LAIKIPIA UNIVERSITY

UNIVERSITY EXAMINATIONS

FIRST SEMESTER 2025/2026 ACADEMIC YEAR

THIRD YEAR EXAMINATION FOR THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND BACHELOR OF SCIENCE IN INFORMATION COMMUNICATION AND TECHNOLOGY

COMP 311: COMPILER CONSTRUCTION DESIGN

STREAM: R ***TIME: 2 HRS***

DAY: WEDNESDAY [11.30 – 13.30 P.M] ***DATE: 04/02/2026***

THIS QUESTION PAPER CONSISTS OF SEVEN (7) PAGES

PLEASE DO NOT OPEN UNTIL THE INVIGILATOR SAYS SO.

INSTRUCTIONS:

QUESTION ONE (30 MARKS)

a) List the **six** main phases of a typical compiler in their logical order of execution. [**3 Mark**]

b) Differentiate between a compiler and an interpreter, providing one advantage of each. [**2 Marks**]

c) Explain the concept of "bootstrapping" in compiler design. Why is it a powerful technique for creating compilers for new programming languages? [**3 Marks**]

d) Define the terms "token" and "lexeme," and provide a concrete example for an integer literal. **[3 Marks]**

e) Construct a Deterministic Finite Automaton (DFA) that recognizes the identifiers in a language where an identifier must start with a letter and can be followed by any number of letters or digits. **[3 Marks]**

f) Contrast the fundamental principles of top-down (e.g., LL(1)) and bottom-up (e.g., LR(1)) parsing. Include one key advantage of each approach. [**3 Marks**]

g) Compute the FIRST and FOLLOW sets for the given grammar rules to determine if it is suitable for a predictive parser. [3 Marks]

- A -> B c | d
- B -> e | ε

h) Given a simple expression grammar, draw the Abstract Syntax Tree (AST) for the expression a + b * c. How does an AST differ from a concrete parse tree? [**3 Marks**]

i) What is the primary purpose of the semantic analysis phase? List two specific tasks it performs. [**2 Marks**]

j) Evaluate the trade-off between the aggressiveness of code optimization and the time it takes to compile a program. In which scenarios might a developer choose to disable optimizations? [**3 Marks**]

k) Differentiate between Just-in-Time (JIT) and Ahead-of-Time (AOT) compilation. Name one programming language commonly associated with each. [**2 Mark**s]

l) Propose a reason why compiler security is a critical concern in modern software development. Provide one example of a vulnerability that could be introduced by poorly generated code. [**3 Marks**]

QUESTION TWO (20 MARKS)

You are developing a static analyzer for a financial programming language that must detect potential errors at compile time. The language has complex type rules and financial-specific semantics

```
class Portfolio:
    def __init__(self):
        self.holdings = {}
        self.cash = 0.0

    def add_stock(self, symbol: str, quantity: int, price: float):
        if symbol not in self.holdings:
            self.holdings[symbol] = 0
        self.holdings[symbol] += quantity
        self.cash -= quantity * price

    def calculate_value(self, current_prices: dict) -> float:
        total_value = self.cash
        for symbol, quantity in self.holdings.items():
            if symbol in current_prices:
                total_value += quantity * current_prices[symbol]
            # Missing else case - what if symbol not in current_prices?
        return total_value

def risky_trade(portfolio: Portfolio, threshold: float) -> bool:
    value = portfolio.calculate_value(current_prices={})
    # Type error: current_prices is empty dict
    # Logical error: always returns False
    if value > threshold:
        return True
```

Based on the above code snippet

a) Develop a static analysis tool that detects financial-specific semantic issues:

i. Potential division by zero in financial calculations

ii. Unchecked null/missing values in price data

iii. Portfolio concentration risks

iv. Cash balance validation

Provide pseudocode for your analysis and show how it would flag issues in the given code.

[10 Marks]

b) Implement type-checking rules for the financial language. Create a set of rules that would catch:

- Type mismatches in arithmetic operations
- Missing return statements
- Uninitialized variables
- Function call argument mismatches **[10 Marks]**

.

QUESTION THREE (20 MARKS)

You are a compiler engineer at a company that develops embedded systems for medical devices. The company utilizes a custom programming language called "MediC," which is similar to C but with specific constraints designed for safety and efficiency. The target hardware has limited memory and processing power, so code optimization is critical. You are tasked with optimizing a critical function that calculates patient vital statistics. The function involves loops, conditionals, and array operations. The initial code has performance issues, and you need to apply various optimization techniques to enhance its performance.

Here's a snippet of the code before optimization:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        if (i % 2 == 0) {
            data[i][j] = data[i][j] * factor + offset;
        } else {
            data[i][j] = data[i][j] * factor - offset;
        }
    }
}
```

a) Analyze the given code snippet and identify at least three optimization opportunities. For each opportunity, explain the type of optimization that could be applied and why it would improve performance. [**6 Marks**]

b) Evaluate the trade-offs between code size and execution speed when applying optimizations in an embedded system context. Discuss whether aggressive optimization is always beneficial for medical devices, considering both safety and reliability. [**4 Marks**]

c) Apply loop optimization techniques to transform the code. Specifically, perform loop unrolling and loop invariant code motion on the inner loop. Show the optimized code after these transformations and explain how each change contributes to performance improvement. [**10 Marks**]

QUESTION FOUR (20 MARKS)

You are developing a lexer for a new configuration language that supports mathematical expressions, nested comments, and domain-specific tokens. Below is the code to tokenize:

```
# Financial portfolio configuration
portfolio {
    name: "Retirement Fund"
    target_return: 7.5%  // Annual return target
    constraints: {
        max_sector_weight: 25%
        min_diversification: 15 stocks
    }

    # Mathematical expression for risk calculation
    risk_score: sqrt(beta^2 * market_variance + specific_risk^2)

    /* Multi-line comment
       with nested expressions */
    rebalance_trigger: (current_allocation - target_allocation) > 5%
}
```

a) Implement a lexical analyzer using Flex that handles:

- Nested comments
- String interpolation
- Context-sensitive tokens (e.g., `%` as percentage vs. modulus)

Provide the Flex specification and explain how it handles each feature. [**10 Marks**]

b) Design regular expressions for the language's tokens, including:

- Identifiers with Unicode support
- Numbers (integers, floats, percentages)
- Strings with escape sequences
- Mathematical operators
- Comments (both single-line and multi-line [**8 marks**]

a. Analyse the challenges in tokenising mathematical expressions mixed with configuration syntax. How would your lexer distinguish between different uses of symbols like `-` (minus vs. range operator) and `*` (multiplication vs. wildcard)? [**2 Marks**]

QUESTION FIVE (20 MARKS)

You are a research engineer at a tech company exploring next-generation compilation techniques. Your team is investigating how modern compiler trends can improve performance for emerging workloads in AI, quantum computing, and secure systems. Design a hybrid AOT-JIT compilation strategy for AI workloads. [**10 Marks**]

a) i) Explain when to use AOT vs JIT compilation for different parts of an AI model

ii) Describe how profiling data guides optimisation decisions in JIT compilation

iii) Propose a caching mechanism for frequently executed computation patterns

iv) Analyse the trade-offs between startup time and peak performance

b) Evaluate the impact of three emerging compiler trends:

- **WebAssembly** as a universal compilation target
- **MLIR (Multi-Level IR)** for domain-specific compilation
- **Differentiable programming** in compilers

For each trend, discuss: Key benefits and limitations in production systems, Potential applications in industry products, and Long-term implications for compiler architecture

[**10 Marks**]