BoinkLang

A Lightweight Interpreted Programming Language

Team: Quad Binary

Members: Aayushya Lakkadwala, Chinmay Neema

BL

TABLE OF CONTENTS

Introduction
Problem Statement
Implementation Overview
Features
Challenges Faced
Results & Future Work
Conclusion

INTRODUCTION

BoinkLang is a simple, interpreted programming language built in Go to explore language design and interpreters. It helps understand how code is processed internally.

What is BoinkLang?

BoinkLang is a lightweight, interpreted programming language built in Go to explore interpreter design. It provides a simple environment for learning tokenization, parsing, and evaluation. With core programming constructs like functions and control flow, it helps bridge theoretical compiler design and practical implementation.

Why was it developed?

BoinkLang was created to gain hands-on experience in building an interpreter and understanding language processing. It helps aspiring language designers grasp compiler fundamentals while deepening expertise in Go. The project also serves as a fun experiment in programming techniques.

PROBLEM STATEMENT

Why do most programming languages hide their internal processing, making code execution hard to understand? BoinkLang addresses this by offering a minimal, transparent implementation that simplifies language processing and enables easy experimentation.

Why do most programming languages hide their internal processing?	What problem does BoinkLang solve?
Programming languages abstract complex internal processes so developers can focus on writing code rather than execution details. While useful, this hides how interpreters handle tokenization, parsing, and execution. BoinkLang provides a minimal implementation, letting users see and modify language processing at a fundamental level.	Most interpreters prioritize efficiency, making them difficult to modify or understand. BoinkLang, being lightweight and modular, breaks language processing into clear steps. This allows easy experimentation with new features and execution behavior without added complexity.

IMPLEMENTATION OVERVIEW

How does BoinkLang process and execute code? Built in Go, it uses a structured approach with key components like a tokenizer, parser, evaluator, and symbol table. Its REPL mode offers multiple execution levels, allowing users to explore language processing step by step.

Why was Go chosen for BoinkLang?

Go was selected for its simplicity, strong standard library, and efficient performance. Its garbage collection handles memory management, allowing focus on interpreter logic. Static typing helps catch errors early, making development smoother.

How does BoinkLang process code?

BoinkLang follows a structured execution flow using key components. The Tokenizer (Lexer) converts code into tokens, the Parser builds an Abstract Syntax Tree (AST), and the Evaluator interprets the AST to execute commands. The Symbol Table manages variable bindings, ensuring proper scope handling.

FEATURES

BoinkLang offers an interactive REPL for real- time code execution, designed for simplicity and ease of use. It supports basic programming constructs like variables, control structures, and functions, ideal for both beginners and experienced users.

What are the core language features of BoinkLang?

BoinkLang supports essential constructs like variables, control flow (if- else), functions, and loops. These features allow users to store data, implement conditional logic, and perform repetitive tasks efficiently.

The language encourages code organization through functions, enabling modularity and reuse. With these core features, BoinkLang is simple yet powerful, catering to both beginners and advanced users.

How does BoinkLang handle errors during code execution?

BoinkLang has a custom error handling system that provides clear and detailed error messages. This helps users quickly identify and fix issues in their code.

The error messages are designed to guide users in debugging, making the process smoother and more intuitive. This system promotes learning and ensures a better programming experience for users of all levels.

CHALLENGES FACED

While designing and programming BoinkLang, we encountered challenges in creating an efficient interpreter that could handle diverse programming constructs while maintaining simplicity. Additionally, ensuring seamless error handling and providing a smooth, intuitive user experience required significant effort.

What challenges did we face in handling parsing complexity, such as operator precedence and associativity in BoinkLang?

We faced the challenge of defining clear rules for operator precedence and associativity to ensure correct evaluation of expressions. Balancing simplicity with accuracy, especially in complex expressions, was a key hurdle during the design process. Additionally, handling edge cases required extensive testing and refinement to avoid errors in parsing.

What challenges did we face in managing memory and variable bindings in BoinkLang?

Managing memory and variable bindings was challenging, especially when dealing with nested function calls and variable scopes. Ensuring efficient memory usage while maintaining accurate references to variables required careful planning and optimization. The complexity of managing dynamic memory allocations added another layer of difficulty to the process.

RESULTS & FUTURE WORK

We have successfully implemented core programming features in BoinkLang, providing a simple and efficient interpreter. Moving forward, we plan to enhance functionality, optimize performance, and introduce more advanced features for broader use.

What core interpreter functionality have we successfully implemented in BoinkLang?	What planned features are we working on for future BoinkLang updates?
We have successfully implemented core interpreter functionality, enabling the execution of basic programming constructs like variables, control flow, and functions. This solid foundation ensures a functional environment for users to experiment with and learn programming concepts. The interpreter is designed to be both simple and flexible, catering to a wide range of use cases.	We plan to add loop constructs (for, while) to enhance control flow capabilities. Additionally, we aim to implement a bytecode compiler for optimized execution, improve error reporting for more effective debugging, and introduce increment and decrement operators for streamlined arithmetic operations. These improvements will make BoinkLang more powerful and user-friendly for a broader audience.

CONCLUSION

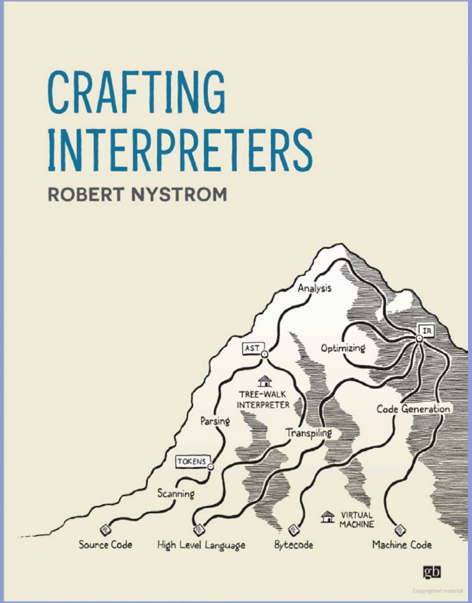
BoinkLang was an exploratory project that helped us learn and experiment with language design and interpreter implementation. While it serves as a foundation, we plan to continue expanding its features and refining its performance in the future.

What key lessons did we learn about interpreter design and language processing while building BoinkLang?	How did working on BoinkLang contribute to our understanding of Go programming and language architecture?
Building BoinkLang taught us essential concepts of interpreter design, such as tokenization, parsing, and expression evaluation. It deepened our understanding of how programming languages process code, manage control flow, and handle errors, reinforcing key principles of language execution. Additionally, we gained practical experience in optimizing language processing for efficiency and scalability.	Working on BoinkLang helped us gain practical experience with Go programming, particularly in areas like memory management and function handling. Additionally, we learned about language architecture, including how interpreters are structured, optimized, and executed, providing valuable insights into the inner workings of programming languages. This project also sharpened our skills in debugging, performance tuning, and implementing language features efficiently.

REFERENCES

Do Check Out These References

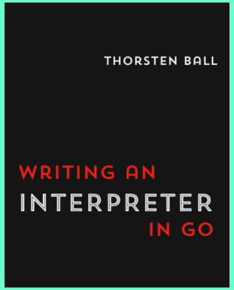
Robert Nystrom, " Crafting Interpreters"



Vaughan Pratt,
" Top-Down Operator Precedence Parsing." https://tdop.github.io

The Go Programming Language Documentation

Thorsten Bell,
" Writing An Interpreter In Go"



THANKS

Do you have any questions?

Keep Boinking