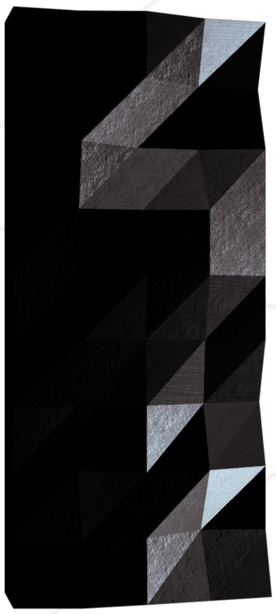

Computor v2

"Your homemade basic calculator"

Summary: This project is the first of a series to renew your relationship with mathematics. It will be very useful—even essential—for many more projects.

Version: 3.1

Contents

Chapter I

Préambule

A trebuchet (French: trébuchet) is a type of siege engine that uses a swinging arm to hurl projectiles at the enemy. The traction trebuchet first appeared in China during the 4th century BC as a siege weapon. It spread westward, likely introduced by the Avars, and was adopted by the Byzantines in the mid-6th century AD. It uses manpower to swing the arm. The later counterweight trebuchet, also known as the counterpoise trebuchet, uses a counterweight to swing the arm. It appeared in both Christian and Muslim territories around the Mediterranean in the 12th century and was reintroduced to China via the Mongol conquests in the 13th century. When the term "trebuchet" is used without further specification, it typically refers to the counterweight trebuchet.

Chapter II

Introduction

After familiarizing yourself with polynomials in the first part of the Computor project, you will now expand your mathematical knowledge by implementing an interpreter in the programming language of your choice, similar to the command-line tool bc. This project consists of two main components: the first is variable assignment (i.e., the ability to store variables within the program's context), and the second is equation resolution.

The project will in a sense be the best of both worlds between the Google and the bc, command to make it a powerful yet easy to use interpreter.

Chapter III

Objectives

This project aims to introduce or reinforce certain aspects of mathematics:

- Complex numbers
- Real numbers
- Matrices
- The theory of functions

You will also explore the operational priorities and computational properties of these various elements. The goal of this project is to provide you with a solid foundation in understanding these tools, which will be valuable in the realization of your projects at 42, as well as beyond 42, particularly those that require mathematical knowledge.

Chapter IV

General Instructions

- This project will be corrected by humans only. You're allowed to organise and name your files as you see fit, but you must follow the following rules.
- The language is up to you.
- You cannot use any native type of variables to the language chosen for the management of complex numbers.
- No library easing the management of complex numbers/matrices is authorized, except for those you will implement yourself.
- If you are working in a compilable language (C/C++ for ex) you will submit a `Makefile`. This `Makefile` must have all the usual rules. It must recompile and re-link the program only if necessary.
- You are responsible for the proper installation of the dump environment on which you will be developing and on which you will be corrected.
- The Norm is not applied to this project. However, we ask you to be clear and verbose about the design of your source codes.
- You are allowed to use other functions to complete the bonus part as long as their use is justified during your defence. Be smart!

Chapter V

Mandatory part

Before fully immersing yourselves in this subject, we recommend reviewing sets, what they contain, the constraints they adhere to, etc. This will help you understand the mathematical notation used in this subject.

For the moment, everything you need is accessible on Wikipedia.

V.1 General presentation

`computor-v2` is an instruction interpreter that, like a shell, retrieves user inputs for advanced computations.

It will present the form of a simple command interpreter, and will have to be able to answer the following specifications (features detailed in the following pages):

- Support for the following mathematical types:
  - Rational numbers
  - Complex numbers (with rational coefficients)
  - Matrices
  - Polynomial equations of degrees less than or equal to 2
- Assignment of an expression to a variable by type inference
- Reassignment of an existing variable with an expression of another type
- Assignment of a variable to another variable (existing or not)
- Resolution of a mathematical expression with or without defined variable (s)
- Resolution of an equation of degree less than or equal to 2
- Operations between types, as much as possible
- Exit the program itself (by keyword, signal, keyboard shortcut ...)

V.2 Assignment part

Your `computor-v2` must be able to assign variables of the types borrowed to mathematics, which you will not necessarily find in a conventional programming language. You MUST create these types of variables and embed them transparently in your program.

Thus, you will create a type for:

- Rational numbers (for any $x \in \mathbb{Q}$)

```
$./computorv2
> varA = 2
  2
> varB = 4.242
  4.242
> varC = -4.3
  -4.3
>
```

- Imaginary numbers for any $x = a + ib$ such as $(a, b) \in \mathbb{Q}^2$

```
$./computorv2
> varA = 2*i + 3
  3 + 2i
> varB = -4i - 4
  -4 - 4i
>
```

- Matrices for any $A \in \mathbb{M}_{n,p}(\mathbb{Q})$

```
$./computorv2
> varA = [[2,3];[4,3]]
  [ 2 , 3 ]
  [ 4 , 3 ]
> varB = [[3,4]]
  [ 3 , 4 ]
>
```

- Functions (with only one variable)

```
$./computorv2
> funA(x) = 2*x^5 + 4x^2 - 5*x + 4
  2 * x^5 + 4 * x^2 - 5*x + 4
> funB(y) = 43 * y / (4 % 2 * y)
    43 * y / (4 % 2 * y)
> funC(z) = -2 * z - 5
  -2 * z - 5
>
```

Your program must be able to reassign a variable and change the type of variable by inference in a way that:

```
$./computorv2
> x = 2
  2
> y = x
  2
> y = 7
  7
> y = 2 * i - 4
  -4 + 2i
>
```

It will also have to be able to reassign the result of a computation to a variable so that:

```
$./computorv2
> varA = 2 + 4 *2 - 5 %4 + 2 * (4 + 5)
  27
> varB = 2 * varA - 5 %4
  53
> funA(x) = varA + varB * 4 - 1 / 2 + x
  238.5 + x
> varC = 2 * varA - varB
  1
> varD = funA(varC)
  239.5
>
```

V.3 Computational part

- The 4 conventional operations can be used, namely: $* \ / + -$.
  The program must also manage the modulos with the % operator as well as the matrix multiplication that will be noted $**$. The term-to-term multiplication of two matrices or of a scalar and a matrix is noted with a $*$.
- It must also manage the computations of integer and positive powers (or zero) with the ^ operator
- The program must handle parenthesis and computation priorities.
- Set a function/variable then = ? should be used to return the value of this variable in the context of the program.
- The resolution of a computation is symbolized by the ? operator at the end of the input

```
$./computorv2
> a = 2 * 4 + 4
  12
> a + 2 = ?
  14
>
```

- It will have to manage image computation:

```
$./computorv2
> funA(x) = 2 * 4 + x
  8 + x
> funB(x) = 4 -5 + (x + 2)^2 - 4
  (x + 2)^2 - 5
> funC(x) = 4x + 5 - 2
  4 * x + 3
> funA(2) + funB(4) = ?
  41
> funC(3) = ?
  15
>
```

- It will also have to manage the computation of the square roots of polynomials when the degree is less than or equal to 2, proposing on $\mathbb{R}$ or $\mathbb{C}$.Good news, you have already done it in part on `computor-v1`

```
$./computorv2
> funA(x) = x^2 + 2x + 1
  x^2 + 2x + 1
> y = 0
  0
> funA(x) = y ?
  x^2 + 2x + 1 = 0
  Une solution sur R :
  -1
>
```

V.4 Syntax part

Variable/Function names should only contain letters and must be case insensitive so that `varA` and `vara` are identical. No variable can be called i (for obvious reasons).

At each input validation, you must display the value stored in the variable in a clear and consistent format. Expressions must be simplified when possible (e.g., $(x^2 + 3x + 2)/(x + 1)$ should be simplified to x+2), and numerical results should avoid redundant representations (e.g., 2×5 should be displayed as 10).

Let us look at the particular cases. All the following syntaxes are valid

V.4.1 Rational or imaginary

```
$./computorv2
> varA = 2
  2
> varB= 2 * (4 + varA + 3)
  18
> varC =2 * varB
  36
> varD    =     2 *(2 + 4 *varC -4 /3)
  289.333333333
>
```

However, $2 * xx$ does not mean $2 * x^2$ nor $2 * x$, xx here is considered as a variable.

V.4.2 Matrices

```
$./computorv2
> matA = [[1,2];[3,2];[3,4]]
  [ 1 , 2 ]
  [ 3 , 2 ]
  [ 3 , 4 ]
> matB=   [[1,2]]
  [ 1 , 2 ]
>
```

The matrix syntax is of the form $[[A_{0,0}, A_{0,1}, ...]; [A_{1,0}, A_{1,1}, ...]; ...]$
The semicolon is used to separate the rows of a matrix, so it is not present in the assignment of a matrix that has only one row. On the other hand, the comma is used to separate the columns of a matrix, which on the other hand will not be present in the assignment of a matrix that has only one column.

V.4.3 Functions

```
$./computorv2
> funA(b) = 2*b+b
  2 * b + b
> funB(a)   =2 * a
  2 * a
> funC(y) =2* y + 4 -2 * 4+1/3
  2 * y + 4 - 8 + 0.333333...
> funD(x)    =     2 *x
  2 * x
>
```

The syntax for functions is of the form: functionName(variable) = ...

Chapter VI

Bonus part

You are free to add much more advanced features to your program, such as:

- Function curve display
- Added usual functions (exponential, square root, absolute value, cosine, sinus, tangent, etc.)
- Radian computation for angles
- Function Composition

```
$./computorv2
> funA(x) = 2*x+1
  2 * x + 1
> funB(x) = 2 * x+1
  2 * x + 1
> funA(funB(x)) = ?
  4 * x + 3
>
```

- Norm computation
- Display of the list of stored variables and their values
- History of commands with results
- Matrix inversion
- An extension of the matrix computation applied to the vector computation
- What you feel is necessary and useful in this project

⚠ The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.