(1)

$S \rightarrow AT$

$A \rightarrow aAa \mid bAb \mid \#T$

$T \rightarrow aT \mid bT \mid \varepsilon$

a) Give the First and Follow sets for each grammar variable:

$First(S) = \{a, b, \#\}$

$First(A) = \{a, b, \#\}$

$First(T) = \{a, b\}$

↓

primeiros terminais atingidos

$Follow(S) = \{\}$

$Follow(A) = \{a, b\}$

$Follow(T) = \{a, b\}$

↓

os terminais que estão imediatamente a seguir / à direita.

$A \rightarrow aAa \mid bAb \mid \#T$

em $T \rightarrow aT \mid bT \mid \varepsilon$, T não tem follow, mas no nível anterior temos $A \rightarrow \#T$, ou seja, podemos substituir o A por #T, Assim, T conterá os followers de A

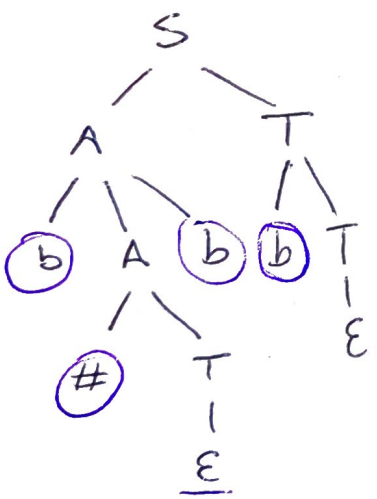
b) Show the table for the parser LL(1)

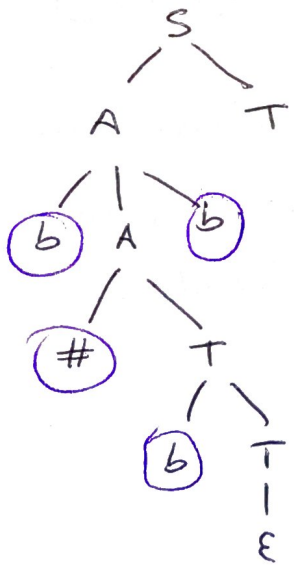
	a	b	#
S	S → AT	S → AT	S → AT
A	A → aAa	A → bAb	A → #T
T	T → aT T → ε	T → bT T → ε	

se T for ε conseguimos consumir na mesma o terminal a pois no nivel a cima há uma produção (A → aAa) que consome esse terminal logo à primeira

c) Indicate the possible problems this grammar may have and that need to be solved in order it can be implemented as a top-down recursive

A gramática é ambigua:





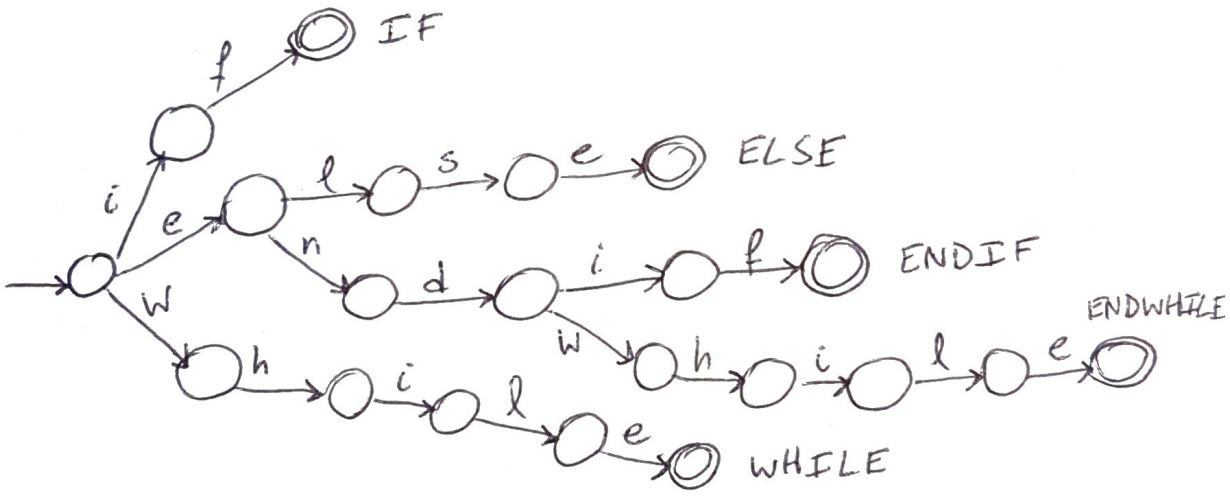
Considerando a entrada b#bb
é possivel gerar duas árvores (há mais do que um caminho possivel)

(2)

a) Considering that OP represents the arithmetic operations, -, +, *, /, and CMP the comparison operations, !=, ==, >, <, >=, <=, show the definitions of these tokens as regular expressions.

OP = - | + | * | /
CMP = != | == | > | < | >= | <=

b) Show the DFA for the tokens WHILE, ENDWHILE, IF, ELSE and ENDIF.



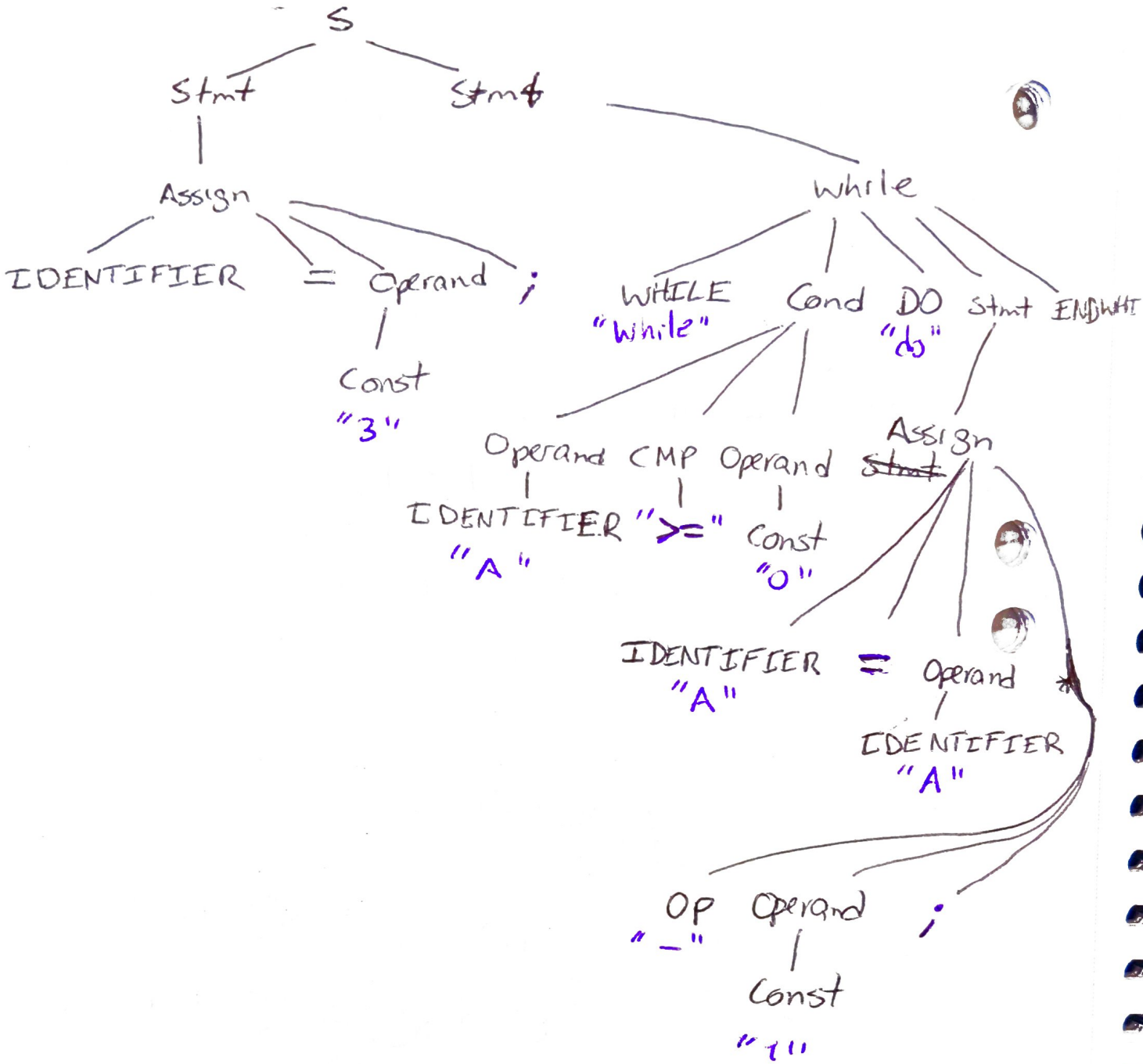
c) Show the concrete syntax trees for the code example below and considering the depicted CFG.

Code Example:

```
A = 3;
while A >= 0 do
    A = A-1;
endWhile
```

Grammar G1:

S → (Stmt)*
Stmt → While | Assign | If
While → While Cond Do (Stmt)* ENDWHILE
Assign → IDENTIFIER = Operand (OP Operand)?;
Operand → IDENTIFIER | CONST
If → IF Cond THEN Stmt (ELSE Stmt)? ENDIF
Cond → Operand CMP Operand


S
Stmt
Stmt
Assign
while
IDENTIFIER
=
Operand
;
Const
"3"
WHILE
"while"
Cond
DO
"do"
Stmt
ENDWHI
Operand
CMP
Operand
Assign
Stmt
IDENTIFIER
"A"
">="
Const
"0"
IDENTIFIER
"A"
=
Operand
IDENTIFIER
"A"
Op
"-"
Operand
Const
"1"
;

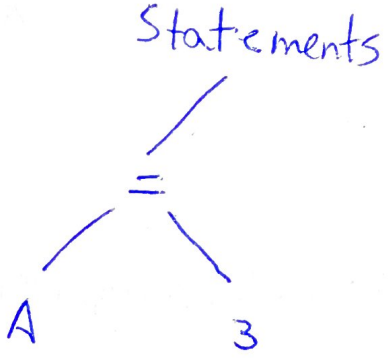
COMP

(2)

d) show a possible abstract syntax tree (AST²) for the concrete syntax tree of the previous question.

Tirar os nós da árvore concreta que não são necessários.

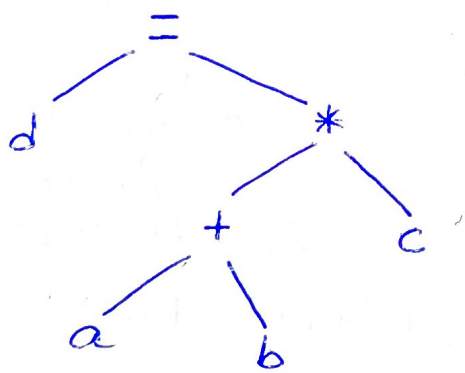
```
A = 3;
While . A >= 0 do
    A = A - 1;
endWhile
```



Temos de manter a ordem/sequência de instruções do código.

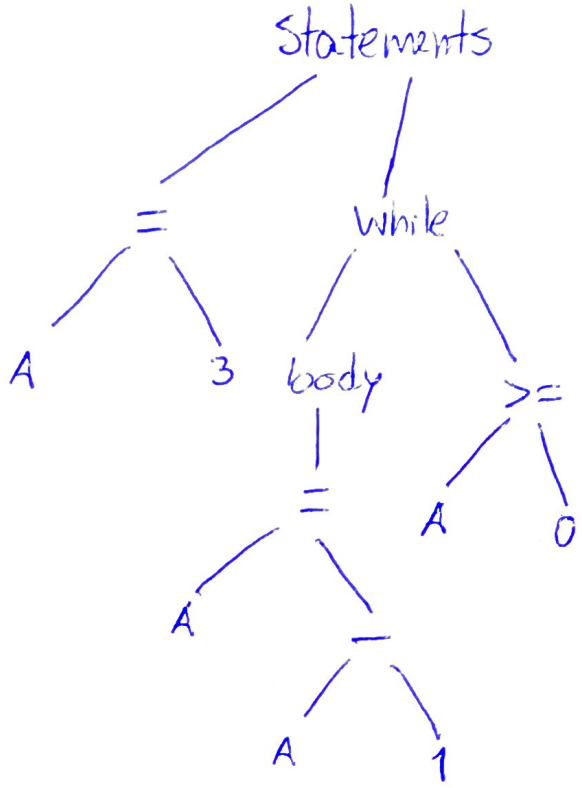
Exemplo com `d = (a+b) * c;`

AST



⚠ Ao contrário da CST (concrete syntax tree), na AST não precisamos de representar os parêntesis e o ponto e vírgula.

```
A = 3;
while A >= 0 do
    A = A - 1;
endwhile
```



(2) e) Stmt → While | Assign | If

```
bool Stmt() {
    if(token == WHILE)
        return while();
    if ( token == IDETIFIER)
        return Assign();
    return If();
}
```

Não fazer:

```
if(while()) return true;
if(Assign()) return true;
return If();
```

pois se entrar no primeiro while() pode falhar essa função, mas ao entrar no Assign() este pode retornar true, apesar de o input inicial estar errado

While → WHILE Cond Do (Stmt)* ENDWHILE

```
bool while() {
  if (token == WHILE){
    next();
    if(!Cond()) return false;
    if(token == Do) {
      next();
    } return false;
  } return false(); }
```

(*) → ver página seguinte

(*)

```
while ( token != ENDWHILE ){
    if(!stmt()) return false;
}
next(); // token a seguir ao ENDWHILE
return true;
```

f) Suppose we want to modify the grammar in order to make possible to input values from the keyboard and print values of variable to the screen.
Present the grammar modifications you suggest.

stmt → While | Assign | If | Read | Write

Read → cin >> IDENTIFIER ;

Write → cout << IDENTIFIER ;