COMPILER DESIGN
(CSEN 4137)

Time Allotted : 2½ hrs **Full Marks : 60**

Figures out of the right margin indicate full marks.

Candidates are required to answer Group A and any 4 (four) from Group B to E, taking one from each group.

Candidates are required to give answer in their own words as far as practicable.

Group – A

1. Answer any twelve: **12 × 1 = 12**

Choose the correct alternative for the following

(i) If *x* is a terminal, then *FIRST{x}* is
(a) €
(b) {x}
(c) x*
(d) None of these.

(ii) A given grammar is not LL(1) if the parsing table of a grammar may contain
(a) any blank field
(b) any ε-entry
(c) duplicate entry of same production
(d) more than one production rule.

(iii) Given the grammar:
S → ABc
A→ a| ε
B→ b| ε
FOLLOW(A) is the set :
(a) {$}
(b) {b}
(c) {b, c}
(d) {a, b, c}

(iv) The grammar A-> AA | (A) | € is not suitable for predictive parsing because the grammar is
(a) Ambiguous
(b) Left Recursive
(c) Right Recursive
(d) None of these.

(v) Reduction in strength means
(a) replacing run time computation by compile time computation
(b) removing loop invariant computation
(c) removing common sub expression
(d) replacing a costly operation by a relatively cheaper one.

(vi) YACC builds up
(a) SLR parsing table
(b) Canonical LR parsing table
(c) LALR parsing table
(d) None of the above.

(vii) A parse tree showing the values of attributes at each node is called in particular
(a) Syntax Tree
(b) Annotated Parse Tree
(c) Syntax Directed Parse Tree
(d) Direct Acyclic Graph.

(viii) Preservation of functional dependency is ensured by which of the correctness rule of the fragmentation?
(a) Top down parsers are more powerful than Bottom up parsers
(b) In any parser for an ambiguous grammar, ambiguity can be removed by using 'k' lookahead symbols, by choosing the value 'k' substantially high
(c) An unambiguous grammar may create two different parse trees for a given string using Top down and Bottom up approaches
(d) An LR(K) parsing table may have some conflicts which may be removed in an LR(k+1) parsing table.

(ix) Consider the following grammar:
A → bBq | Gr | Qk
B→ sB | t | ε
Q → qQ | ε
G → Bt
(a) This grammar is not LL(1)
(b) This grammar is LL(1) but not LR(0)
(c) This grammar is SLR(1) but not LR(0)
(d) This grammar is LR(k) for all k

(x) Consider the following syntax-directed definition D1 over a grammar defined below:
S → A Sign { S.val = A.val; A.sign = Sign.sign; print(A.val); }
Sign → + { Sign.sign = 1; }
Sign → − { Sign.sign = 0; }
A → n { A.val = value(n); }

A → A1 , n { A1.sign = A.sign;
If (A.sign = 1) then
A.val = min (A1.val,value(n));
else
A.val = max(A1.val,value(n));
}

Here value(n) return the numerical value of 'n' and print(n) prints the numerical value of n.
Consider the string "5,2,3−".
(a) The string is not a valid one according to the grammar
(b) '5' will be printed after the string is parsed and evaluated
(c) '3' will be printed after the string is parsed and evaluated
(d) '2' will be printed after the string is parsed and evaluated.

Fill in the blanks with the correct word

(xi) White spaces and tabs are removed in _____.

(xii) A Top-Down Parser generates _____.

(xiii) One form of intermediate code is _____.

(xiv) Dead Code Elimination is a type of code _____ technique.

(xv) Statement starting from a leader up to the next leader is called _____.

Group - B

2. (a) Convert the following statement into tokens:
if (x>3) then y=5 else y=10; *[(CO4)(Analyse/IOCQ)]*
(b) Give the NFA for the following Regular expression and then find a DFA for this:
(a | b)* abb *[(CO4)(Understand/LOCQ)]*
(c) What are the analysis phase and synthesis phase of a compiler? *[(CO3)(Remember/LOCQ)]*
4 + 6 + 2 = 12

3. (a) What are the different phases of a Compiler? *[(CO1)(CO3)(Remember/LOCQ)]*
(b) Show the output of each phase, using the example of the following sequence of statements:

```
while (i < 0) do
    begin
        j = j + 1;
    end
```

[(CO2)(CO3)(Understand/LOCQ)]
(c) (i) Assume you have no compiler for any high level languages available with you. You are to design a new compiler for a high level language FTRN. How will you proceed?
(ii) Assuming you have successfully built a compiler for FTRN, explain how you can use it to create a new compiler for another high level language PTHN more efficiently than the first compiler (for FTRN). *[(CO1)(CO2)(Analyze/IOCQ)]*
2 + 6 + (2 + 2) = 12

Group – C

4. Consider the following grammar:
S → A B d A → a b B→c
(i) Construct the LR(0) sets of items for the above grammar. *[(CO4)(CO5)(Understand/LOCQ)]*
(ii) Construct the SLR(1) parsing table for the same. *[(CO4)(CO5)(Apply/IOCQ)]*
(iii) Comment whether "abc" can be a viable prefix of a language generated by this grammar. *[(CO4)(CO5)(Analyze/HOCQ)]*
(iv) Explain why can't you have recursive procedures in static storage based activation record mechanism, whereas for stack based it is possible. *[(CO4)(CO5)(Understand/LOCQ)]*
(3 + 4 + 2 + 3) = 12

5. (a) Consider the following grammar and construct the SLR table:
E→ E+T|T
T→T*F|F
F→(E)|id *[(CO5)(Analyse/IOCQ)]*
(b) Explain the terms "shift-reduce" conflict and "reduce-reduce" conflict in the context of LR parsers. *[(CO5)(Remember/LOCQ)]*
(c) Consider the grammar.
E → E+E | E*E | id
Find the handles of the reductions for the string *id+id*id.* *[(CO2)(Apply/IOCQ)]*
6 + 3 + 3 = 12

Group – D

6. (a) Generate an annotated parse tree for the string "*3 + 2 – 4*" using the grammar.
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow 0 \mid 1 \mid 2 \mid ..... \mid 9$ *[(CO3)(Analyse/IOCQ)]*

(b) What is the significance of flow graph? What are the contents of Activation Record? *[(CO4)(Remember/LOCQ)]*

(c) Construct the DAG for the following statement :
*A = (x+y) * (x-y) + x / (x-z) + (x+y)* *[(CO2)(Apply/IOCQ)]*

5 + (2 + 2) + 3 = 12

7. Consider the following grammar for a certain programming language.

$S \rightarrow$ for (E ; E ; E) S
$S \rightarrow$ id := E;
$E \rightarrow E + E$
$E \rightarrow E < E$
$E \rightarrow$ id | num;

You may assume E.loc, E.type, id.loc, num.val and other necessary attributes are available as usual. You have to design the intermediate code generator in such a manner that all statements, labels and other necessary items are generated in a single pass.

(i) Make necessary enhancements to the above grammar to accomplish this. *[(CO4,CO5)(Understand/LOCQ)]*

(ii) Design an Intermediate Code Generator using the grammatical rules above. *[(CO4)(Remember/LOCQ)]*

(iii) Translate the following code into intermediate representation using the scheme above:

```
for (i=0; i <6; i:= i + 1)
        x := x + 5;
```

[(CO4)(Apply/HOCQ)]

(4 + 4 + 4) = 12

Group - E

8.

```
i=0
B2: j=0
B3: t1 = 10*i
t2 = t1 + j
t3 = 4 * t2
t4 = t3 - 44
a[t4] = 1
j = j + 1
If j<10 goto B3
i = i + 1
If i < 10 goto B2
i=1
B6: t5 = i - 1
t6 = 44 * t5
a[t6] = 1
i=i+1
If i <10 goto B6
```

(i) Show the Basic Building Blocks and the corresponding Flow Graph for the above intermediate code. *[(CO4,CO6)(Understand/LOCQ)]*

(ii) Optimize the above using elimination of the common sub-expressions, followed by copy propagation / dead code elimination and then induction variable elimination. *[(CO4,CO6)(Apply/LOCQ)]*

(iii) What is the utility of Peephole optimization? Show an example of Jump-over-jump and how this can be removed. *[(CO4,CO6)(Remember/LOCQ)]*

[(2 + 2) + (1 + 2 + 2) + (1 + 2)] = 12

9. (a) Translate the following code into machine code and show the register and address descriptors while the instructions are generated. Assume that three registers are available.

```
A = B * C
F = B + D
G = A + F
B = E
E = A/ F
```

[(CO6)(Understand/LOCQ)]

(b) Consider the following intermediate code:

```
        i=0
        j=0
L0:     if (i < n) goto L1
        goto L3
L1:     if (j  < n) goto L2
        goto L3
L2:     t1 := 4*j
        t2 := t1 + i
        x := t2
        y:= i – t1
```

```
        i:= i + 1
        j:= j + 1
        goto L0
L3:     < Some code not relevant to the question>
```

What types of optimization scopes are present in the code? Incorporate the associated optimization techniques.

[(CO4,CO6)(Apply/IOCQ)]

6 + (2 + 4) = 12

Cognition Level	LOCQ	IOCQ	HOCQ
Percentage distribution	57.3	36.5	6

Course Outcome (CO):

After the completion of the course students will be able to

CO1. To identify the key concept areas of language translation and compiler design.
CO2. To be familiar with compiler architecture.
CO3. To understand the working principles of various phases of a compiler and their interdependencies.
CO4. To apply the theory of compilation, in particular, lexical analysis, syntax and semantic analysis, code generation, and optimization phases of compilation for solving real life examples.
CO5. To analyze the similarities and differences among various parsing techniques and grammar transformation techniques.
CO6. To persuade different state of the art register allocation and code optimization strategies.

**LOCQ: Lower Order Cognitive Question; IOCQ: Intermediate Order Cognitive Question; HOCQ: Higher Order Cognitive Question.*