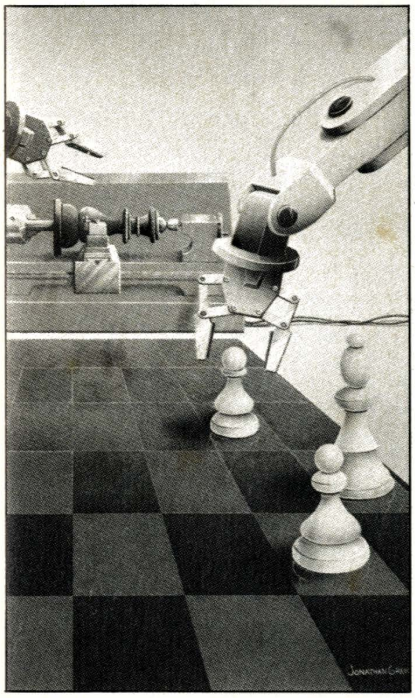
LISP

Ce langage se dispute avec d'autres l'hégémonie de la programmation en matière d'intelligence artificielle. Voici pour vous l'occasion de mieux le connaître.

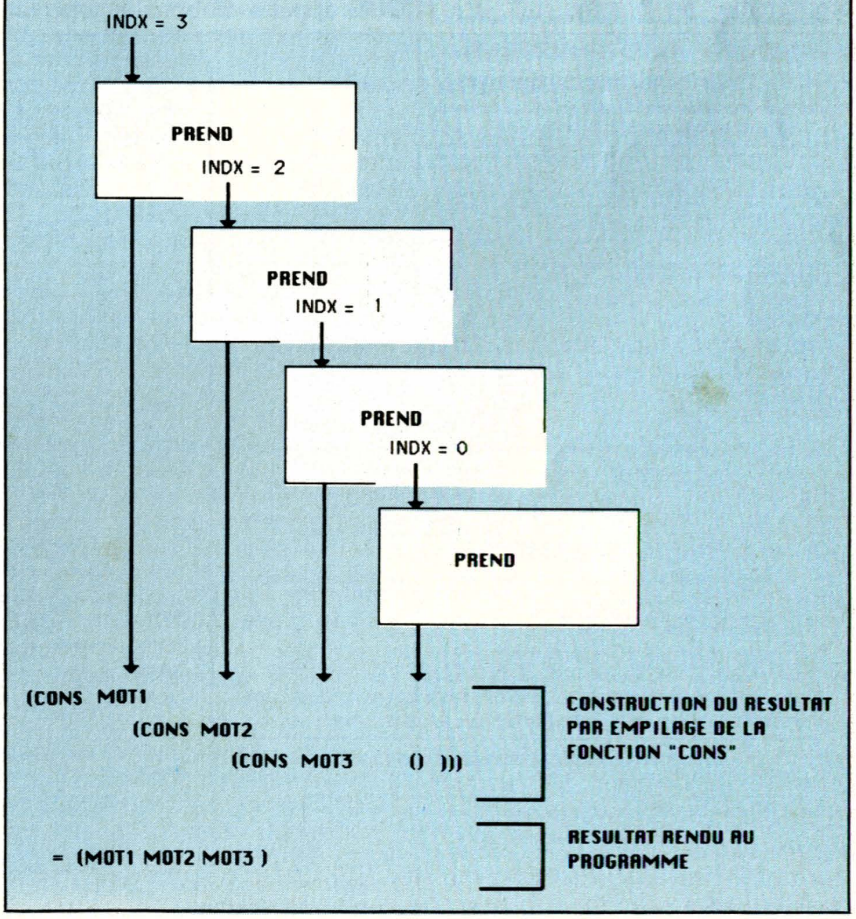
Ce troisième article à propos du langage LISP, va être le premier réellement consacré à la programmation en LISP. Les exemples et programmes présentés pourront être écrits en NANOLISP dont le listing a été publié dans *Micro et Robots* de mars 85. Rappelons que les lecteurs disposant d'un APPLE II (+, C ou E) et le langage PASCAL UCSD peuvent commander les disquettes contenant les sources commantées de NANOLISP et MICROLISP en s'adressant à *Micro et Robots*, 2-12, rue de Bellevue, 75019 Paris qui leur précisera les conditions de vente.



La dernière fois, nous terminions par un petit jeu à programmer en LISP dont une des solutions est présentée dans l'**encadré 1** et que nous

SCHEMA 1 : LA RECURSIVITE

Comment la fonction PREND lit 3 mots a l'écran et les retourne au programme appelant :



allons utiliser comme support pour illustrer quelques notions LISP de base.

Un (petit) jeu de mémoire

Il s'agit pour ce programme de lire un certain nombre de mots entrés au clavier par le joueur, puis après mémorisation de comparer une nouvelle série de mots avec ceux mémorisés afin d'afficher le nombre de mots retrouvés (sans attacher d'importance à leur ordre).

On peut, comme c'est souvent le cas pour la conception de programmes informatiques, découper le problème en tâches indépendantes : deux grandes étapes dans le jeu, l'enregistrement des mots tout d'abord puis la recherche d'un mot nouveau dans l'ensemble des mots connus. On peut aussi remarquer que ces «modules» du programme ont tous les deux besoin de lire des mots au clavier, ce qui fera alors l'objet d'une petite procédure utilisée par les deux «sous-programmes». Cette fonction de lecture, la plus simple des 4 nécessaires au programme, se nomme LIT, n'a besoin d'aucun paramètre en entrée et rend le mot lu au programme appelant en sortie. Son rôle est simplement d'afficher un message sur l'écran et de lire le clavier de la machine. La définition d'une fonction LISP est très simple : la fonction DE placée en tête d'une liste annonce la définition d'une fonction, tout de suite après DE doit se trouver le nom de cette fonction puis la liste des paramètres ou simplement une liste vide dans le cas (comme ici) où il n'y a aucun paramètre en entrée et enfin le corps de la fonction, c'est-à-dire une suite d'expressions LISP qui seront évaluées au moment de l'appel. Une fonction LISP rend toujours un résultat dont le programme principal peut ou non faire usage : ce résultat est la dernière expression évaluée dans la fonction; dans le cas présent ce sera toujours (READ) et la fonction LIT transmettra bien en retour le mot lu au clavier.

La fonction principale JOUE est aussi très facile à comprendre : il s'agit uniquement d'activer la lecture des mots, de vider l'écran puis d'activer la recherche et enfin d'afficher le résultat. C'est ainsi que JOUE appelle d'abord la fonction PREND avec le nombre 5 (rappelons que NANOLISP ne connaît pas les nombres en notation usuelle mais sous forme de liste dont la longueur est le nombre représenté) lui indiquant qu'il y a 5 mots à lire, réalise ensuite un certain nombre de PRINT pour vider l'écran puis active enfin la fonction CHERCHE en lui indiquant aussi qu'il y a 5 mots à demander et à rechercher. La fonction CHERCHE étant en dernière position dans JOUE, celle-ci rendra directement le résultat de la recherche qui sera bien sûr le score du joueur et le résultat sera affiché automatiquement par LISP. Signalons une autre écriture possible pour JOUE :

```
(DE JOUE(NOMBRE)
    (PREND NOMBRE)
    (PRINT ' .) ... (PRINT ' .)
    (PRINT 'A-VOUS)
    (CHERCHE NOMBRE))
```

Il est alors possible de jouer avec n'importe quel nombre de mots à condition d'activer la fonction sous la forme :

```
(JOUE ' (I I I I I))
```

Remarquons aussi qu'au moment des appels de PREND et CHERCHE, NOMBRE puisqu'il n'est pas précédé de l'apostrophe (appelée

Encadré 1 : programme jeu en LISP.

```
;LA FONCTION PRINCIPALE
(DE JOUE()
        (SETQ MOTSLUS (PREND '(I I I I I)))  ;LIT 5 MOTS ET LES STOCKE
                                             ;DANS MOTSLUS
        (PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.) ;REMPLISSAGE
        (PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.) ;D'UN ECRAN POUR
        (PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.) ;FAIRE DISPARAITRE
        (PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.) ;LES 1ERS MOTS
        (PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)(PRINT '.)
        (PRINT 'A-VOUS)
        (CHERCHE MOTLUS '(I I I I I)))       ;DONNE 5 ESSAIS POUR RETROUVER
                                             ;LES MOTS DE MOTSLUS

;LECTURE D'UN MOT PRECEDEE D'UNE "INVITE"
(DE LIT() (PRINT 'MOT:) (READ))

;LECTURES DES INDX PREMIERS MOTS
(DE PREND (INDX)
    (COND
      ((EQ INDX ()) () )                     ;PLUS RIEN A LIRE
      ( T (CONS (LIT) (PREND (CDR INDX))))))  ;LIT UN MOT ET PASSE A LA SUITE

;COMPARE *INDX* NOUVEAUX MOTS AVEC LES MOTS DE MOTSLUS
(DE CHERCHE (MOTLUS INDX)
  (COND
    ((EQ INDX ()) (PRINT 'RESULTAT) () ) ;PLUS D'ESSAIS POSSIBLES
    (T (CONS    ;CONSTRUCTION DE LA LISTE RESULTAT DONT LE 1ER
               ;ELEMENT (LE CAR) EST LE RESULTAT POUR LE MOT COURANT ET LE
               ;RESTE (CDR) UN APPEL RECURSIF A CHERCHE
      (COND
            ((EQ (PRESENT (LIT) LL) T) '1) ;1 - LIT UN MOT
                                           ;2 - VERIFIE S'IL EST DANS MOTLUS
                                           ;    EN APPELANT PRESENT
                                           ;3 - SI PRESENT REND VRAI (T),
                                           ;    I SERA "RENDU" PAR L'EXPRESSION
                                           ;    POUR SIGNIFIER 1 SUCCES
            (T () )) ;LE MOT COURANT N'EST PAS
                     ;DANS MOTSLUS, LE RESULTAT EST DONC ()

      (CHERCHE (MOTSLUS CDR INDX)) )))) ;APPEL RECURSIF AVEC INDX DIMINUEE DE 1
                                        ;ELEMENT POUR LES ESSAIS SUIVANTS

;TESTE SI MOT EST PRESENT DANS MOTLUS
(DE PRESENT (MOT MOTSLUS)
  (COND
    ((EQ MOTSLUS ()) ())        ; MOTSLUS EST VIDE -> ECHEC
    ((EQ MOT (CAR MOTSLUS)) T) ;MOT EST LE 1ER ELEMENT DE MOTSLUS -> SUCCES
    (T (PRESENT MOT (CDR MOTSLUS)))))  ;SINON ON CONTINUE
```

Note : les mots suivants les points-virgules sont des commentaires et ne font pas partie du programme.

Encadré 2 : fonctions du Nanolisp.

```
NOM          FONCTION
'       le QUOTE indique a LISP que l'expression immediatement
        suivante ne doit pas etre evaluee mais consideree telle
        quelle .
        A = *** ERREUR ATOME INDEFINI
        Mais
        'A= A

CAR     Rend le premier element d'une liste
         (CAR '(A B))=A
CDR     Rend la liste sans son  premier element
        (CAR '(A B C))=(B C)
CONS    Rend la liste constituee du premier et du second parametre .
        le 2eme doit etre une liste ou NIL .
        (CONS '(A B) '(C D))=((A B)C D)
SETQ    Affecte au premier parametre (non evalue) la valeur
        du second .
        (SETQ A '(B C))=(B C)
        Donne ensuite : A = (B C)
        Les exemples suivants conservent cette valeur de A
ATOM    Rend T si le parametre est un atome
        (ATOM A)=()
EQ      Rend T si les 2 parametres ont exactement la meme
        valeur atomique
        (SETQ E 'C)
        (SETQ F 'C)
        (EQ E F)=T
        Mais
        (EQ A '(B C))=()
        On pourra facilement ecrire la fonction EQUAL qui verifie
        si les 2 parametres sont egaux au sens large (meme contenu)
COND    Execute la premiere des listes dont le CAR
        n'est pas NIL .
        (COND
          ((ATOM A) (PRINT 'ATOME))
          (T        (PRINT 'LISTE))) = LISTE
LAMBDA  Sert a la definition de fonctions
              ( (LAMBDA(X) (COND
                        ((ATOM X) (PRINT 'ATOME))
                        ( T       (PRINT 'LISTE))))
                '(A B)) = LISTE
        Le premier element de la liste est une definition de
        fonction sans nom (elle sera oubliee immediatement) et cette
        fonction est appliquee sur les parametres constituant le
        reste de la liste . La liste suivant immediatement le LAMBDA
        est la liste des parametres de la fonction .
READ    Rend l'expression LISP lue en entree
              (SETQ EXPR (READ))
PRINT   Imprime son parametre
              (PRINT A) = (B C)
TRACE   Trace les fonctions centrales de l'interprete
        (TRACE)
UNTRACE L'inverse
        (UNTRACE)
LOAD    Lit le fichier specifie comme parametre . L'extension
        .TEXT est automatiquement ajoutee par l'interprete .
        (LOAD 'MONFICHIER) = ...lecture du fichier MONFICHIER.TEXT
OBLIST  Affiche la liste des atomes connus
        (OBLIST)= ... liste des atomes
DE      Definit une fonction avec un nom.
        (DE CADR (LISTE)    (CAR (CDR LISTE)))
            nom  liste        corps de la fonction
                 parametres
                 ou NIL
QUIT    Sort de NANOLISP .
```

QUOTE en LISP) va être évalué et c'est sa valeur (soit : (I I I I I)) qui va être transmise à PREND et CHERCHE ce qui est parfaitement équivalent à l'écriture de l'encadré 1 où les appels ont directement lieu avec la valeur elle-même (dans ce dernier cas, la présence de QUOTE indique à LISP que le paramètre est directement la valeur à passer à la fonction appelée).

Le passage de paramètres

Profitons aussi des appels de fonctions pour signaler une particularité de LISP à propos du passage de paramètres. Dans tous les langages informatiques autorisant la définition de sous-programmes ou procédures ou fonctions avec paramètres (c'est-à-dire la plupart sauf le BASIC standard), il est possible d'écrire des sous-parties de programme indépendantes des noms réellement manipulées dans le programme appelant. Ainsi, la fonction

CARRE(Y) = Y * Y

Pourrait être appelée successivement pour calculer

...CARRE(NOMBRECAROTTES)...
... CARRE(NOMBRENAVETS)...

Néanmoins de nombreuses questions restent ambiguës avec une telle écriture et en particulier la suivante : une fonction a-t-elle le droit de modifier de façon visible par le programme appelant un ou plusieurs de ses paramètres ? La réponse de PASCAL à ce problème est de préciser cela au moment de l'écriture de la fonction ou procédure :

PROCEDURE EXEMPLE(VAR X:INTEGER ; Y:CHAR);

Ici les modifications sur X seront répercutées à l'extérieur, en revanche Y sera rétabli à sa valeur précédent l'activation de la fonction même s'il a été modifié par la procédure EXEMPLE . En LISP, il n'y a pas cette distinction puisque seul le deuxième cas est envisagé : *une fonction ne peut pas modifier directement les paramètres d'appels.* Un exemple pour fermer la parenthèse (LISP bien sûr) :

1 — La définition de la fonction
(DE EXEMPLE(X)
(SETQ X 'DANS-EXEMPLE))
2 — Initialisation d'un atome Y
(SETQ Y 'HORS-EXEMPLE)
3 — La valeur de Y avant l'appel
Y
= HORS-EXEMPLE
4 — Appel de la fonction exemple
(EXEMPLE Y)
5 — On constate que le paramètre est modifié à l'intérieur de la fonction puisque celle-ci rend :
=DANS EXEMPLE
6 — Mais il a déjà récupéré son ancienne valeur après cet appel :
Y
=HORS-EXEMPLE

Les lecteurs disposant des listings NANOLISP et MICROLISP pourront étudier de plus près la fonction PAIRLIS qui réalise le passage et le rétablissement des valeurs des paramètres en entrée et sortie de fonctions. Retournons maintenant à l'examen du programme JEU.

La recursivité : tout un art

Si cette série concernait l'initiation au BASIC, la programmation des fonctions PREND et CHERCHE aurait été effectuée par un magnifique

FOR I=1 TO 5 etc.

et beaucoup d'énergie aurait été dépensée pour expliquer que I est une «variable de boucle», c'est-à-dire une sorte de compteur qui etc...

La programmation LISP repose souvent sur analyse plus poussée de la fonction à programmer qui débouche la plupart du temps sur une écriture plus réduite (donc moins de possibilités d'erreurs). Soit donc à lire un certain nombre de mots (représenté par une liste) au clavier, si ce nombre est nul (liste vide) il n'y a rien à faire, dans le cas inverse il suffit de lire un mot et de réappliquer la même méthode en diminuant le nombre de 1. C'est exactement ce que réalise la fonction PREND dont le paramètre est le nombre de mots à lire : si celui-ci est nul elle rend nil (un synonyme de LISTE VIDE qui s'écrit ( ) en LISP), sinon elle lit un mot et s'appelle elle-même avec le CDR (c'est-à-dire la liste sans le premier élément) du nombre ce qui réalise bien une soustraction de une unité. Comme il faut aussi construire la liste des mots lus (et pas seulement compter leur passage), la fonction «rend» la liste composée du mot lu et du résultat de l'appel récursif qui lui-même est la liste composée du mot (suivant) lu etc. La mémorisation du résultat est simplement effectuée au niveau de la fonction principale JOUE grâce au (SETQ MOTLUS...). Le **schéma 1** illustre l'empilement des appels recursifs de la fonction PREND. Les fonctions CHERCHE et PRESENT obéissent au même principe de récursivité : le rôle de CHERCHE est de comparer chaque nouveau mot à la liste MOTSLUS et ceci un certain nombre de fois. CHERCHE fonctionne donc de façon symétrique à PREND : l'appel récursif réalise le comptage grâce à la liste INDX, appelle la fonction PRESENT avec un mot lu au clavier et «rend» la liste dont le CAR (la partie gauche) est le résultat pour ce mot uniquement (sous la forme d'un I pour réussite et de ( ) pour échec) et le CDR (la partie droite) est le résultat de l'appel récursif à elle-même pour les mots suivants jusqu'à épuisement de la liste de comptage INDX.

Enfin la fonction PRESENT est d'une simplicité biblique : elle rend vrai (note T en LISP) si le nouveau mot est le même que le CAR de la liste MOTSLUS et sinon s'appelle elle-même pour continuer sur le CDR de la liste MOTSLUS jusqu'à épuisement de celle-ci (et c'est alors l'échec de la recherche) ou correspondance avec un mot de cette liste (succès de la recherche).

En guise de conclusion

Un langage informatique s'apprend avant tout par la pratique, en essayant d'écrire des programmes de plus en plus performants, et non en apprenant par cœur la liste des instructions à disposition du programmeur. Cet article comme ceux qui suivront sont donc plus destinés à éclairer des aspects et des techniques de LISP plutôt que passer en revue la liste des fonctions disponibles.

Les points importants illustrés aujourd'hui sont au nombre de 3 :

— Les expressions LISP se lisent, comme en arithmétique, de l'intérieur vers l'extérieur des parenthèses.

— Une expression, une fonction LISP rend toujours un résultat et il est important de bien savoir lequel.

— La récursivité est une des forces de LISP, le passage de paramètres, une de ses faiblesses.

L'**encadré 2** rappelle les fonctions disponibles dans NANOLISP et le résultat de leur appel. A vos claviers...

J.-M. Husson