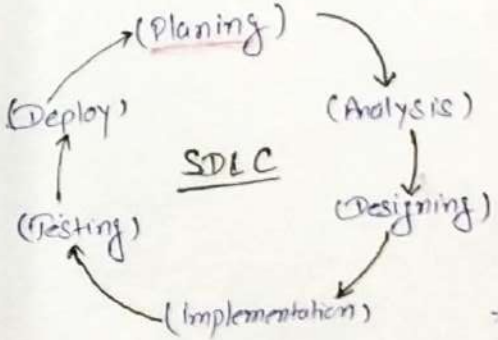
* Software Engineering *

* it is Systematic and disciplined, cost effective techniques for software development.



Classical waterfall	Iterative waterfall	Prototype Model	Incremental Model	RAD Model	Agile Model	Evolutionary Spiral Model.
» Basic » Rigid » Inflexible » Req. are already fixed » Not for the real world » Doesn't have feedback module	» every thing is same as classical but here we add feedback loop. » Problem is well understood (mummy)	» user req. is not clear. » Costly » No early look on Requirment » High user involment » Reusability (Girlfriend)	» Module to module delivery » easy to test and debug » Req. lock (friend)	* Time and cost are fixed » user at all level » Reusebility (Carrer)	→ Risk, Not for the Small project → No early look on Req. → less experience can work (B.Tech degree)	→ Similar as incremental but it use for large project. (friends)

* Agile Model:
i) flexible ii) Advanced
iii) parallel work
iv) process divided into Sprint.
v) used in most of real word (Best friend)

* Methodlogy to devlope software
i) DMAIC: Define, measure Analyze, improve, control
ii) DMADV: Define, measure Analyze, Design, verify

* Spiral model is an iterative Sotware development process that focus on risk analysis and mitigation (reduce to Risk).

* for a software project, the Spiral model was employes it stop when software product is retired/decommissioned.

Functional Requirments:
» Related to working aspect
» Req. that end user specify demands
» Eg: user Reg., Product catalog Shopping Cart, Search fun, Payment getway etc: AMAZON web.

Non-functional Requirments:
» Related to expacted characteristic
» Performance & Scalability
» Relibility and availbility
» Security and confidentiality
» maintainability.

Software Requirment Specification
» it is description of a sotware System to be devloped.
» it layes out function and non-functional requirment of software
» Does not include:
i) Design detail ii) Coding imp.
iii) Testing Plan iv) leagal Contracts
v) Project managment details
vi) user interface mockup

User requirment Specification
» easy & simple to operate
» Quick Response
» Effectively handle operational error.
» Customer support

Data Flow Diagram
» AKA: Bubble chart
» DFD is graphical representation of flow of data through information system.
» Logical DFD :» Physical DFD
» PDFD: Show how System is actually implimented either at the moment (current physical DFD) or how designer inted it to be in Funcure required DFD.

Software design Approaches
① functional oriented deg.
↳ function viewpoint
↳ Top to down decomposition
→ Divide and Conqure
→ DFD is used.
② object oriented deg.
→ Bottom to up approach.
→ UML is used.

Risk Managment
* Reactive Risk:
↳ Reduce damage of potential threat. [Car & life insurance]
* Proactive Risk
→ identify and act acordingly.
[Helmet or sheet belt during driving]
RA = Probability x Security (Risk Assessment)
Risk Control == Reactive Risk
Risk mitigation == proactive Risk

Project Estimation technique
① Basic COCOMO Model
* it compute effort, time and cost as a function of program size. [SLOC, KLOC]

$$Effort = a_1 \times (KLOC)^{a^2} \times PM$$

KLOC: estimated no. of thousand line of code.
a_1, a_2 / b_1, b_2: Coefficients constant

$$T_{dev} = b_1 \times (Effort)^{b2} \times Months$$

↳ time to devlope a Software
Effort: total efforts

↳ cocomo Continue---

* Organic: $E = 2.4 (KLOC)^{1.05}$ PM
Semi-detached: $E = 3.0 (KLOC)^{1.12}$ PM
Embedded: $E = 3.6 (KLOC)^{1.20}$ PM
(PM: product of 5 scale factor)
a_b = multiplicative factor
b_b = exponent factor.

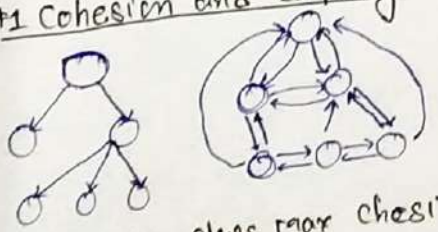
* Duration: $C_b \times (E)^{db}$ (T_{dev})

Person esti(N) = Estimation / Timel

Productivity = $\frac{KLOC}{E} \times$ KLOC/PM

Software testing

#1 Cohesion and Coupling



* To make class max chesive = $n(n-1)/2$
→ bcz it related to each and every method

* To make class min cohesive: 0
→ each class has independant among the method.

* it is used to measure quality of software design.
↑ Coupling & ↓ Cohesion: Difficult to change and test.
↓ Coupling & ↑ Cohesion: easier to maintain and improve

Content, Common, Control, Stamp, data
worst to Best Coupling →
Chamke chamke chand se Din
Coffe cafe day

① Coincidental — logical
procedural — Temporal
Communicational
② Function — Sequence
low to high Cohesion →

* Snake ka muhh sabse jaroori ur khatarnak hota hai whi functional hota v hai.

* Massage coupling is weakest coupling, where two module are connected through a message passing such a method call / function call.

* Pathological coupling is a situation, where two or more module are tightly coupled such that change are difficult

→ In a graph given of module:

Coupling = $\frac{\text{no. of external link}}{\text{no. of modules}}$

Cohesion = $\frac{\text{no of internal link}}{\text{no of method.}}$

#2 white Box testing

* it applied mainly at unit testing level, now even in integration and System testing.

* it focus on internal working and implimentation detail.

Types: [Technique to test]
① Code Coverage analysis
② Control flow testing
③ Data flow testing } CCDM
④ mutation testing
↳ involve small change to source code

Black box testing

* apply for every level of testing.
* focus on functional req. rather than internal implementation

Types: [Technique to test]
① functional testing
② Regration testing
③ usability testing
④ Boundary value analysis
⑤ equivalence partioning
⑥ Exploratory testing.

* helpful for detecting any error ↳ FRUITBEE

Testing.

① Unit testing
② Integration
i) Bang Bang
ii) Top-Down
iii) Bottom up
iv) Mixed
③ System testing
Functional: » Alpha, » Beta, » Acceptance
NON-FUN: » Volume, » Load, » Stress, » Security etc
④ Regression testing
(for maintenance / upgradation)

Alpha: Company or organisation.
Beta: Group of friends/.
Acceptance: for genral public.
→ Alpha & Beta is comes under acceptance testing.

* Boundary Condition testing or analysis is type of black box testing method that focus on edge/boundary of input domain

① Types of maintenance (ACPP)
i) Adaptive: update software so it work under new hardware/software
ii) Corrective: fix defects/error.
iii) Perfective: Enhacing software
iv) Preventive: Prevent from future problem and early improve.

maintenance metrics

» to calculate availabilit of System with given MTBF & MTTR figure

Ava. = MTBF / (MTBF + MTTR)

Eg: mean time blw failure = 20 days
mean time to Repair: 20 days
Ava = 20/(20+20) = 0.9608

Cyclomatic Complexity

M = E - N + 2P
E = no of edge in Control flow graph
N = no of node " "
P = no of Connected component
[my observation = loop + 1]

Reliability

R = (Probability of Success)^N
(N = no of Independant trail)

* Ability of Software to Perform intend function with accuracy / consistency over the time.
R = module of (code * probabilit of sucess.
* it measure through MTBF
⇒ Failure rate = $\frac{\text{no. of failure}}{\text{total operating time}}$

MTBF = $\frac{\text{total operating time}}{\text{no. of failure.}}$

Availablity = $\frac{\text{op. time - downtime}}{\text{total op. time}}$

* Musa and Rayleigh model used for Software reliability

* UML diagm of a class stak is Rep as an attribute.

* use case diagram helps to understand req. of UML

Algorithm

* Step to Step representation of Computer program.

Time Complexity

① `for(i=1; i<=n; i=i+c)` or `for(i=1; i>=n; i=i-c)` } $O(n)$

② `for(i=1; i<=n; i=i*c)` or `for(i=1; i>=n; i=i/c)` } $O(\log n)$

③ `for(i=c; i<=n; i=i^c)` or `for(i=n; i>=c; i=i^{1/c})` } $O(\log\log n)$

Space Comp: $O(1)$ ↑

Asymptotic Notations

	R	S	T
Big(O), $f(n) \le C \cdot g(n)$	✓	×	✓
Omega(Ω) $f(n) \ge C \cdot g(n)$	✓	×	✓
Theta(θ) $C_1 g(n) \le f(n) \le C_2 g(n)$	✓	✓	✓
Small(o) $f(n) < C \cdot g(n)$	×	×	✓
Small(ω) $f(n) > C g(n)$	×	×	✓

Comparision of various time comp.

$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2})$

$O(n^{1/2}) < O(n) < O(\log n) < O(n^2)$

$O(n^2) < O(n^3) \dots O(n^k) < O(2^n)$

$O(2^n) < O(n^n) < O(2^{2^n})$

decreasing constant < logarithmic < polynomial < exponential.

(Desi Chadha Kei laute pure exponential)

Solving the Recurrence Reln

i) Substitution method

ii) Recursive tree method

iii) master Theorem.

* master theorem is used if the subproblem are of equal size.

$T(n) = aT(n/b) + f(n)$

$(a \ge 1 \text{ & } b > 0)$

Case 1: if $n^{\log_b a}$ Polynomial bigger than f(n) then $T(n) = O(n^{\log_b a})$

Case 2: $n^{\log_b a} = f(n)$: Asymptotically Then $T(n) = O(n^{\log_b a} \times \log_b n)$

Case 3: $f(n) > n^{\log_b a}$: Polynomially then $T(n) = O(f(n))$

Most used Recurrence Reln

» $T(n) = 2T(\sqrt{n}) + c \to O(\log n)$

» $T(n) = T(n-1) + c \to O(n)$

» $T(n) = T(n-1) + n + c \to O(n^2)$

» $T(n) = 2T(n-1) + c \to O(2^n)$

» $T(n) = 2T(n/2) + c \to O(n)$

» $T(n) = 2T(n/2) + n \to O(n \log n)$

» $T(n) = T(n/2) + c \to O(\log n)$

» $T(n) = T(\sqrt{n}) + c \to O(\log\log n)$

* Tower of Honai

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

$T(n) \ge 2^n$

* Stable sorting: identical element of i/p arry must occupy same order in result of sorting algo.

Divide and Conquer

① Binary Search ⑪ Quick Sort

⑪ merge sort ⑫ max - min Algo

⑪ Stasser's integer multiplication

⑪ Stasser's matrix multi.

viii) Fast exponential Algo.

* $T(c) = O(n)$, $S(c) = O(n)$ in finding max/min

* No of Comparision: $\frac{3n}{2} - 2$

* no of Comparision in best case of merging two Sorted subarray ...

Comparision = $\min(m, n)$

worst Case = $\max(m+n-1)$

* Linear Search

$T(c) = 1$ in Best Case

$T(c) = O(n)$ in worst case/Avg

* Binary Search

$T(c) = 1$ Best case

$T(c) = O(\log n)$ worst Case

	Best	Average	worst	Comparision	Swap	Recurrence
Quick	$n\log n$	$n\log n$	n^2	n^2 / $n*\log n$	$n\log n$	$T(k)+T(n-k-1)+O(n)$
Heap	$n\log n$	$n\log n$	$n\log n$	$n\log n$	$n\log n$	$2T(n/2)+O(\log n)$
Merge	$n\log n$	$n\log n$	$n\log n$	$n\log n$	—	$2T(n/2)+O(n)$
Selection	n^2	n^2	n^2	$n(n-1)/2$	$n-1$	$T(n-1)+O(n)$
Bubble	n	n^2	n^2	$n(n-1)/2$	$n(n-1)/2$	$T(n-1)+O(n)$
Insertion	n	n^2	n^2	$n(n-1)/2$	$n(n-1)/2$	$T(n-1)+O(n)$

$C(n-1)$ - when array is sorted. → $O(\text{level})$

QHMSBI : NNYNYY : Stable

QHMSBI : YYNYYY : Inplace

↳ if Sorting algo use same i/p array to sort list then inplace sorting algo.

* matrix multiplication time Complexity : $O(n^3)$

* merge Sort is least dependant on intial order of input.

Swapping in bubble sort:

A[7] = {8, 22, 7, 9, 31, 5, 13};

» Require for ASC order.

key : 8 22 7 9 31 5 13

swap 2 4 1 1 2 0 0

key se kitna ur chota element

total = 10 swapping need.

» Require for Dsc order

key : 8 22 7 9 31 5 13

swap : 4 1 3 2 0 1 0

total : 11 swap

(key se kitna bda element)

» Binary/merge/Quick : D & C

» Buble/Insertion/selection is Comparision based (S & C)

» Binary : $T(n) = T(n/2) + O(1)$

» Linear : $T(n) = T(n-1) + 1$

* Selection sort take min no. of swap.

* QIB : $O(n^2)$ Selection : $O(n)$ Heap/Merge : $O(n\log n)$: Swap

Greedy Technique

Application:
i) knapsack problem
ii) Job sequence with deadline
iii) Huffman coding
iv) optimal pattern
v) minimum cost Spanning tree
vi) Sigle source shortest path.

MEST: KP
i) Kruskal Algo ii) prism

SSSP:
i) Dijkstra ii) Bellman ford

* Floyd-warshall is all pair shortest path problem. {O(n³)}

* Greedy method is design technique for solving problem where result making a sequence of decisions.

* Simple graph: Graph with no self loops and no parallel edges.

* no of undirected graph possible for n vertices = n(n-1)/2

Knapsack >> O(nlogn)
Job sequence deadline >> nlogn / n² use
Optimal merge pattern >> O(nlogn)
Huffman coding >> O(nlogn)
Prims with any vertex >> O(E+V)logV
Prims, Sorted arry and >> O(EV)
Adjacent list.
Kruskal after take min. edge > O(ElogV)
Kruskal edge is already sorted > O(E+V)

Dijkstra Algo:

>> using min heap & adjancy list: O(E+V)logV
>> using adjacency matrix & min heap = O(V²ElogV)
>> using " list & unsorted array = O(V²)
>> using " " & sorted Doubly LL = O(EV)

Bellman ford = O(EV)

Heap and Graph traversal

* In complete k-ary tree if every internal node has exactly k children then (k-1) key contain no leaves if there are n internal node no. of leaves = n(k-1)+1

* No of leaf node in a tree of n node with each node having 0 or 3 child.
= (2n+1)/3

* min Heap: CBT + parent min
* Max Heap: CBT + max parent
* logn is time complexity for all type of operation in both.

* Dense graph: Complete Connected graph.
* BFS used queue DS.
* DFS used Stack DS.
* BFS & DFS can be used to find no of connected components and T(n) = O(m+n)
* BFS & DFS used for both weighted and unweighted graph:
Find the path: DFS
Find shortest path: BFS

Dynamic programming

* used to solve problem of optimality design technique.

* Fibonacci >> T(n) = O(nlogn)
* LCS >> T(n) = O(mn) (Longest common sub sequence)
* MCM >> T(n) = O(n³) matrix chain multiplications
* 0/1 Knapsack: T(n) = O(PQ)
* sum of subset: T(n) = O(nm)

⇒ DP is used to find all pair of shortest distance in a graph: floyd-warshall

Imp key concept overall

* OK JOB nlogn Sath kr
ElogV or (E+V) Sorted hai
Pr (E+V)logV or EV Sorted h.

* Sum of "MN" & LCS = 0/1K
Pr "nlogn" (inn logo me)
fibonanice ko "n³" se
Matrix multiple kr diya.

* Kadane Algorithm is used to find max sum subarray in an array.

* Backtracking is used to solve 4 Queen problem.

* Bubble Sort: Slowest Sorting procedure

* DFS is most effiecient to check cycle is present in graph or not?

COMPILER DESIGN

* Convert High to low level language. Eg "C" to Bytee code.
* there are 6 phase of Compiler.

① Lexical Analyzer
» Regular expression and Finite automata are used to design.
» Check Spelling mistake.
» Divide Source code into Stream of token.

② Syntax Analysis phase
» token[Input] → Parse tree[O/P]
» CFG and PDA used to design
③ Semantic Analysis phase
» Typing checking done.
④ Intermediate code Generation
» enforce reusability & portability
⑤ Code optimization
⑥ Symbol table
» Store meta data about data.

[Phase 1 2 3 = Error detection.
Phase 4 5 6 = translation.]
» it is Data Structure that contain information about identifiers and Constant present in the program.

* lexical Analyzer also Called Scanner.

Q) int a = 4;
/* ISRO */
printf("I got the Job");

no of token: 10
→ Comment are ignored
"_" → 1

Q) if (z == a) function (10);
token = 11

A) abcd + C 2 - 5 + X b / 2 - + ;
token = 14

→ Can't detect all Syntax & Semantic error. [Lexical]

* Syntax (Parser) *
- Top-down
  - with full backtracking
  - without backtracking
    - Recursive descent
    - non-Recursive descent [LL(1)]
- Bottom up

Bottom-up
- Operator precedence operater
- LR Parser.
  - LR(0) SLR(1) CLR(1) LALR(1)

Check lang belong to Grammer or not also vice-versa.
* 1 Lang → Many Grammer
* 1 Grammer → 1 Language

Derivation

	S	S
1) S → ABC	ABC	ABC
A → a	aBC	ABd
B → b	abC	Abd
C → d	abd	abd
	(Left most) ↙	(Right most) ↗

⇒ T → U / T * O
P → Q + P / Q

* + is Right Associative while * is left ".
⇒ E → E * F } - has high
F → F - F } precedence than * }

Ambiguous vs unambiguous

* one String has more than 1 parse tree exist is Ambiguous.

Eg ⇒ i) E → E+E / E*E / n
ii) S → AaAb / BbBa
A → ∈
B → ∈
iii) S → ∈ / a / as

* All terminal transition = Shif action
* Non-terminal transition = Goto action
* Any ambiguous grammer is not LR(0)
* Any ambiguous grammer S/R conflict or R/R conflict occur in Parsing table.
[S = Shift, R = Reduce]

Type of Grammer

⇒ Regular Grammer [Type-3]
* Right linear Grammer
* A → xB / x
* Left linear Grammer
A → Bx / x

⇒ Context Free Grammer [T2]
* A → α, α ∈ (V+T)*
Eg: A → BC || B → a || C → b

⇒ Context Sensitive Grammer [T1]
* α → β, |α| ≤ |β|, α, β ∈ (V+T)+
Eg: Sa → aB || S → b

⇒ Unrestricted Grammer (T0]
* α → β, α ∈ (V+T)+, β = (V+T)*
Eg: aBC → a
[V → variable, T = terminal]

L Regular Lang
LHS (α) = only one non-terminal
RHS (β) = Atmost one non-terminal
Eg: S → as | a S → sa | b.
S → as | sb | a Combination one not allowed
↓ RL ↓ LL

L Linear Grammer.
S → asb | ab
↳ Linear but not Regular.
↳ Every RG is Linear but every Linear can't be RG.

# Number of State Relation
CLR(1) ≥ LALR(1) = SLR(1) = LR(0)
(CLR LALR SLR LR)

# Power of parser
LR(0) < SLR(1) < LALR(1) < CLR(1)
* T(c) LR parser = O(n)
* T(c) operater precedence = O(n
* T(c) of CYK algorithm = O(n^3)
* LR is more powerful than Operater precedence.
* Ambiguous grammer is suitable for Operater precedenc parser.
* Ambiguity may or may not be removed from AG.
* Any operater is left Recursive and Right Recursive both then the grammer is ambigous.
E → E+E / E*E / n
* Bottom up is more powerful than top down parser.

LL(1) Grammer detection:-

* Single production grammer: LL(1)
* $A \to \alpha_1 | \alpha_2 | \alpha_3 | \dots | \alpha_n$
* $\to$ first(α_1) $\cap$ first(α_2) $\cap$ first(α_n) $= \phi$ then LL(1)
* $\to A \to \alpha | \epsilon$
  first(α) $\cap$ follow(A) $= \phi$ then LL(1)
* Left Recursive grammer are not LL(1)
* Ambiguous Gr. are not LL(1)
* Common prefix Gr. not LL(1)

Eg: $S \to AB$
$A \to a \| B \to a$ Yes

* If Grammer LL(1): Unambiguous

First and Follow

$S \to AB | CD$
$A \to aA | a$
$B \to bB | b$
$C \to cC | c$
$D \to dD | d$

	First	Follow
S	a, c	$
A	a	b, $
B	b	$
C	c	d
D	d	$

* Left Recursive is not LL(1)

$S \to Sa | b$ → Left Recursion.
$S \to aS | b$ → Right "
$S \to aSb | a$ → Middle "

Direct LR: $A \to A\alpha | \alpha$
Indirect LR: $A \to Ba | b$, $B \to Ab | a$

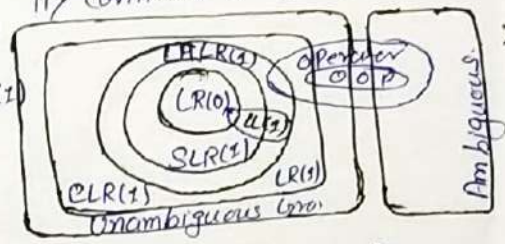
* After removing Left Rec. a grammer need not necessarily be LL(1).
* Common prefix grammar are not LL(1) but Left factoring (Removal of LR) grammer may result into LL(1) grammer.

$S \to ab | ac$ [Common prefix]

* In power:

LL(k) > LL(3) > LL(2) > LL(1) > LL(0)
All are unambiguous.

* Predictive parser is not suitable for.
  i) Left Recursive grammer
  ii) Common prefix grammer

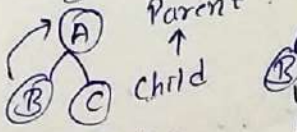


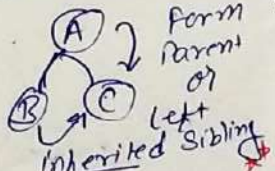
Semantic Analysis

→ Context Sensitive Analysis.
→ Semantic Analysis = Designing of Syntax Directed translation
→ SDT = CFG + Semantic Rule

↔ Why SDT is required?
i) Detection of Semantic error
Eg: Divide by zero.
ii) Type checking
iii) Type Casting.
iv) Symbol table Construction.
v) & intermediate code

* Type of attributes
→ S-attributed SDT
using only synthesized att.
→ L-attributed SDT
using both Synthesized & inherited





* Every S-Attributed is L-Attributed

Key point

» Reduce-Reduce Conflict
  * LR(0) mei hota h
  * SLR(1) mei nhi
» Shift reduce conflict (SR)
  * LR(0) mei hota hai
  * SLR(1) mei nhi

* SDT Application
  > Evaluate an expression
  > Convert infix to prefix/postfix
  > Produce parse tree
  > Produce intermediate rep.

Example of inherited & synthesized

> $S \to A\overset{\downarrow}{B}$
  $\{Ax = f(Bx | Sx)\}$ { Inherited }
> $S \to \overset{\uparrow}{A}B$
  $\{Sx = f(Ax | Bx)\}$ { Synthesized }

Intermediate, Code optimization

↔ Optimization
1) Constant Folding
   $x = 2*3+y$, $x = 6+y$: YES
   $x = 2+y*3$ Can't fold
2) Copy propagation
3) Strength Reduction
   $x = 2*y \Rightarrow x = y+y$ (* → +)
4) Dead Code Elimination.
   [Line of code that never excute during execution, so dit such code]
5) Common sub expression elimination
   DAG is used to eliminate common sub expression.
   Eg: $x = (a+b) + (a+b) + c$;
   $x = t_2 + t_2 + c$ [$t_1 = a+b$]
6) Loop optimization
   → remove unused line of code From loop.
   → merge two loop
   → loop unrolling etc
7) Peephole optimization
   → Applied to intermediate code
   → following optimization can done
   i) Redundant Instn elimination
   ii) Flow of control optimization
   iii) Algebric simplification
   iv) Use of machine idioms
   v) Strength reduction vi) Dead code optimization

PYQ - Section

* SLR(1) & LALR(1) parsing table for CFG.
  Goto part & shift entry same may differ
  Reduce entry & error entry

two pass assembler

task performed in 1st pass:
→ Assign add. of all statement
→ save the value
→ processing of assembler directive
→ table Creating, Add literal to table
→ Allocate space for literal
→ Compute total length of program.

task performed in 2nd pass
→ Generate object oriented code
→ writting / printing assembly listing.
→ Generate code for all load and store instruction.

* loader always reside in RAM.
* Top-down: Left most derivation
* Bottom up: Reverse of RD.
* In bottom-up evaluation of syntax directed definition inherit attribute can be evaluted if the definition is L-attribute.
* time to convert ambiguous to unambiguous, Language must remain unchanged.

* Max no. of ~~token~~ reduce move taken by a bottom up parser with no-epsilon and unit production to parse a string with n-token is N-1.

* Shift reduce parsing belong to a class of bottom up parsing.

* A/c to operator precedence Grammar two production should not be Adjacent and not be eps production.

* Every SLR Grammar is unambiguous but not every unambiguous grammar is SLR.

* LR(1) is sufficient for deterministic context free lang.

* purpose of using intermediate code compiler increase the chance the machine independent code optimize in other compilers.

* peephole optimization is form of local optimization.

* Software pipelining is a method that used to optimise loop, in parallel hardware pipelining.

* Symbol table is not intermediate code representation.

* Peephole is machine dependent and applied on small and target code.

SDT Story

Samastipur Direct train hai, wha ja kr lardke ka, Cast type, ~~ICS~~ type checking, aur ye dekh lenge kai semantic error (Samas mai koi error hai ki nhi) fir SDT se train pakna hai.

(Evalute the expression) ~~Street~~ then Convert infix to pre/post fix also produce tree and finally Intermiadal code generate.

* Out of order execution means Dynamic execution
  - Software pipelining is example of OOOE

* Incremental Compiler: Compiles only those portion of source code that have been modifird.

DIGITAL LOGIC

- isse Spiral [Sunrise] wala se sh Revise kr lo.
- PYQ Note Dairy se revise kr lena.
- isko 2no ka pdf bana kr rakh lena. — Completed —.

#C PROGRAMMING

Memory format

i) little endian [LSB – MSB]
ii) Big Endian [MSB – LSB]
7 is represented in..
Big En: 00000111
Little En: 11100000

* for modulus (%) operation operator must be integer type [both operator] other wise error will be raised
Eg: 4%.2 = 0

Keypoint regading module:

1%.2 = 1 ; 5%.8 = 5
(Chota jo rahega usse likh lenge)
0%.Anything = 0 , X%.X = 1

* Just like rational operater the result of every logical operator is either 0 or 1
* logical not is unary operator.
*(!) Convert 0→1, 1→0

Bitwise Operater.

work on binary rep. of number.
faster than arithmetic operator.
• AND (&) • OR (|) • XOR (^)
• Left shift (<<) • Right Shift (>>)
• Bitwise Not (~)
Eg: -x has o/p as -(x+1)
Printf(~2) ⇒ -(2+1) = -3
* Comma has lowest precedence among all operator in C language.

• Left Shift
a = 5
b = 2
(a<<2)
101 <<2
↳ 10100

Another Ex:
5 = 00000101 , 5<<2
00010100 = 20
20 = 00010100 , 20>>2
= 5

• Right Shift
a=5, b=2
a >> 2
101 >> 2
001

Sizeof()

→ find no of element present in an array
→ to dynamically allocate block of memory.

Ternary operater.

a = 7>2 ? 3×2 +5 : 6! = 7
a = Left ? Middle : Right
(True / False)
a = 2>5 ? 20 : 30 = 30

Operators

() [] →
! ~ ++ -- + - *
* / %
+ -
<< >>
<< = >> =
== !=
&
^
|
&&
||
?:
= += -= *= etc
,

Associativity

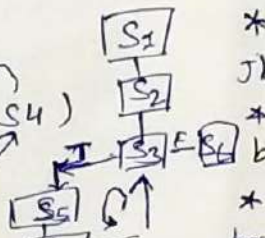
Left to R
Right to left
Left to Right

Right to left
Right to left
Left to Right

PUMA SRE BLACK TRACK

⇒ Top to bottom
AUTO : Right to left Asso.
↳ printf(): return the no of character printed
↳ Scanf(): return no of argument
Eg: int num = printf("ANOBHAV");
Print("%d\n", num)
O/P: ANOBHAV 7
int num = Scanf("%d %d", &x, &y);
Print("%d\n", num);
O/P: 2

For loop S1;
for(S2; S3; S4)
S5;
Switch S6;
while Do-while



* Goto Statement is harmful because make difficult to verify program.
• All following give error (CTE)
i) if() ii) while()
iii) if(1||-); iv) for(;0;);
v) if(0||-);
• Run in compiler
for(;;) → Assume true
if(1,2,3,4), only Condition Matter
↳ Same for while and for
* First Assign then check
if(x=5)
* S1, S2, S3; → S1 S2 S3 run sequentially

<Break> early exist from loop
<exit(0)> Terminate prg. Succesful
<exit(1)> terminat prog. unsuccessful
<Continue> Skip iteration of loop.
<Goto> Jump to given instn
*Auto used as local variable
*%.3f : take 3 place after point
Eg: 0.000
* Perform Left Shift, n×2 (no of LS)
* Perform Righ Shift, n%.2
(Agar 1 LS krna hai toh)
* Global var tab access hoga Jb local var nhi hoga.
* local var func call time bnta hai fir distroy ho jta h.
* Static var local mai ek baar bnta hai fir hamesa uss pr operation perform hota h.
* C lang. mai CSL use hota h.

	Auto	Register	Static	Global
Memory Region	Stack	Stack	Data	Data
Lifetime	Block	Block	Program	All p.
Scope	Block	Block	Program	p.
intial value	Garbage	Garbage	0	0
Declaration	1	1	≥1	≥1
definition.	1	1	1	1
local/global	local	local	L/G	G

int x = 1;
void foo() { int x = 2;
bar(); }
void bar() { printf(x); }
int main() {
foo();
return 0;
}

Static scoping refer Global variable where as DS use use calling function

bar() = 1 : Static
bar() = 2 : Dynamic

□ Array's

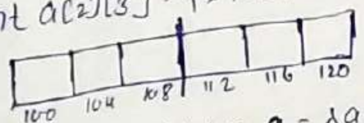
* C langage support static array &A[i] = A + i × size of element.

* In array delclaration:
→ it mandatory to provide size of array but we can omit only first dimension [if element is ddc properly]

Eg: int A[]; valid
int A[][3] = {1,2,3,4,5,6}
int A[][2]; invalid

* Exception: with & operator array name is treated as object not address.

int a[2][3] = {1,2,3,4,5,6}



Printf("%u", a); 100 a = &a[0]
P("%u", &a); 100 = full A to point
P("%u", a[0]); 100 a[0] = &a[0][0]
P("%u", a+1); 112 = &a[0] + 1 × 12
P("%u", &a+1); 124 = 100 + 1 × 24
P("%u", a[0]+1); 104 = &a[0][0] + 1 × 4

* (a[0]+1) = Valural(ML a[0][1])

* whenever we passed array as function "OR" a pointer passes through call by refrence.

* arr++ is invalid but ptr++ are valid.

a[i] = *(a+i) = *(i+a) = i[a]

□ Pointer and Array → Continue →

* int *p[10]

* int(*p (char *a)) [10]
→ p is a 10 element of pointer to integer.
→ p is function that take an argument as pointer to char and return pointer to a 10 element integer array.

↳ int A[100];
* A → Name of array
* &A → Add of array
{should be same.

* A+0
* &A[0]
{ Add of 0th element.

* A+i = &A[i] = Add of 'i' th element
* A[i] = *(A+i) = value of 'i';
* &A[i] = A + i × size of element

↳ int A[2][3];
&A[0] → Add of Row '0'
&A[1] → Add of Row '1'
&A[i] = A + i
&A[i][j] = A[i] + j

* int A[5]; → Local (All Garbage) / → Global (all 0)
* int A[5] = {10,20,30,40,50}
* int A[5] = {10,20} vr sab '0' ley lenge.
* int A[] = {10,20,30,40,50} [Calculate total no of element itself]
* int [2][] = {1,2,3,4,5,6}
↳ Compilation error.

→ A var that hold the add of another var.
→ All pointer are of same size

Arithmetic operation on pointer
• (P+i) = P + i × size of data pointer
• P+ Constant / P− Constant (A); P* Constant / P / Constant (NA)
* P+q → Not possible
* P−q → Possible
* P*q or P/q → NA

Double pointer: **P
Point to another pointer

* (*P)[5] → P is pointer to array element of 5

* int *p[5] → p is array which point containt 5 element that point to an integer.

Problem with pointer

① wild pointer / un intizalied.
int *P; P [Garbage]
*p = 20; ↳ *garbage : Invalid

② Null pointer
int *p = NULL; P [null]
*p = 20; *(Null) → ✗ (error)

③ Dangling pointer → Promble occur (when auto/Register variable)
```
int * f()
{ int x = 10;
  return &x
} → with end of function x get deleted. but
void main()   P still pointing to x
{ int *p;     ↳ {invalid}
  P = f();
}
```
↳ Dangling pointer.
→ [x dlted] → invalid error

(II) lost memory:
```
main ()
{ int *P
  P = malloc(4)
  P = malloc(6)
  free (P);
}
```
→ this memory is lost when the new block is assigned

□ String or array of char
→ Sequence array of char ending with '\0' null termination

way to store ---

[Array of char]:
char A[] = "gate" [g|a|t|e|\0]

[char pointer]
char *A = "gate" [g|a|t|e|\0]

* %c → print only 1 char. (has to provided with char itself)
* %s → print untill "\0" [has provided with start Add.]

[has provided this string literal return "gate" only the add. of which it is store in memory.

□ function

Call by value
→ int sum (int a, int b) formals
return a+b;
c = sum (x, y) actuals
Actuals and formals have diff memory. so formal can't change actual

→ Call by ref. / Add.
int sum (int *a, int *b) F
return (*a + *b)
c = sum (&x, &y) A
Actual and formal have diff memory, formal can change actual

Recursion

↳ function that call itself.
↳ Stack can support recursion
↳ Same fun have more activation

```
Int x = 10;
void f()
{ if (x>0) {
    x--;
    printf("%d", x);
    f(); }
}
```
op: 9 8 7 6 5 4 3 2 1 0

① while(true){ }
infinite loop: the program will run indefinitely will not crash

② foo() { foo(); } — Infinite Recursion
Stack will overflow and program will crash.

Note:-
In C, if there are many variable having the same name then var with nearest scope is visible.

function(function());
function taking function as parameter.

Strick evaluation: the parameter are first evaluted before function call.

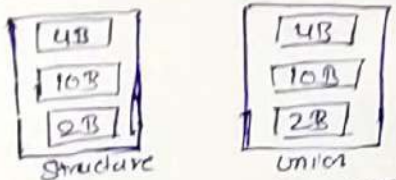
Structure

Collection of element of hetrogenius type.

```
Struct tagname
{ m1 decl
  m2 decl
  ...
}
```

Size = total size of all member

- Union (Similar to Structure)



Str Size = (2+4+10)B = 16B
Union size = Max(2,4,10)B = 10B
Struct: diff. memory for all the member
Union. Same memory for all the member. → any one var exist at a time

Note: Struct & union can be nested at any level.

Note:
Const int *p; value(*p) = Constant.
int Const. *p; value = constant
int * Const P; Add(P) = Constant
Const int * Const P;
↳ value(*p) and Add(P) both are Constant.

Constant: Must be initialized at dlc.
* Can't be modified through out execution.
* Can't be used as L value.

SCOPING

① Static Scope
the scope of var and function depend on structure or Syntax.
→ Access link is know before execution

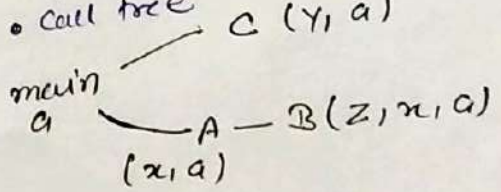
② Dynamic Scope
the scope of var and func. depend on call sequence and call tree.
Access link: know during execution.

```
Int a
Procedure A
  begin
    int x
    B()
  end
Procedure C
  begin
    int Y
    Procedure B
      begin
        int Z
      end
  end
```

```
main
begin
  C();
  A();
end
```

Statically Scoped
Scope of A: a, x
Scope of B: Z, Y, a
Scope of C: Y, a
Scope of main: a

Dynamically Scoped
• call tree



Pointer: can be reassigned after it is established
Reference cannot be reassigned after it establishment.
Pointer can be null but reference can't be null, it may valid or unvalid.

DATA Structure : ARRAY

* All element store in Con secutive memory location
Def: Collection of homogenous element.

Find Size

C lang →
A[size], A[10], LB = 0, UB = 9
A[LB : UB];
* Size = UB - LB + 1
* loc(A(i)) = BA + w(i - LB)
(w = size of each element)

Eg: A [0 | 1 | 2 | 3 | 4] (addresses 100, 102, 104 — 108)

A(3) = 100 + 2(3 - 0) = 106

* Linear Search take : O(n) O(1)
* Binary Search : O(log n)
(Array should be Sorted)

2D Array RMO
loc(A[i][j]) = Base + w[(i - LB_i) × n + (j - LB_j)]

2D Array CMO
loc(A[i][j]) = Base + w[(i - LB_i) + (j - LB_j) × m]
Here n = no of row
m = no of Coloumn

* to find max & min no. of comparision required $\frac{3n}{2} - 2$ and Complexity is O(n)
T(c) S(c)

Linked List

* Dlt in Singly linkedlist take O(n) in worst case and O(1) in best case.
↳ when we alredy position that is best case like At beginning or After given node.

* insertion in Singly & doubly linkedlist take O(1) time in most of case but it can take O(n) in case of: insertion at the end & in sorted list.
↳ O(n) Can be optimise to O(1) by keeping extra pointer to tail.

* dlt in doubly linked list take O(1) but at the end it take O(n)
* Linked is not Suitable for binary search, it use only linear search, insertion sort, Radix sort
→ it is useful for polynomial manipulation.

→ Circular doubly linklist has O(1) Complexity on concatention of two list
→ worst case, singly linklist mai tb hota h, jb data nhi hoga or ekdum last mai Present hoga.

* Insertion / dlt / Search in all type of sorted list take O(n).
* dlt in doubly linklist take O(1) time but Singly take O(n) time.
* dlt the last element of the link list depend on the length of link.
* there is no rule of inst/dlt of an element for circular link.

Stack & Queue

Queue : FIFO
Stack : LIFO

* Queue using Circular array take O(1) enque & dequeue
* Queue using linke list: it take O(1) using 2 pointer for enqueue & dequeue.
* Queue using array
↳ enqueue: O(1)
↳ deque : O(n)

⇒ Doubly ended Queue

Types	Enqueue	dequeue
Input Restricted	Rear	Both
output Restricted	both	Front

* Stack using array and linklist both operation can be perform in Constant time [O(1)] but when array is full then it may take O(n).

* Valid Stack permutation = $= {}^{2n}C_n / n+1$
* Invalid SP = $n! - {}^{2n}C_n / n+1$
* two stack in same/single array can cause overflow
top1 = top2 - 1
* Evalution using stack.
Push = 2n + 1, POP = 2n
— Min stack for evalution = 2
* max stack size = n + 1
let no of operands.
* In a Circular linklist insertion involves modification of two pointer, for singly & doubly link list it depends on position of insertion.
* min no of stack of size n required to implement a queue to size n is 2

* implementation of list in linkedlist is more efficient then implementing list in array. {we have
+ linkedlist perform better insertion/ deletion.
* implementation Queue in circular array is more efficient then Queue on linear array.

Infix to postfix

① priorities of operators
^ / * / + − → H to L

② No two operator of same priority can stay together in stack column.

③ lowest priority can't be placed before highest priority.

Eg: A+B*(C+D)/F+D*E
→ ABCD+*F/DE*++

* An operator stack need for infix to postfix.
* Concatenation of two list is to perform on O(1) time in circular linked list.
* stack is used to check arithmetic exp has blanced parenthesis.

Tree : Binary tree

* total node$_n^{(N)}$ = leaf node(L) + internal node(I)
* Max node with height $h = 2^{h+1} - 1$
* min node with height $h = h+1$
* max node at level $L = 2^L$
* min node at level $L = 1$
* no of BT with n unlabeled node $= {}^{2n}C_n / n+1 = \frac{2n!}{(n+1)!\,n!}$
* no of BT with n distinct key $= {}^{2n}C_n / n+1 \times n! = \frac{2n!}{(n+1)!\times n!}$
* Preorder & postorder can provide unique tree if all internal node have max allowed child otherwise inorder is required with n node to get unique tree.

Tree : Complete binary tree

* max node with height $h = 2^{h+1} - 1$
* min node with height $h = 2^h$
* H(tree) = 0 with Single node

Array Representation of tree

↳ Root at index 1
↳ Left child of node at index $i = 2i$
↳ Right child of node at index $i = 2i+1$
↳ parent of node at index $i = [i/2]$

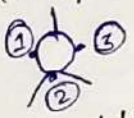
→ leaf node in Full binary tree = internal node + 1 [L = I + 1]
→ total node(N) = 2I + 1
→ in K-tree
$L = (K-1) \times I + 1$
→ no of node with degree 2 in a binary tree = no of leaf node − 1
N = (n − 1)

Binary Search tree

* Inorder is Sorted Sequence in ASC order.
* NO of BST with n distinct key $= {}^{2n}C_n / n+1$
* Searching / insertion / dlt take
Avg: O(log n)
worst: O(n)

AVL tree

* Ser / ins / dlt take O(log n)
* min height of an AVL tree with n node $= \lfloor \log(n+1) \rfloor$
* max height with n node = 1.44 log n
* Avg no of comparision for Sequential Search $= (n+1)/2$
* pre / In / post : 1 / 2 / 3



Tree	max node (h)	min node (h)
Binary tree	$2^{h+1} - 1$	h + 1
Complete BT	$2^{h+1} - 1$	2^h
Full BT	$2^{h+1} - 1$	$2^{(h+1)} - 1$

* A CBT with n-leaf node Contain 2n + 1 nodes
* No of way to insert n element in BST so height of n−1 : $2^n - 1$
* BST is not height balanced tree so it may be left or Right Skewed so insertion or dlt become O(n)

Hashing

* Load factor = total key / total slots
* Space utilization = occupied Space / total slots

Inorder: 2 3 4 6 7 9 13 15 17 18 20
predecessor ↙ ↘ Successor

* In linear hasing, if block factor bfr, loading factor (I) and file buckets (N) are know the no of record will be
[R = I × bfr × N]

B+ Tree

* order = max no of child = max of key + 1
* Degree = order / 2
max key = order × 2 − 1
min key = order − 1
* Min node in an AVL tree with height H(n) =
= H(n−1) + H(n−2) + 1

Data Structure : Heap

Min heap: CBT + node small then child
Max heap: CBT + Parent is ≥ child

in max heap:
* find length of tree : O(log n)
* After applying BS : O(log log n)

* time to find smallest element on a min-heap retrive operation : O(1)

Construction of Heap:
↳ insertion one after another : O(n log n)
↳ using heapify : O(n)

* Infix to postfix conversion
⇒ (A+B/C*(D+C)-F)
⇒ ABC/DC+*+F-

* infix to prefix
→ (A+B)-(C/D)+E
→ [+AB] - [/CD] + E
→ [- [+AB] [/CD]] + E
⇒ +-+AB/CDE =

* Convert postfix to infix
↳ two operand stay in stack.
AB-DE+F*/
→ (A-B)DE+F*/
→ (A-B)(D+E)F*/
⇒ [(A-B)(D+E)*F]/
⇒ (A-B)/(D+E)*F
=

* Prefix to infix
↳ Reverse the infix operation and perform like post to infix.
→ +-*AB/CDE
→ EDC/BA*-+
→ E(D/C)BA*-+
→ E(D/C)(B*A)-+
→ E[(D/C)-(B*A)]+
⇒ E+(D/C)-(B*A)

* Evaluation of postfix
P : 5, 6, 2, +, *, 12, 4, 1, -
⇒ 5, 8, *, 12, 4, 1, -
⇒ 40, 12, 4, 1, -
⇒ 40, 8, -
= 32

* Evaluation of prefix
↳ Reverse the prefix notation.
Q : +, -, *, 2, 2, 1, 16, 8, 5
↳ P : 5, 8, 16, 1, 2, 2, *, -, +
↳ 5, 2, 1, 2, 2, *, -, +
↳ 5, 2, 4, -, +
↳ 5, 2, + → 7

* Insertion Sort :
↳ 2-2 element Compare krege
→ 10, 4, 7, 23, 67, 12, 5
1st pass → 4, 10, 7, 23, 67, 12, 5
2nd pass → 4, 7, 10, 23, 67, 12, 5
3rd pass → 4, 7, 10, 23, 67, 12, 5
etc

* Selection Sort :
→ 64, 25, 12, 22, 11
1st : 11, 25, 12, 22, 64
2nd : 11, 12, 25, 22, 64
3rd : 11, 12, 22, 25, 64

* Bubble Sort
last se First sort krta hai
→ A : 6, 3, 0, 5
1st : 0, 3, 5, 6
2nd : 0, 3, 5, 6 etc.

* Merge Sort
» Strp Split array in two equal part untill two element then Sort one by one.

* Quick Sort : use pivot element to Sort.

D tree Construction in Heaffman *

Insertion Sort mai Left to Right sort hoga.
Selection Sort mai Left to Right sort hoga.
Bubble Sort mai Right to Left sort hoga.

Digital logic

+ whenever logic are designed by TTL the floting terminal always work as a high.

TTL = High, ECL = Low & fastest

Ass: $(A+B)+C = A+(B+C)$

Comm: $A+B = B+A$

* Every logic gate follow comm ulative other than NAND & NOR every gate follow associative.

$\overline{(A+B)}+C \neq A+\overline{(B+C)}$

AND Gate: O/P = 0 || Any I/P = 0

OR Gate: O/P = 1 || Any I/P = 1

NAND Gate: O/P = 1 || Any one I/P = 0

NOR Gate: O/P = 0 || Any one I/P = 1

Ex-OR: O/P = 1 || every i/p are diff

Ex-NOR: O/P = 1 || every i/p are Same

[I am NOT going class AND lab OR library bcz Ex-OR called me today I have to say Ex-NOR, NAND, NOR]

	NAND	NOR
NOT	1	1
AND	2	3
OR	3	2
X-OR	4	5
X-NOR	5	4
NAND	1	4
NOR	4	1

* $A\bar{B}+A\bar{C}+B = A+B$

Q. Find no of NAND & NOR Gate?

$\gg f = A\cdot\bar{B}\cdot C\cdot\bar{D}\cdot E$

NAND: $(2n-2)+K$

NOR: $(3n-3)+K$

n = no of variable

$\gg f = \bar{A}+B+C$

NAND: $(3n-3)+K$

NOR: $(2n-2)+K$

K = no of Complement

* Jab V SOP mai rahega, use AND-OR-Inverter.

↳ No of NAND Gt = No of AND-OR Gt

Eg: $f = AB+CD$ = 3 NAND

* For POS: No of AND-OR = No of NOR Gt.

→ $f(A,B,C,D) = (A+B)(C+D)$ = 3 NOR

Q. $f(A,B,C) = A\bar{B}+CD$

NAND: Normal SOP + K = 3+1 = 4

Fuctionally Complete, Universal gate. Can be formed.

(i) NAND (ii) AND & OR & NOT (iii) OR & NOR (iv) NOR (v) MUX (vi) Decoder + OR logic //

Property of Ex-OR & Ex-NOR.

$A\oplus 0 = A$ $A\oplus A = 0$ $A\oplus 1 = \bar{A}$

$\gg A\oplus\bar{A}B = A\oplus B$

$\gg AB\oplus BC = B(A\oplus C)$

* $A\oplus A\oplus A\cdots n$ → A, when n is odd; 0, when n is even

* $A\odot 0 = \bar{A}$ * $A\odot 1 = A$ * $A\odot A = 1$ * $A\odot\bar{A} = 0$

* $\overline{A\odot B} = A\oplus B$

* $A\odot B = \overline{A\oplus B} = \bar{A}\oplus B = A\oplus\bar{B} = \bar{A}\odot\bar{B}$

⇒ For even no of Ex-NOR $A\odot B = \overline{A\oplus B}$

⇒ For odd no of Ex-NOR $(A\odot B)\odot C = A\oplus B\oplus C$

$A\odot B\odot C\cdots n$ → 1, n is even; $\bar{A}$, n is odd

k-Map

Fun: SOP: Minterm (Σm) ⇒ $AB+A\bar{B}$

POS: Maxterm (Πm) ⇒ $(\bar{A}+B)\cdot(A+\bar{B})$

Imp theorem

(i) $(A+B)(A+C) = A+BC$

(ii) $f(A,B,C) = AB+\bar{A}C+BC$

$= AB+\bar{A}C$

(2 baar Sabb present rahega aur ek par Complement present hoga)

(iii) $(A+B)(\bar{A}+C) = \bar{A}B+AC$

↳ Same var pr Comp rahega.

(iv) $\overline{ABC} = \bar{A}+\bar{B}+\bar{C}$

$\overline{A+B+C} = \bar{A}\bar{B}\bar{C}$

$A+0 = A$	$A\times 1 = A$
$1+A = 1$	$A\times 0 = 0$
$A+A = A$	$A\times A = A$
$A+\bar{A} = 1$	$A\times\bar{A} = 0$

For n-variable's

- no of diff exp = 2^{2^n} (2^n for min term, 2^n for max term)
- Gray code is called as
  - i) Unity hamming code
  - ii) Cyclic code
  - iii) Reflecting code

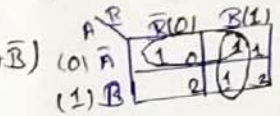
& Ko-Map is based on k-map.

Binary – B_3 B_2 B_1 B_0

Gray – G_3 G_2 G_1 G_0

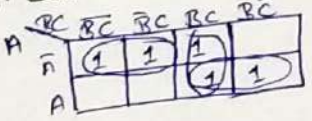
* SOP is used when we want to use more AND Gt and POS is used when we want more OR Gt.

Q. $f(A,B) = \bar{A}\bar{B}+\bar{A}B+AB = \Sigma m(0,1,3)$



$= \bar{A}+B$

Q. $\Sigma m = 0,1,3,6,7$



$\Rightarrow \bar{A}\bar{B}+\bar{A}C+AB$ Ans

* Sum of all minterm = 1
* Sum of all maxterm = 0
* Implicant: Group of 1's
* prime imp: largest possible Group 1's
* Ess prime imp: At least 1 Single which can't Combine in any other.

Q. $f = \Sigma m(0,1,2,4,6,7,9,12,13,14,15)$

AB \ CD	C̄D̄	C̄D	CD	CD̄
ĀB̄	0	1	3	2
ĀB	4	5	7	6
AB	12	13	15	14
AB̄	8	9	11	10

Number System

* to Convert decimal to any other
'n' divide integer part by 'n' &
& multiple fractional part by 'n'.

$(25.625)_{10} = (11001.101)_2$

1) Binary to decimal

$(1101)_2 = (13)_{10}$

$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 13$

2) Octal to decimal

$(57.4)_8 = 47.5$

$5\times8^1 + 7\times8^0 + 4\times8^{-1} = 47.5$

3) Hexa decimal to decimal

$(BAD)_{16} = 2989$

$11\times16^2 + 10\times16^1 + 13\times16^0 = 2989$

4) Binary to octal

$(010\ 110.110) = (26.6)_8$

5) Hexa to binary

$(259A)_{16} = 0010\ 0101\ 1001\ 1010$

* Hexa to octal not possible so first H→B then B→C can do.

Signed Magnitude

→ valid for both +ve & -ve
→ Sign bit Concept is used.
0 = +ve no
1 = -ve no.

Range: $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

Sign	Magnitude

$-(2^{n-1})$ to $(2^{n-1}-1)$ 2's Complement

unsigned Magnitude

→ only for +ve no.
→ +5 = 101, −5 = not allowed
→ Range 0 to $2^n - 1$

	unsigned	Signed	1's	2's
+5	101	0101	0101	0101
−5	x	1101	1010	1011

* +ve no always store in 1's formet/Nor.

Floating point Representation

S	M	E

Sign bit always 1 bit = 0/1

E = Exponent in biased form

M = Mantissa is normalized [E|M]

$E = e + bias$

Explict Nor: $0.1 = (-1)^S \times 0.M \times 2^{E-bias}$

Implict Nor: $1.- = (-1)^S \times .M \times 2^{E-bias}$

↳ Conventional floating point can't Rep. 0

IEEE 754 Floating point Standard

- Single precision: Bias = 127

S	E	M
1	8	23

- Double precision: Bias = 1023

S	E	M
1	11	52

$E = \begin{cases} 00\,00 \text{ —— } 00 \text{ used for special} \\ 1111 \text{ —— } 11 \text{ no Representation.} \end{cases}$

Sign	Exponent	Mantissa	Result
0/1	0000 —— 00	0000 —— 00	+0/−0
0/1	1111 —— 11	1111 —— 11	+∞/−∞
0/1	1111 —— 11	M ≠ 0	Not A Number
0/1	0000 —— 00	M ≠ 0	Fraction
0/1	E ≠ 00 —— 00 & E ≠ 111 —— 11	M = x—x	Implicit Normalized number

BCD (Binary Coded decimal)

» Represent in 4 bit (each digit)

Excess-3, normal +3

1 → 0000 → 0011
2 → 0001 → 0100 etc.

Combinational Circuit

* Mux is also called universal
as AND-OR logic Gate.
* Demux : AND logic

① Design Higher Mux by using lower Mux.

2×1 Mux $\xrightarrow{(2^n - 1)}$ $2^n \times 1$ Mux

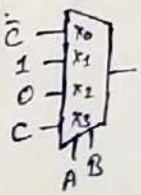
↳ 4×1 M : 8×1 ⇒ $\frac{8}{2} + \frac{4}{2} + \frac{2}{1} = 7$ Mux Req.

↳ 4×1 M : 64×1 ⇒ $\frac{64}{4} + \frac{16}{4} + \frac{4}{4} = 16 + 4 + 1$ Req.

» 3, 2:1 Mux are required.

② Mux as a universal gate.

③ Minimization



$x_0 = \bar{A}\bar{B}$ ⇒ $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}1 + \bar{A}\bar{B}0 + \bar{A}B$

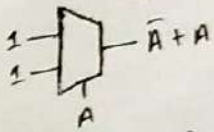
$x_1 = \bar{A}B$ = $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}(\bar{C}+C) + \bar{A}BC$

$x_3 = A\bar{B}$ = $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}$

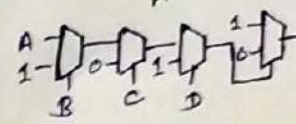
$x_4 = AB$ = $\Sigma m(0,2,3,7)$

= $\bar{A}\bar{C} + BC$ (K-Map)

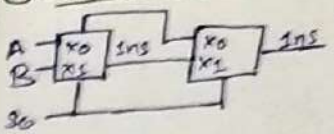
④ Cascading of mux



$\bar{A} + A = 1$



⑤ Delay in Mux



N-bit Magnitude Comparator

» total Combination = 2^{2n}
» equal combination = 2^n
» Unequal Combination = $2^{2n} - 2^n$
» Greater = Less = $\frac{2^{2n} - 2^n}{2}$

Half Adder

» Sum = $\bar{A}B + A\bar{B} = A \oplus B$
» Carry = AB
» total no of NAND/NOR gate required to implement = 5 (4: sum, 1 carry)

Full Adder

» Sum = $A \oplus B \oplus C$
» Carry = $AB + C(A \oplus B)$
» NAND/NOR : 9 (Fore Ful Sub too)
» 1FA = 2HA + 1OR

Half Subtractor

» Diff = $A \oplus B$, Borrow = $\bar{A}B$
» total no of NAN/NOR = 5

Parallel Adder

n-bit parallel Adder

↳ (n−1) FA + 1HA
↳ (2n−1) HA + (n−1) OR Gate
↳ N. Full Adder Req.

Max delay

$T = (n-1)T_{carry} + Max(T_{sum}, T_{carry})$

$T = (n-1)\{T_{AND} + T_{OR}\} + Max\{T_{sum}, T_{carry}\}$

n = no of bit

Semiminimized expression

$X(A>B) = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$

$Y(A<B) = \bar{A}_1B_1 + (A_1 \odot B_1)\bar{A}_0B_0$

$Z(A=B) = (A_1 \odot B_1)(A_0 \odot B_0)$

Sequential Circuit

* Adv of Sequential ckt over Comb national ckt, it include memory to store past data.

Characteristic equation of FF

» S : $Q_{n+1} = S + \bar{R}Q_n$

» JK : $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

» D : $Q_{n+1} = D\bar{Q}_n + DQ_n = D$

» T : $Q_{n+1} = \bar{T}Q_n + T\bar{Q}_n = T \oplus Q_n$

SR FF

S	R	Q_{n+1}	$\bar{Q}_{n+1}$	
0	0	Q_n	$\bar{Q}_n$	→ Hold
0	1	0	1	→ Reset
1	0	1	0	→ Set
1	1	x	x	→ Invalid

JK FF

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\bar{Q}_n$

D FF

D	Q_{n+1}
0	0
1	1

T FF

T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

Synchronous Counter

» All FF are connected with same clock.
» Fast
» All type of Counter are possible
» Eg: Ring, Johnson etc.

Asynchronous Counter

» only one FF having external clock & the o/p of that FF. Counter
» Slow » Generally up-down possible
» Max no of State = 2^n (n = no of FF)

fin → [MOD M] → fout = fin/M

fin → [MOD M] → [MOD N] → fout = fin/(M×N)

FYA Series

* Syn Counter are faster than Asyn
* Adv of Syn Sequential CKT over Asyn is ease of avoiding problem due to hazards.
* Min no of D-FF need to design a mod 258 Counter is

$2^n \geq K \Rightarrow 2^n \geq 258, n \geq 9$

* Ring Counter is Stepping Switch Counter
* Johnson Counter is Switch tail ring counter in which a circular Shift register with the complemented o/p of the last FF connected to the i/p of the first FF

0000 → 1000 → 1100 → 1110 → 1111 → 0111 → 0011 → 0001 → 0000

* fun diff b/w SR FF & JK FF is that JK accept both input 1.
* Min no of FF = no of bit.

0-1-0-2-0-3 = 3 unique (3FF)

* Modulus-12 Ring Counter : need 12 FF
* Synchronous Counter:
  i) Ring C : N no of FF = N no of State
  ii) Johson : N no of FF = 2N no of State

Combinational CKT

* Min no of mux needed for implement of any function of n variable

n-variable → $2^n : 1$ MUX, $2^{n-1} : 1$ MUX + NOT

* Consider n-bit ripple Carry adder

total delay = delay sum + (n-1) delay carry

* For m input line max 2^m o/p line get.

n → [ ] → $2^n = m$.

» total line = m + n

Number System

* No represent in 2's Complement
  i) +ve as they are
  ii) -ve are in 2's Complement

So overflow Condition are---

i) two positive no are added & get sign bit 1.
ii) 2, -ve no added & get sign bit 0.

[x + y = 1 ABC.]
[(-x) + (-y) = 0 ABC]

» NO overflow, if two opposite sign are added, in case of overflow n-bit, n+1 bit are added.

* In normalized form,
  L Exponent can't be all 0 or 1
  L Mantisa Can be 0 or 1.
* Excess-3 Code is called Self Complementing code.
* ASCI : used 7 bit to Represent
  BCD : 4 bit
  EBCDIC : 8 bit
  Unray code : 3 bit
* Gray code is termed as min error code
* XOR Gate used to detect overflow in 2's Complement.
* for +ve no, booth algo, to find addition/subtraction req = no of group 1's

Eg: multiplier = 0111101111001111001

= 4+4 = Add + Sub = 8

* for Signed & Negative number:
  L no of addition = no of group 1's - 1
  L no of subtrac. = no of group 1's
  L in case of total no of arithmatic Operation = 2n - 1

Misc.

* 1 mux and 1 inverter are allowed to implement any boolean function of n variable min size of mux needed : $2^{n-1} : 1$
* -57 in booth's: 0-100100-1
* Sign extension is a step in converting a signed integer from one size to another.
* 0 has 2 Representation in
  i) Sign magnitude & 1's Compliment
  * for n-bit data Select input: $2^n : 1$
  * 0x00000000 hexadecimal
    L the special value +0
    8x00000000 hexadecimal
    L the Special value -0
  * the worst case of an implementation using booths algorithms is when pairs of 01's or 10's occur very frequently in multiplier.
  * Static '1' Hazard : SOP : AND-OR logic
  * Static 0 Hazard POS : AND logic
  * look ahead Carry adder.

Carry block., no of AND = n(n+1)/2
no of OR = n

Theory of Computation

	English	Binary	C lang.
Symbol	a,b,.. z	0, 1	characters
Alphabet	{a,b,...z}	{0,1}	character set
String	words	numbers	token
language	Sentence	binary lang	programs

* For $\Sigma = \{0,1\}$, How many K-length string possible $= 2^K = |\Sigma|^K$
* No of prefix, suffix for n-length string is $n+1$
* no of substring for n-length string $\le \frac{n(n+1)}{2}+1$
  $n+1 \le$ No of substring $\le \frac{n(n+1)}{2}+1$
  if all symbol is same — if all symbol in string is distinct
* To Represent one REG language there are infinite REG expression.
  ↳ $L = \{w \mid w \in \{a,b\}^*$, w end with a$\}$
    ↳ $R = (a+b)^*a$
  ↳ $L = \{w \mid w \in \{a,b\}^*, |w|$ is odd$\}$
    ↳ $R = [(a+b)^2]^*(a+b)$
  ↳ $L = \{w \mid w \in \{a,b\}, \#_a w = \text{odd}\}$
    ↳ $b^*(b^*ab^*ab^*)^*b^*ab^*$
* if L has min DFA with n state then $\bar{L}$ has same no of state.
* Min NFA → Min DFA
  $n \le$ no of state in DFA $\le 2^n$

$(a+b)^*$ equivalence to
i) $(a+b+\varepsilon)^+$ iii) $(a^*b^*)^*$
ii) $\varepsilon+(a+b)^+$ iv) $(b^*a^*)^+$

$w \in \{a,b\}^*$

$w \in \{a,b\}^*$	Min DFA	Min NFA
$\|w\| = k$	$k+2$	$k+1$
$\|w\| \le k$	$k+2$	$k+1$
$\|w\| \ge k$	$k+1$	$k+1$
$ab\Sigma^*$	$\|min\|+2$	$\|min\|+1$
Σ^*ab	$\|min\|+1$	$\|min\|+1$
$\Sigma^*ab\Sigma^*$	$\|min\|+1$	$\|min\|+1$
$\#_a(w) = k$	$k+2$	$k+1$
$\#_a(w) \le k$	$k+2$	$k+1$
$\#_a(w) \ge k$	$k+1$	$k+1$
$\|w\| \% k = 0$	k	k
$\#_a\|w\| \% k = 0$	k	k
k^{th} Symbol is 'a'	$k+1$	$k+1$
k^{th} Symbol from end a	2^k	$k+1$

Find Regular lang from REG Grammer
>> $S \to Sa \mid b$, $L = ba^*$
>> $S \to Sa$, $L = \phi$
>> $S \to Aa$, $A \to Aa \mid Ab \mid \varepsilon$, $L = (a+b)^*a$
>> $S \to aA \mid bA$ $S \to (a+b)A$
   $A \to aB \mid bB$ $\to (a+b)^2(a+b)^*$
   $B \to aB \mid bB \mid \varepsilon$
>> $S \to Sa \mid Ab \mid a$ $L = S = a^+$
   $A \to Ab$
   $B \to Ba \mid Bb \mid \varepsilon$

Pumping Lemma
* PL satisfies REG language
* PL prove Non-REG lang using contradiction.
* PL uses Pigeon hole principle.
* $L = (a+b)^*aaa(a+b)^*$.
  Min length = 3, # State = 4. So PL ≥ 4
* Every Finite lang is REG.
* Every infinite lang is May/May not REG.

identify REG and Non-REG.
> $\{a^m b^n \mid m < n < 100\}$ = FL = REG
> $\{a^n b^{2n}\}$ = non-Regular.
> $\{a^{Prime}\}$ = Non-REG
> $\{a^{Prime}\}^*$ = REG
> $\{a^{n!} b^{Prime}\}$ = Non-REG
> $\{ww \mid w \in \{a,b\}^+\}$: Non REG
> $\{w_1w_2 \mid w_1 w_2 \in a^*\} = a^*$: REG

* Jaha pe dependency milega wo REG nhi hoga.

* Equivalence classes
  ↳ no of state in min DFA for REG = no of equivalence class in REG.

* RG
  - LLG : $V \to VT^*$ or T^*
  - RLG : $V \to T^*V$ or T^*
* CFG : $V \to (V \cup T)^*$ any sequence
  Eg: $S \to SaAbS \mid a$, $A \to SAb \mid \varepsilon$
  ↳ FA
  - DFA : $Q \times \Sigma \to Q$
  - NFA : $Q \times \Sigma \to 2^Q$
  ↳ PDA : DPDA < NPDA
* Every DCFL is CFL but vv not true.
* Every DPDA is PDA but vv not true
* NPDA
  - PDA with final State
  - PDA with empty Stack
* PDA = $(Q, \Sigma, \delta, q_0, F, Z_0, \Gamma)$

$\delta_{PDA} = Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon^* \to 2^{Q \times \Gamma^*}$

$\delta_{DPDA} = Q \times \Sigma \times \Gamma \to Q \times \Gamma^*$

$\delta(q, a, x) = (q', \varepsilon)$
(present state, current input symbol, top of stack; next state)
→ invalid for DPDA but valid for PDA

* For DPDA non- of state are ε
* $\delta(q,a,b) = (q', ab)$, PDA, DPDA.
* Every FA is countable to PDA & DPDA.

Find out REG | DCFL | CFL
> $\{a^m b^n\} = a^*b^*$ = REG = DCFL = CFL
> $\{a^n b^n a^n\}$ = Not CFL
> $\{a^n b^m c^{n+m}\}$ = DCFL but ! REG
* lang over 1 symbol is FA = PDA
> $\{w\#w \mid w \in \{a,b\}^*\}$ Nt CFL
> $\{w\#w^R \mid w \in \{a,b\}^*\}$ DCFL but ! REG
> $\{ww \mid w \in a^*\}$ = REG = DCFL = CFL
> $\{wwx \mid w, x \in \{a,b\}^*\}$ – Not CFL
> $\{ww^Rx \mid$ " $\}$ CFL but ! DCFL
> $\{w\#w^R\#x\#x^R \mid$ " $\}$: DCFL
> $\{a^m b^n c^k \mid m \ne n, \text{or } m \le k\}$: CFL

Turing Machine & RE
$\delta(q, a) = (q', a, R)$
(Present state, Read; next state, write, Direction)

* TM represent REL.
* TM [Accept, REL]: logic exist for valid string.
* HTM [Accept RL] : logic exit for both valid & invalid : decidable

	TM	HTM
1. Recursive lang (Decidable)	✓	✓
2. REL (Semi decidable)	✓	M/MN
3. REL but not Recursive	✓	✗
4. Not REL	✗	✗
5. Undecidable lang.	May/May not	✗

⇒ L is Recursive language iff
1. L is decideble language.
2. L has HTM
3. logic exist for valid & invalid Input
4. L & $\bar{L}$ both are REL
5. L has TM & $\bar{L}$ has TM

⇒ L is REL iff
1. L has TM & semi decideable
2. Logic exist for valid input
3. L is enumerable.
4. L has unrestricted grammer

⇒ L is REL but not Recursive iff
1. L has TM but not Recursive
2. L has TM but L has no HTM.
3. L has TM but $\bar{L}$ has no TM.
4. L is Semi-decidable & undecidable

⇒ L is not REL iff
1. L is not semi decidable
2. L has no TM
3. valid string of L has no logic

⇒ L is undecidable iff
1. L is not decidable
2. L is not Recursive & no HTM

** TOC closure properties

Operation	RL	DCFL	CFL	CSL	REC	RE
(1) Union (∪)	-	x	x	-	-	x
(2) Intersection (∩)	-	x	x	-	-	x
(3) Complement (c)	-	-	x	-	-	x
(4) Concat (.)	-	x	x	-	-	x
(5) Set diff (-)	-	x	-	-	-	-
(6) Kleene (*)	-	x	x	-	-	x
(7) +ve clo (+)	-	x	-	-	-	-
(8) Reversal (L^R)	-	x	x	-	-	x
(9) $L \cap RL$	-	-	-	-	-	-
(10) $L - RL$	-	-	x	-	-	x
(11) $RL - L$	-	-	-	-	-	-
(12) $L \cup RL$	-	-	-	-	-	-
(13) Prefix / init	x	x	x	x	x	x
(14) Subset	x	x	x	x	x	x
(15) infinite operation	x	x	x	x	x	x

(1) FA | FA accept ε
(2) ⟨FA, ω⟩ | FA accept ω] D
(3) FA# ω | FA accept ω
(4) TM | TM accept ε/ω
(5) TM accept some/only string] SDUD
(6) TM reached q within 5 step UD
(7) $L = a^*b^*$ / FA / FL / RL / DCFL / CFL / CSL] D
(8) Recursive lang : D
(9) REL / REL but not Recursive : SDUD
(10) $a^n b^n$: D
(11) Set of all Finite Language] SDUD
(12) Set of all Regular language
(13) Set of DCFL / CFL / CSL
(14) Set of not REL] NR
(15) Set of infinite language

	FA/ RegL/ RL	DPDA/ DCFL	PDA/ CFL/ CFL	HTM/ Recursive	TM/ REL/ U
Halting	D	D	D	D	UD
Membership	D	D	D	D	UD
Emptiness	D	D	D	UD	UD
Finiteness	D	D	D	UD	UD
totality	D	D	UD	UD	UD
Equivalence	D	D	UD	UD	UD
Disjoint	D	UD	UD	UD	UD
Set Containment	D	UD	UD	UD	UD