JPC

MARS 1988

NUMERO 52

Le numéro 40 F

A PROPOS DU CLUB

P. David — Editorial *1*

Du nouveau à propos du HP-28S *2*
Courrier du coeur *2*

HP28

P. Courbis — Choc en retour *4*
G. Le Stum — Questions, réponses *4*

HP75

J.Y. Hervé — Graphiques *8*

HP71

P. David & J. Taillandier — Programmation structurée (acte I) *10*

L. Istria — Hanoï et la récursivité *33*
X. Bille — Démontrez vos théorèmes *34*

Le coin des Lhex *41*

EDITORIAL

Chers Membres,

Je tiens à vous annoncer deux nouvelles importantes : une bonne et une mauvaise.

Commençons par la mauvaise : Jacques Baudier s'est fait voler son HP-71, ce qui a comme conséquence directe l'arrêt (nous espérons momentané) de sa série d'articles sur l'assembleur du HP-71. Que les amateurs d'assembleur se rassurent : ils ont de quoi se creuser les méninges ce mois-ci ! Souhaitons tous que Jacques puisse retrouver une machine rapidement.

La bonne nouvelle : nous avons pu récupérer la Rom du HP-28C (version 1BB) grâce aux efforts de Paul Courbis, Sébastien Lalande et Guillaume Le Stum. Le désassemblage est en cours, grâce au HP-71 et à JPC Rom...

Avant de vous laisser découvrir la suite de ce JPC, je tiens à vous signaler que ce numéro ne contient pas d'article sur la HP-41. C'est dommage, n'est-ce pas ? Alors, envoyez-nous vos oeuvres, nous les publierons avec plaisir.

En attendant, faites de beaux rêves. Et n'oubliez pas de lire votre journal...

Pierre David (37)

DU NOUVEAU
A PROPOS DU HP-28S

En direct des Etats-Unis,

William C. Wickes

Depuis l'annonce du HP-28S dans le précédent JPC, nous avons reçu les précisions suivantes à propos des nouvelles caractéristiques. Ces informations nous sont directement parvenues de William C. Wickes, le créateur des HP-28, via le réseau Unix.

- 32 Ko de Ram,
- système de sous-répertoires pour le menu USER,
- menus modifiables pour combiner des variables utilisateur et des commandes système,
- sauvegarde, rappel et fusion d'images graphiques,
- combinaisons et arrangements,
- pas d'arrêt du tracé de fonctions en cas d'erreur mathématique,
- une nouvelle fonction POS cherche un objet dans une liste,
- A(I) renvoie le $i^{ème}$ élément d'une liste ou d'un tableau, permettant ainsi l'utilisation des variables indexées dans les algebraics (A(I,J) fonctionne aussi),
- les fonctions définies par l'utilisateur peuvent l'être dans des programmes,
- les commandes prenant comme arguments des réels dans des listes permettent l'utilisation de programmes ou d'objets symboliques à la place des nombres,
- un mode double interligne est ajouté pour l'imprimante,
- les programmes sont décompilés et listés avec une indentation automatique pour les structures (NDLR : ça existe aussi sur le HP-71, voir JPC 53),
- les opérateurs mono-caractères ajoutent automatiquement les espaces nécessaires durant l'introduction des programmes,
- l'appui simultané sur [ON][L] imprime l'écran sur l'imprimante à n'importe quel moment,
- RND (round) fonctionne sur les objets complexes et matriciels,
- l'inversion des matrices singulières est liée à l'exception « division par zéro »,
- Enfin, les erreurs identifient automatiquement la commande coupable, ce qui est d'un grand secours durant le débogage.

D'autre part, le HP-28S intègre une électronique complètement nouvelle : le microprocesseur, un des display drivers et 64 Ko de Rom sont maintenant réunis dans un seul boîtier. Les 32 Ko de Ram ne sont pas des Ram HP. Conséquences directes : il semblerait que la vitesse de l'ensemble soit supérieure de 50% à celle d'une HP-28C, mais d'autre part, on ne peut pas ajouter de mémoire supplémentaire.

COURRIER DU COEUR

Jean-Claude Fourès
100 rue de la Chapelle
75018 Paris
Tél : (1) 42 01 12 10 ou (1) 42 40 32 58

Vend :
HP-71B + lecteur de cartes + interface HPIL + imprimante ThinkJet + lecteur de disquettes HP9114A + livres sur Basic HP71 et Forth. Le prix est à débattre.

André Vialaron
11, rue Darquié
31000 Toulouse
Tél : 61 52 20 03

Vend :
Imprimante thermique HP82143A pour HP-41 (révisée par HP) : 1000 F, soit 50% du prix EduCALC.

Christophe Berthet
14 rue Abbé Cottard Josserand
01000 Bourg en Bresse
Tél : 74 21 18 17

Vend :
HP-41CX n'ayant jamais servie (cadeau) encore sous garantie à la FNAC. Achetée 2500 F, prix à débattre.

HP28

P. Courbis
G. Le Stum

Choc en retour *4*
Questions, réponses *4*

CHOC EN RETOUR

Racourcissaisissant

Dans JPC 51, je vous proposais le programme →LEX. Celui-ci peut être remplacé par #3EEA0 SYSEVAL qui fait exactement la même chose...

Erratum

Dans l'article *Le tour du HP-28...* se sont glissées quelques petites erreurs :

page 15, deuxième colonne, en haut :
Il fallait lire 1C3B5 et non pas 1C285

page 23, le listing de ASS commence par :
```
« → LM « HEX "" 1 LM SIZE FOR X
```
et non pas par :
```
« → LM « HEX "" 1'LM SIZE FOR X
```

Paul Courbis (392)

QUESTIONS, REPONSES

Racines cubiques

En réponse à une partie du S.O.S. de Laurent Vergnes : `'-8^INV(3)'` donne `-2` car l'opérateur ^ (puissance) est exécuté avant la négation (-). `'(-8)^INV(3)'` donne un résultat complexe.

Une solution au problème des racines cubiques de $f(x)$ est de définir une nouvelle fonction ainsi :

```
« → a 'SIGN(a)*ABS(a)^INV(3)' »
```

Si on l'appelle CRT, `'CRT(-8)'` donnera bien `-2`.

Pgcd et fractions

Tout d'abord voici un programme de calcul de *pgcd*. Il utilise l'algorithme classique :

- $pgcd\ (a, b) = pgcd\ (b, a \bmod b)$
- $pgcd\ (a, 0) = a$

Voici donc PGCD :

```
« DO
    SWAP OVER MOD      a,b -> b,MOD(a,b)
    DUP                duplique pour tester
  UNTIL 0 ==
  END                  stoppe si MOD(a,b)=0
  DROP
»
```

Ce programme occupe 43 octets

Une application immédiate est la simplification de fractions à l'aide du programme FRAC que voici :

```
« DUP C→R PGCD / »
```

Ce programme utilise 34 octets

Les fractions étant introduites sous la forme `(n,d)` où n est le numérateur et d le dénominateur.

Le programme suivant est destiné à tous ceux qui comme moi sont nuls en calcul mental et trouvent plus agréable d'être l'esclave de leur machine que de s'entrainer à additionner et simplifier des fractions aussi vite qu'elle.

Toujours avec la même notation on a :
$(a,b)+(c,d) = (ad+bc,bd)$

Le programme utilise le fait que :
$(a+bi)(c+di) = ac-bd + (ad+bc)i$

```
«
  DUP2 IM SWAP IM *  calcul de bd
  ROT ROT * IM       calcul de ad+bc
  SWAP R→C           remise sous la forme (n,d)
  FRAC               simplification
»
```

ADFR occupe 54 octets

Vous aurez certainement remarqué qu'aucune erreur n'apparaît si vous mettez un dénominateur nul. N'en déduisez surtout pas que mon programme n'est pas parfait : c'est le fruit d'une décision délibérée visant à laisser une part de réflexion à l'utilisateur (si, si, je vous assure...)

Décomposition en facteurs premiers

Ce programme décompose le nombre impair n en facteurs premiers en le divisant par tous les impairs successifs d, si le reste est nul on remplace n par n/d, sinon on remplace d par $d+2$, on s'arrête quand $d \times d > n$. Les termes de la décomposition sont mis dans une liste.

```
«
  { } SWAP 3
  DO
    WHILE
      DUP2 MOD 0 ==
    REPEAT
      ROT OVER 1 →LIST +
      SWAP ROT
      OVER / SWAP
    END
    2 +
  UNTIL
    DUP2 SQ <
  END
  DROP
  1 →LIST +
»
```

Ce programme occupe 106 octets

Par exemple, `100000001 PREM` donne `(17 5882353)` ...au bout de 69 secondes. Notez bien que je n'ai pas osé parler de test de vitesse !

On peut aisément modifier le programme pour qu'il traite aussi les nombres pairs mais ça prend à mon avis trop de place : le simili forth de notre calculette préférée est assez compact, mais 2 Ko, c'est quand même vraiment juste.

Par exemple, on peut remplacer `3 DO` par `2 DO` et `2 +` par :

```
IF DUP 2 == THEN 1 + ELSE 2 + END
```

Cette solution n'est pas la plus coûteuse en octets, mais c'est sans aucun doute celle qui ralentit le plus le programme.

Combinaisons et permutations

Il est écrit dans la publicité de la HP-28C qu'elle contient les combinaisons et permutations. Or, je n'en ai trouvé aucune trace. J'ai donc été obligé de sacrifier de précieux octets de mémoire pour `PANP` et `PCNP` qui calculent respectivement le nombre de permutations et de combinaisons de p éléments parmi n. Il faut d'abord introduire dans la pile n puis p. `PCNP` utilisant `PANP`, vous ne pouvez donc pas l'utiliser séparément.

PANP(n,p) = $n!/(n-p)!$

```
«
  OVER SWAP - 1 3 ROLLD
  WHILE
    DUP2 >
  REPEAT
    1 + ROT OVER *
    3 ROLLD
  END
  DROP2
»
```

Ce programme occupe 71 octets

PCNP(n,p) = $n!/(p!(n-p)!)$

```
«
  DUP2 -
  MIN DUP FACT
  3 ROLLD PANP
  SWAP /
»
```

Ce programme occupe 48 octets

PS : si vous savez comment accéder à ces fonctions, n'hésitez pas à dire comment vous procédez...

Récursivité

Eh oui, ça marche aussi sur notre machine favorite. Mais mis à part la satisfaction de savoir que c'est possible, je n'ai pas trouvé d'application réellement intéressante à vous proposer. Je pense que la HP-28C est trop limitée en vitesse comme en mémoire pour que ce soit intéressant (tout contre-exemple à mes assertions sera cependant le bienvenu).

Cette fonction étant intégrée à la machine, je vous ferai grâce du classique exemple de factorielle. Voyons plutôt comment ça se passe avec le *pgcd* : on a déjà vu l'algorithme utilisé. Le programme peut donc s'écrire en version récursive :

```
« → a b
  'IFTE(b==0,a,PGCD(b,MOD(a,b)))'
»
```

Ce programme occupe 95 octets

L'utilisation des variables locales et de la forme algébrique consomme beaucoup de mémoire, mais elle permet d'utiliser `PGCD` comme une fonction. On pourra donc faire `'PGCD(21,7)' EVAL`. On ne pouvait pas mettre le programme sous cette forme sans utiliser la récursivité et on est donc très content. L'étape suivante est `'PGCD' PURGE`, à moins qu'on soit

persuadé qu'il est indispensable de pouvoir utiliser des *pgcd* dans des objets du type *algebraic*.

On peut de même refaire PANP et PCNP sachant que :
- $PANP(n,p) = PANP(n,p\text{-}1)(n\text{-}p+1)$
- $PANP(n,0) = 1$

et que :
- $PCNP(n,p) = PCNP(n\text{-}1,p\text{-}1) + PCNP(n\text{-}1,p)$ si $p>n$
- $PCNP(n,p) = 0$
- $PCNP(n,0) = 1$

Si vous faites le programme PCNP de cette manière vous aurez un excellent exemple des limites de l'utilisation de la récursivité sur la HP28C.

Nota : toutes les tailles de programme sont données en supposant que vous leur donnez un nom de 4 lettres.

Guillaume Le Stum (432)

HP75

J.Y. Hervé	Graphiques	*8*
J.Y. Hervé	Programme "GRAPH22"	*36*

GRAPHLEX... POUR HP-75

Voici un Lex qui, accompagné de son programme Basic associé, permettra aux possesseurs de HP-75 et de l'imprimante HP-82905B de faire presqu'aussi bien que s'ils avaient un HP-71, une HP-2225 et l'un des logiciels graphiques maintenant disponibles pour cette machine (Cf *Graphic* de Pierre David, *Plot Package* de Paul Grimmer, *Graph-71* de Chris Bunsen ou les fonctions `GLINE` et `GPSET` de la JPC Rom).

Le Lex fournit cinq fonctions programmables d'édition de chaînes graphiques sur imprimante matricielle de type HP-82905B (*).

RFILL$ (H$,C,L)

Fonction alphanumérique, qui complète à droite une chaîne H$ avec des caractères de code C, pour obtenir une chaîne de longueur L. Si L est inférieure à la longueur de la chaîne H$, cette dernière sera tronquée à la longueur demandée (et retournée vide si L ≤ 0).

ARPT$ (H$,N)

Fonction alphanumérique permettant de répéter N fois la chaîne H$. Si N ≤ 0, la chaîne retournée sera vide.

SBIT$ (H$,B)

Fonction mettant à 1 le bit B (0 à 7) de tous les caractères de la chaîne H$. Génère l'erreur 11 (`Argument out of range`) si B n'est pas compris entre 0 et 7 inclus.

CBIT$ (H$,B)

Fonction inverse de la précédente, met à zéro le bit numéro B de tous les caractères de la chaîne H$. Comme précédemment, l'erreur 11 (`Argument out of range`) est générée si B n'est pas compris dans l'intervalle allant de 0 à 7 inclus.

VBIT$ (H$)

Cette dernière fonction inverse les bits de la chaîne H$, c'est-à-dire met à 1 les bits 0 et réciproquement.

Le programme Basic proposé à titre d'exemple permet de tracer le graphe de toute fonction donnée sous la forme $y=f(x)$. Il demande de fournir une fenêtre de tracé, et effectue celui-ci avec un réticule (les anglophones diraient un *grid*) approprié. Notez au passage la méthode utilisée pour spécifier la fonction (le programme s'auto-modifie lignes 3 et 4), avec la conséquence que ce programme ne peut être implanté en PMS ou Eprom. Le Lex peut l'être, lui !

Jean-Yves Hervé (450)

(*) NDLR : Les fonctions proposées concernent toutes la préparation des chaînes graphiques, mais non leur impression (laquelle demeure classique, par `PRINT`) : la rédaction de programmes équivalents pour les imprimantes en mode raster graphics est donc tout à fait possible. A suivre...

```
GRAPHLEX  319 bytes

Line ----------------Code----------------Chk
0001 C18623018D4C506E23A347524150484C45586900 92
0002 0A00220014003D00400044007400AE00EF000901 1F
0003 400040004000400040000FFFF5246494C4CA44152 F3
0004 5054A456424954A453424954A443424954A4FFFF C3
0005 61179E00382ECE8B3EF5019306E5CE8B3E54CE12 5A
0006 1F7EE36E06E3CE25FCF8146E8BF4107E8BF40742 1F
0007 14E016E4F0F16616E4F0EC9E282ECE8B3EF50193 2A
0008 A306E554CE121F5EE36E931E668BF4046EC3F0F8 4D
0009 CE25FCF8186E06E36E8BF4115C1EA15214A15C8B 66
0010 F4F25012E016E4F0F59E182E54CE121F6E0AE3CE 71
0011 25FCF80C6E8BF4084314E08E16E4F0F49ECE8B3E FC
0012 6690FD04C808FA04CEAC4C0B5093886690F7058A 86
0013 5084F0F754CE121F6E0AE3CE25FC9E282E54B1A3 FD
0014 8214C6C700F80D6E8BF4094314E0109416E4F0F3 DF
0015 9E282E54B1A38214C6C700F80F508E6E8BF40943 E5
0016 14E010C716E4F0F39E                       4B
0017 8A3A
```

ASSEMBLEUR

P. David & J. Taillandier — Programmation structurée (acte I) *10*

BASIC

L. Istria — Hanoï et la récursivité *33*
X. Bille — Démontrez vos théorèmes *34*

J.Y. Hervé — Programme "GRAPH22" (pour HP-75) *36*
L. Istria — Programme "HANOI" *36*
X. Bille — Programme "DEMOS" *37*

LE COIN DES LHEX

41

PROGRAMMATION STRUCTUREE
(ACTE I)

Le HP-71 *de base* possède peu d'instructions de programmation structurée. Bien sûr, il possède DEF FN, FOR et NEXT, GOSUB, CALL et SUB, ce qui est déjà très bien pour un Basic.

Cependant, il est très pauvre par rapport à des langages réputés plus évolués, comme Pascal, C ou encore Ada.

L'objet de ce Lex est de fournir des structures de contrôle dont certaines sont largement plus souples que les langages mentionnés ci-dessus.

Mais pourquoi vouloir faire de la programmation structurée ?

Nous ne tenterons pas de répondre ici à cette question. Une discussion sur le sujet prendrait vraiment trop de place et trop de temps. Nous citerons juste les avantages que notre expérience de la programmation nous a permis de discerner :

La programmation structurée a pour objectif de décomposer les problèmes complexes en une suite de problèmes plus simples. Cette décomposition conduit à une *rédaction* plus facile des programmes, une *mise au point* plus aisée, et une plus grande *lisibilité*.

Cette structuration est maintenant possible sur HP-71 avec le Lex STRUC2.

LES BOUCLES

La boucle WHILE ... END WHILE

La première forme de boucle, et la plus intéressante, est la boucle du type **tant que**. Par exemple, on écrira :

```
10 X=0
20 WHILE KEY$=""
30   DISP X;X*X
40   X=X+1
50 END WHILE
```

Cette boucle se lit comme :

tant que aucune touche n'est appuyée
faire
    afficher *x* et son carré
    incrémenter *x*
fin tant que

L'affichage et l'incrémentation constituent le *corps* de la boucle. La condition est la condition de répétition de la boucle. Autrement dit, la boucle est répétée tant que la condition est vraie.

Cette boucle peut ne jamais être exécutée. Si votre condition est fausse la première fois, le corps ne sera jamais parcouru, et l'exécution continuera tout de suite après le END WHILE.

Il a été prouvé en 1966 (Théorème de Böhm et Jacopini) que tout algorithme exprimé avec des GOTO pouvait être exprimé par des boucles de type **tant que**.

Mais le plus intéressant est que cette boucle est le point départ des preuves formelles de programmes. Pour toute boucle de type **tant que**, on est sur que la condition est vraie dans le corps juste après le **tant que**, et fausse juste après la boucle elle-même. On peut ainsi bâtir des assertions. Une assertion valide à chaque tour de boucle est un *invariant de boucle*. Tout le travail de preuve consiste à déterminer l'*invariant*. La preuve est ensuite facile.

Il est bien évident que peu de personnes programment en prouvant leurs programmes. Néanmoins, s'entrainer à déterminer des invariants est une excellente pratique. Programmer une boucle en déterminant d'abord l'invariant assure presque à coup sûr l'exactitude de la boucle.

La boucle REPEAT ... UNTIL

La deuxième boucle est la boucle de type **répéter** ... **jusqu'à**. Elle s'écrit :

```
10 X=0
20 REPEAT
30   DISP X;X*X
40   X=X+1
50 UNTIL KEY$#""
```

C'est une variante de la précédente. Le test est effectué en fin de boucle, c'est à dire que le corps est parcouru au moins une fois. D'autre part, la condition est l'inverse de celle testée par la boucle précédente.

Pour résumer, la boucle est parcourue au moins une fois, et la sortie se fait lorsque la condition est vraie.

La boucle LOOP ... END LOOP

La dernière forme de boucle, et la plus simple, est la répétition infinie :

```
10 X=0
20 LOOP
30   DISP X;X*X
40   X=X+1
50 END LOOP
```

Dans cet exemple, la variable X est initialisée à 0 avant d'entrer dans la boucle. Le corps de la boucle est formé des instructions 30 et 40, qui affichent le contenu de la variable X et son carré.

La boucle ne contient pas de test de sortie. Elle est parcourue à l'infini. En pratique, l'appui sur [ATTN] nous ramène à la finitude des concepts.

Cependant, cette boucle est intéressante dans deux principaux cas :
- une utilisation simple et rapide, pour un programme fabriqué vite et qui ne sert qu'une fois. Dans cas, un appui sur [ATTN] n'est pas un problème.
- une boucle pour attendre un événement extérieur. Si, par exemple, vous désirez afficher les carrés pendant 70 secondes (pourquoi pas ?), vous placerez un ON TIMER judicieux avant la boucle.

Le complice indispensable de cette boucle est le mot-clef LEAVE que nous allons voir maintenant.

La sortie prématurée avec LEAVE

L'ordre LEAVE permet de sortir prématurément d'une des boucles ci-dessus. On l'utilisera principalement pour des cas exceptionnels.

```
10 X=INT(RND*10)
20 E=0
30 REPEAT
40   INPUT "Devinez le chiffre : ";N
50   IF N<0 THEN LEAVE
60   E=E+1
70 UNTIL X=N
80 DISP "C'était";N;". Essai No";E
```

L'ordre LEAVE est typiquement utilisé pour sortir des boucles infinies LOOP. C'est une facilité à placer à peu près au même niveau que le GOTO : c'est quelquefois la solution la plus simple, mais il ne faut surtout pas en abuser...

LES CHOIX

Le choix simple avec IF

Une des principales raisons de GOTO est le IF standard du HP-71. En effet, si un traitement un peu gros doit être fait dans une des branches d'un IF, il n'y avait jusqu'ici pas de solution autre que l'instruction maudite.

Maintenant, vous pouvez utiliser le IF multi-lignes. Par exemple :

```
10 IF X>=0 THEN
20   Y=SQRT(X)
30   DISP X;Y
40 ELSE
50   DISP X;"est négatif"
60   BEEP 1400,.075
70 END IF
```

Le segment de programme ci-dessus teste la valeur de la variable X. Notez le THEN terminant la ligne. Ceci indique un IF multi-lignes. Nous reviendrons sur la distinction entre le IF standard et le IF de JPC Rom.

Dans le cas où X est positif, on extrait sa racine carrée, et on affiche les deux nombres.

Dans le cas contraire, un message d'erreur est affiché, suivi d'un Biiiiiip de circonstance.

Notez le END IF qui signifie la fin du IF multi-lignes.

Comment distinguer un IF mono-ligne d'un IF multi-lignes ? C'est en fait très simple : un IF mono-lignes contient un THEN directement suivi d'un ordre Basic. Par exemple :

```
10 IF X>=0 THEN Y=SQRT(X) @ DISP X;Y
```

Un IF multi-lignes contient un THEN suivi d'un séparateur de mot-clef. Un tel séparateur est la fin de la ligne, le caractère @ ou encore le début d'une remarque (caractère !). Notre premier exemple aurait pu être écrit comme :

```
10 IF X>=0 THEN @ Y=SQRT(X) @ DISP X;Y
20 ELSE @ DISP "..." @ BEEP 1400,0.75 @ END IF
```

Notez également que le ELSE doit être précédé par un séparateur de mot-clef (début de ligne, @ ou !) et suivi par un séparateur.

Cette structure de choix vous permet une plus grande liberté dans la rédaction de vos programmes : vous n'êtes plus limités par la seule ligne du IF, et vous

pouvez plus facilement rédiger des blocs indépendants.

Le choix multiple avec SELECT et CASE

Le test IF est plutôt restrictif : soit une condition est vraie, soit elle est fausse. Et que se passe-t-il si une question a plusieurs réponses possibles ?

Eh bien, c'est extrêmement simple. Pour vous en convaincre, regardez ci-dessous :

```
110 DISP "Question :"
120 REPEAT              ! peut être remplacé
130   K$=KEY$           ! par la fonction
140 UNTIL K$#""         ! KEYWAIT$
150 SELECT K$
160   CASE "#43"        ! [ATTN]
170     STOP
180   CASE "#47","#48","#50","#51"
190     CALL DEPLACE(K$)
200   CASE "fG"         ! [f][INPUT]
210     CALL INTRO
220   CASE "fZ"         ! [f][EDIT]
230     CALL MODIF
240   CASE "A" TO "Z"
250     CALL LETTRE(K$)
260   CASE ELSE
270     DISP "Erreur"
280     BEEP 1400,.075
290 END SELECT
```

Avouez que c'est lisible ! Si vous n'en êtes pas convaincus, vous pouvez essayer de programmer ça avec des GOTO, pour voir... Et vous n'avez pas encore tout vu ! Prenons un autre exemple :

```
510 X=R^Z/LOG(A(I,J))   ! Compliqué...
520 SELECT X
530   CASE <0,>10
540     DISP "Erreur"
550     BEEP 1400,.075
560   CASE 0,>7.75
570     ...
580   CASE <=2.25
590     ...
500   CASE ELSE
510     ...
520 END SELECT
```

Tout commence par SELECT. Vous sélectionnez la valeur à comparer. Dans le premier cas, il s'agit de K$, X dans le deuxième. Cette valeur selectionnée est testée avec tous les CASE.

La première condition de CASE satisfaite est choisie, et l'exécution reprend au mot-clef juste après le CASE. Si aucune condition de CASE n'est satisfaite, l'exécution continue après le END SELECT à moins qu'il y ait eû un CASE ELSE.

La syntaxe des CASE est difficile à expliquer, mais simple à comprendre. Schématiquement, nous avons :

CASE *élément* , ...
CASE *élément* TO *élément* , ...
CASE *opérateur relationnel élément* , ...

Le premier cas est simplement une valeur à comparer. Si elle est égale à la valeur sélectionnée, l'exécution reprend après le CASE.

Le deuxième cas apparaît dans le premier exemple (ligne 240). Si la touche appuyée est entre "A" et "Z", l'exécution continue après le CASE.

Le troisième cas est utile pour décomposer des intervalles. Par exemple :

```
100 SELECT R
110   CASE <-10,>+10
120     DISP "Valeur aberrante"
130     BEEP 1400,.075
140   CASE <=-0.5
150     DISP "C'est grave"
160   CASE <=-0.1
170     DISP "Ca ne va pas très bien"
180   CASE <=0.1
190     DISP "On fait aller"
200   CASE <=0.5
210     DISP "Il fait beau aujourd'hui"
220   CASE ELSE
230     DISP "Yad'lajoie !"
240 END SELECT
```

Le premier cas (ligne 110) élimine tout de suite les valeurs aberrantes. Le deuxième cas (ligne 140) correspond à l'intervalle [-10,-0.5], le troisième (ligne 160) à l'intervalle]-0.5,-0.1], etc. Le dernier cas (ligne 220) correspond à l'intervalle]0.5,10] puisque les valeurs supérieures à 10 ont été éliminées par le premier cas.

La structure SELECT mérite vraiment son nom de *choix multiple*.

LA SUITE AU PROCHAIN NUMERO

Vous avez peut-être remarqué que la clarté des exemples de cet article tenaient en partie à l'indentation des structures, c'est à dire au bon décalage des instructions pour faire apparaître des blocs.

Le problème est que vous ne pouvez reproduire cette indentation avec les ordres LIST et PLIST.

Pour y remédier, nous vous présenterons dans un prochain numéro une refonte de BLIST et PBLIST (de Jean-Pierre Bondu, voir JPC 38). Cela vous rendra la vie nettement plus agréable.

CONCLUSION

Pour conclure, sachez que STRUC2 est strictement compatible avec Basic 4.0, le Basic utilisé par les « gros » ordinateurs HP (HP9000). Cela ne vous réconforte-t-il pas de savoir que le HP-71 est digne maintenant de ses grands frères ?

Nous espérons que vous saurez profiter de ce Lex pour structurer vos programmes. Et n'hésitez plus à faire publier vos oeuvres dans JPC...

Pierre David (37)
Janick Taillandier (246)

```
        LEX     'STRUC2'

=ARITH  EQU     #061E0
=AVMEMS EQU     #2F594
=BSERR  EQU     #0939A
=CNFLCT EQU     #0BD15
=COMCK+ EQU     #032AE
=CURREN EQU     #2F56C
=CURRST EQU     #2F55D
=D1=AVE EQU     #18651
=DROPDC EQU     #05470
=EOLXC* EQU     #052EC
=EOLXCK EQU     #05405
=EXPEX- EQU     #0F178
=EXPEXC EQU     #0F186
=EXPPAR EQU     #03FD9
=EXPRDC EQU     #05922
=F-RO-3 EQU     #2F8AA
=FINDA  EQU     #023E3
=FIXP   EQU     #02A6E
=IOFNDO EQU     #118C1
=IVEXPe EQU     #02E35
=LDCSPC EQU     #2F6C1
=MEMBER EQU     #1B098
=MEMERR EQU     #0944D
=MPOP1N EQU     #0BD8D
=MPOP2N EQU     #0BD54
=MTHSTK EQU     #2F599
=NTOKEN EQU     #0493B
=NUMCK  EQU     #0369D
=NXTSTM EQU     #08A48
=OBCOLL EQU     #01435
=OUTBS  EQU     #2F58F
=OUTBYT EQU     #02CE8
=OUTELA EQU     #05303
=OUTNBC EQU     #05423
=OUTNIB EQU     #02D28
=POP1S  EQU     #0BD38
=POPUPD EQU     #08F3E
=PRGMEN EQU     #2F567
=PSHGSB EQU     #08F13
=PgmRun EQU     #0000D
=RESPTR EQU     #03172
=REST*  EQU     #03035
=RUNRT1 EQU     #074E7
=S-RO-1 EQU     #2F876
=S-RO-3 EQU     #2F880
=S-R1-3 EQU     #2F890
=STMTDO EQU     #2F891
=STMTD1 EQU     #2F896
=STMTRO EQU     #2F871
=STRTST EQU     #1B1C7
=TBLJMC EQU     #02426
=TKSCN7 EQU     #08A99
=TRACEM EQU     #2F7B0
=TRFROM EQU     #0FE59
=TRTO+  EQU     #0FE7B
=Trace  EQU     #0000F
=WRDSCN EQU     #02C2A
=bSTMT  EQU     #00801
=fBASIC EQU     #0E214
=oFTYPh EQU     #00010
=tCOMMA EQU     #000F1
=tEOL   EQU     #000F0
=tRELOP EQU     #0008A
=tTHEN  EQU     #000F4
=tTO    EQU     #000F3
=tXWORD EQU     #000EF
=uTEST  EQU     #0D435

=tEND2   EQU    66
=tWHILE  EQU    67
=tREPEAT EQU    68
=tUNTIL  EQU    69
=tLEAVE  EQU    70


=tLOOP   EQU    96
=tSELECT EQU    97
=tCASE   EQU    98
=tIF2    EQU    99
=tELSE2  EQU    100


=id      EQU    #E1

=STRUCt EQU     9       Type de GOSUB stack
*
* quartets de reconnaissance de :
*
=qENDL  EQU     0       END LOOP
=qENDS  EQU     1       END SELECT
=qENDI  EQU     2       END IF
```

```
SAUVDO EQU    =STMTDO
CSETYP EQU    =S-R1-3  Type de l'objet (SELECT)

chEW   EQU    0        cherche END WHILE
chEL   EQU    1        ch. END LOOP
chU    EQU    2        ch. UNTIL
chEI   EQU    3        ch. END IF
chES   EQU    4        ch. END SELECT
chE|EI EQU    5        ch. ELSE ou END IF
chC|ES EQU    6        ch. CASE ou END SELECT
chEI-E EQU    7        ch. END IF, saute ELSE
chES-C EQU    8        ch. END SELECT, saute CASE
chL/EW EQU    9        LEAVE cherche END WHILE
chL/EL EQU    10       LEAVE cherche END LOOP
chL/U  EQU    11       LEAVE cherche UNTIL

* Predicats de test
 Pred<  EQU    %0001
 Pred=  EQU    %0010
 Pred>  EQU    %0100
 Pred?  EQU    %1000

        CON(2) =id    ID equivalent
        CON(2) =tEND2
        CON(2) =tLEAVE
        REL(5) nxtlex
        NIBHEX F
        REL(4) 1+TxTbSt
        MSG    MSGTBL
        POLL   0

 *
 * M A I N   T A B L E
 *

        CON(3) (TxEn01)-(TxTbSt)
        REL(5) =END2e
        CON(1) 13

        CON(3) (TxEn02)-(TxTbSt)
        REL(5) =WHILEe
        CON(1) 13

        CON(3) (TxEn03)-(TxTbSt)
        REL(5) =REPEATe
        CON(1) 13

        CON(3) (TxEn04)-(TxTbSt)
        REL(5) =UNTILe
        CON(1) 13

        CON(3) (TxEn05)-(TxTbSt)
        REL(5) =LEAVEe
        CON(1) 13

 *
 * T E X T   T A B L E
 *

TxTbSt
TxEn01 CON(1) 5
       NIBASC 'END'
       CON(2) =tEND2

TxEn05 CON(1) 9
       NIBASC 'LEAVE'
       CON(2) =tLEAVE

TxEn03 CON(1) 11
       NIBASC 'REPEAT'
       CON(2) =tREPEAT

TxEn04 CON(1) 9
       NIBASC 'UNTIL'
       CON(2) =tUNTIL

TxEn02 CON(1) 9
       NIBASC 'WHILE'
       CON(2) =tWHILE
       NIBHEX 1FF

       STITLE TABLE DE MESSAGES

=eNOWHL EQU    3
=eNORPT EQU    4
=eLEAVE EQU    5
=eNOENW EQU    6
=eINVLD EQU    236

MBASE  EQU    3
=eSTRUC EQU    (MBASE)+0    Structure Mismatch
=eBLN  EQU    (MBASE)+1

MSGTBL
       CON(2) 3        Lowest message #
       CON(2) 3        Highest message #

*
       CON(2) 16
       CON(2) =eBLN    Message # 1
       CON(1) 4
       NIBASC '     '
       CON(1) 12

* Structure Mismatch
       CON(2) 44
       CON(2) =eSTRUC  Message # 0
       CON(1) 11
       CON(1) 15
       NIBASC 'Structur'
       NIBASC 'e Mismat'
       CON(1) 1
       NIBASC 'ch'
       CON(1) 12

       NIBHEX FF       Table terminator
```

```
        STITLE PARSE
*
* Syntaxes :
*
*   WHILE <exp>
*   END WHILE
*   LOOP
*   END LOOP
*   REPEAT
*   UNTIL <exp>
*   LEAVE
*   SELECT <exp>
*   CASE <clause> [, <clause>]*
*   CASE ELSE
*       (<clause> ::= <exp> TO <exp> |
*                     <relop> <exp> | <exp>)
*   END SELECT
*   IF <exp> THEN <eol>
*   ELSE (en dehors d'un IF normal)
*   END IF
*


=LEAVEp
=REPEATp
=LOOPp  RTNCC

=UNTILp
=WHILEp GOVLNG =FIXP

 wrdscn GOVLNG =WRDSCN

=END2p  GOSUB  wrdscn
        CON(6) (=tWHILE)~(=id)~(=tXWORD)
        REL(3) ENDWp
        CON(6) (=tLOOP)~(=id)~(=tXWORD)
        REL(3) ENDLp
        CON(6) (=tSELECT)~(=id)~(=tXWORD)
        REL(3) ENDSp
        CON(6) (=tIF2)~(=id)~(=tXWORD)
        REL(3) ENDIp
        CON(2) 0
        GOTO   rest*

*
* Tokenisation du END WHILE / LOOP / SELECT / IF :
*
* Dans la chaine tokenisee (pointee par D0), nous
* avons les 6 quartets pour le codage du END (le
* notre, pas celuyi du systeme), puis un quartet
* de reconnaissance, servant a identifier laquelle
* des 4 structures nous terminons :
*   qENDL = 0    (necessaire)
*   qENDS = 1
*   qENDI = 2
* Si il n'y a pas de quartet de reconnaissance,
* c'est un END WHILE
*

 ENDWp
        D0=D0- 6        On oublie tWHILE
        RTN             Cy := 0

 ENDSp  P=      =qENDS
        GOTO    ENDLp
 ENDIp  P=      =qENDI
 ENDLp
*
* tXWORD id tEND2 tXWORD id tLOOP/SELECT/IF
*                                       ^
*                                       D0
*
        D0=D0- 6        On retire le token reconnu
*
* tXWORD id tEND2 tXWORD id tLOOP/SELECT/IF
*               ^
*               D0
*
        C=P     0       C(0) := qEND L/S/I
        P=      0       c'est plus propre...
 outnib GOVLNG =OUTNIB

*
* CASE ELSE
*
 CS00p  D0=D0- 6        tELSE nous interesse pas
        RTN             Cy := 0 par le D0=D0- 6

************************************************
* CASEp
*
* Syntaxes :
*   CASE ELSE :
*     tXWORD id tCASE
*   <relop> <exp>
*     tRELOP <1 quartet> texp
*   <exp> TO <exp>
*     texp tTO texp
*   <exp>
*     texp
*   CASE <clause> , <clause>
*     tXWORD id tCASE tclause tCOMMA tclause
************************************************

 EXPTYP EQU     =F-R0-3

=CASEp
*
* EXPTYP := 0 ;
*
        CD1EX
        D1=(5) EXPTYP
        C=0     S
        DAT1=C S
        D1=C
*
* ELSE ?
```

```
*
        GOSUB   wrdscn
        CON(6) (=tELSE2)~(=id)~(=tXWORD)
        REL(3) CS00p
        CON(2) 0
*
* analyse des clauses
*
        GOSUB   resptr
 CSp10
*
* <relop> ?
*
        GOSBVL =NTOKEN
        LC(2)  =tRELOP
        ?A#C   B
        GOYES  CSp20    <exp> TO <exp> ou <exp>
*
* CASE <relop> <exp>
*
        GOSBVL =OUTBYT  stocker tRELOP
        C=A    XS       C(XS) := specificateur
        CSR    X
        CSR    X        C(0) := specifier
        GOSUB  outnib   stocker le "specifier"
*
* "CASE <25 @"
*        ^
*        D1
* tXWORD id tCASE tRELOP q
*                          ^
*                          D0
*


        GOTO   CSp30    parse expression & fin
 *
 * CASE <exp1> [ TO <exp2 ]
 *
  CSp20
        GOSUB  resptr
        GOSUB  expck    parse et verifie exp1
 *
 * "CASE 32 TO 127 @"
 *           ^
 *           D1
 * tXWORD id tCASE t32
 *                    ^
 *                    D0
 *
        LC(2)  =tTO
        ?A#C   B
        GOYES  CSp40
        GOSBVL =OUTBYT  On n'incremente pas D1
 *
 * "CASE 32 TO 127 @"
 *           ^
 *           D1
 * tXWORD id tCASE t32 tTO
*                     ^
*                     D0
*
 CSp30  GOSUB  expck
 CSp40
*
* "... , ..."
*       ^
*       D1
* ... t<dernier token reconnu>
*                             ^
*                             D0
*
* A(B) = token a analyser (tCOMMA ou tEOL)
* D0 = ^ flot de sortie
* D1 = ^ passe le token A(B)
*
        GOSBVL =COMCK+
        GOC    CSp10    "," trouvee, on recommence


 resptr GOVLNG =RESPTR


=SELECTp
        GOSBVL =EXPPAR
        ?ST=1  0
        GOYES  ivexp
        GONC   resptr   B.E.T.


************************************************
* expck
*
* But: analyser l'expression pointee par D1, et
*   verifier que le type correspond a celui des
*   expressions precedentes.
* Entree:
*   - D0 = ^ flot de sortie
*   - D1 = ^ flot d'entree
* Sortie:
*   - A(B) = token suivant
*   - P = 0
*   - A(S) = C(S) = type de l'expression
*   - D0 = ^ flot de sortie (passee l'exp)
*   - D1 = ^ passe le 1er token non reconnu
* Appelle: EXPPAR
* Niveaux: 4 (EXPPAR)
* Abime: A-D, R0-R1, ST(0-3,7,11), FUNCD0, F-R0-3
* Detail: verification du type :
*   type := type de l'expression
*   si EXPTYP = 0
*     alors EXPTYP := type
*     sinon si EXPTYP # type alors erreur
*   fin si
* Historique:
*   87/02/28: conception & codage
************************************************
```

```
 expck  GOSBVL =EXPPAR
        ?ST=1  0        not valid expression ?
        GOYES  ivexp
        LC(1)  1        num exp. <==> 1
        ?ST=1  3        not string expression ?
        GOYES  expck1
        C=C+1  A        string exp. <==> 2
*
* C(0) := type de l'expression
*
 expck1 CSRC   W        C(S) := type
        CD1EX
        D1=(5) EXPTYP
        A=DAT1 S        A(S) := (EXPTYP)
        ?A=0   S
        GOYES  expck2
        D1=C            restaurer D1
        ?A=C   S
        RTNYES          expressions meme type
 ivexp  GOVLNG =IVEXPe  "Invalid Expr"

 expck2 DAT1=C S        EXPTYP := type
        D1=C
        RTN


=IF2p
*
* "IF A+B THEN <eol>"
*    ^
*    D1
*
* tXWORD id tIF2
*                ^
*                D0
*
        GOSBVL =NUMCK
*
* "IF A+B THEN <eol>"
*             ^
*             D1
* tXWORD id tIFSTR tA tB t+
*                          ^
*                          D0
*
        GOSUB  resptr
*
* "IF A+B THEN <eol>"
*        ^
*        D1
*
        GOSUB  wrdscn
        CON(2) =tTHEN
        REL(3) THENp
        CON(2) 0
        GOTO   rest*
*
* "IF A+B THEN <eol>"
*      ^
*            D1
* tXWORD id tIFSTR tA tB t+ tTHEN
*                                ^
*                                D0
*
 THENp  D0=D0- 2        On oublie le token de THEN
*
* On fait maintenant le test decisif :
* si on trouve tEOL, t@, t!, c'est a nous !
* sinon, on repasse la main, via REST*, au IF
* interne.
* On ne peut pas utiliser EOLCK, car il accepte la
* presence de tELSE. Hum...
*
        GOSBVL =NTOKEN

        LC(2)  =tEOL
        ?A=C   B
        GOYES  Resptr
        LCASC  '@'
        ?A=C   B
        GOYES  Resptr
        LC(2)  '!'
        ?A=C   B
        GOYES  Resptr
        GOTO   rest*

 Resptr GOTO   resptr
*
* Tokenisation finale :
* "IF A+B THEN <eol>"
*                ^
*                D1
* tXWORD id tIFSTR tA tB t+ tEOL
*                             ^
*                             D0
*

=ELSE2p
        CD1EX           Sauve D1
        D1=(5) =S-RO-3  "IF statement in progress"
        C=DAT1 S
        D1=C            restaure D1
        ?C#0   S        "IF statement in progress"
        GOYES  rest*
        RTN             Ok, on prend (RTNCC)
 rest*  GOVLNG =REST*   C'est pas a nous

        STITLE DECOMPILE

=SELECTd
=UNTILd
=WHILEd GOVLNG =DROPDC

=END2d  A=DAT1 B
        GOSUB  eolxck
        GOC    ENDWd    trouve tEOL (ou equival)
        C=A    A        C(0) := quartet lu
```

```
       D1=D1+ 1
       GOSBVL =TBLJMC
       REL(3) ENDLd
       REL(3) ENDSd
       REL(3) ENDId

ENDId  LCASC  'FI'
       P=     2*2-1
       GOTO   ENDd10
ENDLd  LCASC  'POOL'
       P=     2*4-1
       GOTO   ENDd10
ENDSd  LCASC  'TCELES'
       P=     2*6-1
       GOTO   ENDd10
ENDWd  LCASC  'ELIHW'
       P=     2*5-1
ENDd10 GOSUB  outnbc
       A=DAT1 B
*
* ATTENTION : le code continue !!!
*
=LEAVEd
=REPEATd
=LOOPd
=ELSE2d GOVLNG =OUTELA

=IF2d  GOSUB  exprdc
       LCASC  'NEHT '
       P=     2*5-1
       GOTO   ENDd10

=CASEd GOSUB  eolxck
       GONC   CSd10
       LCASC  'ESLE'
       P=     4*2-1
       GOTO   ENDd10
 CSd10 A=DAT1 X
       LC(2)  =tRELOP
       ?A#C   B
       GOYES  CSd20
*
* <relop> trouve. Il faut le sortir en ASCII
*
       GOSBVL =ARITH
       GOSUB  outnbc
       D1=D1+ 3
       GONC   CSd30    B.E.T.
*
* <relop> n'existe pas. Il faut decompiler une
* expression, puis voir ce qu'il y a apres...
*
 CSd20 GOSUB  exprdc
       LC(2)  =tTO
       ?A#C   B
       GOYES  CSd40    c'est donc tCOMMA ou EOL
*
* TO : on s'attend donc maintenant a une exp.
*
       LCASC  ' OT '
       P=     4*2-1
       GOSUB  outnbc

       D1=D1+ 2
 CSd30 A=DAT1 B
       GOSUB  exprdc
 CSd40 GOSBVL =EOLXC*
*
* Si on est revenu, c'est qu'il y a une tCOMMA
*
       LCASC  ','      on l'envoie...
       P=     2*1-1    ... dans le flot
       GOSUB  outnbc   ... de sortie
       D1=D1+ 2        on l'oublie...
       GOTO   CSd10    ... et on continue

 outnbc GOVLNG =OUTNBC
 exprdc GOVLNG =EXPRDC

       STITLE EXECUTION

************************************************
* SELECT
*
* But: executer l'ordre SELECT
* Historique:
*   87/03/01: conception & codage
*   87/08/29: correction SELECT 1 !
*   87/08/29: correction SELECT 1 / CASE 0 ! / END
************************************************

*
* tokenisation des structures SELECT / END SELECT
*
* tXWORD id tSELECT <exp. alpha ou num.>
*    tXWORD id tCASE tRELOP specifier <exp>
*    tXWORD id tCASE <exp>
*    tXWORD id tCASE <exp> tTO <exp>
*    tXWORD id tCASE                    (CASE ELSE)
* tXWORD id tEND2 qENDS
*

       REL(5) =SELECTd
       REL(5) =SELECTp
=SELECTe
       D0=D0- 6+2
       GOSUB  stoadr   SAUVD0 := ^ Stlen
       D0=D0+ 6+2
       GOSBVL =EXPEX-  Evaluer le SELECT
*
* A(W) := top 16 nib. of MS
*
       LCHEX  9        Legal BCD digit
       ?A<=C  P
       GOYES  SL020    Real number
       LCHEX  0E       Complex number signature
```

```
        ?A=C   B
        GOYES  SL020    Complex number
*
* Nous sommes dans le cas "string" ou autre...
*
 SL010  GOSBVL =POP1S   Erreur si "autre..."
        GOSUB  stoMS    Proteger la chaine
        D0=(5) =STMTRO
        CD1EX           C(A) := ^ dernier caract.
        C=C+A  A        C(A) := ^ premier caract.
        DAT0=C A        S-R0-0 := ^ chaine
        D0=D0+ 5
        DAT0=A A        S-R0-1 := len en quartets
        C=0    S        Type := alpha
        GONC   SL030    B.E.T.
*
* Cas reel ou complexe
*
 SL020  GOSUB  stoMS
        C=0    S
        C=C+1  S
*
* C(S) = type (0 = alpha / 1 = numerique)
* AVMEME = pointeur de M.S.
*
 SL030  D0=(5) CSETYP
        DAT0=C S        CSETYP := type (alpha/num)

*
* Debut de la boucle de recherche de CASE ou
* END SELECT
*
 SL100  GOSUB  recadr
        GOSUB  eol

        P=     chC|ES
        GOSUB  cherche
        GOSUB  stoadr   SAUVD0 := adresse trouvee
        D0=D0+ 6        D0 := ^ tCASE ou tEND2
        LC(2)  =tEND2
        A=DAT0 B
        ?A=C   B
        GOYES  sl900    END SELECT seulement

        D0=D0+ 2        D0 := ^ apres tCASE

        A=DAT0 B
        GOSUB  eolxck
        GONC   SL150    Il y a quelque chose.
 sl900  GOTO   SL900    CASE ELSE
*
* D0 = ^ debut de ce qu'il y a a analyser
*
 SL150
*
* Il y a quelque chose a analyser
*
        LC(2)  =tRELOP
        ?A#C   B
        GOYES  SL200    <exp> ou <exp> TO <exp>
*
* tRELOP specifier <exp>
*
        D1=(5) =S-R0-3
        D0=D0+ 2
        C=DAT0 S        C(S) :=specifier
        DAT1=C S        S-R0-3 := predicat
        D0=D0+ 1
        GOSUB  evalMS
        CD0EX
        D0=(5) =S-R0-3
        C=DAT0 S
        D0=C
        P=C    15
        GOSUB  compar
        GONC   SL300
        GOC    SL900    B.E.T.
*
* <exp>
* <exp> TO <exp>
*
 SL200  GOSUB  evalMS
        A=DAT0 B
        LC(2)  =tTO
        ?A=C   B
        GOYES  SL250
*
* <exp>
*
        P=     Pred=
        GOSUB  compar
        GONC   SL300
        GOC    SL900    B.E.T.
*
* <exp> TO <exp>
*
 SL250  P=     (Pred>)+(Pred=)
        GOSUB  compar
        GONC   SL290
        D0=D0+ 2        On passe tTO
        GOSUB  evalMS
        P=     (Pred<)+(Pred=)
        GOSUB  compar
        GONC   SL300
        GOC    SL900    B.E.T.

 SL290  D0=D0+ 2        On passe tTO
        GOSUB  evalMS   Evaluation de <exp2> sans
          *             prendre le resultat
*
* Fin de l'evaluation. Si on est arrive jusqu'ici,
* toutes les conditions etaient fausses,
*
 SL300  A=DAT0 B
        LC(2)  =tCOMMA
        ?A#C   B
```

```
        GOYES  sl100    c'est donc tEOL
        D0=D0+ 2        on passe la tCOMMA, et
        A=DAT0 B        on recommence
        GOTO   SL150
 sl100  GOTO   SL100
*
* Sortie et branchement. On a trouve :
* - une clause vraie, ou
* - CASE ELSE, ou
* - END SELECT
* Dans tous les cas, SAUVD0 = adresse de debut de
* la ligne (Stlen).
*
 SL900  GOSUB  recadr
        GOTO   goeol


************************************************
* compar
*
* But: comparer l'objet pointe par D1 et l'objet
*   pointe par AVMEME. Traite les verification de
*   type (numerique / alpha).
* Entree:
*   - P = predicat de comparaison
*   - D1 = ^ objet2
*   - AVMEME = ^ objet1
*   - si CSETYP = 1 : type(objet1) = num
*   - si CSETYP = 0 : type(objet1) = alpha, et
*       S-R0-0 = adresse debut de objet1
*       S-R0-1 = longueur en quartets
* Sortie:
*   - Cy = 1 : vrai
*   - Cy = 0 : faux
* Abime: A-D, D1, R0-R3
* Appelle: uTEST, FPOLL, STRTST
* Niveaux: 4 (FPOLL + pCMPLX)
* Note: D0 est sauvegarde dans STMTD1
* Historique:
*   87/03/01: conception & codage
************************************************


 s1=2   EQU    0
 s1>2   EQU    1

 compar
*
* Sauvegarde P dans C(S)
*
        C=P    15       C(S) := Predicat
        P=     0        Propre et net...
*
* Sauvegarde de D0
*
        CD0EX
        D0=(5) =STMTD1
        DAT0=C A
        D0=(5) CSETYP   Type de l'objet
        A=DAT0 S        A(S) := type de objet1
*
* CSETYP = 1 si objet = num
* CSETYP = 0 si objet = str
*
        A=A-1  S        Cy := 1 si (obj1 = str)
        GOC    cmpstr
*
* Comparaison numerique
*
 cmpnum R1=C            R1(S) := predicat
        GOSBVL =MPOP2N
*
* R0   = I(objet1)
* A(W) = R(objet1)
* R2   = I(objet2)
* C(W) = R(objet2)
*
        SETHEX
        GOC    cmpcmp   compare complex
        CR1EX
        P=C    15       P = predicat
        CR1EX
        GOSBVL =uTEST
        SETHEX
        P=     0
        GOTO   cmpfin   et puis voila...

*
* Comparaison de complexes
*
 cmpcmp
*
* Pour le moment... on ne supporte pas les
* complexes.
*
        GOVLNG =CNFLCT  Conflict !!!
*
* Comparaison alphanumerique
*
 cmpstr D=C    S        D(S) := predicat de comp.
        GOSBVL =POP1S
        CD1EX           C(A) := ^ fin objet2
        C=C+A  A        C(A) := ^ debut objet2
        D1=C            D1 := ^ 2eme chaine
        D0=(5) =S-R0-1  D0 := ^ longueur objet1
        C=DAT0 A        C(A) := longueur objet1
        ST=0   s1>2     len1 < len2
        ST=0   s1=2     len1 # len2
        A=0    S
        ?C=0   A        len1 = 0 ?
        GOYES  cmps00
        ?A=0   A        len2 = 0 ?
        GOYES  add>     obj1 > ""
        ?C#A   A
        GOYES  cmps10
        ST=1   s1=2     len1 = len2
```

```
        GONC    cmps20    B.E.T.
*
* Longueur de obj1 = 0
*
 cmps00 ?A=0    A
        GOYES   add=
        GONC    add<      B.E.T. (len1 < len2)


 cmps10 ?C<A    A         ?len1 < len2
        GOYES   cmps20
        ST=1    s1>2      len1 > len2
        C=A     A
*
* C(A) = longueur min
* ST(s1>2) = 1 si (len1 > len2)
* ST(s1=2) = 1 si (len1 = len2)
* D0 = ^ obj1
* D1 = ^ obj2
*
 cmps20 D0=D0- 5          D0=(5) =S-R0-0
        A=DAT0 A
        D0=A              D0 := ^ objet1
        GOSBVL =STRTST
        P=      0
        A=0     S         A(S):= predicat de sortie
        GOC     cmps50    Strings not equal
*
* Les chaines sont egales jusqu'au minimum des
* longueurs.
*
        ?ST=0   s1=2
        GOYES   cmps30
 add=   LC(1)   Pred=
        CSRC
        A=A!C   S
        GOTO    cmps90    goto Bilan


 cmps30 ?ST=1   s1>2
        GOYES   add>


 add<   LC(1)   Pred<
        CSRC
        A=A!C   S
        GOTO    cmps90
*
* Les chaines ne sont pas egales.
*
 cmps50 A=DAT0 B          A(B) := premier de objet1
        C=DAT1 B          C(B) := premier de objet2
        ?A<C   B          obj1 < obj2
        GOYES  add<

 add>   LC(1)   Pred>
        CSRC
        A=A!C   S
*
* Bilan :
*   A(S) = predicat en sortie
*   D(S) = predicat en entree
*
 cmps90 C=D     S
        A=A&C   S
        ?A#0    S
        GOYES   cmpfin    Cy := 1
 cmpfin
*
* Restauration de D0 sans perturber la Cy
*
        D0=(5) =STMTD1
        C=DAT0 A
        D0=C
        RTN

************************************************
* CASE
*
* But: executer l'ordre CASE
* Hstorique:
*   87/03/01: conception & codage
************************************************

        REL(5) =CASEd
        REL(5) =CASEp
=CASEe
        D0=D0- 8
        GOSUB   eol
        P=      chES-C
        GOSUB   cherche
        GOTO    goeol

************************************************
* LEAVE
*
* But: executer l'ordre LEAVE
* Hstorique:
*   87/03/01: conception & codage
************************************************

        REL(5) =LEAVEd
        REL(5) =LEAVEp
=LEAVEe
        GOSUB   stoadr
        GOSBVL =POPUPD
        GOC     struce    Pile vide
        LC(1)   =STRUCt
        CSRC              C(S) := type de retour
        ?D#C    S
        GOYES   struce
*
* Le type est reconnu. Quel est le token qu'il faut
* chercher ?
*
        C=D     A         C(A) := ^ apres le token
```

```
        D0=C
        D0=D0+ 4        D0 := ^ token
        A=DAT0 B
        GOSBVL =FINDA
        CON(2) =tWHILE
        REL(3) LV10
        CON(2) =tLOOP
        REL(3) LV20
        CON(2) =tREPEAT
        REL(3) LV30
        NIBHEX 00
 struce GOTO   STRUCe   "Structure Mismatch"

*
* REPEAT / UNTIL
*
 LV30   P=P+1
*
* LOOP / END LOOP
*
 LV20   P=P+1
*
* WHILE / END WHILE
*
 LV10   C=P    0        C(0) := chX
        P=     0
        D=C    P        D(0) := chX
        LC(1)  (chL/EW)
        C=C+D  P        C(0) := chL/EW + P
        P=C    0        P := chL/EW, chL/EL, chL/U
        GOSUB  recadr
        GOSUB  cherche  Chercher la fin de struct.
        GOTO   goeol

************************************************
* ENDe
*
* But: traiter les ordres END WHILE, END LOOP,
*   END SELECT et END IF
* Entree:
*   - D0 = ^ quartet de reconnaissance
* Sortie:
*   - voir chaque branche
* Utilise: voir chaque branche
* Abime: voir chaque branche
* Niveaux: voir chaque branche
* Historique:
*   87/02/27: reconception
************************************************

        REL(5) =END2d
        REL(5) =END2p
 =END2e
        A=DAT0 B        token suivant / quartet
        GOSUB  eolxck
        GOC    ENDWe    tEOL ==> END WHILE
*
* A(0) = quartet de reconnaissance
*
        LC(1)  =qENDL
        ?A=C   P
        GOYES  ENDLe    END LOOP
        GONC   nxtstm   B.E.T.  END SELECT/IF

************************************************
* ENDWe
*
* But: proceder a l'execution de END WHILE
* Historique:
*   87/02/27: reconception
*   87/08/02: correction de "10 WHILE 1 !"
************************************************

 ENDWe  LC(2)  =tWHILE
        GOSUB  pop
        GOSUB  eval     Cy := 0 si e=0
        GONC   nxtstm

        GOSUB  recadr   D0 := adresse depilee
        GOSUB  push
        GOTO   go

 nxtstm GOVLNG =NXTSTM

************************************************
* ENDLe
*
* But: proceder a l'execution de END LOOP
* Historique:
*   87/02/27: reconception
*   87/08/02: correction de "10 LOOP !"
************************************************

 ENDLe  LC(2)  =tLOOP
        GOSUB  pop
        GOSUB  push
        GOTO   go

************************************************
* WHILEe
*
* Executer le statement WHILE
* Entree:
*   - D0 = ^ expression a analyser
* Sortie:
*   par go ou NXTSTM
* Historique:
*   87/02/28: reconception & codage
************************************************

*
* Tokenisation d'une boucle WHILE / END WHILE :
*
* tXWORD id tWHILE <exp>
*    <corps de la boucle>
* tWORD id tEND2
```

AH ! VOUS ECRIVEZ

Vous vous sentez en verve, mais vous ne savez pas sous quelle forme "l'équipe de rédaction" souhaite recevoir votre prose. C'est ici que se trouvent les réponses à vos questions.

Dans la mesure du possible, vous devez nous envoyer vos écrits sur support magnétique (carte, cassette ou disquette). Soyez sans crainte, nous vous retournerons vos biens après copie.

Si vous ne pouvez pas utiliser de support magnétique, ou ne pouvez vous rendre aux réunions, alors et alors seulement faites le sur papier.

Que ce soit sur une feuille de papier, ou sur support magnétique, ne dépassez pas 50 caractères par ligne.

Pour nous épargner du travail, insérez dans votre texte les commandes de formattage suivantes (et non les commandes du formatteur HP) :

"^" centre un titre, par exemple :
^TITRE

"\" (CHR$(92)) marque le début et la fin d'un paragraphe. Par exemple :

\Début de paragraphe exprimant le contenu de vos idées qui, même si vous en doutez, intéressera certains des membres du Club. Surtout si vous vous sentez débutant. Les articles pour débutants écrits par des débutants sont ceux qui manquent le plus. Fin de paragraphe.\

N'oubliez pas de mettre les accents. Utilisez le jeu de caractères Roman8. Les possesseurs de HP71 utiliseront les redéfinitions de touches ci-dessous, ainsi que le fichier CHARLEX listé dans le coin des Lhex.

Jean-Jacques Dhénin (177)

```
DEF KEY 'fW', CHR$(197);      (é)
DEF KEY 'fE', CHR$(193);      (ê)
DEF KEY 'fR', CHR$(201);      (è)
DEF KEY 'fY', CHR$(203);      (ù)
DEF KEY 'fU', CHR$(195);      (û)
DEF KEY 'fI', CHR$(209);      (î)
DEF KEY 'fO', CHR$(194);      (ô)
DEF KEY 'f/', CHR$(92);       (\)
DEF KEY 'fA', CHR$(192);      (â)
DEF KEY 'fS', CHR$(200);      (à)
DEF KEY 'fD', CHR$(205);      (ë)
DEF KEY 'fJ', CHR$(207);      (ü)
DEF KEY 'fK', CHR$(221);      (ï)
DEF KEY 'f*', CHR$(124);      (|)
DEF KEY 'fC', CHR$(131);      (ç)
```

PPC PARIS SE REUNIT
UNE FOIS PAR MOIS

Comme vous le savez peut être déjà, PPC Paris se réunit une fois par mois, en plein coeur de Paris. Amenez votre matériel, votre bonne volonté et vos idées ! Plus vous en apporterez, et plus vous en trouverez chez vos collègues de PPC.

Ces réunions se déroulent de manière très libre, aucun ordre du jour, discussion ou autre n'étant imposé. Un membre du bureau est toujours présent. Ainsi, si vous désirez remettre votre article tout frais au Journal, si vous avez des suggestions à faire, si vous voulez vous procurer des anciens numéros de JPC, ce sera en principe toujours possible.

Si donc cela vous intéresse, n'hésitez plus un seul instant, venez nous rejoindre tous les premiers samedis de chaque mois (sauf en période de vacances scolaires) au :

Centre de Jeunesse et de Loisirs Jean Verdier
11 rue de Lancry
75010 Paris

et en montant au deuxième étage, vous entendrez des éclats de rire et des discussions passionnées vers la salle 215. Attention, toutefois, de venir entre 16 et 19h.

Pour l'accès en métro, trois possibilités s'offrent à vous :
- Métro Strasbourg Saint Denis :
Sortie porte St Martin / Bd St Denis, coté pairs
- Métro République :
Sortie Bd St Martin, coté pairs
- Métro Jacques Bonsergent :
Sortie Bd Magenta, coté impairs.

Ah, j'oubliais ! JPC est (souvent) distribué en avant première lors de ces réunions... A bon entendeur, salut !

Les dates des prochaines réunions sont :
Samedi 5 mars 1988
Samedi 16 avril 1988
Samedi 7 mai 1988
Samedi 4 juin 1988

Pierre David (37)

NOUS EN AVONS

La coopérative du Club dispose :

- de lecteurs de cartes magnétiques pour HP-71, neufs, dans leur boîte d'origine, avec 5 cartes magnétiques, pour 500 F (port compris),

- des anciens numéros de JPC, au prix de 40 F + 7,40 F de frais d'affranchissement,

- d'une année complète de numéros de JPC (février à décembre / janvier) pour 300 F (offre spéciale) port compris,

- des I.D.S. du module Forth / Assembleur (listing interne commenté par HP) pour 250 F (port compris),

- des VASM pour HP-41 (listings des Roms internes commenté par HP) pour 300 F (port compris).

- de manuels de service du HP-41 au prix de 75 F (port compris).

- de manuels de service du HP-75 au prix de 75 F (port compris).

Si vous souhaitez d'autres renseignements, n'hésitez pas à nous contacter.

VOUS EN VOULEZ

Nom :
Prénom :
No de membre :
Adresse :

Commande :

	Qté	Prix Unitaire	Prix Total
lecteur de cartes pour HP-71	x	500 F	
anciens numéros de JPC	x	47,40 F	
année complète de JPC	x	300 F	
I.D.S. du module Forth	x	250 F	
V.A.S.M.	x	300 F	
Manuel de service pour HP-41	x	75 F	
Manuel de service pour HP-75	x	75 F	
		Total	F

Préciser éventuellement les
numéros de JPC commandés :

PPC PARIS

Association régie par la loi de 1901, enregistrée
à Paris le 2 décembre 1982 sous le numéro 82/3240

BULLETIN D'ADHESION

Nom |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Prénom |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_| Date de naissance |_|_|/|_|_|/|_|_|

Adresse |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Code Postal |_|_|_|_|_| Ville |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Téléphone Domicile |_|_|_|_|_|_|_|_|_|_|_|_| Bureau |_|_|_|_|_|_|_|_|_|_|_|_|_|_|

Profession ______________________________

Intérêts ______________________________

____

Matériel HP en votre possession ______________________________

____

Autre matériel informatique ______________________________

____

Comment avez-vous connu PPC Paris ? ______________________________

____

Que recherchez-vous au sein de PPC Paris ? ______________________________

____

____

La Loi No 78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés, garantit à toute personne justifiant de son identité un droit d'accès et de rectification auprès des services ou organismes chargés de mettre en oeuvre des traitements informatiques comportant des informations nominatives le concernant.

Je souhaite adhérer au club PPC Paris conformément aux statuts de l'Association. Au mieux de ma connaissance, je déclare avoir le droit de fournir tous les programmes et informations que je vous enverrai (sans enfreindre des obligations de secret à l'égard d'autres personnes ou organismes) pour publication dans le Journal de liaison, sans obligations ni responsabilité d'aucune sorte (en cas d'utilisation frauduleuse) de la part des dirigeants de PPC-Paris.

Date |_|_|/|_|_|/19|_|_|

Signature, précédée de la mention "Lu et Approuvé" ______________

Le montant de la cotisation s'élève à 350.00 F pour un an.
Etudiants: 300.00 F (justificatif indispensable)

Paiement effectué le |_|_|/|_|_|/19|_|_| à l'ordre de PPC Paris,
par [ ] chèque bancaire, [ ] chèque postal ou [ ] mandat lettre

Veuillez envoyer toute correspondance à :
PPC Paris, BP 604, 75028 Paris Cedex 01, France

```
*
        REL(5) =WHILEd
        REL(5) =WHILEp
=WHILEe
        GOSUB  stoadr
        GOSUB  eval
        GONC   WHL10
*
* expression # 0 (vrai) : on continue en sequence
*
        GOSUB  recadr   D0 := ^ debut expression
        GOSUB  push
        GOTO   nxtstm
*
* expression = 0 (faux) : on cherche END WHILE
* (D0 = ^ apres la fin de l'expression)
*
 WHL10  P=     chEW     chercher END WHILE
        GOSUB  cherche
        GOTO   goeol

************************************************
* LOOP / REPEAT
*
* But: executer les ordres REPEAT et LOOP
* Entree:
* Sortie: par NXTSTM
* Appelle: push
* Historique:
*   87/02/28: conception & codage
************************************************

*
* Tokenisation d'une boucle LOOP / END LOOP
*
* tWORD id tLOOP
*   <corps de la boucle>
* tXWORD id tEND2 qENDL
*

*
* Tokenisation d'une boucle REPEAT / UNTIL
*
* tXWORD id tREPEAT
*    <corps de la boucle>
* tXWORD id tUNTIL <exp>
*
        REL(5) =REPEATd
        REL(5) =REPEATp
=LOOPe
=REPEATe
        GOSUB  push
        GOTO   nxtstm

************************************************
* UNTILe
*
* But: executer l'ordre UNTIL
* Entree:
*   - D0 = ^ debut de l'expression
* Sortie:
*   - par go ou nxtstm
* Appelle: pop, push, eval, recadr, eval
* Historique:
*   87/02/28: conception & codage
*   87/08/02: correction de "10 REPEAT !"
************************************************

        REL(5) =UNTILd
        REL(5) =UNTILp
=UNTILe
*
* Sauver D0 pour eval
*
        CD0EX
        D0=(5) =STMTD1
        DAT0=C A        STMTD1 := ^ debut de l'exp
*
* Depiler l'adresse de retour et verifier REPEAT
*
        LC(2)  =tREPEAT
        GOSUB  pop
*
* Evaluer l'expression
*
        D0=(5) =STMTD1
        C=DAT0 A
        D0=C
        GOSUB  eval
        GOC    Nxtstm
*
* On boucle :
*
        GOSUB  recadr   D0 := ^ fin du REPEAT
        GOSUB  push
        GOTO   go       GOTO "fin du REPEAT"

 Nxtstm GOTO   nxtstm

************************************************
* IF2
*
* But: executer l'ordre IF
* Hstorique:
*   87/03/01: conception & codage
************************************************

*
* Tokenisation d'une structure IF / END IF
*
* tXWORD id tIF2 <exp>
*   <alternant si vrai>
* tXWORD id tELSE2
*   <alternant si faux>
* tXWORD id tEND2 qENDI
*
```

```
* ou encore :
*
* tXWORD id tIF2 <exp>
*   <alternant si vrai>
* tXWORD id tEND2 qENDI
*
        REL(5) =IF2d
        REL(5) =IF2p
=IF2e
        GOSUB  eval
        GOC    Nxtstm
        P=     chE|EI
        GOSUB  cherche
        GOTO   goeol

************************************************
* ELSE2
*
* But: executer l'ordre ELSE
* Hstorique:
*   87/03/01: conception & codage
************************************************

        REL(5) =ELSE2d
        REL(5) =ELSE2p
=ELSE2e
        P=     chEI
        GOSUB  cherche
        GOTO   goeol

************************************************
* STRUCe
*
* But: generer une erreur et retourner a Basic
* Entree: -
* Sortie: par BSERR
* Historique:
*   87/02/28: conception & codage
************************************************

 STRUCe P=     0
        LC(4)  (=id)~(=eSTRUC)
        GOVLNG =BSERR

 eolxck GOVLNG =EOLXCK

************************************************
* go, goeol
*
* But: revenir a Basic, en faisant un GOTO sur
*   le statement suivant D0 (go), ou sur le token
*   suivant la ligne dont la longueur est pointee
*   par D0 (goeol).
* Entree:
*   go :
*   - SAUVD0 = ^ Stlen du debut de la ligne courante
*   goeol:
*   - D0 = ^ Stlen du debut de la ligne courante
* Sortie: par RUNRT1
* Algorithme:
*   si execution au clavier
*     alors
*       si adresse de destination n'est pas dans
*          dans bSTMT
*         alors "Structure Mismatch"
*       fin si
*   fin si
*   si trace
*     alors trace
*   fin si
*   RUNRT1
* Historique:
*   87/02/27: conception & codage
*   87/03/02: integration de goeol
*   87/03/04: correction du bug "trace"
*   87/08/02: modification de go, bug "10 LOOP !"
************************************************


 go     GOSUB  recadr
        D0=D0- 6+2      6 : token, 2 : Stlen


 goeol  GOSUB  eol


*
* D(A) := D0
*
        CD0EX
        D0=C
        D=C    A
*
* si execution au clavier :
*
        ?ST=1  =PgmRun
        GOYES  go10
        LC(3)  =bSTMT   Statement buffer (clavier)
        GOSBVL =IOFND0  D1 := ^ statement buffer
*
* Le buffer existe toujours. Du moins, on espere !
*
        CD1EX           C(A) := ^ buffer
        ?D<C   A        dest < debut buffer ?
        GOYES  STRUCe
        C=C+A  A        C(A) := ^ fin buffer
        ?D>C   A        dest > fin buffer ?
        GOYES  STRUCe
*
* trace ?
*
 go10   ?ST=0  =Trace
        GOYES  runrt1
*
* le mode TRACE est actif
*
```

```
*
* Ce qui suit est pompe dans trflck (#0FE18)
* (avec modificatio)
*
        ?ST=0   =PgmRun
        GOYES   runrt1
        D1=(5) =TRACEM
        P=      0
        LCHEX   2
        A=DAT1 B
        A=A&C   P
        A=A-1   P
*
* Jusqu'au 4 mars 1987, l'absence de cette
* instruction declenchait abusivement le mode
* TRACE FLOW lors de l'execution d'une fonction de
* STRUCLEX
*
        GOC     runrt1
*
* Elle est toute petite, hein ?
*
        D1=(5) =CURRST
        C=DAT1 A
        D1=C
        D1=D1+ (=oFTYPh)-1
        A=DAT1 A
        ASR     A
        LC(5)   =fBASIC
        ?A#C    A
        GOYES   runrt1
*
* code Taillandier inspire par les dieux d'HP
*
        CDOEX
        R2=C
        D0=C
*
* Est-on a la fin du fichier ?
*
        D1=(5) =CURREN
        A=DAT1 A        A(A) := CURREN
        C=C+1   A
        C=C+1   A       C(A) := ^ apres le tEOL
        ?C>=A   A
        GOYES   runrt1
*
* On n'est pas hors du fichier. On peut tracer en
* toute quietude : le numero de ligne est valide.
*
        GOSBVL =TRFROM  partie FROM de trace
        C=R2
        D0=C
        GOSBVL =TRTO+   partie TO de trace
        A=R2
        D0=A            restaure D0 et fin
 runrt1 GOVLNG =RUNRT1
```

```
************************************************
* eval
*
* But: evaluer l'expression pointee par D0, et
*   renvoyer le resultat sous forme booleenne (Cy)
* Entree:
*   - D0 = ^ expression tokenisee
* Sortie:
*   - P = 0
*   - D0 = ^ apres l'expression tokenisee
*   - Cy = 0 : expression = 0
*   - Cy # 0 : expression # 0
* Appelle: EXPEX-, MPOP1N
* Niveaux: 5 (EXPEX-)
* Abime: A-D, D0, D1, R0-R4, FUNCtion scratch
* Historique:
*   87/02/27: ajout de documentation
*   87/02/27: clarification du code
************************************************

 eval   GOSBVL =EXPEX-  vide la M.S.
        GOSBVL =MPOP1N  depile le nombre (C ou R)
        SETHEX
        A=0     S       ne prend pas le signe (-0)
        GONC    eval10  nombre reel
*
* nombre complexe
*
        ?A#0    W
        RTNYES
        A=R0
        A=0     S
 eval10 ?A#0    W
        RTNYES
        RTN

************************************************
* push
*
* But: mettre D0 dans la pile des GOSUB.
* Entree:
*   - D0 = valeur a empiler
* Sortie:
*   - D0 = valeur en entree
* Appelle: PSHGSB
* Niveaux: 4 (PSHGSB)
* Abime: A-D, D1
* Historique:
*   87/02/27: conception & codage
************************************************

 push   LC(1)   =STRUCt C(0) := type de l'adresse
        CSRC            C(S) := return type
        D0=D0- 6        D0 := ^ token
        ADOEX           A(A) := adresse du token
        GOSBVL =PSHGSB
        CSRC            C(A) := adresse empilee
        D0=C            D0 := ^ adresse du token
```

```
       D0=D0+ 6          D0 := adresse en entree
       RTN


*************************************************
* pop
*
* But: depile une adresse sur la GOSUB stack,
*   verifie le type de l'adresse (tSTRUC), pointe
*   D0 a cette adresse et verifie que le token
*   pointe est le meme que C(B).
* Entree:
*   - C(B) = token a verifier (2 q. seulement)
* Sortie:
*   - D0 = ^ adresse depilee (passe le token)
* Appelle: POPUPD
* Niveaux: 2 (POPUPD)
* Utilise: A-D, D0, D1, R0
* Detail:
*   si type de l'adresse # tSTRUC
*     alors "Structure Mismatch"
*   si token pointe n'est pas bon
*     alors "Structure Mismatch"
* Historique:
*   87/02/27: conception & codage
*************************************************

 pop    R0=C              R0(B) := token a verifier
        GOSBVL =POPUPD
        GOC    STRUce     pile vide
        LC(1)  =STRUCt    type de retour
        CSRC              C(S) := STRUCt
        ?D#C   S          ce qui est depile ne nous
        GOYES  STRUce       appartient pas...
*
* le type est reconnu, c'est a nous. Est-ce qu'on
* pointe sur la bonne structure ?
*
        C=D    A
        D0=C               D0 := ^ adresse de retour
        A=0    W
        A=DAT0 6           A(W) := token lu
        D0=D0+ 6           D0 := ^ apres le token

        GOSUB  stoadr      SAUVD0 := D0 + 6

        C=R0               C(B) := token
        B=C    B           B(B) := token
        C=0    W
        C=B    B           C(W) := token
        CSL    W
        CSL    W
        CSL    W
        CSL    W           C(5-4) := token
        LC(4)  (=id)~(=tXWORD)
        ?A=C   W
        RTNYES             Tokens identiques : OK
 STRUce GOTO   STRUCe


*************************************************
* recadr
*
* But: restaurer dans D0 l'adresse sauvee par pop.
* Entree: -
* Sortie:
*   - D0 = adresse sauvee par pop
* Niveaux: 0
* Abime: C(A), D0
* Historique:
*   87/02/28: conception & codage
*************************************************


 recadr D0=(5) SAUVD0
        C=DAT0 A
        D0=C
        RTN

*************************************************
* stoadr
*
* But: sauver D0 dans SAUVD0
* Entree:
*   - D0 = valeur a sauver
* Sortie:
*   - D0 inchange
* Niveaux: 0
* Abime: C(A)
* Historique:
*   87/02/28: conception & codage
*************************************************

 stoadr C0EX
        D0=(5) SAUVD0
        DAT0=C A
        D0=C
        RTN

*************************************************
* stoMS
*
* But: sauver D1 dans LDCSPC
* Entree:
*   - D1 = valeur a sauver
* Sortie:
*   - D0 inchange
* Niveaux: 0
* Abime: C(A)
* Historique:
*   87/03/01: conception & pompage sur stoadr
*************************************************


 stoMS  C1EX
        D1=(5) =LDCSPC
        DAT1=C A
        D1=C
        RTN
```

```
************************************************
* evalMS
*
* But: Evaluer l'expression pointee par D0, et la
*   ranger dans la pile a l'adresse contenue dans
*   LDCSPC.
* Entree:
*   - D0 = ^ debut de l'exp. tokenisee
*   - LDCSPC = ^ M.S.
* Sortie:
*   - D0 = ^ apres l'expression (token non util.)
*   - D1 = ^ M.S.
*   - A(W) = 16 quartets au sommet de la M.S.
* Niveaux: 5
* Abime: A-D, R0-R4, ST, D0, D1, FUNC scratch
* Historique:
*   87/03/01: conception & codage
*   87/03/02: modification pour integration EXPEXC
************************************************

 evalMS D1=(5) =LDCSPC
        C=DAT1 A
        D1=(4) =MTHSTK
        DAT1=C A
        GOVLNG =EXPEXC

************************************************
* eol
*
* But: calculer l'adresse de la fin du statement
*   courant (adresse de tEOL ou t@) connaissant
*   l'adresse du debut de la ligne (^ Stlen).
* Entree:
*   - D0 = ^ longueur de la ligne (Stlen)
* Sortie:
*   - D0 = ^ t@ ou tEOL ou...
* Abime: A(A), C(A), D0
* Historique:
*   87/03/01: extraction de "goeol"
************************************************

 eol    A=0    A
        A=DAT0 B        A(B) := longueur
        CD0EX           C(A) := ^ longueur
        C=C+A  A        C(A) := ^ fin de la ligne
        D0=C
        RTN

 STRuce GOTO   STRUCe

************************************************
* cherche
*
* But: chercher pour l'objet specifie par P
* Entree:
*   - P = quartet de recherche
*       chEW   : chercher END WHILE
*       chEL   : END LOOP
*       chU    : UNTIL
*       chEI   : END IF
*       chES   : END SELECT
*       chE|EI : ELSE ou END IF
*       chC|ES : CASE ou END SELECT
*       chEI-E : END IF, autorise ELSE
*       chES-C : END SELECT, autorise CASE
*       chL/EW : END WHILE, autorise (*)
*       chL/EL : END LOOP, autorise (*)
*       chL/U  : UNTIL, autorise (*)
*       (*) = CASE, ELSE, END SELECT, END IF
*   - D0 = tEOL ou t@ du statement courant
* Sortie:
*   si l'objet a ete trouve
*   - D0 = ^ Stlen du statement reconnu
* Abime: A-D, D0, D1, P, SAUVD0
* Niveaux:
* Appelle: chpush, chpop, FINDA, TBLJMC, stoadr,
*   recadr, TKSCN7
* Historique:
*   87/03/01: conception & codage
************************************************

*
* Version recursive :
*
* SUB cherche (P)
*   T1$ = premier token a chercher (selon P)
*   T2$ = deuxieme token a chercher (si il y a)
*   LOOP
*     TKSCN7
*     SELECT XWORD
*       CASE "LOOP","WHILE","REPEAT","SELECT","IF"
*         P= "ch" & (token final correspondant)
*         CALL cherche (P)
*       CASE T1$, T2$
*         LEAVE
*       CASE "END WHILE/LOOP/SELECT/IF","UNTIL",
*             "CASE","ELSE"
*         "Structure Mismatch"
*     END SELECT
*   END LOOP
* END SUB
*

*
* Version derecursivee :
*
* cherche:
* initialiser la pile
* cher:
*   T1$ = premier token a chercher (selon P)
*   T2$ = deuxieme token a chercher (si il y a)
*   TKNSCN7
*   LOOP
*     SELECT XWORD
*       CASE "LOOP","WHILE","REPEAT","SELECT","IF"
*         PUSH P
```

```
*         Avancer sur la fin de la ligne
*         P= "ch" & (token final correspondant)
*         GOTO cher (Arghhhhhhh !)
*       CASE T1$, T2$
*         IF pile_vide THEN GOTO sortie
*         Avancer sur la fin de la ligne
*         POP P
*         GOTO cher (Arghhhhhhh !)
*       CASE "END WHILE/LOOP/SELECT/IF","UNTIL",
*           "CASE","ELSE"
*         "Structure Mismatch"
*     END SELECT
*   END LOOP
* Sortie:
*
*
* Zones memoires utilisees :
*
* AVMEME : pour voir si on ne depasse pas les
* limites de la memoire du HP-71.
* OUTBS : bas de la pile
* AVMEMS : sommet de la pile
*
 SP     EQU    =AVMEMS
 BP     EQU    =OUTBS
 Pvar   EQU    =S-R0-3

 cherche
        C=P    15       C(S) := chX
        P=     0
        GOSBVL =OBCOLL  initialiser la pile
* OBCOLL n'a pas modifie D0
        P=C    15       P := chX
 cher   C=P    0        C(0) := quartet de rech.
        P=     0        Ouf !
        D1=(5) Pvar
        DAT1=C P        Pvar := quartet de rech.
 che010
*
* adresse de la fin d'analyse :
*
        D1=(5) =PRGMEN
        C=DAT1 A        C(A) := PRGMEN par defaut
        ?ST=1  =PgmRun
        GOYES  che020
*
* On cherche dans le buffer bSTMT
*
        LC(3)  =bSTMT
        GOSBVL =IOFND0
        CD1EX
        C=C+A  A
 che020 D=C    A
*
* On cherche le token tXWORD :
*
        LC(2)  =tXWORD
        GOSBVL =TKSCN7
*
* Cy = 0 : non trouve : "Structure Mismatch"
* Cy = 1 : trouve. Les ennuis commencent...
*
        GONC   STRuce   "Structure Mismatch"
*
* Sauvegarde de l'adresse de tXWORD
*
        GOSUB  stoadr   SAUVD0 := ^ tXWORD
*
* Dans SAUVD0, nous avons :
*
* Stlen tXWORD id tTOKEN
*       ^
*       SAUVD0
*
        D0=D0+ 2        D0 := ^ id
*
* tXWORD id tTOKEN
*        ^
*        D0
*
        A=DAT0 B        A(B) := id trouve
        LC(2)  =id      C(B) := NOTRE id
        ?A=C   B
        GOYES  che022
        GOTO   che900   EOL(D0-2) ; GOTO cher10
 che022
*
* C'est notre id
*
        D0=D0+ 2        D0 := ^ TOKEN
*
* tXWORD id tTOKEN
*           ^
*           D0
*
        A=DAT0 B        A(B) := token
*
* Est-ce un token de structure ?
* (WHILE, LOOP, REPEAT, SELECT, IF, END, UNTIL,
*  CASE et ELSE)
*
        LC(2)  =tEND2
        ?A=C   B
        GOYES  che025
*
* On pourrait aller directement au TBLJMC, mais ca
* nuit a la lisibilite et a la maintenabilite.
* Vive la programmation structuree !
*
        NIBHEX 3        LC
        CON(1) 2*8-1      (15)
        CON(2) =tWHILE
        CON(2) =tLOOP
        CON(2) =tREPEAT
        CON(2) =tSELECT
```

```
        CON(2) =tIF2
        CON(2) =tUNTIL
        CON(2) =tCASE
        CON(2) =tELSE2
        P=     2*8-1
        GOSBVL =MEMBER
        GOC    che900   Byte not in set
 che025
*
* Est-ce un debut de structure ?
*
        GOSBVL =FINDA
        CON(2) =tWHILE
        REL(3) che200
        CON(2) =tLOOP
        REL(3) che210
        CON(2) =tREPEAT
        REL(3) che220
        CON(2) =tIF2
        REL(3) che230
        CON(2) =tSELECT
        REL(3) che240
        CON(2) 00
*
* Non. Va-t-on trouver ce qu'on cherche ?
*
        D1=(5) Pvar
        C=DAT1 P

 che099 GOSBVL =TBLJMC
        REL(3) che100
        REL(3) che110
        REL(3) che120
        REL(3) che130
        REL(3) che140
        REL(3) che150
        REL(3) che160
        REL(3) che170
        REL(3) che180
        REL(3) che190
        REL(3) che1a0
        REL(3) che1b0
*
* "WHILE", "LOOP", "UNTIL", "IF", "SELECT"
*
 che240 P=     chES-C   END SELECT, sans CASE
        GOTO   che200
 che230 P=     chEI-E   END IF, sans ELSE
        GOTO   che200
 che220 P=P+1
 che210 P=P+1
 che200
*
* P = nouveau quartet de recherche
*
        C=P    15       C(S) := nouveau quartet
        P=     0
        GOSUB  chpush   empiler le courant
        D1=(5) Pvar
        DAT1=C S
*
* ATTENTION : le code continue !!!
*
 che900 GOSUB  recadr
        D0=D0- 2
        GOSUB  eol
        GOTO   che010

*
* chercher END WHILE
*
 che100 GOSUB  recadr
        D0=D0+ 6        D0 := ^ q. de reconnais.
*
* tXWORD id tEND qENDx tEOL ou encore
* tXWORD id tEND tEOL
*               ^
*               D0
*
        A=DAT0 B
        GOSUB  eolxck
        GONC   Struce   "Structure Mismatch"
        GOTO   che300   B.E.T. a fin
*
* chercher UNTIL
*
 che120 LC(2)  =tUNTIL
        GOSUB  che400
        GOC    che299   Trouve
 Struce GOTO   STRUCe   Non trouve
*
* chercher ELSE ou END IF
*
 che150 LC(2)  =tELSE2
        GOSUB  che400
        GOC    che299   Trouve
        P=     =qENDI   Non. Chercher END IF
        GOTO   che110   B.E.T.
*
* chercher CASE ou END SELECT
*
 che299 GOTO   che300

 che160 LC(2)  =tCASE
        GOSUB  che400
        GOC    che299   Trouve
        P=     =qENDS    Non. Chercher END SELECT
        GONC   che110   B.E.T.

* chercher END WHILE, mais pas CASE, ELSE,
* END SELECT, END IF
*
 che1b0 P=P+1
 che1a0 P=P+1
 che190 C=P    15
```

```
        P=      0
        D=C     S       D(S) := chEW, chEL ou chU
        A=DAT0  B
        LC(4)   (=tCASE)~(=tELSE2)
        P=      2*2-1
        GOSBVL  =MEMBER
        GONC    che175  Trouve : on recommence
        LC(2)   =tEND2
        ?A#C    B
        GOYES   che195
*
* tEND2 trouve : est-ce END SELECT ou END IF ?
*
        D0=D0+  2
        A=DAT0  P
        LC(1)   =qENDS
        ?A=C    P
        GOYES   che175  Trouve, on recommence
        LC(1)   =qENDI
        ?A=C    P
        GOYES   che175  Trouve, on recommence
 che195 C=D     S
        CSLC            C(0) := quartet de rech.
        GOTO    che099  ch. END WHILE/LOOP/UNTIL
*
* chercher END SELECT, mais pas CASE
*
 che180 LC(2)   =tCASE
        GOSUB   che400
        GOC     che175  On repart pour une rech.
        P=      =qENDS
        GONC    che110  B.E.T.

 che175 GOSUB   recadr
        D0=D0-  2
        GOSUB   eol
        GOTO    che010  C'est reparti comme en 14
*
* chercher END IF, mais pas ELSE
*
 che170 LC(2)   =tELSE2
        GOSUB   che400
        GOC     che175  On repart pour une rech.

*
* ATTENTION : le code continue !!!
*


*
* chercher END IF
*
 che130 P=P+1
*
* chercher END SELECT
*
 che140 P=P+1
*
* chercher END LOOP
*
 che110
*
* tXWORD id tEND qENDx tEOL ou encore
* tXWORD id tEND tEOL
*
        C=P     15      C(S) := qENDx
        P=      0
        GOSUB   recadr
        D0=D0+  6       D0 := ^ qENDx ou tEOL
        A=DAT0  B
        GOSUB   eolxck
        GOC     STruce  tEOL... ==> END WHILE
*
* A(B) = token lu
*
        CSLC            C(0) = qENDx
        ?A#C    P
        GOYES   STruce
*
* ATTENTION : le code continue !!!
*


*
* Si la pile est vide
*   alors
*     sortir
*   sinon
*     eol
*     chpop
*     goto che010
* fin si
*
 che300 GOSUB   recadr
        D0=D0-  2       D0 := ^ Stlen
        GOSUB   chpop
        RTNC            Sortie sur pile vide
        GOSUB   eol
        GOTO    che010

 STruce GOTO    STRUCe

*
* Est-ce le token C(B) ?
*
 che400 A=DAT0  B
        ?A=C    B
        RTNYES          Cy := 1
        RTN             Cy := 0

*************************************************
* chpush
*
* But: empiler les tokens courants (T1$, et T2$)
*   sur la pile de recherche.
* Entree:
*   - P = 0
*   - Pvar = recherche courante
```

```
*   - AVMEME = limite sup. de la memoire dispon.
*   - SP = sommet de la pile de recherche
* Sortie:
*   - si pas assez de place : memerr
*   - SP reactualise
* Abime: A(A), C(A), D1
* Appelle: D1=AVE
* Niveaux: 1
* Historique:
*   87/03/01: conception & codage
************************************************

 chpush D1=(5) SP
        A=DAT1 A        A(A) := SP
        A=A+1  A        A(A) := SP + 1
        DAT1=A A        Sp := SP + 1
        GOSBVL =D1=AVE  C(A) := (AVMEME)
        ?A>=C  A
        GOYES  memerr
        D1=(5) Pvar
        C=DAT1 P        C(0) := Pvar
        D1=A            D1 := SP + 1
        D1=D1- 1
        DAT1=C P        empiler Pvar
        RTN

 memerr GOVLNG =MEMERR

************************************************
* chpop
*
* But: depiler les tokens au sommet de la pile de
*    recherche, et les mettre dans les tokens
*    courants.
* Entree:
*   - P = 0
*   - BP = bas de la pile
*   - SP = sommet de la pile de recherche
* Sortie:
*   - Cy = 0 : pile non vide
*      - SP reactualise
*      - Pvar, C(0) = recherche courante
*   - Cy = 1 : la pile etait vide
* Abime: A(A), C(A), D1
* Niveaux: 0
* Historique:
*   87/03/01: conception & codage
************************************************

 chpop  D1=(5) BP
        C=DAT1 A        C(A) := bas de la pile
        D1=D1+ 5        D1=(5) SP
        A=DAT1 A        A(A) := sommet de la pile
        ?A=C   A        pile vide ?
        RTNYES          oui : Cy := 1
        A=A-1  A
        DAT1=A A        nouveau SP
        D1=A            D1 = SP
        C=DAT1 P        C(0) := valeur depilee
        D1=(5) Pvar
        DAT1=C P        Pvar := valeur depilee
        RTN             Cy = 0 (A=A-1)


************************************************
* nxtlex
************************************************


 nxtlex CON(2) =id      ID equivalent
        CON(2) =tLOOP
        CON(2) =tELSE2
        CON(5) 0
        NIBHEX F
        REL(4) 1+TxTbSt2
        CON(4) 0
        CON(5) 0
*
* M A I N   T A B L E
*
        CON(3) (TxEn06)-(TxTbSt2)
        REL(5) =LOOPe
        CON(1) #D

        CON(3) (TxEn07)-(TxTbSt2)
        REL(5) =SELECTe
        CON(1) #D

        CON(3) (TxEn08)-(TxTbSt2)
        REL(5) =CASEe
        CON(1) #D

        CON(3) (TxEn09)-(TxTbSt2)
        REL(5) =IF2e
        CON(1) #9

        CON(3) (TxEn10)-(TxTbSt2)
        REL(5) =ELSE2e
        CON(1) #D

*
* T E X T   T A B L E
*

 TxTbSt2
 TxEn08 CON(1) 7
        NIBASC 'CASE'
        CON(2) =tCASE

 TxEn10 CON(1) 7
        NIBASC 'ELSE'
        CON(2) =tELSE2

 TxEn09 CON(1) 3
        NIBASC 'IF'
        CON(2) =tIF2
```

```
TxEn06 CON(1) 7
       NIBASC 'LOOP'
       CON(2) =tLOOP

TxEn07 CON(1) 11
       NIBASC 'SELECT'
       CON(2) =tSELECT

       NIBHEX 1FF

       END
```

HANOI ET LA RECURSIVITE

Eh bien me revoilà encore une fois, et dans cette rubrique pour vous parler d'un petit délire informatique nommé *récursivité*. Qu'est-ce donc que cette drôle de bestiole au nom qui nous fait penser à Monsieur Propre (allusion à celui qui récure, bien entendu) ? Eh bien, il s'agit d'un outil informatique très pratique qui nous permet de simplifier bien des problèmes

Une petite explication s'impose, non ? Imaginons que vous vouliez effectuer une tâche quelconque, *A* assez compliquée ; vous ne savez pas quelle est la marche à suivre, mais vous savez qu'en agissant d'une manière *a* sur *A* vous vous ramenez à un cas *A* plus simple. De proche en proche, en agissant par tâches *a*, vous allez arriver à un cas *A* évident. Simple à réaliser. Donc, vous savez simplifier *A*, et lorsque *A* est simplifié au maximum, vous pouvez effectuer *A*. il vous suffit ensuite de réappliquer les taches *a* dans l'autre sens pour vous ramener au cas du début, mais vous vous êtes facilité la tache, et la recursivité, c'est ça.

Un exemple concret : la fonction Factorielle(n) (notée n!). On ne connaît pas *n*!, mais on sait :

```
n!=n×(n-1)
1!=1
```

A partir de là, les choses deviennent très simples. En effet :

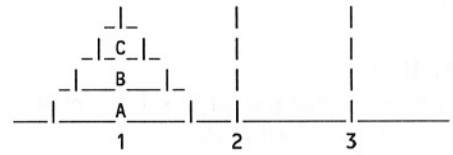
```
n!=n×(n-1)!
  =n×(n-1)×(n-2)!
  =n×(n-1)×(n-2)×(n-3)!
   .
   .
   .
   .
  =n×(n-1)×(n-2)×(n-3)×...×3×2×1
```

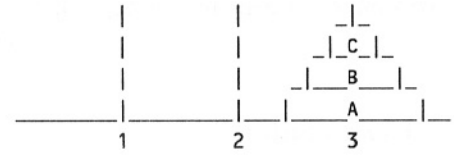
Je pense que vous avez tous compris quelles peuvent être les utilisations de ce genre de raisonnement. Pour s'amuser un peu (car il faut dans la vie, ne croyez vous pas ?), je vous propose donc de voir comment arriver à formaliser le problème des *Tours de Hanoï*. Tout le monde, je pense, connaît ce petit jeu vieux comme le monde : devant soi sont alignés 3 pieux ; dans le premier, une série de disques de tailles différentes, empilés par ordre décroissant. Le but du jeu est de tous les faire passer sur le troisième en suivant deux règles on ne peut plus simples :

- ne déplacer qu'un disque à chaque fois,
- ne jamais mettre un disque plus grand sur un plus petit.

On peut bien entendu se servir des 3 pieux pour tenter d'arriver à ses fins.

Voici un petit dessin, j'espère clair, des positions de départ et d'arrivée avec 3 disques :





Eh bien, comme dit plus haut, tentons un peu de formaliser le problème. Pensons-le avec 3 disques, il sera grand temps de voir plus après. Je ne sais pas déplacer ABC de 1 à 3. En fait, je sais, mais les règles m'interdisent de prendre ABC et de le mettre sur 3. Alors réfléchissons un peu. Je me simplifie la vie en disant que je vais déplacer BC de 1 sur 2, je verrais plus tard comment, puis A de 1 sur 3, j'ai le droit. Puis à nouveau BC de 2 sur 3. Et maintenant, attention Mesdames et Messieurs, je récurre : pour déplacer BC de 1 sur 2, je vais déplacer C de 1 sur 3, B de 1 sur 2, puis C de 3 sur 2. Je suis le même mode de pensée que pour ABC. Bon, j'ai BC sur 2 et A sur 1. Je le mets sur 3, et je redéplace BC de 2 sur 3 :

C : 2 sur 1
B : 2 sur 3
C : 1 sur 3

et le tour est joué. ABC est sur 3.

Pour 4, ou 5, ou *n* disques, on réalise le même raisonnement qui fait déplacer *n*-1 disques, puis le $n^{ième}$, puis rempiler *n*-1 disques.

Voyons maintenant comment programmer cela en Basic récursif bien sûr. J'ai besoin de 3 pointeurs :

1- le nombre de disque (N)
2- la case de départ (D)
3- la case d'arrivée (A)

donc, si j'appelle H la fonction *Tours de Hanoï*, j'ai : H(N,D,A).

- au départ, D=1 et A=3
- si N=1, j'affiche D et A
- sinon, je descends d'un niveau : N=N-1, D ne change pas et A devient la case restante, c'est à dire, ni D, ni A. On réalise cela par A=6-A-D.

Puis, une fois tout dépilé, j'affiche A et D, pour déplacer le plus grand disque du moment, et je rempile sur lui :

- si N=1 j'affiche D et A,
- sinon, N=N-1, A ne change pas, et D subit la même modification que A avant : D=6-D-A.

Et c'est fini.

Et voila le travail. Simple non ? J'espère, comme toujours, vous avoir intéressé avec mes petits délires. A bientôt.

Laurent Istria (3)

DEMONTREZ VOS THEOREMES

Ce programme a pour objet la logique des propositions et les règles dites de *Wang* pour l'algorithme de démonstration. En d'autres termes, il s'agit d'un logiciel capable de déduire P=>Q à partir de P<=>Q, par exemple. Bien entendu, DEMOS n'a pas la prétention d'égaler Prolog, beaucoup s'en faut ! Néammoins, il peut résoudre grand nombre de questions, comme vérifier des formules de cablage en logique électronique (compteur, bascule, etc).

Démontrer : Quelques précisions sur le sens

L'énoncé du théorème ne se pose pas tout à fait à la manière des livres de Maths.

Ici, il y a une hypothèse et une conclusion (des formules propositionnelles) et le but du jeu est de montrer que de la première, la seconde s'en déduit. Donc il faut que l'hypothèse et la conclusion soient ensemble vraies ou fausses en fonction de leurs variables composantes.

En résumé, le théorème « hypothèse donne conclusion » est vrai si et seulement si :

- hypothèse et conclusion sont vraies,
- hypothèse et conclusion sont fausses.

Signes cabalistiques des formules

Pour construire une formule, il faut tout d'abord des variables propositionnelles : P,Q,R,D0,α,Γ... ; elles sont composées de 2 caractères au mieux dans DEMOS. Ces variables prendront selon demande les valeurs *vrai* ou *faux*.

L'articulation de ce beau monde se fait autour de connecteurs :

/ (négation) : si P est *vrai*, alors /P est *faux* et vice versa. On a //P qui vaut P.

+ (ou) : si l'un des deux P, Q est *vrai* alors P+Q devient *vrai*.

. (et) : si les deux P, Q sont *vrais* alors P.Q devient *vrai*.

> (implique) : P>Q est une autre manière d'écrire /P+Q . L'implication est donc *fausse* seulememt dans le cas P *vrai*, Q *faux*.

= (equivaut à) : P=Q s'écrit aussi (P>Q).(Q>P) . L'équivalence est *vraie* seulement lorsque P *vrai*, Q *vrai*.

() : les parenthèses ne sont pas des connecteurs mais elles permettent de séparer les parties de formules.

Ce tour d'horizon met en évidence ceci : une formule peut se démontrer en agissant sur les valeurs *vrai/faux* de chacune des variables composantes.

Une *tautologie* est une formule *vraie* quelles que soient les valeurs des variables (par exemple : /P+P ou P>(Q>P))

La clause vide, notée °, est une formule sans variable qui prend la valeur *faux*.

Mode d'emploi

Deux manière de s'en servir, avec ou sans contrôle des valeurs prises par les variables propositionnelles. La différence tient en deux points :

- le programme demande explicitement les valeurs (avec contrôle), il ne se prononce pas (sans contrôle).
- le programme conserve sur fichier la valeur des variables et se sert de la sauvegarde dans les démonstrations futures (avec contrôle).

Entrée des formules

- donner la « formule hypothèse » à l'aide de variables (2 caractères au mieux) et des connecteurs du paragraphe précédent.
- Donner la conclusion dans le même style. attendre la laconique réponse YES, théorème démontré ou NO dans le cas inverse.

Interprétation des α et β

Le programme commence par transformer les formules brutes (avec des > et =) en formules entièrement faites de +,.,() et /. L'algorithme étant récursif, α indique un appel, β un retour.

En cas d'erreur signalée, modifier l'expression en conséquence.

Interprétation des Γ et δ

Par l'affichage des appels (Γ) et des retours (δ), l'utilisateur suit le processus de démonstration.

Contrôle des valeurs

Lever au clavier le drapeau 4 avant de lancer DEMOS. Procéder à l'entrée normale de l'hypothèse et conclusion. Selon nécessité, le programme va demander une fois pour toute la valeur *vrai* (T comme True) ou *faux* (F en Anglais !).

L'affichage de : "P, T or F ?", signifie "variable P est-elle *vrai* ou *faux*?".

En fin de démonstration, il est possible de garder le fichier VALUE renfermant la « vérité ». Les utilisations postérieures se serviront du contenu pour mener à bien les cogitations, sauf si le drapeau 4 est baissé.

Conclusions et remarques

Sans le contrôle des valeurs, le programme prend les risques minima. Il échoue donc parfois, ne sachant pas traiter l'expression à un niveau plus élevé. La faute est celle de l'algorithme de *Wang* insuffisamment puissant.

Un jeu d'essais ?

(H) p=q
(C) p>q
c'est *vrai*, l'inverse est *faux*.

(H) ((q>p).(p>r))
(C) (q>r)
c'est *vrai* mais avec contrôle des variables !

(H) p
(C) (p>p)
vrai, on s'en doutait.

(H) (s>/r).r
(C) /s
c'est encore un théorème.

(H) /a+/b
(C) /(a.b)
théorème classique.

Xavier Bille (203)

Programme "GRAPH22" pour HP-75 (tracé de courbes sur HP82905B, nécessite GRAPHLEX)

```
   - GRAPH22 graphiques HP82905B (GRAPHLEX)
     (c) J.Y. HERVE et JPC 22/11/1987 V1.0
0003 DEF FNF(X) = X*PI/180-2*SIN(X)
0004 DEF FNH$ = 'X*PI/180-2*SIN(X)'
0010 PRINTER IS ":P1" @ PWIDTH INF @ DELAY 0
0020 DIM G$[321],H$[321],A$[18]
0030 EDIT "GRAPH22" @ DEF KEY CHR$(0),"R.100"
0040 ON ERROR CONT
0050 PURGE "FN"
0060 ASSIGN # 9 TO "FN",TEXT
0070 INPUT "Fonction Y=",FNH$;H$ @ PRINT # 9,3 ; "DEF FNF(X)="&H$
0080 PRINT # 9,4; "DEF FNH$='"&H$&"'" @ ON ERROR GOTO 40
0090 TRANSFORM "FN" INTO BASIC @ OFF ERROR @ PUT CHR$(0) @ MERGE "FN"
0100 PURGE "FN" @ PURGE KEYS
0110 OPTION ANGLE DEGREES
0120 INPUT "Ymin,Ymax,Tic ?","-4,7,1";Y0,Y1,Y3 @ Y2=(Y1-Y0)/320
0130 INPUT "Xmin,Xmax,Pas,Tic ?","-180, 450,1,45";X0,X1,X2,X3
0140 PRINT TAB(20);"Fonction F(X)=";FNH$ @ PRINT
0150 FOR J=Y0 TO Y1 STEP Y3
0160 H$=STR$(J) @ K=((J-Y0)/Y2+4.5)\6-LEN(H$)\2+20
0170 PRINT TAB(K);H$;CHR$(13); @ NEXT  J
0180 PRINT @ PRINT CHR$(27)&"&l9D"
0190 Y3=Y3/Y2 @ X=X0 @ K=IP((FNF(X)-Y0)/Y2+1.5) @ K=MAX(MIN(K,321),1)
0200 G$=RFILL$("",0,320)&CHR$(255)
0210 FOR J=1 TO 321 STEP Y3 @ G$[J,J]=CHR$(255) @ NEXT J
0220 DISP "Tracé de X=";X @ H$=G$ @ A$="" @ FOR I=7 TO 0 STEP -1
0230 IF X>X1 THEN H$=CBIT$(H$,I) @ GOTO 280
0240 IF MOD(X,X3)=0 THEN A$=STR$(X)&" " @ H$=SBIT$(H$,I)
0250 ON ERROR Y=K
0260 Y=IP((FNF(X)-Y0)/Y2+1.5) @ OFF ERROR @ Y=MAX(MIN(Y,321),1)
0270 J=MIN(K,Y) @ L=MAX(K,Y) @ H$[J,L]=SBIT$(H$[J,L],I) @ K=Y
0280 X=X+X2 @ NEXT I
0290 PRINT TAB(20-LEN(A$));A$;CHR$(27)&"*b321G"&H$
0300 IF X1>X-X2 THEN 220
0310 PRINT CHR$(27)&"E" @ BYE
```

****

Programme "HANOI" (le problème classique des tours de Hanoï)

```
  10 INPUT 'Nombre de disque ? ';N
  20 CALL HANOI(N,1,3)

------------------------------------------------------------

  30 SUB HANOI(N,D,A)
  40     IF N=1 THEN DISP D;A @ END
   -     on affiche les cases de départ et d'arrivée, et on remonte d'un disque.
  50     CALL HANOI(N-1,D,6-A-D)
   -     un disque de moins et on change la déstination.
  60     DISP D;A
   -     le plus grand du moment en position pour pouvoir rempiler.
  70     CALL HANOI(N-1,6-D-A,A)
   -     un disque de moins et on change l'origine
  80 END SUB
```

```
*****************************************************************************************************

Programme "DEMOS" (pour démontrer vos théorèmes, nécessite module Forth / Assembler)


   - Démonstrateur de théorèmes en logique des propositions.
     Auteur : Xavier BILLE (203)
     Nécéssite des fonctions BASIC du module forth/assembleur

     programme principal.
  10 DISP "Theorems demos." @ DESTROY ALL
   - H$ hypothèse , C$ conclusion.
     h$ et c$ sont initialisées avec la clause vide ° .
  20 DIM C$[99],H$[99] @ C$='°' @ H$='°'
   - introduction des deux formules du théorème.
  30 CALL INP("Hyppo.",H$) @ CALL INP("Conclu.",C$)
   - Appel au moteur de la démonstration.
  40 CALL MOS(H$,C$,F) @ DISP
   - f=drapeau marquant la réussite de la démonstration(1) ou l'échec(0)
  50 IF F THEN DISP "Yes." ELSE DISP "No."
   - drapeau 4 indique le contrôle des valeurs de variables propositionnelles.
  60 IF NOT FLAG(4) THEN 100
   - les valeurs sont gardées dans le fichier text VALUE.
  70 DISP "Keep file value Y/N "
  80 K$=UPRC$(KEY$) @ IF NOT POS("YN",K$) THEN 80
  90 IF K$='N' THEN PURGE VALUE
 100 END
   - ici débute la cascade de routines
     routine d'entrée des données et de leur transformations

-------------------------------------------------------------

 110 SUB INP(M$,S$) @ DIM I$[99]
   -     m$ message de l'input, s$ chaine à affecter.
         i$ intermédiaire de calcul
 120     DISP M$; @ INPUT ": ",S$;I$ @ S$=I$ @ CALL SYN(I$)
   -     la routine syn() le nombre de parenthèses et lève le drapeau 0 en conséquence
 130     IF FLAG(0,0) THEN 120 ELSE S$=I$ @ CALL TRANS(I$,I$)
   -     trans() découpe l'expression afin de modifier les = et les >
 140     IF FLAG(1,0) THEN 120 ELSE S$=I$
 150     DISP " ok." @ END SUB
   - cette routine vérifie la quantité de parenthèses.
     elle lève le drapeau 0 si le compte n'y est pas

-------------------------------------------------------------

 160 SUB SYN(S$) @ DIM R$[99]
 170     FOR I=1 TO LEN(S$) @ V=NUM(S$[I])
   -         (=chr$(40) et )=chr$(41). Accéssoirement on supprime les espaces(32).
 180         IF V=40 THEN P=P+1
 190         IF V=41 THEN P=P-1
 200         IF V#32 THEN R$=R$&CHR$(V)
 210     NEXT I @ IF P THEN DISP "Missing parenthesis" @ SFLAG 0
 220     S$=R$ @ END SUB
   - routine qui extrait l'argument dissimulé dérrière une paire de parenthèses.
     si s$='asdf)+(xcv)' alors a$='asdf'. Explore dans le sens droite-gauche.

-------------------------------------------------------------
```

```
230 SUB EXTR(S$,A$)
240     FOR I=LEN(S$) TO 1 STEP -1 @ V=NUM(S$[I])
250         IF V=40 THEN P=P+1
260         IF V=41 THEN P=P-1
270         IF P>0 THEN A$=S$[I+1,LEN(S$)] @ GOTO 290
280     NEXT I @ A$=S$
290 END SUB
  - routine qui reconnait les opérandes et l'opérateur dans une sentence, s$
    a$ 1er opérande reconnu, o code ascii de l'opérateur. Explore le sens droite-gauche.
    s$ la chaîne restante une fois oté a$. Commence par le 2ème opérande.

-----------------------------------------------------------------

300 SUB FIARG(S$,A$,O)
310     FOR I=LEN(S$) TO 1 STEP -1
320         V=NUM(S$[I])
330         IF V=41 THEN DIM E$[99] @ E$=S$[1,I-1] @ CALL EXTR(E$,E$) @ I=I-LEN(E$)-2 @ GOTO 320
340         D=V=43 OR V=46 OR V=61 OR V=62
350         IF D THEN O=V @ A$=S$[I+1,LEN(S$)] @ S$=S$[1,I-1] @ GOTO 370
  -         lorsqu'il n'y a pas d'opérateur (o=0),la chaîne est renvoyée en sa totalité (a$) et s$=''
360     NEXT I @ A$=S$ @ S$='' @ O=0
370 END SUB
  - routine qui modifie les connecteurs = et >.
    le résultat se récupère dans a$ sauf érreur (drapeau 1 levé)

-----------------------------------------------------------------

380 SUB TRANS(S$,A$) @ IF FLAG(1) THEN 550
  -     r$=intermédiaire
390     DIM R$[99] @ DISP CHR$(4);
  -     on recherche les connecteurs
400     CALL FIARG(S$,R$,O)
410     IF O THEN 'LBL1'
  -     il n'y a pas de connecteurs à ce niveau.
        y-a-t-il des parenthèses autour de l'expression ?
420     IF R$[LEN(R$)]=')' THEN 'LBLO'
  -     non, il s'agit d'une simple variable
        il faut tester sa qualité syntaxique
430     IF NOT LEN(R$) OR LEN(R$)>2+(R$[1,1]='/') THEN SFLAG 1 ELSE A$=R$ @ GOTO 540
440     IF FLAG(1) THEN DISP ' Var. err: '&R$ @ GOTO 550
  -     420 suite, on extrait l'argument pour le traiter.

450     'LBLO': DIM T$[99],U$[99] @ U$=R$[1,LEN(R$)-1]
460     CALL EXTR(U$,T$) @ U$=T$ @ CALL TRANS(U$,U$)
  -     une fois traité, on remet les parenthèses originelles
470     A$=R$[1,LEN(R$)-LEN(T$)-1]&U$&')' @ GOTO 540
  -     410 suite, on lance le traitement des opérandes avant de reformer l'expression.

480     'LBL1': DIM P$[99],Q$[99] @ CALL TRANS(R$,Q$) @ CALL TRANS(S$,P$)
  -     l'opérateur n'est pas un = ou un >
490     IF O<60 THEN A$=P$&CHR$(O)&Q$ @ GOTO 540
  -     l'opérateur ne peut être qu'un =(61) ou un >(62) à ce niveau.
        on rajoute des parenthèses si l'expression est composée: P$,Q$
500     IF LEN(P$)>2+(P$[1,1]='/') AND P$[LEN(P$)]#')' THEN P$='('&P$&')'
510     IF LEN(Q$)>2+(Q$[1,1]='/') AND Q$[LEN(Q$)]#')' THEN Q$='('&Q$&')'
  -     on forme la transformation de P$>Q$==/P$+Q$
520     A$='/'&P$&'+'&Q$ @ A$=A$[3*(A$[1,2]="//")]
  -     si c'est un = on forme /Q$+P$, puis A$.(/Q$+P$)
```

```
530     IF O=61 THEN DIM B$[99] @ B$='/'&Q$&'+'&P$ @ A$='('&A$&").("&B$[3*(B$[1,2]="//")]&')'
540     DISP CHR$(5);
550 END SUB
  - le moteur de la démonstration. Application de l'Algorithme dit de WANG
    h$ l'hypothèse, c$ la conclusion,
    f=1 la conclusion se déduit de l'hypothèse, 0 sinon


------------------------------------------------------------


560 SUB MOS(H$,C$,F) @ DISP CHR$(5);
  -     s$,r$ sont des variables qui contiennent des morceaux de h$,c$
570     DIM S$[99],R$[99]
  -     on découpe l'hypothèse
580     CALL FIARG(H$,S$,O)
590     IF O THEN 'LBL4'
  -     on ne peut plus découper, il faut chercher à extraire le contenu d'
        Il faut chercher à extraire le contenu d'éventuelles parenthèses.
600     IF S$[LEN(S$)]#')' THEN 'LBL2'
  -     on cherche à enlever les parenthèses afin de traiter l'intérieur.
610     IF S$[1,1]#'/' THEN S$=S$[1,LEN(S$)-1] @ CALL EXTR(S$,H$) @ CALL MOS(H$,C$,F) @ GOTO 730
  -     si l'hyp. commence par une négation, il faut faire passer l'expression
        correspondante dans la conclusion.
620     S$=S$[2] @ H$='°' @ CALL CON(C$,S$,43,C$) @ CALL MOS(H$,C$,F) @ GOTO 730
  -     600 suite. L'hyp. n'est plus séquable.
        On découpe alors la conclusion.

630     'LBL2': CALL FIARG(C$,R$,P)
  -     Les mêmes principes (lignes 590-620) sont appliqués à la conclusion.
640     IF P THEN 'LBL3'
  -     si conc. et hyp. sont réduites à de simples variables,
        on essaye d'attribuer (val) une valeur au th. s$ -> r$ .
650     IF R$[LEN(R$)]#')' THEN CALL VAL(S$,R$,F) @ GOTO 730
660     IF R$[1,1]#'/' THEN R$=R$[1,LEN(R$)-1] @ CALL EXTR(R$,C$) @ CALL MOS(S$,C$,F) @ GOTO 730
670     R$=R$[2] @ C$='°' @ CALL CON(S$,R$,46,H$) @ CALL MOS(H$,C$,F) @ GOTO 730
  -     640 suite. On essaye de travailler séparémemt les 2 sous-expressions
        de la conclusion.

680     'LBL3': H$=S$ @ CALL MOS(S$,R$,G)
690     IF P=43 AND G=0 OR P=46 AND G=1 THEN CALL MOS(H$,C$,F) ELSE F=G
700     GOTO 730
  -     590 suite. On tente de traiter séparément les 2 sous-expressions
        l'hypothèse.

710     'LBL4': R$=C$ @ CALL MOS(S$,C$,H)
720     IF O=46 AND H=0 OR O=43 AND H=1 THEN CALL MOS(H$,R$,F) ELSE F=H
730     DISP CHR$(18); @ END SUB
  - routine qui concatène r$ et s$, via l'opérateur o, dans t$

------------------------------------------------------------


740 SUB CON(S$,R$,O,T$)
750     IF S$='°' THEN T$=R$ ELSE T$=S$&CHR$(O)&R$
760 END SUB
  - routine qui décide au niveau de l'atome
    Ici se décide le théorème élémentaire.

------------------------------------------------------------
```

```
770 SUB VAL(H$,C$,F)
  -     flag(2) et flag(3) indique des négations devant les variables.
780     IF H$[1,1]='/' THEN SFLAG 2
790     IF C$[1,1]='/' THEN SFLAG 3
800     F=H$=C$ @ IF F THEN 970
  -     le contrôle des valeurs se fait avec le drapeau 4
810     IF FLAG(4) THEN 'LBL5'
  -     formule qui décide en l'abscence de connaissance certaine sur les variables.
820     F=FLAG(2) AND C$='°' OR H$='°' AND FLAG(3)
830     GOTO 970
  -     Option drapeau 4, on ouvre value pour extraire la connaissance.

840     'LBL5': X=FILESZR("VALUE") @ IF X<0 THEN CREATE TEXT VALUE @ X=0
850     ASSIGN #1 TO VALUE
  -     Pas de recherche pour la clause vide de valeur de vérité FAUX.
860     IF H$='°' THEN 'LBL6'
  -     codage : [variable]&chr$(31)&[valeur, T ou F]
870     H=SEARCH(H$[1+FLAG(2)]&CHR$(31),1,0,X,1)
880     IF NOT H THEN CALL INS(H$[1+FLAG(2)],#1,I,X) @ GOTO 'LBL6'
890     READ #1,H;S$ @ P=POS(S$,CHR$(31)) @ I=VAL(S$[P+1])
  -     Même travail sur la conclusion, on récupère sa valeur.

900     'LBL6': IF C$='°' THEN 'LBL7'
910     C=SEARCH(C$[1+FLAG(3)]&CHR$(31),1,0,X,1)
920     IF NOT C THEN CALL INS(C$[1+FLAG(3)],#1,J,X) @ GOTO 'LBL7'
930     READ #1,C;S$ @ P=POS(S$,CHR$(31)) @ J=VAL(S$[P+1])
  -     Prise en compte des négations.

940     'LBL7': IF FLAG(3) THEN J=NOT J
950     IF FLAG(2) THEN I=NOT I
  -     F prend une valeur selon celle des variables.
960     F=NOT (I EXOR J) @ ASSIGN #1 TO *
970     CFLAG 2,3 @ END SUB
  - routine qui inscrit au fichier une variable nouvelle.
    T pour True, F pour False

------------------------------------------------------------

980 SUB INS(S$,#1,F,X)
990     DISP S$; @ INPUT ' ,T or F :';F$ @ F$=UPRC$(F$[1,1])
1000    IF NOT POS("TF",F$) THEN 990 ELSE F=F$='T'
  -     codage : [variable]&chr$(31)&'00'&[valeur de la variable]
1010    S$=S$&CHR$(31)&'00'&STR$(F)
  -     x est le nombre d'élément du fichier Value.
1020    IF X THEN INSERT #1,0;S$ ELSE PRINT #1;S$
1030    X=X+1 @ END SUB
```

LE COIN DES LHEX

Comme de coutume, cette rubrique contient la liste des codes hexadécimaux des fichiers Lex parus ce mois-ci.

Rappelons ce qu'est un fichier Lex : c'est un programme pour le HP-71, en assembleur, qui apporte de nouvelles fonctions. Celles-ci sont utilisables directement, ou dans des programmes Basic.

Pour bénéficier de ces nouvelles fonctions, vous n'avez pas besoin de programmer vous-même en assembleur, ni de posséder un module Forth/Assembleur.

Il suffit de recopier le petit programme basic "MAKELEX" ci-dessous, de le lancer et de recopier les codes du fichier Lex désiré. Quand vous avez fini, les nouvelles fonctions sont accessibles, après avoir éteint et rallumé votre HP-71.

Si l'erreur "Erreur de somme" apparaît, vérifiez la ligne que vous avez introduite.

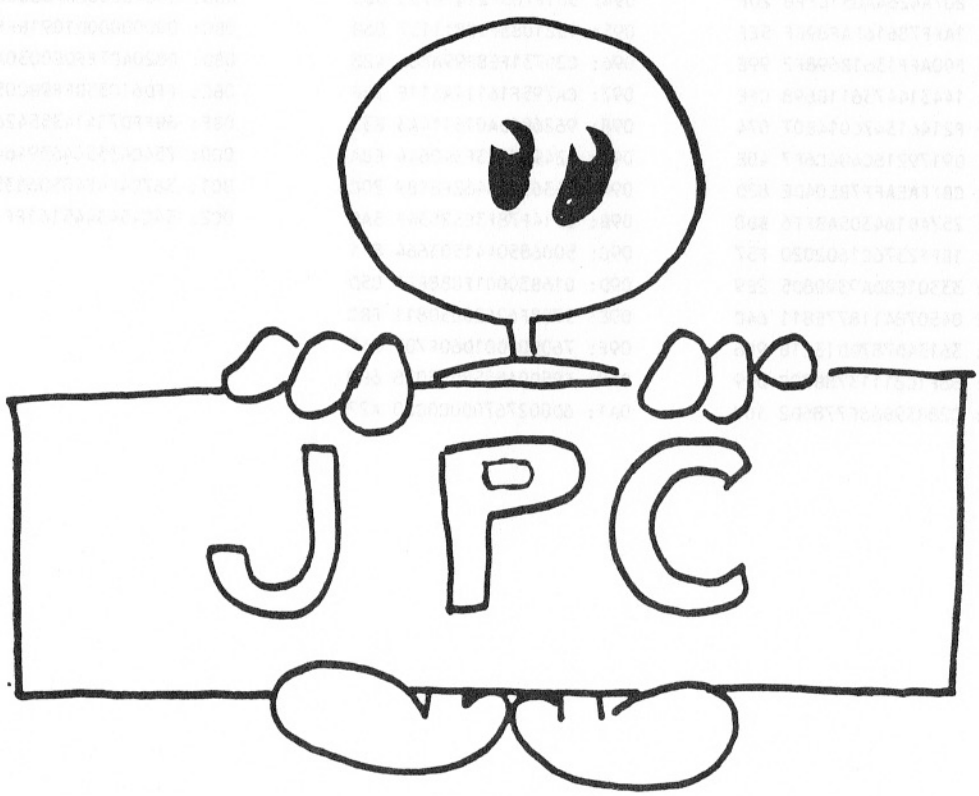
Vous trouverz donc le Lex CHARLEX nécessaire à la rédaction de votre article (voir "Ah ! Vous écrivez !"), ainsi que le Lex de programmation structurée de ce mois-ci. Notez que nous franchissons la barre des 100 tokens avec ELSE...

```
CHARLEX

STRUC2    END       XWORD   225066        WHILE     XWORD   225067
          REPEAT    XWORD   255068        UNTIL     XWORD   225069
          LEAVE     XWORD   255070        LOOP      XWORD   225096
          SELECT    XWORD   255097        CASE      XWORD   225098
          IF        XWORD   255099        ELSE      XWORD   225100
```

```
 10 CALL MLEX @ SUB MLEX @ SFLAG -1 @ PURGE AH @ INPUT "Nb. d'octets: ";N @ LC OFF
 20 CREATE DATA AH,1,N-4 @ A=HTD(ADDR$("AH")) @ B=A @ GOSUB 130
 30 Q=1 @ X=0 @ INPUT "000: ",P$;A$ @ C$=A$ @ S=0 @ GOSUB 90
 40 Q=2 @ X=1 @ GOSUB 80 @ A$=A$&C$ @ A=A+37 @ N=N*2+37 @ Q=3 @ SFLAG 5 @ FOR X=2 TO N DIV 16-1
 50 GOSUB 80 @ C$=C$[5*FLAG(5)+1] @ POKE DTH$(A),C$ @ A=A+16-5*FLAG(5,0) @ NEXT X @ Q=4
 60 DISP DTH$(X)[3]; @ INPUT ": ",P$[1,MOD(N,16)];C$ @ GOSUB 90
 70 POKE DTH$(A),C$ @ POKE DTH$(B),A$ @ CFLAG -1 @ END
 80 DISP DTH$(X)[3]; @ INPUT ": ",P$;C$
 90 DISP DTH$(X)[3]; @ INPUT " sm ","---";D$
100 M=S @ FOR Z=1 TO LEN(C$) @ M=NUM(C$[Z])+M+1 @ NEXT Z
110 IF D$=DTH$(MOD(M,4096))[3] THEN GOSUB 130 @ S=M @ RETURN
120 DISP "Erreur de somme" @ BEEP @ P$=C$ @ POP @ ON Q GOTO 30,40,50,60
130 P$="----------------" @ RETURN
```

```
CHARLEX  ID#E1  624 octets

     0123456789ABCDEF sm

000: 34841425C4548502 35E
001: 802E009212422088 6B3
002: 5E4001E000000000 9F7
003: FE00000000800001F D51
004: F31BF961400032BF 0E4
005: 38F14A11DB10AD23 47E
006: 07D532BFB8FD7911 831
007: 11AD754D7A101743 BB4
008: 11014D1CB15D0000 F1F
009: 71450375FF864834 29C
00A: 5655581008355654 5F3
00B: 5810002455565870 940
00C: 0026555658700836 C92
00D: 5556581008364545 FE8
00E: 4A30000A49724000 33B
00F: 0808094A2C180814 6A4
010: A464242008355455 9FE
011: 581000054C714000 D44
012: 0C3142404C700832 0A0
013: 41414A70002078A0 3F8
014: 2F30000000000000 723
015: 0000000000000000 A33
016: 0000000000000000 D43
017: 0000000000000000 053
018: 0000000000000000 363
019: 0000000000000000 673
01A: 0000000000000000 983
01B: 0000000000000000 C93
01C: 0000000000000000 FA3
01D: 0000000000000000 2B3
01E: 0000000000000000 5C3
01F: 0000000000000000 8D3
020: 0000000000000000 BE3
021: 000000000000080C F0E
022: 1A28080008080A2C 278
023: 180008040E340800 5C1
024: 08001E3018000000 8FB
025: 0000000000000000 C0B
026: 0000000000000000 F1B
027: 0000000000000000 22B
028: 0201000000010200 541
029: 0000000201020000 856
02A: 0001000100000002 B6A
02B: 0102010000000000 E7E
02C: 0000000000000000 18E
02D: 045E755142400101 4DA
02E: 0101010000000000 7ED
02F: 0000000000000000 AFD
030: 0000070507000000 E20
031: 0000000008344C4 15E
032: 44400D7901112D70 4BE
033: 050D750509700000 808
034: 0D70000000384540 B4B
035: 4020014E322E3140 E9F
036: 084E794142400000 1EF
037: 00000000002E4559 52D
038: 3200000000000000 842
039: 0000000000000026 B5A
03A: 5556587008365556 EB9
03B: 5810083645464830 20A
03C: 0832414248700024 54B
03D: 5655587008345655 8A8
03E: 5810083446454830 BF7
03F: 0C3042414C700024 F4C
040: 5556587008355654 2A9
041: 5810083546444830 5F8
042: 0C3142404C700025 94E
043: 5455587008355455 CA8
044: 5810083544454830 FF6
045: 0C3140414C700875 358
046: 14141870000A4972 6A9
047: 40000E3159454E30 A09
048: 0C7A0F7949400024 D81
049: 5554587000084A71 0DD
04A: 40000C523A262D10 43E
04B: 0424587458400875 795
04C: 1415187000094A70 AE5
04D: 4000083544454830 E29
04E: 0C3140414C300C74 191
04F: 5655545000054C71 4E8
050: 40000            5E1

STRUC2   ID#E1 1541 octets

     0123456789ABCDEF sm

000: 3545255534230202 342
001: 802E000312422088 68F
002: F0C001E246448B00 A0C
003: FB30087000000000 D56
004: 072600D230F6600D 0C4
005: 61019600D520A960 426
006: 0D90069500D554E4 7A2
007: 44249C4541465546 B08
008: 4B25540554144544 E62
009: 955E44594C454975 1E9
00A: 8494C454341FF303 56C
00B: 0014040202020202 88F
00C: CC230BF354727573 C1D
00D: 647572756 02D4963 F8A
00E: 7D6164713686CFF0 324
00F: 38DE6A208DA2C207 6CC
010: 5FFFE1E34420FE1E A9C
011: 06820FE1E16710FE E37
012: 1E36410006351185 188
013: 0121650022185800 4D4
014: 0208D82D20185011 832
015: 371FAA8F2AC21554 BDA
016: 135719FFE1E46DDF FA3
017: 007E408FB394031A 328
018: 8966A18F8EC20AA6 6DF
019: BB6BB679AF6B1072 A9C
01A: 207430313F966D08 E0A
01B: F8EC2070208FEA23 1B4
01C: 044B8D271308F9DF 558
01D: 30870935CE8F9DF3 907
01E: 0870A230187340E6 C6E
01F: 8161371FAA8F2153 FFA
020: 4948111359420080 359
021: 53E201554135018F 6BF
022: D96307B9F7BCE4F9 A91
023: 000068401818FB39 DF9
024: 40310F9628131049 152
025: 62F03112962606F1 4BB
026: 0626F1371F088F21 83A
027: 57413594E40018D5 BAB
028: 30308D0745014B7B F1D
029: D4474D61708F6242 2A2
02A: 0510220300339464 5DC
02B: 23653037C4F4F405 959
02C: 2765203B3554C454 CC5
02D: 34452B6110397584 025
02E: 94C4542974901 4B8 3A0
02F: D303507D80390245 709
030: 8454E4296FDF7D54 ABC
031: 5213754C43554276 E1E
032: 8CF153331A896631 1A1
033: 8F0E160754017252 50C
034: 27240313F966C137 87B
035: 0245F40227722017 BCE
036: 114B7F108FCE2503 F64
037: 1C2217700171 66AF 2D7
038: 8D324508D229506F 65B
039: EFF83EFF18770351 A18
03A: 678F871F030998A1 DA8
03B: 331E0962828F83DB 13E
03C: 07C151B178F2137C 4C7
03D: 2144164140AC25C0 830
03E: 7DF4AC2B461B098F BF1
03F: 2154477C47015267 F52
040: D1578C4165312414 2BD
041: A962F016114A7D43 645
042: 5606D9031A8966A3 9C9
043: 1F088F2161156415 D36
044: 541607AA41361B08 0A8
045: 8F2156413480DF77 430
046: 605444B5778414A3 7A1
047: 13F962E0227B4058 B1F
048: 24F3267F30551161 E89
049: 7F54237F205C0432 204
04A: 1617C4414A311F96 57E
04B: 6C016114A6D6F673 90B
04C: F7BF365B280CF201 CB6
04D: 361B698F21441B09 036
04E: 8F21524A4C413109 3AC
04F: 8F45DB0044B11298 738
050: 0DF1298F534D0042 AC2
051: 065B08D51DB0AC78 E6A
052: F83DB0137C21351B 1FD
053: 678F2146841840AC 582
```

```
054: 08AA218A8868A601 90E
055: 8505418A8C25C38B CA0
056: 670851D618414213 FFD
057: 08F7C1B120AC0482 38D
058: 860013028160E4E6 6F4
059: B20871B13018160E A63
05A: 4E681014A14F9E29 DF5
05B: E3048160E4EACB0E 1A9
05C: 4694C201B698F214 52C
05D: 61340163DFF47BFF 8DD
05E: 1877623287336D8 C4B
05F: 1EFCFF9FAFF70D28 047
060: FE3F804033098169 3CE
061: 4752DB13416314A8 746
062: F3E3203471006010 A9F
063: 449000066210C0C8 DFD
064: 0C020A87309A0B80 179
065: D07A6276C26021D2 4FE
066: CFFC9AFF14A7EF04 8F2
067: D0300902725D1313 C4E
068: 479E17F915E07032 FD5
069: 7CB16FD08D84A803 386
06A: 10677C175A168C05 703
06B: DBFFD3AFF7212776 AD9
06C: 15E078F174816BCF E7F
06D: 207A4264A051CFF0 20F
06E: 1AFF78616FAF89BF 5EF
06F: F00AFF1361B698F2 99E
070: 144314473611B698 CFE
071: F21461347C014E07 074
072: D9179216C406C6F7 40E
073: CBFFAEAFF7BE04DE 82D
074: 257AD164305ABFF6 BD8
075: 1BFF2376C1602020 F57
076: 33301E8DA93908D5 2E9
077: 045078411877E811 64C
078: 36134D787DD13210 9C6
079: 88F1C8111378B70C D59
07A: C28B39B86F7786D2 10C
07B: 71F0B7F22030214B 489
07C: 0E06A0C4951FD55F 835
07D: 214713517E143F43 B9E
07E: 4412E08A68313610 F03
07F: A1341FC65F2143E6 29C
080: E68BAC18F95EF011 65F
081: A1348FB7EF011213 9F2
082: 08D7E4708F871F08 D91
083: FD8DB004AC05D097 14A
084: C00110AC097C0001 4B7
085: 3098161851328F31 818
086: F808161341650110 B6B
087: 88FE3F8043430981 EF7
088: 694783DB134AF015 287
089: A51657630118AE5A 60F
08A: F2AE9BF2BF2BF2BF A0C
08B: 233FE1E9720061AE DA4
08C: 1B198F2146134011 106
08D: 361B198F21441340 46D
08E: 11371F1C6F214513 7E0
08F: 5011F1C6F21471E9 B6A
090: 95F1458D681F0D01 EFE
091: 4A136C213401614E 26B
092: 80CF208F5341080D 5F5
093: F80C0201F088F215 97D
094: 501F765F214787D3 D05
095: 1321088F1C811137 06B
096: C2D731FE8F99A805 428
097: CA795F16114A311E 7BF
098: 962606DA016114A3 B32
099: 124962023F340644 E8A
09A: 16365426462F8F89 20C
09B: 0B14F78F3E32034F 5A8
09C: 5006850441503664 8F1
09D: 016B30001F088F21 C5D
09E: 5708F62420850811 FBC
09F: 760E01D01D60F707 342
0A0: F05D0A8058008028 6B9
0A1: 6D002767000C0C80 A27
0A2: CF2070211F088F21 DAC
0A3: 554798E1817FCE63 14F
0A4: 0F7A7E16514A7D1D 4FB
0A5: 51160D031547FD04 86E
0A6: 716BFC314670D048 BFC
0A7: 02261906CA031267 F5D
0A8: BB043F215C70C0C8 2FE
0A9: 0CF20AC714A33462 68C
0AA: 6238F890B15C3312 A0A
0AB: 4966911611520301 D51
0AC: 902423029O2C1ACB 0CC
0AD: 8126F0F31267F504 44B
0AE: 702150275DD1817B 7C3
0AF: 1E6F4E3146704048 B43
0B0: E0C0C80CF2071BD1 EF0
0B1: 6514A745C4028129 25E
0B2: 0681789D18172504 5C5
0B3: 0077DD6B0E632C14 95B
0B4: A96200011F495F21 CD0
0B5: 43E41418F156818B 053
0B6: E911F088F2157013 3D2
0B7: 11C01550018DD449 744
0B8: 01FF85F214717414 AC3
0B9: 38A200CC14113115 E28
0BA: 701F088F21550011 18B
0BB: E064600000FB3000 4EB
0BC: 00000000D1091BFF 858
0BD: D8204C7FFD00030A BF8
0BE: FFD61035BFF9B005 FB5
0BF: 6BFFD73414355426 349
0C0: 754C435544639464 6B5
0C1: 367C4F4F40506B35 A45
0C2: 54C4543445161FF  D91
```


JPC

Le Journal JPC est le bulletin de liaison entre les membres de l'Association "PPC Paris", régie par la loi de 1901. Le Club est éditeur de JPC, et son siège social est au 33, boulevard saint Martin, 75003 Paris.

La maquette de ce numéro a été préparée et réalisée par Pierre David et Janick Taillandier, grâce à un système comprenant un HP71B, deux lecteurs de disquettes HP9114A, un HP9807A, un HP9153 un HP9134 et une imprimante LaserJet.

Directeur de la publication : Pierre David
Numéro ISSN : 0762 - 381X

Imprimé par Copy-Express, 42 86 91 94.

ENGLISH SUMMARY
JPC 52 - MARCH 1988

We are happy to announce you a good news not to be found in JPC : Paul Courbis, Sebastien Lalande and Guillaume Le Stum have been able to download the contents of the HP-28C operating system Rom to an HP-71. We are working on a disassembling of the Rom, using JPC Rom functions, of course !

Since last month, we have received additional informations from Bill Wickes about the HP-28S : the hardware is a new one, and the machine seems to be faster (some algorithms have been completely modified, and the clock speed has been increased).

In the HP-28C columns, Paul Courbis indicates that the program →LEX described in JPC 51 can be changed to a simple #3EEA0 SYSEVAL !

Next, Guillaume Le Stum provides some handy routines, mainly on integer programming.

We are still very involved in HP-75 assembly language programming as can be seen with a nice Lex providing graphic functions on a HP82905B by Jean-Yves Hervé.

The greatest part of the Journal is dedicated to a major achievement by Pierre David and Janick Taillandier. It extends the functionalities of a Lex published a few years ago, and now provides a complete set of structured programming statements, including an incredibly powerful SELECT ... CASE. These statements are fully compatible with HP9000 series Basic. In a future JPC, these guys will present two statements allowing structured listing of Basic programs using the new keywords.

Laurent Istria explains recursive programming with the Hanoï towers problem on page 33. Finally, Xavier Bille has realized a program to demonstrate formal logical propositions.

Until next month,

Happy Programming and JPC reading !