Semantic Analysis Questions

1. Given the following grammar for unsigned numbers:

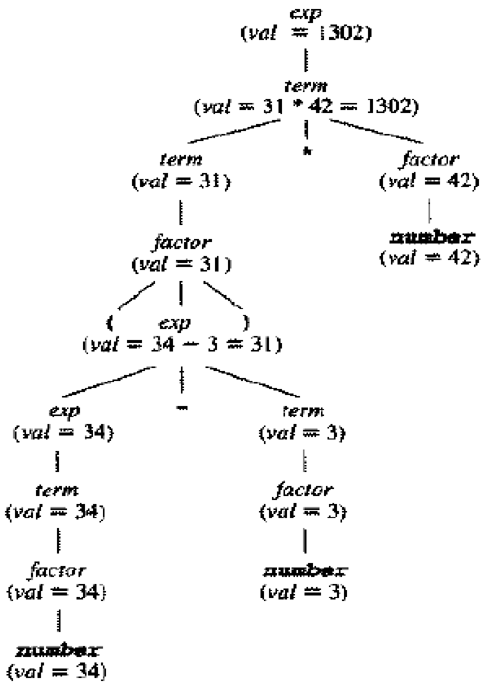
   (Compiler construction.Kenneth Louden (1997).

exp → exp + term | exp - term | term
term → term * factor | factor
factor → (exp) | number

(a) Write an attribute grammar for value of exp.

GRAMMAR RULE	SEMANTIC RULES
$exp_1 \rightarrow exp_2 + term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2 - term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2 * factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow (exp)$	$factor.val = exp.val$
$factor \rightarrow$ **number**	$factor.val =$ **number**$.val$

(b) Draw a parse tree for the **(34-3)*42**and show the attribute calculation performed at each node.

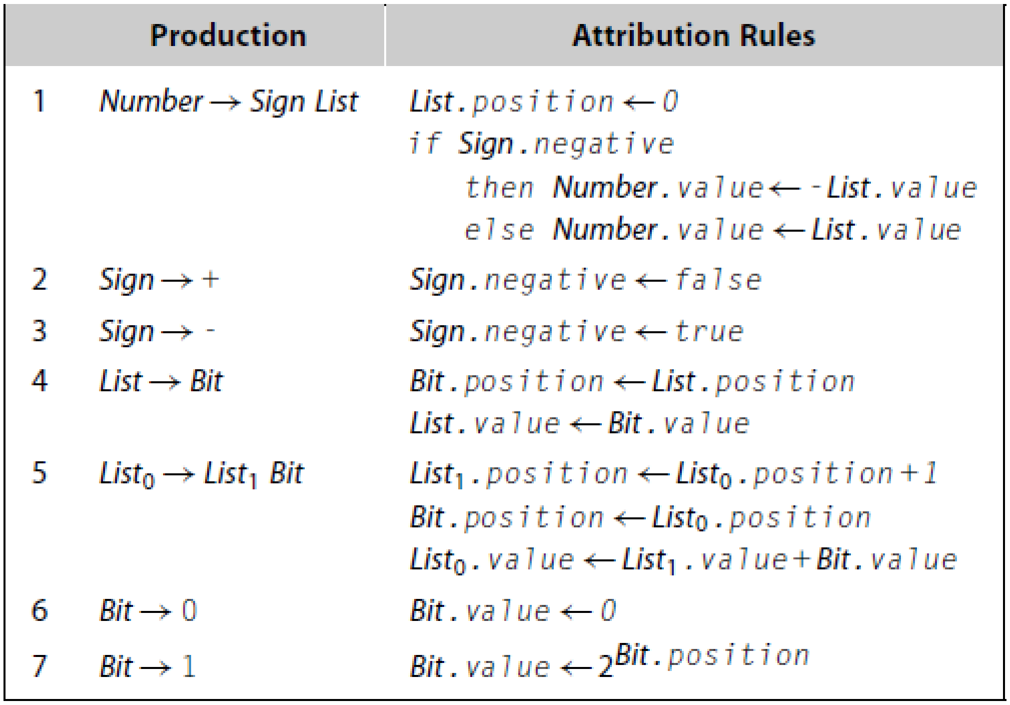


2. Given the following grammar for unsigned numbers:

(Engineering a Compiler)

Number → Sign List
Sign → + | –
List → List Bit | Bit
Bit → 0|1

a. Write an attribute grammar for position ,sign and value of a *negative Binary Number*.

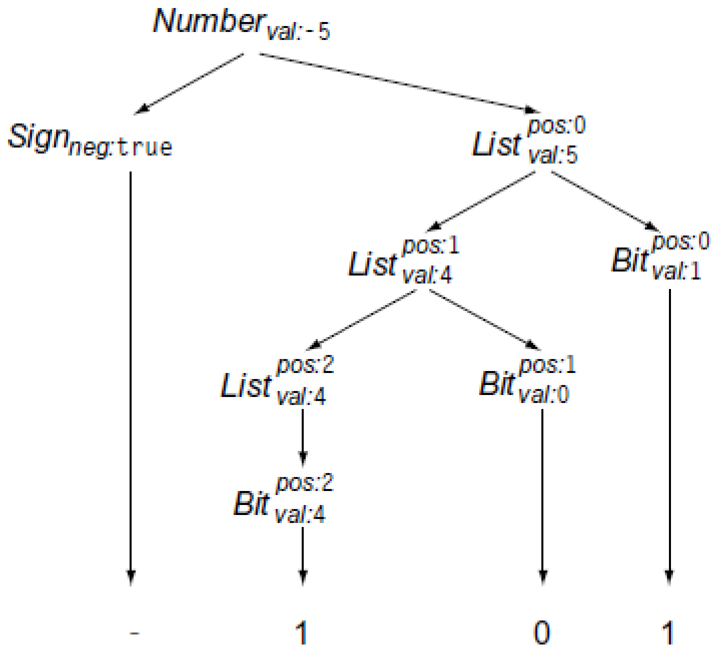


	Production	Attribution Rules

1	*Number* → *Sign List*	*List*.position ← 0
		if *Sign*.negative
		then *Number*.value ← - *List*.value
		else *Number*.value ← *List*.value

2	*Sign* → +	*Sign*.negative ← false
3	*Sign* → -	*Sign*.negative ← true
4	*List* → *Bit*	*Bit*.position ← *List*.position *List*.value ← *Bit*.value
5	$List_0$ → $List_1$ *Bit*	$List_1$.position ← $List_0$.position + 1 *Bit*.position ← $List_0$.position $List_0$.value ← $List_1$.value + *Bit*.value
6	*Bit* → 0	*Bit*.value ← 0
7	*Bit* → 1	*Bit*.value ← $2^{Bit.position}$

b. Draw a parse tree for the number **-101** and show the attribute calculation performed at each node.



Code Generation

1. **Give the sequence of three address code instructions corresponding to the given Expressions:**
    1. **x+y*z**
    2. **a = b * c + b * d**
    3. **if (a < b + c)**

```
if (a < b + c)
        a = a - c
c = b * c
```

Solution

1.

```
t1=y*z
t2=x+t1
```

2.

```
t1 = b * c
t2 = b * d
t3 =t1 +t2
a =t3;
```

3.

```
t1 = b + c
t2 = a < t1
If false t2 Goto L0
```

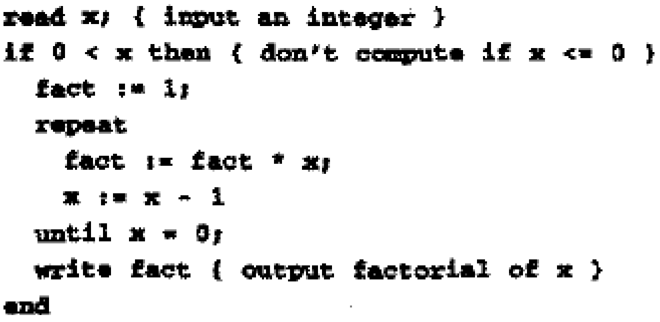
```
        t3 = a - c
        a =t3
L0:
        t4 = b * c
        c = t4
```

2. Write Three Address code for the following code fragments:

1.

```
void main()
{
        int b;
        int a;
        b = 3;
        a = 12;
        a = (b + 2)-(a*3)/6;
}
```

2.



```
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact { output factorial of x }
end
```

3.

```
int i=0;
int sum=0;
While (i<10)
{
        i++;
        sum+=i;
}
```

Solution

```
        a = 12
        b =  3
        _t3 = b + 2
        _t5 = a * 3
        _t7 = _t5 / 6
        _t8 = _t3 - _t7
        a = _t8
```

```
read x
t1=x>0
if false t1 goto L1
        fact=1;
L2:
        t2=fact*x
        fact=t2
        t3=x-1
        x=t3
        t4=x==0
        if false t4 goto L2
        write fact
L1:
        Exit
```

```
i=0;
sum=0;
L0:
        t1=i<10
        if false t1 goto L1
        t2 = i+1
        i=t2;
        t3=sum+t2
        sum=t3
        goto L0
```

```
L1:
        Exit
```