

**SRM Institute of Science and Technology SET-C**

**College of Engineering and Technology**

**School of Computing**

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamil Nadu

**Academic Year: 2024 (EVEN)**

**Test: CLA-T2** **Date: 26/3/2024**

**Course Code & Title: 18CSC304J & Compiler Design Duration: 12.30 PM to 2.15 PM (2 periods)**

**Year & Sem: III Year /VI Sem** **Max. Marks: 50**

| **S.No.** | **Course Outcome** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** | **PSO3** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **CO1** | **3** | **2** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **3** |
| **2** | **CO2** | **-** | **3** | **3** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **1** |
| **3** | **CO3** | **-** | **3** | **3** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **1** |
| **4** | **CO4** | **-** | **3** | **3** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **2** |
| **5** | **CO5** | **-** | **3** | **3** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **-** | **3** |

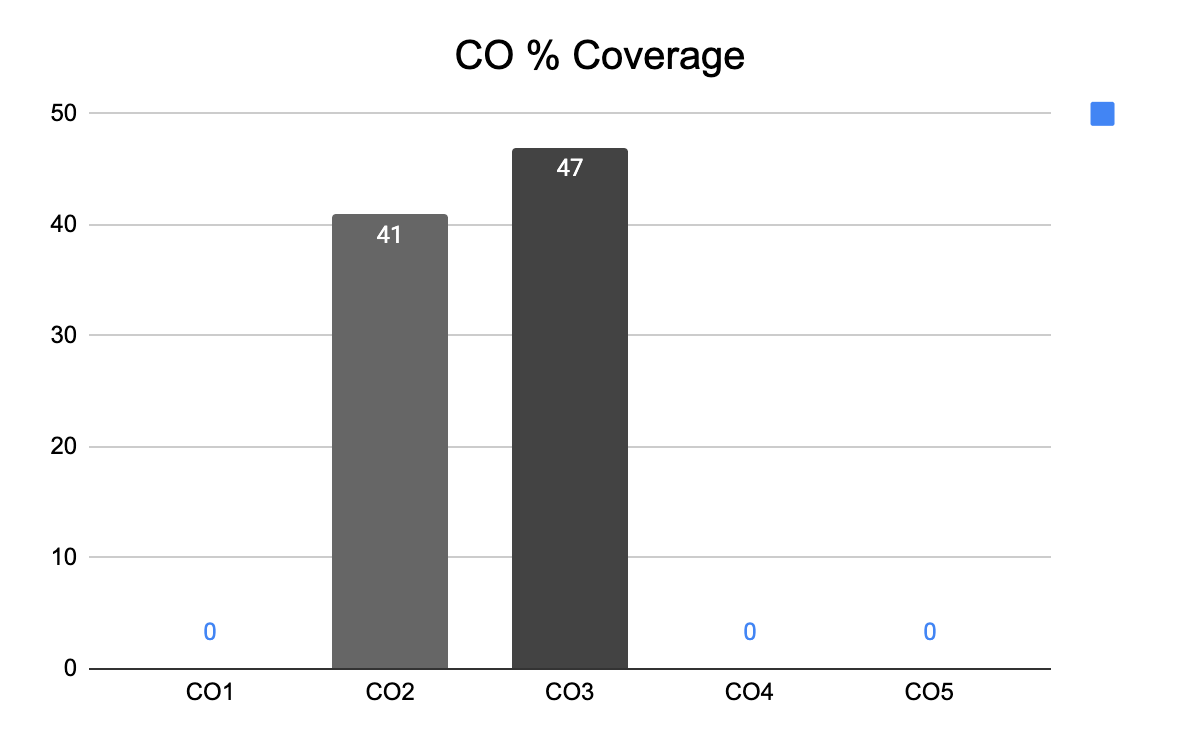
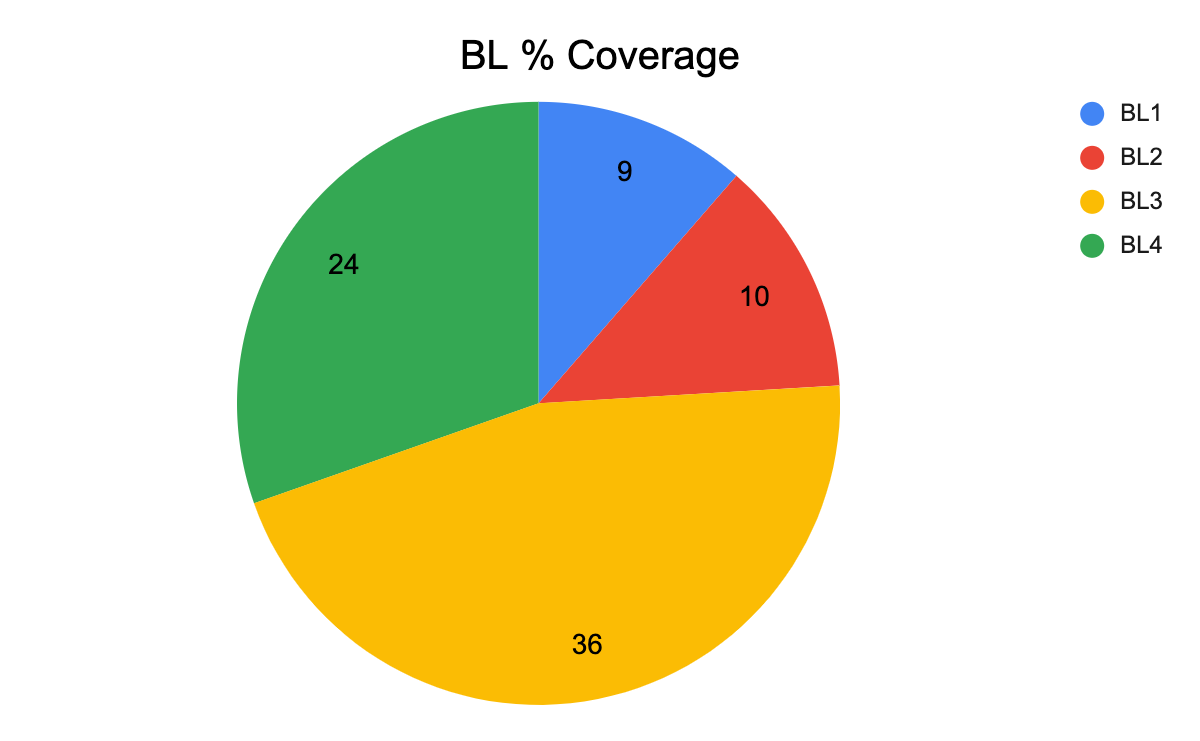
| **Part – A Answer ALL Questions**  **Answer all (10 x 1= 10 marks)** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Q. No** | **Questions** | **Mark** | **BL** | **CO** | **PO** | **PI Code** |
| 1 | In which parsing, the parser constructs the parse tree from the start symbol and transforms it into the input String?  a)Bottom Up Parsing  **b)Top-down parsing**  c)None of the above  d)Both a and b | 1 | 2 | 2 | 3 | 3.6.1 |
| 2 | What is the grammar for the below equations?  S → C C  C → c C | d  **a) LL(1)** b) SLR(1) but not LL(1) c) LALR(1) but not SLR(1) d) LR(1) but not LALR(1) | 1 | 2 | 2 | 3 | 3.6.1 |
| 3 | Grammar that produces more than one leftmost or rightmost parse tree for same sentence is \_\_\_\_\_\_\_\_\_\_\_ **a) Ambiguous** b) Unambiguous c) Complementation  d) Concatenation Intersection | 1 | 2 | 2 | 3 | 3.6.1 |
| 4 | Which phenomenon happens when the non-terminal on the left side is repeated as the first symbol on the right side?  a)Left-most derivation  **b)Left recursion**  c)Left factoring  d)Left parsing | 1 | 1 | 2 | 2 | 2.5.1 |
| 5 | Which part of the compiler highly used the grammar concept?  a)Code optimization  b)Code generation  **c)Parser**  d)Lexical Analysis | 1 | 1 | 2 | 1 | 1.6.1 |
| 6 | Leaf nodes in a parse tree indicate  a)sub-terminals  b)half-terminals  c)non-terminals  **d)terminals** | 1 | 2 | 3 | 1 | 1.6.1 |
| 7 | Which of the following function is used to calculate the canonical collection of LR(0) items?.  a)FIRST  **b)GOTO**  c)COMPUTE  d)FOLLOW | 1 | 2 | 3 | 1 | 1.6.1 |
| 8 | Which of the following grammar rules violate the requirements of an operator grammar?  (i) P -> QR  (ii) P -> QsR  (iii) P -> ε  (iV) P -> QtRr  a) (i) only **b) (i) and (iii) only** c) (ii) and (iii) only d) (iii) and (iv) only | 1 | 1 | 3 | 1 | 1.6.1 |
| 9 | 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 \* | 1 | 1 | 3 | 3 | 3.6.1 |
| 10 | The parser which has higher number of states is  (a) SLR  **(b) CLR**  (c ) LALR  ( d) No relation can be referred to. | 1 | 1 | 3 | 2 | 2.5.1 |

| **Part –B**  **Answer any Four (4x 4=16 marks)** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Q. No** | **Question** | **Mark** | **BL** | **CO** | **PO** | **PI Code** |
| 11 | List the rules for computing FIRST and FOLLOW with an example. | 4 | 1 | 2 | 1 | 1.6.1 |
| 12 | Eliminate left recursion in the following grammar  A → ABd | Aa | a  B → Be | b  A → ABd | Aa | a  B → Be | b  Answer:  The grammar after eliminating left recursion is-  A → aA’  A’ → BdA’ / aA’ / ∈  B → bB’  B’ → eB’ / ∈ | 4 | 3 | 2 | 2 | **2.5.2** |
| 13 | Mention error recovery strategies for predictive parsing.  Answer:  The error may occur at various levels of compilation, so error handling is important for the correct execution of code. There are mainly five error recovery strategies, which are as follows:   1. Panic mode 2. Phrase level recovery 3. Error production 4. Global correction 5. Symbol table  Panic Mode: This strategy is used by most parsing methods. In this method of discovering the error, the parser **discards input symbols one at a time.**This process is continued until one of the designated sets of synchronizing tokens is found. Synchronizing tokens are delimiters such as **semicolons or ends.** These tokens indicate an end of the input statement.  Thus, in panic mode recovery a considerable amount of input checking is for additional errors. If there is less number of errors in the same statement, then this strategy is best choice. Phrase Level Recovery: In this strategy, on discovering an error, parser performs local correction on the remaining input. It can replace a prefix of the remaining input with some string. This actually helps the parser to continue its job. The local correction can be replacing the comma with semicolons, omission of semicolons, or, fitting missing semicolons. This type of local correction is decided by the compiler developer. Error Production: It requires good knowledge of common errors that might get encountered, then we can augment the grammar for the corresponding language with **error productions that generate the erroneous constructs.**If error production is used during parsing, we can generate an appropriate error message to indicate the error that has been recognized in the input. This method is extremely difficult to maintain, because if we change grammar, then it becomes necessary to change the corresponding productions. Global Correction: We often want such a compiler that makes very few changes in processing an incorrect input string to the correct input string. Given an incorrect input string x and grammar G**,**the algorithm itself can find a parse tree for a related string y (Expected output string); such that a number of insertions, deletions, and changes of token require to transform x into y is as low as possible. Global correction methods increase time & space requirements at parsing time. This is simply a theoretical concept.  Symbol Table:  In semantic errors, errors are recovered by using a symbol table for the corresponding identifier and if data types of two operands are not compatible, automatically [type conversion](https://www.geeksforgeeks.org/type-conversion-c/) is done by the compiler. | 4 | 2 | 2 | 2 | **2.5.1** |
| 14 | Construct the Precedence Relation table for the Grammar.  E → E + E | E ∗ E/id  Answer:  The terminal symbols in the grammar are { id, + , x , $ }  We construct the operator precedence table as- | 4 | 3 | 3 | 2 | **2.5.2** |
| 15 | Explain the types of conflicts occurring during Shift Reduce Parser.   1. **Shift/Reduce Conflict**: This type of conflict occurs when the parser has the option to either shift the next input symbol onto the stack or to reduce the current stack contents using a production rule. Shift/reduce conflicts arise because the parser cannot determine whether to shift or reduce based solely on the current input symbol and the symbols on the stack. This ambiguity typically arises due to the grammar being ambiguous, meaning that the same input sequence can be derived by different parse trees. 2. **Reduce/Reduce Conflict**: This conflict occurs when the parser can choose between two or more production rules to reduce the current stack contents. Like shift/reduce conflicts, reduce/reduce conflicts arise from ambiguity in the grammar, where multiple production rules can be applied at the same point in the parsing process. | 4 | 3 | 3 | 2 | **2.5.1** |

| **Part –C**  **Answer all ( 2 x 12=24 marks)** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Q. No** | **Question** | **Marks** | **BL** | **CO** | **PO** | **PI Code** |
| 16 | (i) Write an algorithm for recursive descent parsing with an example.  (ii) Compute FIRST and FOLLOW for the following productions  E-> E+T|T  T->T\*F|F  F->(E)|id  A basic algorithm for recursive descent parsing:   1. Define parsing functions for each non-terminal symbol in the grammar. 2. Start parsing from the start symbol of the grammar. 3. For each parsing function:    * Match the current input token with the expected token(s) for the corresponding non-terminal.    * If the token matches, consume it and proceed.    * If the token does not match, report an error.    * If the non-terminal has production rules, recursively call the parsing functions for the symbols on the right-hand side of the production rules. 4. Continue this process until all input tokens are consumed and the parsing is successful, or until an error is encountered.   eliminating left recursion and left factoring from grammar, the resulting grammar will be a grammar that can be parsed by a recursive desent parser.  (ii)   | **Before removing left recursion** | **After removing left recursion** | | --- | --- | | E –> E + T | T  T –> T \* F | F  F –> ( E ) | id | E –> T E’  E’ –> + T E’ | e  T –> F T’  T’ –> \* F T’ | e  F –> ( E ) | id |   (ii)Compute FIRST and FOLLOW for the following productions  E-> E+T|T  T->T\*F|F  F->(E)|id  Answer:  After eliminating left recursion the grammar is  E → TE'  E' → +TE' | ε  T → FT '  T'→ \*FT ' | ε  F → (E) | id  FIRST( ) :  FIRST(E) = { ( , id}  FIRST(E’) ={+ , ε }  FIRST(T) = { ( , id}  FIRST(T’) = {\*, ε }  FIRST(F) = { ( , id }  FOLLOW( ):  FOLLOW(E) = { $, ) }  FOLLOW(E’) = { $, ) }  FOLLOW(T) = { +, $, ) }  FOLLOW(T’) = { +, $, ) }  FOLLOW(F) = {+, \* , $ , ) } | 6+6 | 3 | 2 | 2 | **2.5.1** |
|  | **Or** |  |  |  |  |  |
| 17 | Construct a predictive parsing table for the following grammar  S -> abSa|aaAb  A -> baAb|b  and Check whether it is LL(1) or not.  **Step 1: First and Follow Sets**  First, let’s compute the **First** and **Follow** sets for each non-terminal symbol:   1. **First Sets**:    * (First(S)):      + (First(S) = {a})    * (First(A)):      + (First(A) = {b}) 2. **Follow Sets**:    * (Follow(S) = {$}) (where ($) represents the end-of-input marker)    * (Follow(A) = {a, b})   **Step 2: Constructing the Predictive Parsing Table**  The predictive parsing table is a matrix where rows correspond to non-terminals, columns correspond to terminals, and each cell contains the production rule to apply. Let’s create the table:  **Table**   | **Non-Terminal** | **Terminal (a)** | **Terminal (b)** | **Terminal ($)** | | --- | --- | --- | --- | | (S) | (abSa) |  |  | | (A) |  | (baAb) | (b) |   **Step 3: Checking for LL(1) Property**  For a grammar to be LL(1), the following conditions must hold:   1. No two distinct productions for the same non-terminal can have the same terminal as their first symbol. 2. If a production (A \to \alpha) has a terminal (a) in its first set, then no other production for (A) can have (a) in its first set.   Let’s verify these conditions:   * For (S):   + (First(abSa)) and (First(aaAb)) both contain (a), but they are distinct productions. So, the first condition is satisfied.   + No conflict in (S)'s first sets. * For (A):   + No conflict in (A)'s first sets.   Since there are no conflicts, the grammar is **LL(1)**. | 12 | 4 | 2 | 2 | **2.8.4** |
| 18 | Show SLR parsing table for the following grammar  S → Aa | bAc | Bc | bBa  A → d  B → d  and check whether the sentences "bdc" and "dd” are accepted or not.  Augmented Grammar: S' -> S S -> Aa | bAc | Bc | bBa A -> d B -> d  Closure of Items:  I0: {S' -> .S} Closure(I0): {S' -> .S, S -> .Aa, S -> .bAc, S -> .Bc, S -> .bBa, A -> .d, B -> .d}  I1: {S -> A.a, S -> b.Ac} Closure(I1): {S -> A.a, S -> b.Ac, A -> .d}  I2: {S -> bA.c} Closure(I2): {S -> bA.c, B -> .d} I3: {S -> bB.a} Closure(I3): {S -> bB.a, B -> .d}  I4: {S -> bAc.}  I5: {S -> B.c}  I6: {S -> bBa.}  I7: {S -> Aa.}  I8: {S' -> S.}  Transitions:   * + Transition from I0:     - On 'A': Go to I1     - On 'B': Go to I2     - On 'a': Go to I7     - On 'b': Go to I3     - On 'c': Go to I5   + Transition from I1:     - On 'd': Go to I4   + Transition from I2:     - On 'd': Go to I4   + Transition from I3:     - On 'd': Go to I4   + Transition from I4:     - No transitions (accept state)   + Transition from I5:     - No transitions (accept state)   + Transition from I6:     - No transitions (accept state)   + Transition from I7:     - No transitions (accept state)   + Transition from I8:     - No transitions (accept state)   SLR Parsing Table:   |  | a | b | c | d | A | B | | --- | --- | --- | --- | --- | --- | --- | | 0 |  | S3 |  | S6 | 1 | 2 | | 1 |  |  |  |  |  |  | | 2 |  |  |  |  |  |  | | 3 | S7 |  |  |  |  |  | | 4 |  |  |  |  |  |  | | 5 |  |  |  |  |  |  | | 6 |  |  |  |  |  |  | | 7 |  |  |  |  |  |  | | 8 |  |  |  |  |  |  |   Parsing the sentence "bdc":  Stack Input Action  0 bdc$ Shift (goto S3)  0b3 dc$ Reduce using S -> bAc  0S3 dc$ Shift (goto S5)  0S5d4 c$ Reduce using S -> Bc  0S5S4 c$ Shift (goto S7)  0S5S7c4 $ Accept  Parsing the sentence "dd":  Stack Input Action  0 dd$ Shift (goto S6)  0d6 d$ Reduce using B -> d  0S6 d$ Shift (goto S6)  0S6d4 $ Reduce using S -> Bc  0S6S4 $ Reduce using S -> bBa  0S6S $ Reduce using S -> bAc  0S $ Accept | 12 | 3 | 3 | 2 | **2.5.2** |
|  | **Or** |  |  |  |  |  |
| 19 | Consider the following grammar, and construct the operator precedence parsing table and check whether the input string (i) \*id=id (ii)id\*id=id are successfully parsed or not?  S→L=R  S→R  L→\*R  L→id  R→L  Answer:  1.Computation of LEADING:  LEADING(S) = {=, \* , id} LEADING(L) = {\* , id} LEADING(R) = {\* , id} 2.Computation of TRAILING:  TRAILING(S) = {= , \* , id}  TRAILING(L)= {\* , id} TRAILING(R)= {\* , id} | 12 | 4 | 3 | 2 | **2.8.4** |

**\*Performance Indicators are available separately for Computer Science and Engineering in AICTE examination reforms policy.**

**Course Outcome (CO) and Bloom’s level (BL) Coverage in Questions**

****

**Approved by the Audit Professor/Course Coordinator**