UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B.Sc. Examination 2015

DEPARTMENT OF COMPUTING

IS53011A Language Design and Implementation

Duration: 2 hours 15 minutes

Date and time: Monday 12 January, 2.30pm

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) Define the notion of a regular expression over a given alphabet. **[5]**

b) Explain briefly which kind of programming language constructs can not be described by regular expressions, and reason whether these constructs can be specified using context-free grammars. **[4]**

c) Rewrite the following regular expression in a more compact format: (*b** *c**)*. **[5]**

d) Demonstrate elimination of immediate left recursion from the following context-free grammar: **[6]**

(1) $S \rightarrow SE$
(2) $S \rightarrow SSa$
(3) $S \rightarrow ES$
(4) $S \rightarrow b$

e) Give the algorithm for construction of predictive parsing tables. **[5]**

Question 2.

a) Consider the following grammar: c^* (bcc^*)* (b | ∈). For each of the following strings, say whether it could be generated by the grammar:

i) ∈
ii) ccbccbbc
iii) bccbccc

[3]

b) Develop a nondeterministic finite state automaton (NFA) for the simple language defined by the regular expression: $ca(a|c)b$ using the Thompson's construction algorithm. **[6]**

c) Convert the NFA from part (b) above to a deterministic finite-state automaton (DFA) using the subset construction algorithm. **[12]**

d) Create the transition graph of the constructed DFA from part (c). **[4]**

Question 3.

a) What are the three main advantages of the LR parsing technique? **[5]**

b) Let the following LR grammar suitable for top-down parsing be given:

(1) $E \rightarrow T$
(2) $T \rightarrow bTc$
(3) $T \rightarrow bc$

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

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

iii) Interpret the operation of the LR parsing algorithm on the input: *bbbccc* $, and show the contents of the stack, the input and the output. **[10]**

	Action			Goto
State	*b*	*c*	$	*T*
0	*s1*			*3*
1	*s1*	*s4*		*2*
2		*s5*		
3			*accept*	
4	*r3*	*r3*	*r3*	
5	*r2*	*r2*	*r2*	

Question 4.

a) Describe briefly the main four components of a context-free grammar. **[4]**

b) Give a definition of grammar derivations in context of programming language compilers. **[3]**

c) Consider the following LL(*1*) grammar for top-down parsing:

$E \rightarrow T\ F$
$T \rightarrow$ **double** $D \mid (E)$
$F \rightarrow +E \mid \in$
$D \rightarrow *T \mid \in$

i) Derive the functions *FIRST* and *FOLLOW* necessary for building the corresponding parsing table for a top-down nonrecursive predictive parsing algorithm. **[8]**

ii) Assume that the following parsing table for the LL(*1*) grammar from part (c) is given:

	double	+	*	(	)	$
E	$E \rightarrow TF$			$E \rightarrow TF$		
T	$T \rightarrow$ **double** D			$T \rightarrow (E)$		
F		$F \rightarrow +E$			$F \rightarrow \in$	$F \rightarrow \in$
D		$D \rightarrow \in$	$D \rightarrow *T$		$D \rightarrow \in$	$D \rightarrow \in$

Using this parsing table show the stack, the input and the output of the nonrecursive predictive parsing algorithm on the input string:
double * **double** . **[10]**

Question 5.

a) i) What are the two most important properties that an optimising compiler should provide? **[4]**

ii) Explain briefly where the name “three-address code” in the field of computer programming language design comes from? **[3]**

b) Consider the following simple function which iteratively performs summation of multiplied consecutive integers from a given array:

```
int Summation( int a[], int N )
{
    int j, sum;
     j = 1; sum = 0;
    while ( j < N )
    {
        sum = sum + a[ j-1 ] * a[ j ];
        j = j + 1;
    }
     return sum;
}
```

i) Develop three-address intermediate code from this simple function. **[13]**

ii) Optimise the developed three-address code using the techniques code motion and reduction in strength. **[5]**