DCU

DUBLIN CITY UNIVERSITY

SAMPLE PAPER

MODULE: CA4003 - Compiler Construction

PROGRAMME(S):

CASE - BSc in Computer Applications (Sft.Eng.)

YEAR OF STUDY: 4

EXAMINERS: Dr David Sinclair

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*	☐	*Thermodynamic Tables*
☐	*Graph Paper*	☐	*Actuarial Tables*
☐	*Dictionaries*	☐	*MCQ Only - Do not publish*
☐	*Statistical Tables*	☐	*Attached Answer Sheet*

Note: In the following questions, non-terminal symbols are represented by strings starting with an upper case letter and terminal symbols by strings starting with a lower case letter. 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 at least two consecutive '1's, for example 0100110, 0111, 00010011001.

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 gammar..

$S \rightarrow u\ B\ D\ z$
$B \rightarrow B\ v$
$B \rightarrow w$
$D \rightarrow E\ F$
$E \rightarrow y$
$E \rightarrow \epsilon$
$F \rightarrow x$
$F \rightarrow \epsilon$

[End Question 2]

QUESTION 3 ***[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.

$$
\begin{aligned}
&S \rightarrow A \\
&A \rightarrow Ba \mid C \\
&B \rightarrow aC \mid b \\
&C \rightarrow B
\end{aligned}
$$

[End Question 3]

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

[10 Marks]

Construct the SLR parse table for the grammar in Question 3, and using this table determine whether or not it is an SLR grammar.

[End Question 4]

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

[10 Marks]

Construct the LR(1) parse table for the following grammar and use it to determine whether or not the following grammar is LR(1).

$$
\begin{aligned}
&S' \rightarrow S\$ \\
&S \rightarrow A \\
&S \rightarrow xb \\
&A \rightarrow aAb \\
&A \rightarrow B \\
&B \rightarrow x
\end{aligned}
$$

[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) [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.

```
max = −999;
i = 0;
while (i < 10)
{
  if (a[i] > max}
  {
    max = 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) [5 Marks]

Describe how *Data Flow Analysis* is used to calculate *reaching definitions*. Briefly indicate how this can be used to detect if undefined variable are passed into functions.

8(b) [5 Marks]

For the following intermediate code, construct the *control flow graph* and calculate the *reaching definitions*.

	$a = 10$
	$b = 11$
	if $e == 1$ **goto** L1
	$a = 1$
	$b = 2$
	goto L2
L1:	$c = a$
	$a = 4$
L2:	

[End Question 8]

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

[10 Marks]

Construct a directed acyclic graph for the following code fragment which identifies all common sub-expressions.

```
G := C * (A + B) + (A + B);
C := A + B;
A := (C * D) + (E - F);
```

[End Question 9]

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

10(a) [5 Marks]

Describe the design of a symbol table that efficiently handles scope. Clearly describe and justify the data structures used.

10(b) [5 Marks]

Describe how target code is generated from (optimised) intermediate code.

[End Question 10]

[END OF EXAM]

[APPENDICES]

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

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

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

[END OF APPENDICES]