MATH20029: Example Sheet 2

John ff

March 19, 2004

1. Using YACC, Bison or CUPS, create a parser which makes and outputs a parse tree for the language defined by the following grammar.

goal	⇒	statementlist
statementlist	⇒	statement **;** | statement **;** statementlist
statement	⇒	variable **=** expression | **D** varlist | **U** varlist
varlist	⇒	variable | varlist **,** variable
expression	⇒	expression **+** term | term
term	⇒	term ***** factor | factor
factor	⇒	variable | number | **(** expression **)**
variable	⇒	[**a-z**]
number	⇒	digit | digit number
digit	⇒	[**0-9**]

Note: You should not need a lexical phase as all keywords are single characters. Treat integers syntactically.

2. Take the grammar

goal	⇒	expression **;**
goal	⇒	ifstatement **;**
expression	⇒	term **+** term
expression	⇒	term **−** term
expression	⇒	term
expression	⇒	variable **=** expression
term	⇒	factor ***** factor
term	⇒	factor **/** factor
term	⇒	factor
factor	⇒	variable
factor	⇒	integer
factor	⇒	string
factor	⇒	**(** expression **)**
ifstatement	⇒	**if (** expression **)** goal **else** goal
ifstatement	⇒	**if (** expression **)** goal

and add a simple code generator to M68000 assembler. You can assume that variable allocation is provided elsewhere.

Note: this is very nearly the same grammar as the draft first assignment so you should not need to do much re-writing of the lexer/parser.

3. For the previous exercise, by scanning the symbol table generate assembler to allocate the variables and strings.

4. Write a code generator for the grammar

goal	⇒	statementlist
statementlist	⇒	statement ; | statement ; statementlist
statement	⇒	variable = expression | **D** varlist | **U** varlist
varlist	⇒	variable | varlist , variable
expression	⇒	expression + term | term
term	⇒	term * factor | factor
factor	⇒	variable | number | (expression)
variable	⇒	[**a-z**]
number	⇒	digit | digit number
digit	⇒	[**0-9**]

   You should use the YACC parser of the first exercise on this sheet. Ignore the **D** and **U** statements.

5. The **D** and **U** declare and undeclare variables. Add semantic operations to your parser/codegenerator to police this.

6. Comment critically on the code you produce.

7. For one (or all) of the parsers generate TAC rather than assembler.

8. Extend the YACC grammar above so the declarations are of either integer or floating type, using D/U for integers and N/R for floats. Then change the semantic process to check that the types of all the operators are the same, and the assignment is of the right type. At the same time annotate the operations as to their type.

9. As previous exercise, but add type coercions as needed.