The Open University of Sri Lanka
Faculty of Engineering Technology
Department of Electrical and Computer Engineering

019

Study Programme	: Bachelor of Software Engineering in Honours
Name of the Examination	: Final Examination
Course Code and Title	: **EEX6363 Compiler Construction**
Academic Year	: 2021/22
Date	: 17th February 2023
Time	: 14:00 – 17:00 hrs.
Duration	: **3 hours**

General Instructions

1. Read all instructions carefully before answering the questions.
2. This question paper consists of **four (4)** questions in **four (4)** pages.
3. Answer **all** questions in **Section A** and any **TWO** questions from **Section B.**
4. Answer for each question should commence from a new page.
5. Answers should be in clear handwriting and do not use Red colour pen.
6. Clearly state your assumptions, if any.
7. This is a **Closed Book Test** (CBT).

Section A – Answer all questions [60 marks]

Q1 The following section presents the description of a language, called **MulCalc** for multiplying numerical values.

MulCalc is a very simple language that accommodates two forms of numerical data types (double and integer), allows computation and printing of numerical values, and offers a small set of variable names to hold the results of computations. As most programming languages, the conversion from integer type to double type is accomplished automatically in **MulCalc**. The production rules for the grammar of this language are given below.

1	S	$\rightarrow$	*dcls stmts \$*
2	*dcls*	$\rightarrow$	*dcl dcls*
3		|	ε
4	*dcl*	$\rightarrow$	*intdcl id*
5		|	*doubledcl id*
6	*stmts*	$\rightarrow$	*stmt stmts*
7		|	ε
8	*stmt*	$\rightarrow$	*id assign val expr*
9		|	*print id*
10	*val*	$\rightarrow$	*id*
11		|	*intnum*
12		|	*doublenum*
13	*expr*	$\rightarrow$	*mul val expr*
14		|	*div val expr*
15		|	ε

where S is the start symbol, ε denotes the empty string and the special symbol *\$* is a terminal which represents the end of the input stream.

The specification of tokens in **MulCalc** is accomplished by associating a regular expression with each token, as shown below.

Terminal	Regular Expression
intdcl	i
doubledcl	d
print	p
id	$[a-c] \mid [e-h] \mid [j-o] \mid [q-z]$
assign	=
mul	×
div	/
intnum	$[0-9]^+$
doublenum	$[0-9]^+ . [0-9]^+$
blank	$(\text{" "})^+$

(a) Define the grammar for the given language **MulCalc** by clearly stating the *terminals* and *non-terminals*. [06]

(b) Write the input stream for this language that satisfies the following requirement. Your input stream should be syntactically valid. [10]

> *"Declare two numbers, the first one is integer type and second one is double type, then assign values and getting the multiplication of two assigned values, finally print the result."*

Note: the sequence of the data type is important.

(c) Validate the input stream written in (b) using the grammar of **MulCalc**. [10]

(d) Construct the non-deterministic finite automata for the input obtained in (b). [06]

(e) Draw the parse tree for the input stream written in (b). [08]

(f) Draw the abstract syntax tree (AST) for the parse tree obtained in (e). Choose whatever variables and numerical values from the given specification of tokens. [08]

(g) Re-draw the AST drawn in (f) after applying semantic analysis. [04]

(h) Write LEX implementation syntax for the language **MulCalc**. [08]

Section B – Answer any TWO questions [40 marks]

Q2 Four production rules (R1 to R4) for a specific grammar are given by

$$\begin{array}{lll} \text{R1:} & \text{FCTR} & \rightarrow & \text{NEST } \mathit{id} \text{ NEST } \mathit{num} \\ \text{R2:} & \text{FCTR} & \rightarrow & \text{LIST } \mathit{num} \text{ LIST } \mathit{id} \\ \text{R3:} & \text{NEST} & \rightarrow & \varepsilon \\ \text{R4:} & \text{LIST} & \rightarrow & \varepsilon \end{array}$$

where CAPITAL terms are non-terminals (three) while all others are terminals, and the starting term is FCTR.

(a) Construct the FIRST and FOLLOW sets for the grammar above. [03]

(b) Construct the LL(1) top-down predictive parsing table for this grammar. [06]

(c) Using the definition of LL(1), explain why the grammar is or not LL(1). [03]

(d) Convert the given grammar into Chomsky normal form. [08]

Q3

(a) Explain, two differences between top-down and bottom-up parsing methods. [04]

(b) State, what is an LR(0) item. [03]

(c) Construct an SLR parsing table for the grammar below: [10]

$$S \rightarrow L = R \mid R$$
$$L \rightarrow * R \mid id$$
$$R \rightarrow L$$

where S, L and R are non-terminals while all others are terminals.

(d) Explain, whether the given grammar in (c) is SLR(1) type or not. [03]

Q4

(a) Explain the followings briefly:

i) Any two factors affecting code generation phase. [04]

ii) With an example, what is mean by three-address code representation? [03]

iii) Importance of the *symbol table manager* and *error handler* in a compiler. [06]

iv) Why we need a code optimization in a compiler? [03]

(b) Optimize the following code-segment written in C language: [04]

<u>**Note**</u>: assume that this code-segment is perfectly executed.

```
int j, x, y, r;
x = 0;
y = 1;
r = 1;
for ( j = 0; j <= N; j++ )
{
    x = (j + 1) * (j + 1) * (j + 1) * x + (8 * a / b) * j;
    x = x + r * y;
}
```

-- End –