

**COP3530**

**Project #1**

**Calculator**

For this project you need to implement a scientific calculator program. This calculator should be able to read mathematical expressions and assignments, compute the result of the expressions and execute the assignments.

## Expressions.

A mathematical expression can be made up of numbers, variables, mathematical operations, constants, mathematical functions, and parentheses. Some examples of mathematical expressions:

- $2.3 + 4 * 3 - 2$
- $(0.4 + 2) * (4 - 12.5)$
- $2^2 + (4-3) ^ 2 * (1-3) )$
- $(count + 1) ^ (growth/10)$
- $x^{(0-y)^2}$
- $(((((1+2)+3)+4)+5)+6)$
- $2 * \pi * radius^2$
- $4.5 - e ^ \log(4.5)$
- $\sin(angle)^2 + \cos(angle)^2$
- $\sin(2.4)$

In order to compute the result of an expression you need to find out the value of variables used in the expression (if any), and calculate the final value of the expression based on the operators and functions.

## Constants.

Your program should support using the following mathematical constants:

- $\pi = 3.14169$
- $e = 2.718$

## Functions.

Your program should support the use of the following mathematical functions in the expressions:

- $\sin$  : the given argument is in Radian.
- $\cos$  : the given argument is in Radian.
- $\log$  : computes the natural logarithm of the given argument.

## Assignments.

An assignment is made up of a variable and an expression and is used to store the result of the expression in the variable. The format of an assignment is `let <variable-name> = <expression>` and the given variable could be used in any expression after this assignment.

## Variables.

The variables' names are strings of alphabetical characters, i.e. 'a' to 'z' and 'A' to 'Z'. You should use a hash-map to store the values of the variables. Note that the names of the variables are case-sensitive, in other words `Count` and `count` are two different variables and can have different values. The names `quit`, `let`, `Pi`, `e`, `sin`, `cos`, `log` will not be used as variable names since they are reserved for constants, functions, to signal the termination of the program and to declare the variables. Some examples of assignments:

- `let x = 2`
- `let count = 0`
- `let BigValue = 2^2^(2+2^(2+2*2))`
- `let count = count + 1`
- `let AveryVeryLongVariableNameIsAlsoPossible = log( BigValue * sin(1.4) )`

## Numbers.

All numbers are double-precision floating-point numbers. When entering the numbers, the fractional part may be omitted (which means the fractional part is equal to zero, i.e. 1 is equal to 1.0), but the integer part will always be present. Example expression:

- `1 + 1.0 + 1.1 + 0.1`

## Operators.

Your program should support the following five operators.

- `+` : Addition.
- `-` : Subtraction.
- `*` : Multiplication.
- `/` : Division.
- `^` : Exponentiation.

## Operator Precedence.

Operator precedence determines the order of the operations in the absence of the parentheses. For example, in the expression `2 + 3 * 4` the result would be different based on which operator we are computing first. Operator precedence clears up this ambiguity. For instance, in the same example, since we know that `*` has a higher precedence than `+`, we will compute the sub-expression `3 * 4` first. In other words, the expressions `2 + 3 * 4` and `2 + (3 * 4)` are equivalent.

Op.	Precedence	Example Exp.	Equivalent Exp.
$\wedge$	3 (Highest)	$2^3$	$(2^3)$
$*$	2	$2 * 4 \wedge 2 * 3$	$(2 * (4 \wedge 2) * 3)$
$/$	2	$4 / 2 \wedge 4$	$(4 / (2 \wedge 4))$
$+$	1 (Lowest)	$2 + 4 * 3 + 2 \wedge 3 * 4$	$(2 + (4 * 3) + ((2 \wedge 3) * 4))$
$-$	1 (Lowest)	$4 - 2 * 3 \wedge 4$	$(4 - (2 * (3 \wedge 4)))$

## Operator Associativity.

Operator associativity determines the order of multiple operations of the same precedence in the absence of the parentheses. For example, in the expression  $4 / 2 * 2$  the result would be different based on which operator we are computing first. Operator associativity clears up this ambiguity. For instance, in the same example, since we know that the operators  $*$  and  $/$  are associative from left to right, we will compute the sub-expression  $4 / 2$  first. In other words, the expressions  $4 / 2 * 2$  and  $(4 / 2) * 2$  are equivalent.

Op.	Associativity	Example Exp.	Equivalent Exp.
$\wedge$	Right to left	$2 \wedge 3 \wedge 4 \wedge 5$	$(2 \wedge (3 \wedge (4 \wedge 5)))$
$*, /$	Left to right	$2 * 3 / 4 * 5$	$((2 * 3) / 4) * 5$
$+, -$	Left to right	$2 - 3 - 4 + 5$	$((2 - 3) - 4) + 5$

The following table contains a few more examples regarding the precedence and associativity of the operators.

Example Exp.	Equivalent Exp.
$2 - 3 * 4 + 5$	$((2 - (3 * 4)) + 5)$
$2 + 3 * 4 / 5 - 6 * 7 \wedge 8 \wedge 9 * 10$	$((2 + ((3 * 4) / 5)) - ((6 * (7 \wedge (8 \wedge 9))) * 10))$

## Parentheses.

The parentheses will always override the order of the operations, regardless of the precedence and/or the associativity of the operators.

## Whitespaces.

An expression may contain any number of whitespaces between the operators, numbers, constants and parentheses.

## Error Handling.

You do not need to check the expression for any syntax errors. The expressions are never malformed, i.e. they will not contain errors such as two consecutive operators, two numbers without any operators, undefined constants, etc., but you still need to check and report two possible errors. One is the division-by-zero error which happens if some number is being divided by zero in the expression. The other is the undeclared-variable error which happens if the expression is using a variable that has not been declared in a previous assignment.

## Input/Output.

Each line of input contains an expression or an assignment. If the input is an assignment your program should compute the result of the right hand side expression and update the value of the left hand side variable. Nothing should be printed for assignments. If the input is an expression, your program should compute the result of the expression and print the result to the output. The result is either a number, the string "Division-By-Zero" if there is a division-by-zero error in the expression, or the string "Undeclared-Variable" if there is an undeclared-variable error in the expression. The final line of input will be the string "quit", and after your program reads that line it should terminate without printing anything. An example running of the program (input is black, output is blue):

```
2 + 4 / 2
4
2.3+2.7^2
9.59
4 + 3 / ( 2 - 4 / 2)
Division-By-Zero
2 + 3 * 2 - 4 * 10
-32
4 / myVar
Undeclared-Variable
let myVar = 4
4 / myVar
1
let myVar = myVar ^ myVar
myVar
256
let x = (2 + 4 * 12 - (14 - 6)*2)
myVar - x*2
188
quit
```

## Notes

- You need to upload your code to Canvas. You will need to demo your project to the TAs after the deadline of the project.
- For questions, please use the Canvas discussion board.
- You can use C++11's `unordered_map` for your hash map.
- Your code needs to be compiled with GCC/G++. It's your responsibility to make sure there is no compilation errors.
- Calculating an expression should be in  $O(n)$  with  $n$  being the number of characters in the expression.