UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B.Sc. Examination 2005

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 are not allowed.*

THIS EXAMINATION PAPER MUST NOT BE

REMOVED FROM THE EXAMINATION ROOM

Question 1.

a) Enumerate the main phases and the additional activities according to which programming language compilers operate. **[5]**

b) Let the following ambiguous grammar for expressions be given:

$$S \rightarrow E$$
$$E \rightarrow E + E \mid E - E \mid N$$
$$N \rightarrow 1 \mid 2 \mid 3$$

i) Develop two different parse trees for the expression: $1 - 2 + 3$. **[6]**

ii) Show the formula for the interpretation of this expression from each tree using parentheses to indicate the order of execution of the operands. **[2]**

iii) Convert the above grammar into two different unambiguous grammars from which these parse trees can be derived. **[4]**

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

$$S \rightarrow AaBb$$
$$A \rightarrow Ab \mid b$$
$$B \rightarrow aB \mid a$$

Demonstrate which of the following sentences are in the language generated by this grammar using leftmost derivations:

i) $bbbab$ **[4]**

ii) $bbaab$ **[4]**

Question 2.

a) Define the notion of a nondeterministic finite state automaton (NFA) and its components. **[5]**

b) Using Thompson's construction algorithm, build a nondeterministic finite-state automaton (NFA) for the following regular expression: $(a \mid b)b*$. **[6]**

c) Transform the NFA for the expression $(a \mid b)b*$ into a deterministic finite-state automaton (DFA) with the subset construction algorithm:

i) Compute the $\in$-*closure* and *move* functions. **[9]**

ii) Construct the transition table for the DFA. **[5]**

Question 3.

Interpret the performance of the nonrecursive predictive parsing algorithm to determine whether the string $b\ id := (\ c+d\ )\ e$ is correct according to the following context-free grammar:

$$
\begin{aligned}
S &\rightarrow bSe \mid A \\
A &\rightarrow id\ E \\
E &\rightarrow := TE \mid \in \\
T &\rightarrow (\ T+T\ ) \mid c \mid d
\end{aligned}
$$

and its parsing table:

	id	c	d	b	e	+	:=	(	)	$
S	$S \rightarrow A$			$S \rightarrow bSe$						
A	$A \rightarrow id\ E$									
E					$E \rightarrow \in$		$E \rightarrow := TE$			
T		$T \rightarrow c$	$T \rightarrow d$					$T \rightarrow (T+T)$		

At each algorithmic step show the stack, the input and the output. **[25]**

Question 4.

a) Which are the two steps for building simple LR (SLR) parsing tables for bottom-up syntax analysis? **[4]**

b) Suppose we have the following augmented grammar:

$$S' \to S$$
$$S \to Sc \mid SA \mid A$$
$$A \to aSb \mid ab$$

i) Construct the canonical LR(0) collection of items from this grammar. **[15]**

ii) Develop the DFA whose states are these sets of valid items. **[6]**

Question 5.

a) Give the five most commonly used three-address code statements. Explain each component in them. **[10]**

b) Generate three-address intermediate code for the following source program fragment using a symbol table *S*, arithmetic and jump statements. **[15]**

```
int i, x, z;
int y[ 5 ];

i = 0;
x = 1;
z = 5;

while ( i < z )
{
   y[ i ] = x + i;
   if ( y[ i ] >= z-1 )
      y[ i ] = 0;
   ++i;
}
```

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 are not allowed.*

THIS EXAMINATION PAPER MUST NOT BE REMOVED FROM THE EXAMINATION ROOM

Solutions CIS324

Question 1.

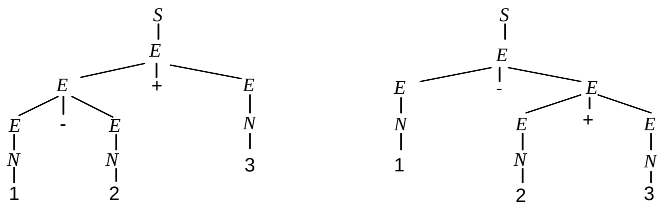
a) Enumerate the main phases and the additional activities according to which programming language compilers operate. **[5]**

Programming language compilers operate in the following phases: *lexical analysis*, *syntax analysis*, *intermediate code generation*, *code optimisation*, and *code generation*.
The two additional activities are: *symbol table management*, and *error handling*.

b) Let the following ambiguous grammar for expressions be given:

$$S \rightarrow E$$
$$E \rightarrow E + E \mid E - E \mid N$$
$$N \rightarrow 1 \mid 2 \mid 3$$

i) Develop two different parse trees for the expression: $1 - 2 + 3$. **[6]**



ii) Show the formula for the interpretation of this expression from each tree using paremtheses to indicate the order of execution of the operands. **[2]**

$(1 - 2) + 3$ and $1 - (2 + 3)$

iii) Convert the above grammar into two different unambiguous grammars from which these parse trees can be derived. **[4]**

$S \rightarrow E$	$S \rightarrow E$
$E \rightarrow E + T \mid E - T \mid T$	$E \rightarrow T + E \mid T - E \mid T$
$T \rightarrow N$	$T \rightarrow N$
$N \rightarrow 1 \mid 2 \mid 3$	$N \rightarrow 1 \mid 2 \mid 3$

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

$$S \rightarrow AaBb$$
$$A \rightarrow Ab \mid b$$
$$B \rightarrow aB \mid a$$

Demonstrate which of the following sentences are in the language generated by this grammar using leftmost derivations:

i) *bbbab*. **[4]**

$S \Rightarrow_{lm} AaBb \Rightarrow_{lm} AbaBb \Rightarrow_{lm} AbbaBb \Rightarrow_{lm} bbbaBb$, therefore NO

ii) *bbaab*. **[4]**

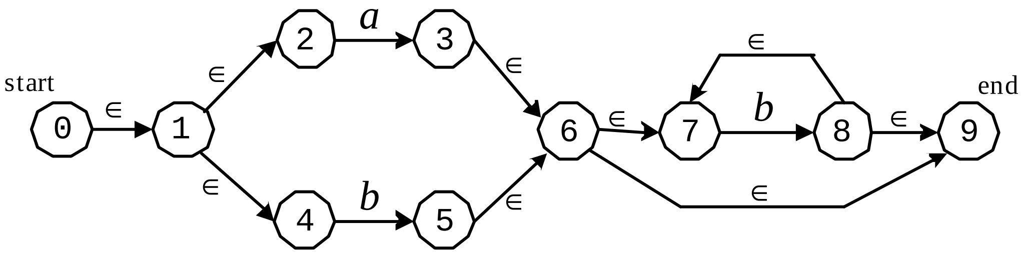
$S \Rightarrow_{lm} AaBb \Rightarrow_{lm} AbaBb \Rightarrow_{lm} bbaBb \Rightarrow_{lm} bbbaab$, therefore YES

Question 2.

a) Define the notion of a nondeterministic finite state automaton (NFA) and its components. **[5]**

A nondeterministic finite state automaton (NFA) is a mathematical model describing a recognizer of a programming language. An NFA implements a diagram of transitions from state-symbol pairs to sets of states, the transitions being carried out on seeing input symbols from the chosen alphabet. The NFA components are: a set of states *S*, input alphabet: ∑, transition function `move` that maps pairs `state-symbol` to `sets-of-states`, input state S_0 and final states *F*.

b) Using Thompson's construction algorithm, build a nondeterministic finite-state automaton (NFA) for the following regular expression: $(a \mid b)b^*$. **[6]**



c) Transform the NFA for the expression $(a \mid b)b^*$ into a deterministic finite-state automaton (DFA) with the subset construction algorithm:

i) Compute the ∈*-closure* and *move* functions. **[9]**

∈*-closure*({0}) = { 1, 2, 4 } = *A*

∈*-closure*(*move*(*A*, *a*)) = ∈*-closure*(*move*({1, 2, 4 }, *a*)) = { 3, 6, 7, 9 } = *B*

∈*-closure*(*move*(*A*, *b*)) = ∈*-closure*(*move*({1, 2, 4 }, *b*)) = { 5, 6, 7, 9 } = *C*

∈*-closure*(*move*(*B*, *a*)) = ∈*-closure*(*move*({ 3, 6, 7, 9 }, *a*)) = {}

∈*-closure*(*move*(*B*, *b*)) = ∈*-closure*(*move*({ 3, 6, 7, 9 }, *b*)) = { 7, 8, 9 } = *D*

∈*-closure*(*move*(*C*, *a*)) = ∈*-closure*(*move*({ 5, 6, 7, 9 }, *a*)) = {}

∈*-closure*(*move*(*C*, *b*)) = ∈*-closure*(*move*({ 5, 6, 7, 9 }, *b*)) = { 7, 8, 9 } = *D*

∈*-closure*(*move*(*D*, *a*)) = ∈*-closure*(*move*({ 7, 8, 9 }, *a*)) = {}

∈*-closure*(*move*(*D*, *b*)) = ∈*-closure*(*move*({ 7, 8, 9 }, *a*)) = { 7, 8, 9 } = *D*

ii) Construct the transition table for the DFA. **[5]**

state	*a*	*b*
A	*B*	*C*
B	{}	D
C	{}	D
D	{}	D

Question 3.

Interpret the performance of the nonrecursive predictive parsing algorithm to determine whether the string $b\ id := (\ c+d\ )\ e$ is correct according to the following context-free grammar:

$$
\begin{aligned}
S &\rightarrow bSe \mid A \\
A &\rightarrow id\, E \\
E &\rightarrow := TE \mid \in \\
T &\rightarrow (\ T+T\ ) \mid c \mid d
\end{aligned}
$$

and its parsing table:

	id	*c*	*d*	*b*	*e*	+	:=	(	)	$
S	$S\rightarrow A$			$S\rightarrow bSe$						
A	$A\rightarrow id\ E$									
E					$E\rightarrow\in$		$E\rightarrow :=TE$			
T		$T\rightarrow c$	$T\rightarrow d$					$T\rightarrow(T+T)$		

At each algorithmic step show the stack, the input and the output. **[25]**

Stack	Input	Output
$S	*b id* := (*c*+*d*) *e* $	
$*eSb*	*b id* := (*c*+*d*) *e* $	$S \rightarrow bSe$
$*eS*	*id* := (*c*+*d*) *e* $	
$*eA*	*id* := (*c*+*d*) *e* $	$S \rightarrow A$
$*eEid*	*id* := (*c*+*d*) *e* $	$A \rightarrow id\ E$
$*eE*	:= (*c*+*d*) *e* $	
$*eT*:=	:= (*c*+*d*) *e* $	$E \rightarrow := TE$
$*eET*	(*c*+*d*) *e* $	
$*eE*)*T*+*T*(	(*c*+*d*) *e* $	$T \rightarrow (\ T+T\ )$
$*eE*)*T*+*T*	*c*+*d*) *e* $	
$*eE*)*T*+*c*	*c*+*d*) *e* $	$T \rightarrow c$
$*eE*)*T*+	+*d*) *e* $	
$*eE*)*T*	*d*) *e* $	
$*eE*)*d*	*d*) *e* $	$T \rightarrow d$
$*eE*)	) *e* $	
$*eE*	*e* $	
$*e*	*e* $	$E \rightarrow \in$

Question 4.

a) Which are the two steps for building simple LR (SLR) parsing tables for bottom-up syntax analysis? **[4]**

The two steps of the SLR method for building parsing tables are:
- construction of a DFA to recognize viable prefixes from the given grammar (using sets-of-items);
- determination of the parsing action and goto entries of the parsing table using this DFA..

b) Suppose we have the following augmented grammar:

$$
\begin{aligned}
&S' \rightarrow S \\
&S \rightarrow Sc \mid SA \mid A \\
&A \rightarrow aSb \mid ab
\end{aligned}
$$

i) Construct the canonical LR(0) collection of items from this grammar. **[15]**

I_0: $S' \rightarrow \bullet S$ $S \rightarrow \bullet Sc$ $S \rightarrow \bullet SA$ $S \rightarrow \bullet A$ $A \rightarrow \bullet aSb$ $A \rightarrow \bullet ab$	I_1: $S \rightarrow S \bullet c$ $S \rightarrow S \bullet A$ $A \rightarrow \bullet aSb$ $A \rightarrow \bullet\ ab$	I_2: $S \rightarrow A \bullet$	I_3: $A \rightarrow a \bullet Sb$ $A \rightarrow a \bullet b$ $S \rightarrow \bullet Sc$ $S \rightarrow \bullet SA$ $S \rightarrow \bullet A$ $A \rightarrow \bullet aSb$ $A \rightarrow \bullet ab$
I_4: $S \rightarrow Sc \bullet$	I_5: $S \rightarrow SA \bullet$	I_6: $A \rightarrow aS \bullet b$ $S \rightarrow S \bullet A$ $A \rightarrow \bullet aSb$ $A \rightarrow \bullet ab$	I_7: $A \rightarrow ab \bullet$ I_8: $A \rightarrow aSb \bullet$

ii) Develop the DFA whose states are these sets of valid items. **[6]**

$$\begin{tikzcd}
 & & I_4 & \\
 & I_1 \arrow[ur, "c"] \arrow[r, "A"] & I_5 & \\
I_0 \arrow[ur, "S"] \arrow[r, "A"] \arrow[dr, "a"'] & I_2 & & \\
 & I_3 \arrow[u, "A"] \arrow[r, "S"] \arrow[dr, "b"] \arrow[loop below, "a"] & I_6 \arrow[l, bend right, "a"'] \arrow[r, "b"] & I_8 \\
 & & I_7 &
\end{tikzcd}$$

Question 5.

a) Give the five most commonly used three-address code statements. Explain each component in them. **[10]**

- *assignment statements* of the kind: `x := y op z`, including copy statements: `x := y`;
- *unconditional jumps*: `goto L` where `L` is the label of the next instruction;
- *conditional jumps*: `if x relop y goto L` where `relop` is relation operator;
- *procedure calls*: `param x1`, `param x2`,… `call p,n` and `return y`, where `p` denotes a procedure `p(x1, x2,…, xn)` with `n` arguments provided in advance;
- *indexed assignments*: `x := y[ i ]`, and `x[ i ] := y`, where `i` is the index that points to a location at `i` points after the beginning of the array `x`.

b) Generate three-address intermediate code for the following source program fragment using a symbol table s, arithmetic and jump statements. **[15]**

```
int i, x, z;
int y[ 5 ];

i = 0;
x = 1;
z = 5;

while ( i < z )
{
   y[ i ] = x + i;
   if ( y[ i ] >= z-1 )
      y[ i ] = 0;
   ++i;
}
```

```
( 1 )    s := mktable( nil )
( 2 )    enter( s, i, int, 4 )
( 3 )    enter( s, x, int, 4 )
( 4 )    enter( s, z, int, 4 )
( 5 )    enter( s, y, array, 5*4 )
( 6 )    i := 0
( 7 )    x := 1
( 8 )    z := 5
( 9 )    if i >= z goto (19)
( 10 )   t1 := 4 * i
( 11 )   t2 := x + i
( 12 )   y[ t1 ] := t2
( 13 )   t3 := y[ t1 ]
( 14 )   t4 := z - 1
( 15 )   if t3 <= t4 goto (17)
( 16 )   y[ t1 ] := 0
( 17 )   i := i + 1
( 18 )   goto (9)
( 19 )   end
```