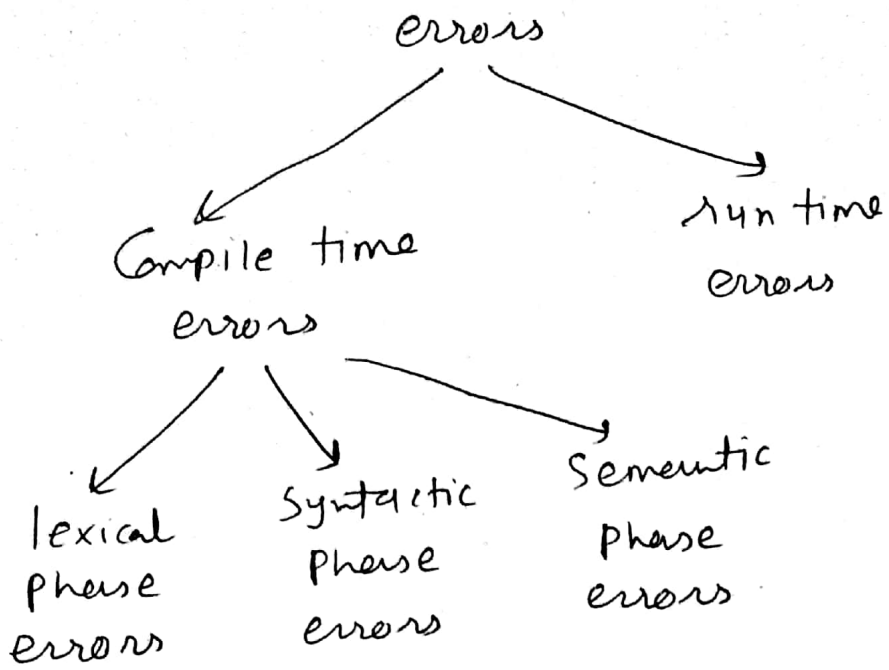
# Ch-4 Error recovery :

## Types of errors :




1) lexical phase errors :
   1) Spelling errors : get incorrect tokens.
   2) Exceeding length of identifier or numeric Constant
   3) Appearance of illegal characters.
      → ex : printf(" ");$

ex : swtich ($choice)
{
---
---
}

2) Syntactic phase errors :
   1) Errors in structure
   2) Missing operators
   3) Unbalanced parenthesis.

3) Semantic Phase errors:

1) Incompatible types of operands
2) Undeclared variables
3) Not matching of actual arguments with formal arguments.

ex: int a[10], b;

....

....

a = b;

int a;

a = b;

# Errors recovery strategies in Compilers:

## 1) panic mode:

→ This strategy is used by most parsing methods.

→ Simple to implement

→ on discovering errors, the parser discards input symbol one at a time. This process is continued until one of designated set of synchronizing tokens is found. Synchronizing tokens are delimeters such as semicolon or end. it indicates end of input statement

→ guarntees not to go in infinite loop.

→ If There are less number of errors in the same statement then this strategy is best choice

## 2) Phrase level recovery:

→ on discovering error parser perform local correction on remaining input.

→ Replace a prefix of remaining input by some strings. This helps the parser to continue its job.

→ The local correction: Replacing comma by semicolon, deletion of extra semicolons or inserting missing semicolon. The type of local correction is decided by compiler designer.

→ drawback: it finds difficult to handle the situations where the actual error has occured before the point of detection.

→ It should taken the care for not going in infinite loop.

3) Error production:

If we have a knowledge of common errors that can be encountered then we can incorporate these errors by augmenting the grammar of the corresponding language with error production that generate the erroneous constructs.

→ If error production is used, we can generate appropriate error message, and parsing can be continued.

→ extremely difficult to maintain.

Date: / /
Page No.

4) Global Production :

→ We often want such a compiler that makes very few changes in processing an incorrect input string.

→ we expect less number of insertions, deletions and changes of tokens to recover from erroneous input.

→ increase time and space requirements at parsing time.

→ Theoretical Concept.

How can panic mode and phrase level recovery be implemented in LR parsers.

$E \to E+E \;(1) \mid E*E \;(2) \mid (E) \;(3) \mid id \;(4)$. prepare SLR parsing table with errors detection and recovery routines.

Augmented grammar:

$E' \to E$

$E \to E+E$

$E \to E*E$

$E \to (E)$

$E \to id$

$E : \{+, *, ), id\}$

$I_0$: $E' \to .E$
$E \to .E+E$
$E \to .E*E$
$E \to .(E)$
$E \to .id$

$I_1$: goto $(I_0, E)$
$E' \to E. //$
$E \to E.+E$
$E \to E.*E$

$I_2$: goto $(I_0, ()$
$E \to (.E)$
$E \to .E+E$

$I_3$: goto $(I_0, id)$
$E \to id. //$

$I_4$: goto $(I_4, +)$
$E \to E+.E$
$E \to .E+E$
$E \to .E*E$
$E \to .(E)$
$E \to .id$

$I_5$: goto $(I_4, *)$
$E \to E*.E$
$E \to .E+E$
$E \to .E*E$

$I_6$ : goto ($I_2$, E)

E → (E.)

E → E. + E

E → E. * E

$I_4$ : goto ($I_2$, ()

E → (. E)

E → . E + E

E → . E * E

E → . (E)

E → . id

$I_5$ : goto ($I_2$, id)

E → id. //

$I_7$ : goto ($I_4$, E)

E → E + E. //

E → E. + E

E → E. * E

($I_2$) : goto ($I_4$, ()

($I_3$) : goto ($I_4$, id)

$I_8$ : goto ($I_5$, E)

E → E * E. //

E → E. + E

E → E. * E

($I_2$) : goto ($I_5$, ()

($I_3$) : goto ($I_5$, id)

$I_9$ : goto ($I_6$, ))

E → (E). //

($I_4$) : goto ($I_8$, +)

($I_5$) : goto ($I_6$, *)

($I_4$) : goto ($I_7$, +) ($I_4$) : ($I_8$, +)

($I_5$) : goto ($I_7$, *) ($I_5$) : ($I_8$, *)

| | + | * | ( | ) | id | $ | E |
|---|---|---|---|---|---|---|---|
| 0 | $e_1$ | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 1 |
| 1 | $s_4$ | $s_5$ | $e_3$ | $e_2$ | $e_3$ | acc. | |
| 2 | $e_1$ | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 6 |
| 3 | $r_{b_1}$ | $r_{b_1}$ | $r_4$ | $r_{b_1}$ | $r_{b_2}$ | $r_4$ | |
| 4 | $e_1$ | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 7 |
| 5 | $e_1$ | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 8 |
| 6 | $s_4$ | $s_5$ | $e_3$ | $s_9$ | $e_3$ | $e_4$ | |
| 7 | $s_4/r_1$ | $s_5/r_1$ | $r_1$ | $r_1$ | $r_1$ | $r_1$ | |
| 8 | $s_4/r_2$ | $s_5/r_2$ | $r_2$ | $r_2$ | $r_2$ | $r_2$ | |
| 9 | $r_3$ | $r_3$ | $r_3$ | $r_3$ | $r_3$ | $r_3$ | |

id + *

| Stack | Input string | Action |
|---|---|---|
| $0 | id + * $ | $s_3$ |
| $0 id 3 | + * $ | $r_4$ |
| $0 E 1 | + * $ | $s_4$ |
| $0 E 1 + 4 | * $ | $e_1$ |

id id

| Stack | Input string | Action |
|---|---|---|
| $0 | id id $ | $s_3$ |
| $0 id 3 | id $ | $r_4$ |
| $0 E 1 | id $ | $e_3$ |

id (

| Stack | Input string | Action |
|---|---|---|
| $0 | id ( $ | $s_3$ |
| $0 id 3 | ( $ | $r_4$ |
| $0 E 1 | ( $ | $e_3$ |

for $

2) ( $

| Stack | Input string | Action |
|---|---|---|
| $0 | ( $ | $s_2$ |
| $0 ( 2 | $ | $e_1$ |

6) ( id $

| Stack | Input string | Action |
|---|---|---|
| $0 | ( id $ | $s_2$ |
| $0 ( 2 | id $ | $s_3$ |
| $0 ( 2 id 3 | $ | $r_4$ |

$ 0 ( 2 E 6 $ $e_4$

0) $ → missing operand : $e_1$

4) +$ → missing operand : $e_1$

5) *$ → missing operand : $e_1$

$e_1$ : Missing operand
$e_2$ : unbalanced right parenthesis
$e_3$ : Missing operator
$e_4$ : Missing right parenthesis.

Ex-2

$S \to (L) \mid a$

$L \to L, S \mid S$

Prepare SLR parsing table with error detection and recovery routines.

Augmented grammar:

0. $S' \to S$
1. $S \to (L)$
2. $S \to a$
3. $L \to L, S$
4. $L \to S$

$I_0$: $S' \to .S$
$S \to .(L)$
$S \to .a$

$I_1$: goto($I_0$, S)
$S' \to S.$ //

$I_2$: goto($I_0$, ()
$S \to (.L)$
$L \to .L, S$
$L \to .S$
$S \to .(L)$
$S \to .a$

$I_3$: goto($I_0$, a)
$S \to a.$

$I_4$: goto($I_2$, L)
$S \to (L.)$
$S \to L., S$

$I_5$: goto($I_2$, S)
$L \to S.$ //

(I2): goto($I_2$, ()

(I3): goto($I_2$, a)

$I_6$: goto($I_4$, ))
$S \to (L).$ //

$I_7$: goto($I_4$, ,)
$S \to L, .S$

$I_8$: goto($I_7$, S)
$S \to L, S.$

SLR Parsing table:

| | i | ( | ) | + | * | S | L |
|---|---|---|---|---|---|---|---|
| 0 | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 1 | |
| 1 | $e_1$ | $e_3$ | $e_2$ | $e_3$ | Acc. | | |
| 2 | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 5 | 4 |
| 3 | $r_2$ | $r_2$ | $r_2$ | $r_2$ | $r_2$ | | |
| 4 | $s_7$ | $e_3$ | $s_6$ | $e_3$ | $e_4$ | | |
| 5 | $r_4$ | $r_4$ | $r_4$ | $r_4$ | $r_4$ | | |
| 6 | $r_1$ | $r_1$ | $r_1$ | $r_1$ | $r_1$ | | |
| 7 | $e_1$ | $s_2$ | $e_2$ | $s_3$ | $e_1$ | 8 | |
| 8 | $r_3$ | $r_3$ | $r_3$ | $r_3$ | $r_3$ | | |

for $

0) $ → missing operand → $e_1$
2) ( $ → missing operand → $e_1$
4) ( id $ → missing right parenthesis → $e_4$
7) + $ → missing operand → $e_1$.