# 4. LA DEFINITION DE FONCTIONS

Nous avons vu qu'il est possible de construire une liste, graˆce à la fonction **CONS**, de prendre le premier élément d'une liste, graˆce à la fonction **CAR**, de récupérer la liste sans son premier élément, graˆce à la fonction **CDR**, et, par des combinaisons de ces fonctions, d'accéder à n'importe quel élément d'une liste ou de combiner des éléments arbitraires de listes quelconques.

Maintenant imaginons que nous ayons un programme (un programme est, pour nous, un ensemble d'appels de fonctions) dans lequel nous ayons à calculer à plusieurs reprises le quatrième élément d'une liste, comme par exemple dans

**(CAR (CDR (CDR (CDR '(A B C D E)))))**

ou

**(CAR (CDR (CDR (CDR '(1 2 3 4 5)))))**

ou

**(CAR (CDR (CDR (CDR '((0) (1) (1 0)(1 1) (1 0 0)(1 0 1)(1 1 0)(1 1 1))))))**

Clairement, après un certain temps il devient lassant de taper et retaper la même suite d'appels **(CAR (CDR (CDR (CDR . . .** On aimerait bien, dans de tels cas, avoir la possibilité d'abréger cette suite d'appels : par exemple, pouvoir écrire

**(4-IEME '(A B C D E))**

au lieu de

**(CAR (CDR (CDR (CDR '(A B C D E)))))**

LISP permet de résoudre ce problème, et propose comme solution la possibilité de *définir* de nouvelles fonctions qui, une fois définies, ne pourront pas être distinguées des fonctions déjà existantes. Nous nommerons les fonctions existantes des *fonctions-standard*, et celles que l'utilisateur de LISP définit lui-même des *fonctions-utilisateurs*.

Mais regardons d'abord comment on peut définir de nouvelles fonctions. Voici un exemple de définition de fonction :

```
(DE 4-IEME (L)
      (CAR (CDR (CDR (CDR L)))))
```

**DE** est une fonction standard indiquant à la machine qu'on est en train de DEfinir une nouvelle fonction. Naturellement, pour pouvoir ensuite s'en servir il faut lui associer un *nom* pour pouvoir s'y référer par la suite. Ce nom peut être quelconque, mais il est préférable de choisir un nom mnémonique, c'est-à-dire un nom qui rappelle ce que la fonction doit faire. Dans l'exemple donné, nous avons choisi le nom **4-IEME**, qui dit bien ce que la fonction est censée faire : chercher le 4-ième élément d'une liste. Puisqu'il s'agit d'une liste il est nécessaire de donner à la fonction un *argument*. Un seul argument ici, puisque on ne veut calculer que le 4-ième élément d'*une* liste à la fois. Dans l'exemple, **L** est le nom de cet argument. C'est un paramètre de la fonction, c-à-d. le nom d'une *variable*. La deuxième ligne de la définition de la fonction utilisateur **4-IEME** indique ce que la fonction doit faire : calculer le **CAR** du **CDR** du **CDR** du **CDR** de son argument **L**.

Après cet exemple informel, regardons la définition syntaxique de la fonction standard **DE** :

**(DE** *nom-de-la-fonction* **(**{$var_1$ $var_2$ ... $var_n$}**)** *corps-de-la-fonction***)**

avec

*nom-de-la-fonction*

un nom nouveau, qui n'existe pas encore dans la machine. De préférence ne prenez pas alors des noms comme **CAR**, **CDR** ou **CONS**, ou plus généralement, des noms de fonctions standard, puisque si vous définissez une fonction du nom d'une des fonctions standard, vous perdez sa définition originale.

$var_1$, $var_2$ etc

sont les noms des paramètres. Là aussi, choisissez de préférence des noms 'mnémoniques'.

*corps-de-la-fonction*

est la suite d'instructions décrivant le calcul que la fonction est censée réaliser.

Le premier argument de la fonction **DE** est donc le nom de la fonction à définir, le deuxième argument est une liste de paramètres (vous avez remarqué les parenthèses dans la définition syntaxique de **DE** ?!) Le reste de la définition d'une fonction est son corps, une suite d'appels de fonctions LISP standard ou de fonctions utilisateurs (dont vous devez également donner à LISP la définition avant une exécution d'un appel de la fonction). La valeur ramenée par une définition de fonction (par l'appel de la fonction **DE**) est le *nom* de la fonction que vous avez définie. Ici dans la définition de la fonction **4-IEME** c'est donc l'atome **4-IEME** qui sera ramené en valeur. C'est une indication du langage LISP, signifiant qu'elle a enregistré votre définition et que vous pouvez vous en servir par la suite.

La fonction une fois *définie*, on l'*appelle* (pour la tester ou pour l'exécuter). Un appel de la fonction **4-IEME** s'écrit comme suit :

**(4-IEME '(A B C D E F))**

La valeur ramenée par un appel de fonction utilisateur est la valeur ramenée par l'évaluation du corps de la fonction *avec les variables paramètres liées à la valeur des arguments de l'appel*. L'argument à l'appel de l'exemple ci-dessus est **(QUOTE (A B C D E F))**. Le paramètre **L** de la fonction **4-IEME** sera donc lié à la liste **(A B C D E F)**, la valeur de l'évaluation de l'appel de la fonction **QUOTE** avec l'argument **(A B C D E F)**.

Cela peut paraître un peu lourd (par écrit !). Toutefois, il est important de distinguer la *définition* d'une fonction de son *appel*.

Le résultat de l'appel

**(4-IEME '(A B C D E F))**

sera donc la valeur de l'expression

**(CAR (CDR (CDR (CDR L))))**

avec **L** lié à la liste **(A B C D E F)**, ceci est équivalent à évaluer l'expression

**(CAR (CDR (CDR (CDR '(A B C D E F)))))**

Donc :

**(4-IEME '(A B C D E F))** $\rightarrow$ **D**

Voici deux autres appels ainsi que le résultat de ces appels :

**(4-IEME '(1 2 3 4 5))** → **4**
**(4-IEME '((0) (1) (1 0)(1 1)(1 0 0)(1 0 1)(1 1 0)(1 1 1)))** → **(1 1)**

Voici maintenant une fonction qui construit une liste de ses trois arguments :

```
(DE LISTE3 (ARG1 ARG2 ARG3)
    (CONS ARG1 (CONS ARG2 (CONS ARG3 () ))))
```

et voici un appel de cette fonction :

**(LISTE3 1 2 3)** → **(1 2 3)**

Rappelons que l'évaluation de cette fonction se fait dans l'ordre suivant :
• d'abord la variable **ARG1** est liée à la valeur **1**
• ensuite la variable **ARG2** est liée à la valeur **2**
• ensuite la variable **ARG3** est liée à la valeur **3**
• ensuite le corps de la fonction est exécuté. Chaque fois qu'il y a une référence à une des trois variables **ARG1**, **ARG2** ou **ARG3**, le langage LISP calcule leur valeur respective. (La valeur d'une variable est la valeur qui lui est liée.)
et voici encore deux appels de cette fonction :

**(LISTE3 'TIENS 'TIENS 'TIENS)** → **(TIENS TIENS TIENS)**
**(LISTE3 'ENCORE '(UNE LISTE) 'OK?)** → **(ENCORE (UNE LISTE) OK?)**

Que fait alors la fonction suivante :

```
(DE TRUC (QQC)
    (CONS QQC (CONS QQC (CONS QQC ()))))
```

voici un appel :

**(TRUC 'HMMM)** → ?

Examinons le pas à pas : il s'agit d'une fonction, de nom **TRUC**, qui prend à l'appel un argument qui sera lié à la variable appelée **QQC**. Donc à l'appel de la fonction **TRUC**, LISP lie la variable **QQC** à l'argument **HMMM**, qui est la valeur de l'expression **(QUOTE HMMM)**. Ensuite, toute occurrence de la variable **QQC** à l'intérieur du corps de la fonction sera remplacée par cette valeur. Ce qui donne la même chose que si on avait écrit :

**(CONS 'HMMM (CONS 'HMMM (CONS 'HMMM ())))**

Ensuite, il ne reste qu'à évaluer cette expression. Comme toujours, LISP évalue d'abord les arguments des fonctions standard et ensuite seulement nous pouvons calculer la valeur de la fonction. Détaillons :

1. La première chose à faire est d'évaluer les deux arguments du premier **CONS** : l'expression **'HMMM**, le premier argument, et **(CONS 'HMMM (CONS 'HMMM ()))**, le deuxième argument. L'évaluation de **'HMMM** donne l'atome **HMMM** tel quel. Cet atome sera le nouveau premier élément de la liste résultant de l'évaluation de **(CONS 'HMMM (CONS 'HMMM ()))**.
2. Pour calculer cette liste, nous devons donc faire un **CONS** de **'HMMM** et de **(CONS 'HMMM ()))**. Comme précédemment, la valeur de l'expression **(QUOTE HMMM)** est l'atome **HMMM**. Cet atome sera le nouveau premier élément de la liste résultant de l'évaluation de **(CONS 'HMMM ()))**.
3. Ensuite il suffit de construire une paire de parenthèses autour de l'atome **HMMM**, ce qui donnera la

liste **(HMMM)**.

4. Maintenant, enfin, il est possible de calculer le **CONS** que nous avions laissé en suspens en ’2’. Il faut donc calculer le résultat du **CONS** de **HMMM** et de **(HMMM)**. Ce qui donne la liste **(HMMM HMMM)**.
5. En ’1’ il reste encore un **CONS** à calculer : le **CONS** de l’atome **HMMM** et de cette liste **(HMMM HMMM)**. Le résultat final est donc la liste

**(HMMM HMMM HMMM)**

La fonction **TRUC** nous retourne donc une liste avec trois occurrences de l’élément passé en argument ! C’est très puissant : après avoir défini une fonction il est possible de s’en servir de la même manière que des fonctions standard.

S’il subsiste encore des problèmes, relisez ce paragraphe (il est *très* important pour la suite !), et faites les exercices qui suivent.

### 4.1. EXERCICES

**1.** Voici la définition d’une fonction très intéressante :

```
(DE TRUC1 (L1 L2)
    (CONS
        (CONS (CAR L1)(CDR L2))
        (CONS (CAR L2)(CDR L1))))
```

que fait-elle pour les appels suivants :

**(TRUC1 ’(A 2 3) ’(1 B C))** → ?
**(TRUC1 ’(JA CA VA) ’(OUI ES GEHT))** → ?
**(TRUC1 ’(A VOITURE) ’(UNE CAR))** → ?

2. Que sont les résultats des appels de fonctions suivants ?

**(TRUC (4-IEME ’(DO RE DO MI DO FA)))** → ?
**(TRUC1 (LISTE3 ’UN ’CUBE ’ROUGE)**
**(TRUC1 (LISTE3 ’SUR ’LA ’TABLE)(TRUC ’BRAVO)))** → ?

3. Ecrivez une fonction à zéro argument qui ramène à l’appel la liste :

**(BONJOUR)**

4. Ecrivez une fonction à un argument qui ramène une liste contenant 4 occurrences de cet argument
5. Ecrivez une fonction à trois arguments qui ramène une liste contenant les trois arguments dans l’ordre inverse de l’appel. Exemple : si la fonction s’appelle **REVERSE3** un appel possible pourrait être :

**(REVERSE3 ’DO ’RE ’MI)**

et la valeur ramenée serait alors la liste **(MI RE DO)**.

chapitre suivant

solutions

chapitre suivant