Remarques :
- *Aucun document ni machine électronique n'est permis à l'exception de la carte de référence de Scheme.*
- *Les questions peuvent être résolues de façon indépendante. Il est possible, voire même utile, pour répondre à une question, d'utiliser les fonctions qui sont l'objet des questions précédentes.*
- *La clarté des réponses et la présentation des programmes seront appréciées. Toutes les fonctions apparaissant dans les réponses doivent être accompagnées de leur spécification (sauf si elle est donnée dans la question).*
- *Le barême (total sur 50) n'est donné qu'à titre indicatif.*

Les nombres heureux

Voici un extrait d'un exercice proposé par Jacques Arsac, Jeux et casse-tête à programmer, Dunod 85 :

« L'adjudant réunit ses hommes pour décider qui serait de corvée de patates.

« Mettez-vous en file indienne et comptez-vous en partant de 2 ». Le premier homme de la file dit 2, le suivant dit 3, le suivant 4 et ainsi de suite.

« Le premier de la file, sortez du rang. Vous êtes dispensé de corvée. Quel est votre numéro ? »

« 2 » répondit le soldat.

« Les hommes de 2 en 2 en commençant à celui qui vient de sortir, sortez des rangs, vous êtes de corvée. »

Et le processus recommença. Le premier restant dans les rangs avait le numéro 3, et il était heureux : dispensé de corvée. Les hommes de 3 en 3 en commençant à lui sortirent des rangs pour la corvée...

Pour être sûr que vous avez bien compris, voici les premiers nombres heureux :

2 3 5 7 11 13 17 23 25 29

Les nombres heureux ne sont pas nécessairement premiers et les nombres premiers ne sont pas nécessairement heureux... »

Nous expliciterons ce texte par la suite.

Première partie : simulation du processus

Le but de la première partie de cet exercice est d'écrire une fonction qui simule le processus ci-dessus (la donnée du problème est le nombre de soldats).

Question 1 (4 points) :

Écrire la signature et une définition de la fonction, nommée `intervalle`, qui, étant donnés deux entiers n et m, rend la liste des entiers compris entre n et m (inclus) ; elle rend la liste vide lorsque $n > m$. Par exemple :

```
(intervalle 5 9)  → (5 6 7 8 9)
(intervalle 9 5)  → ()
```

Attention : la sémantique n'est pas demandée

```
;;; intervalle : int * int -> LISTE[int]
;;; (intervalle n m) rend la liste des entiers de n à m (rend la liste vide
;;; lorsque n > m)
(define (intervalle n m)
  (if (> n m)
      '()
      (cons n
            (intervalle (+ n 1) m))))
```

Question 2 (5 points) :

Écrire la signature et une définition de la fonction, nommée `moins-i-eme`, qui, étant donnés un entier positif i et une liste L, rend la liste obtenue en supprimant le $i^{ème}$ élément de la liste L, la numérotation des éléments de la liste se faisant à partir du numéro 1 ; la fonction doit rendre L lorsque i est supérieur à la longueur de L. Par exemple :

```
(moins-i-eme 2 '(1 2 3 4))  →  (1 3 4)
(moins-i-eme 2 '(10 21 34 43 56))  →  (10 34 43 56)
(moins-i-eme 5 '(1 2 3 4))  →  (1 2 3 4)
```

Remarque : cette fonction n'est pas utile pour la suite !

Attention : la sémantique n'est pas demandée

```
;;; moins-i-eme : nat/>0/ * LISTE[alpha] -> LISTE[alpha]
;;; (moins-i-eme i L) rend la liste obtenue en supprimant le «i»-ème élément
;;; de la liste «L» (elle rend la liste «L» lorsque «i» est supérieur à la
;;; longueur de la liste «L»).
(define (moins-i-eme i L)
  (if (pair? L)
      (if (= i 1)
          (cdr L)
          (cons (car L) (moins-i-eme (- i 1) (cdr L))))
      '()) )
```

Considérons la fonction `moins-j-mult-i-emes` qui a comme spécification :

*;;; moins-j-mult-i-emes : nat/>0/ * nat/>0/ * LISTE[alpha] -> LISTE[alpha]*
;;; (moins-j-mult-i-emes j i L) rend la liste obtenue en supprimant, de la
;;; liste L, ses «j»-ème, «j+i»-ème, «j+2i»-ème, «j+3i»-ème... éléments.

Par exemple :

```
(moins-j-mult-i-emes 1 2 '(3 4 5 6 7 8))  →  (4 6 8)
```

(on enlève les 1^{er}, $3^{ème}$ et $5^{ème}$ éléments de la liste donnée).

Question 3 (6 points) :

Écrire une définition de cette fonction.

```
(define (moins-j-mult-i-emes j i L)
  (if (pair? L)
      (if (= j 1)
          (moins-j-mult-i-emes i i (cdr L))
          (cons (car L) (moins-j-mult-i-emes (- j 1) i (cdr L))))
      '()))
```

Question 4 (4 points) :

En utilisant la fonction précédente, écrire la signature et une définition de la fonction `moins-mult-i-emes`, qui, étant donnés un entier i et une liste L, supprime les éléments de la liste de i en i. Par exemple :

`(moins-mult-i-emes 2 '(3 4 5 6 7 8 9 10 11))` $\rightarrow$ `(3 5 7 9 11)`

(on enlève les $2^{ème}$, $4^{ème}$, $6^{ème}$ et $8^{ème}$ éléments de la liste donnée).

Attention : la sémantique n'est pas demandée

```
;;; moins-mult-i-emes : nat/>0/ * LISTE[alpha] -> LISTE[alpha]
;;; (moins-mult-i-emes i L) rend la liste obtenue en supprimant, de la
;;; liste L, ses «i»-ème, «2i»-ème, «3i»-ème... éléments.
(define (moins-mult-i-emes i L)
  (moins-j-mult-i-emes i i L))
```

On peut maintenant définir la fonction `liste-dispenses` qui, à partir d'une liste d'entiers supérieurs ou égaux à 2, calcule la liste des éléments de cette liste qui sont dispensés de corvée en suivant le processus expliqué dans l'introduction. Par exemple

`(liste-dispenses '(3 5 7 9 11 13 15 17 19 21))` $\rightarrow$ `(3 5 7 11 13 17)`

En effet

a) `3` (le premier élément de la liste) est dispensé et on enlève de la liste privée de son premier élément, les éléments de 3 en 3 ; la nouvelle liste est donc : `(5 7 11 13 17 19)` ;

b) `5` (le premier élément de la nouvelle liste) est dispensé et on enlève de la nouvelle liste privée de son premier élément, les éléments de 5 en 5 ; la nouvelle liste est donc : `(7 11 13 17)` ;

c) `7` (le premier élément de la nouvelle liste) est dispensé et on enlève de la nouvelle liste privée de son premier élément, les éléments de 7 en 7 (il n'y en a pas) ; la nouvelle liste est donc : `(11 13 17)` ;

d) de même que pour 7, en trois étapes, 11, 13 et 17 sont dispensés.

Question 5 (6 points) :
Écrire une définition de la fonction `liste-dispenses`.

```
;;; liste-dispenses : LISTE[nat/>0/] -> LISTE[nat/>0/]
;;; (liste-dispenses L) rend la liste des numéros des soldats qui sont
;;; dispensés de corvée parmi la liste «L» en suivant le processus.
(define (liste-dispenses L)
  (if (pair? L)
      (cons (car L) (liste-dispenses (moins-mult-i-emes (car L) (cdr L))))
      '()))
```

Question 6 (4 points) :
Écrire la signature et une définition de la fonction `liste-heureux` qui calcule la liste des nombres heureux inférieurs ou égaux à un entier donné (qui correspond au nombre de soldats).

Attention : la sémantique n'est pas demandée

```
;;; liste-heureux : nat/>0/ -> LISTE[nat/>0/]
;;; (liste-heureux n) rend la liste des nombres heureux compris entre 2 et «n»
(define (liste-heureux n)
  (liste-dispenses (intervalle 2 n)))
```

Seconde partie : calcul du $n^{ème}$ nombre heureux

Dans la première partie, nous avons calculé la liste des nombres heureux inférieurs à un entier donné. Dans cette seconde partie, nous voudrions calculer, pour un entier i donné, le $i^{ème}$ nombre heureux (le 1^{er} nombre heureux est 2, le $2^{ème}$ nombre heureux est 3...). Noter qu'il s'agit d'un tout autre problème, même si, naturellement, la situation pratique est la même.

Pour ce faire, ne connaissant pas le nombre de soldats nécessaire pour avoir ce $i^{ème}$ nombre heureux, considérons les suites, aussi grandes que l'on veut, d'entiers successives dans le processus donné en introduction de l'exercice ; ce sont les suites suivantes :

a) la première suite d'entiers est la suite $2, 3, 4, 5, 6, 7, 8, 9 \ldots$;

b) la deuxième suite est la suite obtenue en supprimant, de la suite précédente, 2, le premier élément (qui est heureux) et les éléments de 2 en 2 ; c'est la suite $3, 5, 7, 9, 11, 13, 15, 17, 19, 21 \ldots$;

c) la troisième suite est la suite obtenue en supprimant, de la suite précédente, 3, le premier élément (qui est heureux) et les éléments de 3 en 3 ;c'est la suite $5, 7, 11, 17, 19 \ldots$;

d) ...

Nommons $g(i, j)$ le $j^{ème}$ élément (le premier élément étant numéroté 0) de la $i^{ème}$ suite. Par exemple, $g(1, 0)$ est égal à 2, $g(1, 3)$ est égal à 5, $g(2, 0)$ est égal à 3.

Notons que le $i^{ème}$ nombre heureux est alors égal à $g(i, 0)$, d'où une définition de la fonction qui rend le $i^{ème}$ nombre heureux :

```
;;; heureux : nat/>0/ -> nat/>0/
;;; (heureux i) rend le «i»-ème nombre heureux ;
;;; par exemple, (heureux 1) rend 2)
(define (heureux i)
  (g i 0))
```

On peut montrer (nous vous demandons de l'admettre) que

- $g(1, j) = j + 2$
- pour $i > 1$, $g(i, j) = g(i - 1, 1 + g(i - 1, 0) * (j \div (g(i - 1, 0) - 1)) + j$ **mod** $(g(i - 1, 0) - 1))$,
  $n \div m$ étant égal au quotient de n par m et n **mod** m étant égal au reste de la division entière de n par m.

Question 7 (7 points) :

En suivant la définition mathématique donnée ci-dessus, écrire la signature et une définition Scheme de la fonction g (attention à l'efficacité !).

Attention : la sémantique n'est pas demandée

```
;;; g : nat/>0/ * nat -> nat
;;; (g i j) rend le «j»-ème élément de la «i»-ème suite du processus de calcul
;;; des nombres heureux.
(define (g i j)
  (if (= i 1)
      (+ j 2)
      (let ((a (g (- i 1) 0)))
        (g (- i 1) (+ (* a (quotient j (- a 1)))
                      (remainder j (- a 1))
                      1)))))
```

En suivant la définition mathématique de g, on est amené à calculer plusieurs fois les valeurs de $g(k, 0)$ pour toutes les valeurs de k inférieurs à i, d'où inefficacité. Nous vous proposons maintenant d'écrire une définition beaucoup plus efficace.

Question 8 (7 points) :

En utilisant la fonction `g1` de spécification :

;;; *g1 : LISTE[nat/>0/] * nat -> nat*
;;; *HYPOTHÈSE: «L» contient les premiers nombres heureux, le dernier élément de*
;;; *la liste étant le premier nombre heureux (i.e. 2), l'avant-dernier élément*
;;; *de la liste étant le deuxième nombre heureux (i.e. 3)...*
;;; *(g1 L j) rend la valeur de g(i j), i étant égal à la longueur de la liste*
;;; *«L» plus 1*

écrire une définition de la fonction `liste-premiers-heureux` de spécification :

;;; *liste-premiers-heureux : nat/>0/ -> LISTE[nat/>0/]*
;;; *(liste-premiers-heureux i) rend la liste des «i» premiers nombres heureux,*
;;; *le dernier élément de la liste étant le premier nombre heureux (i.e. 2),*
;;; *l'avant-dernier élément de la liste étant le deuxième nombre heureux*
;;; *(i.e. 3), ..., le premier élément de la liste étant le «i»-ème nombre*
;;; *heureux*

```
(define (liste-premiers-heureux i)
  (if (= i 1)
      '(2)
      (let ((liste-h-i-1 (liste-premiers-heureux (- i 1))))
        (cons (g1 liste-h-i-1 0) liste-h-i-1))))
```

Question 9 (7 points) :

Écrire une définition de la fonction `g1`.

```
(define (g1 L j)
  (if (pair? L)
      (g1 (cdr L)
          (+ (* (car L) (quotient j (- (car L) 1)))
             (remainder j (- (car L) 1))
             1))
      (+ j 2)))
```