Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

SEMESTER 1 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 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]*

FIRST & FOLLOW

[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]***

Equivalent LL Grammar

[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 $\rightarrow$ id = Exp
S $\rightarrow$ id (Arglist)
Arglist $\rightarrow$ Arglist , Exp
Arglist $\rightarrow$ Exp
Exp $\rightarrow$ id (Arglist)
Exp $\rightarrow$ id

[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.

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

[End Question 4]

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

LR(1) Grammar

[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):

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

[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 using it 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 intermediate code using the syntax-directed approach given in the appendix. Assume that all variables are stored in 4 bytes.

```
sum = 0;
i = 0;
while (i < 9)
{
  diff = a[i] - a[i+1];
  sum = sum + diff;
  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]***

Data Flow Analysis & Liveness

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.

$$
\begin{aligned}
&t_1 = j + 4\\
&g = a[t_1]\\
&h = k - 1\\
&f = g * h\\
&t_2 = j + 12\\
&e = a[t_2]\\
&t_3 = j + 8\\
&m = a[t_3]\\
&b = a[f]\\
&c = e + 24\\
&d = c\\
&k = m + 4\\
&j = b
\end{aligned}
$$

[End Question 8]

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

Register Allocation & Graph Colouring

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]***

Runtime Environment

[10 Marks]

With the aid of example code, describe the *Visitor* pattern. Describe how the *Visitor* pattern can be used to perform 2 common tasks during *compilation*.

[End Question 10]

APPENDICES

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

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