Problem 1

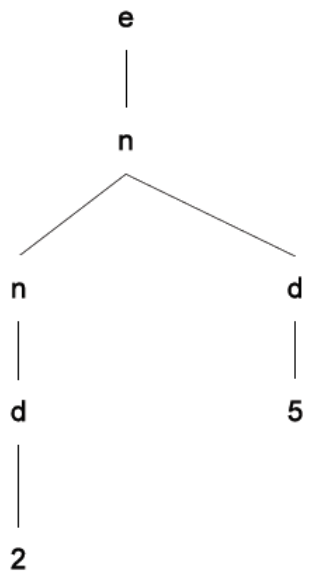
4.1 Draw the **parse tree** for the derivation of the expression 25 given in Subsection 4.1.2. Is there another derivation for 25? Is there another parse tree?

Grammar:

e ::= n | e + e | e - e
n ::= d | nd
d ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Derivation: e → n → nd → dd → 2d → 25

Another derivation: e → n → nd → n5 → d5 → 25



Problem 1

4.2 Draw parse trees for the following expressions, assuming the grammar and precedence described in Example 4.2:

a) 1 - 1 * 1

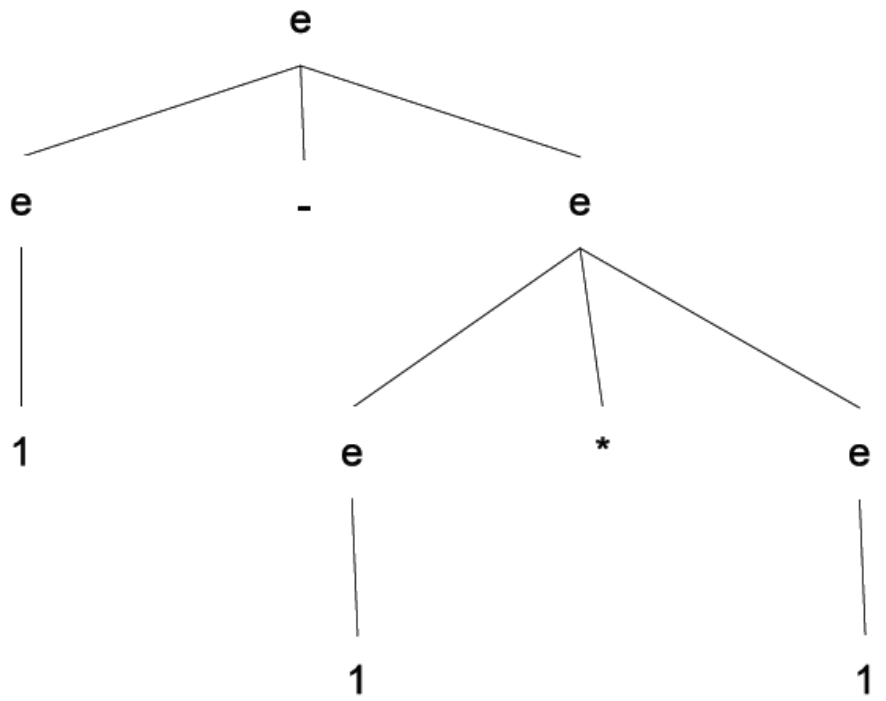
b) 1 – 1 + 1

c) 1 – 1 + 1 – 1 + 1 , if we give + higher precedence than -

Problem 1

a) 1 - 1 * 1

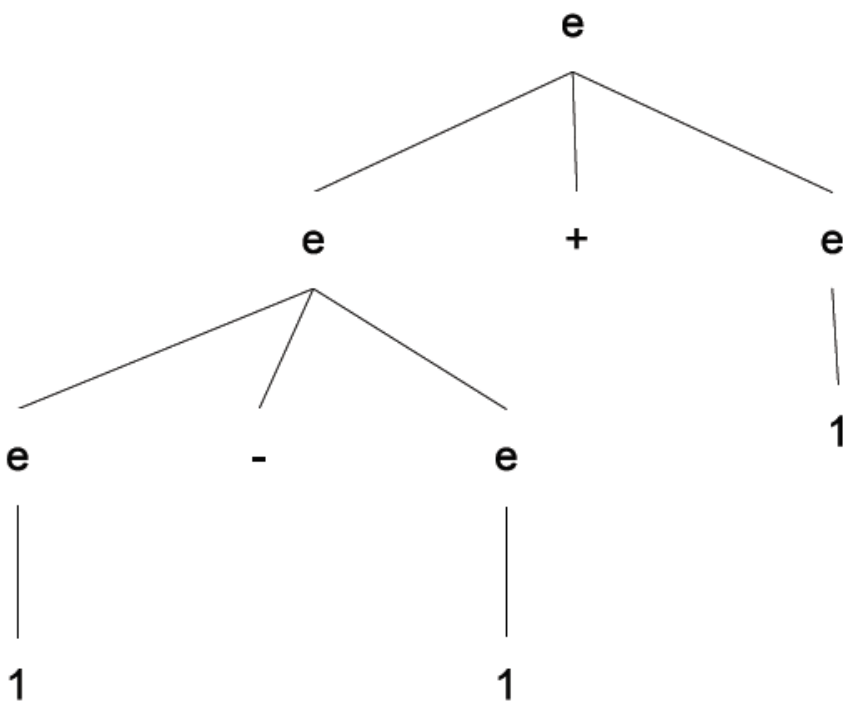
The corresponding grammar:
e ::= 0| 1| e+e | e-e | e*e



Problem 1

b) 1 – 1 + 1

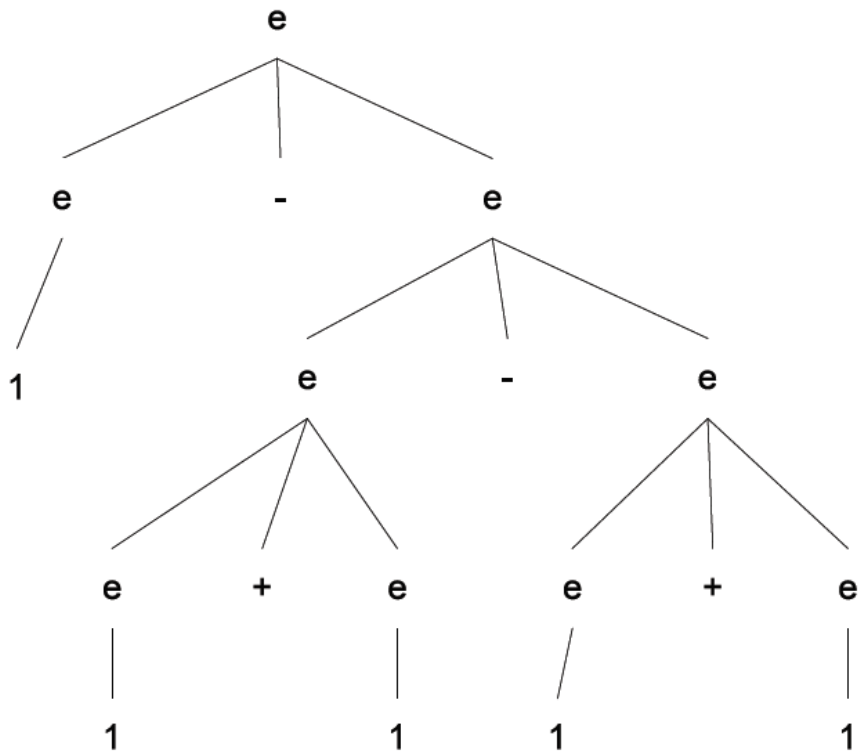
The corresponding grammar:
e ::= 0| 1| e+e | e-e | e*e



Problem 1

c) 1 – 1 + 1 – 1 + 1 , if we give + higher precedence than -

The corresponding grammar:
e ::= 0| 1| e+e | e-e | e*e



Problem 2

Make syntax diagram for the following Extended BNF

```
<program> ::= { <statement>* }
<statement> ::= <assignment> | <if-then-else> | <while>
<assignment> ::= <identifier> = <exp>
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
<while> ::= while <exp> { <statement>+ }
<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>
```

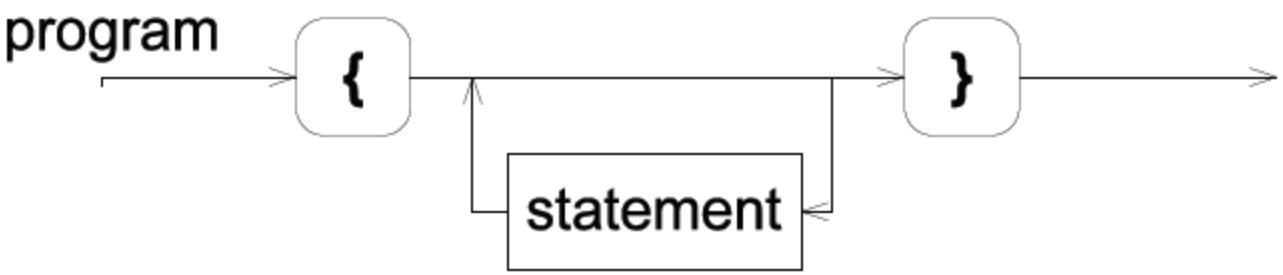
|	alternatives
?	optionality
*	zero or more times
+	one or more times

e ::= n | e + e | e - e
n ::= d | nd
d ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Problem 2

<program> ::= { <statement>* }

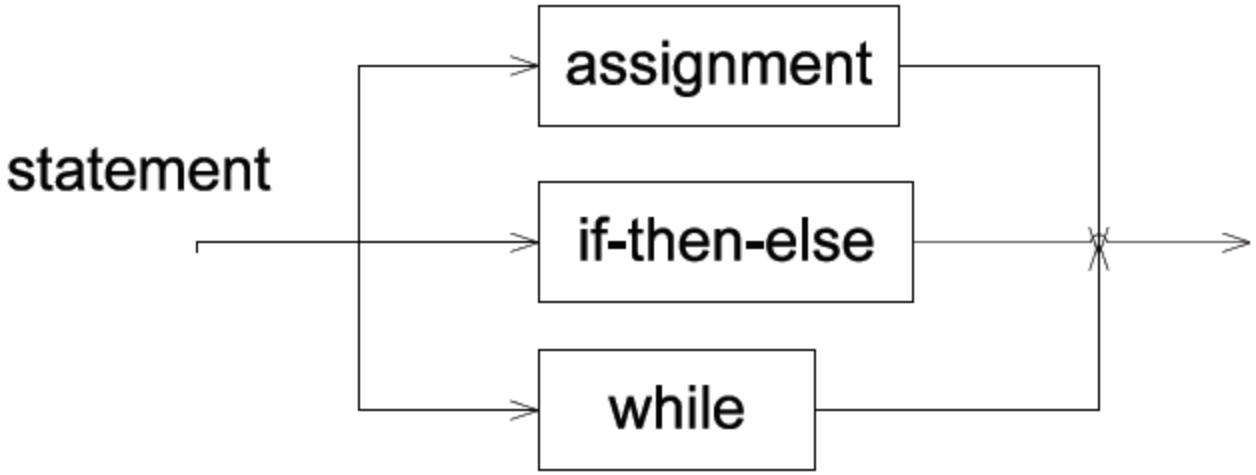
* zero or more times



Problem 2

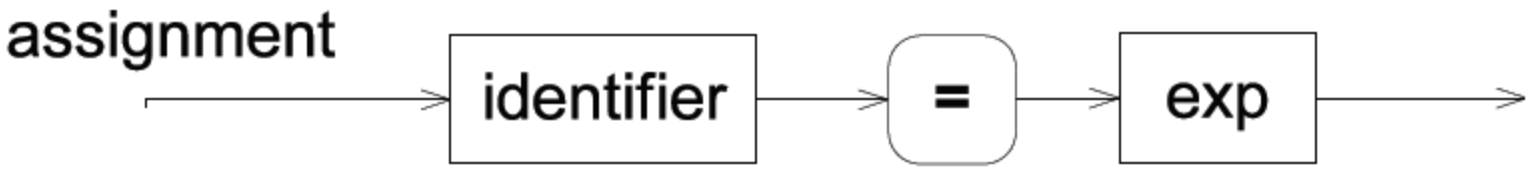
```
<statement> ::= <assignment> | <if-then-else> | <while>
```

| alternatives



Problem 2

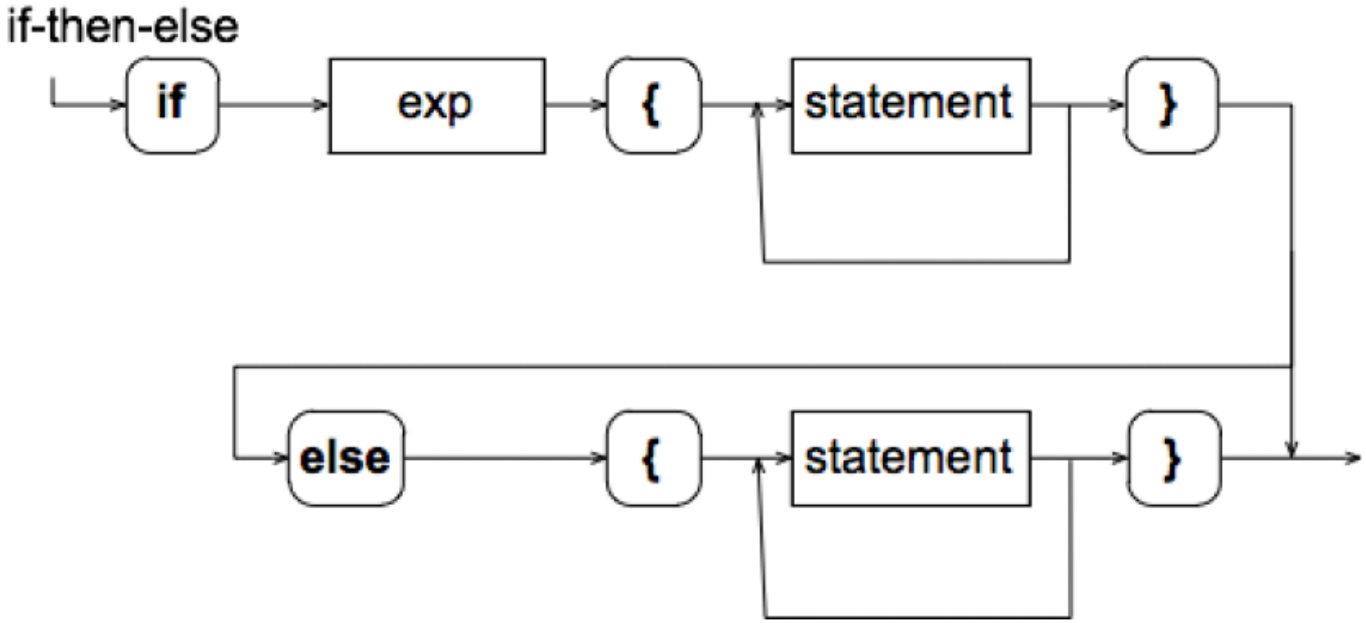
<assignment> ::= <identifier> = <exp>



Problem 2

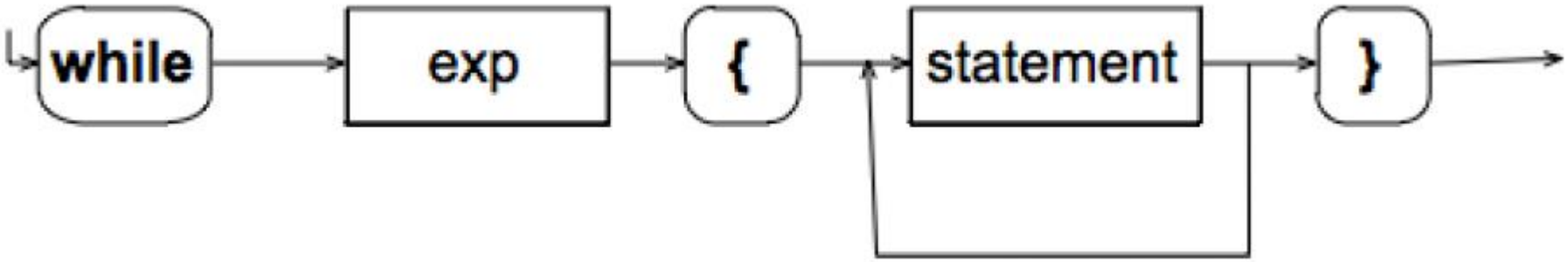
```
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
```

\+ one or more times



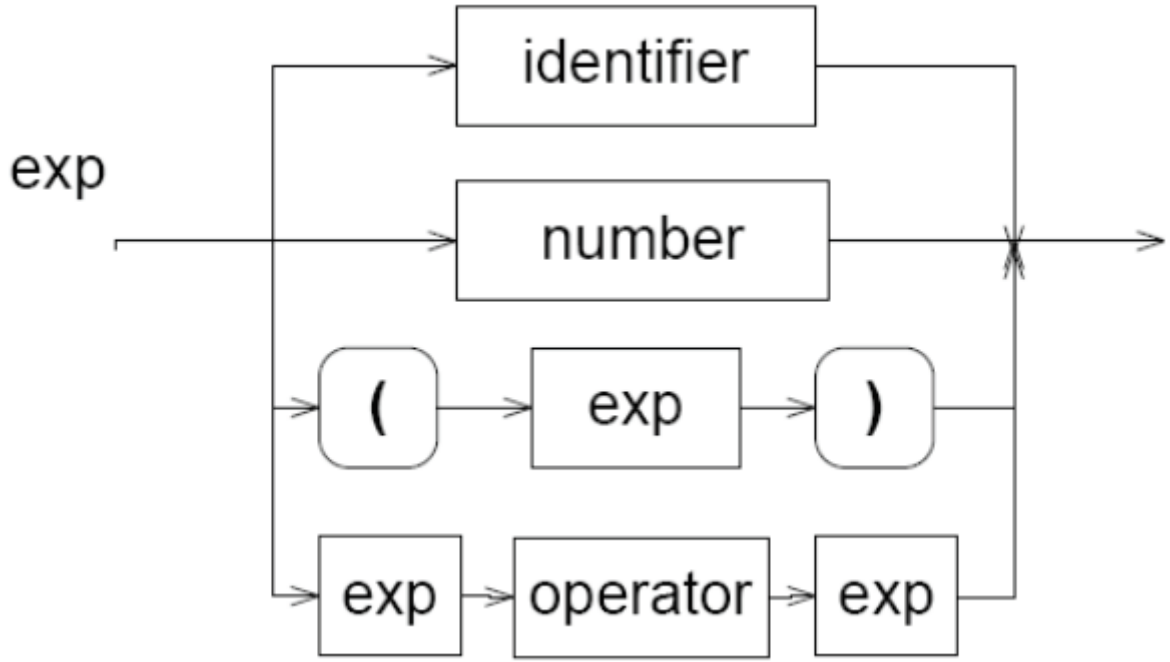
Problem 2

<while> ::= **while** <exp> **{** <statement>+ **}**



Problem 2

<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>



Problem 3

Make the parse tree (according to the grammar in Problem 2) for

```
{ i=1
  while i<10 {
    i=i+1
    j=j+i
  }
}
```

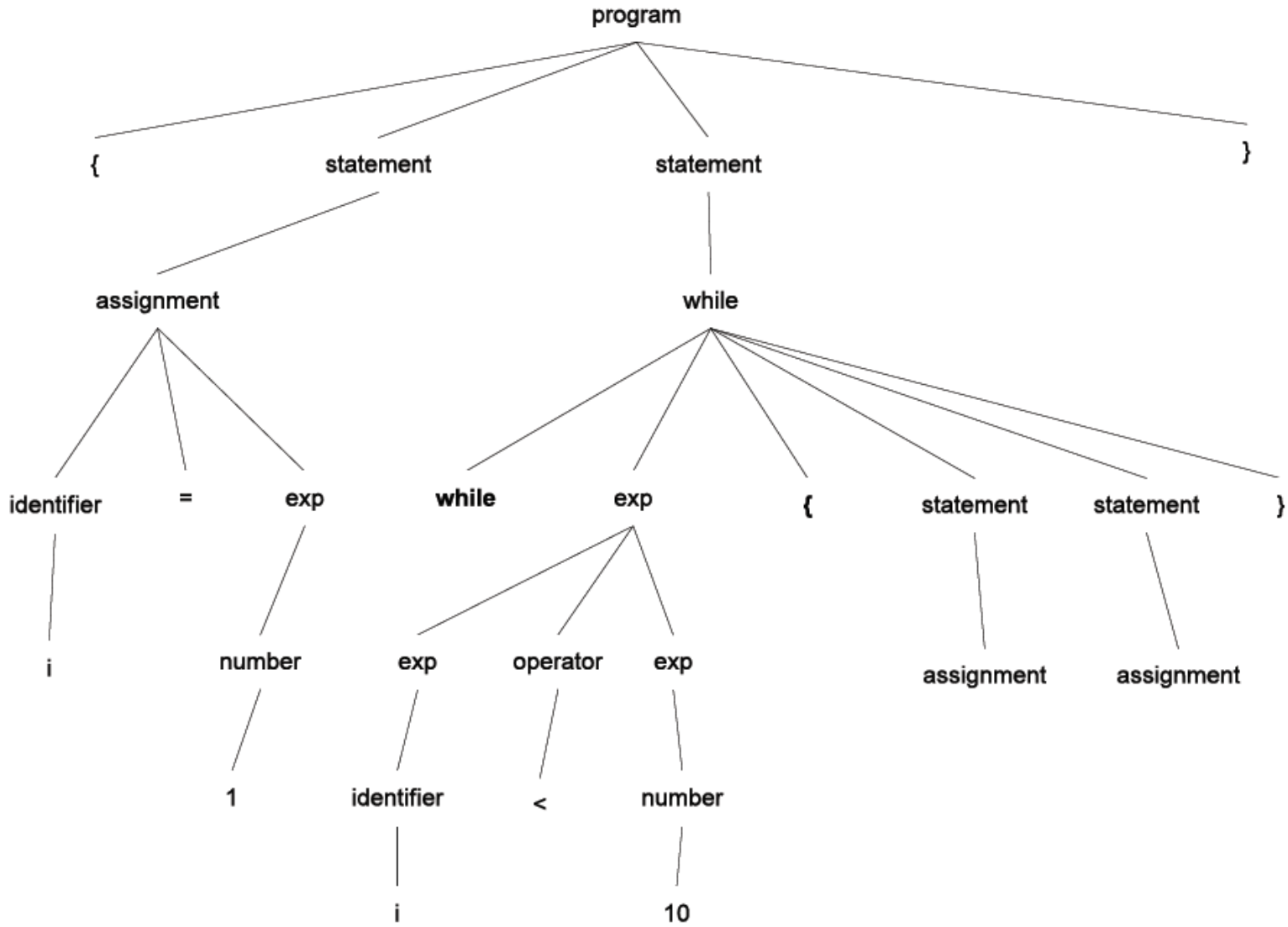
Assume that <operator> may be one of the ordinary operators and that 'i' og 'j' are legal <identifier>s.

Propose a corresponding abstract syntax tree.

```
{ i=1
  while i<10 {
    i=i+1
    j=j+i
  }
}
```

```
<program> ::= { <statement>* }
<statement> ::= <assignment> | <if-then-else> | <while>
<assignment> ::= <identifier> = <exp>
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
<while> ::= while <exp> { <statement>+ }
<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>
```

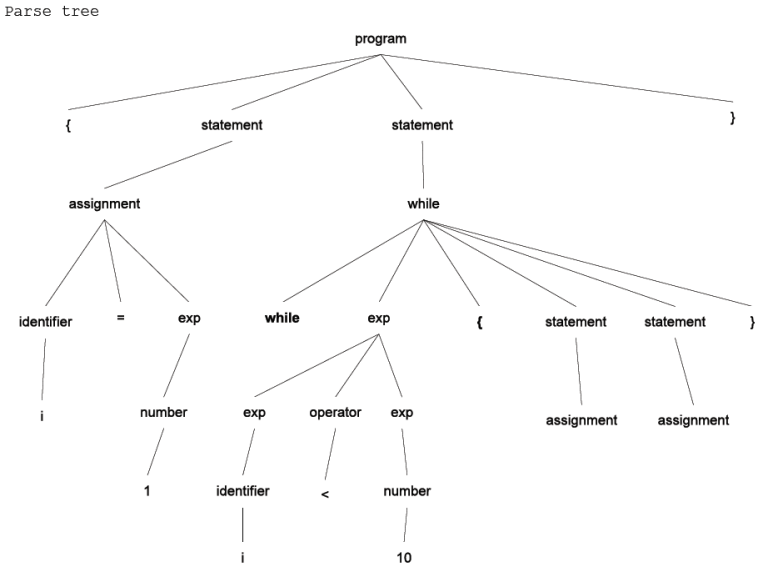
Parse tree



Difference of Parse Tree and Abstract Syntax Tree

- A parse tree is a *concrete* syntax tree.
- The parse tree contains all the artifacts (parenthesis, brackets, etc.) from the original document you parse
- An AST describes the source code conceptually
  - doesn't need to contain all the syntactical elements required to parse (curly braces, keywords, parenthesis etc.).
  - It only contains all 'useful' elements that will be used for further processing

```
{ i=1
  while i<10 {
    i=i+1
    j=j+i
  }
}
```



Problem 4

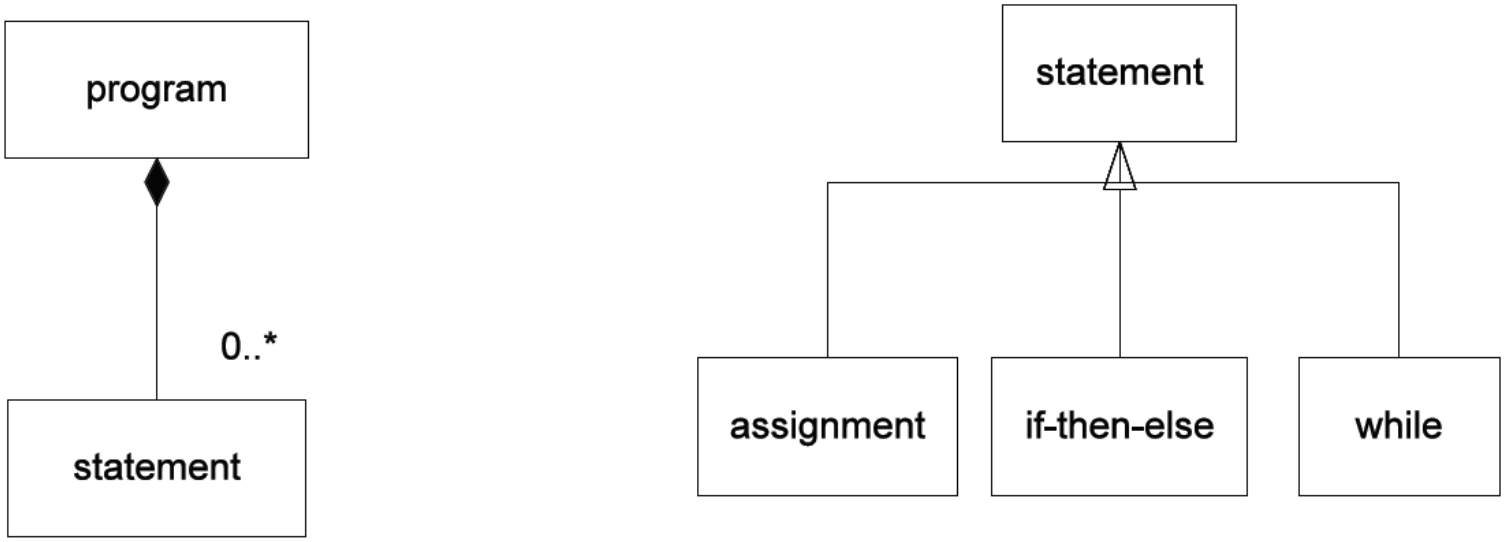
a) Make a meta model corresponding to the grammar in Problem 2.

Hint: Think of how an abstract syntax tree would be represented by means of a structure of objects, and make the class model that reflects this.

Note that there is not one and only one correct meta model, so just make one that you think may represent programs in this language.

```
<program> ::= { <statement>* }
<statement> ::= <assignment> | <if-then-else> | <while>
<assignment> ::= <identifier> = <exp>
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
<while> ::= while <exp> { <statement>+ }
<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>
```

a) Metamodel corresponding to the grammar in Problem 2.

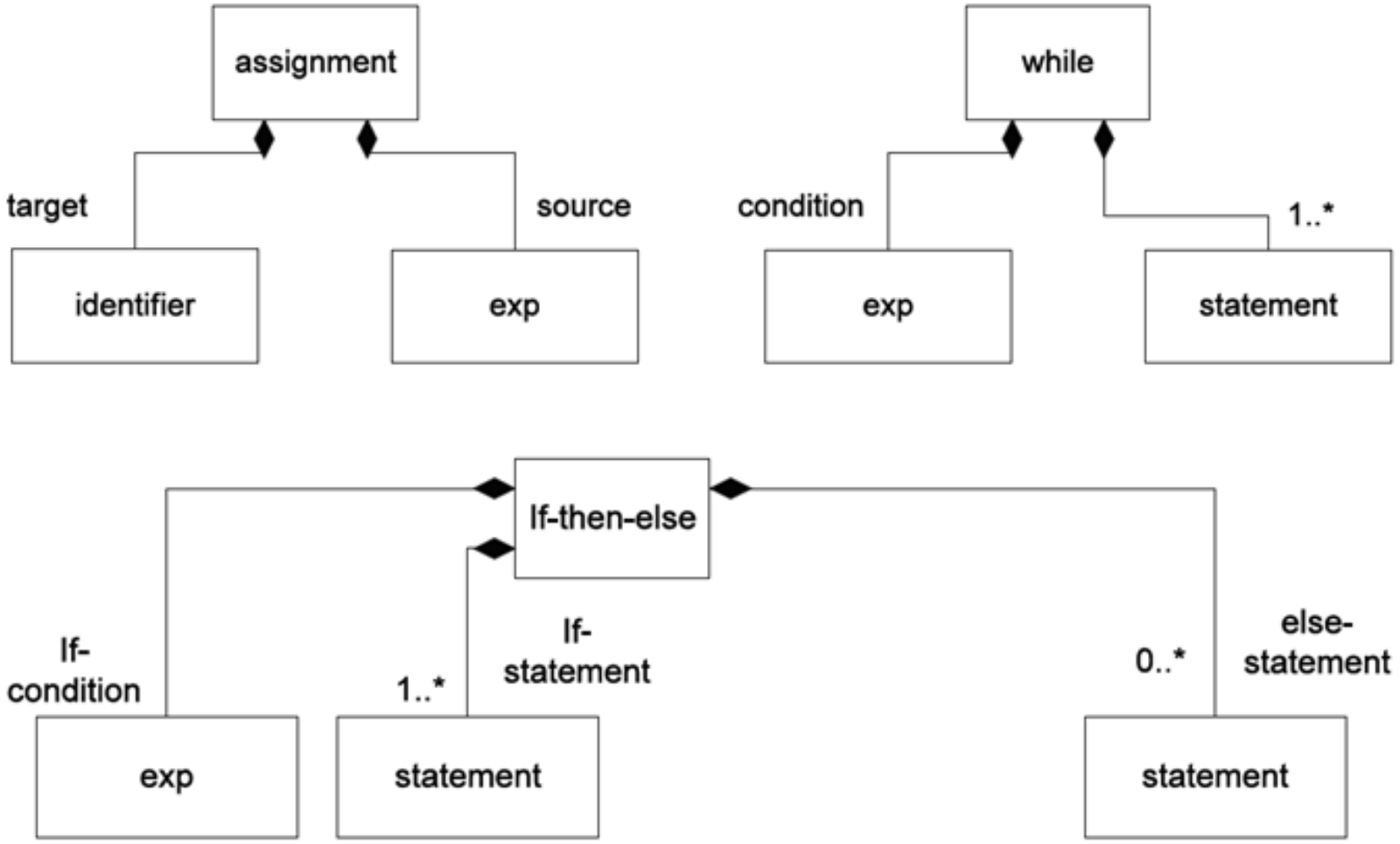


To be more precise we should really model the statements as a list of statements and not just as a set of statements, so in this metamodel we assume that compositions are made by means of lists.

Generalization: “Is-a” relationship;

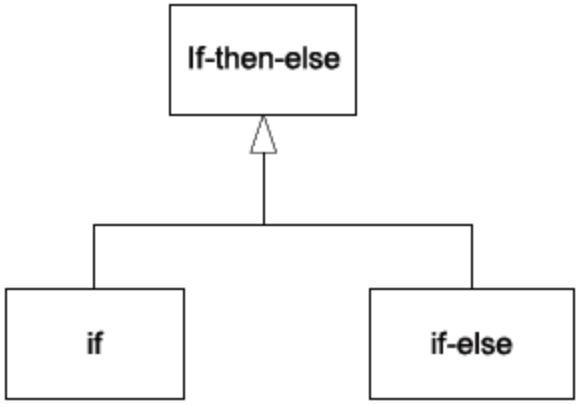
Composition: “Has-a” relationship

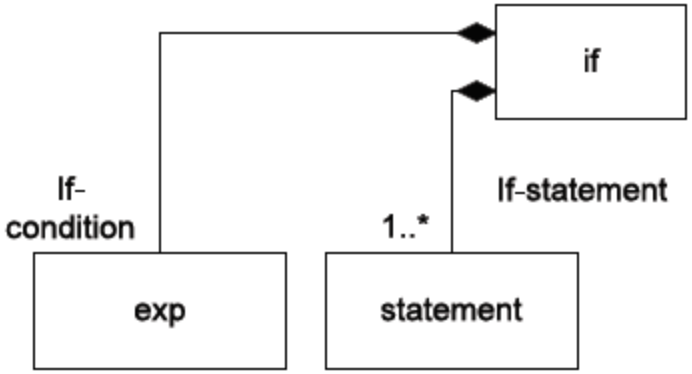
```
<program> ::= { <statement>* }
<statement> ::= <assignment> | <if-then-else> | <while>
<assignment> ::= <identifier> = <exp>
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
<while> ::= while <exp> { <statement>+ }
<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>
```

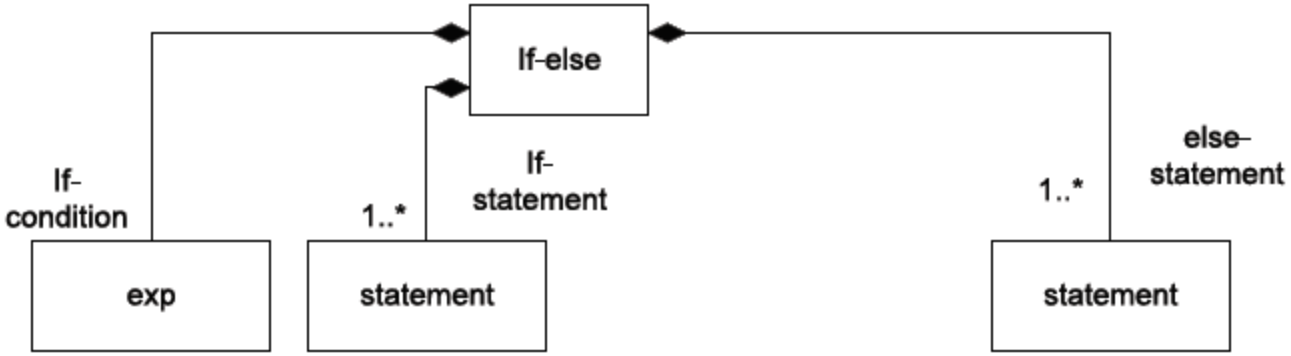


Alternative if-then-else:

```
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
```







Problem 4

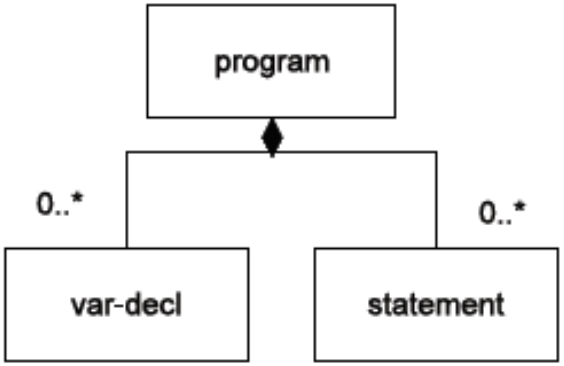
b) Assume that the grammar is extended, so that <program> starts with a number of declarations of variables, each with a name and a type. Assume further that the meta model therefore will have a class 'type' that represent the common properties of all types in the language. Modify the meta model so that programs may now have declarations and so that <identifier> is modeled by a link to the object representing the declaration of the variable with this identifier.

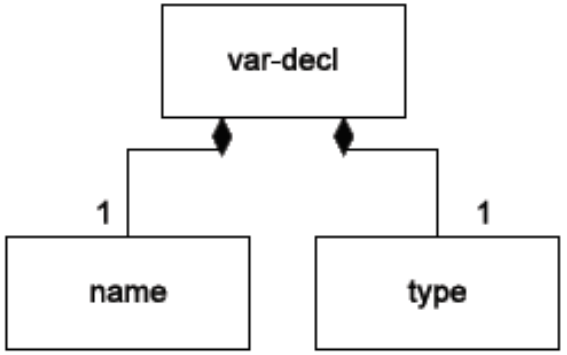
```
<program> ::= { <statement>* }
<statement> ::= <assignment> | <if-then-else> | <while>
<assignment> ::= <identifier> = <exp>
<if-then-else> ::= if <exp> { < statement>+ } |
                   if <exp> { < statement>+ } else {< statement>+ }
<while> ::= while <exp> { <statement>+ }
<exp> ::= <identifier> | <number>| (<exp>) | <exp> <operator> <exp>
```

The grammar is changed like this:

```
<program> ::= { <var-decl>* <statement>* }
<var-decl> ::= <name> <type>
```

The metamodel is changed accordingly, by adding var-decl to program and define var-decl to have a name and a type:

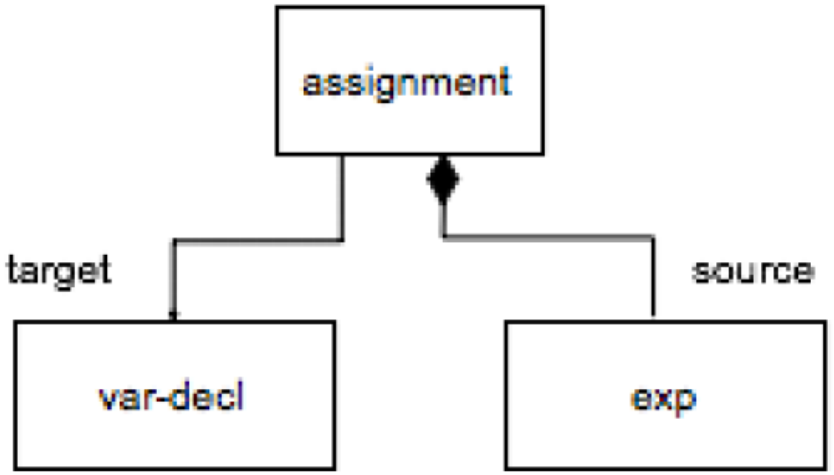




We assume that all types will be modeled by subclasses of class 'type'.

We assume that all types will be modeled by subclasses of class 'type'.

We also have to change the metamodel in places where we have 'identifier' so that it now links to the corresponding var-decl, e.g.:



That is, an assignment consists of a link to the target variable and an expression.

Here is an example with a declaration of variable 'i':

```
{ int i;
  i=1
  while i<10 {
     i=i+1     j=j+i
  }
}
```

Here is part of the corresponding object model representing the program according to the metamodel above: