IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Bhaashankit: A Marathi Code Interpreter

Miss. Disha Ravindra Salunke[1], Prof. Prashant Shimpi[2]

[1]*Miss. Disha Ravindra Salunke, MTech Student :: 0009-0008-6578-1641 GF's Godavari College of Engineering, Jalgaon, Maharashtra, 425003*

[2]*Prof. Prashant Shimpi, Assistant Professor :: 0009-0004-9588-4298 GF's Godavari College of Engineering, Jalgaon, Maharashtra, 425003*

Abstract *- Programming languages are generally designed using English-based syntax, which can make learning difficult for individuals who are more comfortable with regional languages. This paper presents Bhaashankit, a Marathi-based programming interpreter that allows users to write and execute programs using familiar linguistic constructs.*

The main objective of this system is to make programming easier for beginners by reducing the dependency on English. Instead of focusing on memorizing syntax, users can directly understand programming logic through their native language.

Bhaashankit supports core programming features such as arithmetic operations, variables, conditionals, loops, functions, arrays, and basic data types. The interpreter follows a step-by-step execution process, where the input code is passed through stages like lexical analysis, syntax analysis, semantic checking, and execution.

Each stage has been designed carefully to ensure correct interpretation of the program and meaningful error reporting. The modular structure of the system also makes it easier to extend and improve in future.

The overall results suggest that using regional languages for programming can improve understanding and make learning more accessible, especially for beginners. This work highlights the potential of developing similar systems for other native languages.

Keywords - Marathi Programming, Interpreter Design, Regional Language Computing, Lexer, Parser, Abstract Syntax Tree, Programming Education

INTRODUCTION

Programming is an important skill in today's world, both in academics and industry. However, most programming languages rely on English keywords, which can be challenging for students who are not fluent in English. When beginners start learning programming, they often face two difficulties at the same time: understanding logical concepts and dealing with unfamiliar language syntax. In many cases, the difficulty is not due to logic, but because of the language barrier. This can reduce confidence and slow down the learning process.

To make programming more approachable, it is important to align it with the learner's natural language. With this idea in mind, Bhaashankit has been developed as a Marathi-based programming interpreter. It allows users to write programs using keywords such as "छापा" (print), "जर" (if), "अन्यथा" (else), and "जोपर्यंत" (while).

Rather than acting as a simple translator, the system works as a complete interpreter that processes and executes Marathi code step by step. This helps users focus more on problem-solving instead of syntax memorization.

By introducing programming in a familiar language, Bhaashankit aims to create a smoother and more intuitive learning experience.

LITERATURE REVIEW

The development of programming environments aimed at improving accessibility for beginners has gained attention in recent years. Traditional programming languages rely heavily on English-based syntax, which often creates a barrier for learners who are more comfortable with regional languages. As a result, several research efforts have focused on designing systems that use native language constructs to simplify learning and improve understanding.

One of the early contributions in this area is DEEP (Devanagari Enabled Programming), which supports Marathi syntax using the Devanagari script [2]. It allows users to write programs using familiar linguistic constructs, reducing the initial learning difficulty. The system provides a desktop-based interface for execution. However, its functionality is mostly limited to basic programming features and does not support advanced constructs such as complex data structures or real- time execution.

Another important work is the Marathi Programming Language (MPL), which follows a compiler-based approach using Marathi keywords [1]. MPL supports core constructs such as variables, loops, and functions, demonstrating that structured programming can be implemented using a regional language. However, due to its compiler-based nature, interactivity is limited, and some output still depends on English, which affects full localization.

Similar efforts have been made in other regional languages as well. For instance, Ezhil, a Tamil programming language, enables users to learn programming concepts in their native language [3]. It offers a simple syntax and is mainly focused on educational use. However, its feature set is limited and may not support more complex programming requirements.

The Hindawi Programming System also explores Indic text-based programming for Hindi users [4]. It aims to provide a natural interface for programming in Hindi. Like other systems, it faces challenges related to scalability, standardization, and feature completeness.

In addition to these systems, interpreter-based approaches are widely used in programming language design. These approaches typically involve stages such as lexical analysis, syntax parsing, semantic validation, and execution using Abstract Syntax Trees (AST). Such structured methods are preferred because they are flexible, modular, and suitable for educational tools.

Sr. No.	Project / Author	Overview	Method / Key Features
1	**DEEP – Devanagari Enabled Programming** (S. A. Halle et al., 2019)	Marathi-based programming environment using Devanagari script to improve accessibility for beginners.	Desktop-based IDE; supports basic variables, loops, and conditionals; limited advanced features; no real-time execution; partial localization; requires installation.
2	**MPL – Marathi Programming Language** (S. Berad et al., 2018)	Defines a programming language using Marathi syntax to introduce programming concepts in native language.	Compiler-based approach; supports variables, loops, and functions; limited interactivity; partial English output; no real-time execution; requires setup.
3	**Bhaashankit** (Proposed System)	A browser-based Marathi programming interpreter designed for real-time execution and improved learning experience.	Interpreter-based execution; fully Marathi input and output (text and digits); supports arithmetic operations, variables, user input, conditionals, loops, functions, arrays (1D & 2D) and data types; real-time execution; beginner-friendly; no installation required.

Table 1. Comparative Analysis of Existing Marathi Programming Systems and Proposed Bhaashankit Interpreter

Despite the progress made by these systems, several limitations still exist. Many of the existing solutions either provide partial implementation, lack real-time execution capabilities, or do not fully utilize native language output. Furthermore, some systems are not designed with scalability and extensibility in mind, which restricts their practical applicability.

The proposed system, *Bhaashankit*, addresses these gaps by providing a complete Marathi-based interpreter with a structured execution pipeline. Unlike earlier approaches, it supports real-time execution, full Marathi input and output, and advanced programming constructs such as nested loops, functions, and array operations. The system is also designed to be lightweight and modular, making it suitable for beginners as well as for further extension and development. Table 1 presents a comparative analysis of existing Marathi programming systems and the proposed approach.

METHOLOGY

The proposed system, Bhaashankit, is designed as a structured interpreter that processes Marathi programming instructions through a sequence of well-defined stages. The overall methodology follows a pipeline-based approach, where each stage performs a specific transformation on the input code before passing it to the next stage. This layered design ensures clarity, modularity, and efficient error handling, and each stage is carefully designed to ensure accurate interpretation of the program and effective error reporting.

Fig. 1 illustrates the complete flow of execution, starting from input acquisition to final output generation, along with error handling at each stage.

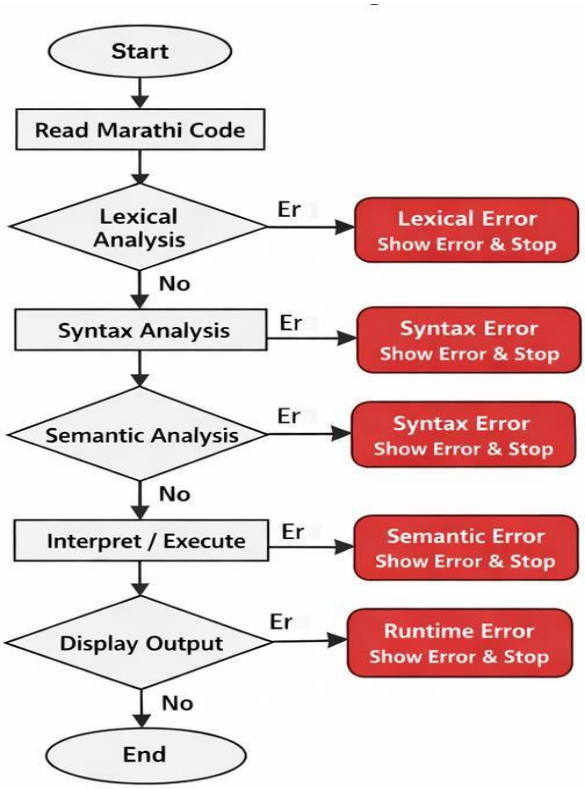

Fig. 1. Execution Flow of Bhaashankit Interpreter

Execution Pipeline Description

The execution of a Marathi program in Bhaashankit follows a systematic sequence:

1. Input Stage
   The user writes the program using Marathi keywords and syntax. This input serves as the raw source code for the interpreter.
2. Lexical Analysis
   In this stage, the source code is scanned character by character and divided into smaller units called tokens. These tokens include keywords (e.g., छापा, जर), identifiers, numeric values, operators, and symbols.
   If any invalid or unrecognized token is encountered, the system immediately reports a lexical error and halts execution.
3. Syntax Analysis
   The tokens generated in the previous stage are analyzed based on predefined grammar rules. The parser constructs an Abstract Syntax Tree (AST), which represents the hierarchical structure of the program.
   If the sequence of tokens does not conform to the grammar, a syntax error is generated, and execution stops.
4. Semantic Analysis
   This stage verifies the logical correctness of the program. It checks for issues such as:
   - Use of undefined variables
   - Type mismatches
   - Invalid operations

   If any inconsistency is detected, a semantic error is reported.
5. Interpretation / Execution
   Once the program passes all validation stages, the interpreter evaluates the AST. Each node of the tree is processed to perform the corresponding operation, such as arithmetic computation, conditional evaluation, or loop execution.
   Any errors occurring during execution, such as division by zero or invalid indexing, are treated as runtime errors.

6. Output Stage
   After successful execution, the final output is displayed to the user in Marathi format. This ensures consistency with the input language and improves user understanding.

Each stage is independent yet interconnected, which makes the system easier to debug, maintain, and extend.

Algorithm

The working of the Bhaashankit interpreter can be described using the following step-by-step algorithm:

Step 1: Start the system

Step 2: Accept Marathi source code from the user

Step 3: Perform Lexical Analysis

- Scan input and generate tokens
- Identify keywords, identifiers, literals, and operators
- If an invalid token is found → display lexical error and terminate

Step 4: Perform Syntax Analysis

- Parse tokens according to grammar rules
- Construct Abstract Syntax Tree (AST)
- If structure is incorrect → display syntax error and terminate

Step 5: Perform Semantic Analysis

- Validate variable declarations and usage
- Check data types and operations
- Ensure logical consistency of expressions
- If any mismatch is found → display semantic error and terminate

Step 6: Execute Program

- Traverse AST nodes
- Evaluate expressions and statements
- Execute loops, conditionals, and functions
- If runtime error occurs → display error and terminate

Step 7: Display Output

- Show final result to the user in Marathi

Step 8: End execution

Error Handling Strategy

An important aspect of the methodology is robust error handling. Errors are detected at different stages of execution, as shown in the flowchart:

- Lexical Errors: Invalid characters or tokens
- Syntax Errors: Incorrect program structure
- Semantic Errors: Logical inconsistencies
- Runtime Errors: Errors during execution

The system immediately stops execution when an error is detected and provides a clear message. This approach prevents incorrect outputs and helps users understand their mistakes.

DESIGN

The Bhaashankit interpreter is designed using a modular architecture in which each component is responsible for a specific stage of program processing. This separation of concerns improves maintainability, simplifies debugging, and allows the system to be easily extended in the future. Each module operates independently but works in coordination with other components to ensure smooth execution of Marathi programs.

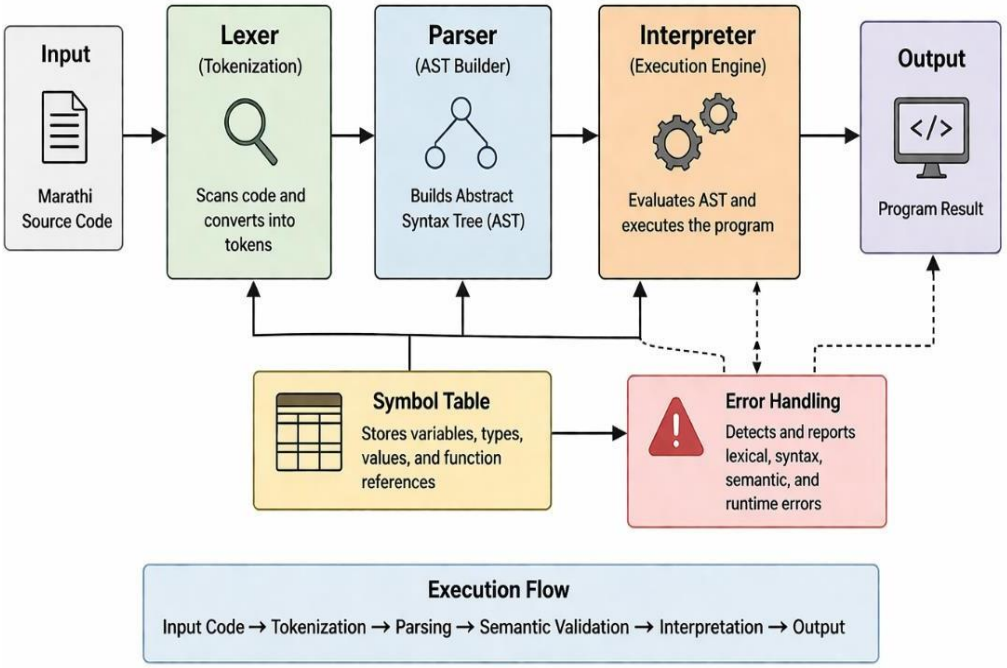

Fig. 2. System Architecture of Bhaashankit

Lexer Module

The lexer (lexical analyzer) is the first stage of the system and plays a crucial role in processing the input source code. It reads the Marathi program character by character and converts it into a sequence of tokens. Tokens are the smallest meaningful units of the program and serve as the input for the parser.

The lexer identifies different categories of tokens, including:

- **Keywords:** Reserved Marathi words such as छापा (print), जर (if), and जोपर्यंत (while)
- **Identifiers:** User-defined names for variables and functions
- **Literals:** Numeric values, strings, and boolean values
- **Operators:** Arithmetic and logical operators such as +, −, *, /
- **Symbols:** Special characters such as parentheses, brackets, and commas

The lexer also removes unnecessary elements such as whitespace and comments. If it encounters an invalid or unrecognized sequence, it generates a lexical error and stops further processing. By converting raw input into structured tokens, the lexer simplifies the task of the parser.

Parser Module

The parser is responsible for analyzing the sequence of tokens generated by the lexer and organizing them into a structured representation known as an Abstract Syntax Tree (AST). The AST reflects the grammatical structure of the program and defines how different elements are related.

In the AST:

- Arithmetic expressions are represented as hierarchical nodes
- Conditional statements form branching structures
- Loops create nested blocks
- Function calls are represented as subtrees

The parser follows predefined grammar rules to ensure that the program is syntactically correct. If the input does not follow these rules, the parser generates a syntax error and halts execution.

The use of an AST makes the program easier to interpret because it provides a clear and organized representation of the code.

Interpreter Module

The interpreter is the core component of the system, responsible for executing the program. It traverses the Abstract Syntax Tree and evaluates each node to perform the required operations.

The interpreter supports a wide range of functionalities, including:

- Evaluation of arithmetic expressions

- Assignment and updating of variable values
- Execution of conditional statements (जर, अन्यथा)
- Iterative execution using loops (जोपर्यंत, साठी)
- Definition and invocation of functions (फंक, परत)
- Handling of arrays, including multi-dimensional structures

The interpreter processes nodes in a step-by-step manner, ensuring that each operation is executed in the correct order. This approach allows dynamic execution and immediate feedback, which is particularly useful for beginners.

Symbol Table

The symbol table is a key data structure used to store information about identifiers such as variables and functions. It acts as a central repository that keeps track of:

- Variable names and their corresponding values
- Data types associated with each variable
- Function definitions and parameters
- Scope information for nested blocks and functions

During execution, the interpreter frequently accesses the symbol table to retrieve and update values. Proper management of the symbol table ensures correct program behavior and efficient execution.

Error Handling Mechanism

Error handling is an essential aspect of the Bhaashankit system. The interpreter is designed to detect and report errors at different stages of execution, ensuring that users receive clear and meaningful feedback.

The system handles the following types of errors:

- **Lexical Errors:** Occur when invalid characters or tokens are encountered during tokenization
- **Syntax Errors:** Arise when the program structure does not follow grammar rules
- **Semantic Errors:** Occur due to logical issues such as undefined variables or type mismatches
- **Runtime Errors:** Happen during execution, such as division by zero or invalid array access

Whenever an error is detected, the system immediately stops execution and displays an appropriate error message. This prevents incorrect results and helps users identify and correct mistakes effectively.

Design Advantages

The modular design of Bhaashankit offers several benefits:

- Improves code readability and organization
- Simplifies testing and debugging
- Allows independent development of components
- Supports easy addition of new features
- Enhances overall system scalability

This design approach ensures that the interpreter is not only functional but also maintainable and extensible for future enhancements.

CONCLUSION

This paper presented *Bhaashankit*, a Marathi-based programming language interpreter designed to make programming more accessible for users who are comfortable with regional languages. By allowing programs to be written and executed using Marathi syntax, the system reduces the dependency on English and enables beginners to focus more on understanding logic rather than memorizing unfamiliar keywords.

The interpreter follows a structured and modular execution pipeline consisting of lexical analysis, syntax analysis, semantic validation, and interpretation. This layered approach ensures that programs are processed systematically, with errors being detected and handled at appropriate stages. The use of an Abstract Syntax Tree further enhances the clarity and correctness of program execution.

The system supports a variety of programming constructs, including variables, arithmetic operations, conditional statements, loops, functions, and arrays. This makes Bhaashankit not only suitable for basic learning but also capable of handling moderately complex programs. The inclusion of a symbol table and structured error handling mechanism improves execution efficiency and user understanding.

From an educational perspective, the use of Marathi keywords significantly improves readability and comprehension for native speakers. Users are able to grasp programming concepts more quickly, which can increase confidence and interest in learning programming. The results indicate that regional language-based programming systems can play an important role in making computer science education more inclusive.

However, the current system is primarily designed for learning purposes and may require further optimization for large- scale or industrial applications. Future work can focus on enhancing performance, developing a graphical user interface, and extending support to additional regional languages. Integration with modern development tools and compiler-based optimizations can also be explored.

In conclusion, Bhaashankit demonstrates that programming in a regional language is both practical and effective. It provides a meaningful step toward bridging the gap between natural language understanding and computational thinking, thereby contributing to more inclusive and accessible programming education.

REFERENCES

[1] S. Berad, N. Kadam, S. Naik, K. Bhagwat, and S. Rathod, "MPL: Marathi Programming Language," International Journal of Computer Engineering in Research Trends, vol. 5, no. 9, 2018.

[2] S. A. Halle, M. S. Jagtap, R. S. Kolte, and N. S. Babhale, "DEEP – Devanagari Enabled Programming: A Way to Program in Marathi," MIT College of Engineering, Savitribai Phule Pune University, 2019.

[3] M. Annamalai, "Ezhil (எழில்): A Tamil Programming Language," International Journal of Scientific Research in Science, Engineering and Technology, vol. 5, no. 10, pp. 82–86, 2020.

[4] A. Chaudhary and S. Chaudhary, "Hindawi Programming System: Indic Text Programming Language for Hindi," 2018