Programmation applicative – L2
Deuxième série de TP : Le compte est bon

O. Sans {Olivier.Sans@umontpellier.fr}
I. Zappatore {Ilaria.Zappatore@umontpellier.fr}
D. Catta {Davide.Catta@umontpellier.fr}

1 Préambule

Cet enoncé concerne 3 séances de TP. L'évaluation des TPs se fera en deux phases :

— Pendant les TPs, votre présence et votre progression seront notées. À la fin de chaque séance de TP, vous indiquerez sur la feuille de présence le numéro de la dernière question à laquelle vous avez entièrement répondu.

— Pendant la dernière séance, une évaluation de votre travail tout au long des séances sera effectuée, sous la forme d'un petit problème auquel il faudra répondre en vous servant des fonctions que vous aurez écrites durant les TPs.

En conséquence, il faudra sauvegarder précieusement votre travail au fur et à mesure des séances, pour pouvoir l'utiliser lors de l'examen final.

2 Description du jeu

Le problème est le suivant : nous disposons d'un ensemble d'entiers naturels (les valeurs de départ) et d'un entier cible n, et nous voulons combiner les valeurs de départ au moyen des quatre opérations $+, -, /, \times$ de manière à obtenir l'entier p le plus proche possible de n, sachant que nous pouvons utiliser au plus une fois chacune des valeurs de départ. Nous sommes intéressés non seulement par la valeur de p mais aussi par la formule qui nous a permis de l'obtenir.

Si vous n'avez jamais vu la version télévisée du jeu, voici le lien vers une vidéo montrant une partie : `http://www.youtube.com/watch?v=h9vH7YGotQc`

3 Structures de données

3.1 Données de départ

Nous travaillerons à partir des données suivantes : une liste d'opérateurs, et une liste de valeurs de départ. Par exemple, on peut définir :

```
; Définitions des objets de base
(define LVal '(1 2 3 4 5 6 7 8 9 10 25 50 75 100))
(define Op '(+ * - /))
```

3.2 Définition d'un tirage

Un tirage se compose d'une part d'un nombre pris aléatoirement entre 100 et 999, et d'une série de 6 plaques prises au hasard parmi les valeurs de départ. Ainsi, plusieurs plaques parmi les 6 peuvent prendre les mêmes valeurs.

Exemple de tirage :

Exercice 1 *Écrire deux fonctions* `make-cible` *et* `make-tirage` *qui réalisent respectivement le tirage du nombre à atteindre (entre 100 et 999), et le tirage aléatoire de 6 valeurs parmi les plaques disponibles. On utilisera la fonction de base du langage* `random`*. À noter que* `(random n)` *renvoie une valeur entière aléatoire entre 0 et* $n - 1$*.*

Exemple :

```
> (make-cible)
833
> (make-tirage)
(4 10 25 10 5 25)
```

Exercice 2 *Écrire une fonction* `estDans?` *qui prend en paramètres un entier et une liste d'entiers, et renvoie vrai si l'entier est dans la liste, faux sinon. Cette fonction nous servira à tester si le nombre cible est dans un tirage ou une liste de nombres issus d'un tirage.*

Exemple :

```
> (estDans? 4 '(1 2 3 4 5 6))
#t
> (estDans? 2 '(10 20 30))
#f
```

3.3 Opérations autorisées

Afin de déterminer si le compte est bon, l'idée de base, sans optimisation, est d'essayer toutes les possibilités de combinaisons deux à deux des plaques du tirage. Mais certaines opérations ne sont pas permises, car il faut que les valeurs restent des valeurs entières et strictement positives à chaque étape.

Exercice 3 *Écrire une fonction* `estValide?` *qui prend en paramètres un opérateur et deux valeurs entières, et renvoie vrai si l'opération renvoie un résultat entier strictement positif. On aura besoin d'évaluer l'opérateur afin de réaliser le calcul, cela se fait avec la fonction* `eval`*.*

Exemple :

```
> (eval +)
#<procedure:+>
> ((eval +) 4 5)
9
> (estValide? - 2 14)
#f
> (estValide? / 3 0)
#f
> (estValide? / 1 3)
#f
> (estValide? + 2 6)
#t
```

4 Moteur de jeu, première version

Dans cette première version, on va répondre au problème simplifié suivant : "est-il possible (oui/non) de trouver une combinaison des plaques de façon à obtenir exactement le nombre cible ?".

Exercice 4 *Écrire une fonction* `opere` *qui prend en paramètre une liste d'opérateurs et deux entiers, et renvoie une liste de valeurs obtenues en appliquant les opérations valides sur ces opérateurs. Le résultat doit être symétrique, c'est à dire que pour les opérateurs non symétriques tels que* `-` *et* `/`*, si l'opération dans un sens n'est pas valide, on doit également tester si l'opération dans l'autre sens est valide. Ce n'est pas la peine pour les opérateurs* `+` *et* `*` *qui sont commutatifs.*

Exemple :

```
> Op
(+ * - /)
> (opere Op 2 10)
(12 20 8 5)
```

Exercice 5 *Écrire une fonction* `genere_plaques` *qui prend en paramètre une liste de valeurs entières, et produit une liste de listes repésentant les plaques que l'on peut générer en choisissant deux valeurs parmi la liste initiale et en appliquant les opérations valides. Lorsque l'on choisit deux éléments, les autres plaques ne sont pas modifiées.*

Exemple :

```
> Op
(+ * - /)
> (genere_plaque Op '(10 3 7))
((13 7) (30 7) (7 7) (3 17) (3 70) (3 3) (10 10) (10 21) (10 4))
```

Remarque : Cette dernière fonction peut nécessiter la définition de plusieurs fonctions auxilliaires intermédiaires, c'est le nœud du problème, on prendra donc le temps de réflêchir sur papier !

Exercice 6 *Écrire une fonction* `ceb` *qui prend en argument une liste d'opérateurs, une liste de plaques et un entier cible, et qui affiche "le compte est bon" si on peut obtenir la cible exactement à partir des plaques et des opérateurs, et qui affiche "le compte n'est pas bon" sinon.*

Exemple :

```
> Op
(+ * - /)
> (define jeu (make-tirage))
> jeu
(4 50 9 2 6 3)
> (ceb Op jeu 2641)
le compte n'est pas bon
> (ceb Op jeu 418)
le compte est bon
```

5 Version approchée

On veut maintenant, si jamais le compte exact n'est pas bon, proposer le nombre qui s'en approche le plus. Pour cela, au lieu de tester si un element entier est dans une liste d'entiers, on va récupérer le nombre qui s'approche le plus du nombre cible.

Exercice 7 *Écrire une fonction* `approche` *qui prend en paramètre un entier et une liste, et renvoie l'élément de la liste le plus proche de cet entier.*

Exemple :

```
> (approche '(1 4 7 9) 3)
4
> (approche '(1 4 7 9) 7)
7
> (approche '(1 4 7 9) 45)
9
> (approche '(1 4 7 9) 8)
7
```

Exercice 8 *Modifier la fonction* `ceb` *pour afficher, si le compte n'est pas bon, la meilleure approximation possible.*

6 Liste des opérations effectuées

Dans la version originale du jeu, il est nécessaire de conserver une trace des calculs effectués jusqu'à l'obtention de la valeur conservée. Pour ce faire, on peut attacher à chaque valeur l'expression qui a permis de la calculer.

Exercice 9 *Modifier les structures de données, pour stocker non seulement les valeurs calculées, mais aussi les calculs qui ont permis de l'obtenir.*