CS3500 - Software Engineering

# Deliverable 1: Design Documentation

Version 1

## Calculator System Comprising of

1.Tokenizer
2. Infix2postfix Converter
3. Code Generator
4. Interpreter/ Virtual Machine

Colin Kelleher - 117303363
Liam de la Cour - 117317853
Karol Przestrzelski - 117360873
Jonathan Hanley - 1174743096

# Table of Contents

# An Overview of the Whole System

## 1.Tokenizer

The purpose of the tokenizer is to convert the program's input into separate “tokens”. For example, 42 is a token, + is a token. The tokenizer has to read the input character by character, checking whether it is a digit, and continues reading the input until a non-digit character is identified. Every operator within the calculator can be expressed by a single character. The tokenizer then exports the token into a format <type> <value>. This will be outputted to a file.

**Types:**

- ❏ I = Integer
- ❏ F = Float
- ❏ R = Right bracket
- ❏ L = Left bracket
- ❏ O = Operator

| **Sample Input** | **Sample Output** |
|---|---|
| 14 + 52 | I 14<br>O +<br>I 52 |
| 12.5* (12 + 5) | F 12.5<br>O *<br>L (<br>I 12<br>O +<br>I 5<br>R ) |

## 2. Infix2postfix Converter

The infix2postfix converter will take the output from the tokenizer in the format specified above, and will change the order, see the example below. This will then be outputted to a file.

The Infix2postfix Converter will use the “shunting-yard” algorithm

❏ The output will be separated by spaces

| **Sample Input** | **Sample Output** |
|---|---|
| `I 1`<br>`O +`<br>`I 2`<br>`O *`<br>`I 3` | `1 2 3 * +` |

## 3. Code Generator

The Code Generator reads the output of the Infix2postfix converter and generates instructions for the Virtual Machine (VM). We have defined an instruction set below-containing instructions to complete the actual operations such as addition, multiplication, division etc. These instructions will be outputted to a file.

**Instruction Set:**

- ❏ LOADINT
- ❏ LOADFLOAT
- ❏ ADD
- ❏ SUB
- ❏ DIV
- ❏ MUL

| **Sample Input** | **Sample Output** |
|---|---|
| `1 2 3 * +` | `LOADINT 1`<br>`LOADINT 2`<br>`LOADINT 3`<br>`MUL`<br>`ADD` |

## 4. Virtual Machine

The virtual machine will read in the output file from the code generator. It will execute the provided instructions to complete the calculation. The output of this calculation will then be printed to the screen using the *printf( ) function*.

## 5. Overall View

The output of the VM will be the final result of the calculation and will be printed to "*stdout*" using the *printf ( )* function in the C language. The instruction set in the code generator is readable by the VM. The Tokenizer, Infix2postfix converter, code generator, and virtual machine all read the input from an input file ( '.txt' files)

# Set of Detailed Requirements

The Calculator should fulfil the following requirements

## 1. Tokenizer Requirements :

The Tokenizer should read a line from the input text file called "input.txt". The input file should contain a single line of integers, floats, brackets and operators. It will create tokens from these. Each token will be placed on a separate line, as key-value pairs.

## 2. Infix2postfix Requirements

The Infix2Postfix converter reads its input from a text file called "tokenized.txt". The tokens in the input file are type, value pairs, with each pair separated by a newline.

It will output the calculation in postfix notation, as a string on a single line, with each value separated by a single space, to a text file called "postfixed.txt"

## 3. Code Generator Requirements

The Code Generator reads its input from a text file called "postfixed.txt", which has been written out to by Infix2postfix.

It will convert this postfixed data into the instruction set used by the virtual machine.

It will write out the output to a text file called "codegenerated.txt".

## 4. Virtual Machine Requirements

This reads its input from a text file called “codegenerated.txt”

The Virtual Machine should execute the code generated and output the result to *stdout*.

# Interface's Description

## Tokenizer -> Infix2postfix

One token per line, in key, value pairs. The key tells the infix2postfix converter what data type the value is.

For example,

- I 42 (Integer, with value 42)
- F 4.2 (Float, with value 4.2)

## Infix2postfix -> Code Converter

Input is a single line, each input separated by a single whitespace, in postfix notation.

E.g `1 2 3 * /`

## Code Converter -> Virtual Machine

Input is a text file, where each line consists of a single instruction (as defined above).

E.g. LOADINT 42

## Test Suite

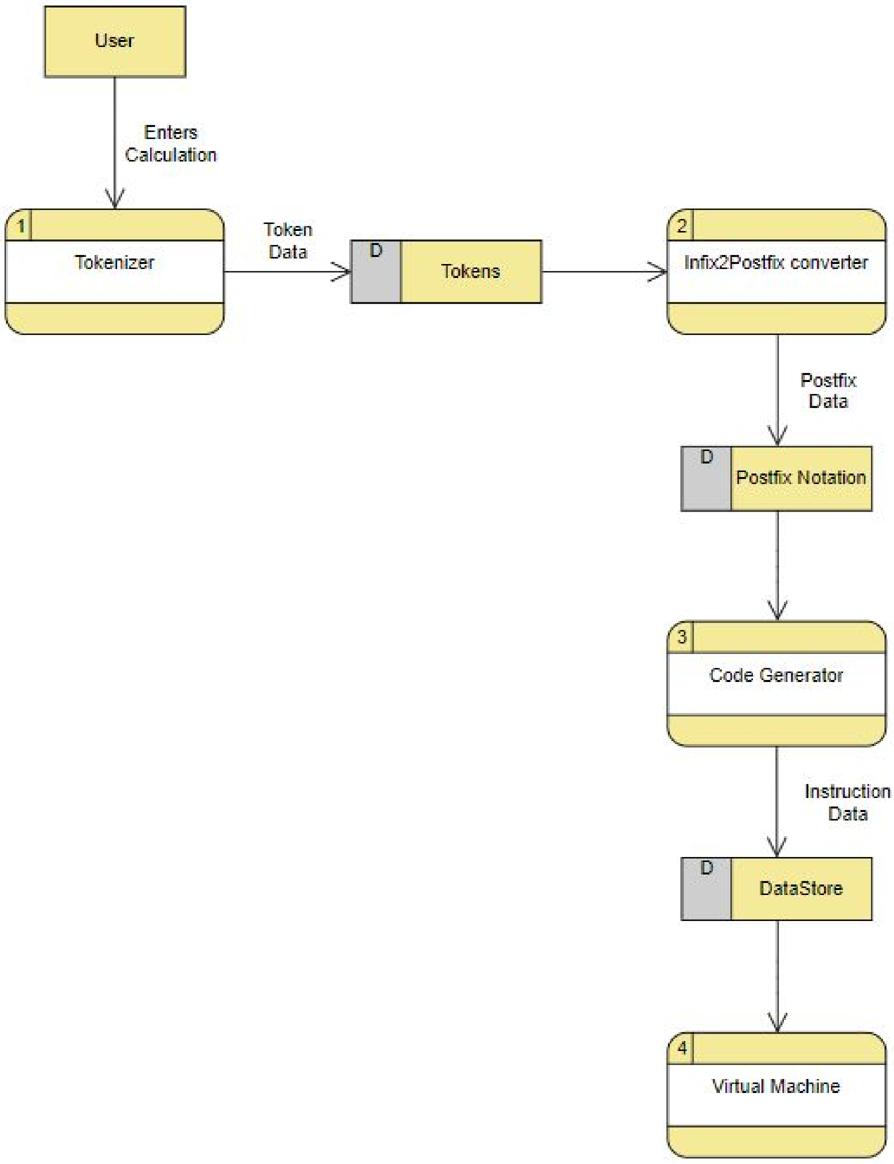
A test suite shall be produced to ensure our program's reliability and accuracy.

I.e. we will provide tests to make sure each component of our software will output in the correct format.

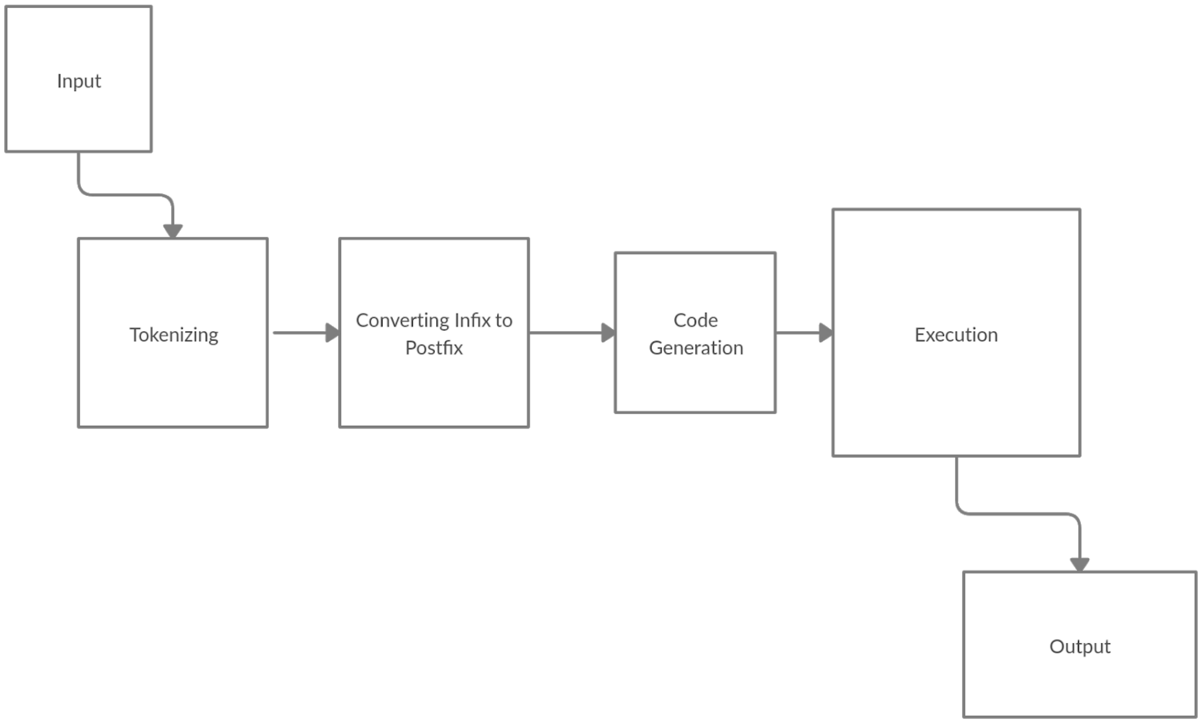
The Test Suite will be available on GitHub along with our code, which will be used throughout the coding and design process to ensure that our code operates as it is supposed to.

The test suites will use unit tests.

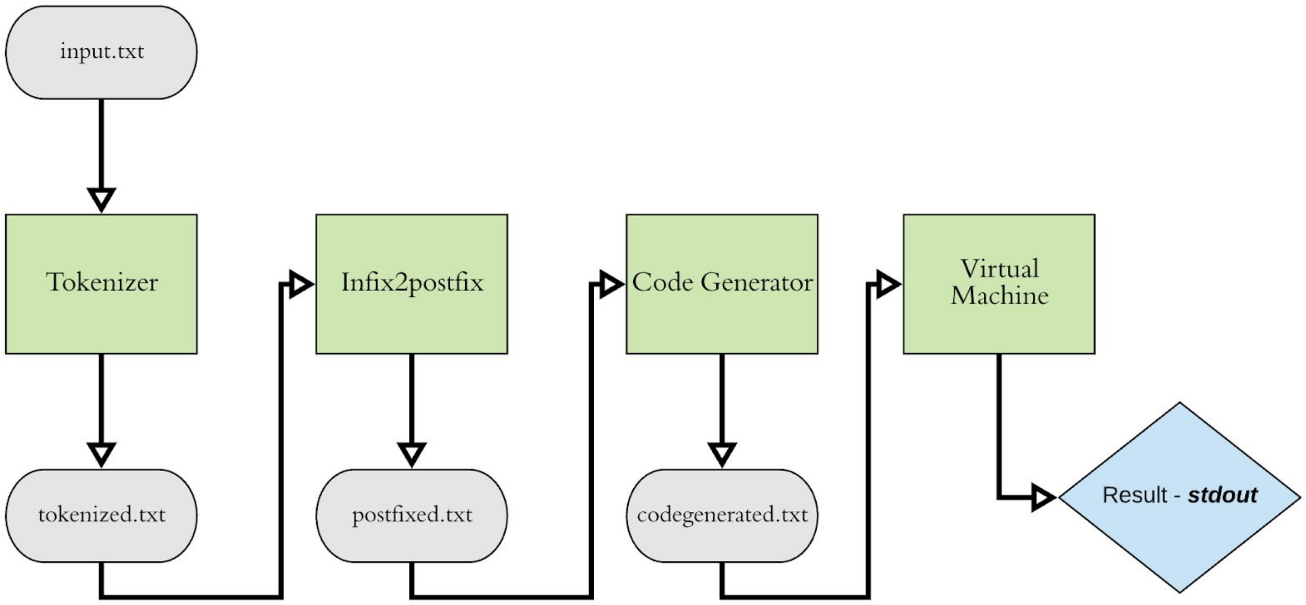
# Design Diagrams



Data-Flow Diagram



High-Level Architecture Diagram


input.txt
Tokenizer
Infix2postfix
Code Generator
Virtual Machine
tokenized.txt
postfixed.txt
codegenerated.txt
Result - ***stdout***

# Additional Notes

## Team Members

| | |
|---|---|
| Colin Kelleher | 117303363 |
| Liam de la Cour | 117317853 |
| Karol Przestrzelski | 117360873 |
| Jonathan Hanley | 1174743096 |

Group 6 on Canvas

## Programming Language

C Programming Language

## GitHub

https://github.com/jonathanhanley/SoftwareEngineeringProject

## Coding Requirements

All code will be commented to ensure everyone within the team can understand what is happening, and ensure maintainability