

# Design Document

## Introduction

The name of my language is called Popper. This is a play on its paradigm, which is StackBased. I do not believe that there is anything especially unique that needs to be addressed.

## Design

1. Basic Data Types and Operations
  - a. We have integer and boolean data types, as well as char data types. Additionally, the operations they provide will be basic math commands such as “multiply”, “divide”, “add”, “subtract”, “tothe”(exponential), and “mod”. The result will be an integer raised to the number. This would fall under the core level as they are integral to the various functions and operations done using the language.
2. Conditionals
  - a. For the conditionals, there will be an if-then-else function that will be able to take in user conditions to branch out depending on if the conditions are satisfied or not. This feature would fall under the core level as without this, it would be practically impossible to branch outwards and that would affect the expressiveness of the language.
3. Recursion/Loops
  - a. We chose to implement a while loop for our program. This is shown with our command “run ex23”. Because we do not have variables, however, we were not able to figure out how to program an example that doesn’t result in an infinite loop. We do believe that the While command itself is written correctly.
4. Procedures/Functions with Arguments
  - a. Because we are using stack based, we can simply do arguments passed normally because they are on the stack itself. This is a core function, and the only way to push arguments onto the stack. Procedures can be created and executed as well.
5. Stack Manipulation Operations
  - a. Some useful stack operators we will use (other than push/pop) is SWAP to swap stack items, DUP to duplicate, and REV to reverse our items on the stack. These are library-level features, as they won’t be used by the user or programmer and will really be used behind the scenes.
6. Static-Type System (2)
  - a. This prevents unauthorized stack modification. This is a library level function. It is under the hood and won’t be noticed unless an error occurs. The stack check is handled by not allowing popping manually, therefore, we will always have a net positive to avoid buffer underflow.
7. Strings and Operators (1)

- a. We would allow strings to be manipulated with ++ for concatenation This is a core function, and couldn't exist without us adding it.

## Implementation

The semantic domain that we decided to use for our language is stack to maybe stack, where the user would be able to manipulate variables contained within the stack, such as ints, strings or bools. This semantic domain is integral to our language as it utilizes a stack-based paradigm, so all of our expressions and commands have the domain of the stack.