BCSE THIRD YEAR EXAMINATION 2018
(2^{nd} Semester)
COMPILER DESIGN

Time : Three hours **Full Marks: 100**

Group-1 (20 marks)	Answer Question No. 1 and any one from 2 and 3 1. (a) Describe the role of lexical analysis in the design of a compiler. What are token, pattern and lexeme? Give example of each. (b) Define *regular grammar*. Write a regular expression for floating point numbers. 4+4 2. (a) Construct an NFA for the following regular expression using McNaughton-Yamada-Thompson algorithm. (a b* c) | (a b c*) (b) Construct a DFA for the above NFA using subset construction. 6+6 3. Construct a DFA directly from a regular expression (a b* c) (a | b)* 12
Group-2 (40 marks)	Answer any two questions from this group. 4. (a) Write a context free grammar for *if* and *else* statement in C. Is your grammar unambiguous? Justify your answer. (b) The following grammar generates all strings that start with one or more a, and end with a single b. This is an ambiguous context free grammar. S →Ab | aaB A →a | Aa B →b (i) Find a string s generated by the grammar that has two leftmost derivations. Show the derivations. (ii) Find an equivalent unambiguous context-free grammar. (iii) Compute the FIRST and FOLLOW sets for the unambiguous grammar and construct the parsing table. 6+(4+4+6)=20 5. Consider the grammar: $S \rightarrow a\ A\ B\ e$ $A \rightarrow A\ b\ c$ $A \rightarrow b$ $B \rightarrow d$ (a, b, c, d, e are terminals and S is the start symbol) (a) Show the parse trees of four strings (at least four characters long) which are generated from the above grammar. (b) Generate the LR(0) item set and SLR parsing table for the above grammar. (c) What is *shift-reduce* conflict? Explain with example. 4+(8+6)+2=20 6. Consider the grammar: $S \rightarrow Bbb \mid aab \mid bBa$ $B \rightarrow a$ (a, b are terminals and S is the start symbol) (a) Generate the LR(1) item set for the grammar. (b) Generate LALR parsing table. (c) What are the strings that can be generated from this grammar? Show the trace of parsing any one string. (d) What is a viable prefix? Give examples of 4 viable prefixes from the above grammar. 6+6+4+4=20
Group-3 (30 marks)	Answer any two questions from this group. 7. (a) Write a syntax-directed definition for generating three address code for an assignment statement S with the following productions. Use attribute 'code' for S and attribute 'addr' and 'code' for an expression E. S → id = E E → - E | E + E | E – E | id

(b) Describe the different ways of managing the scopes of variables in a symbol table.

8+7=15

8. (a) Generate three address code for the following block.

```
int a[10][10], sumofsq;
sumofsq=0;
for (i=0; i<10; i++)
    for (j=0; j<10; j++)
        sumofsq += a[i][j] * a[i][j];
```

(b) If the above code is written in a programming language which stores array variables in column-major order,, then what changes are needed in the three address code?
(c) What is the use of symbol table? How do you check the types of variables using a symbol table?

6+6+3=15

9. (a) Why does a compiler need *semantic analysis* phase?
(b) What do you mean by syntax-directed translation? What is an L-attributed grammar?
(c) What are the operations performed on a symbol table when it is implemented as a single hash table?
(d) Consider the following code block:

```
int sum (int k)
int a = k;
float area;
for (j = 0; j < a; j++) {
   float k = 3.14;
   double area;
   area += k * j * j;
}
return area;
```

With appropriate implementation show how the scope of the identifiers are maintained in the symbol table.

3+4+4+4=15

Group-4 (10 marks)

Answer any one question

10. (a) What is a basic block? Write an algorithm for identifying basic blocks.
(b) Optimize the following code and discuss each optimization technique that you have applied stating their advantages.

```
#include <stdio.h>
int main() {
   int j, n, array[20], k=1;
   for(j=0; j < 10; j++) {
       n=11;
       array[n+j] = j*5;
   }
   for(j=0; j < 10; j++)
       array[j+1] = array[j] + array[j+1];
   return 0;
}
```

5+5=10

11. (a) What are the uses of *register descriptor* and *address descriptor* in code generation.
(b) Explain the following code optimization techniques using examples:
(i) Constant propagation, (ii) Constant folding.
(c) How is the function *getreg* used for register allocation?

3+3+4=10