R/K/TY

Reg. No:

VIT®
Vellore Institute of Technology

Final Assessment Test – November 2025

Course: BCSE307L - Compiler Design

Class NBR(s): 1574/1577/1580/1585/1588/1591/1593/ 1600/1606/1609/1611/1613/1616/1621/1628/1633/ 1639/2829/6947

Slot: A2+TA2

Time: Three Hours

Max. Marks: 100

- KEEPING MOBILE PHONE/ANY ELECTRONIC GADGETS, EVEN IN 'OFF' POSITION IS TREATED AS EXAM MALPRACTICE
- DON'T WRITE ANYTHING ON THE QUESTION PAPER

COs	CO Statements
CO1	Apply the skills on devising, selecting, and using tools and techniques towards compiler design.
CO2	Develop language specifications using context free grammars (CFG).
CO3	Apply the ideas, the techniques, and the knowledge acquired for the purpose of developing software systems.
CO4	Constructing symbol tables and generating intermediate code.
CO5	Obtain insights on compiler optimization and code generation.

BL – Blooms Taxonomy Level (1 – Remember, 2 – Understand, 3 – Apply, 4 – Analyse, 5 – Evaluate, 6 – Create)

Answer ALL Questions

(10 X 10 = 100 Marks)

1. Construct the syntax tree corresponding to the augmented regular expression: *(a*b(ba*b + ab*a)*)#*. Decorate each node with firstpos and lastpos data. Compute followpos for different positions and draw the corresponding Deterministic Finite Automaton. **CO1 BL3**

2. a) Trace the following code snippets through the phases of a compiler and show the output of each phase. **[7] CO1 BL3**

```
fact = 1;
for(i = 1; i <= n; i++)
  fact = fact * i;
return fact;
```

b) Consider the following C code and answer the following: **[3]**

i. Identify all the tokens in this program fragment.

ii. Count the total number of tokens.

```
int sum = 0;
for (int i = 0; i < n; i++)
{
  sum += arr[i];
  arr[i] = arr[i] * 2;
}
if (sum > 100)
  printf("Large sum\n");
else
printf("Small sum\n");
```

3. For given grammar G, construct a LL Parser, Then, parse the input string: *(a + b) * c* and generate the parse tree, showing the expansions of all non-terminals step by step. CO2

G:

expr → term expr_tail

expr_tail → + term expr_tail | ε

term → factor term_tail

term_tail → * factor term_tail | ε

factor → (expr) | a | b | c

4. (i) Consider the following grammar where "id" is a token that represents an integer and "id" value is its integer value. Write appropriate translation scheme for constructing a tree to show productions used and where semantic actions occur and perform post order evaluation for the input string: "2*3+4". [7] CO3 BL3

S → E R

R → * E R | ε

E → F + E | F

F → (S) | id

(ii) Differentiate between synthesized and inherited attributes with proper examples. [3]

5. Given the nested *for* loops that perform adjacent swaps, convert the fragment into three-address code (TAC), identify leaders and partition the TAC into basic blocks (label them), and draw the control-flow graph (CFG) showing all edges and block successors. CO5 BL4

```
FOR I := 1 TO n - 1 DO
  FOR J := 1 TO I DO
IF A[J] > A[J + 1] THEN
BEGIN
   Temp := A[J];
   A[J] := A[J + 1];
   A[J + 1] := Temp;
END
```

CO2

6. Analyse the following C program and Identify all possible code optimization techniques that can be applied in this program, explain where and why each technique can be used, and finally write the fully optimized version of the code — CO5 BL4

```
int computeSum(int arr[], int n) {
  int sum = 0;
  for (int i = 0; i < n; i++)
    sum = sum + arr[i];
  return sum;
}
int main() {
  int a = 10, b = 20, c = 30;
  int x, y, z, arr[5] = {1, 2, 3, 4, 5};
  x = a + b + 5;
  y = a + b + 5;
  z = c * 2;
  int unusedVar = 100, temp = x + y;
  for (int i = 0; i < 5; i++) {
    arr[i] = arr[i] + c * 2;
    x = x + i * 4;
  }
  int p = x;
  if (p > 50)
    y = y + p;
  else
    y = y - p;
int total = computeSum(arr, 5);
  printf("x=%d, y=%d, z=%d, total=%d\n", x, y, z, total);
  return 0;
}
```

7. Explain the various issues encountered during the code generation phase of a compiler. Illustrate each issue with a suitable example to show how it affects the generation of efficient and correct assembly or machine code. — CO5 BL2

8. Explain the various approaches used for register allocation and register assignment in compiler optimization. Compare their strategies and evaluate how each approach impacts the efficiency and performance of the generated code. CO3 B

9.a) Convert the following code fragment into its equivalent three address code (TAC) and then represent the TAC using both triples and indirect triples formats. Briefly discuss the advantages of using triples and indirect triples as intermediate code representations. CO4 BL3

```
if (c[i] != 0)
  a[i] = a[i] + b[i] / c[i];
else
  b[i] = b[i];
```

OR

9.b) Write syntax-directed translation (SDT) scheme that generates three-address code (TAC) using backpatching for flow control constructs, and then apply your SDT to translate the given program fragment into TAC. And draws suitably annotate the parse tree with relevant attributes and clearly illustrate all backpatching steps involved. CO4 BL3

```
while a < b do
  if c < d then
    X = y + z;
  else
    X = y - z;
```

10.a) Consider the following grammar: CO2 BL3

S → P F class I X Y
P → public | ε
F → final | ε
X → extends I | ε
Y → implements I J | ε
I → identifier
J → , I J | ε

Construct the SLR(1) parsing table for the given grammar and determine whether the grammar is SLR(1) or not. If it is not SLR(1), clearly state the reason by identifying any shift-reduce or reduce-reduce conflicts that occur.

OR

10.b) (i) Construct the operator precedence relation table for all terminals in the grammar G. Clearly indicate the precedence relations and justify your assignments based on operator precedence and grammar rules. **[5] CO2 BL3**

G:

$S \rightarrow x = E$

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid x \mid y \mid 0 \mid 1$

(ii) Using the table you constructed, check whether the input string $x = y + 1 * x$ is valid according to operator precedence parsing. Show the step-by-step parsing stack operations. **[5]**

⇔⇔⇔ **R/K/TY** ⇔⇔⇔