# Compiler Design

## chapters:

(1) Overview of the compiler & its structure
(2) Lexical Analysis
(3) Syntactic Syntax Analysis
(4) Error Recovery
(5) Intermediate-Code Generation
(6) Run-Time Environments
(7) Code Generation and Optimization
(8) Instruction-Level parallelism.

## Regular Expressions

(1) Construct R.E for the language which consists of exactly two b's, $\Sigma = \{a, b\}$

$$R.E = a^* b a^* b a^*$$

(2) W.a.R.E for the language of all strings of 0's and 1's with an even number of 0's and odd number of 1's.

$$R.E = 1(11)^* + (00)^*$$

R.E for (1) all strings containing 0's at least one 0 & at least one 1.

$\longrightarrow (0+1)^+$

(2) containing atmost one consecutive pair of 1's.

$$R.E = 0^* 1 1 0^*$$

Page No.
Date

⊗ Construct LMD & RMD for sentence abab over grammar.

$S \to aSbS \mid bSaS \mid \varepsilon$

| L.M.D | | R.M.D. | |
|---|---|---|---|
| $S \to aSbS$ | $\because S \to aSbS$ | $S \to aSbS$ | $\because S \to aSbS$ |
| $\to abSaSbS$ | $\because S \to bSaS$ | $\to aSb$ | $\because S \to \varepsilon$ |
| $\to abaSbS$ | $\because S \to \varepsilon$ | $\to abSaSb$ | $\because S \to bSaS$ |
| $\to ababS$ | $\because S \to \varepsilon$ | $\to abSab$ | $\because S \to \varepsilon$ |
| $\to abab$ | $\because S \to \varepsilon$ | $\to abab$ | $\because S \to \varepsilon$ |

⊗ **Left Recursion :-**

$A \to A\alpha \mid \beta$

$A \to A\alpha_1 \mid A\alpha_2 \mid \dots \beta_1 \mid \beta_2$

⊗ **Remove Left Recursion :-**

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

Ex. ① $A \to A\underbrace{+B}_{\alpha} \mid \underbrace{B}_{\beta}$

$A \to BA'$

$A' \to +BA' \mid \varepsilon$

## First & Follow table :-

(1) $S \to aBDh$
$B \to cC$
$C \to bC | \varepsilon$
$D \to EF$
$E \to g | \varepsilon$
$F \to f | \varepsilon$

Follow
Top to down
& No $\varepsilon$

if $\varepsilon \to$ add to the set
Bottom to up
(First)

First

| Symbols | First | Follow |
|---|---|---|
| S | {a} | {\$} |
| B | {c} | {g, f, h} |
| C | {b, ε} | {g, f, h} |
| D | {g, f, ε} | {h} |
| E | {g, ε} | {f, h} |
| F | {f, ε} | {h} |

(2) $S \to ACB | cbB | Ba$
$A \to da | BC$
$B \to g | \varepsilon$
$C \to h | \varepsilon$

| Symbols | First | Follow |
|---|---|---|
| S | {d, g, h, ε, b, a} | {\$} |
| A | {d, g, h, ε} | {g, h, \$} |
| B | {g, ε} | {h, \$, a, g} |
| C | {h, ε} | {\$, b, g, h} |

(11) $S \to 1AB \mid \varepsilon$

$A \to 1AC \mid 0C$

$B \to 0S$

$C \to 1$

| Symbols | First | Follow |
| --- | --- | --- |
| S | {1, ε} | {$} |
| A | {1, 0} | {0, 1} |
| B | {0} | {$} |
| C | {1} | {0, 1} |

⊗ predictive Parsing table :-

| | 0 | 1 | $ |
| --- | --- | --- | --- |
| S | | S→1AB | S→ε |
| A | A→0C | A→1AC | |
| B | B→0S | | |
| C | | C→1 | |

- This grammar is LL(1) Because every State in LL(1) Table has one entry.

(12) $S' \to S$

$S \to aA \mid b \mid cB \mid d$

$A \to aA \mid b$

$B \to cB \mid d$

| Symbol | First | Follow |
| --- | --- | --- |
| S' | {a, b, c, d} | {$} |
| S | {a, b, c, d} | {$} |
| A | {a, b} | { } |
| B | {c, d} | { } |

For A,

I/p a = Follow(1) ∪ Follow(3) = {1, 2, 3, 4} — B
I/p b = For ∅
I/p ε = Follow(2) = {1, 2, 3} — A

For B,

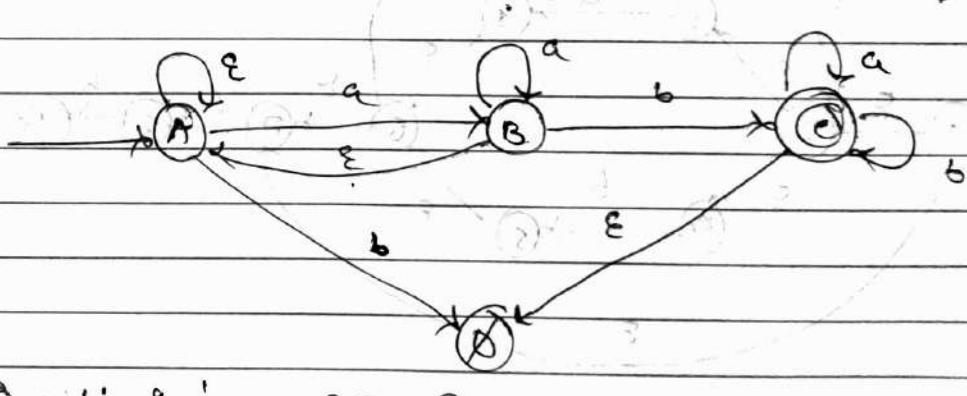
I/p a = Follow(1) ∪ Follow(3) = {1, 2, 3, 4} — B
I/p b = Follow(4) = {5, 6} — C
I/p ε = Follow(2) = {1, 2, 3} — A

For C,

I/p a = Follow(5) = {5, 6} — C
I/p b = Follow(6) = {5, 6} — C
I/p ε = ∅.

TT.

| | a | b | ε |
|---|---|---|---|
| A | B | – | A |
| B | B | c | A |
| * C | c | c | – |




Optimizing DFA

→ {A, B, ∅} = Non accepting state
{C} = Accepting state.

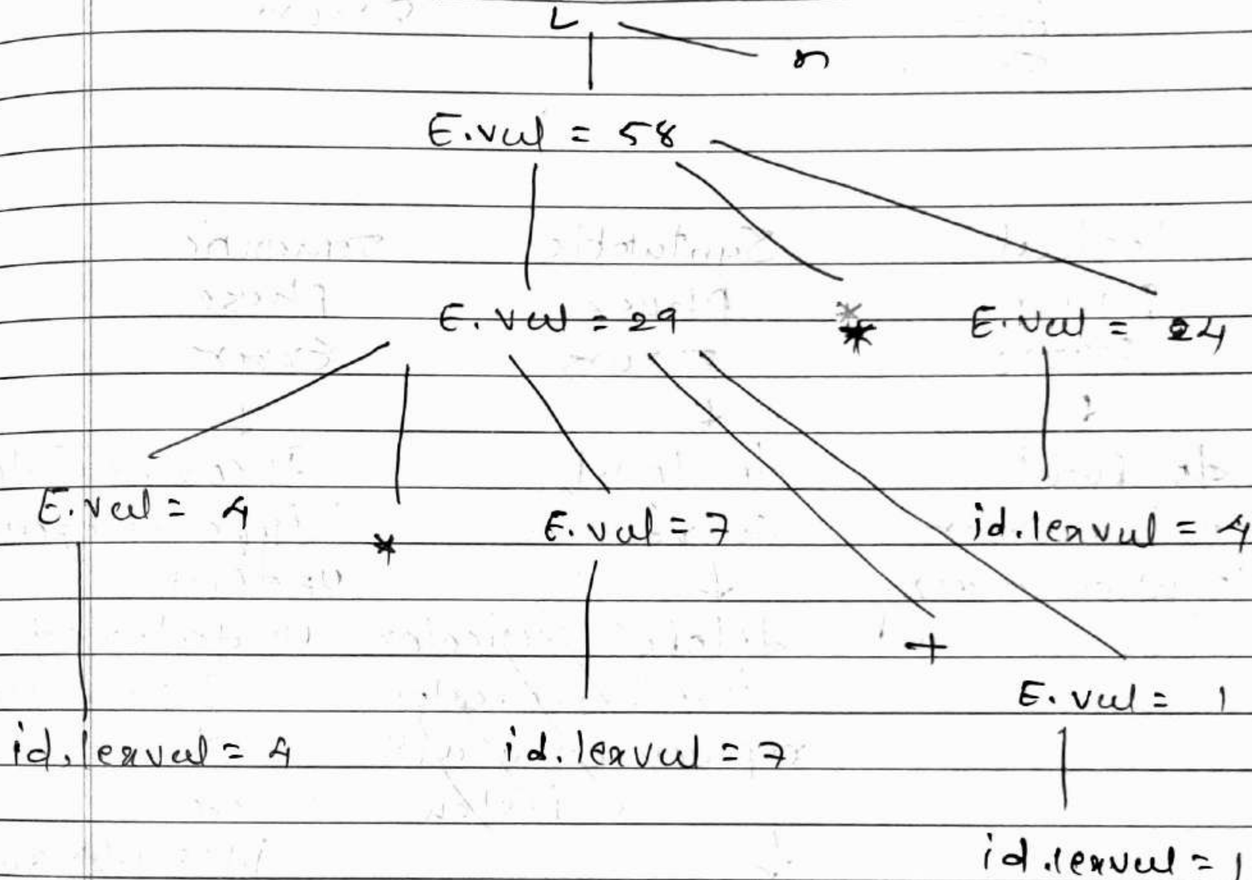
Page No.
Date

(1) $(0+10)^*$

(2) $((0+1)00)^*$.

→ * —— * ←—— * ——

| production | Semantic Rules |
| --- | --- |
| L → En | Print (E.val) |
| ~~E → T~~ E → E + E | E.val = E.val + E.val |
| E → E * E | E.val = E.val * E.val |
| E → id | E.val = id.lexval |

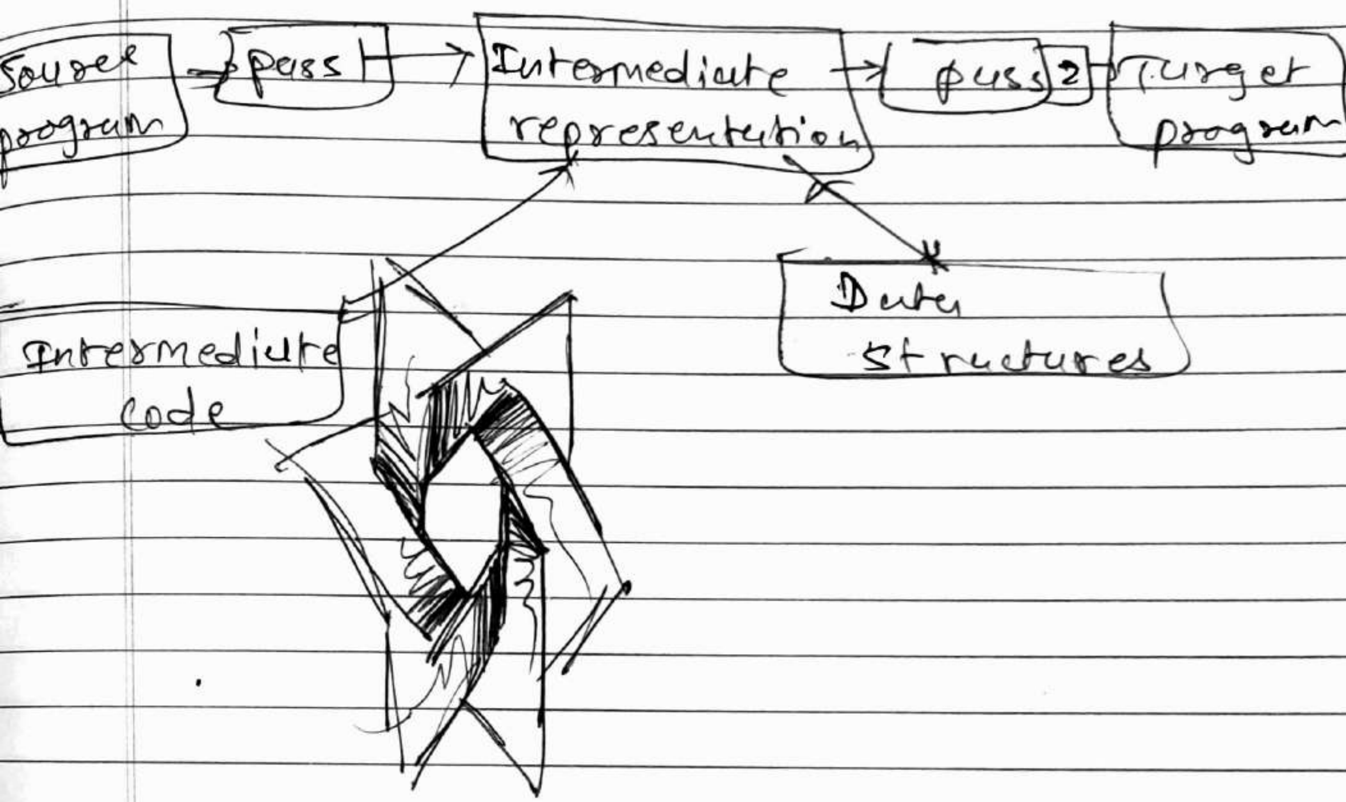



② Annoted parse tree for (4 * 7 + 1) * 2 n ②

## ③ Error Recovery Strategies

(1) Panic mode
(2) Phrase Level Recovery
(3) Error Production
(4) Global Correction.

(2) List Scheduling of Basic Blocks:
(3) Prioritized Topological Orders.




operations in the machine instructions.
data-dependence constrains among the operations.

value is simply resource-reservations.

no earlier source node.

(4) code reuse
(5) Easier debugging.

---

Q.(3) Translate the expression:

$-(a+b) * (c+d) + (a+b+c)$

$\rightarrow$
$t_1 = a+b$
$t_2 = -t_1$
$t_3 = c+d$
$t_4 = t_2 * t_3$
$t_5 = a+b$
$t_6 = t_5 + c$
$t_7 = t_4 + t_6$

(1) Quadruples:

| Op | opr 1 | opr 2 | Result |
|---|---|---|---|
| + | a | b | $t_1$ |
| − | $t_1$ | | $t_2$ |
| + | c | d | $t_3$ |
| * | $t_2$ | $t_3$ | $t_4$ |
| + | a | b | $t_5$ |
| + | $t_5$ | c | $t_6$ |
| + | $t_4$ | $t_6$ | $t_7$ |

(2) Triple :-

Page No.
Date

(3) Control link :-

- points to activation record of caller.

(4) Access link :-

- refers to non-local data held in other activation records.

(5) Machine Status :-

- holds the information about status of machine just before the func^n call.

(6) Local Variables :-

- hold the data that is local to the execution of the procedure.

(7) Temporary Values :-

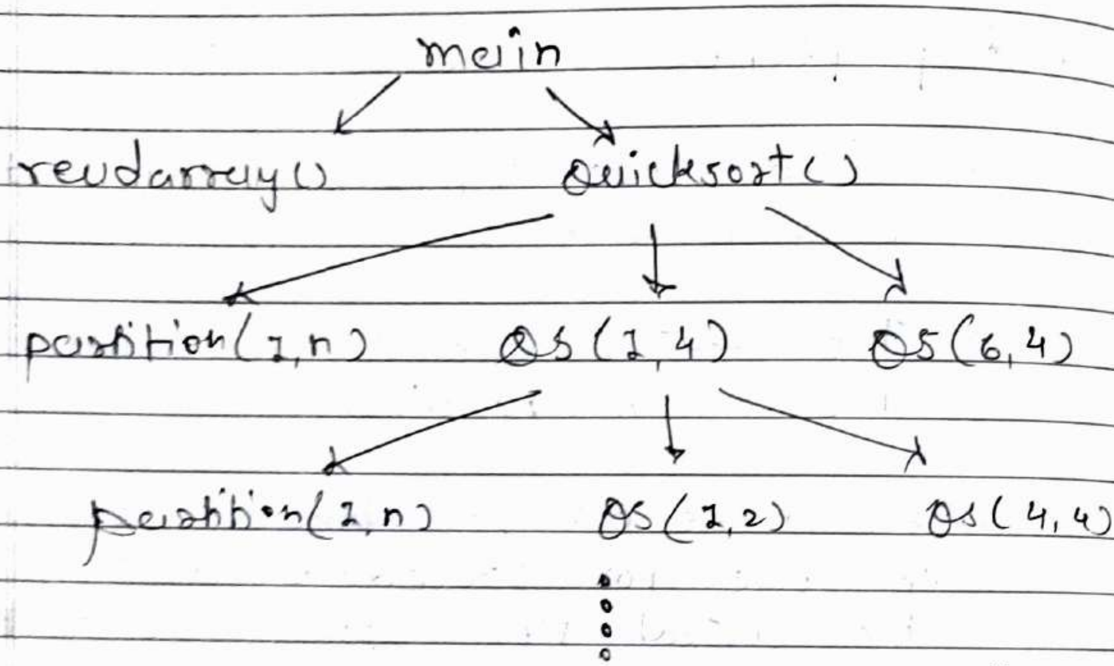
- stores the values that arise in the evaluation of an expression.

---

Q.② Differentiate static Vs Dynamic M. Allocation.

| No. | Static M.A | Dynamic M.A |
|---|---|---|
| 1. | - In SMA, Variables get allocated permanently till the program executes. | In DMA, Variables get allocated only if your program unit gets active. |

∴ The activation tree for this program will be :




- First main function as the root then main calls readarray and Quicksort.

- Quicksort in turn calls partition & Quicksort again.

- The flow of control in a program corresponds to a pre-order depth-first traversal of the activation tree which starts at the root.

---

Q. (3) a write diff between stack & Heap memory allocation.

| No. | Stack | Heap |
|---|---|---|
| 1. | memory is allocated in a continuous block. | memory is allocated in any random order |

Ex. Initial code: Optimized code:

$x = 2 * 3;$ $x = 6;$

## [3] Strength Reduction :-

- The operators that consumes higher execution time are replaced by the operators consuming less execution time.

Ex. $y = x * 2;$ → $y = x + x$ or $y = x << 1;$

$y = x / 2;$ → $y = x >> 1;$

## [4] Combine operations :-

- Several operations are replaced by a single equivalent operation.

Ex. $r2 := r1 \times 2;$ → $r3 := r1 + r1;$

$r3 := r2 \times 1;$

## [5] Null sequences / simplify Algebric Expressions:

- useless operations are deleted.

Ex. $a := a + 0;$

$a := a * 1;$

$a := a / 1;$

$a := a - 0;$