Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

AUGUST/RESIT EXAMINATIONS 2021/2022

MODULE: CA4003 - Compiler Construction

PROGRAMME(S):

CASE	BSc in Computer Applications (Sft.Eng.)
ECSA	Study Abroad (Engineering & Computing)
ECSAO	Study Abroad (Engineering & Computing)

YEAR OF STUDY: 4,O,X

EXAMINERS:

Dr. David Sinclair (Internal) (Ph:5510)

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]***

Regular Expressions & DFAs

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]***

FIRST & FOLLOW

[10 Marks]

Calculate the FIRST and FOLLOW sets for the following grammar, where S is the start (goal) state.

S $\rightarrow$ A b
A $\rightarrow$ a | B | ϵ
B $\rightarrow$ b | ϵ

[End Question 2]

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

Equivalent LL Grammar

[10 Marks]

Consider the following grammar for a language where *print* is a keyword and *I* is a valid identifier for both variables and functions.

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

$$
\begin{aligned}
S &\rightarrow print(L) \\
L &\rightarrow L, E \\
L &\rightarrow E \\
E &\rightarrow I \\
E &\rightarrow I(L)
\end{aligned}
$$

[End Question 3]

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

LL(1) Grammar

[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.

$S \rightarrow Bc$
$S \rightarrow DB$
$B \rightarrow ab$
$B \rightarrow cS$
$D \rightarrow d$
$D \rightarrow \epsilon$

[End Question 4]

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

LR(1) Grammar

[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]***

LALR(1)

[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]***

Intermediate Code

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.

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

7(b) [4 Marks]

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

[End Question 7]

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

Optimisation

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]***

Register Allocation & Graph Colouring

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. $\underline{\mathbf{Clearly\ describe}}$ each step in the process.

[End Question 9]

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

Runtime Environment

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