DCU

DUBLIN CITY UNIVERSITY

AUGUST/RESIT 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 all 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 start with a '0' digit and end with two consecutive '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.

E $\rightarrow$ T E'
E' $\rightarrow$ + T E' | ϵ
T $\rightarrow$ F T'
T' $\rightarrow$ * F T' | ϵ
F $\rightarrow$ (E) | id

[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 (you may assume that the grammar is unambiguous).

F $\rightarrow$ T * F
F $\rightarrow$ T
T $\rightarrow$ int
T $\rightarrow$ int * T
T $\rightarrow$ (F)

[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.

$Z \rightarrow d|XYZ$
$Y \rightarrow c|\epsilon$
$X \rightarrow Y|a$

[End Question 4]

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

[10 Marks]

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

$S' \rightarrow S$
$S \rightarrow aBc$
$S \rightarrow bCc$
$S \rightarrow aCd$
$S \rightarrow bBd$
$B \rightarrow e$
$C \rightarrow e$

[End Question 5]

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

[10 Marks]

Construct the LALR(1) parse table for the grammar in question 5 and using it 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.

```
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 the liveness of variables.

8(b) [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.

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

[End Question 8]

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

9(a) [5 Marks]

Generate the *directed acyclical 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 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
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \| label(S.next)$
$S \rightarrow \textbf{assign}$	$S.code = \textbf{assign}.code$
$S \rightarrow \textbf{if} (B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$
$S \rightarrow \textbf{if} (B) S_1 \textbf{ else } S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \| label(B.true) \| S_1.code$ $gen(\text{'goto'}\ S.next) \| label(B.false) \| S_2.code$
$S \rightarrow \textbf{while} (B) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \| B.code$ $\| label(B.true) \| S_1.code \| gen(\text{'goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel$ $S_2.next = S.next$ $S_1.code \| label(S_1.next) \| S_2.code$

[END OF APPENDICES]

[END OF EXAM]