CHAPITRE 10

LA MULTIPLICATION DES FONCTIONS ET DES ARGUMENTS APPELS RECURSIFS MUTUELS

Si, par exemple, une fonction f de deux variables x et y, n'utilise pour son calcul que les valeurs de x+y, x-y, exp(x), ln(y) et xy, mais à plusieurs reprises, il sera alors plus avantageux de l'exprimer en fonction de ces cinq arguments là. On aura donc une écriture f(x, y) = g(s, d, e, l, p) où g est l'expression f(x, y) écrite en fonction de la somme s, la différence d, l'exponentielle e de x, le logarithme l de y et enfin du produit. En programmation classique on aurait posé s = x+y etc..., ici on compose des fonctions. Cette situation est fréquente et tous les exemples qui suivent, récursifs ou non, vont montrer que l'on peut complètement éliminer la notion de variable locale, en jouant sur plusieurs fonctions ayant autant d'arguments que nécessaire, et se passant la main, d'où le titre de ce chapitre. Par la suite ces fonctions peuvent faire une "ronde" en s'appelant mutuellement, c'est souvent le cas lorsqu'une fonction f donne le départ d'un problème à f', qui elle prend les choses en cours de route, et éventuellement redonne un départ f vers un sous-problème.

10-1° Tri par insertion On souhaite trier par ordre croissant, une liste L en commençant par la fin, et en insérant chacun de ses termes dans la queue de la liste, qui est donc déjà triée.
La fonction d'insertion d'un X dans L, consiste à balayer L depuis le début jusqu'à la bonne position.

```
(de ins (X  L) (cond      ((null L) (list X))
                          ((< X (car L)) (cons X L))
                          (t (cons (car L) (ins X (cdr L)))) ))
```

insère X à la bonne place dans une liste L déjà triée

```
(de trins (L) (cond       ((null L) nil)
                          (t (ins (car L) (trins (cdr L)))) ))
```

consiste à insérer le début dans la suite déjà triée.

Si par exemple on demande l'évaluation de (trins '(4 1 2 6)), une copie de trins est évaluée avec une première instanciation de L_0 = (4 1 2 6), comme cette liste n'est pas vide, cela provoque un premier appel de ins avec $X_1 = 4$ et L_1 = (trins '(1 2 6)).
Ce second appel de trins appelle à son tour ins pour $X_2 = 1$ et pour L_2 = (trins '(2 6)).
Ce troisième appel entraîne ins de $X_3 = 2$ dans L_3 = (trins '(6)).
Ce quatrième appel entraîne ins de $X_4 = 6$ dans L_4 = (trins nil).
Le cinquième et dernier appel de trins, renvoie la valeur nil, donc:
Le quatrième appel de ins renvoie la liste constituée par X_4, c'est-à-dire (6).
Le troisième appel de ins peut alors se terminer en renvoyant l'insertion de X_3 dans la liste (6), soit (2 6) puisque 2 est inférieur à 6.

Le deuxième appel de ins doit renvoyer l'insertion de X_2 dans (2 6), soit la liste (1 2 6).
Pour finir ins de $X_1 = 4$ dans (1 2 6) va s'appeler deux fois avec les arguments 4 et (2 6) puisque 4 est supérieur à 1, et enfin avec les arguments 4 et (6), pour renvoyer (4 6). L'appel précédent renvoyant (2 4 6), et celui d'avant noté comme premier appel de ins, fournissant le résultat (1 2 4 6), qui est donc également le résultat de trins de (4 1 2 6).

10-2° Tri par fusion : il s'agit pour trier une liste suivant cet algorithme, de la couper en deux parties, de trier séparément (suivant le même algorithme) ces deux parties, puis de les fusionner.

On a là un bel exemple de définition récursive pour un algorithme. Définissons d'abord les petites fonctions dont nous avons besoin :

```
(de tete (N L) (cond
                ((eq N 0) nil)
                ((null L) nil)
                (t (cons (car L) (tete (1-  N) (cdr L)))) ))
```

est une fonction qui donne les N premiers éléments de L, elle existe en lelisp sous le nom de "firstn", et "nthcdr" N donne la liste de tout ce qui suit les N premiers.

```
(de fusion (L M) (cond    ; construit une seule liste ordonnée à partir de deux listes triées.
                ((null L) M)
                ((null M) L)
                ((< (car L) (car M)) (cons (car L) (fusion (cdr L) M)))
                (t (cons (car M) (fusion L (cdr M))))))

(de trifus (L) (cond        ; consiste à fusionner les deux moitiés déjà triées, de la liste L.
                ((null L) nil)
                ((null (cdr L)) L)
                (t (fusion          (trifus (tete (quotient (length L) 2) L))
                                    (trifus (nthcdr (quotient (length L) 2) L)))) ))
```

Ce tri est assez rapide, voir l'exercice suivant. Il sera amélioré plus loin en séparant en une seule lecture les éléments d'ordre pairs et impairs.

10-3° Ecrire les passes successives de "trifus" pour la liste (77 231 571 17 299 61 29 8 832 21 120 101 50 70 857 71), sous forme de 5 colonnes en séparant les sous-listes auxquelles s'adressent les appels de trifus. Si une liste est de longueur N, divisible par 2 établir que le nombre de comparaisons k(N) suit une relation de récurrence k(N)=2*k(N/2) + N/2
Montrer alors que c'est $k(N)=N*\log_2(N)$

10-4° Tri par extraction : aller chercher le plus petit élément de la liste, trier ce qui reste (suivant le même algorithme) puis remettre en tête ce plus petit élément.

Cet exemple montre le dédoublement de fonctions qui deviennent mutuellement récursives (on ne veut évaluer X=min(cdr L) qu'une fois)

```
(de ret (X  L) (cond ; consiste à retirer la première occurrence de X dans L.
                ((null L) nil)
                ((eq (car L) X) (cdr L))
                (t  (cons (car L) (ret X (cdr L)))) ))
```

Il est possible de redéfinir une fonction existante (min existe pour deux nombres) cependant c'est une chose à éviter.

```
(de min (L) (cond ; renvoie l'élément minimal d'une liste numérique.
                ((null (cdr L))     (car L))
                ((< (cadr  L) (car  L)  (min (cdr  L))) ; rappel : cadr est le car du cdr
                (t (min (cons (car  L) (cddr  L)))))) ; cddr est le cdr du cdr
```

Lorsqu'une liste a au moins deux éléments, on doit donc la trier, et pour cela la fonction principale "triext" appelle une fonction soeur "tribis" dont les deux arguments sont la liste, et le plus petit élément de sa queue (son "cdr").

Tribis réalise alors l'insertion de ce plus petit élément X extrait, non pas dans L, mais dans la liste L privé de X, ou de son premier élément, et surtout triée au moyen de triext.
Les appels successifs de triext et tribis, consistent à extraire les plus petits éléments à chaque fois des queues des listes, à les comparer aux premiers éléments, et à mettre en attente les minimums obtenus. La liste argument de triext diminue d'un élément à chaque fois, ce qui entraîne bien entendu un arrêt.

```
(de triext (L) (if (null (cdr L)) L (tribis L (min (cdr L)))))

(de tribis (L X) (cond      ; X qui est le minimum du "cdr" de L est mis à sa place.
                    ((< (car L) X) (cons (car L) (triext (cdr L))))
                    (t (cons X (triext (cons (car L) (ret X (cdr L)))))) ))
```

10-5° Développer tous les appels, pour l'évaluation de (triext '(5 1 3 6 2)). Faire un tableau des instanciations successives de tous les paramètres intervenant.

10-6° Déterminant d'une matrice en développant suivant une colonne

On choisit de représenter une matrice par la liste L de ses colonnes, et on va la développer par rapport à sa première colonne. Accéder à une sous-matrice consiste à ignorer la première colonne: c'est facile en considérant (cdr L), et à ignorer une "ligne" de matrice. On construit pour cela deux fonctions consistant à produire des listes obtenues à partir des listes données en argument, en retirant soit le n-ième élément, soit les n-ième éléments de chacun des éléments de la liste.

```
(de ret1 (N L) (cond
                    ((null L) nil)      ; retire le n-ième élément de L
                    ((eq N 1) (cdr L))
                    (t (cons (car L) (ret1 (-  N 1) (cdr L))) )))
```

exemple : (ret1 3 '(a b c d e f)) ⇨ (a b d e f)

```
(de ret2 (N L) (if (null L) nil        ; retire les n-ième él. des él. de L
                    (cons (ret1 N (car L)) (ret2 N (cdr L)))))
```

exemple :

(ret2 3 '((a b c d) (a e f) (a o f) (s n c f)) ⇨ ((a b d) (a e) (a o) (s n f))

Imaginons que le développement soit avancé dans la première colonne jusqu'à la ligne N de façon à ce qu'une somme partielle DF de ce qui va constituer le déterminant soit acquise, et nommons RC la liste de ce qui reste dans la colonne, on sait que le prochain terme à ajouter est le terme suivant dans la colonne (car RC) multiplié par un cofacteur faisant appel à un déterminant et à un signe S .
On écrit tout naturellement deux fonctions mutuellement récursives. (Bien sûr la complexité en temps d'exécution de cet algorithme est proportionnelle à n! alors que celle de l'algorithme de Gauss est en n^3).

```
(de detbis (DF RC S N L) (if
                    (null RC) DF
                    (detbis (+ DF (* S (car RC) (det (ret2  N (cdr L))))) (cdr RC) (- S) (+ 1 N) L) ))

(de det (L) (if     ; donne le départ
                        (null  L) 1
                        (detbis 0 (car L) 1 1 L) ))
```

Voir dans le chapitre suivant, la transposition et le produit des matrices, qui s'écrivent très simplement avec cette structuration en liste de colonnes.

10-7° Fonctions statistiques moyenne, variance et covariance.
Pour une liste X = (x_1 x_2 ...x_n) de nombres, la moyenne m(X) est bien sûr ($\sum x_i$) / n, la variance var (X) = ($\sum x_i^2$) / n - m^2(X) = m(X^2) - m^2(X), enfin pour deux listes de nombres X et Y, cov (X, Y) = ($\sum x_i y_i$) / n - m(X).m(Y)

Nous définissons la fonction "moyenne" pour une liste L grâce à une fonction auxiliaire à trois arguments que nous appelons "mbis".

```
(de moyenne (L) (mbis 0 0 L))
```

Celle-ci considère le problème du calcul d'une moyenne en cours de résolution, lorsque l'on est en train de l'effectuer, on peut dire qu'à chaque instant, nous avons déjà compté N valeurs dont la somme est S, il nous reste à parcourir une liste de valeurs LR (la liste restante).
Deux cas se présentent alors, si cette liste restante est vide (null LR), c'est que le problème est achevé, il ne reste qu'à délivrer le résultat qui est S/N.
Sinon, il suffit de reconsidérer le même problème, mais avec les paramètres N+1, S auquel on ajoute le premier élément de LR, et enfin la liste LR privée de son premier élément. La fonction de trois arguments "mbis" est récursive terminale.

```
(de mbis (N S LR) (if (null LR)    (/ S N)
                                   (mbis (+ N 1) (+ S (car LR)) (cdr LR))))
```

La fonction "moyenne" pour une liste, n'est là que pour lancer le mécanisme, car au début N = 0 et S = 0. Construisons sur le même modèle la fonction que nous appelons "variance" pour une liste L de nombres.

```
(de var (L) (varbis 0 0 0 L))
```

La fonction "varbis" possède quatre arguments : le nombre N de valeurs déjà lues, leur somme S1, la somme S2 de leurs carrés, et la liste LR restant à lire. Rappel, le "if" que nous employons pour changer, est à trois arguments:
a) le test, si celui-ci n'est pas faux, alors c'est l'expression suivante qui est renvoyée
b) cette expression est ici la différence entre S2/N et $(S1/N)^2$ conformément à la définition de la variance
c) enfin l'expression qui doit être évaluée si le test est faux, et ici c'est le même calcul qui recommence, mais avec :
N+1 valeurs,
une somme S1 augmentée de la première valeur (car LR) de LR,
une somme S2 augmentée du carré (* (car LR) (car LR)) de cette même valeur,
et une liste restante diminuée (cdr LR).

```
(de varbis (N S1 S2 LR) (if (null LR)
                            (- (/ S2 N) (/ (* S1 S1) (* N N)))
                            (varbis (+ N 1) (+ S1 (car LR)) (+ S2 (* (car LR) (car LR))) (cdr LR))))
```

Exemple : (moyenne '(1 2 3 4 5 6 7 8 9)) = 5
Si les données sont trop longues, on affecte (chapitre suivant) par exemple sous le nom de LX une liste de valeurs par :

```
(set 'LX '(32 25 21 19 16.5 15 9 5.5)) (moyenne LX) =  1.787500e+1
(var LX) =  6.292188e+1
```

La covariance de deux listes est une fonction construite encore sur le même modèle, au moyen d'une fonction "covbis" à six variables.

```
(de cov (LX LY) (covbis 0 0 0 0 LX LY))
```

Si l'une ou l'autre des listes RX ou RY restant à lire, est vide, le résultat donné est alors la covariance SXY/N - (SX/N)*(SY/N). Sinon on repart avec N+1 couples déjà lus, SX, SY, SXY sont augmentés, et les nouvelles listes restant à parcourir sont diminuées.

```
(de covbis (N SX SY SXY RX RY)
        (if (or (null RX) (null RY))
                (- (/ SXY N) (/ (* SX SY) (* N N)))
                (covbis (+ N 1) (+ SX (car RX)) (+ SY (car RY))
                        (+ SXY (* (car RX) (car RY))) (cdr RX) (cdr RY) )))
```

Exemple : (cov '(2 4 7 9 10 13 16 19) '(32 29 22 26 19 10 14 8)) = -42

10-8° Ajustement linéaire. Construire un ensemble de fonctions amenant à une fonction "corr" à deux arguments X Y liste de réels de mêmes longueurs renvoyant la liste (mX mY sX sY cov cor a b a' b') des moyennes écarts-type, coefficient de corrélation et paramètres des droites de régression. On rappelle que cor (X, Y) = cov (X, Y) / sX.sY, a = cov (X, Y) / var (X) et a' = var (Y) / cov (X, Y) (c'est bien cette formule et non l'inverse), enfin les deux droites passent au point moyen ce qui permet de trouver b et b'.

Il serait possible de calculer la droite de regréssion linéaire et le coefficient de corrélation en utilisant les fonctions précédentes, mais ce serait assez maladroit dans la mesure où les mêmes formules seraient calculées plusieurs fois et les mêmes listes de valeurs seraient lues plusieurs fois. Aussi préférons nous reprendre le problème au départ et le généraliser à une liste de couples pondérés par des effectifs relatifs.
Avec cette nouvelle façon de structurer les données, nous avons une liste de triplets tels que (EF X Y) signifiant que (X, Y) est présent EF fois, (effectif) ainsi par exemple :

```
(set 'donnees '((35 25 20) (6 35 20) (2 45 20) (5 25 30) (25 35 30) (7 45 30) (2 25 40) (3 35 40) (15 45 40)))
```

Pour une seule liste L composées de triplets, la fonction "ajust" va fournir comme résultat les paramètres a, b et ρ que nous souhaitons.

```
(de ajust (L) (ajbis 0 0 0 0 0 0 L))
```

La fonction "ajbis" prend, là encore, le problème en cours de résolution en commençant par tester s'il reste du travail à faire, si la liste LR de triplets restants est vide, alors on renvoie comme résultat une liste constituée par des chaînes de caractères entre guillemets, et les paramètres souhaités.

```
(de ajbis (N SX S2X SY S2Y SXY LR)
(if (null LR) (list "Droite de regression y - " (/ SY N) '" = "
                    (/ (- (* N SXY) (* SX SY)) (- (* N S2X) (* SX SX)))
                    " (x - " (/ SX N) '") et coefficient de corrélation "
                    (/ (- (* N SXY) (* SX SY)) (sqrt (* (- (* N S2X) (* SX SX)) (- (* N S2Y) (* SY SY))))))
(ajter N SX S2X SY S2Y SXY (caar LR) (cadar LR) (caddar LR) (cdr LR))  )  )
```

Dans le cas où le travail n'est pas terminé, on fait cette fois appel à une autre fonction "ajter", à laquelle on fournit les paramètres N, SX, S2X, SY, S2Y, SXY non modifiés et les trois valeurs EF, X, Y du triplet suivant. "Ajter" se charge alors de redonner la main à "ajbis", en augmentant les sommes, l'intérêt de ce va-et-vient (cette récursivité mutuelle) entre les deux fonctions est d'éviter de refaire plusieurs fois le calcul de EF, X, Y, ou de prendre des variables locales. Rappel : "caar" signifie la fonction "car" composé avec elle-même, "cadar" signifie la composition de fonctions $car_o cdr_o car$ etc...

```
(de ajter (N SX S2X SY S2Y SXY EF X Y LR)
        (ajbis  (+ N EF) (+ SX (* EF X)) (+ S2X (* EF X X)) (+ SY (* EF Y))
                (+ S2Y (* EF Y Y)) (+ SXY (* EF X Y)) LR))
```

Exemple : (ajust donnees) = (Droite de regression y - 27.7 = 0.6.669853 (x - 33.2) et coefficient de corrélation 0.6.955562)

10-9° Permutations

Cet exercice nécessite de dédoubler certaines fonctions afin d'éviter d'inutiles évaluations. Construisons d'abord une fonction réalisant la liste des différentes insertions d'un X dans une liste L.

```
(de place (X  L) (if       (null L) (list (list X))
                           (placebis X  nil L nil ) ))
```

Exemple (place 'X '(A B C)) = ((A B C X) (A B X C) (A X B C) (X A B C))

```
(de placebis (X P Q R) (if
                  (null Q) (cons (append P (list X)) R)
                  (placebis X (append P (list (car Q))) (cdr Q) (cons (append P (cons X Q)) R) ))
```

Pour bien comprendre ce que fait "placebis", il faut exécuter à la main "place" avec X et L = (A B C) en faisant un tableau des différentes instanciations de P Q R jusqu'à ce que Q soit vide, en partant de P vide et Q = L

```
(de perm (L) (if   (null (cdr L)) (list L)
                   (permbis (car L) (perm (cdr L)) nil) ))

(de permbis (X Q R) (if
                   (null Q) R
                   (permbis X (cdr Q) (append (place X (car Q)) R) ))
```

Là aussi R symbolise le résultat qui est la concaténation de tous les placements possibles de X = (car L) dans les éléments de Q qui sont au départ les permutations de (cdr L).

10-10° Essayer "perm" sur la liste à 5 éléments (d-amour marquise vos-beaux-yeux me-font mourir).

10-11° Longueur du plus grand plateau d'éléments consécutifs dans une liste (problème donné par J.Arsac). On souhaite par exemple (plat '(A A B B B B C C D)) donne : B 4 fois ou encore (plat '(A B A A A B B)) donne : A 3 fois

Il s'agit par exemple de chercher l'homonyme le plus répandu dans un annuaire. Cet exemple va nous donner l'occasion de montrer qu'il est souvent possible de se tirer d'affaire par des fonctions récursives possédant beaucoup de paramètres.
Si, en parcourant la liste L , X a déjà été lu K fois, et que N représente la longueur déjà reconnue comme la plus longue, pour une certaine valeur V, et si Y est l'élément suivant premier de la liste L restant à parcourir, alors on peut écrire les conditions dans l'ordre :
Si L est vide, on peut donner la réponse qui est V rencontré consécutivement N fois.
Si L n'est pas vide et que X est égal à son suivant Y, alors il faut tester si X n'est pas une réponse potentielle meilleure que V;
Si c'est faux, on continue le balayage de la liste;
Si c'est vrai, on continue également, mais X prend la place de V comme meilleure réponse provisoire.
Si X et Y sont différents, alors on peut abandonner X, il ne sera pas meilleur que V, par contre il faut compter les Y.
La fonction "plabis" réalise ce travail récursif, et la fonction "plat" n'est là que pour fixer le démarrage.

La meilleure méthode de compréhension est encore de faire évaluer à la main la fonction "plat" sur une liste telle que (a a a a f f g g g b b b b b h a a a a a h h) en notant les valeurs successives de tous les paramètres, pour chaque appel.

```
(de plabis (V N K X Y L) (cond
                ((null L) (list V 'présent N 'fois))
                ((eq X Y) (cond
                        ((< K N) (plabis V N (1+  K) X (cadr L) (cdr L)))
                        (t (plabis  X (1+  N) (1+  N) X (cadr L) (cdr L)))))
                (t  (plabis V N 1 Y (cadr L) (cdr L)))))
```

Maintenant cette fonction doit être lancée au moyen de :

```
(de plat (L)  (cond
                ((null L) nil)
                ((atom L) (plat (explodech   L)))          ; pour un mot
                ((null (cdr L)) 'inintéressant)            ; cas d'une lettre seule
                (t (plabis nil 0 1 (car L) (cadr L) (cdr L)))))
```

10-12° Algorithme de Hörner
Le calcul d'un polynôme de degré n pour une valeur de x demande n(n+1)/2 multiplications si on exécute $a_0 + a_1x + ... + a_nx^n$, et seulement n multiplications en faisant $a_0 + x(a_1 + x(a_2 + ... + x(a_n)...)$. L'algorithme de Hörner consiste à utiliser la seconde formule pour calculer une expression polynômiale. Si le sinus n'est pas présent parmi les fonctions prédéfinies, il est possible de le réécrire grâce à un développement limité.

$$\sin(x) = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} = x(1 - \frac{x^2}{6}(1 - \frac{x^2}{20}(1 - \frac{x^2}{42})))$$

La fonction sinus peut alors être calculée par ce polynôme, dans un intervalle suffisamment petit [0 , π/2], pour que l'erreur n'y soit pas trop grande, et en dehors de cet intervalle on utilise les propriétés d'imparité, de périodicité, et de symétrie par rapport à π/2. (PI aura été préalablement affecté de la bonne valeur). Le nombre d'appels récursifs est grossièrement proportionnel à X.

```
(de sin (X)  (cond
                ((< X 0)          (- (sin (- X))))
                ((< PI X)         (sin (- X (* 2 PI))))
                ((< (/ PI 2) X)   (sin (- PI X)))
                (t        (*  X (-  1 (* (/  (*  X  X)  6) (-  1 (* (/ (*  X  X) 20) (- 1(/ (*  X  X) 42)) ))))) )))
```

10-13° Donner une fonction récursive "horner" à deux arguments ; X l'inconnue, et L la liste suivant les puissances croissantes des coefficients du polynôme P dont on veut calculer les images P(X).

```
(de horner (X L ) (hornerbis X L 0))

(de hornerbis (X L R) (if (null L) R (hornerbis X (cdr L) (+ (car L) (* X R))))) ; R est le résultat
```

10-14° En application, on donne la formule de Hastings pour l'intégrale de $\exp(-x^2)$ entre 0 et x multiplié par $2/\sqrt{\pi}$, c'est approximativement $1 - 1/ (0.078108 + 0.000972x + 0.230389x^2 + 0.278393x^3 + x^4)^4$. Construire la fonction "hastings(x)" et comparer avec exp(x2).

```
(de hast0 (X) (horner X '(0.078108 0.000972 0.230389 0.278393 1)))

(de hast1 (X) (- 1 (divide 1 (* X X X X))))

(de hastings (X) (hast1 (hast0 X)))
```

10-15° Calcul de sinus et cosinus mutuellement récursifs avec tests d'arrêts sin(x) = x et cos(x) = 1 - $x^2/2$ (égalités en machine) en utilisant les formules de la trigonométrie élémentaire et sin(x) = 2sin(x/2)cos(x/2) pour se ramener au voisinage de 0.

10-16° **Surface d'un domaine limité par une ligne brisée.** En s'inspirant des calculs de moyenne et de l'algorithme de Hörner, on veut la surface et éventuellement le centre de gravité d'un domaine limité par (M_0, M_1, ... M_{n-1}) ligne brisée sans points double autre que $M_n = M_0$.
On peut toujours décomposer le domaine en rectangles et triangles rectangles dont les côtés de l'angle droit sont parallèles aux axes d'un repère orthonormé fixé à l'avance. En repérant un rectangle par (R x y l h) et un triangle par (T x y l h) on peut calculer son aire par |lh| (resp. |lh|/2) et son centre de gravité (x + l/2, y + h/2) resp. (x + l/3, y + h/3).
Si on ne souhaite que l'aire, c'est une somme de trapèzes relatifs $|\Sigma(y_i + y_{i+1})(x_{i+1} - x_i)| / 2$ soit $| \Sigma (x_i y_{i+1} - x_{i+1} y_i) | / 2$ pour i dans Z / nZ

10-17° On définit les fonctions "bifbis" et "bif" par:

```
(de bifbis  (P Q R N)   (if     (eq R N) Q
                                (bifbis Q (+  P  Q) (+  1  R) N)))
```

et (de bif (N) (bifbis 1 1 1 N))
Calculer (bif 12) en faisant un tableau des différentes valeurs de P Q R N, lors des appels successifs de bifbis.
Quelle relation de récurrence peut-on donner à propos de la suite des valeurs bif (N) pour les entiers N ? Réécrire cette fonction avec un paramètre de moins.

P	1	1	2	3	5	8	13	21	34	55	89	144
Q	1	2	3	5	8	13	21	34	55	89	144	233
R	1	2	3	4	5	6	7	8	9	10	11	12

N étant constant à 12
C'est bien sur la version récursive terminale de Fibonacci où fib(0) = fib(1) = 1 et fib(n) = fib(n-1) + fib(n-2) pour n > 1.
On peut réécrire avec l'appel (bif n) = (bifbis 1 1 n) où bifbis est à trois arguments :

```
(de bifbis (p q n) (if (< n 2) q (bifbis q (+ p q) (- n 1))))
```

10-18° **Calculer en même temps le min et le max d'une liste.**

```
(de minmax (L) (if (null L) L (miax (cdr L) (car L) (car L))))
(de miax (L i j) (cond
                 ((null L) (list i j))
                 ((< (car L) i) (miax (cdr L) (car L) j))
                 ((< j (car L)) (miax (cdr L) i (car L)))
                 (t (miax (cdr L) (i j)) ))
```

10-19° Ecrire la fonction "place" à trois arguments : un objet X, un entier N, une liste L, dont le résultat doit être la liste L où l'objet X a été intercalé à la N-ième place (avec 1 ≤ N ≤ longueur (L)).

Exemple (place 'a 3 '(a z e r t y)) → (a z a e r t y) ou encore (place 'u 7 '(a z e r t y)) → (a z a e r t y u)

```
(de place (X N L) (cond
                  ((null L) (list X))
                  ((eq N 1) (cons X L))
                  (t (cons (car L) (place X (- N 1) (cdr L)))) ))
```

10-20° Reprendre l'exercice sur une suite de N dont les termes et leurs différences successives réalisent une partition de N (chapitre 6).

```
(de sbis (L D) (if (member D L) (sbis L (1+ D)) (print D (+ D (car L))) (sbis (cons (+ D (car L)) L) (1+ D))))
(de suite () (sbis '(1) 2))
```

Attention, "suite" est illimitée, il faut veiller à ce que le lisp utilisé ait des commandes telles que "break" ou "abort".

10-21° **Dérivation des polynômes.** Si les polynômes tels que $2 + 3x + 4x^2 + 5x^3$ sont représentés par la liste de leurs coefficients suivant les puissances croissantes (2 3 4 5), on souhaite calculer le polynôme dérivé, c'est-à-dire ici (3 8 15).
a) Trouver une fonction der pouvant réaliser l'application $P \rightarrow P'$. On pourra s'aider d'une fonction auxiliaire derbis à deux arguments: un polynôme P débutant par x^n, et l'entier n, renvoyant le polynôme P dans lequel seul le premier terme a été dérivé
b) En déduire une fonction dermultiple à deux paramètres P et K calculant la dérivée K-ième du polynôme P.
c) Construire sur le même modèle une fonction calculant une primitive d'un polynôme.
d) Compéter dermultiple afin qu'elle donne les primitives d'ordre K pour K négatif.

```
(de der (P) (if (null (cdr P)) '(0) (derbis 1 (cdr P)) ))
(de derbis (N P) (if        (null P) nil
                            (cons (* N(car P)) (derbis (+ N 1) (cdr P)))))
(de dermultiple (K P) (cond
                            ((eq K 0) P)
                            ((eq K 1) (der P))
                            ((< K 0) (prim (dermultiple (+ K 1) P)))
                            (t (der (dermultiple (- K 1) P)))))
(de prim (P) (cons (gensym) (primbis 0 P))) ; (gensym) est une fonction prédéfinie produisant un symbole
(de primbis (N P) (cond
                  ((null P) nil)
                  ((numberp (car P)) (cons (/ (car P) (+ N 1)) (primbis (+ N 1) (cdr P))))
                  (t (cons (car P) (primbis (+ N 1) (cdr P))))))
```

10-22° **Les polynômes orthogonaux de Tchébicheff sont définis par la relation :**
$xT_n(x) = T_{n+1}(x) + T_{n-1}(x)/4$ pour $n \geq 2$, et $T_0(x) = 1$, et $T_1(x) = x$
Chercher une représentation simple des polynômes, et programmer le calcul des coefficients de T_{n+1} à partir de ceux de T_n et T_{n-1} de façon à ce que les appels récursifs s'enchaînent suivant un ordre total.

10-23° **Enumération diagonale de Cantor,** pour démontrer que les rationnels formaient un ensemble dénombrable, a établi une correspondance bijective f entre N^2 et N débutant par :
$(0,0) \rightarrow 0$, $(1,0) \rightarrow 1$, $(0,1) \rightarrow 2$, $(2,0) \rightarrow 3$, $(1,1) \rightarrow 4$, $(0,2) \rightarrow 5$, etc
a) Faire le dessin sur un quadrillage, et constater que l'on numérote les couples d'entiers en suivant des "diagonales".
b) Donner une définition récursive de f, et en déduire une fonction lisp.
c) Chercher l'expression directe de f(i,j) en considérant que pour f(i) = f(i, 0) on a la relation de récurrence: $\phi(i+1) = \phi(i) + i + 1$.
En déduire une autre fonction Lisp, et faire des comparaisons de temps d'exécution.

Solution : les relations de récurrence sont faciles à trouver. Exemple f(1, 3) = 13 et f(3, 1) = 11

```
(de cantorec (i j) (cond    ; 1+ est la fonction ajoutant 1 à son argument
                  ((eq j 0) (if (eq i 0) 0 (1+ (cantorec 0 (1- i)))))
                  (t (1+ (cantorec (1+ i) (1- j))))))
(de cantor (i j) (+ j (div (* (+ i j) (+ i j 1)) 2)))      ; formule directe
```

10-24° Décomposition d'un entier en facteurs premiers au moyen de fonctions récursives.

```
(de prem0 (N) (cond
                ((eq N 1) '(1) )
                ((eq N 2) '(1 2) )
                ((< N 5)  '(1 2 3) )
                (t (prem1 (prem0 (1- N)) N ))))

(de prem1 (L N) (prem2 L (cdr L) (truncate (sqrt N)) N)))

(de prem2 (LP RP M N) (cond ;  LP liste précédente RP reste à voir M est la racine de N
                null RP) (appendD LP (list N)))          ; il n'y a plus de diviseurs potentiels
                ((< M (car RP)) (append LP (list N)))    ; il n'y a plus de diviseur à voir
                ((eq (modulo N (car RP)) 0) LP)          ; N n'est pas premier
                (t (prem2 LP (cdr RP) M N ))))
(de decomp (N) (decompbis (cdr (prem0 N)) N nil))        ; décomposition en facteurs premiers

(de decompbis (L N R) (cond ; fait la recherche dans la liste L des nb premiers possibles
                ((null L) (reverse R))     ; R est la liste des facteurs
                ((eq (modulo N (car L)) 0)
                        (decompbis L (divide N (car L)) (cons (car L) R)))
                        ; quand un facteur est trouvé on continue à chercher les facteurs du quotient
                (t (decompbis (cdr L) N R))))
```

10-25° Suites de Farey. Produire la suite ordonnée des fractions de dénominateurs inférieurs à N, et premiers avec N (sans diviseur commun), comprises entre 0 et 1

```
(de divise (P N) (fixp (divide N P))) ; ou encore (eq (modulo N P) 0) ; répond vrai ou faux
(de prembis (D P N) (cond
                ((> D (div P 2)) t)
                ((divise D P) (ifn (divise P N) (prembis (1+ D) P N))) ; (ifn A B) évalue B si A est faux
                (t (prembis (1+ D) P N))))

(de prem (P N) (cond
                ((< N P) (prem N P))
                ((divise P N) nil)
                (t (prembis 2 P N)))) ; prédicat disant si p et n premiers entre eux

(de denom (L P N) (cond
                ((eq (1- N) P) (cons P L)); fini
                ((prem P N) (denom (cons P L) (1+ P) N))
                (t (denom L (1+ P) N)))) ; renvoie la liste des dénominateurs
(de farey (N) (denom nil 2 N))
```

Exemple : (farey 28) = (27 26 25 24 23 22 21 20 19 18 17 16 15 13 12 11 10 9 8 6 5 3)

10-26° Polynômes cyclotomiques : pour n > 0, le polynôme cyclotomique d'ordre n est le polynôme normalisé dont les racines sont les racines n-ième primitives de 1. (C'est à dire non racines de 1 pour un ordre inférieur, son degré est $\phi(n)$ indicateur d'Euler).
Exemple $P_1(x) = x-1$, $P_2(x) = x+1$, $P_3(x) = x^2 + x + 1$, $P_4(x) = x^2 + 1$, $P_5(x) = x^4 + x^3 + x^2 + x + 1$, $P_6(x) = x^2 - x + 1$ (il faudra tester jusqu'à n = 105).
a) Programmer la division euclidienne de deux polynômes quelconque et leur pgcd
b) La stratégie est la suivante : si n = 1 P_1 est x-1. Sinon l'algorithme consiste à partir de $P_n = x^{n-1} + x^{n-2} + ... x + 1$ qui admet toutes les racines n-ième de 1, puis de déterminer les diviseurs p premiers successifs de n tels que p < n. Si q = n div p, initialiser P_q avec $x^{q-1} + x^{q-2} + ... + x + 1$, calculer le pgcd normalisé de P_n par P_q et remplacer P_n par son quotient avec ce pgcd puis recommencer. (Si n est premier Pn restera donc dans son état initial.)

10-27° Problème des grands nombres : Le but du programme est d'obtenir des calculs arithmétiques exacts pour de très grandes valeurs numériques, ou bien pour des valeurs comportant beaucoup de chiffres significatifs. En adoptant une représentation des grands entiers sous forme de listes de leurs chiffres à l'envers, par exemple 328 279 481 par (1 8 4 9 7 2 8 2 3), on veut construire des fonctions "mos", "uos" et "lum" récursives réalisant les opérations + - * à l'envers pour deux listes de chiffres et une valeur représentant la retenue. Réexposer l'algorithme d'addition appris à l'école primaire en le détaillant bien, notamment en l'expliquant sur un ou plusieurs exemples.
Construire la fonction "sup" répondant "vrai" ou "faux" dans tous les cas de signes.
Exemple : (sup '(- 7 8 5 4 2 0 0 1 2 3 7 8) '(- 1 0 4 5 6 4 5 6 8 8 2 2 3)) → nil
(sup '(1 2 3 5 6 4 7) '(1 2 3 5 6 4 8)) → t

```
(de retirzero (L) (cond     ; retire les zéros en tête d'une liste
                    ((null (cdr L)) L)
                    ((eq 0 (car L)) (retirzero (cdr L)))
                    (t L)))
(de sup (X Y) (cond         ; test "X supérieur strictement à Y"
                    ((eq '- (car X)) (if (eq '- (car Y)) (sup (cdr Y) (cdr X)) nil))
                    ((eq (car Y) '-) t)
                    ((> (length X) (length Y)) t)
                    ((> (length Y) (length X)) nil)
                    ((null (cdr X)) (> (car X) (car Y)))
                    ((> (car X) (car Y)) t)
                    ((> (car Y) (car X)) nil)
                    (t (sup (cdr X) (cdr Y))) ))
(de mos (A B r) (cond     ; A B positifs et 0 ≤ r < 9 est la retenue
                    ((null A) (if (null B) (list r) (mos B (list r) 0)))
                    ((null B) (mos B A r))
                    (t (cons (modulo (+ (car A) (car B) r) 10) (mos (cdr A) (cdr B) (div (+ (car A) (car B) r) 10))))))
(de som (A B) (cond         ; tous cas de signes
                    ((and (eq (car A) '-) (eq (car B) '-))
                            (cons '- (reverse (mos (reverse (cdr A)) (reverse (cdr B)) 0))))
                    ((eq (car A) '-) (sou B A))
                    ((eq (car B) '-) (sou A B))
                    (t (retirzero (reverse (mos (reverse A) (reverse B) 0)))) ))
(de sou (A B) (cond         ; soustraction dans tous les cas de signes
                    ((equal A B) '(0))
                    ((and (eq (car A) '-) (eq (car B) '-)) (sou (cdr B) (cdr A)))
                    ((eq (car A) '-) (cons '- (som (cdr A) B)))
                    ((eq (car B) '-) (som  A (cdr B)))
                    ((sup B A) (cons '- (sou B A)))
                    (t (retirzero (reverse (uos (reverse A) (reverse B) 0 )))) ))
(de uos (A B R) (cond     ; ne fonctionne que si A > B et tous deux positifs
                    ((null A) nil)
                    ((null B) (if (eq R 0) A (uos A '(1) 0)))
                    ((> (+ (car B) R) (car A)) (cons (- (+ 10 (car A)) (+ (car B) R)) (uos (cdr A) (cdr B) 1)))
                    (t (cons (- (car A) (+ (car B) R)) (uos (cdr A) (cdr B) 0))) ))
(de lumatom (A b r) ; multiplication de A inversée par le chiffre b avec la retenue r
                    (if (null A)        (retirzero (list r))
                    (cons (modulo (+ (* (car A) b) r) 10) (lumatom (cdr A) b (div (+ (* (car A) b) r) 10)))))
(de lum (A B) (ifn (null B) (mos (lumatom A (car B) 0) (cons 0 (lum A (cdr B))) 0) ))
                    ; pour A, B positifs à l'envers
(de mul (A B) (cond ; tous cas de signes
                    ((and (eq (car A) '-) (eq (car B) '-)) (reverse (lum (reverse (cdr A)) (reverse (cdr B)))))
                    ((eq (car A) '-) (cons '- (reverse (lum (reverse (cdr A)) (reverse (cdr B))))))
                    ((eq (car B) '-) (cons '- (reverse (lum (reverse A) (reverse (cdr B))))))
                    (t (reverse (lum (reverse A) (reverse B)))) ))
```

10-28° Que pourrait-on envisager pour compléter ce programme ou l'améliorer ? (prendre une base 100 ou 1000, quels problèmes posent les nombres négatifs, les nombres à virgule, ... ?)

10-29° Division de a par b en utilisant un algorithme du à Knuth. On garde la tête formée des chiffres (en base 10) a' = $a_0a_1a_2.....a_n$ de a, si b est formé de n chiffres $b_1....b_n$, et au cas où a et b ont même nombre de chiffres, on prendra $a_0 = 0$. On pose alors d = E(10/(1+b_1)) et q = min (9, d*E((10a_0+a_1)/db_1) + d - 1) où E désigne la partie entière. Si q*b ≤ a' alors on garde q sinon on essaie avec q-1 (trois essais suffisent). La division se poursuit entre a'-q*b concaténé avec la queue du dividende a.

10-30° Division de a par b avec d décimales par l'algorithme naturel :
a) On concatène d'emblée d fois 0 en queue de a et Q = nil
b) Si a < b on stoppe, alors si Q = nil, le résultat est 0 sinon c'est Q dans lequel une virgule est placée avec d chiffres à sa droite.
c) Sinon b = $b_1....b_n$ on prend a' = $a_1....a_n$ et a" la suite a_{n+1}.... de a.
Si a' = b, alors q = 1
Si a' < b on prend a' = $a_1....a_{n+1}$ et a" = a_{n+2}.... ce qui est toujours possible
Si a' > b soit q = 5 provisoirement et r = qb,
si qb < a' répeter r := r + b, q := q + 1 jusqu'à r > a'
sinon tant que r > a' faire r := r - b, q := q - 1

10-31° Division des grands nombres avec recherche dichotomique

```
(de division (A B) (cond ; renvoie le grand nombre quotient de A par B à  l'unité près
                ((sup B A) '(0))
                ((equal A B) '(1))
                (t  (retirzero (divbis (firstn (length B) A) B (nthcdr (length B) A))))))

(de divbis (A B SA) (cond ; A suivi de SA est le dividende, B le diviseur
                ((sup B A)(if (null SA) nil (divbis (append A (list (car SA))) B (cdr SA))))
                ((equal A B) (cons 1 (division SA B)))
                (t  (divter (dicho A B 0 '(0) 10) A  B SA)) ))

(de divter (Q A B SA) ; le car de Q est le premier chiffre q du quotient, le cdr est B*q
                (cons (car Q) (divbis (sou A (cdr Q)) B SA))) ; seul "sou" retire les zéros en tête

(de dicho (A B n1 P1 n2) (cond ;renvoie le premier chiffre q du quotient suivi de B*q
                ((equal A P1) (cons n1 P1)) ; cette clause n'est pas indispensable mais avantageuse
                ((equal n2 (1+ n1)) (cons n1 P1))
                (t (dichobis A B n1 P1
                        (retirzero (reverse (lumatom (reverse B) (div (+ n1 n2) 2) 0))) n2))))

(de dichobis (A B n1 P1 Pm n2)
(if (sup Pm A) (dicho A B n1 P1 (div (+ n1 n2) 2))  (dicho A B (div (+ n1 n2) 2) Pm n2)))
```

10-32° Relation de Bezout pour les grands entiers X > Y donnés. Trouver A, B, C tels que AX + BY = C (pgcd de X et Y).

On partira avec 6 variables 1X + 0Y = X et 0X + 1Y = Y puis :
$a_0X + b_0Y = c_0$ et $a_1X + b_1Y = c_1$ donneront $a_1X + b_1Y = c_1$ et $(a_0-a_1q)X + (b_0-b_1q)Y = c$ où $c = c_0-c_1q$ et la fin si c_1=0, le reste précédent c_0 sera le pgcd de X et Y.

```
(de bezout (a0 b0 c0 a1 b1 c1) (print a1 b1 c1) (let ((q (divi c0 c1)) (r (sou (mul c1 q) c0)))
                        (if (eq r '(0)) 'fini (bezout a1 b1 c1 (sou a0 (mul a1 q)) (sou b0 (mul b1 q)) r)) )))
```

Exemple
(bezout '(3 4 1 5 6 4 5 2 6 0 5) '(4 3 8 5 1 6 5 1 5)) donne
a = -13886944, b = 1081666775, c = 5

10-33° Résolution de systèmes linéaires. Reprendre la triangulation d'une matrice par la méthode de Gauss, mais en organisant les matrices comme liste de leurs lignes.

La matrice de départ est cette fois, représentée par la liste de ses lignes
La matrice triangulée sera représentée par la liste des lignes à partir de la diagonale de la dernière à la première.

```
(de tete (L N) (cond ; la fonction "firstn" existe dans certains Lisp
                ((eq N 0) nil)
                ((null L) L)
                (t (cons (car L) (tete (cdr L) (1- N))))))
(de unit (L) (if (eq (car L) 0) L (mapcar (lambda (Q) (divide Q (car L))) L)))
; Renvoie la liste obtenue en divisant tous les éléments par le premier d'entre eux s'il est différent de 0

(de sous (L0 L1 A) (if
                (null L0) L0 (cons (- (car L0) (* (car L1) A)) (sous (cdr L0) (cdr L1) A) ) ))
; Renvoie la liste L0 où on a soustrait terme à terme avec les éléments de L1 multipliés par A

(de inc (L R) (incbis (cdr L) R 0))
(de incbis (L R X) (if
                (null R) (+ X (car L))
                (incbis (cdr L) (cdr R) (- X (* (car L) (car R)))) ))
```

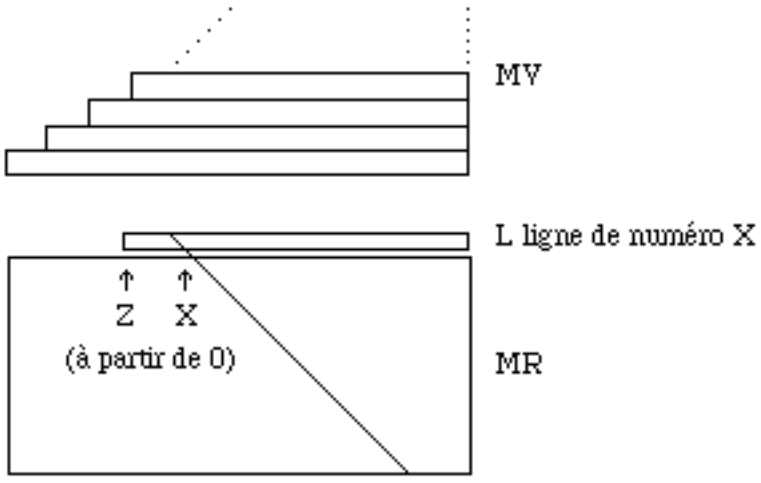
Calcule l'inconnue X à partir des inconnues déjà calculées dans R et des coefficients de la ligne L, dont le dernier représente le terme constant.

```
(de tril (MV X Z L MR) (cond
                ((< Z X) (if (eq (car (nth (1- (- X Z)) MV)) 0)
                        (tril (append (tete MV (1- (- X Z))) ; cas égal à 0
                                (cons (unit L) (nthcdr (- X Z) MV))) X (1+ Z) (cdr (nth (1- (- X Z)) MV)) MR)
                        (tril MV X (1+ Z) (cdr  (sous L (nth (1- (- X Z)) MV) (car L))) MR))) ; sinon
                ((null MR) (cons (unit L) MV))
                ((eq (car L) 0) (tril (cons L MV) (1+ X) 0 (car MR) (cdr MR)))
                (t (tril (cons (unit L) MV) (1+ X) 0 (car MR) (cdr MR)) )))

(de resobis (R MTR) (cond
                ((null MTR) R)
                ((eq (caar MTR) 0) 'pas-de-cramer)
                (t (resobis (cons (INC (car MTR) R) R) (cdr MTR)))))
```

MTR représente la pyramide des lignes d'une matrice triangulaire supérieure à l'envers : la dernière ligne ne contenant que deux élément, est en premier.

```
(de reso (M)  (resobisnil (tril nil 0 0 (car M) (cdr M)) ))
```



10-34° Forme en "te" pour les verbes japonais : Cette forme correspond à une certaine substantivisation, elle est employée comme substantif désignant une action ou dans une succession d'actions de façon à ce qu'il n'y ait toujours qu'un seul verbe dans la phrase.
Exemples :
V-te kudasai impératif poli
(tabete kudasai = mange) tabenasai = mange, est l'impératif plus familier
V-te imasu forme progressive (tabete imasu = être en train de manger)
V1-te kara V2-masu succession V1 puis V2
(te o aratte kara gohan o tabemas = après m'être lavé les mains, je mange)
V1-te V2-te .... Vn-masu succession d'actions
(tabete, nonde, nemasu = je mange, bois et dors)
V-te mo ii desu permission (tabete mo ii deska = peut-on manger ?)
(tabako o sutte mo ii des = il est permis de fumer)

Formation de la forme en "te" à partir de l'infinitif familier (forme du dictionnaire):
Pour un verbe régulier V-ru+te sauf pour miru, mitte (regarder)

manger	tabemas	taberu	tabete
regarder	mimas	miru	mitte

Verbe semi-réguliers

acheter	kaimasu	kau	katte
aller	ikimasu	iku	itte
boire	nomimas	nomiru	nonde
écouter	kikimas	kiku	kiite
lire	yomimas	yomu	yonde
prendre	torimas	toru	totte

Un programme de conjugaison en japonais est beaucoup plus facile à réaliser qu'en français, deux problèmes se posent : analyser une forme verbale, et donc chercher dans un dictionnaire la ou les racines qui lui correspondent, et inversement, à partir d'un radical ou d'un infinitif, construire toutes les formes. En fait ce second problème suffit, et ici, seuls les passages entre les deux infinitifs et la forme en "te", présentent une difficulté.
Il semble qu'il y ait quelques règles phonétiques, on peut en effet distinguer les radicaux se finissant par a) ai, ui b) eri, ori, iri c) omi, obi, ubi d) ogi, iki, oki

```
(de fte (V) (fte1 (reverse (explodech V)))) ; on inverse le radical, la fonction est Infinitif → forme en "te"
(de fte1 (L) ; distingue les deux infinitifs
        (cond   ((equal (tete 4 L)'(u s a m)) (fte2 (nthcdr 4 L)))
                ((equal (tete 2 L)'(u r)) (fte2 (nthcdr 2 L)))  ))
```

Naturellement on peut traîter directement sur les mots, mais la programmation est plus aisée sur des listes.

```
(de fte2 (L) (cond
        ((eq L '(i a r a h)) 'harate)                                   ; cas a
        ((member (tete 2 L) '((i u)(i a))) (fte3 (cdr L) 'tte))
        ((equal L '(i r o)) 'oritte)                                    ; cas b
        ((equal (tete 3 L) '(i r e)) (fte3 (nthcdr 3 L) 'tte))
        ((equal (tete 2 L) '(i r)) (fte3 (nthcdr 2 L) 'tte))
        ((member (tete 3 L) '((i m o)(i b o)(i b u))) (fte3 (nthcdr 2 L) 'nde))  ; cas c
        ((equal (tete 3 L) '(i g o)) (fte3 (nthcdr 2 L) 'ite))          ; cas d
        ((equal L '(i k)) 'kite) ((equal L '(i k i)) 'itte) ((equal L '(i k o)) 'okite)
        ((equal (tete 2 L) '(i k)) (fte3 (nthcdr 2 L) 'te)) ((equal L '(i m)) 'mitte)
        (t (fte3 L 'te))  ))                                            ; cas général
(de fte3 (radicalinv suff) (catenate (implodech (reverse radicalinv)) suff)) ; recolle le radical et suffixe.
```

Exemples :

```
(fte 'taberu) = tabete          (manger)    (fte 'tabemasu) = tabete
(fte 'hairimasu) = haitte       (entrer)    (fte 'ikimasu) = itte          (aller)
(fte 'suimasu) = sutte          (fumer)     (fte 'yomimasu) = yonde        (lire)
(fte 'kaimasu) = katte          (acheter)   (fte 'yasubimasu) = yasunde    (se reposer)
(fte 'torimasu) = totte         (prendre)   (fte 'isogimasu) = isoite      (se dépêcher)
(fte 'kaerimasu) = katte        (rentrer)   (fte 'mimasu) = mitte          (regarder)
(fte 'shimasu) = shite          (faire)
```

10-35° Semi-traduction du japonais au français On donne la liste suivante de suffixes japonais, grâce à eux il est facile d'obtenir sans ambiguité le sens d'une phrase :
-DA = affirmation familière (watashi seito da = j'suis élève)
-DE indique le moyen, par (kurumade ikimaska = irez-vous en voiture?)
-E direction
-I = adjectif verbal (takai = c'est cher, yasui = c'est bon marché)
-JA ARIMASEN = négation familière (anataha shikku ja arimasen = t'es pas chic)
-KAN pendant (itchijikan = pendant une heure)
-KARA provenance (dokokara kimashitaka = d'où venez-vous?, gyûjikara des = c'est à partir de 10 h)
-KATTA = forme polie affirmative passée (takakatta = c'était cher)
-KU = forme polie négative (takaku arimasen = ce n'est pas cher)
-KUN = suffixe de respect, mais plus intime
-MADE jusqu'à (Yokohamamade ikimas = je vais jusqu'à Yokohama)
-MATAWA ou bien
-MO aussi (Nihonmo suki des = j'aime aussi le Japon, daremo kimasen deshita = personne n'est venu, nanimo kaimasen deshita = je n'ai rien acheté)
-NAGARA gérondif (utainagara = en chantant)
-NE = n'est-ce-pas, confirmation attendue
(kirei na onna no ko desne = c'est une jolie fille, n'est-ce-pas?)
-NI point d'arrivée, vers... (dokoni imashitaka = où étiez vous?)
tabeni ikimas = aller manger
-NO préposition "de" (watashi no des = c'est le mien, niwasan no niwa = le jardin de mr Niwa)
-O complément d'objet (gohan o tabemashitaka = avez vous mangé?, tabako o suimasuka = fumez-vous?)
-SAN = monsieur (Niwasan = monsieur Niwa), madame, mademoiselle
-SHI = monsieur utilisé par écrit
-TAI suffixe de désir (tabetai = je veux manger) hana o kaitai = je souhaite acheter des fleurs
kôhii o nomitaimasen = je ne veux pas boire de café, tegami o kakitai = je veux écrire une lettre
anatawa tegami o kakitaika = désires-tu écrire une lettre
-TO accompagnement, avec (nekoto inu = chat et chien)
-YA indique également "avec", mais lorsqu'il y a une énumération
-YO = renforcement (kinô wa atsukatta desuyo = il faisait vraiment chaud hier) (muda desuyo = c'est tout à fait inutile)
-YORI comparatif (Tôkyô wa Pariyori o okii des = Tokyo est plus grand que Paris, kanojowa watashiyori kirei des = elle est plus belle que moi)

Le japonais se prête particulièrement bien (moins que l'espéranto toutefois), à l'analyse automatique des phrases. Le problème informatique, aussi bien dans le domaine de la traduction automatique que dans tout système de questions-réponses, est en effet d'arriver à une "compréhension" minimale du texte, suffisante pour pouvoir aiguiller la machine vers une recherche quelconque ou vers la génération d'une phrase en réponse.
On s'oriente à l'heure actuelle dans l'analyse d'une phrase, sur la voie de la constitution d'un "graphe conceptuel" où sont notés les principaux éléments de la phrase, leur fonction syntaxique, et où leur sont reliés toutes sortes d'éléments circonstanciels visant à infléchir le sens de ces éléments (adjectifs, adverbes, mais aussi subordonnées ...)
Le programme ci-dessous tire profit des suffixes afin d'opérer une analyse de la phrase japonaise, il est bien sûr très limité, mais il suffirait de prendre en compte d'autres suffixes, des préfixes, et aussi les mots isolés ou adverbes et les suffixations en chaîne, pour en faire un système déjà performant. Le problème des ambiguités étant plus difficile à traiter.
Une phrase donnée est une liste L de mots japonais et de particule détachées, la fonction lisp d'analyse donne une liste de mots français précédés de leur rôle au sein de cette phrase.

```
(de trad (X) (trad1 X DICO)) ; fournit la traduction des racines rencontées.

(de trad1 (X L) (cond     ((null L) (concat 'mot-inconnu- X))
                          ((eq X (caar L)) (cadar L))
                          (t (trad1 X (cdr L)))))
(de analyse (L) (ana1 (car L) (cadr L) (cddr L) nil))

(de ana2 (L R) (cond      ((null L) R)
                          ((null (cdr L)) (mcons 'rôle-inconnu (trad(car L)) R))
                          (t (ana1 (car L) (cadr L) (cddr L) R))))
```

```
(de ana1 (X Y L R) (cond ; soit X est une racine et Y un suffixe, soit ce sont deux suffixes
                         ((or (eq Y 'wa) (eq Y 'ga)) (ana2 L (mcons 'sujet- (trad X) R)))
                         ((eq Y 'o) (ana2 L (mcons 'complément- (trad X) R)))
                         ((eq Y 'to) (ana2 L (mcons 'accompagnant- (trad X) R)))
                         ((eq Y 'ni) (ana2 L (mcons 'vers- (trad X) R)))
                         ((eq Y 'no) (ana2 L (mcons '-de/du  (trad X) R)))
                         ((eq Y 'de) (ana2 L (mcons 'au-moyen-de- (trad X) R)))
                         ((eq Y 'ka) (ana2 L (mcons 'interrogation (trad X) R)))
                         ((eq Y 'e) (ana2 L (mcons 'dans- (trad X) R)))
                         ((eq Y 'made) (ana2 L (mcons 'jusqu-a- (trad X) R)))
                         ((eq Y 'ji) (ana2 L (mcons 'heure- (trad X) R)))
                         ((eq Y 'mashita) (ana2 L (mcons 'verbe-passé- (trad X) R)))
                         ((eq Y 'masen) (ana2 L (mcons 'négation (trad X) R)))
                         ((eq Y 'mo) (ana2 L (mcons 'aussi- (trad X) R)))
                         ((eq Y 'masu) (ana2 L (mcons 'verbe- (trad X) R))) ; à compléter!
                         (t (ana2 L (cons 'rôle-inconnu- (trad X) R)))))

(set 'DICO '((neko chat) (inu chien) (issu chaise) (utchi maison) (iki aller) (watashi moi) (anata toi) (niwa jardin)
(densha train)))
```

Exemples d'utilisation :

```
(analyse '(neko to inu wa niwa no utchi ni iki masu))
= (verbe- aller vers- maison -de/du jardin sujet- chien accompagnant- chat)
(analyse '(anata wa watashi mo utchi made iki masu))
= (verbe- aller jusqu-a- maison aussi- moi sujet- toi)
(analyse '(watashi no neko wa niwa e iki mashita))
= (verbe-passé- aller dans- jardin sujet- chat -de/du moi)

(analyse '(watashi wa inu to Tôkyô ni densha de iki masu))
= (verbe- aller au-moyen-de- train vers- mot-inconnu-tôkyô accompagnant- chien sujet- moi)
```

10-36° Conjugaison des verbes français : Le programme doit d'abord pouvoir donner toutes les formes verbales pour un infinitif donné : participes passé et présent, présent, imparfait, futur, conditionnel, passé-simple, impératif et subjonctifs sous forme d'un tableau bien présenté à l'écran.
Tout le problème réside dans la recherche d'une bonne représentation des racines et terminaisons. Chercher 4 listes de terminaisons des verbes au présent, 2 au passé-simple, sur le modèle de celle du futur IF = (ai as a ons ez ont) ou du conditionnel IC = (ais ais ait ions iez aient), puis chercher une repésentation des verbes de façon à rendre compte des différentes altérations.
Dans l'autre sens, le programme doit pouvoir analyser une forme verbale pour retrouver l'infinitif, par exemple "suis" doit être analysé comme verbe être présent singulier première personne et comme verbe suivre présent singulier première personne. Il doit en outre pouvoir reconnaître des formes non attestées mais compréhensibles telles que "disez", "pleuvu" ou "faisez".
On demande également une reconnaissance des adjectifs et des participes passés accordés.
Le fichier de verbes contiendra au minimum : parler, sentir, finir, connaître, assaillir, servir, dormir, cueillir, mourir, tenir, conclure, fuir, voir, croire, pourvoir, recevoir, devoir, savoir, boire, pouvoir, mouvoir, valoir, vouloir, asseoir, rendre, prendre, mettre, plaire, vaincre, suivre, vivre, dire, écrire, cuire, faire, aller, avoir, être, etc...
On tiendra compte ensuite des verbes tels que jeter, appeler, employer, appuyer ou manger, qui présentent des étrangetés orthographiques. On pourra aussi donner des indications sur les formes non attestées comme "traire", "pleuvoir" ou "braire" à certaines personnes ou temps, comment traiter ces manques ? Comment traiter par ailleurs les formes équivalentes acceptées pour "asseoir" ou celles de l'ancien français ?

Remarque : au présent, hormis les verbes irréguliers tels que être ou croire, on peut réunir 4 types de terminaisons V1 = (e es e ons ez ent) V2 = (x x t ons ez ent) V3 = (s s t ons ez ent) V4 = (s s _ ons ez ent), et au passé-simple deux seulement S1 = (ai as a âmes âtes èrent) S2 = (s s t mes tes rent)