Roll No.: _

Amrita Vishwa Vidyapeetham

Amrita School of Engineering, Coimbatore

B.Tech. First Assessment Examinations – January 2019

Sixth Semester

Computer Science and Engineering

15CSE311 Compiler Design

[Time: Two hours Maximum: 50 Marks]

CO	Course Outcomes
CO01	Understand modularization and apply theoretical concepts to compiler construction
CO02	Apply algorithms for analyzing lexemes and structural correctness of source programs
CO03	Analyze the design of type systems and check the correctness of code semantics
CO04	Experiment intermediate representation and perform code control flow checking
CO05	Analyze function executions using activation records

Answer all questions

1.

a. Show the input and output of every phase of the compiler as a flow chart for the code snippet **a ← a* 3 – b + c** [5] [CO01]

b. The lexical (scanner), syntactic (parser), and semantic-analysis phases of a compiler process the source program unambiguously. For each of the following statements, specify which phase of the compiler will verify that a program conforms to the language specification and justify the choice of the phase. If a check could be done equally well in more than one phase of the compiler, briefly discuss the trade-offs between the alternative implementations. [4] [CO01]

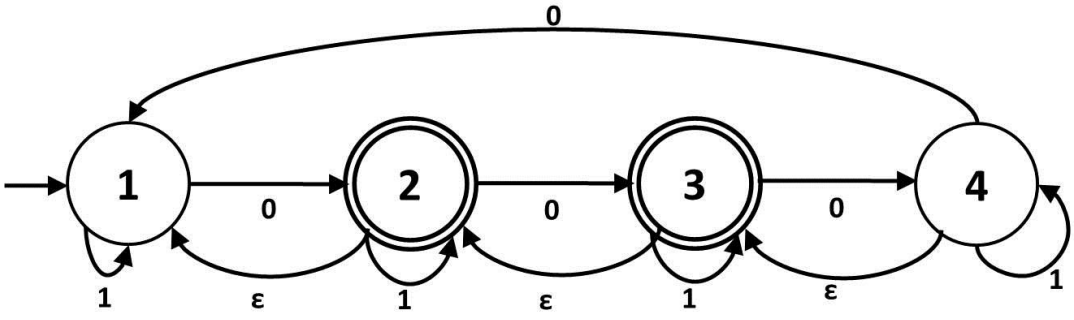
i. A function is called with the correct number of arguments.
ii. Underscore characters (_) may appear in the middle of identifiers, but not at the beginning or end (i.e., this_identifier is legal, but _this_one is not).
iii. Every variable must be declared before it is used in the program (the classic C or Pascal rule).
iv. Assignment statements must end with a semicolon (;)

2. Write a regular expression to match a number in scientific notation. The mantissa can be an integer or floating point number with optional integer part. The exponent part is optional. [2] [CO01]

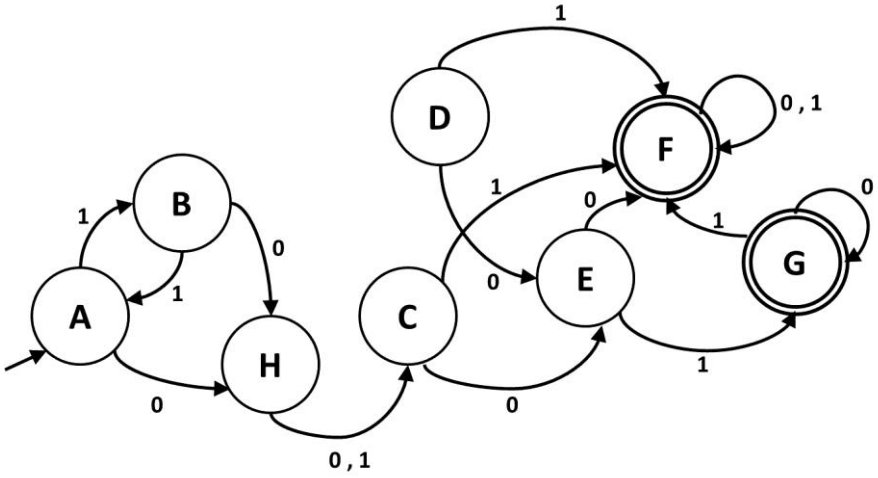
3. For the regular expression given below which accepts hexadecimal or octal digits, draw the NFA using Thompson's construction: [4] [CO01]

(0 [1…7] ([0…7] | ε)*) | (0X [1…9A…F] ([0…9A…F] | ε)*)

4. Convert the below NFA into a DFA using subset construction algorithm: [6] [CO01]



5. Minimize the following DFA: [5] [CO01]



6. Given the grammar G with the following productions (P): [2] [CO02]

assignment → qname := expr ;
qname → id
qname → id . qname
expr → qname | number | expr op expr
op → + | - | * | /

Perform a left-most derivation of the program segment ***id . id := number + number ;*** using assignment as the start symbol. Show all intermediate steps. Give two reasons as to why the above grammar is not LL(1).

7. Consider the following grammar with terminals - (the negation operator) and *int*. [5] [CO02]

S → E
E → E - E | - - E | *int*

Draw **<u>all</u>** possible parse trees for the string ***int - - - int – int***

8. Examine and justify whether the following context-free grammar which describes C-style function declarations involving pointers is LL(1). If it is not LL(1), perform left recursion elimination and left factoring to make the grammar suitable for predictive parsing. [5] [CO02]

Function → *Type* **id (***Arguments***)**

Type → **id**
Type → *Type******
Arguments → *ArgList*
Arguments → ε
ArgList → *Type* **id ,** *ArgList*
ArgList → *Type* **id**

Where Function, Type, Arguments, ArgList *are non-terminals and* id ( ) * , $ are terminals with $ as the end-of-input marker. This grammar could generate declarations like:

id* id()
id** id(id*** id)
id id(id id, id* id)

9. Given a CFG with G = (N={S, A, B, C, D}, T= {a,b,c,d}, P, S) where the set of productions P are: [8] [CO02]
   S → A
   A → BC | DBC
   B → Bb | ε
   C → c | ε
   D → a | d
   Construct a predictive (LL(1)) parse table and show the stack contents, the input and the actions performed during the parsing of the input string w = dbb

10. Write the pseudo code of a recursive decent parser for the following grammar: [4] [CO02]
    S → LS **|** ε
    L → ↑ L′
    L′ → L ↓ **|** ↓
    Show the tracing of the input ↑↑↓↓ on the parser.

****