B. INFO. TECH 3RD YEAR 1ST SEMESTER EXAMINATION, 2019

PRINCIPLES OF COMPILER DESIGN

Time: Three hours Full Marks: 100

Different parts of the same question should be answered together.

CO1
[20]

[1] Answer any one out of (a) and (b) from this block.

(a) A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.

T_1: a?(b|c)*a
T_2: b?(a|c)*b
T_3: c?(b|a)*c

(i) Design DFAs for the three tokens.
(ii) Design an NFA for the Lexical Analyzer.
(iii) Convert the NFA produced in part (ii) to a DFA

[3x3+1+10]

(b) Answer the following questions.
(i) Design a DFA for C++ style comments using the following $\sum = \{/, *, a\}$.
(ii) Design a Lexical Analyzer based on the DFA produced in (i).

[10+10]

CO2
[20]

[2] Answer any two out of (a), (b), and (c) from this block.

(a) Construct the LL(1) parsing table for the following grammar after suitable modifications

$S \rightarrow (L) \mid a$
$L \rightarrow L * S \mid S$
$\sum=\{a, *, (,)\}$

[10]

(b) Consider the following grammar G. Construct the LL(1) parsing table.

$S \rightarrow F \mid H$
$F \rightarrow p \mid c$
$H \rightarrow d \mid c$

[10]

(c) Consider the following grammar

$R \rightarrow R \cup R \mid R \cdot R \mid R^* \mid (R) \mid a \mid b$

Construct a LL(1) parse table for this grammar.

[10]

CO3
[20]

[3] Consider the following grammar.

$E \rightarrow TT$
$T \rightarrow +T \mid \underline{id}$

Construct the LR(1) Sets of Items. There are 10 sets.
Which LR(1) sets can be merged to create LALR sets?

[16+4]

CO4 [20]

[4] Answer two questions out of (a), (b), and (c) from this block.

(a) Consider the following grammar for C style array declaration.

L → id[num]
    | L [num]

Define the syntax directed translation for generating three-address code for sentences obtained using this grammar.

[10]

(b) Consider a programming language in which programmers define variables in the following manner.

int a, b, c ; float d, e, f ;

Consider another programming language in which programmers define variables in the following manner.

a, b, c : integer;
d, e, f : real;

(i) Define a grammar for declarations representing the first style mentioned. Assume *id*, ‘,’, ‘;’, *int*, *float* to be terminals.

(ii) Write a translation scheme to translate the definitions of variables written in the first style to definitions in the form mentioned in second style.

[4+6]

(c) Consider the syntax of the do-loop in FORTRAN as described below. Design a translation scheme to convert it to three address code.

The DO statement repeatedly executes a set of statements.
DO *s* *loop-control*
where *s* is a statement number.
The form of *loop-control* is
variable = *e1*, *e2* [, *e3*]
variable - Variable of type integer, real, or double precision.
e1, *e2*, *e3* - Integer constants, specifying initial, limit, and increment (or decrement) values respectively.

Description of DO loop

A DO loop consists of the following:
- DO statement
- Set of assignment statements called a block
- Terminal statement, a CONTINUE statement

Restrictions
- The DO variable must not be modified in any way within the range of the DO loop. In FORTRAN, every assignment statement appears in a separate line.

[10]

CO5 [20]

[5] Answer any two out of (a), (b), and (c) from this block.

(a) Convert the following code to a three-address code, identify the basic blocks and draw the control flow graph.

```
w = 0;
x = x+ y;
y = 0;
if (x > z) {
   y = x;
   x++;
}
```

```
else {
   y = z;
   z++;
}
w = x+z;
```

[3+5+2]

(b) Consider the following code snippet.

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

With respect to the above code snippet, answer the following questions:

i) What common sub-expression(s) can be eliminated?
ii) Identify the loop invariant computation?
iii) Is there any scope of strength reduction? Explain
iv) What is the possibility of dead code elimination?

[3+2+3+2]

(c) Draw the hierarchical symbol table for the following code segment.

```
void proc()
{
      int one_1;
      int one_2;
      {
         int one_3;
         int one_4;
      }
      int one_5;
      {
         int one_6;
         int one_7;
      }
}
struct xyz
{
   int a;
   int b;
}
```

[10]