Page
Date

Aman Chauhan 1805158

① a) i) Token :- It is a valid sequence of characters which are given by lexeme. keywords, Constent, identifiers etc are example of token

ii) Pattern :- It describes a rule that must be matched by sequence of characters to form a token. It can be defined by regular rules. In the case of a keyword as a token, Pattern is just sequence of characters that form the keyword.

iii) Lexeme :- It is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token

iv) Augmented Grammer :- It is a grammer whose productions are augmented with Conditions expressed using features. Features may be associated with any non terminal symbol in a derivation.

v) Need Of Augmented Grammer :- It helps in generating a grammer from language and also helps for parsing.

vi) Recursive Descent Parser :- It is a kind of top-down parser built from a set

Aman Chauhan 1805158

of mutually recursive procedures where each such procedure implements one of the monterminals of the grammer. Thus the structure of the resulting program closely mirrors that of the grammer it recognizes.

1 b) The lexical analyzer is the first phase of compiler. Its main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis.

- It is implemented by making lexical analyzer be a subroutine.
- Upon receiving a 'get next token' command from parser, the lexical analyzer reads the input character until it can identify the next token.
- It may also perform secondary task at user interface
- One such task is stripping out from the source program comments and white space ~~in~~ in the form of blanks, tabs and newline character.
- Some lexical analyzer are divided into cascade of two phases, the first called called scanning and second is lexical analysis.
- The scanner is responsible for doing simple task while lexical analysis does the more complex task.

Page
Date

Aman Chauhan 1805158

Difficulties faced by Lexical Analyzer & There are several reasons for separating analysis these

- Simpler design is perhaps the most important consideration. The separation of lexical analysis often allows us to simplify one or other of these phases.
- Compiler efficiency is improved
- Compiler portability is enhanced

② a) Let consider a grammer $A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \alpha\beta_3$

This kind of grammer creates a problematic situation for top down parser. Parser can not decide which production must be choosen to parse the string. To remove this problem we use left ~~fa~~ factoring.

Left Factoring :- It is a process by which the grammer with common prefixes is transformed to make it useful for Top down parsers.

Algorithm :- We make one production for each common prefixes.

- The common prefix may be a terminal or non-terminal or a combination of both.
- Rest of the derivation is added by new productions.

Scanned with CamScanner

Aman Chauhan 1805158

$A \rightarrow a\alpha 1 \mid a\alpha 2 \mid a\alpha 3$

$A \rightarrow aA'$

$A' \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$

Example : $S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow \epsilon \mid eS$

$E \rightarrow b$

2 b) Eliminate Recursion :

If $A \rightarrow A\alpha \mid B$

then $A \rightarrow BA'$

$A' \rightarrow \alpha A' \mid \epsilon$

Example : $S \rightarrow S0S1S \mid 01$

$S \rightarrow 01S'$

$S' \rightarrow 0S1SS' \mid \epsilon$

2 c)

Top - Down	Bottom - UP
i) It is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammer.	i) It is parsing strategy that first looks at the lowest level of the parser tree and works up the parse tree by using the rules of grammer.

Aman Chauhan 1805158

ii)	It attempts to find the left most derivations for an input string.	ii) It can be defined as an attempts to reduce the input string to start symbol of a grammer.
iii)	We start parsing from top to down in this	iii) We start parsing from bottom to up in this.
iv)	This parsing uses Left Most Derivation.	iv) This parsing uses Right Most Derivation.
v)	It's main decision is to select what production rule to use in order to construct the string.	v) It's main decision is to select when to use a production rule to reduce the string to get the starting symbol.

② d)

	First	Follow
E → TE'	{(, id}	{$,), +, (, id}
E' → +TE' | E	{+, (, id}	{$,), +, (, id}
T' → FT'	{(, id}	{+, (, id}
T' → *FT' | E	{*, (, id}	{+, (, id}
F → (E) | id	{(, id}	{*, (, id}

Now the predictive Parsing Table is

Page
Date

Aman Chauhan 1805158

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