

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

This paper may be retained by the candidate

No examination materials

Answers must be written in ink.

Question 1. Regular Expressions and Finite Automata**[25 marks]**

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

$$\begin{aligned} A &\rightarrow w \\ A &\rightarrow Bw \end{aligned}$$

- (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 or 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.

Question 2. Context-Free Grammars

[25 marks]

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, G_1 , for the formula files.
- (c) Is G_1 LL(1)? If so, explain briefly that this is the case. If not, motivate why, and then construct an equivalent grammar G_2 that is LL(1). G_2 should be as similar as possible to G_1 .

Question 3. Predictive Recursive-Descent Parsing*[15 marks]*

Consider the following grammar:

$$\begin{array}{lll} Expr & \rightarrow & - Expr \\ & | & (Expr) \\ & | & Var ExprTail \\ ExprTail & \rightarrow & - Expr \\ & | & \epsilon \\ Var & \rightarrow & \mathbf{ID} VarTail \\ VarTail & | & (Expr) \\ & | & \epsilon \end{array}$$

- (a) Construct the **FIRST** and **FOLLOW** sets for all nonterminals.
- (b) Construct the LL(1) parsing table.

Question 4. Attribute Grammars

[15 marks]

Consider the following grammar for specifying the **binary** numbers:

- 1 $bnum \rightarrow num \cdot num$
- 2 $num \rightarrow num \ digit$
- 3 $num \rightarrow digit$
- 4 $digit \rightarrow \mathbf{0} \mid \mathbf{1}$

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.

Question 5. Code Generation

[20 marks]

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.