Consider the following grammar giving the concrete syntax of a language:

\[
\begin{align*}
E & \rightarrow id \\
C & \rightarrow E = E; \\
C & \rightarrow \{B\} \\
C & \rightarrow C \text{ repeatwhile } E \\
C & \rightarrow \text{ if } E \text{ then } C \\
C & \rightarrow \text{ if } E \text{ then } C \text{ else } C \\
B & \rightarrow B C \\
B & \rightarrow C \\
S & \rightarrow C \text{ eof}
\end{align*}
\]

where \( C \text{ repeatwhile } E \) has the same meaning as \texttt{do } C \texttt{ while } E \) in C or Java.

(a) List the terminals and non-terminals of this grammar and explain the significance of \( S \). \hfill [3 \text{ marks}]

(b) Identify any ambiguities in the above grammar and rewrite it to remove them, ensuring that your new grammar generates exactly the same set of strings. \hfill [4 \text{ marks}]

(c) Specify a suitable abstract syntax, for example by giving a type declaration in a programming language of your choice, which might be used to hold parse trees for this language. \hfill [3 \text{ marks}]

(d) Give \textit{either} a recursive descent parser \textit{or} a characteristic finite state machine (e.g. for SLR(1)) with associated parser for your grammar. Your parser need not return a parse tree—it suffices for your parser either to accept or to reject the input string. \hfill [10 \text{ marks}]
