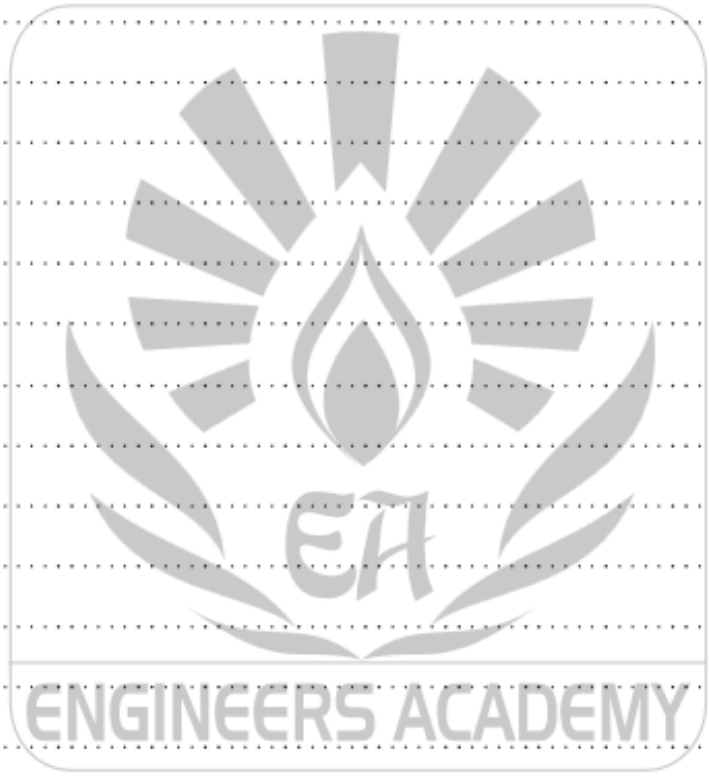
UNIT-XIII

COMPILER DESIGN

ENGINEERS ACADEMY®
Your GATEway to Professional Excellence
IES • GATE • PSUs • JTO • IAS • NET

www. eapublications.org

NOTES



CHAPTER 1

INTRODUCTION TO COMPILER DESIGN

OBJECTIVE QUESTIONS

1. Consider the following issue :

I. Simplify the phase

II. Compiler efficiency is improved

III. Compiler works faster

IV. Compiler protability is enhanced

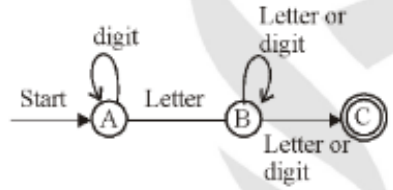
What is/are true in context of lexical analysis

(a) I, II, III (b) I, III, IV

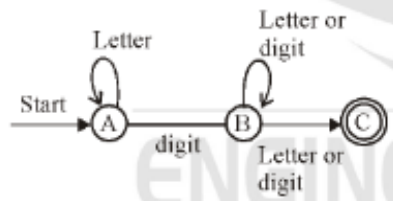
(c) I, II, IV (d) All

2. Which one of the following transition diagram correctly defines identifier and keywords ?

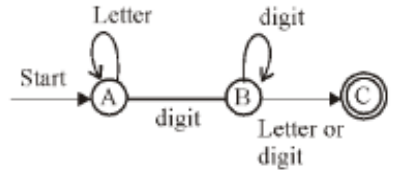
(a)



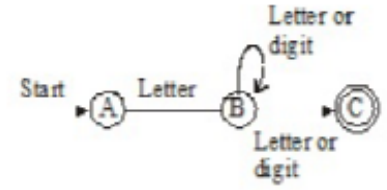
(b)



(c)



(d)



3. If an error is detected within a statement, the type assigned to the statement is

(a) type constructor

(b) type expression

(c) error-type

(d) type-error

4. Type checking is normally done during

(a) syntax analysis

(b) Lexical analysis

(c) code optimization

(d) syntax directed translation

5. System programming such as compilers are designed so that they are

(a) recursive (b) serially usable

(c) non-reusable (d) re-enterable

6. Which of the following m/c provided the data structure to the user program

(a) Compiler

(b) Assembler

(c) Preprocessor

(d) Loader

7. In the grammer

$E \rightarrow E + T/T$

$T \rightarrow i$

(a) + is left associative

(b) + is right associative

(c) sometimes left associative sometimes right

(d) None of these

8. The grammer for regular expression

$R \rightarrow R + R / R^*$

$R \rightarrow R\ R$

$R \rightarrow a/b/(R)$

(a) Ambigous

(b) Unambigous

(c) sometimes ambiguous

(d) None

9. Consider the following statements :

S1 : The set of string described by a rule is called pattern associated with the token

S2 : A lexeme is a sequence of character in the source program that is matched by pattern for a token

Which of the statements is / are true ?

(a) Both statements are true

(b) S1 is true but S2 is false

(c) S2 is true but S1 is false

(d) Both statements are false

10. Consider the following statements :

S1 : If the keyword are not reserved then lexical analyzer must distinguish between keyword and user defined identifier

S2 : The statement

DO 5 1 = 1, 25 is having six token

Which of the statements is/are true ?

(a) Both statements are true

(b) S1 is true but S2 is false

(c) S2 is true but S2 is false

(d) Both statements are false

11. Match the following:

List-I

A. Preprocessor **B.** Assembler

C. Loader **D.** Linker

List-II

1. Resolving external reference

2. Loading the program

3. Producing reloca table machine code

4. Allow user to define shorthand for longer construct

5. Loading program and link editing

Codes:	**A**	**B**	**C**	**D**
(a)	4	3	2	1
(b)	4	3	1	2
(c)	4	3	5	2
(d)	4	3	5	1

12. Consider the grammer : $S \rightarrow OS1|01$

Which one of the following statements is true ?

(a) The grammer is ambiguous

(b) The grammer is unambiguous

(c) The language generated by grammer is $0^n\ 1^n$

(d) Both (b) and (c)

13. Consider the following grammer : $S \rightarrow S(S)\ S \mid e$

Which one of the following statement describe the given grammer best ?

(a) The grammer is ambiguous

(b) The grammer is unambiguous

(c) The grammer is represent all balanced string

(d) Both (b) and (c)

14. Consider the following grammer :

$S \rightarrow aSbS\ |bSaS \mid \in$

How many different parse trees are possible for string ababab ?

(a) 2 (b) 3 (c) 4 (d) 5

15. Suppose I is a binary operation and consider the grammer

$E \rightarrow T \uparrow E \mid T$

$T \rightarrow id$

Which of the following statement is true ?

(a) $\uparrow$ Operation is left associative

(b) $\uparrow$ operating is right associative

(c) Associativity is not decidable

(d) None of these

16. Which of the following statements are TRUE/

I. There exist parsing algorithms for some programming languages whose complexities are less than $0(n^3)$

II. A programming language which allows recursion can be implemented with static storage allocation

III. No L-attributed definition can be evaluated in the framework of bottom-up parsing

IV. Code improving transformations can be performed at both source language and intermediate code level

(a) I and II (b) I and IV

(c) III and IV (d) I, III and IV

17. Match the following according to input (from the left column) to the compiler phase (in the right column) that processes it:

(P) Syntax tree **(i)** Code generator

(Q) Character stream **(ii)** Syntax analyzer

(R) Intermediate representation **(iii)** Semantic analyzer

(S) Token stream **(iv)** Lexical analyzer

(a) P → (ii), Q →(iii), R → (iv), S → (i)

(b) P → (ii), Q → (i), R → (iii), S → (iv)

(c) P → (iii), Q →(iv), R → (i), S → (ii)

(d) P → (i), Q → (iv), R → (ii), S →(iii)

18. A lexical analyzer uses the following patterns to recognize three tokens T_1, T_2, and T_3 over the alphabet {a,b,c }.

T_1 : a? (b/c)*a

T_2 : b? (a/c)*b

T_3 : c? (b/a)*c

Note that 'x'?' means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix.

If the string *bbaacabc* is processed by the analyzer, which one of the following is the sequence of tokens it outputs ?

(a) $T_1T_2T_3$ (b) $T_1T_1T_3$

(c) $T_2T_1T_3$ (d) T_3 T_3

❑❑❑

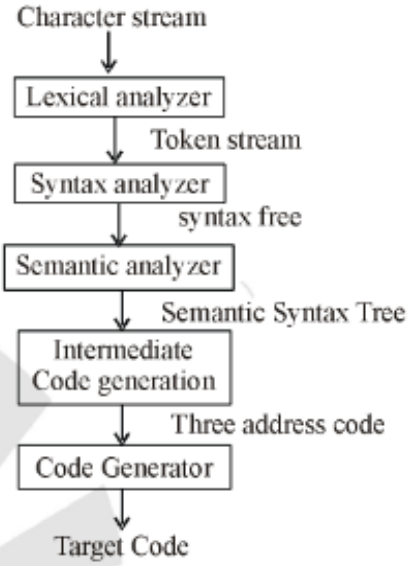
Answer Key

1. *Ans. (c)*
2. *Ans. (d)*
3. *Ans. (d)*
4. *Ans. (d)*
5. *Ans. (d)*
6. *Ans. (c)*
7. *Ans. (a)*
8. *Ans. (a)*
9. *Ans. (a)*
10. *Ans. (b)*
11. *Ans. (d)*
12. *Ans. (d)*
13. *Ans. (d)*
14. *Ans. (c)*
15. *Ans. (b)*
16. *Ans. (b)*

Recursion can not be implemented with static storage allocation. L-attributed definition can be evaluated if all rules are at the end and all attributes are synthesized.

So I and IV are correct.

17. *Ans. (c)*



18. *Ans (d)*

In T_3T_3, From first T_3 bbaac is taken, from second T_3 abc is taken, longest possible prefix.

Hence T_3T_3 token output.

❑❑❑

CHAPTER 2

CLASSIFICATION OF CONTEXT FREE GRAMMARS

OBJECTIVE QUESTIONS

1. After removing left recursion from the following $A \rightarrow A\alpha/\beta$ the resulting grammar will be

(a) $A - \beta A'$ $A' \rightarrow \alpha A'/\in$

(b) $A \rightarrow \alpha A'$ $A' \rightarrow \beta A'/\in$

(c) $A \rightarrow \alpha\beta A'$

(d) None of these

2. Consider the following grammar

Stmt $\rightarrow$ if expr then stmt else stmt

| if expr then smt

After removing left factoring the resulting grammar will be

(a) stmt $\rightarrow$ if expr then E' stmt

E' $\rightarrow$ else stmt | $\in$

(b) stmt $\rightarrow$ if expr then stmt E'

E' $\rightarrow$ else stmt | $\in$

(c) stmt $\rightarrow$ else stmt E

E $\rightarrow$ if stmt else stmt

(d) None of these

3. Consider the following grammar

$S \rightarrow iEtS \mid iEtSeS/a$

$E \rightarrow b$

After removing left factoring the resulting grammar will be

(a) $S \rightarrow iEtSS' \mid a$ $S' \rightarrow eS \mid \varepsilon$ $E \rightarrow b$

(b) $S \rightarrow iEtSS' \mid a$ $S' \rightarrow eS$ $E \rightarrow b$

(c) $S \rightarrow iEtSS' \mid a$ $S' \rightarrow eS \mid \in$

(d) None of these

4. The role of predictive parsing is

(a) To construct a top-down parser that never backtracks

(b) To construct a top-down parser that backtracks

(c) To construct a bottom-up parser that never backtracks

(d) None of these

5. First (α) can be described as,

(a) The set of tokens that appears in first symbol of one or more string generated from α.

(b) The set of tokens that appears in middle symbol of one or more string generated from α.

(c) The set of tokens that appear in last

(d) Data insufficient.

6. Suppose two productions are there $A \rightarrow a$ and $A \rightarrow b$. The recursive descent parsing without backtracking requires,

(a) first (α) and first (β) must be in same set

(b) first (α) and first (β) must be disjoint

(c) first (α) and first (β) must be equal

(d) data insufficient

7. Which of the following is true regarding LL (1) grammar ?

(a) No ambiguous or left recursive grammar can be LL (1)

(b) No ambiguous but left recursive grammar can be LL (1)

(c) No left recursive but ambiguous grammar can be LL (1)

(d) Both ambiguous and left recursive grammar can be LL (1)

8. Every LR (O) grammer is

(a) an LR (1) grammer

(b) may be an LR (1) grammer

(c) sometimes not an LR (1) grammer

(d) None

9. Every LALR (1) grammer is

(a) an SLR (1) grammer

(b) may not be an SLR (1) grammer

(c) Is new an LR (0) grammer

(d) None

10. An ambiguous grammer

(a) is never LR (1) (b) is never SLR (1)

(c) is never LR (0) (d) All

11. Regarding LALR (1) and LL (1)

(a) both are equivalent

(b) LALR (1) is more powerful

(c) LL (1) is more powerful

(d) None

12. Every LL(1) grammer is

(a) SLR (1) (b) LR (0)

(c) LR (1) (d) None of the above

13. When there are no reduce/reduce conflicts in LR(1) in corresponding LALR (1)

(a) no reduce reduce conflict

(b) No shift shift conflict

(c) reduce/reduce conflicts may occur

(d) none

14. The grammer $S \rightarrow T / \in / a$

$T \rightarrow S/a$

(a) is ambiguous and hence not SLR (1)

(b) is ambiguous & LR (0)

(c) is not LALR (1) but is SLR (1)

(d) is LR (1)

15. The grammer $S \rightarrow TA$

$A \rightarrow \in / +TA$

$T \rightarrow i/n$

(a) is LL (1)

(b) is LALR (1)

(c) LR(1) but not LALR (1)

(d) None

16. Eliminate left recursion from the following grammer

$S \rightarrow (L)/a$

$L \rightarrow L, S/S$

(a) $S \rightarrow (L) / a$ (b) $S \rightarrow (L)/a$

$L \rightarrow SA$

$A \rightarrow ,SA / \in$

(c) $S \rightarrow (L)/a$ (d) None

$L \rightarrow L. S/S$

$L \rightarrow SA$

17. The following grammer

$A \rightarrow Bb / Cd$

$B \rightarrow \in / aB$

$C \rightarrow cC / \in$

(a) is left recursive and when a not LL (1)

(b) is right recursive hence not LL (1)

(c) is LL (1)

(d) None of these

18. Which of the following is SR parser

(a) predictive parser (b) Recursive descent

(c) Bottom up parser (d) Brute force method

19. An inadequate state in LR parser is result of

(a) Shift-shift conflict

(b) Reduce-shift conflicts

(c) Bottom up parser

(d) Brute force method

20. Where there are no shift/reduce conflicts in CLR(1), in the corresponding LALR(1)

(a) No reduce-reduce conflicts

(b) No shift-reduce conflicts

(c) Shift-reduce conflicts may occur

(d) None of the above

21. Bottom up parsing techniques

(a) simulates a left most derivation

(b) simulates the reverse of a leftmost derivation

(c) simulates a right most derivation

(d) simulates the reverse of a right most derivation

22. An operator precedence grammer

(a) is always LR(1), as LR(1) is the most powerful parsing technique

(b) is always a superset of the LR (1)

(c) is incomparable with the LR(1) grammar

(d) none of the above

23. Can LL(1) grammar ambiguous

(a) yes (b) never

(c) sometimes (d) cannot say

24. Every SLR(1) grammar is

(a) not an LR(1) grammar

(b) may be an LR (1) grammar

(c) sometimes not an LR (1) grammar

(d) an LR (1) grammar

25. Every LL(1) grammar is

(a) an SLR (1) grammar

(b) LR (0) grammar

(c) LR (1) grammar

(d) none of the above

26. $S \rightarrow a\ XX\ b,\ X \rightarrow a \mid b/ \in$ is

(a) LL (1) (b) Not LL(1)

(c) can not say (d) none

27. $S \rightarrow a\ XX\ b.\ X \rightarrow a|b$ is

(a) LR(1) (b) SLR(1)

(c) LALR (1) (d) all

28. The grammar $S \rightarrow a\ S\ a \mid \in$ is

(a) LR (1) (b) not LR (1)

(c) may be LR (1) (d) none

29. A grammar is CLR(1) but not LALR(1), then there must be

(a) SR conflicts (b) RR conflicts

(c) both (d) none of the above

30. Consider the following grammer :

$S \rightarrow S + S$

$S \rightarrow S * S$

$S \rightarrow id$

What we can say about precedence of operator + and * ?

(a) + is having higher precedence

(b) * is having higher precedence over +

(c) Both using same precendence

(d) None of these

31. After removing left recursion from the following grammer

$S \rightarrow SA \mid SB \mid a \mid b \mid c$

The resulting grammer will be

(a) $S \rightarrow aS' \mid bS'$ (b) $S \rightarrow aS' \mid bS' \mid cS'$

$S \rightarrow AS' \mid BS'$ $S \rightarrow S'A \mid S'B$

(c) $S \rightarrow aS' \mid bS' \mid cS'$ (d) None of these

$S' \rightarrow AS' \mid BS' \mid \in$

32. Consider the following statements :

S1 : Using bottom up parser we can construct efficient parser more easily by hand using bottom-up method.

S2 : Top down parsing can handle large class of grammers end translation schema.

Which of the following statements is/are true ?

(a) Both statements are true

(b) S1 is true and S2 is false

(c) S2 is true and S1 is false

(d) Both statements are false

33. If there is conflict between two right sides in grammar for any look ahead symbol, then parser for which one can't this grammar is,

(a) Bottom up parser

(b) Predictive parser

(c) Both (a) and (b)

(d) None of the above

34. Which of the following statements is true?

(a) SLR parser is more powerful than LALR

(b) LALR parser is more powerful than Canonical LR parser

(c) Canonical LR parser is more powerful than LALR parser

(d) The parsers SLR, Canonical LR, and LALR have the same power

35. Which of the following statements is FALSE?

(a) An unambiguous grammar has same leftmost and rightmost derivation

(b) An LL(1) parser is a top-down parser

(c) LALR is more powerful than SLR

(d) An ambiguous grammar can never be LR(k) for any k

36. Assume that the SLR parser for a grammar G has n_1 states and the LALR parser for G has n_2 states. The relationship between n_1 and n_2 is

(a) n_1 is necessarily less than n_2

(b) n_1 is necessarily equal to n_2

(c) n_1 is necessarily greater than n_2

(d) None ofthe above

37. Consider the grammar shown below :

$S \rightarrow CC$

$C \rightarrow c\ C \mid d$

The grammar is

(a) LL (1)

(b) SLR (1) but not LL (1)

(c) LALR (1) but not SLR (1)

(d) LR (1) but not LALR (1)

38. The grammar $A \rightarrow AA \mid (A) \mid \varepsilon$ is not suitable for predictive-parsing because the grammar is

(a) ambiguous (b) Left-recursive

(c) right-recursive (d) an operator-grammar

39. Consider the grammar

$S \rightarrow (S) \mid a$

Let the number of states in SLR (1), LR (1) and LARL (1) parsers for the grammar be n_1, n_2 and n_3 respectively. The following relationship holds good

(a) $n_1 < n_2 < n_3$ (b) $n_1 = n_3 < n_2$

(c) $n_1 = n_2 = n_3$ (d) $n_1 \geq n_3 \geq n_2$

Linked Questions for 40 & 41.

Consider the context-free grammar

$E \rightarrow E + E$

$E \rightarrow (E * E)$

$E \rightarrow id$

where E is the starting symbol, the set of terminals is {id, (, + ,), * } and the set of non-terminals is {E}.

40. Which of the following terminal strings has more than one parse tree when parsed according to the above grammar ?

(a) id + id + id + id (b) id + (id * (id * id))

(c) (id* (id * id)) + id (d) ((id * id + id) * id)

41. For the terminal string with more than one parse tree obtained as solution to in above Question, how many parse trees are possible ?

(a) 5 (b) 4

(c) 3 (d) 2

42. Consider the following grammar

$S \rightarrow FR$

$R \rightarrow *S|\varepsilon$

$F \rightarrow id$

In the predictive parser table, M of the grammar the entries M[S, id] and M[R, $] respectively

(a) $\{S \rightarrow FR\}$ and $\{R \rightarrow \varepsilon\}$

(b) $\{S \rightarrow FR\}$ and $\{\ \}$

(c) $\{S \rightarrow FR\}$ and $\{R \rightarrow *S\}$

(d) $\{F \rightarrow id\}$ and $\{R \rightarrow \varepsilon\}$

43. Which one of the following is a top-down parser?

(a) Recursive descent parser

(b) Operator precedence parser

(c) An LR(k) parser

(d) An LALR(k) parser

44. The grammar $S \rightarrow aSa\ |bS|c$ is

(a) LL(l) but not LR(l)

(b) LR(l) but not LL(l)

(c) Both LL(l) and LR(l)

(d) Neither LL(l) nor LR(l)

45. What is the maximum number of reduces moves that can be taken by a bottom up parser for a grammar without epsilon and unit productions (i.e. of type $A \rightarrow \varepsilon$ and $A \rightarrow a$) to parse a string with n tokens ?

(a) n/2 (b) n-1

(c) 2n-1 (d) 2^n

46. Consider the following two sets of LR(1) items of LR(1) grammar.

$X \rightarrow c.X,\ c/d$ $X \rightarrow c.X,\ \$$

$X \rightarrow .cX,\ c/d$ $X \rightarrow .cX,\ \$$

$X \rightarrow .d,\ c/d$ $X \rightarrow .d,\ \$$

Which of the following statement related to merging of the two sets in the corresponding parser is/are FALSE?

1. Cannot be merged since look aheads are different.
2. Can be merged but will result in S-R conflict.
3. Can be merged but will resultin R-R conflict.
4. Cannot be merged since go to on c will lead to two different sets.

(a) 1 only (b) 2 only

(c) 1 and 4 only (d) 1, 2, 3 and 4

47. A canonical set of items is given below

$S \rightarrow L.\ >R$

$Q \rightarrow R.$

On input symbol > the set has

(a) a shift-reduce conflict and a reduce-reduce conflict.

(b) a shift-reduce conflict but not a reduce-reduce conflict.

(c) a reduce-reduce conflict but not a shift-reduce conflict.

(d) neither a shift-reduce nor a reduce-reduce conflict.

48. Consider the grammar defined by the following production rules, with two operators * and +

$S \rightarrow T * P$

$T \rightarrow U \mid T * U$

$P \rightarrow Q + P \mid Q$

$Q \rightarrow Id$

$U \rightarrow Id$

Which one of the following is TRUE?

(a) + is left associative, while * is right associative

(b) + is right associative, while * is left associative

(c) Both + and * are right associative

(d) Both + and * are left associative

49. Which one of the following is TURE at any valid state in shift-reduce parsing ?

(a) Viable prefixes appear only at the bottom of the stack and not inside

(b) Viable prefixes appear only at the top of the stack and not inside

(c) The stack contains only a set of viable prefixes

(d) The stack never contains viable prefixes

50. Among simple LR (SLR), canonical LR, and look-ahead LR (LALR), which of the following pairs identify the method that is very easy to implement and the method that is the most powerful, in that order ?

(a) SLR, LALR

(b) Canonical LR, LALR

(c) SLR, canonical LR

(d) LALR, canonical LR

51. Which of the following statements about parser is/are CORRECT?

I. Canonical LR is more powerful than SLR.

II. SLR is more powerful than LALR.

III. SLR is more powerful than Canonical LR.

(a) I only (b) II only

(c) III only (d) II and III only

52. Which one of the following kinds of derivation is used by LR parsers?

(a) Rightmost

(b) Leftmost in reverse

(c) Rightmost in reverse

(d) Leftmost

○○○

Answer Key

1. *Ans. (a)*
2. *Ans. (b)*
3. *Ans. (a)*
4. *Ans. (a)*
5. *Ans. (a)*
6. *Ans. (b)*
7. *Ans. (a)*
8. *Ans. (a)*
9. *Ans. (b)*
10. *Ans. (d)*
11. *Ans. (b)*
12. *Ans. (c)*
13. *Ans. (a)*
14. *Ans. (a)*
15. *Ans. (a)*
16. *Ans. (b)*
17. *Ans. (c)*
18. *Ans. (c)*
19. *Ans. (b)*
20. *Ans. (b)*
21. *Ans. (d)*
22. *Ans. (c)*
23. *Ans. (d)*
24. *Ans. (d)*
25. *Ans. (c)*
26. *Ans. (b)*
27. *Ans. (d)*
28. *Ans. (c)*
29. *Ans. (b)*
30. *Ans. (c)*
31. *Ans. (c)*
32. *Ans. (a)*
33. *Ans. (b)*
34. *Ans. (c)*

LR parsers in term of power is

CLR > LALR > SLR > LR(O)

35. *Ans. (a)*

(a) An unambiguous grammar can have different leftmost and rightmost derivation. However, an unambiguous grammar has only one derivation tree. So option (a) is false.

(b) LL(1) is a top-down parser.

(c) LALR is more powerful than SLR.

(d) For any parser, grammar should be unambiguous.

36. *Ans. (b)*

SLR parser has n_1 states for a grammar G LALR parser has n_2 states for a grammar G. The states of SLR and LALR parser are the states of corresponding states in a deterministic finite automata which recognizes the viable prefixes and both deterministic finite automata contains the equal number of states so $n_1 = n_2$.

37. *Ans. (a)*

Consider the grammar

$S \rightarrow CC$

$C \rightarrow cC \mid d$

The above grammar is LL(1)

[it is unambiguous, not left recursive and left factored].

It is also LR(0) grammar, hence it is SLR(1), LALR(1) and CLR(1).

Therefore option (a) matches correctly.

38. *Ans. (a)*

The string ε has more than one parse tree for the given grammar.

Given grammar is Ambiguous, that is the reason that it is not suitable for predictive parsing

39. ***Ans. (b)***

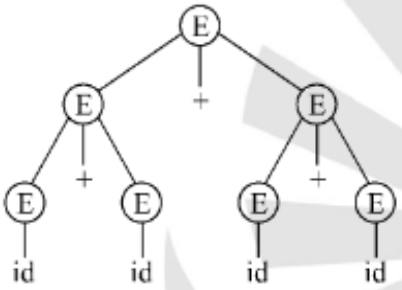
$S \rightarrow (S) \mid a$

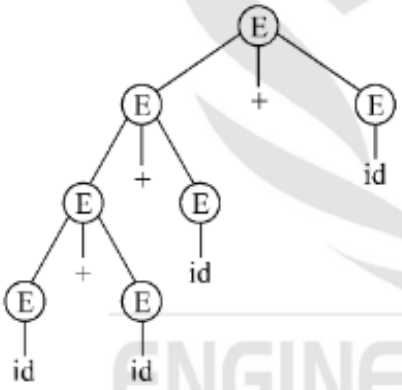
Parser	Number of states
SLR(1)	n_1
LR(1)	n_2
LALR(1)	n_3

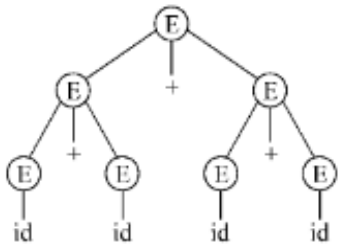
The number of states of deterministic finite automate in SLR(1) and LALR(1) parsers are equal, so $n_1 = n_3$. The number of states of deterministic finite automata in LR(1) is greater than number of states of deterministic finite automata of SLR(1) and LALR(1)

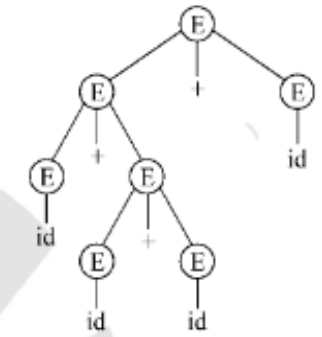
So $n_1 = n_3 < n_2$.

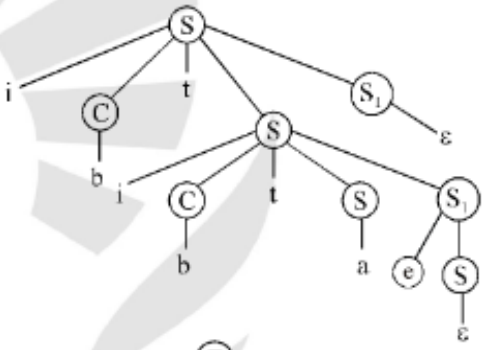
40. ***Ans. (a)***

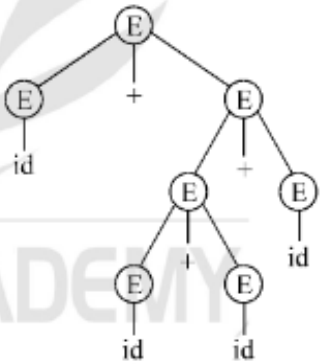






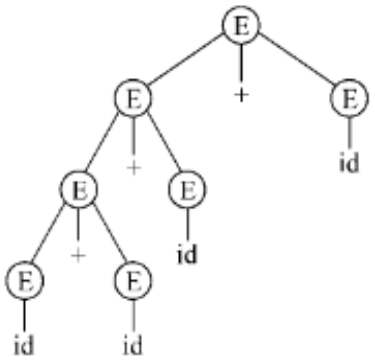


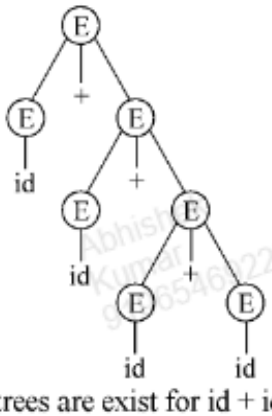




∴ id + id + id + id has more than one parse tree.

41. ***Ans. (a)***





∴ 5 parse trees are exist for id + id + id + id

42. ***Ans. (a)***

Construct Predictive paraser table as follows

Non-terminal	id	*	$
S	S → FR		
R		R → *S	R → ε
F	F → id		

So M{S, id] = {S → FR}

and M[R, $] = {R → ε}

43. ***Ans. (a)***

Predictive parser and recursive descent parser are example of top-down parser. Whereas LR(k) and LALR(k) parser are bottom up parser.

44. ***Ans. (c)***

S → aSa | bS | c

The above grammar is LL(1) because,

First [aSa] ∩ first [bS] = (a) ∩ (b) = ϕ

&&

First [bs] ∩ first [c] = (b) ∩ (c) = ϕ

&&

First [c] ∩ first [aSa] = (c) ∩ (a) = ϕ

As the above grammar is LL(1), also LR(1) because LL(1) grammar is always LR(1) grammar.

45. ***Ans. (c)***

Unit production : A → B

[**Note :** There is typo error in question for unit production. Instead of A → B, it was given as A → a]

Null production : A → ε

Ex. S → AB

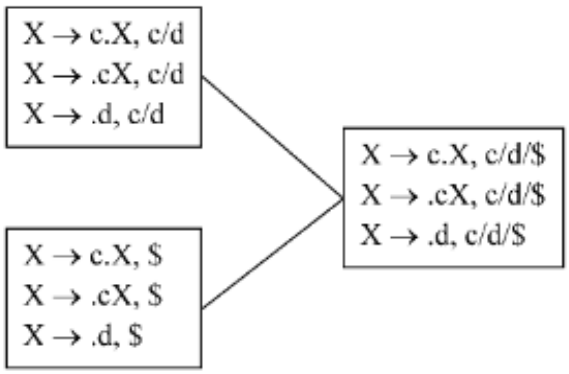
A → a

B → b

[2 length string] ab ⇒ Ab ⇒ AB ⇒ S, it requires three reductions.

∴ For n length string (2n – 1) reductions required.

46. ***Ans. (d)***



1. Cannot be merged since look-a-heads are different: FALSE, because merging does not depend on look ahead.
2. Can be merged but will results in S-R conflict: FALSE the two states are not containing reduce item, so after merging, merged state can not contain any S-R conflict.
3. Can be merged but will result in R-R conflict : FALSE, no RR conflict in merged state.
4. Cannot be merged since goto on c will lead to two different sets : FALSE, merging of states does not depends on goto.

47. ***Ans. (d)***

S → L. > R

Q → R .

In above diagram, we see at I_2 state it has only one reduced item and there is no other item in the state. Therefore for input symbol '>', there is no conflict.

48. ***Ans. (b)***

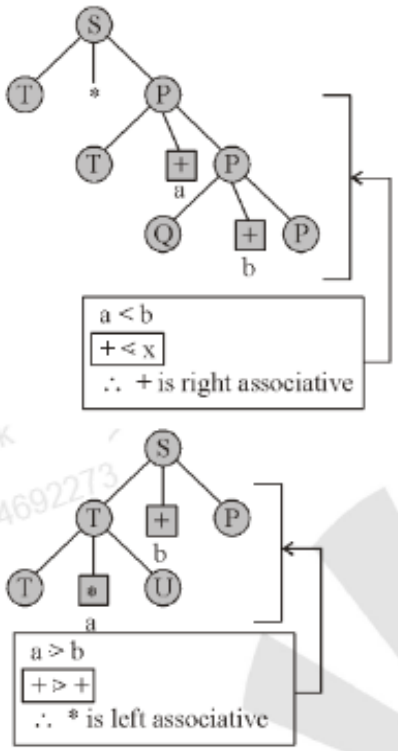
S → T * P

T → U| T * U

P → Q + P|Q

Q → Id

U → Id



49. ***Ans. (c)***

Stack contains only a set of viable prefixes.

50. ***Ans. (c)***

Among SLR, CLR and LALR parsers

(i) **SLR** parser is simple and very easy to implement compared to other parsers.

(ii) **CLR** parser is most powerful than other parsers because it can accepts more languages than other.

51. ***Ans. (a)***

Canonical LR is the most powerful

Parser among all the LR(k) parser.

52. ***Ans. (c)***

LR parser is bottom up parser.

Every bottom up parser uses "Reverse of RMD"

So, RMD in reverse is correct.

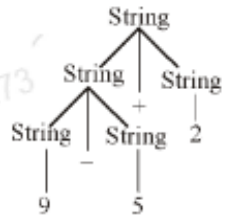
❍❍❍

CHAPTER 3

SYNTAX DIRECTED TRANSLATION

OBJECTIVE QUESTIONS

1. Consider the following parse tree :



Which of the following statement is true ?

(a) Both + and – are having equal precedence

(b) + is having higher precendence over –

(c) – is having higher precedence over +

2. A parse tree showing attribute value at each node is called

(a) Concrete tree

(b) Syntex tree

(c) Annotated parse tree

(d) Both (a) and (b)

3. A context free grammar in which program fragments, called, semantic action are embedded within rightside of the production is called,

(a) Syntex directed definition

(b) Translation schema

(c) Annotated parse tree

(d) None of these

4. Parsing is a process of determining

(a) String of nonterminals that can be generated by grammar

(b) String of terminals generated by grammar

(c) String of tokens that can be generated by the grammar

(d) None of these

5. Activation of, procedures can be implemented by runtime storage. The part of runtime storage that is responsible for this is

(a) Data object (b) Target code

(c) Stack pointer (d) Control stack

6. The optional access link in the activation record is used for

(a) Pointing to the activation record of the caller

(b) Referring the non local data held in other activation records

(c) Temporary values, such as those arising in the evaluation of expressions

(d) None of these

7. Heap allocation is required for languages that support

(a) Recursion

(b) dynamic scope rules

(c) dynamic data structures

(d) none

8. Synthesized attribute can be simulated by

(a) LL (1) grammar

(b) LR (1) grammar

(c) Ambiguous grammar

(d) none

9. A syntax directed definition specifies

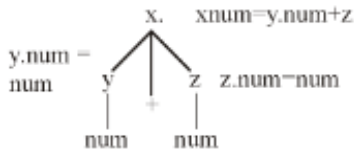
(a) Translation of construct in terms of attributes associated with its syntactic component

(b) Translation of construct in terms of memory associated with its syntactic component

(c) Translation of construct in terms of execution time associated with its syntactic component

(d) None of these

10. Consider the following annotated parse tree :



Which of the following is true for the given annotated tree ?

(a) There is specific order for evaluation of attribute on parse tree

(b) Any evaluation order that computes an attribute a after all othere attribute that a depends on its acceptable

(c) Both (a) and (b)

(d) None of these

11. The grammar

$A \rightarrow B\uparrow \mid A * B \mid B$

$B \rightarrow B - B \mid C / B \mid C$

$C \rightarrow i$

(a) reflects that $\uparrow$ has high precedence than -

(b) reflects that - has high precedence than

(c) reflects that - has high precedence than $\uparrow$

(d) none of the above

12. The following SDT while deriving a string 'abadfefe' produces the output

$A \rightarrow aBCe$	{printf("1");}
$B \rightarrow bA$	{printf("2");}
/d	{printf("3");}
$C \rightarrow cB$	{printf("4");}
/f	{printf("5");}

(a) 12345 (b) 54321

(c) 351521 (d) 351251

13. A linker is given object modules for a set of programs that were compiled separately. What information need to be included in an object module?

(a) Object code

(b) Relocation bits

(c) Names and locations of all external symbols defined in the object module

(d) Absolute addresses of internal symbols

14. In a bottom-up evaluation of a syntax directed definition, inherited attributes can

(a) always be evaluated

(b) be evaluated only if the definition is L-attributed

(c) be evaluated only if the definition has synthesized attributes

(d) never be evaluated

15. Consider the translation scheme shown below :

$S \rightarrow TR$

$R \rightarrow + T$ {print ('+');} $R \mid \varepsilon$

$T \rightarrow$ num {print (num.val);}

Here num is a token that represents an integer and num.val represents the corresponding integer value. For an input string '9 + 5 + 2', this translation scheme will print

(a) 9 + 5 + 2 (b) 9 5 + 2 +

(c) 9 5 2 + + (d) + + 9 5 2

16. Consider the grammar with the following translation rules and E as the start symbol.

$E \rightarrow E_1 \# T$	{E.value = E_1.value * T.value}
$\mid T$	{E.value = T.value}
$T \rightarrow T_1 \& F$	{T.value = T_1.value + F.value}
$\mid F$	{T. value = F.value}
$F \rightarrow$ num	{F.value = num.value}

Compute E. value for the root of the parse tree for the expression: 2 # 3 & 5 # 6 & 4.

(a) 200 (b) 180

(c) 160 (d) 40

17. Consider the grammar $E \rightarrow E + n \mid E \times n \mid n$

For a sentence $n + n \times n$, the handles in the right-sentential form of the reduction are

(a) n, E + n and E + n × n

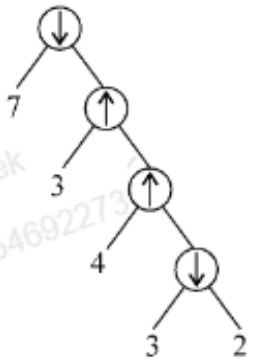
(b) n, E + n and E + E × n

(c) n, n + n and n + n × n

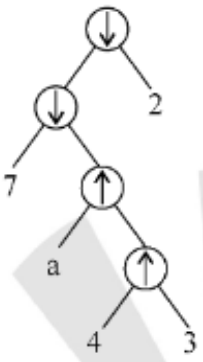
(d) n, E + n and E × n

18. Consider two binary operators '↑' and '↓' with the precedence of operator ↓ being lower than that of the operator ↑. Operator ↑ is right associative while operator ↓ is left associative. Which one of the following represents the parse tree for expression (7↓3↑4↑3↓2)

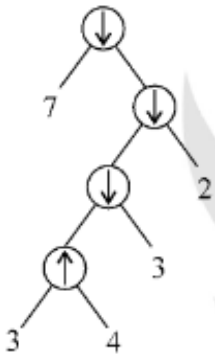
(a)



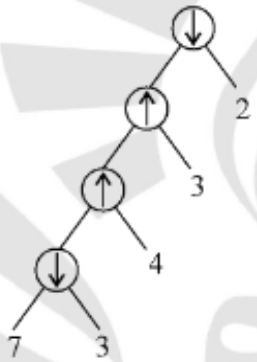
(b)



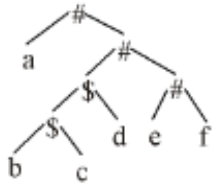
(c)



(d)



19. Consider the following parse tree for the expression $a \neq b\$c\$d \neq e \neq f$, involving two binary operators \$ and $\neq$.



Which one of the following is correct for the given parse tree?

(a) \$ has higher precedence and is left associative; $\neq$ is right associative

(b) $\neq$ has higher precedence and is left associative; \$ is right associative

(c) \$ has higher precedence and is left associative; $\neq$ is left associative

(d) $\neq$ has higher precedence and is right associative; \$ is left associative

❍❍❍

Answer Key

1. *Ans. (c)*
2. *Ans. (c)*
3. *Ans. (b)*
4. *Ans. (c)*
5. *Ans. (b)*
6. *Ans. (b)*
7. *Ans. (c)*
8. *Ans. (d)*
9. *Ans. (a)*
10. *Ans. (b)*
11. *Ans. (c)*
12. *Ans. (d)*
13. *Ans. (d)*

A linker is a computer program that takes one or more object files generated by a compiler and combines them into a single executable program. The linker also takes care of arranging the objects in a program's address space. Therefore absolute addresses of internal symbols need to be included in an object module.

14. *Ans. (c)*

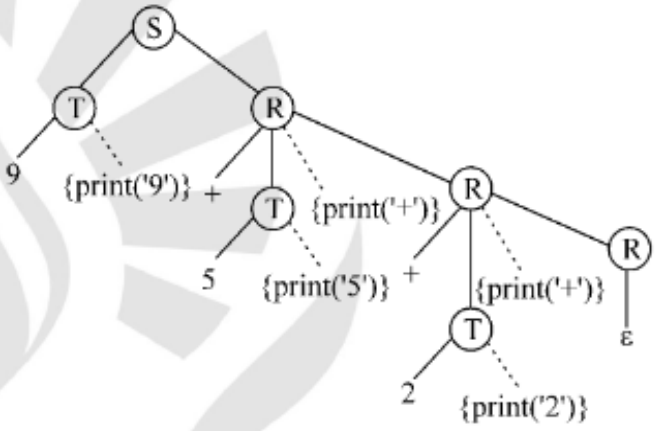
Every S(Synthesized) - attributed definition is L-attributed. For implementing inherited attributed during bottom-up parsing, extends to some, but not LR grammars. Consider the following example

Production	Semantic Rule
$S \rightarrow L$	L. count : = 0
$L \rightarrow L_1$ 1	L. count : = L. count : + 1
$L \rightarrow E$	print (L.count)

In the example above the non-terminal L in $L \rightarrow E$ inherits the count of the number of 1's generated by S. Since the production $L \rightarrow E$ is the first that a bottom-up parser would reduce by, the translator at the time can't know the number of 1's in the input. So in a bottom-up evaluation of a syntax directed definition, inherits attributes can't be evaluated if the definition is L-attributed in the given example. So we can say that L-attributed definition is based on simple LR(1) grammar, but it can't be implemented always but inherit attributes can be evaluated only if the definition has synthesized attributes.

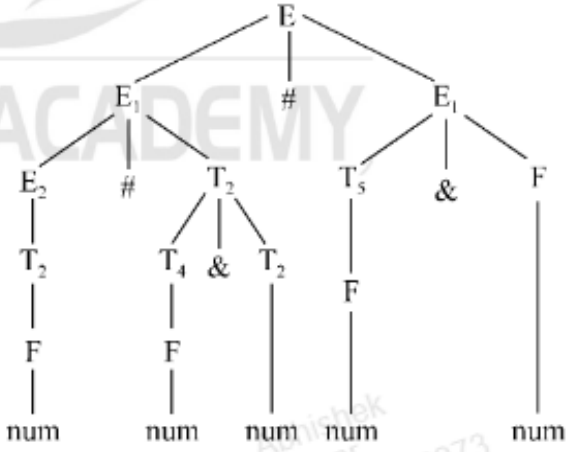
15. *Ans. (b)*

For the input '9 + 5 + 2' the translation scheme is 95 + 2 + shown below:

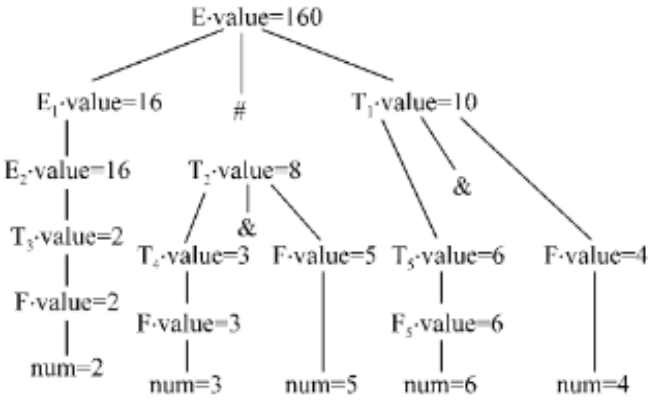


16. *Ans. (c)*

First we have to construct the parse tree.



Then we construct the annonated parse tree or parse tree with value at the leaf node.



17. ***Ans. (d)***

$E \rightarrow E \mid n \mid E \times n \mid n$

Input String $n + n \times n$

$\Rightarrow$ $n + n \times n$

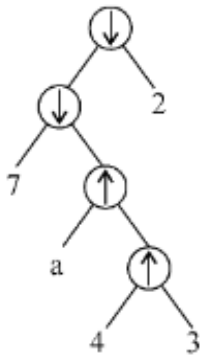
$\Rightarrow$ $E + n \times n$ reduction $E \rightarrow n$

$\Rightarrow$ $E \times n$ reduction $E \rightarrow E + n$

$\Rightarrow$ E reduction $E \rightarrow E \times n$

So the reductions are n, E + n, E × n

18. ***Ans. (b)***



19. ***Ans. (a)***

Since '$' is at lower level in parse tree,

$\therefore$ it has high precedence

also (b $ c) $ d is evaluated

$\therefore$ $ is left associative, # is right associative

❍❍❍

CHAPTER 4

INTERMEDIATE CODE GENERATION

OBJECTIVE QUESTIONS

1. All data objects in FORTRAN language can't be allocated
   (a) dynamically (b) statically
   (c) globally (d) Both (a) and (b)

2. When storage can be de-allocated, the reference to storage that has been allocated is called
   (a) Bug reference (b) Datum reference
   (c) Dangling reference (d) None of these

3. Where does the compiler keep, track of all the scope and binding information about identifiers ?
   (a) Heap (b) Stack
   (c) Symbol table (d) None of these

4. The postfix form of a + b – c * c/d is
   (a) abc – + cd / * (b) ab + c – * c/d/
   (c) ab + c – cd /* (d) ab + cc * d/-

5. Which of the following technique can be used as intermediate code generation ?
   (a) Postfix notation and three address codes
   (b) Quadruples
   (c) Both (a) and (b)
   (d) None of these

6. 'Three address code' technique for intermediate code generation shows that each statement usually contains three addresses. Three addresses are as follows,
   (a) Two addresses for operands, one for operator
   (b) One for all operators, one for all operands and one for result
   (c) One for result, two for operands
   (d) None of these

7. Which of the following statements is true ?
   (a) For optimizing compiler, moving a statement that defines a temporary value requires us to change all references to that statements. It is an overhead for triples notation
   (b) For optimizing compiler, triples notation has important benefit where statements are often moved around as it incurs no movement or change
   (c) Quadruples have some disadvantages over triples notation for an optimizing compiler
   (d) Both (a) and (b)

8. Inputs to the code generator phase of compiler design are,
   (a) Code optimizer, symbol table
   (b) Intermediate code & optimized code
   (c) Intermediate code representation and symbol table
   (d) None of these

9. The output of the code generator is a target program that includes
   (a) Absolute machine language
   (b) Relocatable machine language
   (c) Assembly language
   (d) All of these

10. Replacement of an expensive operation by a cheaper one is termed as
    (a) reduction in strength
    (b) code motion
    (c) loop-in variant computation
    (d) none of these

11. A basic block can be analyzed by

(a) Flow graph

(b) A graph with cycles

(c) DAG

(d) None of these

12. Generation of intermediate code. based on abstract machine model is useful in compilers because

(a) it enhances the probability of the front end of the compiler

(b) syntax directed translations can be written for intermediate code generation

(c) it is possible to generate code for real machines directly from high level language programs

(d) it makes implementation of lexical analysis and syntax analysis easier.

13. What is the formal name of the method that performs optimizing loop execution speed, usually at the cost of code size ?

(a) Reverse Polish Notation

(b) Loop unwinding

(c) Strength Reduction

(d) None of these

14. Multiplication of a positive integer by a power of two can be replaced by left shift, which executes faster on most machines. This is an example of

(a) Loop unwinding

(b) Peephole optimization

(c) Strength Reduction

(d) None of these

15. Which technique generates more optimal codes ?

(a) Starting from a linear sequence of three-address statement

(b) Starting from quadruples

(c) From rearrange of the dag

(d) None of these

16. Code generation can be done by

(a) DAG tree

(b) Labeled tree

(c) Both (a) and (b)

(d) None of these

17. An optimizing compiler

(a) is optimized to occupy less space

(b) optimizes the code

(c) is optimized to take less time for execution

(d) none of these

18. Peephole optimization is a form of

(a) Constant folding

(b) Loop optimization

(c) Local optimization

(d) Data flow analysis

19. The graph showing interdependencies of attributes is

(a) DAG

(b) flow graph

(c) dependency graph

(d) none

20. What will be the optimized code when the expression p = q* – r + q* – r is represented in DAG specification?

(a)
```
t1 = - r
t2 = q * t1
t3 = q * t1
t4 = t2 + t3
a = t4
```

(b)
```
t1 = -r
t2 = q
t3 = t1 * t2
t4 = t3 + t3
a = t4
```

(c)
```
t1 = -r
t2 = q * t1
t3 = t2 + t2
p = t3
```

(d) none of the above

21. Match the correct optimization technique to the given "initial-final" coding

Initial	Final
(A) A = B + C D = B + C	A = B + C D = A
(B) B = 2.0 * A	B = A + A
(C) DO FOR I = 1 to 10 A[1] = B + C END	DO FOR I = 1 to 10 T = B + C A[1] = T END
(D) I = 10 J = 2 * 1	I = 10 J = 20

Optimization technique
1. constant propagation
2. code motion
3. common sub-expression
4. Strength reduction

Codes:	A	B	C	D
(a)	4	3	2	1
(b)	3	4	2	1
(c)	4	3	1	2
(d)	1	2	3	4

22. Match the following

P. Intermediate code	1. X = A + B + C
Q. Quadruple	2. (+, A, B, C)
R. Triple	3. (+, A, B)
S. Indirect Triple	4. Add R1, R2
	5. AB+
	6. (3)

Codes:	P	Q	R	S
(a)	1	5	3	4
(b)	6	4	3	5
(c)	1	3	6	2
(d)	5	2	3	6

23. Consider the following C code segment.

```
for (i = 0; i < n; i ++)
{
    for (j = 0; j < n; j ++)
    {
        if (1 % 2)
        {   x + = (4 * j + 5 * i);
            y + = (7 + 4 * j);
        }
    }
}
```

Which one of the following is false ?

(a) The code contains loop invariant computation

(b) There is scope of common sub-expression elimination in this code

(c) There is scope strength reduction in this code

(d) There is scope of dead code elimination in this code

24. Which languages necessarily need heap allocation in the runtime environment ?

(a) Those that support recursion

(b) Those that use dynamic scoping

(c) Those that allow dynamic data structure

(d) Those that use global variables

Common Data for Questions 25 and 26

The following code segment is executed on a processor which allows only register opeands in its instructions. Each instruction can have almost two source operands and one destinations operand. Assume that all variables are dead after this code segment.

```
c = a + b;
d = c * a;
e = c + a;
x = c * c;
if (x > a) }
y =a * a;
}
else {
    d = d* d;
    e = e * e;
}
```

25. Suppose the instruction set architecture of the processor has only two registers. The only allowed complier optimization is code motion, which moves statements from one place to another while preserving correctness. What is the minimum number of spills to memory in the complied code?

(a) 0 (b) 1

(c) 2 (d) 3

26. What is the minimum number of registers needed in the instruction set architecture of the processor to compile this code segment without any spill to memory ? do not apply any optimization other than optimizing register allocation ?

(a) 3 (b) 4

(c) 5 (d) 6

27. Which one of the following is NOT performed during compilation ?

(a) Dynamic memory allocation

(b) Type checking

(c) Symbol table management

(d) Inline expansion

28. Which of the following statements are CORRECT?

(1) Static allocation of all data areas by a compiler makes it impossible to implement recursion.

(2) Automatic garbage collection is essential to implement recursion.

(3) Dynamic allocation of activation record is essential to implement recursion.

(4) Both heap and stack are essential to implement recursion.

(a) 1 and 2 only (b) 2 and 3 only

(c) 3 and 4 only (d) 1 and 3 only

29. Consider the basic block given below :

```
a = b + c
c = a + d
d = b + c
e = d – b
a = e + b
```

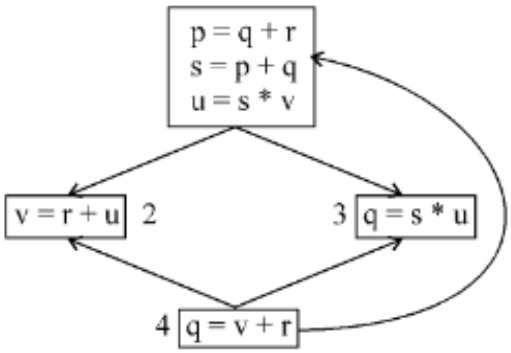
The minimum number of nodes and edges present in the DAG representation of the above basic block respectively are

(a) 6 and 6 (b) 8 and 10

(c) 9 and 12 (d) 4 and 4

30. A variable x is said to be live at a statement S_i in a program if the following three conditions hold simultaneously:

1. There exists a statement S_j that uses x
2. There is a path from S_i to S_j in the flow graph corresponding to the program
3. The path has no intervening assignment to x including at S_i and S_j



The variables which are live both at the statement in basic block 2 and at the statement in basic block 3 of the above control flow graph are

(a) p, s, u (b) r, s, u

(c) r, u (d) q, v

31. Consider the intermediate code given below :

1. i = 1
2. j = 1
3. t1 = 5 * i
4. t2 = t1 + j
5. t3 = 4 * t2
6. t4 = t3
7. a[t4] = –1
8. j = j + 1
9. if j < = 5 goto (3)
10. i = j + 1
11. if i < 5 goto (2)

The number of nodes and edges in the control-flow-graph constructed for the above code, respectively, are

(a) 5 and 7 (b) 6 and 7

(c) 5 and 5 (d) 7 and 8

32. A student wrote two context-free grammars G1 and G2 for generating a single C-like array declaration. The dimension of the array is at least one. For example,

int *a* [10] [13].

The grammars use D as the start symbol, and use six terminal symbols int; id [ ] num.

Grammar G1	Grammar G2
D → int L;	D → int L;
L → id [E	L → idE
E → num]	E → E [num]
E → num] [E	[E → [num]

Which of the grammars correctly generate the declaration mentioned above ?

(a) Both G1 and G2 (b) Only G1

(c) Only G2 (d) Neither G1 nor G2

33. Consider the following intermediate program in three address code

p = a – b

p = a * c

p = u * v

q = p + q

Which one of the following corresponds to a static single assignment form of the above code?

(a) $p_1 = a - b$
$p_1 = p_1 * c$
$p_1 = u * v$
$q_1 = p_1 + q_1$

(b) $p_3 = a - b$
$q_4 = p_3 * c$
$p_4 = u * v$
$q_5 = p_4 + q_4$

(c) $p_1 = a - b$
$q_1 = p_2 * c$
$p_3 = u * v$
$q_2 = p_4 + q_3$

(d) $p_1 = a - b$
$q_1 = p * c$
$p_2 = u * v$
$q_2 = p + q$

❑❑❑

Answer Key

1. *Ans. (b)*
2. *Ans. (c)*
3. *Ans. (c)*
4. *Ans. (d)*
5. *Ans. (a)*
6. *Ans. (c)*
7. *Ans. (a)*
8. *Ans. (c)*
9. *Ans. (d)*
10. *Ans. (c)*
11. *Ans. (c)*
12. *Ans. (d)*
13. *Ans. (b)*
14. *Ans. (c)*
15. *Ans. (c)*
16. *Ans. (c)*
17. *Ans. (d)*
18. *Ans. (c)*
19. *Ans. (c)*
20. *Ans. (c)*
21. *Ans. (b)*
22. *Ans. (a)*
23. *Ans. (d)*

(a) i%2 is inner loop invariant, it can be moved before inner loop.

(b) 4*j is common sub-expression appeared in two statements.

(c) 4*j can be reduced to j<<2 by strength reduction.

(d) There is no dead code in given code segment. So there is no scope of dead code elimination in this code.

Hence only option (d) is FLASE.

24. *Ans. (c)*

Runtime environment means we deal with dynamic memory allocation and heap is a dynamic data structure.

So it is clear that those languages that allow dynamic data structure necessarily need heap allocation in the runtime environment.

25. *Ans. (b)*

```
c = a + b;
x = c * c;                    ...(i)
if (x > a) {
   y = a * a;
}
else {
   d = c * a;                 ...(ii)
   e = c + a;
   d = d * d;
   e = e * e;
}
```

(i) is to store c*c, it needs one memory spill.

(ii) is uses previous same (i) memory spill to store c*a.

Number of memory spills are used in above program is one. With the use of one memory cell and two registers the above optimized code can be executed.

26. *Ans. (b)*

$R_1 \leftarrow R_0 + R_1$	c = a + b
$R_2 \leftarrow R_0 * R_1$	d = c * a
$R_3 \leftarrow R_0 + R_0$	e = c + a
$R_1 \leftarrow R_1 * R_1$	x = c * c
if $(R_1 > R_0)$	if (x > a)
{	
$R_1 \leftarrow R_0 * R_0$	y = a * a
}	
else	
{	
$R_2 \leftarrow R_2 * R_1$	d = d * d
$R_3 \leftarrow R_3 * R_3$	e = e * a
}	

4 registers are required.

∴ Option (b) is correct.

27. ***Ans. (a)***

Dynamic memory allocation performed during runtime whereas type checking, symbol table management and inline expansion is performed during compilation.

28. ***Ans. (d)***

Recursion can not be implemented using static allocation.

Recursion can be implemented using dynamic allocation.

29. ***Ans. (a)***

$a = b + c \Rightarrow a = e + b$

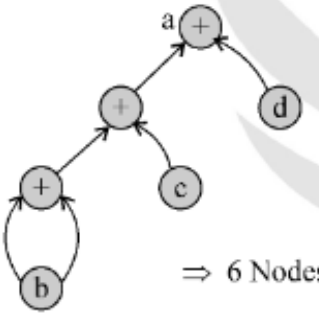
$c = a + d \Rightarrow a = d - b - b = d$

$d = b + c \Rightarrow a = b + c$

$e = d - b \Rightarrow a = b + a + d$

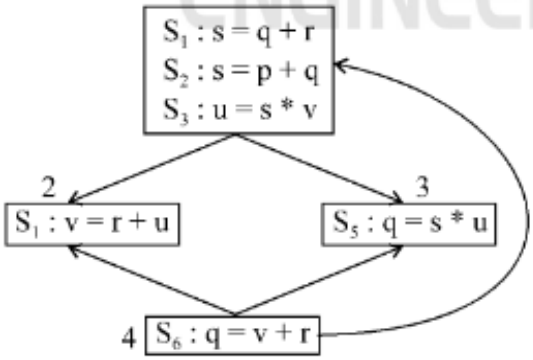
$a = e + b \Rightarrow a = b + b + c + d$

$\therefore$ b + b + c + d is expression



$\Rightarrow$ 6 Nodes and 6 edges

30. ***Ans. (*)***



(i) P is not live both a Block 2 and Block 3.

There is a path from (S_4 and S_6) to S_2, but S_1 has an assignment to p.

(ii) q is not live both at Block 2 and Block 3.

There is a path from (S_4 & S_5) to S_1, but S_6 has an assignment to q in between the path.

(iii) r is live both a Block 2 and Block 3. There is a path from (S_4 & S_5) to S_6, S_6 uses r and no assignment to r in this path.

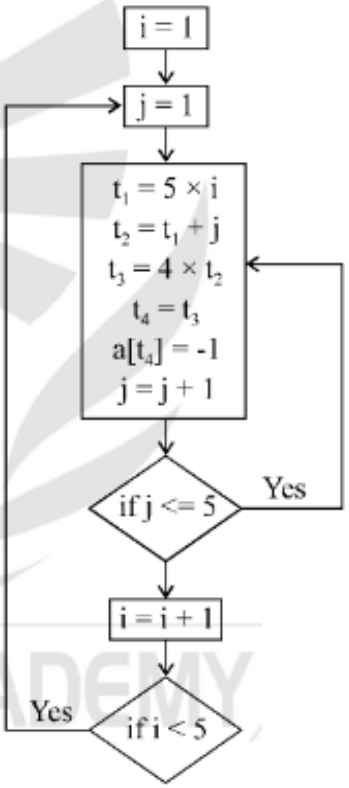
(iv) u is not live both a Block 2 and Block 3.

There is a path from S_4 to S_5 and S_5 to S_4 where both S_4 and S_5 uses u but S_3 has an assignment to u in this path.

(v) Similarly s and v are not live both a Block 2 and Block 3.

$\therefore$ Only r is live both at basic Block 2 and basic Block 3.

31. ***Ans. (b)***



Number of nodes = 6

Number of edges = 7

32. ***Ans. (a)***

Both G1 and G2 can generate int a [10][3];

as follows :

G1 : D → int L; → int id[E ;

→ int id[num] [E;

→ int id[num] [num];

G2 : D → int L ; → int id E;
→ int id E [num];
→ int id [num] [num];

33. ***Ans. (b)***

Correct code is

$P_3 = a - b$

$q_4 = p_3 \times c$

$q_4 = u \times v$

$q_5 = q_4 + p_4$

❑❑❑