Compiler Construction 2005 – Paper 4 Question 1 (AM)
Solution Notes

This question is about grammars (section 3 of the notes) and automated lexing/parser tools (section 5 of the notes).

(a) T is a set of terminals symbols, i.e. symbols which occur in the input to be parsed, e.g. FOR, WHILE, identifier, floating-point-number. N is a set of non-terminals, i.e. symbols which do not occur in the input but which are used to generate strings in terms of other strings, e.g. expression, command. S the start symbol is a distinguished member of N. They are restricted (Chomsky hierarchy) by:

class 0: no restriction

class 1: rules of the form $UAV \rightarrow UBV$ where B, U and V are strings of $N \cup T$ with A is a single non-terminal. ('context sensitive')

class 2: rules of the form $A \rightarrow B$ where B is a string of $N \cup T$ with A is a single non-terminal. ('context free')

class 3: rules of the form $A \rightarrow aB$ or $A \rightarrow a$ where A and B are single non-terminals and a is a single terminal. ('regular grammar')

(b) $S \rightarrow 1; S \rightarrow S + S$.

(c) $S \rightarrow S1; S \rightarrow 2$.

(d) Yes, because type 0 grammars are turing powerful.

(e) I would use `lex` and `yacc`. The former would make tokens and the latter parse them constructing the answer on the fly.

Input to lex:

```
%%
[0-9]+ { yylval = atoi(yytext); return NUM; }
.      { return yytext[0]; }
%%
```

Input to yacc:

```
%token NUM
%left '+' '-'
%left '*' '/'
%%
```

```
comm: comm expr '=' { printf("%d\n", $2); }
    | /* empty */

expr: '(' expr ')' { $$ = $2; }
    | expr '+' expr { $$ = $1 + $3; }
    | expr '-' expr { $$ = $1 - $3; }
    | expr '*' expr { $$ = $1 * $3; }
    | expr '/' expr { $$ = $1 / $3; }
%%
int main() { yyparse(); return 0; }
```