1.)
Grammar:
<S> ::= a <S> c <B> | <A> | b
<A> ::= c <A> | c
<B> ::= d | <A>

a) aabccd Yes,
<S>   ::= a <S> c <B>
       ::= a a <S> c c <B>
       ::= a a b c c d

b) accbcc No

<S>   ::= a <S> c <B>
       ::= a <A> c <A>
       ::= a c <A> c c ←- can't revisit <S> and get a b here, if I go back to step 1 and choose
<S> again I'll have aa.

c) accccc Yes
<S>   ::= a <S> c <B>
       ::= a <A> c <A>
       ::= a c <A> <A>
       ::= a c c <A> <A> ←- recursion!
       ::= a c c c <A> <A>
       ::= a c c c c c
2.)
<integer>   ::= <unsigned> [<sign>]
<unsigned>   ::= {<digits>} <digits>
<digits>   ::= <digit> {<digit>}
<digit>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<sign>   := + | -

3.)
a.) ab or n*a n*b the second option can recurse on itself and generate itself as much and then
terminate
b.) n*a n*b n*c, each option can recurse on itself as many times and then terminate.
c.) <x> leads to
       n*0 n*1 or n*0
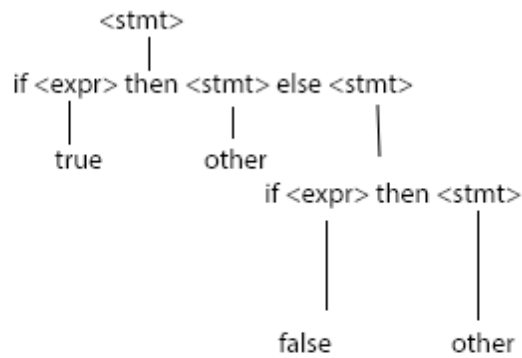   <y> leads to
       n*0 2n1 or n*1

4.)
a.)

```
              <stmt>
                |
   if <expr> then <stmt> else <stmt>
        |            |           |
      true         other        if <expr> then <stmt>
                                      |            |
                                    false        other
```

if true then other else if false then other

```
              <stmt>
                |
   if <expr> then <stmt> else <stmt>
        |            |              |
      false        if <expr> then <stmt>      other
                        |            |
                      true         other
```
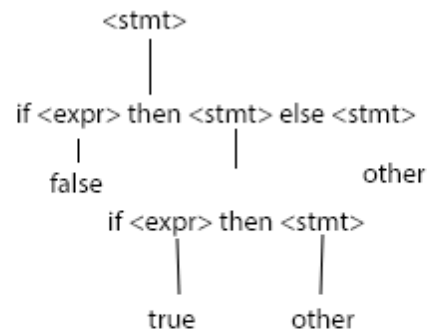
if false then if true then other else other

b.)<stmt>   ::= if <expr> then <expr> | if <expr> then <expr> else <stmt> | other
   <expr>   ::= true | false

5.)
BNF:
a.)<S>   ::= a <S> | a | empty

b.)<S>   ::= a , <S> , a | a

EBNF:
a.)<S>   ::= a [<S>] | empty

b.)<S>   ::= a [a, <S>, a]