FILS

Course: Compiler Techniques

Homework #2

Attribute Grammars

1. **Float-point numbers**

1.1. Consider the following grammar, that represents numbers in the form of integers (eg. 123), reals (eg. 32,6), and numbers in exponent format (eg. 27.6E7).

Note: '^' indicates 'to the power of'.

Note: single quotes indicate a string literal.

```
Num -> SNum            { Num.val = SNum.val }
Num -> CNum            { Num.val = CNum.val }
SNum -> Int            { SNum.val = Int.val }
SNum -> Float          { SNum.val = Float.val }
Float -> Int1 '.' Int2 { Float.val = Int1.val + Int2.val/10^Int2.len }
CNum : - SNum 'E' Int  { CNum .val = SNum.val * 10 ^ Int.val }
Int : - digit          { Int.val = digit.val; Int.len= 1 }
Int -> Int1 digit      { Int.val =Int1.val*10+digit.val;
                         Int.len = Int1.len +1 }
```

Show the parse tree for "7.35E12"

1.2. Draw arrows indicating the dependency of the attributes of the nodes of the tree.

1.3. Using the above grammar of attributes and the input string "72.5E3", construct the derivation tree and calculate the value of all the attributes of the tree.

2.

Consider the following attribute grammar:

Grammar Rule	**Semantic Rules**
S -> A B C	B.u = S.u A.u = B.v + C.v S.v = A.v
A -> a	A.v = 2 * A.u
B -> b	B.v = B.u
C ->c	C.v = 1

2.1 Draw the parse tree for the input '*abc*', (the only string for the language), and show the dependency graph for the associated attributes. Describe one correct order for the evaluation of the attributes.

2.2 Assume that S.u is assigned the value of 3 before starting attribute evaluation, What will be the value of S.v when evaluation has terminated.

2.3 Suppose that the attribute rules were instead:

Grammar Rule	**Semantic Rules**
S -> A B C	B.u = S.u C.u = A.v A.u = B.v + C.v S.v = A.v
A -> a	A.v = 2 * A.u
B -> b	B.v = B.u
C ->c	C.v = C.u - 2

What value does S.v have after evauation of all attributes
if the initial value of S.u is 3 ?
Explain why this result occurred.

3. **Well-balanced parenthesis**

3.1 Write a context-free grammar that generates a well-balanced sequence of parenthesis like:
(), ()(()), ((()) ((() ())))

3.2 Attribute the previous grammar to calculate the maximum depth of parenthesis. For instance the string ((()) ((() ()))) has the depth 4.

3.3 Draw the parse tree and calculate attributes for:
((()) ((() ())))