Graphic Era Hill University
DEHRADUN • BHIMTAL • HALDWANI

PROJECT AND TEAM INFORMATION

Project Title

Pebble:A Custom Language for Easy and Fun Programming.

Student/Team Information

Team Name:	***Alpha Coders***
Team member 1 (Team Leader) *Ritik Parihar* *22011554* *ritikparihar2040@gmail.com*	
Team member 2 *Priyanshi Karayat* *220121615* *priyanshikarayat72@gmail.com*	

Team member 3
Prisha Rana
22012993
prisharana4@gmail.com

PROJECT PROGRESS DESCRIPTION

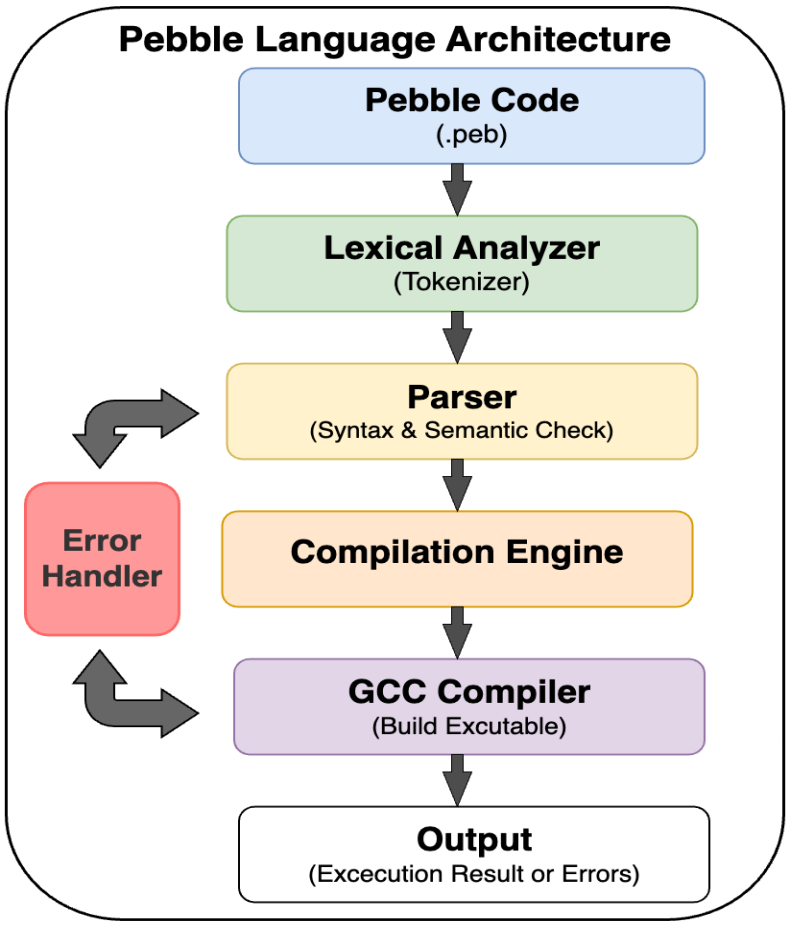
Project Abstract

The Pebble programming language is a beginner-friendly, statically-typed language designed for educational purposes, particularly to teach compiler construction. Its C-like syntax combines simplicity with expressive features, supporting integers, floats, strings, characters, and booleans, alongside arithmetic, comparison, and string concatenation operations. Pebble includes control structures like if-else, repeat, for, and while loops, as well as functions with automatic return type detection and built-in input/output operations. The compiler, built using Flex for lexical analysis and Bison for parsing, translates Pebble source files (.peb) into executable C code, incorporating robust type checking and a symbol table for variable and function management. Enhanced error handling provides detailed messages with line/column numbers and suggestions for common errors, improving the learning experience. The project includes a Makefile for building the compiler, a Bash script (run) for compiling and executing programs, and comprehensive documentation with examples showcasing all supported syntax. Pebble's design emphasizes readability with keyword-style operators (e.g., equal, greater) alongside symbolic ones, making it accessible for novices. The compiler generates C code with built-in functions for printing, input handling, and type conversion, linked with a C backend for execution. The project, licensed under the MIT License, aims to provide a practical platform for students to explore lexical analysis, parsing, semantic checking, and code generation, fostering a deeper understanding of compiler design principles.

Updated Project Approach and Architecture

The Pebble compiler employs a modular architecture to transform Pebble source code (.peb) into executable C code, designed for educational use in compiler construction. The system is built using **Flex** for lexical analysis and **Bison** for parsing, generating a lexer (lex.yy.c) and parser (parser.c, parser.h) respectively. The lexer tokenizes input into keywords (e.g., var, if), operators, literals, and identifiers, tracking line/column numbers for precise error reporting. The parser constructs an abstract syntax tree, enforcing static typing via a symbol table that manages variable/function types, constancy, and parameters. Semantic checks ensure type compatibility for operations and function calls.

The compiler generates C code, incorporating standard libraries (stdio.h, stdlib.h, math.h, string.h, ctype.h) and custom built-in functions (pebble_builtins.c) for input/output (print, display) and type conversion (to_int, to_float, to_bool, to_char). Error handling, implemented in error.c, categorizes issues (e.g., SyntaxError, TypeError) with ANSI-colored output, detailed messages, and suggestions, exiting on fatal errors. A **Makefile** automates compilation with gcc, linking the math library (-lm). The run Bash script streamlines execution by building the compiler, generating C code (output.c), compiling it into an executable (program), and running it.


Pebble Language Architecture
Pebble Code
(.peb)
Lexical Analyzer
(Tokenizer)
Parser
(Syntax & Semantic Check)
Error Handler
Compilation Engine
GCC Compiler
(Build Excutable)
Output
(Excecution Result or Errors)

Tasks Completed

Task Completed	**Team Member**
Scripting (run and Makefile script)	Ritik Parihar
Lexical Analyzer (pebble.l)	Priyanshi Karayat
Error Handling (error.c)	Priyanshi Karayat
Parser (half: variable declarations, assignments, functions, initial code generation).	Ritik Parihar
Parser (majority: control structures, semantic checks, function return types, final code generation)	Prisha Rana

Challenges/Roadblocks

During the development of the Pebble compiler, our team encountered several challenges. One significant issue was ensuring robust type checking in the parser (pebble.y). Prisha Rana and Ritik Parihar faced difficulties with semantic analysis, particularly when handling mismatched types in operations like string concatenation with numbers (e.g., "Hello" + 5). This caused unexpected errors during code generation. To address this, we enhanced the type checking logic in the parser, adding detailed error messages via error.c (developed by Priyanshi Karayat) to guide users on type mismatches, suggesting conversions like to_int() or to_string().

Another challenge was the lack of modulo operator support (%), as identified in pebble.l. Priyanshi noticed that users frequently attempted to use it, leading to frequent errors. Since Pebble doesn't support modulo, we improved the error message to suggest using division-based alternatives and plan to document this limitation clearly in the README.md to manage user expectations.

Additionally, Ritik faced issues with the run script on different operating systems due to variations in tool availability (e.g., bison, flex, gcc). Some environments lacked these tools, causing build failures. We resolved this by adding prerequisite checks in the script and providing installation instructions in the README.md. Moving forward, we plan to explore containerization (e.g., Docker) to ensure consistent build environments across platforms.

Finally, integrating the lexer and parser outputs occasionally led to mismatched token definitions between pebble.l and pebble.y. Priyanshi and Prisha collaborated to align token definitions in parser.h, ensuring seamless communication. We aim to implement automated tests in the tests/ directory to catch such issues early in future iterations, enhancing the compiler's reliability.

Tasks Pending

Task Pending	**Team Member (to complete the task)**
Expand documentation with advanced features and troubleshooting and Optimize generated C code for performance.	Ritik Parihar
Implement comprehensive testing.	Priyanshi Karayat
Add basic standard library support	Prisha Rana

Project Outcome/Deliverables

The Pebble compiler project delivers a fully functional compiler for the Pebble programming language, tailored for educational use in compiler construction. The primary deliverable is the pebblec binary, which translates .peb files into executable C code, supporting features like static typing, control structures (if-else, loops), functions, and input/output operations. The compiler includes a robust error handling system (error.c) that provides detailed, user-friendly error messages with suggestions, enhancing the learning experience. A comprehensive set of documentation, including README.md and Introduction to Pebble Language.docx, offers syntax guides, examples, and usage instructions. The run script simplifies compilation and execution, making the tool accessible to beginners. Example programs like allSyntax.peb demonstrate the language's capabilities, serving as learning resources. The project also delivers a modular architecture with a lexer (pebble.l), parser (pebble.y), and built-in functions (pebble_builtins.c), all integrated via a Makefile for easy building. Licensed under the MIT License, Pebble is an open-source tool that enables students to explore lexical analysis, parsing, semantic checking, and code generation, fostering a deeper understanding of compiler design principles.

Progress Overview

The Pebble compiler project is **approximately 80% complete**. Core components, including the lexer (pebble.l), parser (pebble.y), error handling (error.c), and scripting (run), are fully implemented, ahead of schedule. Ritik Parihar completed the run script and half of the parser earlier than anticipated, while Priyanshi Karayat's work on the lexer and error handling was finalized promptly, ensuring robust tokenization and user-friendly error messages. Prisha Rana's extensive parser contributions, covering control structures and semantic checks, were also completed ahead of schedule, enabling early integration and testing of basic functionality.

However, we are slightly behind schedule on comprehensive testing and documentation expansion. Priyanshi's testing task, intended to validate edge cases, is pending due to focus on error handling refinements. Ritik's documentation updates, including advanced features, are also delayed as he prioritized scripting. The addition of a standard library (Prisha) and code optimization (Ritik) are on track but not yet started, aligning with our timeline.

Codebase Information

- **Repository Link**: https://github.com/RitikParihar09/pebble-lang-compiler

The Pebble compiler project is hosted on GitHub under the repository name pebble-lang-compiler, which clearly reflects its purpose as a compiler for the Pebble language, aligning with the project's educational goals.

- **Branch**: Given the project's current phase, development is likely on the main branch. As core components like the lexer, parser, and error handling are complete, feature branches such as feature/testing or feature/stdlib could be created for pending tasks like testing and standard library additions.
- **Important Commits**:

  - **Initial Setup**: Committed the project structure, including Makefile, run script, and directories (src/, include/, build/, bin/), setting up the foundation for compilation and execution.
  - **Lexer Implementation**: Added pebble.l, enabling tokenization of Pebble code with error reporting for unsupported operators (e.g., %), and line/column tracking for precise diagnostics.
  - **Error Handling**: Introduced error.c, implementing categorized error reporting (SyntaxError, TypeError) with ANSI-colored output and user-friendly suggestions.
  - **Parser Development**: Committed pebble.y, defining grammar rules for variables, control structures, functions, semantic checks, and C code generation, forming the core of the compiler.
  - **Documentation**: Added README.md and Introduction to Pebble Language.docx, providing syntax guides and examples, with further updates planned.

The team should adopt a branching strategy (e.g., Git Flow) to manage remaining tasks like testing and documentation, ensuring smooth collaboration.

Testing and Validation Status

Test Type	Status (Pass/Fail)	Notes
Syntax Parsing	Pass	The parser (pebble.y) successfully processes allSyntax.peb, handling variable declarations, control structures (if-else, loops), and functions. No syntax errors were reported during informal runs.
Semantic Checking	Pass	Type checking in pebble.y correctly identifies valid operations (e.g., integer addition) and flags errors (e.g., string + integer), as seen in error handling outputs during development.
Error Reporting	Pass	The error handler (error.c) generates detailed messages for invalid syntax, type mismatches, and unsupported operators (e.g., %), with suggestions, as validated through manual error injection.
Code Generation	Pass	Generated C code (output.c) from allSyntax.peb compiles and executes via run script, producing expected output (e.g., "Hello, World!").
Comprehensive Unit Testing	Not Started	Formal unit tests for edge cases (e.g., nested structures, invalid inputs) are pending in the tests/ directory, scheduled for implementation.

Deliverables Progress

The Pebble compiler project's key deliverables are advancing toward completion. Below is the current status of each deliverable outlined earlier.

- **Pebble Compiler Binary (pebblec)**: **Completed**
  The pebblec binary is fully functional, translating .peb files into executable C code, supporting static typing, control structures, functions, and input/output operations, as demonstrated by running allSyntax.peb via the run script.
- **Error Handling System (error.c)**: **Completed**
  The error handling system, implemented in error.c, provides detailed, user-friendly error messages with categories (SyntaxError, TypeError), ANSI-colored output, and suggestions, enhancing the learning experience.
- **Documentation (README.md, Introduction to Pebble Language.docx)**: **In Progress**
  Initial documentation covers syntax guides and examples. Updates to include advanced features

(e.g., function return type detection, type conversion) and troubleshooting sections are in progress, with completion expected soon.

- **Run Script (run)**: **Completed**
  The run script is fully operational, automating the build, compilation, and execution of Pebble programs, making the compiler accessible to beginners.
- **Example Programs (allSyntax.peb)**: **Completed**
  The allSyntax.peb example is complete, showcasing all language features (e.g., variables, loops, functions, input/output) and serving as a learning resource.
- **Modular Architecture**: **Completed**
  The project's architecture, with a lexer (pebble.l), parser (pebble.y), and built-in functions (pebble_builtins.c), is fully implemented and integrated via the Makefile.

With most deliverables completed, the focus is on finalizing documentation and upcoming tasks like testing to ensure a robust educational tool.