A PROJECT REPORT
On
KukuLang (Custom Programming Language).
M.Tech 1st year
Guide Name: Dr. Suptashi Chakraborti

The ICFAI University, Tripura

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ICFAI Technical School

Faculty of Science and Technology

Course Title: Mini Project
2024-2026

Submitted by :-

Kuchuk Boram Debbrama. ID: 24IUT0600003

Source code :-https://github.com/kuchuk-borom-debbarma/KukuLang-v2

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my guide Sir Dr.Saptarshi Chakraborty guidance and support in completing of my project. We would thank God for being able to complete the project “KukuLang”, with success.

A special thanks to our Hon’ble VC Sir, Registrar, Dean Sir, Principal Sir for providing us with such opportunities and helping us gain knowledge.

Finally, we extend our sincere thanks to one and all of ICFAI family for the completion of the project.

Student Name

Kuchuk Boram Debbrama.

ABSTRACT

KukuLang is an innovative programming language designed to bridge the gap between natural human communication and computer programming. By implementing a syntax based on natural spoken English grammar, KukuLang makes programming accessible to non-programmers while maintaining the power and functionality expected in modern programming languages.

The language features static typing, custom data structures, function definitions (called "tasks"), and intuitive variable manipulation through natural language constructs. KukuLang's distinctive syntax allows for expressing complex programming concepts like conditionals, loops, and nested variable access using familiar English phrases, making code both readable and approachable.

The project includes a complete interpreter implementation written in C#, following a traditional compilation pipeline of lexical analysis, parsing, and interpretation. This approach balances accessibility with execution capabilities while laying groundwork for potential future developments like LLVM integration or transpilation to other languages.

KukuLang represents a significant step toward democratizing programming by reducing the cognitive barriers typically associated with learning traditional programming languages, while still providing the necessary constructs for expressing complex computational logic.

TABLE OF CONTENTS: Page No.

01. INTRODUCTION

01.1 Problem Definition:

Programming languages have traditionally required users to adopt specialized syntax and vocabulary that differs significantly from natural human communication. This learning curve creates a substantial barrier to entry for newcomers, especially those without a technical background. While existing programming languages prioritize machine efficiency and computational expressivity, they often sacrifice human readability and intuitive understanding.

The disconnect between natural language and programming syntax presents several challenges:

1. **Cognitive Barrier**: Potential programmers must learn abstract syntax rules that don't mirror their everyday communication patterns, leading to steep learning curves.
2. **Accessibility Gap**: Programming remains inaccessible to large segments of the population who could benefit from computational thinking but are deterred by conventional syntax.
3. **Readability Issues**: Code written in traditional languages often requires specialized knowledge to interpret, limiting collaboration between technical and non-technical stakeholders.
4. **Implementation Complexity**: Even simple logical constructs that are easy to express in natural language often require complex syntax patterns in conventional programming languages.

KukuLang addresses these challenges by creating a programming language with syntax that closely resembles natural English grammar, while maintaining the power and functionality of traditional programming languages. By bridging the gap between human communication and computational expression, KukuLang aims to democratize programming and make it accessible to a broader audience without sacrificing capabilities.

01.2 Project Overview:

KukuLang is a minimal yet powerful programming language designed to make coding accessible to non-programmers through natural language syntax while retaining the capability to express complex programming concepts. The project encompasses both the language design and its implementation.

Language Features

- **Natural English Syntax**: Commands follow intuitive English grammar patterns (e.g., "set x to 5" instead of "x = 5")
- **Static Typing**: Provides type safety and early error detection
- **Custom Data Structures**: Users can define and initialize their own complex data types
- **Task (Function) System**: Modular code organization with parameter passing and return values
- **Intuitive Variable Access**: Nested properties use possessive forms (e.g., "person's name" instead of "person.name")
- **Control Flow**: Conditional statements and loops with natural language expressions
- **Expression Evaluation**: Support for mathematical operations and complex expressions

Implementation Components

- **Lexical Analyzer**: Converts source code into meaningful tokens
- **Parser**: Transforms tokens into a structured parse tree representing program logic
- **Interpreter**: C#-based runtime that executes the parsed program statements

Project Goals

1. **Democratize Programming**: Make coding accessible to individuals without technical backgrounds
2. **Maintain Expressivity**: Ensure the language can handle real-world programming tasks
3. **Enhance Readability**: Create code that can be understood by those who didn't write it
4. **Educational Value**: Serve as a stepping stone to more complex programming languages
5. **Exploration of Language Design**: Provide insights into compiler/interpreter implementation techniques

KukuLang represents not only a practical tool for newcomers to programming but also an experimental platform for exploring the intersection of natural language processing and computer programming.

01.3 Objective:

Primary Objectives

1. **Create an Intuitive Programming Syntax**: Develop a programming language with grammar and structure that closely mirrors natural English, reducing the cognitive load required to learn programming.
2. **Build a Functioning Interpreter**: Implement a complete execution environment in C# that can parse and run KukuLang programs reliably.
3. **Support Core Programming Paradigms**: Ensure the language includes essential programming constructs including:
    1. Variable declaration and manipulation
    2. Control structures (conditionals and loops)
    3. Functions/tasks with parameters and return values
    4. Custom data structures

Secondary Objectives

1. **Educational Tool Development**: Create a language that serves as an effective teaching tool for introducing programming concepts without the syntax barriers of traditional languages.
2. **Maintain Performance Viability**: While prioritizing readability, ensure the language implementation is efficient enough for practical use cases.
3. **Document Compiler/Interpreter Implementation**: Provide clear documentation of the lexical analysis, parsing, and interpretation processes to serve as a learning resource for language implementation techniques.
4. **Enable Progressive Learning**: Design the language to facilitate a smooth transition to more traditional programming languages after users grasp fundamental concepts.

01.4 Hardware Specifications:

• Laptop of 16 GB RAM
• 256GB SSD of main storage
• Processor of AMD, Ryzen 5 3600XT

01.5 Software Specifications:

1. **VS Code**: Visual Studio Code, often referred to as VS Code, is a lightweight yet powerful source code editor developed by Microsoft. Renowned for its versatility, VS Code supports a myriad of programming languages and is widely embraced by developers across different domains. What sets VS Code apart is its intuitive user interface, coupled with a robust set of features like IntelliSense for smart code completion, Git integration for version control, and an extensive marketplace offering a plethora of extensions to tailor the editor to individual needs. With its speed, cross-platform compatibility, and a thriving community, Visual Studio Code has become a go-to choice for developers seeking a seamless and efficient

coding experience. Whether you're working on web development, data science, or cloud computing, VS Code stands out as a reliable and user-friendly tool for crafting high-quality code.

3. **GitHub:** GitHub, launched in 2008, is a web-based platform that facilitates collaborative software development using Git version control. It serves as a central hub for hosting and managing code repositories, promoting collaboration through features like pull requests and issues. GitHub is widely utilized by developers for open-source projects, offering tools for project management and automation through features like GitHub Actions. It has become a cornerstone for code sharing, collaboration, and project transparency in the software development community.

02. LITERATURE SURVEY

The concept of using natural language for programming has been explored in various forms over the decades, with each approach offering unique insights into bridging the gap between human communication and computational instruction.

English-like Programming Languages

COBOL (1959): One of the earliest attempts at creating readable code, COBOL (Common Business-Oriented Language) was designed with English-like syntax to make business applications more accessible to non-technical users. While verbose by modern standards, it pioneered the concept that programming languages could approximate natural language constructs.

HyperTalk (1987): The scripting language for Apple's HyperCard system used a syntax deliberately modeled after English grammar. Its approach of "programming by talking" influenced many subsequent efforts in accessible programming languages. Commands like `put "Hello" into message box` demonstrated how programming could mimic conversational instructions.

Inform 7 (2006): A design system for interactive fiction that uses a rule-based natural language approach. Inform 7 code reads like English prose, allowing authors to create complex interactive narratives through declarative statements. For example: `The Kitchen is a room. The Kitchen contains a table. On the table is a cookbook.`

Natural Language Processing in Programming

NLP-based Code Generation: Recent research has explored using large language models to generate code from natural language descriptions. Systems like OpenAI's Codex and GitHub Copilot represent the current frontier of natural language to code translation, though they typically generate code in traditional programming languages rather than executing natural language directly.

NLTK and ANTLR: Tools like the Natural Language Toolkit and ANTLR have been used to build parsers that can process controlled natural language subsets, demonstrating the technical feasibility of parsing English-like syntax into executable instructions.

Educational Programming Languages

Logo (1967): While not strictly using natural language syntax, Logo pioneered the concept of "low floor" programming languages that are accessible to beginners through simple, intuitive commands.

Scratch (2003): Uses visual blocks with natural language labels to create programs without traditional syntax concerns. Its approach of making programming concepts visible and tangible has proven highly effective in educational contexts.

Snap!: An extension of Scratch that adds more advanced programming concepts while maintaining the accessible block-based approach.

Interpreter and Compiler Design

KukuLang's implementation draws from established compiler design principles while adapting them to natural language processing challenges:

Recursive Descent Parsing: The primary parsing technique used in KukuLang is a well-established approach documented in works like "Compilers: Principles, Techniques, and Tools" (the Dragon Book). However, adapting this technique to natural language presents unique challenges in handling ambiguity.

Pratt Parsing for Expressions: For handling mathematical and logical expressions with proper operator precedence, KukuLang implements Pratt parsing (also known as Top-Down Operator Precedence parsing). This technique, introduced by Vaughan Pratt in 1973, efficiently handles expressions with different operator precedence levels while maintaining a simple implementation. The approach has been popularized in modern language implementations through resources like "Crafting Interpreters" by Robert Nystrom and is particularly well-suited for expression parsing in natural language contexts where operator precedence must be intuitive.

Lexical Analysis for Natural Language: Traditional lexing approaches must be modified to handle the nuances of English-like syntax, particularly regarding whitespace significance and multi-word tokens.

Interpreter Design Patterns: KukuLang's interpreter implementation follows the visitor pattern approach common in interpreter design, as described in works like "Crafting Interpreters" by Robert Nystrom.

Gap Analysis

While existing approaches have made strides in accessible programming, several gaps remain that KukuLang addresses:

1. **Balance of Power and Accessibility**: Many natural language programming approaches sacrifice computational expressivity for readability. KukuLang aims to maintain both through careful language design.
2. **Static Typing in Natural Syntax**: Few natural language programming systems have successfully implemented static typing within a natural language framework. KukuLang's approach to declaring types in an English-like manner represents an advancement in this area.
3. **Implementation Transparency**: Many educational languages hide their implementation details. KukuLang's open approach to documenting its lexer, parser, and interpreter serves an educational purpose beyond the language itself.
4. **Possessive Notation for Object Access**: KukuLang's use of the possessive form (e.g., `person's name`) for accessing object properties represents a novel approach to making object-oriented concepts more intuitive.

By addressing these gaps while building on established research in compiler design and natural language processing, KukuLang represents a meaningful contribution to the field of accessible programming languages.

03. LANGUAGE SYNTAX

KukuLang employs a natural language syntax designed to be intuitive for non-programmers while maintaining the power and expressiveness of traditional programming languages. This section details the core syntax elements and provides examples of their usage.

03.1 Basic Syntax Elements

Comments

Comments in KukuLang are enclosed in tilde symbols:

```
~ This is a comment ~
```

Output and Input

Displaying output uses the `print with` command:

```
print with "Hello World";
```

Taking input from the user:

```
set a to input;
```

Variables and Assignment

Variables are created and assigned values using the `set` keyword:

```
set var to 12;
```

03.2 Data Types and Structures

Primitive Types

KukuLang supports the following primitive types:

- `int`: Integer values
- `float`: Floating-point values
- `text`: String values
- `bool`: Boolean values

Custom Data Types

Custom data structures can be defined using the `define` keyword:

```
define Human with name(text), age(int);define Student with humanData(Human), class(int), rollNumber(int);
```

Object Initialization

Objects of custom types are initialized and modified as follows:

```
set sto to Student;
```

Nested Variable Access

KukuLang uses possessive forms to access nested properties:

```
~ Equivalent to sto.class = 69; ~set sto's class to 69;~ Equivalent to sto.humanData.name = "Jack Billdickson"; ~set sto's humanData's name to "Jack Billdickson";~ Equivalent to sto.humanData.age = sto.rollNumber; ~set sto's humanData's age to sto's rollNumber;
```

03.3 Tasks (Functions)

Task Definition

Functions in KukuLang are called "tasks" and are defined with the `define` keyword:

```
define HumanCreator returning Human with age(int), name(text) {      set tempHuman to Human; ~creating human object~      set tempHuman's age to age;     set tempHuman's name to name;  ~setting the name correctly~     return tempHuman;}~function with no return type and no params~define Foo returning nothing {      set a to (12+6)*2;}
```

Task Invocation

Tasks are called with the following syntax:

```
Foo;HumanCreator with age(24), name("kuku");
```

Tasks can also return values that can be assigned to variables:

```
set hum to HumanCreator with age(1), name("kuchuk");~Nested example~set hum to HumanCreator with age(1), name(sto's humanData's name);
```

03.4 Control Structures

Conditional Statements

```
if <condition> then {    ~do stuff~}else {}
```

KukuLang supports the following conditional operators:

- `is`: Equality
- `is_not`: Inequality
- `is_less_than`: Less than
- `is_less_or_is`: Less than or equal to
- `is_greater_than`: Greater than
- `is_greater_or_is`: Greater than or equal to

Loops

KukuLang offers two types of loops:

Until loop (continues until condition becomes true):

```
until <condition = false> repeat {}
```

As long as loop (continues as long as condition remains true):

```
as_long_as <condition = true> repeat {}
```

03.5 Other Language Features

The `return` keyword exits the current scope and, in the context of tasks, returns a value to the caller.

This natural language syntax design enables KukuLang to express complex programming constructs in a form that closely resembles human communication, making it more approachable for non-programmers while maintaining the necessary computational power.

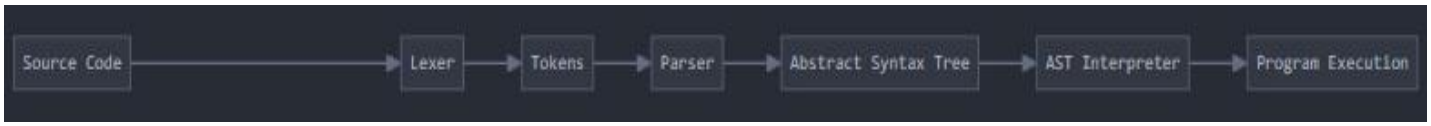
03. METHODOLOGY

System Architecture

The KukuLang implementation follows a classic compiler/interpreter pipeline architecture with three main components:

1. **Lexical Analyzer (Lexer)** - Tokenizes the source code
2. **Parser** - Builds an Abstract Syntax Tree (AST)
3. **Interpreter** - Executes the AST

This architecture provides clear separation of concerns and follows established language implementation patterns.



Component Breakdown

1. Lexer (KukuLexer)

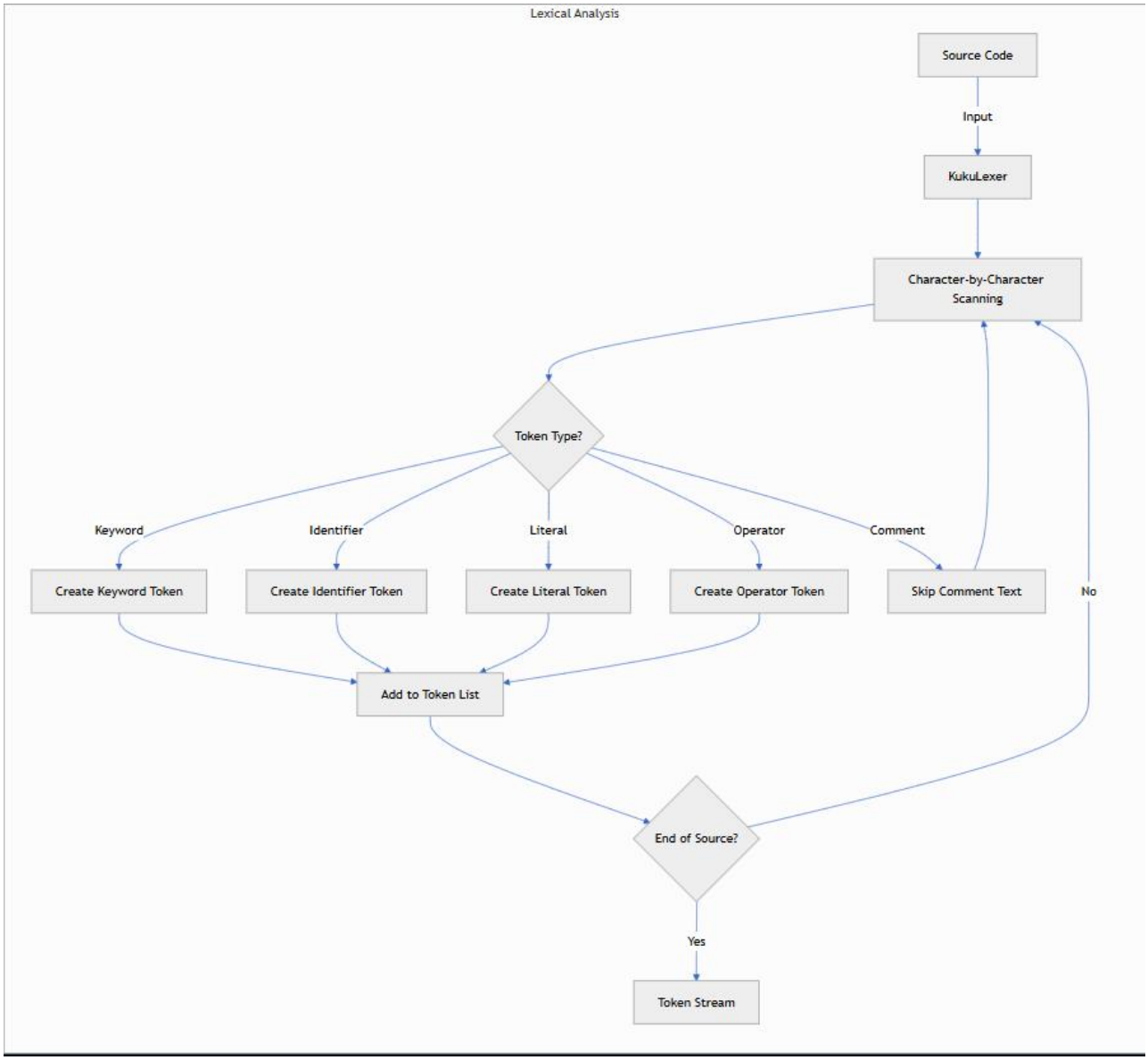
The lexical analyzer is the first phase of the processing pipeline. It transforms the raw source code into a sequence of tokens.

Key Features:

- Token generation based on language grammar
- Classification of identifiers, keywords, literals, and operators
- Line and position tracking for error reporting
- Comment handling (using ~ delimiters)
- String literal processing

Implementation Approach:

- Character-by-character scanning of source code
- State tracking for multi-character tokens
- Token type assignment based on pattern recognition
- Special handling for nested quotes and comments



2. Parser (RecursiveDescentParser and PrattParser)

The parsing phase transforms the token stream into an Abstract Syntax Tree (AST), which represents the hierarchical structure of the program.

Key Features:

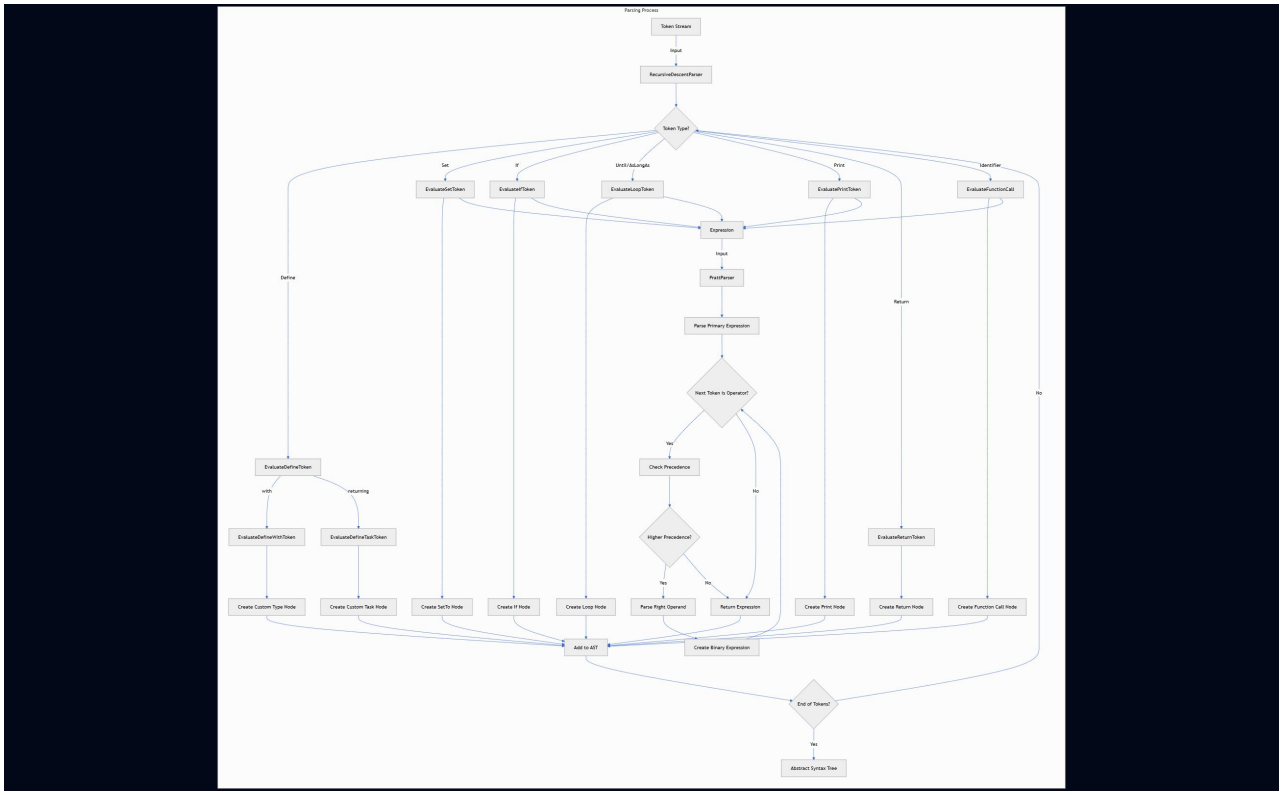
- Hybrid parsing approach combining recursive descent and Pratt parsing techniques
- Strong support for complex expression parsing with proper operator precedence
- Custom type and function declaration handling
- Statement-level structural representation

Implementation Approach:

- Recursive descent for statement-level parsing
- Pratt parsing for expressions with complex operator precedence
- Token validator services to ensure grammar compliance
- AST node creation for each language construct

Expression Parsing Logic: The PrattParser implementation handles expressions with proper precedence using these key concepts:

- Primary expressions (identifiers, literals)
- Infix operators (arithmetic, logical, comparison)
- Precedence levels for operators
- Recursive parsing for nested expressions



3. Interpreter (MainInterpreter)

The interpreter executes the AST directly without generating intermediate code.

Key Features:

- Runtime scope management
- Dynamic type handling (primitive and custom types)
- Variable resolution and assignment

- Function invocation with parameter passing
- Statement execution based on AST node types

Implementation Approach:

- Visitor pattern for traversing the AST
- Runtime object representation with dynamic typing
- Scope-based variable resolution with nested scoping support
- Statement processor service for modular interpretation

Runtime Architecture

Runtime Objects and Types

KukuLang supports both primitive types and custom user-defined types:

Primitive Types:

- `int` - Integer values
- `float` - Floating-point values
- `text` - String values
- `bool` - Boolean values

Complex Types:

- Custom user-defined structures with named properties
- Lists (planned feature)

Scope Management

The language implements a hierarchical scope system:

Features:

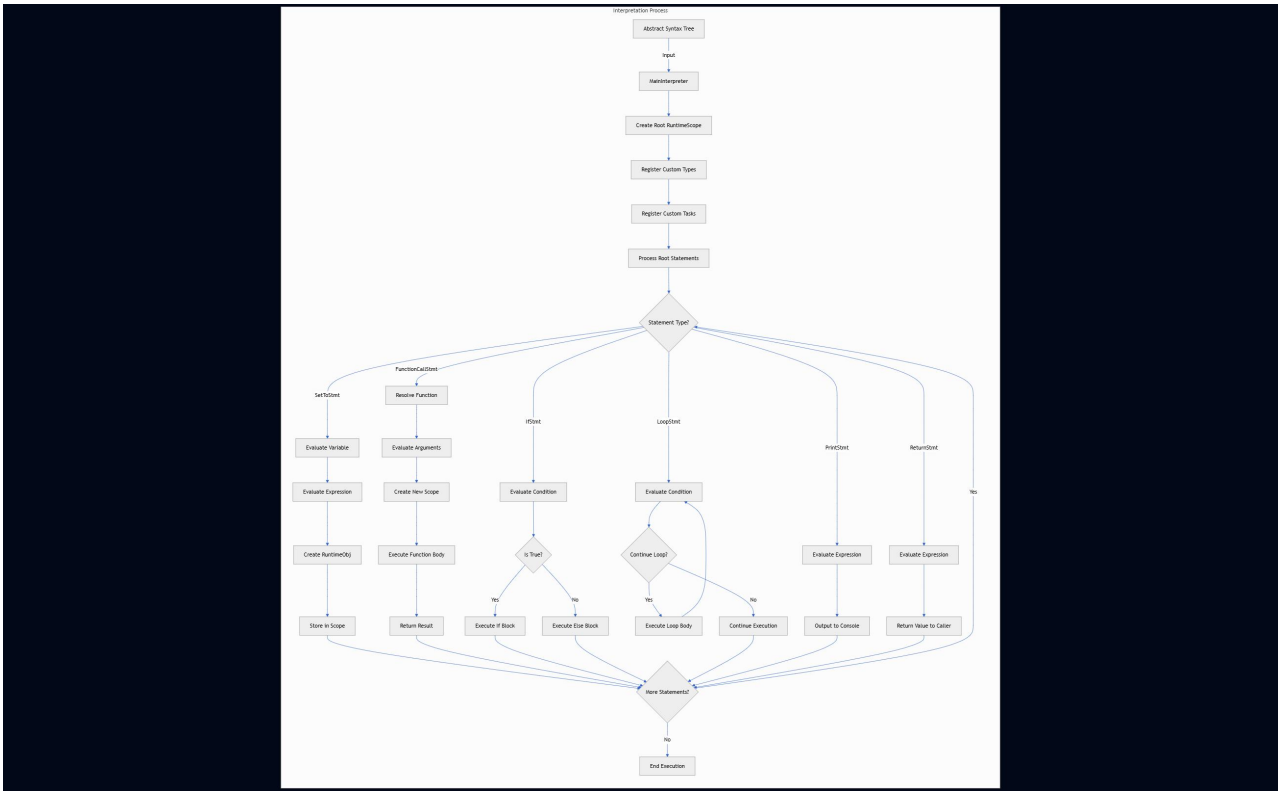
- Variable lookup with proper shadowing
- Parent-child scope relationships
- Isolated function scopes
- Dynamic scope creation and disposal

Memory Management

Memory is managed through the C# runtime with consideration for:

- Proper scope disposal using IDisposable pattern
- Garbage collection for runtime objects

- Reference tracking for complex data structures



Language Features

Natural Language Syntax

The core distinguishing feature of KukuLang is its English-like syntax:

- `set x to 5` instead of `x = 5`
- `if x is_greater_than 10 then {...}` instead of `if (x > 10) {...}`
- `until x is_greater_than 10 repeat {...}` instead of `while (x <= 10) {...}`

Static Typing with Natural Declaration

KukuLang uses static typing with intuitive type declarations:

```
define Human with name(text), age(int);set person to Human;set person's name to "John";
```

Function Definition and Invocation

Functions (called "tasks" in KukuLang) have a natural syntax:

```
define Greet returning nothing with name(text) {      print with "Hello " + name;}Greet with name("World");
```

Control Structures

Control flow statements follow natural language patterns:

```
if age is_less_than 18 then {     print with "You are underage";} else {       print with "You are an adult";}as_long_as counter is_less_than 10 repeat {     print with counter;     set counter to counter + 1;}
```

Testing and Debugging

The development approach includes several debugging features:

- AST visualization using Mermaid diagrams
- Verbose console output of tokens and AST structure
- Runtime tracing of variable assignment and task execution
- Error reporting with line and position information

03. CONCLUSION

KukuLang represents a significant advancement in the field of accessible programming languages by successfully bridging the gap between natural human communication and computer programming. Through its innovative approach of implementing English-like syntax while maintaining the power of traditional programming constructs, KukuLang demonstrates that programming languages can be both accessible to beginners and capable of expressing complex computational logic.

The project has successfully achieved its primary objectives by:

1. **Creating an intuitive programming syntax** that closely mirrors natural English, significantly reducing the cognitive barrier typically associated with learning to code.
2. **Building a functioning interpreter** in C# that can reliably parse and execute KukuLang programs through a well-structured lexical analysis, parsing, and interpretation pipeline.
3. **Supporting core programming paradigms** including variable manipulation, control structures, functions with parameters, and custom data structures, all expressed through natural language constructs.

Beyond these technical achievements, KukuLang demonstrates the viability of using controlled natural language as a programming medium, challenging traditional assumptions about the necessary disconnect between human communication and machine instruction. The language serves as both a practical tool for introducing newcomers to programming concepts and an exploration of compiler/interpreter implementation techniques.

The educational value of KukuLang extends beyond its use as a programming language. The transparent implementation of its lexer, parser, and interpreter provides valuable insights into language processing techniques, particularly the challenges of adapting established parsing methodologies to handle natural language constructs. The hybrid approach of combining recursive descent parsing with Pratt parsing for expressions offers a balanced solution to the challenges of parsing English-like syntax while maintaining proper operator precedence.

While KukuLang currently exists as an interpreted language, its architecture lays the groundwork for potential future enhancements, including:

1. Performance optimization through implementation of a bytecode compiler
2. Integration with LLVM for native code generation

3. Expansion of the standard library with additional functionality
4. Development of integrated development tools specifically designed for natural language programming
5. Further research into the educational effectiveness of natural language programming approaches

In conclusion, KukuLang successfully demonstrates that programming languages can be designed to be more intuitive and accessible without sacrificing their computational power. By reducing the artificial barriers created by traditional syntax, KukuLang opens programming to a broader audience, potentially democratizing computational thinking and programming skills across diverse populations. The project contributes not only a novel programming language but also valuable insights into the intersection of natural language processing and compiler design, pointing toward a future where programming becomes increasingly aligned with natural human communication patterns.