IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

Department: **Computer Science** Program: **BS**

COMPILER CONSTRUCTION

Announced date: MAY, 15, 2025 Due Date: MAY 30, 2025 Total Marks = 10

Complex Computing Problem (CCP)		
Mapped CLO	**SDG**	**Complex Problem Solving Mapped**
CLO2 CLO3	4	WP1 (Range of Conflicting Requirements) WP2 (Depth of Analysis required) WP3 (Depth of Knowledge required)

Problem Statement:

You are part of a team assigned with the exciting challenge of designing and implementing the core components of a compiler for a programming language that you will create from scratch. This project will involve defining the language's unique syntax and grammar, tailoring it to address a specific problem or domain, and then developing the necessary tools to process the source code written in that language. As part of the development process, you will implement the essential components of the compiler, which will include creating a tokenizer, a state machine, a lexical analyzer, and a syntax analyzer. Each of these components will be integral to ensuring that your language is processed efficiently and accurately, turning it into a fully functional system for real-world use.

Objectives:

1. **Tokenizer**: Design rules to break down your language's source code into tokens, identifying its unique keywords, operators, data types, and delimiters. The tokenizer should reflect the distinct features of your language and handle edge cases like nested expressions or special symbols.
2. **State Machine**: Construct a finite state machine tailored to the lexical analysis of your language. The machine should manage transitions between states, such as identifiers, constants, operators, and string literals, while robustly handling errors like invalid characters.
3. **Lexical Analyzer**: Implement a lexical analyzer that generates a token stream from the source code based on your tokenizer and state machine. It should include meaningful error messages and support detailed debugging features like line numbers and contextual information.
4. **Syntax Analyzer**: Develop a syntax analyzer to validate the grammatical structure of the token stream according to the rules of your language. The parser should detect errors, such as mismatched brackets or incorrect statement order, and employ a parsing strategy (e.g., top-down, bottom-up) that best fits your language design.

IQRA UNIVERSITY IU

FACULTY OF ENGINEERING SCIENCES AND TECHNOLOGY

Constraints:

1. Each group must design a completely unique programming language with distinct syntax, grammar, and features tailored to a specific problem domain.
2. The project must include a tokenizer, state machine, lexical analyzer, and syntax analyzer, all implemented from scratch without prebuilt libraries.
3. Robust error handling is required, with meaningful error messages and detailed documentation of grammar and state machine transitions.

Expected Outcome:

1. **Create a Unique Language**: Each group must innovate and define a new programming language with its syntax, features, and design philosophy tailored to its chosen problem domain.
2. **Grammar and Syntax Rules**: Document the language's grammar using formal techniques like Backus-Naur Form (BNF) or Extended BNF.
3. **Error Detection**: Design mechanisms to handle lexical and syntax errors gracefully, providing clear feedback to users.
4. **Test Programs**: Write several test programs in your language to demonstrate its capabilities, covering valid use cases and error scenarios.
5. **Documentation and Justification**: Submit a report detailing the language design, its unique features, and the rationale behind its syntax and grammar.

Deliverables:

A detailed report that must include the following:

1. A fully implemented compiler consisting of a tokenizer, state machine, lexical analyzer, and syntax analyzer, along with a language specification document.
2. At least five sample programs in the custom language, demonstrating functionality and error handling.
3. A final report and presentation showcasing the language design, implementation, and a live demo of the compiler.

CCP Attributes mapped		
Attributes of Complex Problem Solving		**Justification**
WP1	Range of Conflicting Requirements	Designing the compiler involves conflicting requirements, such as balancing accuracy and efficiency, choosing between simplicity for error handling and complexity for advanced parsing, and integrating distinct components (tokenizer, state machine, lexical analyzer, and syntax analyzer) seamlessly.
WP2	Depth of Analysis required	Compiler construction allows for multiple approaches to parsing and tokenization (e.g., recursive descent vs. table-driven parsing). Each method has trade-offs in terms of efficiency and error detection, requiring in-depth analysis to

		identify the best fit for the designed programming language.
WP3	Depth of Knowledge required	The project demands comprehensive knowledge of compiler construction principles, including finite automata, lexical analysis, parsing algorithms, and error recovery techniques. Students must also understand how these elements interconnect to form a functional compiler.
WP4	Familiarity of issues	NA
WP5	Extent of applicable codes	NA
WP6	Extent of stakeholder involvement and level of conflicting requirements.	NA
WP7	Consequences	NA
WP8	Interdependence	NA