BIRLA INSTITUTE OF TECHNOLOGY, MESRA, RANCHI
(END SEMESTER EXAMINATION)

CLASS: BE **SEMESTER : VII**
BRANCH: IT **SESSION : MO/19**

SUBJECT: IT7043 COMPILER DESIGN

TIME:3:00 HOURS **FULL MARKS: 60**

INSTRUCTIONS:
1. The question paper contains 7 questions each of 12 marks and total 84 marks.
2. Candidates may attempt any 5 questions maximum of 60 marks.
3. The missing data, if any, may be assumed suitably.
***4*. Before attempting the question paper, be sure that you have got the correct question paper.**
5. Tables/Data hand book/Graph paper etc. to be supplied to the candidates in the examination hall.

Q.1(a) What are the features of a good language and compiler? [2]

Q.1(b) What is the front-end and the back-end of a compiler? Explain in details. [4]

Q.1(c) Explain the various phases of a compiler with help of the following program. [6]

```
main() {
int a, b;
float c,d;
c=a+b*10; d=a-b/2.0;
   }
```

Q.2(a) Which of the following expressions have *l*-values and / or *r*-values. [2]
(i) A[I+1] (ii) * A (iii) & A (iv) &(* A) (vi) * (&(&A)) (v) *(& A)

Q.2(b) Consider the following C-program. [4]

```
int main( )
{   int  i, n;     fro (i=0; i< N; i++);    }
```

Is there any lexical error in the given program? Justify your answer.

Q.2(c) Write a Lex program that will check no. of *a*'s is divisible by 2 or no. of *b*'s is divisible by 3. [6]

Q.3(a) What are the steps to eliminate left recursion in CFG? [2]

Q.3(b) Consider the following grammar having two non-terminals : {A, D}, [4]
five terminals : {a, b, c, d, f, g} and
two regenerating/production rules : { A→abcDfg, A→abcDgf }
Identify the problem present in the grammar that cause further inconvenient in design of a Top-down parser.

Q.3(c) Refer to question 3(b) [6]
i) Find the precedence table.
ii) Explain the operator precedence parsing algorithm.

Q.4(a) Explain Recursive Descent Parser with suitable example. [2]

Q.4(b) Consider the grammar: [4]
E→TE', E'→E | e, T→F|T', T'→T|e, F→PF', F'→*F'|e, P→(E)|a|b|e
Here, *e* stands for null (epsilon).
i) Compute FIRST & FOLLOW for each non terminal of the above grammar.
ii) Find the predictive parsing table and conclude whether the grammar Is LL (1) or not.

Q.4(c) Consider the grammar [6]
S→ Aa|bAc|Bc|bBa
A → d
B → d
Show the grammar is LR (1) but not LALR (1).

Q.5(a) Differentiate between the Abstract Syntax Tree and the Directed Acyclic Graphs with suitable example. [2]

Q.5(b) Consider the following grammar: [4]
S →xxW { printf "1"}
S → y { printf "2"}
W →Sz { printf "3"}
Construct the annotated parse tree to find out the output for the input expression "xxxxyzz".

PTO

Q.5(c) Write Three address code generation process for the following expression: [6]

```
If  ( (a<b) or ((c<d) and (e>f)))
 {   Z = X+Y;  }
else
     Z=Z+1;
```

Q.6(a) Differentiate between SDD and SDT. Also, discuss different types of SDT's with simple examples. [2]

Q.6(b) What do you mean by runtime storage allocation? Explain the difference between static and dynamic allocations. [4]

Q.6(c) Consider the following grammar [6]

E →E +T, E → T

T →T*F, T→ F, F →id

Write corresponding semantic action for each of the Non-Terminal, so that postfix expression can be obtained for any infix expression. Construct the annotated parse tree for the input expression 2*4+ (3+4/6).

Q.7(a) What are the main purposes for optimization techniques? [2]

Q.7(b) Explain the following with suitable examples: [4]

(i) Loop Fission (ii) Loop Interchange (iii)Loop Reversal (iv) Loop Splitting

Q.7(c) i) Consider the following C code segment. [6]

```
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
        if (i%2)
 {x + = (4 * j + 5 * i);   y + = (7 + 4 * j);  } }  }
```

Modify the program using suitable optimizing techniques

ii) Specify the necessary and sufficient conditions for performing loop optimization and dead code elimination. Give suitable examples.

::::::02/12/2019::::::E