A06507 *No Calculator permitted in this examination*

UNIVERSITY OF BIRMINGHAM

School of Computer Science

First Year – MSc Computer Science
Final Year – BSc Artificial Intelligence/Computer Science with Industrial Year
First Year – UG Aff Computer Science/Software Engineering
Final Year – BSc Computer Science
Final Year – BSc Computer Science with Industrial Year
Final Year – BEng Computer Systems Engineering
Final Year – BEng Computer Science/Software Engineering
Final Year – BEng Computer Science/Software Engineering with Industrial Year

Third Year – MEng Computer Science/Software Engineering
Final Year – BSc Mathematics and Computer Science
Final Year – BSc Pure Mathematics and Computer Science
Final Year – BSc Computer Science with Business Managment

06 02578

Compiler and Languages

Summer Examinations 2013

Time Allowed: 1:30 hours

[Answer ALL Questions]

Turn Over

1. (a) The following list of regular expressions with corresponding token names is used by a lexer generator to construct a lexer:

Regular Expression	Token Name
"a"	T1
"b"	T2
"b*a"	T3
"ab"	T4

Assuming that the usual rules for deciding priorities between conflicting regular expressions are in operation, write out the sequence of tokens that would be returned by the lexer when it is applied to the following string:

```
abaabbbbab
```

[10%]

(b) Write the fragment of a context free grammar that matches an integer *record* language feature. A record appears as a possibly empty comma separated list of elements surrounded by braces. A record element is an identifier followed by a colon character followed by an integer. Thus sample records that must be matched include:

{ }
{ x:10 }
{ x:10, y:20, width:20, height:20 }

Assume that the relevant tokens returned by the lexical analyser are `LBRACE` and `RBRACE`, for left and right brace characters, `ID` for identifier, `INT` for integer and `COMMA` and `COLON` for the comma and colon characters respectively.

[10%]

(c) Consider the following grammar where "a" and "b" are terminal symbols:

$$S \quad ::= \quad aSbS \mid bSaS \mid \epsilon$$

Explain how to show that a grammar is ambiguous and do so for the above grammar using the string:

```
abab
```

[10%]

2. (a) Write out the first and follow sets for the non-terminal symbols X, Y and Z of the following grammar, where S is the start symbol, \$ is the end of file pseudo-token and "a", "b", "c" and "d" are terminal symbols:

$$\begin{array}{lcl} S & ::= & X\ \$ \\ X & ::= & Y\ a \mid b \\ Y & ::= & d\ Z\ Y \mid a \\ Z & ::= & b\ X \mid c\ Z \mid \epsilon \end{array}$$

[10%]

(b) Consider the following ACTION and GOTO tables for an SLR(1) parser where \$ is the end of file pseudo-token, a and b are terminal symbols:

State	ACTION			GOTO
	a	b	\$	S
0	s2			1
1			Accept	
2	s2	r $S ::= a$	r $S ::= a$	3
3		s4		
4	s2			5
5		r $S ::= a\ S\ b\ S$	r $S ::= a\ S\ b\ S$	

Write out a table showing the state of the stack, the remaining input and the action taken in every step when using these tables to parse the string

$$a\ a\ b\ a\ \$$$

[10%]

(c) Consider the following augmented grammar, where S' is the start symbol, \$ is the end of file pseudo-token, and "x", "$=$" and "$*$" are terminal symbols:

$$\begin{array}{lcl} S' & ::= & S\ \$ \\ S & ::= & V\ =\ E \\ S & ::= & E \\ E & ::= & V \\ V & ::= & x \\ V & ::= & *E \end{array}$$

In the context of an LR(1) parser, calculate the start state of the LR(1) automaton for this grammar, i.e. calculate

$$Closure\,(\{S ::= \cdot S\ \$\ ,\ ?\})$$

[10%]

3. (a) In a particular compiler for the C programming language, the activation frame for a function call includes locations for (listed here in alphabetic order):

   - FP: frame pointer
   - LVars: local variables
   - Params: function call parameters
   - RA: return address
   - RV: return value
   - Regs: saved register values
   - Temp: temporary values

   Assume that function `f` calls function `g`, where both functions take a number of parameters and return an integer value. Draw a diagram of the state of the stack while the function `g` is executing that shows the layout of the activation frames, including all locations listed above, for both `f` and `g` and identifies clearly which locations are written by the function `f` and which by function `g` when the call from `f` to `g` is being executed.

   [10%]

   (b) Consider the following Gnu C code involving nested functions. Draw the contents of the call stack immediately after line 6 is executed but before the function `q()` returns. Then explain the mechanism for accessing the variable "a" during the execution.

```
 1   int p()
 2   {
 3       int a = 0;
 4       void q()
 5       {
 6           a = 1;
 7       }
 8       q() ;
 9       return a;
10   }
```

   [10%]

4. Consider the following Java code excerpt:

```
1.   b = v = 0 ;
2.   while (i < a)
3.   {
4.       if (i % 2 == 1)
5.           v += i*i ;
6.       i++ ;
7.   }
8.   b = v;
```

(a) Draw the control flow graph for this code and annotate each node with the *Gen* and *Kill* sets suitable for that node for a live variable analysis.

[10%]

(b) The data flow equations for live variable analysis are as follows:

$$in[n] = gen[n] \cup (out[n] - kill[n])$$
$$out[n] = \bigcup_{s \in succ[n]} in[s]$$

Using these equations, write out the table of in and out values for each node of your flow graph in your answer to part (a) above and explain the optimal order for calculating the values.

[10%]