GATE 2020
PSUs 2020

MADE EASY Publications

Revised & Updated Edition

# POSTAL STUDY PACKAGE

## COMPUTER SCIENCE & IT

### Compiler Design

Objective Practice Sets

# POSTAL Study Package 2020

## Computer Science & IT

Objective Practice Sets

### Compiler Design

*Contents*




MADE EASY Publications

# 1 CHAPTER
# Introduction to Compiler

1. Storage mapping is done by
   (a) operating system (b) compiler
   (c) linker (d) loader

2. Assembly language
   (a) is usually the primary user interface
   (b) requires fixed format commands
   (c) is a mneomonic form of machine language
   (d) is quite different from the SCL interpreter

3. Which translator program converts assembly language program to object program?
   (a) Assembler (b) Compiler
   (c) Microprocessor (d) Linker

4. Which one is a phase of a compilation process?
   (a) Lexical analysis (b) Code generation
   (c) Both (a) and (b) (d) None of these

5. Semantic errors can be detected
   (a) at compile time only
   (b) at run time only
   (c) both at compile and run time
   (d) none of these

6. Undeclared name is _____ error.
   (a) syntax (b) lexical
   (c) semantic (d) not an error

7. A simple two-pass assembler does not do which of the following in the first pass?
   (a) It allocates space for the literals.
   (b) It computes the total length of the program.
   (c) It builds the symbol table for the symbols and their values.
   (d) It generates code for all the load and stores register instructions.

8. A simple two-pass assembler does which of the following in the first pass?
   1. It allocates space for the literals.
   2. It computes the total length of the program.
   3. It builds the symbol table for the symbols and their values.
   4. It generates code for all the load and stores register instruction.
   (a) 1 only (b) 1 and 2
   (c) 1, 2 and 3 (d) 4 only

9. A loader is
   (a) a program that place programs into memory and prepares them for execution
   (b) a program that automates the translations of assembly language into machine language
   (c) a program that accepts a program written in a high level language and produces an object program
   (d) a program that appears to execute a source program as if it were machine language

10. A programming language is to be designed to run on a machine that does not have a big memory. The language should
    (a) prefer a 2 pass compiler to a 1 pass compiler
    (b) prefer an interpreter to a compiler
    (c) not support recursion
    (d) all of the above

11. Cross-compiler is a compiler
    (a) that generates object code for its host machine
    (b) which is written in a language that is the same as the source language
    (c) which is written in a language that is different from the source language
    (d) that runs on one machine and produces object code for another machine

12. Which of following is used for grouping of characters into tokens (in a compiler)
(a) parser (b) code optimizer
(c) code generator (d) scanner

13. Consider the following *C* fragment
fro (int $x = 0; x <= n; x++$)
Which type of error detected by the *C* compiler for the above code?
(a) lexical error (b) syntactic error
(c) semantic error (d) logical error

14. For which of the following reasons, a compiler is preferable to an interpreter?
(a) It can generate stand-alone programs that often take less time for execution.
(b) It is much helpful in the initial stages of program development.
(c) Debugging can be faster and easier.
(d) If one changes a statement, only that statement needs recompilation.

15. An optimizing compiler
(a) is optimized to occupy less space
(b) is optimized to take less time for execution
(c) optimized the code
(d) none of the above

16. Match the following groups:

**List-I**
A. Lexical analyzer
B. Syntax analyzer
C. Type checking
D. Intermediate code generation

**List-II**
1. Checks the structure of the program.
2. Analysis of entire program by reading each character.
3. High level language is translated to simple machine independent language.
4. Checks the consistency requirements in a context of the program.

**Codes:**

| | A | B | C | D |
|---|---|---|---|---|
| (a) | 1 | 2 | 4 | 3 |
| (b) | 2 | 1 | 4 | 3 |
| (c) | 2 | 4 | 3 | 1 |
| (d) | 1 | 4 | 3 | 2 |

■■■■

## Answers Introduction to Compiler

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

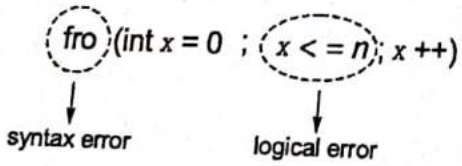
## Explanations Introduction to Compiler

**6. (c)**
It is a semantic error (run-time error).

**8. (c)**
Simple two-pass assembler
1. Allocates space for the literals.
2. Computes the total length of program (syntax analysis)
3. Builds the symbol table for the symbols and their values

**10. (d)**
Since the programming language is to be designed to run on a machine that does not have a big memory. Therefore pass-2 (multiphase), which uses less space is preferred also instead of using a compiler we use an interpreter, which scans the program line by line hence uses less memory at compiler time for each instance. Also it does not support recursion.

**13. (b)**
Although the statement




Has both syntax and logical error
But, it will result in syntax error.
Hence, (b) is correct option.

**14. (a)**
A compiler compiles the program in one time while the interpreter compiles it taking line by line. Hence, security checking is more in interpreter while compiler is fast in compilation. Hence, it can generate stand-alone programs that often take less time for execution.

**16. (b)**
**Lexical analyzer:** Reads every character of the program to identify the tokens.

**Syntax analyzer:** Analyzes the syntax or structure of the program.

**Type checking:** It determines violation of consistency requirements.

**ICG:** Translates the program into intermediate language.

■■■■

# 2 CHAPTER
# Lexical Analysis

1. In some programming languages, an identifier is permitted to be a letter followed by any number of letters or digits. If *L* and *D* denotes the sets of letters and digits respectively, which expression defines an identifier?
   (a) $(LUD)^+$ (b) $L(LUD)^*$
   (c) $(L.D)^*$ (d) $L.(L.D)^*$

2. Consider the following statements:
   1. Lexical analysis removes white spaces and not the comments.
   2. fro($i = 0$) generates lexical error in *C*
   3. Lexeme is the string and not the pattern.
   4. *fi* ($a > b$) generates no lexical error in *C*.

   Identify the false statements:
   (a) Only 1 and 4 (b) Only 1 and 2
   (c) Only 2 (d) Only 4

3. The task of the lexical analysis phase is
   (a) to parse the source program into the basic elements or tokens of the language
   (b) to build a literal table and an identifier table
   (c) to build a uniform symbol table
   (d) all of the above

4. The output of lexical analyser is
   (a) a set of regular expressions
   (b) syntax tree
   (c) set of tokens
   (d) strings of characters

5. How many tokens are contained in the following FORRTAN statement:
   IF(NUMB EQ MAX) GOTO 500
   (a) 8 (b) 10
   (c) 22 (d) 24

6. The number of tokens in the following *C* statement
   printf ("$i$ = %$d$, & $i$ = % $x$", $i$, & $i$);
   (a) 3 (b) 26
   (c) 10 (d) 21

7. Consider the following C-program:
```
int strange (int x)
{
  if (x < = 0) return 0;
  if (x%2! = 0) return x – 1;
  return 1 + strange (x – 1);
}
```
   Find out the number of tokens?
   (a) 44 (b) 42
   (c) 43 (d) 75

8. Consider the following C-prog. Find the number of tokens in the output of the following program if that is passed as a string to a lexical analyzer.
```
main ()
{
    char * s[ ] = {"ice", "green", "water", "hi"};
    char ** ptr[ ] = {s + 3, S + 2, S + 1, s}
    char *** p = ptr;
print f("in, % s", ** ++ P);
}
```
   (a) 5 (b) 1
   (c) 4 (d) 3

9. Consider line-3 of the following c-program.
```
int main ( ) {      /* line1*/
int i, n ;          /* line2*/
Fro (i = 0; i < n; i ++); /* line 3*/
```
   Identify the compiler response about the line 3 while creating the object module.
   (a) No compilation error
   (b) Only a lexical error
   (c) Only Syntactic error
   (d) Both lexical error and syntactic error

10. Find the number of tokens in the following *C* code using lexical analyzer of compiler.

```
main( )
{
    int *a, b;
    b = 10;
    a = &b;
    printf("%d%d", b, *a);
    b = */*pointer*/b;
}
```

(a) 35 (b) 45
(c) 55 (d) 65

11. Find the line number in which lexical error present in the following program.

```
1. main( )
2. {
3.     int x; y; z;
4.     /* comment1*/
5.     /*com/*ment2*/end*/
6.     x = "A";
7. }
```

(a) 3 (b) 5
(c) 6 (d) No lexical error

12. Which of the following equivalent one is used in lexer?
(a) Regular expression
(b) Context free language
(c) Both (a) and (b)
(d) Neither (a) nor (b)

13. Consider the following program.

```
main ( )
{
    char ch = 'A';
    int x, y;
    x = y = 20;
    x ++;
    printf("%d% d", x, y);
}
```

The number of tokens in the above program are _____.

■■■■

## Answers Lexical Analysis

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

## Explanations Lexical Analysis

**2. (b)**
Lexical analysis removes comments as well.
$(i = 0)$ will not generate lexical error. It will tokenise as fro, (, i, =, 0, ). No error is generated.

**5. (a)**
The given statement can be broken into
IF, (, NUMB, EQ, MAX, ), GOTO, 50C
∴ Number of tokens = 8

**6. (c)**
Number of tokens in C statement is 10
Print f, (, "I = %d, & I=%x", ,, i, ,, &, I, ), ;

**7. (b)**

```
① ② ③ ④ ⑤ ⑥
|int| strange| ( |int|x| ) |
⑦

⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯
|if| ( |x| <=| 0| ) | return| 0 | ;|
⑰ ⑱ ⑲ ⑳ ㉑ ㉒ ㉓ ㉔ ㉕ ㉖ ㉗ ㉘ ㉙ ㉚ ㉛
|if| ( | ( |x |%| 2| ) |!=| 0| )| return| x |-| 1| ;|
㉜ ㉝ ㉞ ㉟ ㊱ ㊲ ㊳ ㊴ ㊵ ㊶
|return| 1 | + |strange| ( |x | – | 1| ) | ;|
㊷
|)|
```

**8. (b)**

We need not to find out the actual output of the given C-program. Just by seeing the C-code we can understand that the output is a string and that is passed to lexical analyzer and that would be treated as a single valid lexeme for lexical analyzer. For instance we can see the output of the program will be 'water'.

Water (1) is considered as a single token.

**9. (c)**

The C-code contains syntax error, where 'fro' is written in place of 'for'. The lexical analyzer will not declare it as an error, whether 'fro' is a misspelling of the keyword 'for' or an undeclared function identifier. Since 'fro' is a valid lexeme for the token id, the lexeme analyzer must return the token id to the parser and let some other phase of the compiler (parser in this case) handle the error due to transposition of the letters.

**10. (a)**

```
main ( )
{
  int *a, b;
  b = 10;
  a = &b;
  printf ("%d%d", b, *a);
  b = */*pointer*/ b;
}
```

**11. (d)**

The given program contain no lexical error even though it contains syntax errors. In line number "5", comment started and searches for the first close comment pattern when it finds, it consider a comment. There is no start comment pattern (/*) but there is end comment at last in line 5, hence it is not lexical error but it is syntax error.

```
/* com / * ment2 * / end * /
```

Comment Identifier Operator ⇒ No lexical error

**12. (a)**

Regular expression is used in lexical analysis to identify the tokens.

**13. (33)**

```
main ( )
① ②③
{
④
  char ch = 'A' ;
   ⑤   ⑥⑦ ⑧⑨
  int x , y ;
  ⑩ ⑪⑫⑬⑭
  x = y = 20 ;
  ⑮⑯⑰ ⑱ ⑲ ⑳
  x ++ ;
  ㉑ ㉒ ㉓
  printf ( "%d%d" , x , y ) ;
   ㉔  ㉕    ㉖    ㉗㉘ ㉙ ㉚㉛㉜
}
㉝
```

■■■■

# 3 CHAPTER

# Syntax Analysis (Parser)

1. Which of the following suffices to convert an arbitrary CFG to an LL(1) grammar?
   (a) Removing left recursion alone
   (b) Factoring the grammar alone
   (c) Removing left recursion and factoring the grammar
   (d) None of above

2. Non backtracking form of top-down parser are called
   (a) recursive-descent parsers
   (b) predictive parsers
   (c) shift reduce parsers
   (d) none of these

3. Parsing table in the LR parsing contains
   (a) action, goto (b) state, action
   (c) input, action (d) state, goto

4. Consider the following grammar
   expr → expr op expr
   expr → id
   op → + | *
   Which of the following is true?
   (a) op and expr are start symbols
   (b) op and id are terminals
   (c) expr is start symbol and op is non terminal
   (d) none of these

5. Consider the grammar:
   $S \rightarrow aABe$
   $B \rightarrow d$
   $A \rightarrow Abc \mid b$
   For which the string abbcde is right most derivation
   (a) $S \rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde$
   (b) $S \Rightarrow aABe \Rightarrow aAbcBe \Rightarrow abbcBe \Rightarrow abbcde$
   (c) Both (a) and (b)
   (d) None of these

6. Consider the following grammar G:
   $S \rightarrow V = E$
   $E \rightarrow F \mid E + F$
   $F \rightarrow V \mid int \mid (E) \mid \epsilon$
   $V \rightarrow$ id
   Which of the following will represents the FIRST (F) and FOLLOW (F) respectively?
   (a) FIRST (F) = {id, int, c, ∈}
   FOLLOW (F) = {+ , ) , $}
   (b) FIRST (F) = {id, int, c}
   FOLLOW (F) = { + , ) , $ }
   (c) FIRST (F) = {id, int, c, ∈}
   FOLLOW (F) { + , ) }
   (d) None of these

7. Suppose a production $A \rightarrow \alpha B$ or $A \rightarrow \alpha B\beta$ where FIRST (β) contains ∈ then which of the following is true?
   (a) FOLLOW A is in FOLLOW B
   (b) FOLLOW B is in FOLLOW A
   (c) ∈ will be in FOLLOW A
   (d) Both (a) and (b)

8. stmt → if expr then stmt
   | if expr then stmt else stmt
   | other
   Consider the following statements for above grammar:
   1. The grammar will cause shift reduce conflict.
   2. The grammar is ambiguous.
   3. The grammar is left recursive.
   Which of the above statements is/are true?
   (a) 1 and 2 is true (b) 1, 2 and 3 is true
   (c) 2 and 3 is true (d) all are false

9. Consider the following statements:
   $S_1$: Viable prefixes of grammar G are those prefixes of right-sentenial form that can appear on top of stack of a shift reduce parsing.

$S_2$: Handle is sub-string that matches right side of production whose reduction to non terminal on left side of production represents one step along the reverse of leftmost derivation.

Which of the above statements is/are true?
(a) $S_1$ is true and $S_2$ is false
(b) $S_2$ is true and $S_1$ is false
(c) Both are true
(d) Both are false

10. Consider the following statements:
$S_1$: Every SLR(1) grammar is unambiguous but there are many unambiguous grammar that are not SLR(1).
$S_2$: LALR is most powerful parsing method.
Which of the above statements is/are true?
(a) $S_1$ is true and $S_2$ is false
(b) $S_1$ is false and $S_2$ is true
(c) Both are true
(d) Both are false

11. Choose the false statements.
(a) LL(k) grammar has to be a CFG
(b) LL(k) grammar has to be unambiguous
(c) There are LL(k) grammars that are not Context Free
(d) LL(k) grammars cannot have left recursive non-terminals

12. The operator in a grammar, which is derived first, has _____ precedence.
(a) high
(b) low
(c) depends on grammar
(d) cannot say

13. For an arbitrary grammar, number of states in LALR(1) parser is _____ number of states in LR(1) parser
(a) $\geq$ (b) $\leq$
(c) $=$ (d) $\neq$

14. The grammar is
$S \rightarrow A \mid B$
$A \rightarrow a \mid b$
$B \rightarrow b \mid c$
(a) LL(1) (b) LR(1)
(c) Both (a) and (b) (d) None of these

15. Backtracking is possible in
(a) LR parsing
(b) Predictive parsing
(c) Recursive descent parsing
(d) None of the above

16. 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) LR parser is more powerful than LALR parser.
(d) SLR, LR and LALR parsers have the same power.

17. In which of the following has attribute values at each node?
(a) Associated parse tree
(b) Postfix parse tree
(c) Annotated parse tree
(d) Prefix parse tree

18. Consider the grammar:
$S \rightarrow ABCc \mid ABc$
$BA \rightarrow AB$
$BC \rightarrow bb$
$AB \rightarrow ab$
$Aa \rightarrow aa$,
Which of the following sentence can be derived by this grammar?
(a) *abc* (b) *aab*
(c) *abcc* (d) *abbc*

19. Consider the grammar shown below:
$S \rightarrow i\ E\ t\ SS' \mid a$
$S' \rightarrow e\ S \mid \in$
$E \rightarrow b$
In predictive parsing table $M$, the entries $M[S', e]$ and $M[S', \$]$ respectively are
(a) $\{S' \rightarrow eS\}$ and $\{S' \rightarrow \in\}$
(b) $\{S' \rightarrow eS\}$ and $\{\}$
(c) $\{S' \rightarrow \in\}$ and $\{S' \rightarrow \in\}$
(d) $\{S' \rightarrow eS, S' \rightarrow \in\}$ and $\{S' \rightarrow \in\}$

20. What is the left most derivation for the sentence id + id * id, by considering the following grammar.

$E \to E + E \mid E * E \mid (E) \mid E - E \mid \text{id}$

(a) $E \Rightarrow E * E$
$\Rightarrow E + E * E$
$\Rightarrow \text{id} + E * E$
$\Rightarrow \text{id} + \text{id} * E$
$\Rightarrow \text{id} + \text{id} * \text{id}$

(b) $E \Rightarrow E + E$
$\Rightarrow E + E * E$
$\Rightarrow \text{id} + E * E$
$\Rightarrow \text{id} + \text{id} * E$
$\Rightarrow \text{id} + \text{id} * \text{id}$

(c) Both (a) and (b) (d) None of these

21. Consider the following grammar:

$E \to E / X \mid X$
$X \to T - X \mid X * T \mid T$
$T \to T + F \mid F$
$F \to (E) \mid \text{id}$

The above grammar is used to generate all valid arithmatic expressions in a hypothetical language in which

(a) / associates from the left
(b) * associates from the left
(c) + associates from the left
(d) all of the above

22. In operator precedence parsing, for the input id *(id – id) – id / id, where id is an identifier (terminal), the following grammar is given.

$E \to EAE \mid (E) \mid -E \mid \text{id}$
$A \to + \mid - \mid / \mid \uparrow$

Which symbol has the higher precedence than other?

(a) \$, + (b) id, *
(c) +, * (d) –, *

23. In canonical parsing table

$S \to CC$
$C \to cC \mid d$

How many states are there in canonical parsing table for the given grammar?

(a) 9 (b) 8
(c) 7 (d) 10

24 The following grammar is given for predictive parsing table.

$E \to TE'$ $T \to FT'$ $F \to (E)$
$E' \to +TE'$ $T' \to *FT'$
$E' \to \epsilon$ $T' \to \epsilon$
$F \to \text{id}$

The start symbol $E$ has entry in

(a) +, \$ (b) *, id
(c) id, ( (d) (, \$

25. Consider the following grammar:

1. $S \to S; S$
2. $S \to \text{id} := E$
3. $S \to \text{print (L)}$
4. $E \to \text{id}$
5. $E \to \text{num}$
6. $E \to E + E$
7. $E \to (S, E)$
8. $L \to E$
9. $L \to L, E$

How many states are there in LR parsing table?

(a) 15 (b) 27
(c) 23 (d) None of these

26. Consider the operator precedence relation.

| | id | + | * | \$ |
|---|---|---|---|---|
| id | | ·> | ·> | ·> |
| + | <· | ·> | ·> | ·> |
| * | <· | <· | ·> | ·> |
| \$ | <· | <· | <· | |

Suppose we are evaluating string $\text{id}_1 + \text{id}_2 * \text{id}_3$, give the order in which $\text{id}_1$, $\text{id}_2$ and $\text{id}_3$ will be evaluated

(a) $\text{id}_1, \text{id}_3, \text{id}_2$ (b) $\text{id}_2, \text{id}_1, \text{id}_3$
(c) $\text{id}_1, \text{id}_2, \text{id}_3$ (d) $\text{id}_3, \text{id}_2, \text{id}_1$

27. Operator-precedence parsing method is a parsing method. Which of the following statement is false about it?

1. It is bottom-up parsing method.
2. It must contain $\epsilon$-production.
3. It doesn't contain two adjacent nonterminal symbols.

(a) 1 only (b) 2 only
(c) 3 only (d) 1 and 3 only.

28. Which of the following derivations does a top-down parser use while parsing as input string? The input is assumed to be scaned in left to right order

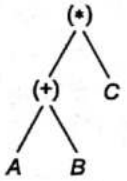
(a) Left most derivation
(b) Left most derivation traced out in reverse
(c) Right most derivation
(d) Right most derivation traced out in reverse

29. 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 CR, and LALR have the same power.

30. A bottom-up parser generates
(a) right-most derivation
(b) right-most derivation in reverse
(c) left-most derivation
(c) left-most derivation in reverse

31. A top-down parser generates
(a) right-most derivation
(b) right-most derivation in reverse
(c) left-most derivation
(d) left-most derivation in reverse

32. Which of the following expressions is represented by the parse tree?




(a) $(A + B) * C$ (b) $A + * BC$
(c) $A + B * C$ (d) $A * C + B$

33. Which of the following grammar is LR(1)?
(a) $A \to a\,A\,a$
$A \to b\,A\,b$
$A \to a$
$A \to a$
(b) $A \to a\,A\,a$
$A \to a\,A\,b$
$A \to c$
(c) $A \to A + A$
$A \to a$
(d) Both (a) and (b)

34. Which of the following parsers is the most powerful?
(a) Operator-precedence (b) Canonical LR
(c) LALR (d) SLR

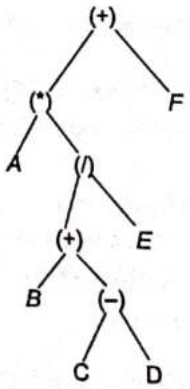
35. Handle pruning is the technique used to obtain
(a) canonical derivation sequence
(b) Canonical reduction sequence
(c) Both (a) and (b) above
(d) None of the above

36. Consider the following left-associative operators, in decreasing order of precedence:
– subtraction (highest precedence)
* multiplication
$ exponentiation (lowest precedence)
What is the result of the following expression?
$3 - 2 * 4 \$ 2 \$ 3$
(a) – 61 (b) 64
(c) 512 (d) 4096

37. Consider the grammar.
$S \to (S) \mid a$
Let the number of states in SLR (1), LR (1) and LALR(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) $n_1 \geq n_3 \geq n_2$

38. Consider the following grammar G:
$E \to TF$
$T \to xT$
$F \to y$
The number of states in LALR parser for G is _____.

39. Which of the following expressions is represented by the parse tree given below?




(a) $A + B * C - D / E + F$
(b) $A * (B + (C - D)) / E + F$
(c) $A + B * C - D / (E + F)$
(d) $A + B * (C - D) / (E + F)$

40. A grammar G is defined by:

$G(\{a, b\}, \{S, A, B\}, P, S)$

When $P$ is the set

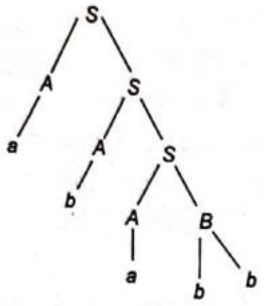
$S \rightarrow AB \mid AS$

$A \rightarrow a \mid aA$

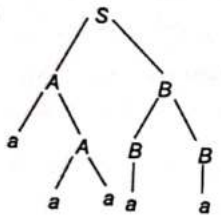
$B \rightarrow b \mid bb$

Consider the following trees which is / are legitimate for the string "*ababb*"?

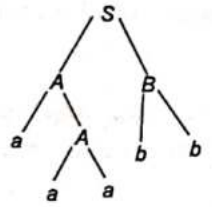
(a)




(b)




(c)




(d) None of the above

41. Which of the following actions an operator-precedence parser may take to recover from an error?
(a) Insert symbols onto the stack
(b) Delete symbols from the stack
(c) Insert or delete symbols from the input
(d) All of the above

42. Which of the following is the important factor that justifies the use of a stack in shift-reduce parsing?
(a) The handle will always appear on top of stack and never inside
(b) The handle will always appear inside the stack and never on top
(c) The handle will never appear in stack
(d) None of the above

43. $S \rightarrow AB$
$S \rightarrow CA$
$B \rightarrow BC$
$B \rightarrow AB$
$A \rightarrow a$
$C \rightarrow aB \mid b$

Check which of the following is correct for above grammar?
(a) the above grammar is in reduce form
(b) the above grammar is not in reduce form
(c) from given non-terminal we get the non-terminal only thus above grammer is in reduce form
(d) none of the above

44. Consider the following expression grammar. The semantic rules for expression calculation are stated next to each grammar production.

| | |
|---|---|
| $E \rightarrow$ number | $E.val = number.val$ |
| $\mid E$ '+' $E$ | $E^{(1)}.val = E^{(2)}.val + E^{(3)}.val$ |
| $\mid E$ '×' $E$ | $E^{(1)}.val = E^{(2)}.val \times E^{(3)}.val$ |

;

The above grammar and the semantic rules are fed to a yacc tool (which is an LALR (1) parser generator) for parsing and evaluating arithmetic expressions. Which one of the following is true about the action of yacc for the given grammar?
(a) it detects recursion and eliminates recursion
(b) it detects reduce-reduce conflict, and resolves
(c) it detects shift-reduce conflict, and resolves the conflict in favor of a shift over a reduce action
(d) it detects shift-reduce conflict, and resolves the conflict in favor of a reduce over a shift action

45. Consider the following grammar production.

$S \rightarrow X$
$X \rightarrow YX \mid \in$
$Y \rightarrow aY \mid b$

Which of the following is not LR(1) item set?
(a) $\{S \to X\cdot, \$\}$
(b) $\{Y \to a\cdot, a \mid b \mid \$\}$
(c) $\{Y \to b\cdot, a \mid b \mid \$\}$
(d) $\{[Y \to a\cdot Y, a \mid b \mid \$], [Y \to \cdot aY, a \mid b \mid \$\}$

46. Consider the following grammar production.
$E \to E+E \mid E \times E \mid (E) \mid$ id and id $\to 0 \mid 1 \mid 2 \mid \ldots \mid 9$
The expression $3 + 2 \times 5$ is parsed by the shift-reduce parser then which of the following is not a handle for it?
(a) $E + E$ (b) $E \times E$
(c) $E + E \times E$ (d) 3

47. Consider the LR(0), LR(k), LR(1) and SLR(1) grammars then which of the following about these grammar is true?
(a) LR(0) ⊂ SLR(1) ⊂ LR(1) ⊂ LR(k)
(b) LR(0) ⊂ LR(k) ⊂ LR(1) ⊂ SLR(1)
(c) SLR(1) ⊂ LR(0) ⊂ LR(1) ⊂ LR(K)
(d) LR(K) ⊂ LR(0) ⊂ LR(1) ⊂ SLR(1)

48. Consider the following grammar:
$S \to ABDh$
$B \to cC$
$C \to bc \mid \epsilon$
$D \to EF$
$E \to g \mid \epsilon$
$F \to f \mid \epsilon$
How many pair/pairs of elements has equivalent FOLLOW sets?
(a) 0 (b) 1
(c) 2 (d) 3

49. Consider the following grammar *G*.
1. $S \to X1$ 2. $X \to A1B$
3. $X \to 2$ 4. $A \to 2$
5. $B \to A$
The above grammar is
(a) not LL(0), not LL(1), not LR(0), but SLR(1), LR(1) and LALR(1)
(b) not LL(0), not LL(1), not LR(0), but SLR(1), LR(1) and not LALR(1)
(c) not LL(0), not LL(1), not LR(0), but SLR(1) but LR(1) and LALR(1)
(d) not LL(0), not LL(1), not LR(0), not SLR(1), not LR(1) and not LALR(1)

50. Fill in the blanks in the following statements:
$S_1$: Merging states with a common core may produce ___I___ conflicts but does not produce ___II___ conflicts in LALR parser.
$S_2$: The LR parsing method is the most general ___III___ ___IV___ parsing method.
(a) I-reduce-reduce; II-shift reduce III non backtracking IV-predictive
(b) I-shift-reduce; II-reduce-reduce; III back tracking IV-shift-reduce
(c) I-shift-reduce; II-reduce-reduce; III back tracking; IV-predictive
(d) I-reduce-reduce; II-shift-reduce; III-non back tracking; IV shift-reduce

51. What will be the resulting grammar after removal of left-recursion from the following grammar?
$E \to Ea \mid Eb \mid a \mid b$
(a) $E \to aE' \mid bE'$ ; $E' \to aE' \mid bE' \mid \epsilon$
(b) $E \to aE' \mid bE'$ ; $E' \to aE \mid bE \mid \epsilon$
(c) $E \to aE' \mid bE' \mid \epsilon$ ; $E' \to aE' \mid bE' \mid \epsilon$
(d) $E \to aE' \mid bE'$ ; $E' \to a \mid b \mid \epsilon$

52. How many conflicts are there in the following grammar
$S \to SS \mid a \mid \epsilon$
(a) 3 (b) 4
(c) 5 (d) 6

53. Let *G* be any arbitary grammar and *G* known to be LR(1). Assume that the number of error entries in the LR(0) table is '*a*' and the number of error entries in the SLR(1) table is '*b*'. What is the relation over *a* and *b*? [Assume an error is SR conflict or RR conflict]
(a) $a > b > 0$ (b) $a \geq b \geq 0$
(c) $a \leq b \leq 0$ (d) $a < b < 0$

54. Let *G* be a grammar and *G* has following set of productions.
$S \to AA$
$A \to AA \mid bA \mid Ab \mid a$
How many strings can be generated by grammar G with atmost four steps in the derivation. [initially start symbol is available without any step]
(a) 3 (b) 4
(c) 5 (d) 6

55. Which of the following is true?
(a) If grammar is LL(1) then it must be CLR(1)
(b) If grammar is SLR(1) then it must be CLR(1)
(c) If grammar is CLR(1) then it must be LL(1) or SLR(1)
(d) Both (a) and (b)

56. Consider the following grammar:
$S \to S + A \mid A$
$A \to B * A \mid B$
$B \to B - C \mid C$
$C \to \text{id}$
In above grammar
(a) + is left associative, * is right associative
(b) + is left associative, – is right associative
(c) * is left associative, – is left associative
(d) None of these

57. Consider the following grammar:
1. $S \to XYX$
2. $X \to Yc$
3. $X \to sX$
4. $X \to \varepsilon$
5. $Y \to eX$

For the above CFG, LL(1) parsing table constructed using production numbers are as follows :

| | c | s | e | $ |
|---|---|---|---|---|
| S | | 1 | 1 | |
| X | 4 | $E_1$ | $E_2$ | 4 |
| Y | | | 5 | |

Find the missing entries $E_1$ and $E_2$ in the above LL(1) table.
(a) $E_1 = \{2\}$, $E_2 = \{3, 4\}$
(b) $E_1 = \{3, 4\}$, $E_2 = \{2\}$
(c) $E_1 = \{2, 4\}$, $E_2 = \{3, 4\}$
(d) $E_1 = \{3, 4\}$, $E_2 = \{2, 4\}$

58. Consider the following grammar:
$S \to abS \mid acS \mid c.$
The number of states in canonical set of LR(0) items for the above grammar is _____.

59. Consider the following grammar:
$S \to Aa \mid bAc \mid Bc \mid bBa$
$A \to d$
$B \to d$
Which of the following is true about above grammar?
(a) LR(1) but not LALR(1)
(b) Both LR(1) and LALR(1)
(c) Neither LR(1) nor LALR(1)
(d) None of these

60. Consider the following grammar:
$S \to M N z S c$
$M \to a M a \mid \epsilon$
$N \to b N b \mid \epsilon$
Which of the following will represent the FOLLOW(S) and FIRST(S) respectively?
(a) FOLLOW (S) = {c}, FIRST (S) = {a, b, z}
(b) FOLLOW (S) = {$, c}, FIRST (S) = {a, b, z}
(c) FOLLOW (S) = {a, c}, FIRST (S) = {a, b, z}
(d) None of these

61. Consider the following grammar:
$S \to Ax \mid By$
$A \to z$
$B \to z$
The total number of inadequate states during LR(0) parser is _____.

62. Consider the following grammar:
$S \to XX$
$X \to aX$
$X \to c$
The total number of reduced states in construction of LALR(1) from CLR(1) parser is _____.

63. Consider the following grammar:
$T \to W \mid aTc$
$W \to b\,W \mid d$
The total number of states in SLR(1) parsing table of the above grammar are _____.

64. Consider the following grammer:
$S \to \text{aAb} \mid \text{Sc}$
$A \to \text{d} \mid \text{Sd} \mid \text{S}$
The above grammer is
(a) SLR(1) (b) LL(1)
(c) LR(0) (d) None of these

65. Let G be a grammar with the following productions.
$E \to E + T \mid T$
$T \to T * F \mid F$
$T \to (E) \mid T - F$
$F \to \text{id}$

If LR(1) Parser is used to construct the DFA using the above productions, then how many look-a-heads are present for an item $T \to gT^* F$ in the initial state?

**66.** During the LR(0) construction of a grammar we get an intermediate state

$S' \to T$
$T \to F$
$T \to T^* F$
$T \to \text{id}$
$F \to \text{id} \mid (T)$
$F \to \text{id} = T;$

Then which of following statement is true
(a) There is only one shift/reduce conflict in intermediate state
(b) There is only one reduce/reduce conflict in intermediate state
(c) There is both shift/reduce and reduce/reduce conflict in intermediate state
(d) There is no such intermediate state.

**67.** Consider the following augmented grammar *G*.

$G: E' \to E$
$E \to abE \mid EcE \mid d \mid e$

Which of the following is correct about construction of LR(1) parser of grammar *G*?
(a) 2 states contain RR conflicts
(b) 2 states contain SR conflicts
(c) 3 states contain SR conflicts
(d) None of these

**68.** Consider the following operator grammar

$S \to BbA \mid bA \mid bBA$
$B \to b \mid c$
$A \to a$

Which of the following precedence relation is correct from above grammar? Assume $x <\cdot y$ is used to represent *y* has highest precedence than *x* and in expression *y* appears first then *x* appears next.
(a) $a <\cdot b$ (b) $c <\cdot a$
(c) Both (a) and (b) (d) None of these

**69.** Consider the following CFG:

$S \to ABC \mid a \mid \in$
$A \to BC \mid b \mid \in$
$B \to c \mid \in$
$C \to d \mid \in$

Compute the FOLLOW set of non-terminal *A*.
(a) {c, $, d} (b) {a, b, c}
(c) {d, c, ∈} (d) {a, b, ∈}

**70.** Let G be the following grammar

$A \to AB \mid a$
$B \to *AC \mid Cb \mid e$
$C \to +ABc \mid e$

Find the total number of reductions using LR(1) parser for the string a*a + ac using grammar G.

**71.** Which of the following sets could result SR conflict in LALR (1)?
(a) $A \to a.b, \{b\}$ $B \to a., \{a\}$ (b) $A \to a.a, \{a\}$ $B \to a., \{b\}$
(c) $A \to b.a, \{b\}$ $B \to b., \{a\}$ (d) $A \to b.b, \{a\}$ $B \to b., \{a\}$

**72.** Consider the following grammar:

$S \to ABC \mid aC$
$A \to bC \mid \in$
$B \to \in$
$C \to e \mid \in$

Which of the following will represent the FOLLOW (A) and FIRST (*S*) set?
(a) FOLLOW (*A*) = {e, $}
FIRST (*S*) = {a, b, e, ∈}
(b) FOLLOW (*A*) = {a, b, $}
FIRST (*S*) = {a, b, e, ∈}
(c) FOLLOW (*A*) = {e, $}
FIRST (*S*) = {a, b, e, $}
(d) FOLLOW (*A*) = {b, $}
FIRST (*S*) = {b, e, ∈}

**73.** Consider the grammar G shown below:

$G: E \to EzE \mid xyE \mid w$

Which of the following conflict is present in the DFA construction of SLR(1) parser for grammar G.
(a) Reduce-Reduce conflict
(b) Shift-Reduce conflict
(c) Both (a) and (b)
(d) No conflict present

**74.** Consider the following grammar

$S \to (L) \mid a$
$L \to L, S \mid S$

The maximum size of stack during *LL*(1) parsing the input string $w = (a, a)$ is _____.

## Answers Lexical Analysis

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | (c) | 2. | (b) | 3. | (a) | 4. | (c) | 5. | (a) | 6. | (a) | 7. | (a) | 8. | (a) | 9. | (a) |
| 10. | (a) | 11. | (c) | 12. | (b) | 13. | (b) | 14. | (d) | 15. | (c) | 16. | (c) | 17. | (c) | 18. | (a) |
| 19. | (d) | 20. | (c) | 21. | (d) | 22. | (b) | 23. | (d) | 24. | (c) | 25. | (c) | 26. | (c) | 27. | (b) |
| 28. | (a) | 29. | (c) | 30. | (b) | 31. | (c) | 32. | (a) | 33. | (d) | 34. | (b) | 35. | (b) | 36. | (d) |
| 37. | (b) | 39. | (b) | 40. | (a) | 41. | (d) | 42. | (a) | 43. | (b) | 44. | (c) | 45. | (a) | 46. | (c) |
| 47. | (a) | 48. | (c) | 49. | (d) | 50. | (d) | 51. | (a) | 52. | (b) | 53. | (b) | 54. | (b) | 55. | (d) |
| 56. | (a) | 57. | (d) | 58. | (8) | 59. | (a) | 60. | (b) | 62. | (3) | 63. | (9) | 64. | (a) | 66. | (c) |
| 67. | (b) | 68. | (a) | 69. | (a) | 71. | (c) | 72. | (a) | 73. | (b) | | | | | | |

## Explanations Lexical Analysis

**5. (a)**

The string abbcde can be obtained using RMD by following steps

$S \rightarrow aABe \rightarrow aAde \rightarrow aAbcde \rightarrow abbcde.$

**6. (a)**

FOLLOW (F) = Contain terminal after F on RHS of grammar, if nothing present right hand side of 'F' then find FOLLOW (LHS) i.e. FOLLOW (E) = { + , )} and again FOLLOW (S) = {$}

So FOLLOW (F) = {+ , ) , & }

FIRST (F) = Contain first terminal on RHS i.e. {id, int, *c*, ∈}

**8. (a)**

The grammar

stmt → if expr then stmt

| if expr then stmt else stmt

| other

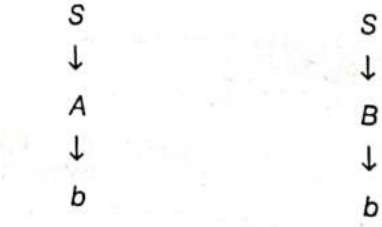
When we construct parsing table it will cause shift reduce conflict because at else we will not be able to decide whether shift or reduce and also the given grammar is ambiguous.

**9. (a)**

"Handle" is a substring that matches the right side of a production, and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a right most derivation.

**14. (d)**

The grammar is ambiguous because the string '*b*' has two parse trees as shown below.




**18. (a)**

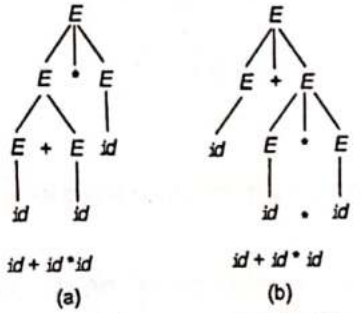
$S \rightarrow ABc \rightarrow abc$

**19. (d)**

| Non-terminal | Input symbol | | | | | |
|---|---|---|---|---|---|---|
| | a | b | e | i | t | $ |
| S | $S \rightarrow a$ | | | $S \rightarrow iEtSS'$ | | |
| S' | | | $S' \rightarrow \epsilon$<br>$S' \rightarrow eS$ | | | $S' \rightarrow \epsilon$ |
| E | | $E \rightarrow b$ | | | | |

20. (c)

$E \to E + E \mid E * E \mid (E) \mid -E \mid \text{id}$




Both (a) and (b) generated string id + id * id by LMD. So answer will be (c).

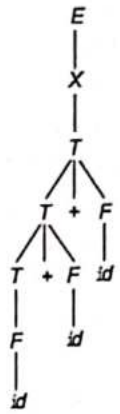
21. (d)

For checking associativity of /, consider the string id / id / id


The parse tree grows down towards the left, hence is left associative. For checking associativity of *, consider the string id * id * id


The parse tree grows down towards the left, hence * is left associative.

For checking associativity of +, consider the string id + id + id



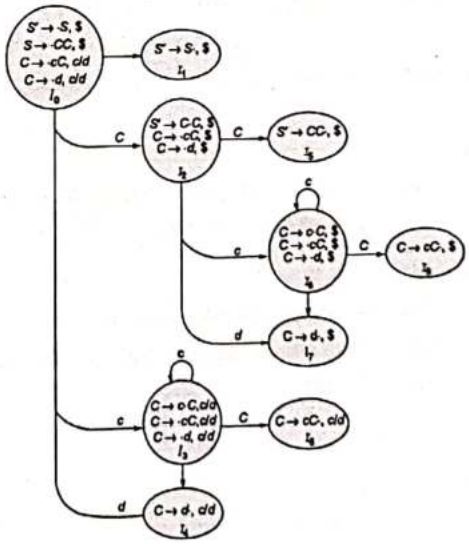

The parse tree grows down towards left, hence + is left associative.

22. (b)

| | + | − | * | / | ↑ | id | ( | ) | $ |
|---|---|---|---|---|---|---|---|---|---|
| + | ·> | ·> | <· | <· | <· | <· | <· | ·> | ·> |
| − | ·> | ·> | <· | <· | <· | <· | <· | ·> | ·> |
| * | ·> | ·> | ·> | ·> | <· | <· | <· | ·> | ·> |
| / | ·> | ·> | ·> | ·> | <· | <· | <· | ·> | ·> |
| ↑ | ·> | ·> | ·> | ·> | <· | <· | <· | ·> | ·> |
| id | ·> | ·> | ·> | ·> | ·> | | | ·> | ·> |
| ( | <· | <· | <· | <· | <· | <· | <· | ≐ | |
| ) | ·> | ·> | ·> | ·> | ·> | | | ·> | ·> |
| $ | <· | <· | <· | <· | <· | <· | <· | | |

23. (d)

By using the algorithm for construction of the canonical LR parsing table.



**24. (c)**

Predictive parsing table for the given grammar are created by calculating FIRST and FOLLOW for each nonterminal first. Then make entries according to the algorithm of predictive parsing table.

| | + | * | id | ( | ) | $ |
|---|---|---|---|---|---|---|
| $E$ | | | $E \to TE'$ | $E \to TE'$ | | |
| $E'$ | $E' \to +TE'$ | | | | $E' \to \epsilon$ | $E' \to \epsilon$ |
| $T$ | | | $T \to FT'$ | $T \to FT'$ | | |
| $T'$ | $T' \to \epsilon$ | $T' \to *FT'$ | | | $T' \to \epsilon$ | $T' \to \epsilon$ |
| $F$ | | | $F \to id$ | $F \to (E)$ | | |

**25. (c)**

| | id | num | print | ; | , | + | := | ( | ) | $ | S | E | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $S_4$ | | $S_7$ | | | | | | | | | | |
| 2 | | | | $S_3$ | | | | | | acc | $g_2$ | | |
| 3 | $S_4$ | | $S_7$ | | | | | | | | | | |
| 4 | | | | | | | | | | | $g_5$ | | |
| 5 | | | | $r_1$ | $r_1$ | | $S_6$ | | | $r_1$ | | | |
| 6 | $S_{20}$ | $S_{10}$ | | | | | | | $S_8$ | | | | |
| 7 | | | | | | | | | $S_9$ | | $g_{11}$ | | |
| 8 | $S_4$ | | $S_7$ | | | | | | | | | | |
| 9 | | | | | | | | | | | $g_{12}$ | | |
| 10 | | | | $r_5$ | $r_5$ | $r_5$ | | | $r_5$ | $r_5$ | | $g_{15}$ | $g_{14}$ |
| 11 | | | | $r_2$ | $r_2$ | $S_{16}$ | | | | | | | |
| 12 | | | | $S_3$ | $S_{18}$ | | | | | $r_2$ | | | |
| 13 | | | | $r_3$ | $r_3$ | | | | | | | | |
| 14 | | | | | | | | | | $r_3$ | | | |
| 15 | | | | | $S_{19}$ | | | | $S_{13}$ | | | | |
| 16 | $S_{20}$ | $S_{10}$ | | | $r_8$ | | | | $r_8$ | | | | |
| 17 | | | | | | | | $S_8$ | | | $g_{17}$ | | |
| 18 | $S_{20}$ | $S_{10}$ | | $r_6$ | $r_6$ | $S_{16}$ | | | $r_6$ | $r_6$ | | | |
| 19 | $S_{20}$ | $S_{10}$ | | | | | | $S_8$ | | | $g_{21}$ | | |
| 20 | | | | $r_4$ | $r_4$ | $r_4$ | | $S_8$ | $r_4$ | $r_4$ | $g_{23}$ | | |
| 21 | | | | | | | | | $S_{22}$ | | | | |
| 22 | | | | $r_7$ | $r_7$ | $r_7$ | | | $r_7$ | $r_7$ | | | |
| 23 | | | | | $r_9$ | $S_{16}$ | | | $r_9$ | | | | |

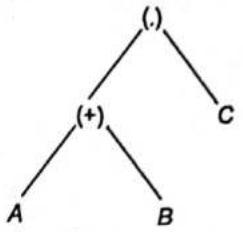
**26. (c)**

Here, + has higher precedence than *. So the order of evaluation will be id, $id_2$ and $id_3$.

**29. (c)**

The canonical LR parser is the most powerful parser that can recognize more grammars than other parsers.

**32. (a)**

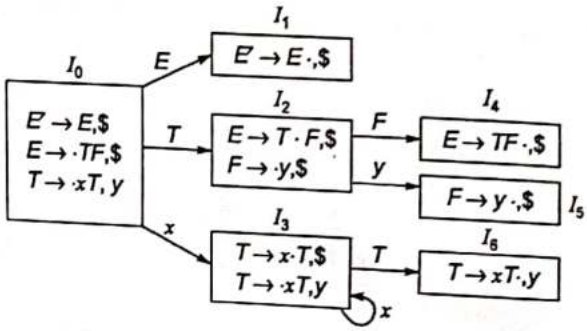



By applying inorder traversal we have $(A + B) * C$.

**35. (b)**

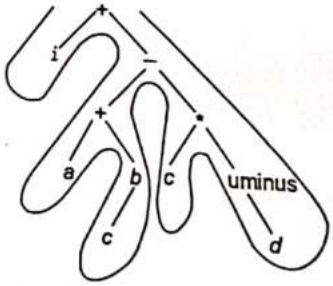
Handle pruning is the technique used in bottom to up parsing i.e. canonical reduction sequence.

**38. (7)**




Above LALR construction has 7 states.

**39. (b)**




Hence, the given parse tree gives the expression

$((A * ((B + (C - D)) / E)) + F$

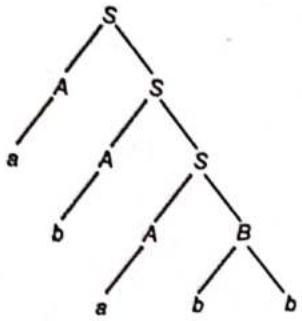
**40. (a)**

$S \to AB \mid AS$

$A \to a \mid aA$

$B \to b \mid bb$

will generate the parse tree




**43. (b)**

$S \to AB$
$S \to CA$
$B \to BC$
$B \to AB$
$A \to a$
$C \to aB \mid b$

The above grammar is not in reduce form and reduced form is

$S \to CA$
$A \to a$
$C \to aB$
$C \to b$

**47. (a)**

The grammar's recognizing powers are as follows

$$LR(0) \subset SLR(1) \subset LR(1) \subset LR(K)$$

more is the look ahead symbol number, more accurate is the recognizing power of the parser.

**48. (c)**

| | First | Follow |
|---|---|---|
| S | c | $ |
| B | c | g, f, h |
| C | b | g, f, h |
| D | g, f, ∈ | h |
| E | g, ∈ | h, f |
| F | f, ∈ | h |

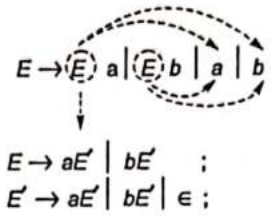
Here 2 pair/pairs of elements has equivalent follow sets.

**50. (d)**

Merging states with a common care may produce reduce-reduce conflicts but does not produce shift-reduce conflicts in LALR parser.

The LR parsing method is the most general non backtracking shift reduce parsing method.

**51. (a)**




**52. (b)**

In this, first we draw the goto graph with LR(0) item then find the number of inadequate state so we get

| | a | | action |
|---|---|---|---|
| 0 | $s_3/r_3$ | 2 | $s_1/r_1/r_3$ \| $r_1/r_3$ \| $r_1/r_3$ |

So there are 3-shift reduce and 1-reduce-reduce conflicts. So that 4 conflict.

**53. (b)**

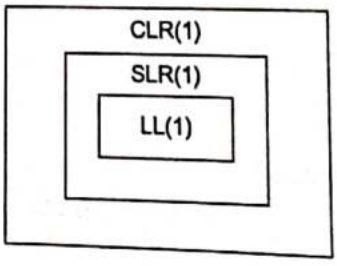
# error entries in LR(0) ≥ # error entries in SLR(1) ≥ # error entries in LR(1)
G is LR(1), so no error in LR(1)
$\therefore \quad a \geq b \geq 0$

**54. (b)**

$S \to AA \to aA \to aa$
$S \to AA \to aA \to abA \to aba$
$S \to AA \to aA \to aAb \to aab$
$S \to AA \to Aa \to bAa \to baa$
$\therefore$ {aa, aba, aab, baa} can be generated within 4 steps.

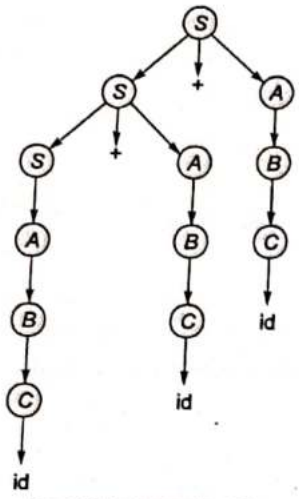
**55. (d)**

LL(1) is CLR(1).
SLR(1) is also CLR(1).
CLR(1) need not be LL(1) or SLR(1).

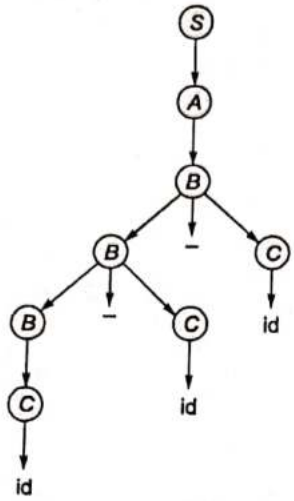


**56. (a)**

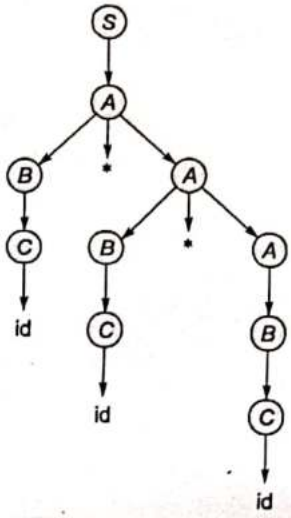
Consider 3 strings id + id + id, id – id – id and id * id * id.




So, '+' is left associative




So, '–' is left associative




So, '*' is right associative

**57. (d)**

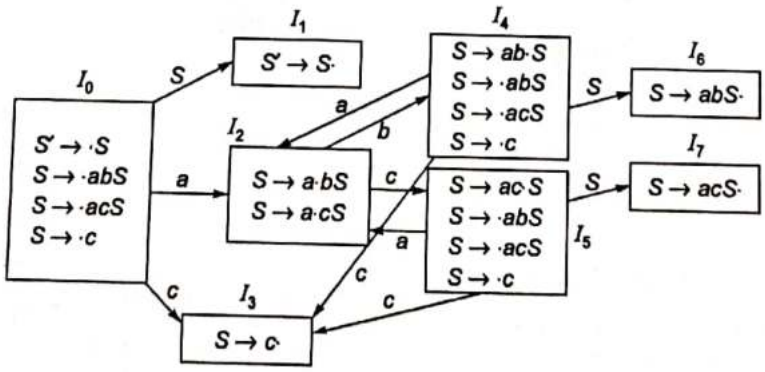
FIRST $(X) = \{s, e, \varepsilon\}$

FOLLOW $(X) = \{e, c, s, \$\}$

| | $c$ | $s$ | $e$ | $\$$ |
|---|---|---|---|---|
| $X$ | $X \to \varepsilon(4)$ | $X \to sX(3)$<br>$X \to \varepsilon(4)$<br>$= E_1$ | $X \to Yc(2)$<br>$X \to \varepsilon(4)$<br>$= E_2$ | $X \to \varepsilon(4)$ |

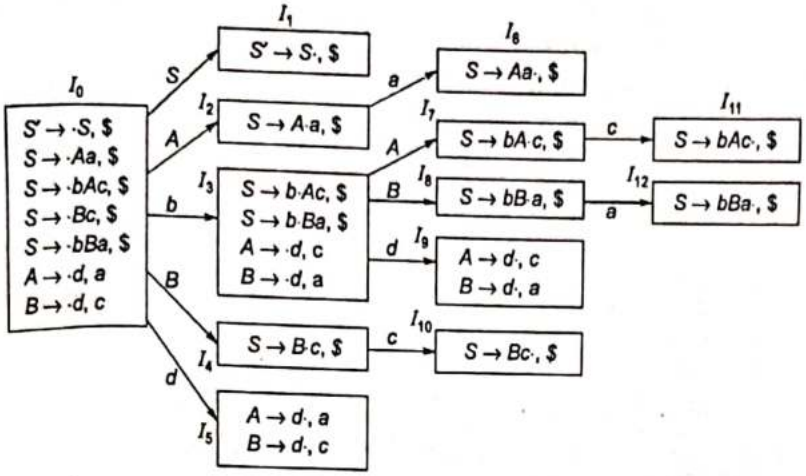
$\therefore$ $E_1 = \{3, 4\}$ and $E_2 = \{2, 4\}$

**58. (8)**




Total 8 states.

59. (a)




Since their is no conflict in any state in parsing table. So given grammar is LR(1) but when we merge $I_5$ and $I_9$ the resulting state will be

$I_5 + I_9 = A \to d\cdot, a|c$

$B \to d\cdot, a|c$ creates reduce-reduce conflict.

So given grammar is not LALR(1). Therefore given grammar is LR(1) but not LALR(1).

60. (b)

FOLLOW (S) = {c, $}

FIRST (S) = FIRST ($M\,N\,z\,S\,c$)

= {a, b, z}

61. (1)

LR(0) item set is given below:

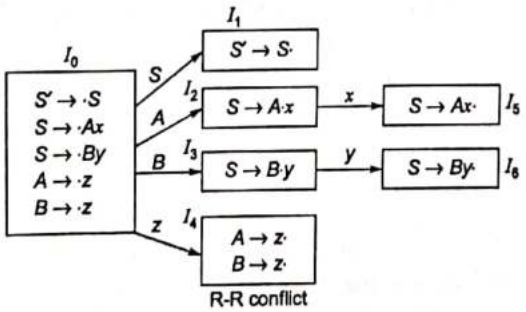
$S' \to .S$

$S \to .Ax$

$S \to .By$

$A \to .z$

$B \to .z$



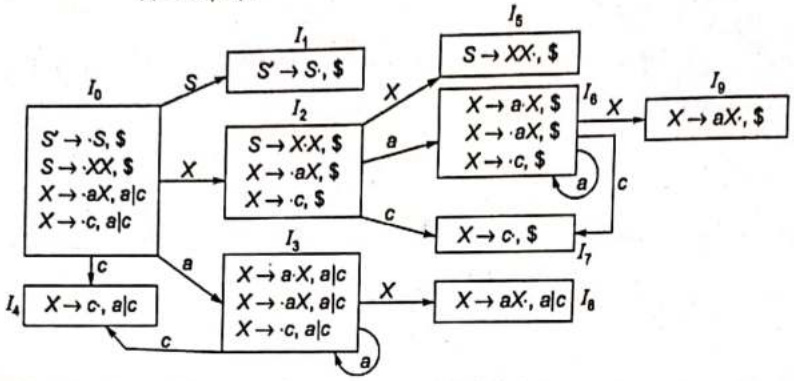

There are total 7 states in which only state $I_4$ is inadequate state (contain R-R conflict).

So there is 1 inadequate state.

**62. (3)**

LR(1) item set is given below:

$S' \to .S, \$$
$S \to .XX, \$$
$X \to .aX, a|c$
$X \to .c, a|c$

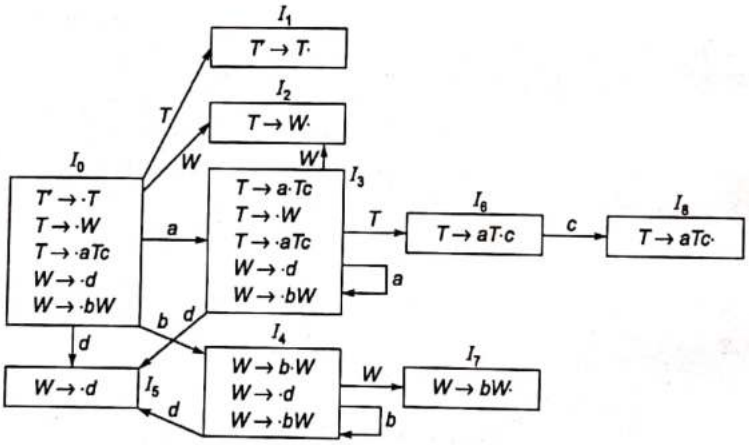



Total 10 states in CLR(1) parser.

Here, state $(I_3, I_6)$, $(I_4, I_7)$ and $(I_8, I_9)$ have same transition item over *a* and *c* respectively which only differ in look ahead symbols. So to make LALR(1) combines $(I_3, I_6 = I_{36})$, $(I_4, I_7 = I_{47})$ and $(I_8, I_9 = I_{89})$.

So total number of states in LALR(1) is 7 and reduced states is 3.

**63. (9)**




**64. (a)**

Since the above grammer is left recursive: It is not LL(1).

LR(0) items have S-R conflict.

No conflict in SLR(1)

**65. (4)**

$E' \to \bullet E, \$$ ⇒ 1 Look-a-heads
$E \to \bullet E + T, \{\$, +\}$ ⇒ 2 Look-a-heads
$E \to \bullet T, \{\$, +\}$ ⇒ 2 Look-a-heads
$T \to \bullet T * F, \{*, -, \$, +\}$ ⇒ 4 Look-a-heads
$T \to \bullet F, \{*, -, \$, +\}$ ⇒ 4 Look-a-heads

$T \rightarrow \bullet (E),\ \{*, -, \$, +\} \Rightarrow$ 4 Look-a-heads
$T \rightarrow \bullet\ T - F,\ \{*, -, \$, +\} \Rightarrow$ 4 Look-a-heads
$T \rightarrow \bullet$ id, $\{*, -, \$, +\} \Rightarrow$ 4 Look-a-heads

66. (c)

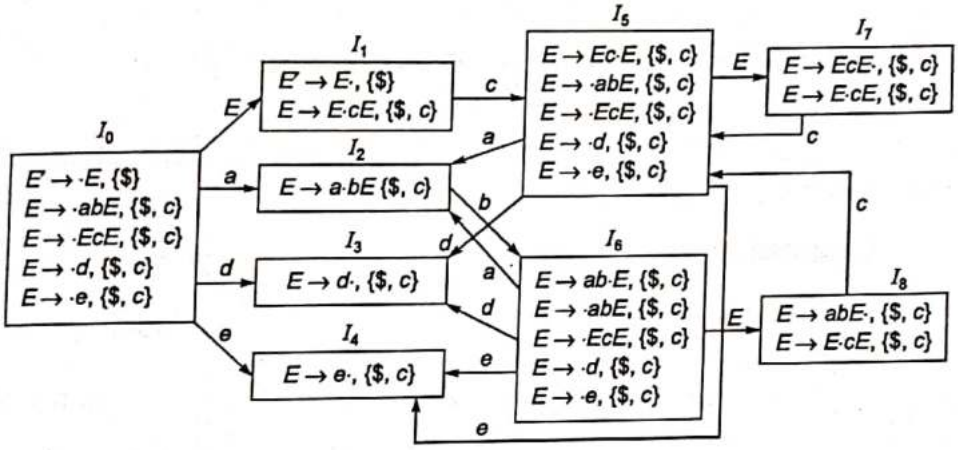
$F \rightarrow$ id $\bullet$
$F \rightarrow$ id $\bullet$ = T

Shift-Reduce conflict

$F \rightarrow$ id $\bullet$
$T \rightarrow$ id $\bullet$

Reduce-Reduce conflict

67. (b)

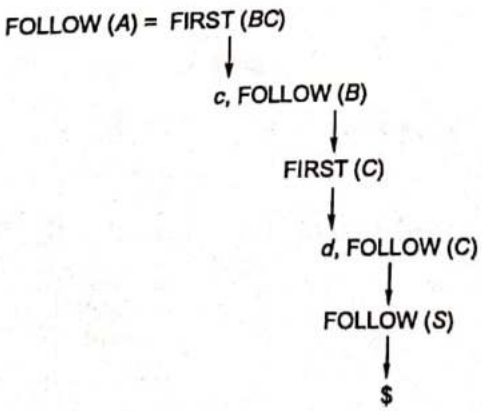



In above LR(1) state $I_7$ and $I_8$ contain SR conflicts.
$\therefore$ 2 states contains SR conflict.

68. (a)

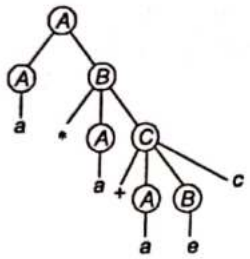
In the given grammar for derivation of any string '*b*' and '*c*' comes first then atlast '*a*' come. So *a* has less precedence than *b* and *c* but *c* don't have less precedence than *a*.

69. (a)




FOLLOW (*A*) is {*c*, *d*, $}

**70. (7)**




String = "a * a + ac".

Number of reductions = Number of intermediate nodes = Number of substitutions = 7 [reduction order in LR parser must be reverse of RMD].

**71. (c)**

$A \rightarrow b.a, \{b\}$

$B \rightarrow b., \{a\}$

**a** belongs to set of look-a-heads of **B**. Shift reduced conflict occurs in option (c).

**72. (a)**

FIRST (*S*) = FIRST (*A*) and {*a*}
↓
{*b*, ∈}, so FIRST (*B*)
↓
{∈}, so FIRST (*C*)
↓
{*e*, ∈}

So, FIRST (*S*) = {*a*, *b*, *e*, ∈}

FOLLOW (*A*) = FIRST (*B*)
↓
{∈}, so FOLLOW (*B*)
↓
FIRST (*C*)
↓
{*e*, ∈}, so FOLLOW (*C*)
↓
FOLLOW (*S*)
↓
{$}

So FOLLOW (*A*) = {*e*, $}

**73. (b)**

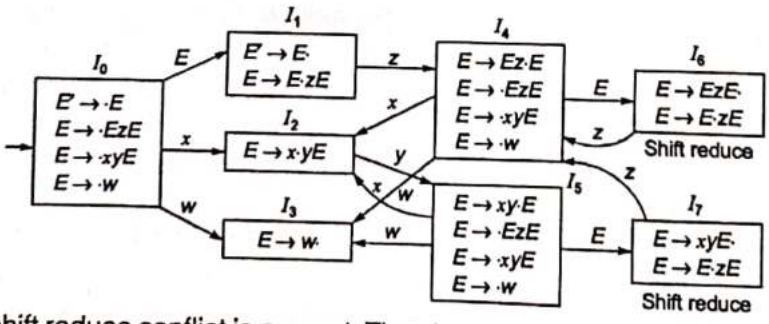
Augmented grammar G:

$E' \rightarrow \cdot E$

$E \rightarrow \cdot EzE$

$E \rightarrow \cdot xyE$

$E' \rightarrow \cdot w$




In state $I_6$ and $I_7$ shift reduce conflict is present. The element to be shifted is reduced so they will come in same column.

74. (4)

Since given grammar is left recursive, so first convert into non-left recursive.

$S \rightarrow (L) \mid a$

$L \rightarrow SL'$

$L' \rightarrow , SL' \mid \in$

First $(S) = \{(, a\}$ Follow $(S) = \{, , ), \$\}$

First $(L) = \{(, a\}$ Follow $(L) = \{)\}$

First $(L') = \{, , \text{follow}(L')\}$

$= \{, , )\}$

| | ( | ) | a | , | $ |
|---|---|---|---|---|---|
| S | $S \rightarrow (L)$ | | $S \rightarrow a$ | | |
| L | $S \rightarrow SL'$ | | $L \rightarrow SL'$ | | |
| L' | | $L' \rightarrow \in$ | | $L' \rightarrow SL'$ | |

**LL(1) Table**

| Stack | Input | Output |
|---|---|---|
| $\$S$ | $(a, a)\$$ | |
| $\$)L($ | $(a, a)\$$ | $S \rightarrow (L)$ |
| $\$)L'$ | $a, a)\$$ | |
| $\$)L'S$ | $a, a)\$$ | $S \rightarrow SL'$ |
| $\$)L'a$ | $a, a)\$$ | $S \rightarrow a$ |
| $\$).L'$ | $,a)\$$ | |
| $\$)L'S,$ | $,a)\$$ | $L' \rightarrow , SL'$ |
| $\$)L'S$ | $a)\$$ | |
| $\$)L'a$ | $a)\$$ | $S \rightarrow a$ |
| $\$)L'$ | $)\$$ | |
| $\$)$ | $)\$$ | $S \rightarrow \in$ |
| $\$$ | $\$$ | |

Maximum size of stack is 4.

■■■■

# 4 CHAPTER
# Syntax Directed Translation and Intermediate Code Generation

1. An intermediate code form is
   (a) postfix notation
   (b) syntax trees
   (c) three address codes
   (d) all of these

2. In a syntax directed translation scheme, if the value of an attribute of a node is a function of the values of the attributes of its children, then it is called a
   (a) synthesized attribute
   (b) inherited attribute
   (c) canonical attribute
   (d) none of the above

3. Synthesized attribute can easily be simulated by an
   (a) LL grammar (b) ambiguous grammar
   (c) LR grammar (d) none of these

4. Which of the following is not an intermediate code form?
   (a) Postfix notation
   (b) Syntax trees
   (c) Three address codes
   (d) Quadruples

5. Which of the following statements is/are correct?
   1. All S attributes are L attributes.
   2. All L attributes are S attributes.
   (a) 1 only (b) 2 only
   (c) Both are true (d) Both are false

6. Choose the correct statement.
   (a) S attributes are evaluated using depth first search
   (b) L attributes are evaluated using depth first search
   (c) Inherited attributes are evaluated using breadth first search
   (d) All of the above

7. 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) none of the above

8. Inherited attribute is a natural choice in
   (a) keeping track of variable declaration
   (b) checking for the correct use of L-values and R-values
   (c) both (a) and (b)
   (d) none of the above

9. Consider the following syntax directed definition

   $A \rightarrow BC$ $C.i = c(A.i)$ ... 1
   $B.i = b(A.s)$ ... 2
   $A.s = f(B.s)$ ... 3

   i for inherited, s for synthesized attribute
   Which of the following is true?
   (a) 1 is violating L attributed definition
   (b) 2 is violating L attributed definition
   (c) 3 is violating L attributed definition
   (d) None of the above

10. Generation of intermediate code based on a abstract machine model is useful in compilers because
    (a) it makes implementation of lexical analysis and syntax analysis easier.
    (b) syntax-directed translations can be written for intermediate code generation.
    (c) it enhances the portability of the front end of the compiler.
    (d) it is not possible to generate code for real machines directly from high level language programs.

11. Which of the following cannot be used as an intermediate code form?
(a) Post fix notation (b) Three address codes
(c) Syntax trees (d) Quadruples

12. A shift-reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar

$S \rightarrow AS$ {print "1"}
$S \rightarrow AB$ {print "2"}
$A \rightarrow a$ {print "3"}
$B \rightarrow bC$ {print "4"}
$B \rightarrow dB$ {print "5"}
$C \rightarrow c$ {print "6"}

This syntax directed translation scheme translates a language whose terminal symbols are *a*, *b*, *c*, and *d* into another language whose terminal symbols are 1, 2 3, 4, 5 and 6. What is the translation of "*aaadbc*"?
(a) 333546 (b) 654211
(c) 333645211 (d) 645233311

13. A shift reduce parser carries out the actions specified with in braces immediately after reducing the corresponding rule of grammar.

$A \rightarrow bbB$ {print "+"}
$A \rightarrow a$ {print "*x*"}
$B \rightarrow Ac$ {print "–"}

What is the translation of *bbbbacc* using the syntax directed translation scheme described by the above rules?
(a) * + – + * (b) * * – + –
(c) + * – + * (d) * – + – +

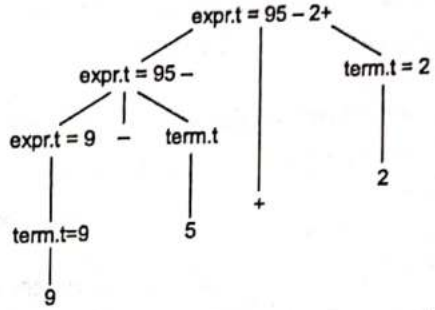
14. Consider the grammar with the following translation rules and E as the start symbol.

| | |
|---|---|
| $E \rightarrow E_1 \# T$ | {*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

15. 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 the above

16. Consider annotated parse tree.

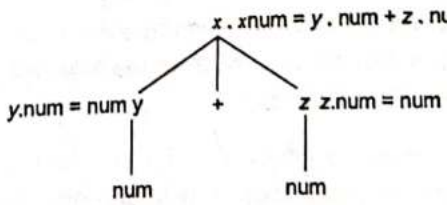



Which one of the following is correct syntax directed definition of above parse tree?

| Production | Semantic Rule |
|---|---|
| (a) expr → expr+term | expr.t=expr.t‖term.t‖'*' |
| expr → expr+term | expr.t=expr.t‖ term.t‖'+' |
| expr → term | expr.t = term.t |
| term → 9 | term.t → '9' |
| term → 5 | term.t → '5' |
| term → 2 | term.t → '2' |
| (b) expr → expr + term | expr.t = expr.t \|\| term.t \|\| '+' |
| expr → expr – term | expr.t = expr.t \|\| term.t \|\| '–' |
| expr → term | expr = term |
| term → 9 | term.t → '9' |
| term → 5 | term.t → '5' |
| term → 2 | term.t → '2' |

(c) both (a) and (b)
(d) none of the above

17. Consider the following annotated parse tree.



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

(a) There is specific order for evaluation of attribute on parse tree
(b) Any evaluation order that computes an attribute $x$ after all other attribute that $x$ depends on is acceptable
(c) Both (a) and (b)
(d) None of the above

**18.** Consider the following grammar with corresponding synthesized attributes.

$F \rightarrow .L$ {F.v = L.v}
$L \rightarrow L\,B$ {L.len = $L_1$.len+1, L.v = $L_1.v + 2^{-L.len} \times B.v$}
$L \rightarrow B$ {L.len = 1, L.v = B.v/2}
$B \rightarrow 0$ {B.v = 0}
$B \rightarrow 1$ {B.v = 1}

If "F.val" gives the value of the binary fraction generated by F in the above grammar then what will be the value of F.val on input .101?

(a) .625 (b) .550
(c) .710 (d) .485

**19.** If the expression 8 #12 & 4 #16 & 12 #4 & 2 is evaluated to 512, then which one of the following correctly represent the rule III?

| Production rule | Translation rule | |
|---|---|---|
| $E \rightarrow E \# T$ | {E.val = $E_1$.val * T.val} | ...I |
| $\mid T$ | {E.val = T.val} | ...II |
| $T \rightarrow T \& F$ | _____ | ...III |
| $\mid F$ | {T.val = F.val} | ...IV |
| $F \rightarrow$ num | {F.val = num} | ...V |

(a) T.val = $T_1$.val * F.val
(b) T.val = $T_1$.val + F.val
(c) T.val = $T_1$.val – F.val
(d) None of these

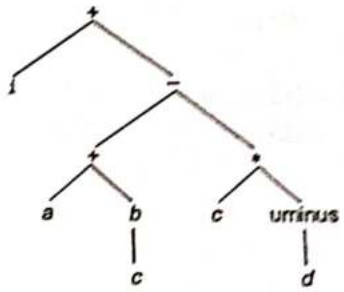
**20.** A synthesized attribute is an attribute whose value at a parse tree node depends on

(a) attributes at the siblings only
(b) attributes at parent node only
(c) attributes at children nodes only
(d) none of the above

**21.** An inherited attribute is the one whose initial value at a parse tree node is defined in terms of

(a) attributes at the parent and/or siblings of that node
(b) attributes at children nodes only
(c) attributes at both children nodes and parent and/or siblings of that node
(d) none of the above

**22.** The intermediate code generated for the following syntax tree is




(a) iac + cdb + – +
(b) iacb + cd uminus * – +
(c) iacb + cd * – +
(d) None of these

**23.** Generation of intermediate code based on an abstract machine model is useful in compilers because

(a) it makes implementation of lexical analysis and syntax analysis easier
(b) syntax-directed translations can be written for intermediate code generation
(c) it enhances the portability of the front end of the compiler
(d) it is not possible to generate code for real machines directly from high level language programs

**24.** 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) none of the above

**25.** Consider the intermediate code given below:

1. $i = 1$;
2. If $i \le n$ goto (4);
3. goto (8);
4. $t_1 = b + c$;

5. $a = t_1$;
6. $a = t_1 * d$;
7. $i = i + 1$ goto (2);
8. end

Which one of the following is correctly represents the above code?

(a)
```
for (i = 1; i ≤ n; i++)
{
    a = b + c * d;
}
```

(b)
```
for (i = 1; i ≤ n;)
{
    a = b + c + d;
    i = i + 2;
}
```

(c)
```
for (i = 1; i ≤ n; i++)
{
    a = b + c;
    a = a * d;
}
```

(d) None of these

**26.** Consider the following SDT.

| | |
|---|---|
| $E \to XY$ | $\{Y.a = X.a\}$ |
| $E \to X\,U\,V\,Y$ | $\{V.a = X.a + U.a\}$ |
| $Y \to 5$ | $\{Y.a = 5\}$ |
| $V \to \epsilon$ | $\{V.a = 0\}$ |
| $U \to \epsilon$ | $\{U.a = 0\}$ |
| $X \to \epsilon$ | $\{X.a = 0\}$ |

The given SDT is _____.

(a) L-attributed grammar only
(b) S-attributed only
(c) Both S-attributed and L-attributed
(d) None of these

**27.** An expression '*e*' evaluated at point P is said to be available at point Q if

(a) *e* is not used between P and Q
(b) any of its operands is redefined between P and Q
(c) any of its operands is not modified between P and Q
(d) None of these

**28.** Let G be a grammar is used for arithmetic expressions. The grammar G is shown below with semantic actions and attribute "sign" can contain either 0 or 1.

$E \to E_1 + E_2$ {*E*.sign = $E_2$.sign}
$E \to E_1 \times (E_2)$ {*E*.sign = $E_1$. sign × $E_2$.sign}
$E \to E_1/E_2$ {if ($E_1$.sign == 0) then *E*.sign = 1 else *E*.sign = 0}
$E \to +E_1$ {*E*.sign = 0}
$E \to -E_1$ {*E*.sign = 1}
$E \to id$ {*E*.sign = 0}

[*Note:* $E \to E_1 + E_2$ is same as $E \to E + E$]

Find the attribute value at the root *E* for the given input: –id × (–id + id)

(a) 0 (b) 1
(c) 2 (d) None of these

**29.** Consider the following syntax directed definition shown below:

$E \to E + T$ {*E*.val = *E*.val + *T*.val +1}
$E \to T$ {*E*.val = *T*.val +1}
$T \to T * F$ {*T*.val = *T*.val + *F*.val +1}
$T \to F$ {*T*.val = *F*.val +1}
$F \to$ id {*F*.val = 1}

For an input $a + b * c + d$ this translation scheme will give output as:

(a) 8 (b) 9
(c) 10 (d) 11

**30.** Consider the following SDT.

| | |
|---|---|
| $L \to E$ | {*L*.val = *E*.val} |
| $E \to E + T$ | {*E*.val = $E_1$.val + *T*.val} |
| $E \to T$ | {*E*.val = *T*.val} |
| $T \to T * F$ | {*T*.val = $T_1$.val * *F*.val} |
| $T \to F$ | {*T*.val = *F*.val} |
| $F \to (E)$ | {*F*.val = *E*.val} |
| $F \to a$ | {*F*.val = *a*} |

What is the output of the expression "10 + 4 * 6"?

(a) 29 (b) 32
(c) 34 (d) 28

**31.** Consider the following production, along with the syntax directed definition.

$A \to QR$,
*Q*.in = *f*.(*A*.*s*) || 1st rule
*R*.in = *f*.(*Q*.*s*) || 2nd rule

Where .in and .s have their regular meaning i.e., inherited and synthesized attributes respectively. Which of the following option is true?

(a) Syntax directed definition are *L*-attributed.
(b) Syntax directed definition is not *L*-attributed because of 2nd rule.
(c) Syntax directed definition is not *L*-attributed because of 1st rule.
(d) Syntax directed definition is not *L*-attributed because of both rule.

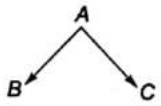
■■■■

## Answers Syntax Directed Translation and Intermediate Code Generation

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

## Explanations Syntax Directed Translation and Intermediate Code Generation

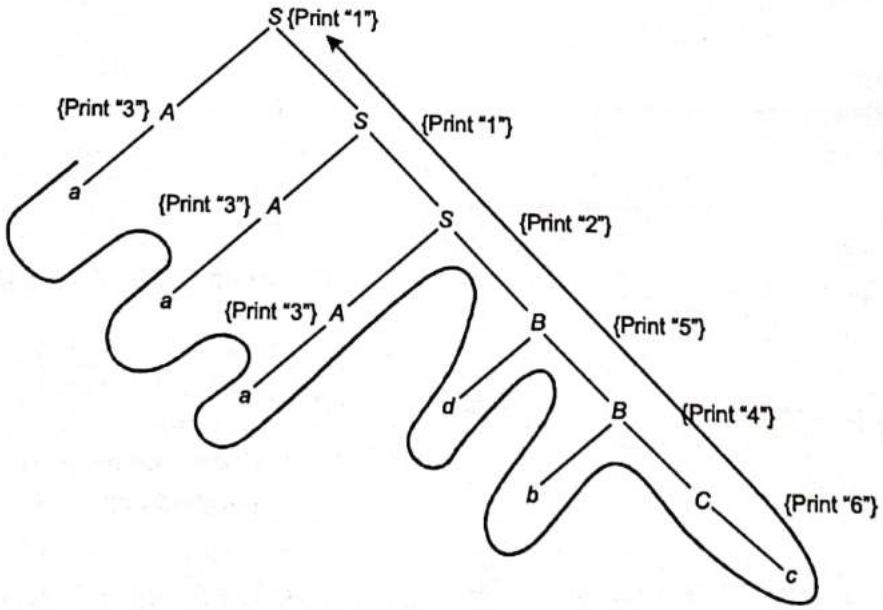
**9. (b)**




According to *L* attributed definition, *B*'s attributes can depend on inherited attributes of *A* but not on synthesized attributes of *A*.

**12. (c)**

The syntax directed tree for the given grammar can be represented as for given input "***aaadbc***"

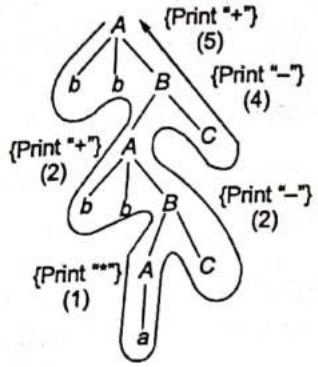



Output printed will be : 3 3 3 6 4 5 2 1 1.

13. (d)

$A \rightarrow bbB$ {Print "+"}
$A \rightarrow a$ {Print "*"}
$B \rightarrow Ac$ {Print "–"}

Input:
*bbbbacc*



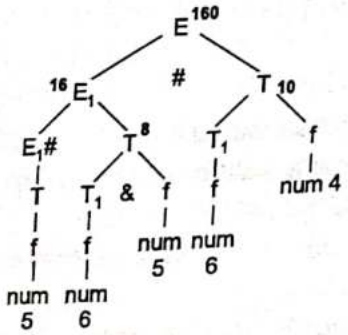

The above syntax tree prints

* – + – +

Hence, (d) is correct option.

14. (c)

2 # 3 & 5 # 6 & 4

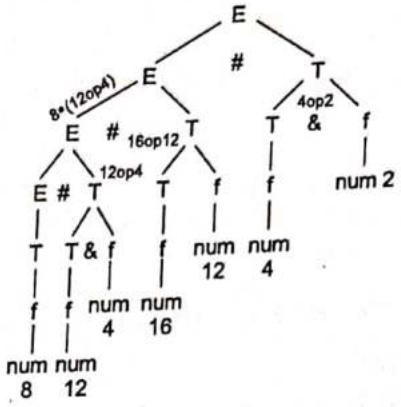



1. 3 & 5 = 3 + 5 = 8
2. 6 & 4 = 6 + 4 = 10
3. 2 # **8** (from 1) = 2 × 8 = 16
4. **16** (from 3) # **10** (from 2) = 16 × 10 = 160

So, option (c) is correct.

19. (c)

8 # 12 & 4 # 16 & 12 # 4 & 2 = 512




Let the rule be T.val=$T_1$.val op f.val

1. 12 & 4 = 12 op 4
2. 8 # (12 & 4) = 8*(12 op 4)
3. 16 & 12 = 16 op 12
4. 4 & 2 = 4 op 2
5. 8 # 12 & 4 & 16 & 12 = (8*(12 op 4))*(16 op 12)
6. 8 # 12 & 4 & 16 & 12 # 4 & 2 = ((8*(12 op 4))*(16 op 12))*(4 op 2)

Now, ((8*(12 op 4))*(16 op 12)*(4 op 2)) = 512

if op = '–'

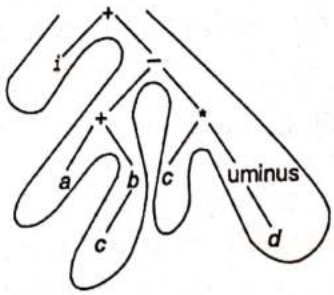
((8 * (12 – 4)) * ((16 – 12))*(4 – 2))

= 8 × 8 × 4 × 2 = 256 × 2 = 512

22. (b)

The intermediate code generated is

*iacb* + *cd* uminus * – +



25. (c)

```
for (i = 1; i ≤ n ; i ++)        i = 1;
{                                if i ≤ n then
 a = b + c;              ⟹          t1 = b + c;
 a = a * d;                         a = t1;
}                                   a = t1 * d;
                                    i = i + 1 goto (2)
                                 else
                                     end
```

Intermediate code represent option (c).

26. (a)

SDT is L-attributed (i.e., restricted inherited attributes and synthesized attributes).

$Y.a = X.a$ rule compute using left sibling.

So not S-attributed.

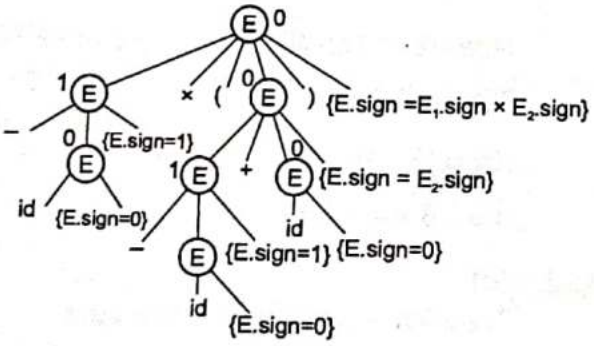
27. (c)

*Example:* $e : a + b$

If $a$ or $b$ is modified then the expression is not said to be available.

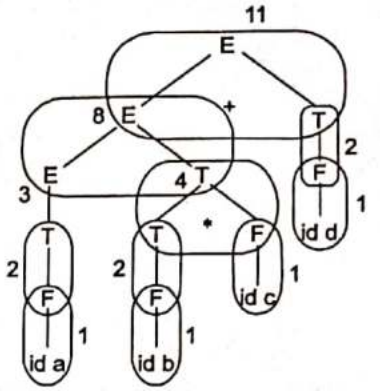
28. (a)

Input: –id × (–id + id)



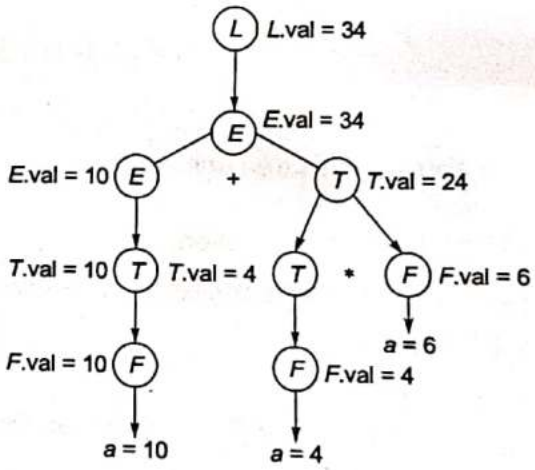

The value computed at root is "0".

29. (d)




The given SDT gives total number of reductions for the input which is 11.

30. (c)




31. (c)

SDD is L-attributed iff:

The inherited attributes of $Y_i$ in the production.

$X \rightarrow Y_1 Y_2 \ldots Y_i Y_{i+1} \ldots Y_j$

- Should only be derived from only attributes $Y_1$ to $Y_{i-1}$.
- Should only be derived from inherited attribute of $X$.

Hence in first production rule $Q$ is being derived from synthesized attribute of $A$. Hence SDD is not L-attribute because of 1's rule.

■■■■

# 5 CHAPTER
# Run-Time Environment and Target Code Generation

1. Heap allocation is required for languages
   (a) that support recursion
   (b) that support dynamic data structures
   (c) that use dynamic scope rules
   (d) none of these

2. Access time of the symbol table will be logarithmic, if it is implemented by
   (a) linear list (b) search tree
   (c) hash table (d) self-organization list

3. Which one is not a storage-allocation strategy?
   (a) Static (b) Symbolic
   (c) Stack (d) Heap

4. In symbol table, each string is stored in
   (a) a separate array (b) a single array
   (c) a hash table (d) a stack

5. The function of the storage assignment is
   (a) to assign storage to all variables referenced in the source program
   (b) to assign storage to all temporary locations that are necessary for intermediate results
   (c) to assign storage to literals, and to ensure that the storage is allocated and appropriate locations are initialised
   (d) all of the above

6. Which of the following symbol table implementation is based on the property of locality of reference?
   (a) Hash table (b) Linear list
   (c) Search tree (d) Self organization list

7. Consider the following statements:
   $S_1$: Static allocation bindings do not change at runtime.
   $S_2$: Heap allocation allocates and de-allocates storage at run time.
   Which of the above statements is/are true?
   (a) $S_1$ is true and $S_2$ is false
   (b) $S_2$ is true and $S_1$ is false
   (c) Both are true
   (d) Both are false

8. Consider the following program
```
main ( )
{
    int* p;
    p = X( );
}
    int * X( )
{   int x = 20;
    return &x;
}
```
   Which of the following is true?
   (a) The program is an example of alias
   (b) The program will generate dangling reference
   (c) The program will allocate variable in heap
   (d) None of the above

9. 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 the above

10. The term "environment" in programming language semantics is said as
    (a) function that maps a name to value held there
    (b) function that maps a name to a storage location
    (c) the function that maps a storage location to the value held there
    (d) none of the above

11. Symbol table can be used for
(a) Checking type compatibility
(b) suppressing duplication of error messages
(c) storage allocation
(d) all of the above

12. Which of the following are true?
1. A programming language which does not permit global variables of any kind and has no nesting of procedures/functions, but permits recursion can be implemented with static storage allocation.
2. Multi-level access link (or display) arrangement is needed to arrange activation records only if the programming language being implemented has nesting of procedures/function.
3. Recursion in programming languages cannot be implemented with dynamic storage allocation.
4. Nesting of procedures/functions and recursion require a dynamic heap allocation scheme and cannot be implemented with a stack-based allocation scheme for activation records.
5. Programming languages which permit a function to return a function as its result cannot be implemented with a stack-based storage allocation scheme for activation records.

(a) 2 and 5 only (b) 1, 3 and 4 only
(c) 1, 2 and 5 only (d) 2, 3 and 5 only

13. Consider the following procedure declaration.

```
procedure p;
x : integer;
procedure q;
begin x : = x + 1 end;
procedure r;
x : integer;
begin x : = 1; q; write(x) end;
begin
x : = 2;
r;
end;
```

The output produced by calling *P* in a language with dynamic scope will be
(a) 1 (b) 2
(c) 3 (d) 4

14. Choose the false statements:
1. Control stack keeps track of live procedure activations.
2. Activation records can be managed with the help of stack.
3. Dangling reference is a reference to a storage that has been deallocated.

(a) 1 and 2 (b) 2 and 3
(c) 1 and 3 (d) None of these

15. Hash tables can contribute to an efficient average case solution for all of the following problems described below except
(a) counting distinct values
(b) dynamic dictionary
(c) range search
(d) symbol table lookup

16. Which of the following is content of activation record?
(a) Control link
(b) Access link
(c) Temporary variable
(d) All of these

17. Which of the following is correct?
(a) The drawback with static storage allocation is it does not support recursion.
(b) The drawback with stack storage allocation is when function complete its execution it will be poped out from stack.
(c) Both (a) and (b)
(d) None of the above

18. Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

**List-I**
A. Static code
B. Stack local variables and parameters
C. Heap variables allocated

List-II

1. Dynamically-resizable strings or arrays; local variable with unlimited extent.
2. Local variables in Fortran; compiler-generated tables for debugging, exception handling, garbage collection.
3. Return addresses; temporary values (spilled registers)

Codes:

| | A | B | C |
|---|---|---|---|
| (a) | 2 | 3 | 1 |
| (b) | 1 | 2 | 3 |
| (c) | 3 | 1 | 2 |
| (d) | 1 | 3 | 2 |

19. Let P be a procedure that for some inputs call itself (i.e. recursive). If P is guaranteed to terminate, which of the following must be true?
1. P has local variable
2. P has an execution path where it does not call itself
3. P either refers to a global variable or has atleast one parameter

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

20. Three address codes can be implemented by
(a) Indirect triples
(b) Direct triples
(c) Both (a) and (b)
(d) None of these

21. Which of the following optimization techniques is typically not applied on loops?
(a) Reduction in strength
(b) Removal of invariant computation
(c) Constant folding
(d) Elimination of induction variables

22. Reduction in strength means
(a) replacing run time computation by compile time computation
(b) removing loop invariant computation
(c) removing common sub-expressions
(d) replacing a costly operation by a relatively cheaper one

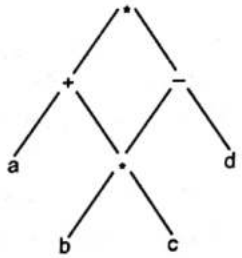
23. Peephole optimization is a form of
(a) loop optimization (b) local optimization
(c) constant folding (d) data flow analysis

24. Consider the syntax directed definition shown below.

$S \rightarrow id := E$ {gen (id.place = E.place;);}
$E \rightarrow E_1 + E_2$ {t = newtemp ( );
gen ($t = E_1$.place + $E_2$.place;);
E.place = $t$}
$E \rightarrow id$ {E.place = id.place;}

Here, gen is a function that generates the output code, and newtemp is a function that returns the name of a new temporary variable on every call. Assume that $t_1$'s are the temporary variable names generated by newtemp. For the statement 'X: Y + Z; the 3-address code sequence generated by this definition is
(a) X = Y + Z
(b) $t_1$ = Y + Z; X $t_1$
(c) $t_1$ = Y; $t_2 = t_1$ + Z; X = $t_2$
(d) $t_1$ = Y; $t_2$ = Z; $t_3 = t_1 + t_2$; X = $t_3$

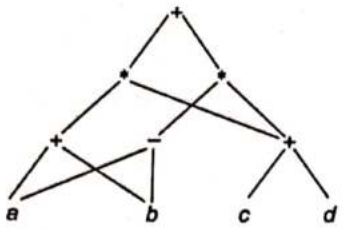
25. The DAG shown here represents:




(a) a + (b * c) – d
(b) a + b * c * b * c – d
(c) (a + b) * (c – d)
(d) (a + b * c) * (b * c – d)

26. Any Directed Acyclic Graph must have
(a) atleast one vertex with indegree 1 and atleast one vertex with outdegree 0
(b) atleast one vertex with indegree 0 or atleast one vertex with outdegree 0
(c) atleast one vertex with indegree 0 and atleast one vertex with outdegree 0
(d) atleast one vertex with indegree 0 and atleast one vertex with outdegree 1

27. The DAG shown here represents:




(a) (a – b) * (a + b) + (c + d) * (c + d)
(b) (a – b) * (c + d) + (a – b) * (c – d)
(c) (a * b) + (c * d) * (a * b) + (c * d)
(d) (a + b) * (c + d) + (a – b) * (c + d)

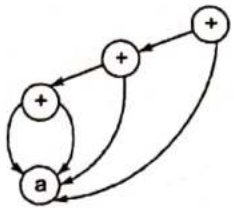
28. A directed acyclic graph represents one form of intermediate representation. The number of non terminals nodes in DAG of a = (b + c) * (b + c) expression is
(a) 2 (b) 3
(c) 4 (d) 5

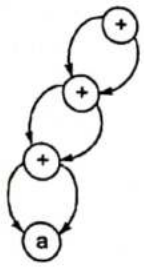
29. Which of the following is not a type of three address statements?
(a) copy statements
(b) index assignment statements
(c) pointer assignments
(d) none of the above

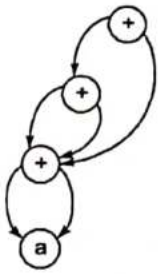
30. Consider the following statements.
$S_1$: Three address code is a linearized representation of a syntax tree.
$S_2$: The syntax tree not depicts the natural hierarchical structure of source program
Which of the above statements is/are true?
(a) $S_1$ is true $S_2$ is false
(b) $S_1$ is false $S_2$ is true
(c) Both $S_1$ and $S_2$ are true
(d) Both $S_1$ and $S_2$ are false

31. Some code optimizations are carried out on the intermediate code because
(a) they enhance the portability of the complier to other target processors
(b) program analysis is more accurate on intermediate code than on machine code
(c) the information from data flow analysis cannot otherwise be used for optimization
(d) the information from the front end cannot otherwise be used for optimization

32. Which of the following is correct directed acyclic graph for the expression.
$a + a + a + a + a + a$

(a)



(b)



(c)



(d) Both (a) and (c)

33. Which of the following statement is/are true?
$S_1$: First statement of three address code is leader.
$S_2$: Every target of goto statement is leader.
$S_3$: Every statement after goto is leader.
(a) $S_1$ only (b) $S_2$ only
(c) $S_1$ and $S_3$ only (d) $S_1$, $S_2$ and $S_3$ only

34. Which of the following is correct?
(a) The drawback for quarduple representation of 3-address code is it requires more space compared to triple notation.
(b) The advantage for triple representation of 3-address code is it requires less space compared to quarduple notation.
(c) Both (a) and (b) are correct.
(d) Neither (a) nor (b) are correct.

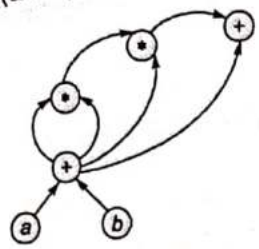
35. Consider the following expression :
$a + b/9 + c - d * 4 + e$
The minimum number of temporary variables required to create a three-address code in static single assignment form for the above expression are _____.

36. Consider the following expression.
$((x + y) - ((x + y) * (x - y))) + ((x + y) * (x - y))$
The number of nodes to represent the DAG for above expression is _____.

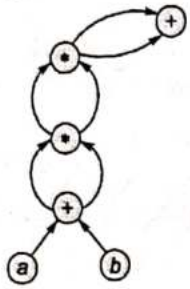
37. Which of the following represents DAG for the following expression correctly?

$[(a + b) * (a + b) * (a + b) + (a + b)]$

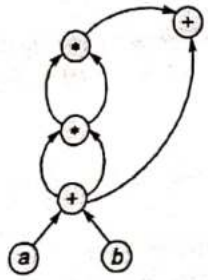
(a)



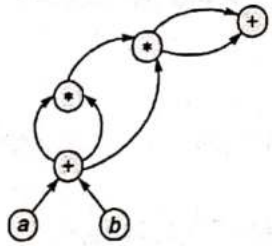
(b)



(c)



(d)



38. Consider the following 3-address code.

$t_1 = a + b$
$t_2 = c + d$
$t_3 = t_1 * t_2$
$t_4 = t_2 + t_2$
$t_5 = t_4 + t_3$

There are 5 temporary variables in above code. What is the minimum number of temporary variables can be used in the equivalent optimized 3-address code of above code?

(a) 2 (b) 3
(c) 4 (d) 1

39. Consider the basic block given below:

$b = (b + c)$
$d = b * d$
$b = d - b$

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

(a) 4 and 5 (b) 5 and 4
(c) 6 and 6 (d) 6 and 7

■■■■

**Answers** Run-Time Environment and Target Code Generation

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | (b) | 2. | (b) | 3. | (b) | 4. | (a) | 5. | (d) | 6. | (d) | 7. | (c) | 8. | (b) | 9. | (b) |
| 10. | (b) | 11. | (d) | 12. | (d) | 13. | (b) | 14. | (d) | 15. | (c) | 16. | (d) | 17. | (c) | 18. | (a) |
| 19. | (d) | 20. | (c) | 21. | (c) | 22. | (d) | 23. | (b) | 24. | (b) | 25. | (d) | 26. | (c) | 27. | (d) |
| 28. | (b) | 29. | (d) | 30. | (a) | 31. | (a) | 32. | (c) | 33. | (d) | 34. | (c) | 37. | (a) | 38. | (a) |
| 39. | (c) | | | | | | | | | | | | | | | | |

## Explanations Run-Time Environment and Target Code Generation

**16. (d)**

Control link points to the activation record of the caller.

Access link points to the activation record associated with nearest enclosing scope of the subprogram definition.

So, control link, access link and temporary variable are part of activation record.

**17. (c)**

Static storage allocation does not support recursion because memory will be allocated at compile time itself and at compile time we don't know how much memory is required. So it is the drawback.

In stack allocation when one function completes it execution then it will be poped out from stack. If in near future again that function called it will be evaluated again. So it consume lots of time to evaluate same function again and again. So it is the drawback.

**18. (a)**

Static code; global variables; constants; own (static) variables; local variables in Fortran; compiler-generated tables for debugging, exception handling, garbage collection, etc.

Stack local variables and parameters in most languages; saved registers; return addresses; temporary values (spilled registers) heap variables allocated; dynamically-resizable strings or arrays; local variables with unlimited extent.

**19. (d)**

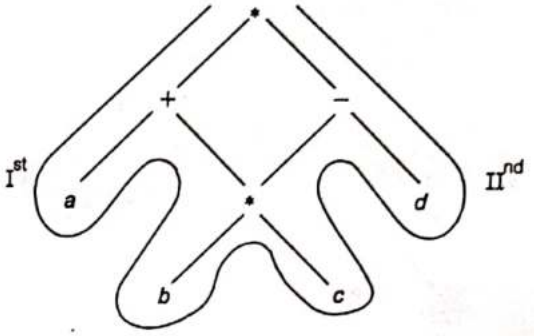
- A recursive procedure may or may not have a local variable. So 1 is false.
- All recursive procedures have an execution path where they do not call themselves. So 2 is correct
- A recursive procedure terminates only its parameter or a global variable is being tested. So true.

**20. (c)**

Three address code is represented in three when

1. Quadruple
2. Triples
3. Indirect triples

**25. (d)**



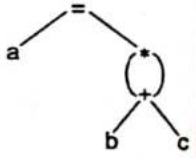

The above DAG represents

$(a + b * C) * (b * C - d)$

**28. (b)**

For a = $(b + c)*(b + c)$

DAG will be like:



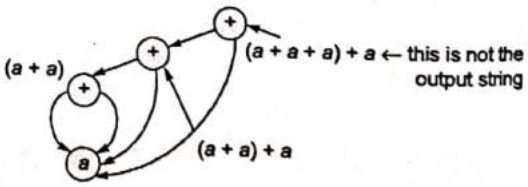

∴ Number of non-terminal nodes = 3. (=, *,+)

**32. (c)**

$a + a + a + a + a + a$

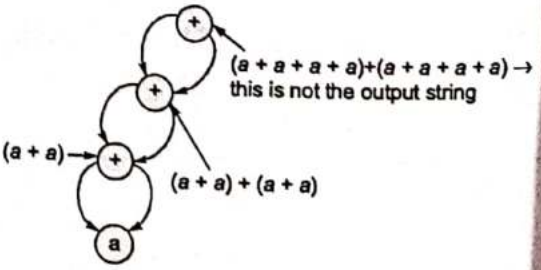
Consider the options:

(a)



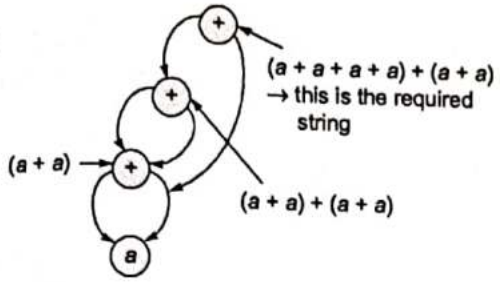

So option (a) is wrong.

(b)




So option (b) is wrong

(c)




So option (c) is right.

33. (d)
The all of three-statement is true about the leader of basic block in flow graph of code optimization technique.

34. (c)
The drawback in quarduple representation is one extra field required to store the result.
In triple representation there is no need of extra field to store the result, so it requires less space.
Both (a) and (b) are correct.

35. (6)
In static single assignment, every variable assigned only once and that variable can be used any number of times without assignment.
**Expression :** $a + b/9 + c - d * 4 + e$

$t_1 = b/9;$

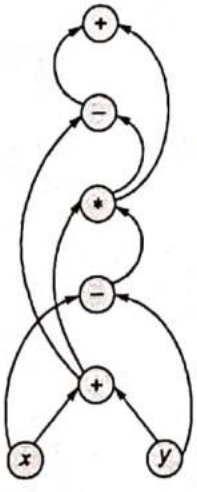
$t_2 = a + b/9;$ $\quad t_3 = t_2 + c$

$t_4 = d * 4$ $\quad t_5 = t_3 - t_4$

$t_6 = t_5 + e$

∴ 6 temporary variables are required.

36. (7)




37. (a)
In the DAG of given expression term "$a + b$" will be evaluated only one time and remaining time this expression will be used.

38. (a)

$$\left.\begin{aligned} t_1 &= a+b \\ t_2 &= c+d \\ t_3 &= t_1 * t_2 \\ t_4 &= t_2 + t_2 \\ t_5 &= t_4 + t_3 \end{aligned}\right\} \Rightarrow \begin{aligned} t_1 &= a+b \\ t_2 &= c+d \\ t_1 &= t_1 * t_2 \\ t_2 &= t_2 + t_2 \\ t_1 &= t_1 + t_2 \end{aligned}$$

Above code represents the following expression:

$((c + d) + (c + d)) + ((a + b) \times (c + d))$

∴ Only two temporary variables are used.

39. (c)

$b = (b + c)$

$d = (b + c) * d$

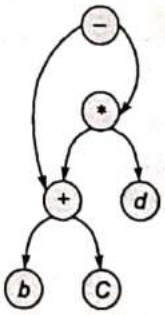
$d = b * d$

$b = (b + c) * d - (b + c)$

Final expresssion is

$\Rightarrow \quad b = (b + c) * d - (b + c)$

So, DAG representation for above expression is:




Number of nodes = 6
Number of edges = 6

■■■■