The Zed Programming Language Evaluation Report

References to the "Initial Report" refer to the documentation delivered to the Client in early April.

Introduction

After 3 months in development, the Zed Programming Language is complete; included is a compiler and editor, forming a complete Interactive Development Environment. From initial requirements gathering from the Client's needs up until the release, the development process has involved programming in TypeScript and Python 3, the use of the Deno JavaScript runtime, creation of documentation in the form of HTML documents, a unit test for internal development to ensure stability in the compiler, and re-evaluation of the requirements to ensure the best product possible can be delivered to the Client.

Client Needs

Addressing the Client's needs, the product includes a compiler which can take Z source code and represent it in Python as "object code". The product also successfully performs lexical and syntax analysis on Z source code, and displays error messages to the user through the editor by means of red-highlighted parts of the source code which are causing errors. In addition to this, the compiler also performs undefined-type checking and displays messages to inform the developer that they referencing variables that are currently undefined at that point in the code.

All control structures required by the Client have been implemented, this includes: Programs, Instruction Sequences, Loops (Pre-Test, Post-Test, Count), Selections (Binary, Multiway). Communication with the client on the 20th of April resulted in the change of specification for the Post-Test loop, which now uses the "REPEAT DO ENDDO UNTIL ENDREPEAT" syntax. The binary selection was added to, to allow for "OTHERWISE IF", allowing for more conventional programming practices found in the software industry. Mathematical operators used in assignments have also been implemented: addition, subtraction, multiplication and division, with the extension of the client's needs including a modulus (%) operator. This new operator was added to enable further mathematical calculations and improve the versatility of the Language. All comparison operators have been implemented: less-than, greater-than, less-than-or-equal-to, greater-than-or-equal-to, is-equal. All logical operators have been implemented too: logical-AND, logical-OR and logical-NOT. Important to note is that the initial report states that the words "AND", "OR" and "NOT" may be used for their logical operators, but further consultation with the client revealed this was not necessary and it was removed from the Language Specification. Variables keep their structure of one letter followed by one or more digits, and are also now checked if they exist when referenced by the Action phase of the compiler. Consultation with the Client clarified the definition of letters and strings in the initial report; a string includes the ASCII characters: [SPACE], "!" and all characters from "#" to "~". Digits and numbers support integers, floating points, and negative forms of the previous two. Negative numbers were added to the specification to replace the cumbersome "= A1 0: = A1 10 -:" that was necessary to attain the value negative-ten. Conditions are unchanged from the initial requirements.

Instructions are implemented as is required in the initial-report; requiring a ":" at the end of each statement. Assignment statements include special rules for when an operator is used between two different data types, these rules can be found in "The Zed Programming Language" manual under "Assignment Statement". In addition to it however, input statements can accept multiple items to form the prompt, in the same way as the OUT [] statement.

One major addition to the original specification is the ability to have comments in Zed source code. They are created using a "#" and make the rest of the line following it a commentary by the developer.

Functional Requirements

All functional requirements from the initial report have been met, with the addition of undefined-type checking for variables referenced before they are assigned.

Non-Functional Requirements

User preferences were thought to be a feature of use, but not originally required by the Client. It was decided that they were unnecessary, and so the storage of them in JSON was not implemented. The GUI is still a dynamically generated web-page, accessible through a browser, and custom operations have been implemented by HTTP GET and POST, and WebSockets on the page and on the IDE internal server. TypeScript was used as the language of choice for development, Deno was use as the runtime for the product's source code, JavaScript is used in the webpages to provide extended functionality, and communication between the browser and IDE server is achieved by WebSockets. The development of the product involved Visual Studio Code, and the creation of a "test.ts" file serving as a unit-test file for internal development to ensure that the core product was stable. All images, stylesheets and webpage design were done from scratch. Standard library functions used include WebSockets, FileSystem, HTTP, and Sha1 hashing for identifying communications between the browser and the IDE server. To clarify the Client's concerns about the version of Python target when compiling and used for running object code, Python v3.8.3 is the chosen version.

Project Timeline

The Gantt chart representing the projected timeline of the project was an optimistic view, and it turns out that the development took longer than initially planned. This was due to the complexity of the compiler, and a major refactor that saw the robustness and stability of the compiler massively improved.

Algorithms

The load source code function was obsoleted by the use of the browser for file functions. The decision to do this simplified security and file-safety issues. Lexical Analysis was simplified greatly by the use of Regular Expressions, which are highly optimised, robust and efficient, enabling this step of the compiler to be near instantaneous. Syntax Analysis follows a similar flow to the algorithm, but employs a Class based approach to construct components of the Language in a modular fashion, greatly simplifying the model of the Syntax Analyser, and improving code maintainability. The compilation to Python follows the same approach as the initial report.

EBNF

The EBNF for Zed has changed to follow the updated Post-Test loop, the inclusion of the modulus operator, the exclusion of “AND”, “OR”, and “NOT”, the modification of strings and numbers, and the “OTHERWISE IF” component of binary selections.

Screen Design

The design of the editor has been modified slightly from the initial report. Changes includes the simplification of the Code Editor menu and the removal of the Settings Screen. A question the Client asked was, “Which screen shows the Compile Error Messages?”. To answer, compile error messages appear when hovering over red-highlighted text in the main code editor screen. The execution screen has also been given a title label at the top to show what program is currently running.

Conclusion

The delay of the product from the timeline given should not affect the client, as it was scheduled to be delivered early. The extension of the product should provide for some easy use, more versatility and a friendlier user experience. The refactor of the compiler which delayed the product should mean that the product is more stable than it would have been if delivered as initially planned. In concluding, The Zed Programming Language compiler, editor, manual and complete software product follow and extend upon the original requirements, and should satisfy the Client’s needs, providing a suitable Development Environment for the Zed Language and a stable platform for the Client to build their products from.