Projet Informatique n°2 -Classe de Seconde (Michel LAGOUGE)
Programmation Scheme

On se propose d'écrire dans ce projet un ensemble de fonctions qui permettent de passer d'un nombre écrit dans une base b à ce même nombre écrit dans une autre base b' (b et b' étant quelconque mais <= 16).
Dans un premier temps, on se limitera aux conversions de nombres entiers puis dans un deuxième partie du projet on étendra à des nombres réels pour aborder le problème de la représentation des nombres sur ordinateur et les problèmes de calcul dus à cette représentation.

Quelques informations sur un nombre écrit en base b

1) **définition de l'écriture d'un nombre en base b**
On se limite dans cette partie aux nombres entiers. Les nombres rééls (avec décimales) seront traités plus tard.
Soit un nombre écrit en base b = 10 (la base habituelle dans laquelle nous travaillons)
Exemple : x = 1515 est une écriture économe qui signifie que le nombre x = $\underline{1} \times 10^3 + \underline{5} \times 10^2 + \underline{1} \times 10^1 + \underline{5} \times 10^0$
De façon générale, un nombre (entier) X écrit dans une base b

a) requiert b symboles pour être représenté
par exemple : les symboles {0,1,2,3,4,5,6,7,8.9} en base 10
les symboles {0.1 } en base 2
les symboles {0,1,2,3,4,5,6,7 } en base 8
les symboles {0,1,2,3,4,5.6,7,8,9, A, B,C,D.E,F} en base 16

b) est écrit de droite à gauche
par exemple millier ← centaine ← dizaine ← unité en base 10 **c)**

a une représentation symbolique $X = x_5\, x_4\, x_3\, x_2\, x_1 x_0$
qui a pour signification que $X = x_5 \times b^5 + x_4 \times b^4 + x_3 \times b^3 + x_2 \times b^2 + x_1 \times b^1 + x_0 \times b^0$
(une règle d'écriture d'un nombre est que l'on n'écrit pas les "zéros à gauche")

2) Passage de la base b quelconque à la base 10
La conversion est évidente compte tenu de la définition de l'écriture d'un nombre en base b quelconque.

Quelques exemples
Soit le nombre X = 10101010 écrit en base 2
D'après la définition ci-dessus: $X = \underline{1} \times 2^7 + \underline{0} \times 2^6 + \underline{1} \times 2^5 + \underline{0} \times 2^4 + \underline{1} \times 2^3 + \underline{0} \times 2^2 + \underline{1} \times 2^1 + \underline{0} \times 2^0$
$= \underline{1} \times 128 + \underline{0} \times 64 + \underline{1} \times 32 + \underline{0} \times 16 + \underline{1} \times 8 + \underline{0} \times 4 + \underline{1} \times 2 + \underline{0} \times 1$
$= 128 + 32 + 8 + 2$
$= 170$

Soit le nombre X = BABA écrit en base 16
D'après la définition ci-dessus : $X = \underline{B} \times 16^3 + \underline{A} \times 16^2 + \underline{B} \times 16^1 + \underline{A} \times 16^0$
$= \underline{11} \times 4096 + 10 \times 256 + 11 \times 16 + 10 \times 16$
$X = 47802$

On remarquera que plus la base est élevée, plus l'écriture d'un nombre donné est courte.

Ainsi le nombre $X = (1010\ 1010)_2 = (170)_{10}$,,, s'écrira (AA) en base 16
Et on notera un moyen simple de passer de la base 2 à la base 16 :

- regrouper les bits (binary digit = chiffre binaire) par paquet de 4 à partir de la droite
- et associer le symbole de la base 16 associé.

Exemple $X = \underline{(1010}\ 1010)_2$ en notant que $(1010)_2 = (A)_{16}$
D'où $X = (A\ A)_{16}$

3) Passage de la base 10 à une base b quelconque
Observons tout d'abord comment s'obtiennent les différents chiffres d'un nombre écrit en base 10
Exemple $X = (1515)_{10}$ on opère par divisions successives par 10

opérande	diviseur	quotient	reste
1515	10	151	5
151	10	15	1
15	10	1	5
1	10	0	1

On voit que les différents chiffres correspondent **au reste** des divisions successives du nombre par la base.

Transposons par exemple à la b ase 16

opérande	diviseur	quotient	reste
1515	16	94	11 => B
94	16	5	14 => E
5	16	0	5
1	10	0	1

D'où X = $(1515)_{10} = (5EB)_{16} = (101\ \ 1110\ \ 1011)_2$ (en « éclatant » chaque chiffre de la base 16 vers la base 2 –page précédente-)

Quelques outils SCHEME pour le projet

1) les opérations sur les nombres entiers

On aura besoin d'obtenir le quotient et le reste de la division entière de deux nombres

a) le quotient

```
> (quotient 26 6)
4
```

b) le reste

```
> (remainder 26 6)
2
```

2) les A – listes ou listes d'associations

L'aide en ligne de DrScheme donne les informations suivantes :

Alist (for ``association list'') must be a list of pairs. These procedures find the first pair in alist whose car field is obj, and returns that pair. If no pair in alist has obj as its car, then #f (not the empty list) is returned. Assq uses eq? to compare obj with the car fields of the pairs in alist, while assv uses eqv? and assoc uses equal?.

```
(define e '((a 1) (b 2) (c 3)))          ; e est une A-liste gérant des associations (a⇔ 1, b⇔2, etc.)
(assq 'a e)              =>  (a 1)       ; retourne dans e l'association correspondant à a
(assq 'b e)              =>  (b 2)       ; retourne dans e l'association correspondant à b
(assq 'd e)              =>  #f          ; retourne #f car il n'y a aucune association avec d dans e
(assq (list 'a) '(((a)) ((b)) ((c))))
                         =>  #f
(assoc (list 'a) '(((a)) ((b)) ((c))))
                                    =>  ((a))
(assq 5 '((2 3) (5 7) (11 13)))
                                    =>  unspecified
(assv 5 '((2 3) (5 7) (11 13)))
                                    ⇨   (5 7)
```

Cette structure de données particulières qu'est la A-liste est particulièrement adaptée pour gérer les associations :
A ⇔ 10 B ⇔ 11 etc. pour les bases de 11 à 16 par exemple.

3) Fonction de Trace : fonction à utiliser pendant la mise au point des fonctions (en mode langage : « Essentials of Programming Languages (2nd ed.) »)

```
(require (lib "trace.ss"))                    ; à mettre au début de la fenêtre de définitions
```

…ensuite dans le fenêtre d'évaluation, écrire :

```
(trace  nom_de fonction_1  nom_de fonction_2…etc)
```

A chaque appel des fonctions nom_de fonction_1 nom_de fonction_2 etc, les fonctions correspondantes seront « tracées »

```
(detrace  nom_de fonction_1  nom_de fonction_2…etc)    ;  pour arrêter la trace
```

4) Fonction error (en mode langage : « Essentials of Programming Languages (2nd ed.) »)

```
(define (error message . args)
; permet d'afficher un message d'erreur et d'arrêter une évaluation
      (display "Error: ")
      (display message)        ; mettre entre guillemets le message à envoyer
      (for-each (lambda (arg)
                  (display " ")
                (write arg))
                args)
  (newline))
```

Première partie du projet sur la numération

Cahier des charges pour l'écriture d'un nombre en base b :
Un nombre écrit en base b sera représenté en Scheme sous la forme d'une liste dont :

- le CAR sera la base b
- le CDR sera une liste dont les éléments représentent l'écriture du nombre

Exemples : $(1515)_{10}$ sera représenté par (10 (1 5 1 5))
$(BABA)_{16}$ sera représenté par (16 (B A B A))
$(1010\ 1010)_2$ sera représenté par (2 (1 0 1 0 1 0 1 0))

♠ ***Pour les différentes fonctions à écrire définies plus bas, s'efforcer d'écrire une fonction qui donne le résultat attendu, peu importe que la fonction soit récursive terminale ou non terminale.***
Une fois la fonction écrite, si elle est non terminale, on pourra ensuite chercher à écrire une fonction optimale, à savoir une fonction récursive terminale.

I Les fonctions de conversions

1. Ecrire la fonction `convert_ecriture`

Qui prend en entrée un nombre « normal » et le transforme dans la structure de donnée voulue

```
> (convert_ecriture 1515)                    =>          (10 ( 1     5     1     5))
```

2. Ecrire la fonction `convert_inverse`

Qui prend en entrée un nombre écrit dans la structure de donnée voulue et retourne le nombre « normal »

```
> (convert_inverse '(10 ( 1   5     1     5))) =>     1515)
```

3. Ecrire la fonction `conv10tob`

Qui prend en entrée un nombre « normal » et la base b (**b < 10**) et retourne son écriture dans la structure de donnée voulue
; attention : on se limite ici à b inférieur à 10 pour ne pas avoir de conversion de symbole

```
> (conv10tob 1515 2)                         =>(2 (1  0  1  1  1  1  0  1  0  1  1))
> (conv10tob 1515 8)                         =>(8 (2     7     5     3))
```

4. Ecrire la fonction `convbto10`

Qui prend en entrée un nombre écrit dans la structure de donnée voulue pour la base b <10 et retourne le nombre « normal »
; attention : on se limite ici à b inférieur à 10 pour ne pas avoir de conversion de symbole

```
> (convbto10 '(2 (1  0  1  1  1  1  0  1  0  1  1)))       =>    1515
> (convbto10  '(8 (2    7     5     3))  )                  =>    1515
```

5. Ecrire la version améliorée `conv10tob_gene` de `conv10tob` qui fonctionne jusque b = 16

6. Ecrire la version améliorée `convbto10_gene` de `convbto10` qui fonctionne jusque b = 16

Pour les fonctions `conv10tob_gene` et `convbto10_gene,` définir localement une A-liste permetant de faire les conversions A ⇔ 10 B ⇔ 11 etc jusque F ⇔ 15 par :

```
(let ((alphabet_10tob '((10 A) (11 B)(12  C)(13  D) (14  E) (15  F)))
      ….)
```

et

```
(let ((alphabet_bto10 '((A  10 A)(B  11)(C  12)(D  13) (E  14)(F  15)))
      ….)
```

Exemple d'utilisation de la fonction let et d'une A-liste :

```
(define (associe nb)
(let ((alphabet_10tob '((10 A) (11 B)(12  C)(13  D) (14  E) (15  F))))
            (cadr (assoc nb alphabet_10tob))))

> (associe 10)
A       ; l'évaluateur peut retourner la lettre en minuscule mais ceci est sans importance. Les versions les plus récentes de DrScheme différentient minuscule ⇔ majuscule.
> (associe 14)
E
```

Quand cela sera nécessaire, voir dans l'aide la fonction cousine de let à savoir let* qui permet de faire un let séquentiel.

II Les deux premières opérations sur des nombres en base b quelconque (b <=16)

Ecrire les deux premières opérations arithmétiques (addition et soustraction) sur des nombres écrits en base b quelconque (b <= 16) et retournant le résultat dans la structure de donnée définie par le cahier des charges.
Pour la soustraction, on admettra que le deuxième opérande est toujours inférieur au premier afin de ne gérer que des nombres entiers naturels (∈ N) donc toujours positifs.
Exemples :

```
> (add      '(16 (5 E B))                ;(1515)10
          '(16 (B A B A)))               ;(47802)10
(16 (C 0 A 5))                           ;(49317)10

> (sous     '(16 (B A B A))              ;(47802)10
          '(16 (5 E B)))                 ;(1515)10
(16 (B 4 C F))                           ;(46287)10
```

♠***Attention : il s'agit d'écrire les fonctions travaillant directement dans la base b SANS REPASSER PAR LA BASE 10. Par contre, on pourra toujours vérifier -ensuite !- les résultats des différentes fonctions écrites en repassant par la base 10***
Exemples pour vérifier

a) le résultat de l'addition

```
> (conv10tob_gene (+    (convbto10_gene '(16 (5 E B)))
                        (convbto10_gene '(16 (B A B A))))
                  16)
(16 (c 0 a 5))
```

b) le résultat de la soustraction

```
> (conv10tob_gene (-  (convbto10_gene '(16 (B A B A)))
                      (convbto10_gene '(16 (5 E B))))
                  16)
(16 (b 4 c f))
```

Il conviendra pour écrire les fonctions correspondantes aux deux opérations de bien réfléchir au algorithmes traditionnels de calcul de ces opéations dans la base 10 (bas habituelle de nos calculs) pour faire la transposition à une base b quelconque (voir le cours !)

Remarques pour aider…
Rem 1 : Il est conseillé AVANT d'écrire la fonction qui additionne deux nombres d'écrire la fonction qui additionne TROIS chiffres dans la base b et qui retourne le résultat de la somme et la retenue…. Soit par exemple :
(Réfléchir : pourquoi additionner TROIS chiffres ?)

```
> (add_chiffre 5  4  0  10)   ; ajoute 5 + 4 + 0 dans la base 10
(9 0)                         ; => résultat : 9 retenue : 0
> (add_chiffre 5  4  1  10)   ; ajoute 5 + 4 + 1 dans la base 10
(0 1)                         ; => résultat : 0 retenue : 1
> (add_chiffre 'a  'a  0  16) ; ajoute A + A + 0 dans la base 16
(4 1)                         ; => résultat : 4 retenue : 1
> (add_chiffre 4  4  1  8)    ; ajoute 4 + 4 + 1 dans la base 8
(1 1)                         ; => résultat : 1 retenue : 1
```

Rem 2 : Comment transformer une soustraction en addition ?
Soit la soustraction (en base 10) 1515 – 987 que l'on pose habituellement sous la forme (on écrit ici les « zéros à gauche » qui ne sont généralement pas écrits mais qui sont implicites) :

```
      1   5   1   5
  -   0   9   8   7
  ---------------------
      0   5   2   8
```

=> autre algorithme
on prend le complément à 9 de chaque chiffre du deuxième opérande
on fait la somme
on ajoute 1

```
      1   5   1   5
  +   9   0   1   2
  ---------------------
  1   0   5   2   7
  +                 1
  ---------------------
  1   0   5   2   8
```

Cette démarche est généralisable à toute base en prenant le complément à (b – 1) et en n'oubliant pas d'ajouter 1 au résultat : la démarche globale { le complément à (b – 1) et incrémentation du résultat} – du deuxième opérande de la soustraction !- s'appelle une complémentation à b.
Cette démarche est particulièrement intéressante pour la base b = 2 puisque le complément de 0 est 1 et celui de 1 est 0.
Les meilleurs, les plus rapides, etc… pourront essayer d'écrire les fonctions qui permettent de faire :
- la multiplication - la division dans une base b quelconque (b <= 16)