9th March 2023.

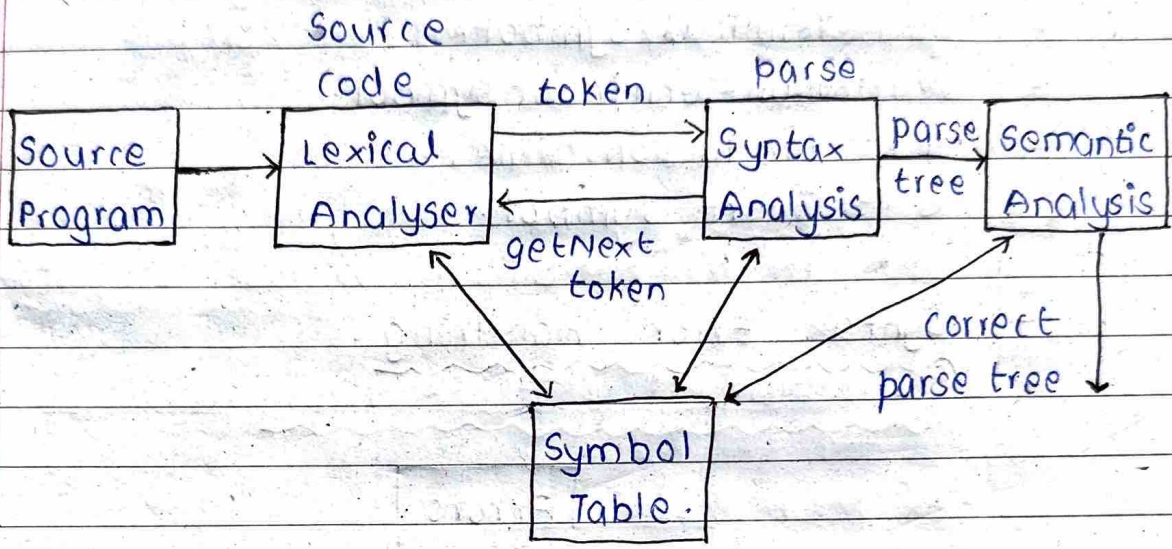
CD Unit-2.

PAGE No.
DATE / /

- Syntax Analysis Definition - Role of Parser
- Lexical versus syntactic analysis.
- Representative Grammars.
- Syntax Error Handling.
- Elimination of Ambiguity, Left Recursion.
- Left Factoring.
- Top Down Parsing
- Recursive Descent Parsing, backtracking.
- Computation of FIRST.
- Computation of FOLLOW.
- Predictive Parsing Table.
- Predictive Parsers LL(1) Grammars.
- Transition diagram for Predictive Parser.
- Error recovery in Predictive Parsing
- Predictive Parsing Algo
- Non Recursive Predictive Parser.

PAGE No.
DATE / /

- Role of Parser / Syntax Analysis.



- Lexical vs Syntactic Analysis.

1] Separating the syntactic structure of a language into lexical and nonlexical parts provide a convenient way of modularizing

2] The lexical rules of a language are simple and we don't need a grammar to describe them.

3] Regular Expressions are easy to understand than grammar.

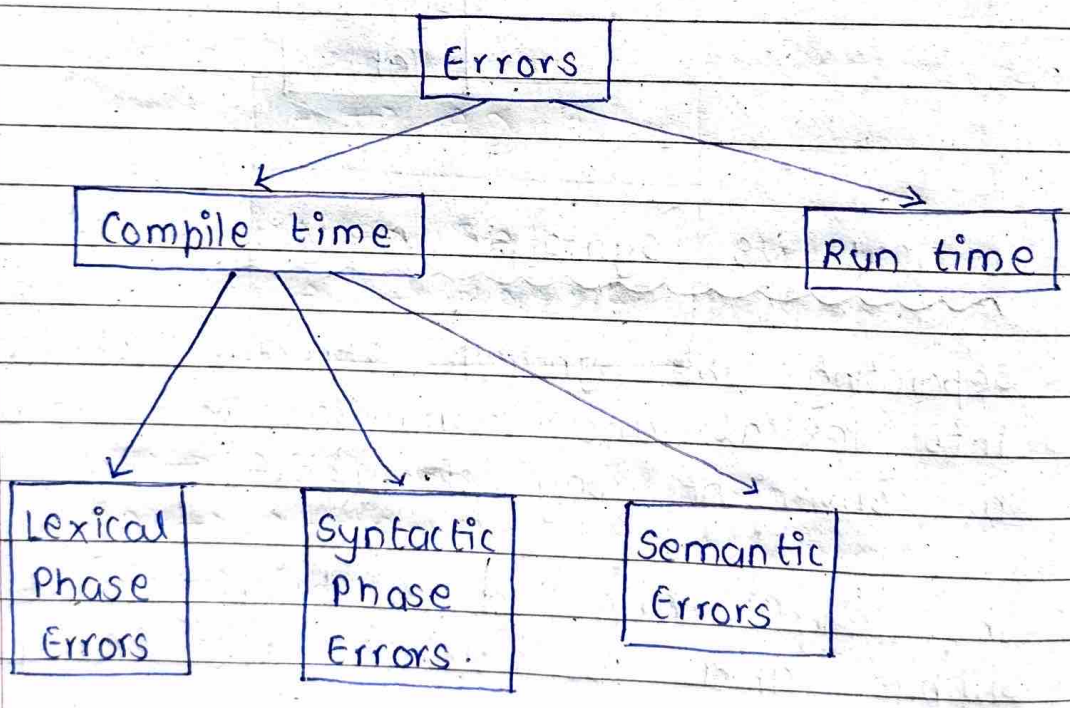
- Grammar.

PAGE No.
DATE / /

$G = (V, T, P, S)$.

G = Grammar.
T = Terminal symbol.
V = Non-Terminal symbol.
P = Production Rules.
S = Start symbol.

- Syntax Error Handling -



1. Lexical Error.

- During lexical phase, lexical error can be detected.
- Lexical error is a sequence of characters that does not match the pattern of any token.

PAGE No.
DATE / /

- Lexical phase error is found during the execution of program.

- Lexical phase error can be :-

- Spelling error
- Exceeding length of identifier.
- Appearance of illegal character.
- Replace a character with an incorrect character.

- Example :-

```
void main()
{
int x = 10, y = 20;
char *a;
a = &x;
 x = 4xab;
}
```

In this code, 4xab is neither a number nor an identifier. So, this code will show the lexical error.

2) Syntax Error.

- Syntax error appears during syntax analysis phase. (2nd phase).
- It is found during execution of program.
- Some syntax errors can be :-
  - Error in structure.
  - Missing operators.

Eg using "=" when "==" is needed.

PAGE No.
DATE / /

3] Semantic Error.

- Semantic error occurs during semantic phase. It is detected at compile time.
- Semantic error can be
  - undeclared variable.

Eg

```
int a = "hello";   // Not compatible
```

- Elimination of Left Recursion.

$A \rightarrow A\alpha / B$
$A \rightarrow BA'$ $A' \rightarrow \alpha A' / \varepsilon$

Q] $S \rightarrow ABC$
$A \rightarrow Aa / Ad / b$
$B \rightarrow Bb / e$
$C \rightarrow Cc / g.$

Soln.
$S \rightarrow ABC.$
$A \rightarrow bA'$
$A' \rightarrow aA' / dA' / \varepsilon$
$B \rightarrow eB'.$
$B' \rightarrow bB' / \varepsilon$
$C \rightarrow gC$
$C' \rightarrow cC' / \varepsilon$

PAGE No.
DATE / /

- Left Factoring.

Grammar

Ambigious ✗ — Unambigious ✓

Grammar

Left Recursion ✗ — Right Recursion. ✓

$A \rightarrow A\alpha / B$

Grammar

~~Ambiguite~~ Deterministic — Non-Deterministic. ✗

$A \rightarrow \alpha B_1, \alpha B_2, \alpha B_3, \ldots$

Q) $A \rightarrow aB \mid abc.$

Soln. $A \rightarrow aA'$

$A' \rightarrow B' \mid bc.$

Q) $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b.$

Soln. ~~S/A/S'/a~~ $S \rightarrow iEtSS' \mid a$

$S' \rightarrow \varepsilon \mid eS$

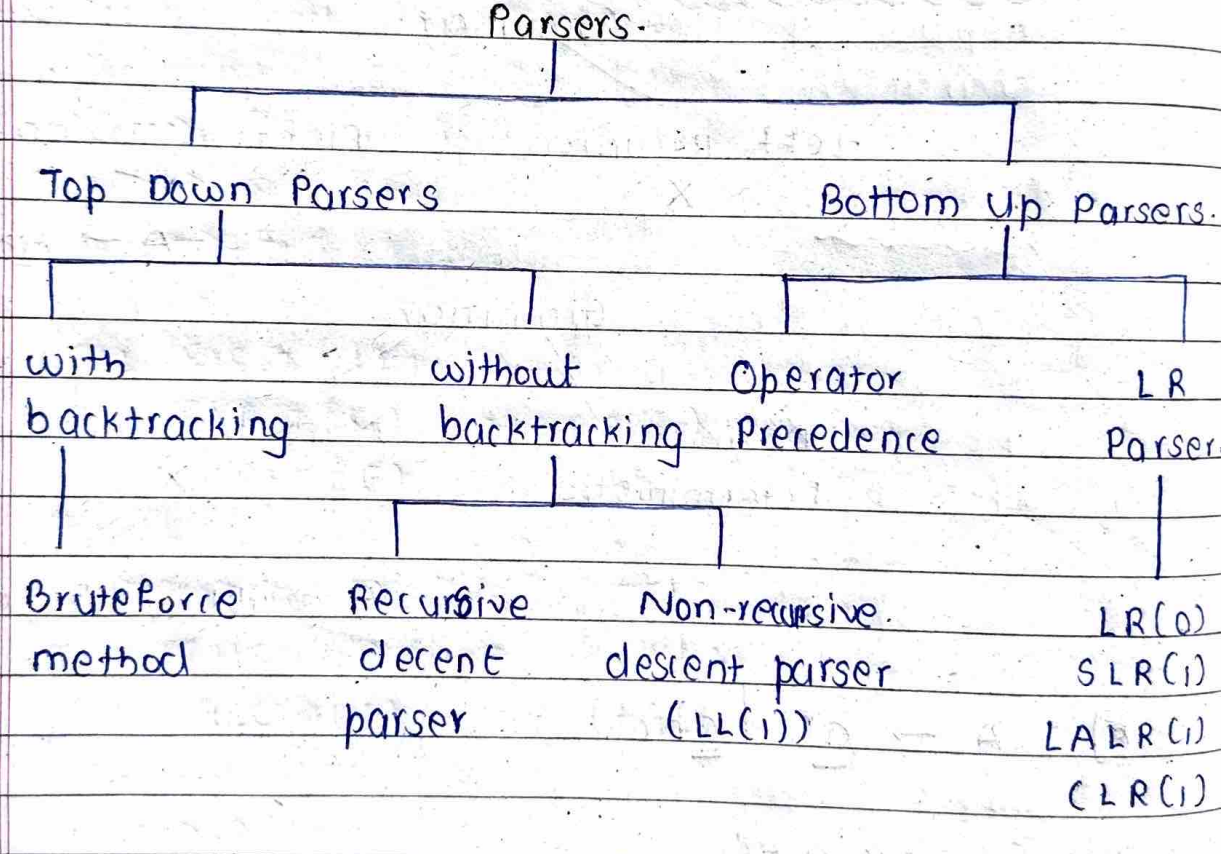
$E \rightarrow b$

PAGE No.
DATE / /

Q] $S \rightarrow aSSbS / aSaSb / abb / b$.

Soln. $S \rightarrow aSS' / abb / b$
$S' \rightarrow SbS / aSb$

- Parsers.



- Recursive descent.

eg $E \rightarrow iE'$
$E' \rightarrow +iE' / \varepsilon$

lookahead l = getchar

PAGE No.
DATE / /

Soln.

```
E()
{
  if(l == 'i')
{
    match('i');
    E'();
}
}

E'()
{
 if (l == '+')
{
   match('+');
   match('i');
    E'();
}
}

match(char t)
{
  if (l == t)
  l = getchar();
  else
  printf("error");
}

main()
{
  E();
  if(l == $)
printf("successfully
        Parsed");
}
```

- Operator Precedence Parser.

- It is mainly used to define mathematical operators for the compiler.
- There are certain rules on operator grammar:-

- 2 variables can't be adjacent to each other.
- There shouldn't be epsilon on RHS.

S → ABb ✗
S → abA ✓
S → ∈ ✗

Follow set will never contain NULL.

Follow of start will have $.

PAGE No. / DATE / /

- FIRST and FOLLOW.

- Rule for FOLLOW.

1] Terminal → write it as it is.
2] Non-Terminal → write its FIRST elements.
3] Last element → write FOLLOW of LHS.

Q] S → iEtSS1 / a
S1 → eS / ε
E → b.

Soln. CHECK LHS.
FIRST(S) = {i, a}
FIRST(S1) = {e, ε}
FIRST(E) = {b}

CHECK RHS
FOLLOW(S) = {e, $}
FOLLOW(S1) = {e, $}
FOLLOW(E) = {t}.

Q] S → A
A → aB / Ad
B → bBCf / f
C → h / gD
D → bE
E → cD

Soln.

FIRST(S) = {a}
FIRST(A) = {a}
FIRST(B) = {b, f}
FIRST(C) = {h, g}
FIRST(D) = {b}
FIRST(E) = {c}

FOLLOW(S) = {$}
FOLLOW(A) = {d, $}
FOLLOW(B) = {h, g, d}
FOLLOW(C) = {f}
FOLLOW(D) = {f}
FOLLOW(E) = {f}

- Predictive Parsing LL(1) table.

	i	t	a	e	b	$
→ S	S → iEtSS$_1$		S → a			
S$_1$				S$_1$ → eS S$_1$ → ε		
E					E → b.	

11th March 2023.

PAGE No.
DATE / /

- Shortcut for LL(1) table / Non-recursive predictive parsing

- Predictive Parsing.

1] Elimination of Left Recursion.
2] Left Factoring.
3] First and Follow Functions.
4] Predictive Parsing table.
5] Parse the ingput string.

6] $E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id.$

Soln.

- Eliminate Left Recursion.

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *F \mid \varepsilon$
$F \rightarrow (E) \mid id.$

- Left factoring not possible.

- FIRST.

If FOLLOW is of case 2 and it contains ε then consider FOLLOW of LHS.

PAGE No. / DATE / /

FIRST(E) = { (, id }.
FIRST(E') = { +, ε }
FIRST (T) = { (, id }
FIRST (T') = { *, ε }
FIRST (F) = { (, id }

- FOLLOW.

FOLLOW(E) = {), $ } FOLLOW of E
FOLLOW(E') = {), $ }
FOLLOW(T) = { +,), $ }
FOLLOW(T') = { +,), $ }
FOLLOW(F) = { *, +,), $ }.

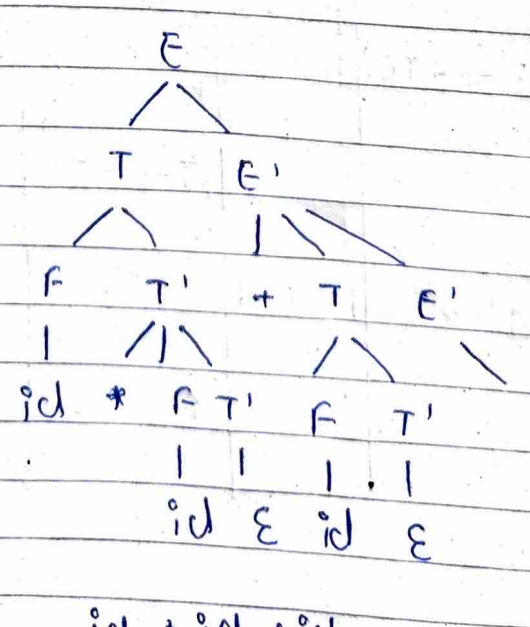
- Predictive Parsing Table. If LHS tends to ε then find FOLLOW of LHS !!

NTs	+	*	(	)	id	$
E			E → TE'		E → TE'	
E'	E' → +TE'			E' → ε		E' → ε
T			T → F T'		T → F T'	
T'	T' → ε	T' → *F		T' → ε		T' → ε
F			F → (E)		F → id	

PAGE No.
DATE / /

- ω = id * id+id

STACK	Input	Output
$ E	id * id+id $	
$ E' T	id * id+id $	E → TE'
$ E'T'F	id * id+id $	T → FT'
$ E'T'id	id * id+id $	F → id
$ E'T'	* id + id $	
$ E'T'F*	* id+id $	T' → * FT'
$ E'T'F	id+id $	
$ E'T'id	id+id $	F → id
$ E'T'	+id $	
$ E'	+id $	T' → ε
$ E'T+	+id $	E' → +TE'
$ E' T	id $	
$ E'T'F	id $	T → FT'
$ E' T'	$	F → id
$ E'	$	T' → ε
$	$	E' → ε.



id * id + id.