REG.NO.:

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2024-2025

SLOT: C1+TC1

Programme Name & Branch : B.Tech CSE
Course Code and Course Name :BCSE307L Compiler Design
Faculty Name(s) : PROF. SAHAAYA ARUL MARY S A, PROF. KANNADASAN R, PROF. VISHNUPRIYA, PROF. VETRISELVI T, PROF. BHUVANESWARI M, PROF. KANAGARAJ R, PROF. KALAIVANI K, PROF. SATHYA K, PROF. BASKARAN P, PROF. SABYASACHI KAMILA, PROF. UMA PRIYA D, PROF. MUKKU NISANTH KARTHEEK, PROF. ARUMUGA ARUN R, PROF. ISLABUDEEN M, PROF. SUGANTHINI C, PROF. NAGA PRIYADARSINI R, PROF. BHAWANA TYAGI, PROF. DEBI PRASANNA ACHARJYA, PROF. BAIJU B V, PROF. UMAMAHESWARI M
Class Number(s) : VL2024250101542, VL2024250101548, VL2024250101555, VL2024250101587, VL2024250101605, VL2024250101612, VL2024250101623, VL2024250101633, VL2024250101641, VL2024250101651, VL2024250101660, VL2024250101669, VL2024250101673, VL2024250101676, VL2024250101684, VL2024250101725, VL2024250101734, VL2024250101740, VL2024250101746, VL2024250107999
Date of Examination : 15-Oct-2024
Exam Duration : 90 minutes **Maximum Marks: 50**

General instruction(s):

Q. No	Question	M	CO	BL
1.	The grammar given below is used to parse all possible expressions in C that involve(only) variables of pointer datatype(ignoring the semi-colon). ***program → expression | assignment*** ***assignment → expression = expression*** ***expression → identifier | *expression*** (Assume that identifier is a terminal symbol) Check whether the above grammar is SLR(1). If yes, then parse the following: ****ptr1 = ptr2***	10		

REG.NO.:

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2024-2025

SLOT: C1+TC1

(Q1) $P \rightarrow E/A$
$A \rightarrow E=E$
$E \rightarrow id/*E$

Augmented Grammar:

I_0
$P' \rightarrow .P$
1. $P \rightarrow .E$
2. $P \rightarrow .A$
3. $A \rightarrow .E=E$
4. $E \rightarrow .id$
5. $E \rightarrow .*E$

(I_0, P) I_1
$P' \rightarrow P.$

(I_0, E) I_2
$P \rightarrow E.$
$A \rightarrow E.=E$

(I_0, A) I_3
$P \rightarrow A.$

(I_0, id) I_4
$E \rightarrow id.$

$(I_0, *)$ I_5
$E \rightarrow *.E$
$E \rightarrow .id$
$E \rightarrow .*E$

$(I_2, =)$ I_6
$A \rightarrow E=.E$
$E \rightarrow .id$
$E \rightarrow .*E$

(I_5, E) I_7
$E \rightarrow *E.$

(I_5, id) I_4
$E \rightarrow id.$

$(I_5, *)$ I_5
$E \rightarrow *.E$
$E \rightarrow .id$
$E \rightarrow .*E$

(I_6, E) I_8
$A \rightarrow E=E.$

(I_6, id) I_4
$E \rightarrow id.$

$(I_6, *)$ I_5
$E \rightarrow *.E$
$E \rightarrow .id$
$E \rightarrow .*E$

Follow(P') = {$}
FOLLOW(P) = {$}
FOLLOW(E) = {$, =}
FOLLOW(A) = {$}

I_1 : $P' \rightarrow P$
I_2 : $P \xrightarrow{1} E \Rightarrow$ {$} R1
I_3 : $P \xrightarrow{2} A \Rightarrow$ {$} R2
I_4 : $E \xrightarrow{4} id \Rightarrow$ {=, $} R4
I_7 : $E \xrightarrow{5} *E \Rightarrow$ {=, $} R5
I_8 : $A \xrightarrow{3} E=E \Rightarrow$ {$} R3

	ACTION				GOTO		
	id	*	=	$	P	E	A
0	S4	S5			1	2	3
1				ACCEPT			
2			S6	R1			
3				R2			
4			R4	R4			
5	S4	S5				7	
6	S4	S5				8	
7			R5	R5			
8				R3			

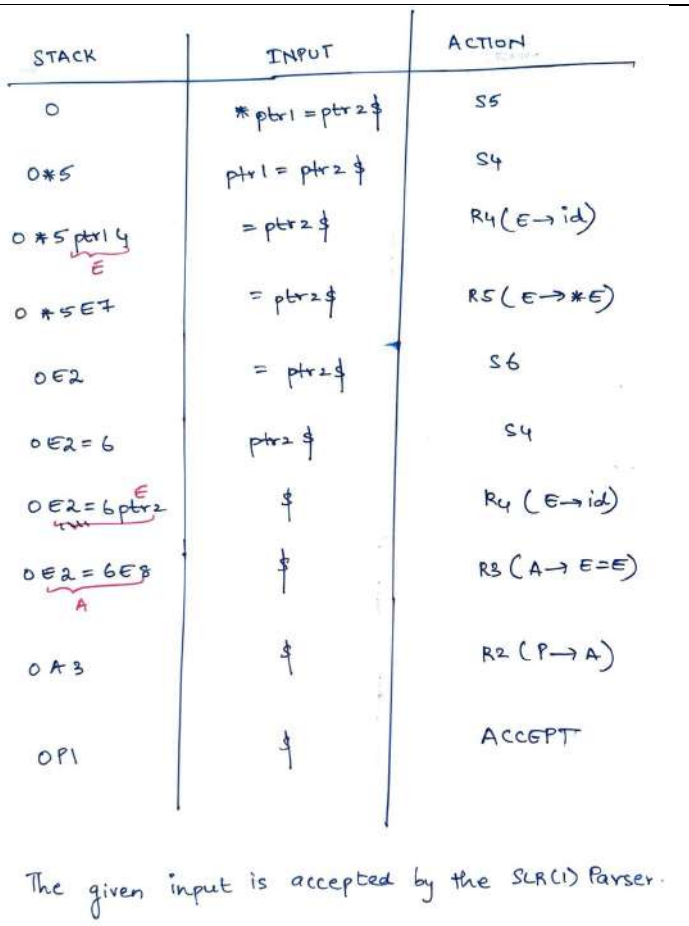
The given grammar is a SLR(1) grammar.

REG.NO.:

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2024-2025

SLOT: C1+TC1

	 The given input is accepted by the SLR(1) Parser.			
2.	Consider the following grammar for **Boolean expressions** and **if-else** statements: ***S → if B then S1 else S2*** ***S → assign id = E*** ***B → E == E | E > E*** ***E → E + T | E – T/T*** ***T → T*F | T/F | F*** ***F → (E) | num*** Write a SDD that evaluates arithmetic expressions(E,T,F). Evaluates Boolean expressions(B) . Given the below input, construct the annotated parse tree showing how your SDD evaluates the Boolean expression and simulates the conditional assignment: ***if (3 + 4) > 5 then assign x = 10 else assign x = 20***	10		

$E \rightarrow E + T \{ E.val = E_1.val + T.val \}$

$E \rightarrow E - T \{ E.val = E_1.val - T.val \}$

$T \rightarrow T * F \{ T.val = T_1.val * F.val \}$

$T \rightarrow T / F \{ T.val = T_1.val / F.val \}$

$F \rightarrow (E) \{ F.val = E.val \}$

$F \rightarrow num \{ F.val = num.lexval \}$

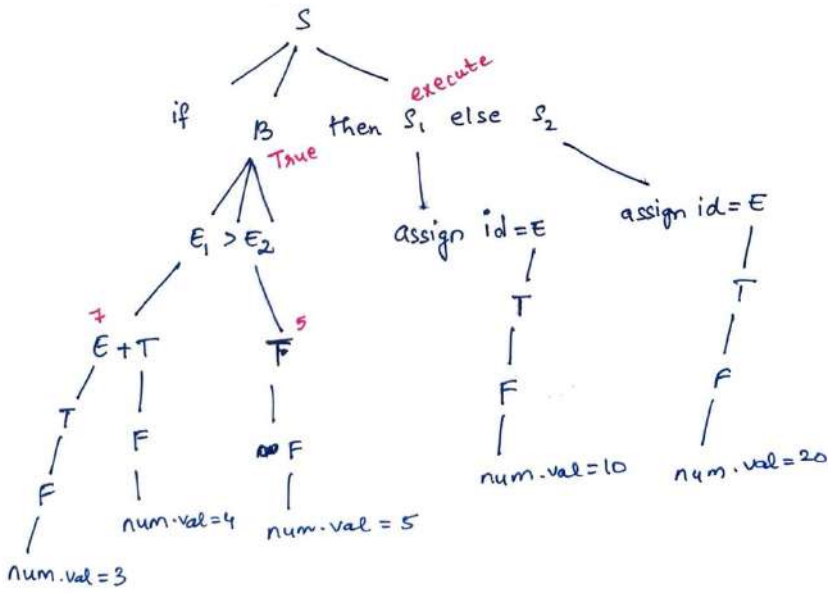
$B \rightarrow E1 == E2 \{ E_1.val == E2.val \}$

$B \rightarrow E1 > E2 \{ E.val > E_2.val \}$

$S \rightarrow$ if B then S1 else S2 :

{ if (B.val == True) then S1.execute
else S2.execute }

$S \rightarrow$ assign id = E (id.val = E.val)



3.	Consider the following C like code that involves pointers and dynamic memory allocation. *int* allocateAndFillArray(int size) {* *int* arr = (int*) malloc(size * sizeof(int));* *for (int i = 0; i < size; i++) {* *arr[i] = i * 2;* *}* *return arr;* *}* *int main() {* *int n = 5;* *int* p = allocateAndFillArray(n);*	5		

```
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += p[i];
    }
    return sum;
}
```

Generate the three-address code(TAC) for the above code. Assume that the TAC has an instruction alloc(n) that performs the malloc().

Function : allocateAndFillArray

t1 = size * 4
arr = alloc(t1) → allocate memory for array.
i = 0
L1 : if i >= size GoTo L2
t2 = i * 2
t3 = arr + i
*t3 = t2
i = i+1
GoTo L1
L2 : return arr → Return the pointer to the array.

Function : main

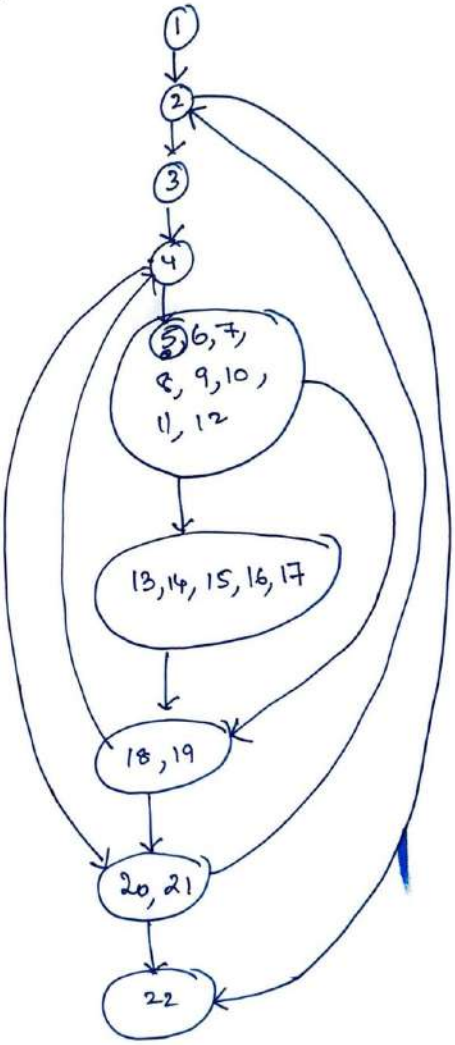
n = 5
p = call allocateAndFillArray(n)
sum = 0
i = 0
L3 : if i >= n GoTo L4
t4 = p + i
t5 = *t4
sum = sum + t5
i = i+1
GoTo L3
L4 : return sum.

4.	Consider the following C program, that transposes a 3 x 3 matrix and sums the diagonal elements: *for (int i = 0; i < 3; i++) {* *for (int j = 0; j < 3; j++) {* *transpose[j][i] = arr[i][j];*	10		

```
        if (i == j) {
            sum = sum + transpose[i][j];
        }
    }
}
```

Convert the above code into three-address code and partition the same into basic blocks. Construct the flow graph for the basic blocks, showing control flow between them.

1. i = 0 (Leader)
2. L1: if i >= 3 GOTO L6 (Leader)
3. j = 0 (Leader)
4. L2: if j >= 3 GOTO L5 (Leader)
5. t1 = 3 * i (Leader)
6. t2 = t1 + j
7. t3 = 4 * t2
8. t4 = 3 * j
9. t5 = t4 + i
10. t6 = 4 * t5
11. transpose[t6] = arr[t3]
12. if i != j GOTO L3
13. t7 = 3 * i (Leader)
14. t8 = t7 + j
15. t9 = 4 * t8
16. t10 = transpose[t9]
17. sum = sum + t10
18. L3: j = j + 1 (Leader)
19. GOTO L2
20. L5: i = i + 1 (Leader)
21. GOTO L1
22. L6: NOP (Leader)



5.	a. Consider the following C code, that uses a switch-case statement to compute different operations based on a input value.			

	int result; *int x = 5;* *switch(x) {* *case 1:* *result = x + 1;* *break;* *case 2:* *result = x * 2;* *break;* *case 3:* *result = x - 3;* *break;* *default:* *result = x;* *}* Write semantic actions to generate intermediate code for the above switch-case statement. Show the annotated parse tree for the switch case statement.			

Translation of Switch statement:

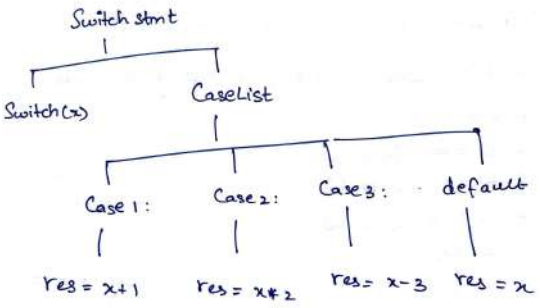
The three address code, for the given code is:

```
t1 = 5
if t1 == 1 GOTO L1
if t1 == 2 GOTO L2
if t1 == 3 GOTO L3
GOTO L4
L1: t2 = t1 + 1
    GOTO L5
L2: t2 = t2 * 2
    GOTO L5
L3: t2 = t1 - 3
    GOTO L5
L4: t2 = t1
L5 : NOP
```

(01)

```
Code to evaluate x into t
goto test
L1:
   result = t+1
   goto next
L2:
   result = t*2
   goto next
L3:
   result = t-3
   goto next
L4:
   result = t
   goto next.
test:
   if t==1 goto L1
   if t==2 goto L2
   if t==3 goto L3
   goto L4.
next:
```

(Both forms can be accepted)



b. Consider the assembly code generated by a compiler. Apply peephole optimization to optimize the code.

1. MOV R1, R2

	2. MOV R2, R1 3. ADD R3, #0 4. JMP L1 5. MOV R4, #5 6. L1: MUL R5, #1 7. MOV R6, #0 8. ADD R6, R7 9. CMP R8, R9 10. JNE L2 11. JMP L3 12. L2: JMP L3 13. L3: NOP			

REG.NO.:

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2024-2025

SLOT: C1+TC1

---

1. MOV R1, R2 : $R_1 \leftarrow R_2$
2. MOV R2, R1 : $R_2 \leftarrow R_1$ (Redudent instruction) (1)
3. ADD R3, #0 : $R_3 \leftarrow R_3 + 0$ (Algebraic Simplification) (2)
4. JMP L1
5. MOV R4, #5 } unreachable code (4)
6. L1: MUL R5, #1 : $R_5 \leftarrow R_5 \times 1$ (Algebraic Simplification) (2)
7. MOV R6, #0 : $R_6 \leftarrow 0$
8. ADD R6, R7 : $R_6 \leftarrow R_6 + R_7$ (instead use Mov - machine idiom) (3),
9. CMP R8, R9
10. JNE L2
11. JMP L3
12. L2 : JMP L3 → flow & control optimization. (5)
13. L3 : NOP

⇓

Optimized code

```
MOV  R1, R2
JMP  L1
L1 : MOV  R6, R7
CMP  R8, R9
JNE  L2
L2 : NOP
```

(1) Redudent instruction removal
(2) Algebraic Simplification
(3) Use machine idioms
(4) Unreachable code removal
(5) Flow & Control optimizatis.

****