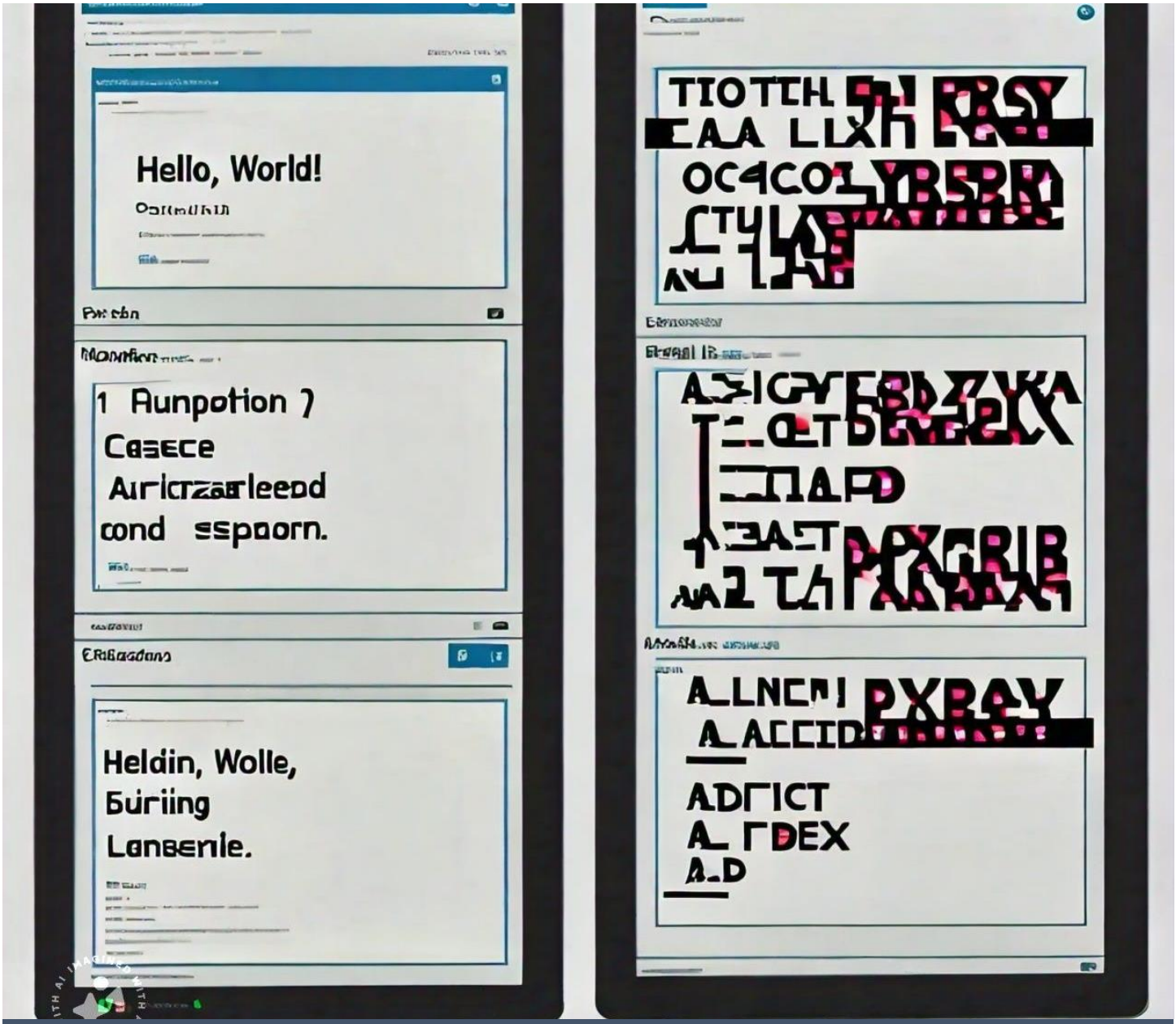


Compiler Chronicles: Revealing the Inner Workings of Lexical Analysis, Parsing, and Interpretation

Compiler Construction

115057

Submitted to: Sir Ubaid Zaman

PROJECT MEMBERS:

11696 Muhammad Abad Khan

11965 Sadia Naz

What is Compiler?

A compiler is a fundamental tool in computer science that translates source code written in a high-level programming language into machine code that can be understood and executed by a computer's processor. The purpose of a compiler is to facilitate the execution of programs by converting human-readable code into a format that a computer can execute directly.

Here are some key purposes of a compiler:

1. ***Translation:*** The primary purpose of a compiler is to translate source code written in a high-level programming language into machine code , which consists of low-level instructions that can be executed by the computer's processor.
2. ***Optimization:*** Compilers often perform various optimizations on the code during the translation process. These optimizations aim to improve the performance of the resulting executable code, such as by reducing execution time, minimizing memory usage, or improving power efficiency.
3. ***Error Detection:*** Compilers analyze the source code for syntax errors, semantic errors, and other potential issues before generating the executable code. This helps programmers identify and fix errors early in the development process, improving the reliability and correctness of the last program.

PURPOSE OF OUR COMPILER PROJECT:

Our Compiler Construction project involves the development of a compiler for python programming language. Python is a simple language designed to facilitate mathematical operations like (add, sub etc) , bitwise operators, mathematical computations (factorization, prime numbers), and basic input/output operations. The compiler consists of three main components:

1. LEXER
2. PARSER
3. INTERPRETER

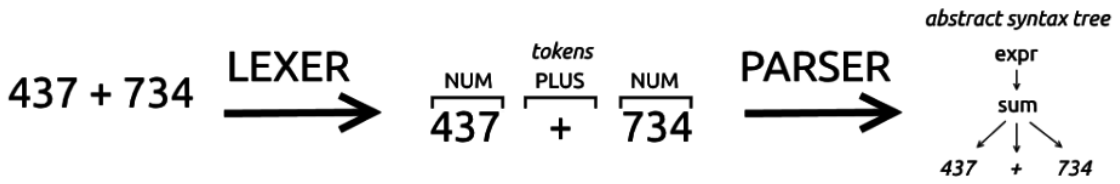
Each component plays a crucial role in the compilation process, from tokenizing input code to executing instructions.

1. **LEXER:**

The lexer component of our compiler is responsible for tokenizing the input code. It identifies keywords, operators, identifiers, and literals in the source code and generates tokens based on predefined token patterns. The lexer utilizes regular expressions to match patterns in the input code and convert them into tokens. For example, the lexer tokenizes the input code VAR num = 10 into tokens representing a variable declaration.

2. **PARSER:**

The parser component of the our compiler parses the tokens generated by the lexer and constructs an abstract syntax tree (AST) representing the syntactic structure of the code. It applies parsing rules for each language construct, such as variable declarations, mathematical operations, and input/output statements. The parser traverses the token stream and generates an AST that serves as an intermediate representation of the code. For example, the parser constructs an AST for the input code ” PLUS num num1” representing an addition operation.



3. **INTERPRETER:**

The interpreter component of the our compiler interprets the AST generated by the parser and executes the instructions specified in the code. It interacts with the lexer and parser to obtain tokens and parse the code, and maintains a dictionary of variables to store values during execution. The interpreter implements operations such as variable assignment, arithmetic operations, input/output. For example, the interpreter executes the addition operation “PLUS num num1” and outputs the result.

MATHEMATICAL OPERATORS:

```
VAR num = 10
VAR num1 = 50
PLUS num num1    # 10 + 50 = 60
WRITE num1      # Output: 60
MINUS num1 num    # 60 - 10 = 50
WRITE num       # Output: 50
WRITE num1      # Output: 60
```

BITWISE OPERATORS:

```
VAR num = 10
VAR num1 = 50
AND num num1
WRITE num1
OR num num1
XOR num num1
NOT num1
```

FACTORIZATION & PRIME NUMBERS:

```
VAR num2 = 5
FACT num2
WRITE num2
IsPRIME num
```

KEYWORDS:

VAR	Declares a variable.
PLUS	Performs addition.
MINUS	Performs subtraction.
MULTIPLY	Performs multiplication.
DIVIDE	Performs division.
SQUARE	Calculates the square of a number.
WRITE	Outputs a value.
READ	Takes user input.
Bitwise Operator	AND OR NOT XOR
Factorization	Performs factorization
Prime Numbers	Takes prime numbers

CONCLUSION:

In conclusion, our compiler construction project demonstrates the process of developing a simple compiler for programming language. The lexer, parser, and interpreter components work together to process source code, analyze its structure, and execute the resulting instructions. It optimized and enhanced the functionality.