HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Professeur : Nastaran Fatemi
Assistant : Jonathan Bischof

Labo 1 : Calculatrice (partie 1)

Objectifs

L'objectif de ce laboratoire est de développer les fonctions nécessaires à l'utilisation d'une calculatrice directement dans une `Worksheet`. Ceci permettra d'importer les fonctions définies lors de ce laboratoire directement dans le deuxième laboratoire, qui sera un programme à part entière.

Indications

Créez un nouveau projet nommé `calculatrice` et ainsi qu'une `Worksheet` nommée `partie1`.

- Ce laboratoire est à effectuer par groupe de `2` (ou `3`), tout plagiat sera sanctionné par la note de `1`.
- Le laboratoire complet (partie 1 et 2) est à rendre pour le `20.03.2017` à `23h55` sur Cyberlearn, une archive nommée `SCALA_labo1_NOM1_NOM2.zip (7z, rar, ...)` contenant les sources de votre projet devra y être déposé.
- Il n'est pas nécessaire de rendre un rapport, un code propre et correctement commenté suffit. Faites cependant attention à bien expliquer votre implémentation.
- Faites en sorte d'éviter la duplication de code.
- Préférez des implémentations récursives de fonctions.
- Préférez l'utilisation de `val`, sauf pour la mémoire, ou il est parfois nécessaire d'utiliser des `var`.

Réalisation

Vous allez développer les fonctions d'utilisation d'une calculatrice, telles que les opérations binaires `+`, `-`, `*`, `/`, `%` (modulo) et `^` (puissance), l'opération unaire factoriel `!`, l'utilisation d'une mémoire pour des variables, la résolution d'équations du 2^e degré ainsi que le plus grand diviseur commun (GCD).

- `3 + 4`
- `15 / 5`
- `12!`
- `a = 5`
- `a * 3`
- `solve(ax2 - 4x + 2)`
- `gcd(a, 167)`
- ...

Dans ce laboratoire, l'implémentation de la mémoire n'a pas encore réellement de sens, étant donné que vous allez travailler dans une `Worksheet`, mais elle sera mise en oeuvre lors du deuxième laboratoire. Vous pouvez donc déjà penser à une implémentation, et aux fonctions pour mémoriser et récupérer la valeur de la mémoire.

Entrées utilisateurs

Nous partons du principe que toutes les entrées utilisateurs sont valides. Il ne vous est pas demandé de vérifier que chaque entrée utilisateur contienne les bonnes valeurs.

Opération unaire et binaire

La liste de ces opérations est volontairement exhaustive, afin de ne pas "trop" compliquer la tâche. Il n'est pas demandé de chaîner les opérations, car ceci demande des connaissances plus approfondies sur la priorité des opérateurs et sur le *pattern matching*. Ces notions seront vues dans le deuxième laboratoire.

Il s'agit ici de simplement interpréter des calculs simples, comme `3 + 5` ou `5^2`. L'opération unaire `!` ne prend qu'un seul argument à gauche de l'opérateur.

Créez une fonction (par exemple *def op(...)*) qui prend en paramètre l'opérateur (`char`) ainsi qu'un ou deux nombres (`Int` ou `Double`) et qui retourne le résultat de l'opération.,

Utilisation de la mémoire

Il doit être possible d'affecter des valeurs à des variables à l'aide de l'opérateur `=`. Lorsqu'une variable a été affectée, il doit être possible de l'utiliser dans n'importe quelle opération, par exemple :

- `a = 5`
- `b = 6`
- `a * b` ⇒ `5 * 6` ⇒ `30`
- `a = b` ⇒ `a = 6`
- ...

Vous pouvez utiliser une `Map` pour gérer la mémoire de la calculatrice.

```
var memory: Map[String, Double] = Map()
```

Nous considérons actuellement qu'une variable est valide si, et seulement si, elle n'est constituée que de lettres (`[a-zA-Z]+`).

Plus Grand Diviseur Commun

Définissez une fonction qui calcule le Plus Grand Diviseur Commun (GCD) de deux nombres. L'implémentation de l'algorithme a été vue pendant le cours.

```
gcd(167, 5) ⇒ 1
```

Equations du 2^e degrés

Définissez une fonction qui résoud une équation du 2^e degrés.

```
solve(x2 + 3x + 4)
```

Il y a plusieurs cas possibles pour `ax2 + bx + c` :

- Si `a = 0`, ce n'est pas une équation du 2^e degrés.
- Si `a = 1`, il peut être écrit soit `x2` soit `1x2`.
- Si `b = 0`, il n'y a pas de `x` dans l'équation, mais elle reste cependant valable.
- Si `c = 0`, il n'y a pas de troisième terme dans l'équation, mais elle reste cependant valable.

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Professeur : Nastaran Fatemi
Assistant : Jonathan Bischof

Rappel

$\Delta = b^2 - 4ac$

Si $\Delta > 0 \Rightarrow x = \frac{-b \pm \sqrt{\Delta}}{2a}$

Si $\Delta = 0 \Rightarrow x = \frac{-b}{2a}$

Si $\Delta < 0 \Rightarrow x = (real : \frac{-b}{2a}, imaginary : \frac{\pm\sqrt{-\Delta}}{2a})$

Dans le cas $\Delta < 0$, nous nous trouvons dans les nombres complexes, dont la partie réelle et la partie imaginaires des valeurs de `x` se calculent avec les éléments déjà connus.

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Professeur : Nastaran Fatemi
Assistant : Jonathan Bischof

Labo 1 : Calculatrice (partie 2)

Racine carrée

Définissez une fonction qui calcule la racine carrée d'un nombre. Avec $\varepsilon = 0.0001$:

```
sqrt(9) = 3.00009155
```

Vous pouvez utiliser une implémentation *tail recursive* pour les itérations.
Première itération pour `n = 9`, on choisit `x = 1`, si `x` est suffisamment bon, on le retourne, car c'est la racine carrée de `n`, sinon, on augmente `x` et on passe à la deuxième itération.
Pour savoir si `x` est suffisamment bon, on vérifie que $\frac{|x^2 - n|}{n} < \varepsilon$. Si c'est vrai, alors `x` est suffisamment proche de la racine carrée de `n`, sinon, on recommence en augmentant `x`.
Pour augmenter `x`, sa nouvelle valeur doit être $\frac{x + \frac{n}{x}}{2}$. Si `x = 1` à la première itération, alors `x = 5` à la deuxième itération, `x = 3.4` à la troisième itération, etc.

Nombre premier

Définissez une fonction qui calcule si un nombre est premier.

```
prime(5) ⇒ 5 is prime !
prime(6) ⇒ Not a prime number
```

Un nombre est premier, s'il n'est divisible que par `1` et par lui-même. Donc si un nombre `n` n'est divisible par aucun autre nombre que `1`, pour tous les nombres plus petit que $\sqrt{n}$, alors `n` est premier.

Algorithme d'Euclide étendu

Définissez une fonction qui calcule l'algorithme d'Euclide étendu.

```
egcd(u, v) ⇒ u*x + v*y = z
egcd(5, 167) ⇒ 5*67 + 167*-2 = 1
```

Les explications de l'algorithme sur la page Wikipedia sont très bien construites. On y trouve également des pseudo-codes pour l'implémentation.

Inverse modulaire

Définissez une fonction qui calcule l'inverse modulaire.

```
ModInvert(3, 11) = 4
```

Selon l'algorithme d'Euclide étendu :

```
egcd(u, v) ⇒ u*x + v*y = z
```

Si `z != 1`, alors il n'y a pas d'inverse modulaire. Sinon `ModInvert(u, v) = x`.