ID _____ **Name** _____

Birla Institute of Technology and Science, Pilani
Department of Computer Science and Information Systems
Compiler Construction (CS F363)
Second Semester 2019-20

Mid Semester Test
Part 1 (Closed Book)

Date: March 3, 2020 **Duration:50 minutes**
Day: Tuesday **Max. Marks: 35**

Q1(10 M)	Q2(3 M)	Q3 (4 M)	Q4(10 M)	Q5(8 M)	Total Marks (35 M)

Rechecks:

Instructions:

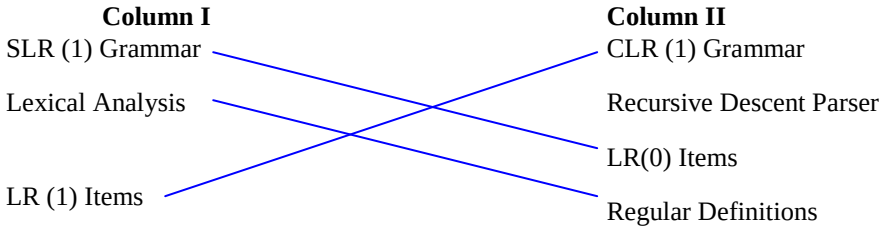
i. *Answer precisely and neatly. Overwritten answers will not be considered for recheck.*
ii. *There are FOUR printed sheets in this part. There is NO partial marking.*
iii. *Ensure that all FOUR sheets are printed, before you start answering the questions.*
iv. *Write your ID and name on all sheets. A sheet without name and ID will not be considered for evaluation.*
v. *As you complete part 1, return this to the invigilator and get the Part 2 question paper.*
vi. *The expected time to complete part 1 is 45 minutes while you can take a maximum of 50 minutes keeping in mind the time required for part 2.*

Q.1. **Fill in the blanks.** **[10 × 1M = 10 M]**

1. Canonical LR (1) Parsing method works with LR(1) items which consist of LR(0) items and lookahead symbol .(LR(0)/ LL(1)/ LR(1) items/ look ahead symbol)
2. In order to avoid the shift-reduce or reduce-reduce conflicts in Canonical LR(1) parsing, the states of such parsers include the sets of LR(1) items where various LR(1) items include the same LR(0) Items (LR(0) Items/ LR(1) Items).
3. The conditions that needs to be verified for any context-free grammar $A \rightarrow \alpha | \beta$, where α and β are the strings of terminals and non-terminals, to be LL(1) are FIRST(α) ∩ FIRST(β) = ϕ, atmost one of α and β can derive the empty string and if β derives an empty string, then FIRST(α) ∩ FOLLOW(A) = ϕ.
4. Consider the grammar G defined over the set of alphabet {(,),a,+} and a set of nonterminals {S, L}. The start symbol of the grammar is S and the productions are S → (L), S → a, L → L+S and L → S. The closure of LR (1) Item [S→(.L),$] is {[S→(•L),$],[L→•L+S,)],[L→•S,)],[S→•(L),)],[S→•a,)],[L→•L+S,+],[L→•S,+],[S→•(L),+],[S→•a,+]} where $ is the end of input marker. *[Note: If the lookaheads) and + both are not taken care of in the set of items, and any one item is missing, then the answer is considered wrong]*
5. Consider the following augmented Grammar, where terminal symbols are {other, if, else}, and the production rules are S' → S$, S → I, S → other, I → if S, I →if S else S. The grammar is none. (LR(0)/ SLR(1)/ Both/ None).
6. For the successful implementation of lexical analyzer, the lexical analyzer must follow the regular definitions, both. (priority rules/ longest match principle/ both/ none)
7. Consider the following grammar rules defined over set of alphabet {@, *, (,), a, b}, A→A@A, A→AA, A →A*, A→(A), A→a|b, where A is the start symbol. Then, First (A) = {a, b, (} and Follow (A) = {a,b,(,), *, @, $}

8. An LR (0) parser is a shift reduce parser that is described by zero token look-ahead. What does this 0 indicate? 0 represents blind reduction i.e. reduction without any lookahead.
9. The SLR parser is a LR(0) parser with one additional condition on reduce action only if the next token is in FOLLOW set of the nonterminal symbol of the production rule where the reduction is done.
10. Recursive descent parsing algorithm cannot parse the grammars which are left recursive and not left factored.(left recursive/ left factored/ LL(1)).

Q.2. Match the following two columns appropriately.[The lines joining the pairs must be straight else will not be considered for evaluation] **[3×1M=3 M]**



Q.3. Answer the following questions **[2+2 = 4M]**

(a) A grammar defined over the alphabet { @, #, %, &, 1, 4, 2, 5, 6} has productions

$A \rightarrow$ @B% | #BCg

$B \rightarrow$ &CB1| 42 | $\in$

$C \rightarrow 5 \mid 6$

Which columns corresponding to the row of B, in predictive parsing table, will have the entry of the rule $B \rightarrow \in$? Give reasons to support your answer.

The columns corresponding to all terminals in the set FOLLOW(B), in the predictive parsing table entry for row of B, contain rule $B \rightarrow \in$. The set FOLLOW(B) = {%}U{1} U FIRST(C) = {%, 1, 5, 6}.

(b) Consider the following grammar defined over the alphabet {=, *, id} with start symbol S' and end of input marker $.

S'→ S$
S→ L=R
S→ R
L→ *R
L→id
R→L

Compute the state I_1 = GOTO(I_0, L), where I_0 is the start state defined as CLOSURE([S'→.S$]) and discuss the conflicts in the state I_1.

I_0= { [S'→ •S$], [S→ •L=R], [S→ •R], [L→ •*R], [L→•id], [R→•L]}
and I_1=GOTO(I_0, L) = {[S→ L•=R] [R→L•]}
The item [S→ L•=R] induces a shift to a state { [S→ L=•R], L→ •*R], [L→•id], [R→•L]} on seeing a lookahead '='. Also, the item [R→L•] induces a reduce action *on all terminals belonging to FOLLOW(R)* which is {=, $}. Hence, a lookahead '=' leads to a shift-reduce conflict at I1.

[Note: Full 2 marks are given if the exact reason of the conflict is mentioned as is highlighted in italics above, else 0 is awarded]

Q.4. **Fill in the blanks.** **[10 × 1M = 10 M]**

1. Consider the grammar G defined over {a,b, c, +, -} with production rules S→ AaBb | b+a, A→ c-bBa | ∈, B→ cb+A |∈. Compute FIRST (S)={a, b, c} *[Note: inclusion of epsilon here is also considered right]* and FOLLOW(B) = {a,b}
2. An item [A→ abB.] in an LR(0) automaton induces a reduce action in SLR parsing table.
3. The two 'L's in LL(1) grammar represent _left to right scanning and left most derivation
4. The handle in bottom up parsing is described as a substring that matches the body of the production, and whose reduction represents one step along the reverse of a right most derivation. *[Note: the understanding of reduction leading to rightmost derivation, if communicated properly is awarded 1 mark, but mere mention of the RHS of a production is given a 0]*
5. Consider a grammar defined over the set of alphabet {u,v,w,q,r,s}with start symbol S and the production rules S→ PvS, P→ QRSw| u| ε, Q→q | ε, R→r| ε , S→ s| ε. Is the grammar LL(1)? Give justification. No, the grammar is not LL(1). FIRST(S) ∩ FOLLOW(S) = (FIRST(PvS) U FIRST(s)) ∩ FOLLOW(S) = {q, r, s, w, u, v} ∩ {w} = {w}≠ ϕ.
6. The key decisions during bottom up parsing are about when to reduce and when to shift based on the lookahead, and which rule to apply for reduction.
7. The key decisions during top down predictive parsing are about which production rule to select.
8. The difference between Recursive Descent parsing and table driven top down predictive parsing is that Recursive Descent algorithm requires backtracking in general while the top down predictive parsing does not require backtracking. *[Note: Only the basic difference of backtracking is considered and awarded 1 mark. Any mention of the recursive and nonrecursive nature of algorithms, or the use of procedures versus table is not considered as correct answer]*

9. The size of the predictive parsing table in terms of number of nonterminals and terminals used in the grammar is Number of rows = Number of nonterminals and Number of colums = 1+number of terminals.
10. If an SLR(1) parser is in configuration (s_0 , s_1 , s_2 , .., s_m , a_i , a_{i+1} , a_{i+2} , ...a_n\$) and ACTION[s_m , a_i] = reduce by A→β where β has r symbols in it, what will be the configuration of the parser after applying reduce action? (s_0 , s_1 , s_2 , .., s_{m-r} , S, a_i , a_{i+1} , a_{i+2} , ...a_n\$) where S = GOTO(s_{m-r} , A) *[Note: If any one of s_{m-r} , S and a_i is not mentioned correctly, and the meaning of S as = GOTO(s_{m-r} , A) is not communicated, then the answer is considered wrong and a 0 is awarded here.]*

Q.5. Write semantic rules to construct an Abstract Syntax Tree (AST) for the nonterminal <paramater_list> given the following grammar . Design semantic rules for creating AST to preserve the sequence of parameters (P) and their associated types(T) in such a way that the AST takes the shape of a linked list of (P,T) pairs. Also, dry run these rules for input string of **boolean : x, array[12..20] of real: z** and compute the ratio of number of nodes in the parse tree and AST of the given input. **[8M]**

[1]. <parameter_list>→<D> COLON ID <R>
[2]. <D>→ <P>
[3]. <D>→ ARRAY SQBO <N> SQBC OF < P >
[4]. <P>→ REAL
[5]. <P>→ BOOLEAN
[6]. <R>→COMMA <parameter_list>
[7]. <R>→ε
[8]. <N> →NUM RANGEOP NUM

Answer here: List the **attributes** used below, specifying the purpose

Attribute	Purpose	Attribute	Purpose
Syn_list	Linked list computed bottom up	addr, node	Node addresses of relevant nodes
Inh_list	Linked list computed top down	List_head	Head of linked list

Ratio of number of nodes of parse tree and AST for the given input string: *[Note: This was used to understand your AST construction]*

Write semantic rules below corresponding to each of the above 8 syntax rules

(Note: Variations in answers are considered given that the construction of the list is correct. Freeing of redundant nodes of the parse tree is an essential part of the answer. While many of you used type as a field for many of the non terminals such as D and P, which is not expected in AST construction, you needed only to propagate addresses of the leaf nodes containing type. If you write D.type = P.type for part 2, then you lose information of line number and lexemes stored in relevant leaf nodes. Similar is for rules 4 and 5)

Rule number	Semantic rules for AST as asked (Approach 1: synthesized list of parameter-type pairs)	Semantic rules for AST as asked (Approach 2: inherited list of parameter -type pairs)

[1] 3M	<parameter_list>.node = makenode(label: parameter-type-pair, D.addr, ID.addr, NULL)
	<parameter_list>.syn_list=insert_at_beginning(R.syn_list, <parameter_list>.node) computed while going from child to parent bottom up
	Free nodes- D, :, R
		Initialize <parameter_list>.inh_list = NULL
		<parameter_list>.node = makenode(label: parameter-type-pair, D.addr, ID.addr, NULL)
		<parameter_list>.list_head = insert_at_end(<parameter_list>.inh_list, <parameter_list>.node)
		R.inh_list= <parameter_list>.list_head //computed while going from parent to child top down
		<parameter_list>.syn_list = R.syn_list //computed while going from child to parent bottom up
		Free nodes- D, :, R

[2] 0.5 M	D.addr = P.addr Free node-P	D.addr = P.addr Free node-P
[3] 0.5 M	D.addr = makenode(label: array, N,addr, P.addr) Free node-ARRAY, [,], OF, N, P	D.addr = makenode(label: array, N,addr, P.addr) Free node-ARRAY, [,], OF, N, P
[4] 0.5 M	P.addr = REAL.addr	P.addr = REAL.addr

[5] **0.5 M**	P.addr = BOOLEAN.addr	P.addr = BOOLEAN.addr

[6]
2 M
	R.syn_list = <parameter_list>.syn_list //computed while going from child to parent bottom up
	Free nodes - , and L
		<parameter_list>. inh_list= R.inh_list //computed while going from parent to child top down
		R.syn_list = <parameter_list>.syn_list //computed while going from child to parent bottom up
		Free nodes - , and L

[7] **0.5 M**	R.syn_list = NULL Free node - epsilon	R.syn_list = R.inh_list Free node - epsilon
[8] **0.5 M**	N.addr = makenode(label:range, NUM1.addr, NUM2.addr)	N.addr = makenode(label:range, NUM1.addr, NUM2.addr)

****