

1997 Paper 3 Question 4

Compiler Construction

Consider the following grammar for expressions ($\langle E \rangle$) and commands ($\langle C \rangle$).

```
 $\langle E \rangle ::= i \mid n \mid \langle E \rangle - \langle E \rangle \mid \langle E \rangle ** \langle E \rangle \mid ( \langle E \rangle )$   
  
 $\langle C \rangle ::= i := \langle E \rangle$   
           $\mid \text{if } \langle E \rangle \text{ then } \langle C \rangle \mid \text{if } \langle E \rangle \text{ then } \langle C \rangle \text{ else } \langle C \rangle$   
           $\mid \langle C \rangle \text{ repeatwhile } \langle E \rangle \mid \langle C \rangle ; \langle C \rangle \mid \{ \langle C \rangle \}$ 
```

Show that there are syntactic ambiguities between (a) the minus (-) and exponentiation (**) operators, (b) the if-command and the if-then-else-command, and (c) the if-then-else-command and the 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 `lex()` that will yield an integer representing the next lexical token from the input stream, and the functions `mk2(op,x)`, `mk3(op,x,y)` and `mk4(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 `then` should be the longest possible and the command before `repeatwhile` should be the shortest possible.

[12 marks]

Briefly outline how you would modify your parser if the command to the left of `repeatwhile` was changed to be the longest (rather than the shortest). [4 marks]