Consider the following grammar for expressions (\(<E>\)) and commands (\(<C>\)).

\[
< E > ::= i \mid n \mid < E > - < E > \mid < E > ** < E > \mid ( < E > ) \\
\]

\[
< C > ::= i ::= < E > \\
    | if < E > then < C > \mid if < E > then < C > else < C > \\
    | < C > repeatwhile < E > \mid < C > ; < C > \mid \{ < C > \}
\]

Show that there are syntactic ambiguities between (a) the minus (\(-\)) and exponentiation (\(**\)) operators, (b) the \(\text{if}\)-command and the \(\text{if-then-else}\)-command, and (c) the \(\text{if-then-else}\)-command and the \(\text{repeatwhile}\)-command.

[4 marks]

Define, in a programming language notation of your choice, a recursive descent parser that will construct the abstract syntax tree for an input stream conforming to the above syntax for commands. You may assume the existence of a function \(\text{lex}()\) that will yield an integer representing the next lexical token from the input stream, and the functions \(\text{mk2}(\text{op}, x)\), \(\text{mk3}(\text{op}, x, y)\) and \(\text{mk4}(\text{op}, x, y, z)\) that will construct abstract syntax tree nodes with a given operator and one, two or three operands. You should assume that exponentiation is right associative and more binding than subtraction which is left associative. The command following \(\text{then}\) should be the longest possible and the command before \(\text{repeatwhile}\) should be the shortest possible.

[12 marks]

Briefly outline how you would modify your parser if the command to the left of \(\text{repeatwhile}\) was changed to be the longest (rather than the shortest).

[4 marks]