DUBLIN CITY UNIVERSITY

SEMESTER 1 EXAMINATIONS 2017/2018

MODULE:	CA4003 - Compiler Construction
PROGRAMME(S):	
CASE	BSc in Computer Applications (Sft.Eng.)
CPSSD	BSc in ComputationalProblem Solv&SW Dev.
YEAR OF STUDY:	4
EXAMINERS:	Dr. David Sinclair (Ph:5510) Dr. Hitesh Tewari External Prof. Brendan Tangney External
TIME ALLOWED:	3 hours
INSTRUCTIONS:	Answer 10 questions. All questions carry equal marks.

PLEASE DO NOT TURN OVER THIS PAGE UNTIL INSTRUCTED TO DO SO

The use of programmable or text storing calculators is expressly forbidden.
Please note that where a candidate answers more than the required number of questions, the examiner will mark all questions attempted and then select the highest scoring ones.

There are no additional requirements for this paper.

Note: In the following questions, non-terminal symbols are represented by strings starting with an upper case letter, e.g. A, Aa, Name, and terminal symbols are represented by either individual symbols (e.g. $+$) or sequence of symbols (e.g. $>=$), or by strings starting with a lower case letter, e.g. a, xyz. The ϵ symbol represents an empty symbol or null string as appropriate. The $ symbol represents the end-of-file.

QUESTION 1 *[Total marks: 10]*

1(a) [4 Marks]

Given a binary alphabet 0,1, write a regular expression that recognises all words that have an even number of '1's and end with a '0'.

1(b) [6 Marks]

Use the subset construction method to derive a deterministic finite state automaton that recognises the language from part (a).

[End Question 1]

QUESTION 2 *[Total marks: 10]*

[10 Marks]

Calculate the FIRST and FOLLOW sets for the following grammar.

$S \rightarrow T; S$
$S \rightarrow \epsilon$
$T \rightarrow UR$
$R \rightarrow *T$
$R \rightarrow \epsilon$
$U \rightarrow x$
$U \rightarrow y$
$U \rightarrow [S]$

[End Question 2]

QUESTION 3 **[Total marks: 10]**

[10 Marks]

Convert the following grammar into an LL(1) grammar which recognises the same language (you may assume that the grammar is unambiguous).

$F \rightarrow F\ B\ a$
$F \rightarrow c\ D\ E$
$F \rightarrow c$

[End Question 3]

QUESTION 4 **[Total marks: 10]**

Construct the LL(1) parse table for the following grammar , and using this table determine whether or not it is an LL(1) grammar.

$S \rightarrow u\ B\ D\ z$
$B \rightarrow B\ v$
$B \rightarrow w$
$D \rightarrow E\ F$
$E \rightarrow y$
$E \rightarrow \epsilon$
$F \rightarrow x$
$F \rightarrow \epsilon$

[10 Marks]

[End Question 4]

QUESTION 5 **[Total marks: 10]**

[10 Marks]

Construct the LR(1) parse table for the following grammar and determine whether or not it is an LR(1) grammar.

$S \rightarrow Statement\$$
$Statement \rightarrow Var = Exp$
$Statement \rightarrow Exp$
$Var \rightarrow *Exp$
$Var \rightarrow id$
$Exp \rightarrow Var$

[End Question 5]

QUESTION 6 ***[Total marks: 10]***

[10 Marks]

Construct the SLR(1) parse table for the grammar in Question 5 and determine whether or not it is an SLR(1) grammar.

[End Question 6]

QUESTION 7 ***[Total marks: 10]***

7(a) [6 Marks]

Convert the following source code into intermediate code using the syntax-directed approach given in the appendix. Assume that all variables are stored in 4 bytes.

```
sum = 0;
i = 0;
while (i < 9)
{
  diff = a[i] - a[i+1];
  sum = sum + diff;
  i = i+ 1;
}
```

7(b) [4 Marks]

Generate a *Control Flow Graph* from the intermediate code generated in part (a). Clearly describe the rules used to generate the *Control Flow Graph*.

[End Question 7]

QUESTION 8 ***[Total marks: 10]***

8(a) [3 Marks]

Describe the following type of *code optimisation*:

- *Peephole Optimisation*
- *Basic Block Optimisation*
- *Global Optimisation*

8(b) [7 Marks]

Give example of 4 different types of *Peephole Optimisation*.

[End Question 8]

QUESTION 9 ***[Total marks: 10]***

9(a) [6 Marks]

For the following intermediate code, assuming variable d, k and j are live on exit from this code, calculate which variables are live on entry.

$$
\begin{array}{l}
t_1 = j + 12 \\
g = a[t_1] \\
h = k - 1 \\
f = g * h \\
t_2 = j + 8 \\
e = a[t_2] \\
t_3 = j + 16 \\
m = a[t_3] \\
b = a[f] \\
c = e + 8 \\
d = c \\
k = m + 4 \\
j = b
\end{array}
$$

9(b) [4 Marks]

For the code in part (a), draw the *interference graph* showing *move-related* edges.

[End Question 9]

QUESTION 10 ***[Total marks: 10]***

[10 Marks]

With the aid of example code, describe the *Visitor* pattern. Why is it particularly suited to *"walking an abstract syntax tree"*?

[End Question 10]

[APPENDICES]

Syntax-directed definition approach to build the 3-address code

Production	Semantic Rule
$S \rightarrow \mathbf{id} = E;$	$gen(get(\mathbf{id}.lexeme)$ '=' $E.addr)$;
$S \rightarrow L = E;$	$gen(L.addr.base$ '[' $L.addr$ ']' '=' $E.addr);$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}Temp();$ $gen(E.addr$ '=' $E_1.addr$ '+' $E_2.addr);$
$E \rightarrow \mathbf{id}$	$E.addr = get(\mathbf{id}.lexeme);$
$E \rightarrow L$	$E.addr = \mathbf{new}Temp();$ $gen(E.addr$ '=' $L.array.base$ '[' $L.addr$ ']'$);$
$L \rightarrow \mathbf{id}[E]$	$L.array = get(\mathbf{id}.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = \mathbf{new}Temp();$ $gen(L.addr$ '=' $E.addr$ '*' $L.type.width);$
$L \rightarrow L_1[E]$	$L.array = L_1.array;$ $L.type = L_1.type.elem$ $t = \mathbf{new}Temp();$ $L.addr = \mathbf{new}Temp();$ $gen(t$ '=' $E.addr$ '*' $L.type.width);$ $gen(L.addr$ '=' $L_1.addr$ '+' $t);$
$B \rightarrow B_1 \vert\vert B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B_1.code \Vert label(B_1.false) \Vert B_2.code$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B_1.code \Vert label(B_1.true) \Vert B_2.code$
$B \rightarrow !B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1 \ \mathbf{rel} \ E_2$	$B.code = E_1.code \Vert E_2.code$ $\Vert gen($'if' $E_1.addr$ **rel** $E_2.addr$ 'goto' $B.true)$ $\Vert gen($'goto' $B.false)$
$B \rightarrow \ \mathbf{true}$	$B.code = gen($'goto' $B.true)$
$B \rightarrow \ \mathbf{false}$	$B.code = gen($'goto' $B.false)$

Production	Semantic Rule
$P \to S$	$S.next = newlabel()$ $P.code = S.code \| label(S.next)$
$S \to \textbf{assign}$	$S.code = \textbf{assign}.code$
$S \to \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$
$S \to \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$ $gen(\text{'goto'}\ S.next) \| label(B.false) \| S_2.code$
$S \to \textbf{while}\ (\ B\ )\ S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \| B.code$ $\| label(B.true) \| S_1.code \| gen(\text{'goto'}\ begin)$
$S \to S_1\ S_2$	$S_1.next = newlabel$ $S_2.next = S.next$ $S_1.code \| label(S_1.next) \| S_2.code$

[END OF APPENDICES]

[END OF EXAM]