CoP

USN | 1 | P | I | | | | | | | |

PES Institute of Technology, Bangalore

14CS351

Department of Computer Science / Information Science & Engineering

SEMESTER END EXAMINATION (SEE) B. E. VI SEMESTER

[Session: **May, 2017**]

14CS351 - COMPILER DESIGN

Time: 3 hrs. **Answer All Questions** Max Marks: **100**

1. a) Consider the following tokens and their associated regular expressions, given as a lex-like specification: **[5]**

```
%%
(01|10)           print ("course")
0(01) *1          print ("compiler")
(1010*1|0101*0)   print ("design")
```

Give an input to this scanner such that the output string is

(compiler ^{11} design ^{2}) ^{4} course ^{3}

Where, A^i denotes A repeated i times. (And, of course, the parentheses are not part of the output.) You may use similar shorthand notation in your answer.

1. b) Recall from the lecture that, when using regular expressions to scan an input, we resolve conflicts in lexer by taking the largest possible match at any point. That is, if we have the following lex scanner specification: **[5]**

```
%%
do                          { return T_Do ; }
[A-Za -z_ ][A-Za -z0 -9_]*  { return T_Identifier ; }
```

and we see the input string "dot", we will match the second rule and emit T_Identifier for the whole string, not T_Do.

However, it is possible to have a set of regular expressions for which we can tokenize a particular string, but for which taking the largest possible match will fail to break the input into tokens.

Give an example of a set of regular expressions and an input string such that:

a) The string can be broken into substrings, where each substring matches one of the regular expressions,

b) and using our usual lexer algorithm, taking the largest match at every step, will fail to break the string in a way in which each piece matches one of the regular expressions. Explain how the string can be tokenized and why taking the largest match won't work in this case.

def	IN	OUT
y	–	y
x	y	x, y
x	x, y	x, y
z	x, y	x, y
y	x	–

39, 57, 41, 92, 29, 43

1. c) Explain the front end of a compiler using the following example: **[10]**

if(x>10) x = x + 100/x;

2. a) Consider the following simple context free grammars: **[10 (3 + 7)]**

G_1:

$S \rightarrow Aa$

$A \rightarrow \varepsilon$

$A \rightarrow bAb$

G_2:

$S \rightarrow Aa$

$A \rightarrow \varepsilon$

$A \rightarrow Abb$

Note that the grammars generate the same language: strings consisting of even numbers of b's (including 0 of them), followed by an a.

54, 18, 34, 41, 41, 57, 39, 32

	a	b	c	d
1	5	5	10	
2	10 (3+7)	10 (6+4)		
3	10	10		
4	5	5	10	
5	5	5	5	5

	First	Follow
S	u, w, x, y, z	$
u	u, y, z, ε	w x y z
v	w, x, ε	y z
w	y, z	v, $

$(compiler)^{11}(design^2)^4$ ... 011001

USN	1	P	I						

		I. Attempt to show a shift-reduce parse of the string bbbba for a parser for grammar G_1. Show the contents of the stack, the input, and the actions (i.e., shift, reduce, error, accept). You don't need to create a parse table; just use your knowledge of the grammar and how the parser works. Be sure to indicate any conflicts and explain why they are conflicts. **II.** Using the definition and properties of various parsers that you studied in Compiler design course, justify whether or not: 1. G1 is LR(1)? 2. G2 is LR(0)? 3. G1 is LL(1)? : 4. G2 is LL(1)? **[You get credit only for correct justification not for writing yes or No]**	

2.	b)	Consider the following grammar over the alphabet $\sum = \{ u, v, w, x, y, z\}$
		`S -> UVW`
		`U -> u | Wv | ϵ`
		`V -> w | xU | ϵ`
		`W -> y | z`
		I. Give the first sets and follow sets of each non-terminal in the grammar.
		II. Is this grammar LL (1)? Justify.
			10

3.	a)	Consider grammar and rules given below for array address translation and generating 3 address code for array references: • L.array.basename means name of the array • L.array.typeofelement means type of the element of the array • L.type.width means width of L.type Assume size of integer to be 4 bytes, and lower bound of the arrays to be 0 Let A, B and C be 10 X 5, 5 X 7, and 10 X 7 arrays of integers respectively. Let i, j, and k be integers. **Construct an annotated parse tree for the expression** **C[i][j] + A[i][k] * B[k][j]** **and show the 3-address code sequence generated for the expression.** $E \rightarrow E_1 + E_2$ {E.addr = newtemp(); gen(E.addr '=' E_1.addr '+' E_2.addr);} $E \rightarrow E_1 * E_2$ {E.addr = newtemp(); gen(E.addr '=' E_1.addr '*' E_2.addr);} | id {E.addr = id.lexeme; } | L {E.addr = newtemp(); gen(E.addr '=' L.array.basname '[' L.addr ']'); } $L \rightarrow$ id [E] {L.array = id.lexeme; L.type = L.array.typeofelement; L.addr=newtemp(); gen(L.addr '=' E.addr '*' L.type.width);} | L_1 [E] {L.array = L_1.array; L.type = L_1.type.typeofelement; t = newtemp(); L.addr = newtemp(); gen(t '=' E.addr '*' L.type.width); gen(L.addr '=' L_1.addr '+' t); }	10

USN	1	P	I							

3.	b)	Provide an implementation of SDT scheme for simple type declaration (grammar is given below for reference) during LR parsing. **For full credit, provide proper explanation and show parser stack when necessary.** Consider for example the input string as **int a, b** **D -> TL** **T -> int | float** **L -> L, id | id** (Note: Provide the general structure of parser stack used during LR parsing.)	10
4.	a)	What are the different ways in which a procedure can access non-local data on a run time stack? Explain each technique using a simple example.	5
4.	b)	Generate Target Code for the following procedure call assuming static allocation.	5

	Code is kept at address	Activation record is kept at address
main	100	600
p	400	800

//main()	//p()
n = 6 i = 0 L1 : if i >= 6 goto L2 i = i + 1 goto L1 L2 : call p halt	i = 60 return

4.	c)	Generate 3-address code for the following program and convert it into a CFG.	10

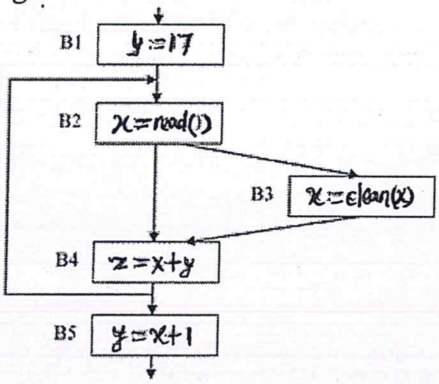
```
int prime(int n)
{
   int n, i, flag = 0;
   for(i=2; i<=n/2; ++i) {
      if(n%i==0){
         flag=1;
         break;
      }
   }
   if (flag==0)
         return 0;
   else
         return 1;
}
```

5.	a)	What are the issues in the design of a code generator?	5
5.	b)	Optimize the code below by applying the following code transformations: constant propagation, constant folding, copy-propagation, dead-code elimination and strength reduction.	5

USN	1	P	I							

```
L0: t1 = t1 + 1
    t2 = 0
    t3 = t1 * 8
    t4 = t3 + t2
    t5 = t4 * 4
    t6 = *t5
    t7 = FP + t3
    *t7 = t2
    t8 = t1
    if (t8 > 0) goto L1
L1: goto L0
L2: t1 = 1
    t10 = 16
    t11 = t1 * 2
    goto L1
```

5. c) Perform Live variable analysis on the following CFG. Provide your answer in the table format as given below next to the CFG. — 5



	use	def	IN	OUT
B1				
B2				
B3				
B4				
B5				

Note : clean(x) is some function performed on the value of x.

5. d) Construct DAG for the following Block and optimize. — 5

1. t1 : = 4 * I
2. t2 := A − 4
3. t3 := t2 [t1]
4. t4 := 4 * I
5. t5 : = B − 4
6. t6 : = t5 [t4]
7. t7 : = t3 * t6
8. t8 : =PROD + t7
9. PROD : = t8
10. t9 : = I + 1
11. I = t9
12. if I ≤ 20 goto (1).