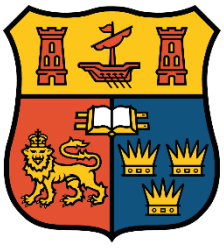


UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

CS3500 Calculator Project

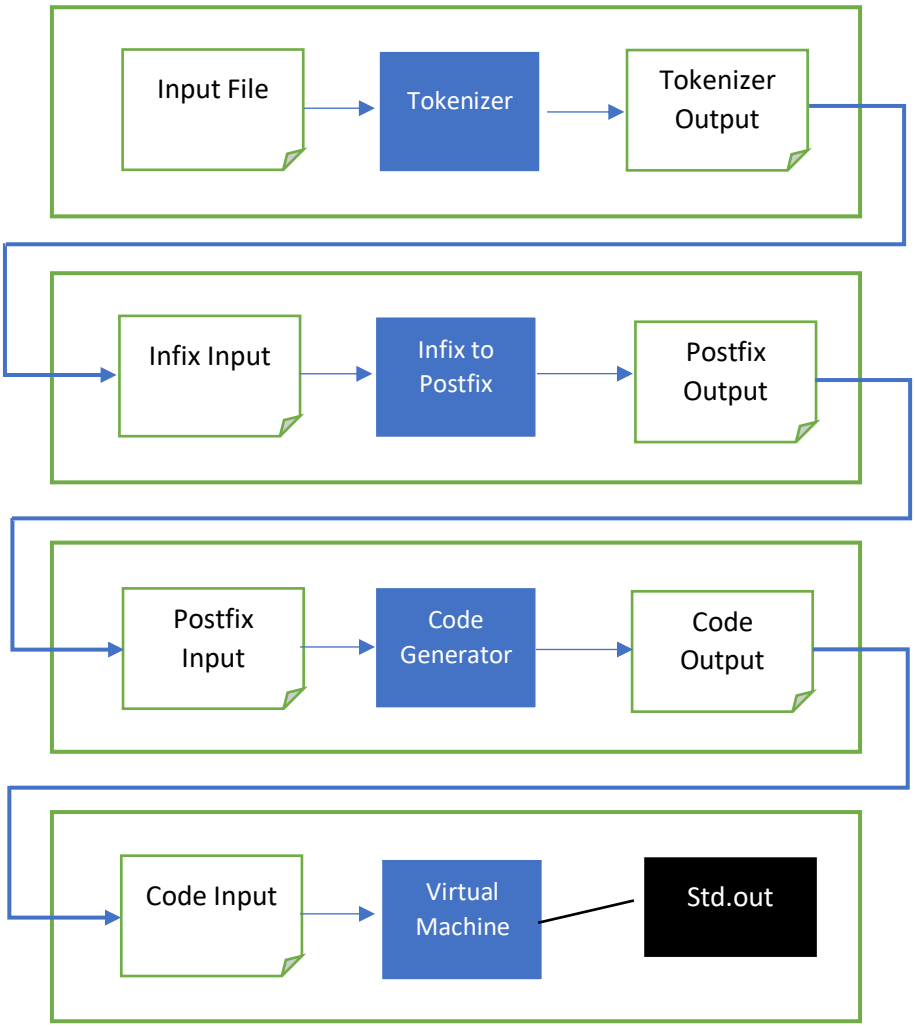
Design

Authors:
Cian Strolla, Tony Flavin, Ronan Murphy

System Overview

1. The Tokenizer will receive the raw input via an input text file. It will separate it into individual tokens, which it will then write to an output file.
2. The output from the Tokenizer will then be used as the input to the Infix to Postfix converter. It will read the formatted data from the file and translate it from infix notation to postfix notation. This postfix notation will be written to an output file.
3. The Code Generator will take the output from the Infix to Postfix converter and determine the instructions needed in order to carry out the calculation. These instructions will be written to an output file.
4. Finally, the Virtual Machine will take the instructions generated by the Code Generator and execute them and determining the result of the calculation. This result will be returned to the user via standard out.
5. A test suite will be developed to test the functionality of the calculator, involving unit tests, integration tests, and system tests.

System Overview Diagram



Requirements

Functional Requirements

1. The calculator must take an input expression from a file and output the result of the expression for the user to see.
2. Error detection must notify the user of mistakes in the input that results in the inability to compute an answer via an error message.
3. Each component of the calculator e.g. Tokenizer must be able to function independent of the other components of the calculator.

Non-Functional Requirements

1. Reliability – The calculator must be able to reliably calculate the correct result of the input expression every time.
2. Testability – The testability of the program is an important feature. The test suite should be flexible in order to allow the ability to find any flaws in the code. Good coding practice to allow for unit testing, integration testing, and system testing must be employed.
3. Ease of use – The program must be easy to use for the target user (Computer Science student/lecturer).
4. Maintainability – The program will be relatively simple so maintainability is not a big issue in this project, however, we will strive to ensure we use best software development practices throughout.

Requirements such as portability, robustness, size, and performance are insignificant for a calculator of this nature.

Interfaces

To interface between each component of the calculator plain text files will be used.

1. Firstly, when the Tokenizer is run, it will take a text file, “input.txt”, containing a single expression written on a single line as input.
   E.g. 2* 3 /2

2. Each character in the input will be converted to a token and written to a new line in the output text file, “infix.txt”.
   E.g. 2
   *
   3
   /
   2

3. The Infix to Postfix converter will then read each token from infix.txt by reading the first token on each line of the file.

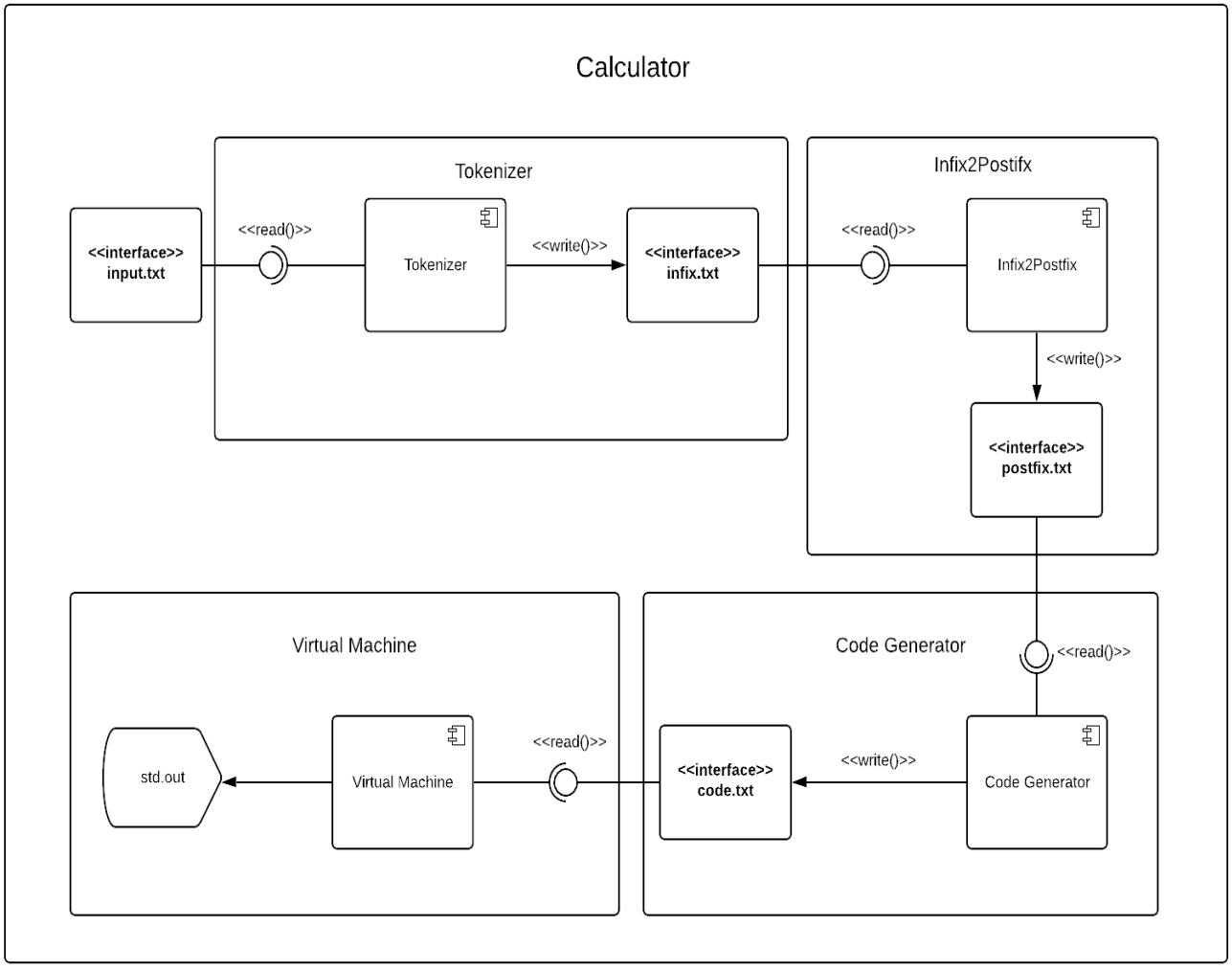
4. The Infix to Postfix converter then writes the postfix output to a file, “postfix.txt”, on a single line with a single space between each token.

E.g. 2 3 * 2 /

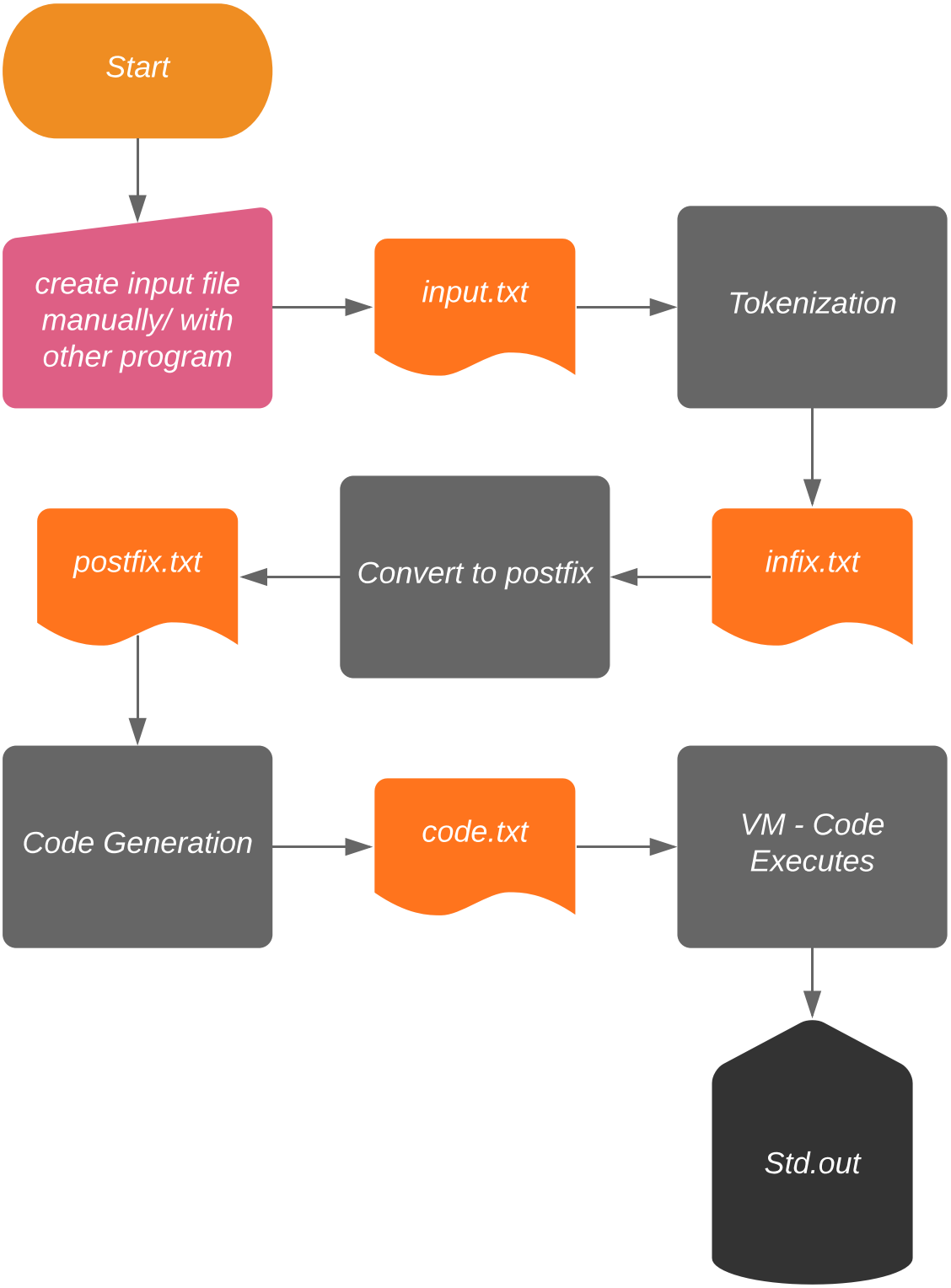
5. The Code Generator reads each token in the first line of postfix.txt.
6. The Code Generator outputs each instruction to be executed by the Virtual Machine to a new line in the output file “code.txt”.
   E.g.
   LOADINT 2
   LOADINT 3
   MUL
   LOADINT 2
   DIV

7. The Virtual Machine then reads each instruction by reading each line in sequence. The final output of the program, a single number, will be to std.out.

High Level Architecture Diagram



Flow Chart


Start
create input file manually/ with other program
input.txt
Tokenization
infix.txt
Convert to postfix
postfix.txt
Code Generation
code.txt
VM - Code Executes
Std.out