1. (a) 

\[ A \rightarrow aB \mid bA \] 

(b) As the code allows success without consuming any input, this suggests an \( \epsilon \)-production:

\[ A \rightarrow aB \mid bA \mid \epsilon \]

However, the code does not quite work. If the input starts with an ‘a’, the code will consume that token and then commit to parse using \( \text{parseB} \). But if that fails, then the entire parsing fails. That is not correct, as it should have succeeded without consuming any input. Here is the corrected code:

```haskell
parseA :: [Token] -> Maybe [Token]
parseA ts@(‘a’ : ts’) =
  case parseB ts’ of
    Just ts’’ -> Just ts’’
    Nothing -> Just ts
parseA (‘b’ : ts) = parseA ts
parseA ts = Just ts
```

(Why can the case for input starting with ‘b’ be left as is?)

(c)

```haskell
parseS :: [Token] -> Maybe [Token]
parseS ts =
  case parseA ts of
    Nothing -> Nothing
    Just ts’ -> case parseB ts’ of
      Nothing -> Nothing
      Just ts’’ -> parseB ts’’
```

parseB :: [Token] -> Maybe [Token]
parseB (‘c’ : ‘c’ : ts) = parseB ts
parseB (‘c’ : ‘d’ : ts) = parseC ts
parseB _ = Nothing

parseC :: [Token] -> Maybe [Token]
parseC (‘e’ : ts) = parseC ts
parseC ts = Just ts
(d) A couple of possibilities. First one that employs limited backtracking:

```haskell
parseB :: [Token] -> Maybe [Token]
parseB ('c' : ts) =
  case parseB ts of
    Just ts' -> Just ts'
    Nothing ->
      case ts of
        ('d' : ts'') -> parseC ts''
        _ -> Nothing
  parseB _ = Nothing
```

Here is another possibility that achieves the same thing, exploiting that patterns are tried in order when pattern-matching in Haskell. The code is simpler, but arguably less clear in that the second pattern seemingly overlaps the first (unless the reader is aware that the patterns are tried in order).

```haskell
parseB :: [Token] -> Maybe [Token]
parseB ('c' : 'd' : ts) = parseC ts
parseB ('c' : ts) = parseB ts
parseB _ = Nothing
```

2. First identify the immediately left-recursive non-terminals. Then group the productions for each such non-terminal into two groups: one where each RHS starts with the non-terminal in question, and one where they don’t:

\[
\begin{align*}
A & \to A\alpha_1 \ | \ \ldots \ | \ A\alpha_m \\
A & \to \beta_1 \ | \ \ldots \ | \ \beta_n
\end{align*}
\]

Then replace those productions with new productions for \(A\) and productions for \(A'\), where \(A'\) is a new name, as follows:

\[
\begin{align*}
A & \to \beta_1 A' \ | \ \ldots \ | \ \beta_n A' \\
A' & \to \alpha_1 A' \ | \ \ldots \ | \ \alpha_m A' | \epsilon
\end{align*}
\]

There are immediately left-recursive productions for \(S\) and \(X\) in the given grammar. Grouping the productions as required and then applying the above transformation rule to the grammar yields:

\[
\begin{align*}
S & \to \ XbS'S' \ | \ aS' \\
S' & \to \ aS' \ | \ \epsilon \\
X & \to \ YYX' \ | \ YYX'X' \\
X' & \to \ XXX' \ | \ YYX' \ | \ \epsilon \\
Y & \to \ cY \ | \ dY \ | \ \epsilon
\end{align*}
\]
3. (a) $N_\epsilon = \{S, A, B\}$. $A$ is nullable because $A \rightarrow \epsilon$ is a production. $B$ is nullable because $B \rightarrow \epsilon$ is a production. $S$ is nullable because $S \rightarrow ABB$ is a production and both $A$ and $B$ are nullable. $C$ is not nullable since the RHS of all productions for $C$ include a terminal ($c$ or $d$), which means it is clear $\epsilon$ cannot be derived from $C$.

(b) Keeping in mind which non-terminals are nullable, we obtain the following equations:

\[
\text{first}(A) = \text{first}(aA) \cup \text{first}(\epsilon) = \{a\} \cup \emptyset = \{a\} \\
\text{first}(B) = \text{first}(Bb) \cup \text{first}(\epsilon) = (\text{first}(B) \cup \text{first}(b)) \cup \emptyset = \text{first}(B) \cup \text{first}(b) \\
\text{first}(C) = \text{first}(cA) \cup \text{first}(d) = \text{first}(c) \cup \text{first}(d) = \{c\} \cup \{d\} = \{c, d\}
\]

The solutions of the equations for first($A$) and first($C$) are manifest. As to the equation for first($B$), we need only observe that it has the form $X = X \cup Y$. The smallest solution to such an equation is simply $X = Y$, so first($B$) = \{b\}.

Now we can turn to setting up and solving the equation for first($S$), again keeping in mind which non-terminals are nullable:

\[
\text{first}(S) = \text{first}(ABB) \cup \text{first}(BBC) \cup \text{first}(CA) \\
= (\text{first}(A) \cup \text{first}(B) \cup \text{first}(B) \cup \text{first}(\epsilon)) \\
\quad \cup (\text{first}(B) \cup \text{first}(B) \cup \text{first}(C)) \\
\quad \cup \text{first}(C) \\
= (\text{first}(A) \cup \text{first}(B) \cup \emptyset) \\
\quad \cup (\text{first}(B) \cup \text{first}(C)) \\
\quad \cup \text{first}(C) \\
= (\{a\} \cup \{b\}) \cup (\{b\} \cup \{c, d\}) \cup \{c, d\} \\
= \{a, b, c, d\}
\]

Thus, we obtained a solution directly.

(c) Note: very detailed account below for clarity. It is sufficient to just state the constraints according to the definitions and then simplify.

Constraints for follow($S$):

\[
\{\$\} \subseteq \text{follow}(S)
\]

Constraints for follow($A$) from the productions where $A$ occurs in the RHS, i.e.

\[
S \rightarrow ABB \\
S \rightarrow CA \\
A \rightarrow aA
\]
Constraints for \( \text{follow}(B) \) from the productions where \( B \) occurs in the RHS, i.e.

\[
S \rightarrow ABB \\
S \rightarrow BBC \\
B \rightarrow Bb
\]

Constraints for \( \text{follow}(C) \) from the productions where \( C \) occurs in the RHS, i.e.

\[
S \rightarrow BBC \\
S \rightarrow CA \\
C \rightarrow cC
\]

Using

\[
\begin{align*}
\text{first}(\epsilon) & = \emptyset \\
\text{first}(A) & = \{a\} \\
\text{first}(B) & = \{b\} \\
\text{first}(C) & = \{c, d\} \\
\text{first}(BB) & = \text{first}(B) \cup \text{first}(B) \cup \text{first}(\epsilon) \\
& = \{b\} \cup \{b\} \cup \emptyset = \{b\} \\
\text{first}(BC) & = \text{first}(B) \cup \text{first}(C) \cup \emptyset \\
& = \{b\} \cup \{c, d\} = \{b, c, d\}
\end{align*}
\]

and eliminating trivial constraints yields:
\{\$\} \subseteq \text{follow}(S)

\{b\} \subseteq \text{follow}(A)
\text{follow}(S) \subseteq \text{follow}(A)

\{b\} \subseteq \text{follow}(B)
\text{follow}(S) \subseteq \text{follow}(B)
\{b, c, d\} \subseteq \text{follow}(B)

\text{follow}(S) \subseteq \text{follow}(C)
\{a\} \subseteq \text{follow}(C)

This is equivalent to

\{\$\} \subseteq \text{follow}(S)
\{b\} \cup \text{follow}(S) \subseteq \text{follow}(A)
\{b\} \cup \text{follow}(S) \cup \{b, c, d\} \subseteq \text{follow}(B)
\text{follow}(S) \cup \{a\} \subseteq \text{follow}(C)

which can be further simplified to the final constraints:

\{\$\} \subseteq \text{follow}(S)
\{b\} \cup \text{follow}(S) \subseteq \text{follow}(A)
\{b, c, d\} \cup \text{follow}(S) \subseteq \text{follow}(B)
\{a\} \cup \text{follow}(S) \subseteq \text{follow}(C)

(d) The smallest set satisfying the constraint for \text{follow}(S) is obviously just \{\$\}. Substituting this into the remaining constraints makes the smallest sets satisfying those obvious too. Thus:

\text{follow}(S) = \{\$\}
\text{follow}(A) = \{b\} \cup \{\$\} = \{b, \$\}
\text{follow}(B) = \{b, c, d\} \cup \{\$\} = \{b, c, d, \$\}
\text{follow}(C) = \{a\} \cup \{\$\} = \{a, \$\}