DUBLIN CITY UNIVERSITY

AUGUST/RESIT 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: 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*
- [ ] *Graph Paper*
- [ ] *Dictionaries*
- [ ] *Statistical Tables*
- [ ] *Thermodynamic Tables*
- [ ] *Actuarial Tables*
- [ ] *MCQ Only - Do not publish*
- [ ] *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 even number of '1's and starts and ends 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.

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

[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).

E → T + E
E → T
T → int
T → int * T
T → (E)

[End Question 3]

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

[10 Marks]

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

$S \rightarrow Xa$
$S \rightarrow YX$
$X \rightarrow bc$
$X \rightarrow aS$
$Y \rightarrow d$
$Y \rightarrow \epsilon$

[End Question 4]

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

[10 Marks]

Determine whether or not the following grammar is LR(1) by constructing its LR(1) parse table.

$S' \rightarrow S\$$
$S \rightarrow a\ E\ a$
$S \rightarrow b\ E\ b$
$S \rightarrow a\ F\ b$
$S \rightarrow b\ F\ a$
$E \rightarrow e$
$F \rightarrow e$

[End Question 5]

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

[10 Marks]

Construct the LALR(1) parse table for the grammar in question 5 and use it to determine whether or not the grammar is LALR(1).

[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.

```
prod = 1;
i = 0;
while (i < 10)
{
  prod = prod * 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]

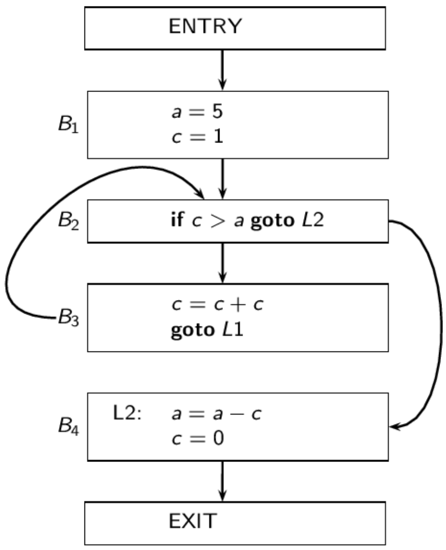
QUESTION 8 **[Total marks: 10]**

8(a) [4 Marks]

Describe how *Data Flow Analysis* is used to calculate the *liveness* of variables.

8(b) [6 Marks]

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



[End Question 8]

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

9(a) [5 Marks]

Generate the *directed acyclic graph* (DAG) representation of the following basic block.

```
w = v + x
v = v − y
x = x + y
z = v + x
```

9(b) [5 Marks]

Explain and demonstrate, using the DAG in part (a) of this question and assuming that v and z are not live on exit from the block, how *dead code* can be eliminated from a basic block.

[End Question 9]

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

10(a) [5 Marks]

What is a *Runtime Environment* ? What does the use of a *Runtime Environment* enable?

10(b) [5 Marks]

Using a piece of code as an example, describe how the stack frame in procedure A is modified when another procedure, procedure B, with arguments, is invoked from within procedure A.

[End Question 10]

[APPENDICES]

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

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

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

The attributes $S.code$, $E.code$ and $E.addr$ respectively represent the code for S, the code for E and the location that holds the value of E.

The function $gen(x\ \text{'='}y\ \text{'+'}\ z)$ will produce the 3-address code `x = y + z` where + is any binary operator $\mathbf{and}$ concatenates it with the previously generated code (incremental translation).

$C_1 \| C_2$ concatenates the code for C_2 after the code for C_2. The functions $newlabel()$ and $label()$ generate and instantiate labels.

[END OF APPENDICES]

[END OF EXAM]