00073

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

Study Programme	: Bachelor of Software Engineering in Honours
Name of the Examination	: Final Examination
Course Code and Title	: **EEX6363 Compiler Construction**
Academic Year	: 2023/2024
Date	: 09th February 2025
Time	: 09:30 – 12:30 hrs.
Duration	: **3 hours**

General Instructions

1. Read all instructions carefully before answering the questions.
2. This question paper consists of **three (3)** questions on **four (4)** pages.
3. Answer **all three** questions.
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).

Q1

Suppose the grades for *Compiler Construction* course are stored in a file called, 'GradeFile' with the following format.

- The file contains one or more student records.
- Each record includes student's name, registration number, and zero or more grades for different assessments, separated by commas.
- Grades are represented as integers, followed by zero or more caret (^) symbols to indicate the number of days for late submissions.

The above format simplifies record-keeping and clarifies student's performance. The course advisor decided to write a parser to read the grades, making it easy to summarize. This file is updated after every assignment is graded and the context free grammar (CFG) is developed as follows.

file	→	*record tail $*
tail	→	*file* | ε
record	→	*name regno optGrades*
optGrades	→	*grades* | ε
grades	→	*oneGrade* | *oneGrade comma grades*
oneGrade	→	*intval optLate*
optLate	→	*symbols* | ε
symbols	→	*caret* | *symbols caret*

where *file* is the start term, ε denotes the empty string and the special token, *$* that represents the end of the sequence of tokens.

(a) Define the grammar for the given GradeFile by clearly stating the 'terminals' and 'non-terminals'. [08]

(b) Although the given CFG is not LL(1), a predictive parser can parse some correct inputs.

I How do you check whether the given CFG is LR(0) or not, explain briefly. [04]

II Write the **shortest** input stream (as a sequence of tokens) which syntactically valid for the GradeFile. [04]

III Validate the input stream written in (b)-II by using the grammar above. [04]

IV Draw the parse tree that the parser would build for the input you gave for part (b)-II. Do not include the token, *$* in the parse tree. [08]

(c) Since the given CFG is not LL(1), the parser encounters issues when parsing some inputs. That means the parser would not know how to continue creating the parse tree in the top-down method, even if a sequence of tokens is valid input.

I Write a sequence of tokens (a prefix of a valid input) which the parser starts to be stuck when selecting the corresponding CFG rules given. [06]

II Draw the (partial) parse tree that the predictive parser would have built before being stuck. [08]

III Write the CFG rules that the predictive parser cannot choose between to continue to grow the parse tree. [06]

(d) Further issues are identified in the given CFG as follows.

I The CFG has not been left factored. Find the CFG rules with a common prefix and transform them by doing **left factoring**. [06]

II This CFG has left recursion. Find the CFG rules that cause this and transform them to **remove the left recursion**. [06]

Q2

(a) The production rules for a certain context free grammar (CFG) are given below.

$$stmt \rightarrow id = exp\ ;$$
$$exp \rightarrow id\ \ tail \mid id$$
$$tail \rightarrow +\ exp \mid -\ exp$$

where *stmt* is the starting term of the grammar.

I Define the Chomsky Normal Form (CNF) for CFGs. [02]

II Convert the given CFG into CNF after carefully identifying the terminals and non-terminals. Show **all** the steps. [08]

(b) Compute the FIRST and FOLLOW sets for the non-terminals of this grammar. [10]

$$prog \rightarrow \{\ stmts\ \}\ eof$$
$$stmts \rightarrow stmt\ \ stmts \mid \varepsilon$$
$$stmt \rightarrow id = exp\ ; \mid if\ (\ exp\ )\ stmt$$
$$exp \rightarrow id\ tail$$
$$tail \rightarrow +\ exp \mid -\ exp \mid \varepsilon$$

Q3

(a) Consider the grammar by the following thirteen production rules. [10]

$$dnum \rightarrow num\ .\ num$$
$$num \rightarrow num\ digit \mid digit$$
$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

where dot (.) indicates the decimal point of a floating-point representation.

Write an attribute grammar for the floating-point representation in the table-form.

Production	Semantic rules

Hint:

Use a *count* attribute to count the number of digits to the right of the decimal point.

(b) Explain the following briefly:

I Any two factors affecting code generation phase. [03]

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. [04]

-- End –