NANKAI UNIVERSITY

JINNAN CAMPUS

College of Software

Programming with High Level Language December 30, 2024

Simple Calculator

Group Members

1. LAGGAH JULIUS ____________________ ID 2120246034
2. JUSU ABDUL KARIM ____________________ ID 2120246025
3. TURAY ZIDIDA-DAMEKHALY ___________ ID 2120246056
4. SANDAR WIN ____________________ ID 2120246058

Table of Contents

Introduction

The simple calculator program is a C++ command-line application made to accurately and efficiently evaluate mathematical expressions. Built using the C++ programming language, this program demonstrates core concepts of ObjectOriented Programming (OOP), including encapsulation, abstraction, inheritance and polymorphism. By following a predetermined order of operations, this program makes complicated arithmetic computations easier. It integrates sophisticated capabilities like variable handling and trigonometric operations and provides strong support for input through external files or direct user interaction. The calculator was created with the user experience in mind and is flexible, easy to use, and able to handle errors with grace.

Problem Statement

Mathematical problem-solving often requires tools capable of handling operations in a structured order. This program solves mathematical expressions using the BODMAS (Brackets, Orders, Division, Multiplication, Addition and Subtraction) rule. It is intended to:

1. Solve complex mathematical expressions while following to the BODMAS rule.
2. Support trigonometric functions for more advanced calculations.
3. Handle expressions with variables, enabling dynamic and adaptable problem-solving.

Order of Operations and Execution

The calculator evaluates expressions using the **BODMAS** rule to ensure accurate and consistent computation. The operations are prioritized and executed in the following sequence:

1. **Parentheses**: Resolves expressions within parentheses first, starting from the innermost to the outermost.

2. **Exponents**: Computes any exponential values after resolving parentheses.

3. **Multiplication and Division**: Processes these operations from left to right as they appear in the expression.

4. **Addition and Subtraction**: Finalizes the computation by solving these operations from left to right.

Inputs and Input Types

Input Methods

The program accepts two primary types of input:

- **Direct Input**: Mathematical expressions entered interactively by the user.
- **File Input**: A text file containing multiple expressions, processed line by line.
    - This is the most recommended input method for this program.

Supported Number Formats

The calculator supports the following number formats:

- **Decimal Numbers**: For example, 3.14.
- **Binary Numbers**: Indicated with a b suffix, e.g., 1010b.
- **Hexadecimal Numbers**: Indicated with a 0x prefix, e.g., 0xFF.

Process Flow

The program evaluates expressions by following these steps:

1. Accepts input (either interactively from the user or via a file).
2. Parses the expressions into tokens (numbers, operations, and parentheses).
3. Evaluates the tokens using the **BODMAS rule**.
4. Displays the results or returns errors if the input is invalid.

Arithmetic Operations in the Calculator

The calculator supports the following arithmetic operators:

S/N	Operator	Function
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	^	Exponents (Power)
6	()	Grouping (Parentheses)

These operators are evaluated in the order of precedence defined by the **BODMAS rule**, ensuring accurate results for both simple and complex expressions.

Expressions of Trigonometric Functions

S/N	Function	Example	Result
1	SIN	SIN (0)	0
2	COSINE	COS (0)	1

Expressions of Variables and Their Inputs

S/N	Variable Definition	Expression	Result
1	Pi = 3.14, Radius = 3	Pi * Radius ^ 2	28.26
2	I = 5, J = 3	I + J	8

Outputs

The calculator provides results in the following formats:

1. **Decimal Numbers**: e.g., 3.14
2. **Integer Numbers**: e.g., 5
3. **Fractional Results**: e.g., 1/2 or 0.5

Functionalities of the Program

S/N	Functionalities	Inputs	Expected Output
1	Single numbers	5	5
2	Simple calculations	1 + 2	3
3	Trigonometric functions	SIN (PI / 2)	1
4	Parentheses calculations	(1 + 2) * 3	9
5	Variable-based expressions	I = 3	3
6	Complex expressions	PI * Radius ^ 2	28.26

Design of the Program

The main components that constitute the development of the program are described below:

1) **ArithmeticOperations Class:** Execute mathematical calculations such as addition, subtraction, multiplication, division, and exponentiation based on parsed input and handle operations involving decimals, integers, and fractions.

2) **BaseConversion Class:** Handles conversions between different number systems such as binary, hexadecimal, decimal). It supports the interpretation

and conversion of binary (b suffix) and hexadecimal (0x prefix) integers and transforms input numbers into a unified format for arithmetic processing.

3) **ConstantManager Class:** Manages the use of mathematical constants (e.g., PI, E) throughout the program. It implements trigonometric functions like sine (SIN) and cosine (COS), permits dynamic usage of predefined constant values, and offers quick access to constants for expression evaluation.
4) **VariableManager Class:** Handles user defined dynamic variables during runtime. In addition to enabling users to define, modify, and retrieve variables, it also verifies and ensure variables are initialized prior to use.
5) **ExpressionParser Class:** Mathematical expressions are parsed and tokenized into manageable components such as operators, operands, parentheses. It analyzes user input to identify numbers, operators, and functions and validates expressions to ensure structural integrity (e.g., matching parentheses). Additionally, it can organize tokens into a format suitable for evaluation by the ArithmeticOperations class.
6) **Main Class:** Serves as the entry point for the program, coordinating input processing, computation, and output. It can accept input directly from the user or via a file then coordinate the flow between ExpressionParser, Arithmetic Operations and associated classes. Finally, display results or error messages to the user.

The design of the program follows object-oriented programming (OOP) principles, ensuring a clear and modular structure where each component is focused on specific responsibility. This approach enables debugging, testing, and future expansion easier.

Error Handling in the Program

The calculator program solid error-handling features to ensure a smooth user experience and deliver meaningful feedback for incorrect inputs or unexpected situations. The following are the primary methods used:

1. Exception Handling: Address runtime errors to keep programs from crashing and maintain uninterrupted operation.
2. File Handing: Check for the presence of necessary files and handles errors if files are inaccessible.
3. Input Validation: Assure that user inputs follow the correct formats and ranges, minimizing potential calculation errors.
4. Logical Error Detection: Detect invalid mathematical operations, such as division by zero or invalid expressions.

5. Boundary Handling: Manages mismatched parentheses or other structural irregularities in mathematical expressions.
6. Unassigned Variable Handling: Reports errors if an unassigned variable is referenced.

The program ensures computational integrity and improves the user experience overall by resolving these frequent problems. These error handling mechanisms reflect reliability, and adaptability in a range of mathematical contexts.

Conclusion

The calculator program offers an example of how object-oriented programming concepts can be applied effectively to solve challenging mathematical issues in an organized and approachable manner. The program is appropriate for both basic and complex use cases since it guarantees accurate computations by following the BODMAS rule. Including variable handling, trigonometric operations, and support for multiple input methods increases the ability of the program. Along with making testing and debugging easier, its object-oriented and modular architecture also creates the framework for future improvements. In addition to error-handling features, the program provides reliability and a smooth user experience.

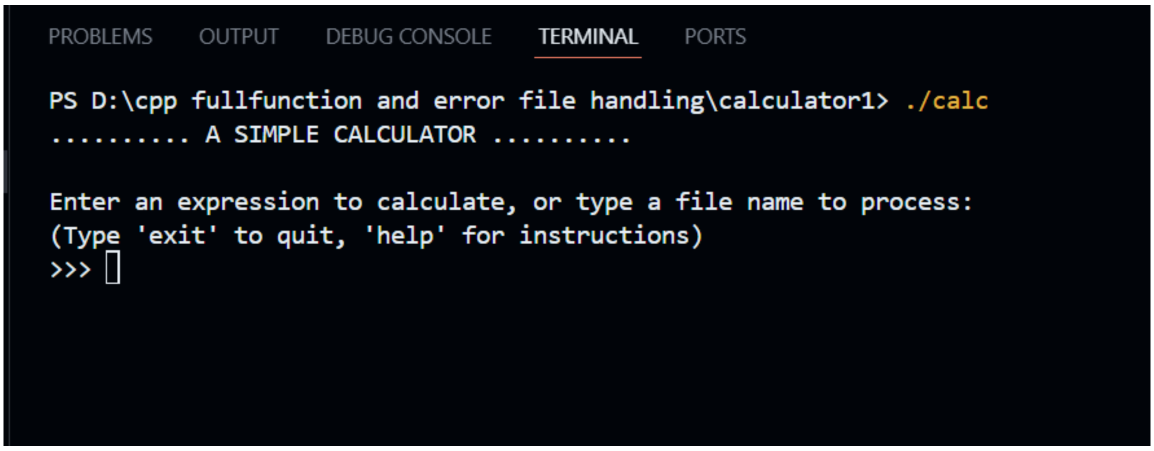
In order to successfully run and compile this program, the following steps are considered.

1. Extract the zip folder into a desirable drive folder
2. Import the folder into an IDE
3. On the terminal, use a standard compiler e.g

```
g++ -o calculator main.cpp ExpressionParser.cpp VariableManager.cpp
ArithmeticOperations.cpp ConstantManager.cpp Calculator.cpp BaseConversion.cpp -std=c++11
```

4. Run the program after compilation using the command ./calc
5. After running the command, you will see a prompt like:

Enter an expression to calculate, or type a file name to process (press exit to quit, type help for instructions)

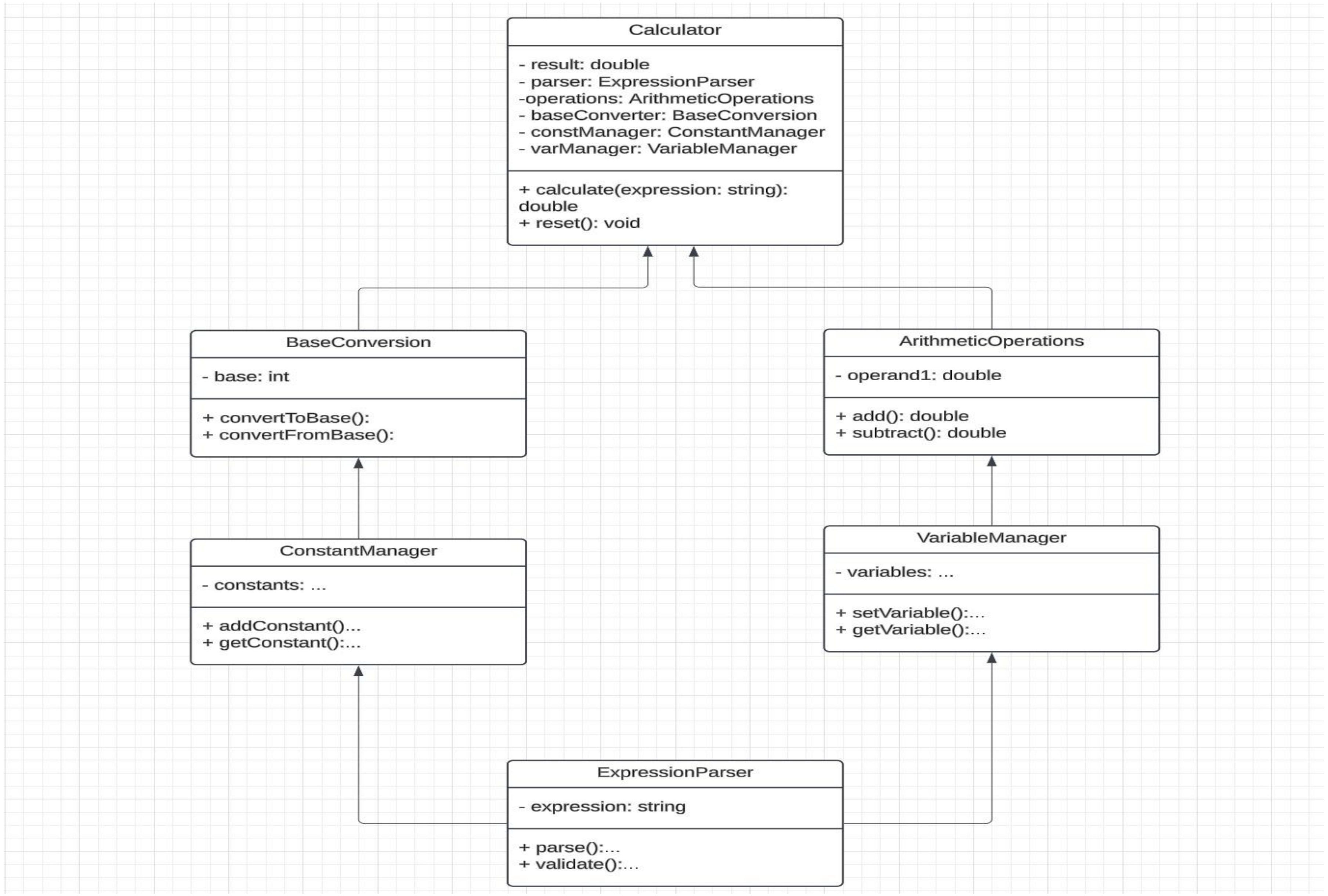


```
PS D:\cpp fullfunction and error file handling\calculator1> ./calc
.......... A SIMPLE CALCULATOR ..........

Enter an expression to calculate, or type a file name to process:
(Type 'exit' to quit, 'help' for instructions)
>>> 
```

6. At this point you can write a direct mathematical expression to calculate, import a file to calculate, type exit to quit or type help for instructions.
7. If you encounter any error double check the file name and location during compilation and ensure the input file (e.g. expressions.txt) is in the same directory as the program or provide a full path.

UDM Class Diagram


Calculator
- result: double
- parser: ExpressionParser
-operations: ArithmeticOperations
- baseConverter: BaseConversion
- constManager: ConstantManager
- varManager: VariableManager
+ calculate(expression: string): double
+ reset(): void
BaseConversion
- base: int
+ convertToBase():
+ convertFromBase():
ArithmeticOperations
- operand1: double
+ add(): double
+ subtract(): double
ConstantManager
- constants: ...
+ addConstant()...
+ getConstant():...
VariableManager
- variables: ...
+ setVariable():...
+ getVariable():...
ExpressionParser
- expression: string
+ parse():...
+ validate():...