UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B.Sc. Examination 2014

COMPUTING AND INFORMATION SYSTEMS

IS53011A Language Design and Implementation

Duration: 2 hours 15 minutes

Date and time:

There are five questions on this paper. You should answer no more that THREE questions. Full marks will be awarded for complete answers to a total of THREE questions. Each question carries 25 marks. The marks for each part of a question are indicated at the end of the part in [.] brackets.

There are 75 marks available on this paper.

THIS PAPER MUST NOT BE REMOVED FROM THE EXAMINATION ROOM

Question 1.

a) Explain briefly what is the ambiguity problem in programming language grammars. **[3]**

b) i) Define the notion of parse tree in context of programming language design. **[4]**

ii) Let the following simple programming language grammar be given:

$$P \rightarrow \{ \ D \ ; E \ \}$$
$$D \rightarrow \ var \ \ x$$
$$E \rightarrow \ x := E \ | \ (E) \ | \ E * F \ | \ F$$
$$F \rightarrow \ 1 \ | \ 2 \ | 3$$

Using this grammar develop a parse tree for the expression: { *var x*; *x* := ((1*2)*3). **[8]**

c) i) Explain what kind of strings are generated by the following regular expression: *1*(*0*|*1*)**1*| *1*. **[4]**

ii) Two regular expressions are considered equivalent when they denote the same set of strings. Give a different regular expression that is equivalent to (*0* | *1*)*. **[6]**

Question 2.

a) Design a nondeterministic finite state automaton (NFA) using the Thompson's construction algorithm for the following regular expression: $a^* \; (\; b \; | \; c \;) \; b$. **[6]**

b) Transform the designed NFA into a corresponding deterministic finite-state automaton (DFA) using the subset construction algorithm. Show the computation of the functions $\in$-*closure* and *move* leading to the DFA. **[11]**

c) Build the transition table for the constructed DFA. **[4]**

d) Draw the transition graph for the constructed DFA. **[4]**

Question 3.

a) Define formally the notion of context-free grammar using a tuple and explain each component in it. **[2]**

b) Consider the following grammar for top-down nonrecursive predictive parsing:

$S \rightarrow \{\ D\ E\ \}$
$D \rightarrow var$
$E \rightarrow x := F\ T$
$T \rightarrow *\ F\ T \mid \in$
$F \rightarrow x \mid 1$

i) Compute the functions *FIRST* and *FOLLOW* necessary for parser construction. **[10]**

ii) Suppose that the ready parsing table for this grammar is:

	var	*x*	1	:=	*	{	}	$
S						$S \rightarrow \{D\ E\ \}$		
D	$D \rightarrow var$							
E		$E \rightarrow x := F\ T$						
T					$T \rightarrow *F\ T$		$T \rightarrow \in$	
F		$F \rightarrow x$	$F \rightarrow 1$					

Demonstrate the moves of the nonrecursive predictive parsing algorithm on the input string: $\{\ var\ x := x * 1\ \}$. Show the stack, the input and the output of the nonrecursive parser. **[13]**

Question 4.

a) Give the main four advantages of LR bottom-up parsers. **[4]**

b) Consider the following grammar suitable for bottom-up parsing:

(1) $S \rightarrow E$
(2) $E \rightarrow E + F$
(3) $E \rightarrow a$
(4) $F \rightarrow b$

i) Develop the canonical collection of items for this grammar using the sets-of-items construction algorithm. **[6]**

ii) Build the DFA whose states are these sets of valid items. **[4]**

ii) Demonstrate the operation of the bottom-up parsing algorithm on the input string: $a + b + b$ $ using the parsing table given below. Show the input, the stack and the output in a table. **[11]**

	Action				Goto	
State	*a*	*b*	+	$	*E*	*F*
0	*s2*				*1*	
1			*s3*	*accept*		
2			*r3*	*r3*		
3		*s5*				*4*
4			*r2*	*r2*		
5			*r4*	*r4*		

Question 5.

a) i) What are the main two benefits of using machine-independent intermediate code generation in programming language compilers? **[3]**

ii) Explain briefly where is the position of the intermediate code generation phase in a programming language compiler? **[3]**

b) Consider the following function which scans through an array to locate the element with the largest index and exchanges it with the *N*-th element:

```
int Exchange()
{
    int j, v, N, max;
     max = 1; N = 5;
     for ( j = 2; j <= N; j++ )
         if ( a[ j ] > a[ max ] )
             max = j;
     v = a[ max ]; a[ max ] = a[ N ];
     return v;
}
```

i) Generate three-address intermediate code for this function. **[14]**

ii) Optimise the generated three-address code using the technique reduction in strength. **[5]**