Additional Questions
Syntax Directed Translation - Semantic Analysis – Code Generation

1- Consider the following grammar and suppose that the type of each identifier is a subrange of integers.

P → D **;** E
D → D **;** D | **id :** T
T → **num1 .. num2**
E → E + Term | Term
Term → Term **div** F | F
F → **id**

a- Construct a syntax-directed translation schema that assigns to each subexpression, the lower and upper bounds of the subrange its value must lie within.
b- Using the constructed syntax-directed translation schema give an annotated parse tree for the following program fragment, showing the lower and upper bounds of each subexpression.
J: 1 .. 10 ; K : 100 .. 200;
K div J

2 – Construct a syntax-directed translation scheme that converts arithmetic expressions containing {+,*,(,),num} to prefix notation with parentheses removed. Let the output of the translation be attached to the root of the parse tree and be sure to preserve the usual precedence and associativity of arithmetic operators.

3 – For the following grammar :

E → E + T | T
T → intconst | realconst

a- Construct a syntax-directed translation scheme to determine types as well as translating expressions into prefix notation. Assume that the values of **intconst** and **realconst** are given by the lexical analyzer.
b- Eliminate left recursion in the syntax – directed translation scheme obtained for the grammar in (a).

4 – Consider the following grammar for generating binary numbers :

S → L **.** L | L
L → L B | B
B → 0 | 1

a- Give a syntax-directed definition to compute the decimal value of each binary number generated.
b- Give an annotated parse tree for the input **101.101**

GOOD LUCK

5 – Consider the following simplified syntax of case statements:

```
Case expr of
    Const1 : stmt1;
    Const2 : stmt2;
    .
    .
    else stmtn
end
```

and the corresponding grammar :

case-stmt → **case** expr **of** caselist **else** stmt **end**
caselist → const **:** stmt | caselist **;** const **:** stmt

Construct a syntax-directed definition that translates case statements into three-address code of the following form (where V_i is the value of the ith constant)

```
        Code to evaluate expr into a temporary t
        If t < > V1 goto L1
        Code for stmt1
        Goto Lnext
L1:     If t < > V2 goto L2
        Code for stmt2
        Goto Lnext
L2:         :
            :
Ln-1:   Code for stmtn
Lnext:
```

Note: Assume that the next attribute of the entire case-stmt has been set to Lnext and that nested case statements are not allowed

GOOD LUCK