# Project Report: Roman Urdu Compiler

## Overview

This report details the development and functionality of a Roman Urdu Compiler. The project explores the essential components and methodologies involved in compiler construction, including lexical analysis, parsing, error handling, and interpretation.

### Team Members

* Junaid Ameer Khan
* Samreen Fatima Kazmi
* Laraib Sultana

### Supervisor

**Instructor**: Ramzan Shahid

## Objectives

The primary goal of this project is to create a compiler that:

* Accepts Roman Urdu as input.
* Processes the input through lexing, parsing, and interpreting stages.
* Executes the input code and provides meaningful outputs or error messages.

## Workflow

### 1. Input

The user provides a line of code written in Roman Urdu.

### 2. Lexical Analysis (Lexer)

The lexer processes the input to:

* Recognize token types using a **finite state machine** and **regular expressions**.
* Generate tokens for further processing.
* Handle illegal characters with informative error messages.

Creating rules to generate Tokens

### 3. Parsings

The parser employs:

* **Recursive Descent Parsing** to interpret tokens and build an Abstract Syntax Tree (AST).
* **Operator Precedence and Associativity** to handle complex expressions.
* **Node Classes** (e.g., NumberNode, BinOpNode) to represent constructs hierarchically.

### 4. Interpretation

The interpreter evaluates the AST:

* Executes code constructs like variable assignments, arithmetic operations, and control flow structures.
* Displays results or errors to the console.

### 5. Output

The program prints execution results or detailed error messages to the user.

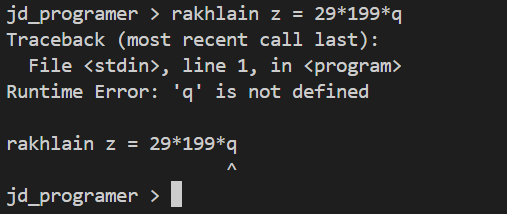
Input Output

## Features and Functionality

The Roman Urdu Compiler supports:

### Core Features

* **Variables and Arithmetic Operations**: Assign values to variables and perform basic arithmetic operations.
* **Control Flow**: Implement if-else conditions and loops.
* **Functions**: Define and call reusable code blocks.
* **Error Handling**: Provide user-friendly error messages for syntax and runtime issues.



Error handling

### Additional Enhancements

* **Class Support**: Built-in classes to facilitate object-oriented programming.

## Technical Implementation

### Codebase

The project comprises three main files:

1. basic.py: Implements the main compiler components, including the lexer, parser, AST nodes, and interpreter.
2. strings\_with\_arrows.py: Generates detailed error messages with visual indicators for the error's location.
3. **Presentation**: Explains the project's methodology, tools, and results visually.

### Key Highlights

#### Lexer

* Recognizes tokens like integers, identifiers, and operators.
* Uses error classes (IllegalCharError, ExpectedCharError) for precise feedback.

#### Parser

* Builds an AST using recursive descent parsing.
* Handles expressions, statements, and control structures.

#### Interpreter

* Processes AST nodes to execute the input code.
* Supports arithmetic, logical operations, and function calls.

#### Error Handling

* Includes syntax and runtime error detection.
* Utilizes string\_with\_arrows for error visualization.

## Results

### Code Output

* **Phase 1**: Variable assignments and arithmetic calculations.
* **Phase 2**: Control structures (if-else, loops) and functions.
* **Error Handling**: Effective detection and reporting of illegal operations and syntax errors.

### Demonstrations

* Variable manipulation and calculations.
* Conditional statements and loops.
* Function definitions and calls.
* Error scenarios with visual indicators.

## Conclusion

The Roman Urdu Compiler project successfully demonstrates the principles and processes of compiler construction. By supporting Roman Urdu as the input language, the project highlights the adaptability of modern compilers to non-standard programming languages.

The integration of robust error handling, an intuitive lexer and parser, and a well-designed interpreter ensures a comprehensive understanding of compiler architecture.

## Future Work

* Expand language features to include advanced constructs like data structures and libraries.
* Improve performance for large input programs.
* Develop a user-friendly graphical interface for enhanced usability.

## References

* Lecture slides (Ramzan Shahid).
* https://www.geeksforgeeks.org/compiler-construction-tools/.