Naylang: A REPL interpreter and debugger for the Grace educational programming language

Director: José Luis Sierra Rodríguez

Keywords: [Keywords] .

Contents

List of Figures

1 Introduction

Naylang is an open source REPL interpreter, runtime and debugger for the Grace programming language written in modern C++; It currently implements a subset of Grace described later, but as both the language and the interpreter evolves the project strives for feature-completeness.

Motivation

Grace is a language aimed to help novice programmers get acquainted with the process of programming, and as such it provides safety and flexibility in it's design.

However, this flexibility comes at a cost, and most of the current implementations of Grace are in themselves opaque and obscure. Since Grace is open source, most of it's implementations are also open source, but this lack of clarity in the implementation makes them hard to extend and modify by third parties and newcomers, severely damaging the growth opportunities of the language.

Objectives and Methodology

Naylang strives to be an exercise in interpreter construction not only for the creators, but also for any possible contributor. Therefore, the project focuses on the following goals:

- To provide a solid implementation of a relevant subset of the Grace language.
- To be as approachable as possible by both end users, namely first-time programmers, and project collaborators.
- To be itself a teaching tool to learn about one possible implementation of a language as flexible as Grace.

To that end, the project follows a Test Driven Development approach [?], in which unit tests are written in parallel to or before the code, in very short iterations. This is the best approach for two reasons:

Firstly, it provides an easy way to verify which part of the code is working at all times, since tests strive for complete code coverage [?]. Therefore, newcomers to the project will know where exactly their changes affect the software as a whole, which will allow them to make changes with more confidence.

Secondly, the tests themselves provide documentation that is always up-to-date and synchronized with the code. This, coupled with descriptive test names, provide a myriad of **working code examples**. Needless to say that this would result in vital insight gained at a much quicker pace by a student wanting to learn about interpreters.

Tradeoffs

Since Naylang is designed as a learning exercise, clarity of code and good software engineering practices will take precedence over performance in almost every case. More precisely, if there is a simple and robust yet naïve implementation of a part of the system, that will be selected instead of the more efficient one.

However, good software engineering practices demand that the architecture of the software has to be modular and loosely coupled. This, in addition to the test coverage mentioned earlier, will make the system extensible enough for anyone interested to modify the project to add a more efficient implementation of any of it's parts.

In short, the project optimizes for **approachability** and **extensibility**, not for **execution time** or **memory usage**.

2 State of the art

Minigrace

Kernan

TreeGraph

JavaScript interpreters

JavaScript is similar to a fully dynamically typed version of Grace (our subset).

3 Implementation

Abstract Syntax Tree

Object and Execution Model

Built-in methods and Prelude

Heap and Garbage Collection

Debugger

Frontend

4 Appendix A: Title of appendix A

Subtitle 1

Subtitle 2

5 Appendix B: How was this document made?

Author

Note: the process described in this Appendix was devised by Álvaro Bermejo, who published it under the MIT license in 2017 [@persimmon].

Process

This document was written on Markdown, and converted to PDF using Pandoc.

Document is written on Pandoc's extended Markdown, and can be broken amongst different files. Images are inserted with regular Markdown syntax for images. A YAML file with metadata information is passed to pandoc, containing things such as Author, Title, font, etc... The use of this information depends on what output we are creating and the template/reference we are using.

Diagrams

Diagrams are were created with LaTeX packages such as tikz or pgfgantt, they can be inserted directly as PDF, but if we desire to output to formats other than LaTeX is more convenient to convert them to .png files with tools such as `pdftoppm`.

References

References are handled by pandoc-citeproc, we can write our bibliography in a myriad of different formats: bibTeX, bibLaTeX, JSON, YAML, etc..., then we reference in our markdown, and that reference works for multiple formats