p5q6
ACN
p12q7

1 Compiler Construction 2000

Give a brief description of the main features of *either* Lex and Yacc *or* the corresponding Java tools JLex and Cup.

[5+5 marks]

Illustrate their use by outlining how you would construct a parser for expressions composed of identifiers, integers, unary minus and binary operators +, -, * and /. Your parser is expected to create a parse tree in a format of your choice representing the expression that is presented to it. If it helps you may assume that expressions will be terminated by writing a semicolon after tham.

[10 marks]

1.1 Marking notes

```
LEX/JLex generates lexical analysers based on (extended) regular
expressions that describe the tokens it must recognize.

Yacc/Cup uses a context free grammar to specify what it should handle.

Both tools allow the user to write arbitrary C or Java code as
"semantic actions" that are triggered when a "REDUCE" (as distinct
from "SHIFT") is performed by LR-parsing (or when a regular
expresison ACCEPTS).  These actions have access to items parsed on the
way.

To parse the given grammar we need 3 things:

(a) a Jlex/lex input script. The bit in the middle goes long the lines

"+"    { return new Symbol(sym.PLUS); }
[a-z]+ { return new Symbol(sym.ID, yytext()); }

and it has some guff at the start that is not very important here(!)

(b) Class or struct defs for a parse tree, eg
   class Tree {}
   class Int extents Tree { int n; Int(int n){this.n = n; } }
   class Binary extends Tree {String op; Tree left; Tree right ... }

(c) parser statements to parse
I want evidence of
  basic understanding of how yacc/cup is written
  approx grammar for arith expressions
  some sort of semantoic actions

  nonterminal Tree Expr, Term

  Expr ::= Term
       |   Expr:a PLUS Term:b {: RESULT = new Binary("+", a, b); :}
```


etc