CD MID-1 QB

1. (b) Interpreter

2. (d) Lexical Analyser

3. Function of Loader: A loader is a program that places programs into memory and prepares them for execution.

4. (c) Relocation

5. Pattern: A set of strings in the input for which the same token is produced as output. This set of strings is described by a rule called a pattern associated with the token.

7. (c) Error handler

8. Symbol table: It is an important datastructure created and (manipulate) maintained by compilers inorder to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces etc.
   → It is used by both analysis and the Synthesis parts of a compiler.

8 6) (a) code optimization.

9. Lexical Analysis: It is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a lexical analyser or scanner.

10. Cousins of Compiler: 1) Preprocessor 2) Assembler 3) Loader / Linker - editor

11.

Interpreter	Compiler
• Translates program one statement at a time.	• Scans the entire program and translates it as a whole into machine code.
(It takes less amount)	
• high level → machine code	• high level → assembly code
• Python, Ruby use interpreters.	• C, C++ use compilers.

12. The only problem left with the lexical Analyser is how to verify the validity of a regular expression used in specifying the patterns of keywords of a language.
[(a) Defining the standard rules.]

13. Role of linker and Loader:

→ A loader is a program that places programs into memory and prepares them for execution.

→ Linker is a computer program that links and merges various object files together in order to make an executable file.

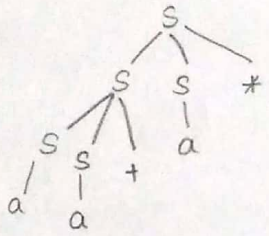
14 → Three Address Code
→ Quadruples
→ Triples
→ Indirect Triples

15. Compiler construction Tools

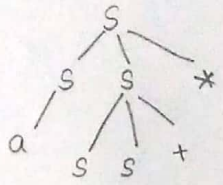
- Parser generators
- Scanner generators
- Syntax-directed translation engines
- Automatic code generators
- Data-flow engines

16. $S \rightarrow SS+ \mid SS* \mid a$

String : $aa+a*$



Left derivation tree



Not possible.

Not ambiguous.

17. Structure of LEX.

① Declaration section

```
%{
  ...
  ...
%}
```

② Rule section

```
%%
  ...
  ...
%%
```

③ Auxillary procedure / subroutine section

```
main()
{
  ...
  ...
}
```

18. A pass refers to the traversal of a compiler through the entire program.

19. (P)
Phase and Pass are two terms used in the area of compilers. A pass is a single time the compiler passes over the source code.

Phase is used to classify the compilers according to the constructions, while pass is used to classify compiler according to how they operate.

20. Phases of the compiler are categorised based on the way they compile.

(a) Analysis phase:
- → Machine independent
- → Language dependent

(b) Synthesis phase
- → Machine dependent
- → Language independent

21. Left Factoring: Left factoring is removing the common left factor that appears in two productions of the same non-terminal. It is done to avoid back-tracing by the parser.

Ex. $S \rightarrow Saa \mid Sbb$

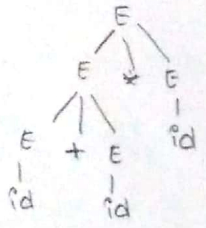
Sol. $S \rightarrow S'$

$S' \rightarrow aaS' \mid bbS' \mid \epsilon$

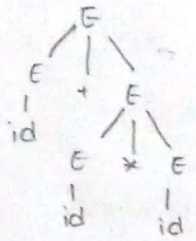
(P) A phase is logically interrelated operation that takes the source program in one representation and produces output in another representation.

22. Ambiguity: If a string can be derived from more than one parse tree, then it is called ambiguity.

Ex. $E \rightarrow E+E \mid E*E \mid E-E \mid id$.

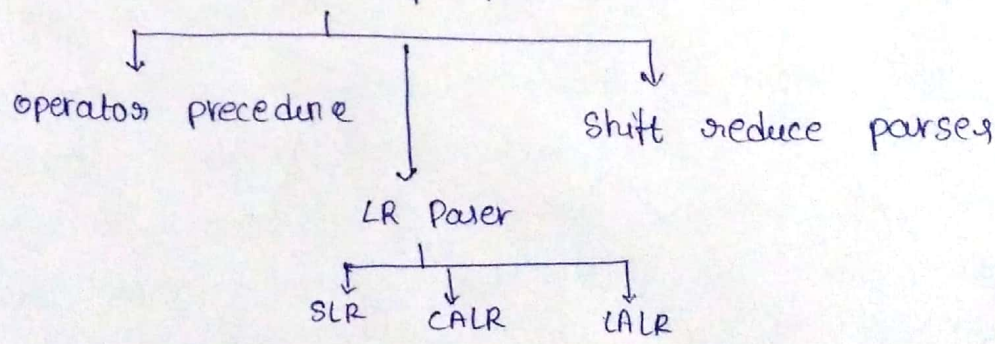


Left derivation



Right most derivation.

23 Different types of Bottom up parsers



24. Backtracking: It is defined as if one derivation of a production fails, the syntax analyzer restarts the process using different rules of same production. this technique may process the input string more than once to determine the right production.

27. Rules to remove ambiguity.

→ Highest preceedence operator should always be in the lowest level.

→ For left to right associativity: add a production where RHS first variable is equal to LHS variable.

→ Don't write the same variable name on both sides of operator.

28. Removing left recursion or left factoring cannot introduce ambiguity. this kind of transformation preserves ambiguity. If the CFG is already ambiguous, the result will be ambiguous too, and if the original is not, the resulting neither.

29. First(): It is applied to the r.h.s of a production rule, and tells us all the terminal symbols that can start sentences derived from that r.h.s.

Follow(): It is used only if the current non-terminal can derive; then we're interested in what could have followed it in a sentential form. A string can derive if and only if it is its First set.

30. A → A!

It is not left recursive and also not a valid grammar as '!' is not a part of the grammar.

80