University of Nankai

College of Software Engineering

PROGRAMMING WITH HIGH-LANGUAGE

REPORT

A SIMPLE CALCULATOR USING C++

Professor

Bo ZHANG

Student Name	**Student ID**
1. SOMOEURN VIRAKDEN	2120246050
2. UCHITA HIKARU	2120246057

Contents

1 Introduction **2**

2 Problem Analysis **2**

2.1 Understanding the Requirements . . . 2

2.2 Challenges . . . 2

3 System Design **3**

3.1 System Overview . . . 3

3.2 Class Diagram . . . 4

3.2.1 File-Handler . . . 5

3.2.2 Session-Handler . . . 5

3.2.3 Variable-Handler . . . 5

3.2.4 Expression-Handler . . . 6

3.2.5 Relationships . . . 6

1 Introduction

In this project, I developed a command line calculator for multiple systems(decimal, binary and hex)of numbers implemented in C++ having many operations such as addition, subtraction, multiplication, division sin, cos, exponentiation, trigonometric functions and other Mathematical calculations. This calculator is written to calculate arbitrary length mathematical expressions and output will be in float form. In this report we analyze the problem and propose a system design that allows us to implement an optimal solution, providing class diagram example of its implementation as well.

2 Problem Analysis

2.1 Understanding the Requirements

For this project, we wanted a basic calculator which can be run in command-line. The calculator must support:

- Three number systems (bases):
  - Binary (base 2), Decimal (base 10), and Hexadecimal (base 16).
- Operators:
  - Arithmetic operators: $+, -, \times, \div$
  - Exponentiation: ˆ
  - Trigonometric functions: sin(), cos().
  - Parentheses () for grouping expressions.
- Variables:
  - Users can define variables during runtime using expressions such as $i = 10$
  - Variables can be reused in subsequent expressions and calculations.
  - Variable names are case-sensitive and stored with their respective values for dynamic computations.
- Arbitrary-length expressions: The calculator must handle expressions of any length, ensuring correct operator precedence and grouping.
- Output in Decimal: Regardless of the base of the input, the output must always be in decimal format.

2.2 Challenges

- **Base conversion**: One of the key challenge is how to make sure that calculator can interpret numbers in different bases & convert them into decimal for computation.
- **Operator Precedence**: It is very important to cover the order of operator priority and using parentheses right, so handling complex expressions can be evaluated properly.
- **Real Division**: Moreover, making sure that the division returns a real number and not integer division is another headache lure present especially in other kind of few numbers system.
- **Trigonometric Functions**: These must be applied directly to the numbers provided, taking special care with angle units radians.

- **Variable Management**:
  - Managing variables dynamically during runtime, including:
    - Detecting and storing user-defined variables.
    - Allowing variables to be reused in subsequent calculations.
    - Differentiating between valid and invalid variable definitions.
  - Addressing challenges such as:
    - Conflicts between variable names and reserved keywords.
    - Providing meaningful error messages for uninitialized or undefined variables.
    - Optimizing memory usage for efficient storage and retrieval of variables.

3 System Design

The calculator was implemented with the following system components to address the problem requirements:

3.1 System Overview

The flowchart provides a detailed breakdown of the sequence of steps our program follows to process and evaluate data from an input file and generate an output file with the results:

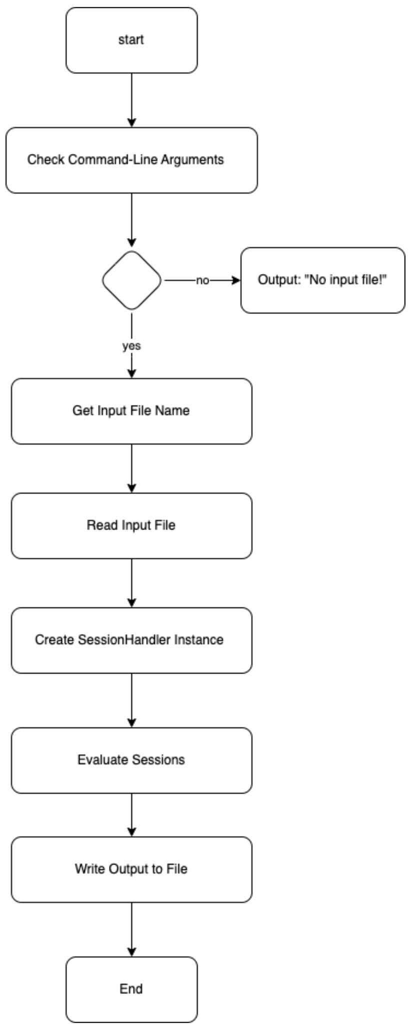

Figure 1: flow-chart

- **Start**: The program begins execution in the main function.
- **Check Command-Line Arguments**: The program checks if an input file was provided as a command-line argument, which is the file name:
  - If yes, it retrieves the input file name.
  - If no, it outputs an error message and terminates.
- **Read Input File**: The program reads the content of the specified input file using FileHandler::readFromFile().
- **Create SessionHandler Instance**: An instance of SessionHandler is created to manage the sessions read from the file.
- **Evaluate Sessions**: The evaluateSession() method of SessionHandler is called to evaluate the mathematical expressions and store the results.
- **Write Output to File**: The results are written to an output file using FileHandler::writeOutputFile().
- **End**: The program completes its execution.

3.2 Class Diagram

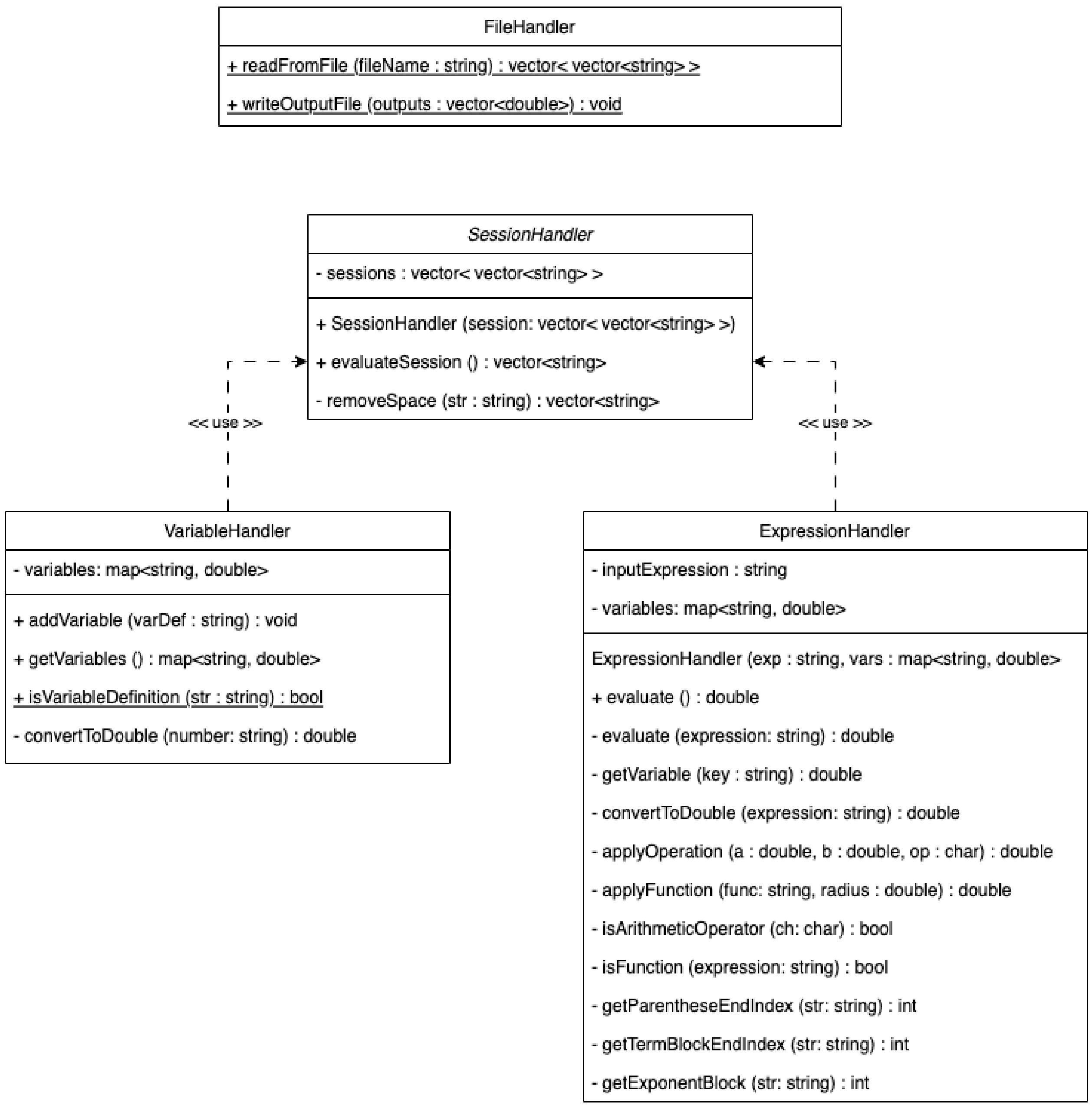
FileHandler
+ readFromFile (fileName : string) : vector< vector<string> >
+ writeOutputFile (outputs : vector<double>) : void
SessionHandler
- sessions : vector< vector<string> >
+ SessionHandler (session: vector< vector<string> >)
+ evaluateSession () : vector<string>
- removeSpace (str : string) : vector<string>
<< use >>
<< use >>
VariableHandler
- variables: map<string, double>
+ addVariable (varDef : string) : void
+ getVariables () : map<string, double>
+ isVariableDefinition (str : string) : bool
- convertToDouble (number: string) : double
ExpressionHandler
- inputExpression : string
- variables: map<string, double>
ExpressionHandler (exp : string, vars : map<string, double>
+ evaluate () : double
- evaluate (expression: string) : double
- getVariable (key : string) : double
- convertToDouble (expression: string) : double
- applyOperation (a : double, b : double, op : char) : double
- applyFunction (func: string, radius : double) : double
- isArithmeticOperator (ch: char) : bool
- isFunction (expression: string) : bool
- getParentheseEndIndex (str: string) : int
- getTermBlockEndIndex (str: string) : int
- getExponentBlock (str: string) : int

Figure 2: Class-Diagram

This class diagram provides an overview of the main components in the program and their relationships. This structure allows for modular handling of sessions, variable storage, and expression evaluation in a clean, object-oriented way. Here's a description of each class and its responsibilities:

3.2.1 File-Handler

- **Responsibilities**: Manages file input and output operations.
- **Attributes**:
  - None.
- **Methods**:
  - $\underline{+readFromFile(fileName : string) : vector < vector < string \gg}$ Reads session data from the input file and returns a nested vector of strings containing the sessions.
  - $\underline{+writeOutputFile(outputs : vector < double >) : void :}$ Writes a vector of computed outputs (as doubles) to an output file.

3.2.2 Session-Handler

- **Responsibilities**: Handles and processes multiple sessions of data.
- **Attributes**:
  - $-sessions : vector < vector < string \gg$: Stores a list of sessions, each represented as a vector of strings.
- **Methods**:
  - SessionHandler $(session : vector < vector < string \gg)$: Constructor that initializes the $SessionHandler$ with a session vector.
  - $+evaluateSession() : vector < string >$: Evaluates all sessions and returns a vector of results as strings.
  - $-removeSpace(str : string) : vector < string >$: Utility function to remove spaces from a given string and return a vector of strings. This function aids in cleaning the input before processing.

3.2.3 Variable-Handler

- **Responsibilities**: Manages variables, storing them as a map and providing methods for variable handling.
- **Attributes**:
  - $-variables : map < string, double >$: Stores variable names and their corresponding values.
- **Methods**:
  - $+addVariable(varDef : string) : void)$:Adds a variable to the map, interpreting the definition from a string input.
  - $+getVariables() : map < string, double >$: Returns all stored variables as a map.
  - $\underline{+isVariableDefinition(str : string) : bool}$: Determines if a given string represents a variable definition expression.
  - $-convertToDouble(number : string) : double$: Converts a string representation of a number into a double.

3.2.4 Expression-Handler

- **Responsibilities**: Evaluates mathematical expressions and performs related operations.
- **Attributes**:
  - $-inputExpression : string$: Holds the input mathematical expression as a string.
  - $-variables : map < string, double >$: Stores variable names and values for use in expression evaluation.
- **Methods**:
  - $ExpressionHandler(exp : string, vars : map < string, double >)$:Constructor that initializes $ExpressionHandler$ with an expression and a variable map.
  - $+evaluate() : double$: Evaluates the stored $inputExpression$ directly and returns the result as a double (for public call).
  - $-evaluate(expression : string) : double$: Evaluates the provided expression and returns the result as a double.
  - $-getVariable(key : string) : double$: Retrieves the value of a specified $variable$ from the variables map.
  - $-convertToDouble(expression : string) : double$: Converts a string expression which can be a number or a variable to a double.
  - $-applyOperation(a : double, b : double, op : char) : double$: Applies a basic arithmetic operation between two numbers based on the specified operator.
  - $-applyFunction(func : string, radius : double) : double$: Applies a mathematical function (Sine, Cosine) based on the function name and input value.
  - $-isArithmeticOperator(ch : char) : bool$: Checks if a character is an arithmetic operator.
  - $-isFunction(expression : string) : bool$: Determines if a given expression represents a function.
  - $-getParenthesesEndIndex(str : string) : int$: Finds the index of the closing parenthesis for a given string expression.
  - $-getTermBlockEndIndex(str : string) : int$: Finds the end index of a block term in a complex expression.
  - $-getExponentBlock(str : string) : int$: Identifies the index where an exponent block ends.

3.2.5 Relationships

- **FileHandler** is a utility static class in the program that is used mainly to read and write files. This class has no direct relation with other classes.
- **SessionHandler** uses **VariableHandler**, and **ExpressionHandler** to perform its session evaluation tasks.
- **VariableHandler** and **ExpressionHandler** are utility classes used by **SessionHandler** to manage variables and evaluate expressions, respectively.