Calculator Design Documentation

For CS3500

Authors: Team CS3500.2020.X 9

Conor Holden

Jack McCabe

William Yang

Xi Chen

Revisions

Revision	Revision Details
0	Initial Version
1	Numbered requirements, Added figures to diagrams, Added new requirements, redone diagrams

System Overview

The goal of this project is to create a calculator using the coding language of C. This project will allow the user to solve simple mathematical problems by inputting the equation they want into an input file. The calculator will then take the file with the equation and go through the process of tokenizing, converting infix to postfix, generating the code, and have the code interpreted by a virtual machine.

The system will consist of four components: a tokenizer, infix to postfix converter, code generator, and a virtual machine or interpreter.

- Tokenizer: This will allow the calculator to determine what and how many tokens are present. It will also allow for the tokens to be distinguished.
- Infix-to-Postfix Converter: This conversion will allow for smoother and faster calculations due to the fact that parentheses would not be required while calculating an equation. https://gist.github.com/AnthonyDiGirolamo/1179218/2faf71295ee853facac6dd4f2984672bd9dcb9ad
- Code Generator: This allows for the converted equation to be correctly generated so that it can be read by the virtual machine.
- Virtual Machine/Interpreter: The virtual machine reads and interprets the generated code.

System Requirements

- **A. General Requirements**
  - A.1. Read an input file
  - A.2. Provide an output
  - A.3. Handle blank or no input
  - A.4. Handle errors and exceptions
  - A.5. Adhere to the predefined interfaces
- **B. Tokenizer Requirements**
  - B.1. Find the start and end of a string of numbers
  - B.2. Detect valid and invalid characters
  - B.3. Detect decimal place as part of number
  - B.4. Detect if number is integer or float
- **C. Infix-to-Postfix Requirements**
  - C.1. Produce correct order of operations
  - C.2. Handle parentheses
- **D. Code Generator Requirements**
  - D.1. Identify Operators
  - D.2. Generate Integer or floating point load instruction
- **E. Virtual Machine Requirements**
  - E.1. Detect invalid instruction
  - E.2. Detect impossible instructions e.g dividing by zero
  - E.3. Operations with both integers and float point numbers
  - E.4. Handle Missing instructions or numbers
  - E.5. Detect empty stack or stack with more than one element on output

System Architecture

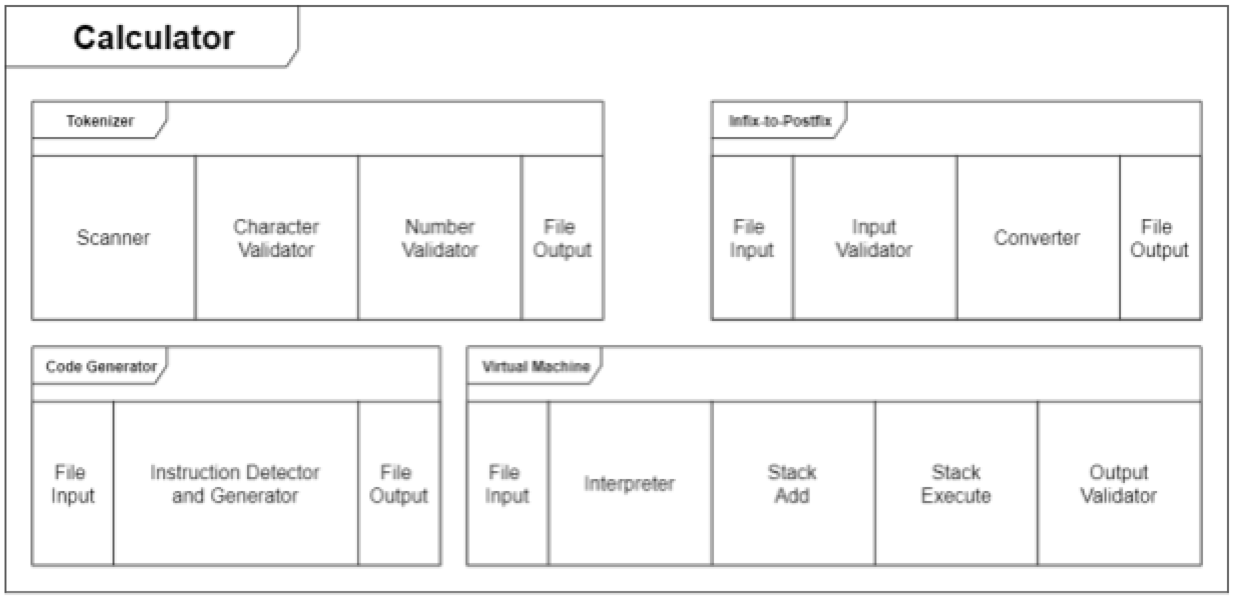

Fig.1 — A diagram for the general system architecture.

Tokenizer

- Scanner: Scans input for numbers and characters
- Number Validator: Validates correct number layout
- Character Validator: Validates legal characters

Infix to Postfix

- Input Validator: Validates correct expression layout
- Converter: Converts infix to postfix

Code Generator

- Instruction Detector and Generate: Detects instruction type and corresponding format

Virtual Machine

- Interpreter: Interprets and runs instructions
- Stack Add/Execute: Adds numbers to stack/Executes instructions on stack
- Output Validator: Validates stack after code execution

Data Flow Diagrams

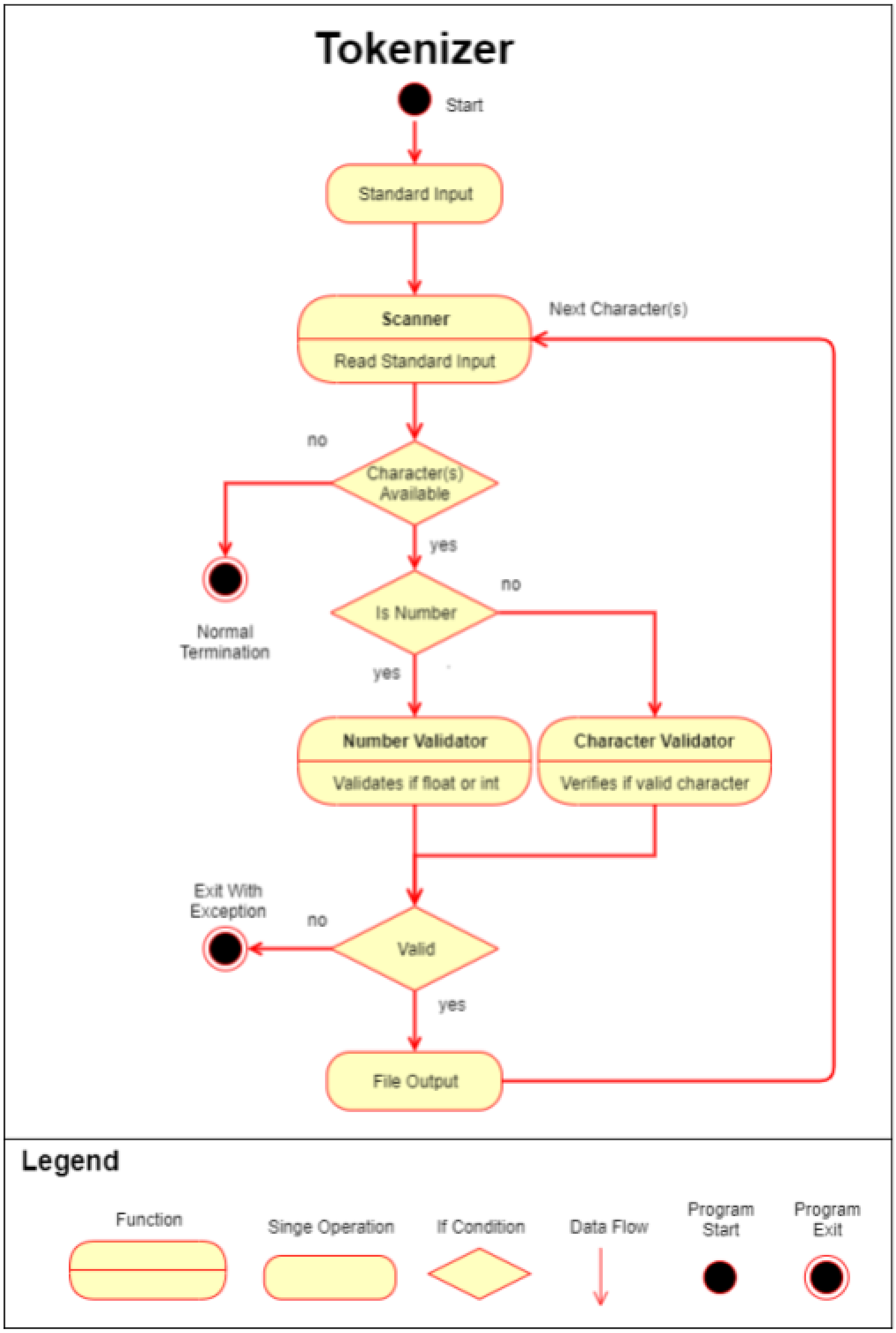

Fig.2 — Tokenizer Data flow Diagram

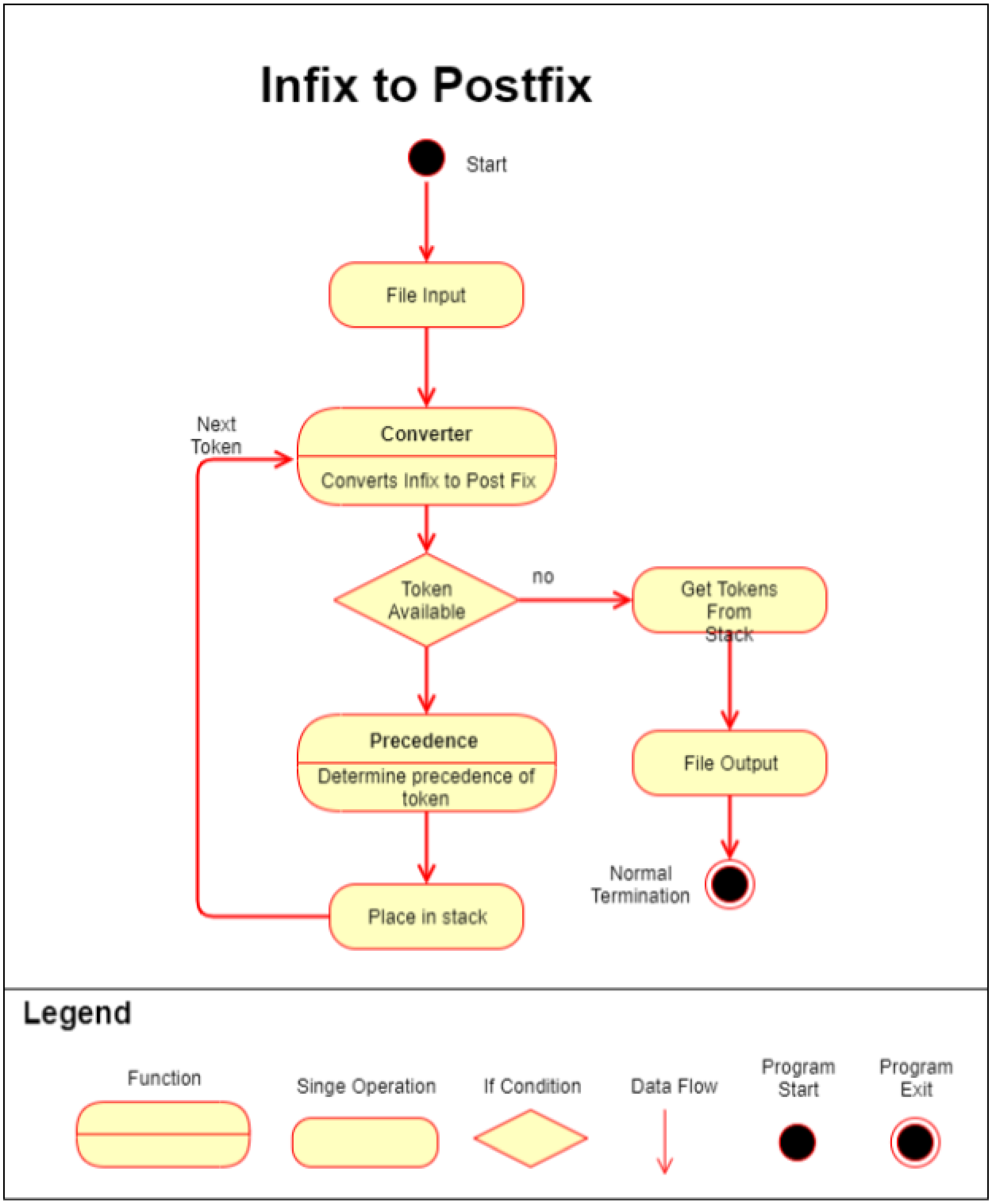

Fig.3 — Infix-to-postfix data flow diagram

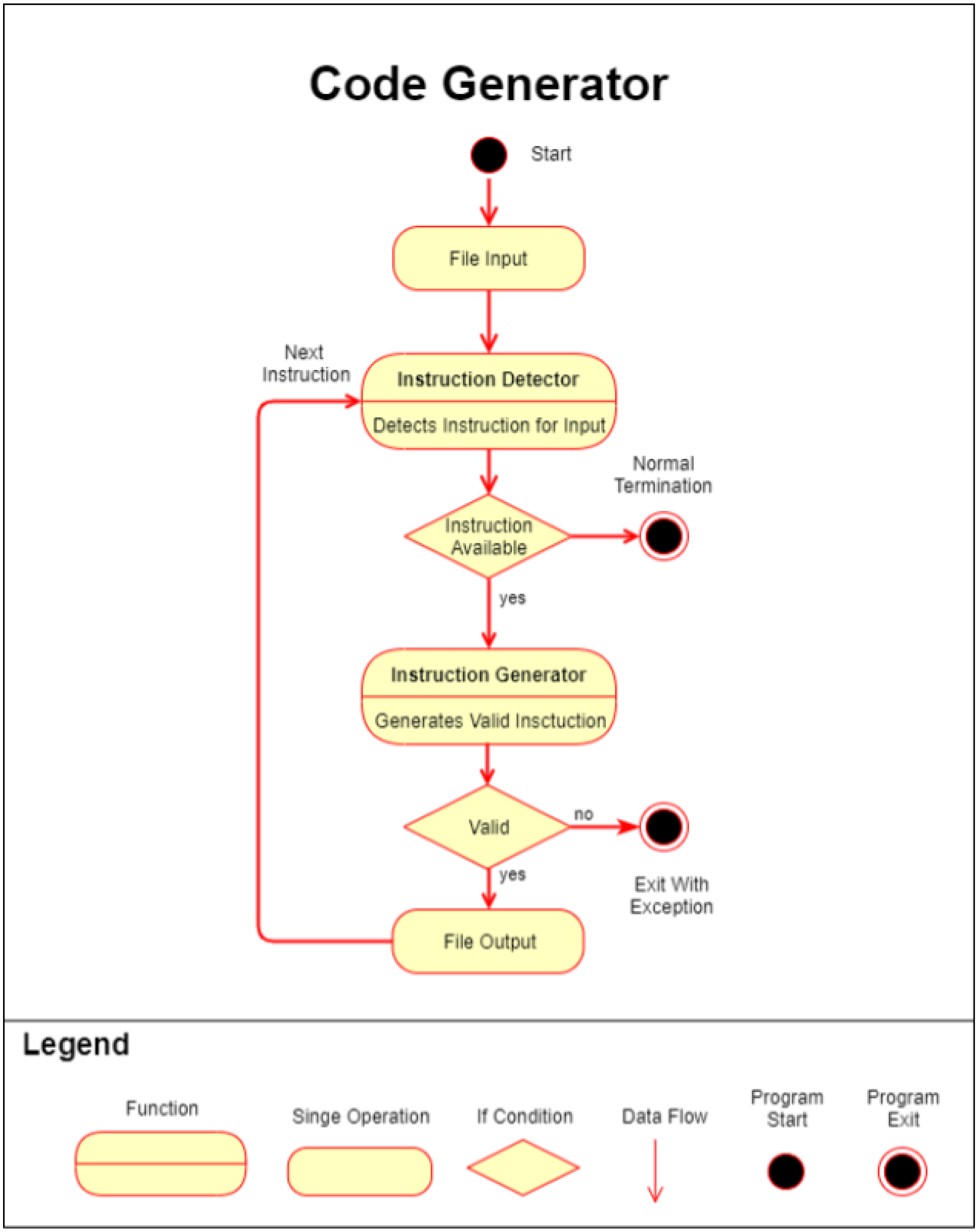

Fig.4 – Code Generator data flow diagram.

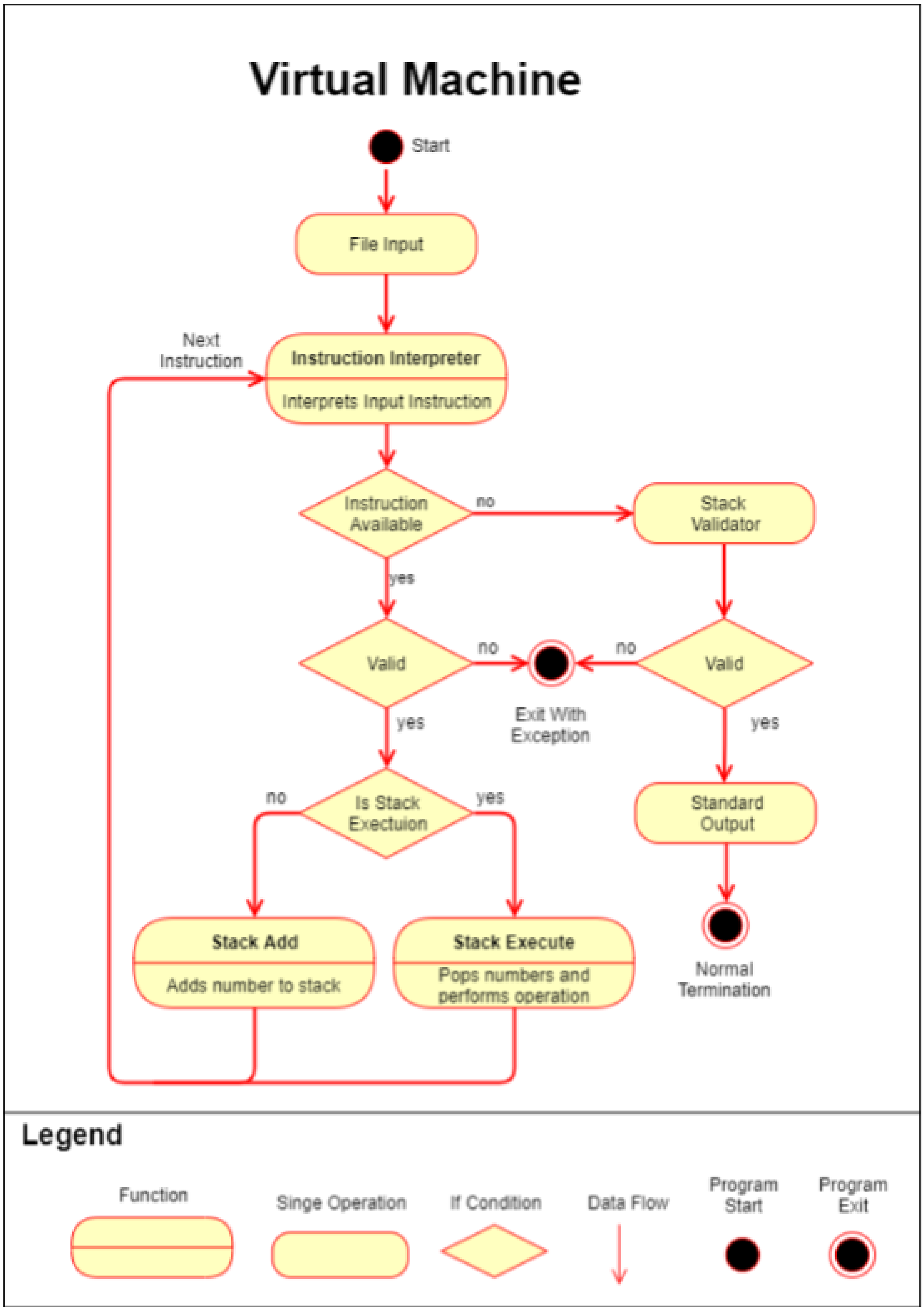

Fig.5 — Virtual machine data flow diagram.

System Interfaces

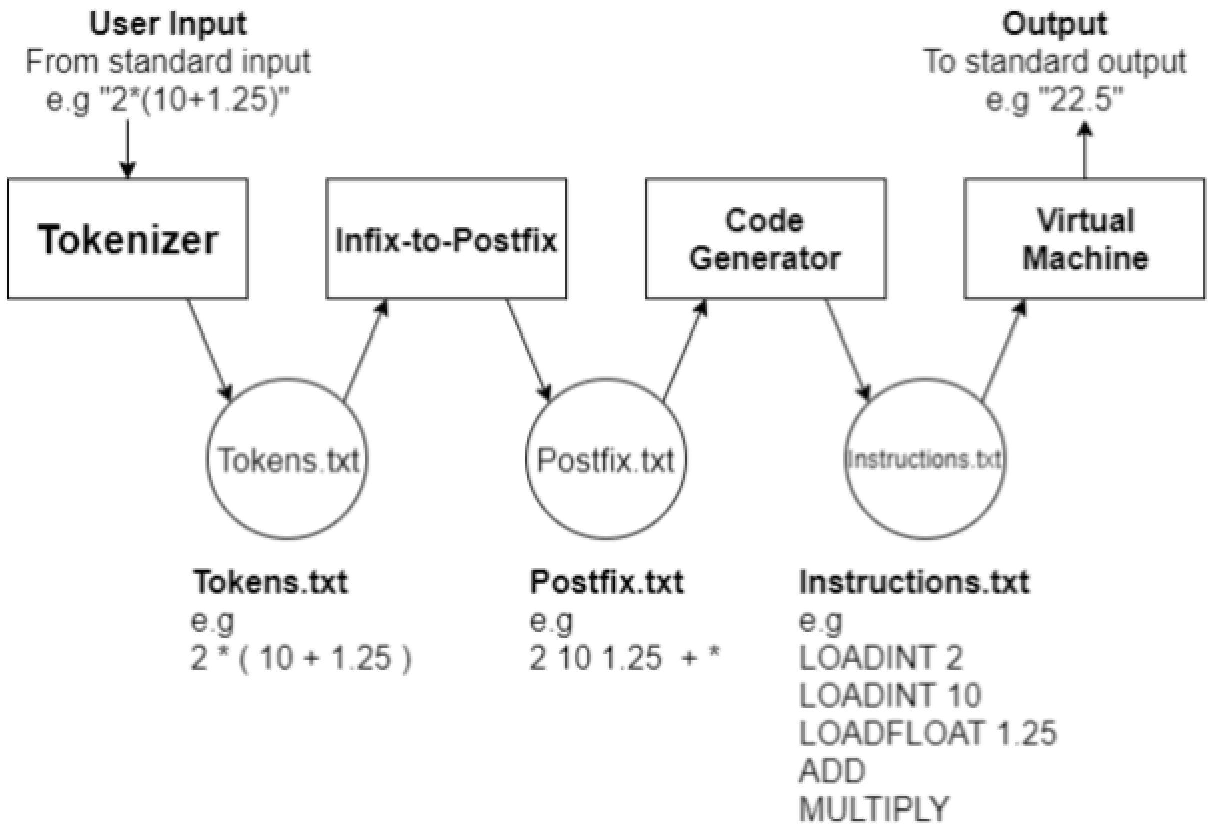

Fig.6 — Interfaces between system components.

User Input: A string mathematical expression read from standard input .
Valid Characters: Numbers 0 - 9, Decimal Point ". ", Operators (+, -, *, /), Parentheses (), Spaces.

Tokens.txt: A text file containing the relevant tokens, separated by spaces, in infix notation ready to be passed as input to the Infix-to-Postfix converter.

Postfix.txt: A text file containing the tokenized data, separated by spaces, in postfix notation ready to be passed as input to the Code Generator.

Instructions.txt: A text file containing the expressions represented as instructions, separated by newlines, that can be interpreted by the Virtual Machine and executed
Valid Instructions: LOADINT (int), LOADFLOAT (float), ADD, SUBTRACT, MULTIPLY, DIVIDE

Output: A single integer or floating point number printed to standard output