Describe the execution process of source program in Compiler.

Describe the different phases of compiler with suitable block diagram.

Describe one pass compiler with suitable diagram.

Describe multi pass compiler with suitable diagram.

Differentiate one pass and multi pass compiler.

Describe the different working principles of Syntax Analysis.

How Semantic Analysis is maintaining the Rule of a language?

Implement this below mentioned high level language with different stages of Compiler.

f = ((c * 9) / 5) + 32

Describe the contexts of Analytics and Synthesis in terms of Pass.

How Front End and Back End terms are related with Compiler Passes?

Why 1st Pass is platform independent and why 2nd pass is platform dependent in Multi Pass Compiler?

How Code Optimization and Code Generator are related with Compiler passes?

Describe the contexts of Three Address Code.

If we want to design a compiler for different programming language for same machine, in that case for each programming language there is requirement of making Front end/first pass for each of them and only one Back end/second pass. Implement and discuss this case study with suitable block diagram.

If we want to design a compiler for same programming language for different machine/system, in that case we make different Back end for different Machine/system and make only one Front end for same programming language. Implement and discuss this case study with suitable block diagram.

Write a program to calculate the sum of 1st n natural numbers. Now calculate the following:

a) Total Numbers of tokens and lexemes.

b) Total numbers of non-tokens.

Write a program to calculate the total numbers of even, odd, positive and negative numbers. Now calculate the following:

a) Total Numbers of tokens and lexemes.

b) Total numbers of non-tokens.

Write a program to swap two numbers without using 3rd variable. Now calculate the following:

a) Total Numbers of tokens and lexemes.

b) Total numbers of non-tokens.

Write a program to calculate the factorial of a given number. Now calculate the following:

a) Total Numbers of tokens and lexemes.

b) Generate the Three Address Codes for the main logic of factorial calculation.

Write a program to generate a Fibonacci series. Now calculate the following:

a) Total Numbers of tokens and lexemes.

b) Generate the Three Address Codes for the main logic of Fibonacci series.

Explain the concepts of Quadruples and Triples.
a := -b * c + d

Implement this above mentioned statement with Quadruples and Triples.
Code Optimization is not mandatory phase of Compilation process - Justify your answer.
What is the significance of Intermediate Code in Compilation Process?
How Semantic Analyser keeps track of identifiers, their types and expressions?
Why Lexical Analysis is known as a Scanner?
How DFA is related with Lexical Analysis?
Explain the different steps of Finite Automata with respect to a Compilation Process.
Explain the concepts of Transition Function.
Implement a suitable block diagram of Lexical Analysis.
What are the significances of Tokens and Non-Tokens?
Explain the concepts of FLEX.
What is the significance of yylex()?
Describe the working principle of FLEX with a block diagram.
Explain the concepts of Definition Section of LEX.
Explain the concepts of Rule Section of LEX.
Describe the following patterns:

[0-9]

[0+9]

[0, 9]

[^A-Z]

a{2, 4}
How User Code Section will be implemented?
What is Symbol Table?
Explain the syntax of a Symbol Table.
Describe different implementional area of Symbol Table.
Describe insert() of Symbol Table.
Describe lookup() of Symbol Table.
How a Data Structure of Symbol Table will be used?
How Scope Information will be described in Symbol Table?
Draw the schematic diagram of Non-Recursive Predictive Parsing.
What are the data structures used in Non-Recursive Predictive Parsing?
What do you need to do in order to avoid backtracking in top-down parsing?
Make the following grammar suitable for predictive parsing:

$$A \rightarrow a\beta_1 | a\beta_2$$

Where A is a non-terminal, “a” is a terminal and β_1, β_2 are strings of non-terminals and terminals.
Left recursive grammar is not suitable for which broad category of parsing? What remedy do we take?

Which phase of the compiler processes linear input and produces hierarchical output?

Which phase of the compiler processes hierarchical input and produces linear output?

Which phase of the compiler processes hierarchical input and produces hierarchical output?

What kinds of grammars are used in Lexical Analysis and Syntax Analysis?

Which phases of a compiler are target machine-dependent and which phases are independent of the target machine?

What is the right-most derivation? Give an example.

What is the left-most derivation? Give an example.

Explain canonical derivations.

What is the left-sentential form?

What is the right-sentential form?

What is a sentence of a grammar G?

What is Context-Free Language?

Convert the following grammar so that it can be parsed by Top-Down Predictive Parser

$$S \rightarrow Aa|b$$

$$A \rightarrow Ac|Sd|\epsilon$$

Give an example of left linear grammar. Convert it to right equivalent linear grammar.

Which kind of grammar is required to specify the statement syntax of a language? Describe how to specify such a language. (What are the parameters?) Can the same grammar be used to specify tokens?

Explain the Top-Down and Bottom-Up Approach with an example.

What is ambiguous grammar? Is the following ambiguous? Explain with example.

$$E \rightarrow E + E|E - E|E^*E|id|num$$

Give the parse tree for the statement $a = b + c * 60.$

For the following grammar:

$$list \rightarrow list + digit|list - digit|digit$$

$$digit \rightarrow 0|1|\ldots|9$$

Derive the sentence $(9 + 5 - 4)$ using left-most derivation

Consider the following left-linear grammar

$$S \rightarrow Sab|Aa$$
$$A \rightarrow Abb|bb$$

Find out an equivalent right-linear grammar.

What is the purpose of Context-Free Language?

How do you describe the grammar for it?

Is it possible to describe tokens using this grammar?

What is ambiguous grammar? Give an example. How would you handle ambiguity?

What is left recursive grammar? Give an example. Convert the same to right recursive grammar.

Given the grammar

$$exp \rightarrow exp\ addop\ term|term$$
$$addop \rightarrow +|-$$
$$term \rightarrow term\ mulop\ factor|factor$$
$$mulop \rightarrow *$$
$$factor \rightarrow (exp)|number$$

Write down leftmost derivations, parse trees, and abstract syntax trees for the following expression: $3 + 4 * 5 - 6$.

Consider the context-free grammar

$$S \rightarrow SS^{+}|SS^{*}|a$$

a. Show how the string $aa + a^{*}$ can be generated by this grammar.
b. Construct a parse tree for this string
c. What language does this grammar generate? Justify your answer.

What are the data structures used in

Lexical Analysis, Top Down Parsing and Bottom Up parsing.

Given grammar:

$$X \rightarrow cYd$$
$$Y \rightarrow a|ab$$

Where X and Y: Non terminals; a,b,c,d: terminals
(a) Show the steps of building top-down parse tree for the input string: **cabd** using the above grammar.
(b) What is this method called? Do you get a unique parse tree?
(c) Can you build a predictive parser for this grammar? Explain.

Given grammar:

$$E \rightarrow E + E|E * E|num$$

+ means addition operator
* means multiplication operator
num represents number

(a) Draw Parse Trees for the string 5 + 3 * 4 + 6 using the above grammar and any order of derivation
(b) Do you get a unique parse tree? Provide reason.
(c) What kind of grammar is this? Why is it called so?
(d) Can you design a top-down predictive parser using this grammar?

(a) What are the types of top-down $E \rightarrow E + T$ parsing?
(b) State the major difference between them
(c) Why is the following production not suitable for top-down parsing?

(d) What measures would you take?
(e) In which parsing method the above grammar will not be a problem?

Consider the following grammar representing simplified LISP like expressions:

$$lexp \rightarrow atom|list$$
$$atom \rightarrow number|identifier$$
$$list \rightarrow (lexp - seq)$$
$$lexp - seq \rightarrow lexp - seqlexp|lexp$$

a. Write a leftmost and a rightmost representing derivation for the string
(a 23 (m x y)
b. Draw a parse tree for the string of part (a)

Given the following grammar

$$statement \rightarrow if - stmt|other|\epsilon$$
$$if - stmt \rightarrow if(exp)statementelse - part$$
$$else - part \rightarrow else\ statement|\epsilon$$
$$exp \rightarrow 0|1$$

a. Draw a parse tree for the string
if (0) if(1) other else else other
b. What is the purpose of the two else-s?
c. Is similar code permissible in C? Explain.

Construct a predictive parsing table for the grammar

$$S \rightarrow iEtSS^{'}|a$$
$$S^{'} \rightarrow eS|\epsilon$$
$$E \rightarrow b$$

Here S is the start symbol and $S^{'}$, E are non-terminals,
i, t,,a,e,b are terminals.

The following grammar generates all regular expressions over the alphabet of letters:

$$rexp \rightarrow rexp|rexp|rexp\ rexp|rexp*|(rexp)|letter$$
$$letter \rightarrow [a-b]$$

Give a derivation for the regular expression $(ab|b)^*$ using this grammar.
b. Give a derivation for the regular expression abb* using this grammar.
c. Show that this grammar is ambiguous.
d. Rewrite this grammar to establish the correct precedence for the operators
e. What associativity does your answer above give to the binary operators? Why?

Given the grammar $A \rightarrow AA|(A)|\epsilon$
(a) List 2 strings generated by this grammar
(b) Describe the language it generates
(b) Show that it is ambiguous

Convert the following NFA to DFA:

Given the grammar

$$exp \rightarrow expaddop\ term|term$$
$$addop \rightarrow +|-$$
$$term \rightarrow termmulopfactor|factor$$
$$mulop \rightarrow *$$
$$factor \rightarrow (exp)|number$$

Write down leftmost derivations, parse trees, and abstract syntax trees for the following expressions:
a. 3+4*5-6 b. 3*(4-5+6)
c. 3-(4+5*6)*(4*5+6)+8

Grammar given is

$$dtype \rightarrow simple|id|array\ [simple]\ of\ dtype$$
$$simple \rightarrow integer|char|num\ to\ num$$

Explain if the string “array [1 to 100] of integer” can be generated from the non-terminal dtype.
“array [1 to 100] of ^myval”
Programmer defined identifier and num are in bold letters.
Here, 1 , 100 are of token num, myval is an identifier.

Draw Parse tree for the following statement:

a = (b + c + d)*e

State true or false: Every context-free grammar is ambiguous.

Draw Parse tree for the following statement:

e = a*b + c*d

Consider the following grammar-
S → aB / bA

S → aS / bAA / a

B → bS / aBB / b

Let us consider a string w = aaabbabbba

Derive the string w using leftmost derivation.
Consider the following grammar-

S → aB / bA

S → aS / bAA / a

B → bS / aBB / b

Let us consider a string w = aaabbabbba

Derive the string w using rightmost derivation.
Consider the grammar-

S → A1B

A → 0A / ∈

B → 0B / 1B / ∈

For the string w = 00101, find Leftmost derivation, Rightmost derivation, Parse Tree.
Consider the grammar-

S → bB / aA

A → b / bS / aAA

B → a / aS / bBB

For the string w = bbaababa, find Leftmost derivation, Rightmost derivation, Parse Tree.
Consider the following grammar and eliminate left recursion-

A → ABd / Aa / a

B → Be / b
Consider the following grammar and eliminate left recursion-

E → E + E / E x E / a
Consider the following grammar and eliminate left recursion-

E → E + T / T

T → T x F / F

F → id
How Left Factoring works in a Parse Tree?
Do left factoring in the following grammar-

A → aAB / aBc / aAc
Do left factoring in the following grammar-

S → bSSaaS / bSSaSb / bSb / a
Bottom up parsing is used to construct a parse tree for an input string. How this phenomenon will be justified?
In the top down parsing, the parsing starts from the start symbol and transform it into the input symbol. How this phenomenon will be justified?
Consider the following Production Rules:

E → T
T → T * F
T → id

F → T
F → id

Input String: id * id

Organize a Parse Tree.
Explain the concepts of Shift Reducing Parsing.
Consider the following Grammar:

S → S+S
S → S-S
S → (S)
S → a

Input string:

a1-(a2+a3)

Organize a Shift Reducing Parsing Table.
What are the different data structures are used in Shift Reducing parsing?
Consider the following grammar-

E → E – E

E → E x E

E → id

Parse the input string id – id x id using a shift-reduce parser.
Consider the following grammar-

S → (L) | a

L → L , S | S

Parse the input string (a , (a , a)) using a shift-reduce parser.
Consider the following grammar-

S → T L

T → int | float

L → L , id | id

Parse the input string int id , id ; using a shift-reduce parser.
Considering the string “10201”, design a shift-reduce parser for the following grammar-

S → 0S0 | 1S1 | 2
Explain the concepts of Operator Precedence Parsing?
Consider the following grammar-

E → EAE | id

A → + | *

Construct the operator precedence parser and parse the string id + id * id.
Consider the following grammar-

S → (L) | a

L → L , S | S

Construct the operator precedence parser and parse the string (a , (a , a)).

Write Rules to construct FIRST Function and FOLLOW Function.

Explain the different techniques of LR parser.

Write the rules to construct the SLR parsing table.

What is syntax directed translation (SDD)?

Differentiate between synthesized translation and inherited translation.

Write quadruples, triples for the expression: -(a*b)+(c+d)-(a+b+c+d)

Write the three address statement with example for:
a. Assignment
b. If-then-else statement
c. While, do-while statement
d. Switch case statement

What is code optimization? Explain machine dependent and independent code optimization.

What is common sub-expression and how to eliminate it? Explain with example.

Explain the following with example to optimize the code:

a. Dead code elimination
b. Variable elimination
c. Code motion
d. Reduction in strength

In syntax directed translation, every non-terminal can get one or more than one attribute or sometimes 0 attribute depending on the type of the attribute. - Justify this statement with your views.

What are the different attributes should be considered in SDD?

Consider the following Production Rules.
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (F)$
$F \rightarrow num$

Derive the Semantic Rules for every statements.

Explain Postfix notation.

How postfix notation is a linear representation of a syntax tree?

Consider the following Production

$E \rightarrow E1\ op\ E2$
$E \rightarrow (E1)$
$E \rightarrow id$

Show the semantic rule and program fragment.

Explain the concepts of Syntax Tree.

Consider a sentence: id + id * id. Design a syntax tree.

Describe the contexts of Translation of Assignment Statements.

Consider the grammar

$S \rightarrow id := E$
$E \rightarrow E1 + E2$

Derive the translation scheme of above grammar.

What is the responsibility of Emit function in Three Address Code?

Describe the different features of YACC.

Describe the concepts of Lexical Error.

Describe the concepts of Syntax Error in terms of Complier Construction.

Describe the concepts of Semantic Errors in terms of Complier Construction.

Explain the concepts of Code Generator.

What are the issues can be happened when input will be fed to the Code Generator?

What are the different classifications of Target Program regarding a Code Generator?

What are the different memory management issues can be created for a Code Generator?

Nature of instruction set of the target machine should be complete and uniform. Describe this statement with your views.

How register allocation process can be acclaimed during code generation?

How loop optimization process can be happened?

How induction variable elimination process works in Code Optimization?

How DAG will be represented for basic blocks in Code Optimization?

Consider the following three address statement:

```
S1:= 4 * i
S2:= a[S1]
S3:= 4 * i
S4:= b[S3]
S5:= s2 * S4
S6:= prod + S5
Prod:= s6
S7:= i+1
i := S7
if i<= 20 goto (1)
```

Design different DAG steps for this above mentioned code.

What is Data Flow Graph Analysis?

Consider the following code:

```
a = 1;
b = 2;
c = 3;
if (....) x = a + 5;
else x = b + 4;
c = x + 1;
```

Implement the strategy of global optimization policy for this above mentioned code.

Describe static semantics of programming languages.

Discuss the dynamic semantics of the programming languages.

State true or false: Attribute grammars are extensions of context-free grammar.

Describe the mathematical structure of attribute grammar.

State true or false: Let G be an attribute grammar. We can associate both synthesis and inherited attributes with the symbols present in the grammar G.

State true or false: Synthesis attributes are computed in bottom-up fashion.

Let G be an attribute grammar. Describe the synthesis and inherent attributes of the grammar G using an example.

Let G be an attribute grammar with the following rules:

1. $DList \rightarrow D \mid DList\ ;\ D$ **2.** $D \rightarrow T\ L\ \{L.type \downarrow := T.type \uparrow\}$
3. $T \rightarrow int\ \{T.type \uparrow := integer\}$ **4.** $T \rightarrow float\ \{T.type \uparrow := real\}$
5. $L \rightarrow ID\ \{ID.type \downarrow := L.type \downarrow\}$
6. $L_1 \rightarrow L_2\ ,\ ID\ \{L_2.type \downarrow := L_1.type \downarrow; ID.type \downarrow := L_1.type \downarrow\}$
7. $ID \rightarrow identifier\ \{ID.name \uparrow := identifier.name \uparrow\}$

Derive the parse tree for the string **int a; float x.** Next derive the corresponding dependence graph.

Describe S-attributed grammar using an example.

Let G be an attribute grammar with the following rules:

Production	Semantic Rules
$D \rightarrow T\ L$	L.in = T.type
$T \rightarrow$ **int**	T.type = integer
$T \rightarrow$ **real**	T.type = real
$L \rightarrow L_1$, **id**	L_1.in = L.in, addtype(**id**.entry,L.in)
$L \rightarrow$ **id**	addtype(**id**.entry,L.in)

L.in means the symbol L is associated with an inherited attribute *in.* Similarly, type is a synthesized attribute and

T.type represents that the symbol T is associated with a synthesized attribute *type*.

Derive the parse tree for the string **real id1,id2,id3.** Moreover, derive the dependency graph from the parse tree based on the above-mentioned rules.

Let G be an attribute grammar with the following rules:

Production	Semantic Rules
$D \rightarrow T\ L$	L.in = T.type
$T \rightarrow$ **int**	T.type = integer
$T \rightarrow$ **real**	T.type = real
$L \rightarrow L_1$, **id**	L_1.in = L.in, addtype(**id**.entry,L.in)
$L \rightarrow$ **id**	addtype(**id**.entry,L.in)

Derive the parse tree for the string **int p, q.**

Also derive the dependency graph from the parse tree.

Describe circular dependency of the attributes using an example.

Describe L-attributed grammar.

Give an example of a non-L-attributed SDD and explain why it is non-L-attributed.

Derive the three-address code, quadruples and triples of the following instruction: a + b * c - d / (b * c).

Derive the syntax tree and directed acyclic graph of the following instruction: a + b * c - d / (b*c).

Write down a program for calculating the dot product of two vectors. Derive the intermediate code from the program.

Write down a program to calculate the factorial of an integer n. Next, derive the intermediate code for the corresponding program.

Derive the three-address code, quadruples and triples for the following instruction: a = b * -c + b * -c.

Derive the syntax tree and directed acyclic graph for the following statement: a = b * -c + b * -c.

State true or false: An attribute associated with the symbols in an attribute grammar can be both inherited and synthesized.

Derive the three-address code, quadruples, triples, syntax tree and directed acyclic graph (if possible) for the following instruction:

a + b * c + d.

Discuss some use of the information gathered from the symbol table.

State true or false: Symbol tables typically need to support multiple declarations of the same identifier within a program.

State true or false: Intermediate codes can be easily translated into machine code.

Derive the machine code of the following instruction: e = a + b * c + d.

Describe the jump instructions in three-address codes using an example.

State true or false: In the three-address code, the right-hand side of any assignment instruction has at most two operands and one operator.

Draw the parse tree for the following instruction: a + a * (b – c) + (b – c) * d

Draw the directed acyclic graph derived from the parse tree for the following instruction: a + a * (b – c) + (b – c) * d.

Also derive the triples for the above instruction.