abstraction.

### **Design of the Use Case Approach**

*The following components are used for the design of the use case approach*



Actor or external agent lies outside the system model but interacts with it in some way. An actor may be a person, machine or an information system.

Use case is initiated by a user with a particular goal in mind and competes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver services that satisfies the goal.

## **Requirement Analysis**

Requirement analysis allows the system analyst to refine the software allocation and build conceptual models of the data, functional and behavioural domains that will be treated by software.

### Data Modeling

- Define data objects attributes and relationships.
- We use *E-R* diagrams for this purpose.

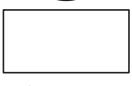
### Behavioural Modeling

- Finding out different states of the system.
- Specifying events that cause the system to change state.
- We use state transition diagrams for behavioural modeling.

### Function Modeling

- Identify functions that transform data objects .
- Indicate how data flows through system.
- Represent producers and consumers of data.

### Symbols used in a DFD

Symbol	Name	Function
	Data flow	Used to connect processes to each other to sources or sinks, the arrowhead indicates direction of data flow.
	Process	Performs some transformations of input data to yield output data.
	Source or sink (external entity)	A source of system inputs or sink of system outputs.
	Data store	A repository of data, the arrowheads indicate net inputs and the net outputs to store.

### Requirement Documentation

Requirement document is the way to represent requirements in a consistent format. Requirement document is called SRS *i.e.*, Software Requirements Specification. The SRS should be correct, unambiguous, complete, consistent, verifiable, modifiable, traceable.

### Key Points

- ♦ For function modeling, we use Data Flow Diagrams (DFDs). DFD shows the flow of data through a system.
- ♦ The requirement review process is carried out to improve the quality of the SRS.
- ♦ The requirement review process may also be called as requirements verification.
- ♦ For maximum benefits, review and verification should not be treated as a discrete activity to be done only at the end of preparation of SRS. It should be treated as continuous activity that is incorporated into the elicitation, analysis and documentation.

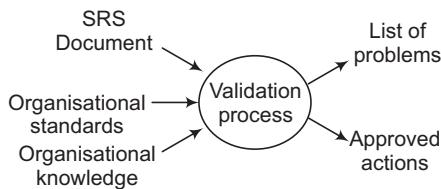
## Requirement Validation

After the completions of SRS document, we may like to check the documents for

- Completeness and consistency
- Conformance to standards
- Requirements conflicts
- Technical errors
- Ambiguous requirements

### Validation Process with Inputs and Outputs

The objective of requirements validation is to certify that the SRS is an acceptable document of the system to be implemented.



## Software Risk Management

Risk is a problem that could cause some loss or threaten the success of the project but which has not happened yet. Risk management means dealing with a concern before it becomes a crisis.

### Typical Software Risks

*These can be classified as*

#### Requirement Issue

Many project face uncertainty around the products's requirements. If we do not control requirements related risk factors, we might either build the wrong product or build the right product badly.

#### Management Issue

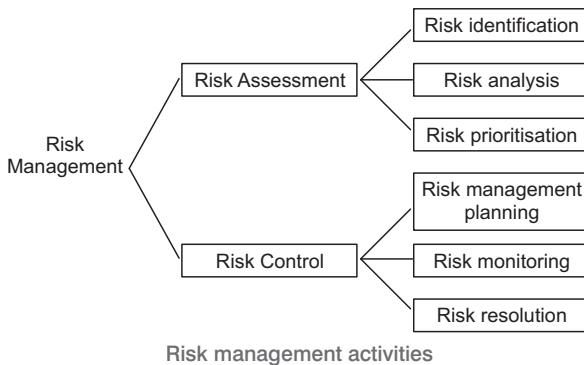
Project manager usually write the risk management plan and most people do not wish to air the weaknesses in public. If we do not confront such touch issues, we should not be surprised, if they bite us at some points.

## Lack of Knowledge

The rapid rate of change in technologies and the increasing change of skilled staff, means that our project teams may not have the skills we need to be successful.

## Risk Management Activities

*Risk management activities consists of two key areas that are risk assessment and risk control.*



### Risk Assessment

It is the process of examining a project and identifying areas of potential risk. Risk identification can be facilitated with the help of a check list of common risk areas of software projects. Risk analysis involves examining how project outcomes might change with modification of risk input variables. Risk prioritisation helps the project focus on its most severe risks of assessing the risk exposure.

### Risk Control

It is the process of managing risks to achieve the desired outcomes. Risk management planning produces a plan for dealing with each significant risk. We should also monitor the project as development progress by periodically re-evaluating the risks and their probabilities. Risk resolution is the execution of the plans for dealing with each risk.

#### Risk Estimated Method

Priority of each risk can be computed as

$$P = r * s$$

where,  $r$  = The likelihood of a risk coming true

$s$  = The consequences of the problems associated with that risk.

# Software Design

Software design translates requirements into a blueprint for constructing the software. The purpose of design phase is to produce a solution to a problem given in Software Requirement Specification (SRS) document. Here, we generate a Software Design Document (SDD).

## Design Concepts

*It basically describes the three things given below*

- (i) Separation of function/data structure detail from a conceptual representation of the software (abstraction and refinement).
- (ii) Deciding criteria to partition software into individual components (i.e., modularity).
- (iii) Criteria to define the technical quality of a software design.

## Abstraction

Abstraction can be defined as the process of looking what is essential to your perspective or understanding without looking at the complex or low level details.

### Types of Abstraction

*Two types of abstractions are available in modern programming language.*

- **Data Abstraction** i.e., a named collection of data that describes a data object.
- **Procedural Abstraction** i.e., a named sequence of instructions with a specific and limited function.

Consider a sentence of two words i.e., **open door**. In this sentence, **open** could be an example of procedural abstraction and **door** could be an example of data abstraction.

*Open implies a long sequence of procedural steps such as*

- Walk to the door
- Turn knob and pull door
- Reach out and grasp knob
- Step away from moving door

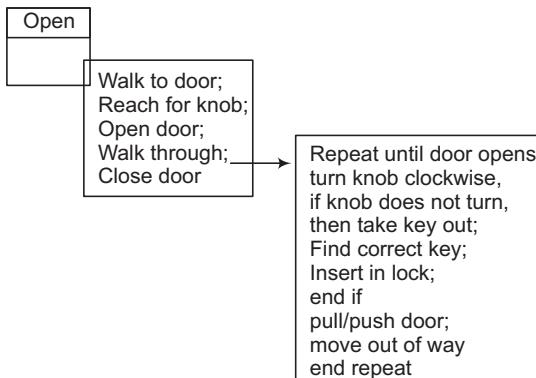
*The data abstraction for door would encompass a set of attributes that describes the door such as*

- Door type
- Model number
- Weight and dimensions
- Manufacturer
- Swing direction

## Refinement/Stepwise Refinement

A hierarchy is developed by decomposing a macroscopic statement of function (a procedural abstraction) in a stepwise fashion until programming language statements are reached.

**Top-down Decomposition** In each step of the refinement, one or several instructions of the given program are decomposed into more detailed instructions. This successive decomposition or refinement of specifications terminates when all instructions are expressed in terms of underlying computer or programming language.



Refinement- A process of elaboration

## Modularity

Modularity is the process, wherein software is divided into separately named and addressable components, often called modules, that are integrated to satisfy problem requirements. It uses the divide and conquer approach for solving a complex problem by breaking it or modularising it into smaller modules.

## Control Hierarchy

It represents the organisation of program components (modules) and implies a hierarchy of control. Does it represent procedural aspects of software such as sequence of processes, occurrence or order of decisions or repetitions, nor is it necessarily applicable to all architectural styles.

### **Horizontal Partitioning**

A module that controls other module is called as manager module, while a module controlled by another is known as subordinate of the manager module. Horizontal partitioning defines separate branches of the hierarchy for each major program function. The simplest approach to horizontal partitioning defines three partitions; input, data transformations or processing and output.

### **Vertical Partitioning**

It is also called factoring. It suggests that control (decision-making) and work should be distributed top-down in the program structure.

---

#### **Key Points**

- ♦ Top-level modules should perform control functions and do little processing work.
  - ♦ Modules that reside lower in the structure should be the workers, performing all input, computations and output tasks.
- 

### **Functional Independence**

Functional independence is achieved by developing modules with single-minded functionality (functionally cohesive) and an aversion to excessive interactive with other modules (weak coupling). Functional independence is a key to good design and good design is key to software quality.

*Functional independence is measured using two qualitative criteria*

- (i) Cohesion
- (ii) Coupling

#### **Cohesion**

Cohesion is a measure of the degree to which the elements of a module are functionally related.

---

#### **Key Points**

- ♦ A strongly cohesive module implements functionality that is related to one feature of the solution and requires little or no interaction with other modules.
  - ♦ Cohesion is equal to strength of relations within modules.
- 

Here, an important design objective is to maximize the module cohesion and minimize the module coupling.

### Strength of Different Types of Cohesion

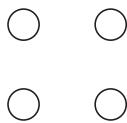
Types of Cohesion	Best (High)
Functional cohesion	
Sequential cohesion	
Communicational cohesion	
Procedural cohesion	
Temporal cohesion	
Logical cohesion	
Coincidental cohesion	
	Worst (Low)

### Coupling

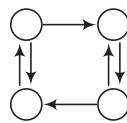
Coupling is the measure of the degree of interdependence or interconnection between modules.

#### Key Points

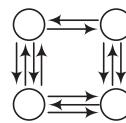
- Two modules with high coupling are strongly interconnected and thus dependent on each other.
- Two modules with low coupling are less or not dependent on one another.



(a) Uncoupled



(b) Loosely coupled



(c) Highly coupled

### Strength of Different Types of Coupling

Types of Coupling	Best (High/Strong)
Data coupling	
Stamp coupling	
Control coupling	
External coupling	
Common coupling	
Content coupling	
	Worst (Low/Loose)

# **Testing**

Testing refers to a defect detection mechanism and its purpose is to find errors. Testing is a process of executing a program with intent of finding an error.

## **Verification**

It is a process of determining whether or not the product of a given phase of software development fulfill the requirements established during the previous data. Verification is all about; are we building the product right.

Verification means software product should meet user expectations, checking that users expectations are satisfied. *We can do this with the help of below*

- Functional testing
- Intergration and interface testing
- System testing
- Acceptance criteria
- Regression testing

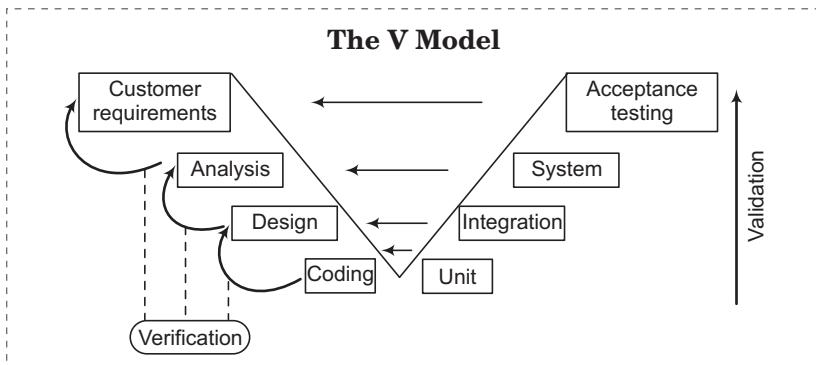
## **Key Points**

- ♦ A good test case is one that has a high probability of finding a yet undiscovered error.
- ♦ A successful test is one that uncovers a yet undiscovered error.

## **Validation**

Validation is a process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements. Validation is all about; are we building the right product. Validation ensures whether the software product is behaving according to its specification. *We can do this with the help of below*

- Technical reviews and inspections
- Buddy checks, peer reviews
- Root cause analysis
- Metric definition
- Certification demonstrations



## Testing Techniques

Testing is the process of execution of a program with the intent of finding errors.

*There are two types of testing techniques which are given below*

### **White Box Testing (Structural Testing)**

Testing based on the internal specification with knowledge of how system is constructed. In this testing approach, we have to analyse the code and use the knowledge about the program structure to derive test data.

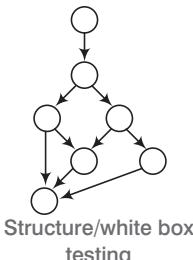
*White box testing techniques are as given below*

#### **Basic path testing**

- Flow graph notation
- Cyclomatic complexity
- Graph matrices

#### **Control structure testing**

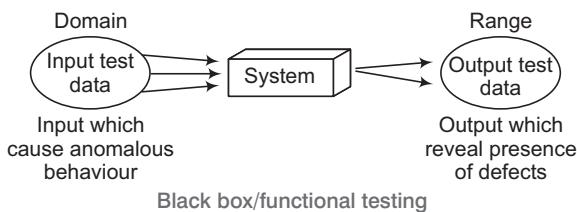
- Loop testing



Structure/white box testing

### **Black Box Testing (Functional Testing)**

Testing based on the external specification without the knowledge of how system is constructed. In this approach, testers need not to have explicit knowledge of internal workings of the item being tested.



*Black box testing techniques are given below*

- (i) Equivalence partition
- (ii) Boundary value analysis
- (iii) Robustness testing

*Here, some instances are given below, where white box testing is better than black box testing*

- (i) Logical error
- (ii) Memory overflow undetected
- (iii) Topological error

*Some instances, where black box testing is better than white box testing.*

- (i) Functional requirements not met
- (ii) Integration errors
- (iii) Incorrect parameters passed between functions

## **Types of Testing / Level of Testing**

There are mainly three levels of testing. A software product goes through these levels of testing.

- 1. Unit testing
- 2. Integration testing
- 3. System testing

## **Unit Testing**

Unit testing is the process of taking a module (the smallest unit of software design) and running it in isolation from the rest of the software product by using prepared test cases and comparing the actual results with the results predicted by the specification and design of the module.

*There are number of reasons in support of unit testing than testing the entire product.*

- The size of a single module is small enough that we can locate an error fairly easily.
- Confusing interactions of multiple errors in widely different parts of the software are eliminated.

Unit Testing is white box oriented.

## Integration Testing

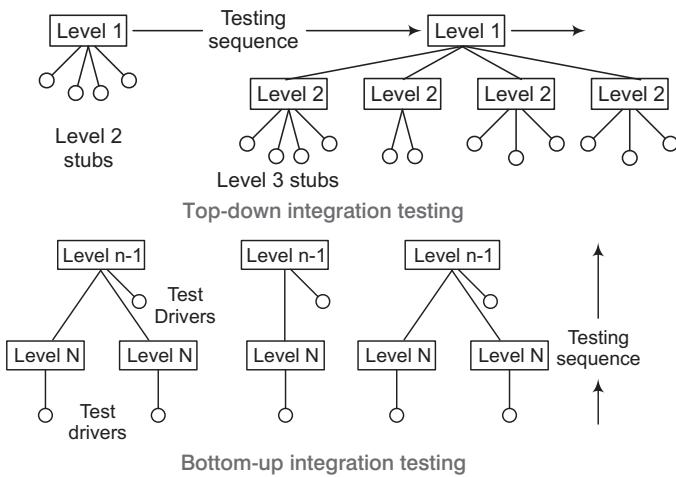
This integration testing is used to test the integration and consistency of an integrated subsystem. Integration testing is applied incrementally as modules are assembled into larger subsystems. It is done using a combination of both black box and white box testing techniques.

### Testing Applied to Integrated Part of the System

*Testing is applied to subsystems, which are assembled in either*

- **Top-down** Assembles down from the highest level modules replacing the lower level modules test stubs.
- **Bottom-up** Assembles from the lowest level modules replacing the higher level by test drives.

A stub is a simplified program or dummy module designed to provide the response that would be provided by the real sub-element.



## System Testing

If focuses on complete integrated system to evaluate compliance with specified requirements. It is basically used for performance, stress and security testing. System testing includes the following testing techniques

### Acceptance Testing

This testing is performed before to handover the system to the customer.

Here, the customer may write the test criteria and request the developer to execute them or the developer can write the criteria and take the customer's approval.

Acceptance testing focuses on complete integrated system to evaluate fitness of use. It is done from the users perspective.

### **Alpha and Beta Testing**

Alpha testing is done at developer's site by customer. In alpha testing, developers are present and environment is controlled environments. Beta testing is done at one or more customer's site by end users/customers. In beta testing, we face live situation and here developer may or may not be present. Beta testing usually comes in picture when the number of users are millions.

### **Performance Testing**

This testing is concerned with assessing the time and memory aspects of the system. Performance testing may be concerned with checking that the operation completes within the fixed deadline and only a fixed size of memory is allocated.

### **Regression Testing**

Regression testing is applied after changes have been made to the system. The operation of the new version is compared with the previous version to see, if there are any unexpected results.

## **Software Measurement**

In software measurement, software is measured to find its efficiency and accuracy. The functionality of a software is totally dependent on the measurement process, if a software gives correct output in the software measurement process, then it can be sure that software will give meaningful and valuable information within a defined time.

*Measurement in the physical world can be categorised into two ways.*

#### **Direct measures of software engineering process**

- It includes cost and effort applied      • It also includes
  - (i) Lines of Code (LOC) produced
  - (ii) Execution speed                        (iii) Memory size
  - (iv) Defects reported over some period of time

#### **Indirect measures of the product**

- It includes
  - (i) Functionality                                (ii) Quality
  - (iii) Complexity                                (iv) Efficiency
  - (v) Reliability and maintainability etc.

## Software Metrics (Direct Measure)

Software metrics can be defined as the continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information together with the use of those techniques to improve that process and its products.

### Categories of Software Metrics

*There are three main categories of software metrics which are given below*

#### Product metrics

These metrics describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability etc.

#### Project metrics

These metrics describe the project characteristics and execution. *Example of project metrices are*

- Number of software developers.
- Staffing pattern over the life cycle of the software.
- Cost and schedule.
- Productivity.

#### Process metrics

These metrics describe the effectiveness and quality of the process that produce the software product. *Examples of process metrices are*

- Effort required in the process.
- Time to produce the product.
- Maturity of the process.
- Number of defects found during testing.
- Effectiveness of defect removal during developments.

### Some Common Software Metrics

**LOC** (Lines of Code) It specifies size of a software. LOC can also be used to given a more precise characterisation to the common motions of small, large or very large projects e.g.,

Size	Small	Medium	Large	Very large
LOC	<2 k	2 k – 8k	8k – 32 k	> 32 k

**Person months** (Man-months) It specifies the effort needed or spent on a project.

## Normalized Metrics

If measures are normalized, it is possible to create software metrics that enable comparison to broader organisational averages.

### Size Oriented Metrics

These metrics are derived by normalizing quality and/or productive measures by considering the size of software that has been produced.

Productivity = LOC/Person month ; Quality = Defects/LOC ; Cost = Price/LOC

### Function Oriented Metrics

These metrics use a measure of the functionality delivered by the application as a normalization value. Functionality is measured indirectly using other direct measures.

#### Function Point (FP) Metrics

This metric can easily be used to estimate the size of a software product directly from the problem specification. The conceptual idea underlying the function point metric is that the size of a software product is directly dependent on the number of different functions and features it supports.

*Function point is computed in two steps*

(i) The first step is to compute the Unadjusted Function Point (UFP).

$UFP = (\text{Number of inputs}) * 4 + (\text{Number of output}) * 5 + (\text{Number of inquiries}) * 4 + (\text{Number of files}) * 10 + (\text{Number of interfaces}) * 10$

Once the UFP is computed the Technical Complexity Factor (TCF) is computed next. The TCF refines the UFP measure by considering 14 other factors such as high transaction rates, throughput and response time requirements etc. Each of these 14 factors is assigned a value from 0 (not present or no influence) to 6 (strong influence). The resulting numbers are summed, yielding the total degree of influence.

$TCF = (0.65 + 0.01 * D1)$  as  $D1$  can vary from 0 to 70, the TCF can vary from 0.65 to 1.35. Finally,  $FP = UFP * TCF$

#### Key Words Related to Normalized Matrices

- **Number of inputs** Each data item by user is counted.
- **Number of outputs** A set of related data items is counted as one output. The output considered refer to reports printed, screen outputs, error messages produced etc.
- **Number of inquiries** Number of distinct interactive queries which can be made by the users. These inquiries are user commands which require specific action by the system.
- **Number of files** Each logical file is counted. A logical file means group of logically related data. Thus, logical files can be data structures or physical files.

## Indirect Measures

*Indirect measures focus on software reliability and software quality*

### Software Reliability

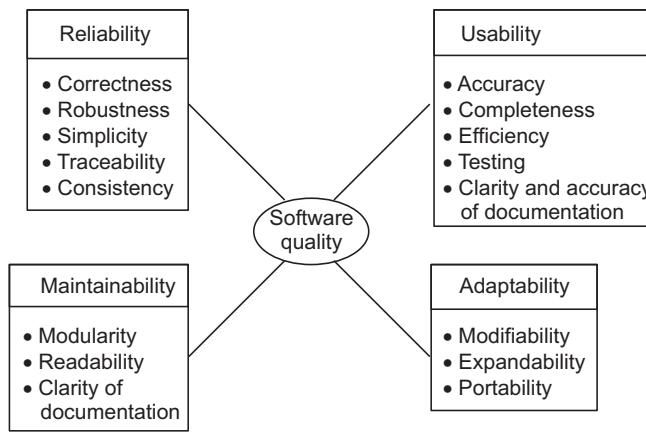
It is defined as the ability of a system or component to perform its required function under stated conditions for a specified period of time. Before the deployment of software products, testing verification and validation are necessary steps which can ensure better software reliability in the product.

### Software Quality

One objective of software engineering is to produce good quality maintainable software in time and within budget. If a product is meeting its requirements, we may say, it is a good quality product.

### Key Points

- ♦ Software reliability is the probability of failure free software operation for a specified period of time in a specified environment.
- ♦ When we deal with software quality, a list of attributes (reliability, usability, maintainability and adaptability) is required to be defined that are appropriate for software.



Software quality model

## **Project Estimation Techniques**

The estimation of various project parameters is a basic project planning activity. The important project parameter that are estimated include project size, effort required to develop the software; project duration and cost.

*There are three broad categories of estimation techniques*

### **Empirical Estimation Techniques**

These techniques are based on making an educated guess of the project parameters.

### **Heuristic Techniques**

This technique assumes that the relationships among the different project parameters can be modelled using suitable mathematical expression.

### **Analytical Estimation Techniques**

This technique derive the required result starting with certain basic assumptions regarding the project. Thus, unlike emperical and heuristic approach, this approach do have a scientific basis. Example of analytical technique is Halstead's software science.

### **Halstead Software Science**

Halstead used a set of primitive measures, which can be derived once the design phase is completed and the code is generated.

*There measures are as follows*

$n_1$  = Number of distinct operators in a program

$n_2$  = Number of distinct operands in a program

$N_1$  = Total number of operators

$N_2$  = Total number of operands

Program length ( $N$ ) can be calculated by using below equation

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

Program volume ( $V$ ) can be calculated by using equation given below, volume of information's unit is in bits

$$V = N \log_2 (n_1 + n_2)$$

Volume ration ( $L$ ) must be less than 1 and can be calculated as

$$L = \frac{2}{n_1} * \frac{n_2}{N_2}$$

$$\text{Program difficulty level } (D) = \left( \frac{n_1}{2} \right) * \left( \frac{N_2}{n_2} \right)$$

$$\text{Effort } (E) = D * V$$

# COCOMO Model

It is a heuristic estimation technique. This is also known as constructive cost model. Software development project can be classified into one of the three categories based on the development complexity *i.e.*, organic, semidetached and embedded.

## Organic

We can consider a development project to be of organic type, if the project deals with developing a well understood application program. The size of the development team is reasonably small and the team members are experienced in developing similar types of project.

## Semidetached

If the development team consists of a mixture of experienced and unexperienced staff. Team members may have limited experience on related system but may be unfamiliar with some aspects of the system being developed.

## Embedded

We apply this approach, if the software being developed is strongly coupled to complex hardware or if stringent regulations on the operational procedure exist.

According to Boehm, software cost estimation should be done through three stages *i.e.*, basic COCOMO, intermediate COCOMO and complete COCOMO.

## Basic COCOMO Model

Barry Boehm introduced a hierarchy of software estimation models named Constructive Cost Model. Basic equations of COCOMO model are

$$\text{Efforts in person-months } (E) = a (\text{KLOC})^b$$

$$\text{Development time in months } (D) = c (E)^d$$

Where  $a, b, c$  and  $d$  are coefficients that have fixed values for different classes of projects.

**Table for Coefficients in COCOMO Model**

Project	$a$	$b$	$c$	$d$
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

# Cyclomatic Complexity

It is a software metric that provides a quantitative measure of the logical complexity of a program. It defines the independent paths in a program which provides us with an upper bound for the number of tests that must be conducted to ensure that all statements are executed atleast once.

Cyclomatic complexity  $V(G)$ , for a flow graph  $G$  is defined as

$$V(G) = E - N + 2$$

where,  $N$  = Number of nodes

$E$  = Number of edges

Cyclomatic complexity  $V(G)$ , for a flow graph  $G$  is defined as  $V(G) = P + 1$  where,  $P$  = Predicate nodes

Let's take an example graph as shown in figure

Path 1

$\rightarrow 1 - 9$

Path 2

$\rightarrow 1 - 3 - 8 - 1 - 9$

Path 3

$\rightarrow 1 - 2 - 4 - 7 - 8 - 1 - 11$

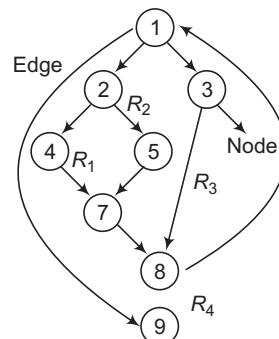
Path 4

$\rightarrow 1 - 2 - 5 - 7 - 8 - 1 - 11$

Here, we can see that each new path introduces a new edge.

The flow graph has four regions ( $R_1, R_2, R_3, R_4$ )

$$V(G) = 10 \text{ edges} - 8 \text{ nodes} + 2 = 4$$



Graph for cyclomatic complexity

## Key Points

- Independent path is any path through the program that introduces atleast one new condition or new set of processing statements.
- The number of regions of the flow graph correspond to cyclomatic complexity.

## Evolution of Quality System

- **Product Inspection** This method gave a way to quality control.
- **Quality Control** It aims for correcting the causes of errors and not just rejecting the defective products.
- **Quality Assurance** If an organisation's process are good and are followed rigorously, the product are bound to be of good quality.
- **Total Quality Management (TQM)** The process followed by an organisation must be continuously improved through process measurements.



### ISO 9000 Certification

It specifies a set of guidelines for repeatable and high quality product development. ISO 9000 standard mainly addresses operational and organisational aspects such as responsibilities, reporting etc.

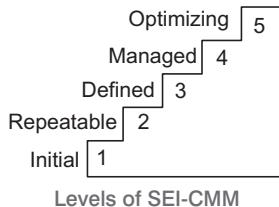
*It is a series of three standards ISO 9001, ISO 9002 and ISO 9003.*

- ISO 9001 standard applies to the organisations engaged in design, development, production and servicing of goods.
- ISO 9002 standard applies to those organisations which do not design products but are only involved in production.
- ISO 9003 standard applies to organisations involved only in installation and testing of the products.

## Software Engineering Institute Capability Maturity Model (SEI-CMM)

It is a strategy for improving the software process, irrespective of the actual life cycle model used. CMM used to judge the maturity of the software processes of an organisation and to identify the key practices that are required to increase the maturity of the process.

CMM is organized into five maturity levels as shown below



### **Description of Levels of CMM (Maturity Levels)**

CMM Level	Focus	Key Process Area
Initial	Complete people	—
Repeatable	Project management	Software project planning, software configuration management.
Defined	Definition of processes	Process definition, training program, pre reviews.
Managed	Product and process quality	Qualitative process metrics, software quality managements.
Optimizing	Continuous process improvement	Defect prevention, process change management, technology change management.

## **Six Sigma ( $\sigma$ )**

It is a disciplined, data driven approach to eliminate defects in any process from manufacturing to transactional and from product to service. A six sigma defect is defined as any system behaviour that is not as per customer specifications.

### **Key Points**

- ♦ Total number of six sigma opportunities equals to total number of chances for a defect
- ♦ Six sigma can be used to improve every facet of business from production to human resources to order entry to technical support.

*The six sigma sub-methodologies are given below*

**DMAIC** It is (Define, Measure, Analyse, Improve, Control) an improvement for existing processes falling below specification and thus looking for incremental improvement.

**DMADV** It is (Define, Measure, Analyse, Design, Verify) an improvement system used to develop new processes or products at six sigma quality level.

# **Software Maintenance**

It is an activity in which program is modified after it has been put into use. In this, usually it is not preferred to apply major software changes to system's architecture.

## **Types of Software Maintenance**

### **Corrective**

It focuses to rectify the bugs observed while the system is in use.

### **Adaptive**

This kind of maintenance comes into the picture when customer need the product to run on new platform, on new operating system or when they need the product to be interfaced with new hardware or software.

### **Perfective**

This focuses on to enhance the performance of the system.

## **Software Re-engineering**

It means re-structuring or rewriting part or all of the system. The software re-engineering is needed for the application which require frequency maintenance. Advantages of software re-engineering are reduced risk and reduced cost.

# 9

# Web Technology

---

## HTML (Hyper Text Markup Language)

HTML is a method where ordinary text can be converted into hyper text. It is basic tools for designing a web page.

### Hyper Text

A way of creating multimedia documents, also a method for providing links within the documents.

### Markup Language

A method for embedding special tags that describe the structure as well as behaviour of a document.

### HTML Tags

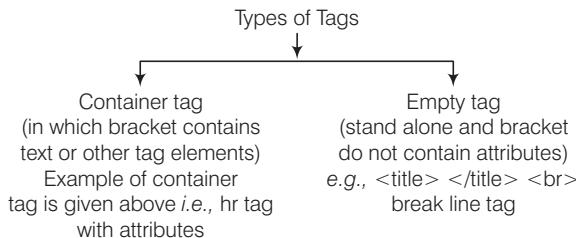
*Developing a web page is concerned with*

- Contents of the page
- Appearance of the page

The appearance of the page is coded in HTML language, using HTML tags. An attribute of a tag is additional information that is included inside the start tag.

The diagram shows a code snippet: <hr>. A circle highlights the 'hr' tag. Three arrows point from three attributes to their respective labels: 'size= 5' points to 'size', 'width= 50%' points to 'width', and 'align= right' points to 'align'. Below these labels is the word 'attributes'.

Tag used for horizontal line



### Basic Structure of HTML Page

The HTML page can be saved as. html or . htm extension.

```

<HTML>
<HEAD>
    //Head elements are to be written here
</HEAD>
< BODY >
    //Body elements are to be written here
</BODY>
</HTML>
  
```

Here, the `<>` is called Starting Tag and `</>` is called Closing Tag. The page will start with HTML tag, which will declare that , this page is an HTML page.

*The HTML page is divided into two major parts.*

1. Head element
2. Body element

## Head Element

Head element basically contains those tags which are useful to store some information about the current HTML page but they are not visible to web explorer. Some of head tags are also used to handle some events in HTML documents. *Head elements are described as follows*

### **<title> Tag**

This tag is used to define title of the web page. The text that is written in between the opening and closing tags of title tag is visible at the topmost bar of web page.

```
<title> First web page </ title >
```

### **<base > Tag**

Base tag defines the default address or default target for all links on a page.

```

< html >
< head >
    < base href = "http://www.abc.com/myfolder" />
    < base target = "- blank" />
  
```

```

</head>
<body>
    <a href = "mypage.html"> Click Here </a>
</body>
</html>

```

Here, `<a>` `</a>` defines the link to another page. This tag is also known as Anchor Tag. When you click on Click Here, it will redirect you to the `www.abc.com/myfolder/mypage.html`.

The href element in the base tag defines a base URL for all relative URLs in the page and the target element in base tag defines the location, where to open all links in a page.

*Attributes of target are as follow*

Target = `_blank` means open the link page to the new window.

Target = `_parent` means open the link into the same \* frame.

Target = `_self` means open the link into the same \* frame.

Target = `_top` means open the link into the top \* frame.

\* frame specified by `frame_name`

### **<link> Tag**

The `<link>` tag defines the relationship between a document and an external resource.

```

<head>
    <link rel = "stylesheet" type = "text/css" href = "abc.css"/>
</head>

```

Here, the link tag is used to define that an external css file, i.e., abc.css has to be used in this html page.

### **Key Points**

---

*The attributes in link element are as follows*

- ♦ **Rel** Specifies the relationship between the current document and the linked document.
  - ♦ **Type** Specifies the MIME type of the linked document.
  - ♦ **Href** Specifies the location of the linked document.
  - ♦ **Rev** Specifies the relationship between the linked document and the current document.
  - ♦ **Charset** Specifies the character encoding of the linked document.
- 

### **<Meta> Tag**

This provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parseable. The `<meta>` tag

always goes inside the <head>. The Metadata can be used by browsers (how to display content or reload page), search engines (keywords) or other web services.

```
< head >
< meta name = "description" content = "free tutorials"/>
< meta name = "keywords" content = "HTML, CSS, XML"/>
< meta name = "author" content = "Mohan"/>
< meta http_equiv = "Content_Type" content = "text.html;
charset =ISO-8859-1"/>
</head>
```

Here,

- **Name** Provides a name for the information in the content attribute.
- **Http\_equiv** Provides an HTTP header for the information in the content attribute values for this, attributes are content type, content-style-type, expires, refresh, set\_cookie.
- **Content** Specifies the content of the meta information.
- **Scheme** Specifies a scheme to be used to interpret the values of the content attribute.

### **<script> Tag**

It is used to define client-side script, such as Java script. This script element either contains scripting statements or it points to an external script file through the src attribute.

```
<head>
    <script type = "text/java script">
        document.write ("Hello World!")
    </script>
</head>
```

### **<style> Tag**

Style tag is used to define style information for an HTML document. Inside the style element we specify how HTML elements should render in a browser.

```
<html>
<head>
<style type = "text/css">
h1 {color : red}
p {color : blue}
</style>
</head>
```

```
<body>
  <h 1> Header 1 </h 1>
  <p > A paragraph. </p>
</ body>
</html1>
```

### **<body> Tag**

The body tag contains all those text that is to be displayed in web tag. The tag inside the body tag are used to format the text written and images to be displayed in web page. It also helps to align the various objects in web page.

### **Key Points**

- ♦ The only possible value is “text/css” for this tag.
- ♦ The style element always goes inside the head sections.
- ♦ The tag inside the body tag are used to format the text written and images to be displayed in web page.
- ♦ <body> Tag also helps to align the various objects in web page.

## **Text Formatting Tags**

- **<b> Tag** b stands for Bold tag. The text written between <b> and </b> tag will be displayed as bold letters.
- **<i> Tag** i stands for italic format means the text written between <i> and </i> will be displayed as italic format.
- **< big > Tag** This tag helps the text to be displayed in bigger size.
- **< small> Tag** This tag renders a smaller text.
- **<u> Tag** The text written between <u> and </u> will be displayed as underlined text.
- **<q> Tag** This tag defines the short quotation. The text will be quoted, if we use this tags.

```
<html1>
<body>
<q> Here comes a short quotation </q>
</body>
</html1>
```

**Output** “Here comes a short quotation”

- **<strong> Tag** This tag is used to display the text in bold format.

- **<sub>** Tag This tag defines subscript text.

e.g.,

H <sub> 2 </sub> O

Output H<sub>2</sub>O

- **<sup>** Tag This tag is used to display superscripted text.

e.g.,

4<sup>th</sup>Edition

Output 4<sup>th</sup> Edition

- **Headings <h<sub>1</sub>> ...<h<sub>6</sub>> Tags** The <h<sub>1</sub>> to <h<sub>6</sub>> tags are used to define HTML headings. <h<sub>1</sub>> defines the largest and <h<sub>6</sub>> defines the smallest heading.

- **<font>** Tag This tag is used to describe the font style of a text.

e.g.,

< font color = "red" size = "16" face = "arial" > This is my  
font tag test </font>

The color attribute defines the color of the text. The value in the color attribute can be given by directly by name or by specifying a particular hexadecimal 6 digit number like color = "#FF0EE2" or by RGB combination like color = "rgb (50, 50, 50)" size attribute defines the size of the text and the face attribute defines the font type of the text.

- **<abbr>** Tag The <abbr> tag describes an abbreviated phrase.

### The <abbr> Tag

The <abbr title = "World Health Organisation">WHO </abbr> was founded in 1948.

Here, when we will move the mouse over WHO it will display

Output "World Health Organisation"

- **<blockquote>** Tag This tag defines a long quotation. A browser inserts white space before and after blockquote element. It also inserts margins for the blockquote element.

- **<del>** Tag The <del> tag is used to define the text that has been deleted from a document.

e.g., The leader of this class is <del> Rahul </del> Neha.

Output The leader of this class is Neha.

## Text Alignment Tags

There are some tags that are used to align text into left, center, right or justified position.

*These tags are described as follows*

(a) **<p> Tag** This tag is used to define a new paragraph. The attributes in paragraph tag is align values of which is right, center, justified or left.

```
<p align = "left" >
```

(b) **<br > Tag** This tag is used to start next line or line break . The text written after **<br>** tag will be stated from next line

e.g., This is my first web page <br> And I learned everything about HTML.

**Output** This is my first web page

And I learned everything about HTML.

**Note** that **<br>** tag has no closing element.

(c) **<pre> Tag** As we know that without **<br>** tag line break is not applied to web page.

Thus, for every alignment corresponding tag is to be defined. **<pre>** tag eliminates this over head. The text written between **<pre>** tag is displayed in web page in the same manner as it is written in coding.

e.g.,

```
<pre>  
Hi! my name is Rahul and  
I love web technology.  
</pre>
```

**Output**

```
Hi! my name is Rahul and  
I love web technology.
```

(d) **<center> Tag** This tag is used to align a text to the center. This means the text that is written between center tag will be displayed on the center of page.

## Lists

HTML supports three types of lists as follows

- Unordered List (Unnumbered or Bulleted List)
- Ordered List (Numbered List)
- Definition List

## Unordered List

This defines the element using bullets.

e.g.,

```
<body>
<ul>
<li> Coffee </li>
<li> Milk </li>
<li> Bread </li>
</ul>
</body>
```

**Output**

```
Coffee
Milk
Bread
```

## Ordered List

In ordered list, elements are defined in ordered manner i.e., numbered.

e.g.,

```
<ol>
<li> Oranges </li>
<li> Peaches </li>
<li> Grapes </li>
</ol>
```

**Output**

```
1. Oranges
2. Peaches
3. Grapes
```

## Definition List

A definition list is a list of items, with a description of items. The `<dl>` tag defines a definition list . The `<dl>` tag is used in conjunction with `<dt>` (defines the item in the list) and `<dd>` (describes the item in the list).

e.g.,

```
<dl>
<dt> CS </dt>
<dd> - Computer Science </dd>
<dt> IT</dt>
<dd>- Information Technology </dd>
</dl>
```

**Output**

```

CS
-Computer Science
IT
-Information Technology

```

**Tables**

Tables are very useful for presentation of tabular information. For this purpose, we use `<table>` tag.

e.g.,

```

<table border = "5" >
<caption><b> Student Record </b> </caption>
<tr>
<th> Roll No </th>
<th> Student Name </th>
</tr>
<tr>
<td> 001 </td>
<td> Ram </td>
</tr>
<tr>
<td> 002 </td>
<td> Shyam </td>
</tr>
</table>

```

**Output****Student Record**

Roll No	Student Name
001	Ram
002	Shyam

**Table Elements**

*Table elements are as follows*

`<table> </table>` Defines a table in HTML.

`<caption> </caption>` Defines caption for the title of the table. Attribute Align = Both can be used to position the caption below the table.

`<tr> </tr>` Specifies a table row within a table.

`<th> </th>` Defines a table header cell. By default the text in this cell is bold and centered.

`<td> </td>` Defines the table data cell. By default the text in this cell is aligned left and centered vertically.

Some of the table attributes are described below

- **bgcolor** This is used to define the background color in table.
- **border** This specifies width around the border.
- **cellpadding** Specifies the space between the cell wall and the cell content.
- **cellspacing** Specifies the space between cells.
- **width** Specifies the width of a table.
- **align** This attribute aligns the content of row in three manners i.e., left, right, center.
- **valign** This attribute aligns the content of a cell in three values for this are middle, top and bottom.
- **image** The `<img>` tag allows to display images on web page. Images are the binary representation of data in pixel form.

Inline image Image is inserted at a particular location “in a line” within a web page.

- **syntax** `<img src = "myfolder/abc.jpg" height = "50%" width = "50%" alt = "mypics" border = "3" align = "center">`

Here, the src attribute defines the path of the image. Height defines the height of the image that can be given in form of % or pixel value a height = “100”. Similarly, width attribute defines the width of the image.

## Key Points

- ◆ The alt attribute defines the alternate text that will be displayed, if the image is not visible.
- ◆ Border attribute defines the border width around the image.
- ◆ Align attribute defines the alignment of the image, values can be left, center, right, top, bottom.

## Link

Links are used to jump from one html page to another html page. `<a>` anchor tag is used to create link between two html pages.

e.g., `<a href = "myfolder/nextpage.html"> Click Here </a>` when, we will click on the “Click Here” it will redirect the current page to nextpage.html.

## HTML Form

Forms not only provide communication from user to server, but also provide powerful stuff and can add a lot of value to the web pages.

Forms are used to get inputs from the user. The user input is submitted to the server. A form is defined with `<form> ... </form>` tag.

*Form tag has three attributes as follows*

### Action Attribute

This attribute specifies where to send the form-data when a form is submitted or what action needs to be performed when form is submitted.

e.g., If we want a program “`\cgi\bin\comments.exe`” to be executed when a form is submitted, then we write

```
action = “/cgi/bin/comments. exe”
```

### Method Attribute

This attribute defines how the data will be submitted to the server to go to the next page. Two methods are there, get and post. In get method, the data of the form is appended in the URL of next page.

e.g., `<form method = “get/post” action = “a. exe”>`.

### Encrypt Attribute

This is used to inform the server the way to handle the encryption process.  
`encrypt = “application/x-www. form-urlencoded”`

## Elements of a Form

*The elements of a form can be of three types*

- Input box
- Selection list box or combo box
- Text area

### Input Box and Type Values

Input Types	Example
Check Box	<code>&lt; input type = “check box” name = CB&gt;</code>
Radio Button	<code>&lt; input type = “radio” name = RB&gt;</code>
Text Field	<code>&lt; input type = “text” name = TF&gt;</code>
Password	<code>&lt; input type = “password” name = Pwd&gt;</code>
Hidden Field	<code>&lt; input type = “Hidden” name = HF&gt;</code>
Button	<code>&lt; input type = “Button” name = Btn&gt;</code>
Submit Button	<code>&lt; input type = “Submit” name = SB&gt;</code>
Reset Button	<code>&lt; input type = “Reset” name = RB&gt;</code>

**Selection Box (Combo Box)** The `<select>` tag is used to create a select list (drop down list).

The `<option>` tag inside the select element define the available options in the list.

e.g.,

```
<select>
<option value = "volvo" > volvo </option>
<option value = "scoda" > scoda </option>
<option value = "Mercedes" > mercedes </option>
</select>
```

**Text Area** Multiline area in which user can type the text as input. It has three attributes as number of rows, number of columns in text area and the name of the variable.

e.g.,

```
<html>
<body>
<center> Enter your text
<text area Name = "Remarks" rows = 5 column = 50>
</text area>
</center>
</body>
</html>
```

## Frames

Frames allow us to divide a browser window into several independent parts. It is a way of displaying two or more pages at once.

*Frames could be applied for the following on the web page as given below*

- To display the log or a stationary information in one fixed portion of the page.
- For the table of contents in a page, where people can just click and move around the website without having to move constantly to the contents page.

### Different Frames Tag

Tag	Name	Description
<code>&lt;frame&gt;</code>	Frame definition	Defines a single frame in a frame set
<code>&lt;frame set&gt;</code>	Frame group definition	Container for frame elements
<code>&lt;iframe&gt;</code>	inline frame	Defines an inline frame

### Example for Cols

< frameset cols = 200, 400> : size is in pixel

cols specify the number of vertical windows or frames and values in them specify the width of the frame in frameset. We can provide values either in pixel or in percentage. For percentage we need to mention values with % sign. For pixel, we should not give any sign with the value as in above example.

### Example for Rows

Row specifies number of horizontal windows or frames and values in rows specify the height of the frame < frameset rows = 100, 40%, \*>

Frame window will have three frames, where the height of the first frame is of 100 pixels, the second is 40% of the main window and the height of the 3rd one will occupy the rest of the window space.

### Attributes of Frame Tag

Attributes	Description
1. Frames border (1/0)	Renders a 3-D border around the frame. The default (1) inserts a border, 0 displays no border.
2. Margin height (number /percentage)	Controls the margin height (in pixels or in percentage) for the frame.
3. Margin width (number / percentage)	controls the margin width.
4. Name = "Text"	Provides a target name for frames
5. No resize	Prevents the user from resizing the frame.
6. Scrolling (yes/no/auto)	Creates a scrolling frame.
7. SRC = "URL"	Displays the source file for the frame.

e.g.,

```
<frameset rows = "50%", "50%">
<frame set cols = "50%", "50%">
<frame src = A.html>
<frame src = B.html>
</frameset>
<frame cols = "50%", "50%">
<frame src = C.html>
<frame src = D.html>
</ frameset>
```

Output

A	B
C	D

# XML

XML stands for Extensible Markup Language. XML is designed to transport and store data but HTML is designed to display data.

XML tags are not predefined. You must define your own tags.

e.g.,

```
<?Xml version = ''1.0''
encoding = ''ISO-8859-1''?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Do not forget me this weekend !<body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1=Latin-1/West European character set).

The next line describes the root element of the document (like saying: “this document is a note”) The next 4 lines describe 4 child elements of the root (to, from, heading and body) and finally the last line defines the end of the root element.

## Key Points

- ♦ XML is used in many aspects of web development, often to simplify data storage and sharing.
- ♦ XML simplifies data sharing because in the real world, computer systems and databases contain data in incompatible formats.
- ♦ XML documents form a tree structure that starts at “the root” and branches to “the leaves”.

## Difference between XML and HTML

	XML	HTML
1.	XML is designed to describe data and to focus on what data is, XML uses a document type definition or XML schema to describe the data.	HTML is designed to display data and to focus on how data looks.
2.	XML tags are not predefined. We need to define our own tags.	The tags used to markup HTML documents and the structure of HTML documents both are predefined.
3.	In XML, data is stored in a separate XML file.	In HTML, data is stored inside the HTML.
4.	XML is about carrying information.	HTML is about displaying information.

## Applications of XML

- **Refined Search Results** With XML specific tags, search engines can give users more refined search results. A search seeks the term in the tags, rather than the entire documents.
- **EDI Transactions** XML allows data to be exchanged, regardless of the computing system.
- **File Converters** Many applications have been written to convert existing document into the XML standards.

## XML Syntax Rules

```
< ? XML version = "1.0" encoding = "ISO-8859-1"?>
<Student>
    <Name> RAM </Name >
    <Roll No> 04</Roll No>
</Student>
```

- **XML Declaration** The XML page is declared as `< ? XML version = "1.0" encoding = "ISO - 8859 - 1"?>`. The declaration must be included in XML page. It defines the XML version and the character encoding used in the documents.
- **Root Element** The XML root element is defined as a tag in which all the other tags are included in our example `<student>` is root tag.
- **Syntax rules of XML page are given below**
  - (i) All XML elements must have a closing tag.
  - (ii) XML tags are case sensitive.
  - (iii) XML tags must be properly nested.
  - (iv) XML documents must have a root element.
  - (v) Always quote the XML attribute values.

## XML Schema

XML schema is a definition language for describing the structure of XML document.

### Elements of XML Schema

- `<XS : element>` It is used to define elements.
- `<XS : attribute>` It is used to define individual attribute.
- `<XS : simple type>` Marks the beginning of type that has no attributes and only text contents.
- `<XS : choice>` Describes the set of subelements an element can contain
- `<XS : sequence>` Describes the set of subelements in a particular order.

## Valid XML Document

An XML document is said to be valid, if it has a Document Type Definition (DTD) or XML schema associated with it and if the document complies with it.

## Well Formed XML Document

XML document with correct syntax is called well formed XML document.

## DTD (Document Type Definition)

A document type definition defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be defined inside a XML document or an external reference can be declared.

- **Internal DTD** If the DTD is defined inside the XML document, it should be wrapped in a DOCTYPE definition with the following syntax.  
`<!DOCTYPE root element [element declarations]>`
- **External DTD** If the DTD is defined in an external file, it should be wrapped in a DOCTYPE definition with the following ‘syntax’:  
`<!DOCTYPE root element system “filename”>`

## Importance of DTD

1. With a DTD, an XML file carries a description of its own format.
2. With a DTD, Independent groups of people can agree to use a standard DTD for interchanging data.
3. DTD can be used to verify data.

## XML Parser

To use XML document in an application, we need to parse it. An XML parser read an XML document and separates it into start tags, attributes and end tags. *Two types of XML parser are*

1. DOM (Document Object Model)
2. SAX (Simple API for XML)

## DOM (Document Object Model)

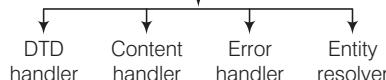
The DOM parser presents an XML document as a tree structure with the elements, attributes and text defined as nodes. The DOM provides an XML document into a collection of objects in an object model in a tree structure.

- Benefits of using DOM DOM makes easier to modify data, to remove it or to insert new one.
- Advantages of using DOM DOM is very useful when a document is small.
- Application DOM model is useful for interactive applications.

## SAX (Simple API for XML)

SAX works in serial access modes to parse XML documents. SAX is also called as an event driven protocol because it implements the technique to register the handler to invoke the callback methods whenever an event is generated.

### Handler classes in SAX



## Key Points

- ♦ SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the DOM.
- ♦ Where the DOM operates on the document as a whole, SAX parser operate on each piece of the XML document sequentially.
- ♦ DOM provides access to the information stored in our XML documents as a hierarchical object model.

## Sample XML Document

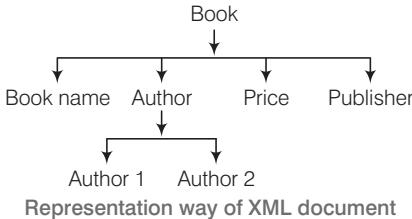
```

< ? XML Version = "1.0"?>
<Book>
  <Book Name> WebTech</BookName>
  <Author>
    <Author 1> RAM</Author 1>
    <Author 2> Shyam</Author 2>
  </Author>
  <Publisher> Arihant</Publisher>
  <Price> ₹ 500</Price>
</Book>
  
```

### SAX Approach for the Sample XML Document

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>▪ Start Element Book</li> <li>▪ End Element Book Name</li> <li>▪ Start Element Author 1</li> <li>▪ Start Element Author 2</li> <li>▪ End Element Author</li> <li>▪ End Element Publisher</li> <li>▪ End Element Price</li> </ul> | <ul style="list-style-type: none"> <li>▪ Start Element BookName</li> <li>▪ Start Element Author</li> <li>▪ End Element Author 1</li> <li>▪ End Element Author 2</li> <li>▪ Start Element Publisher</li> <li>▪ Start Element Price</li> <li>▪ End Element Book</li> </ul> |
|---|--|

### DOM Approach for the Sample Document XML



## Client/Server Model of Computing

In the client/server computing, all clients communicate with a server in the computer network. Both the clients and server are the nodes (communication points) on the network. The arrangement of the nodes in a network is called the network topology. In client/server computing model, *there are two computer which play an important role*

- (1) Client      (2) Server

### Client/Server Definition

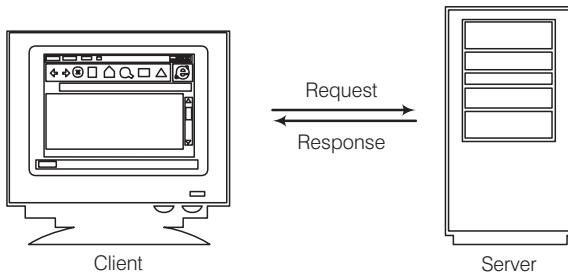
Clients are personal computers or workstations on which users run applications. A network architecture is that in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to disk drives (file servers), printers.

Client/server model is a form of distributed computing where one program (the client) communicates with another program (the server) for the purpose of exchanging information.

### Client/Server Architecture

A server is a computer system that selectively shares its resources; a client is a computer or computer program that initiates contact with a server in order to make use of a resource.

The Client/server architectures are sometimes called 2-tier architecture.



*The client's responsibilities are usually to*

- Handle the user interface.
- Translate the user's request into the desired protocol.
- Send the request of the server.
- Wait for the server's response.
- Translate the response into human readable format.
- Present the results to the user.

*The server's functions include*

- Listen for a client's query/request.
- Process that query/request.
- Return the results back to the client.

*A typical client/server interaction goes like this*

- The user runs client software to create a query.
- The client connects to the server.
- The client sends the query to the server.
- The server analyses the query.
- The server computes the results of the query.
- The server sends the results to the client.
- The client presents the results to the user.
- Repeat as necessary.

## Key Points

- ♦ Specific types of clients include web browsers, e-mail clients and the online chat clients.
- ♦ Specific types of servers include web servers, FTP servers, application servers, database servers, mail servers, file servers, print servers and terminal servers.
- ♦ Most web services are also types of servers.

# **Scripting Language**

A scripting language, script language or extension language is a programming language that allows control of one or more applications.

Scripts are distinct from the core code of the application, as they are usually written in a different language and are often created or atleast modified by the end user. Scripts are often interpreted from source code or byte code whereas the application is typically first compiled to native machine code.

## **Java Script**

- A scripting language *i.e.*, a light weight programming language.
- Designed to add interactivity to HTML pages.
- Usually embedded directly into HTML pages.
- Interpreted language (execute without preliminary compilation)

## **Key Points**

- ♦ Java and Java script are not same in both concept and design.
- ♦ Java is a powerful and much more complex programming language.

## **Advantages of Java Script**

*The advantages of Java script are as given below*

- (i) Java script gives HTML designers a programming tool.
- (ii) Java script can put dynamic content into an HTML page. Statement like `document.write ("< h1>" + name + "</h1>")` can write a variable text into an HTML page.
- (iii) Java script can react to events.
- (iv) Java script can read and write HTML elements.
- (v) Java script can be used to validate data before it is submitted to a server.
- (vi) Java script can be used to detect the visitors's browser.
- (vii) Java script can be used to create cookies.

## Loops in Java Script

Loops in Java script are used to execute the same block of code a specified number of times or while a specified condition is true.

e.g.,

```
<html>
<body>
<script type = "text/javascript">
    var i = 0;
    for (i = 0 ; i <= 10 ; i++)
    {
        document . write ("The number is;" + i);
        document . write ("<br>");
    }
</script>
</body>
</html>
```

In same way, we can use while loop for this.

e.g.,

```
for (variable in objects)
{
    code to be executed
}
```

The for in statement is used to loop/iterate through the elements of an array or through the properties of an object. Here, variable can be a named variable, an array element or a property of an object.

```
<html>
<body>
    <script type = "text/javascript">
var x;
    var mycars = new Array [ ] ;
    mycars [0] = "Audi"
    mycars [1] = "BMW"; for (x in mycars)
{
    document. write (mycars [x] + "<br/>");
}
</script>
</body>
</html>
```

**Output**

Audi  
BMW

## Java Script Functions

- To keep the browser free from executing a script when the page loads, we can put our script into a function.
- A function contains code that will be executed by an event or by a call to that function.
- Functions can be defined in both the <head> and in the <body> section of a document. However, to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

e.g.,

```
<html>
<head>
<script type = "text/javascript">
function displaymessage ( )
{
    alert ("All the Best");
}
</script>
</head>
<body>
<form>
<input type = "button" value = "click me" onclick =
"displaymessage ()">
</form>
</body>
</html>
```

## Key Points

- ♦ Java script is case sensitive In java script “my function” and “My Function” are not same.
- ♦ White space Java scripts ignores extra spaces. We can add white spaces to our script to make it more readable. *Following lines are same*  
name = “Arihant”;  
name = “Arihant”;
- ♦ **Breaking up a code line** We can breakup a code line within a text string by using a backslash“\".

e.g.,

```
document.write ("Hello World !");
However, we can't breakup a code line like this:
documents write \ ("Hello World !");
```

## HTML Comments to Handle Simple Browsers

Browsers that do not support Java script will display Java script as page content. To prevent them from doing this and as a part of the Java script standard, the HTML comment can be used to hide the Java script.

```
<script type = "text/javascript">
<! ...
    document . write ("Hello");
//...></script>
```

`Document . Write ()` It is a standard Java script command for writing output to a page or displaying text to the user.

## The return Statement

The return statement is used to specify the value that is returned from the function.

e.g.,

```
<html>
<head>
<script type = "text/javascript">
function product (a, b)
{
    var x = a * b;
    return x;
}
</script>
</head>
<body>
<script type = "text/javascript">
var a = product (2, 3);
document. write ("product is =" + a);
</script>
</body>
</html>
```

## Conditional Statements

We can refer the below example to understand this.

e.g.,

```
<html>
<head>
<body>
<script type = "text/javascript">
    var d = new Date ( );
    var time = d.getHours ( ) ;
    if (time < 10)
    {
        document. write ("< b> Good Morning </b>" );
    }
    else if (time > 10 && time < 16)
    {
        document. write ("< b> Good Day </b>" );
    }
    else
    {
        document. write ("< b> Hello world ! </b >" );
    }
</script>
</body>
</head>
</html>
```

## Java Script Events

Events are action that can be detected by Java script. Every element on a web page has certain events, which can trigger Java script functions.

We define the events in the HTML tags.

### Example of Events

- (a) A mouse click
- (b) A web page or an image loading
- (c) Mouse over a hot spot on the web page
- (d) Selecting an input box in an HTML form
- (e) Submitting an HTML form
- (f) A key stroke

Events are normally used in combination with functions and the function will not be executed before the event occurs.

### **Onload and Unload**

These events are triggered when the user enters or leaves the page. The onload event is often used to check the visitor's browser type and browser version and load the proper version of the web page based on the information.

### **Onsubmit**

This event is used to validate all form fields before submitting it.

```
<form method = "post" action = "abc. html"  
onsubmit = "return check From ()">
```

Function check Form ( ) returns either true or false. If it returns true, the form will be submitted, otherwise the submission will be cancelled.

### **Java Script switch Statement**

As we have seen we can use if...else conditional statements, loops, in same way we can use switch statement in Java script to select one of many blocks of code to be executed.

# 10

## Switching Theory and Computer Architecture

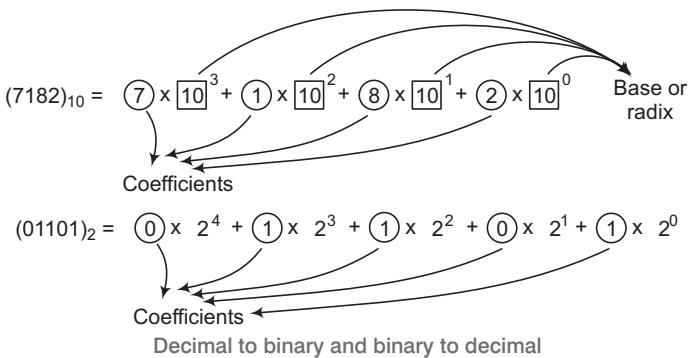
### Number System

This basically means the presentation of number with limited number of symbols. The decimal number system is said to be of base or radix 10 because it uses ten digits (0-9) to represent any number.

To distinguish between numbers of different bases, we enclose the value in parenthesis and with a subscript equal to the base is used.

e.g., If 10 is a decimal value, then  $(10)_{10}$

If 10 is a binary value, then  $(10)_2$



So, a number system with radix  $r$  will have  $r$  symbols and any number will be interpreted as

$$a_n r^n + a_{n-1} r^{n-1} + \dots + a_0 r^0$$

### Basic Number Systems

Number System	Symbols Used	Base or Radix
Binary	0,1	2
Octal	0,1,2,3,4,5,6,7	8
Decimal	0,1,2,3,4,5,6,7,8,9	10
Hexa decimal	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	16

### Key Points

- The symbols  $(a_n, a_{n-1}, \dots, a_0)$  should be one of the  $r$  symbols allowed in the number system (i.e., from 0 to  $r - 1$ ). In this,  $a_n$  is called as **most significant digit** and  $a_0$  is called the **least significant digit**.

## Number Base Conversions

### Decimal to Binary

We can understand this with the help of an example, converting decimal 31 to binary.

#### Decimal to Binary Conversion

Integer Quotient	Remainder	Coefficient
$\frac{31}{2} = 15$	1	$a_0 = 1$
$\frac{15}{2} = 7$	1	$a_1 = 1$
$\frac{7}{2} = 3$	1	$a_2 = 1$
$\frac{3}{2} = 1$	1	$a_3 = 1$
$\frac{1}{2} = 0$	1	$a_4 = 1$

The binary number will be written as  $(a_4 a_3 a_2 a_1 a_0)_2$  i.e.,  $(11111)_2$ .

### Process

The given number should be divided by 2 to give an integer quotient and a remainder. The quotient is again divided by 2 to give a new quotient and remainder. The process is continued, until the integer quotient becomes 0. The coefficient of the desired binary number are obtained from the remainder as shown in the table.

e.g.,

For fractional part

Convert  $(0.1245)_{10}$  to binary upto 6 bits.

### Process

First  $(0.1245)_{10}$  is multiplied by 2 to give an integer and a fraction. The new fraction is multiplied by 2 to give a new integer and a new fraction. This process is continued until the fraction becomes 0 or until the number of digits has sufficient accuracy. The coefficient of binary number are obtained from the integers as shown in the table.

**Coefficient of Binary Number**

Fraction $\times 2$	Fraction	Integer	Coefficient
$0.1245 \times 2 = 0.249$	0.249	0	$a_{-1} = 0$
$0.249 \times 2 = 0.498$	0.498	0	$a_{-2} = 0$
$0.498 \times 2 = 0.996$	0.996	0	$a_{-3} = 0$
$0.996 \times 2 = 1.992$	0.992	1	$a_{-4} = 1$
$0.992 \times 2 = 1.984$	0.984	1	$a_{-5} = 1$
$0.984 \times 2 = 1.968$	0.968	1	$a_{-6} = 1$

Thus,

$$(0.1245)_{10} = (0.000111)_2$$

i.e.,

$$(a_{-1} a_{-2} a_{-3} a_{-4} a_{-5} a_{-6})_2$$

### Decimal to Octal Conversion

We follow the same procedure as in case of decimal to binary except the base is changed now i.e., instead of 2 it is 8 now. So, wherever we have used 2 for multiplication or for division (in case of decimal to binary) we will use 8 in case of octal conversion.

e.g.,

$$(41)_{10} = (51)_8$$

**Decimal to Octal Conversion**

Integer Quotient	Remainder	Coefficient
$\frac{41}{8} = 5$	1	$a_0 = 1$
$\frac{5}{8} = 0$	5	$a_1 = 5$

### Binary to Octal or Hexadecimal Conversion

Since,  $2^3 = 8$  and  $2^4 = 16$ , each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The conversion from binary to octal is easily accomplished by partitioning the binary numbers into group of 3 digits each as given below.

$$(10 \quad 110 \quad 001 \quad 101 \quad . \quad 011 \quad 111 \quad 100 \quad 000 \quad 10)_2 = (26153.7402)_8$$

2      6      1      5      3      7      4      0      2  
 ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓      ↓  
 ←      ←      ←      ←      ←      ←      ←

Conversion, from binary to hexadecimal is similar, except the binary number is divided into group of 4 digits.

$$( \begin{array}{c} 10 \\ 2 \end{array} \quad \begin{array}{c} 1100 \\ 12 = C \end{array} \quad \begin{array}{c} 0110 \\ 6 \end{array} \quad \begin{array}{c} 1011 \\ 11 = B \end{array} \quad . \quad \begin{array}{c} 1111 \\ 15 = F \end{array} \quad \begin{array}{c} 0000 \\ 0 \end{array} \quad \begin{array}{c} 010 \\ 2 \end{array} )_2 = (2C6B.F02)$$

↔      ↔

### Key Points

- Partitioning is done from right to left for integer part and from left to right for fractional part.

#### Binary to Decimal Conversion

$$(1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \cdot 0 \ 1 \ 1)_2 = (?)_{10}$$

$$\begin{array}{cccccccccc} \downarrow & \downarrow \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 1 & 2 & 3 \end{array}$$

$$\begin{aligned} &= 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &\quad + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= 64 + 0 + 0 + 8 + 0 + 0 + 1 + 0 + \frac{1}{4} + \frac{1}{8} \\ &= (73.375)_{10} \end{aligned}$$

### Octal to Decimal Conversion

The same technique is applied over here as applied in case of binary to decimal except here our radix is 8, so we use powers of 8.

### Complements

Complements are used in digital computer system for simplifying the subtraction operation and for logical manipulation.

*There are two types of complements for each base r system,*

1. The  $r'$ s complement
2. The  $(r - 1)'$ s complement

### The $r'$ s Complement

Given a positive number  $N$  with base  $r$  with an integer part of  $n$  digits. The  $r'$ s complement of  $N$  is defined as  $r^n - N$  for  $N \neq 0$  and 0 for  $N = 0$ .

e.g., 10's complement of  $(25.639)_{10}$  is  $(10^2 - 25.639)$

$100 - 25.639 = 74 - 36.1$ , here the number of digits in integer part is 2 means  $n = 2$ .

## The $(r-1)$ 's Complement

Given, a positive number  $N$  in base  $r$  with an integer part of  $n$  digits and a fraction part of  $m$  digits, the  $(r-1)$ 's complement of  $N$  is defined as  $r^n - r^{-m} - N$ .

- 1's complement of  $(52520)_{10}$  is  $(10^5 - 1 - 52520)$

$$= 99999 - 52520 = 47479$$

Because number of integer part is 5, so  $r^n = 10^5$  and no fractional part is present' so  $r^{-m} = 10^{-0} = 1$

- 1's complement of  $(0.3267)_{10}$  is  $(10^0 - 10^{-4} - 0.3267)$

$$= 1 - 0.0001 - 0.3267$$

$$= 0.9999 - 0.3267 = 0.6732$$

No integer part, so  $10^n = 10^0 = 1$

- 1's complement of  $(101100)_2$  is  $(2^6 - 2^{-0})_{10} - (101100)_2$

$$= (64 - 1)_{10} - (101100)_2$$

$$= (63)_{10} = (101100)_2$$

$$= 111111 - 101100 = 010011$$

## Key Points

- ♦ 520 → Here,  $n = 3$ , but  $(052) \rightarrow$  here  $n = 2$ .
- ♦ In the later example 0 is of no significance.

## Alternative Conversion Process

### For 1's Complement

Replace 0's by 1's and 1's by 0's.

e.g., 1's complement of

$$(1 \ 0 \ 0 \ 1 \ 1 \ 0)_2 = (011001)_2$$

$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$

$$0 \ 1 \ 1 \ 0 \ 0 \ 1$$

## For 2's Complement

To start at the Least Significant Bit (LSB) and copy all the zeroes (working from LSB towards the most significant bit) until the first 1 is reached, then copy that 1 and take complement all the remaining bits.

e.g.,      2's complement of

$(1\ 0\ 0\ \textcircled{1}\ 1\ 0)_2$   
 ↓ ↓ ↓ ↓ ↓ ↓      first 1, when moving from left to right.  
 $(0\ 1\ 1\ 0\ 1\ 0)_2$

## Properties of 2's Complement

2's complement representation allows the use of binary arithmetic operations on signed integers, yielding the current 2's complement result.

### 2's Complement of Positive and Negative Number

- **Positive Numbers** Positive 2's complement numbers are represented as the simple binary.
- **Negative Numbers** Negative 2's complement numbers are represented as the binary number that when added to a positive number of the same magnitude equals zero.

## Key Points

- ♦ The most significant (left most) bit indicates the sign of the integer, therefore, it is sometimes called the sign bit. If the sign bit is zero, then the number is greater than or equal to zero or positive.
- ♦ If the sign bit is one, then the number is less than zero or negative.

## Subtraction Using 1's Complement

e.g.,       $(19 - 3) = ?$

$$(19)_{10} = (0001\ 0011)_2$$

$$(3)_{10} = (0000\ 0011)_2$$

Now we take 1's complement of 3 i.e.,  $(1111\ 1100)_2$

Now, we add binary equivalent of 19 and 1's complement of 3.

$$\begin{array}{r}
 0001 & 0011 \\
 + 1111 & 1100 \\
 \hline
 1\ 0000 & 1111 \\
 & \ 1 \\
 \hline
 (00010000)_2 = (16)_{10}
 \end{array}$$

This is carry bit

We need to add carry bit in case of 1's complement method.

e.g.,  $(3 - 19) = ?$

$$(3)_{10} = (0000\ 0011)_2$$

$$(19)_{10} = (0001\ 0011)_2$$

Now, we take 1's complement of 19 i.e.,  $(1110\ 1100)_2$

Now, we add binary equivalent of 3 and 1's complement of 19.

$$0000\ 0011 \rightarrow \text{Binary equivalent of } 19$$

$$1110\ 1100 \rightarrow 2^{\prime}\text{'s complement of } 3$$

$$\underline{1110\ 1111}$$

Here, we don't find any carry bit, so in this case we need to take 1's complement of the sum and put a negative sign with it. This will be our final answer.

$$(1110\ 1111)_2 \rightarrow 1^{\prime}\text{'s complement is } (0001\ 0000)_2 = (16)_{10}$$

But we should a negative sign so  $-16$  is our answer.

## Subtraction Using 2's Complement

e.g.,  $(19 - 3) = ?$

$$(19)_{10} = (0001\ 0011)_2$$

$$\text{and } (3)_{10} = (0000\ 0011)_2$$

$$2^{\prime}\text{'s complement of } 3 = (1111\ 1101)_2$$

Add binary equivalent of 19 and 2's complement of 3

$$0001\ 0011 \rightarrow \text{Binary equivalent of } 19$$

$$+ 1111\ 1101 \rightarrow 2^{\prime}\text{'s complement of } 3$$

$$\underline{1001\ 0000}$$

So, result is  $(00010000)_2 = (16)_{10}$

e.g.,  $(3 - 19) = ?$

$$\begin{array}{r}
 0001\ 0011 \rightarrow \text{Binary equivalent of } 3 \\
 + \underline{1111\ 1101} \rightarrow \text{2's complement of } 19 \\
 \hline
 1001\ 0000
 \end{array}$$

Since, no carry bit is there, we need to take 2's complement of the sum. This will be our final result with a negative sign.

$$\text{2's complement of } (11110000) = (0001\ 0000)_2 = (16)_{10}$$

So, result will be  $-16$ .

### **Example of 2's Complement**

Find out  $(-3 - 19)$  using 2's complement. This is a special case.

Here, we need to add 2's complement of 3 and 2's complement of 19.

$$\begin{array}{r}
 1111\ 1101 \rightarrow \text{2's complement of } 3 \\
 \text{This 1 is carry but } 1 \quad \underline{+1110\ 1101} \rightarrow \text{2's complement of } 19 \\
 \hline
 1110\ 1010
 \end{array}$$

Drop the carry and take the 2's complement of result because here both the numbers are negative. So, the result will be 2's complement of  $(1110\ 1010)_2$  i.e.,  $(0001\ 0110)_2 = (22)_{10}$ , additionally we need to put a  $-$  (minus) sign with the result i.e.,  $-22$ . This will be our final result.

## **Binary Codes**

Binary codes are codes which are represented in binary system with modification from original ones.

### **Weighted Binary System**

Weighted binary codes are those which obey the positional weighting principles, each position of a number represents a specific weight.

e.g., 8421, 2421, 5211.

### **Sequential Code**

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. The 8421 and excess-3 codes are sequential, whereas 2421 and 5211 codes are not.

### **Non-weighted Codes**

Non-weighted codes are codes that are not positionally weighted. That is each position within the binary number is not assigned a fixed value.

## Reflective Code

A code is said to be reflective when code for 9 is complement for the code for 0 and so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211 and excess-3 are reflective, whereas the 8421 code is not.

## BCD (Binary Coded Decimal)

It is a straight assignment of the binary equivalent. To encode a decimal number using the common BCD encoding. Each decimal digit is stored in a 4-bit number.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

BCD encoding for number 127 would be

1      2      7

$(0001\ 0010\ 0111) \rightarrow$  BCD equivalent of 127

whereas the pure binary number would be  $(01111111)_2$

## BCD Addition

Add  $(148 + 157) = ?$

148	$\xrightarrow{BCD}$ $\xrightarrow{BCD}$	<table style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">0001</td><td style="padding: 0 5px;">0100</td><td style="padding: 0 5px;">1000</td></tr> <tr> <td style="padding: 0 5px;">0001</td><td style="padding: 0 5px;">0101</td><td style="padding: 0 5px;">0111</td></tr> <tr> <td style="padding: 0 5px;">0010</td><td style="padding: 0 5px;">1001</td><td style="padding: 0 5px;">1111</td></tr> <tr> <td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0110</td><td style="padding: 0 5px;">0110</td></tr> <tr> <td style="padding: 0 5px;">0011</td><td style="padding: 0 5px;">0110</td><td style="padding: 0 5px;">10101</td></tr> </table>	0001	0100	1000	0001	0101	0111	0010	1001	1111	1	0110	0110	0011	0110	10101
0001	0100	1000															
0001	0101	0111															
0010	1001	1111															
1	0110	0110															
0011	0110	10101															
157		<table style="margin-left: 20px; border-collapse: collapse;"> <tr> <td style="padding: 0 5px;">0001</td><td style="padding: 0 5px;">0100</td><td style="padding: 0 5px;">1000</td></tr> <tr> <td style="padding: 0 5px;">0001</td><td style="padding: 0 5px;">0101</td><td style="padding: 0 5px;">0111</td></tr> <tr> <td style="padding: 0 5px;">0010</td><td style="padding: 0 5px;">1001</td><td style="padding: 0 5px;">1111</td></tr> <tr> <td style="padding: 0 5px;">1</td><td style="padding: 0 5px;">0110</td><td style="padding: 0 5px;">0110</td></tr> <tr> <td style="padding: 0 5px;">0011</td><td style="padding: 0 5px;">0110</td><td style="padding: 0 5px;">10101</td></tr> </table>	0001	0100	1000	0001	0101	0111	0010	1001	1111	1	0110	0110	0011	0110	10101
0001	0100	1000															
0001	0101	0111															
0010	1001	1111															
1	0110	0110															
0011	0110	10101															
Answer		3      0      5															

When sum of 2 digits is greater than or equal to 9, then we need to add 6 i.e., 0110.

## 2421 Code

This is a weighted code, its weights are 2,4,2 and 1. A decimal number is represented in 4 bit form and the total 4 bits weight is

$$2 + 4 + 2 + 1 = 9.$$

Hence, 2421 code represents the decimal numbers from 0 to 9.

Decimal	0	1	2	3	4	5	6	7	8	9
2421	0000	0001	0010	0011	0100	1011	1100	1101	1110	1111

## Excess-3 Code

Excess-3 is a non-weighted code used to represent decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

e.g.,

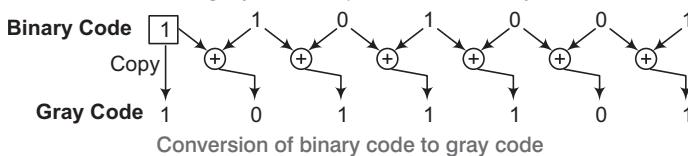
Decimal	8421	Excess-3
8	1000	$1000+0011=1011$
6	0110	$0110+0011 = 1001$

## Gray Code

This is a variable weighted code and is cyclic. This means that it is arranged, so that every transition from one value to the next value involves only one bit change.

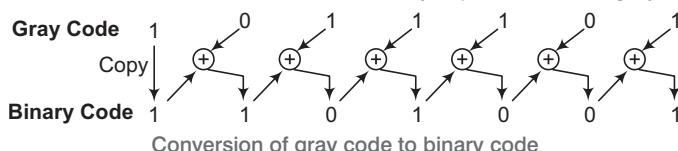
### Binary to Gray Code Conversion

1. Write down the number in binary codes.
2. The Most Significant Bit (MSB) of the gray code will be same as the MSB of binary code.
3. Perform XOR operation on MSB and next bit to the MSB in binary number.
4. Repeat step 3 till all bits of binary number have been XORED, the resultant code is the gray code equivalent to binary code.



### Gray Code to Binary Conversion

1. Start with the MSB of gray coded number.
2. Copy this bit as the MSB of the binary number.
3. Now, perform Ex-OR operation of this bit with the next bit of the binary number.
4. Repeat step 3 till all bits of gray coded number have been used in XOR operation. The resultant number is the binary equivalent of the gray number.



## Fixed Point Representation

Because of computer hardware limitation everything including the sign of number has to be represented either by 0's or 1's.

So, for a positive number the leftmost bit or sign bit is always 0 and for a negative number the sign bit should be 1.

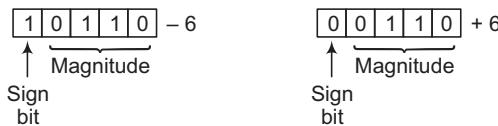
## Representation of Integers

*The are three possible ways to represent a number*

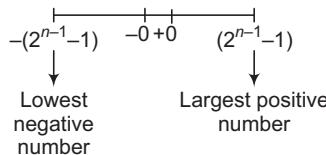
- (i) Signed magnitude method
- (ii) 1's complement method
- (iii) 2's complement method

## Signed Magnitude Method

Number is divided into two parts, one is sign bit and other part for magnitude. In example we are using 5 bit register to represent - 6 and + 6.



**Range of Number** For  $n$  bit register, MSB will be a sign bit and  $(n - 1)$  bits will be magnitude.



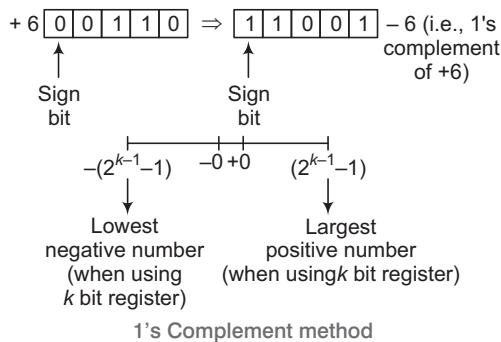
## Key Points

- ♦ Drawback of signed magnitude method is that 0 will be having 2 different representation one will be 10000 i.e., -0 and the other one will be 00000 +0.

## 1's Complement Method

Positive numbers are represented in same way as in sign magnitude method. If number is negative, then it is represented using 1's complement method. For this, we first need to represent the number with positive sign and then take 1's complement of this number.

e.g., Suppose we are using 5 bit register. The representation of  $-6$  will be as below.

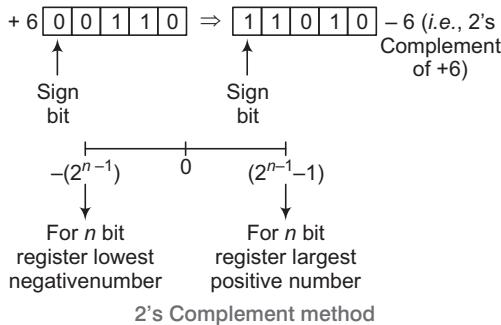


### Key Points

- The only drawback of 1's complement method is that there are two different representation for zero, one is  $-0$  and other is  $+0$ .

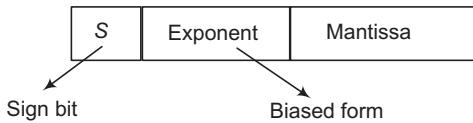
### 2's Complement Method

Positive numbers are represented in same way as in sign magnitude. For representing negative number, we take 2's complement of the corresponding positive number.



### Floating Point Representation

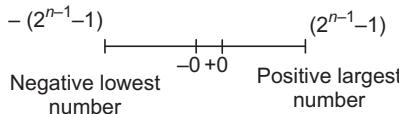
A floating point number can be represented using two points. First is called mantissa ( $m$ ) and other one is exponent ( $e$ ). Thus, in a number system with base  $r$ , a floating point number with mantissa  $m$  and exponent  $e$  will be represented as  $(m \times r^e)$ .



The value of  $m$  may be a fraction or an integer. Thus, a number  $(2.25)_{10}$  can be represented as  $0.225 \times 10^1$ .

Here,  $m = 225$  and  $e = 1, r = 10$ .

For  $n$  bit register, MSB will be sign bit and  $(n - 1)$  bits will be magnitude. So, positive largest number that can be stored is  $(2^{n-1} - 1)$  and negative lowest number is  $-(2^{n-1} - 1)$ .



### Actual Number Finding Technique

Here, we always store exponent in positive. Biased number is also called excess number. Since, exponent is stored in biased form, so bias number is added to the actual exponent of the given number. Actual number can be calculated from the contents of the registers by using following formula

$$\text{Actual number} = (-1)^s (1 + m) \times 2^{e - \text{Bias}}$$

$S$  = Sign bit

$m$  = Mantissa value of register

$e$  = Exponent value of register

Bias = Bias number of  $n$  bits used to represent exponent, then

$$\text{Bias number} = (2^{n-1} - 1)$$

$$\text{Range of exponent} = -(2^{k-1} - 1) \text{ to } 2^{k-1}.$$

## Boolean Algebra

Boolean algebra is an algebraic structure defined on a set of elements together with two binary operators (+) and (.)

### Closure

For any  $x$  and  $y$  in the alphabet  $A$ ,  $x + y$  and  $x \cdot y$  are also in  $A$ .

### Duality

If an expression contains only the operations AND, OR and NOT. Then, the dual of that expression is obtained by replacing each AND by OR, each OR by AND, all occurrences of 1 by 0 and all occurrences of 0 by 1.

**Table of Some Basic Theorems**

Law/Theorem	Law of Addition	Law of Multiplication
Identity Law	$x + 0 = x$	$x \cdot 1 = x$
Complement Law	$x + x' = 1$	$x \cdot x' = 0$
Idempotent Law	$x + x = x$	$x \cdot x = x$
Dominant Law	$x + 1 = 1$	$x \cdot 0 = 0$
Involution Law	$(x')' = x$	
Commutative Law	$x + y = y + x$	$x \cdot y = y \cdot x$
Associative Law	$x + (y + z) = (x + y) + z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Distributive Law	$x \cdot (y + z) = x \cdot y + x \cdot z$	$x + y \cdot z = (x + y) \cdot (x + z)$
Demorgan's Law	$(x + y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$
Absorption Law	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

**Boolean Value** The value of Boolean variable can be either 1 or 0.

### Key Points

- Principle of duality is useful in determining the complement of a function.

## Boolean Operators

There are four Boolean operators

1. AND () operator ( $A \cdot B$ )
2. OR (+) operator ( $A + B$ )
3. NOT ( $\bar{A}$  /  $A'$ ) operator
4. XOR ( $\oplus$ ) operator ( $A + B = \bar{A} \cdot B + A \cdot \bar{B}$ )

### Operator Precedence

The operator for evaluating Boolean expression is

1. Paranthesis
2. AND
3. NOT
4. OR.

## Boolean Function

- A Boolean function is an expression formed with binary variables, the binary operators (+, .), the unary operator ( $-$ ), parenthesis and equal sign.
- For a given value of variable, the function can take only one value either 0 or 1.
- A Boolean function can be shown by a truth table. To show a function in a truth table we need a list of the  $2^n$  combinations of 1's and 0's of the  $n$  binary variables and a column showing the combinations for which the function is equal to 1 or 0. So, the table will have  $2^n$  rows and columns for each input variable and the final output.

## Canonical and Standard Form

- A canonical form is a unique representation for a given symbolic expression. If two expressions have the same canonical form, they must represent the same function.
- Boolean functions expressed as a sum of minterms i.e., sum of products.
- A minterm is a special product of literals in which each input variable appears exactly once. Function with  $n$  variable has  $2^n$  minterms.
- A maxterm is represented as sum of variable.  $n$  variables can be combined to form  $2^n$  maxterms. Each maxterm is obtained from OR of the  $n$  variables, with each variable being unprimed, if the corresponding bit of the binary number is 0 and primed, if the binary number is 1.

Wherever value of function  $F_1$  is 1, corresponding minterm will be considered.

### Key Points

- Two functions of  $n$  binary variables are said to be equal, if they have same value for all possible  $2^n$  combinations of the  $n$  variables.

### Minterm and Maxterm for a Function

$F_1$ Function	$x$	$y$	$z$	Minterm	Shorthand for Minterms	Maxterm	Shorthand for Maxterms
0	0	0	0	$\bar{x}\bar{y}\bar{z}$	$m_0$	$(x + y + z)$	$M_0$
0	0	0	1	$\bar{x}\bar{y}z$	$m_1$	$(x + y + \bar{z})$	$M_1$
1	0	1	0	$\bar{x}y\bar{z}$	$m_2$	$(x + \bar{y} + z)$	$M_2$
0	0	1	1	$\bar{x}yz$	$m_3$	$(x + \bar{y} + \bar{z})$	$M_3$
1	1	0	0	$x\bar{y}\bar{z}$	$m_4$	$(\bar{x} + y + z)$	$M_4$
0	1	0	1	$x\bar{y}z$	$m_5$	$(\bar{x} + y + \bar{z})$	$M_5$
1	1	1	0	$xy\bar{z}$	$m_6$	$(\bar{x} + \bar{y} + z)$	$M_6$
1	1	1	1	$xyz$	$m_7$	$(\bar{x} + \bar{y} + \bar{z})$	$M_7$

## Representation of Boolean Function using Minterm

A Boolean function may be expressed algebraically from given truth table by forming a minterm for each combination of the variable which produces a 1 in the function and taking the OR of all those terms. We can refer the preceding table.

Thus, function  $F_1$  will be written as

$$F_1 = \bar{x}y\bar{z} + x\bar{y}\bar{z} + xy\bar{z} + xyz$$

Here, we have selected only those minterms for which

$F_1 = 1$  and  $F_1$  will be represented (using minterm)

$$F_1 = \Sigma (2, 4, 6, 7) = \Sigma(m_2, m_4, m_6, m_7)$$

As,  $F_1 = 1$  for 010, 100, 110 and 111 those decimal equivalents are 2, 4, 6 and 7, respectively.

### Representation of Boolean Function using Maxterm

For maxterm representation use those maxterms for which  $F_1 = 0$  i.e., which produce 0 in the function. Then, form the AND of all those maxterm.

Thus,  $f_1$  will be given by the below Boolean expression

$$\begin{aligned} F_1 &= (x + y + z) \cdot (x + y + \bar{z}) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + y + \bar{z}) \\ &= M_0 \cdot M_1 \cdot M_3 \cdot M_5 \end{aligned}$$

$F_1$  will be represented as

$$F_1 = \Pi (0, 1, 3, 5)$$

### Karnaugh Map

- K-map is used to simplify the Boolean expression.
- K-map provides a pictorial method of grouping together, expressions with common factors of their, for eliminating unwanted variables.

	a	b	0	1
0				
1			1	1

2 variable K-map

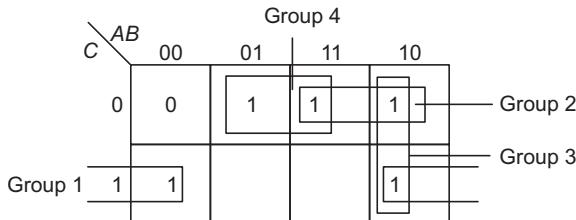
K-map for function  $F(a, b) = ab + a\bar{b}$

AB	CD		00	01	11	10
	00	01	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}CD$
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}CD$		
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BC\bar{D}$	$\bar{A}BCD$		
11	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABC\bar{D}$	
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}C\bar{D}$	$A\bar{B}CD$	$A\bar{B}\bar{C}D$	

Four variable K-map

### Rules for K-map

- Adjacent squares that have 1's are combined in groups of 2, 4 or 8 allowing the removal of 1, 2 or 3 variable from a term.
- The maps are considered to "Warp around", so that the top and bottom corresponding squares are adjacent and the left and right squares are also adjacent.
- Make largest groups and use each square only once.
- No redundant and unnecessary group are allowed.



### Don't Care Condition

- There are applications where certain combinations of input variables never occur. We don't care what the function output is to be for these combinations of the variables because they are guaranteed never to occur.
- A don't care combination is marked with an X. When choosing adjacent squares to simplify the function in the map, the X's may be assumed to be either 0 or 1, whichever gives the simplest expression.
- In addition, an X need not be used at all, if it don't contribute to cover a larger area. In each case, the choice depends only on the simplification that can be achieved.

# Combinational Circuit

Combinational circuits are Logic circuits that perform arithmetic functions (e.g., addition, subtraction, multiplication and division).

These circuits don't have memory and the output depends only on inputs provided. Combinational circuit is a logic circuit containing only logic gates.

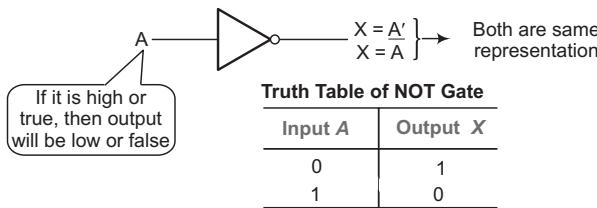
## Key Points

- ♦ Logic gates are not always required because simple logic functions can be performed with switches or diodes.
  - (a) Switches in series (AND function)
  - (b) Switches in parallel (OR function)
  - (c) Combining IC outputs with diodes (OR function)

## Basic Logic Gates

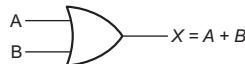
### NOT Gate

- One input, one output
- Whatever logical state (1, 0) is applied to the input the opposite state (0,1 respectively) will appear at output.
- Also known as inverter.



### OR Gate

The output will be high / true / 1, if any or all of its inputs are high / true / 1. The output will be low / zero / false only if all of inputs are low / zero / false.

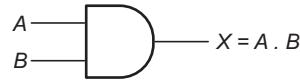


Truth Table for OR Gate

Input A	Input B	Output X = (A · B)
0	0	0
0	1	1
1	0	1
1	1	1

## AND Gate

The output will be high / true / 1, if all of its inputs are high / true / 1. If any of its inputs is low / false / 0, then output will be low / false / 0.



**Truth Table for AND Gate**

Input A	Input B	Output $X = (A \cdot B)$
0	0	0
0	1	0
1	0	0
1	1	1

To change the type of gate, such as changing OR to AND, you must do three things

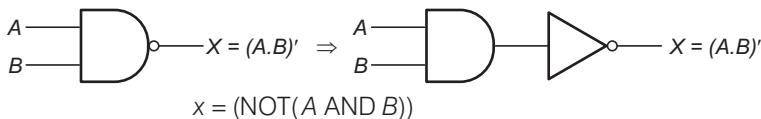
- Invert (NOT) each input.
- Change the gate type (OR to AND or AND to OR)
- Invert (NOT) the output.

## Universal Gates

Any function can be implemented with the help of these (NAND, NOR) gates.

### NAND Gate

This is an AND gate with the output inverted or we can say (AND + NOT). Therefore, the output expression of the two input NAND gate is  $X = (A \cdot B)'$ .



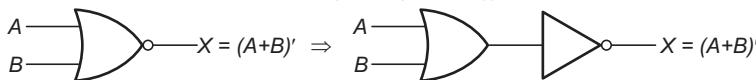
**Truth Table of NAND Gate**

Input A	Input B	Output $X = (A \cdot B)'$
0	0	1
0	1	1
1	0	1
1	1	0

### NOR Gate

This is an OR gate with the output inverted or we can say (OR + NOT). Therefore, the output expression of the two input NOR gate is  $X = (A + B)'$ .

$$X = (\text{NOT}(A \text{ OR } B))$$



**Truth Table of NOR Gate**

Input A	Input B	Output $X = (A + B)'$
0	0	1
0	1	0
1	0	0
1	1	0

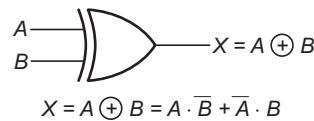
## Other Types of Gates

### Exclusive OR (EX-OR / XOR) Gate

The XOR gate provides 1 as an output only if its two inputs are different. If the inputs are same, the output will be a '0'. Unlike standard OR / NOR and AND / NAND functions the XOR function always has exactly two inputs.

$$X = A \oplus B = A \cdot \bar{B} \oplus \bar{A} \cdot B$$

If A and B are different, then output will be high.

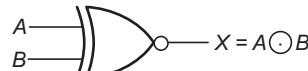
**Truth Table of XOR Gate**

Input A	Input B	Output $X = A\bar{B} + \bar{A}B$
0	0	0
0	1	1
1	0	1
1	1	0

### Exclusive NOR (EX- NOR / XNOR) Gate

The XNOR gate produces output 1 only if the inputs are same. If the inputs are different the output will be a zero 0.

$$X = A \cdot B = \bar{A} \cdot \bar{B} + A \cdot B$$



If A and B are different, then the output will be low or zero.

**Truth Table of XNOR Gate**

Input A	Input B	Output $X = A\bar{B} + \bar{A}B$
0	0	1
0	1	0
1	0	0
1	1	1

## Substituting one Type of Gate for Another

### Any Logic Gate can be built from NAND or NOR Gates

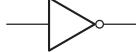
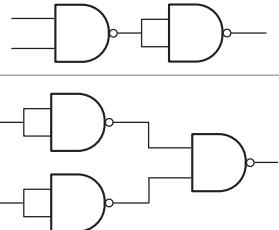
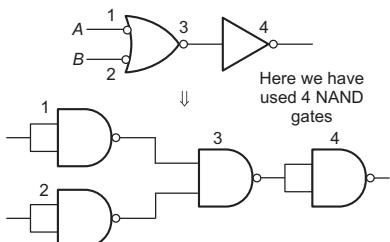
NAND or NOR gates can be combined to create any type of gate. This enables a circuit to be built from just one type of gate, either NAND or NOR. e.g., an AND gate is a NAND gate, then a NOT gate (to undo the inverting function).

#### Key Points

- AND and OR gate can't be used to create other gates because they lack the inverting (NOT) function.

e.g., an OR gate can be built from NOTed inputs fed into a NAND (AND + NOT) gate.

**Chart for NAND Equivalents**

Gate	Equivalent in NAND Gates
NOT	
AND	
OR	
NOR	 <p style="text-align: center;">              Here we have used 4 NAND gates         </p>

## Combinational Circuit for Arithmetic Operations

### Half Adder

Addition of 2 binary digits requires 2 inputs, one for result and one for carry.

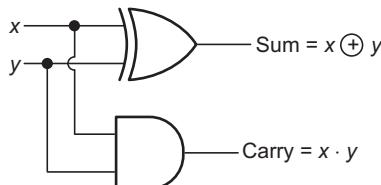
Combinational circuit for result is



and for carry, combinational circuit is



combining both we will get the below circuit



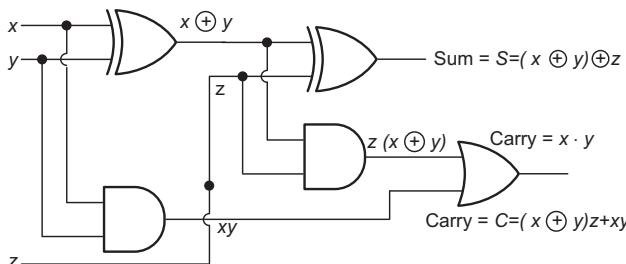
Half adder logic diagram

### Truth Table for Half Adder

Input A		Output	
x	y	$S = x + y$ (Result)	$C = x \cdot y$ (Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

### Full Adder

A full adder is a combinational circuit that forms the arithmetic sum of 3 input bits. It consists of 3 inputs and 2 outputs. Two of the input variables are the 2 significant bits to be added. The 3rd input represents the carry from previous lower significant position.



Full adder logic diagram

$$S = z \oplus (x \oplus y) = z' (xy' + x'y) + z (xy' + x'y')$$

$$S = z' (xy' + x'y) + z (xy + x'y')$$

$$S = xy' z' + x' yz' + xyz + x' y' z$$

and the carry output is  $C = z (xy' + x'y) + xy = xy' z + x'y z + xy$

### Positive and Negative Logic

- If the signal that activates the circuit (the 1 state) has a voltage level that is more positive than the 0 state, then the logic polarity is considered to be positive. Thus, in positive logic 1 is considered as high value and 0 is considered as low value.
- If the signal that activates the circuit (the 1 state) has a voltage level that is more negative than the 0 state, then the logic polarity is considered to be negative. Thus in negative logic 1 is considered as low value and 0 is considered as high value.

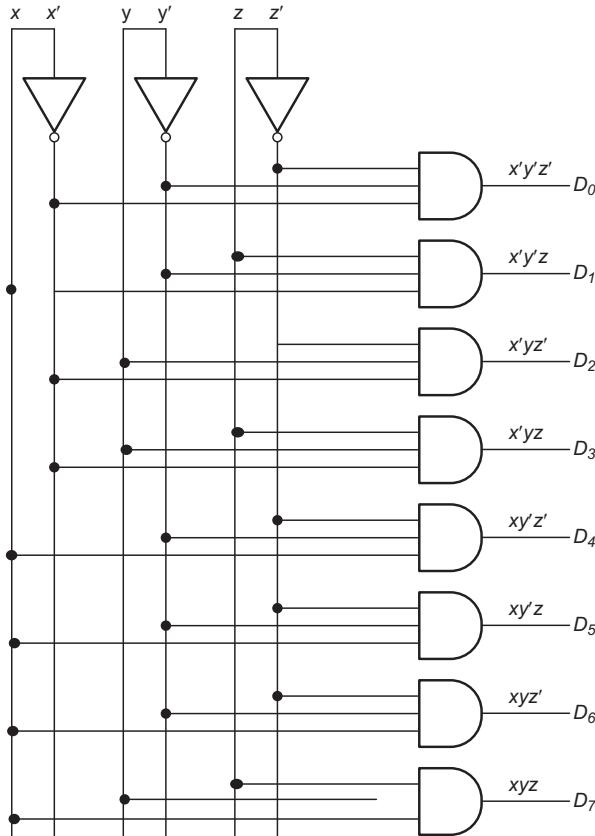
## Decoders

A decoder is a combinational circuit that converts binary information from  $n$  input lines of a maximum of  $2^n$  unique output lines. If the  $n$  bit decoded information has unused or don't care combinations, the decoded output will have less than  $2^n$  outputs. The decoders are also called as  $n$  to  $m$  line decoders where  $m \leq 2^n$ .

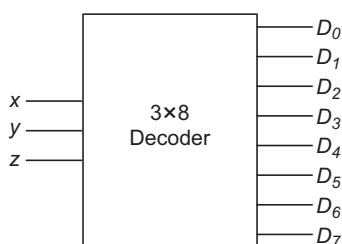
### Truth Table of 3 Bit Binary to Decimal Decoder

Input			Output	Minterm
$x$	$y$	$z$		
0	0	0	$D_0 = 0$	$x'y'z'$
0	0	1	$D_1 = 1$	$x'y'z$
0	1	0	$D_2 = 2$	$x'yz'$
0	1	1	$D_3 = 3$	$x'yz$
1	0	0	$D_4 = 4$	$xy'z'$
1	0	1	$D_5 = 5$	$xy'z$
1	1	0	$D_6 = 6$	$xyz'$
1	1	1	$D_7 = 7$	$xyz$

This decoder takes binary values as input and produces decimal value at output. Suppose, if  $D_3$  is high it means the binary combination of 3 that is 011, means  $x = 0, y = 1$  and  $z = 1$ .



Combinational circuit for  $3 \times 8$  decoder



IC presentation of decoder

## Encoder

An encoder is a digital function that produces a reverse operations from that of a decoder. An encoder has  $2^n$  (or less) input lines and  $n$  output lines. The output lines generate the binary code for  $2^n$  input variables.

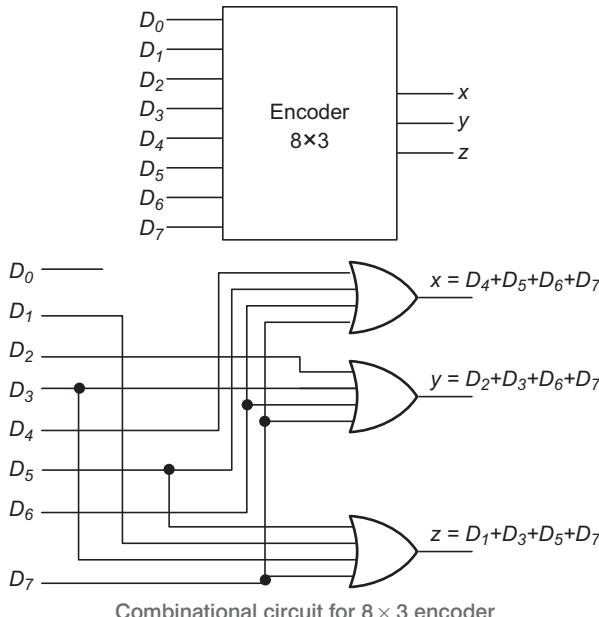
Input								Output		
$D_0$	$D_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	$D_7$	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

From the above truth table, we get

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$



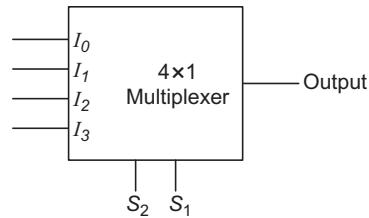
Combinational circuit for  $8 \times 3$  encoder

## Multiplexer

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to the output line. The selection of a particular input line is controlled by a set of selection line. Normally, there are  $2^n$  lines and  $n$  selection lines whose bit combinations determine which input is selected.

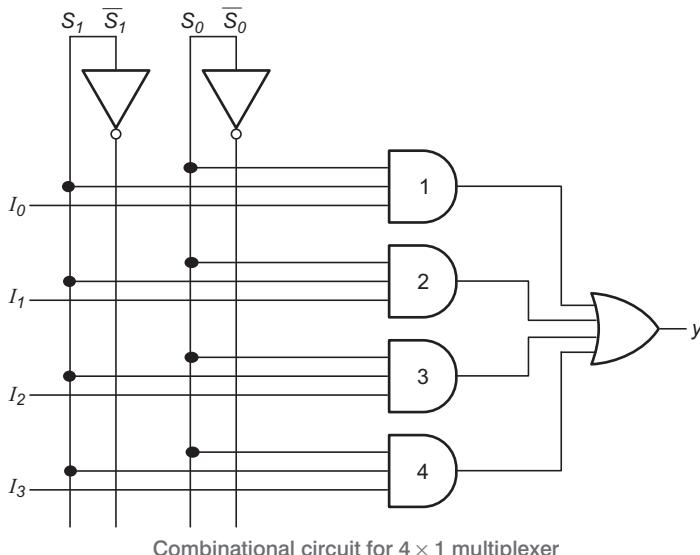
### Selection Lines For Input Combinations

$S_1$	$S_2$	Output will depend on
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$



The Selection of Inputs will be as of the above Table

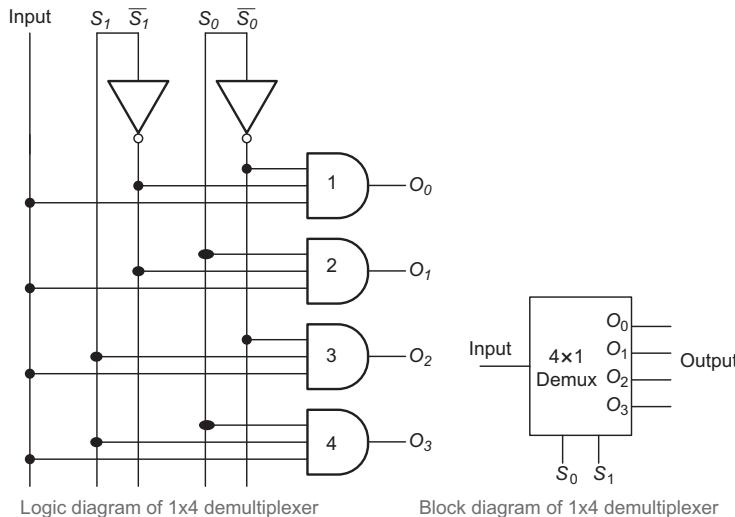
When  $S_1 = 0$  and  $S_0 = 0$ . Output of gate 2, 3 and 4 will be 0, and the output of gate 1 will depend upon  $I_0$  (because  $\bar{S}_0 = 1$  and  $\bar{S}_1 = 1$ , thus output of gate 1 =  $I_0 \cdot 1 \cdot 1 = I_0$ )



Combinational circuit for  $4 \times 1$  multiplexer

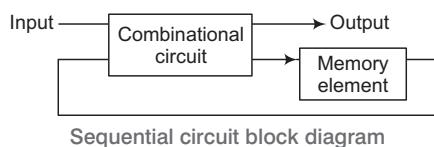
## Demultiplexer

A demultiplexer is a circuit that receives information on a single line and transmit that information on one of  $2^n$  possible output lines. The output line to which input will be transmitted is selected by the selection line.



## Sequential Circuit

A switching circuit whose output depends not only on the present state of its input but also what its input conditions have been in the past.



Sequential circuit block diagram

Sequential logic output depends on stored levels and also the input levels. The memory elements are capable of storing binary information. A sequential circuit is specified by a time, sequence of inputs, output and internal states.

## Key Points

- ♦ There are two types of sequential circuits. This classification depends on the timing of their signals.
- ♦ Asynchronous sequential circuits
- ♦ Synchronous sequential circuits

### Asynchronous Sequential Circuits

This is a system whose outputs depend upon the order in which its input variable changes and can be affected at any instant of time.

### Synchronous Sequential Circuits

These circuits have a clock signal as one of their inputs to force the change in output only at discrete instance of time. As state transition in such circuits occurs only when the clock value is either 0 or 1 or happen at rising or falling edge of the clock.

The memory element used in the synchronous sequential circuits is called flip-flops.

## Flip-Flops

These are the binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it.

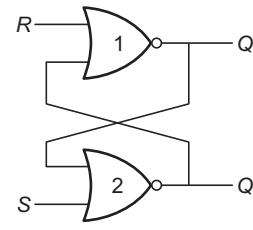
A flip-flop circuit can maintain a binary state indefinitely until directed by an input signal of switch states.

### A Basic Flip-Flop

A flip-flop circuit can be constructed from two NAND gates or two NOR gates.

**SR Flip Flop Truth Table**

S.No.	Input		Output	
	S	R	Q	Q'
1	0	0	Unchanged	Unchanged
2	0	1	0	1
3	1	0	1	0
4	1	1	Undefined	Undefined



Flip-flop using NOR gate

For more understanding let us consider a case when  $S = 0$  and  $R = 1$  and the current output of both the gates are  $Q$  and  $Q'$  respectively, these outputs are working as inputs for the gate 2 and gate 1, respectively. So, for the first cycle.

$$R = 1 \text{ and } Q' \text{ will be input for gate 1 so}$$

$$Q = 1 + Q' = 1 \text{ and}$$

$$S = 0 \text{ and } Q = 1 \text{ will be input for gate 2 so}$$

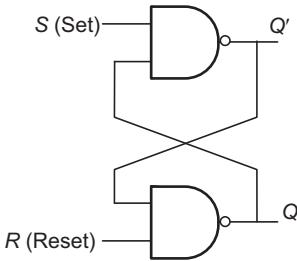
$$Q' = (0 + 1)' = 0$$

Now, the output will again work as inputs for the gates. We will keep on putting these values again and again, until we are not getting the same output each time.

Below diagram illustrates a basic flip flop circuit using NAND gate

### Truth Table For NAND Gate Flip-Flop

Input		Output	
S	R	Q	Q'
0	0	X	X
0	1	0	1
1	0	1	0
1	1	No change i.e., Q	No change i.e., Q'



Flip-flop using NAND gate

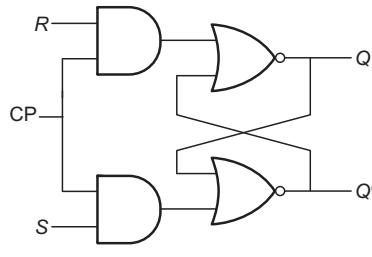
### RS Flip - Flop

An RS flip-flop is similar to an SR latch. It functions only when clock pulse is 1 or an active clock edge is there.

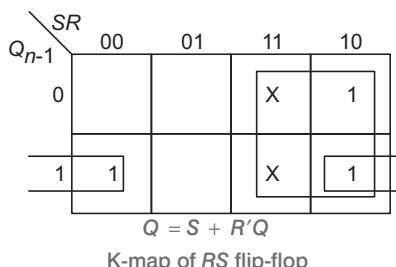
In RS flip-flop  $S = 1$  sets the next state value of  $Q$  to 1 and  $R = 1$  resets the next value of  $Q$  to 0.

### Truth Table For RS Flip-Flop When Clock Pulse is 1

Present state of $Q (Q_{n-1})$	S	R	Next state of $Q (Q_n)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Invalid
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Invalid

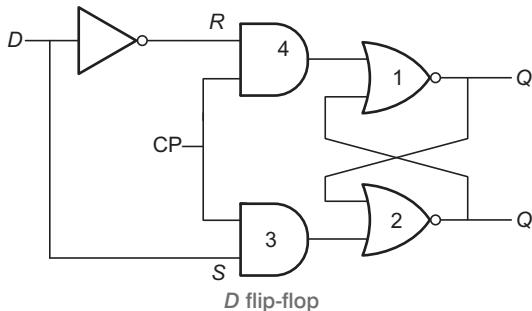


RS flip-flop



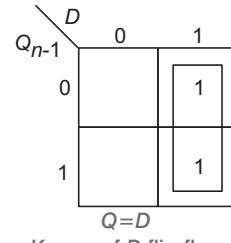
## D Flip-Flop

- Modified of RS flip-flop.
- $D$  flip-flop has 2 inputs, data input ( $D$ ) and clock pulse.
- This will function only when clock pulse is 1 or when the appropriate pulse edge of the clock input is encountered.



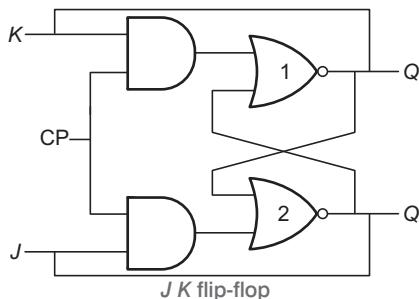
**Truth Table of  $D$  Flip-Flop When Clock Pulse is 1**

Present state of ( $Q_{n-1}$ )	$D$	Next State of $Q$ ( $Q_n$ )
0	0	0
0	1	1
1	0	0
1	1	1



## JK Flip-Flop

- $JK$  flip-flop is an extended version of the  $RS$  flip-flop.
- It has 3 inputs .  $J$ ,  $K$  and Clock Pulse (CP). The  $J$  input corresponds to  $S$  input and the  $K$  input corresponds to the  $R$  input.



**Truth Table of JK Flip-Flop When Clock Pulse is 1**

Present state of Q ( $Q_{n-1}$ )	T	K	Next State of Q ( $Q_n$ )
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$JK$

$Q_{n-1}$	00	01	11	10
0	0	1	<span style="border: 1px solid black; padding: 2px;">1</span>	<span style="border: 1px solid black; padding: 2px;">1</span>
1	<span style="border: 1px solid black; padding: 2px;">1</span>			<span style="border: 1px solid black; padding: 2px;">1</span>

$$Q = JQ' + K'Q$$

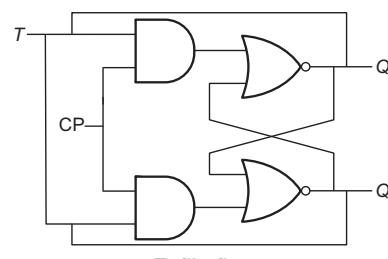
K-map of JK flip-flop

**T Flip-Flop**

- Also known as toggle flip-flop.
- Frequently used in building counters.
- It has 2 inputs, T and Clock Pulse (CP).
- When  $T = 1$ , the flip-flop changes state after the active edge of the clock. If  $Q_{n-1} = 0$  (present state of Q). The next state value of Q, will be set to 1. If  $Q_{n-1} = 1$ , the next state value of Q will be reset to 0. When  $T = 0$ , no state change occurs.

**Truth Table of Flip-Flop When Clock Pulse is 1**

Present state of ( $Q_{n-1}$ )	T	Q
0	0	0
0	1	1
1	0	1
1	1	0



T- flip-flop

$T$

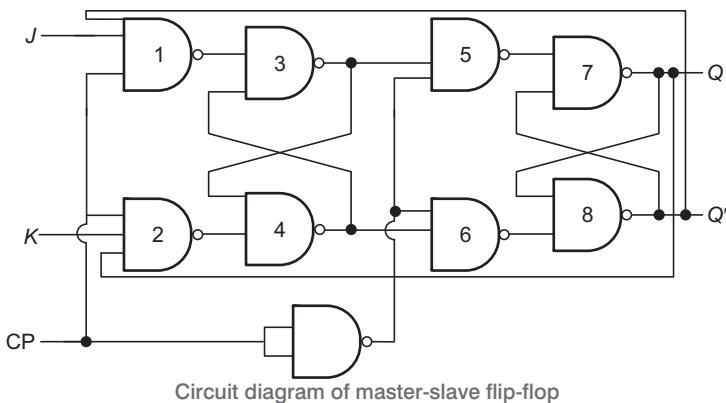
$Q_{n-1}$	0	1
0		1
1	1	

$$Q = TQ' + T'Q$$

K-map of T flip-flop

## Master-Slave Flip-Flops

- Constructed from two separate flip-flops.
- One circuit serves as master and the other as a slave.
- Gates 1 through 4 are from master flip-flop and gates 5 through 8 are from slave.
- Master works on the positive edge of clock pulse and slave works on the negative edge of clock pulse.

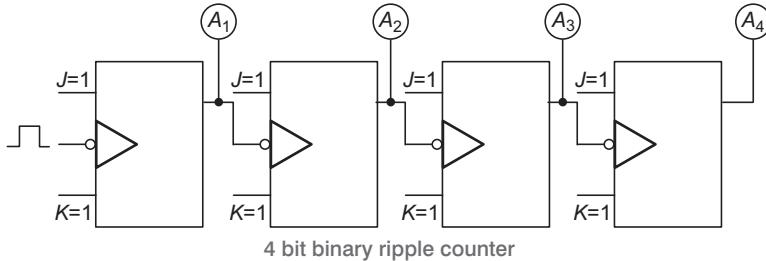


## Counters

- Sequential circuit that goes through a prescribed sequence of states upon the application of input pulses is called a counter.
- In a counter, the sequence of states may follow a binary count or any other sequence of states.
- Counters are used for counting the number of occurrences of an event and are useful for generating timing sequences to control operations in a digital system.
- A counter that follows the binary sequence is called a binary counter.
- An  $n$ -bit binary counter consists of  $n$  flip-flop and can count in binary from 0 to  $2^{n-1}$ .
- Up counters increase in value and down counters decrease in value.

## Asynchronous Counter (Ripple Counter)

In ripple counter, the flip-flop output transition serve as a source for triggering other flip-flop. e.g., in a binary ripple counter consists of a series connection of complementing flip-flops ( $T$  or  $JK$ ), with the output of each flip flop connected to the CP input of next higher order flip-flop.



## Synchronous Counter

Synchronous counter is distinguished from ripple counter such that in synchronous counter clock pulses are applied to the CP inputs of all flip-flop simultaneously, rather than one at a time in succession as in ripple counter.

The decision whether flip-flop is to be complemented or not is determined from the value of  $J$  and  $K$  inputs at the time of the pulse.

## Registers

- A register consists of a group of flip-flops with a common clock input.
- Register are commonly used to store and shift binary data.

### Key Points

- ♦ An  $n$  bits register has a group of  $n$  flip-flop which can store a binary information of  $n$  bits.

## Shift Register

It is a register in which binary data can be stored and then shifted left or right when a clock pulse is applied. Bits shifted out one end of the register may be either lost or shifted back in on the other end.

## **Machine Instructions**

### **Computer Instruction**

A binary code used for specifying micro operations for computer.

**Instruction Code** Group of bits used to instruct the CPU to perform a specific operation.

**Instruction Set** Collection of instructions.

**Instruction Representation** Each instruction has a unique bit pattern, but for human beings a corresponding symbolic representation has been defined.

e.g., ADD, SUB, LOAD, etc.

<b>Operation code (OPCODE)</b>	This refers to the operation to be performed e.g., Add, Subtract, Multiply.
<b>Source operand reference</b>	Location where source operand can be found.
<b>Result operand reference</b>	Location where result is to be put.
<b>Next instruction reference</b>	Location where reference to the next instruction can be found.

Subparts of instruction

## **Instruction Cycles**

*Instruction cycle consists of following phases*

- Fetching an instruction from memory.
- Encoding the instruction.
- Reading the effective address from memory in case of the instruction having an indirect address.
- Execution of the instruction.

## **Instruction Format**

An instruction consists of bits and these bits are grouped up to make fields.

*Some fields in instruction format are as follows*

1. Opcode which tells about the operation to be performed.
2. Address field designating a memory address or a processor register.
3. Mode field specifying the way the operand or effective address is determined.

## Different types of Instruction formats

Some common types are as given below

- Three address instruction format
- Two address instruction format
- One address instruction format
- Zero address instruction format

### Three Address Instruction Format

- This system contains three address fields (address of operand 1, address of operand 2 and address where result needs to be put).
- The address of next instruction is held in a CPU register called Program Counter (PC).

	Add	Result address	OP1 address	OP2 address
Bits	8	24	24	24

Here, the number of bytes required to encode an instruction is 10 bytes i.e., each address requires 24 bit = 3 bytes. Since, there are three addresses and one opcode field so,

$$3 \times 3 + 1 = 10 \text{ bytes.}$$

The number of memory access required is 7 words, i.e., 4 words for instruction fetch, 2 words for operand fetch and 1 word for result to be placed back in memory.

### Two Address Instruction Format

- In this format, two addresses and an operation field is there.
- The result is stored in either of the operand address i.e., either in address of first operand or in the address of second operand.
- CPU register called Program Counter (PC) contains the address of next instruction.

Op Code  
↓

	Add	Result address	OP1 address
Bits	8	24	24

## One Address Instruction Format

- One address field and an operation field.
- This address is of the first operand.
- The second operand and the result are stored in a CPU register called Accumulator Register (AR). Since, a machine has only one accumulator, it needs not be explicitly mentioned in the instruction.
- A CPU register (*i.e.*, Program Counter (PC) holds the address of next instruction.
- In this scenario, two extra instructions are required to load and store the accumulator contents.

Op Code		
Bits	Add	OP1 address
	8	24

- Number of bits required to encode an instruction is 4 bytes. *i.e.*, each address requires 24 bits = 3 bytes. Since, there are one address and one operation code field,  $1 * 3 + 1 = 4$  bytes.
- The number of memory access required is 3 words *i.e.*, 2 words for instruction fetch + 1 word for code for operand fetch.

### Key Points

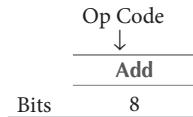
- ♦ Total number of bytes to encode an instruction = number of address fields \* Bytes required to store an address + bytes required to store operation code.  

$$2 * 3 + 1 = 7 \text{ bytes.}$$
- ♦ The number of memory access required is 6 words *i.e.*, 3 words for instruction fetch + 2 words for operand fetch + 1 word for result to be placed back in memory.

## Zero Address Instruction Format

- Here, an stack is included in the CPU for performing arithmetics and logic instructions with no addresses.
- The operands are pushed onto the stack from memory and ALU operations are implicitly performed on the top elements of the stack.

- The address of the next instruction is held in a CPU register called program counter.



e.g., Add

Top of stack  $\leftarrow$  Top of stack + second top of stack.

## Addressing Modes

Addressing modes are the ways how architectures specify the address of an operand of an instruction. *There are various addressing modes*

### **Implied Mode**

In this mode the operands are specified implicitly in the definition of an instruction.

### **Immediate Mode**

In this mode the operand is specified in the instruction itself or we can say that, an immediate mode instruction has an operand rather than an address.

### **Register Mode**

In this mode, the operands are in registers.

### **Direct Address Mode**

In this mode, the address of the memory location that holds the operand is included in the instruction. The effective address is the address part of the instruction.

### **Indirect Address Mode**

In this mode the address field of the instruction gives the address where the effective address is stored in memory.

### **Relative Address Mode**

In this mode the content of program counter is added to the address part of the instruction to calculate the effective address.

### **Indexed Address Mode**

In this mode, the effective address will be calculated as the addition of the content of index register and the address part of the instruction.

## **Data Transfer Instructions**

Data transfer instructions cause transfer of data from one location to another without changing the information content.

The common transfers may be between memory and processor registers, between processor registers and input/output.

**Typical Data Transfer Instructions**

Name	Mnemonic
LOAD	LD
STORE	ST
MOVE	MOV
EXCHANGE	XCH
INPUT	IN
OUTPUT	OUT
PUSH	PUSH
POP	POP

## **Data Manipulation Instructions**

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.

*There are three types of data manipulation instructions.*

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions.

**Typical Arithmetic Instructions**

Name	Mnemonic
INCREMENT	INC
DECREMENT	DEC
ADD	ADD
SUBTRACT	SUB
MULTIPLY	MUL
DIVIDE	DIV
ADD WITH CARRY	ADDC
SUBTRACT WITH BORROW	SUBB
NEGATIVE	NEG

**Typical Logical and Bit Manipulation Instructions**

Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE-OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

**Typical Shift Instructions**

Name	Mnemonic
LOGICAL SHIFT RIGHT	SHR
LOGICAL SHIFT LEFT	SHL
ROTATE RIGHT	ROR
ROTATE LEFT	ROL

**Program Control Instructions**

- Program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data processing operations.
- The change in value of a program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

**Typical Program Control Instructions**

Name	Mnemonic
BRANCH	BR
JUMP	JMP
SKIP	SKP
CALL	CALL
RETURN	RET
COMPARE	CMP
TEST	TST

## **Program Interrupt**

- The program interrupts are used to handle a variety of problems that arise out of normal program sequence.
- Program interrupts are used to transfer the program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.

## **Types of Interrupts**

*There are three major types of interrupts*

1. External interrupt
  2. Internal interrupt
  3. Software interrupt
- External interrupts come from Input-Output (I/O) devices or from a timing device.
  - Internal interrupts arise from illegal or erroneous use of an instruction or data.
  - External and internal interrupts from signals that occur in the hardware of the CPU.
  - A Software interrupt is initiated by executing an instruction.

## **Complex Instruction Set Computer (CISC)**

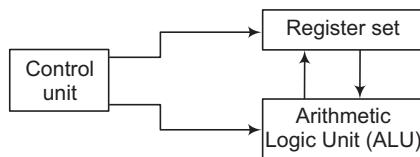
- Computer architecture is described as the design of the instruction set for the processor.
- The computer with a large number of instructions is classified as a complex instruction set computer. The CISC processors typically have the 100 to 250 instructions.
- The instructions in a typical CISC processor provide direct manipulation of operands residing in memory.
- As more instructions and addressing modes are incorporated into a computer, the more hardware logic is needed to implement and support them and this may cause the computations to slow down.

## **Reduced Instruction Set Computer (RISC)**

- RISC architecture is used to reduce the execution time by simplifying the instruction set of the computer.
- In the RISC processors, there are relatively few instructions and few addressing modes. In RISC processors, all operations are done within the registers of the CPU.

## Design of Control Unit

- The function of the control unit in a digital computer is to initiate sequences of micro operations (the operations executed on data stored in registers are called micro operations).
- The number of different types of micro operations that are available in a given system is finite.
- When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.
- Microprogramming is a second alternative for designing the control unit of a digital computer.



Block diagram of a control unit

### Key Points

- The control unit initiates a series of sequential steps of micro operations. During any given time certain micro operations are to be initiated, while others remain idle.
- The control variables at any given time can be represented by a string of 1s and 0s called a control word.

## Peripheral Devices

- The I/O system provides an efficient mode of communication between the central system and the outside environment.
- Programs and data must be entered into computer memory for processing and results obtained from computations must be displayed for the user. The most familiar means of entering information into a computer is through a type writer-like keyboard. On the other hand the central processing unit is an extremely fast device capable of performing operations at very high speed.
- To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks. The information in the disk is then transferred into a high-speed storage, such as disks.
- Input or output devices attached to the computer are called the peripheral devices. The most common peripherals are keyboards, display units and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.

## **Input-Output Interface**

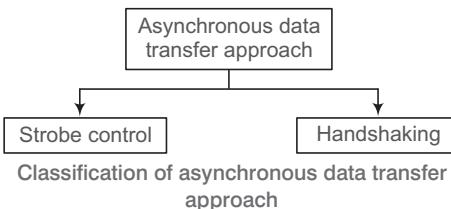
- Input-Output interface provides a method for transferring information between internal storage and external I/O devices.
- Peripherals are connected to the central processing unit with a special communication links (I/O bus).
- The I/O bus from processor is attached to all peripheral interfaces.

### **I/O Communication**

- There is a need of I/O bus for communication between CPU and peripheral devices because of many reasons
  - (a) Data formats of internal memory of CPU and the peripheral devices (I/O devices) are different.
  - (b) Data transfer rates CPU and the I/O devices are different.

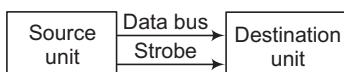
## **Asynchronous Data Transfer**

- The two units such as CPU and I/O interface, are designed independently of each other. If the registers in the interface does not have a common clock (global clock) with the CPU registers, then the transfer between the two units is said to be asynchronous.
- The asynchronous data transfer requires the control signals that are being transmitted between the communicating units to indicate the time at which data is being transmitted.

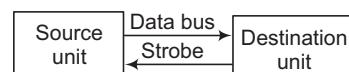


### **Strobe Control**

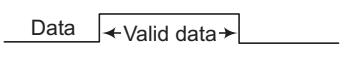
- Strobe is a pulse signal supplied by one unit to another unit to indicate the time at which data is being transmitted.



Block diagram

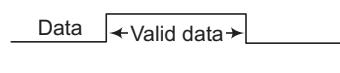


Block diagram



Timing diagram

Source initiated strobe for data transfer



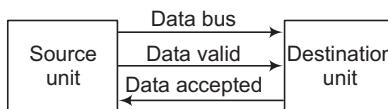
Timing diagram

Destination initiated for data transfer

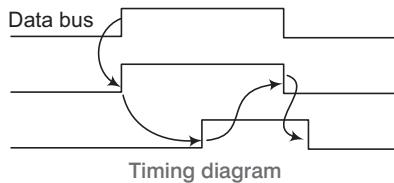
- Strobe may be activated by either the source or the destination unit.
- The strobe pulse is controlled by the clock pulses in the CPU. The data bus carries the binary information from source unit to the destination unit. In source initiated strobe for data transfer, the strobe is a single line that informs the destination unit when a valid data word is available in the bus.
- But in destination initiated for data transfer it informs the source to provide the data. Then source unit places the data on the data bus.

### **Handshaking**

- The disadvantage of the strobe method is that the source unit has no information whether the destination unit has actually received the data item, if the source unit initiates the transfer. But if the destination unit initiates the transfer it has no way of knowing whether the source unit has actually placed the data on the bus. The handshake method solves this problem.
- The basic approach of handshaking is as follows. In handshaking method, there are two control signals unlike strobe control method. One control signal is in the same direction as the data flow in the bus from the source to the destination. This signal is used to inform the destination unit whether there are valid data in the bus. The second control signal is in the other direction from the destination to the source. It is used to inform the source whether it can accept data.



Block diagram of handshaking



Timing diagram

### **Synchronous Data Transfer**

In synchronous data transfer a global or shared clock is provided to both sender and receiver. The sender and receiver works simultaneously.

## Modes of Transfer

The information from external device is stored in memory. Information transferred from the central computer into an external device via memory unit. Hence, this data transfer between the central computer and I/O devices is handled in various modes.

1. Programmed I/O
2. Interrupt- initiated I/O
3. Direct Memory Access (DMA)

### **Programmed I/O**

In this mode, each data item is transferred by an instruction in the program. The transfer is to and from a CPU register and peripherals. In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. Once the data transfer is initiated, the CPU is required to monitor the interface to see, when the transfer can again be made. This is a time-consuming process since, it keeps the CPU busy needlessly.

### **Interrupt-initiated I/O**

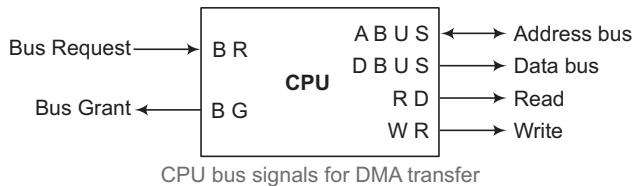
This mode removes the drawback of the programmed I/O mode. In this mode, interrupt facility is used to inform the interface to issue an interrupt request signal when the data are available from the device. In the mean time the CPU can proceed to execute another program.

### **Direct Memory Access (DMA)**

In programmed I/O mode, the transfer is between CPU and peripherals. But in direct memory access mode, the interface, transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory.

The Bus Request (BR) input is used by the DMA controller to request the CPU to get the control of buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus and the data bus. The CPU activates the Bus Grant (BG) output to inform the external DMA that the buses are available. The DMA now takes the control of the buses to conduct the memory transfer. When DMA

terminates the transfer, it disables the bus request line. The CPU disables the bus grant, takes the control of the buses.



## Parallel Processing

Parallel processing provides simultaneous data processing tasks for the purpose of increasing the computational speed of a computer system rather than each instruction is processed sequentially, a parallel processing system is able to perform concurrent data processing to achieve faster execution time and increase throughput.

There are more advantages with parallel processing but it has some issues also. Due to parallel processing, the amount of hardware increases and the cost of system increases. Parallel processing is established by distributing the data among the multiple functional units.

### Flynn's Classification

- MJ Flynn introduced the parallel processing classification.
- This classification considers the organisation of a computer system by the number of instructions and data items that are manipulated simultaneously.

### Key Points

- ◆ The sequence of instructions read from the memory constitutes an instruction stream.
- ◆ The operations performed on the data in the processor constitutes a data stream.

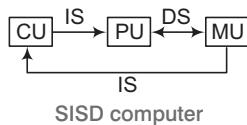
### Flynn's Classification

*Flynn's classification divides computer into four major groups as follows*

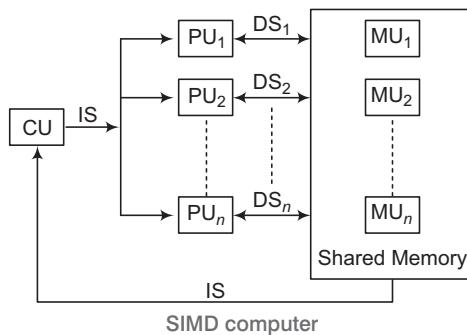
- Single Instruction stream, Single Data stream (SISD)
- Single Instruction stream, Multiple Data stream (SIMD)
- Multiple Instruction stream, Single Data stream (MISD)
- Multiple Instruction stream, Multiple Data stream (MIMD)

**SISD**

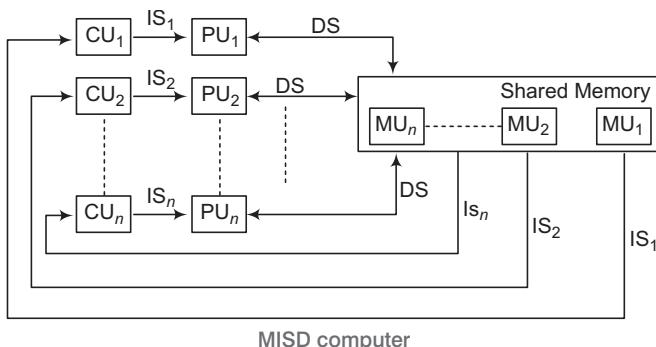
It represents the organisation of a single computer containing a control unit, a processor unit and a memory unit. Instructions are executed sequentially.

**SIMD**

It represents an organisation that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data.

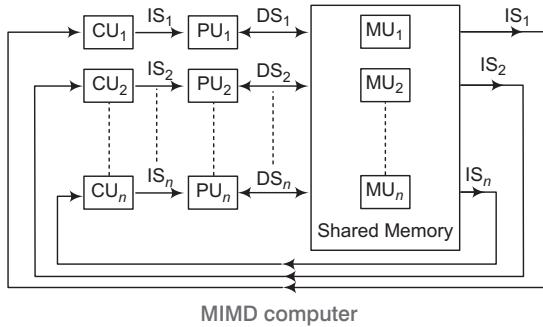
**MISD**

Its architecture contains  $n$  processor units, each receiving instruction streams and providing the same data stream. MISD structure is only of theoretical interest, since no practical system has been constructed using this organisation.



## MIMD

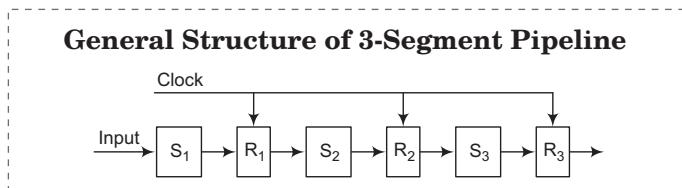
Its organisation refers to a computer system capable of processing several programs at the same time.



## Pipelining

Pipeline processing is an implementation technique, where arithmetic suboperations or the phases of a computer instruction cycle overlap in execution. A pipeline can be visualised as a collection of processing segments through which information flows.

The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment.



- Each segment consists of a combinational circuit  $S_i$  that performs a suboperation over the data stream flowing through pipe. The segments are separated by register  $R_i$  that hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- The behaviour of a pipeline can be illustrated with a space-time diagram. This is a diagram that shows the segment utilisation as a function of time. The horizontal axis displays the time in clock cycles and the vertical axis gives the segment number.

- The space-time diagram shows the four segment pipeline with  $T_1$  through  $T_6$  six tasks executed.

Segment		1	2	3	4	5	6	7	8	9
	1	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$			
2		$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$			
3			$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$		
4				$T_1$	$T_2$	$T_3$	$T_4$	$T_5$	$T_6$	

## Technical Description

- Consider if  $K$ -segment pipeline with clock cycle time  $t_p$  is used to execute  $n$  tasks. The first task  $T_1$  requires a time  $= Kt_p$ .
- The remaining  $(n - 1)$  tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time  $= (n - 1)t_p$ .
- Therefore, to complete  $n$  tasks using a  $K$ -segment pipeline requires  $K + (n - 1)$  clock cycles.
- A non-pipeline unit perform the same operation and takes a time of  $t_n$  to complete each task. The total time required for  $n$  tasks is  $nt_n$ .
- The speedup ( $S$ ) is the ratio of a pipeline processing over an equivalent non-pipeline processing.

$$S = \frac{nt_n}{(K + n - 1)t_p}$$

### Special Case in Speedup

As number of tasks increases,  $n$  becomes larger than  $K-1$ , then  $K + n - 1$  is approximately  $n$ . Then, speedup becomes

$$S = \frac{t_n}{t_p}$$

If we assume  $t_n = Kt_p$ ;  $S = \frac{nt_p}{t_p} = K$

## Instruction Pipeline

- Pipeline processing can occur not only in the data stream but in the instruction stream.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations and consider a computer with a instruction fetch unit and an instruction execution unit designed to provide two-segment pipeline.
- Computers with complex instructions requires other phases in addition to fetch and execute to process an instruction completely.

*The instructions cycle is as follows.*

- |                                     |   |
|-------------------------------------|---|
| • Fetch the instruction from memory | • Decode the instruction                |
| • Calculate the effective address   | • Fetch the operands from memory        |
| • Execute the instruction           | • Store the result in the proper place. |

## Difficulties in Instruction Pipeline

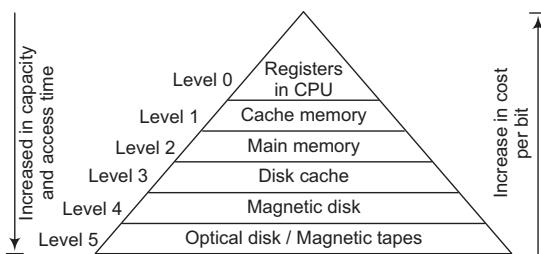
1. **Resource conflicts** It is caused by access to memory by two segments at the same time. Most of these conflicts can be solved by using separate instruction and data memories.
2. **Data dependency conflicts** It arises when an instruction depends on the result of a previous instruction but this result is not yet available.
3. **Branch difficulties arise** It arises from branch and other instructions that change the value of PC.

### Key Points

- ◆ The memory unit that communicates directly with the CPU is called the Main memory.
- ◆ Devices that provide backup storage are called auxiliary memory. i.e., magnetic tapes or magnetic disks.

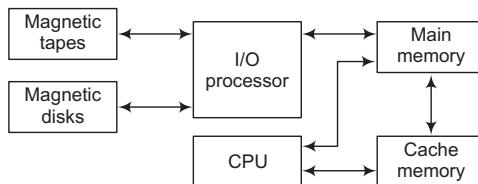
## Memory Hierarchy

- The memory unit is used for storing programs and data. It fulfills the need of storage of the information.
- The additional storage with main memory capacity enhance the performance of the general purpose computers and make them efficient.
- Only those programs and data, which is currently needed by the processor, reside in main memory. Information can be transferred from auxiliary memory to main memory when needed.



## Cache Memory

- A small, fast storage memory used to improve average access time.  
Or
- We can say that cache is a very high-speed memory that is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.



Memory connection in computer system

## Cache Performance

When the processor needs to read or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache.

**Cache hit** If the processor immediately reads or writes the data in the cache line.

**Cache miss** If the processor does not find the required word in cache, then cache miss has occurred.

**Hit ratio** Percentage of memory accesses satisfied by the cache.

$$\text{Miss ratio} = 1 - \text{Hit ratio}$$

## Key Points

- The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

## Main Memory

- The main memory refers to the physical memory and it is the central storage unit in a computer system.
- The main memory is relatively large and fast memory used to store programs and data during the computer operation.
- The main memory in a general purpose computer is made up of RAM integrated circuit.

## Latency

- The latency is the time taken to transfer a block of data either from main memory or caches.
- As the CPU executes instructions, both the instructions themselves and the data they operate on must be brought into the registers, until the instruction/data is available, the CPU cannot proceed to execute it and must wait. The latency is thus the time the CPU waits to obtain the data.
- The latency of the main memory directly influences the efficiency of the CPU.

## Auxiliary Memory

- The common auxiliary memory devices used in computer systems are magnetic disks and tapes.

## Magnetic Disks

- A magnetic disk, is a circular plate constructed of metal or plastic coated with magnetised material.
- Often, both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- All disks rotate together at high speed. Bits are stored in the magnetised surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors.

## Magnetic Tapes

- A magnetic tape is a medium of magnetic recording, made of a thin magnetisable coating on a long, narrow strip of plastic film.
- Bits are recorded as magnetic spots on the tape along several tracks. Magnetic tapes can be stopped, started to move forward or in reverse. Read/write heads are mounted one in each track, so that data can be recorded and read as a sequence of characters.

### Mapping of Cache Memory

- The transformation of data from main memory to cache memory is referred to as a mapping process.
- There are three types of mapping procedures considered
  1. Associative mapping
  2. Direct mapping
  3. Set associative mapping

### Associative Mapping

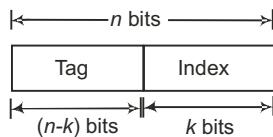
- The fastest and most flexible cache organisation uses an associative memory.
- The associative memory stores both the address and the data of the memory word.



- This memory permits to store any word in cache from main memory.

### Direct Mapping

- Associative memories are expensive compared to Random Access Memories (RAM), because of the added logic associated with each cell.
- The CPU address is divided into two fields



### Set-Associative Mapping

In direct mapping, each word of cache can store two or more words of memory under the same index address. But in set-associative method, each data word is stored together with its tag and the number of tag data items in one word of cache is said to form a set.

### Key Points

- ♦ The number of bits in the index field is equal to the number of address bits required to access the cache memory.
- ♦ In general, if there are  $2^k$  words in cache memory and  $2^n$  words in main memory. Then, the  $n$ -bit memory address is divided into two fields  $k$ -bits for the index field and  $n-k$  bits for the tag field.

# Appendix

---

## Abbreviations

ALU	Arithmetic Logic Unit	CRM	Customer Relationship Management
ANSI	American National Standards Institute	CRT	Cathode Ray Tube
API	Application Program Interface	CSS	Cascading Style Sheet
ARP	Address Resolution Protocol	DBMS	Database Management System
ASCII	American Standard Code for Information Interchange	DCIM	Digital Camera IMages
ASP	Active Server Page or Application Service Provider	DDL	Data Definition Language
ATA	Advanced Technology Attachment	DDR	Double Data Rate
ATM	Asynchronous Transfer Mode	DFS	Distributed File System
BIOS	Basic Input/Output System	DHCP	Dynamic Host Configuration Protocol
Blob	Binary Large Object	DLL	Dynamic Link Library
BMP	Bitmap	DMA	Direct Memory Access
CAD	Computer-Aided Design	DNS	Domain Name System
CD	Compact Disc	DOS	Disk Operating System
CD-R	Compact Disc Recordable	DRAM	Dynamic Random Access Memory
CD-ROM	Compact Disc Read-Only Memory	DSL	Digital Subscriber Line
CD-RW	Compact Disc Re-Writable	DTD	Document Type Definition
CDMA	Code Division Multiple Access	DV	Digital Video
CGI	Common Gateway Interface	DVD	Digital Versatile Disc
CISC	Complex Instruction Set Computing	DVD-R	Digital Versatile Disc Recordable
CLOB	Character Large Object	DVD-RAM	Digital Versatile Disc Random Access Memory
CMOS	Complementary Metal Oxide Semiconductor	DVD-RW	Digital Versatile Disk Rewritable
CMYK	Cyan Magenta Yellow Black	DVI	Digital Video Interface
CPU	Central Processing Unit	DVR	Digital Video Recorder
		ECC	Error Correction Code

EDI	Electronic Data Interchange	ISP	Internet Service Provider
EXIF	Exchangeable Image File Format	IT	Information Technology
FDDI	Fiber Distributed Data Interface	IVR	Interactive Voice Response
FIFO	First In, First Out	JPEG	Joint Photographic Experts Group
FTP	File Transfer Protocol	JRE	Java Runtime Environment
Gbps	Gigabits Per Second	JSON	JavaScript Object Notation
GIF	Graphics Interchange Format	JSP	Java Server Page
GIGO	Garbage In, Garbage Out	Kbps	Kilobits Per Second
GIS	Geographic Information Systems	KVM	Switch-Keyboard, Video, and Mouse Switch
GPS	Global Positioning System	LAN	Local Area Network
GUI	Graphical User Interface	LCD	Liquid Crystal Display
HDD	Hard Disk Drive	LDAP	Lightweight Directory Access Protocol
HDMI	High-Definition Multimedia Interface	LED	Light-Emitting Diode
HDTV	High Definition Television	LIFO	Last In First Out
HDV	High-Definition Video	MAC	Address-Media Access Control Address
HTML	Hyper-Text Markup Language	MANET	Mobile Ad Hoc Network
HTTP	HyperText Transfer Protocol	Mbps	Megabits Per Second
HTTPS	HyperText Transport Protocol Secure	MIDI	Musical Instrument Digital Interface
I/O	Input/Output	MIPS	Million Instructions Per Second
ICF	Internet Connection Firewall	MMS	Multimedia Messaging Service
ICMP	Internet Control Message Protocol	MPEG	Moving Picture Experts Group
IDE	Integrated Device Electronics or Integrated Development Environment	MTU	Maximum Transmission Unit
IEEE	Institute of Electrical and Electronics Engineers	NetBIOS	Network Basic Input/Output System
IM	Instant Message	NIC	Network Interface Card
IMAP	Internet Message Access Protocol	NTFS	New Technology File System
IP	Internet Protocol	OLAP	Online Analytical Processing
IPX	Internetwork Packet Exchange	OLE	Object Linking and Embedding
ISDN	Integrated Services Digital Network	OOP	Object-Oriented Programming
ISO	International Organization for Standardization	OSPF	Open Shortest Path First
		P2P	Peer To Peer

PC	Personal Computer	SMS	Short Message Service
PCB	Printed Circuit Board	SMTP	Simple Mail Transfer Protocol
PDF	Portable Document Format	SNMP	Simple Network Management Protocol
PHP	Hypertext Preprocessor	SOA	Service Oriented Architecture
PNG	Portable Network Graphic	SOAP	Simple Object Access Protocol
POP3	Post Office Protocol	SQL	Structured Query Language
PPP	Point to Point Protocol	SRAM	Static Random Access Memory
PPPOE	Point-to-Point Protocol Over Ethernet	SSD	Solid State Drive
PPTP	Point-to-Point Tunneling Protocol	SSH	Secure Shell
PRAM	Parameter Random Access Memory	SSID	Service Set Identifier
PROM	Programmable Read-Only Memory	SSL	Secure Sockets Layer
PS/2	Personal System/2	TCP/IP	Transmission Control Protocol/Internet Protocol
RAID	Redundant Array of Independent Disks	TFT	Thin-Film Transistor
RAM	Random Access Memory	TIFF	Tagged Image File Format
RFID	Radio-Frequency Identification	TTL	Time To Live
RGB	Red Green Blue	UAT	User Acceptance Testing
RISC	Reduced Instruction Set Computing	UDDI	Universal Description Discovery and Integration
ROM	Read-Only Memory	UDP	User Datagram Protocol
RPC	Remote Procedure Call	UML	Unified Modeling Language
RTE	Runtime Environment	UPnP	Universal Plug and Play
RTF	Rich Text Format	UPS	Uninterruptible Power Supply
SAN	Storage Area Network	URI	Uniform Resource Identifier
SATA	Serial Advanced Technology Attachment	URL	Uniform Resource Locator
SCSI	Small Computer System Interface	USB	Universal Serial Bus
SD	Secure Digital	UTF	Unicode Transformation Format
SDK	Software Development Kit	VCI	Virtual Channel Identifier
SDRAM	Synchronous Dynamic Random Access Memory	VDU	Visual Display Unit
SLA	Software License or Service Level Agreement	VFAT	Virtual File Allocation Table
SMART	Self-Monitoring Analysis And Reporting Technology	VGA	Video Graphics Array
		VoIP	Voice Over Internet Protocol
		VPI	Virtual Path Identifier
		VPN	Virtual Private Network
		W3C	World Wide Web Consortium
		WAN	Wide Area Network

Wi-Fi	Wireless Fidelity	XML	Extensible Markup Language
WWW	World Wide Web	XSLT	Extensible Style Sheet
XHTML	Extensible Hypertext Markup Language		Language Transformation

## **Famous Scientists and their Discoveries**

- **Wil Vander Aalst** Business process management, process mining
- **Hal Abelson** Intersection of computing and teaching
- **Serge Abiteboul** Database theory
- **Samson Abramsky** Game semantics
- **Leonard Adleman** RSA, DNA computing
- **Frances E Allen** Compiler optimization
- **Gene Amdahl** Supercomputer developer, founder of Amdahl Corporation
- **Bruce Arden** Programming language compilers (GAT, MAD), virtual memory architecture, MTS
- **John Vincent Atanasoff** Computer pioneer
- **Ali Aydar** Computer scientist and CEO of Sporcle
- **Charles Babbage** Invented first mechanical computer
- **Roland Carl Backhouse** Mathematics of program construction
- **John Backus** FORTRAN, Backus-Naur form, first complete compiler
- **Rudolf Bayer** B-tree
- **Steven M Bellovin** Network security
- **Tim Berners-Lee** World Wide Web
- **Daniel J Bernstein** Qmail, software as protected speech
- **Manuel Blum** Cryptography
- **Barry Boehm** Software engineering economics, spiral development
- **George Boole** Boolean logic
- **Bert Bos** Cascading Style Sheets
- **Jonathan Bowen** Z notation, formal methods
- **David J Brown** Unified memory architecture, binary compatibility

- **Per Brinch Hansen** Concurrency
- **Sjaak Brinkkemper** Methodology of product software development
- **Tracy Camp** Wireless computing
- **Vinton Cerf** Internet, TCP/IP
- **Peter Chen** Entity-relationship model, data modeling, conceptual model
- **Edgar F Codd** Formulated the database relational model
- **Stephen Cook** NP-completeness
- **James Cooley** Fast Fourier Transform (FFT)
- **Seymour Cray** Cray Research, supercomputer
- **Andries van Dam** Computer graphics, hypertext
- **Christopher J Date** Proponent of database relational model
- **Richard DeMillo** Computer security, software engineering, educational technology
- **Dorothy E Denning** Computer security
- **Vinod Dham** P5 Pentium processor
- **Whitfield Diffie** Public key cryptography, Diffie-Hellman key exchange,
- **Edsger Dijkstra** Algorithms, Goto considered harmful, semaphore
- **Susan Dumais** Information Retrieval
- **Brendan Eich** JavaScript, Mozilla
- **Philip-Emeagwali** Supercomputing
- **Douglas Engelbart** Tiled windows, hypertext, computer mouse
- **Don Estridge** Led development of original IBM Personal Computer (PC) known as father of the IBM PC
- **Oren Etzioni** MetaCrawler, Netbot
- **David C Evans** Computer graphics
- **Edward Felten** Computer security
- **Tommy Flowers** Colossus computer
- **Robert Floyd** NP-completeness
- **Michael Garey** NP-completeness
- **Seymour Ginsburg** Formal languages, automata theory, AFL theory, database theory
- **Kurt Gödel** Computability - not a computer scientist per se, but his work was invaluable in the field
- **Adele Goldberg** Smalltalk

- **Ian Goldberg** Cryptographer, off-the-record messaging
- **Oded Goldreich** Cryptography, computational complexity theory
- **Shafi Goldwasser** Cryptography, computational complexity theory
- **Gene Golub** - Matrix computation
- **Martin Charles Golumbic** - Algorithmic graph theory
- **James Gosling** Java
- **Paul Graham** Viaweb, On Lisp, Arc
- **Susan L Graham** Compilers, programming environments
- **Jim Gray** Database
- **Sheila Greibach** Greibach normal form, AFL theory
- **Ramanathan V Guha** RDF, Netscape, RSS, Epinions
- **Neil J Gunther** Computer performance analysis, capacity planning
- **Peter G Gyarmati** Adaptivity in operating systems and networking
- **Richard Hamming** Hamming code, founder of the Association for Computing Machinery
- **Juris Hartmanis** Computational complexity theory
- **Martin Hellman** Encryption
- **James Hendler** Semantic Web
- **John L Hennessy** Computer architecture
- **Danny Hillis** Connection Machine
- **CAR Hoare** Logic, rigor, Communicating sequential processes (CSP)
- **John Henry Holland** Genetic algorithms
- **John Hopcroft** Compilers
- **David A Huffman** Huffman coding, used in data compression.
- **Watts Humphrey** Personal Software Process (PSP), Software quality, Team Software Process (TSP)
- **Ivar Jacobson** Unified Modeling Language, Object Management Group
- **Cliff Jones** Vienna Development Method (VDM)
- **Robert E Kahn** TCP/IP
- **Avinash Kak** Digital image processing
- **Richard Karp** NP-completeness
- **Marek Karpinski** NP optimization problems
- **Carl Kesselman** Grid computing

- **Stephen Cole Kleene** Kleene closure, recursion theory
- **Leonard Kleinrock** ARPANET, queueing theory, packet switching, hierarchical Routing
- **Andrew Koenig** C++
- **Andrey Nikolaevich Kolmogorov** Algorithmic complexity theory
- **Robert Kowalski** Logic programming
- **John Koza** Genetic programming
- **Leslie Lamport** Algorithms for distributed computing, LaTeX
- **Manny M Lehman** Laws of Software Evolution
- **Max Levchin** Gausebeck-Lechin Test and PayPal
- **Leonid Levin** Computational complexity theory
- **Richard J Lipton** Computational complexity theory
- **Barbara Liskov** Programming languages
- **Paul Mockapetris** Domain Name System (DNS)
- **Cleve Moler** Numerical analysis, MATLAB
- **Edward F Moore** Moore machine
- **Gordon Moore** Moore's law
- **Hans Moravec** Robotics
- **Mark Overmars** Game programming
- **David Parnas** Information hiding, modular programming
- **Yale Patt** Instruction-level parallelism, speculative architectures
- **David John Pearson** CADES, computer graphics
- **Alan Perlis** Programming Pearls
- **Radia Perlman** Spanning tree protocol
- **Simon Peyton Jones** Functional programming
- **William H Press** Numerical algorithms
- **Michael O Rabin** Nondeterministic machines
- **Dragomir R Radev** Natural Language Processing, Information Retrieval
- **Brian Randell** Dependability
- **Joyce K Reynolds** Internet
- **Dennis Ritchie** C (programming language), UNIX
- **Ron Rivest** RSA, MD5, RC4
- **Colette Rolland** REMORA methodology, meta modelling

- **Douglas T Ross** Structured Analysis and Design Technique
- **Winston W Royce** Waterfall model
- **Rudy Rucker** Mathematician, writer, educator
- **James Rumbaugh** Unified Modeling Language, Object Management Group
- **Carl Sassenrath** Operating systems, programming languages, Amiga, REBOL
- **Mahadev Satyanarayanan** File systems, distributed systems, mobile computing, pervasive computing
- **Ben Shneiderman** Human-computer interaction, information visualization
- **Larry Stockmeyer** Computational complexity, distributed computing
- **Michael Stonebraker** Relational database practice and theory
- **Olaf Storaasli** Finite element machine, linear algebra, high performance computing
- **Christopher Strachey** Denotational semantics
- **Madhu Sudan** Computational complexity theory, coding theory
- **Bert Sutherland** Graphics, Internet
- **Andrew S Tanenbaum** Operating systems, MINIX
- **Avie Tevanian** Mach kernel team, NeXT, Mac OS X
- **Linus Torvalds** Linux kernel, Git
- **Godfried Toussaint** Computational geometry - computational music theory
- **Joseph F Traub** Computational complexity of scientific problems
- **Murray Turoff** Computer-mediated communication
- **Alan Turing** British computing pioneer, Turing Machine, algorithms, cryptology, computer architecture.
- **Jeffrey D Ullman** Compilers, databases, complexity theory
- **Leslie Valiant** Computational complexity theory, computational learning theory
- **David Wagner** Security, cryptography
- **Manfred K Warmuth** Computational learning theory

### Some Recent Inventions

Equipment Name	Created By
The Stark Hand	Mark Stark
The PrintBrush	Alex Breton
Dynamic Eye Sunglasses	Chris Mullin
The Bed Bug Detective	Chris Goggin
The Medical Mirror	Ming-Zher Poh

### Intel 805xx Processor Series

Product Code	Marketing Name (s)	Code Name (s)
80500	Pentium	P5 (A-step)
80501	Pentium	P5
80502	Pentium	P54C, P54CS
80503	Pentium with MMX Technology	P55C, Tillamook
80521	Pentium Pro	P6
80522	Pentium II	Klamath
80523	Pentium II, Celeron, Pentium II Xeon	Deschutes, Covington, Drake
80524	Pentium II, Celeron	Dixon, Mendocino
80525	Pentium III, Pentium III Xeon	Katmai, Tanner
80526	Pentium III, Celeron, Pentium III Xeon	Coppermine, Cascades
80528	Pentium 4, Xeon	Willamette (Socket 423), Foster
80529	canceled	Timna
80530	Pentium III, Celeron	Tualatin
80531	Pentium 4, Celeron	Willamette (Socket 478)
80532	Pentium 4, Celeron, Xeon	Northwood, Prestonia, Gallatin
80533	Pentium III	Coppermine (cD0-step)
80534	Pentium 4 SFF	Northwood (small form factor)
80535	Pentium M, Celeron M 310-340	Banias
80536	Pentium M, Celeron M 350-390	Dothan
80537	Core 2 Duo T5xxx, T7xxx, Celeron M 5xx	Merom
80538	Core Solo, Celeron M 4xx	Yonah
80539	Core Duo, Pentium Dual-Core T-series	Yonah
80541	Itanium	Merced

<b>Product Code</b>	<b>Marketing Name (s)</b>	<b>Code Name (s)</b>
80542	Itanium 2	McKinley
80543	Itanium 2	Madison
80546	Pentium 4, Celeron D, Xeon	Prescott (Socket 478), Nocona, Irwindale, Cranford, Potomac
80547	Pentium 4, Celeron D	Prescott (LGA 775)
80548	canceled	Tejas and Jayhawk
80549	Itanium 2 90xx	Montecito
80550	Dual-Core Xeon 71xx	Tulsa
80551	Pentium D, Pentium EE, Dual-Core Xeon	Smithfield, Paxville DP
80552	Pentium 4, Celeron D	Cedar Mill
80553	Pentium D, Pentium EE	Presler
80554	Celeron 800/900/1000 ULV	Shelton
80555	Dual-Core Xeon 50xx	Dempsey
80556	Dual-Core Xeon 51xx	Woodcrest
80557	Core 2 Duo E4xxx, E6xxx, Dual-Core Xeon 30xx, Pentium Dual-Core E2xxx	Conroe
80560	Dual-Core Xeon 70xx	Paxville MP
80562	Core 2 Quad, Core 2 Extreme QX6xxx, Quad-Core Xeon 32xx	Kentsfield
80563	Quad-Core Xeon 53xx	Clovertown
80564	Xeon 7200	Tigerton-DC
80565	Xeon 7300	Tigerton
80566	Atom Z5xx	Silverthorne
80567	Itanium 91xx	Montvale
80569	Core 2 Quad Q9xxx, Core 2 Extreme QX9xxx, Xeon 33xx	Yorkfield
80570	Core 2 Duo E8xxx, Xeon 31xx	Wolfdale
80571	Core 2 Duo E7xxx, Pentium Dual-Core E5xxx, Pentium Dual-Core E2210	Wolfdale-3M
80573	Xeon 5200	Wolfdale-DP
80574	Core 2 Extreme QX9775, Xeon 5400	Harpertown
80576	Core 2 Duo P7xxx, T8xxx, P8xxx, T9xxx, P9xxx, SL9xxx, SP9xxx, Core 2 Extreme X9xxx	Penryn

Product Code	Marketing Name (s)	Code Name (s)
80577	Core 2 Duo P7xxx, P8xxx, SU9xxx, T6xxx, T8xxx	Penryn-3M
80578	LE80578	Vermilion Range
80579	EP80579	Tolapai
80580	Core 2 Quad Q8xxx, Q9xxx, Xeon 33xx	Yorkfield-6M
80581	Core 2 Quad Q9xxx	Penryn-QC
80582	Xeon 74xx	Dunnington
80583	Xeon 74xx	Dunnington-QC
80584	Xeon X33x3 LV	Yorkfield CL
80585	Core 2 Solo SU3xxx, Celeron 7xx, 9xx	Penryn-L
80586	Atom 2xx, N2xx	Diamondville
80587	Atom 3xx	Diamondville DC

### Intel 806xx Processor Series

Product Code	Marketing Name (s)	Code Name (s)
80601	Core i7, Xeon 35xx	Bloomfield
80602	Xeon 55xx	Gainestown
80603	Itanium 93xx	Tukwila
80604	Xeon 65xx, Xeon 75xx	Beckton
80605	Core i5-7xx, Core i7-8xx, Xeon 34xx	Lynnfield
80606	Canceled	Havendale
80607	Core i7-7xx QM, Core i7-8xx QM, Core i7-9xx XM	Clarksfield
80608	Canceled	Auburndale
80609	Atom Z6xx	Lincroft
80610	Atom N400, D400, D500	Pineview
80611	Canceled	Larrabee
80612	Xeon C35xx, Xeon C55xx	Jasper Forest
80613	Core i7-9xxX, Xeon 36xx	Gulftown
80614	Xeon 56xx	Westmere-EP
80615	Xeon E7-28xx, Xeon E7-48xx, Xeon E7-88xx	Westmere-EX
80616	Pentium G6xxx, Core i3-5xx, Core i5-6xx	Clarkdale
80617	Core i5-5xx, Core i7-6xxM/UM/LM	Arrandale

Product Code	Marketing Name (s)	Code Name (s)
80618	Atom E6x0	Tunnel Creek
80619	Core i7-3xxx	Sandy Bridge-EP
80620	Xeon E5-24xx	Sandy Bridge-EP-8, Sandy Bridge-EP-4
80621	Xeon E5-16xx, Xeon E5-26xx, Xeon E5-46xx	Sandy Bridge-EP-8, Sandy Bridge-EP-4
80622		Sandy Bridge-EP-8
80623	Xeon E3-xxxx, Core i3/i5/i7-2xxx, Pentium Gxxx, Xeon E3-12xx	Sandy Bridge-HE-4, Sandy Bridge-M-2
80627	Core i3/i5/i7-2xxxM, Pentium Bxxx, Celeron Bxxx	Sandy Bridge-HE-4, Sandy Bridge-H-2, Sandy Bridge-M-2
80631	Itanium 95xx	Poulson
80632	Atom E6x5C	Stellarton
80637	Core i5/i7-3xxx, Xeon-E3	Ivy Bridge
80638	Mobile Core i5/i7-3xxxM	Ivy Bridge
80640	Atom Z24xx	Penwell
80641	Atom D2xxx, Atom N2xxx	Cedarview
80647		Haswell
80649	Xeon Phi	Knight's Corner
80650	Atom Z27xx	Cloverview

## Generations of Programming Language

- **First generation programming language** is pure machine code that is just ones and zeros, e.g. 0011001110000101001
- **Second-generation programming languages** are a way of describing Assembly code which you may have already met.
- **Third-generation programming languages** brought many programmer-friendly features to code such as loops, conditionals, classes etc. This means that one line of third generation code can produce many lines of object (machine) code, saving a lot of time when writing programs.
- **Fourth-generation languages** are designed to reduce programming effort and the time it takes to develop software, resulting in a reduction in the cost of software development. (SQL), languages to make reports (Oracle Reports) and languages to construct user interface (XUL).