Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

SEMESTER 1 EXAMINATIONS 2018/2019

MODULE: CA4003 - Compiler Construction

PROGRAMME(S):

CASE	BSc in Computer Applications (Sft.Eng.)
CPSSD	BSc in ComputationalProblem Solv&SW Dev.
ECSAO	Study Abroad (Engineering & Computing)

YEAR OF STUDY: 4,O

EXAMINERS:

Dr. David Sinclair	(Internal)	(Ph:5510)
Dr. Hitesh Tewari	(External)	External
Prof. Brendan Tangney	(External)	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 the alphabet $\{0,1\}$, write a regular expression that represents all the binary strings that contain an even number of '1' digits.

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]

Clearly show the step(s) involved in converting the following grammar into an equivalent LL(1) grammar which recognises the same language.

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

[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 an LL(1) grammar.

$S \rightarrow ABe$
$A \rightarrow dB|S|c$
$B \rightarrow AS|b$

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

[End Question 6]

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

7(a) [7 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.

```
min = a[0];
i = 1;
while (i < 10)
{
  if (a[i] < min) min = a[i];
  i = i+ 1;
}
```

7(b) [3 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]

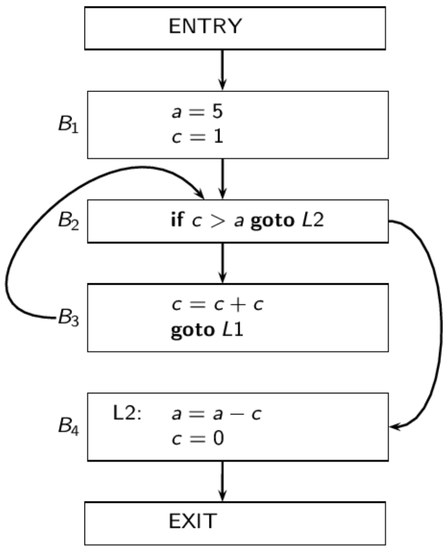
QUESTION 8 ***[Total marks: 10]***

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) [4 Marks]

The following code has been analysed and the *live-in* and *live-out* variables have been determined as indicated below. Draw the *interference graph* showing *move-related* edges.

	live-in	live-out
$g = a[j + 12]$	k, j	g, k, j
$h = k - 1$	g, k, j	g, h, j
$f = g * h$	g, h, j	f, j
$e = a[j + 8]$	f, j	e, f, j
$m = a[j + 16]$	e, f, j	e, f, j, m
$b = a[f]$	e, f, j, m	b, e, j, m
$c = e + 8$	b, e, j, m	b, c, j, m
$d = c$	b, c, j, m	b, d, j, m
$k = m + 4$	b, d, j, m	b, d, k, j
$j = b$	b, d, k, j	d, k, j

9(b) [6 Marks]

Using *Colouring by Simplification* assign the variables in the code in Question 9(a) to 4 registers.

[End Question 9]

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

10(a) [6 Marks]

Describe the layout of a *MIPS Stack Frame*.

10(b) [4 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
$S \rightarrow \mathbf{id} = E;$	$gen(get(\mathbf{id}.lexeme)$ '=' $E.addr)$;
$S \rightarrow L = E;$	$gen(L.addr.base$ '[' $L.addr$ ']' '=' $E.addr);$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}Temp();$ $gen(E.addr$ '=' $E_1.addr$ '+' $E_2.addr);$
$E \rightarrow \mathbf{id}$	$E.addr = get(\mathbf{id}.lexeme);$
$E \rightarrow L$	$E.addr = \mathbf{new}Temp();$ $gen(E.addr$ '=' $L.array.base$ '[' $L.addr$ ']'$);$
$L \rightarrow \mathbf{id}[E]$	$L.array = get(\mathbf{id}.lexeme);$ $L.type = L.array.type.elem;$ $L.addr = \mathbf{new}Temp();$ $gen(L.addr$ '=' $E.addr$ '*' $L.type.width);$
$L \rightarrow L_1[E]$	$L.array = L_1.array;$ $L.type = L_1.type.elem$ $t = \mathbf{new}Temp();$ $L.addr = \mathbf{new}Temp();$ $gen(t$ '=' $E.addr$ '*' $L.type.width);$ $gen(L.addr$ '=' $L_1.addr$ '+' $t);$
$B \rightarrow B_1 \vert\vert B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B_1.code \Vert label(B_1.false) \Vert B_2.code$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B_1.code \Vert label(B_1.true) \Vert B_2.code$
$B \rightarrow !B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1 \ \mathbf{rel} \ E_2$	$B.code = E_1.code \Vert E_2.code$ $\Vert gen($'if' $E_1.addr \ \mathbf{rel} \ E_2.addr$ '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
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \| label(S.next)$
$S \rightarrow \textbf{assign}$	$S.code = \textbf{assign}.code$
$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$
$S \rightarrow \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$ $gen(\text{'goto'}\ S.next) \| label(B.false) \| S_2.code$
$S \rightarrow \textbf{while}\ (\ B\ )\ S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \| B.code$ $\| label(B.true) \| S_1.code \| gen(\text{'goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel$ $S_2.next = S.next$ $S_1.code \| label(S_1.next) \| S_2.code$

[END OF APPENDICES]

[END OF EXAM]