DUBLIN CITY UNIVERSITY

SEMESTER 1 EXAMINATIONS 2016/2017

MODULE: CA4003 - Compiler Construction

PROGRAMME(S):

CASE - BSc in Computer Applications (Sft.Eng.)
CPSSD - BSc in ComputationalProblem SolvandSW Dev.
ECSAO - Study Abroad (Engineering and Computing)

YEAR OF STUDY: 4,O

EXAMINERS: Dr. David Sinclair (Ph:5510)
Prof. David Bustard
Dr. Ian Pitt

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.

Requirements for this paper (Please mark (X) as appropriate)

☐	*Log Tables*	☐	*Thermodynamic Tables*
☐	*Graph Paper*	☐	*Actuarial Tables*
☐	*Dictionaries*	☐	*MCQ Only - Do not publish*
☐	*Statistical Tables*	☐	*Attached Answer Sheet*

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 odd number of '1's and ends with a '1'.

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.

$S \rightarrow u\ A\ B\ z$
$A \rightarrow A\ v$
$A \rightarrow w$
$B \rightarrow C\ D$
$C \rightarrow y$
$C \rightarrow \epsilon$
$D \rightarrow x$
$D \rightarrow \epsilon$

[End Question 2]

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

[10 Marks]

Consider the following grammar for a language that allows the user to print the value of identifiers and the values returned by functions. *print* is a keyword and *id* is a valid identifier for both variables and functions.

Convert the grammar into an LL(1) grammar which recognises the same language:

$$\begin{array}{lll} S & \rightarrow & print(Arglist) \\ Arglist & \rightarrow & Arglist, Exp \\ Arglist & \rightarrow & Exp \\ Exp & \rightarrow & id \\ Exp & \rightarrow & id(Arglist) \end{array}$$

[End Question 3]

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

[10 Marks]

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

$S \rightarrow Ac$
$S \rightarrow BA$
$A \rightarrow ab$
$A \rightarrow cS$
$B \rightarrow d$
$B \rightarrow \epsilon$

[End Question 4]

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

[10 Marks]

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

$S \rightarrow E\$$
$E \rightarrow L = R$
$E \rightarrow R$
$L \rightarrow *R$
$L \rightarrow id$
$R \rightarrow L$

[End Question 5]

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

[10 Marks]

Construct the SLR(1) parse table for the following grammar and use it to determine whether or not the following grammar is SLR(1).

$S \rightarrow E\$$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow id$
$F \rightarrow (E)$

[End Question 6]

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

7(a) [6 Marks]

Convert the following source code into 3-address intermediate code using the syntax-directed approach given in the appendix. Assume that all variables are stored in 4 bytes.

```
max = 0;
i = 0;
while (i < 10)
{
  if (a[i] > max)
  {
    max = a[i];
  }
  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]

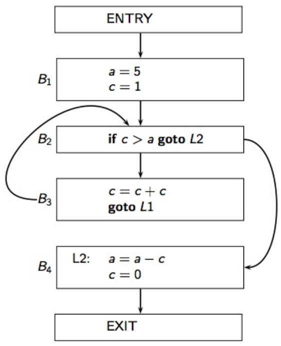
QUESTION 8 **[Total marks: 10]**

8(a) [4 Marks]

Describe how *Data Flow Analysis* is used to calculate *reaching definitions*.

8(b) [6 Marks]

For the following control flow graph, calculate the *reaching definitions* on exit from each block.



[End Question 8]

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

Consider the following basic block.

```
x = 5
a = x + 5
b = x + 3
v = a + b
a = x + 5
z = v + a
```

9(a) [5 Marks]

Calculate the *liveness* at each point in the basic block assuming that only z is live on exit.

9(b) [5 Marks]

From the *liveness* information in part (a), generate the interference graph and using the *graph colouring* algorithm allocate the variables to 3 registers.

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

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

[END OF APPENDICES]

[END OF EXAM]