Grammars, Functional Programming, and Scheme

Context-Free Grammar: A context-free grammar is a quadruple *(NT, T, R, S)* where:

- *NT* is a finite set of symbols (non-terminal symbols, variables, or syntactic categories)
- *T* is a finite set of terminal symbols
- *R* is a finite set of productions (or rules), each of which is composed of an expression of the form
$$V \to w \qquad (\Sigma = T \cup NT,\ w \in \Sigma^*)$$
where *V* (the head of the production) is a single non-terminal symbol and *w* (the body) is a string composed of zero or more terminal or non-terminal symbols:
- *S* is the initial symbol and is an element of *NT*

1. Give an ambiguous CFG for **`if-then-else`**. Draw a derivation tree to prove that the grammar is ambiguous. Use the non-terminal *S* as your initial symbol and *C* for the if condition. You do not need to provide the production rules for *C*.

2. Backus-Naur form (BNF) is another notation used to describe a grammar where:
    - Non-terminal symbols are written between angle brackets
    - Productions with the same head are grouped into a single block using “|” to separate productions

   Excerpt from the Scheme lexical structure in BNF:

⟨expression⟩ → ⟨variable⟩
    | ⟨literal⟩
    | ⟨procedure call⟩
    | ⟨lambda expression⟩
    | ⟨conditional⟩
⟨literal⟩ → ⟨quotation⟩ | ⟨self-evaluating⟩
⟨self-evaluating⟩ → ⟨boolean⟩ | ⟨number⟩
    | ⟨character⟩ | ⟨string⟩
⟨boolean⟩ → #t | #f

⟨number⟩ → ⟨integer⟩ | ⟨decimal⟩
⟨integer⟩ → ⟨sign⟩ ⟨digit⟩+
⟨sign⟩ → ⟨empty⟩ | + | -
⟨digit⟩ → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨conditional⟩ → (if ⟨test⟩ ⟨consequent⟩ ⟨alternate⟩)
⟨test⟩ → ⟨expression⟩
⟨consequent⟩ → ⟨expression⟩
⟨alternate⟩ → ⟨expression⟩ | ⟨empty⟩

Write out the derivation tree for `(if #t 3)`

3. Write a scheme procedure `reverse-list` that takes in a list and outputs the reversed list. *Hint*: use the builtin procedure `append`; when given all list arguments, the result is a list that contains all of the elements of the given lists in order

4. Define a **tail-recursive** fibonacci function in Scheme. *Hint:* use a helper function