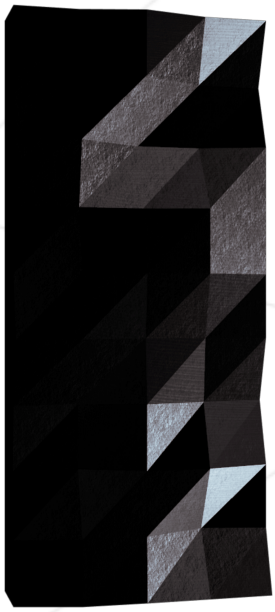

Computor v2

"Ton bc fait maison"

Dylan Djian ddjian@student.42.fr
42 Staff pedago@staff.42.fr

Résumé: Ce projet est le second d'une série qui vous permettra de renouer avec les maths, qui vous seront très utiles -voire nécessaires- pour de nombreux autres projets.

Table des matières

Chapitre I

Préambule

Le trébuchet fait partie des pièces d'artillerie médiévales dites à contrepoids. Il s'agit d'un engin de siège qui a été utilisé au Moyen Âge, soit pour détruire la maçonnerie des murs, soit pour lancer des projectiles par-dessus les fortifications. Il est parfois appelé « trébuchet à contrepoids » afin de le différencier d'une arme plus ancienne qu'on appelait « trébuchet à traction », une version primitive de l'engin où la force de propulsion était fournie par des hommes et non par un contrepoids. Le trébuchet à contrepoids est apparu dans la première partie du XIIe siècle dans les pays du pourtour méditerranéen à la fois dans les terres chrétiennes et dans les régions contrôlées par les musulmans. Il pouvait lancer des projectiles de trois cents livres (140 kg) et les projeter à grande vitesse contre les fortifications ennemies. Dans certaines circonstances, des cadavres infectés par différentes maladies ont été catapultés dans les villes dans le but de propager des épidémies parmi les assiégés, il s'agit d'une variante médiévale de la guerre biologique. Le trébuchet à contrepoids est apparu en Chine vers le IVe siècle av. J.-C. en Europe au VIe siècle de notre ère et il ne deviendra obsolète qu'au XVIe siècle, bien après l'introduction de la poudre à canon. Le trébuchet est beaucoup plus précis que les autres catapultes du Moyen Âge. Au cours des croisades, Richard Cœur de Lion a donné des noms évocateurs et pittoresques aux deux trébuchets utilisés au cours du siège de Saint-Jean-d'Acre en 1191 « la catapulte de Dieu » et « mauvais voisin ». Les techniques de construction des trébuchets ont été perdues au début du XVIe siècle.

Chapitre II

Introduction

Après vous être familiarisé avec les polynômes dans le premier volet du Computor, vous allez maintenant élargir le spectre de vos connaissances mathématiques aux travers d'un interpréteur, dans le langage de votre choix, un peu à la manière de la commande bc. Le projet comporte deux grosses parties, la première étant l'assignation (c'est-à-dire pouvoir sauvegarder des variables dans le contexte du programme) et la seconde est la résolution.

Le projet sera en quelque sorte la réunion du meilleur des deux mondes entre la calculatrice Google et la commande bc, pour en faire un interpréteur à la fois puissant et simple d'utilisation.

Chapitre III

Objectifs

Ce projet a pour but de vous faire découvrir ou redécouvrir certains aspects des mathématiques :

- Les complexes
- Les réels
- Les matrices
- Les fonctions numériques

Vous allez aussi voir les priorités opératoires ainsi que les propriétés calculatoires de ces différents élements.
L'objectif de ce projet est de vous apporter des bases sur la compréhension de ces outils dans la réalisation de vos projets 42 et hors 42 qui pourraient nécessiter des maths.

Chapitre IV

Consignes générales

- Ce projet ne sera corrigé que par des humains. Vous êtes donc libres d'organiser et nommer vos fichiers comme vous le désirez, en respectant néanmoins les contraintes listées ici.
- Le choix du langage est vôtre, vous n'avez aucune restriction dessus.
- Interdiction pour vous d'utiliser un type de variable natif au langage choisi pour la gestion des complexes.
- Aucune bibliothèque/librairie facilitant la gestion des complexes/matrices n'est autorisée, sauf celle que vous implémentez vous-mêmes.
- Dans le cas d'un langage compilé, vous devrez rendre un `Makefile`. Ce `Makefile` doit compiler le projet, et doit contenir les règles habituelles de compilation présentes dans la Norme C. Il ne doit recompiler et relinker le programme qu'en cas de nécessité.
- Vous êtes responsable de la bonne installation de l'environnement de travail du dump sur lequel vous allez developper ainsi que sur lequel vous serez corrigé.
- La norme n'est pas appliquée sur ce projet. Toutefois, nous vous demandons d'être clair et verbeux sur la conception de vos codes sources.
- Vous devrez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Vous avez l'autorisation d'utiliser d'autres fonctions dans le cadre de vos bonus, à condition que leur utilisation soit dûment justifiée lors de votre correction.

Chapitre V

Partie obligatoire

Avant de vous lancer corps et âme dans ce sujet, nous vous conseillons de jeter un oeil sur les ensembles, ce qu'ils contiennent, les contraintes qu'ils respectent etc. Au moins pour comprendre la notation mathématique utilisée dans ce sujet.

Pour le coup, tout ce dont vous avez besoin est accessible sur Wikipedia.

V.1 Présentation générale

`computor-v2` est un interpréteur d'instructions qui, à la manière d'un shell, va récupérer des inputs utilisateurs pour faire des calculs avancés.

Il présentera la forme d'un interpréteur de commandes simple, et devra pouvoir répondre au cahier des charges suivants (features détaillées dans les pages suivantes) :

- Prise en charge des types mathématiques suivants :
  - Nombres rationnels
  - Nombres complexes (à coéfficients rationnels)
  - Matrices
  - Equations polynômiales de degrés inférieur ou égal à 2
- Assignation d'une expression à une variable par inférence de type
- Réassignation d'une variable existante avec une expression d'un autre type
- Assignation d'une variable à une autre variable (existante ou non)
- Résolution d'une expression mathématique avec ou sans variable(s) définies
- Résolution d'une équation de degré inférieur ou égal à 2
- Opérations entre types, dans le domaine du possible
- Sortie du programme propre (par mot-clé, par signal, par raccourci clavier...)

V.2 Partie assignation

Votre `computor-v2` doit être capable d'assigner à des variables des types empreintés aux mathématiques, que vous ne retrouverez pas forcément dans un langage de programmation classique. Vous devrez OBLIGATOIREMENT créer ces types de variables et les incorporer de façon transparente dans votre programme.

Ainsi, vous devrez créer un type pour :

- Les nombres rationels (soit tout $x \in \mathbb{Q}$)

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

- Les nombres imaginaires soit tout $x = a + ib$ tels que $(a, b) \in \mathbb{Q}^2$

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

- Les matrices soit tout $A \in \mathbb{M}_{n,p}(\mathbb{Q})$

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

- Les fonctions (uniquement à une 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
>
```

Votre programme doit être capable de réassigner une variable et changer le type de variable par inférence, de telle sorte que :

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

Il devra aussi pouvoir assigner le résultat d'un calcul à une variable, de telle sorte que :

```
$./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 Partie calculatoire

- On peut utiliser les 4 opérations classiques, à savoir : $*$ / $+$ $-$.
  Le programme doit aussi gérer les modulos avec l'opérateur % ainsi que la multiplication matricielle qui se notera $**$. La multiplication terme à terme de deux matrices ou d'un scalaire et d'une matrice se note avec un $*$.
- Il devra aussi gérer les calculs de puissances entières et positives (ou nulles) avec l'opérateur ^
- Le programme doit gérer les parenthèses ainsi que les priorités de calculs.
- Mettre une fonction/variable puis =? doit permettre de retourner la valeur de cette variable dans le contexte du programme
- La résolution d'un calcul est symbolisée par l'opérateur ? à la fin de l'input

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

- Il devra gérer le calcul d'image :

```
$./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
>
```

- Il devra aussi gérer le calcul de racine de pôlynome quand le degré est inférieur ou égal à 2, en proposant sur $\mathbb{R}$ ou $\mathbb{C}$. Bonne nouvelle, vous l'avez déjà fait en partie sur `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 Syntaxe

Les noms de variables/fonctions doivent uniquement comporter des lettres et doivent être insensibles à la casse de telle sorte que `varA` et `vara` soient identiques. Aucune variable ne peut s'appeller i (pour des raisons évidentes).

A chaque validation d'input, vous devez afficher la valeur stockée dans la variable. Libre à vous de la formater comme il vous semble, tant que cela reste cohérent.

Voyons les cas particuliers. Toutes les syntaxes qui suivent sont valides.

V.4.1 Rationnels ou imaginaires

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

Cependant, $2 * xx$ ne veut pas dire $2 * x^2$ ni $2 * x$, ici xx est considérée comme une variable.

V.4.2 Matrices

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

La syntaxe matricielle est de la forme $[[A_{0,0}, A_{0,1}, ...]; [A_{1,0}, A_{1,1}, ...]; ...]$
Le point-virgule sert à séparer les lignes de la matrice, il n'est donc pas présent dans l'assignation d'une matrice qui ne comporte qu'une seule ligne. En revanche, la virgule sert à séparer les colonnes de la matrice, qui elle par contre ne sera pas présent dans l'assignation d'une matrice qui ne comporte qu'une seule colonne.

V.4.3 Fonctions

```
$./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
>
```

La syntaxe pour les fonctions est de la forme : nomDeLaFonction(variable) = ...

Chapitre VI

Partie bonus

Évidemment, la partie bonus ne sera évaluée que dans le cas où les parties obligatoires ont été parfaitement réalisées. Par parfaitement, nous entendons que TOUS les points cités en partie obligatoire sont développés de manière parfaite et sans aucun comportement indéfini.

Vous êtes libres d'ajouter à votre programme des fonctionnalités beaucoup plus avancées tels que :

- Affichage de courbe de fonctions
- Ajout de fonctions usuelles (exponentielle, racine carrée, valeur absolue, cosinus, sinus, tangente, etc.)
- Ajout du calcul en radian pour les angles
- Composition de fonction

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

- Calcul de norme
- Afficher la liste des variables stockées ainsi que leurs valeurs
- Historique des commandes avec résultats
- Inversion de matrice
- Une extension du calcul matriciel appliqué au calcul vectoriel
- Ce qui vous semble nécessaire et utile dans le cadre de ce projet

Chapitre VII

Rendu et peer-évaluation

Rendez votre travail sur votre dépot `git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance.