Alex Molotkov
Ashley Vieira
Connor Wilson
Zeyad Shureih

# Design Document

## Introduction

The name of our language is Crypt0. We chose this name because we aim to include encryption in our language. Crypt0 is an imperative language that very closely resembles C. The language is imperative because it includes top-level definitions to define variables/procedures, statements which make up the function definitions, and expressions to evaluate values. Although not currently implemented, our goal is to have password-locked executables.

## Design

We currently have all basic data types and operations consisting of Doubles, Ints, Strings, and Bools. These features fall under the core feature level as they're necessary for storing various types of values. With these data types we have implemented the necessary operations of declaring variables, adding, subtraction, multiplication, division, and if statements. Conditionals are represented with if-then-else statements, and they are a core feature. We plan to use explicit looping constructs. Both while and for loops are syntactic sugar, with while loops being based on if-else statements. Currently, variables can be named and assigned to values under the command data type with a declare call, but they do not include scope. Variables are a core feature. We will be using functions with pass by reference for reusing repeated code and altering the state. Functions are syntactic sugar, as they are not necessary to run a program in the language, unlike in C where a main function is a requirement. The additional features we will be including are string operations (such as concatenation) and a static-type system.

The static-type system is the safety property of our language. The program will check for type errors before it is executed. This is to ensure that there are no type errors caused by applying an operation, function, or procedure to an argument of the wrong type.

Implementation

For our semantic domain, we use a state to state domain implemented with Maps. The Map takes a Name and a Var, storing them as a key-value relationship. We decided on this because storing Vars in a list to represent state created slower programs and was harder to work with. Maps are much easier to work with if we want to access the data in a variable quickly.

On “compilation” of the program, the executable program will be encrypted in a manner much similar to a one time pad, where a key defines how offset each character in the source code will be. In order to avoid a simple decryption as a result of similar beginnings to program, and the fact that Haskell does not play well with pseudo random number generators, we are applying a trigonometric function based off of position in the program’s source code to further encrypt the program. When executing the program, the key must be passed along with the executable otherwise the source code will not be decrypted.