UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B.Sc. Examination 2004

COMPUTING AND INFORMATION SYSTEMS

IS53011A (CIS324) Language Design and Implementation

Duration: 2 hours 15 minutes

Date and time:

- *Full marks will be awarded for complete answers to THREE questions. Do not attempt more than THREE questions on this paper. Although each question carries 25 marks, and therefore the result from three questions sums up to 75 marks, the final result will be additionally scaled to 100.*

- *Electronic calculators may be used. The make and model should be specified on the script. The calculator must not be programmed prior to the examination. Calculators which display graphics, text or algebraic equations are not allowed.*

THIS EXAMINATION PAPER MUST NOT BE

REMOVED FROM THE EXAMINATION ROOM

Question 1.

a) Define formally the concept of a parse tree in context of compiler design. **[4]**

b) Give answers to the following questions concerning the ambiguity of computer programming
language grammars: **[6]**

i) When does the problem of ambiguity of programming language grammars arise?

ii) Is it important to resolve the ambiguity problem in the design of language grammars?

c) Consider the following grammar for expressions with balanced brackets:

$$T \rightarrow [\, T\,] \mid TT \mid \in$$

Demonstrate whether this grammar is ambiguous or unambiguous by drawing parse trees for the following string: [][][]. **[8]**

d) Let the following grammar for expressions be given:

$$E \rightarrow E + T \mid E - T \mid T$$
$$F \rightarrow T * F \mid T / F \mid F$$
$$T \rightarrow D \mid (E)$$
$$D \rightarrow 1 \mid 2 \mid 3 \mid 4$$

Give the unique parse tree for the following expression: (1 + 2) * (3 – 4). **[7]**

Question 2.

a) Explain which are the values and operations that enable to define recursively regular expressions over a given alphabet. **[4]**

b) Develop a nondeterministic finite-state automaton (NFA) for the regular expression: $(ab^* \mid c)$ using the Thompson's construction algorithm. **[6]**

c) Convert the NFA for the expression: $(ab^* \mid c)$ into a deterministic finite-state automaton (DFA) using the subset construction algorithm. Illustrate the development in the following way: **[15]**

i) Show the derivation of the states and transitions of the DFA, demonstrating the computation of the $\in$-*closure* and *move* functions;

ii) Develop the transition table for the DFA;

iii) Draw the transition graph for the resulting DFA.

Question 3.

a) Which are the components of a context-free grammar? Explain each of them briefly. **[6]**

b) What is the role of the parser in programming language compilers? **[4]**

c) Suppose we have the following context-free grammar for describing simple statements:

$$
\begin{aligned}
S &\rightarrow aS \mid bLc \mid d \\
L &\rightarrow ST \\
T &\rightarrow ;L \mid \in
\end{aligned}
$$

which can be analysed with the parsing table:

	a	b	c	d	;	\$
S	$S \rightarrow aS$	$S \rightarrow bLc$		$S \rightarrow d$		
L	$L \rightarrow ST$	$L \rightarrow ST$		$L \rightarrow ST$		
T			$T \rightarrow \in$		$T \rightarrow ;L$	

Use the nonrecursive predictive parser to decide the syntactic correctness of the input string: $bad;dc\$$ (show the stack, the input and the output of the parser). **[15]**

Question 4.

a) Explain the four main advantages of LR parsers. **[4]**

b) Present the algorithmic framework of the LR parser. **[8]**

c) Let the following context-free grammar be given:

$S \rightarrow E$

$E \rightarrow E + T \mid T$

$T \rightarrow$ **int**

as well as its LR parsing table:

	Action			Goto	
State	**int**	+	$	*E*	*T*
0	*s1*			*2*	*3*
1		*r3*	*r3*		
2		*s4*	*acc*		
3		*r2*	*r2*		
4	*s1*				*5*
5		*r1*	*r1*		

Show the moves of the LR (bottom-up) parser that will be made when processing the expression: **int** + **int** + **int** $ (the stack, the input and the action). **[13]**

Question 5.

a) Where is the position of the intermediate code generation phase in a programming language compiler? **[3]**

b) What is the advantage of having an intermediate code generation language with a small operator set? **[3]**

c) Give the two benefits of using machine-independent intermediate code generation in programming language compilers? **[4]**

c) Translate the following program fragment into three-address intermediate code, assuming that this is a procedure in a global environment with a symbol table denoted by *s*. Start enumerating the code instructions from one. **[15]**

```
void Sort( int a[], int N )
{
  int i, j, k;

  i = 1;
  while ( i <= N )
  {
    k = a[ i ];
      j = i;
    while ( a[ j-1 ] > k )
    {
      a[ j ] = a[ j-1 ];
         --j;
      }
    a[ j ] = k;
  }
}
```