NAME: _____

STUDENT ID: _____

SIGNATURE: _____

The University of New South Wales

Final Examination

November/December 2003

COMP3131/COMP9102

Programming Languages and Compilers

Time allowed: **2 hours**

Total number of questions: **5**

Answer **all** questions

The questions are **not** of equal value

Answer Questions 1 – 2 in one book,
Questions 3 – 4 in another book, and Question 5 in a third book

No examination materials

Answers must be written in ink.

Question 1. A *regular grammar* is a grammar whose productions are in one of the following two forms (where A an B are nonterminals and w represents a sequence of zero or more terminals):

$$\begin{array}{lcl} A & \rightarrow & w \\ A & \rightarrow & Bw \end{array}$$

(a) Give a regular grammar which generates the binary floating-point numbers specified exactly by the following regular expression:

$$(0|1)^{+}.(0|1)^{*}[e(0|1)^{+}]$$

where "( )" indicates grouping, "[ ]" indicates optional item, "ρ^{+}" indicates one of more repetitions of ρ and "ρ^{*}" indicates zero or more repetitions of ρ.

You are **not** allowed to use any regular operator in your answer.

(b) Give a non-regular CFG with fewer productions than your answer to (a) but which generates the same set of strings.

You are **not** allowed to use any regular operator in your answer.

(c) Convert the regular expression given in (a) into an NFA that must not also be a DFA. You are allowed to use any algorithm to perform the conversion.

(d) Use the **subset construction** to convert the NFA from (c) into a DFA.

(e) Convert the DFA from (d) to a minimal-state DFA.

********* ANSWER *********

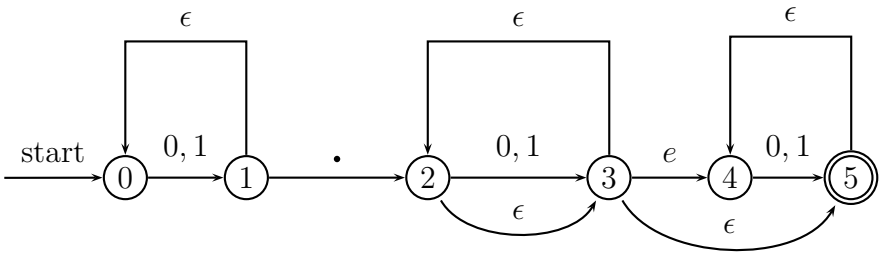
(a)

$$\begin{array}{lcl} S & \rightarrow & E0 \mid E1 \mid F \\ E & \rightarrow & E0 \mid E1 \mid Fe \\ F & \rightarrow & F0 \mid F1 \mid I. \\ I & \rightarrow & I0 \mid I1 \mid 0 \mid 1 \end{array}$$

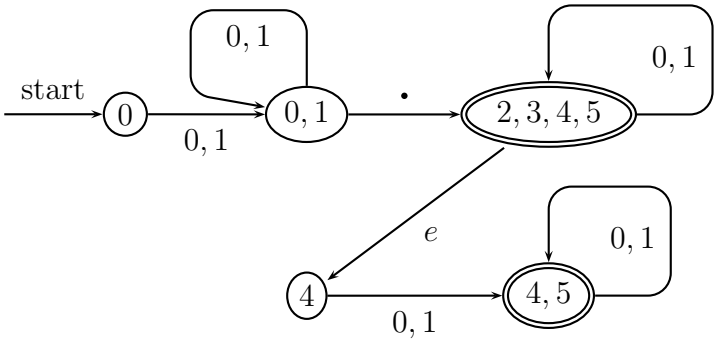
(b)

$$\begin{array}{lcl} S & \rightarrow & I.FE \\ I & \rightarrow & I0 \mid I1 \mid 0 \mid 1 \\ F & \rightarrow & I \mid \epsilon \\ E & \rightarrow & eI \mid \epsilon \end{array}$$

(c)



(d)



(e) same as (d)

NOTE: This question is similar to Q1 of 2001 except that

- the students are asked to produce a left-linear regular grammar
- (e) is new but (d) == (e)

Question 2. This problem is about the design and implementation of a simple spreadsheet program. The spreadsheet consists of a matrix of cells where formulas are stored. As a first step, before implementing any interactive GUI for the program, we wish to be able to read in spreadsheet formulas from formula files, written in textual representation. For simplicity, the spreadsheet supports only integers. An example formula file is shown below:

```
A1 = 6;
A2 = 3;
A4 = A1 + A2 * BL85 + A3;
BL85 = 12;
```

A1 refers to the cell at column A and row 1. BL85 refers to the cell at column BL and row 85. The example above states, e.g., that the cell A1 holds a formula returning the value 6, and that the cell A4 holds a formula returning the value of A4 = A1 + A2 * BL85 + A3. Cells that are not explicitly given any formula, like A3, are considered to hold a formula returning the value of 0. Hence, evaluation of cell A4 will yield the result 42.

Cell formulas can contain integer literals, additions, multiplications, parenthesised expressions, and function calls. The usual precedence and associativity rules apply. Arguments to a function are separated by semicolons. An example is shown below:

```
A5 = A4*2+(4+A6)*foo()+max(B8;5)*square(CZ34)+bar(C2;G34;BA18);
```

In the following you will define some grammars for the formulas. For details where the description above is incomplete, you can make your own choices. However, your grammars must cover the examples above.

(a) What tokens will be needed? Define them using regular expressions.

(b) Define a non-ambiguous CFG, G1, for the formula files.

(c) Is G1 LL(1)? If so, explain briefly that this is the case. If not, motivate why, and then construct an equivalent grammar G2 that is LL(1). G2 should be as similar as possible to G1.

********* ANSWER *********

(a)

```
COL = [A-Z]+
ROW = [0-9]+
INT = [0-9]+
FUNCID = [a-z]+
```

(b)

(i) an ambiguous grammar:

```
FormulaFile -> CellDef ; FormulaFile
            | eps
CellDef     -> CellRef = Expr
CellRef     -> CellCol CellRow
CellCol     -> COL
CellRow     -> ROW
Expr        -> Add
            |  Mul
            |  Call
            |  IntLit
            |  CellRef
            |  ( Expr )
Add         -> Expr + Expr
Mul         -> Expr * Expr
Call        -> FuncName ( Args )
FuncName    -> FUNCID
Args        -> Expr MoreArgs
            |  eps
MoreArgs    -> ; Expr MoreArgs
            |  eps
IntLit      -> INT
```

(ii) a non-ambiguous grammar:

```
FormulaFile -> CellDef ; FormulaFile
            | eps
CellDef     -> CellRef = Expr
CellRef     -> CellCol CellRow
CellCol     -> COL
CellRow     -> ROW
Expr        -> Term ExprP
ExprP       -> + ExrpP
            |  eps
```

```
Term        -> Factor TermP
TermP       -> * TermP
            |  eps
Fact        -> ( Expr)
            |  Call
            |  IntLit
            |  CellRef
Call        -> FuncName ( Args )
FuncName    -> FUNCID
Args        -> Expr MoreArgs
            |  eps
MoreArgs    -> ; Expr MoreArgs
            |  eps
IntLit      -> INT
```

NOTE: This question is a bit more challenging than before.
The problem is well-defined but the students need to come with their own grammar for the problem.

Question 3. Consider the following grammar:

$$
\begin{array}{lcl}
\textit{Expr} & \rightarrow & - \textit{ Expr} \\
 & | & (\textit{ Expr }) \\
 & | & \textit{Var ExprTail} \\
\textit{ExprTail} & \rightarrow & - \textit{ Expr} \\
 & | & \epsilon \\
\textit{Var} & \rightarrow & \textsf{ID } \textit{VarTail} \\
\textit{VarTail} & | & (\textit{ Expr }) \\
 & | & \epsilon
\end{array}
$$

(a) Construct the FIRST and FOLLOW sets for all nonterminals.

(b) Construct the LL(1) parsing table.

********* ANSWER *********

(a)

```
FIRST(Expr)      =  { -, (, ID }
FIRST(ExprTail)  =  { -, ϵ }
FIRST(Var)       =  { ID }
FIRST(VarTail)   =  { (, ϵ }

FOLLOW(Expr)     =  FOLLOW(ExprTail) =  { ), $ }
FOLLOW(Var)      =  FOLLOW(VarTail)  =  {  -, ), $ }
```

(b)

	-	(	)	ID	$
Expr	1	2		3	
ExprTail	4		5		6
Var				7	
VarTail	9	8	9		9

NOTE: This question is straightforward.

Question 4. Consider the following grammar for specifying the **binary** numbers:

$$
\begin{array}{llll}
1 & \mathit{bnum} & \rightarrow & \mathit{num} \bullet \mathit{num} \\
2 & \mathit{num} & \rightarrow & \mathit{num}\ \mathit{digit} \\
3 & \mathit{num} & \rightarrow & \mathit{digit} \\
4 & \mathit{digit} & \rightarrow & \mathbf{0} \mid \mathbf{1}
\end{array}
$$

Let *val* be a synthesised attribute that gives the value of a binary number generated by this grammar. For example, on input 101.101, bnum.val=5.625.

Write an attribute grammar to compute the value of a binary number given by this grammar **using only synthesised attributes**.

You are **not** allowed to modify the the grammar given above.

********* ANSWER *********

```
     Production                          Semantic Rule
-------------------------------------------------------------------------
1. bnum -> num1.num2    bnum.val = num1.val + 2^(-num2.count) * num2.val
2  num1 -> num2 digit   num1.val = 2 * num2.val + digit.val
                        num1.count = num2.count + 1;
3. num  -> digit        num.val = digit.val
                        num.count = 1;
4. digit -> 0           digit.val = 0
   digit -> 1           digit.val = 1
-------------------------------------------------------------------------
```

NOTE: the decimal part can be a bit tricky.

Question 5. Consider the following Pascal for loop:

```
for v:=first to last do stmt
```

where `v` is an integer variable, `first` and `last` are integer literals and `stmt` represents any Pascal statement.

The Pascal standard defines that the for statement to have the same meaning as the following code sequence:

```
t1 = first;
t2 = last;
if (t1 <= t2) then {
   v = t1;
   stmt
   while (v ≠ t2) {
      v = v + 1;
      stmt
   }
}
```

Suppose we used the following class to represent the Pascal **for** statements in the AST:

```
package VC.ASTs;
import VC.Scanner.SourcePosition;

public class ForStmt extends Stmt {
  public Ident I;
  public IntLiteral first, last;
  public Stmt S;

  public ForStmt(Ident iAST, IntLiteral firstAST,
            IntLiteral lastAST, Stmt sAST, SourcePosition Position) {
    super (Position);
    I = iAST;
    first = firstAST;
    last = lastAST;
    S = sAST;
    I.parent = first.parent = last.parent = S.parent = this;
  }

  public Object visit(Visitor v, Object o) {
    return v.visitForStmt(this, o);
  }
}
```

Write Emitter.visitForStmt in Java for generating Jasmin code for the **for** statement.

********* ANSWER *********

```
  public Object visitForStmt(ForStmt ast, Object o) {
    Frame frame = (Frame) o;
    String whileLabel = frame.getNewLabel();
    String endLabel = frame.getNewLabel();

    ast.first.visit(ast.first, o);
    ast.visitIntLiteral(ast.final, o);

    emit(JVM.IF_ICMPGT, endLabel);

    ast.visitIntLiteral(ast.initial, o);
    emitISTORE(ast.I);

    ast.S.visit(this, o);

    emit(whileLabel + ":");
    ast.visitIntLiteral(ast.final, o)
    emitILOAD(ast.I);
    emit(JVM.IFEQ, endLabel);

    emitILOAD(ast.I);
    emit(JVM.ICONST_1);
    emit(JVM.IADD);
    emitISTORE(ast.I);

    ast.S.visit(this, o);

    emit(JVM.GOTO, whileLabel);

    emit(endLabel + ":");

    return null;
}
```