Preposal

### Smaller

# Scripture

Mormon 9:34

"34 But the Lord knoweth the things which we have written, and also that none other people knoweth our language; and because that none other people knoweth our language, therefore he hath prepared means for the interpretation thereof."

# 2 Abstract

**The focus of this project is** to learn how to parse and execute a programing language using a lexer, parser and abstract syntax tree which will help deepen my understanding of how programing languages go from code to execution.

**The application focus is** to make a simple, easy to use programing language for people first learning to program.

# 3 Background

## 3.1 Definitions

*Abstract syntax tree (AST)*

An abstract syntax tree (AST) is a tree representation of the abstract syntactic structure of source code written in a programming language.

*Executer*

This is the part that turns the AST into action. Action meaning making the computer do what the AST said to do based on the rules of the language.

*Tokenizer*

Breaks a string of characters into "tokens" or the pieces of the programing language (if block, a function, etc...)

*Parser*

The part of the program that turns the tokens into an AST.

*Spec*

Stands for "requirements specification" which will be a document describing what my programing language is required to preform.

## 3.2 Why this topic is of interest

Programing is a very useful skill but the barrier to entry is quite high for beginners. For this reason I propose to create a simple programing language with a visual representation of the internals. Providing a more visual way to see what is going on at each stage of the execution helps beginners find bugs as well as understand what the program is actually doing. This is interesting because many people, if taught to code, could solve many useful problems that otherwise would not of been possible for them.

## 3.3 Prior work by others

* [http://sonic-pi.net](http://sonic-pi.net/) a music creation programing enviorment.
* [DrRacket](http://docs.racket-lang.org/drracket/index.html) programing with pictures.
* [GameMaker](https://www.yoyogames.com/studio) a programing envorment that I learned to program in.
* Python's editor for windows
* [Scratch](http://scratch.mit.edu/) a lego based programing language.
* Codeacademy / Codeschool (both online tutorials that take you step-by-step through programing)
* [Daisy the Dinosaur](https://itunes.apple.com/us/app/daisy-the-dinosaur/id490514278) an app for young children to learn to program on an I-pad.
* [Khan Academy Computer science courses](https://www.khanacademy.org/cs).
* [CODE.ORG](http://code.org/) has many basic interactive programing tutorials aimed at children.
* etc... etc...

## 3.4 Prior work by you

* Brainstorming for the language syntax and other language ideas.
* Research in how various languages other languages work
* Research into the high level concepts of a compiler

# 4 Description

The project consists of making a simple, easy to use programing language focused at people who are first learning to program.

1. Preliminary research and proposal preparation
2. Research
3. Requirements specification
4. Design
5. Writing behavior driven tests.
6. Writing the Code to preform the given behaviors previously specified.

## 4.1 What is success?

The following is considered a "success":

* The code from the spec parses
* The code from the spec runs
* The code from the spec has the required output for any of the input examples from the spec

# 5 Scope

## 5.1 What is included

* A simple executable programing language which will have the following features
* Variables which can hold values and be replaced with other values
* Basic math operators including addition, subtraction, division and multiplication.
* Lists
* A basic way to repeat blocks of code a number of times
* Functions
  + Defining them
  + Running them
  + Returning values from them
  + Returning values from them
* A simple view of the output in a separate screen
* A visual way to view the internals variables

## 5.2 What is not

* Many language features including but not limited to:
* Classes
* Local scope, everything will be global.
* Normal integers (everything will be doubles)
* Pointers

# 6 Tasks and Schedule

1. Preliminary research and proposal preparation
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**
2. Research
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**
3. Requirements Specification document
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**
4. Design
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**
5. Writing behavior driven tests.
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**
6. Writing the Code to preform the given behaviors previously specified.
   * **Estimated Hours till completion:**
   * **Start Date:**
   * **Stop Date:**

**Total Hours:** 156

# 7 Deliverables

* [ ] An executable that can interpret and execute the code written in a plain text file according to the spec.

# 8 Applicability

This project helps me deepen my understanding of how the code that I write is actually doing something, what process it has to go through to reach execution. This would expand on many of the topics taught in CS 124 where we learned about the low level constructs of programing. Also there is a lot of knowledge I will be able to utilize from CS 238 when we talked about languages and grammar which have to be defined in any programing language. It is outside of the curriculum because there is no class that focuses on constructing programing languages.

# 9 Required Resources with Costs

There is no requirements with costs.

# 10 References

* <http://en.wikipedia.org/wiki/Abstract_syntax_tree>

Also all the things listed in "prior work by others"