Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Hymavati Ratul Divyanshu & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Compiler design PYQs with Solutions

(Lexical analysis)

Question 1: [**GATE CS 2011**]

The lexical analysis for a modern computer language such as Java needs the power of which one of the following machine models in a necessary and sufficient sense?

A) Finite state automata

B) Deterministic pushdown automata

C) Non-Deterministic pushdown automata

D) Turing Machine

Answer: A

Explanation: In compiler, lexical analyzer categorizes the character sequence into lexemes and produces tokens as output for parser. And tokens are expressed in regular expressions so a simple Finite Automata is sufficient for it.

Question 2: **[GATE CS 2018]**

Which one of the following statements is FALSE?

A)Context-free grammar can be used to specify both lexical and syntax rules.

B)Type checking is done before parsing.

C)High-level language programs can be translated to different Intermediate Representations.

D)Arguments to a function can be passed using the program stack.

Answer: B

Explanation: Type checking is done at semantic analysis phase and parsing is done at syntax analysis phase. And we know Syntax analysis phase comes before semantic analysis. So Option **(B) is False**. All other options seems Correct.

Question 3: **[GATE CS 2009]**

Match all items in Group 1 with correct options from those given in Group 2.

Group 1:-

P. Regular expression

Q. Pushdown automata

R. Dataflow analysis

S. Register allocation

Group 2:-

1. Syntax analysis

2. Code generation

3. Lexical analysis

4. Code optimization

A)P-4. Q-1, R-2, S-3

B)P-3, Q-1, R-4, S-2

C)P-3, Q-4, R-1, S-2

D)P-2, Q-1, R-4, S-3

Answer: B

Question 4: [**GATE CS 2000**]

The number of tokens in the following C statement is:

```
printf("i = %d, &i = %x",i, &i);
```

A) **3**

B) 26

C) 10

D) 21

Answer: C

Question 5: [**GATE CS 2011**]

Consider the following statements:
(1) The output of the lexical analyzer is groups of characters.
(2) Total tokens count in the printf("pt=%d, &pt=%x", pt, &pt); are 11.
(3) Symbol table can be implemented with an array and hash table but not a tree.
Which of the given statement(s) is/are correct?

A) Only (1).

B) Only (2) and (3).

C) Only (1), (2), and (3).

D) None of the above.

Solution: (D).

Explanation: The correct statement is as follows
(1) The output of the lexical analyzer is tokens.
(2) Total tokens count in printf("pt=%d, &pt=%x", pt, &pt); are 10.
(3) We can implement a symbol table with an array, hash table, tree, and linked lists.
So, option (D) is correct.

Question 6: [GATE CS 2018]

The lexical analyzer uses the given patterns for recognizing three tokens, T1, T2, and T3, over the alphabets a,b,c. T1: a?(b|c)*a & T2: b?(a|c)*b & T3: c?(b|a)*c. Note: 'x?' means 1 or 0 occurrences of the symbol x. Also, the analyzer outputs the token matching the longest possible prefix. If the analyzer processes the string "bbaacabc", which of the below is the sequence of tokens it outputs?

A) T1T2T3
B) T1T1T3
C) T2T1T3
D) T3T3

Solution: (D).

Explanation: We can rewrite the tokens as: T1 : (b+c)* a + a(b+c)* a and T2 : (a+c)* b + b(a+c)* b and T3 : (b+a)* c + c(b+a)* c.
The given String is "bbaacabc" the longest matching prefix is "bbaac" (That may be generated by T3). The remaining part "abc" can again be generated by T3. Hence, the answer is T3T3.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 7: [**GATE CS 2015**]

Match the following:

P.	Lexical analysis	1.	Graph coloring
Q.	Parsing	2.	DFA minimization
R.	Register allocation	3.	Post-order traversal
S.	Expression evaluation	4.	Production tree

A. P-2, Q-3, R-1, S-4
B. P-2, Q-1, R-4, S-3
C. P-2, Q-4, R-1, S-3
D. P-2, Q-3, R-4, S-1

Answer: C

1. Regular expression uses FA & Regular Sets.

2. Expression can be evaluated with postfix Traversals.

3. Register allocation can be modeled by graph coloring.

4. The parser constructs a production tree.

So, answer is C.

Question 8: [**GATE CS 2018]**

Consider the following statements: (I) The output of a lexical analyzer is groups of characters. (II) Total number of tokens in printf("i=%d, &i=%x", i, &i); are 11. (III) Symbol table can be implementation by using array and hash table but not tree. Which of the following statement(s) is/are correct?

A)Only (I)

B)Only (II) and (III)

C)All (I), (II), and (III)

D)None of these

Answer: D

Explanaiton: (I) The output of a lexical analyzer is tokens. (II) Total number of tokens in printf("i=%d, &i=%x", i, &i); are

10. (III) Symbol table can be implementation by using array, hash table, tree and linked lists. So, option (D) is correct.

Question 9: [**GATE CS 2011**]

In a compiler, keywords of a language are recognized during

A)parsing of the program

B)the code generation

C)the lexical analysis of the program

D)dataflow analysis

Answer: C

Explanation: Typically, the lexical analysis phase of compilation breaks the input text up into sequences of lexemes that each belongs to some particular token type that's useful in later analysis. Consequently, keywords are usually first recognized during lexical analysis in order to make parsing easier. Since parsers tend to be implemented by writing context-free grammars of tokens rather than of lexemes (that is, the *category* of the lexeme rather than the *contents* of the lexeme), it is significantly easier to build a parser when keywords are marked during lexing. Any identifier is also a token so it is recognized in lexical Analysis .
Hence, option **C** is True.

Question 10: [**GATE CS mock**]

The number of tokens in the following C statement is

```
printf("HELLO WORLD");
```

A) 3

B) 5

C) 9

D) 8

Answer: B

Explanation:

```
1 printf
2 (
3 "HELLO WORLD"
4 )
5 ;
```

Question 11: [**GATE CS 1998**]

Type checking is normally done during

A)Lexical analysis

B)syntax analysis

C)Syntax directed translation

D)Code optimization

Answer: C

Question 12: [**GATE CS 1998**]

Consider the following ANSI C Program

```
int main () {
  Integer x;
  return 0;
}
```

Which one of the following phases in a seven-phase of C compiler will throw an error?

A)Lexical analyzer

B)Syntax analyzer

C)Semantic analyzer

D)Machine dependent optimizer

Answer: B

Question 13: [**GATE CS 1998**]

In compiler optimization, operator strength reduction uses mathematical identities to replace slow math operations with faster operations. Which of the following code replacements is an illustration of operator strength reduction ?

A)Replace P + P by 2 * P or Replace 3 + 4 by 7.

B)Replace P * 32 by P < < 5

C)Replace P * 0 by 0

D)Replace (P < <4) – P by P * 15

Answer: B

Explanation: In option (B), Multiplication operation is replaced with Shift Operator . It reduce the operator strength because shift operator is less expensive than multiplication operation. So option (B) is correct.

Question 14: [**GATE CS 1998**]

The number of tokens in the following C statement is printf("i=%d, &i=%x", i,&i);

A)3

B)6

C)10

D)9

Answer: C

Explanation:

```
printf ( "i=%d, &i=%x" , i , & i ) ;

          1    2      3        4 5 6 7 8 9 10
```

Question 15: [**GATE CS 1995**]

In some programming languages, an identifier is permitted to be a letter following by any number of letters or digits. If L and D denotes the sets of letters and digits respectively, which of the following expressions defines an identifier?

A

A)(L U D)+

B)L(L U D)*

C)(L.D)*

D)L(L.D)*

Answer: B

Explanation: Identifier has to be started by a letter followed by any number of letters (or) digits. Hence option B is Correct

(Syntax Analysis
[and]
Semantic analysis)

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 16: [**GATE CS 2013**]

What is the maximum number of reduce moves that can be taken by a bottom-up parser for a grammar with no epsilon- and unit-production (i.e., of type A -> ϵ and A -> a) to parse a string with n tokens?

A) n/2

B) n-1

C) 2n-1

D)2^n

Answer: B

Explanation:Given in the question, a grammar with no epsilon- and unit-production (i.e., of type A -> ϵ and A -> a). To get maximum number of Reduce moves, we should make sure than in each sentential form only one terminal is reduced. Since there is no unit production, so last 2 tokens will take only 1 move. So To Reduce input string of n tokens, first Reduce n-2 tokens using n-2 reduce moves and then Reduce last 2 tokens using production which has . So total of n-2+1 = n-1 Reduce moves.Suppose the string is abcd. (n = 4).

We can write the grammar which accepts this string as follows: `s->aB`

`B->bC`

`C->cd`

The Right Most Derivation for the above is:

```
S -> aB  ( Reduction 3 )
-> abC  ( Reduction 2 )
  -> abcd  ( Reduction 1 )
```

We can see here that no production is for unit or epsilon. Hence 3 reductions here. We can get less number of reductions with some other grammar which also doesn't produce unit or epsilon productions,

`S->abA`

`A-> cd`

The Right Most Derivation for the above as:

```
S -> abA  ( Reduction 2 )
  -> abcd  ( Reduction 1 )
```

Hence 2 reductions.

But we are interested in knowing the maximum number of reductions which comes from the 1st grammar. Hence total 3 reductions is maximum, which is (n - 1) as n = 4 here.

Thus, Option B.

Question17:[GATE CS 2009]

Which of the following statements are TRUE?

I. There exist parsing algorithms for some programming languages whose complexities are less than O(n3).

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

Answer:B

Explanation: II is false, in recursion, compiler cannot determine the space needed for recursive calls. III is false.

Question 18: [GATE CS 2008]

Which of the following describes a handle (as applicable to LR-parsing) appropriately?

A)It is the position in a sentential form where the next shift or reduce operation will occur

B)It is non-terminal whose production will be used for reduction in the next step

C)It is a production that may be used for reduction in a future step along with a position in the sentential form where the next shift or reduce operation will occur

D)It is the production p that will be used for reduction in the next step along with a position in the sentential form where the right hand side of the production may be found

Answer: D

Explanation: Handle is a substring of sentential form from which the start symbol can be reached using reduction at each step.

Question 19:[GATE CS 2007]

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

Answer: A

Question 20:[GATE CS 2007]

Consider the CFG with $\{S, A, B\}$ as the non-terminal alphabet, $\{a, b\}$ as the terminal alphabet, S as the start symbol and the following set of production rules:

$S \rightarrow aB \qquad S \rightarrow bA$

$B \rightarrow b \qquad A \rightarrow a$

$B \rightarrow bS \qquad A \rightarrow aS$

$B \rightarrow aBB \qquad S \rightarrow bAA$

For the string $aabbab$, how many derivation trees are there?

A. 1
B. 2
C. 3
D. 4

Answer: B

[Conti….]
Explanation:

$S \to aB$
$\to aaBB$
$\to aabB$
$\to aabbS$
$\to aabbaB$
$\to aabbab$

$S \to aB$
$\to aaBB$ (till now, only 1 choice possible)
$\to aabSB$ (last time we took $B \to b$, now taking $B \to bS$)
$\to aabbAB$
$\to aabbaB$
$\to aabbab$

So, totally 2 possible derivation trees.

Correct Answer: B

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 21:[GATE CS 2007]

In the context of compilers, which of the following is/are NOT an intermediate representation of the source program?

A. Three address code
B. Abstract Syntax Tree (AST)
C. Control Flow Graph (CFG)
D. Symbol table

Answer: D

Explanation:

Concept:

Intermediate Representation of Source Program: Means We have various stages in the Compilation process each of these stages **accepts some form of source program as an input** and **produces a different form as output**. These forms are called **Intermediate Representation**.

Control Flow Graph: Control flow Graph is a type of Structure of Intermediate Representation which represent entire flow of a program in terms of some variable notation.

Symbol Table: Symbol Table is a Data Structure used by various stages of compilers to capture information regarding variables, functions, classes, objects etc. Hence it is **not** a Intermediate Representation.

Three Address code: Three address code is a Linear form of Syntax Trees. It is generated by Code optimization phase of the compiler. **It is a Intermediate Representation.**

Abstract Syntax Tree: Abstract syntax Tree basically tree like structure of the source code. It is generated by Parser. **It is a Intermediate Representation**.

Question 22:[GATE CS 2005]

The grammar A → AA | (A) | ε is not suitable for predictive-parsing because the grammar is

A)ambiguous

B)left-recursive

C)right-recursive

D)an operator-grammar

Answer: A

Explanation: Since given grammar can have infinite parse trees for string 'ε', so grammar is ambiguous, and also A → AA has left recursion. For predictive-parsing, grammar should be:

- Free from ambiguity
- Free from left recursion
- Free from left factoring

Given grammar contains both ambiguity and left factoring, so it can not have predictive parser. We always expect first grammar free from ambiguity for parsing. Option (A) is more strong option than option (B) here

Question 23:[GATE CS 2005]

Consider the grammar

E → E + n | E × n | n For a sentence n + n × 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

Answer: D

Explanation:
E → E * n {Applying E → E * n }
→ E + n * n {Applying E → E + n }
→ n + n * n {Applying E → n }

Hence, the handles in right sentential form is n, E + n and E × n.

Question 24 [GATE CS 2005]

Consider the grammar

$S \rightarrow (S) \mid a$

Let the number of states in SLR(1), LR(1) and LALR(1) parsers for the grammar be n1, n2 and n3 respectively. The following relationship holds good

A) $n1 < n2 < n3$

B) $n1 = n3 < n2$

C) $n1 = n2 = n3$

D) $n1 \geq n3 \geq n2$

Answer: B

Explanation: LALR(1) is formed by merging states of LR(1) (also called CLR(1)), hence no of states in LALR(1) is less than no of states in LR(1), therefore $n3 < n2$. And SLR(1) and LALR(1) have same no of states, i.e ($n1 = n3$).
Hence $n1 = n3 < n2$

Question 25:[GATE CS 2004]

Which of the following grammar rules violate the requirements of an operator grammar ? P, Q, R are nonterminals, and r, s, t are terminals.

1. $P \rightarrow Q\ R$
2. $P \rightarrow Q\ s\ R$
3. $P \rightarrow \varepsilon$
4. $P \rightarrow Q\ t\ R\ r$

A)1 only

B)1 and 3 only

C)2 and 3 only

D)3 and 4 only

Answer: B

Explanation: Operator grammar should not have two or more variables in its production side by side, and should not have null productions.

Question 26:[GATE CS 2003]

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 these

Answer: D

Explanation: Removing left recursion and factoring the grammar do not suffice to convert an arbitrary CFG to LL(1) grammar.

Question 27:[GATE CS 1996]

The activities are listed below. What is the pass numbers of each of the activities respectively?

a. Object code generation
b. Literals added to the literal table
c. Listing printed
d. Address resolution of local symbols that occur in a two pass assembler

(A) 1, 2, 2, 2
(B) 2, 1, 1, 1
(C) 1, 2, 1, 2
(D) 2, 1, 2, 1

Answer: B

Explanation: The functions performed in pass 1 and pass 2 in 2 pass assembler are

Pass 1

1. Assign addresses to all statements in the program.
2. Save the values assigned to all labels for use in pass 2
3. Perform some processing of assembler directives.

Pass 2

1. Assemble instructions.
2. Generate data values defined by BYTE, WORD etc.
3. Perform processing of assembler directives not done during pass 1.
4. Write the program and the assembling listing

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 28:[GATE CS 2003]

Assume that the SLR parser for a grammar G has n1 states and the LALR parser for G has n2 states. The relationship between n1 and n2 is:

A) n1 is necessarily less than n2

B) n1 is necessarily equal to n2

C)n1 is necessarily greater than n2

D)none of these

Answer: B

Explanation: The number of entries in the LALR(1) parse table ≤ no. of entries in the CLR(1) parse table.
The number of entries in the SLR(1) parse table is ≤ no of entries in CLR(1) parse table.

Relation between them: n1 (SLR) = n2(LALR) ≤ n3 (CLR) Hence n1 is necessarily equal to n2 is the correct answer.

Question 29:[GATE CS 2001]

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

Answer: A

Explanation: A grammar is ambiguous if there exists a string s such that the grammar has more than one leftmost derivations for s. We could also come up with more than one rightmost derivations for a string to prove the above proposition, but not both of right and leftmost. An unambiguous grammar can have different rightmost and leftmost derivations.

Question 30:[GATE CS 2000]

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

A)Leftmost derivation

B)Leftmost derivation traced out in reverse

C)Rightmost derivation

D)Rightmost derivation traced out in reverse

Answer: A

Explanation: In top down parsing, we just start with the start symbol and compare the right side of the different productions against the first piece of input to see which of the productions should be used.A top down parser is called LL parser because it parses the input from **L**eft to right, and constructs a **L**eftmost derivation of the sentence

Question 31:[GATE CS 2000]

Given the following expression grammar:

```
E -> E * F | F + E | F
 F -> F - F | id
```

which of the following is true?

A)* has higher precedence than +

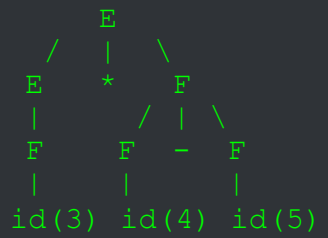
B)– has higher precedence than *

C)+ and — have same precedence

D)+ has higher precedence than *

Answer: B

Explanation:Let say input is 3*4-5 when we draw parse tree according to grammar



As we can see first '- ' will be evaluated then ' * ' is evaluated so ' - ' has higher precedence than *.

Question 32:[GATE CS 2015]

Which one of the following is True 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

Answer:C

Explanation: The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. By definition, a viable prefix is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

Question 33:[GATE CS 2015]

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

Answer: C

Explanation: SLR parser is a type of LR parser with small parse tables and a relatively simple parser generator algorithm. Canonical LR parser or LR(1) parser is an LR(k) parser for k=1, i.e. with a single lookahead terminal. It can handle all deterministic context-free languages. LALR parser or Look-Ahead LR parser is a simplified version of a canonical LR parser

Question 34:[GATE CS 1997]

In the following grammar
X :: = X ⊕ Y / Y
Y :: = Z * Y / Z
Z :: = id

Which of the following is true?

A) '⊕' is left associative while '*' is right associative

B) Both '⊕' and '*' are left associative

C) '⊕' is right associative while '*' is left associative

D) None of the above

Answer: A

Explanation: X :: = X ⊕ Y is left recursive grammar, so '⊕' is left associative and Z * Y is right recursive grammar, so '*' is right associative.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 35:[GATE CS 1997]

Which of the following is essential for converting an infix expression to the postfix from efficiently ?

A)An operator stack

B)An operand stack

C)An operand stack and an operator stack

D) A parse tree

Answer: A

Explanation: Operator stack is used for converting infix to postfix expression such that operators like as +, *, (,), / are pushed in stack where as operand stack is used for converting Postfix to Prefix evaluation such that operands are 7,2,1,2 etc. Hence, option (A) is correct.

Question 36:[GATE CS 2018]

Which is True about SR and RR-conflict:

A)If there is no SR-conflict in CLR(1) then definitely there will be no SR-conflict in LALR(1)

B)RR-conflict might occur if lookahead for final items(reduce-moves) is same.

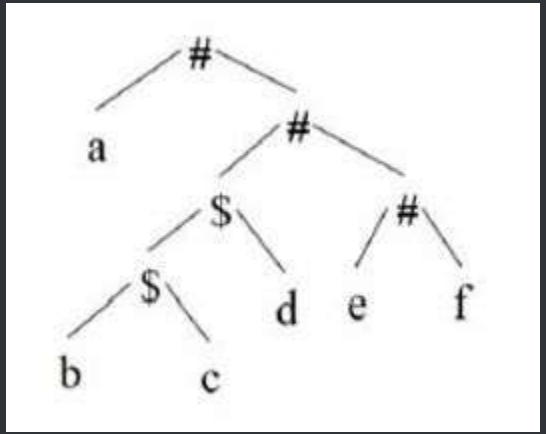
C)Known that CLR(1) has no RR-conflict, still RR-conflict might occur in LALR(1).

D)All of the above

Answer: D

Question 37:[GATE CS 2018]

Consider the following parse tree for the expression a#bcd#e#f, involving two binary operators $ and #.



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

A)$ has higher precedence and is left associative; # is right associative

B)# has higher precedence and is left associative; $ is right associative

C)$ has higher precedence and is left associative; # is left associative

D)# has higher precedence and is right associative; $ is left associative

Answer: A

Explanation: Since **$** will be evaluated first, so has higher precedence with left associativity. Whereas **#** is right associative. In d#e#f, e#f will be evaluated first (refer given parse tree). Therefore, **option (A) is Correct.**

Question 38:[GATE CS 2018]

Consider the following statements.

- **S1:** Every SLR(1) grammar is unambiguous but there are certain unambiguous grammars that are not SLR(1).
- **S2:** For any context-free grammar, there is a parser that takes at most $O(n^3)$ time to parse a string of length n.

Which one of the following options is correct?

A)S1 is true and S2 is false

B)S1 is false and S2 is true

C)S1 is true and S2 is true

D)S1 is false and S2 is false

Answer: C

Explanation: Statement (S1) is correct. Using CYK algorithm we can test the membership problem of CFG. It takes at most $O(n^3)$ time to parse a string of length n. Statement (S2) is also correct.

Question 39: **[GATE CS 2011]**

Which one of the following statements is TRUE?

A)The LALR(1) parser for a grammar G cannot have a reduce-reduce conflict if the LR(1) parser for G does not have reduce-reduce conflict

B)The symbol table is accessed only during the lexical analysis phase.

C)Data flow analysis is necessary for run-time memory management.

D)LR(1) parsing is sufficient for deterministic context-free languages

Answer:D

Explanation:

LR(1) parsing is a parsing technique for context-free grammars. A context-free grammar is deterministic if it has at most one production rule for each nonterminal symbol that can be applied in any given parsing state. A language is called deterministic context-free language (DCFL) if there exists a deterministic pushdown automaton (DPDA) that recognizes it. LR(1) parsing is sufficient to handle all DCFLs. In fact, LR(k) parsing can handle all DCFLs, for any value of $k \geq 1$.

Hence Option D is **True**. The LR parser can recognize any deterministic context-free language in linear-bounded time

Question 40: **[NTA UGC NET 2019-II]**

Shift-reduce parser consists of (a) input buffer (b) stack (c) parse table

choose the correct option from those given below:

A)(a) and (b) only

B)(a) and (c) only

C)(c) only

D)(a), (b) and (c)

Answer:D

Explanation: SR parser is a bottom up parser. It consists of input buffer, stack, parse table. Where Input buffer is used for storing strings to be parsed, stack is used for holding grammar symbols and parse table is used for parsing the string

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 41: [**GATE CS 2021]**

Consider the following augmented grammar with {#,@,<,>,a,b,c} as the set of terminals.
S' → S
S → S#cS
S → SS
S → S@
S → <S>
S → a
S → b
S → c
Let I0=CLOSURE({S'→ · S}). The number of items in the set GOTO(GOTO(I0 <), <) is _____

A)8

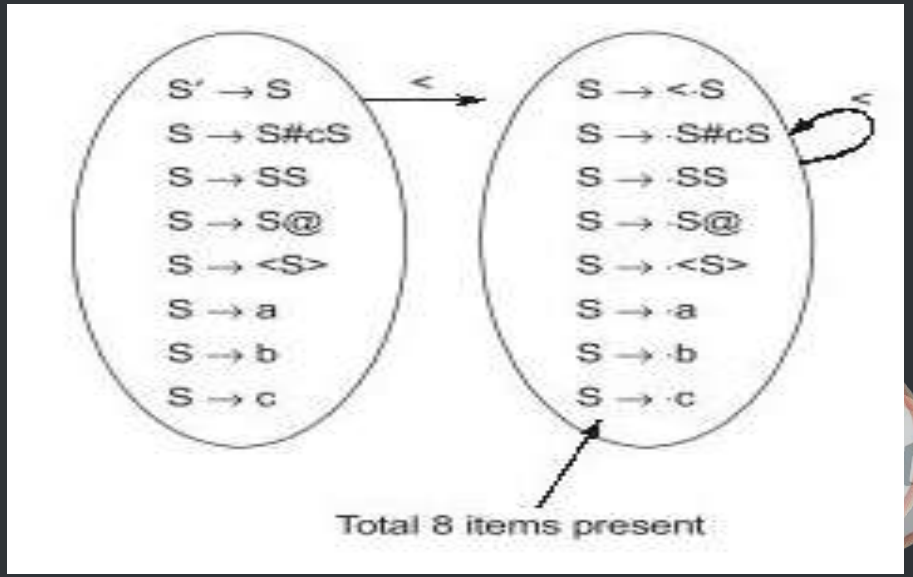
B)7

C)5

D)9

Answer: A

The number of items in the set GOTO(GOTO(I0 <), <) is 8.



Question 42: [**GATE CS 2021]**

Consider the following grammar (that admits a series of declarations, followed by expressions) and the associated syntax directed translation (SDT) actions, given as pseudo-code

$$
\begin{aligned}
P &\rightarrow D^* E^* \\
D &\rightarrow \text{int ID}\{\text{record that ID.lexeme is of type int}\} \\
D &\rightarrow \text{bool ID}\{\text{record that ID.lexeme is of type bool}\} \\
E &\rightarrow E_1 + E_2\{\text{check that } E_1.\text{type} = E_2.\text{type} = \text{int}; \text{set } E.\text{type} := \text{int}\} \\
E &\rightarrow !E_1\{\text{check that } E_1.\text{type} = \text{bool}; \text{ set } E.\text{type} := \text{bool}\} \\
E &\rightarrow \text{ID}\{\text{set } E.\text{type} := \text{int}\}
\end{aligned}
$$

With respect to the above grammar, which one of the following choices is correct

A)The actions can be used to correctly type-check any syntactically correct program

B)The actions can be used to type-check syntactically correct integer variable declarations and integer expressions

C)The actions can be used to type-check syntactically correct boolean variable declarations and boolean expressions.

D)The actions will lead to an infinite loop

Answer:B

Explanation: Rule 2 and 3 are used for entry into the symbol table. Rule 4 is used for type checking of the integer expression.

Question 43: **[GATE CS 2020]**

Consider the productions A → PQ and A → XY. Each of the five non-terminals A,P,Q,X, and Y has two attributes: s is a synthesized attribute, and i is an inherited attribute. Consider the following rules.
Rule 1: P.i=A.i+2, Q.i=P.i+A.i, and A.s=P.s+Q.s
Rule 2: X.i=A.i+Y.s and Y.i=X.s+A.i
Which one of the following is TRUE ?
A)Both Rule 1 and Rule 2 are L-attributed

B)Only Rule 1 is L-attributed

C)Only Rule 2 is L-attributed

D)Neither Rule 1 nor Rule 2 is L-attributed

Answer: B

Explanation: **According to L-attributed SDT:**

1. **If an SDT uses both synthesized attributes and inherited attributes with a restriction that inherited attribute can inherit values from left siblings only, it is called as L-attributed SDT.**
2. **Attributes in L-attributed SDTs are evaluated by depth-first and left-to-right parsing manner.**
3. **Semantic actions are placed anywhere in RHS. Therefore,**

Rule 1: P.i=A.i+2, Q.i=P.i+A.i, and A.s=P.s+Q.s

Rule 1 is L-attributed by definition. However,
Rule 2: X.i=A.i+Y.s and Y.i=X.s+A.i

Rule 2 is not L-attributed because this expression (X. Rule 2: i = A.i + Y.s) fails to satisfy the definition of L-attributed. It is failed for X.i =A.i +Y.s since X take value from its sibling which is present of its right, i.e., (A → XY). Option (B) is correct.

Question 44: **[GATE CS 2020]**

Consider the following grammar.
S → aSB | d
B → b
The number of reduction steps taken by a bottom-up parser while accepting the string aaadbbb is _____ . **Note -** This question was Numerical Type.

A)6

B)7

C)8

D)4

Answer: B
Explanation: According to Bottom Up or Shift Reduce Parsers: Initially: aaadbbb
1. aaaSbbb {S→d}
2. aaaSBbb {B→b}
3. aaSbb {S→aSB}
4. aaSBb {B→b}
5. aSb {S→aSB}
6. aSB {B→b}
7. S {S→aSB}
Therefore, total 7 steps required. Option (B) is correct.

Question 45: **[GATE CS 2020]**

Consider the following statements.

I. Symbol table is accessed only during lexical analysis and syntax analysis.

II. Compilers for programming languages that support recursion necessarily need heap storage for memory allocation in the run-time environment.

III. Errors violating the condition ‘any variable must be declared before its use’ are detected during syntax analysis.

Which of the above statements is/are TRUE ?

A)I only

B)I and III only

C) Ⅱ only

D)None of Ⅰ, Ⅱ and Ⅲ

Answer:D

Explanation: **I.** False. Symbol Table is used by *all 6 phases of compiler*: lexical analysis, syntax analysis, semantic analysis, intermediate code generation, code optimization, and target code generation phase. **II.** False. Heap allocation is used to dynamically allocate memory to the variables. Heap allocation can be used for recursion, but recursion is also supported in Stack allocation. **III.** False. Syntax analysis checks only syntax, but variable declare before its use must be done by Semantic analyses. Option (D) is correct.

Question 46: [**GATE CS 2020**]

Which of the following is true?

A)Canonical LR parser is LR (1) parser with single look ahead terminal

B)All LR(K) parsers with K > 1 can be transformed into LR(1) parsers.

C)Both (A) and (B)

D)None of the above

Answer: C

Explanation: Canonical LR parser is LR (1) parser with single look ahead terminal. All LR(K) parsers with K > 1 can be transformed into LR(1) parsers. For more information on canonical parser and SLR parser Refer:Parsing | Set 3 (SLR, CLR and LALR Parsers) Option (C) is correct.

Question 47: [**UGC NET CS 2014**]

Shift-Reduce parsers perform the following:

A)Shift step that advances in the input stream by K(K > 1) symbols and Reduce step that applies a completed grammar rule to some recent parse trees, joining them together as one tree with a new root symbol.

B)Shift step that advances in the input stream by one symbol and Reduce step that applies a completed grammar rule to some recent parse trees, joining them together as one tree with a new root symbol.

C)Shift step that advances in the input stream by K(K = 2) symbols and Reduce step that applies a completed grammar rule to form a single tree

D)Shift step that does not advance in the input stream and Reduce step that applies a completed grammar rule to form a single tree.

Answer: B

Explanation: Shift step that advances in the input stream by one symbol and Reduce step that applies a completed grammar rule to some recent parse trees, joining them together as one tree with a new root symbol. For more information on Shift-Reduce parsers Refer:Shift Reduce Parser in Compiler Option (B) is correct.

Question 48: **[UGC NET CS 2015]**

Which one from the following is false?

A)LALR parser is Bottom - Up parser

B)A parsing algorithm which performs a left to right scanning and a right most deviation is RL (1)

C)LR parser is Bottom - Up parser.

D)In LL(1), the 1 indicates that there is a one - symbol look - ahead.

Answer: B

Explanation:LALR parser is Bottom - Up parser.**True**

- A parsing algorithm which performs a left to right scanning and a right most deviation is RL (1).**False (there is no such algo)**
- LR parser is Bottom - Up parser. **True**
- In LL(1), the 1 indicates that there is a one - symbol look - ahead.**True**

So, option (B) is correct.

Question 49: [GATE CS Mock]

A translation scheme is shown as-

S->S+S_2 {S.val S_1.val + S_2.val}

IS*S_2 (S.val= S_1.val S_2.val)

| id{S.val = id}

Output for 32+ 4 is?

Answer: 18

Explanation: Given grammar is an ambiguous grammar.

Therefore two outputs are possible for single input i.e. 3*2+4.

output 1 = 10

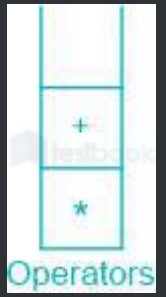
output 2 = 18

This problem is resolved by YACC Tool. So, according to rule of YACC Tool:

(1) S-R conflict is resolved by 'S'

(2) R_1 - R; conflict is resolved by R, where i<j.

So, scanning input from left to right



Operand 324

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 50 [**ISRO CS 2013**]

Which of the following productions eliminate left recursion in the productions given below: S → Aa | b A → Ac | Sd | ε

A)S → Aa | b A → bdA' A' → A'c | A'ba | A | ε

B)S → Aa | b A → A' | bdA', A' → cA' | adA' | ε

C)S → Aa | b A → A'c | A'd A' → bdA' | cA | ε

D)S → Aa | b A → cA' | adA' | bdA' A' → A | ε

Answer: B

Explanation: To remove left recursion from the grammar of the

form : A → Aα | β

We rewrite the production rules as:

A → βA'

A'→ αA'| ε

Given Grammar: S → Aa | b

A → Ac | Sd | ε

after finding indirect left recursion, grammar:

S → Aa | b

A → Ac | Aad | bd | ε

here, α = c, ad, β = bd

So, Grammar after removing left recursion =

S → Aa | b

A → A' | bdA'

A'→ CA'| ada'| ε

So, option (B) is correct.

Question 51: **[ISRO CS 2013]**

Shift reduce parsing belongs to a class of

A)bottom up parsing

B)top down parsing

C)recursive parsing

D)predictive parsing

Answer: A

Explanation: Shift reduce are the class of parsers which built a parse tree in bottom-up manner and scans it from left to right. In these parsers, through shift step, a single character from the token stream is pushed on the parse stack and becomes a new single-node parse tree.A reduce step applies a complete grammar production rule to the recent parse trees, joining them together as one tree with a new root symbol.These parsers include LR parsers which are non-backtracking in nature. Examples: SLR, CLR, LALR etc.

Question 52: **[ISRO CS 2015]**

Which grammar rules violate the requirement of the operator grammar? A, B, C are variables and a, b, c are terminals

1) $A \rightarrow BC$

2) $A \rightarrow CcBb$

3) $A \rightarrow BaC$

4) $A \rightarrow \varepsilon$

A)1 only

B)1 and 2 only

C)1 and 3 only

D)1 and 4 only

Answer:D

Explanation: In operator grammar production rules which have two adjacent non-terminals on right hand side are not allowed.

Additionally empty production rules are also not allowed. So, $A \rightarrow BC$ and $A \rightarrow \varepsilon$ are not allowed. Correct option is (D).

Question 53: **[ISRO CS 2015]**

Given the following expression grammar:

$E \rightarrow E * F \mid F + E \mid F$

$F \rightarrow F - F \mid id$

Which of the following is true?

A) * has higher precedence than +

B) - has higher precedence than *

C) + and - have same precedence

D) + has higher precedence than *

Answer: B

Explanation: Operator at lower level in tree has higher precedence than operator at upper levels

- has higher precedence than +and *

+ and * have equal precedence

Question 54: [**ISRO CS 2015**]

Consider the following statements related to compiler construction :

I. Lexical Analysis is specified by context-free grammars and implemented by pushdown automata.

II. Syntax Analysis is specified by regular expressions and implemented by finite-state machine.

Which of the above statement(s) is/are correct ?

A) Only I

B) Only II

C) Both I and II

D) Neither I nor II

Answer: D

Question 55: [**UGC NET CS 2019**]

Which of the following statements is/are TRUE ? (i) The grammar S → SS | a is ambiguous (where S is the start symbol). (ii) The grammar S → 0S1 | 01S | e is ambiguous (the special symbol e represents the empty string and S is the start symbol). (iii) The grammar (where S is the start symbol).

S → T/U

T → x S y / xy / e

U → yT

generates a language consisting of the string yxxyy

A) Only (i) and (ii) are TRUE

B) Only (i) and (iii) are TRUE

C) Only (ii) and (iii) are TRUE

D) All of (i), (ii) and (iii) are TRUE

Answer: D

Explanation: We can generate more than 1 parse tree for a single string from the grammars. For statement (iii)

S → U

U → yT

yT → yxSy

yxSy → yxTy

yxTy → yxxyy

All statement are correct. So, option (D) is correct.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

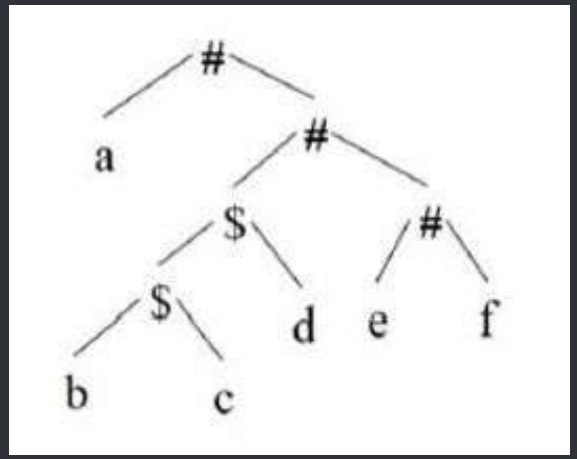
For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 56: [**GATE CS 2018**]

Consider the following parse tree for the expression a#bcd#e#f, involving two binary operators $ and #.



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

A)$ has higher precedence and is left associative; # is right associative

B)# has higher precedence and is left associative; $ is right associative

C)$ has higher precedence and is left associative; # is left associative

D)# has higher precedence and is right associative; $ is left associative

Answer: A

Explanation: Since **$** will be evaluated first, so has higher precedence with left associativity. Whereas **#** is right associative. In d#e#f, e#f will be evaluated first (refer given parse tree). Therefore, **option (A) is Correct.**

Question 57: [**GATE CS 1996**]

Consider the syntax-directed translation schema (SDTS) shown below:

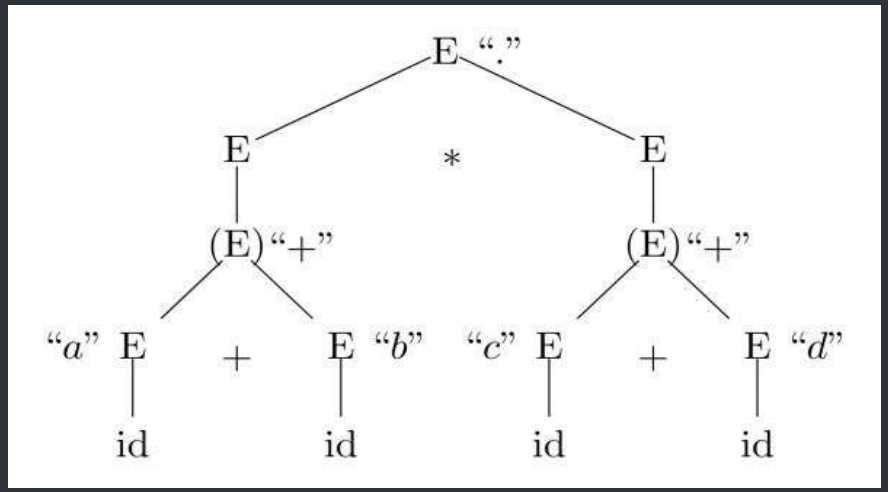
- $E \rightarrow E + E$ {print "+"}
- $E \rightarrow E * E$ {print "."}
- $E \rightarrow id$ {print id.name}
- $E \rightarrow (E)$

An LR-parser executes the actions associated with the productions immediately after a reduction by the corresponding production. Draw the parse tree and write the translation for the sentence.

$(a + b) * (c + d)$, using SDTS given above.

Answer: ab+cd+.

Explanation:



Question 58: **[GATE CS 2018]**

Consider the following grammars

(S1) :
A --> aBCD
B --> bc|c
C --> d|∈
D -> b

(S2) :
A --> aBCD
B --> bc|∈
C --> d|c
D -> b

(S3) :
A --> aBCD
B --> bc|∈
C --> d|∈
D -> b

(S4) :
A --> aBCD
B --> bc|c
C --> d|c
D -> b

Which of the following grammar has same follow set for variable B?

A)Only (S1), (S2) and (S3), (S4)

B)Only (S1), (S3) and (S2), (S4)

C)Only (S2), (S3) and (S1), (S4)

D)None of the above

Answer: B

Explanation:

Grammar (S1) and (S3) has follow sets :

A = {$},
B = {b, d},
C = {b},
D = {$}

So, option (B) is correct.

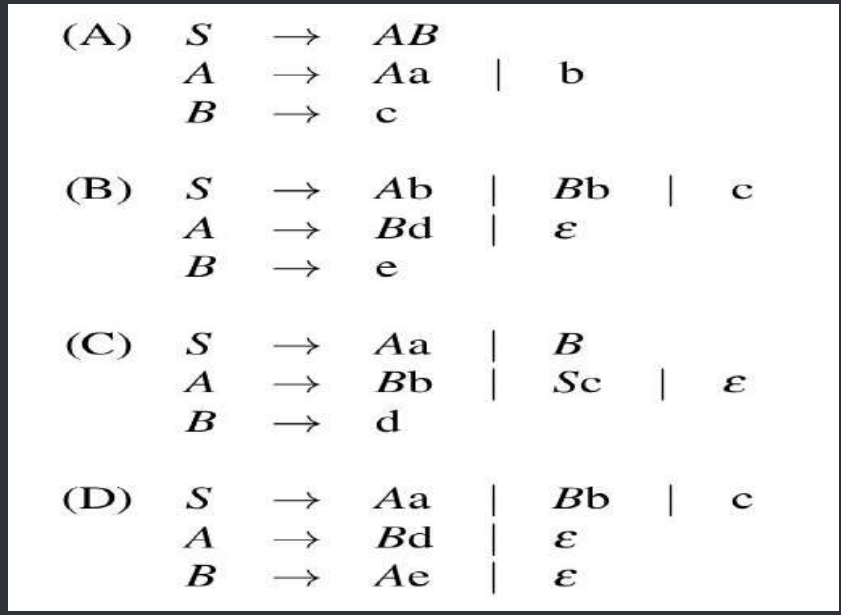
Grammar (S2) and (S4) has follow sets :

A = {$},
B = {c, d},
C = {b},
D = {$}

Question 59: **[GATE CS 2016]**

Which one of the following grammars is free from left recursion?

A) A
B) B
C) C
D) D



Answer: B

Explanation: Grammar A has direct left recursion because of the production rule: A->Aa.

Grammar C has indirect left recursion because of the production rules:S-> Aa and A->Sc

Grammar D has indirect left recursion because of production rules : A-> Bd and B-> Ae

Grammar B doesn't have any left recursion (neither direct nor indirect).

Question 60: **[GATE CS 2016]**

Consider the following Syntax Directed Translation Scheme (SDTS), with non-terminals {S, A} and terminals {a, b}}.

S	⟶	**aA**	{ print 1 }
S	⟶	**a**	{ print 2 }
A	⟶	**Sb**	{ print 3 }

Using the above SDTS, the output printed by a bottom-up parser, for the input **aab** is

A) 1 3 2
B) 2 2 3
C) 2 3 1
D) Syntax Error

Answer: C

Explanation: Bottom up parser builds the parse tree from bottom to up, i.e from the given string to the starting symbol. The given string is aab and starting symbol is S. so the process is to start from aab and reach

S. =>aab (given string)

=>aSb (after reduction by S->a, and hence print 2)

=>aA (after reduction by A->Sb, and hence print 3)

=>S (after reduction by S->aA, and hence print 1)

As we reach the starting symbol from the string, the string belongs to the language of the grammar. Another way to do the same thing is :- bottom up parser does the parsing by RMD in reverse. RMD is as follows:

=>S => aA (hence, print 1)

=> aSb (hence, print 3)

=> aab (hence, print 2) If we take in Reverse it will print : 231

Question 61: **[GATE CS 2015]**

Consider the following grammar G.

S → F | H

F → p | c

H → d | c

Where S, F and H are non-terminal symbols, p, d and c are terminal symbols. Which of the following statement(s) is/are correct?

S1: LL(1) can parse all strings that are generated using grammar G.

S2: LR(1) can parse all strings that are generated using grammar G.

A) Only S1

B) Only S2

C) Both S1 and S2

D) Neither S1 and S2

Answer: D

Explanation: The given grammar is ambiguous as there are two possible leftmost derivations for string "c".

First Leftmost Derivation

S → F

F → c

Second Leftmost Derivation

S → H

H → c

An Ambiguous grammar can neither be LL(1) nor LR(1)

Question 62: [**GATE CS 2015]**

In the context of abstract-syntax-tree (AST) and control-flow-graph (CFG), which one of the following is True?

In both AST and CFG, let node N2 be the successor of node N1. In the input program, the code corresponding to N2 is present after

A)the code corresponding to N1

B)For any input program, neither AST nor CFG will contain a cycle

C)The maximum number of successors of a node in an AST and a CFG depends on the input program

D)Each node in AST and CFG corresponds to at most one statement in the input program

Answer: C

Explanation:

(A) is false, In CFG (Control Flow Graph), code of N2

may be present before N1 when there is a loop or goto.

(B) is false, CFG (Control Flow Graph) contains cycle

when input program has loop.

(C) is true, successors in AST and CFG are dependent on

input program

(D) is false, a single statement may belong to a block

of statements.

Question 63: [**GATE CS 2015**]

Which one of the following is True 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

Answer: C

Explanation: The prefixes of right sentential forms that can appear on the stack of a shift-reduce parser are called viable prefixes. By definition, a viable prefix is a prefix of a right sentential form that does not continue past the right end of the rightmost handle of that sentential form.

Question 64: [**GATE CS 2003**]

Consider the syntax directed definition shown below.

S → id : = E {gen (id.place = E.place;);}

E → E1 + E2 {t = newtemp ( ); gen (t = El.place + E2.place;); E.place = t}

E → 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 ti'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) t1 = Y + Z; X = t1
C) t1 =Y; t2 = t1 + Z; X = t2
D) t1 = Y; t2 = Z; t3 = t1 + t2; X = t3a

Answer: B

Explanation: It must be B. The production E --> E + E is used only one time and hence only one temporary variable is generated.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 65: **[GATE CS 2003]**

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

E → number	E.val = number. val
| E '+' E	E(1).val = E(2).val + E(3).val
| E '×' E	E(1).val = E(2).val × 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

Answer: C

Explanation: shift is preferred over reduce while shift/reduce conflict. first reduce is preferred over others while reduce/reduce conflict. You can answer to this question straightforward by constructing LALR(1) parse table, though its a time taking process. To answer it faster, one can see intuitively that this grammar will have a shift-reduce conflict for sure. In that case, given this is a single choice question, (C) option will be the right answer. Fool-proof explanation would be to generate LALR(1) parse table, which is a lengthy process. Once we have the parse table with us, we can clearly see that i. reduce/reduce conflict will not arise in the above given grammar ii. shift/reduce conflict will be resolved by giving preference to shift, hence making the expression calculator right associative. According to the above conclusions, only correct option seems to be (C).

Question 66: [**GATE CS 2007**]

Consider the grammar with non-terminals N = {S,C,S1 },terminals T={a,b,i,t,e}, with S as the start symbol, and the following set of rules:

```
S --> iCtSS1|a
S1 --> eS|□
C --> b
```

The grammar is NOT LL(1) because:

A)it is left recursive

B)it is right recursive

C)it is ambiguous

D)It is not context-free.

Answer: C

Explanation: A LL(1) grammar doesn't give to multiple entries in a single cell of its parsing table. It has only single entry in a single cell, hence it should be unambiguous.

A parsing table of a grammar will **not** have multiple entries in a cell(i.e. will be a LL(1) grammar) **if and only if** the following conditions hold for each production of the form A->α|β 1) FIRST(α) ∩ FIRST(β) = Φ 2) if FIRST(α) contains ' ε ' then FIRST(α) ∩ FOLLOW (A) = Φ and vice-versa.

- For the production , S->iCtSS1|a, rule 1 is satisfied, because FIRST(iCtSS1) ∩ FIRST(a) = {i} ∩ {a} = Φ
- For the production, S1->eS|ε, rule 1 is satisfied, as FIRST(eS) ∩ FIRST(ε) = {e} ∩ {ε} = Φ . But here due to 'ε' in FIRST, we have to check for rule 2. FIRST(eS) ∩ FOLLOW(S1) = {e} ∩ {e, $} ≠ Φ . Hence rule 2 fails in this production rule. Therefore there will be multiple entries in the parsing table, hence the grammar is ambiguous and not LL(1).

Question 67: [**GATE CS 2013**]

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

X -> c.X, c/d
X -> .cX, c/d
X -> .d, c/d
X -> c.X, $
X -> .cX, $
X -> .d, $

Which of the following statements related to merging of the two sets in the corresponding LALR 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 result in R-R conflict.
4. Cannot be merged since goto 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

Answer: D

Question 68: [**GATE CS 2013]**

For the grammar below, a partial LL(1) parsing table is also presented along with the grammar. Entries that need to be filled are indicated as E1, E2, and E3. is the empty string, $ indicates end of input, and, | separates alternate right hand

$S \rightarrow a\,A\,b\,B \mid b\,A\,a\,B \mid \varepsilon$

$A \rightarrow S$

$B \rightarrow S$

	a	b	$
S	**E1**	**E2**	$S \rightarrow \varepsilon$
A	$A \rightarrow S$	$A \rightarrow S$	error
B	$B \rightarrow S$	$B \rightarrow S$	**E3**

(A) $FIRST(A) = \{a, b, \varepsilon\} = FIRST(B)$
$FOLLOW(A) = \{a, b\}$
$FOLLOW(B) = \{a, b, \$\}$

(B) $FIRST(A) = \{a, b, \$\}$
$FIRST(B) = \{a, b, \varepsilon\}$
$FOLLOW(A) = \{a, b\}$
$FOLLOW(B) = \{\$\}$

(C) $FIRST(A) = \{a, b, \varepsilon\} = FIRST(B)$
$FOLLOW(A) = \{a, b\}$
$FOLLOW(B) = \emptyset$

(D) $FIRST(A) = \{a, b\} = FIRST(B)$
$FOLLOW(A) = \{a, b\}$
$FOLLOW(B) = \{a, b\}$

Answer: A

Explanation: First(X) - It is the set of terminals that begin the strings derivable from X.
Follow(X) - It is the set of terminals that can appear immediately to the right of X in some sentential form.
Now in the above question,
FIRST(S) = { a, b, epsilon}
FIRST(A) = FIRST(S) = { a, b, epsilon}
FIRST(B) = FIRST(S) = { a, b, epsilon}
FOLLOW (A) = { b , a }
FOLLOW (S) = { $ } U FOLLOW (A) = { b , a , $ }
FOLLOW (B) = FOLLOW (S) = { b ,a , $ } epsilon corresponds to empty string.

Question 69: **[GATE Mock test]**

Consider the following grammar. How many backtracks are required to generate the string aab from the above grammar?

S → aB | aAb

A → bAb | a

B → aB | ε

Answer: 2

Explanation:

S->aB->aaB->aa unsuccessful

backtrack 1 time

S->aAb->abAbb->ababb unsuccessful

backtrack

S->aAb->aab //successful , no backtrack

So, 2 backtracking required

Question 70: [**GATE cs 2010]**

The grammar S → aSa | bS | c is

A)LL(1) but not LR(1)

B)LR(1)but not LR(1)

C)Both LL(1)and LR(1)

D)Neither LL(1)nor LR(1)

Answer: C

Explanation: First(aSa) = a

First(bS) = b

First(c) = c

All are mutually disjoint i.e no common terminal between them, the given grammar is LL(1).

As the grammar is LL(1) so it will also be LR(1) as LR parsers aremore powerful then LL(1) parsers. and all LL(1) grammar are also LR(1)So option C is correct.Below are more details. A grammar is LL(1) if it is possible to choose the next production by looking at only the next token in the input string.

Formally, grammar G is LL(1) if and only if

For all productions $A \rightarrow \alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha n$,

$First(\alpha i) \cap First(\alpha j) = \varnothing,\ 1 \leq i,j \leq n,\ i \neq j.$

For every non-terminal A such that First(A) contains ε,

$First(A) \cap Follow(A) = \varnothing$

Question 71: [**GATE CS 2014]**

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

S --> T * P
T --> U | T * U
P --> Q + P | Q
Q --> Id
U --> 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

Answer: B

Explanation: From the grammar we can find out associative by looking at grammar.Let us consider the 2nd production

T -> T * U T is generating T*U recursively (left recursive) so * is left associative.Similarly

P -> Q + P Right recursion so + is right associative.

Question 72: **[GATE Mocktest]**

Consider the following grammar. How many back tracks are required to generate the string aab from the above grammar?

S → aB | aAb

A → bAb | a

B → aB | ε

Answer: 2

Explanation:

S->aB->aaB->aa unsuccessful

backtrack 1 time

S->aAb->abAbb->ababb unsuccessful

backtrack

S->aAb->aab //successful , no backtrack

So, 2 backtracking required

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 73: **[GATE MockTest]**

Let G be any grammar with the following productions:
X → X+Y | Y
Y→ Y*Z | Z
Z → (X)
Z → id
If LR(1) parser is used to parse the above grammar, then total how many look-a-heads are present for the item "X→.Y" and "Z →.id" in the initial state?

Solution:

$X' \rightarrow .X, \{\$\}$
$X \rightarrow .X + Y, \{\$, +\}$
$X \rightarrow .Y, \{\$\}$ ...(i) << Possibly {$,+} should have been here
$Y \rightarrow .Y * Z, \{\$, *\}$
$Y \rightarrow .Z, \{\$, *\}$
$Z \rightarrow .(X) \{\$, *\}$
$Z \rightarrow .id, \{\$, *\}$ ...(ii)

So, total (3) look-a-heads are there.

Question 74: [**GATE cs 2007]**

Consider the following two statements:

- P: Every regular grammar is LL(1)
- Q: Every regular set has a LR(1) grammar

Which of the following is **TRUE**?

A. Both P and Q are true
B. P is true and Q is false
C. P is false and Q is true
D. Both P and Q are false

Answer: C

Explanation: P : FALSE because a left-linear regular grammar can be left-recursive and left recursive languages cannot be LL(1)

Q: TRUE because every regular set (or language) has a right-linear deterministic (or left-factored) unambiguous grammar and thus, every regular language can have an LL(1) grammar. Since every LL(1) grammar is also LR(1), Q is true.

NOTE : For, every regular language, there exists an unambiguous grammar because regular languages are acceptable by DFAs and unambiguity is a property of non-determinism.

Question 75: [**GATE CS 2007**]

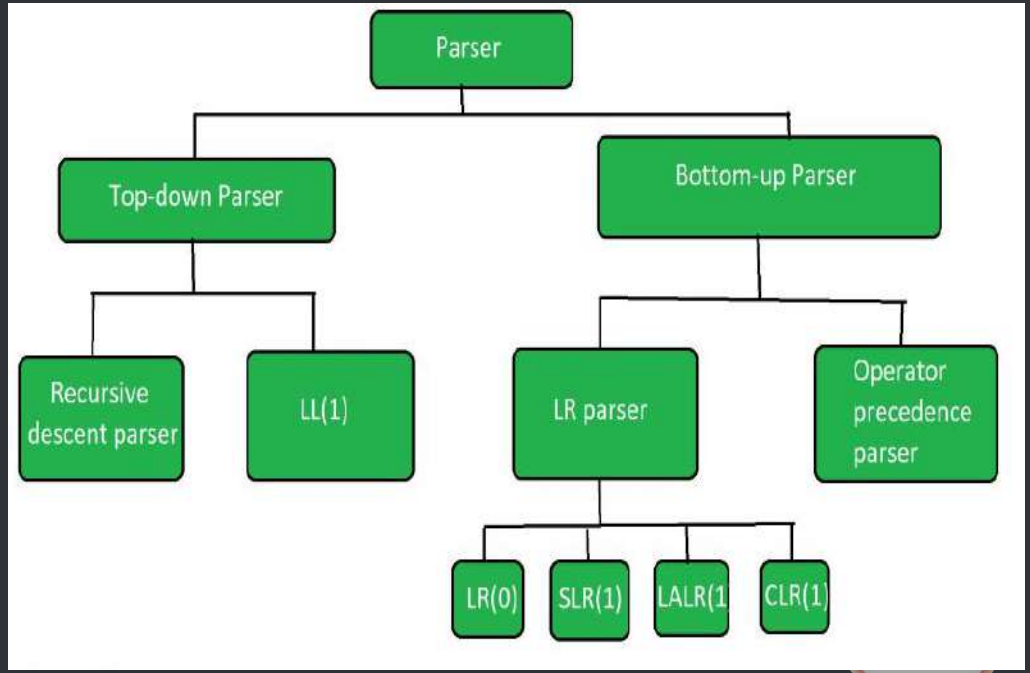
Which one of the following is a top-down parser?

A. Recursive descent parser.
B. Operator precedence parser.
C. An LR(k) parser.
D. An LALR(k) parser.

Answer: A

Explanation:

1. Recursive descent parser-TOP DOWN PARSER
2. Operator precedence parser-BOTTOM UP PARSER
3. An LR(k) parser.-BOTTOM UP PARSER
4. An LALR(k) parser-BOTTOM UP PARSER



Question 76: [**GATE CS 2007**]

Consider the CFG with {S,A,B) as the non-terminal alphabet, {a,b) as the terminal alphabet, S as the start symbol and the following set of production rules

```
S --> aB        S --> bA
B --> b         A --> a
B --> bS        A --> aS
B --> aBB       A --> bAA
```

Which of the following strings is generated by the grammar?

A) aaaabb

B) aabbbb

C) aabbab

D)abbbba

Answer: C

Explanation: We can derive **aabbab** using below sequence

```
S  -> aB      [Using S --> aB]
   -> aaBB    [Using B --> aBB]
   -> aabB    [Using B --> b]
   -> aabbS   [Using B --> bS]
   -> aabbaB  [Using S --> aB]
   -> aabbab  [Using B --> b]
```

Question 77: [**GATE CS 2006**]

Consider the following statements about the context free grammar

G = {S → SS, S → ab, S → ba, S → E}

I. G is ambiguous

II. G produces all strings with equal number of a's and b's

III. G can be accepted by a deterministic PDA.

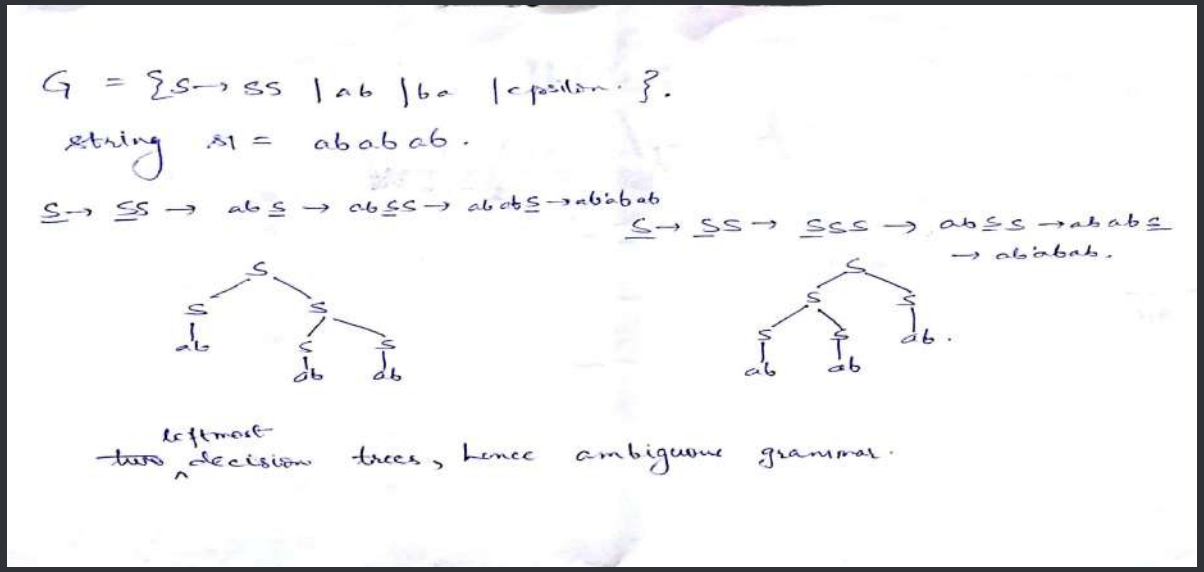
Which combination below expresses all the true statements about G?

(A) I only
(B) I and III only
(C) II and III only
(D) I, II and III

Answer: B

Statement I: G is ambiguous because, as shown in the image below there can be two decision tree for string S = ababab [TRUE]

[cont....]



Statement II: G produces all strings with equal number of a's and b's [FALSE]

string 'aabb' cannot be produced by G

Statement III: G can be accepted by a deterministic PDA [TRUE]

Assume there is a PDA which pushes if top of the stack is $ (bottom most alphabet of the stack) and pops otherwise. A string is rejected while popping if the current letter and top of the stack are same. This PDA can derive G.

Hence, correct answer should be (B) I and III only

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 78: [**GATE CS 2006**]

Consider the following translation scheme.

S → ER

R → *E{print(“*”);}R | ε

E → F + E {print(“+”);} | F

F → (S) | id {print(id.value);}

Here id is a token that represents an integer and id.value represents the corresponding integer value.

For an input ‘2 * 3 + 4’, this translation scheme prints

(A) 2 * 3 + 4

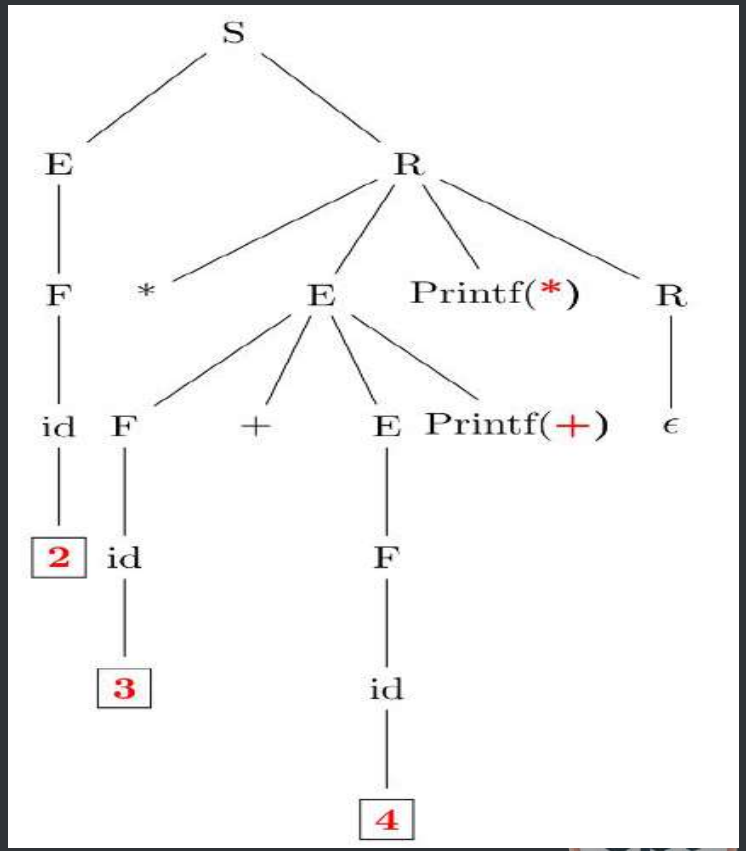
(B) 2 * +3 4

(C) 2 3 * 4 +

(D) 2 3 4+*

Answer: (D)

Explanation: perform post order evaluation of parse tree



Question 79: [**GATE CS 2006**]

Consider the following grammar:

```
S → FR

R → S | ε

F → id
```

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

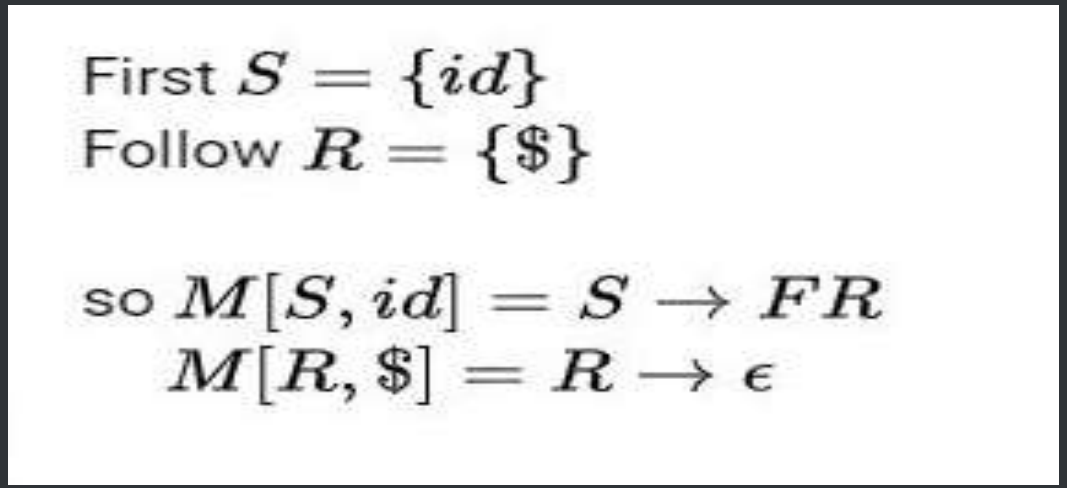
(A) {S → FR} and {R → ε }

(B) {S → FR} and { }

(C) {S → FR} and {R → *S}

(D) {F → id} and {R → ε}

Answer: A



Question 80: **[GATE CS 2006]**

Which one of the following grammars generates the language $L = \{a^i b^j \mid i \neq j\}$?

A. $S \rightarrow AC \mid CB$
$C \rightarrow aCb \mid a \mid b$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow Bb \mid \varepsilon$

B. $S \rightarrow aS \mid Sb \mid a \mid b$

C. $S \rightarrow AC \mid CB$
$C \rightarrow aCb \mid \varepsilon$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow Bb \mid \varepsilon$

D. $S \rightarrow AC \mid CB$
$C \rightarrow aCb \mid \varepsilon$
$A \rightarrow aA \mid a$
$B \rightarrow Bb \mid b$

Answer: D

Explanation: Language L contains the strings : {**abb, aab, abbb, aabbb, aaabb, aa, bb, ...**}, i.e, all a's appear before b's in a string, and "number of a's" is not equal to "number of b's", So i ≠ j.

Here **Grammar A, B & C** also generate the string "ab", where i = j, and many more strings with i = j, hence these grammars do not generate the language L, because for a string that belongs to language L, exponent i should not be equal to exponent j.

Grammar D : This Grammar never generates a string with equal no of a's and b's, i.e. i=j. Hence this grammar generates the language L. Hence Option D.

Question 81: **[GATE CS 2006]**

- $S \rightarrow AC \mid CB$
- $C \rightarrow aCb \mid \epsilon$
- $A \rightarrow aA \mid a$
- $B \rightarrow Bb \mid b$

generates the language $L = \{a^i b^j \mid i \neq j\}$. In this grammar what is the length of the derivation (number of steps starting from S) to generate the string $a^l b^m$ with $l \neq m$

A. $\max(l, m) + 2$
B. $l + m + 2$
C. $l + m + 3$
D. $\max(l, m) + 3$

Answer: A

Explanation: Let us take *aabbb* as an example to check in the options[l=2, m=3]

Productions used in this sequence:

S-->CB

C-->aCb

C-->aCb

C-->ε

B-->b

total 5 sequences are used. Only option A satisfies this.

Question 82: [**GATE CS 2012**]

For the grammar below, a partial LL(1) parsing table is also presented along with the grammar. Entries that need to be filled are indicated as E1, E2, and E3. & is the empty string, $ indicates end of input, and, separates alternate right hand sides of productions.

- **Sa AbB | bAaB | €**
- **A → S**
- **B → S**

	a	b	$
S	E1	E2	$S \rightarrow \varepsilon$
A	$A \rightarrow S$	$A \rightarrow S$	error
B	$B \rightarrow S$	$B \rightarrow S$	$E3$

The appropriate entries for $E1$, $E2$, and $E3$ are

A. $E1 : S \rightarrow aAbB, A \rightarrow S$
$E2 : S \rightarrow bAaB, B \rightarrow S$
$E3 : B \rightarrow S$

B. $E1 : S \rightarrow aAbB, S \rightarrow \varepsilon$
$E2 : S \rightarrow bAaB, S \rightarrow \varepsilon$
$E3 : S \rightarrow \varepsilon$

C. $E1 : S \rightarrow aAbB, S \rightarrow \varepsilon$
$E2 : S \rightarrow bAaB, S \rightarrow \varepsilon$
$E3 : B \rightarrow S$

D. $E1 : A \rightarrow S, S \rightarrow \varepsilon$
$E2 : B \rightarrow S, S \rightarrow \varepsilon$
$E3 : B \rightarrow S$

[Conti.....]

Answer: C

To make $LL(1)$ parsing table first we have to find $\mathbf{FIRST}$ and $\mathbf{FOLLOW}$ sets from the given grammar.

- $\mathbf{FIRST}(S) = \{a, b, \epsilon\}$
- $\mathbf{FIRST}(A) = \{a, b, \epsilon\}$
- $\mathbf{FIRST}(B) = \{a, b, \epsilon\}$
- $\mathbf{FOLLOW}(S) = \{a, b, \$\}$
- $\mathbf{FOLLOW}(A) = \{a, b\}$
- $\mathbf{FOLLOW}(B) = \{a, b, \$\}$

Now lets make $LL(1)$ parse table

Non Terminal	a	b	$
S	$S \to aAbB$, $S \to \epsilon$	$S \to bAbB$, $S \to \epsilon$	$S \to \varepsilon$
A	$A \to S$	$A \to S$	
B	$B \to S$	$B \to S$	$B \to S$

Here is the explanation of entries asked in question

1. For E1 and E2 Look into $\mathbf{FIRST}(S) = \{a, b, \epsilon\}$.
   a is because of $S \to aAbB$ and b is because of $S \to bAaB$
   So $M[S, a]$ and $M[S, b]$ will contain $S \to aAbB$ and $S \to bAaB$ respectively. For epsilon look into $\mathbf{FOLLOW}(S) = \{a, b, \$\}$. So $S \to \epsilon$ will be in $M[S, a]$, $M[S, b]$ and $M[S, \$]$
2. Now for E3 look into $\mathbf{FIRST}(B) = \{a, b, \epsilon\}$. a and b are because of $B \to S$.
   So $M[B, a]$ and $M[B, b]$ will contain $B \to S$ and for ϵ, look into $\mathbf{FOLLOW}(B) = \{a, b, \$\}$.
   Hence $M[B, \$]$ will contain $B \to S$.

Now we get the answer as E1 is $S \to aAbB$, $S \to \epsilon$, E2 is $S \to bAaB$, $S \to \epsilon$ and E3 is $B \to S$.

Hence, **Option (C)** is correct.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 83: [**GATE CS 2006**]

Consider a grammar with the following productions

- $S \rightarrow a\alpha b \mid b\alpha c \mid aB$
- $S \rightarrow \alpha S \mid b$
- $S \rightarrow \alpha bb \mid ab$
- $S\alpha \rightarrow bdb \mid b$

The above grammar is:

A. Context free
B. Regular
C. Context sensitive
D. $LR(k)$

Answer: D

Explanation:

- $S\alpha \rightarrow$

This violates the condition of context-free grammar that the LHS must be a single non-terminal symbol.

- $S\alpha \rightarrow b$

This violates even the weaker requirement for CSG that the length of RHS of a production must be at least same as that of LHS. So, the grammar is not even context-sensitive.

Question 84: [**GATE CS 1995**]

A. Translate the arithmetic expression $a^* - (b + c)$ into syntax tree.

B. A grammar is said to have cycles if it is the case that $A \overset{+}{\Rightarrow} A$

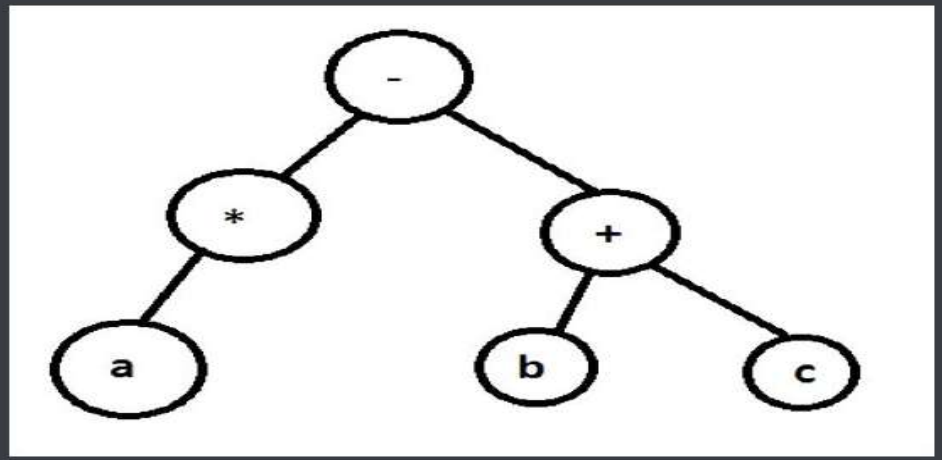
Show that no grammar that has cycles can be $\mathrm{LL}(1)$.

Explanation:

We have the infix (inorder) expression.

This will be the equivalent postfix: $a^*(bc+)-$

You have infix, you have postfix, now make the tree.



Cycle => Loop => Usage of Left Recursive Grammar. So, can't be LL(1)

Question 85: [**GATE CS 1995**]

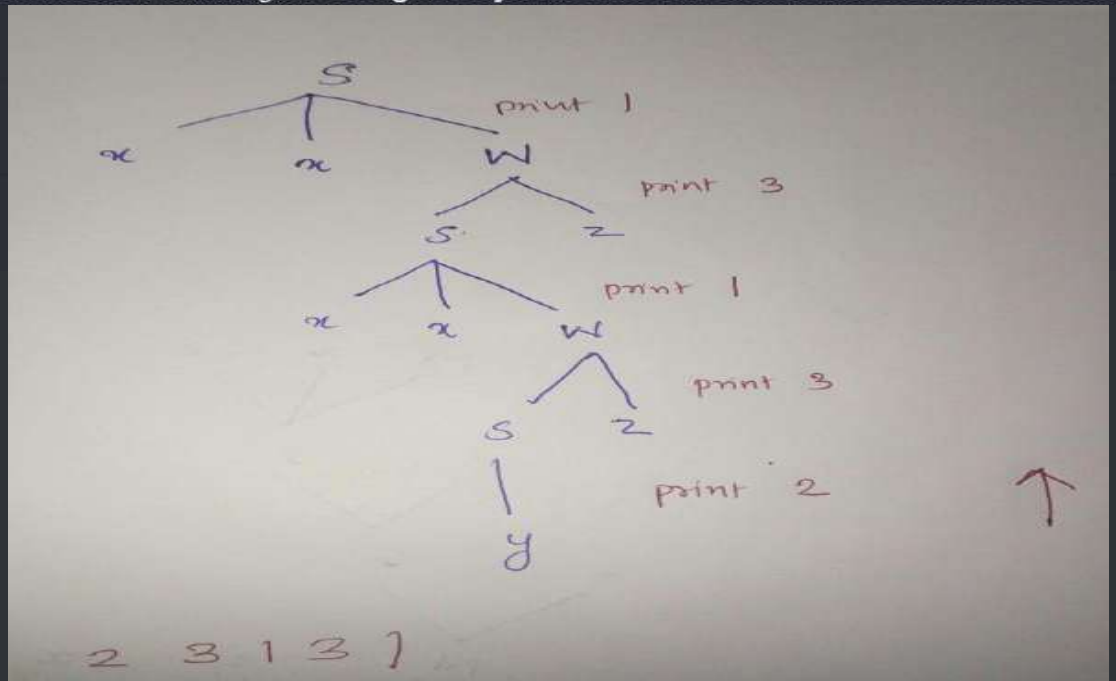
A shift reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar

- $S \rightarrow xxW$ {print "1"}
- $S \rightarrow y$ {print "2"}
- $W \rightarrow Sz$ {print "3"}

What is the translation of $xxxyzz$ using the syntax directed translation scheme described by the above rules?

A. 23131
B. 11233
C. 11231
D. 33211

Answer: A



Question 86: [**GATE CS 1995**]

Construct the $\mathrm{LL}(1)$ table for the following grammar.

1. $Expr \rightarrow \_Expr$
2. $Expr \rightarrow (Expr)$
3. $Expr \rightarrow Var\ ExprTail$
4. $ExprTail \rightarrow \_Expr$
5. $Expr \rightarrow \lambda$
6. $Var \rightarrow Id\ VarTail$
7. $VarTail \rightarrow (Expr)$
8. $VarTail \rightarrow \lambda$
9. $Goal \rightarrow Expr\$$

Answer: A

Non-Terminals: $\{Goal\ , Expr\ , ExprTail\ , Var\ , VarTail\ \}$

Terminals : $\{\_, (,), Id, \lambda\}$

Starting symbol : $\{Goal\}$ ($\because$ The end delimiter $ is at the end of it in RHS)

Rewriting the grammar

1. $Goal \rightarrow Expr\$$
2. $Expr \rightarrow \_Expr \mid (Expr) \mid Var\ ExprTail \mid \lambda$
3. $ExprTail \rightarrow \_Expr$
4. $Var \rightarrow Id\ VarTail$
5. $VarTail \rightarrow (Expr) \mid \lambda$

[cont....]

Non-terminal	First	Follow
$Goal$	$\{\_, (, Id, \lambda\}$	$\{\$\}$
$Expr$	$\{\_, (, Id, \lambda\}$	$\{\$,)\}$
$ExprTail$	$\{\_\}$	$\{\$,)\}$
Var	$\{Id\}$	$\{\_\}$
$VarTail$	$\{(, \lambda\}$	$\{\_\}$

The $\mathrm{LL}(1)$ table will be as follows:

	_	(	)	Id	λ	$
Goal	$Goal \rightarrow Expr\$$	$Goal \rightarrow Expr\$$		$Goal \rightarrow Expr\$$	$Goal \rightarrow Expr\$$	
Expr	$Expr \rightarrow \_Expr$	$Expr \rightarrow (Expr)$		$Expr \rightarrow Var\ ExprTail$	$Expr \rightarrow \lambda$	
ExprTail	$ExprTail \rightarrow \_Expr$					
Var				$Var \rightarrow Id\ VarTail$		
VarTail		$VarTail \rightarrow (Expr)$			$VarTail \rightarrow \lambda$	

Question 87: [**GATE CS 1992**]

Consider the $\mathrm{SLR}(1)$ and $\mathrm{LALR}\ (1)$ parsing tables for a context free grammar. Which of the following statement is/are true?

A. The *goto* part of both tables may be different.
B. The *shift* entries are identical in both the tables.
C. The *reduce* entries in the tables may be different.
D. The *error* entries in tables may be different

Answer:

- Goto part & shift entry must be same.
- Reduce entry & error entry may b different due to conflicts.

Correct Answer: B;C;D.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 88: **[GATE CS Mock]**

Consider the following grammar:

S → aPbSQ/a

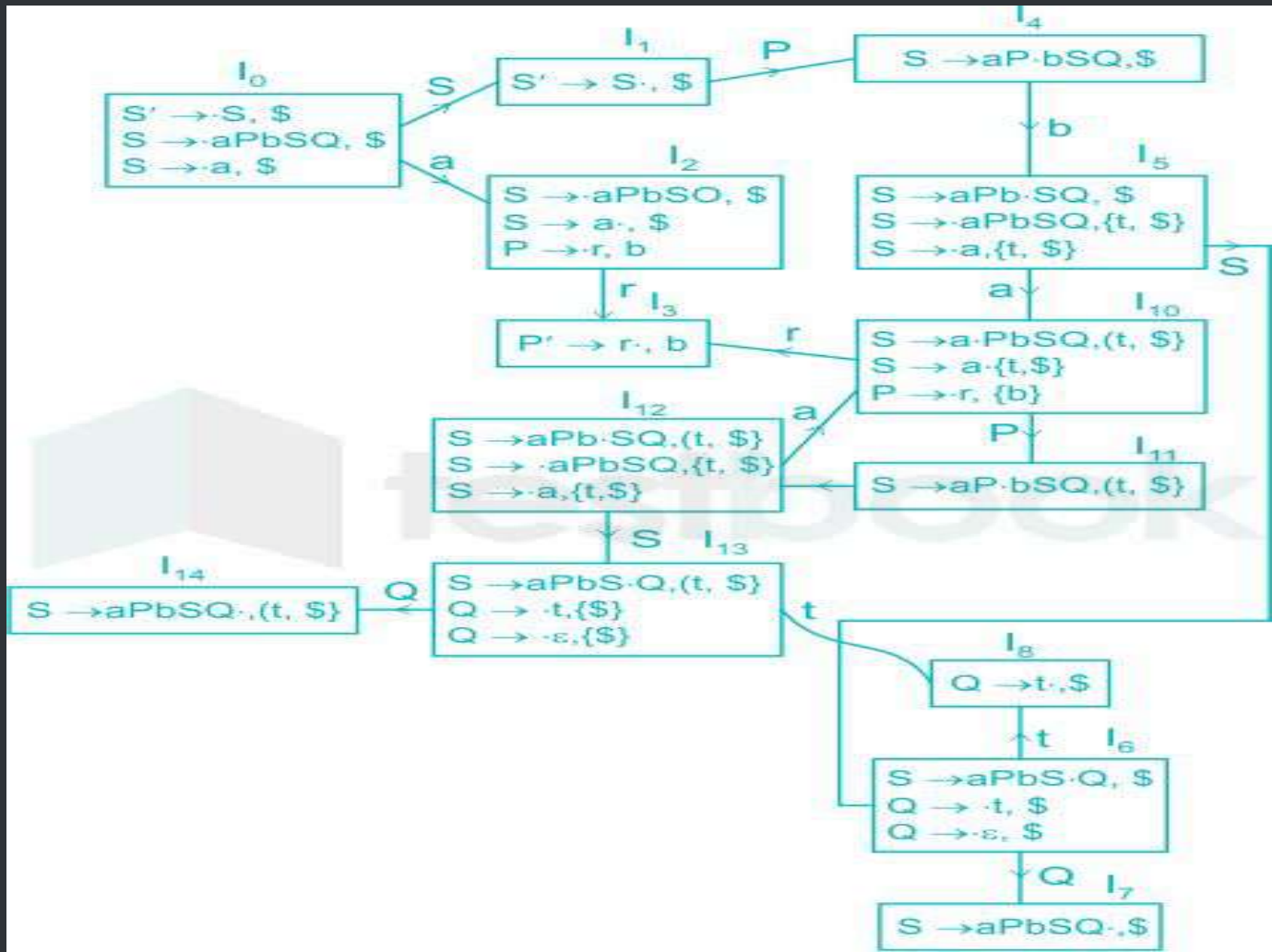
Q → t/e

P→ r

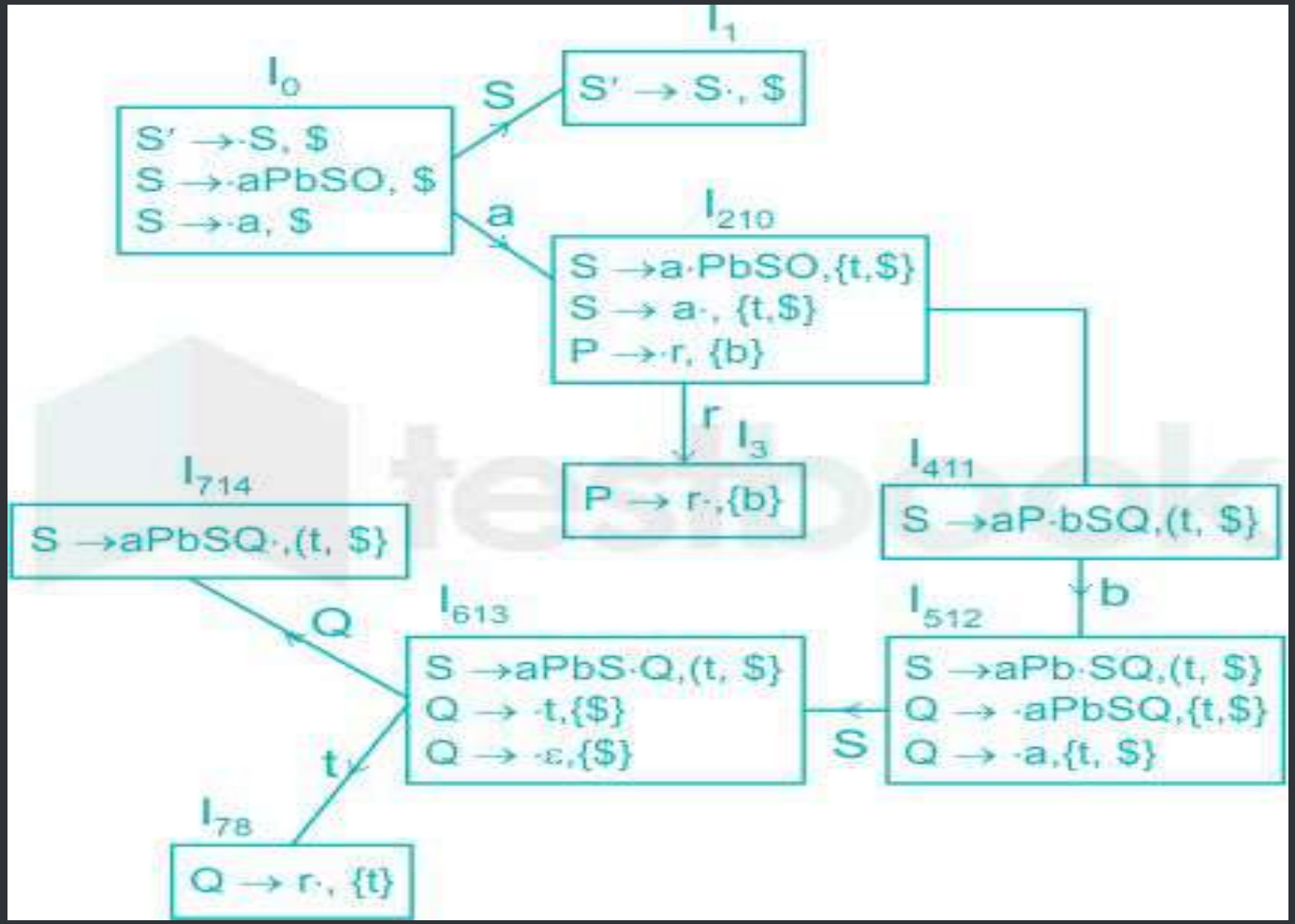
The number of states will reduce when a LALR(1) parser is computed out of CLR(1) parser of the above grammar—---?

Answer: 5

==========>



[Conti.....] The Merged States are {I_0,I_{10}}, {I_4,I_{11}}, {I_5,I_{12}}, {I_7,I_{14}}



Question 89: [**GATE CS 1992**]

Let G be a context-free grammar where $G = (\{S, A, B, C\}, \{a, b, d\}, P, S)$ with the productions in P given below.

- $S \to ABAC$
- $A \to aA \mid \varepsilon$
- $B \to bB \mid \varepsilon$
- $C \to d$

(ε denotes the null string). Transform the grammar G to an equivalent context-free grammar G' that has no ε productions and no unit productions. (A unit production is of the form $x \to y$, and x and y are non terminals).

Answer: Final grammar is

- $S \to ABAC \mid ABC \mid BAC \mid BC \mid AC \mid AAC \mid d$
- $A \to aA \mid a$
- $B \to bB \mid b$
- $C \to d$

Question 90: [**GATE CS 1992**]

A $2-3$ tree is such that

a. All internal nodes have either 2 or 3 children
b. All paths from root to the leaves have the same length

The number of internal nodes of a $2-3$ tree having 9 leaves could be

A. 4
B. 5
C. 6
D. 7

Answer:

Correct Options: **A;D**

$4 \rightarrow$ When each leaf has 3 childs. So $9/3 = 3$ Internal nodes, Then one internal node those internal nodes.

$7 \rightarrow$ When each leaf has 2 childs & one leaf out of 4 get 3 childs. Ex $\rightarrow 8/4 = 2$ child per internal node. Then one of that internal node get extra third child. Then 2 internal nodes to connect these 4. Then 1 internal node to connect this 2. So $4 + 2 + 1 = 7$.

No other way is possible.

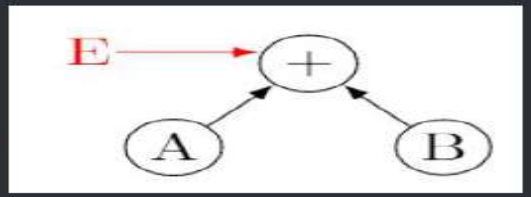
Question 91: [**GATE CS 1998**]

Construct a DAG for the following set of quadruples:

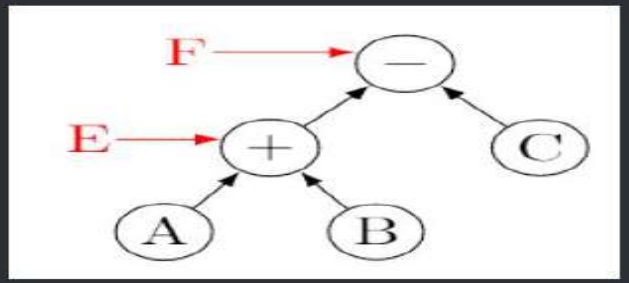
- E:=A+B
- F:=E-C
- G:=F*D
- H:=A+B
- I:=I-C
- J:=I+G

The Steps for constructing the DAG are shown below.

$I.\ E = A + B$

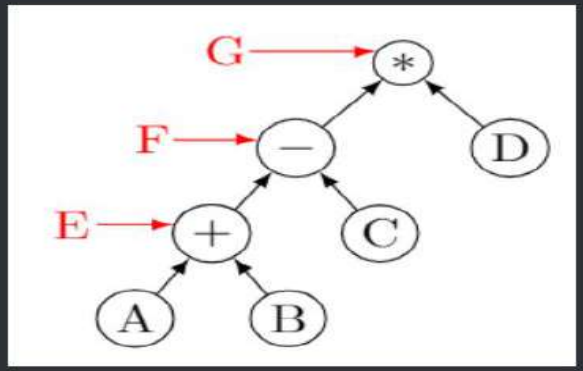


$II.\ F = E - C$

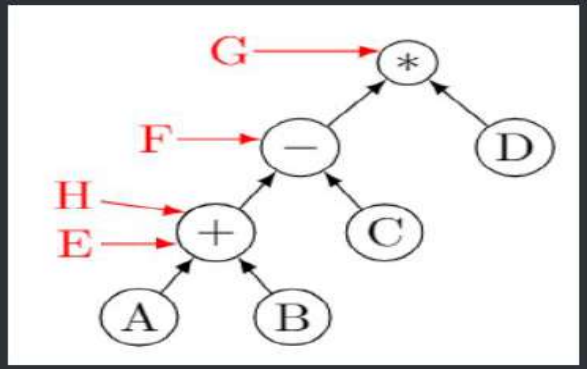


[Conti....]

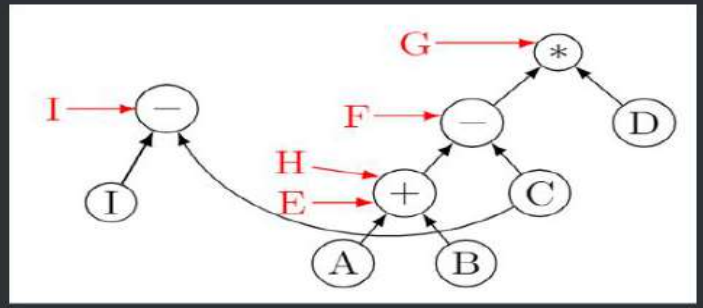
III. $G = F * D$



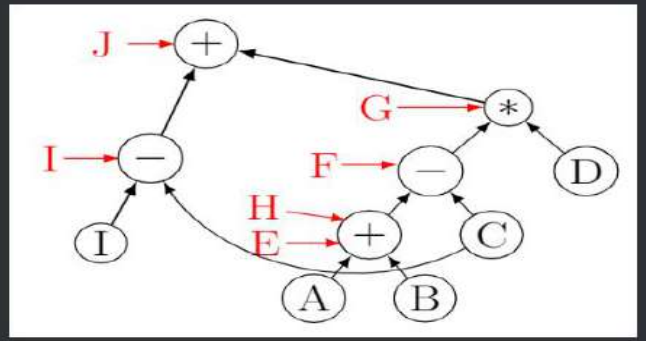
IV. $H = A + B$



V. $I = I - C$



VI. $J = I + G$



ept

Question 92: [**GATE CS 1991**]

Consider the following grammar for arithmetic expressions using binary operators $-$ and $/$ which are not associative

- $E \rightarrow E - T \mid T$
- $T \rightarrow T/F \mid F$
- $F \rightarrow (E) \mid id$

(E is the start symbol)

Is the grammar unambiguous? Is so, what is the relative precedence between $-$ and $/$? If not, give an unambiguous grammar that gives $/$ precedence over $-$.

Answer:

Yes. It is unambiguous grammar since for any string no more than 1 parse tree is possible.

For precedency draw the parse tree and find the depth of operator - and / .

Here "/" having more depth than " - " operator so precedency of" /" is higher than "-".

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 92: [**GATE CS 1991**]

Consider the following grammar for arithmetic expressions using binary operators $-$ and $/$ which are not associative

- $E \rightarrow E - T \mid T$
- $T \rightarrow T / F \mid F$
- $F \rightarrow (E) \mid id$

(E is the start symbol)

Does the grammar allow expressions with redundant parentheses as in (id/id) or in $id - (id/id)$? If so, convert the grammar into one which does not generate expressions with redundant parentheses. Do this with minimum number of changes to the given production rules and adding at most one more production rule.

Convert the grammar obtained above into one that is not left recursive.

[Conti....]

Answer:

Here we have to convert the grammar into one which does not generate expressions with redundant parentheses.So the grammar which does not generate expressions with redundant parentheses

- $E \rightarrow E - T \mid T$
- $E' \rightarrow E - T$
- $T \rightarrow id \mid T'/F$
- $T' \rightarrow F \mid T'/F$
- $F \rightarrow id \mid (E')$

Now equivalent grammar after removing left recursion

- $E \rightarrow TX'$
- $X' \rightarrow -TX' \mid \epsilon$
- $E' \rightarrow E - T$
- $T \rightarrow id \mid T'/F$
- $T' \rightarrow FY'$
- $Y' \rightarrow /FY' \mid \epsilon$
- $F \rightarrow id \mid (E')$

Question 93: [**GATE CS 2023**]

Consider the following statements regarding the front-end and back-end of a compiler.

S1: The front-end includes phases that are independent of the target hardware.
S2: The back-end includes phases that are specific to the target hardware.
S3: The back-end includes phases that are specific to the programming language used in the source code.

Identify the CORRECT option.

A. Only **S1** is TRUE.
B. Only **S1** and **S2** are TRUE.
C. **S1**, **S2**, and **S3** are all TRUE.
D. Only **S1** and **S3** are TRUE.

Answer: B

S1: The Front-End Phase of the compiler doesn't contain any hardware specific process. So, the front-end phases of the compiler are independent of target hardware. So, **S1 is true**.

S2 : The Target Code Generation phase of back-end is hardware specific. So, **S2 is also true**.

S3 : It's front-end which convert the programming code into Intermediate code. The back-end phase of the compiler has nothing to do with programming language. It is Independent of programming language. So, **S3 is false**.

Only **S1 & S2 are true**. Hence, **B is correct answer.**

Question 94: **[GATE CS 2008]**

Some code optimizations are carried out on the intermediate code because

A)they enhance the portability of the compiler to other target processors

B) program analysis is more accurate on intermediate code than on machine code

C) the information from dataflow analysis cannot otherwise be used for optimization

D) the information from the front end cannot otherwise be used for optimization

Answer: A

Explanation:

Option (B) is also true. But the main purpose of doing some code-optimization on intermediate code generation is to enhance the portability of the compiler to target processors. So Option A) is more suitable here. Intermediate code is machine/architecture independent code. So a compiler can optimize it without worrying about the architecture on which the code is going to execute (it may be the same or the other). So that kind of compiler can be used by multiple different architectures. In contrast to that, suppose code optimization is done on target code, which is machine/architecture dependent, then the compiler has be specific about the optimizations on that kind of code. In this case the compiler can't be used by multiple different architectures, because the target code produced on different architectures would be different. Hence portability reduces here.

Question 95: [**GATE CS 2008**]

Which of the following are true?

I. 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
II. Multi-level access link (or display) arrangement is needed to arrange activation records only if the programming language being implemented has nesting of procedures/functions
III. Recursion in programming languages cannot be implemented with dynamic storage allocation
IV. Nesting procedures/functions and recursion require a dynamic heap allocation scheme and cannot be implemented with a stack-based allocation scheme for activation records
V. 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

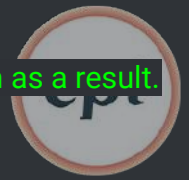
A. II and V only
B. I, III and IV only
C. I, II and V only
D. II, III and V only

Answer: A

Explanation:

I. False. Recursion cannot be implemented using static allocation.
II. True. Yes, we do need multi level access link in case of nested functions. Each level to traverse ARB of same level of nesting.
III. False. Recursion can only be implemented using dynamic memory allocation.
IV. False. Recursion is done using memory in stack (ARBs in stack), not in heap.
V. True. Yes, they cannot, once a function returns its activation record is no longer valid, so we cannot return a function as a result.

So, option (**A**) is correct.



Question 96: [**GATE CS 2007**]

In a simplified computer the instructions are:

OP R_j, R_i	- Performs R_j OP R_i and stores the result in register R_i.
OP m,R_i	- Performs *val* OP R_i and stores the result in R_i. *val* denotes the content of memory location *m*.
MOV m, R_i	- Moves the content of memory location *m* to register R_i.
MOV R_i, m	- Moves the content of register R_i to memory location *m*.

The computer has only two registers, and OP is either ADD or SUB. Consider the following basic block:

$$t_1 = a + b$$
$$t_2 = c + d$$
$$t_3 = e - t_2$$
$$t_4 = t_1 - t_3$$

Assume that all operands are initially in memory. The final value of the computation should be in memory.

What is the minimum number of MOV instructions in the code generated for this basic block?

A)2

B)3

C)5

D)6

Answer: 3

Explanation:For Instructions of t2 and t3 1. MOV c, t2 2. OP d, t2(OP=ADD) 3. OP e, t2(OP=SUB) For Instructions of t1 and t4 4. MOV a, t1 5. OP b, t1(OP=ADD) 6. OP t1, t2(OP=SUB) 7. MOV t2, a(AS END Value has To be in the MEMORY) Step 6 should have been enough, if the question hadn't asked for final value in memory and rather be in register. The final step require another MOV, thus a total of 3.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 97: **[GATE CS 2014]**

Which one of the following is FALSE?

A)A basic block is a sequence of instructions where control enters the sequence at the beginning and exits at the end.

B) Available expression analysis can be used for common subexpression elimination.

C) Live variable analysis can be used for dead code elimination.

D)x = 4 * 5 => x = 20 is an example of common subexpression elimination.

Answer: D

A basic block is a sequence of instructions where control enters the sequence at the beginning and exits at the end is TRUE. (B) Available expression analysis can be used for common subexpression elimination is TRUE. Available expressions is an analysis algorithm that determines for each point in the program the set of expressions that need not be recomputed. Available expression analysis is used to do global common subexpression elimination (CSE). If an expression is available at a point, there is no need to re-evaluate it. (C)Live variable analysis can be used for dead code elimination is TRUE. (D) x = 4 * 5 => x = 20 is an example of common subexpression elimination is FALSE. Common subexpression elimination (CSE) refers to compiler optimization replaces identical expressions (i.e., they all evaluate to the same value) with a single variable holding the computed value when it is worthwhile to do so. Below is an example

```
In the following code:

a = b * c + g;

d = b * c * e;
```

```
it may be worth transforming the code to:

tmp = b * c;

a = tmp + g;

d = tmp * e;
```

Question 98: **[GATE CS 2014]**

One of the purposes of using intermediate code in compilers is to

A) make parsing and semantic analysis simpler.

B) improve error recovery and error reporting.

C) increase the chances of reusing the machine-independent code optimizer in other compilers.

D)improve the register allocation.

answer:C

Explanation: After semantic Analysis, the code is converted into intermediate code which is platform(OS + hardware) independent, the advantage of converting into intermediate code is to improve the performance of code generation and to increase the chances of reusing the machine-independent code optimizer in other compilers. So, option (C) is correct.

Question 99: [**GATE CS 2006**]

Consider the following C code segment.

```
for (i = 0, i<n; i++)
{
   for (j=0; j<n; j++)
   {
       if (i%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 of strength reduction in this code

D) There is scope of dead code elimination in this code

Answer: D

Explanation:4*j is common subexpression elimination so B is true.

5*i can be moved out of inner loop so can be i%2.

Means, A is true as we have loop invariant computation.

Next, 4*j as well as 5*i can be replaced with a = - 4;

before j loop then a = a + 4; where 4*j is computed,

likewise for 5*i. C is true as there is scope of strength

reduction.

By choice elimination, we have D.

Question 100: [**GATE CS 2004**]

Consider the grammar rule E → E1 - E2 for arithmetic expressions. The code generated is targeted to a CPU having a single user register. The subtraction operation requires the first operand to be in the register. If E1 and E2 do not have any common sub expression, in order to get the shortest possible code

A)E1 should be evaluated first

B)E2 should be evaluated first

C) Evaluation of E1 and E2 should necessarily be interleaved

D)Order of evaluation of E1 and E2 is of no consequence

Answer : B

Explanation: E -> E1 - E2 Given that E1 and E2 don't share any sub expression, most optimized usage of single user register for evaluation of this production rule would come only when E2 is evaluated before E1. This is because when we will have E1 evaluated in the register, E2 would have been already computed and stored at some memory location. Hence we could just use subtraction operation to take the user register as first operand, i.e. E1 and E2 value from its memory location referenced using some index register or some other form according to the instruction. Hence correct answer should be (B) E2 should be evaluated first.

Question 101: **[GATE CS 2015]**

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 = i + 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

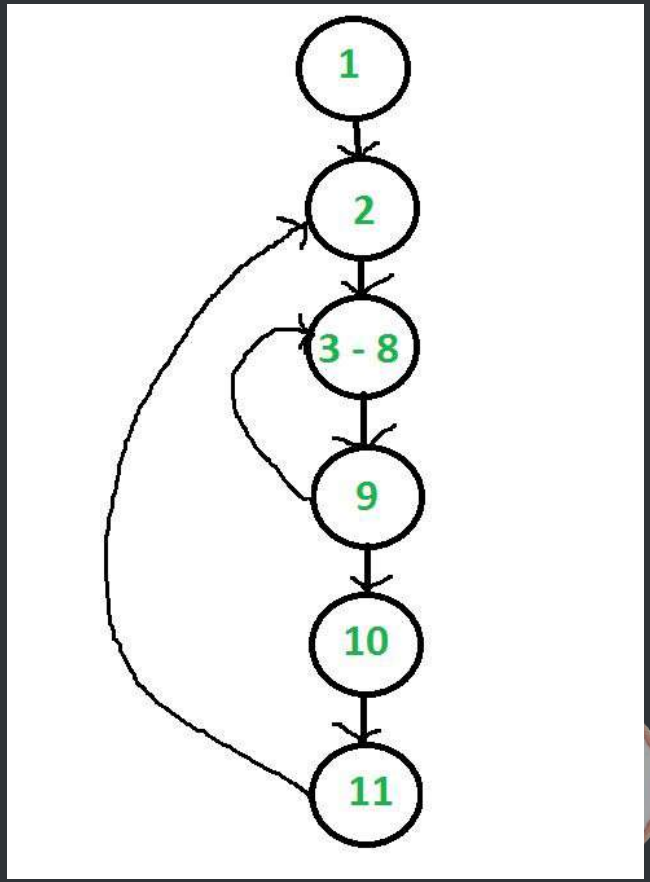
B) 6 and 7

C)5 and 5

D

D) 7 and 8

Answer: B



Question 102: [**GATE CS 2016**]

Consider the following code segment.

```
x = u - t;
y = x * v;
x = y + w;
y = t - z;
y = x * y;
```

The minimum number of total variables required to convert the above code segment to static single assignment form is

A) 6

B) 8

C) 9

D)10

Answer: D

Static Single Assignment is used for intermediate code in compiler design. In Static Single Assignment form(SSA) each assignment to a variable should be specified with distinct names. We use subscripts to distinguish each definition of variables. In the given code

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

segment, there are two assignments of the variable x

```
x = u - t;
x = y + w;
```

and three assignments of the variable y.

```
y = x * v;
y = t - z;
 y = x * y
```

So we use two variables x1, x2 for specifying distinct assignments of x and y1, y2 and y3 each assignment of y. So, total number of variables is 10 (x1, x2, y1, y2, y3, t, u, v, w, z). Static Single Assignment form(SSA) of the given code segment is:

```
x1 = u - t;
y1 = x1 * v;
x2 = y1 + w;
y2 = t - z;
y3 = x2 * y2;
```

Question 103: [**GATE CS 1999**]

A grammar that is both left and right recursive for a non-terminal is

A)Ambiguous

B)Unambiguous

C)Information is not sufficient to decide whether it is ambiguous or Unambiguous.

D)None of the above

Answer: C

Explanation:Suppose we have grammar like this :

S → n

B → BbB

Here we see that grammar is left as well as right recursive but still it is unambiguous grammar because A is useless production but it is still part of grammar. So we can say that a grammar having both left as well as right recursion may or may not be ambiguous . Lets understand with another example as we have grammar like this A→AA using this grammar we can not produce any string in finite steps as language of this grammar is empty set {}. Hence, we finally get conclusion as if grammar is having both left as well as right recursion, then grammar may or may not be ambiguous. Option (C) is correct

Question 104: [**GATE CS 1999**]

The least number of temporary variables required to create a three-address code in static single assignment form for the expression a=b * d - c + b * e - c is _____

Answer: 4

Explanation:

a=b * d - c + b * e - c

t1 = b * d

t2 = b * e

t3 = t1 + t2

t4 = t3 -c

a= t4 -c

Total temp variables = 4

Question 105: [**GATE CS 1996**]

The grammar whose productions are

- ⟨stmt⟩→ if id then ⟨stmt⟩
-
- ⟨stmt⟩→ if id then ⟨stmt⟩ else ⟨stmt⟩
-
- ⟨stmt⟩→ id := id

is ambiguous because

(a) the sentence

`if a then if b then c:= d` has more than two parse trees

(b) the left most and right most derivations of the sentence

`if a then if b then c:= d` give rise to different parse trees

(c) the sentence

`if a then if b then c:= d else c:= f` has more than two parse trees

(d) the sentence

`if a then if b then c:= d else c:= f` has two parse trees

Answer: (d)

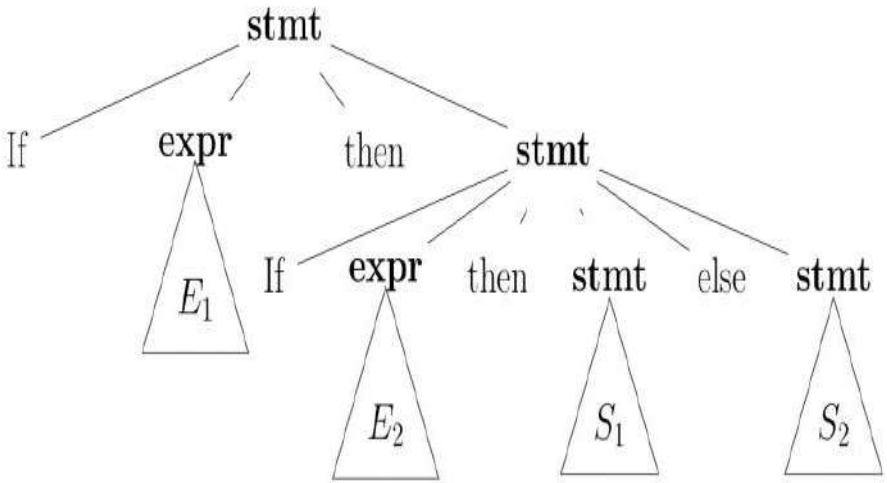
if a then if b then c: = d else c:= f has two parse trees as follows:

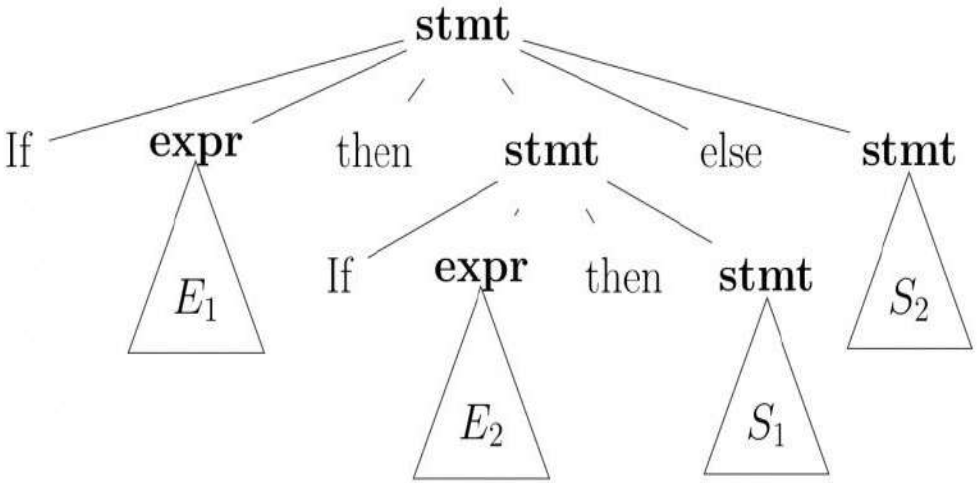
1. if a then (if b then c:= d) else c:= f
2. and if a then (if b then c:=d else c:= f)

Ambiguity - “dangling else"

```
stmt -> if expr then stmt |
        if expr then stmt else stmt | other stmts
```

if E1 then if E2 then S1 else S2





Question 106: **[GATE CS 1998]**

In a resident- OS computer, which of the following system software must reside in the main memory under all situations?

A)Assembler

B)Linker

C)Loader

D)Compiler

Answer: C

Explanation: Loader is the part of an operating system that is responsible for loading programs and libraries. it places the programs into memory and also prepares them for execution. Loading a program involves tasks such as reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running so any software must reside in the main memory under all situations. The operating system starts the program by passing control to the loaded program code once the loading process is completed. Option (C) is correct.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 107: [**GATE CS 1998**]

A linker reads four modules whose lengths are 200, 800, 600 and 500 words respectively. If they are loaded in that order, what are the relocation constants?

A)0, 200, 500, 600

B)0, 200, 1000, 1600

C)200, 500, 600, 800

D)200, 700, 1300, 2100

Answer: B

Explanation:

according to question a linker reads four modules whose lengths are 200, 800, 600 and 500 words respectively. If first module loaded then it will start at address 0 and we know that Size is 200. hence it will occupy first 200 address and last address being 199 because it start with 0 therefore Second module will be present from 200 to 999 as 2nd module having length 800 and third module will start from 1000 to 1599 as its length is 600 similarly fourth module will start from 1600 to till 500 B. Therefore relocation constant is 0, 200, 1000, 1600. For better understanding see the table below.

Module Number	relocation base	limit (length)
1	0	200
2	200	800
3	1000	600
4	1600	500

Question 108: **[GATE CS 1997]**

A language L allows declaration of arrays whose sizes are not known during compilation. It is required to make efficient use of memory. Which of the following is true?

A)A compiler using static memory allocation can be written for L

B)A compiler cannot be written for L, an interpreter must be used

C)A compiler using dynamic memory allocation can be written for L

D)None of the above

Answer: C

Explanation:If a language L allows declaration of arrays whose sizes are not known during compilation time. It is required to use efficient use of memory. So a compiler using dynamic memory allocation can be written for L. An array is a collection of data items, all of the same type, accessed using a common name. C dynamic memory allocation refers to performing manual memory management for dynamic memory allocation in the C programming language via a group of functions in the C standard library, namely malloc, realloc, calloc and free.

Question 109: [**GATE CS 1997]**

The expression *(a*b)* c* op..... where 'op' is one of '**+**', '*****' and '↑' (exponentiation) can be evaluated on a CPU with a single register without storing the value of *(a * b)* if

A)'op' is '+' or '*'

B) 'op' is '↑' or '*'

C) 'op' is '↑' or '+'

D) not possible to evaluate without storing

Answer: A

Explanation: Given expression is :-

(a*b)* c op

Here op is one of the '+', '*' and '↑' (exponentiation). (a* b)* having high precedence so it will evaluate first in CPU register. But we have given one single register as we cannot store any value from reg to memory. Now (a * b) is evaluated in register R and precedence order is (↑ , * or / , + or –). If we put op as (a*b)* c op ' ↑ ' then expression becomes as (a*b)* c ↑ d here c ↑ d will evaluate first. But we have not extra register to evaluate (a * b). Therefore we cannot put any operator having precedence greater than " * ". Hence, Operator is either " + " or "- " .

Question 110: [**GATE CS 1997]**

Consider the grammar

S→ bSe

S→ PQR

P→ bPc

P→ ε

Q→ cQd

Q→ ε

R→ dRe

R→ ε

where S,P,Q,R are non-terminal symbols with S being the start symbol; *b,c,d,e* are terminal symbols and 'ε' is the empty string. This grammar generates strings of the form b_i, c_j, d_k, e_m for some i, j, k, m ≥ 0.

- **(a).** What is the condition on the values of *i, j, k, m* ?
- **(b).** Find the smallest string that has two parse trees

Explanation:(a). Condition on the values of i, j, k, m i+k=j+m+k i+k = j+m

where i, j, k, m >= 0

(b). Smallest string that has two parse trees = bcde
Production used to generate the smallest string is :-

```
S- > bSe
S- > bSe
S- > bSe
S -> PQR
P-> null
Q ->cQd
Q -> nll
  R- > null
```

Finally you will get the string like " bbbcdeee"
which means that i=3, j=1, k=1, m=3 and hence the answer i+k=j+m .

Now production used to generate the smallest string is :- `bSe`

```
S ->  bPQRe
S ->  bεQRe
S ->  bcQdRe
S ->  bcεdRe
S ->  bcdεe
  S -> bcde
```

Hence smallest string is bcde.
Therefore we can see same number of b, c, d, e is generated.
Power of b, c, d, e are respectively i, j, k, m.
Hence relation between i, j, k, m is :-

```
  i+k = j+m
```

Question 111: [**GATE CS 2018]**

Consider the following expression `u*v+a-b*c` Which one of the following corresponds to a static single assignment from the above expressions?

A) $x_1 = a - b\ y_1 = p * c\ x_2 = u * v\ y_2 = p + q$

B) $x_1 = a - b\ y_1 = x_2 * c\ x_3 = u * v\ y_2 = x_4 + y_3$

C) $x_1 = a - b\ y_2 = x_1 * c\ x_2 = u * v\ y_3 = x_2 + y_2$

D) $p = a - b\ q = p * c\ p = u * v\ q = p + q$

Answer: C

Explanation: According to Static Single Assignment

1. A variable cannot be used more than once in the LHS.
2. A variable should be initialized atmost once.

So, only option (C) is correct.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 112: [**GATE CS 2018**]

Consider the following source code :

```
c = a + b
d = c
c = c - e
a = d - e
b = b * e
b = d/b
```

Which of the following is correct optimization of given code?

A)
```
c = a + b
t = b * e
a = d - e
b = d/t
c = a
```

B)
```
c = a + b
    d = c
    c = c - e
    a = d - e
    b = d/b
```

C)
```
d = c
    c = c - e
    a = d - e
    b = b * e
     b = d/b
```

D) None of the above

Answer: D

Explanation: A) It modified by a = d - e, but d must contain a+b. (B) It used old value of b but b = b*e. (C) Variable d used old value of c, but d = a+b. All optimizations are not correct.

Question 113: [**GATE CS 2021**]

For a statement S in a program, in the context of liveness analysis, the following sets are defined:

$\mathrm{USE}(S)$: the set of variables used in S

$\mathrm{IN}(S)$: the set of variables that are live at the entry of S

$\mathrm{OUT}(S)$: the set of variables that are live at the exit of S

Consider a basic block that consists of two statements, S_1 followed by S_2. Which one of the following statements is correct?

A. $\mathrm{OUT}(S_1) = \mathrm{IN}\ (S_2)$
B. $\mathrm{OUT}\ (S_1) = \mathrm{IN}\ (S_1) \cup \mathrm{USE}\ (S_1)$
C. $\mathrm{OUT}\ (S_1) = \mathrm{IN}\ (S_2) \cup \mathrm{OUT}\ (S_2)$
D. $\mathrm{OUT}\ (S_1) = \mathrm{USE}\ (S_1) \cup \mathrm{IN}\ (S_2)$

Answer: A

Explanation: When a basic block S_2 immediately follows another basic block S_1, all variables that are live at exit of S_1 must be live at entry of S_2 (no intermediate place where they can get killed) and no other variable can be live at entry of S_2 as a basic block is always **single entry** and **single exit**.

So, $\mathrm{OUT}(S_1) = \mathrm{IN}\ (S_2)$

Correct option: A

Question 114: **[GATE CS 2021]**

Consider the following C code segment:

```
a = b + c;
e = a + 1;
d = b + c;
f = d + 1;
g = e + f;
```

In a compiler, this code segment is represented internally as a directed acyclic graph (DAG) The number of nodes in the DAG is—----------

Answer: 6

Explanation:

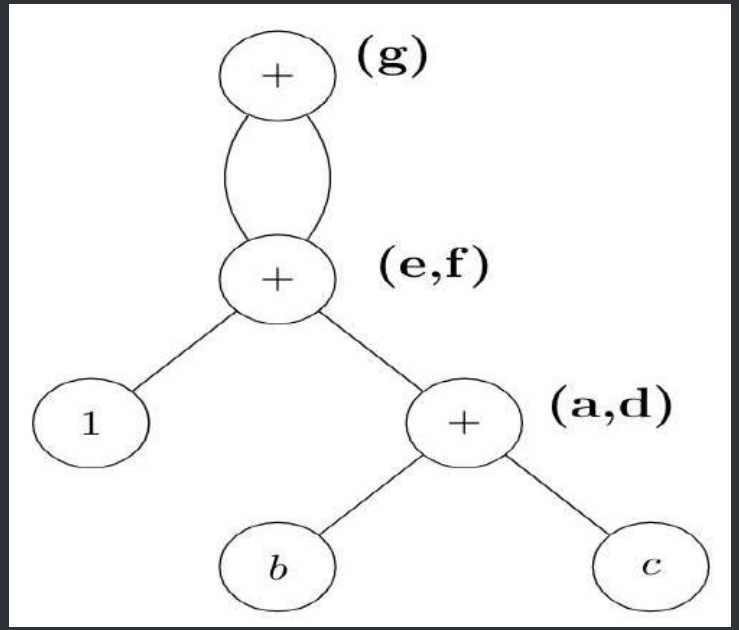
Here a and d are same as both add same values (bc) (common sub-expression elimination)

Since a and d are same f and e are also same as they compute $a+1$ and $d+1$ respectively.

- $a = d = b + c$
- $e = f = a + 1$
- $g = e + e$ (f and e being same)

So total no of nodes is 6 $(a, b, c, e, 1, g)$

Ans : 6 nodes

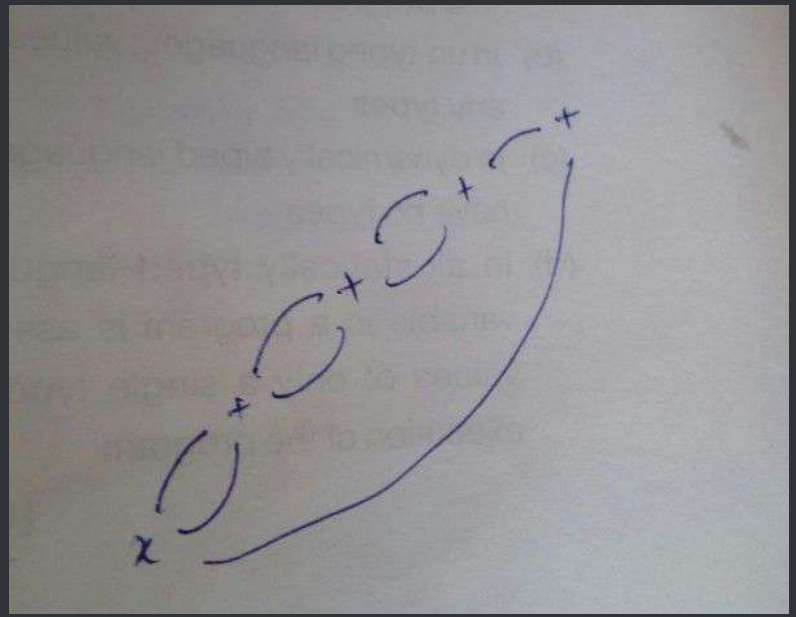


Question 115: [**GATE CS Mock]**

Minimum number of edges in the dag that represents the expression : x + x + x + x + x + x + x + x + x

Answer: 8 Edges and 5 Nodes

Explanation:



Question 116: **[GATE CS 2021]**

Consider the following ANSI C code segment:

```
z=x + 3 + y->f1 + y->f2;
for (i = 0; i < 200; i = i + 2)
{
   if (z > i)
   {
       p = p + x + 3;
       q = q + y->f1;
   } else
   {
       p = p + y->f2;
       q = q + x + 3;
   }
}
```

Assume that the variable Y points to a struct (allocated on the heap) containing two fields f1 and f2, and the local variables

x, y, z, p, q, and i are allotted registers. Common sub-expression elimination (CSE) optimization is applied on the code. The number of addition and the dereference operations (of the form y ->f1 or y ->f2) in the optimized code, respectively, are:

A) 403 and 102
B) 202 and 2
C) 303 and 102
D) 303 and 2

Answer: D

```
t1 = x + 3 // 1 addition
t2 = y->f1; // 1 dereference
t3 = y->f2; // 1 dereference
z = t1 + t2 + t3 // 2 additions
for (i = 0; i < 200; i += 2) {
    if (z > i) {
      p = p + t1; // 1 addition
      q = q + t2; // 1 addition
    } else {
      p = p + t3; // 1 addition
      q = q + t1; // 1 addition
    }
}
```

Whether we take if or else block we get 2 additions, the loop runs exactly 200/2=100 times, so from loop we get 2×100=200 additions plus

100 additions for incrementing the value of i , before loop we had perform 3 additions, so total additions 303.We only do two de-reference outside the for loop, so total de-references =2.

Question 117: **[GATE CS Mock]**

Consider the following CFG:

$E \rightarrow SaA \mid aS$

$S \rightarrow bA \mid A$

$A \rightarrow aA \mid a$

(Where E, S, A are nonterminal and a, b are terminals) The above grammar is

a. LALR(1) but not SLR(1)

b. Not LALR(1) but CLR(1)

c. Only CLR(1)

d. None of these

Answer: D

It is not SLR(1) asfollow(A) ∩ first(A) ≠Phi

As they were having SR conflict but even when you put reduce moves in Follow(A) it has SR conflict again

Even after adding lookahead, we are not able to resolve SR conflict.

SO it is not LALR, not CLR, etc.

It is none of these

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 118: [**GATE CS Mock**

S→(X

S→E]

S→E)

X→E)

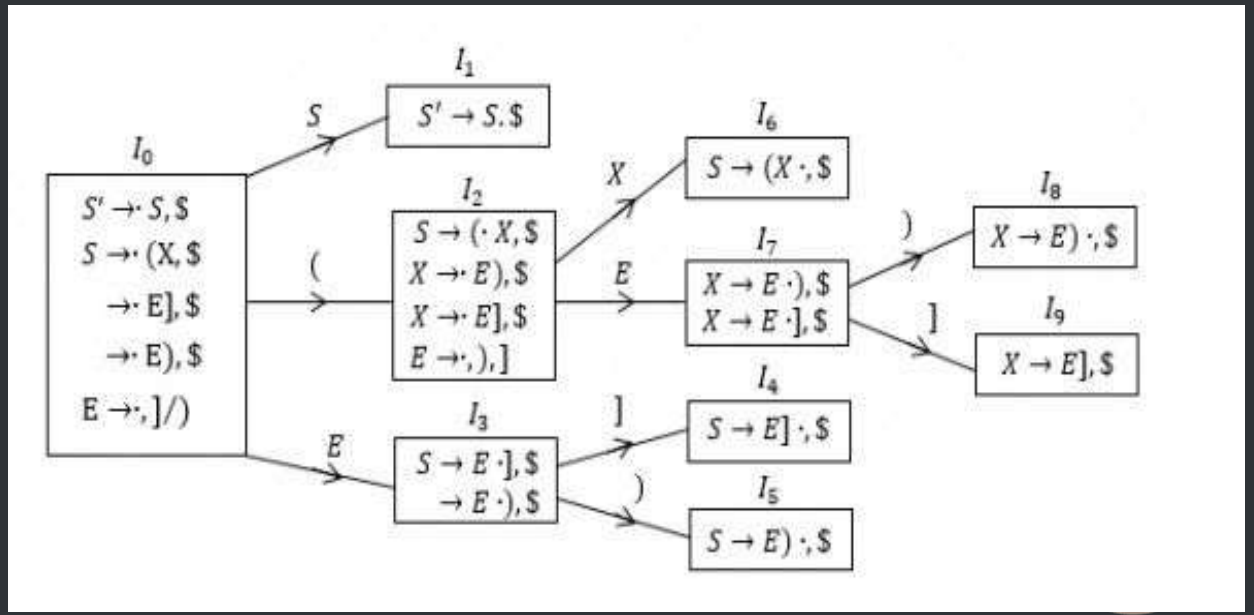
X→E]

E→ϵ

Is this grammar CLR(1)?

Answer: Yes, Given grammar is CLR(1).

Explanation: LR 1 for given grammar is



Question 119: [**GATE CS 2015**]

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

Answer: B

Explanation: A Syntax Directed Definition (SDD) is called S Attributed if it has only synthesized attributes.

L-Attributed Definitions contain both synthesized and inherited attributes but do not need to build a dependency graph to evaluate them.

Question 120: [**GATE CS 2003**]

Consider the grammar shown below

```
S → i E t S S' | a

S' → e S | ε

 E → b
```

In the predictive parse table. M, of this grammar, the entries M[S', e] and M[S', $] respectively are]?

A){S' → e S} and {S' → e}

B){S' → e S} and {}

C){S' → ε} and {S' → ε}

D){S' → e S, S'→ ε} and {S' → ε}

Answer: D

Explanation: Here representing the parsing table as M[X , Y], where X represents rows(Non terminals) and Y represents columns(terminals). Here are the rules to fill the parsing table.

For each distinct production rule A->α, of the grammar, we need to apply the given rules: **Rule 1**: if A –> α is a production, for each terminal ‘a’ in FIRST(α), add A–>α to M[A , a]
Rule 2 : if ‘ ε ‘ is in FIRST(α), add A –> α to M [A , b] for each ‘b’ in FOLLOW(A). As Entries have been asked corresponding to Non-Terminal S', hence we only need to consider its productions to get the answer. **For S' → eS**, according to rule 1, this production rule should be placed at the entry M[S', FIRST(eS)], and from the given grammar, FIRST(eS) ={e}, **hence S'->eS is placed in the parsing table at entry M[S' , e].** Similarly, **For S'->ε**, as FIRST(ε) = {ε}, hence rule 2 should be applied, therefore, this production rule should be placed in the parsing table at entry M[S',FOLLOW(S')], and FOLLOW(S') = FOLLOW(S) = { e, $ }, **hence R->ε is placed at entry M[S', e] and M[S' , $].** Therefore Answer is option D.

Question 121: [**GATE CS 2003**]

Consider the grammar shown below.

```
S → C C

C → c C | 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)

Answer: D

Explanation: Since there is no conflict, the grammar is LL(1). We can construct a predictive parse table with no conflicts. This grammar also LR(0), SLR(1), CLR(1) and LALR(1).

Question 122: [**GATE CS 2003**]

Consider the translation scheme shown below

```
S → T R

R → + T {print ('+');} R | ε

T → 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

Answer: B

Explanation: Let us make the parse tree for 9+5+2 in top down manner, left first derivation.

```
Steps:
 1) Exapnd S->TR
 2) apply T->Num...
```

```
3) apply R -> +T...
4) appy T->Num...
5) apply R-> +T..
6) apply T-> Num..
 7) apply R-> epsilon
```

After printing through the print statement in the parse tree formed you will get the answer as 95+2+

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 123: [**GATE CS 2014**]

A canonical set of items is given below

```
S --> L. > R

Q --> 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.

Answer: D

Explanation: The question is asked with respect to the symbol '<' which is **not present** in the given canonical set of items. Hence it is neither a shift-reduce conflict nor a reduce-reduce conflict on symbol '<'. Hence **D** is the correct option.

But if the question would have asked with respect to the symbol '>' then it would have been a shift-reduce conflict.

Question 124: **[GATE CS 2003]**

Consider the syntax directed definition shown below.

```
S → id : = E   {gen (id.place = E.place;);}

E → E1 + E2    {t = newtemp ( ); gen (t = E1.place + E2.place;); E.place = t}

E → 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 ti'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)t1 = Y + Z; X = t1

C)t1 =Y; t2 = t1 + Z; X = t2

D)t1 = Y; t2 = Z; t3 = t1 + t2; X = t3

Answer: B

Explanation: It must be B. The production E --> E + E is used only one time and hence only one temporary variable is generated.

Question 125: **[GATE CS 2004]**

Which of the following grammar rules violate the requirements of an operator grammar ? P, Q, R are nonterminals, and r, s, t are terminals.

1. P → Q R
2. P → Q s R
3. P → ε
4. P → Q t R r

A) 1 only

B)1 and 3 only

C)2 and 3 only

D)3 and 4 only

Answer: B

Explanation: Operator grammar should not have two or more variables in its production side by side, and should not have null productions.

Question 126: **[GATE CS 2004]**

The least number of temporary variables required to create a three-address code in static single assignment form for the expression

q+r/3+s−t∗5+u∗v/w is_____.

Answer: 8

Explanation: In compiler design, **static single assignment form** (often abbreviated as **SSA form** or simply **SSA**) is a property of an intermediate representation (IR), which requires that each variable is assigned exactly once, and every variable is defined before it is used. Existing variables in the original IR are split into *versions*, new variables.

We will need a temporary variable for storing the result of each binary operation as SSA (Static Single Assignment) implies the variable cannot be repeated on LHS of assignment.

$$q + r/3 + s - t * 5 + u * v/w$$

$$\begin{aligned} t1 &= r/3; \\ t2 &= t * 5; \\ t3 &= u * v; \\ t4 &= t3/w; \\ t5 &= q + t1; \\ t6 &= t5 + s; \\ t7 &= t6 - t2; \\ t8 &= t7 + t4 \end{aligned}$$

Question 127: [**GATE CS 2004**]

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

```
E → E1 # T { E.value = E1.value * T.value }
        | T{ E.value = T.value }
T → T1 & F { T.value = T1.value + F.value }
         | F{ T.value = F.value }
 F → 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

Answer: C

Question 128: [**GATE CS 2005**]

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

```
E → number    E.val = number. val
| E '+' E     E(1).val = E(2).val + E(3).val
  | E '×' E   E(1).val = E(2).val × 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

Answer: C

Explanation: **Background** yacc conflict resolution is done using following rules: shift is preferred over reduce while shift/reduce conflict. first reduce is preferred over others while reduce/reduce conflict. You can answer to this question straightforward by constructing LALR(1) parse table, though its a time taking process. To answer it faster, one can see intuitively that this grammar will have a shift-reduce conflict for sure. In that case, given this is a single choice question, (C) option will be the right answer.
Fool-proof explanation would be to generate LALR(1) parse table, which is a lengthy process. Once we have the parse table with us, we can clearly see that i. reduce/reduce conflict will not arise in the above given grammar ii. shift/reduce conflict will be resolved by giving preference to shift, hence making the expression calculator right associative. According to the above conclusions, only correct option seems to be (C).

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 129: **[GATE CS 2005]**

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

```
E → number        E.val = number. val
    | E '+' E     E(1).val = E(2).val + E(3).val
     | E '×' E    E(1).val = E(2).val × E(3).val
```

Assume the conflicts in Part (a) of this question are resolved and an LALR(1) parser is generated for parsing arithmetic expressions as per the given grammar. Consider an expression 3 × 2 + 1. What precedence and associativity properties does the generated parser realize?

A)Equal precedence and left associativity; expression is evaluated to 7

B)Equal precedence and right associativity; expression is evaluated to 9

C)Precedence of '×' is higher than that of '+', and both operators are left associative; expression is evaluated to 7

D)Precedence of '+' is higher than that of '×', and both operators are left associative; expression is evaluated to 9

Answer: B

Explanation: Answer is B as the productions belong to the same non-terminal and since YACC resolves by shift over reduce, the associativity will be right associative.

Question 130: [**GATE CS 2006**]

Consider the following grammar.

```
S -> S * E
S -> E
E -> F + E
E -> F
 F -> id
```

Consider the following LR(0) items corresponding to the grammar above.

```
(i) S -> S * .E
(ii) E -> F. + E
 (iii) E -> F + .E
```

Given the items above, which two of them will appear in the same set in the canonical sets-of-items for the grammar?

A)(i) and (ii)

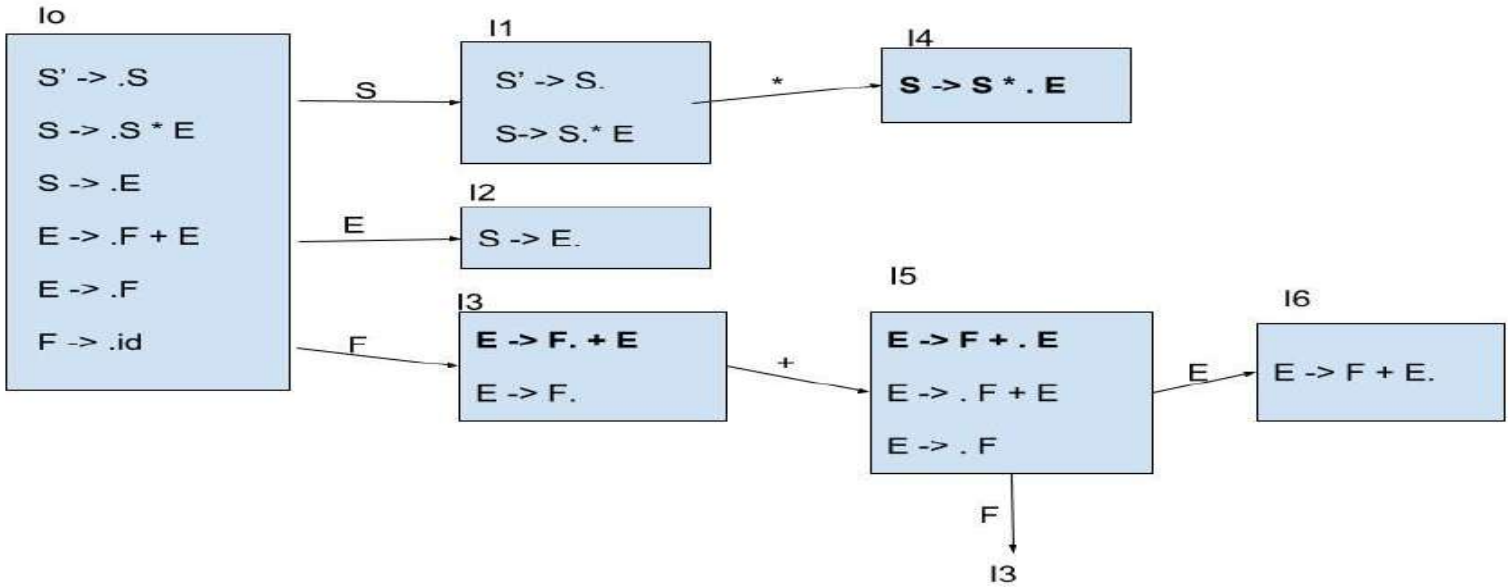
B)(ii) and (iii)

C)(i) and (iii)

D)None of the above

Answer: D

Explanation: Let's make the LR(0) set of items. First we need to augment the grammar with the production rule S' -> .S , then we need to find closure of items in a set to complete a set. Below are the LR(0) sets of items.



In the LR(0) sets of items formed so far, we can see that the given three items belong to the different sets (I3, I4, I5).

Here, S' -> .S is an augmented production rule.

Question 131: **[GATE CS 2006]**

Consider the following translation scheme. S → ER R → *E{print("*");}R | ε E → F + E {print("+");} | F F → (S) | id {print(id.value);} Here id is a token that represents an integer and id.value represents the corresponding integer value. For an input '2 * 3 + 4', this translation scheme prints

A)2 * 3 + 4

B)2 * +3 4

C)2 3 * 4 +

D)2 3 4+*

Answer: D

Explanation : We are given L-Attributed Syntax Directed Translation as

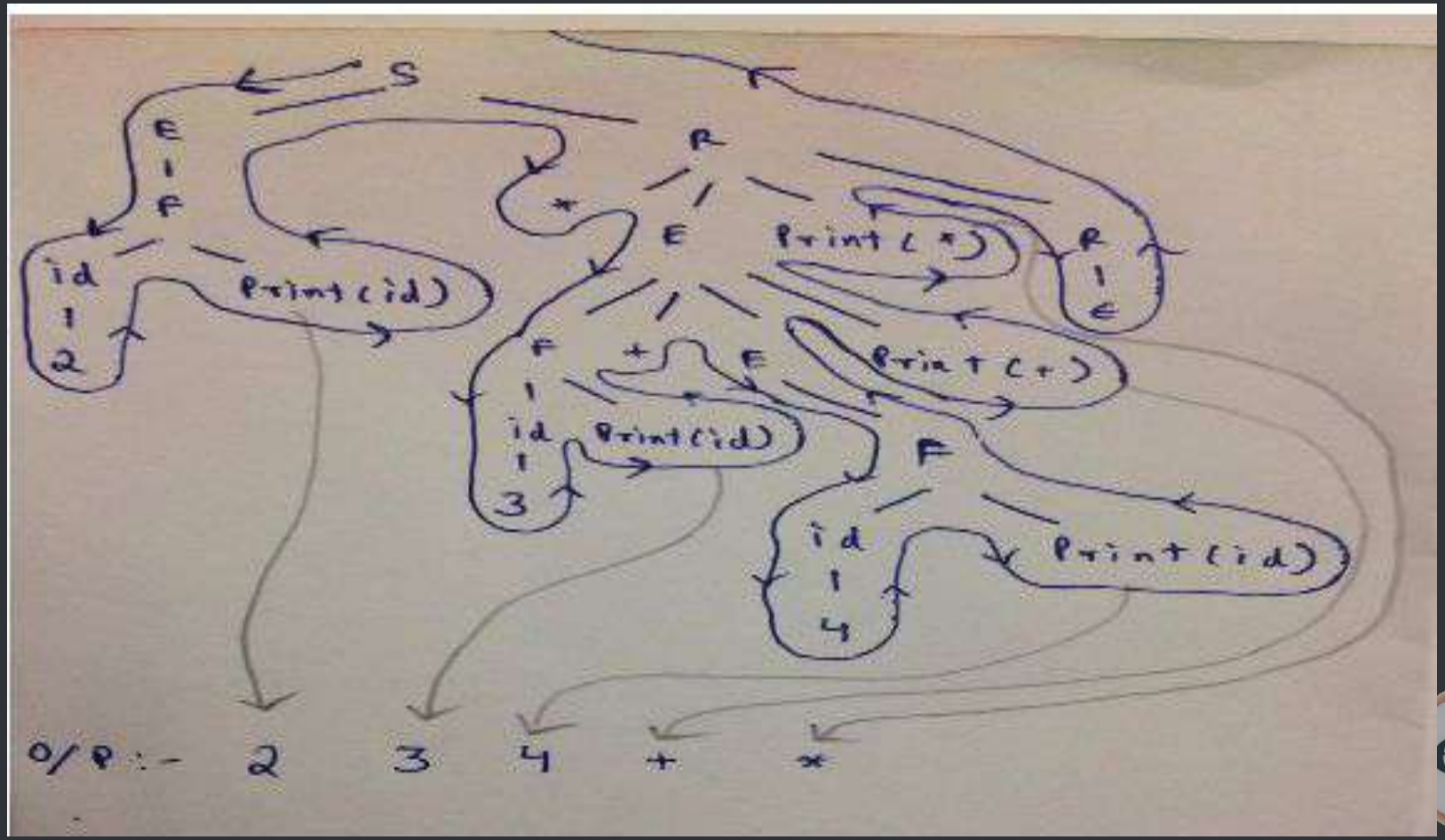
semantic actions like printf statements are inserted anywhere on the

RHS of production (R → *E{print(“*”);}R). After constructing the parse tree

as shown below from the given grammar, we will follow depth first order left

to right evaluation in order to generate the final output.

Just follow the arrows in the picture (This is actually Depth first left to right evaluation) and the moment we take exit from any child which is printf statement in this question, we print that symbol which can be a integer value or '*' or '+'.



Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 132: [**GATE CS 2006**]

Consider the following C code segment.

```
for (i = 0, i<n; i++)
{
   for (j=0; j<n; j++)
   {
       if (i%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 of strength reduction in this code

D)There is scope of dead code elimination in this code

Answer: D

Explanation: 4*j is common subexpression elimination so B is true.

5*i can be moved out of inner loop so can be i%2.

Means, A is true as we have loop invariant computation.

Next, 4*j as well as 5*i can be replaced with a = - 4;

before j loop then a = a + 4; where 4*j is computed,

likewise for 5*i. C is true as there is scope of strength reduction.

By choice elimination, we have D.

Question 133: **[GATE CS 2007]**

An LALR(1) parser for a grammar G can have shift-reduce (S-R) conflicts if and only if

A)the SLR(1) parser for G has S-R conflicts

B)the LR(1) parser for G has S-R conflicts

C)the LR(0) parser for G has S-R conflicts

D)the LALR(1) parser for G has reduce-reduce conflicts

Answer: B

Explanation: Both LALR(1) and LR(1) parser uses LR(1) set of items to form their parsing tables. And LALR(1) states can be find by merging LR(1) states of LR(1) parser that have the same set of first components of their items. i.e. if LR(1) parser has 2 states I and J with items A->a.bP,x and A->a.bP,y respectively, where x and y are look ahead symbols, then as these items are same with respect to their first component, they can be merged together and form one single state, let's say K. Here we have to take union of look ahead symbols. After merging, State K will have one single item as A->a.bP,x,y . This way LALR(1) states are formed (i.e. after merging the states of LR(1)). Now, S-R conflict in LR(1) items can be there whenever a state has items of the form :

A-> a.bB , p

C-> d. , b

i.e. it is getting both shift and reduce at symbol b,

hence a conflict.

Now, as LALR(1) have items similar to LR(1) in terms of their first component, shift-reduce form will only take place if it is already there in LR(1) states. If there is no S-R conflict in LR(1) state it will never be reflected in the LALR(1) state obtained by combining LR(1) states. Note: But this process of merging may introduce R-R conflict, and then the Grammar won't be LALR(1).

Question 134: [**GATE CS 2017**]

Consider the following grammar

```
p --> xQRS
Q --> yz|z
R --> w|∈
S -> y
```

Which is FOLLOW(Q)?

A){R}

B){w}

C){w, y}

D){w, ∉}

Answer: C

Explanation: follow(Q) = First(RS) = (w) ∪ First(S) = { w } ∪ { y } = { w,y }.

Question 135: [**GATE CS 2017**]

Match the following according to input to the compiler phases that process it?

P. Syntax tree	i. Code generator
Q. Character stream	ii. Syntax analyser
R. Intermediate representation	iii. Semantic analyser
S. Token stream	iv. Lexical analyser

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

Answer: C

Explanation: Q - iv because Character stream is given as input to lexical analyser

- P - iii Syntax tree is given as input to semantic analyser
- R - i Intermediate code given as input to code generator
- S - ii Token stream given as input to syntax analyser

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 136: [**GATE CS 2019**]

Consider the following given grammar:

S → Aa

A → BD

B → b | ε

D → d | ε

a	b	d	$
3	2	1	0

Let a,b,d and $ be Indexes as follows

Compute the FOLLOW set of the non-terminal B and write the index values for the symbols in the FOLLOW set in the descending order. (For example, if the FOLLOW set is {a, b, d, $}, then the answer should be 3210). **Note:** This was Numerical Type question.

Answer: 31

Explanation: Follow (B) = {d, a}

Hence their index in descending order is **31**.

Question 137: **[GATE CS 2019]**

Consider the following grammar and the semantic actions to support the inherited type declaration attributes. Let X_1, X_2, X_3, X_4, X_5 and X_6 be the placeholders for the non-terminals D, T, L or L_1 in the following table:

Production rule	Semantic action
D → TL	X1. type = X2. type
T → int	T. type = int
T → float	T.type = float
L → L1, id	X3.type = X4.type addType(id.entry, X5type)
L → id	addType(id.entry, X6type)

Which one of the following are the appropriate choices for X_1, X_2, X_3 and X_4?

A) $X_1 = L, X_2 = T, X_3 = L_1, X_4 = L$

B) $X_1 = L, X_2 = L, X_3 = L_1, X_4 = T$

C) $X_1 = T, X_2 = L, X_3 = L_1, X_4 = T$

D) $X_1 = T, X_2 = L, X_3 = T, X_4 = L_1$

Answer: A

Explanation: According to Inherited attribute definition,
an attribute is inherited if the attribute value of a parse-tree node is determined from attribute values of its parent and siblings. So, semantic rules should be as following below:
D → TL {L.idtype = T.stype}
T → int {T.stype = int}
T → float {T.stype = float}
L → L1, id {L1.itype = L.itype}
addtype(id.entry, L.itype)
L → id addtype(id.entry, L.itype)

Option (A) is correct.

Question 138: [**GATE CS 2021]**

Consider the following statements.

- **S1:** The sequence of procedure calls corresponds to a preorder traversal of the activation tree.
- **S2:** The sequence of procedure returns corresponds to a postorder traversal of the activation tree.

Which one of the following options is correct?

A)S1 is true and S2 is false

B)S1 is false and S2 is true

C)S1 is true and S2 is true

D)S1 is false and S2 is false

Answer: C

Question 139: [**GATE CS 2021**]

A shift reduce parser carries out the actions specified within braces immediately after reducing with the corresponding rule of grammar S—-> xxW (PRINT “1”) S—-> y { print ” 2 ” } S—-> Sz { print ” 3 ”) What is the translation of xxxxyzz using the syntax directed translation scheme described by the above rules ?

a)23131

b)11233

c)11231

d)33211

Answer: A

Explanation: Derive the string w = xxxxyzz from the given grammar and attach the Symantec actions as follows

S→y 2

W→Sz 23 231

S→xxW 23 13

W →Sz 23131

S → x x W

Question 140: [**GATE CS 2021]**

Consider the following statements.

- **S1:** Every SLR(1) grammar is unambiguous but there are certain unambiguous grammars that are not SLR(1).
- **S2:** For any context-free grammar, there is a parser that takes at most $O(n^3)$ time to parse a string of length n.

Which one of the following options is correct?

A)S1 is true and S2 is false

B)S1 is false and S2 is true

C)S1 is true and S2 is true

D)S1 is false and S2 is false

Answer: C

Question 141: **[GATE CS 2021]**

Consider the following context-free grammar where the set of terminals is {a,b,c,d,f}.

$$S \rightarrow daT \mid Rf$$
$$T \rightarrow aS \mid baT \mid \epsilon$$
$$R \rightarrow caTR \mid \epsilon$$

The following is a partially-filled LL(1) parsing table.

	a	b	c	d	f	$\$$
S			(1)	$S \rightarrow daT$	(2)	
T	$T \rightarrow aS$	$T \rightarrow baT$	(3)		$T \rightarrow \epsilon$	(4)
R			$R \rightarrow caTR$		$R \rightarrow \epsilon$	

Which one of the following choices represents the correct combination for the numbered cells in the parsing table ("blank" denotes that the corresponding cell is empty)?

A. (1) $S \rightarrow Rf$ (2) $S \rightarrow Rf$ (3) $T \rightarrow \epsilon$ (4) $T \rightarrow \epsilon$

B. (1) blank (2) $S \rightarrow Rf$ (3) $T \rightarrow \epsilon$ (4) $T \rightarrow \epsilon$

C. (1) $S \rightarrow Rf$ (2) blank (3) blank (4) $T \rightarrow \epsilon$

D. (1) blank (2) $S \rightarrow Rf$ (3) blank (4) blank

Answer: A

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

[Conti...]

Explanation:

Let's Compute First and Follow of non terminals.

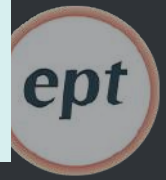
	First	Follow
S	d,c,f	c,f,$
T	a,b,∈	c,f,$
R	c,∈	f

Now using first and follow we will construct LL(1) parsing table.

1. S ->Rf production entry will come under first of S terminals (c,f) in parsing table.

2. T->∈ production entry will come under follow of T terminals (c,f,$) in parsing table.

	a	b	c	d	f	$
S			S ->Rf	S->daT	S ->Rf	
T	T->aS	T->baT	T->∈		T->∈	T->∈
R			R->caTR		R->∈	



Question 142: [**GATE CS 2022]**

Which one of the following statements is TRUE?

A)The LALR(1) parser for a grammar G cannot have a reduce-reduce conflict if the LR(1) parser for G does not have reduce-reduce conflict.

B)The symbol table is accessed only during the lexical analysis phase.

C)Data flow analysis is necessary for run-time memory management.

D)LR(1) parsing is sufficient for deterministic context-free languages.

Answer: D

Explanation: Option A: **False**. Not necessary. Merging or combining the states with the same state and different lookahead may also lead to the R-R conflict in LALR(1) parser.

Option B: **False.** Symbols table can be used with all phases of compilation and will be used at different stages. Not necessarily with lexical analysis phase only

Option C: **False**. It is not necessary but we may apply it for the optimizations and finding the minimum number of registers and analyzing the code etc.

Option D: **True**. The LR parser can recognize any deterministic context-free language in linear-bounded time.

Question 143: **[GATE CS 2022]**

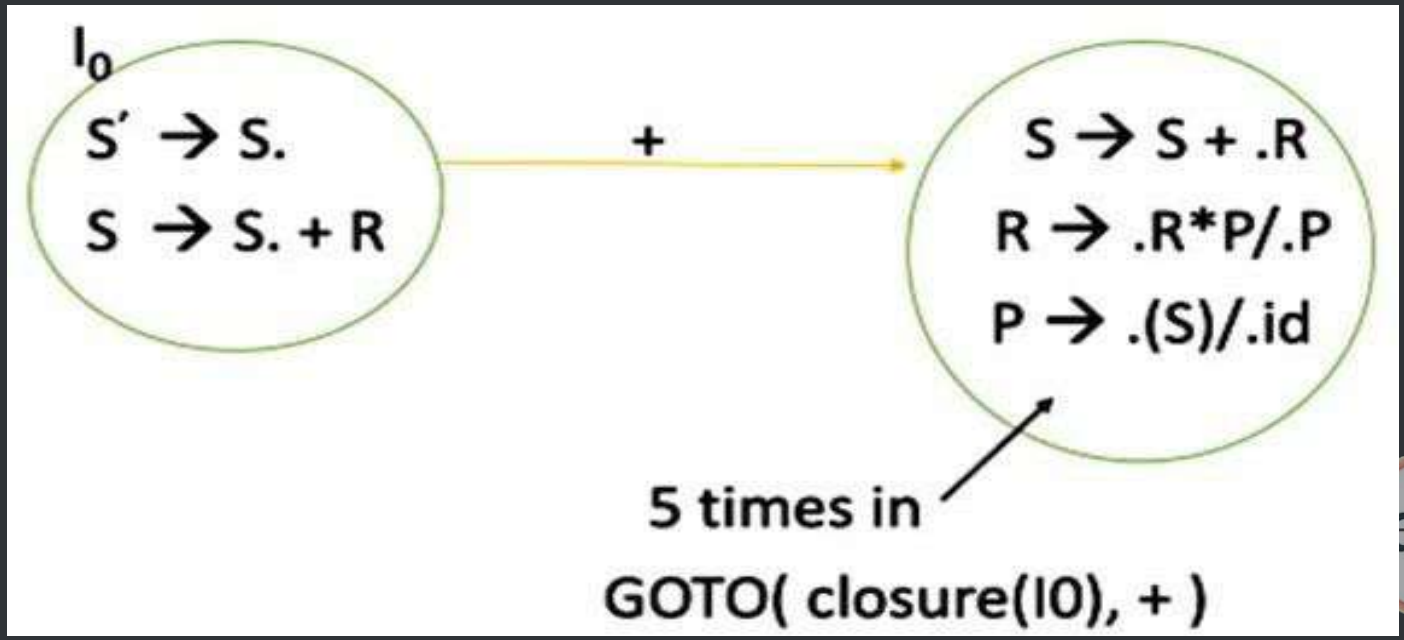
Consider the augmented grammar with { + , *, (,), id } as the set of terminals

```
S'->  S

S  ->S + R | R

R->  R* P| P

P-> (S) | id
```

Goto(Io,+) Contains how many items

Answer: 5

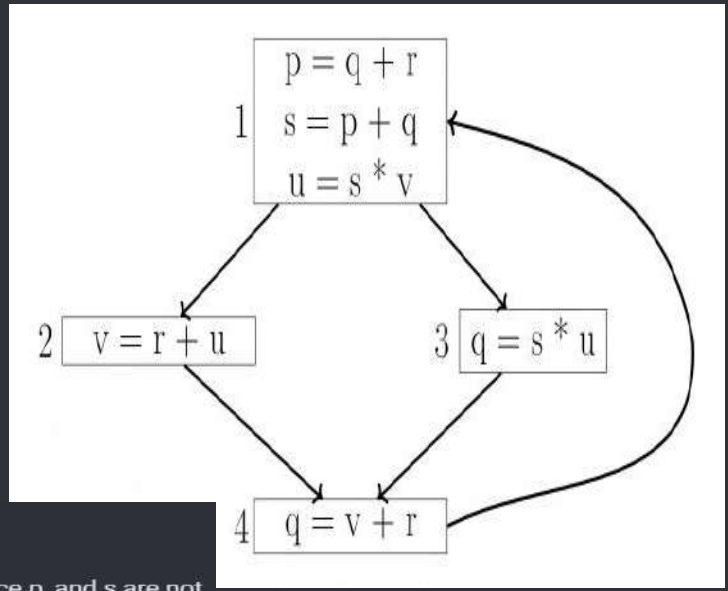
Explanation:



Question 144: **[GATE CS 2015]**

A variable x is said to be live at a statement S1 in a program if the following three conditions hold simultaneously:

i. There exists a statement S2 that uses x
ii. There is a path from S1 to S2 in the flow graph corresponding to the program
iii. The path has no intervening assignment to x including at S1 and S2



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

Answer:

r, u.

p, and s are assigned to in 1 and there is no intermediate use of them before that. Hence p, and s are not live in both 2 and 3.
q is assigned to in 4 and hence is not live in both 2 and 3.

v is live at 3 but not at 2.

u is live at 3 and also at 2 if we consider a path of length 0 from $2 - 2$.

So, r, u is the answer.

Question 145: [**GATE CS 2022**]

Consider the following grammar along with translation rules=========>

Here # and % are operators and id is a token that represents an integer and id. val represents the corresponding integer value. The set of non-terminals is {S, T, R, P} and a subscripted non-terminal indicates an instance of the non-terminal. Using this translation scheme, the computed value of S.val for root of the parse tree for the expression 20#10%5#8%2%2 is _____.

$S \rightarrow S_1 \# T \quad \{S_{.val} = S_{1.val} * T_{.val}\}$

$S \rightarrow T \quad \{S_{.val} = T_{.val}\}$

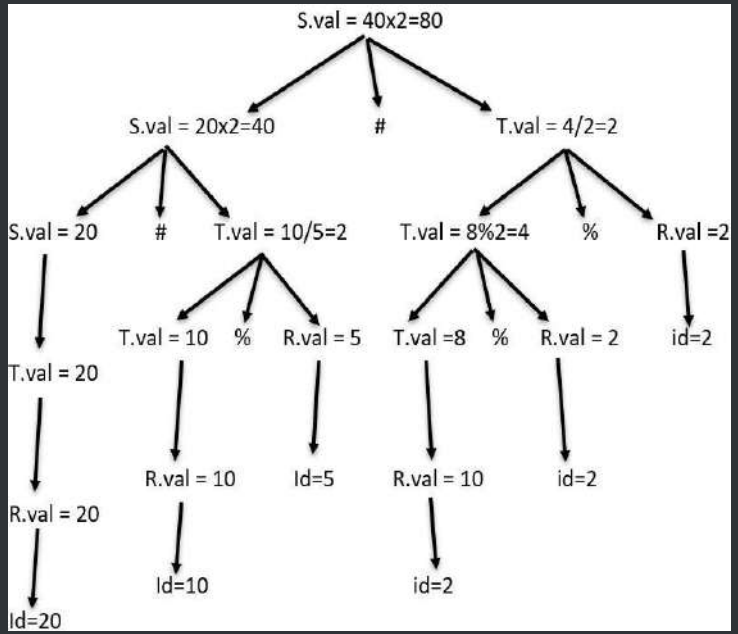
$T \rightarrow T_1 \% R \quad \{T_{.val} = T_{1.val} \div R_{.val}\}$

$T \rightarrow R \quad \{T_{.val} = R_{.val}\}$

$R \rightarrow id \quad \{R_{.val} = id_{.val}\}$

Answer: 80

Explanatiyon:



Question 146: [**GATE CS 2023**]

Consider the syntax directed translation given by the following grammar and semantic rules. Here N, I, F and B are non-terminals. N is the starting non-terminal, and $\#$, $\mathbf{0}$ and $\mathbf{1}$ are lexical tokens corresponding to input letters "#", "0" and "1", respectively. $X.val$ denotes the synthesized attribute (a numeric value) associated with a non-terminal X. I_1 and F_1 denote occurrences of I and F on the right hand side of a production, respectively. For the tokens $\mathbf{0}$ and $\mathbf{1}$, $\mathbf{0}.val = 0$ and $\mathbf{1}.val = 1$.

$N \rightarrow I\#F$	$N.val = I.val + F.val$
$I \rightarrow I_1B$	$I.val = (2I_1.val) + B.val$
$I \rightarrow B$	$I.val =$ B.val
$F \rightarrow BF_1$	$F.val = \frac{1}{2}(B.val + F_1.val)$
$F \rightarrow B$	F.val $= \frac{1}{2}B.val$
$B \rightarrow \mathbf{0}$	$B.val = \mathbf{0}.val$
$B \rightarrow \mathbf{1}$	$B.val = \mathbf{1}.val$

The value computed by the translation scheme for the input string

$$10\#011$$

is _____. (Rounded off to three decimal places)

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

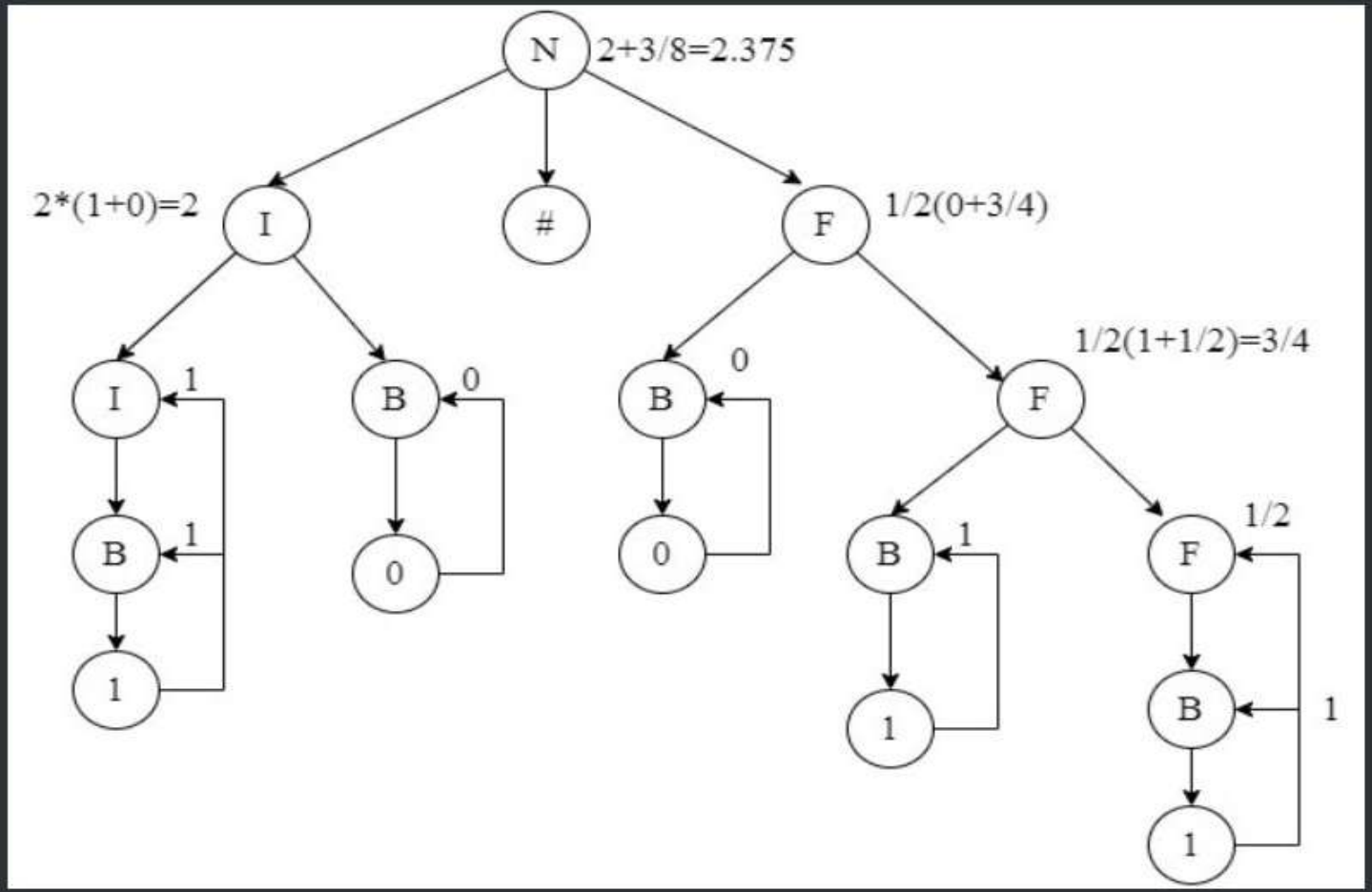
For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Answer: 2.375

Explanation:



The correct answer is 2.375

Question 147: [**GATE CS 2023**]

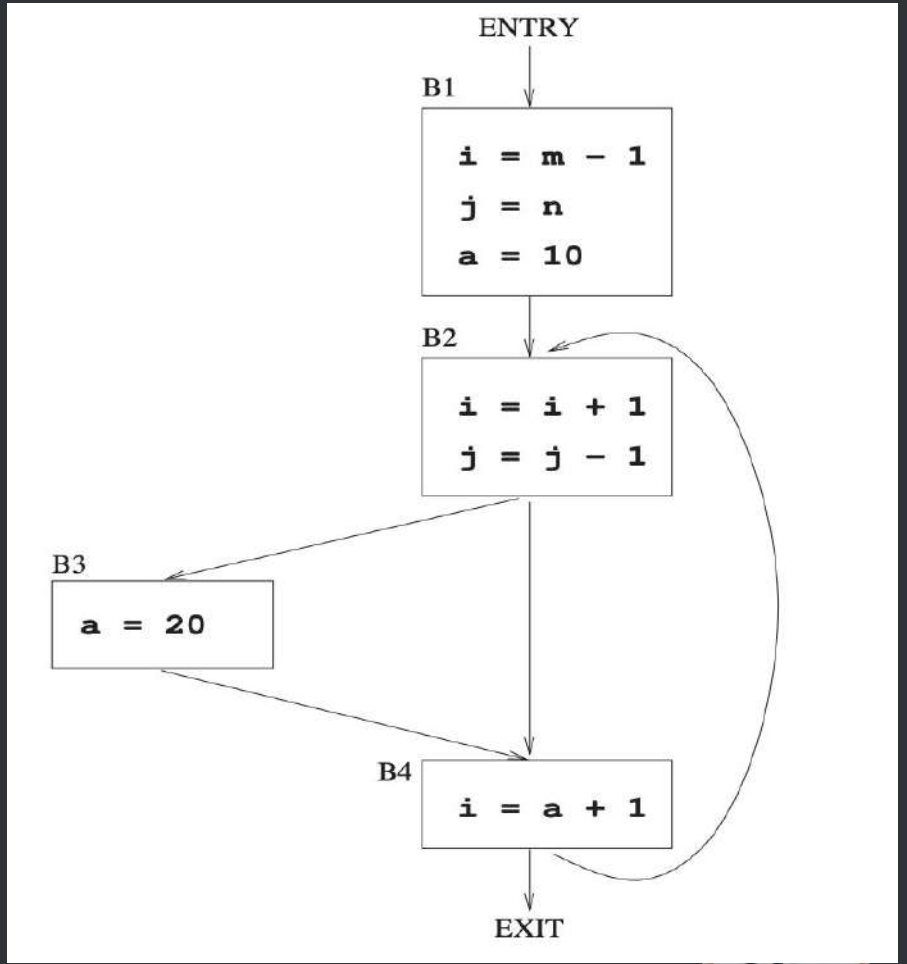
Consider the control flow graph shown.

Which one of the following choices correctly lists the

set of *live* variables at the exit point of each basic block?

A. B1: { }, B2: {a}, B3: {a}, B4: {a}
B. B1: {i, j}, B2: {a}, B3: {a}, B4: {i}
C. B1: {a, i, j}, B2: {a, i, j}, B3: {a, i}, B4: {a}
D. B1: {a, i, j}, B2: {a, j}, B3: {a, j}, B4: {a, i, j}

Answer: D



(Practice Questions)

Question 148: [**GATE CS Mock**]

In some phases of compiler input

```
temp1 := inttoreal(60)
temp2 := id3*temp1
temp2 := id2 + temp2
Id1 := temp3
```

Output:

```
temp1 := id3 * 60.0
Id1 := id2+temp1
```

Where temp1, temp2, temp3 are temporary storage, id1, id2, Id3 are identifier into real is converting int 60 to real number. The above phase is

A)Code optimizer
B)Code generator
C)intermediate code generator
D)None of the above

Answer: A

Explanation: Input:

temp1: = inttoreal(60)

temp2: = id3*temp1

temp3 : = id2 + temp2

id1 : = temp3

Output:

temp1 : = id3*60.0

id1 : id2 + temp1

The above phase is code optimization, because it is optimizing the intermediate code using only storages as temp 1 and id1.

Question 149: [**GATE CS Mock**]

Match List-1 with List-2 and select the correct answer using the codes given below the lists:

List1	**List2**
A. Compilation	1. RunTime specification
B. Interpretation	2. Exception
C. Macro definition	3. Object Specification
D. Loading	4. Abbreviation

Codes:

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

Answer: D

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 150: [**GATE CS Mock**]

Match List-1 with List-2 and select the correct answer using the codes given below the lists:

List1	**List2**
A. Preprocessor	1. Resolving external reference
B. Assembler	2. Loading the program
C. Loader	3. Producing relocatable machine code
D. Linker	4. Allow users to define shorthand for longer construct

Codes:

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

Answer: A

Question 151: [**GATE CS Mock**]

Match List-1 with List-2 and select the correct answer using the codes given below the lists:

List1

A. Lexical Analyzer
B. Syntax Analyzer
C. Type Checking
D. Intermediate code generation

List2

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 requirement 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

Answer: B

Question 152: [**GATE CS Mock**]

We can implement the symbol table data structure for a typical compiler by using which of the following ?

1. Linear list
2. Array
3. Hash Table

A) 2 only B) 1 and 2 only C) 1 and 3 Only D) 3 only

Answer: C

We can also use array but this approach is not efficient and not used in practice.

Symbol table using List , hash and binary search tree is preferred more

Question 153: [**GATE CS Mock**]

The cost of developing a compiler is proportional to

a)complexity of the source language

b.)complexity of the architecture of the target machine

c.)flexibility of the available instruction set

d.)all of these

Answer: D

Question 154 : [**GATE CS Mock**]

Backend of the compiler includes those phases that depends on

A) Target Machine
B) Source Language
C) Both (A) and (B)
D) None of the above

Answer: A

Question 155: [**GATE CS Mock**]

Frontend of the compiler does not include the phases

A) Code generation
B) Intermediate code generation
C) Code optimization
D) Both A and C

Answer: D

Question 156: [**GATE CS Mock**]

Which of the following pahses of compiler is an optional phase

A) Lexical Analysis

B) Syntax Analysis

C) COde Optimizaiton

D) Code Generaiton

Answer: C

Question 157: [**GATE CS Mock**]

Which of the following is invalid preprocessor command in c language?

A) # define B)# Include

C) # ifdef D) None of these

Answer: D

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 158: [**GATE CS Mock**]

Which of the following is not a function of c compiler?

A)Identifying Tokens

B)Identifying Syntax errors

C)Linking

D)None of these

Answer: C

Explanation: Linking is done by a linker after compilation process. Compiler can identify tokens, generates compilation errors (lexical, syntax and semantic).

Question 159: **[GATE CS Mock]**

Match the following Groups:

List 1:

A. Allocation
B. Relocation
C. Loading
D. Linking

1. Resolve the symbol reference
2. Alters the address of the instructions and data
3. Makes the program ready to execute by keeping the machine code in the main memory
4. Assigning the required memory space for the program

Codes:

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

Answer: D

Question 160: [**GATE CS Mock**]

The number of tokens in the following C statementws

Print(“i=%d,&i=%x”,i,&i);

A)3 **B)26** **C)10** **D)21**

Answer: C

Explanation: Print (“i=%d,&i=%x” , i , & i) ;

1 2 3 4 5 6 7 8 9 10

Question 161: [**GATE CS Mock**]

Identify the rules that must be followed in the case of identifiers:

1. The identifiers must begin with a numerical digit.

2. The first character used in an identifier should be either an underscore or an alphabet.

3. The lowercase and uppercase letters are not distinct in an identifier.

4. An identifier does not specify blank spaces or commas.

5. The maximum length of an identifier is 31 characters.

A. 1, 3, and 5

B. 2, 4, and 5

C. 1, 2, and 4

D. 2, 3, and 5

Answer – B. 2, 4, and 5

Question 162: [**GATE CS Mock**]

Which of these is not a constant used in the C language?

A. Octal Constants

B. String Constants

C. Integral Constants

D. Floating Constants

Answer – C. Integral Constants

Question 163: [**GATE CS Mock**]

Consider the following issues:

1. Simplify the Phases
2. Compiler efficiency is improved
3. Compiler works faster
4. Compiler portability is enhanced

Which is/are true in the context of Lexical Analyser?

Which is/are true in context of lexical analysis?

a)1,2 and 3
b)1,3 and 4
c)1,2 and 4
d)All of these
Answer: C

Explanation: Lexical analysis consists of the phases that
(i) Simplify the subsequent phases by the help of tokens instead of arbitrary elements.
(ii) Since interrelated elements are converted to stream of tokens hence compiler efficiency is improved as no arbitrary storage or value is to be checked independently.
(iii) Computer portability is enhanced.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 164: [**GATE CS Mock**]

Which is considered as the sequence of characters in a token?

a. Mexeme

b. Lexeme

c. Texeme

d. Pattern

Answer: B

Explanation: Lexemes are the string of alphanumeric characters in a single token. In the source program, lexemes are characters which are identified by the pattern for a token.

Question 165: [**GATE CS Mock**]

Type checking is normally done during----------

A. Lexical analysis
B. Syntax Analysis
C. Syntax directed translation
D. Code optimization

Answer: C

Explanation: Type checking is done at Semantic analysis nothing bus Syntax directed translation phase.

Question 166: [GATE CS Mock]

Consider the syntax-directed transition scheme given below with non-terminals {A, B, C} and terminals (a, b, c)

S-AC (print) | aB(print ***}

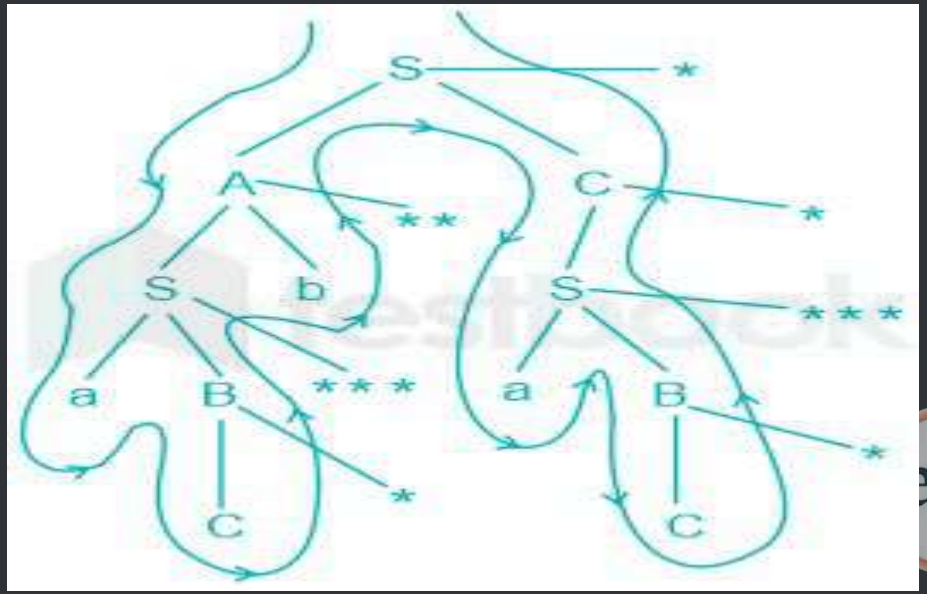
A → Sb {print **}

B → c{print *}

C-S {print *}

Using the above SDT, calculate the total number of * that are going to be printed in the output for the input **acbac**.

Answer: Total 12 * printed



Question 167: **[GATE CS Mock]**

Consider the following two Statements:

S1: The set of Strings 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 following statement is/are true?

1. Both S1 and S2 are true
2. S1 is true S2 is false
3. S2 is true S1 is false
4. Both S1 and S2 are false

Answer: 3

Explanation: The first statement "The set of string described by a rule is called pattern associated with the token" is false. The pattern is not associated with the token, but rather it is used to match a sequence of characters in the source program to identify a token.

The second statement "A lexeme is a sequence of character in the source program that is matched by Pattern for a token" is true. A lexeme is a sequence of characters in the source program that matches a pattern for a token, and it is the basic unit of analysis in lexical analysis.

Therefore, the correct answer is "The first statement is false, and the second statement is true."

Question 168: [**GATE CS Mock**]

if (z==a) function1(10);

The number of tokens in the above statement are

A)9 B)11 C)10 D)12

Answer: B

Explanation: if (z == a) function1 (10) ;

1 2 3 4 5 6 7 8 9 10 11

Question 169: **[GATE CS Mock]**

D→bool ID { record that ID.lexeme is of type bool}

E→ E_1+ E2 (check that E_1.type = E2.type = int; set E.type := int}

E→!E_1 {check that E_1.type = bool; set E.type = bool}

E →> ID {set E.type = int}

With respect to the above grammar; which one of the following choices is correct

A) The actions will lead to infinite loop.

B) The actions can be used to correctly type-check any syntactically correct program.

C) The actions can be used to type-check syntactically correct boolean variable declarations and boolean expressions.

D) The actions can be used to type-check syntactically correct integer variable declarations and integer expressions.

Answer: D

Explanation: Option 1:The actions will lead to an infinite loop. This statement is not correct.

Option 2: The actions can be used to correctly type-check any syntactically correct program.

This statement is not correct. Because These SDT actions can not type-check boolean expressions or float expressions.

for example:

```
float a=3.46;
bool b = 0,c;
c = a+b;
```

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Option 3: The actions can be used to type-check syntactically correct boolean variable declarations and boolean expressions.

This statement is not correct. Because the SDT actions can not be used to correctly type check boolean expressions.

for example:

bool a = 0;

bool b = 1;

bool c = = a+b;

Option 4: The actions can be used to type-check syntactically correct integer variable declarations and integer expressions.

This statement is Correct. Because These SDT actions can be used to correctly type chec both integer variables and integer expressions as well.

Question 170: [**GATE CS Mock**]

Find the C statement Which has the Syntax error.

A) fi(z);
B) for(a,b,c);
C) Whil (a,b);
D) None of these

Answer: B

Explanation: options A and C are considered as function definitions but Option B ‘for’ is a keywords which gives syntax error

Question 171: [**GATE CS Mock**]

Consider the productions A → PQ and A → XY. Each of the five non-terminals A, P, Q, X and Y has two attributes: s is a synthesized attribute, and i is an inherited attribute. Consider the following rules.

Rule 1: P.i = A.i + 2, Q.i = P.i + A.i, and A.s = P.s + Q.s

Rule 2: X.i = A.i + Y.s and Y.i = X.s + A.i

Which one of the following is TRUE?

A) Both Rule 1 and Rule 2 are L-attributed.

B) Only Rule 1 is L-attributed.

C) Only Rule 2 is L-attributed.

D) Neither Rule 1 nor Rule 2 is L-attributed.

Answer: B

Explanation: Rule 1: P.i=A.i + 2, Q.i = P.i + A.i, and A. s = P.s + Q.s

P.i = A.i +2 ...{This is an inherited attribute, as child P is getting value from parent A} Q.i = P.i + A.i___...{This is also inherited attribute and Q is getting value from parent P and left sibling A}

A.s=P.s+Q.s ...{This is synthesized attribute as Parent A is taking values from children P and Q}

Rule 2 is not L- attributed

Rule 2: X.i= A.i + Y.s and Yi = X.s + A.i

Explanation: X.i = A.i + Y.s ...{Child X is taking attribute value from parent A and right sibling_Y} Therefore, this violates the conditions of L-attributed definitions.

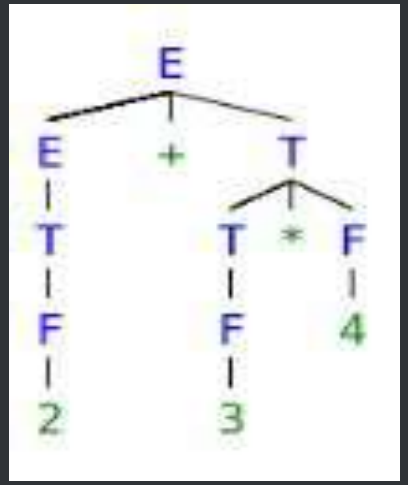
Question 172: **[GATE CS Mock]**

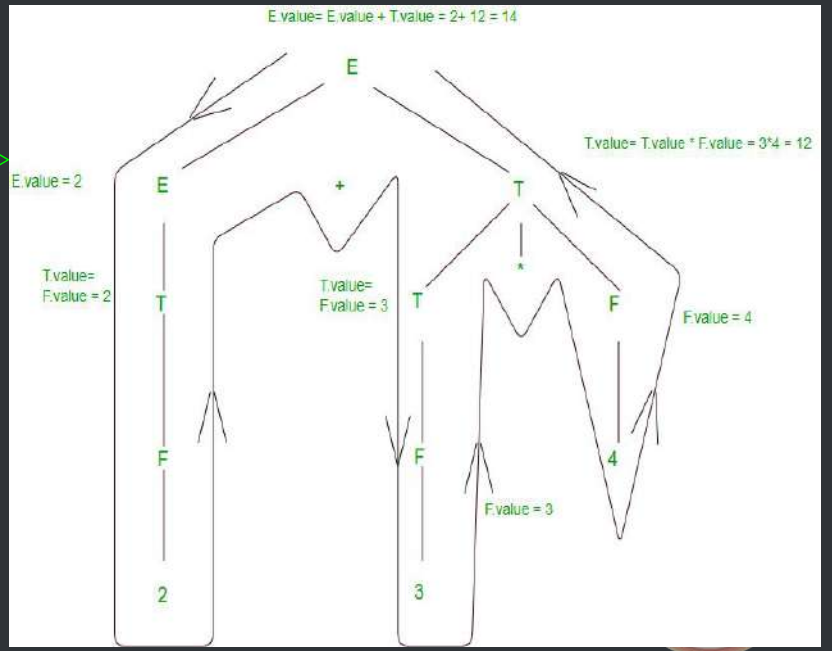
Consider the following SDT for the grammar

```
E -> E+T      { E.val = E.val + T.val }    PR#1

E -> T        { E.val = T.val }            PR#2

T -> T*F      { T.val = T.val * F.val }    PR#3

T -> F        { T.val = F.val }            PR#4

F -> INTLIT   { F.val = INTLIT.lexval }    PR#5
```

Find the output when this SDT applied on string S=2+3*4

Answer: Parse tree Evaluating the attributes=====>





Question 173: [**GATE CS Mock]**

Find the Leaders and draw the basic block flowchart for for converting a 10 x 10 matrix to an identity matrix.

```
for r from 1 to 10 do
        for c from 1 to 10 do
            a [ r, c ] = 0.0;
for r from 1 to 10 do
        a [ r, c ] = 1.0;
```

Answer:

Step1: create the 3-address code for the given source code

1) r = 1
2) c = 1
3) t1 = 10 * r
4) t2 = t1 + c
5) t3 = 8 * t2
6) t4 = t3 - 88
7) a[t4] = 0.0
8) c = c + 1
9) if c <= 10 goto (3)
10) r = r + 1
11) if r <= 10 goto (2)
12) r = 1
13) t5 = c - 1
14) t6 = 88 * t5
15) a[t6] = 1.0
16) r = r + 1
17) if r <= 10 goto (13)

[Conti.....]

There are six basic blocks for the above-given code, which are:

- B1 for statement 1
- B2 for statement 2
- B3 for statements 3-9
- B4 for statements 10-11
- B5 for statement 12
- B6 for statements 13-17.

Explanation:

According to the definition of leaders given in the above algorithm,

- **Instruction 1** is a leader as the first instruction of a three-address code is always a leader.
- **Instruction 2** is a leader because it is followed by a goto statement at Instruction 11.
- **Instruction 3 and Instruction 13** are also leaders because they are followed by a goto statement at Instruction 9 and 17, respectively.
- **Instruction 10 and Instruction 12** are also leaders because they are followed by a conditional goto statement at Instruction 9 and 17, respectively.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 173: [**GATE CS Mock**]

What are the problems with top-down parser?

Answer:

- Backtracking
- Left recursion
- Left factoring
- Ambiguity

Question 174: [**GATE CS Mock]**

Write the algorithm for FIRST and FOLLOW?

Answer:

FIRST SET:

1. If X is terminal, then FIRST(X) IS {X}.
2. If X → ε is a production, then add ε to FIRST(X).
3. If X is non terminal and X → Y1,Y2..Yk is a production, then place a in FIRST(X) if for some i , a is in FIRST(Yi) , and ε is in all of FIRST(Y1),...FIRST(Yi-1);

FOLLOW SET: Place $ in FOLLOW(S),where S is the start symbol and $ is the input right endmarker.

1. If there is a production A → αBβ, then everything in FIRST(β) except for ε is placed in FOLLOW(B).
2. If there is a production A → αB, or a production A→ αBβ where FIRST(β) contains ε , then everything in FOLLOW(A) is in FOLLOW(B).
3. List the advantages and disadvantages of operator precedence parsing.

Question 175: **[GATE CS Mock]**

Consider the following grammar:

S-> aAb | Sc

A-> d | Sd | S

The above grammer is:

a) SLR(1)

b) LL(1)

c) LR(0)

d) none of the above

Answer: A

Explanation: ==>as the given grammar is left recursive (S->Sc),so it is not LL(1) grammar.

==>as the given grammar has SR conflict,it is not LR(0) grammar(A->S.d and A->S.)

==>it is SLR(1) as there is no any SR conflict

so answer is A) SLR(1)

Question 176: [**GATE CS Mock**]

Consider the following grammar find whether the given grammar is LL(1) or not?

$$S \rightarrow A \mid B$$
$$A \rightarrow a \mid \epsilon$$
$$B \rightarrow b \mid \epsilon$$

Answer: No It is not a LL(1) Grammar

Explanation: Now if I want to generate empty string (ε) from above grammar, I have more than one option to generate it. Obviously grammar is ambiguous. No parser will work.

Question 177: [**GATE CS Mock**]

Consider the following C program

```
#include<stdio.h>
Main()
{
int x=0, y=12;
Char * a;
a=&x;
x=OxAB;
print("%d %d", x, *a);
}
```

Which of the following type of error is identified during the compilation of the program?

A)Lexical

B) Syntrax

C)Semantic

D)None of these

Answer: C Type incompatible in Line a=&x;

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

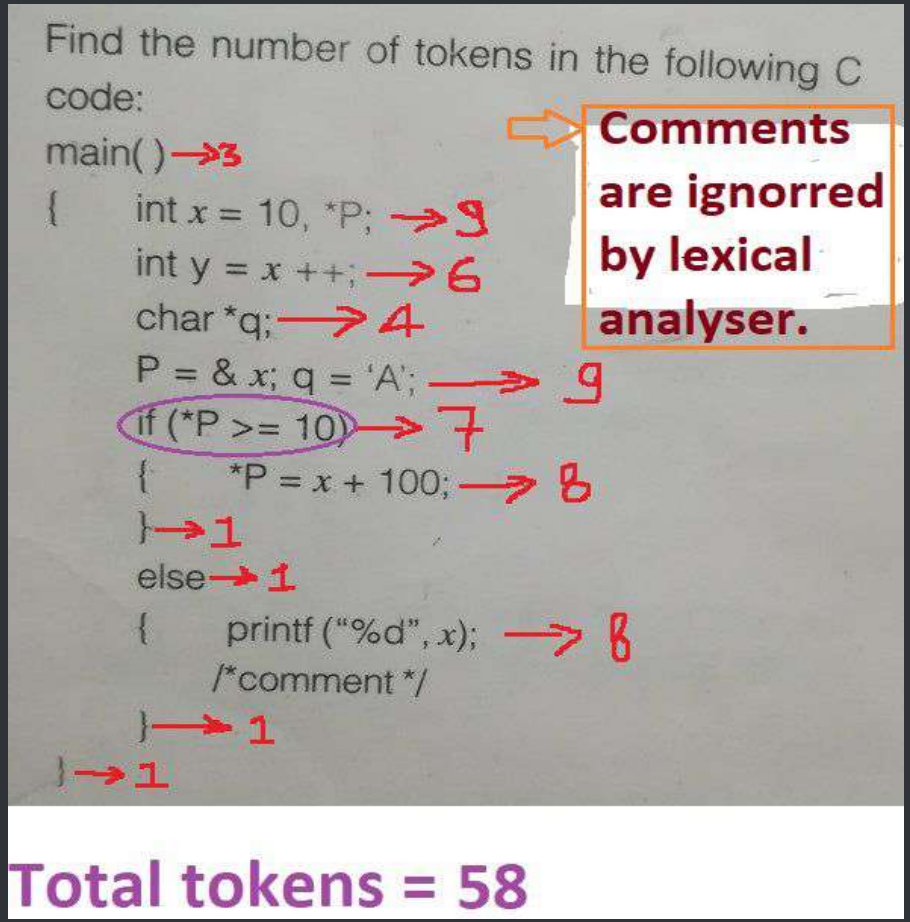
Note: If any doubt in any question post with subject & question number in EPT App.

Question 178: **[GATE CS Mock](NAT)**

Find The number of tokens in the Following C program----------

```
main()
{
  int x = 10 , *P ;
  int y = x ++ ;
  char * q ;
  P = & x ; q = 'A' ;
  if(*P>=10)
  {
    *P = x + 100;
  }
  else
 {
   printf("%d" , x);
    /*comment*/
 }
}
```

Answer: 58



Question 179: **[GATE CS Mock]**

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

Answer: D

Explanation:

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

Question 180: **[GATE CS Mock]**

How many Tokens are generated by the lexical analyzer if the following c program has no lexical errors—-------

```
main()
{
    int x, y;
    fl/*gate oat z;
    x =/*exam*/10;
    y = 20;
}
```

Answer: 17

Explanation: Total number of tokens = **17**.

These are the tokens:

1. main
2. (
3.)
4. {
5. int
6. x
7. ,
8. y
9. ;
10. fl
11. 10
12. ;
13. y
14. =
15. 20
16. ;
17. }

The comment starts with a **/*** and ends to the nearest ***/**. Therefore

```
/*gate oat z;
x =/*exam*/
```

will be considered a **single token**.

And "**fl**" and "**10**" will be separated as two different tokens.

Question 181: [**GATE CS Mock**]

Consider the following Program segment:

```
int foo(unsigned int n)
{
int c, x=0;
While(n!=0)
{
if(n & 01) X++;
n>>=1;
}
Return cl
```

} The number of tokens generated by the lexical analyzer in the above program are—----------

Answer: 40

Explanation: a) Lexical analyser is a DFA .So it has capability to identify compound operators ; so they will be considered as 1 lexeme only under the token class "operator"..

b) Numbers are treated as constants irrespective of the base used whether it is octal or decimal..
c) compiler follows greedy approach and will try to make longest pattern available. like x++ will be 2 token instead of x , + ,+ (3)
So here "01" , "!=" and ">>=" will be treated as 1 token only..

Question 182: [**GATE CS Mock**]

Find the number of tokens in the following C Code using the lexical analyzer of the compiler

```
main()
{
/* int a=10 */
int * u, *v, s;
u = &s;
V = &s;
printf("%d %d", s, *u);
//code ended
}
```

Answer: 34

Question 183: [**GATE CS Mock**]

Find the number of Lexical errors in the following C Code using the lexical analyzer of the compiler

```
main()
{
int x=10;
It ( x < 20 ;
else
y=20;
}
```

Answer: No lexical error will be produced

Question 184: [**GATE CS Mock**]

An LR parser can detect a —----------as soon as it possible when a left to right scan of the input is done

A) Semantic error
B) Syntactic error
C) Logical error
D) Lexical error

Answer: B

Explanation: LR-parser and detecting syntactic errors—---**LR-parser:** An LR-parser is a type of bottom-up parser that reads input text from left to right and constructs a rightmost derivation of the input. It uses a stack to keep track of previously seen symbols and a parsing table to determine the next action based on the current state and the next input symbol.

Syntactic error: A syntactic error occurs when the input text does not conform to the grammar of the language being parsed. For example, a missing semicolon or a mismatched parenthesis can cause a syntactic error.

Detection of syntactic error: An LR-parser can detect a syntactic error as soon as it is possible to do so with a right-to-left scan of the input. This is because LR-parsers are bottom-up parsers that attempt to construct a rightmost derivation of the input. If the parser encounters an input symbol that cannot be reduced to a valid production in the grammar, it can backtrack and try a different production. If all possible productions fail, the parser reports a syntax error.

Left-to-right scan: While it is possible to detect some syntactic errors with a left-to-right scan of the input, it is not always sufficient. This is because LR-parsers attempt to construct a rightmost derivation of the input, which means they need to look ahead to determine the correct action to take. For example, consider the input "a + b * c". A left-to-right scan would detect the missing semicolon at the end of the input, but it would not detect the fact that the expression is ambiguous without additional parsing information. An LR-parser, on the other hand, can use its parsing table to determine the correct parse tree for the input.

Conclusion: In conclusion, an LR-parser can detect syntactic errors as soon as it is possible to do so with a right-to-left scan of the input. This is because LR-parsers attempt to construct a rightmost derivation of the input and need to look ahead to determine the correct action to take. While it is possible to detect some syntactic errors with a left-to-right scan, it is not always sufficient for LR-parsers.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 185: [GATE CS Mock]

Which of the following statement is correct about parsing?

1. Top-down parsing expands the start symbol to the required string needed whereas in bottom-up parsing reduces the syntax to the start symbol.

2. Top-down parsing is implemented using the set of recursive procedures and implementation of bottom-up parsing is done using the stacks and input buffer.

3. Backtracking is required for both bottom-up and top-down parsing implementation.

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

Answer: A

Explanation: Top down parsing expands the start symbol to the required string needed where as in bottom up parse string reduces to start symbol by handle purring.

Top down parser also known as recursive descent parser i.e. uses recursive procedure, where as bottom up parser user stack and input buffer.Backbacking is required in top-down parser but not in bottom up

Question 186: [**GATE CS Mock**]

The set of prefixes of the right sentential forms that can appear on the stack of a shift reduce parser are called....

A) Right prefix

B) SR Prefixes

C) Viable prefixes

D) Non-viable prefixes

Answer: C

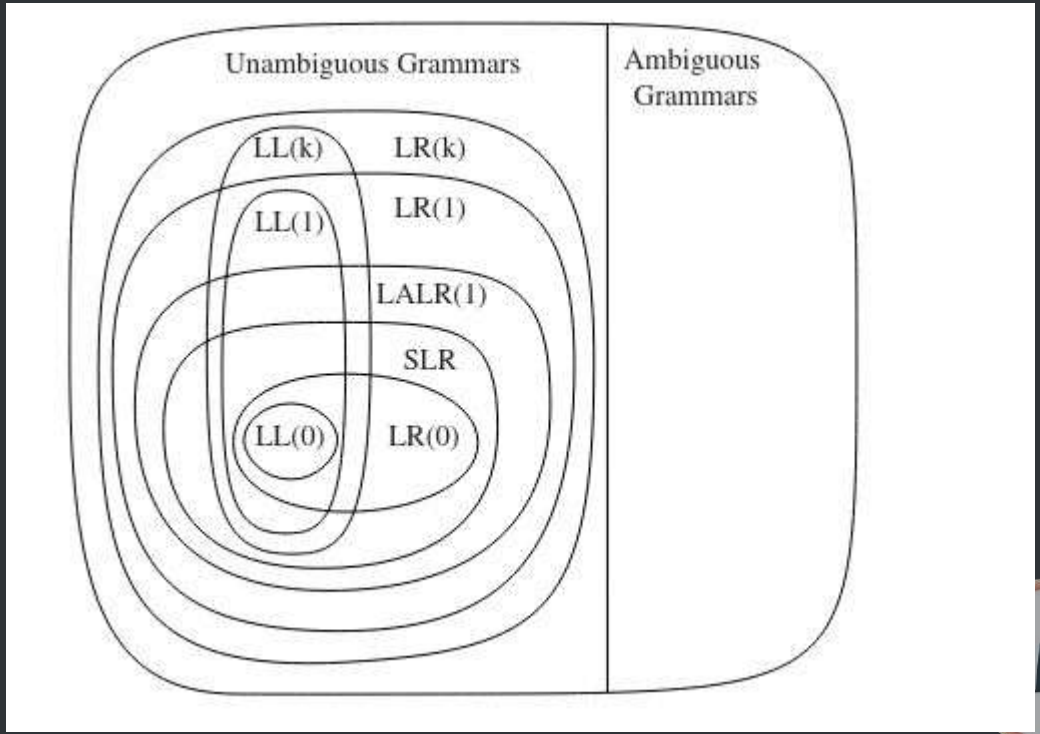
Question 187: [**GATE CS Mock**]

Consider the LR(0), LR(K), and SLR(1) grammars then which of the following about these grammers 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)

Answer: A

Explanation: Hierarchy of Grammers



Question 188: [**GATE CS Mock]**

Which of the following statements is false?

a) Left as well as right most derivations can be in Unambiguous grammar

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

c) LALR is more powerful than SLR

d) Ambiguous grammar can’t be LR (k)

Answer: A

Explanation: If a grammar has more than one leftmost (or rightmost) derivation the grammar is ambiguous. Sometimes in unambiguous grammar the rightmost derivation and leftmost derivations may differ.

Question 189: [**GATE CS Mock**]

Parsers are expected to parse the whole code.

a) True

b) False

Answer: True

Explanation: Parsers are expected to parse the whole code even if some errors exist in the program.

Question 190: [**GATE CS Mock]**

Assume that the SLR parser for a grammar G has n1 states and the LALR parser for G has n2 states.

a) n1 is necessarily less than n2

b) n1 is necessarily equal to n2

c) n1 is necessarily greater than n2

d) none of the mentioned

Answer: B

Explanation: SLR parser has less range of context free languages than LALR but still both n1 & n2 are same for SLR & LALR respectively

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 191: [**GATE CS Mock**]

A grammar for a programming language is a formal description of _____

a) Syntax

b) Semantics

c) Structure

d) Library

Answer: C

Explanation: The grammar clearly indicates which type of structure does a program has.

Question 192: [**GATE CS Mock]**

Select a Machine Independent phase of the compiler.

a) Syntax Analysis

b) Intermediate Code generation

c) Lexical Analysis

d) All of the mentioned

Answer: D

Explanation: All of them work independent of a machine.

Question 193: [**GATE CS Mock**]

Consider the grammar defined by the following production rules:

```
S --> T * P
T --> U | T * U
P --> Q + P | Q
Q --> Id
U --> 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

Answer: B

Explanation: It is associative we can see and tell.Second productions latter part shows left recursion and is left associative.

Question 194: [**GATE CS Mock]**

DAG representation of a basic block allows _____

a) Automatic detection of local common sub expressions

b) Detection of induction variables

c) Automatic detection of loop variant

d) None of the mentioned

Answer: A

Explanation: It detects local sub expression.

Question 195: [**GATE CS Mock**]

Inherited attribute is a natural choice in _____

a) Tracking declaration of a variable

b) Correct use of L and R values

c) All of the mentioned

d) None of the mentioned

Answer: A

Explanation: These attribute keep a check on variable declaration.

Question 196: [**GATE CS Mock**]

An intermediate code form is _____

a) Postfix notation

b) Syntax Trees

c) Three Address code

d) All of the mentioned

Answer: D

Explanation: Intermediate code generator receives input from its predecessor phase, semantic analyzer, in the form of an annotated syntax tree.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 197: [**GATE CS Mock]**

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) Inserting or deleting symbols from the input

d) All of the mentioned

Answer:D

Explanation: All these symbols are used to recover operator precedence parser from an error.

Question 198: [**GATE CS Mock**]

Input to code generator is _____

a) Source code

b) Intermediate code

c) Target code

d) All of the mentioned

Answer: B

Explanation: Intermediate code is the input to the code generator.

Question 199: [**GATE CS Mock]**

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 mentioned

Answer: C

Explanation: Synthesized attribute's value depend on children node only.

Question 200: [**GATE CS Mock**]

In a bottom up evaluation of a syntax direction definition, inherited attributes can _____

a) Always be evaluated

b) Be evaluated only if the definition is L –attributed

c) Evaluation only done if the definition has synthesized attributes

d) None of the mentioned

Answer: C

Explanation: Bottom-up parsing identifies and processes the text's lowest-level, before its mid-level structures, and the highest-level overall structure to last are left.

Question 201: [**GATE CS Mock**]

The graph that shows basic blocks and their successor relationship is called _____

a) DAG

b) Flow Chart

c) Control Graph

d) Hamilton graph

Answer: b

Explanation: Flow chart shows basic blocks.

Question 202: **[GATE CS Mock]**

_____ is a graph representation of a derivation.

a) The parse tree

b) Oct tree

c) Binary tree

d) None of the mentioned

Answer:A

Explanation: Parse tree is a representation of the derivation.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 203: [**GATE CS Mock]**

What is the similarity between LR, LALR and SLR?

a) Use same algorithm, but different parsing table

b) Same parsing table, but different algorithm

c) Their Parsing tables and algorithm are similar but uses top down approach

d) Both Parsing tables and algorithm are different

Answer: A

Explanation: The common grounds of these 3 parser is the algorithm but parsing table is different.

Question 204: [**GATE CS Mock**]

If a state does not know whether it will make a shift operation or reduction for a terminal is called _____

a) Shift/reduce conflict

b) Reduce /shift conflict

c) Shift conflict

d) Reduce conflict

Answer: A

Explanation: As the name suggests that the conflict is between shift and reduce hence it is called shift reduce conflict.

Question 205: [**GATE CS Mock**]

The construction of the canonical collection of the sets of LR (1) items are similar to the construction of the canonical collection of the sets of LR (0) items. Which is an exception?

a) Closure and goto operations work a little bit different

b) Closure and goto operations work similarly

c) Closure and additive operations work a little bit different

d) Closure and associatively operations work a little bit different

Answer: A

Explanation: Closure and goto do work differently in case of LR (0) and LR (1).

Question 206: [**GATE CS Mock]**

In operator precedence parsing whose precedence relations are defined _____

a) For all pair of non-terminals

b) For all pair of terminals

c) To delimit the handle

d) None of the mentioned

Answer: A

Explanation: There are two important properties for these operator precedence parsers is that it does not appear on the right side of any production and no production has two adjacent no terminals.

Question 207: [**GATE CS Mock]**

The method which merges the bodies of two loops is?

a) Loop rolling

b) Loop jamming

c) Constant folding

d) None of the mentioned

Answer: B

Explanation: In computer science, loop fusion (or loop jamming) is a compiler optimization and loop transformation which replaces multiple loops with a single one.

Question 208: [**GATE CS Mock]**

Dividing a project into segments and smaller units in order to simplify design and programming efforts is called?

a) Modular approach

b) Top down approach

c) Bottom up approach

d) Left right approach

Answer: A

Explanation: Modular design, or “modularity in design”, is a design approach that subdivides a system into smaller parts called modules or skids that can be independently created and then used in different systems.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 209: [**GATE CS Mock**]

The identification of common sub-expression and replacement of run-time computations by compile-time computations is _____

a) Local optimization

b) Loop optimization

c) Constant folding

d) Data flow analysis

Answer: C

Explanation: Constant folding is the process of recognizing and evaluating constant expressions at compile time rather than computing them at runtime. Terms in constant expressions are typically simple literals they may also be variables whose values are assigned at compile time.

Question 210: [**GATE CS Mock**]

Consider the following C code segment.

```
for if i # i } } }
```

Which one to the following false?
a) The code contains loop-in variant 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

Answer: D

Explanation: All the statements are true except option last since there is no dead code to get eliminated.

Question 211: [**GATE CS Mock]**

In the correct grammar above, what is the length of the derivation (number of steps starting from S to generate the string a l b m with l ! m?

a) max (l, m) + 2

b) l+m+2

c) l + m + 3

d) max (l, m) + 3

Answer: A

Explanation: It is very clear from the previous solution that the no. of steps required depend upon the no. of a' s & b ' s which ever is higher & exceeds by 2 due to S " AC CB & C "! So max(l , m) + 2.

Question 212: [**GATE CS Mock]**

Which languages necessarily need heap allocation in the runtime environment?

a) Those that support recursion

b) Those that use dynamic scoping

c) Allow dynamic data structure

d) Those that use global variables

Answer: C

Question 213: [**GATE CS Mock**]

Peep-hole optimization is a form of

A. loop optimization
B. local optimization
C. data flow analysis
D. constant folding

Answer: D

Explanation: Redundant instructions may be discarded during the final stage of compilation by using a simple optimizing technique called peephole optimization.It is a kind of optimization performed over a very small set of instructions in a segment of generated code. The set is called a peephole or a window.

Question:214 [DRDO 2009]

A variable X has been assigned fresh values in statements numbered 6, 9 and 12 in a 25- statement program which does not have any jump instructions.This variable is used in statements numbered 7, 8, 10, 16 and 17.the statement range where the register, used by the variable X, could be assigned to some other variable are-

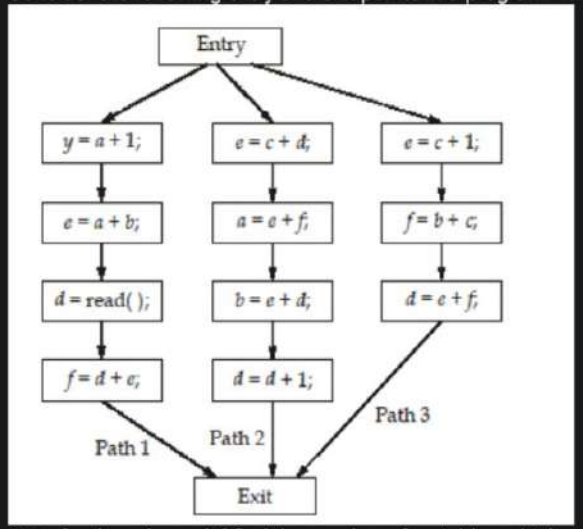
(A.) 8-9, 10- 12, 17- 25
(B.) 11, 18-25
(C.) 17-25
(D.) None of the above

Answer: B

Explanation: After the last use of a variable and next assignment, there is no use to preserve the variable. So, option B is the answer.

Question:215 [DRDO 2009]

Consider the following entry and exit points in a program. The variables *a, b, c, d, e, f* are live at the entry point.



Out of *a, b, c, d, e* and *f*, find the number of variables which are live at the exit point.

Answer:

Live variable → A variable V is live at a program point P, if some path from p to program exit contains an r-value occurrence of v which is not preceded by an l-value occurrence of v.

$a \rightarrow$ live in path 1

$b \rightarrow$ live in path 1

$c \rightarrow$ live in path 2

$d \rightarrow$ live in path 2

$e \rightarrow$ not live in all 3 paths

$f \rightarrow$ live in path 2

So, *a, b, c, d, f* are live at exit.

5

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 216: **[GATE CS Mock]**

Which of the following are true?

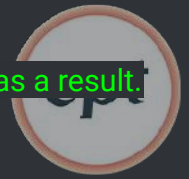
I. A programming language which does not permit global variables of any kind and has no nesting of procedures/functions, but permits recursion can be implemented with static storage allocation
II. Multi-level access link (or display) arrangement is needed to arrange activation records only if the programming language being implemented has nesting of procedures/functions
III. Recursion in programming languages cannot be implemented with dynamic storage allocation
IV. Nesting procedures/functions and recursion require a dynamic heap allocation scheme and cannot be implemented with a stack-based allocation scheme for activation records
V. 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. II and V only
B. I, III and IV only
C. I, II and V only
D. II, III and V only

Answer: False. Recursion cannot be implemented using static allocation.

I. True. Yes, we do need multi level access link in case of nested functions. Each level to traverse ARB of same level of nesting.
II. False. Recursion can only be implemented using dynamic memory allocation.
III. False. Recursion is done using memory in stack (ARBs in stack), not in heap.
IV. True. Yes, they cannot, once a function returns its activation record is no longer valid, so we cannot return a function as a result.

So, option (**A**) is correct.



Question 217: [**GATE CS Mock**]

For the C program given below the statement that does not hold true is

```
for (i = 0; ifor(j = 0; jif (i%c)
{
x += (4*j+5*i);
y+=(7+4*j);
}
}
}
```

A) There is a scope strength reduction
B) There is a scope of dead code elimination
C) There is a scope of common sub-expression elimination

D) None of the above

Answer: **There is a scope of dead code elimination**

Question 218: [**GATE CS Mock]**

The processor allows only register operands in its instructions and the given code segment is executed in that processor. For each instruction almost two source operands and one destination operand is available. Assume that all the variables are dead after this code segment.

```
z = x + y;

p = z * x;

q = c + x;

j = z * z;

If (j > x)

{

M = x*x;

}

else

{

p = p*p;

q = q*q;

}
```

(Refer the above data)

Question 219: [**GATE CS Mock**]

Only two registers are available in the instruction set architecture of the processor. The code motion moves the statements from one place to another while preserving correctness. The only allowed complier optimization is code motion. In the compiled code, the minimum number of spills to memory is

(A) 0

(B) 1

(C) 2

(D) 3

Answer: B

Question 220: [**GATE CS Mock**](**Refere the above questions data**]

Assume that no other optimization other than optimizing register allocation is applied. To compile this code segment without any spill to memory the minimum number of registers needed in the instruction set architecture of the processor is

(A) 3

(B) 6

(C) 4

(D) 5

Answer: C

Question 221: [**GATE CS Mock]**

he languages that need heap allocation in the runtime environment are

(A) Those that use global variables

(B) Those that use dynamic scoping

(C) Those that support recursion

(D) Those that allow dynamic data structure

Answer: D

Question 222: [**GATE CS Mock**]

An SDT that involves only synthesized attributes is called--------

A. S-attributed
B. L-attributed
C. Both A and B
D. None of the above.

Answer: A

Question 223: [**GATE CS Mock**]

is done by attaching rules or algorithms or program fragments to productions in a grammar.

A) Syntax-directed translation

B) Lexical analysis

C) Execution

D) Loading

Answer: A

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 224: [**GATE CS Mock**]

Which of the following is correct?

S_1: In heap allocation, memory is allocated at compile time whereas in stack allocation, memory is allocated at run time.

S2: Both, stack allocation and static allocation supports recursion.

A) S_1 is correct, S2 is not

B) S2 is correct, S_1 is not

C) Both are correct

D) Both are wrong

Answer: D

Explanation: Heap Allocation: This allocation is used to allocate memory at run time to the variables dynamically and then claim the memory back when the use of variables ends.

Stack Allocation: This allocation uses stack to allocate memory at run time to the data objects. Index and registers perform the memory addressing here. Recursive procedures are supported in this allocation.

Static Allocation: This allocation is used to allocate the memory at compile time to fixed named data objects. It does not support recursion.

S_1: In in heap allocation, memory is allocated at compile time whereas in stack allocation, memory is allocated at run time.

False. In both stack and heap allocation, memory is allocated at run time.

S2: Both, stack allocation and static allocation supports recursion. False. Static allocation does not support recursion whereas stack allocation allows recursion by organising the memory as stack and pushing and popping the activation records as the activation begins and ends respectively

Question 225: **[GATE CS Mock]**

Which of the following is NOT represented in a subroutine's activation record frame for a stack-based programming language?

A) Values of local variables

B) Return address

C) Heap area

D) Information needed to access non local variables

Answer: C

Explanation:

Actual Parameters:

The actual parameters used by the calling procedure. Commonly, these values are placed in the activation record but rather in registers, when possible, for greater efficiency.

However, space is shown for them to be completely general.

Control Link: A control link, pointing to the activation record of the caller.

Temporaries:Temporary values, such as those arising from the evaluation of expressions, in cases where those cannot be held in registers.

Therefore, a Direct Link is not present in the activation record of a procedure.

Explanation:

Heap area generally not represented in a subroutine's activation record frame because it can be globally accessible in the program. So if some space is allocated in the Heap area that space exists even after subroutine exits if not freed explicitly.

Question 226: [**GATE CS Mock**]

Consider the following code segment.

x = u - t;

y = x * v;

x = y + w;

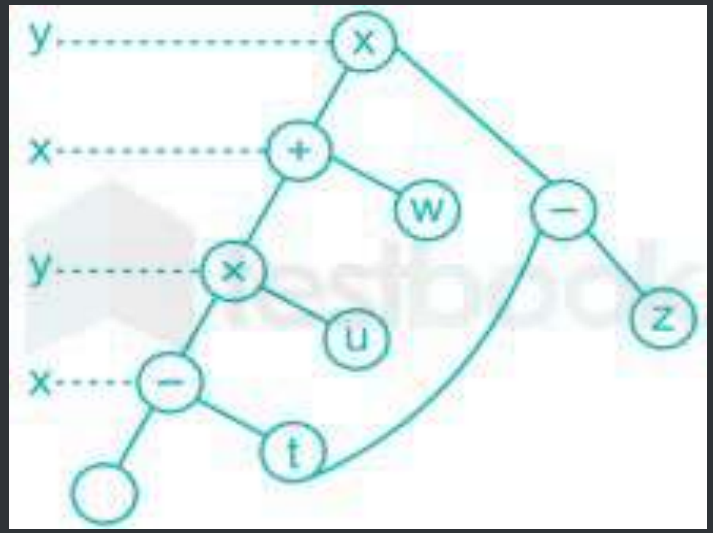
y = t - z;

y = x * y;

The minimum number of total variables required to convert the above code segment to static single assignment form is—----

Answer: 10

Explanation:



[Conti.....]

Given Grammar is

x = u - t;

y = x * v;

x = y + w;

y = t - z;

y = x * y;

Static single variable assignment

t1 = u-t

t2 = t1 * v

t3 = t2+ w

t4 = t - z

t5 = t3 t4

So, total 10 variables are needed in static single assignment form which are (t1, t2, t3, t4, t5, u, t, v, w, z).

Question 227: [**GATE CS Mock**]

Match the following.....

A. Activation record	p. Linking loader
B. Location counter	q. Garbage collection
C. Reference counts	r. Subroutine call
D. Address relocation	s. Assembler

A) p, q, r, s

B) q, r, s, p

C) r, s, q, p

D) r, s, p, q

Answer: C

Question 228: [**GATE CS Mock]**

S → E $ { printE.VAL }

E → E + E {E.VAL := E.VAL + E.VAL }

E → E * E {E.VAL := E.VAL * E.VAL }

E → (E) {E.VAL := E.VAL }

E → I {E.VAL := I.VAL }

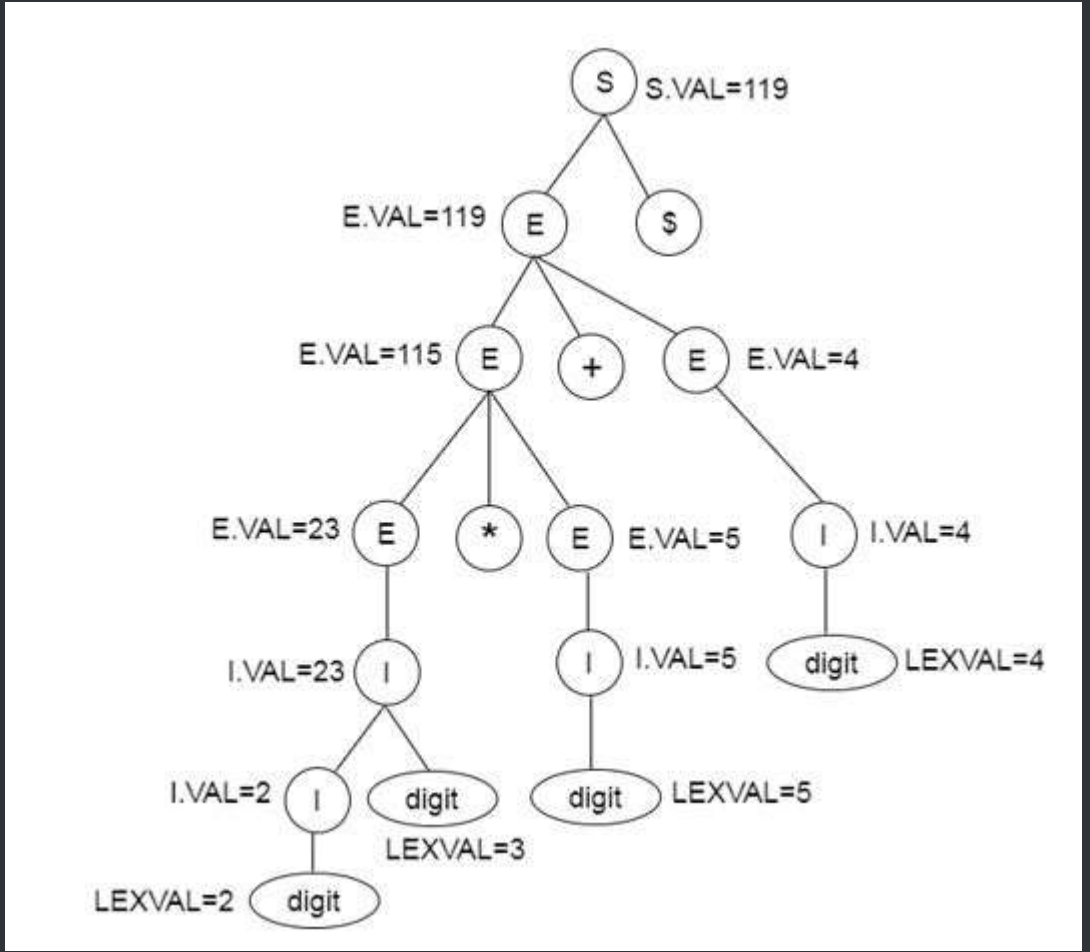
I → I digit {I.VAL := 10 * I.VAL + LEXVAL }

I → digit { I.VAL:= LEXVAL}

Input: 23*5+4$

Explanation: ParseTree ==========>

Output:119



Question 229: [**GATE CS Mock**]

Convert the given code into 3-address code

a := (-c * b) + (-c * d)

Answer:

t_1 := -c

t_2 := b*t_1

t_3 := -c

t_4 := d * t_3

t_5 := t_2 + t_4

a := t_5

Question 230: [**GATE CS Mock**]

Convert the given code into quadra tuple

a := -b * c + d

Answer:

3-address code

t_1 := -b

t_2 := c + d

t_3 := t_1 * t_2

a := t_3

Quadratuple:-

	Operator	Source 1	Source 2	Destination
(0)	uminus	b	-	t_1
(1)	+	c	d	t_2
(2)	*	t_1	t_2	t_3
(3)	:=	t_3	-	a

Question 231: **[GATE CS Mock]**

Consider the below grammar

1. S → id := E
2. E → E1 + E2
3. E → E1 * E2
4. E → (E1)
5. E → id

Convert the grammer into translation scheme for assignment statements

Answer:

Production rule	**Semantic actions**
S → id :=E	{p = look_up(id.name);If p ≠ nil then Emit (p = E.place) Else Error;}
E → E1 + E2	{E.place = newtemp(); Emit (E.place = E1.place '+' E2.place)}
E → E1 * E2	{E.place = newtemp(); Emit (E.place = E1.place '*' E2.place)}
E → (E1)	{E.place = E1.place}
E → id	{p = look_up(id.name); If p ≠ nil then Emit (p = E.place) Else Error;}

Note: [The Emit function is used for appending the three address code to the output file. Otherwise it will report an error.The newtemp() is a function used to generate new temporary variables]

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 232: [**GATE CS Mock**]

p>q AND r<s OR u>r Write the 3-address code for this Statement

Answer: Label

1. 100: **if** p>q **goto** 103
2. 101: t1:=0
3. 102: **goto** 104
4. 103: t1:=1
5. 104: **if** r>s **goto** 107
6. 105: t2:=0
7. 106: **goto** 108
8. 107: t2:=1
9. 108: **if** u>v **goto** 111
10. 109: t3:=0
11. 110: **goto** 112
12. 111: t3:= 1
13. 112: t4:= t1 AND t2
14. 113: t5:= t4 OR t3

Question 233: [**GATE CS Mock**]

Write the grammar for the given for statements

S **for** L = E1 step E2 to E3 **do** S1

Answer:

F → **for** L

T → F = E1 by E2 to E3 **do**

S → T S1

Question 234: **[GATE CS Mock]**

1. S → call id(Elist)
2. Elist → Elist, E
3. Elist → E

Solution:

A suitable transition scheme for procedure call would be:

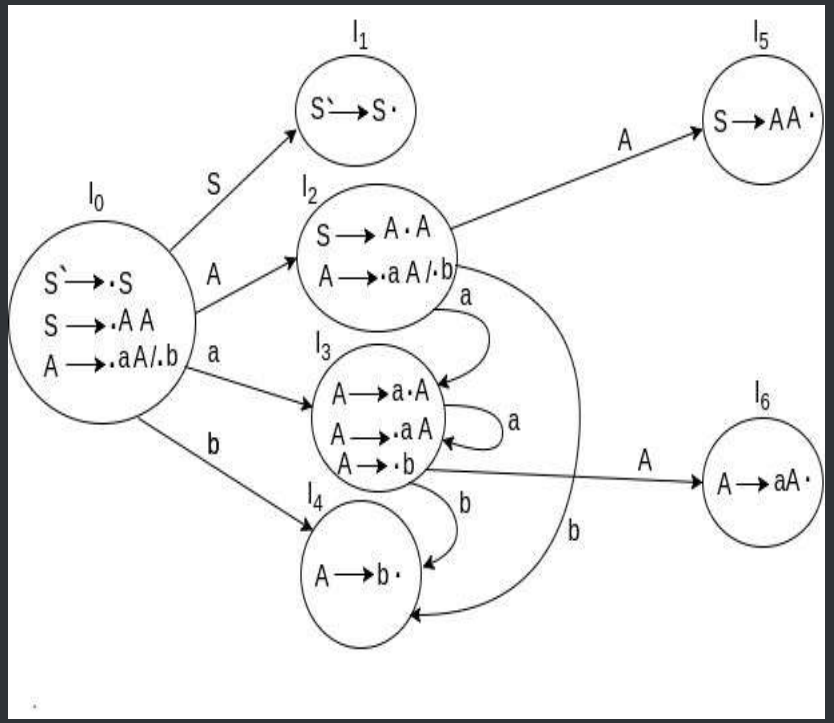
Production Rule	Semantic Action
S → call id(Elist)	for each item p on QUEUE do GEN (param p) GEN (call id.PLACE)
Elist → Elist, E	append E.PLACE to the end of QUEUE
Elist → E	initialize QUEUE to contain only E.PLACE

Question 235: [**GATE CS Mock**]

Consider the LR(0) for the following grammar?

1. $S \rightarrow AA$
2. $A \rightarrow aA \mid b$

DFA for canonical items.....



[Conti....]

LR parsing table

States	Action			Go to	
	a	b	$	A	S
I_0	S3	S4		2	1
I_1			accept		
I_2	S3	S4		5	
I_3	S3	S4		6	
I_4	r3	r3	r3		
I_5	r1	r1	r1		
I_6	r2	r2	r2		

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 236: [**GATE CS Mock**]

Consider the following Grammar construct the SLR(1) parsing table

S` → •E

E → •E + T

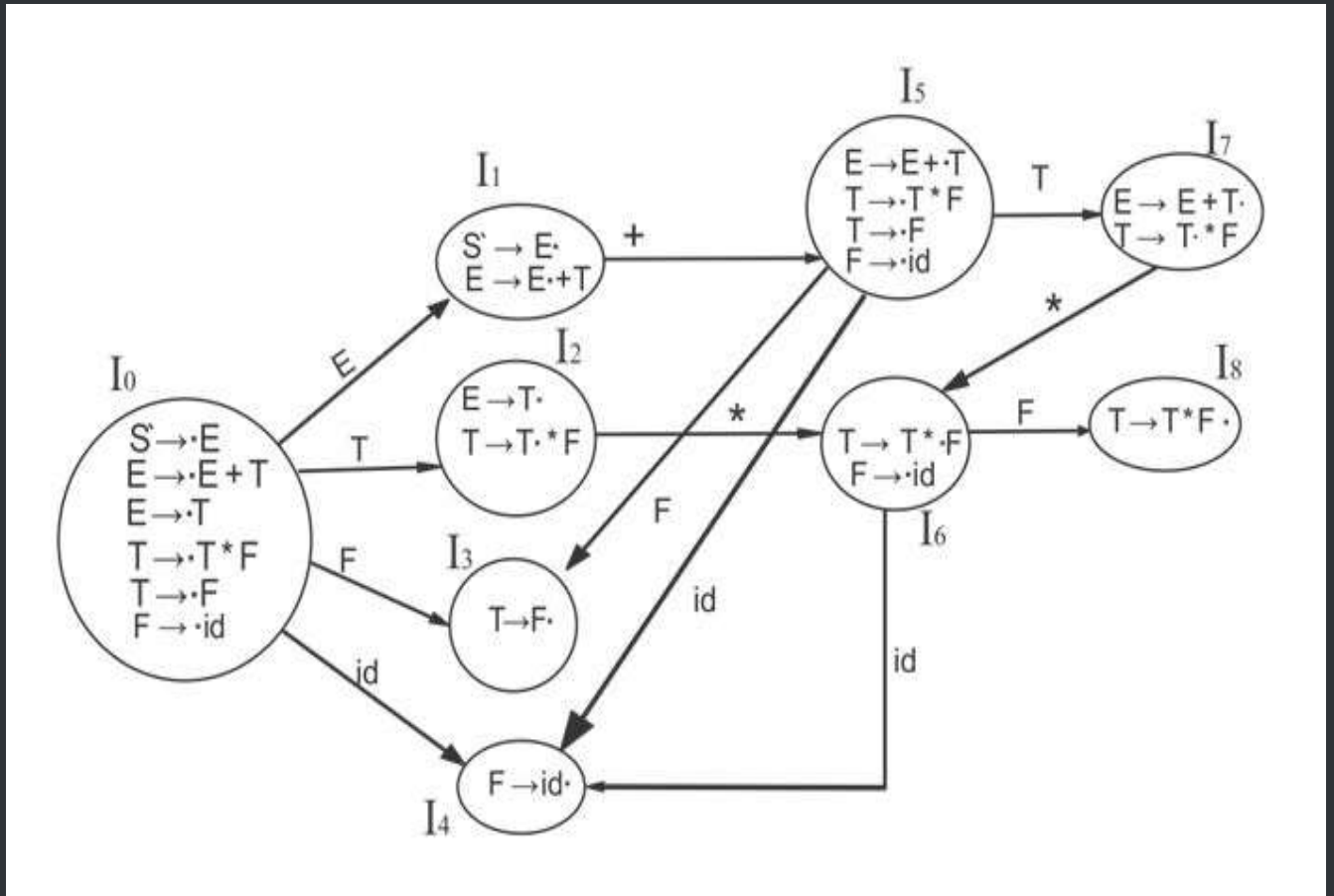
E → •T

T → •T * F

T → •F

F → •id

Solution: DFA for the Canonical items



[Conti....]

SLR (1) Table

States	Action				Go to		
	id	+	*	$	E	T	F
I_0	S_4				1	2	3
I_1		S_5		Accept			
I_2		R_2	S_6	R_2			
I_3		R_4	R_4	R_4			
I_4		R_5	R_5	R_5			
I_5	S4					7	3
I_6	S4						8
I_7		R1	S6	R1			
I_8		R3	R3	R3			

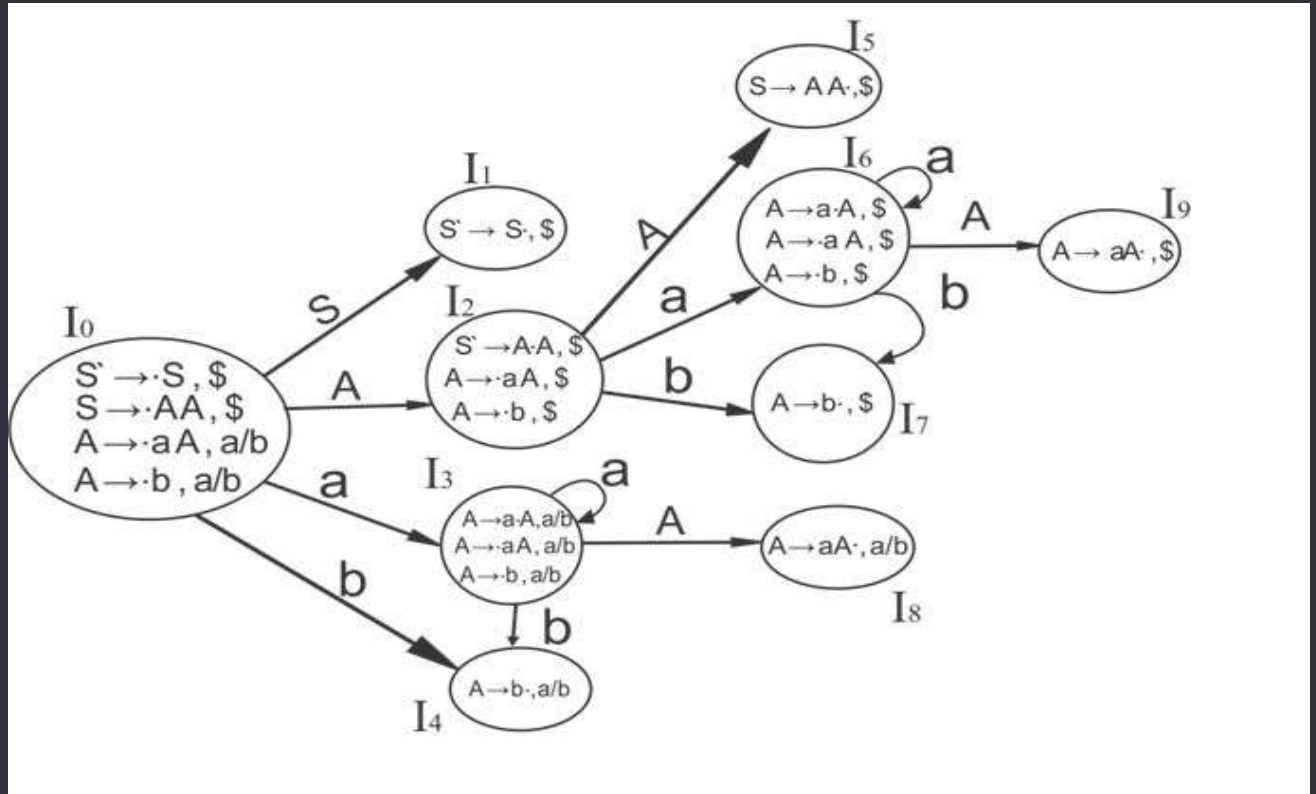
First (E) = First (E + T) ∪ First (T)
First (T) = First (T * F) ∪ First (F)
First (F) = {id}
First (T) = {id}
First (E) = {id}
Follow (E) = First (+T) ∪ {$} = {+, $}
Follow (T) = First (*F) ∪ First (F)
= {*, +, $}
Follow (F) = {*, +, $}

Question 237: [**GATE CS Mock**]

Construct the CLR(1) parser for the following grammar

1. S → AA
2. A → aA
3. A → b

Solution:



[Conti....]

States	a	b	S	S A
I_0	S_3	S_4		2
I_1			Accept	
I_2	S_6	S_7		5
I_3	S_3	S_4		8
I_4	R_3	R_3		
I_5			R_1	
I_6	S_6	S_7		9
I_7			R_3	
I_8	R_2	R_2		
I_9			R_2	

Question 238: [**GATE CS Mock]**

Construct the LALR(1) Parser for the given grammar

1. S → AA
2. A → aA
3. A → b

Solution:**I0 State:**

Add Augment production to the I0 State and Compute the ClosureL

I0 = Closure (S` → •S)

Add all productions starting with S in to I0 State because "•" is followed by the non-terminal. So, the I0 State becomes

I0 = S` → •S, $
S → •AA, $

Add all productions starting with A in modified I0 State because "•" is followed by the non-terminal. So, the I0 State becomes.

I0= S` → •S, $
S → •AA, $
A → •aA, a/b
A → •b, a/b

[Conti.....]

I1= Go to (I0, S) = closure (S` → S•, $) = S` → S•, $
I2= Go to (I0, A) = closure (S → A•A, $)

Add all productions starting with A in I2 State because "•" is followed by the non-terminal. So, the I2 State becomes

I2= S → A•A, $
A → •aA, $
A → •b, $

I3= Go to (I0, a) = Closure (A → a•A, a/b)

Add all productions starting with A in I3 State because "•" is followed by the non-terminal. So, the I3 State becomes

I3= A → a•A, a/b
A → •aA, a/b
A → •b, a/b

Go to (I3, a) = Closure (A → a•A, a/b) = (same as I3)
Go to (I3, b) = Closure (A → b•, a/b) = (same as I4)

I4= Go to (I0, b) = closure (A → b•, a/b) = A → b•, a/b
I5= Go to (I2, A) = Closure (S → AA•, $) =S → AA•, $
I6= Go to (I2, a) = Closure (A → a•A, $)

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

[Conti....]

Add all productions starting with A in I6 State because "•" is followed by the non-terminal. So, the I6 State becomes

I6 = A → a•A, $
A → •aA, $
A → •b, $

Go to (I6, a) = Closure (A → a•A, $) = (same as I6)
Go to (I6, b) = Closure (A → b•, $) = (same as I7)

I7= Go to (I2, b) = Closure (A → b•, $) = A → b•, $
I8= Go to (I3, A) = Closure (A → aA•, a/b) = A → aA•, a/b
I9= Go to (I6, A) = Closure (A → aA•, $) A → aA•, $

If we analyze then LR (0) items of I3 and I6 are same but they differ only in their lookahead.

I3 = { A → a•A, a/b
A → •aA, a/b
A → •b, a/b
}

I6= { A → a•A, $
A → •aA, $
A → •b, $
}

[Conti....]

Clearly I3 and I6 are same in their LR (0) items but differ in their lookahead, so we can combine them and called as I36.

I36 = { A → a•A, a/b/$
A → •aA, a/b/$
A → •b, a/b/$
}

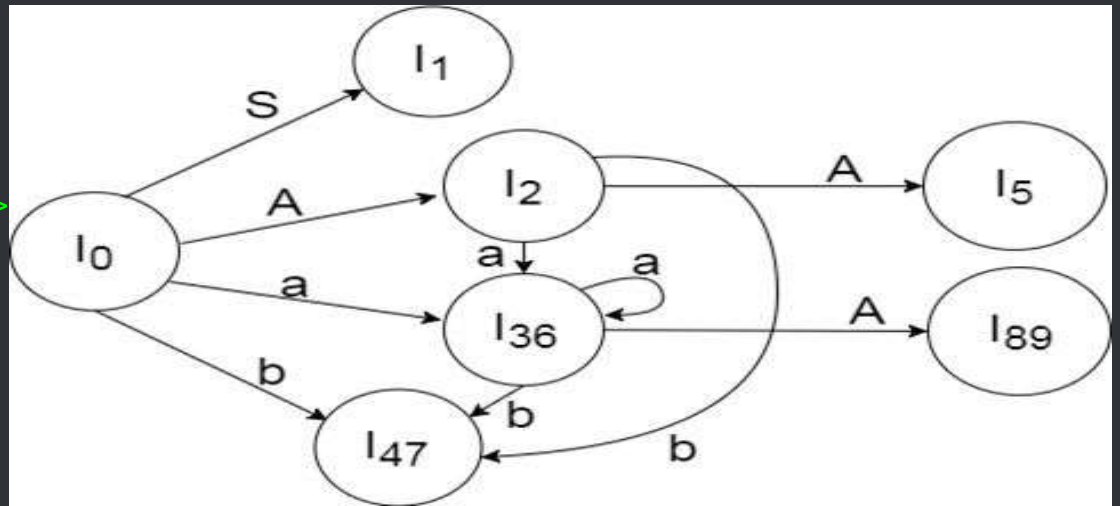
The I4 and I7 are same but they differ only in their look ahead, so we can combine them and called as I47.

I47 = {A → b•, a/b/$}

The I8 and I9 are same but they differ only in their look ahead, so we can combine them and called as I89.

I89 = {A → aA•, a/b/$}

DFA for the Canonical set of items==========>



[Conti....]

LALR (1) Parsing table:

States	a	b	$	S	A
I_0	S_{36}	S_{47}		12	
I_1		accept			
I_2	S_{36}	S_{47}			5
I_{36}	$S_{36}S_{47}$				89
I_{47}	R_3R_3	R_3			
I_5			R_1		
I_{89}	R_2	R_2	R_2		

Question 239: **[GATE CS Mock]**

What is not TRUE about data flow analysis?

A) Useful in register allocation

B) Dead code elimination is not possible

C) Eliminates common sub expression

D) Used in constant and variable propagation

Answer: B

Explanation: Data Flow Analysis: Analysis of flow of data in Control Flow Graph is called Data Flow Analysis. In simpler words, it is the analysis which determines the information regarding the definition and use of data in the program. This analysis can help in the optimisation of the program. Advantages of data flow analysis are:

1. Useful in register allocation: An expression is busy along a path if its evaluation already exists in one of the registers and none of the operands definition exists before its evaluation along the path, hence, helping in effectively utilising the register memory.

2. Dead code elimination: Data flow analysis helps in recognising dead code (code that is no more useful or is not required). Further we can eliminate such expressions from the code.

3. Eliminates common sub expressions: In data flow analysis, if a definition D reaches a point x and if there is path from D to x in which the original expression D is not eliminated then that path is considered. This helps in removing unnecessary duplicate expressions and keeping only the one copy which is required.

4. Used in constant and variable propagation: Data flow analysis helps a variable/constant to be alive from some point 'p' to end before it is redefined somewhere else and it becomes dead then.

Question 240: **[GATE CS Mock]**

Which of the following is a machine independent optimization?

A) Constant folding

B) Copy propagation

C) Peephole optimization

D) Loop jamming

Answer: A, B, D

Explanation: Machine Independent code optimization tries to make the intermediate code more efficient by transforming a section of code that doesn't involve hardware components like: CPU registers or any absolute memory location. Generally, it optimizes code by eliminating redundancies, reducing the number of lines in code, eliminating useless code or reducing the frequency of repeated code. Thus can be used on any processor irrespective of machine specifications.

Constant Folding - Constant Folding is a technique where the expression which is computed at compile time is replaced by its value. like:
int a = 12 + 5+ b; (Before Constant Folding)
int a = 17+b; (After Constant Folding)

Copy Propagation - Use one variable instead of other, in cases where assignments of the form a = b are used. These assignments are copy statements. We can efficiently use b at all required place instead of assign it to a. In short, elimination of copies in the code is Copy Propagation.

Loop jamming - Combining the loops that carry out the same operations. And this is totally machine independent.

Peephole optimization is machine dependent optimization. And constant folding, copy propagation and loop jamming are machine independant optimization.

Question 241: **[GATE CS Mock]**

In the given C-code which line is a part of loop invariant?

```
for (i = 0; i < n; i++)
{
1. J = J + i;
2. x = y * z;
}
```

A) Only line 1

B) Only line 2

C) Both 1 and 2 line

D) None of the above

Answer: B

Explanation: Only line 2, because x = yz (does not depend on loop it remains constant) so it is loop invariant.

and line 1 depends on loop. J = J + i, where i is loop variable. So, it is not loop inveriant.

Question 242: [**GATE CS Mock**]

Consider the following C-program.

```
a = 1;
b = c * d;
e = c + a;
If (e < 10)
{
t1 = 2 * d;
}
else
{
t1 = e/2;
}
```

Which of the following optimization methods can be used?

A) Constant folding

B) Constant propagation

D) Dead code elimination

D) None of the above

Answer: B, D

Explanation: b = c * d is a dead code. Because there is no use of this expression.

a = 1; e = c + a; is an example of constant propagation. because we can directly use 1 inplace of a. like: e = c + 1.

Question 243: **[GATE CS Mock]**

Which of the following is a type of an out-of-order execution, with the reordering done by a compiler

A) loop unrolling

B) dead code elimination

C) strength reduction

D) Software Pipelinging

Answer: D

Explanation:

• Out of Order Execution is also called Dynamic Execution. In this mode of execution, the instructions aren't executed precisely as per the program control flow.

• It is a technique used to get back some of that wasted execution bandwidth.

• With out-of-order execution, the processor would issue each of the instructions in program order, and then enter a new pipeline stage called "read operands" during which instructions whose operands are available would move to the execution stage, regardless of their order in the program.

• The term issue could be redefined at this point to mean "issue and read operands." • The software pipelining is a method that is used to optimize loops, in a way that parallels hardware pipelining. It is out of order execution, except that the reordering is done by the compiler.

Question 244: [**GATE CS Mock**]

Which one of the following is NOT performed during compilation?

A) Dynamic memory allocation

B) Type checking

C) Symbol table management

D) Inline expansion

Answer: A

Explanation: Dynamic Memory Allocation:

Performed during run time when a program executing and require a memory block.

Type Checking:

Check performed during the semantic analysis of compilation.

Symbol Table:

Management is to store and retrieve the information about tokens during compilation phase.

Inline Expansion:

Type of macro in pre-processor which happens before even compilation started.

Replace a function call by the body of respective function.

Hence option A is the correct answer.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 245: **[GATE CS Mock]**

Consider the expression (a − 1) * (((b + c) / 3) + d). Let X be the minimum number of registers required by an optimal code generation (without any register spill) algorithm for a load/store architecture, in which (i) only load and store instructions can have memory operands and (ii) arithmetic instructions can have only register or immediate operands. The value of X is—--------

Answer: 2

Explanation: Register Spilling means moving a variable from register to memory while moving a variable from memory to register is filling. Advantage of using a register rather than memory is fast processing.

Explanation:
Expression: (a - 1)*(((b + c) / 3) + d)
Load Ro, b // Load b into register Ro
Load R1, c // Load c in register R1
Add R1, Ro // R_1 = R1 + Ro
Now, Ro register is free.
DIV R1, // R_1 = R1/3
Load Ro, d // Load d in register Ro
ADD R1, Ro // R_1 = R1 + Ro
Load Ro, a // Load a in register Ro
SUB Ro, 1 // Ro=Ro - 1

MUL R1, Ro

So minimum 2 registers are needed

Question 246: [**GATE CS Mock**]

Which of the following comment about peep-hole optimization is true?

A) It is applied to small part of the code and applied repeatedly

B) It can be used to optimize intermediate code

C) It can be applied to a portion of the code that is not contiguous

D) It is applied in symbol table to optimize the memory requirements

Answer: A

Explanation: Peephole optimization is an optimization technique performed on a small set of compiler- generated instructions; the small set is known as the peephole or window.

Important Points:

An optimizing compiler is a compiler that tries to minimize or maximize some attributes of an executable computer program.

Common requirements are to minimize a program's execution time, memory requirement, and power consumption.

Question 247: **[GATE CS Mock]**

Some code optimizations are carried out on the intermediate code because:

A) they enhance the portability of the compiler 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

Answer: A

Explanation: • The main purpose of doing some code-optimization on intermediate code generation is to enhance the portability of the compiler to target processors

Important points:

Program analysis is more accurate on intermediate code than on machine code. It is also correct but the most optimal answer is 1.

Question 248: **[GATE CS Mock]**

Consider the following grammar:Consider the following grammar:

S- aBCbD

BbBh/ BD/d

C→ CE/ a

D→ EC/b/E

E→CDb / €

Which of the following is/are FALSE?

A) Follow(B) = Follow(C)

B) Follow(C) = Follow(D)

C) Follow(B) = Follow(E)

D) Follow(C) = Follow(D) = Follow(E)

Answer: A, C

Explanation: ==========>

Follow for given non-terminals

Hence option A and C are false

	Follow
S	{$}
B	{a, b, h}
C	{a, b, h, $}
D	{a, b, h, $}
E	{a, b, h, $}

Question 249: **[GATE CS Mock]**

Consider the following grammar

S→m|mn| mno

Choose correct statement from the following:

1. The grammar is LL (2)
2. The grammar is LL (4)
3. The grammar is LL (1)
4. The grammar is LL (3)

Answer:

Given grammar:

S→m|mn| mno

Case 1: For LL(1)

First(m) n first (mn) = m (which is not empty).

So, grammar is not LL(1).

Case 2: For LL(2)

Second (mn) N second(mno) = n (which is not empty)

So, grammar is not LL(2).

Case 3: For LL(3)

Third (mn) n third (mno) = empty

So, grammar is LL(3).

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 250: **[GATE CS Mock]**

Consider the following

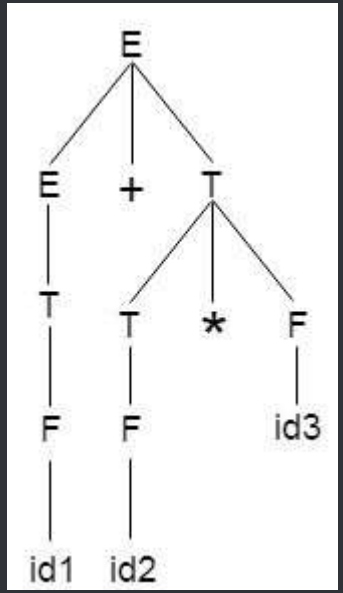
E → E+T/T

T → T*F/F

F → id

Construct the operator precedence parser for the string w = id + id * id

Solution: Let us consider a parse tree for it as follows:



On the basis of above tree, we can design following operator precedence table:

	E	T	F	id	+	*	$
E	X	X	X	X	≐	X	⋗
T	X	X	X	X	⋗	≐	⋗
F	X	X	X	X	⋗	⋗	⋗
id	X	X	X	X	⋗	⋗	⋗
+	X	≐	⋖	⋖	X	X	X
*	X	X	≐	⋖	X	X	X
$	⋖	⋖	⋖	⋖	X	X	X

[Conti.....]

Now let us process the string with the help of the above precedence table:

$ ⋖ id1 ⋗ + id2 * id3 $

$ ⋖ F ⋗ + id2 * id3 $

$ ⋖ T ⋗ + id2 * id3 $

$ ⋖ E ≐ + ⋖ id2 ⋗ * id3 $

$ ⋖ E ≐ + ⋖ F ⋗ * id3 $

$ ⋖ E ≐ + ⋖ T ≐ * ⋖ id3 ⋗ $

$ ⋖ E ≐ + ⋖ T ≐ * ≐ F ⋗ $

$ ⋖ E ≐ + ≐ T ⋗ $

$ ⋖ E ≐ + ≐ T ⋗ $

$ ⋖ E ⋗ $

Accept.

Question 251: **[GATE CS Mock]**

Production Rules of Grammar

E -> TE'

E' -> +T E'|Є

T -> F T'

T' -> *F T' | Є

F -> (E) | id

Construct the first set for the non-termnals of above grammer

Answer:

FIRST(S) = FIRST(ACB) U FIRST(Cbb) U FIRST(Ba)

= { d, g, h, b, a, Є}

FIRST(A) = { d } U FIRST(BC)

= { d, g, h, Є }

FIRST(B) = { g , Є }

FIRST(C) = { h , Є }

Question 251: **[GATE CS Mock]**

Calculate the first and follow functions for the given grammar-

S → (L) / a

L → SL’

L’ → ,SL’ / ∈

Answer:

First Functions-

- First(S) = { (, a }
- First(L) = First(S) = { (, a }
- First(L’) = { , , ∈ }

Follow Functions-

- Follow(S) = { $ } ∪ { First(L’) – ∈ } ∪ Follow(L) ∪ Follow(L’) = { $, , ,) }
- Follow(L) = {) }
- Follow(L’) = Follow(L) = {) }

Question 252: [**GATE CS Mock**]

Calculate the first and follow functions for the given grammar-

S → AaAb / BbBa

A → ∈

B → ∈

Answer:

First Functions-

- First(S) = { First(A) – ∈ } ∪ First(a) ∪ { First(B) – ∈ } ∪ First(b) = { a , b }
- First(A) = { ∈ }
- First(B) = { ∈ }

Follow Functions-

- Follow(S) = { $ }
- Follow(A) = First(a) ∪ First(b) = { a , b }
- Follow(B) = First(b) ∪ First(a) = { a , b }

Question 253: **[GATE CS Mock]**

Calculate the first and follow functions for the given grammar-

$E \rightarrow E + T / T$

$T \rightarrow T \times F / F$

$F \rightarrow (E) / id$

Answer:

We have-

- The given grammar is left recursive.
- So, we first remove left recursion from the given grammar.

After eliminating left recursion, we get the following grammar-

$E \rightarrow TE'$

$E' \rightarrow + TE' / \in$

$T \rightarrow FT'$

$T' \rightarrow \times FT' / \in$

$F \rightarrow (E) / id$

[Conti.....]

First Functions-

- First(E) = First(T) = First(F) = { (, id }
- First(E') = { + , ∈ }
- First(T) = First(F) = { (, id }
- First(T') = { x , ∈ }
- First(F) = { (, id }

Follow Functions-

- Follow(E) = { $,) }
- Follow(E') = Follow(E) = { $,) }
- Follow(T) = { First(E') – ∈ } ∪ Follow(E) ∪ Follow(E') = { + , $,) }
- Follow(T') = Follow(T) = { + , $,) }
- Follow(F) = { First(T') – ∈ } ∪ Follow(T) ∪ Follow(T') = { x , + , $,) }

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 257: **[GATE CS Mock]**

Calculate the first and follow functions for the given grammar-

S → ACB / CbB / Ba

A → da / BC

B → g / ∈

C → h / ∈

Answer:

First Functions-

- First(S) = { First(A) – ∈ } ∪ { First(C) – ∈ } ∪ First(B) ∪ First(b) ∪ { First(B) – ∈ } ∪ First(a) = { d , g , h , ∈ , b , a }
- First(A) = First(d) ∪ { First(B) – ∈ } ∪ First(C) = { d , g , h , ∈ }
- First(B) = { g , ∈ }
- First(C) = { h , ∈ }

Follow Functions-

- Follow(S) = { $ }
- Follow(A) = { First(C) – ∈ } ∪ { First(B) – ∈ } ∪ Follow(S) = { h , g , $ }
- Follow(B) = Follow(S) ∪ First(a) ∪ { First(C) – ∈ } ∪ Follow(A) = { $, a , h , g }
- Follow(C) = { First(B) – ∈ } ∪ Follow(S) ∪ First(b) ∪ Follow(A) = { g , $, b , h }

Question 258: **[GATE CS Mock]**

Calculate the first and follow functions for the given grammar-

S → A

A → aB / Ad

B → b

C → g

Answer: We have-

The given grammar is left recursive. So, we first remove left recursion from the given grammar.

After eliminating left recursion, we get the following grammar-

S → A

A → aBA'

A' → dA' / ∈

B → b

C → g

First Functions-

- First(S) = First(A) = { a }
- First(A) = { a }
- First(A') = { d , ∈ }
- First(B) = { b }
- First(C) = { g }

Follow Functions-

- Follow(S) = { $ }
- Follow(A) = Follow(S) = { $ }
- Follow(A') = Follow(A) = { $ }
- Follow(B) = { First(A') – ∈ } ∪ Follow(A) = { d , $ }
- Follow(C) = NA

Question 259: **[GATE CS Mock]**

Calculate the first and follow functions for the given grammar-

S → aBDh

B → cC

C → bC / ∈

D → EF

E → g / ∈

F → f / ∈

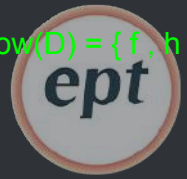
Answer:

First Functions-

- First(S) = { a }
- First(B) = { c }
- First(C) = { b , ∈ }
- First(D) = { First(E) – ∈ } ∪ First(F) = { g , f , ∈ }
- First(E) = { g , ∈ }
- First(F) = { f , ∈ }

Follow Functions-

- Follow(S) = { $ }
- Follow(B) = { First(D) – ∈ } ∪ First(h) = { g , f , h }
- Follow(C) = Follow(B) = { g , f , h }
- Follow(D) = First(h) = { h }
- Follow(E) = { First(F) – ∈ } ∪ Follow(D) = { f , h }
- Follow(F) = Follow(D) = { h }



Question 260: **[GATE CS Mock]**

Consider the following grammar and construct the first, follow sets for all non-terminals

E -> TE'

E' -> +T E'|Є

T -> F T'

T' -> *F T' | Є

F -> (E) | id

FIRST set

FIRST(E) = FIRST(T) = { (, id }

FIRST(E') = { +, Є }

FIRST(T) = FIRST(F) = { (, id }

FIRST(T') = { *, Є }

FIRST(F) = { (, id }

FOLLOW Set

FOLLOW(E) = { $,) } // Note ')' is there because of 5th rule
FOLLOW(E') = FOLLOW(E) = { $,) } // See 1st production rule
FOLLOW(T) = { FIRST(E') – Є } U FOLLOW(E') U FOLLOW(E) = { + , $,) }
FOLLOW(T') = FOLLOW(T) = { + , $,) }
FOLLOW(F) = { FIRST(T') – Є } U FOLLOW(T') U FOLLOW(T) = { *, +, $,) }

Question 261: [**GATE CS Mock**]

Consider the following expression and construct a DAG for it—> (a + b) x (a + b + c)

Answer:

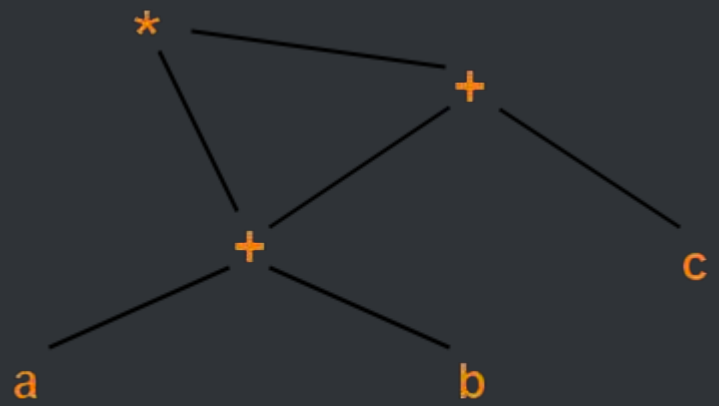
Three Address Code for the given expression is-

T1 = a + b

T2 = T1 + c

T3 = T1 x T2

Now, Directed Acyclic Graph is-



Directed Acyclic Graph

Question 262: **[GATE CS Mock]**

Consider the following expression and construct a DAG for it- (((a + a) + (a + a)) + ((a + a) + (a + a)))

Answer:

Directed Acyclic Graph for the given expression is-

Directed Acyclic Graph

Question 263: [**GATE CS Mock**]

Consider the following block and construct a DAG for it-

(1) a = b x c

(2) d = b

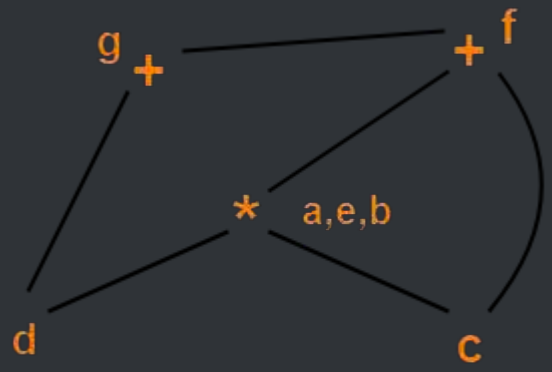
(3) e = d x c

(4) b = e

(5) f = b + c

(6) g = f + d

Answer: Directed Acyclic Graph for the given block is-



Directed Acyclic Graph

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 264: [**GATE CS Mock]**

Consider the following basic block-

B10:

S1 = 4 x I

S2 = addr(A) – 4

S3 = S2[S1]

S4 = 4 x I

S5 = addr(B) – 4

S6 = S5[S4]

S7 = S3 x S6

S8 = PROD + S7

PROD = S8

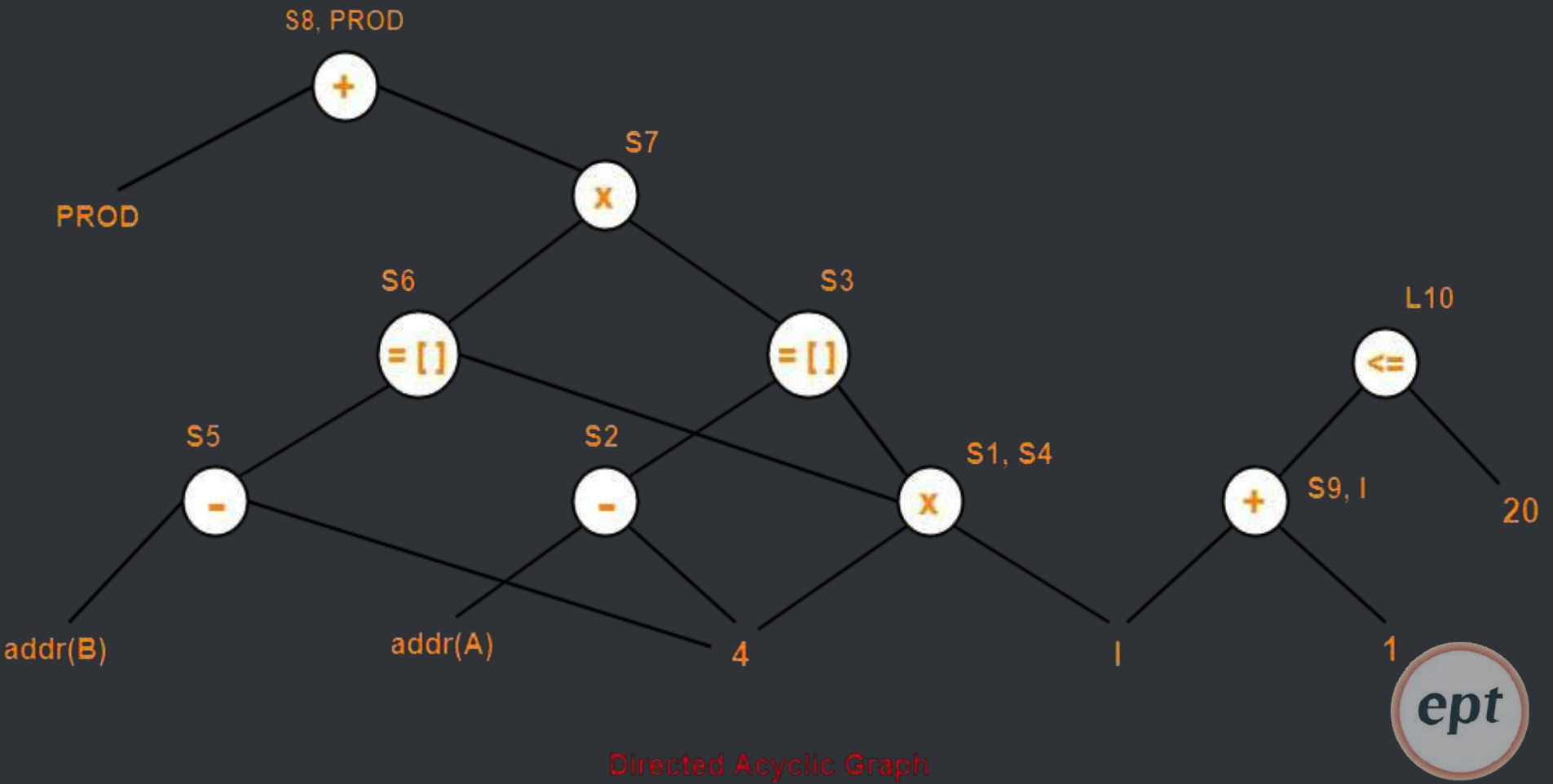
S9 = I + 1

I = S9

If I <= 20 goto L10

1. Draw a directed acyclic graph and identify local common sub-expressions.
2. After eliminating the common sub-expressions, re-write the basic block.

Answer: Directed Acyclic Graph for the given basic block is-



Directed Acyclic Graph

[Conti....]

In this code fragment,

- 4 x I is a common sub-expression. Hence, we can eliminate because S1 = S4.
- We can optimize S8 = PROD + S7 and PROD = S8 as PROD = PROD + S7.
- We can optimize S9 = I + 1 and I = S9 as I = I + 1.

After eliminating S4, S8 and S9, we get the following basic block-

B10:

S1 = 4 x I

S2 = addr(A) – 4

S3 = S2[S1]

S5 = addr(B) – 4

S6 = S5[S1]

S7 = S3 x S6

PROD = PROD + S7

I = I + 1

If I <= 20 goto L10

Question 265: [**GATE CS Mock]**

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;
printf("in, %s", ** ++ P);
}
```

A) 5

B) 1

C) 4

D) 3

Answer: B

Explanation: 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|======>single token

Water is considered as a single token.

Question 266: [**GATE CS Mock**]

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

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

A) 3 B)5 C)6 D)No lexical error

Answer: D

Explanation: 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.

Question 267: [**GATE CS Mock**]

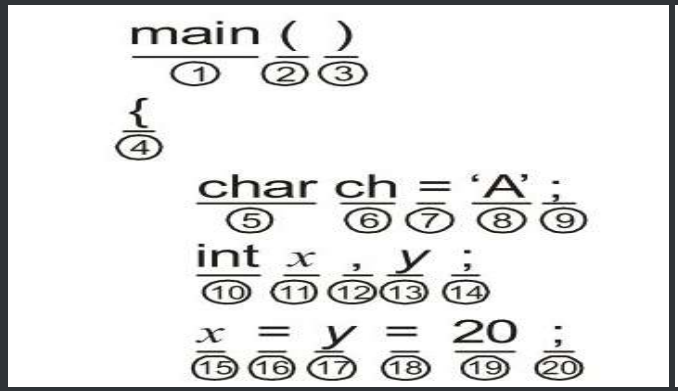
Consider the following program.

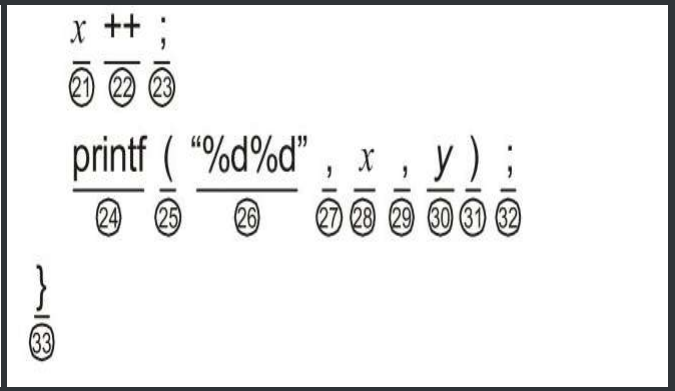
```
1. main ()
2.  {
3. char ch = 'A';
4. int x, y;
5. x = y = 20;
6. x ++;
7. printf("%d %d", x, y);
8. }
```

The number of tokens in the above program are—---------

Answer: 33

Explanation:





Question 268: [**GATE CS Mock**]

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)

Answer: A

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 269: [**GATE CS Mock**]

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;
}
```

Answer: 35

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

Question 270: [**GATE CS Mock**]

Find the number of tokens in the following C code using lexical analyzer of compiler. —----------

main()----------->3

{ —--------------->1

int a; ---------->3

int* a;---------->4

For (i = 0; i < n; i++)---------->13

{---------------------------->1

Printf ("True");---------->5

++* a;---------------------->4

a = a + a;-------------------->6

}-------------------->1

}-------------------->1

==========

Total 42 tokens

Question 271: [**GATE CS Mock]**

Consider the following code

```
int a = 10;
Float b = 20.5;
Printf("c= %d, d = %", ++a, b++);
```

The number of tokens in the given code fragment are—-------

Answer: 21

Explanation: int a = 10; Number of tokens are 5.

Float b = 20.5; Number of tokens are 5.

Printf("c %d, d = %F", ++a, b++); =

Number of tokens are 11

Total number of tokens are = 21

Question 272: **[GATE CS Mock]**

Consider the following C Program Void main()

{

int i = 20;

whil(i - -)

Printf("GATE 2020\n");

}

The output of the given program is?

A) Print Gate 2020 20 times
B) Print Gate 2020 19 times
C) Lexical error
D) Syntax error

Answer: D

Explanation: the keyword while is misspelled hence produces syntax error.

Question 273: **[GATE CS Mock]**

Which of the following errors that are detected in the Lexical Analysis Phase.

1. Numeric literals that are too long.

2. Identifies that are too long

3. ICC-formed numeric literals

4. Input characters that are not in language.

A) 1 and 2 only

B) 1, 3 and 4 only

C) 3 and 4 only

D) 1, 2, 3 and 4

Answer: D

Explanation: Lexical Analysis will detect all the mentioned errors.

1. Numeric literals that are too long.

2. ICC-formed numeric literals.

3. Identifiers that are too long.

4. Input characters that are not in the language.

Question 274: [**GATE CS Mock**]

Construct the syntax tree for the following expression (a + b) * (c – d) + ((e / f) * (a + b))

Answer:

We convert the given arithmetic expression into a postfix expression as-

(a + b) * (c – d) + ((e / f) * (a + b))

ab+ * (c – d) + ((e / f) * (a + b))

ab+ * cd- + ((e / f) * (a + b))

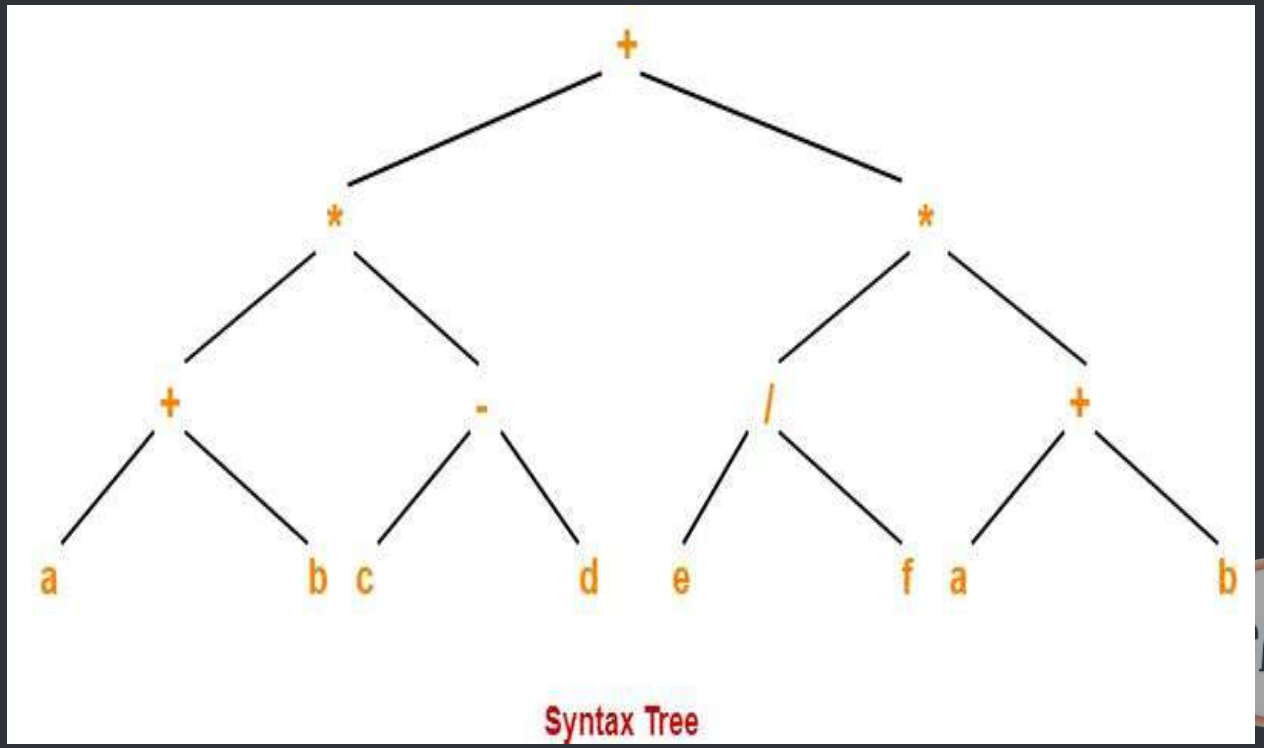
ab+ * cd- + (ef/ * (a + b))

ab+ * cd- + (ef/ * ab+)

ab+ * cd- + ef/ab+*

ab+cd-* + ef/ab+*

ab+cd-*ef/ab+*+



Syntax Tree

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 278: [**GATE CS Mock**]

Generate three address code for the following code-

```
c = 0
do
{
    if (a < b) then
        x++;
    else
        x–;
        c++;
} while (c < 5)
```

Answer:

Three address code for the given code is-

1. c = 0
2. if (a < b) goto (4)
3. goto (7)
4. T1 = x + 1
5. x = T1
6. goto (9)
7. T2 = x – 1
8. x = T2
9. T3 = c + 1
10. c = T3
11. if (c < 5) goto (2)

Question 279: [**GATE CS Mock**]

Generate three address code for the following code-

```
while (A < C and B > D) do
    if A = 1 then C = C + 1
    else
        while A <= D
            do A = A + B
```

Answer:

Three address code for the given code is-

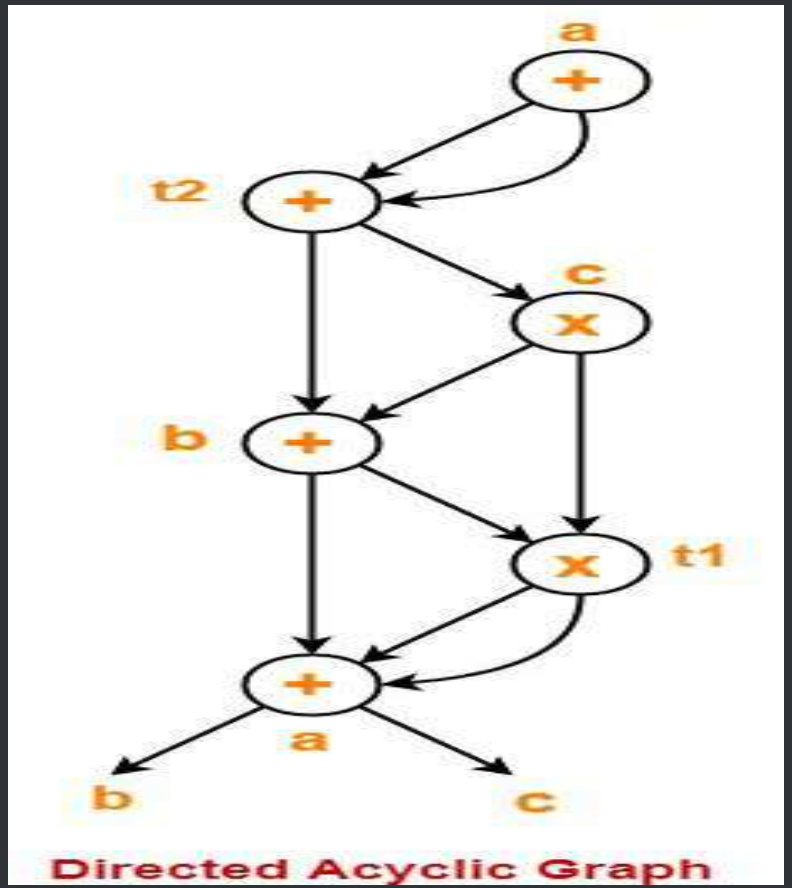
1. if (A < C) goto (3)
2. goto (15)
3. if (B > D) goto (5)
4. goto (15)
5. if (A = 1) goto (7)
6. goto (10)
7. T1 = c + 1
8. c = T1
9. goto (1)
10. if (A <= D) goto (12)
11. goto (1)
12. T2 = A + B
13. A = T2
14. goto (10)

Question 280: **[GATE CS Mock]**

Construct a DAG for the following three address code-

1. a = b + c
2. t1 = a x a
3. b = t1 + a
4. c = t1 x b
5. t2 = c + b
6. a = t2 + t2

Answer:



Directed Acyclic Graph

Question 281: [**GATE CS Mock**]

Consider the following code-

```
prod = 0 ;
i = 1 ;
do
{
    prod = prod + a[ i ] x b[ i ] ;
    i = i + 1 ;
} while (i <= 10) ;
```

Compute the basic blocks and draw the flow graph?

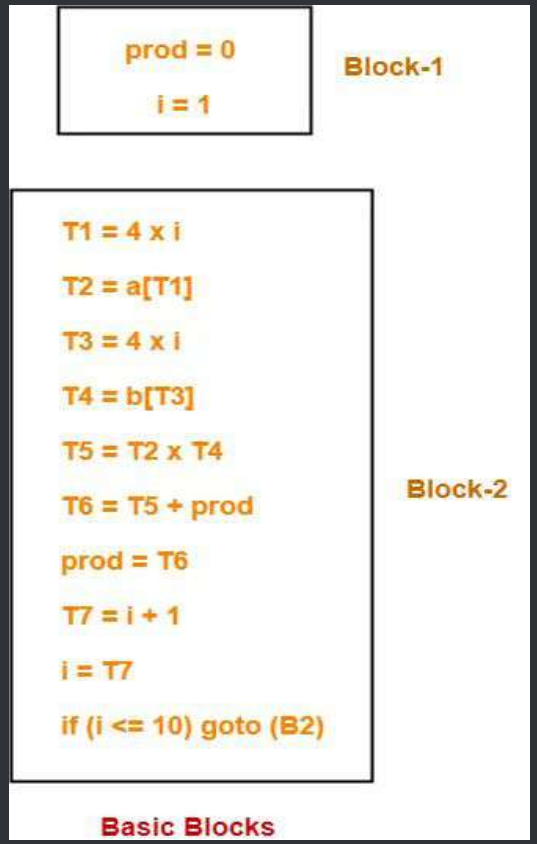
Answer: Three address code for given code

```
prod = 0
i = 1
T1 = 4 x i
T2 = a[T1]
T3 = 4 x i
T4 = b[T3]
T5 = T2 x T4
T6 = T5 + prod
prod = T6
T7 = i + 1
i = T7
if (i <= 10) goto (3)
```

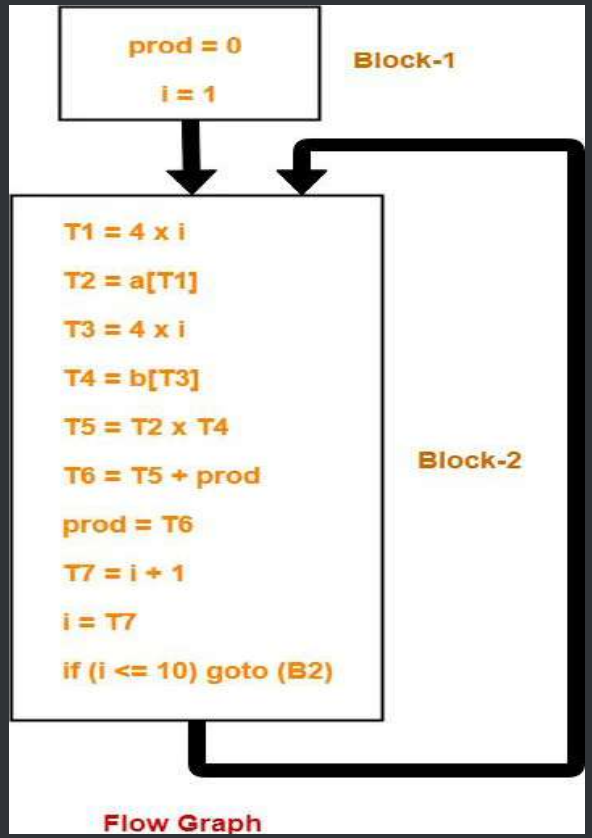
[Conti.....]

We identify the leader statements as-

- prod = 0 is a leader because first statement is a leader.
- T1 = 4 x i is a leader because target of conditional or unconditional goto is a leader.



The flow graph is:



Question 282: [**GATE CS Mock**]

pi = 3.14

radius = 10

Area of circle = pi x radius x radius

What type of optimization can be performed on the above code?

Answer: Constant Propagation

Explanation:

- his technique substitutes the value of variables ‘pi’ and ‘radius’ at compile time.
- It then evaluates the expression 3.14 x 10 x 10.
- The expression is then replaced with its result 3.14.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 283: [**GATE CS Mock**]

S1 = 4 x i

S2 = a[S1]

S3 = 4 x j

S4 = 4 x i // **Redundant Expression**

S5 = n

S6 = b[S4] + S5

Optimize the above code?

Answer: Here we can perform common sub-expression elimination

- The redundant expressions are eliminated to avoid their re-computation.
- The already computed result is used in the further program when required.

Code after optimization:

S1 = 4 x i

S2 = a[S1]

S3 = 4 x j

S5 = n

S6 = b[S1] + S5

Question 284: [**GATE CS Mock**]

Optimize the below code

```
for ( int j = 0 ; j < n ; j ++)
{
x = y + z ;
a[j] = 6 x j;
}
```

Answer: Here we can perform code movement

Code after optimization:

```
x = y + z ;
for ( int j = 0 ; j < n ; j ++)
{
a[j] = 6 x j;
}
```

Question 285: [**GATE CS Mock**]

Optimiza the below code?

```
i = 0 ;
if (i == 1)
{
a = x + 5 ;
}
```

Answer: Here we can perform Dead code elimination.

Code after optimization:

```
i = 0 ;
```

Question 286: [**GATE CS Mock**]

How can you optimiza below code?

B = A x 2

Answer:

We can perform strength reduction

Code after optimization:

B = A <<1

Question 287: [**GATE CS Mock**]

Consider the following grammar and eliminate left recursion-

A → ABd / Aa / a

B → Be / b

Answer:

The grammar after eliminating left recursion is-

A → aA'

A' → BdA' / aA' / ∈

B → bB'

B' → eB' / ∈

Question 288: [**GATE CS Mock**]

Consider the following grammar and eliminate left recursion-

$E \rightarrow E + E / E \times E / a$

Answer:

The grammar after eliminating left recursion is-

$E \rightarrow aA$

$A \rightarrow +EA / xEA / \in$

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 289: [**GATE CS Mock**]

Consider the following grammar and eliminate left recursion-

$E \rightarrow E + T / T$

$T \rightarrow T \times F / F$

$F \rightarrow id$

Answer:

The grammar after eliminating left recursion is-

$E \rightarrow TE'$

$E' \rightarrow +TE' / \in$

$T \rightarrow FT'$

$T' \rightarrow xFT' / \in$

$F \rightarrow id$

Question 290: **[GATE CS Mock]**

Consider the following grammar and eliminate left recursion-

S → (L) / a

L → L , S / S

Answer:

The grammar after eliminating left recursion is-

S → (L) / a

L → SL'

L' → ,SL' / ∈

Question 291: **[GATE CS Mock]**

Consider the following grammar and eliminate left recursion-

A → Ba / Aa / c

B → Bb / Ab / d

Answer:

this is a case of indirect left recursion.

Step-01:

First let us eliminate left recursion from A → Ba / Aa / c

Eliminating left recursion from here, we get-

A → BaA' / cA'

A' → aA' / ∈

Now, given grammar becomes-

A → BaA' / cA'

A' → aA' / ∈

B → Bb / Ab / d

Step-02:

Substituting the productions of A in B → Ab, we get the following grammar-

A → BaA' / cA'

A' → aA' / ∈

B → Bb / BaA'b / cA'b / d

Step-03:

Now, eliminating left recursion from the productions of B, we get the following grammar-

A → BaA' / cA'

A' → aA' / ∈

B → cA'bB' / dB'

B' → bB' / aA'bB' / ∈

This is the final grammar after eliminating left recursion.

Question 292: [**GATE CS Mock**]

Convert the following ambiguous grammar into unambiguous grammar-

$R \rightarrow R + R / R . R / R^* / a / b$

where * is kleen closure and . is concatenation.

Answer:

To convert the given grammar into its corresponding unambiguous grammar, we implement the precedence and associativity constraints.

We have-

- Given grammar consists of the following operators- **+** , **.** , ***
- Given grammar consists of the following operands- **a** , **b**

The priority order is- (**a** , **b**) > * > **.** > **+**

Where-

- **.** operator is left associative
- **+** operator is left associative

sing the precedence and associativity rules, we write the corresponding unambiguous grammar as-

$E \rightarrow E + T / T$
$T \rightarrow T . F / F$
$F \rightarrow F^* / G$
$G \rightarrow a / b$

OR

$E \rightarrow E + T / T$
$T \rightarrow T . F / F$
$F \rightarrow F^* / a / b$

Question 293: [**GATE CS Mock**]

Convert the following ambiguous grammar into unambiguous grammar-

bexp → bexp or bexp / bexp and bexp / not bexp / T / F

where bexp represents Boolean expression, T represents True and F represents False.

Answer:

To convert the given grammar into its corresponding unambiguous grammar, we implement the precedence and associativity constraints.

We have- Given grammar consists of the following operators- **or** , **and** , **not**

- Given grammar consists of the following operands- **T** , **F**

The priority order is- (**T** , **F**) > **not** > **and** > **or**

where-

- **and** operator is left associative
- **or** operator is left associative
- Using the precedence and associativity rules, we write the corresponding unambiguous grammar as-

bexp → bexp or M / M
M → M and N / N
N → not N / G
G → T / F

(OR)

bexp → bexp or M / M

M → M and N / N

N → not N / T / F

Unambiguous Grammar

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 294: [**GATE CS Mock**]

Consider the given grammar-

E → E + T / T

T → F x T / F

F → id Evaluate the following expression in accordance with the given grammar 2+3x5x6+2

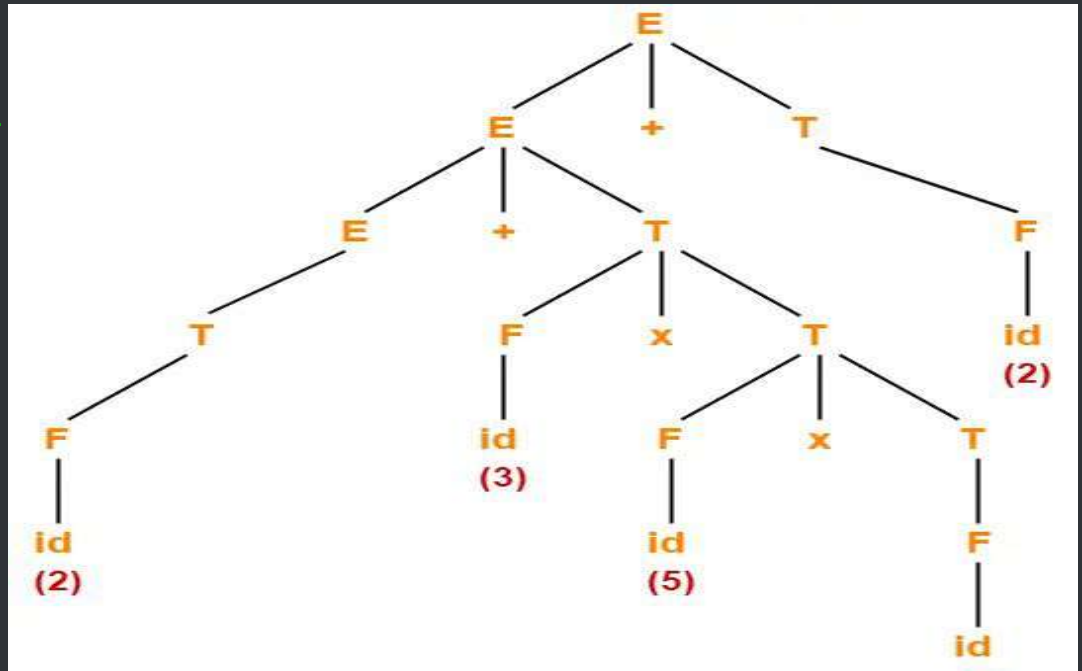
Answer:

Method-01:

- Let us draw a parse tree for the given expression.
- Evaluating the parse tree will return

the required value of the expression.

The parse tree for the given expression is-



[Conti.....]

Method-02:

The priority order and associativity of operators on the basis of given grammar is-

id > x > +

where-

- x operator is right associative.
- + operator is left associative.

Now, we parenthesize the given expression based on the precedence and associativity of operators as-

(2 + (3 x (5 x 6))) + 2

Now, we evaluate the parenthesized expression as-

= (2 + (3 x **(5 x 6)**)) + 2

= (2 + **(3 x 30)**) + 2

= **(2 + 90)** + 2

= **92 + 2**

= 94

Question 295: [**GATE CS Mock**]

Consider the following grammar-

E → E – E

E → E x E

E → id

Parse the input string id – id x id using a shift-reduce parser and see whether the given string accepted or not?

Answer: The priority order is: id > x > –

Stack	Input Buffer	Parsing Action
$	id – id x id $	Shift
$ id	– id x id $	Reduce E → id
$ E	– id x id $	Shift
$ E –	id x id $	Shift
$ E – id	x id $	Reduce E → id
$ E – E	x id $	Shift
$ E – E x	id $	Shift
$ E – E xid	$	Reduce E → id
$ E – E x E	$	Reduce E → E x E
$ E – E	$	Reduce E → E – E
$ E $		Accept

Question 296: **[GATE CS Mock]**

Consider the following grammar-

S → (L) | a

L → L , S | S

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

Answer:

Stack	Input Buffer	Parsing Action
$	(a , (a , a)) $	Shift
$ (	a , (a , a)) $	Shift
$ (a	, (a , a)) $	Reduce S → a
$ (S	, (a , a)) $	Reduce L → S
$ (L	, (a , a)) $	Shift
$ (L ,	(a , a)) $	Shift
$ (L , (a , a)) $	Shift	
$ (L , (a	, a)) $	Reduce S → a
$ (L , (S	, a)) $	Reduce L → S
$ (L , (L	, a)) $	Shift
$ (L , (L ,	a)) $	Shift
$ (L , (L , a	)) $	Reduce S → a
$ (L , (L , S)	)) $	Reduce L → L , S
$ (L , (L	)) $	Shift
$ (L , (L)	) $	Reduce S → (L)
$ (L , S	) $	Reduce L → L , S
$ (L)	$	Shift
$ (L)	$	Reduce S → (L)
$ S	$	Accept

Question 297: **[GATE CS Mock]**

Consider the following grammar-

S → (L) | a

L → L , S | S

Construct the operator precedence parser and parse the string

(a , (a , a)).

Answer: The terminal symbols in the grammar are { (,) , a , , }

We construct the operator precedence table as-

	a	(	)	,	$
a		>	>	>	>
(	<	>	>	>	>
)	<	>	>	>	>
,	<	<	>	>	>
$	<	<	<	<	

Parsing Given String-

Given string to be parsed is **(a , (a , a))**.

We follow the following steps to parse the given string-

Step-01:

We insert $ symbol at both ends of the string as-**$ (a , (a , a)) $**

We insert precedence operators between the string symbols as-

$ < (< a > , < (< a > , < a >) >) > $

Step-02: We scan and parse the string as-

$ < (**< a >** , < (< a > , < a >) >) > $

$ < (S , < (**< a >** , < a >) >) > $

$ < (S , < (S , **< a >**) >) > $

$ < (S , **< (S , S) >**) > $

$ < (S , **< (L , S) >**) > $

$ < (S , **< (L) >**) > $

$ **< (S , S) >** $

$ **< (L , S) >** $

$ **< (L) >** $

$ **< S >** $

$ $

Question 298: [**GATE CS Mock**]

Consider the following grammar-

E → E + E | E x E | id

1. Construct Operator Precedence Parser.
2. Find the Operator Precedence Functions.

Answer: The terminal symbols in the grammar are { + , x , id , $ }

We construct the operator precedence table as-

f ↓ \ g →	id	+	x	$
id		>	>	>
+	<	>	<	>
x	<	>	>	>
$	<	<	<	

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

[Conti....]

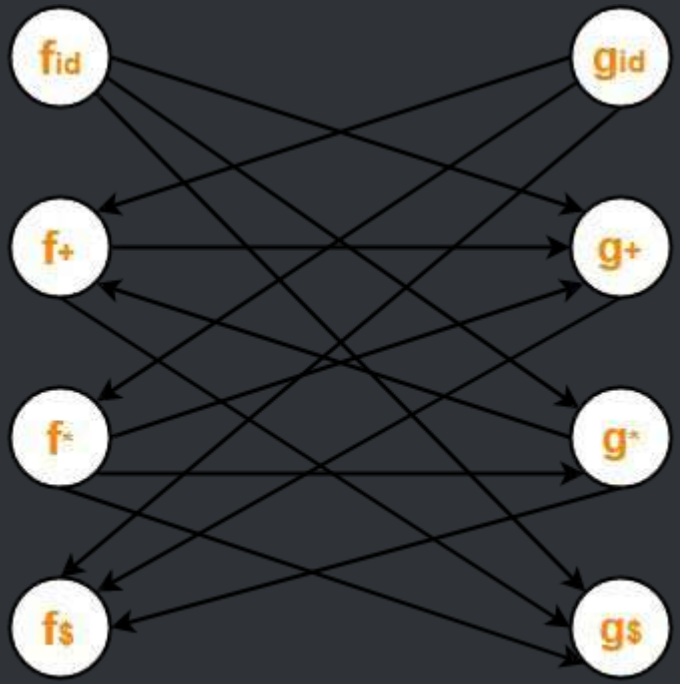
The graph representing the precedence functions is-

Here, the longest paths are-

- $f_{id} \rightarrow g_x \rightarrow f_+ \rightarrow g_+ \rightarrow f_\$$
- $g_{id} \rightarrow f_x \rightarrow g_x \rightarrow f_+ \rightarrow g_+ \rightarrow f_\$$

The resulting precedence functions are-

	+	x	id	$
f	2	4	4	0
g	1	3	5	0



Graph Representing Precedence Functions

Question 299: [**GATE CS Mock**]

Find the type of error produced by the following c code:

```
main()
{
        in/*comment t c;
        float/*comment*/t gate;
}
```

a. Lexical error
b. syntax error
c. both a) and b)
d. None of these

Answer: D

Explanation:

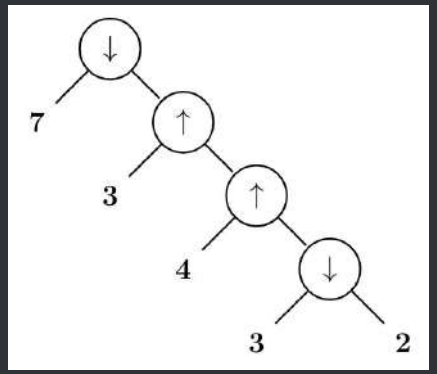
There is no lexical issue as the tokens are identified here and all of them follow the naming convention of tokens here ..There is no such violation like :

a) There is some token which starts with the capital letter
b) There is some token which starts with some special character(except underscore) etc.

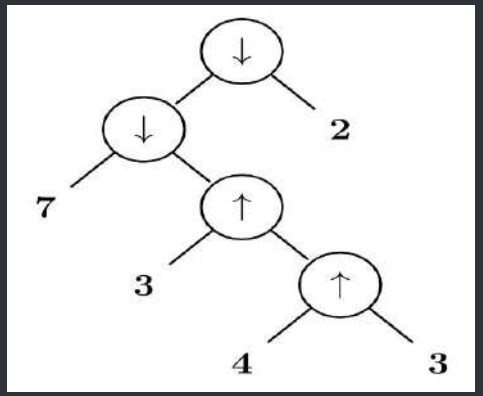
Hence no lexical error here..
And as far as the comment sign is concerned so it expects the closing comment sign also which is there in the given code..And after the token "gate" , we have the semicolon also.
So no syntax error also.
But there is a ***semantic error*** as we have "in" , so compiler treats it as an identifier which is undeclared..
Hence D) is correct answer..

Question 300: **[GATE CS Mock]**
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)?

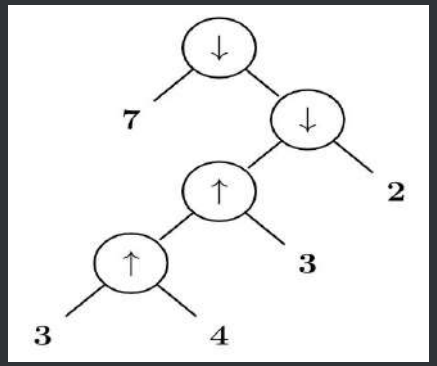
A)



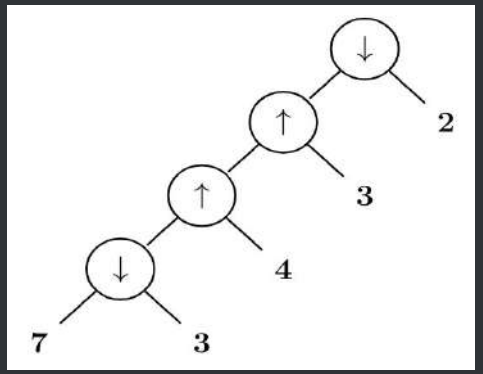
B)



C)



D)



[Conti.....]

Answer: B

Explanation: To make the parse tree start compiling the identifiers into blocks based on associativity and precedence.
Grouping: (7↓(3↑(4↑3)))↓2

Tree can be made by opening inner braces and move towards braces.

Question 301: **[GATE CS Mock]**
Consider the following intermediate program in three address code

```
p = a - b
q = p * c
p = u * v
q = p + q
```

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

A)
```
p1 = a  - b
q1 = p1 * c
p1 = u  * v
q1 = p1 + q1
```

B)
```
p3 = a  - b
q4 = p3 * c
p4 = u  * v
q5 = p4 + q4
```

C)
```
p1 = a  - b
q1 = p2 * c
p3 = u  * v
q2 = p4 + q3
```

D)
```
p1 = a - b
q1 = p * c
p2 = u * v
q2 = p + q
```

Answer: B
Explanation: In option A – the 3rd expression is overridden when again p1 is used.
In option C – the 2nd expression is wrong as p2 is not initialized before but used.

In option D – the 2nd expression is wrong as p is not initialized before but used.

Here each and every variable used are distinct. So the answer is (B).

Question 302: [GATE CS Mock]
Consider the following code segment

```
a=b+c
k=a-d
a=k+b
u=v+w
u=a-u
```

The minimum number of total variables required to convert the above code segment to static single assignment form is

Answer: 11
Explanation:
The static single assignment form is:

```
t1=b+c
t2=t1-d
t3=t2+b
t4=v+w
t5=t3-u
```

Question 303: [**GATE CS Mock**]

Consider the following intermediate code segment in 3-address code form :
x=x*x
y=y*y
x=x*y
y=y*x
The minimum number of total variables in the static single assignment form of the given code segment is _____

A) 5 **B) 4** **C)6** **D)8**

Answer: 6
Explanation:
The SSA from of the given code segment is :

$x_1=x_0*x_0$
$y_1=y_0*y_0$
$x_2=x_1*y_1$
$y_2=y_1*x_2$

So, total 6 variable(x_0,y_0,x_1,x_2,y_1,y_2).

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.

Question 304: **[GATE CS Mock]**
Number of temporary variable required to create 3 address code in static single assignment form for the expression:
$P+Q*R-S/(Q*R)$

A) 6 B)5 C)4 D)3

Answer: **C**
Explanation:
$P+Q*R-S/(Q*R)$
$T_1=Q*R$

$T_2=P+T_1$

$T_3=S/T_1$

$T_4=T_2-T_3$

So, total 8 variables
$[P,Q,R,S,T_1,T_2,T_3,T_4]$.

Here, 4 temporary variables.

Question 305: **[GATE CS Mock]**
Consider the following expressions which calculates one solution of the quadratic equation:
$ax_2+bx+c=0$

$x=(-b+sqrt(b_2-4\times a\times c))/(2\times a)$ where sqrt is an unary operator which performs the square root.The least number of temporary variables required to create a three address code in static single assignment form for the above expression is _____?

Answer: 8

Explanation: The correct option is A 8

We will need a temporary variable for storing the result

(Static Single Assignment) implies the variable cannot

$t_1=b*b$

$t_2=4*a$

$t_3=t_2*c$

$t_4=t_1*t_3$

$t_5=sqrt(t_4)$

$t_6=t_5*b$

$t_7=2*a$

$t_8=t_7/t_8$

$x=t_8$

Here temporary variable = 8

Question 306: **[GATE CS Mock]**
Consider the following 'C' expression

x = (a + b) * (a + b+ c) * a + b;

How many temporary variables (in minimum) are required to generate the equivalent 3-address code for the above expression?

Answer: 1
Explanation:
x = (a+ b) * (a + b + c)* a + b
t_1=a+b
c=t_1+c
t_1=t_1*c
t_1=t_1*a
t_1=t_1+b
x=t_1
Only 1 temporary variable is needed.

Download Exampreptool (EPT App)

https://play.google.com/store/apps/details?id=com.ept.requestteacher

For important Questions & Answers with video solutions

(Prepared by rankers: Divyanshu Hymavati Ratul & Soumya)

Note: If any doubt in any question post with subject & question number in EPT App.