10. LES ENTREES / SORTIES (PREMIERE PARTIE)

Nous n'avons envisagé qu'une seule manière de communiquer avec notre machine : soit en évaluant l'appel d'une fonction, soit en évaluant une variable. Les seules valeurs que nous puissions donner à des fonctions sont celles données à l'appel, et les seules valeurs apparaissant sur l'écran de votre terminal, sont celles résultant d'une évaluation demandée au niveau de l'interaction avec la machine, c'est-à-dire : au *top-level*. Eventuellement, on aimerait également pouvoir imprimer des résultats intermédiaires, ou d'autres informations, voire même écrire des programmes permettant un dialogue avec la machine. Les fonctions permettant d'imprimer ou de lire au milieu de l'évaluation (ou de l'exécution) d'un programme sont appelées : les fonctions d'*entrées/sorties*. C'est de la définition et de l'utilisation de ces fonctions que traitera ce chapitre.

10.1. LA FONCTION GENERALE D'IMPRESSION : PRINT

PRINT est la fonction LISP utilisée pour imprimer des valeurs. Cette fonction est la deuxième que nous rencontrons, qui est non seulement utilisée pour la valeur qu'elle ramène après évaluation de son argument mais aussi à cause de l'*effet-de-bord* qu'elle provoque.[1] Bien entendu, cet effet de bord est l'impression de la valeur des arguments.

Voici la définition de la fonction **PRINT** :

$$\textbf{(PRINT } arg_1\ arg_2 \ldots arg_n\textbf{)} \quad \rightarrow \quad arg_n$$

PLUS : impression des valeurs des différents arguments

PRINT est donc une fonction à nombre quelconque d'arguments. A l'appel chacun des arguments sera évalué *et* imprimé sur le périphérique standard de sortie (écran de visualisation, papier du télétype, fichier disque ou bande magnétique, dépendant du contexte dans lequel vous vous trouvez).[2] La valeur du dernier argument sera ensuite ramenée en valeur, de la même manière que toutes les autres fonctions ramènent une valeur utilisable dans la suite du calcul.

Voici deux exemples simples d'utilisation de cette fonction :

```
? (PRINT 1 2 3 4)
  1 2 3 4                              ; l'impression ;
= 4                                    ; la valeur ramenée ;
? (PRINT (CONS 'VOILA '(CA MARCHE)))
  (VOILA CA MARCHE)                    ; l'impression ;
= (VOILA CA MARCHE)                    ; la valeur ramenée ;
```

A la fin de l'impression, **PRINT** effectue automatiquement un saut à la ligne suivante.

---

[1] Les deux autres fonctions à *effet de bord* que nous connaissons sont la fonction **PUT** (ou **PUTPROP**) et la fonction **REMPROP**, qui, d'une part ramènent une valeur, d'autre part provoquent l'effet de bord d'une modification de la P-liste.

[2] Nous verrons plus tard comment on peut changer de périphérique standard.

En LE_LISP, l'impression des arguments n'est pas séparée par des espaces. Ainsi, le premier exemple ci-dessus imprime **1234**, mais ramène toutefois **4** en valeur. Afin d'obtenir quand même des espaces, il faut les donner comme chaînes de caractères explicites. Par exemple, pour obtenir le même résultat, il faut donner l'expression suivante :

```
? (PRINT 1 " " 2 " " 3 " " 4)
  1 2 3 4
= 4
```

Les guillemets entourent des chaînes de caractères. Ici les trois chaînes de caractères sont réduites au caractère espace. Dans la suite de ce chapitre nous verrons les chaînes en plus de détail.

Voici à présent une petite fonction qui imprime tous les éléments d'une liste :

```
(DE IMPRIME-TOUS (L)
    (IF (NULL L) '(VOILA Y EN A PUS)
        (PRINT (CAR L))
        (IMPRIME-TOUS (CDR L))))
```

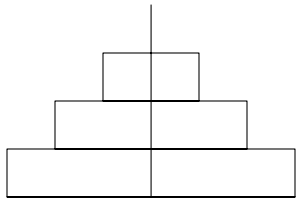
et voici quelques appels de cette fonction :

```
?  (IMPRIME-TOUS '(DO RE MI FA SOL))
   DO
   RE
   MI
   FA
   SOL
=  (VOILA Y EN A PUS)

?  (IMPRIME-TOUS '(UN DEUX TROIS))
   UN
   DEUX
   TROIS
=  (VOILA Y EN A PUS)
```

Après chaque impression la machine effectue un *retour-chariot* et un *line-feed* (ce qui veut dire en français : elle remet le curseur au début de la ligne suivante).

Pour donner un exemple moins trivial d'utilisation de la fonction **PRINT**, voici une solution au problème des *tours de Hanoi.*[3] Le problème est le suivant : vous avez une aiguille sur laquelle sont empilés des disques de diamètres décroissants, comme ceci :



[3] D'après un mythe ancien, quelques moines dans un temple de l'orient font écouler le temps en transférant 64 disques d'or d'une aiguille vers une autre, en suivant les règles données. L'existence de l'univers tel que nous le connaissons sera achevée, dès l'instant où ils auront fini cette tâche.

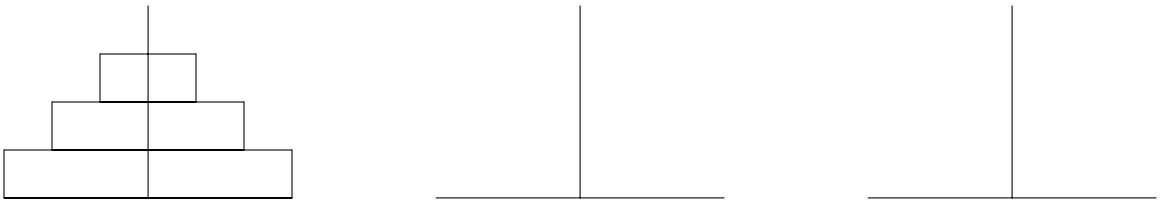
D'ailleurs, quel sera l'âge total de l'univers si le déplacement d'un disque prend une seconde ?

A proximité vous avez deux autres aiguilles. Il faut alors transporter les disques de la première aiguille vers la deuxième, en se servant éventuellement de la troisième aiguille comme lieu de stockage temporaire et en prenant en compte, toutefois, les restrictions suivantes :

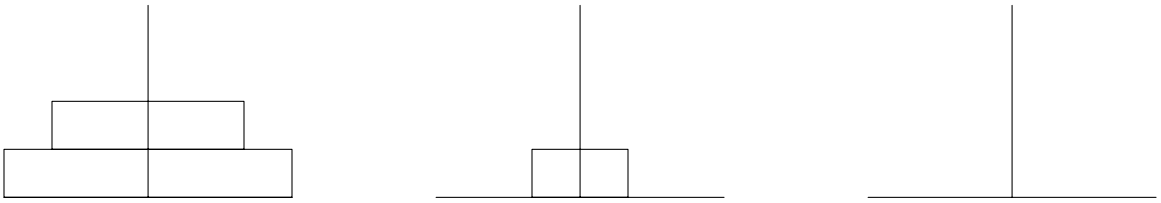
1. on ne peut déplacer qu'un disque à la fois
2. à aucun instant, un disque ne peut se trouver sur un disque de diamètre inférieur.

Regardez, voici la solution pour 3 disques :

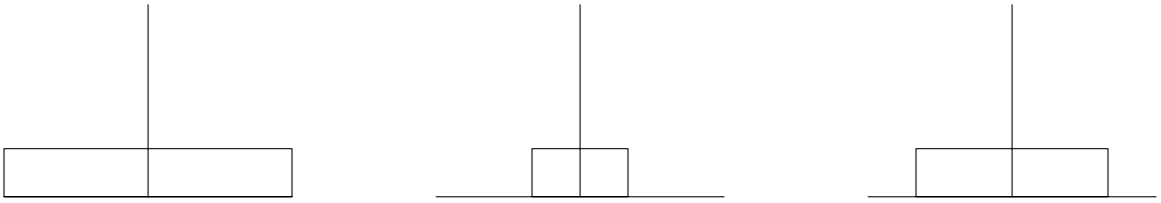
1. état initial



2. premier déplacement:

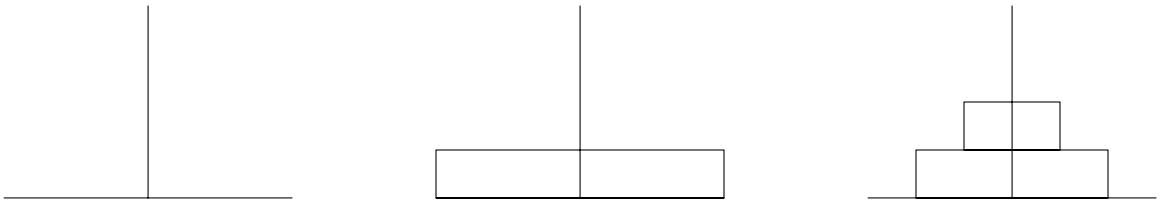


3. deuxième déplacement:

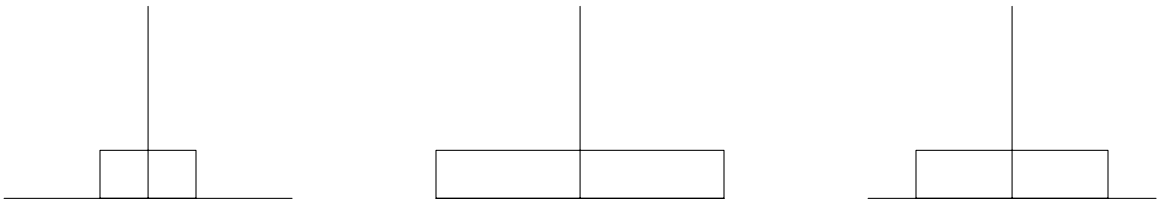


4. troisième déplacement :

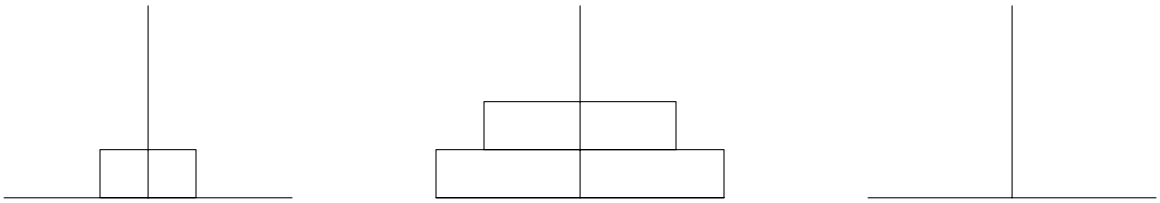
5. quatrième déplacement :



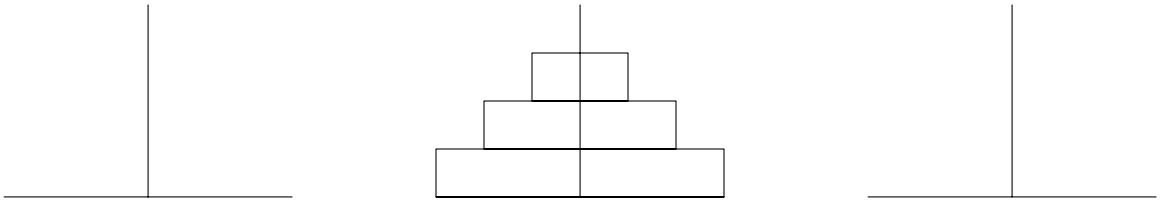
6. cinquième déplacement :



7. sixième déplacement :



8. état final :



Dans la quatrième image on a, sur l'aiguille servant d'intermédiaire, une configuration similaire à celle de l'état initial. La différence entre les deux états est que le disque le plus grand ne fait pas partie de la configuration sur l'aiguille intermédiaire. C'est l'analyse de cette observation qui nous livre la solution au problème. Observez, les trois premiers déplacements n'ont fait rien d'autre que de reconstruire, sur l'aiguille *intermédiaire*, une *tour de Hanoi* comportant un disque de moins que la tour originale. L'aiguille *arrivée* étant libre à cet instant, on peut y placer maintenant le disque de diamètre maximal qui se trouve tout seul sur l'aiguille de *départ*. Il suffit de recommencer les opérations avec, toutefois, une permutation du *rôle* des aiguilles : l'aiguille qui servait au début d'intermédiaire est évidemment devenue l'aiguille de départ (puisque tous les disques à déplacer s'y trouvent). Par contre, l'aiguille originale de départ peut servir maintenant d'aiguille intermédiaire.

La généralisation de cette observation nous livre l'algorithme suivant :

```
(DE HANOI (N DEPART ARRIVEE INTERMEDIAIRE)
   (IF (= N 0) '(ENFIN, C EST FINI)
      (HANOI (1- N) DEPART INTERMEDIAIRE ARRIVEE)
      (PRINT 'DISQUE N 'DE  DEPART 'VERS ARRIVEE)
      (HANOI (1- N) INTERMEDIAIRE ARRIVEE DEPART)))
```

et voici l'exemple de quelques appels :

```
? (HANOI 2  'DEPART 'ARRIVEE 'INTERMEDIAIRE)
  DISQUE 1 DE DEPART VERS INTERMEDIAIRE
  DISQUE 2 DE DEPART VERS ARRIVEE
  DISQUE 1 DE INTERMEDIAIRE VERS ARRIVEE
= (ENFIN, C EST FINI)


? (HANOI 3 'DEPART 'ARRIVEE 'INTERMEDIAIRE)
  DISQUE 1 DE DEPART VERS ARRIVEE
  DISQUE 2 DE DEPART VERS INTERMEDIAIRE
  DISQUE 1 DE ARRIVEE VERS INTERMEDIAIRE
  DISQUE 3 DE DEPART VERS ARRIVEE
  DISQUE 1 DE INTERMEDIAIRE VERS DEPART
  DISQUE 2 DE INTERMEDIAIRE VERS ARRIVEE
  DISQUE 1 DE DEPART VERS ARRIVEE
= (ENFIN, C EST FINI)


? (HANOI 4 'DEPART 'ARRIVEE 'INTERMEDIAIRE)
  DISQUE 1 DE DEPART VERS INTERMEDIAIRE
  DISQUE 2 DE DEPART VERS ARRIVEE
  DISQUE 1 DE INTERMEDIAIRE VERS ARRIVEE
  DISQUE 3 DE DEPART VERS INTERMEDIAIRE
  DISQUE 1 DE ARRIVEE VERS DEPART
  DISQUE 2 DE ARRIVEE VERS INTERMEDIAIRE
  DISQUE 1 DE DEPART VERS INTERMEDIAIRE
  DISQUE 4 DE DEPART VERS ARRIVEE
  DISQUE 1 DE INTERMEDIAIRE VERS ARRIVEE
  DISQUE 2 DE INTERMEDIAIRE VERS DEPART
  DISQUE 1 DE ARRIVEE VERS DEPART
  DISQUE 3 DE INTERMEDIAIRE VERS ARRIVEE
  DISQUE 1 DE DEPART VERS INTERMEDIAIRE
  DISQUE 2 DE DEPART VERS ARRIVEE
  DISQUE 1 DE INTERMEDIAIRE VERS ARRIVEE
= (ENFIN, C EST FINI)
```

Analysez bien cette fonction et construisez-en l'arbre d'appel : elle est intéressante, pas seulement à cause de l'utilisation de la fonction **PRINT**.

10.2. LES CHAINES DE CARACTERES

Jusqu'à maintenant nous ne connaissons que trois types d'objets LISP : les *listes*, les *nombres* et les *atomes*. Afin de pouvoir rendre les sorties les plus agréables possible (pour l'oeil), nous avons éventuellement besoin d'imprimer des caractères spéciaux tels que des *espaces*, des *points* ou tout autre caractère jusqu'à maintenant inaccessible à cause de son rôle de *séparateur* qu'il possède normalement en LISP. Les *chaînes-*

de-caractères sont des objets LISP permettant l'accès à ces caractères et aucun contrôle n'y est effectué quant au rôle syntaxique que pourrait avoir normalement des caractères pouvant faire partie d'une *chaîne* (forme courte pour *chaîne de caractère*).

Syntaxiquement, toute chaîne de caractères s'écrit entre des guillemets (") :

chaîne de caractère ::= "*suite-de-caractères*"

où une *suite-de-caractères* peut être constituée de n'importe quels caractères.

Voici quelques exemples de chaînes de caractères bizarres mais tout à fait correctes :

"aBcDeFgHiJk"
"((((((((((("
".<>.<>."
"! _$%_&'()_@"

Les chaînes de caractères sont des objets LISP de type *atome*. Le prédicat **ATOM** appliqué à une chaîne de caractères, ou à une variable ayant comme valeur une chaîne de caractères, ramène donc *vrai*, i.e.: **T**, comme dans l'exemple suivant :

(ATOM "aBc.cBa") → **T**

Afin de néanmoins pouvoir distinguer entre des chaînes de caractères et d'autres atomes, LISP fournit le prédicat **STRINGP** défini comme suit :

(STRINGP *e***)**
→ **T** en VLISP, la chaîne *e* en LE_LISP
si l'argument *e* est une chaîne de caractères
→ **()** dans tous les autres cas

Voici quelques exemples d'applications de ce prédicat : d'abord en VLISP :

(STRINGP "LISP is beautiful") → **T**
(STRINGP 'CUBE1) → **NIL**
(STRINGP (CAR '("aie" est-ce vrai?))) → **T**
(STRINGP '("tiens" "voila" "du boudin")) → **NIL**

ensuite en LE_LISP :

(STRINGP "LISP is beautiful") → **"LISP is beautiful"**
(STRINGP 'CUBE1) → **()**
(STRINGP (CAR '("aie" est-ce vrai?))) → **"aie"**
(STRINGP '("tiens" "voila" "du boudin")) → **()**

NOTE

Remarquez qu'il ne faut pas *quoter* les chaînes de caractères. Elles sont des *constantes* comme l'atome **T** et l'atome **NIL** et peuvent être considérée comme des atomes qui ont eux-mêmes comme valeur.

Voici quelques exemples d'utilisation de la fonction **PRINT** avec des chaînes de caractères :

```
? (PRINT 1 2 3)                          ; normalement ;
  1 2 3                                  ; en LE_LISP : 123 ;
= 3

? (PRINT 1 "      " 2 "      " 3)        ; avec des espaces ;
  1       2       3
= 3

? (PRINT 1 "........." 2 "........." 3)   ; voyez vous l'utilité ? ;
  1 ......... 2 ......... 3
= 3
```

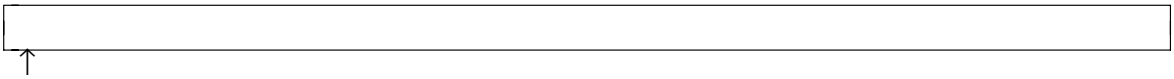
Pour l'instant rappelons-nous juste que chaque fois que nous aurons besoin d'imprimer des caractères spéciaux, nous utiliserons des chaînes de caractères.

10.3. LES FONCTIONS PRIN1, TERPRI ET PRINCH

Les impressions ou, plus généralement, les sorties s'effectuent normalement en deux étapes : d'abord on copie ce qu'on veut imprimer dans une zone spéciale de la mémoire de l'ordinateur, zone généralement appelée *buffer* (ou *tampon*), ensuite on envoie cette zone mémoire, ce buffer, vers le périphérique de sortie : on *vide le tampon*.

La fonction **PRINT** combine les deux étapes : elle remplit le tampon de sortie et ensuite elle l'envoie vers le périphérique. Mais il est également possible de séparer les étapes grâce aux fonctions **PRIN1**, en VLISP, ou la fonction **PRIN**, en LE_LISP, et **TERPRI**. **PRIN1** est une fonction qui remplit le tampon, on dit qu'elle *édite* le tampon. Cette fonction procède à l'impression effective seulement si, au fur et à mesure du remplissage, le tampon a été complètement rempli. Tant que ce n'est pas le cas, on ne voit rien apparaître sur le terminal (si le terminal est le périphérique standard de sortie). Pour vider le tampon, il est indispensable d'appeler la fonction **TERPRI**, qui, après l'avoir vidé repositionne un pointeur (invisible) au début du tampon. Si le tampon est vide à l'appel de la fonction, **TERPRI** n'imprime rien et passe à la ligne.

Afin d'être un peu plus explicite, supposons que nous ayons le tampon suivant :



avec un pointeur vers le début du tampon, représenté ici par la petite flèche en dessous du rectangle représentant le tampon. Si l'on évalue l'appel

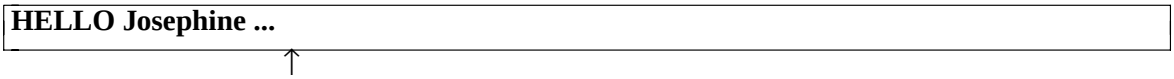
(PRIN1 'HELLO) *; en* VLISP
(PRIN 'HELLO) *; en* LE_LISP

l'effet sera le suivant :

La suite des caractères constituant l'atome **HELLO** a été introduite en tête de tampon et le pointeur a avancé pour pointer maintenant sur la première position vide. L'appel

(PRIN1 "Josephine" "...") *; en* VLISP
(PRIN1 "Josephine" "...") *; en* LE_LISP

modifiera l'état du tampon comme suit :



Aucune impression effective n'a encore eu lieu. **PRIN1** remplit son tampon de sortie, c'est tout. Pour l'imprimer effectivement, il est nécessaire d'appeler explicitement la fonction **TERPRI** :

(TERPRI)

ce qui entrainera l'apparition de la ligne suivante :

HELLO Josephine ...

le tampon retrouvant son état initial :

c'est-à-dire : le tampon est vidé et son pointeur pointe vers la première position.

Chaque fois que l'on utilise la fonction **PRIN1**, il faut faire suivre tous les appels de cette fonction par au moins un appel de la fonction **TERPRI**.

Formellement, **PRIN1** est défini comme suit :

(PRIN1 arg_1 arg_2 ... arg_n**)** $\rightarrow$ arg_n

PLUS : remplissage du tampon de sortie avec les valeurs des différents arguments.

PRIN1 est donc, comme **PRINT**, une fonction à nombre quelconque d'arguments. **PRIN1** évalue tous les arguments, les introduit dans le tampon de sortie, et ramène la valeur du dernier argument (arg_n) en valeur. **PRIN1** ne procède à l'impression effective que si le tampon est complètement rempli, et qu'aucun autre élément ne peut être introduit.

(TERPRI $\{n\}$**)** $\rightarrow$ n ou **NIL**

TERPRI possède un argument optionnel qui, s'il existe, doit être un nombre. **TERPRI** imprime le tampon de sortie. Si le tampon était vide l'opération est identique à un saut de ligne. Si un argument n est présent, la fonction **TERPRI** vide le tampon (c'est-à-dire : imprime le contenu du tampon) et effectue *n-1 sauts de lignes* supplémentaires. **TERPRI** ramène en valeur la valeur de son argument, s'il est présent, sinon **NIL**.

Les fonctions **PRINT** et **PRIN1** séparent les différents éléments à imprimer par un espace. Eventuellement il peut arriver que l'on ait besoin d'imprimer *sans* espaces. La fonction **PRINCH**, acronyme pour *PRIN*1 *CH*aracter, possède deux arguments : un caractère à imprimer et un nombre indiquant combien de fois on veut imprimer ce caractère. Les *éditions* successives de ce caractère ne sont pas séparées par des espaces, mais 'collées' les unes aux autres. La valeur ramenée par un appel de la fonction **PRINCH** est le caractère donné en premier argument. Voici quelques appels exemples de cette fonction :

```
? (PRINCH "~" 5)
~~~~= ~
```
; C;a imprime 5 fois le caractère '~' et le ramène en valeur. Ici, la valeur est imprimée sur la même ligne que les 5 impressions du caractère, puisque la fonction **PRINCH** *ne procède pas à l'impression, elle remplit juste le tampon. L'impression est effectuée par le top-level de LISP ;*

```
? (PRINCH "." 10)
.........= .
```
; le dernier point est celui qui est ramené en valeur ;

```
? (PRINCH 1)
1= 1
```
; s'il n'a pas de deuxième argument, la valeur 1 est supposée par défaut ;

Afin de donner un exemple d'utilisation de ces fonctions, voici maintenant la définition d'une fonction de *formatage* - comme en FORTRAN. Cette fonction a deux arguments, une liste d'éléments à imprimer et une liste de colonnes donnant les tabulations auxquelles les éléments successifs doivent être imprimés.[4]

```
(DE FORMAT (ELEMENTS COLONNES)
  (TERPRI)        ; pour une ligne vide ;
  (LET ((N 0)(ELEMENTS ELEMENTS)(COLONNES COLONNES))
       (IF (NULL ELEMENTS) ()      ; c'est fini ;
           (IF (NULL COLONNES) ()  ; c'est fini ;
               (SELF (+ (LET ((N N))(COND
                                    ((= N (CAR COLONNES))
                                         (PRIN1 (CAR ELEMENTS)) N)
                                    (T (PRINCH " " 1)
                                    (SELF (1+ N)))))
                                    ; pour la place de l'élément ;
                        (PLENGTH (CAR ELEMENTS)))
                     (CDR ELEMENTS)
                     (CDR COLONNES))))))
```

La construction :

```
(IF (NULL ELEMENTS) ()
    (IF (NULL COLONNES) ()
        . . .
```

précise que le calcul sera terminé dès que la liste **ELEMENTS** ou la liste **COLONNES** est vide. Pour tester si au moins *un* test d'un ensemble de tests est vrai, LISP fournit la fonction **OR** (anglais pour *ou*) qui est définie comme suit :

(OR $test_1$ $test_2$ ... $test_n$**)**

→ le résultat du premier *test* qui ne s'évalue pas à **NIL**

→ **NIL** si tous les $test_i$ s'évalue à **NIL**

Nous pouvons donc réécrire ce test comme :

```
(IF (OR (NULL ELEMENTS) (NULL COLONNES)) ()
    . . .
```

[4] La fonction **PLENGTH** qui calcule le nombre de caractères d'un nom d'atome sera vue au paragraphe suivant.

La fonction **OR** correspond donc tout à fait à un *ou logique*.

Nous avons également la fonction **AND** (anglais pour *et*) qui correspond à un *et logique*, qui teste donc la vérification de tous les tests d'un ensemble de tests. Voici sa définition :

(AND $test_1$ $test_2$ . . . $test_n$**)**

→ $test_n$ si aucun des $test_i$ ne s'évalue à **NIL**

→ **NIL** si au moins un des $test_i$ s'évalue à **NIL**

La fonction auxiliaire **F1** prendra comme arguments une liste contenant des sous-listes, chacune donnant les éléments à imprimer dans le format indiqué par le deuxième argument. La voici :

```
(DE F1 (LISTE-D-ELEMENTS COLONNES)
  (IF (NULL LISTE-D-ELEMENTS) (TERPRI)
     (FORMAT (CAR LISTE-D-ELEMENTS) COLONNES)
     (F1 (CDR LISTE-D-ELEMENTS) COLONNES)))
```

et voici, finalement, quelques exemples des impressions que cette fonction produit :

```
(F1 '((ANGLAIS FRANCAIS ALLEMAND)
    (---------- ---------- ----------)
    (COMPUTER ORDINATEUR COMPUTER)
    (STACK PILE KELLER)
    (MEMORY MEMOIRE SPEICHER)
    (DISPLAY ECRAN BILDSCHIRM)
    (KEYBOARD CLAVIER TASTATUR))
   '(5 25 45))

   →

ANGLAIS              FRANCAIS             ALLEMAND
-------------------- -------------------- --------------------
COMPUTER             ORDINATEUR           COMPUTER
STACK                PILE                 KELLER
MEMORY               MEMOIRE              SPEICHER
DISPLAY              ECRAN                BILDSCHIRM
KEYBOARD             CLAVIER              TASTATUR
= NIL
```

Encore un exemple : un tableau de prix mensuels de deux réseaux français de télécommunications :

(F1 '((COUTS TELEPHONE LIAISON CADUCEE TRANSPAC)
(RACCORDEMENT 4800F 3200F 3200F 4000F)
(ABONNEMENT 189F 4003F 3130F 1810F)
(UTILISATION 13536F 0F 4060F 117F))
'(4 20 30 40 50))

→

COUTS	**TELEPHONE**	**LIAISON**	**CADUCEE**	**TRANSPAC**
RACCORDEMENT	**4800F**	**3200F**	**3200F**	**4000F**
ABONNEMENT	**189F**	**4003F**	**3130F**	**1810F**
UTILISATION	**13536F**	**0F**	**4060F**	**117F**

= NIL

10.4. L'ACCES AUX NOMS DES ATOMES

Comme en physique, en LISP l'atome n'est indivisible qu'apparemment. Correspondant aux particules élémentaires constituant l'atome physique, les noms des atomes LISP sont constitués de caractères. En VLISP, ces caractères sont rendus accessibles par la fonction **EXPLODE** et la fusion d'un atome à partir de particules élémentaires est effectuée à l'aide de la fonction **IMPLODE**. En LE_LISP, les deux fonctions s'appellent respectivement **EXPLODECH** et **IMPLODECH**.

Allons pas à pas : d'abord la fonction **EXPLODE** qui est définie comme suit :

(EXPLODE *atome***)** → une liste de caractères correspondant à la suite de caractères composant la forme externe donc imprimable, de la valeur de *atome*.

La fonction **EXPLODE** prend donc un argument, l'évalue, et, si la valeur est un atome, elle livre la liste de caractères constituant cet atome, sinon, si la valeur est une liste, elle livre **NIL**. Voici quelques exemples d'utilisation de cette fonction :

```
? (EXPLODE 'ATOM)
= (A T O M)

? (EXPLODE 123)
= (1 2 3)

? (EXPLODE (CAR '(IL S EN VA)))
= (I L)

? (CDR (EXPLODE 'UNTRESLONGAT))))
= (N T R E S L O N G A T)

? (LENGTH (EXPLODE 'CIEL-MON-MARI))
= 13
```

La fonction LE_LISP **EXPLODECH** est plus générale : elle prend en argument une expression LISP quelconque et livre la suite de caractères composant cette expression. Ainsi, si vous appelez :

(EXPLODECH '(CAR '(A B)))

le résultat est la liste :

(|(| C A R | | |'| |(| A | | B |)| |)|)

Les caractères spéciaux sont entourés par deux '|'. En LE_LISP, c'est un moyen de quoter les caractères, pour faire abstraction de leurs rôles particuliers.

Déterminer la longueur de la liste résultant d'un appel de la fonction **EXPLODE**, revient au même que de calculer le nombre de caractères constituant le *P-name* d'un atome. Cette indication peut être très utile si l'on construit manuellement des belles impressions, comme nous l'avons fait dans la fonction **FORMAT** afin de calculer la position exacte du pointeur du tampon de sortie. Etant donné que nous avons relativement souvent besoin de la longueur du P-name d'un atome, LISP a prévu une fonction spéciale à cet effet : c'est la fonction **PLENGTH**, qui pourrait être définie comme :

```
(DE PLENGTH (ELE)
    (IF (ATOM ELE)(LENGTH (EXPLODE ELE)) 0))
```

Voici quelques appels de la fonction **PLENGTH** :

(PLENGTH 'HOP)	$\rightarrow$	**3**
(PLENGTH 'MARYLIN)	$\rightarrow$	**7**
(PLENGTH (CAR '(A B C)))	$\rightarrow$	**1**

Naturellement, si nous pouvons séparer le nom d'un atome en une suite de caractères, nous aimerions éventuellement également pouvoir construire un atome à partir d'une liste de caractères. Cette opération, l'inverse de la fonction **EXPLODE**, est effectuée par la fonction **IMPLODE**.

IMPLODE prend comme argument une liste de caractères et livre en résultat un atome dont le nom est constitué par ces caractères. Ainsi nous avons la relation suivante :

(IMPLODE (EXPLODE *atome*)**)** $\rightarrow$ *atome*

Naturellement, de même que la fonction LE_LISP **EXPLODECH**, la fonction inverse **IMPLODECH** de LE_LISP admet comme argument une suite de caractères quelconque correspondant aux caractères constituants une expression LISP, et livre comme résultat l'expression LISP correspondante. Nous avons donc, en LE_LISP la relation plus générale :

(IMPLODECH (EXPLODECH *expression*)**)** $\rightarrow$ *expression*

Regardons quelques exemples d'utilisation de cette fonction :

```
? (IMPLODE '(A T O M))
= ATOM

? (IMPLODE '(C U R I E U X))
= CURIEUX

? (IMPLODE (APPEND (EXPLODE 'BON)(EXPLODE 'JOUR)))
= BONJOUR

? (IMPLODE (EXPLODE 'IDENTITE))
= IDENTITE
```

IMPLODE est un moyen fort commode pour créer des nouveaux atomes, ce qui peut - surtout dans des

applications concernant le langage naturel - être bien utile de temps à autre.

La suite de ce paragraphe sera un petit programme pour conjuguer des verbes français réguliers des premiers groupes, c'est-à-dire : des verbes réguliers se terminant en 'er', 'ir' et 're'.

; la fonction **CREE-MOT** *crée un nouveau mot à partir d'une racine et d'une terminaison ;*

```
(DE CREE-MOT (RACINE TERMINAISON)
  (IF (NULL TERMINAISON) RACINE
    (IMPLODE (APPEND (EXPLODE RACINE)
                     (EXPLODE TERMINAISON)))))
```

; la fonction **RACINE** *trouve la racine d'un verbe, simplement en y enlevant les deux derniers caractères ;*

```
(DE RACINE (VERBE)
  (IMPLODE (REVERSE (CDR (CDR (REVERSE (EXPLODE VERBE)))))))
```

; la fonction **TYPE** *ramène l'atome constitué des deux derniers caractères de l'atome donné en argument ;*

```
(DE TYPE (VERBE)
   (LET ((AUX (REVERSE (EXPLODE VERBE))))
      (IMPLODE (CONS (CADR AUX)
                     (CONS (CAR AUX) ())))))
```

; **CONJUGUE** *est la fonction que l'utilisateur va appeler. Elle prépare les arguments pour la fonction auxiliaire* **CONJ1** *;*

```
(DE CONJUGUE (VERBE)
  (TERPRI)                                  ; pour faire plus joli ;
  (CONJ1 (RACINE VERBE)                       ; trouve la racine ;
      (TYPE VERBE)                              ; et son type ;
      '(JE TU IL ELLE NOUS VOUS ILS ELLES)))
```

; **CONJ1** *fait tout le travail : d'abord elle prépare dans* **TERMINAISONS** *la bonne liste de terminaisons, ensuite, dans le* **LET** *intérieur, elle fait imprimer la conjugaison pour chaqu'un des pronoms ;*

```
(DE CONJ1 (RACINE TYP PRONOMS)
  (LET ((TERMINAISONS (COND
        ((EQ TYP 'ER) '(E ES E E ONS EZ ENT ENT))
        ((EQ TYP 'RE) '(S S T T ONS EZ ENT ENT))
        (T '(IS IS IT IT ISSONS ISSEZ ISSENT ISSENT)))))
    (LET ((PRONOMS PRONOMS)(TERMINAISONS TERMINAISONS))
      (IF (NULL PRONOMS) "voila, c'est tout"
        (PRINT (CAR PRONOMS)
            (CREE-MOT RACINE (CAR TERMINAISONS)))
        (SELF (CDR PRONOMS)(CDR TERMINAISONS))))))
```

Etudiez bien ce programme : c'est le premier programme, dans cette introduction à LISP, qui se constitue d'un ensemble de plusieurs fonctions. Afin de comprendre son fonctionnement, donnez le à la machine et faites le tourner. Ensuite, une fois que ça marche, commencez à le modifier, en y ajoutant, par exemple, les verbes en 'oir'.

Mais d'abord regardons quelques appels, afin de nous assurer qu'il tourne vraiment et qu'il fait bien ce qu'on voulait.

? (CONJUGUE 'PARLER) *; d'abord un verbe en 'er' ;*

JE PARLE
TU PARLES
IL PARLE
ELLE PARLE
NOUS PARLONS
VOUS PARLEZ
ILS PARLENT
ELLES PARLENT
= voila, c'est tout

? (CONJUGUE 'FINIR) *; ensuite un verbe en 'ir' ;*

JE FINIS
TU FINIS
IL FINIT
ELLE FINIT
NOUS FINISSONS
VOUS FINISSEZ
ILS FINISSENT
ELLES FINISSENT
= voila, c'est tout

? (CONJUGUE 'ROMPRE) *; et finalement un verbe en 're' ;*

JE ROMPS
TU ROMPS
IL ROMPT
ELLE ROMPT
NOUS ROMPONS
VOUS ROMPEZ
ILS ROMPENT
ELLES ROMPENT
= voila, c'est tout

Naturellement, ce programme est *très* limité : des verbes réguliers donnent des résultats corrects. Pour tous les autres, n'importe quoi peut arriver, en fonction de leurs terminaisons. Dans les exercices suivants je vous propose quelques améliorations, ou extensions, à apporter à ce programme.

10.5. EXERCICES

1. Rajoutez à la fonction **CONJUGUE** un argument supplémentaire, indiquant le temps dans lequel vous voulez avoir la conjugaison du verbe donné en premier argument. Ainsi, par exemple, l'appel

(CONJUGUE 'FINIR 'SUBJONCTIF-PRESENT)

devrait vous imprimer :

QUE JE FINISSE
QUE TU FINISSES
QU'IL FINISSE
QU'ELLE FINISSE
QUE NOUS FINISSIONS
QUE VOUS FINISSIEZ
QU'ILS FINISSENT
QU'ELLES FINISSENT

Modifiez le programme de manière à savoir conjuguer dans les temps suivants :

a. présent (ex : je parle)
b. imparfait (ex : je parlais)
c. futur (ex : je parlerai)
d. passé-simple (ex : je parlai)
e. conditionnel (ex : je parlerais)
f. subjonctif-présent (ex : que je parle)
g. subjonctif-imparfait (ex : que je parlasse)
h. passé-composé (ex : j'ai parlé)
i. plus-que-parfait (ex : j'avais parlé)
j. passé-antérieur (ex : j'eus parlé)
k. conditionnel-passé 1 (ex : j'aurais parlé)
l. conditionnel-passé 2 (ex : j'eusse parlé)

Pour cela, l'appel simple de la fonction **PRINT** à l'intérieur de la fonction **CONJ1** ne suffit plus. Introduisez donc une fonction **VERBPRINT** qui remplacera l'appel de **PRINT** juste mentionné, et qui se charge, dans le cas du subjonctif, d'effectuer l'impression des 'que′ supplémentaires.

Bien évidemment, pour les temps composés, il faut que le programme connaisse les divers conjugaisons du verbe 'avoir'. Introduisez une autre fonction supplémentaire, **PR-COMPOSE**, responsable de l'impression des temps composés.

2. Ecrivez un petit programme qui prend en argument une liste représentant une phrase et qui livre cette phrase transformée au pluriel. Exemple :

(PL '(JE FINIS AVEC CE CHAPITRE)) $\rightarrow$ **(NOUS FINISSONS AVEC CE CHAPITRE)**

ou

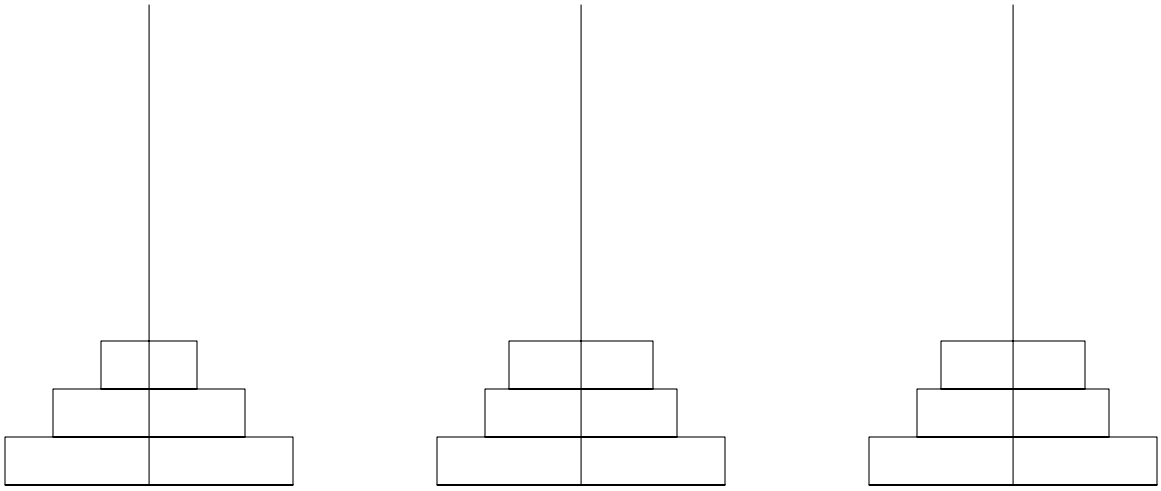
(PL '(IL ME PARLE)) $\rightarrow$ **(ILS ME PARLENT)**

ou encore

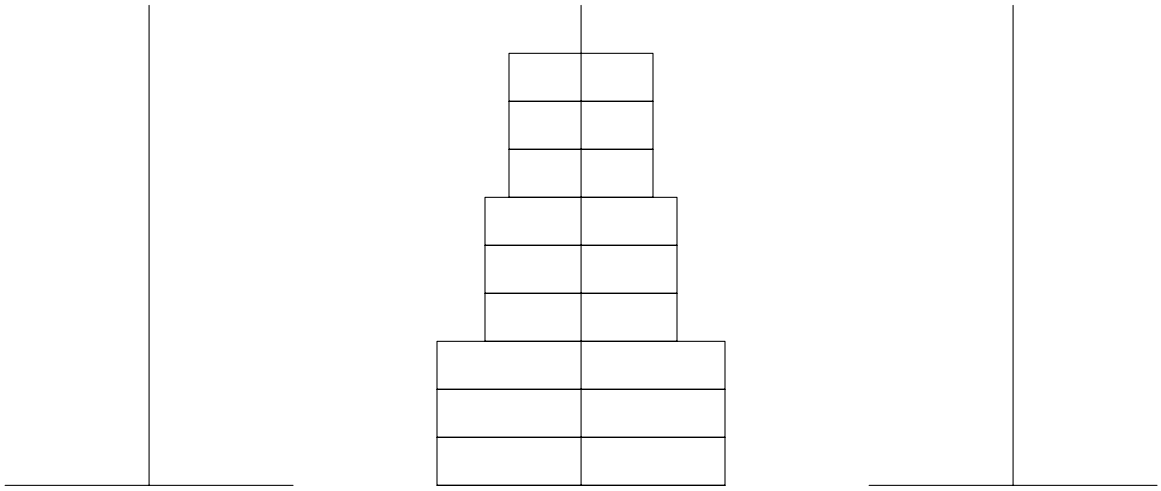
(PL '(TU NE FOULERAS PLUS CETTE TERRE))
$\rightarrow$ **(VOUS NE FOULEREZ PLUS CETTE TERRE)**

3. (pour les algorithmiciens) Ecrivez un programme résolvant le problème de la tour de Hanoi, mais où sont empilés des disques sur chacune des aiguilles à l'état initial. Naturellement, aucune des règles de déplacement n'est modifiée excepté, qu'un disque peut être déposé sur un autre de la même taille. A la fin, l'aiguille d'arrivée doit comporter dans un ordre de grandeur décroissant, tous les disques. Voici graphiquement l'état initial et l'état final :

état initial :



état final :



chapitre précédent

solutions

chapitre suivant