JPC

JUIN 1988

NUMERO 55

Le numéro 40 F

A PROPOS DU CLUB

HP28

HP75

HP71

EDITORIAL

Chers amis,

Vous avez certainement remarqué les nouvelles créatures qui hantent les colonnes de votre journal favori. Non, ne vous affolez pas, elles n'ont pas mangé les nibs farceurs ! Ceux-ci sont l'oeuvre de Jean-Jacques Dhénin, alors que les nouvelles créatures sont le fruit de l'imagination de Paul Courbis. Et toute cette ménagerie vit très bien ensemble, pour votre plus grand plaisir...

Pour votre plus grand plaisir également, le Club met à votre disposition tous les programmes pour HP-71 et HP-75 parus dans *JPC*. Philippe Assouline tient la programmathèque. Dans un premier temps, la marche à suivre est simple : vous envoyez au Club trois disquettes vierges et 75 F, et nous les renvoyons pleines avec un catalogue de ces programmes.

A la fin de cette période d'essai, nous ferons le point en fonction du volume, de vos remarques et suggestions, et nous verrons si nous arrêtons cette initiative, si nous la continuons telle quelle, ou si nous l'étendons aux autres supports (cassettes et cartes magnétiques) voire aux autres matériels.

Alors, faites vous entendre...

Pierre David (37)

COURRIER DU COEUR

Eric Gengoux
8 rue de Furstenberg
75006 Paris

Vend :

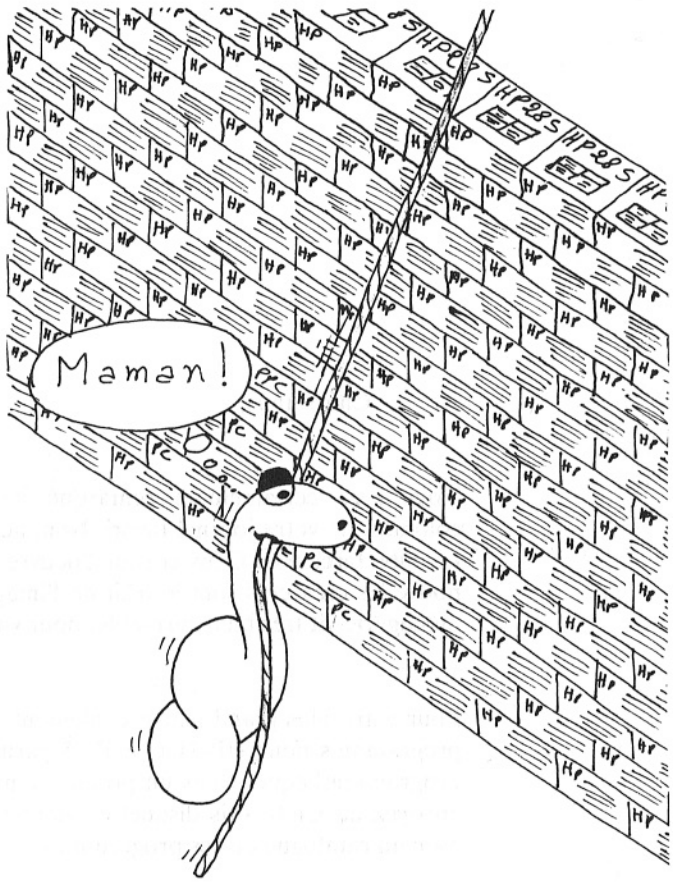
Lecteur de cassettes HP82161A : 1300 F, imprimante thermique HP82162A : 1000 F, interface RS232 HP82164A : 1500 F, modules Ram 4 Ko pour HP-71 : 400 F pièce (à débattre).

René Fagnon
81 rue de Belfort
25000 Besançon

Vend :
HP-71B avec différents modules, HP-IL ; HP-75C avec différents modules ; imprimante thermique avec papier ; lecteur de cassettes avec cassettes neuves ; lecteur de disquettes et disquettes ; imprimante ThinkJet avec housse et encre ; convertisseur HP-IL / HP-IB ; interface vidéo HP ; interface vidéo Pacscreen ; moniteur HP (grand format) : prix très intéressants, le tout en très bon état avec la documentation.

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

Vend:
Pour cause double emploi un HP-71B, une imprimante ThinkJet, un lecteur de disquettes HP9114 : prix à débattre.



HP28

ASSEMBLEUR POUR HP28S

Ce programme est l'équivalent des programmes ASS et →LEX conçus pour le HP28C et parus dans *JPC 51.*

Données en entrée : chaîne de caractères contenant les codes hexadécimaux représentant l'objet à créer.

En sortie : l'objet ou un message d'erreur Bad Argument Type, Too Few Arguments, ou Too Few Memory.

Entrée du programme :

- Taper le programme ASS,
- Entrer les codes sous forme d'une chaîne de caractères, en une seule ligne, sans espace. La stocker dans S,
- Rappeler S, exécuter ASS et stocker le résultat dans SC, dans le menu HOME,
- Rappeler S et faire #D0000 SC SIZE 2 * - SYSEVAL.

Le résultat (deux System Objects) sera stocké dans ASSEMBLEUR, dans le menu de votre choix... ASS, S et SC peuvent être alors détruits.

Le programme en RPL :

```
« → LM
  « HEX "" 1 LM SIZE
    FOR X
      "#" LM X DUP2 1 + DUP SUB 3 ROLLD
      DUP SUB + + STR→ B→R CHR + 2
    STEP
  »
»
```

Le programme ASSEMBLEUR :

```
        CON(5) #02C67
        CON(5) #0C36A  Mise à jour de STACKSIZE
        CON(5) #02C96  Assembly code
debut   CON(5) (fin)-(debut)
        GOSBVL #05081  Sauvegarde D0 D1 B D
        D0=(5) #C015F
        C=0    A
        DAT0=C A       Cmd number=00000
        D0=(5) #C00F9
        A=DAT0 A       A(A) = STACKSIZE
        LC(5)  10
        ?A>=C  A       Y-a-t'il assez d'objets ?
        GOYES  OK1
ERR1    LCHEX  00201   Too Few Argument
ERR     A=C A
        GOSBVL #050B8  Récupération D0 D1 B D
        GOVLNG #0396A  Erreur
OK1     D0=(5) #C0116
        A=0 W
        A=DAT1 A       Sauvegarde éventuelle de
        C=DAT0 XS      l'argument dans la pile
        C=C+C  XS      du LAST
        C=C+C  XS
        GONC   L1
        D0=(5) #C0096
        DAT0=A 15
L1      D1=A           D1 = ^Objet au niveau 1
        A=DAT1 A       A(A) = prologue
        LCHEX  02A4E
        ?A=C A         Est-ce une chaîne ?
        GOYES  OK2
        LCHEX  00202   Bad Argument Type
        GOTO   ERR
OK2     ST=0   15      Premier essai
        D1=D1+ 5
        A=DAT1 A       A(A) = longueur chaîne
        LC(5)  5
        ?A=C   A       Chaîne vide ?
        GOYES  ERR1
        A=A-C  A
        ASRB           A(A) = Nb codes dans la chaîne
        R0=A
        GOSBVL #050B8  Récupération D0 D1 B et D
L2      C=R0           C(A) = place à réserver
        GOSBVL #053AE
        GONC   OK3     Réservation réussie : OK3
        GOSBVL #050B8  Récupération D0 D1 B D
        ?ST=0  A       premier essai ?
        GOYES  RETRY   oui : on recommence
        GOLNG  #0393E  Too Few Memory
RETRY   GOSBVL #04A94  Garbage Collector
        ST=1   15      Second essai
        GOTO   L2
OK3     CD0EX          Sauvegarde de D0 qui contient
        R1=C           l'adresse de la place réservée
        GOSBVL #050B8
        C=R1
        D0=C
        A=R0
        B=A    A
        B=B-1  A       B=nb. quartets à coder-1
        A=DAT1 A
        D1=A
        D1=D1+ 10      D1 = ^ contenu de la chaîne
L3      A=DAT1 B       Lecture 1 caractère
        LCASC  '0'     Conversion ASCII en hexa
        A=A-C  B
        LCHEX  09
        ?A<=C  B
        GOYES  L4
        LCHEX  07
        A=A-C  B
L4      DAT0=A 1       Ecriture quartet
        D0=D0+ 1       Quartet suivant
        D1=D1+ 2
        B=B-1  A
```

```
        GONC   L3       Si pas fini on continue
        GOSBVL #050B8   Récupération D0 D1 B D
        A=R1
        DAT1=A A        Rés. à la place de la chaîne
        A=DAT0 A        Fin de routine
        D0=D0+ 5
        PC=(A)
fin     CON(5) #02F90   Fin de structure
```

Et voici les codes hexadécimaux du programme à rentrer :

```
76C20A63C069C20D31008F180501BF510CD21441B9F00C14234A
00008BE913410200DA8F8B0508DA69301B6110CAF01431562A26
A265D01B6900C158E13114334E4A208A2D0342020061BF84F174
14334500008A249EA81C1008F8B0501188FEA3505328F8B05086
F908DE39308F49A4085F65DF1361098F8B050119134110D8CD14
313117914B3103B6A31909EA903170B6A1580160171CD59D8F8B
0501111411421648081009F20
```

Pour bien comprendre le fonctionnement de ce programme, reportez vous à *JPC 51*.

Attention : Ne jamais assembler de codes fantaisistes ou trop longs (par exemple : E4A205000000 au lieu de E4A2050000) sous peine de problèmes avec STO... Dans ce cas, faire un arrêt système.

A bientôt pour PEEK et POKE sur HP-28S...

Paul Courbis (392)

SAUVEGARDE D'ECRAN POUR HP-28C

Et voici le début de ce que vous attendiez tous avec impatience : l'ère de l'adaptation des commandes du HP-28S sur le HP-28C. Malheureusement, je ne possède en plus de mon HP-28S qu'un HP-28C version 1BB : l'adaptation pour 1CC aurait été encore plus facile, mais je n'en ai pas. Et pourtant, je me doute que les possesseurs de 1CC doivent se sentir lésés. C'est pourquoi, à la fin de cet article, je vous proposerai une méthode pour trouver les équivalences. Mais passons au vif du sujet.

Le programme LCD→ s'exécute directement, il fixe l'écran dans une chaîne de caractères que vous pouvez stocker, transformer, et ressortir sous forme d'écran avec →LCD. Vous pourrez ainsi sauvegarder vos plus beaux graphismes en rajoutant LCD→ à la fin des programmes qui les produisent ou si c'est une courbe utiliser le programme :

```
DRAW→ :


« CLLCD DRAW LCD→ »
```

au lieu du DRAW simple.

Voici LCD→ pour HP-28C (1BB) :

```
76C20   prologue prgm
C53C1   ajustement de début
068B0   transformation de la
        première ligne en chaîne
47D60   short integer 00003
7D9F3   short integer FFFFF
92B30   SUB
        (ces 3 dernières adresses
        font un 3 FFFFF sub, c'est
        à dire suppriment les 2
        premiers caractères de la
        chaîne qui servent pour
        l'impression par infra rouge.)
078B0   seconde ligne
47D60   short integer 3
7D9F3   short integer FFFFF
92B30   sub
CD430   on somme les deux premières
        chaînes
088B0   troisieme ligne
47D60   short integer 3
7D9F3   short integer FFFFF
92B30   sub
CD430   somme
098B0   quatrième ligne
47D60   short integer 3
7D9F3   short integer FFFFF
92B30   sub
CD430   somme
09F20   fin prgm
```

Il vous suffit à présent de l'assembler avec ASS (*JPC 51*) et →LEX (*JPC 51*), je vous rappele que →LEX est déjà en Rom : il suffit de le remplacer (pour les 1BB toujours) par #3EEA0 SYSEVAL et de l'exécuter comme une commande normale. Je vous rappele aussi qu'il ne faut pas visiter un programme en langage machine sous peine de destruction de celui-ci ; vous pouvez faire RCL mais pas VISIT.

A présent, pour →LCD, c'est plus difficile car contrairement à LCD→ tout n'est pas déjà en Rom. Il y a donc une partie en langage machine proprement dit. Attention, ce programme teste la présence d'un élément dans la pile mais pas sa nature : ainsi il ne faut l'exécuter qu'avec une chaîne sinon vous provoquerez un arrêt système. Ceci est dû au fait qu'il n'y a pas que des adresses qui suivent le test. Il aurait fallu inclure le test de la nature dans la routine en

assembleur ; ceci paraîtra peut être dans un prochain *JPC*.

Voici le programme :

```
76C20       prologue prgm
91C70       fige l'écran après exécution.
            c'est ce qui permet de
            conserver le résultat à
            l'écran sinon on reviendrait
            tout de suite, à la fin du
            programme, au menu et à la
            pile
5B3C1       test d'un objet dans la pile
307C1       je pense que c'est la mise
            en condition du début
47D60       idem
69C20       prologue assembleur
6D000       longueur
143         A=DAT1  A
174         D1=D1+  5
E7          D=D+1   A
8F2EE40     GOSBVL  04EE2 (sauvegarde
            de D1 D0 B D en Ram)
844         ST=0    4
130         D0=A
164         D0=D0+  5
AF1         B=0     W
146         C=DAT0  A
164         D0=D0+  5
137         CD1EX
1C4         D1=D1-  5
137         CD1EX
D5          B=C     A
8A9         ?B=0    A
43          GOYES   L1
81D         BSRB
864         ?ST=0   4
A2          GOYES   L2
31A0        LCHEX   0A
AE7         D=C     B
1FA31F4     D1=(5)  #4F13A
143         A=DAT1  A
CC          A=A-1   A
CC      L4  A=A-1   A
421         GOC     L2
14E     L3  C=DAT0  B
161         D0=D0+2
CD          B=B-1   A
967         ?D#C    B
5F          GOYES   L3
5DE         GONC    L4
CD      L2  B=B-1   A
4C5     L1  GOC     L5
D3          D=0     A
303         LCHEX   3
816         CSRC
348700 4L8  LCHEX   40078 (début écran)
CB          C=C+D   A
135         D1=C
31C4        LCHEX   4C
            (nb d'octets de la première
            partie de l'écran)
840         ST=0    0
14A     L6  A=DAT0  B
149         DAT1=A  B
177         D1=D1+  8
            (incrémentation du
            pointeur écran)
            (on procède par octet)
161         D0=D0+  2
            (incrémentation du
            pointeur chaîne)
CD          B=B-1   A
403         GOC     L5
A6E         C=C-1   B
5BE         GONC    L6
870         ?ST=1   0
B1          GOYES   L7
3402100     LCHEX   00120
            (longeur de la partie vide
            située entre les deux parties
            de l'écran)
133         AD1EX
CA          A=A+C   A
131         D1=A
31B3        LCHEX   3B
            (nb d'octets de la seconde
            partie)
850         ST=1    0
5DC         GONC    L6
E7      L7  D=D+1   A
E7          D=D+1   A
A4E         C=C-1   S
50B         GONC    L8
8F91F40     GOSBVL  #04F19
            (récupération de D1 D0 B D)
142         A=DAT0  A
164         D0=D0+  5
808C        PC=(A)
```

Il ne reste plus qu'à l'assembler.

COMMENT TROUVER LES ADRESSES ?

Cherchons d'abord les adresses des commandes préexistantes dans la machine. Pour cela, il vous faut un PEEK provisoire. Il vous suffit de taper la chaîne suivante (que nous devons à Paul Courbis), de l'assembler, de la placer une fois assemblée en première place du menu (c'est primordial, sans quoi l'utilisation pourrait provoquer un MEMORY LOST) et d'exécuter le programme PEEK qui sera donc forcément placé après la chaîne dans le menu USER :

```
PEEK

« VARPEEK #4FF5C SYSEVAL SWAP DROP »
```

Dans VARPEEK vous placez une chaîne de la longueur sur laquelle vous voulez *peeker*. Par exemple, 100 caractères qui peuvent être quelconques.

Et voici la chaîne à assembler avec ASS mais pas →LEX ! car vous n'avez peut-être pas encore →LEX pour votre machine :

```
69C20       prologue assembleur
F9000       longeur
133         AD1EX
103         R3=A
133         AD1EX
132         AD0EX
102         R2=A
AFC         ABEX   W
101         R1=A
174         D1=D1+ 5
143         A=DAT1 A
133         AD1EX
179         D1=D1+ 10
143         A=DAT1 A
132         AD0EX
113         A=R3
133         AD1EX
143         A=DAT1 A
133         AD1EX
174         D1=D1+ 5
143         A=DAT1 A
DC          ABEX
174         D1=D1+ 5
3450000     LCHEX  00005
E1      L1  B=B-C  A
8A9         ?B=0   A
43          GOYES  L2
AE0         A=0    B
15A0        A=DAT0 1
3103        LCHEX  30
A6A         A=A+C  B
3193        LCHEX  39
9EA         ?A<=C  B
90          GOYES  L3
3170        LCHEX  07
A6A         A=A+C  B
149     L3  DAT1=A B
160         D0=D0+ 1
171         D1=D1+ 2
3420000     LCHEX  00002
6 ACF       GOTO   L1
113     L2  A=R3
133         AD1EX
112         A=R2
132         AD0EX
111         A=R1
AFC         ABEX   W
142         A=DAT0 A
164         D0=D0+ 5
808C        PC=(A)
```

Vous l'assemblez avec ASS et non →LEX, vous stockez la nouvelle chaîne dans PEEK.XX et faites à chaque fois qu'elle n'est plus en première position dans le menu { PEEK.XX } ORDER.

A présent, que vous avez PEEK vous pouvez trouver l'adresse de toutes les commandes. Il vous suffit de placer la commande dans un programme, par exemple :

```
TEST

« SWAP »
```

puis de placer ce programme en seconde position dans le menu juste après PEEK.XX et de faire #4FF30 PEEK. Les 5 premiers chiffres forment l'adresse écrite à l'envers, les 5 suivants sont ceux du code » puis la fin du programme 09F20.

Si vous faites SYSEVAL à l'adresse trouvée, vous obtenez l'exécution de la commande ; en peekant à cette adresse vous listez le programme "76C20...09F20" et récoltez ainsi différentes adresses (toujours écrites à l'envers) formant les étapes du programme.

Par exemple, cherchons l'adresse de sauvegarde de D0, D1, B, D : c'est à dire, pour les machines version 1BB : 04EE2. On trouve l'adresse de BEEP, on liste le programme on prend la dernière adresse avant 09F20, on peek à cette adresse, on liste, on trouve 76C20 puis 69C20 : une suite de codes puis 5 adresses. On prend l'avant dernière avant 09F20. On peek à nouveau, on trouve : cette dernière adresse + 5, écrite à l'envers, 8F une adresse, D6068F, une adresse et enfin 8F et l'adresse équivalente à 04EE2 que nous cherchions ! Cette succession d'étapes n'est pas sortie de ma tête, mais ceci a été obtenu en comparant les résultats de la 1BB et de la 1CC. En effet, en cherchant 2EE40 je l'ai trouvée dans le BEEP, il suffit alors de faire le même chemin dans l'autre machine.

Au sein d'un programme en assembleur cherchez toutes les valeurs à changer : adresse de début d'écran, longueur entre les deux parties d'écran, la sauvegarde et la récupération de D0, D1, B et D... Pour cela, trouvez une machine version 1BB, cherchez dans une commande ou elles apparaissent (dans cet exemple-ci, elles interviennent dans presque tous les progammes) et faites le parcours équivalent dans votre machine.

Il serait bon pour faciliter les recherches de posséder un programme qui cherche les adresses où se trouvent un groupe de chiffres donné. Mettez la chaîne contenant le groupe de chiffres dans LIKE et faites SEARCH :

```
«
  VARPEEK SIZE LIKE DUP SIZE
  → l a b
  «
    #0 'AD' STO { PEEK.XX } ORDER
    DO
      AD DUP 1 DISP l + b + PEEK a POS DUP
      IF
      THEN
        AD + 1 - DUP 1 DISP 1 +
        'AD' STO 1000 .1 BEEP HALT
      ELSE
        DROP AD l + 'AD' STO
      END
    UNTIL AD #40000 >
    END
  »
»
```

Une fois le programme lancé et une première adresse obtenue notez la, mettez vous dans le menu PROGRAM CTRL et faites SST pour laisser continuer le programme.

Voilà et bonne chance !

Sébastien Lalande (442)

UP POUR HP-28S

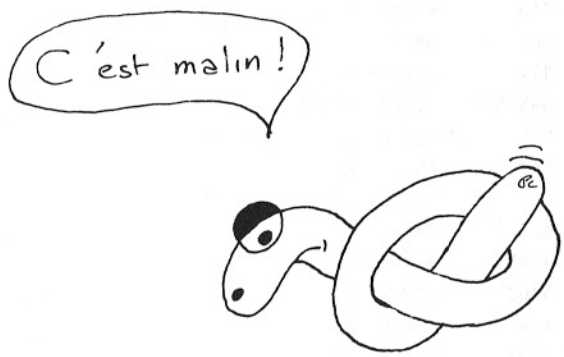
Vous possédez une 28S, fantastique ! Vous commencez à comprendre les menus (directories). Toutefois, une fois que vous avez penétré dans un sous sous ... sous menu, vous ne pouvez plus remonter au menu précédent, vous ne pouvez que faire HOME et redescendre jusqu'où vous vouliez aller. Voici un programme simple, mais utile, vous permettant de remonter au menu précédent. Il s'exécute directement. Pour pouvoir l'avoir toujours à portée de la main, je vous conseil de le stocker dans le menu principal (le tout premier) et de le rappeler dans le menu CUSTOM ou mieux de stocker le programme up (en minuscule) dans tous vos sous-directories :

up
« UP » (en majuscules)

et de stocker UP dans le menu principal. Ainsi, dans tout menu vous possédez directement up sans que cela ne vous prenne trop de mémoire. Voici UP :

```
UP

«
  PATH DUP SIZE
  IF 2 >
  THEN
    DUP SIZE 1 - GET EVAL
  ELSE
    DROP HOME
  END
»
```

Sébastien Lalande (442)



HP75

J.Y. Hervé

VARLEX *10*

VARLEX

Ce fichier Lex de 258 octets permet de lister ou d'imprimer toutes les variables employées dans un programme Basic du HP-75. Il est donc un outil précieux de documentation ou d'exploration. Il comporte un seul mot-clé, VARLIST.

Pour l'utiliser, il suffit d'insérer l'instruction VARLIST à n'importe quel endroit du programme dont on veut connaître les variables, et d'exécuter ensuite la ligne correspondante dudit programme en faisant RUN <numéro ligne>. Il est recommandé, si la ligne ajoutée n'est pas la dernière du programme traité, d'y inclure un END, comme indiqué dans les exemples.

```
1 VARLIST @ END  puis faire RUN
9999 VARLIST     puis RUN 9999
```

Les variables sont listées dans l'ordre chronologique de leur première apparition dans le programme, avec les informations suivantes :

- dimension des tableaux (numériques ou alpha),
- longueur maximale des chaînes de caractères,
- fonctions utilisateur (FNx et FNx$).

Lorsqu'une imprimante est branchée, la liste des variables est automatiquement imprimée. S'il n'y a pas d'imprimante, la liste est affichée sur le vidéo et / ou l'écran LCD ; dans ce dernier cas, on aura intérêt à spécifier DELAY 99 avant exécution, ce qui permettra de passer d'une variable à l'autre en pressant une touche quelconque (autre que [ATTN], car celle-ci annule l'effet de DELAY et arrête l'exécution juste après l'instruction VARLIST).

VARLIST fonctionne tout aussi bien en mode direct, et fournit alors les variables de calcul de CALCPROG.

Jean-Yves Hervé (450)

```
VARLEX   L290 Bytes  ROM ID: 119

Line -------- Data -------- Check

   1 7C8A10018D4CDE8214A65641 A5
   2 524C4558202077000A001200 10
   3 0C001A001D0049001E00FFFF AA
   4 5641524C4953D4FFFF61179E BE
   5 6E21A16DA8B406E54CE4CE9A 82
   6 0C6C06E30AE59E7810A10AE5 0B
   7 CE8A015810A1CE5AF398780A 9C
   8 E310A358A39EA19858B14382 3C
   9 1AA354E118C3561AE18B14C3 85
  10 C1FAEACEA0FE20E181F48153 63
  11 A80150A9E1817E14A51EA31C 1D
  12 A3CF0020F707A9464E10E553 1A
  13 88CE292E10E45EC820F701E4 C8
  14 5C90F415CF3000F725C810F6 E3
  15 04A825F002A82110E4F017F0 7C
  16 B7A9245BE56414A51CA35CB1 B2
  17 A382C63100A85DE453885E14 57
  18 E1CF4000F727E1E15CA82810 12
  19 E4B1A3821CC631005E14E1C9 EE
  20 FFFFF70C5CA82C10E4B1A382 02
  21 1CC6310058A82910E45E14E1 87
  22 538AF6F9560AE554E5CE98E3 9A
  23 5EA9140056A9E18150A801CE 48
  24 E5FDCE56FE540AE356E31498 31
  25 F091                    82
  26 8772
```

FORTH

ASSEMBLEUR

BASIC

LE COIN DES LHEX

CHAINES ALPHANUMERIQUES EN FORTH

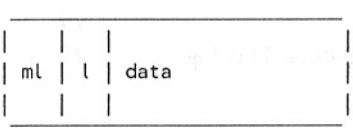
Le Forth est approprié au traitement numérique, c'est indéniable. Cependant, celui implanté sur le HP-71 possède un vocabulaire de création et de manipulation des chaînes alphanumériques. Partons à la découverte de ces facilités...

CREATION D'UNE CHAINE

Lorsqu'on lit le manuel d'utilisation ou des livres d'introduction au Forth, on ne peut manquer de tomber sur la représentation de l'alphanumérique. Une chaîne, c'est une adresse couplée à une longueur. Par exemple :

```
" HOP" .S
```

fait apparaître une adresse dans le *pad* et le nombre 3. Cette adresse est en fait début de *pad* plus 4. Pourquoi ? La représentation d'une chaîne en mémoire est la suivante :



ml est la longueur maximum de la chaîne ; à comparer avec `DIM C$[ML]` de Basic. *l* est la longueur actuelle de la chaîne ; à comparer avec `LEN(C$)` de Basic.

l et *ml* sont chacun codé sur 1 octet. Conséquence logique, une chaîne ne peut contenir plus de 255 caractères.

data est la chaîne elle-même, codée sous forme ASCII.

Maintenant, on peut essayer de créér le mot `CHAINE` capable de refléter cette structure. L'usage sera : `n CHAINE NOM`, où *n* est la taille maximum.

```
: CHAINE                                    ( n -- )
  CREATE
    DUP C,                                ( on stocke ML)
    0 C,                                  ( L=0 au départ)
    255 MOD                          (on s'assure que ML<256)
    ALLOT                                 ( allocation mémoire)
  DOES>
    2+ DUP C@                        ( on récupère la longueur)
    >R 2+ R>                                ( début de data)
;
```

Le lecteur voudra bien comparer les effets de `CHAINE` à ceux de `STRING` (voir manuel).

DEUX MOTS DE BASE

Le plus simple : celui qui fournit la **longueur maximum** (*ml*) d'une chaîne.

```
: MAX-LONG                                  ( chaîne -- ML)
  DROP                          ( la longueur ne sert à rien)
  2- 2- C@                           ( on se place et on lit)
;
```

Application : `" HOP" MAX-LONG` On trouve 80. Tiens, c'est curieux : une chaîne créée dans le *pad* n'a que 80 caractères au mieux. Cette particularité peut devenir source d'ennui (croyez-en mon expérience !) lors de la concaténation.

A propos de *pad* : c'est une zone de mémoire dans laquelle le système stocke les résultats d'opérations sur les chaînes. Essayez la suite suivante :

```
" machin" " hop" TYPE BL EMIT TYPE
```

Sans commentaires : ne pas laisser trainer une chaîne dans le *pad*, du moins si on tient à la récupérer.

L'assignation de chaînes. La forme proposée est : `ch1 ch2 ASSIG`, ce qui met *ch1* dans *ch2*.

```
: ASSIGN                                  ( A1 L1 A2 L2 -- )
  OVER SWAP MAX-LONG      ( vérifier qu'il y a assez )
  >R SWAP R>                                  ( de place. )
  OVER < ABORT" impossible"          ( pile : A1 A2 L1 )
  2DUP SWAP 2- C!          ( Nouvelle longueur de ch2 )
  CMOVE                                       ( Stockage.)
;
```

`ASSIGN` existe dans le manuel sous le nom `S!`. La conclusion à tirer de ces manipulations se résume ainsi : tout le vocabulaire des chaînes peut facilement se reproduire.

LES MOTS FINISSANT PAR $

`RIGTH$, LEFT$, END$, SUBST$`, ces mots permettent le traitement des sous-chaînes. Le résultat de leur action se trouve dans le *pad*, prudence donc.
Bizarrement des ordres tels que :

```
" bonjour chez vous" 8 END$ TYPE
```

ne posent aucun problème.

D'autres mots finissant par $ sont disponibles dans votre manuel préféré.

LA CONCATENATION

Il y a deux opérateurs : `S<&` et `S>&`. Ils sont puissants mais peu aisés à comprendre. Soit *ch1* la chaîne contenant `" bonjour "`, et *ch2* la chaîne contenant `" chez vous"`. Alors `ch1 ch2 S<&` renvoie l'adresse de *ch1*, laquelle contient `" bonjour chez vous"` ; *ch2* n'est pas modifiée.

`ch1 ch2 S>&` renvoie l'adresse de *ch2*, laquelle contient la concaténation ; *ch1* n'est pas altérée.

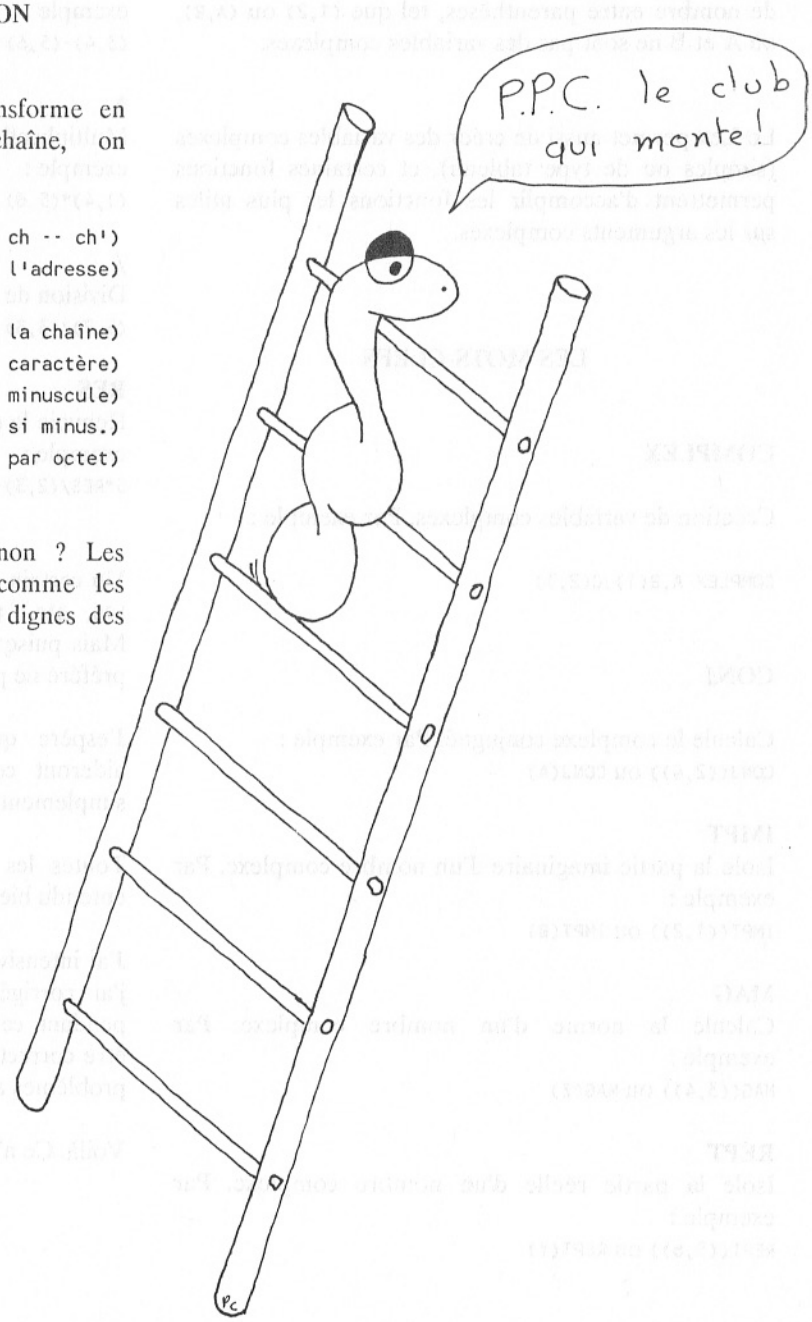
POUR FINIR, UNE APPLICATION

Soit à construire une fonction qui transforme en majuscules les minuscules d'une chaîne, on l'appelera... `UPRC$`. Forme : `ch UPRC$`

```
: UPRC$                                  ( ch -- ch')
  2DUP                          ( sauvegarde de l'adresse)
  1+ 2* OVER + SWAP
  DO                              ( boucle sur la chaîne)
    I C@                                  ( 1 caractère)
    DUP 96 > OVER 123 < AND             ( si minuscule)
    32 * + I C!                  ( enlève 32, si minus.)
  2 +LOOP ;                     ( boucle octet par octet)
```

Très chirurgicale comme application, non ? Les chaînes sont des ensembles d'octets comme les autres. A quand des programmes Forth dignes des meilleures applications en Basic ?

Xavier Bille (203)



UN LEX PAS SI COMPLEXE

Ceci est ma première contribution à notre Journal et j'espère que celle-ci sera publiée puisque c'est un des Lex les plus utiles que j'ai écrits.

Ce Lex pour HP-71B permet de faire des opérations sur des nombres complexes aussi simplement qu'avec les fonctions standard. Il intercepte le *poll* pCMPLX envoyé par le HP-71B à chaque fois qu'il trouve un nombre complexe ou une opération sur des nombres complexes.

Un nombre complexe pour le HP-71B est un couple de nombre entre parenthèses, tel que `(1,2)` ou `(A,B)`, où A et B ne sont pas des variables complexes.

Le Lex permet aussi de créer des variables complexes (simples ou de type tableau), et certaines fonctions permettent d'accomplir les fonctions les plus utiles sur les arguments complexes.

LES MOTS-CLEFS

COMPLEX

Création de variables complexes. Par exemple :

```
COMPLEX A,B(1),C(2,3)
```

CONJ

Calcule le complexe conjugué. Par exemple :
`CONJ((2,4))` ou `CONJ(A)`

IMPT

Isole la partie imaginaire d'un nombre complexe. Par exemple :
`IMPT((1,2))` ou `IMPT(B)`

MAG

Calcule la norme d'un nombre complexe. Par exemple :
`MAG((3,4))` ou `MAG(Z)`

REPT

Isole la partie réelle d'un nombre complexe. Par exemple :
`REPT((5,6))` ou `REPT(Y)`

LES OPERATEURS STANDARD

Le Lex redéfinit également les opérateurs standard pour accepter les nombres complexes. Ces opérateurs sont :

+

Addition de deux nombres complexes. Par exemple :

`(1,2)+(3,4)` ou `5+(6,7)`

-

Soustraction de deux nombres complexes. Par exemple :
`(3,4)-(5,6)` ou `(2,3)-5`

*

Multiplication de deux nombres complexes. Par exemple :
`(1,4)*(5,6)` ou `6*(4,4)`

/

Division de deux nombres complexes. Par exemple :
`(5,7)/(3,3)` ou `(2,1)/6`

RES

Renvoie le dernier résultat calculé par le HP-71B. Par exemple :
`5*RES/(2,3)-14*(1,1)*3`

Un certain nombre d'autres opérations telles que `SQR`, `LOG`, etc.) pourraient aisément être implémentées. Mais puisque je ne m'en servirais pas tellement, j'ai préféré ne pas les écrire.

J'espère que mes commentaires dans le source aideront ceux qui désireront étendre ce Lex ou simplement comprendre comment il fonctionne.

Toutes les suggestions ou optimisations sont bien entendu bienvenues.

J'ai intensivement utilisé ce Lex depuis deux mois, et j'ai corrigé toutes les bogues qui sont apparues pendant cette période, donc le programme devrait être correct à 90%. Faites moi savoir si vous avez des problèmes avec ce Lex.

Voilà. Ce n'était pas si compliqué...

Gigi Filippini (327)

```
        LEX   'COMPLEX'

AD2-12  EQU   #0C35F  ajouter deux 12-chiffres
AD2-15  EQU   #0C363  ajouter deux 15-chiffres
AVMEME  EQU   #2F599
D1=AVE  EQU   #18651  mettre D1 à AVMEME
D=AVMS  EQU   #1A460  mettre D(A) à AVMEME
DECDC   EQU   #05287  décompiler déclaration variable
DECP    EQU   #0328F  parse déclaration variable
DMNSN   EQU   #0AE39  crée et alloue un tableau
DPVCTR  EQU   #0AC50  crée une variable
DV2-15  EQU   #0C4AC  diviser deux 15-chiffres
EXAB1   EQU   #0D3E7  échanger AB avec Scratch1
EXPR    EQU   #0F23C  adresse de retour des fonctions
FINDA   EQU   #023E3  selon A(B), faire...
FNRTN4  EQU   #0F238  adresse de retour des fonctions
FUNCD1  EQU   #2F8C0
MEMERR  EQU   #0944D  ERR:Insufficient Memory
MP2-12  EQU   #0C432  multiplier deux 12-chiffres
NXTSTM  EQU   #08A48  revient dans la Basic Loop
POP1N   EQU   #0BD1C  dépile un nombre de la M.S.
POP2N   EQU   #0BC8C  dépile deux nombres de la M.S.
PREP    EQU   #0ADAF  prépare la création de variable
R<RSTK  EQU   #014DD  sauve des niveaux de RSTK
RCCD1   EQU   #0D3F5  rappelle CD depuis Scratch1
RCCD2   EQU   #0D41C  rappelle CD depuis Scratch2
RCSCR   EQU   #0E954  dépile 15-chiffres dans Scratch
RDATTY  EQU   #17CC6  ERR: Data Type
RSTK<R  EQU   #014A8  restaure des niveaux de RSTK
SPACE   EQU   #0AD9D  calcule la place pour 1 tableau
SQR-15  EQU   #0C534  racine carrée de 15-chiffres
STAB1   EQU   #0D3D9  stocke AB dans Scratch1
STKCHR  EQU   #18504  empile un caractère sur la M.S.
STSCR   EQU   #0E92C  empile 15-chiffres dans Scratch
XYEX    EQU   #0C697  échange AB avec CD
uRES12  EQU   #0C994  convertit de 15 en 12-chiffres

        ID    #5E
        MSG   0
        POLL  P       il y a un poll handler
        ENTRY DECL
        CHAR  #D
        ENTRY CONJE
        CHAR  #F
        ENTRY IMPTE
        CHAR  #F
        ENTRY MAGE
        CHAR  #F
        ENTRY REPTE
        CHAR  #F
        KEY   'COMPLEX'
        TOKEN 27
        KEY   'CONJ'
        TOKEN 28
        KEY   'IMPT'
        TOKEN 29
        KEY   'MAG'
        TOKEN 30
        KEY   'REPT'
        TOKEN 31
        ENDTXT

*****************************************************

P       LCHEX 38      si poll = pCMPLX
        ?B=C  B
        GOYES 0       alors le prendre
        RTNSXM        sinon retour

0       GOSBVL D1=AVE D1 := pointeur sur M.S.
        P=    5       sauve des niveaux de pile...
        GOSBVL R<RSTK ... pour plus tard

        D0=(5) FUNCD1 rappeler le numéro de token...
        A=DAT0 B      ... dans A(B)
        GOSBVL FINDA  selon le token :
        CON(2) #7A      nombre complexe :
        REL(3) NUM        aller en NUM
        CON(2) #67      rappel :
        REL(3) RCL        aller en RCL
        CON(2) #84      division :
        REL(3) DIV        aller en DIV
        CON(2) #83      multiplication :
        REL(3) MUL        aller en MUL
        CON(2) #87      addition :
        REL(3) ADD        aller en ADD
        CON(2) #82      soustraction
        REL(3) SUB        aller en SUB
        CON(2) #69      stocker réel dans complexe :
        REL(3) RSTO       aller en RSTO
        CON(2) #68      stocker complexe dans compl :
        REL(3) CSTO       aller en CSTO
        CON(2) #7F      RES ou () en mode CALC :
        REL(3) RES        aller en RES
        CON(2) 00     sinon : on ne reconnait pas
        P=    5       le token, donc on restaure
        GOSBVL RSTK<R les niveaux de pile et
        RTNSXM        on sort.

*****************************************************

RES     C=R1          C(W) := partie réelle
        GOSUB  PUSH   empiler sur la pile
        C=R0          C(W) := partie imaginaire
        GONC   EXIT2  B.E.T.

*****************************************************

ADD     SETDEC        pour SPLITA dans AD2-12
        A=R1          prendre...
        C=R3          ... les parties réelles
        GOSBVL AD2-12 et les additionner
        GOSUB  ures12 les convertir en 12-chiffres
        DAT1=C W      et les empiler sur la M.S.
        A=R0          prendre...
        C=R2          ... les parties imaginaires
```

```
       GOSBVL AD2-12  et les additionner
EXIT3  GOSUB  ures12  les convertir en 12-chiffres
EXIT2  GOSUB  PUSH    et les empiler
       P=     0       ajouter un "stack-header"
       LCHEX  0E      aux nombres sur la M.S.
       GOSBVL STKCHR
EXIT1  SETHEX         pour RSTK<R
       P=     5       restaurer les niveaux sauvés
       GOSBVL RSTK<R
       XM=0           le poll a été traité
       B=0    S       les lignes suivantes sont
       B=B-1  S       requises par RCL
       B=B-1  S       B(S) = type variable = E
       A=R1           B(A) = R1(A) = adresse variable
       B=A    A
       RTN            retour à l'appelant


*************************************************

SUB    SETDEC
       C=R1           C(W) := partie réelle
       C=-C-1 S       changement de signe
       DAT1=C W       empiler C(W) sur la M.S.
       C=R0           idem avec partie imaginaire
       C=-C-1 S
       GONC   EXIT2   B.E.T.

*************************************************

DOIT   SETDEC         utilisé par MUL et DIV
       A=R0           A(W) := Im1
       C=R3           C(W) := Re2
       GOSUB  mp2-12  AB := Im1 * Re2
       GOSUB  stscr   empiler AB sur la MathScrStk
       A=R1           A(W) := Re1
       C=R2           C(W) := Im2
       GOSUB  mp2-12  AB := Re1 * Im2
       GOSUB  stscr   empiler AB sur la MathScrStk
       A=R1           A(W) := Re1
       C=R3           C(W) := Re2
       GOSUB  mp2-12  AB := Re1 * Re2
       GOSUB  stscr   empiler AB sur la MathScrStk
       A=R0           A(W) := Im1
       C=R2           C(W) := Im2
       GOSUB  mp2-12  AB := Im1 * Im2
rcscr  GOVLNG RCSCR   et sortie par RCSCR

*************************************************

MUL    GOSUB  DOIT    calculs communs avec DIV
       A=-A-1 S       AB = - Im1 * Im2
       GOSUB  ad2-15  Re1 * Re2 - Im1 * Im2
       GOSUB  ures12  conversion en 12-chiffres
       DAT1=C W       et résultat sur la M.S.
       GOSUB  rcscr   AB := Re1 * Im2
       GOSBVL XYEX    CD := Re1 * Im2
       GOSUB  rcscr   AB := Im1 * Re2
       GOSUB  ad2-15  AB := Re1 * Im2 - Im1 * Re2
       GOTO   EXIT3   ... et sortie

*************************************************

DIV    GOSUB  DOIT    calculs communs avec MUL
       GOSUB  ad2-15  AB := Re1 * Re2 + Im1 * Im2
       GOSBVL STAB1   sauvegarde dans Scratch1
       A=R3           A(W) := Re2
       C=A    W       C(W) := Re2
       GOSUB  mp2-12  AB := Re2 * Re2
       C=B    W       stocker AB dans Scratch2
       AR2EX          et rappeler
       R3=C           Im2
       C=A    W       C(W) := Im2
       GOSUB  mp2-12  AB := Im2 * Im2
       GOSBVL RCCD2   CD := Re2 * Re2
       GOSUB  ad2-15  AB := Re2 * Re2 + Im2 * Im2
       GOSBVL EXAB1   échanger Re1 * Im2 + Re2 * Im1
       GOSUB  rccd1   CD := Re2 * Re2 + Im2 * Im2
       GOSBVL DV2-15  AB := (Re1 * Im2 + Re2 * Im1) /
       GOSUB  ures12  ...   (Re2 * Re2 + Im2 * Im2)
       DAT1=C W       et empiler sur la M.S.
       GOSUB  rcscr   CD := Re1 * Im2
       C=-C-1 S       CD := - Re1 * Im2
       GOSBVL XYEX    AB := - Re1 * Im2
       GOSUB  rcscr   CD := Im1 * Re2
       GOSUB  ad2-15  AB := Im1 * Re2 - Re1 * Im2
       GOSUB  rccd1   CD := (Re2 * Re2 + Im2 * Im2)
       GOSBVL DV2-15  AB := (Im1 * Re2 + Re1 * Im2) /
       GOTO   EXIT3   ...   (Re2 * Re2 + Im2 * Im2)

*************************************************

PUSH   A=C    W       A(W) := nombre à empiler
       GOSBVL D=AVMS  D(A) := available memory start
       D1=D1- 16      faire de la place pour le nb
       CD1EX          C(A) := D1
       D1=C           et restaurer D1
       ?D>=C  A       assez de place ?
       GOYES  memerr  non : erreur
       DAT1=A W       oui : empiler le nombre
       RTNCC                et revenir à l'appelant
memerr GOVLNG MEMERR  ERR: Insufficient Memory

*************************************************

       REL(5) Decdc   Routine de décompilation
       REL(5) Decp    Routine de parse
DECL   LCHEX  E       Ecrire le type de la variable
       D1=(5) #2F890  dans S-R1-3
       DAT1=C P       car recquis par PREP
RIP    GOSBVL PREP    préparation de la création
       GOSBVL DPVCTR  get dope vector
       R1=C           R1 := dope vector
       ?A#0   A       si variable = tableau
       GOYES  ARR     alors on continue
       A=A+1  A       sinon on fait comme si c'etait
```

```
        ST=1   0        un tableau dans la suite.
ARR     GOSBVL SPACE    Calculer place nécessaire
        GOSBVL DMNSN    et allouer la mémoire
        GOC    RIP      on répète si il y en a d'autres
        GOVLNG NXTSTM   sinon, on revient à Basic

Decdc   GOVLNG DECDC
Decp    GOVLNG DECP

*****************************************************

RCL     A=R1            A(A) := adresse de la variable
        D0=A            D0 := ^ partie réelle
        C=DAT0 W        C(W) := partie réelle
        D0=D0+ 16       D0 := ^ partie imaginaire
        GOSUB  PUSH     empiler partie réelle sur M.S.
        C=DAT0 W        C(W) := partie imaginaire
        GOTO   EXIT2    et sortie

*****************************************************

CSTO    GOSUB  CKVAR    Vérifier si variable valide
        D1=D1+ 2        sauter le "header" 0E
        C=DAT1 W        C(W) := partie imaginaire
        D1=D1+ 16       D1 := ^ partie im. sur la M.S.
WRITR   DAT0=C W        écrire partie imaginaire
        D0=D0- 16       D0 := ^ partie réelle de la var
        C=DAT1 W        C(W) := partie réelle
        DAT0=C W        écrire partie réelle
        C=D    A        restaurer D1
        D1=C            à son ancienne valeur
        GOTO   EXIT1    et sortie

*****************************************************

RSTO    GOSUB  CKVAR    vérifier si variable valide
        C=0    W        Partie imaginaire := 0
        GOC    WRITR    B.E.T.

*****************************************************

CKVAR   CD1EX           sauver D1 dans D(A)
        D=C    A
        D1=C
        D0=(5) #2F880   lire type de la variable
        C=DAT0 S        dans la zone scratch
        D0=D0- 15       D0 := ^ adr. var dans scratch
        C=DAT0 A        C(A) := adresse de la variable
        D0=C            D0 := ^ variable
        D0=D0+ 16       D0 := ^ partie imaginaire
        C=C+1  S        vérifier le type
        C=C+1  S        si c'est E : complexe
        RTNC            alors retour avec Carry = 1
rdatty  GOVLNG RDATTY   sinon : ERR: Data Type

*****************************************************

        NIBHEX 811      1 paramètre numérique (R ou C)
REPTE   GOSUB  pop1n    dépiler partie imaginaire
expr    GOVLNG EXPR     et retour...

*****************************************************

        NIBHEX 811      1 paramètre numérique (R ou C)
CONJE   GOSUB  pop1n    dépiler l'argument
        GONC   expr     si réel, c'est terminé
        C=R0            C(W) := partie imaginaire
        C=-C-1 S        C(W) := - partie imaginaire
        GOSUB  PUSH     empiler C(W) sur la M.S.
        LCHEX  0E       et mettre le "header" 0E
        GOSBVL STKCHR   sur la M.S.
        GONC   expr     B.E.T.

*****************************************************

NUM     GOSBVL POP2N    dépiler deux nombres de la M.S.
        GOC    rdatty   erreur si il y en a un complexe
        GOTO   EXIT2    sinon sortie

*****************************************************

        NIBHEX 811      1 paramètre numérique (R ou C)
IMPTE   GOSUB  pop1n    dépiler un argument de la M.S.
        C=0    W        si c'est un réel,
        GONC   fnrtn4   alors renvoyer 0
        C=R0            sinon partie imaginaire
        GOC    fnrtn4

*****************************************************

        NIBHEX 811      1 paramètre numérique (R ou C)
MAGE    GOSUB  pop1n    dépiler un argument de la M.S.
        C=A    W        si c'est un réel,
        GONC   fnrtn4   alors on peut le renvoyer
        SETDEC
        GOSUB  mp2-12   AB := Re * Re
        GOSBVL EXAB1    sauver dans Scratch1
        C=A    W        C(W) := Partie imaginaire
        GOSUB  mp2-12   AB := Im * Im
        GOSUB  rccd1    CD := Re * Re
        GOSUB  ad2-15   AB := Re * Re + Im * Im
        GOSBVL SQR-15   AB := SRQ (Re * Re + Im * Im)
        GOSUB  ures12   conversion en 12-chiffres

*****************************************************

fnrtn4  GOVLNG FNRTN4   pour sauver quelques quartets
pop1n   GOVLNG POP1N
ures12  GOVLNG uRES12
ad2-15  GOVLNG AD2-15
mp2-12  GOVLNG MP2-12
stscr   GOVLNG STSCR
rccd1   GOVLNG RCCD1

        END
```

FULL ERROR JACKET

Voici trois mots-clef de plus qui vont accroître la souplesse de vos programmes Basic. Leur but est de clarifier les dialogues entre vos sous programmes et l'utilisateur. Vous faites des programmes qui affichent des messages d'erreur, mais si vous êtes restés au temps du DISP... @ BEEP, ces programmes deviennent vite un paquet inextricable de GOTO, et divers ON ERROR, qui ne favorisent que l'apparition de nibs farceurs.

La Solution Finale contre ces nibs farceurs, c'est ERRLEX. Il possède trois mercenaires impitoyables, qui tirent à vue. Le plus rustique est encore ERROR. Il n'a peur que d'une chose, c'est que le ON ERROR lui tombe sur la tête.

Son complice, plus astucieux, se nomme WARNING. Il est insensible aux ON ERROR, mais seuls les nibs futés penseront à faire SFLAG -1 pour le déjouer.

Le dernier, arme lourde uniquement programmable, engendrera une erreur dans l'environnement antérieur à l'environnement courant. Il s'appelle EEND. E comme Erreur, END comme fin des ennuis.

Prenons tout de suite un exemple. Si vous êtes en train de faire un programme qui doit traiter les cas d'erreur, commencez-le donc par un :

```
ON ERROR GOSUB ERR
```

et à 'ERR':, traitez vos erreurs, avec des IF ERRN=...AND ERRL=...THEN..., qui utilisent au choix un POP ou un RETURN, et après vos tests, mettez un DISP ERRL; @ ERROR ERRN pour la mise au point.

Pour des programmes de longs calculs, vous pouvez afficher des messages à l'aide de WARNING, dont on pourra s'affranchir au besoin par un SFLAG -1.

Il serait intéressant de faire un *compilateur de messages*, sorte de TRANSFORM INTO LEX qui fabriquerait un fichier Lex contenant des messages personnalisés accessibles, donc, par MSG$, ERROR, WARNING, et EEND. Cela augmenterait encore plus la souplesse d'utilisation des messages.

Si maintenant vous voulez simuler une véritable fonction avec un programme, si, par exemple, vous voulez créer la fonction TAN, bien qu'elle existe déjà, faites le programme suivant :

```
SUB TAN(A,B)
    IF COS(A)=0 THEN EEND 4
    B=SIN(A)/COS(A)
END SUB
```

Si vous faites un programme comportant la ligne 50 CALL TAN(A,B) et que A vaut 90 degrés, son exécution s'arrêtera sur un ERR L50: TAN=INF, et donc bien dans l'environnement du CALL. EEND effectue donc l'équivalent d'un END avant d'engendrer l'erreur. Notez qu'en mode TRACE FLOW, l'instruction EEND est signalée.

Remarquez enfin que ces trois instructions ne s'occupent que de la partie entière de la valeur absolue de leur argument, qu'elles obéissent à BEEP OFF/ON et à DELAY. ERROR et WARNING sont dispensées des préfixes de message habituels tels ERR Lxxx: ou WRN:, mais pas EEND. Si l'environnement de l'erreur est le clavier, le Lxxx n'existe pas comme préfixe. ERROR et EEND tiennent compte des ON ERROR, s'ils sont effectivement dans l'environnement concerné. Les trois instructions affectent bien entendu ERRN, ERRL, et ERRM$.

J'allais oublier le principal : la syntaxe. EEND est uniquement programmable. Les trois instructions sont valides dans un IF, et demandent un unique paramètre qui est une expression dont la valeur est un entier représentant un message. Il est donc du format *iimmm* où *ii* est l'ID du fichier Lex concerné (0 pour les messages standard) et *mmm* le numéro de message.

Les nibs farceurs n'ont donc qu'à bien se tenir. Ceci va nous... Bip ! End of file

Serge Vaudenay (124)

```
        LEX     'ERRLEX'
        ID      #5C
        MSG     0
        POLL    0
        ENTRY   eEEND
        CHAR    #C
        ENTRY   eERROR
        CHAR    #D
        ENTRY   eWARN
        CHAR    #D
        KEY     'EEND'
        TOKEN   1
        KEY     'ERROR'
        TOKEN   2
        KEY     'WARNING'
        TOKEN   3
        ENDTXT
t       EQU     1
=BSERR  EQU     #0939A
=CLOSEA EQU     #120E4
=CLPSTK EQU     #07D29
=DECHEX EQU     #1B2D2
=EXPEXC EQU     #0F186
=FIXDC  EQU     #05493
```

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$(181);      (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 !

Note : la salle est fermée pendant les congés scolaires. Les réunions reprendront à la rentrée de septembre. Les dates précises seront publiées dans un prochain *JPC*.

Pierre David (37)

NOUS EN AVONS

La coopérative du Club vous propose :

- 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 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).

En outre, le module **JPC Rom** pour HP-71 est disponible. Vous nous adressez votre Eprom CMT (de préférence 64 Ko), et nous la programmons suivant une des options ci-dessous :
- JPC Rom + Manuel, pour 600 F,
- JPC Rom + Manuel + vos propres programmes, pour 800 F.

Si vous souhaitez des renseignements complémentaires, 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	FF	
anciens numéros de JPC		x	47,40	FF	
année complète de JPC		x	300	FF	
I.D.S. du module Forth		x	250	FF	
V.A.S.M.		x	300	FF	
Manuel de service pour HP-41		x	75	FF	
Manuel de service pour HP-75		x	75	FF	
JPC Rom + Manuel		x	600	FF	
JPC Rom + Manuel + vos propres programmes		x	800	FF	
			Total		FF

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

```
=FIXP    EQU    #02A6E
=FLTDH   EQU    #1B223
=FUNCDO EQU    #2F8BB
=HEXDEC EQU    #0ECAF
=LNSKP- EQU    #089FF
=MFWRNQ EQU    #093C5
=NXTSTM EQU    #08A48
=POP1R  EQU    #0E8FD
* Attention: point d'entré non normalisé
POPSTK  EQU    #1978A
=SCOPCK EQU    #0915B
=STMBUF EQU    #090DF
=TRCLIN EQU    #0FEC4
=TRFLCK EQU    #0FE18
=TRTOEN EQU    #0FEBC
p811    GOVLNG =FIXP
d811    GOVLNG =FIXDC
        REL(5) d811
        REL(5) p811
eERROR  GOSUB  GETARG
        P=     2
bserr   GOVLNG =BSERR
        REL(5) d811
        REL(5) p811
eWARN   GOSUB  GETARG
        P=     10
        GOSBVL =MFWRNQ
        GOVLNG =NXTSTM
        REL(5) d811
        REL(5) p811
eEEND   GOSUB  GETARG
        DO=(5) =FUNCDO
        DATO=C A
        GOSBVL =STMBUF
        GOSBVL =TRFLCK
        GOC    EEND1
        GOSBVL =TRCLIN
        LCASC  'DNEE '
        DATO=C 10
        DO=DO+ 10
        GOSBVL =TRTOEN
EEND1   GOSBVL POPSTK
        GOC    EEND2
        GOSBVL =LNSKP-
        RO=A
        C=R1
        C=C+C  P
        GOC    EEND4
        ?C#O   P
        GOYES  EEND3
        GOSBVL =SCOPCK
        GONC   EEND4
EEND2   GOSBVL =CLPSTK
        GOSBVL =CLOSEA
EEND3   ST=0   #D
EEND4   DO=(5) =FUNCDO
        C=DATO A
        GOTO   bserr

GETARG  GOSBVL =EXPEXC
        GOSBVL =POP1R
        A=0    S
        GOSBVL =FLTDH
        GOSBVL =HEXDEC
        C=0    W
        ACEX   X
        ASR    W
        ASR    W
        ASR    W
        D=C    W
        GOSBVL =DECHEX
        ACEX   W
        CDEX   W
        ACEX   W
        GOSBVL =DECHEX
        C=D    W
        CSL    A
        CSL    A
        C=A    B
        RTN
        END
```

TENSION DE BATTERIE
FREQUENCE D'HORLOGE

Le problème

Il peut se produire qu'un disfonctionnement du HP-71 semble provenir d'une tension insuffisante de la batterie. Ainsi tel phénomène aberrant se produit si le HP-71 est alimenté sur piles, pourtant l'indicateur BAT ne s'affiche pas, et ne se produit plus lorsqu'il est relié au secteur par le transformateur HP82066.

Il serait souhaitable de disposer d'un moyen de mesure de l'état des batteries afin de pouvoir invalider ou non l'hypothèse de la cause du disfonctionnement, car il pourrait aussi résulter d'une erreur de programmation !

Comment le HP-71 détermine-t-il le niveau de la batterie ?

L'écran à cristaux liquides du HP-71 est contrôlé par trois circuits intégrés, chacun responsable d'une partie de l'écran. L'un de ces circuits est également pourvu d'un système de détermination du niveau de batterie. Ce système est réduit à un indicateur tout ou rien.

Le bit 3 du quartet 2E3FF peut être lu. La lecture de ce bit déclenche une lecture du niveau de batterie par

le circuit. Il faut environ 100 micro secondes pour que l'évaluation du niveau de batterie soit effective. Après ce délai, le bit est armé si la tension de la batterie est inférieure à un seuil, désarmé si la batterie est satisfaisante. Une deuxième lecture permet donc de savoir si la tension est suffisante.

Le système d'exploitation effectue une lecture de la batterie toutes les minutes. Si nécessaire, l'indicateur binaire BAT (-61) est armé.

Est-il possible d'obtenir une information plus précise ?

Le HP-71 contient deux horloges de technologie différentes, chacune destinée à une application particulière :

- l'horloge à quartz pour la mesure du temps,

- l'horloge à circuit LC pour rythmer le fonctionnement du CPU.

L'horloge à quartz a une fréquence de 32768 Hz (2^{15}) ; après une série de divisions par 2, cette oscillation est utilisée pour actualiser un compteur placé en mémoire en 2F3FF, sur six quartets. Ce compteur est mis à jour 512 fois par seconde. Le système procure une bonne précision pour la réalisation d'un chronomètre ainsi que pour le maintien d'une horloge au centième de seconde.

L'horloge LC est constituée par un condensateur et une bobine subminiature formant un circuit électrique oscillant sous une tension. Dans sa version actuelle, ce circuit a une fréquence d'environ 600 kHz. Il est prévu que le CPU puisse accepter une horloge d'un MHz. Cette horloge est 18 fois plus rapide que le quartz. C'est elle qui scande les opérations effectuées par le CPU. Chaque oscillation est appelé un cycle.

Il y a un peu plus de mille cycles CPU effectués entre deux mises à jour du compteur de l'horloge à quartz.

Ce qui nous importe, c'est que les variations des deux types d'horloge en fonction de la tension sont très différentes ; l'horloge à quartz est en pratique insensible tandis que l'horloge LC ralentit au fur et à mesure que la tension de la batterie diminue.

Par conséquent, si l'on compte le nombre d'oscillations du circuit LC pendant une durée délimitée par l'horloge à quartz, nous aurons la fréquence de l'horloge du CPU, et par comparaison avec une fréquence à une tension déterminée, nous pourrons estimer la tension de la batterie.

Le Lex

Ce n'est à coup sûr pas un exploit de programmation puisqu'il est une simple copie du code de la Rom.

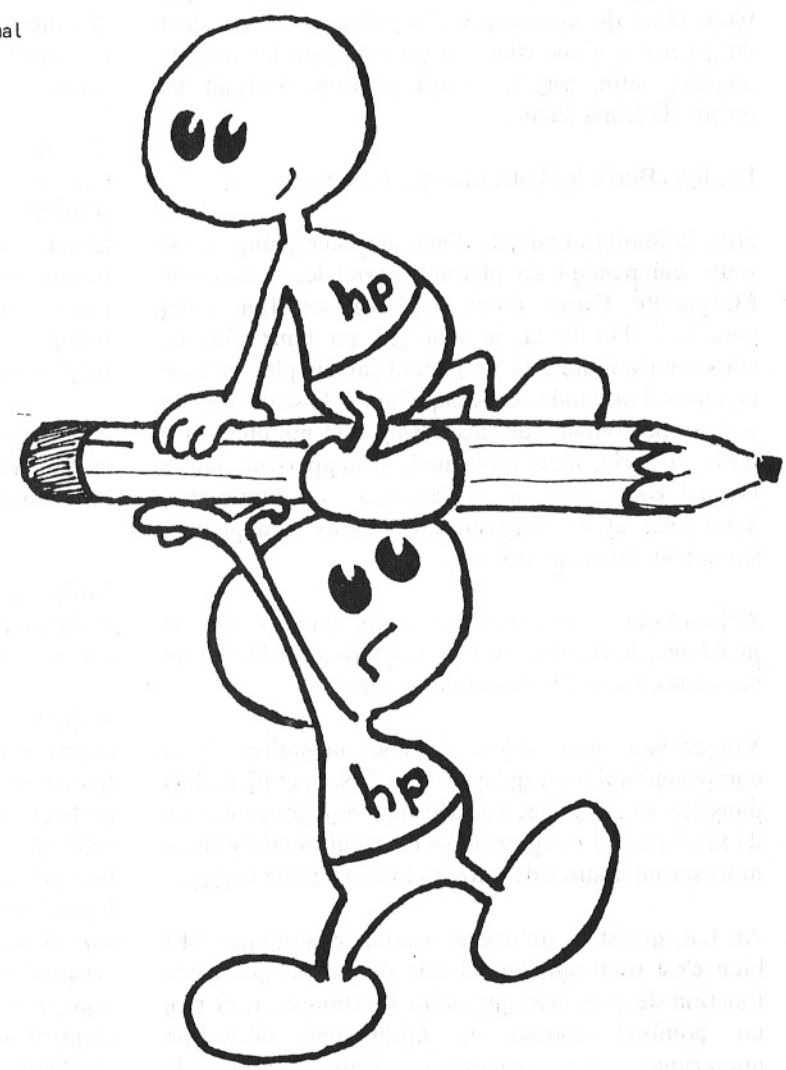
Après avoir chargé le Lex, au moyen de MAKELEX ou en utilisant le module Forth / Assembler, vous pourrez connaître la fréquence de votre HP-71. Si vous branchez un chargeur HP82066, vous obtiendrez la fréquence la plus élevée. Dès que vous serez en présence d'un jeu de piles provoquant l'affichage du témoin BAT, déterminez alors la fréquence du circuit LC. Vous aurez ainsi les deux bornes extrêmes de fonctionnement de votre HP-71. Il vous sera possible par la suite de connaître l'état de vos batteries.

Pour mon HP-71, BATST renvoie 650576 Hz lorqu'il sur secteur et 640784 Hz avec un jeu de piles défaillant.

Jean-Jacques Dhénin (177)

```
        LEX    'ACBATLEX'
        ID     #5C
        MSG    0
        POLL   0

        ENTRY  eBAT
        CHAR   #F
        KEY    'CLKSPEED'
        TOKEN  99
=FNRTN1 EQU    #0F216
=HDFLT  EQU    #1B31B
=TIMER2 EQU    #2E2F8
        ENDTXT

        NIBHEX 00
eBAT
        GOSUB  stD0D1   Sauve D0 D1 dans R4

clkspd  SETHEX
        P=     0        Nécessaire si 2 exécutions
        B=0    A        Compteur nombre de boucle
        A=R4            R4[A]=0...
        A=0    A        afin de déterminer...
        R4=A            s'il y a eu interruption
        D0=(5) =TIMER2
        INTOFF
        A=DAT0 B        Lit l'octet de pds faible
clks10  C=DAT0 B        Attend une mise à jour
        ?C=A   P        Top ?
        GOYES  clks10   Non. 7 cycles si passe
        C=DAT0 B        Relecture 15 cycles.
        P=     1        Détermine la durée. 2 c
        C=C-1  P        1/32 de sec. 4 cycles
        C=C-1  P        1/16 de sec. 4 cycles.
clks20  B=B+1  X        Compteur+1   6 cycles.
        A=DAT0 B        Lecture du timer. 15c
```

```
        ?C#A    B       Temps écoulé ? non 15c
        GOYES   clks20  Continue si temps. 8c
        A=0     A       "coton" de 7c
        B=B+1   X       termine le compte. 6c
        A=DAT0  B       Vérification.     15c
        ?C#A    B       Erreur ?   si oui 15c
        GOYES   clks20
        INTON
        A=R4            Y-a-t-il eu interruption
        ?A#0    A
        GOYES   clkspd  Recommencer
        P=      0
        B=B+B   A
        A=0     W
        A=B     A       2*B
        ASL     A       20*B
        B=B+B   A       4*B
        A=A+B   A       24*B
        C=0     A
        LCHEX   13
        A=A-C   A       24*B-13 (HEX)
        ASL     A       res*16
        GOSBVL  =HDFLT  Conversion en décimal
        C=A     W
        GOSUB   rcD0D1
        GOVLNG  =FNRTN1
*
stD0D1  AD0EX
        ASL     W
        ASL     W
        ASL     W
        ASL     W
        ASL     W
        AD1EX
        ASL     W
        ASL     W
        ASL     W
        ASL     W
        ASL     W
        R4=A
rcD0D1  A=R4
        ASR     W
        ASR     W
        ASR     W
        ASR     W
        ASR     W
        AD1EX
        ASR     W
        ASR     W
        ASR     W
        ASR     W
        ASR     W
        AD0EX
        RTNCC
        END
```



CALCUL SYMBOLIQUE (ACTE I)
LE DERIVATEUR (ACTE II)

JPC numéro 44 : le dérivateur acte 1.1 ; numéro 47 : le dérivateur acte 1.2, quelques erreurs corrigées et surtout le repérage de grosses fautes telles la dérivation des fonctions composées ; numéro 49 : le dérivateur acte 1.3 : correction d'une partie des erreurs, mais le problème relatif aux fonctions composées reste entier ; numéro 50 : enfin ! le problème relatif aux fonctions composées est résolu.

C'en était trop, une véritable marée d'indignation montait chez tout bon matheux qui se respecte. Il fallait faire quelque chose : nous l'avons fait. LE dérivateur existe, nous l'avons rencontré...

Par un beau mercredi d'automne, tandis que les petits oiseaux gazouillaient dans leurs nids douillets (en plein mois de novembre), 2 fondus se retrouvaient dans l'alcôve d'une chambre sordide, sous les toits du quartier latin. Sur un vieux pick-up, tournait un disque de Boris Vian...

Les faits Boris, les faits, rien que les faits !

Sous la lumière blafarde d'une ampoule jaunie de 35 watts qui pendait du plafond, parmi les oeuvres de Marguerite Duras (vous avez vu son film : les enfants ? Ho la la, je n'ai pas pu tenir plus de vingt-cinq minutes. Je ne m'était jamais plus embêté au cours d'une toile. J'ai craqué après la scène où l'on voit un personnage de dos durant huit minutes. Et il parle, il parle, mais moi aussi, je m'aperçois que le journal coûte cher, alors j'arrête.) , de Marguerite Yourcenar et de Stockhausen, 2 têtes bien pleines, sinon bien faites, pensaient...

Aujourd'hui, après maintes nuits passées sur le problème, le résultat de leur cogitation est là, devant vos yeux ébahis, LE dérivateur est né.

Voulez vous nous suivre dans les méandres de sa complexité qui n'est qu'apparente ? Non, et bien alors dans les 15 secondes, l'article que vous êtes en train de lire va se désintégrer. Mais ceux qui viendront avec nous seront assurés de faire un bien agréable voyage...

Au fait, qu'est ce qu'une dérivation symbolique ? Et bien c'est tout simplement une dérivation qui à une fonction de x associe une autre fonction de x, et non un nombre, comme le ferait une dérivation numérique. Et comment cette chose là fonctionne-t-elle ? C'est très simple quand on considère la chose pas à pas.

Nous avons, en construisant ce dérivateur, suivi le même mode de pensée que lorsqu'on dérive à la main. Chaque fois que l'homme, dans sa grande quête du quotidien, dérive une fonction $f(x)$, il suit le même raisonnement, lequel est le suivant :

Repérage des opérateurs, puis des fonctions de la variable, application des règles de dérivation, et assemblage de tous les morceaux. Sort de ce cheminement de la pensée humaine, une fonction $f'(x)$, dérivée symbolique de $f(x)$. Mais, j'en vois au fond de la salle qui lèvent le doigt en un signe désespéré de non compréhension de ce qui précède, quelles sont ces fameuses règles de la dérivation ? Le proverbe dit : "Tout vient à point à qui sait attendre". Je ne vous ferai quand même pas languir plus longtemps, et exposerai icelles dans l'instant.

Si nous notons u et v, deux fonctions de x, C une constante et ' l'opération de dérivation, alors nous avons :

$C' = 0$
$C \times x' = C$
$(C \times x^N)' = C \times N \times x^{N-1}$
$(C \times u)' = C \times (u)'$
$(u+v)' = u'+v'$
$(u \times v)' = u' \times v + u \times v'$
$(u/v)' = (u' \times v - u \times v')/v^2$
$(u^v)' = v \times u^{v-1} \times u' + u^v \times \ln(u) \times v'$

Ajoutons que cette dernière dérivée était inconnue de la plupart des personnes qu'il nous a éte donné de rencontrer lors de la mise au point de ce programme.

Voilà pour les opérateurs diadiques. Passons maintenant aux opérateurs monadiques, appelés ainsi car il ne traitent qu'une seule fonction :

$\cos(u)' = -\sin(u) \times u'$
$\sin(u)' = \cos(u) \times u'$
$\tan(u)' = 1/\cos(u)^2 \times u'$
$\cosh(u)' = \sinh(u) \times u'$
$\sinh(u)' = \cosh(u) \times u'$
$\tanh(u)' = 1/\cosh(u)^2 \times u'$
$\log(u)' = 1/u \times u'$
$\exp(u)' = \exp(u) \times u'$
$\text{acos}(u)' = -1/\text{sqr}(1-u^2) \times u'$
$\text{asin}(u)' = 1/\text{sqr}(1-u^2) \times u'$
$\text{atan}(u)' = 1/(1-u^2) \times u'$
$\text{acosh}(u)' = \pm 1/\text{sqr}(u^2-1) \times u'$
$\text{asinh}(u)' = 1/\text{sqr}(u^2+1) \times u'$
$\text{acosh}(u)' = 1/(1-u^2) \times u'$

Il nous suffisait donc, connaissant ces quelques règles, de les apprendre à notre bon vieux HP-71. Après une première mouture du programme entièrement en Basic, il nous a semblé judicieux d'écrire 3 nouvelles fonctions en assembleur. Elles sont : CHRTYP, ISNUM et MNOP. Voici leur utilité :

- CHRTYP rend le type du premier caractère de la chaîne :

```
0:  si parenthèse
1:  si signe +
2:  si signe -
3:  si signe *
4:  si signe /
5:  si signe ^
8:  si la chaîne est vide
10: si compris entre 0 et 9
12: si compris entre A et Z
15: dans tous les autres cas.
```

- ISNUM, rend un booléen. Si la chaîne de caractères entrée est un nombre, utilisable comme tel, la fonction rend 1, sinon elle rend 0. Des exemples:

1E4 → 1
2.3E-3 → 1
.0E1 → 1
+ + + - - + 121234.23145E + + - - + - + + + + - - + + 6387 → 1
1EF → 0
X → 0
1E-.2 → 0

- MNOP rend la position de l'opérateur, extérieur à des parenthèses, de plus basse priorité. Si un même opérateur se trouve plusieurs fois, la dernière occurence est prise en compte :

```
SIN(COS(TAN(2*X)-X)-1)*SQR(EXP(2*X))
                        ^
```

donne 23.

```
SIN(COS(TAN(2*X)-X)*SQR(EXP(2*X)))
```

donne 0, tous les opérateurs sont entourés de parenthèses.

```
X+X+X+X
    ^
```

donne 6.

Voici donc le raisonnement que la machine suit, ligne par ligne :

- 60
Si, en entrée, on a une constante, repérée par l'absence de la variable de dérivation dans la chaîne F$, contenant la fonction à dériver, on rend immédiatement le résultat dans la chaîne qui contient la dérivée F1$ = '0'

- 70

Si, en entrée, on a à dériver la fonction identité, le résultat est immédiat, la valeur de la dérivée est 1.

- 80

On dimensionne les variables U$ et V$ à 192 caractères, pour plus de sureté.

- 90

Le but de cette fonction est expliqué plus haut.

- 100

Ligne clé du programme. Si O n'est pas nulle, alors un opérateur a été trouvé, et le travail se poursuit en ligne 110. Si O vaut 0, alors il n'y a pas d'opérateur à l'extérieur d'une parenthèse, et on continue en ligne 420.

- 110

CHRTYP rend dans P la priorité de l'opérateur trouvé 2 lignes plus haut.

- 120
U$ et V$ sont dimensionnées de façon à utiliser le moins de mémoire possible. Il ne faut pas oublier que ce programme est récursif, et qu'il prend donc beaucoup de place en mémoire.

- 130
F$ est découpée en deux, d'une part U$, contenant la première partie de l'expression, avant l'opérateur, d'autre part V$, contenant la seconde partie, après icelui.

- 140,150
C'est quand même beau la récursivité. En effet, il nous suffit maintenant de dériver U$ dans U1$, et V$ dans V1$.

- 160
Et hop, une des toutes nouvelles fonctions de P.D. et J.T. SELECT, CASE, CASE ELSE, END SELECT. Comme toujours un vrai plaisir de les utiliser (amis, quand donc vous arrêterez-vous ? jamais nous l'espérons).

- 170,180

Dans le cas où P vaut 1 ou 2, on effectue une addition ou une soustraction à l'aide d'une fonction qui sera décrite plus bas, en appliquant la règle décrite, elle, plus haut. En sautant de CASE en CASE, on va jusqu'en ligne 410.

- 190

On raisonne de façon identique, et on applique la règle de dérivation pour multiplication ou division.

- 200,210

Ces lignes ne sont parcourues que si l'opérateur est une division, et que le diviseur est un nombre, et non une fonction. Le cas de la division par 0 est traité à cet endroit.

- 230

La fonction de construction sera décrite plus bas.

- 240 à 270 :

. 240

Considérons le cas où le dénominateur est différent de 0.

. 250

Si le dénominateur est différent de 1, on construit le carré d'icelui.

. 260

S'il faut mettre les parenthèses, on les met ! Sinon, et ben, on les met pas, et toc !

- 290 à 400

Cas 5, il s'agit de la fonction puissance, dont peu de gens, pensons-nous, connaissent la dérivée. La ligne 370 nous donne un aperçu de la capacité de traitement des fonctions utilisateurs du HP-71.

. 300

Si l'exposant est numérique, on le raccourcit d'une unité.

. 310

Si l'exposant est nul, on renvoie 1.

. 320

Si l'exposant vaut 1, on renvoie U$.

. 330

Enfin, si l'exposant est négatif, on l'entoure de parenthèses.

. 350

Dans le cas où l'exposant n'est pas numérique, on construit la partie exponentiation de la dérivée.

. 370

Et comme la lutte du même nom, nous renvoyons le résultat final de la dérivée. Notez les 5 appels de fonctions utilisateurs.

. 380

Dans le cas où U$ vaut 0, la dérivée n'est pas définie, et on retourne NaN.

. 390

Si U$ vaut 1, on retourne 0.

- 410

On tombe ici lorsqu'aucun opérateur n'a été repéré dans la chaîne F$. Nous sommes dans le domaine des fonctions composées.

- 420

Q contient la position de la première parenthèse, ou 0 si la chaîne n'en contient pas.

- 430

Si F$ ne contient pas de parenthèses, nous avons affaire à une constante, et, bon an, mal an, suivant notre bon petit bonhomme de chemin, nous rendons 0.

- 440

Autre partie clé du programme : si la parenthèse est en première position de la chaîne, cela signifie qu'il faut éliminer icelle (la parenthèse, bien sur, pour ceusses qui n'auraient pas compris, cong), la parenthèse de fin, et envoyer cette sous-chaîne à analyser dans le dérivateur. La ligne se termine par un END, car en fin de dérivation, le travail est fini. Aristoteliciennement parlant, PDJT ne va pas aimer. On s'en fiche.

- 450,460

A cet endroit du programme, la chaîne F$ contient une fonction monadique d'une fonction de X. Appelons *M* cette fonction, on sait alors que *M'(U)* est fonction de *U*, et de *U'*. Il nous faut donc calculer *U'* et, pour cela, connaître la valeur de *U*. C'est le but de ces lignes qui effectuent la dérivation (toujours de façon récursive) de *U*.

- 470

On dérive cette fonction.

- 480

Si U1$ vaut 0, alors on poursuit en ligne 700 sinon, on passe à la ligne suivante.

- 490

Si U$ contient au moins un opérateur, et que dans la construction de la dérivée, U$ doit être élevé au carré, alors, on l'habille de parenthèses.

- 500 à 680

Cette partie du programme contient la table des dérivées pour les fonctions monadiques. Si par le plus grand des hasards, il en manquait, la structure même du programme permettrait de les ajouter de façon très simple.

Si la fonction *M* n'est pas reconnue, elle est dérivée symboliquement en i', en ligne 670.

- 690
On construit *M'(U)×U'* dans la chaîne F1$.

- 700
U$ étant une constante, U1$ est nul, donc F1$ vaut 0.

- 710 à 730
Fermeture de tous les THEN multilignes, et du reste. C'est fini !

- 740 FNS$(U$,V$,S$)
Cette fonction rend possible l'addition de deux chaînes de caractères. Il est bien entendu nécessaire de s'affranchir de résultats tels X+0, 2*X-0, ou +COS(X).

- 750

Si U$ et V$ sont numériques, on effectue l'addition au sens propre du terme, et on branche à END DEF en ligne 940.

- 770

Un petit dimensionnement n'a jamais fait de mal à personne.

- 780 à 820

On traite le cas de l'addition :

. 790

On évite les +- disgracieux.

. 800

U$+U$=2*U$

. 810

V$+0=V$

. 820

U$+0=U$

- 840 à 900

On traite le cas de la soustraction.

. 840
-- donne +.

. 850 à 880
U$ vaut 0, et le résultat est traité en fonction du signe de V$.

. 890
U$-0=U$.

. 900
U$-U$=0.

- 920
Visiblement, on gère les parenthèses autour du résultat. OA était fatigué, moi aussi, et en plus, on avait faim..., si vous ne comprenez pas, faites 3615 code JNCP.

- 950 FNP$(U$,V$)
Fonction de multiplication des deux chaînes U$ et V$. Comme pour l'addition, on simplifie les résutats d'une multiplication par 0 ou par 1.

- 960
U = 0, ça s'impose.

- 970 à 1020
Le résultat de la multiplication est différent de 0, en effet, nous sommes dans un anneau intègre.

. 980,990
On mémorise, grâce à la très utilisée fonction EXOR le signe du produit. De plus, fort de cette connaissance, on se permet de travailler sur des valeurs absolues.

. 1000
On construit le produit, précédé de son signe, dans le cas général.

. 1010,1020
on traite le cas où l'un des facteurs vaut 1.

- 1040
Cas où l'un des facteurs est nul. Nous ne vous ferons pas l'offense de vous préciser le résultat.

Et voilà le travail. En fait, dans son intégralité, ce programme suit le mode de raisonnement de l'esprit humain. Il est totalement récursif, en ce sens qu'il dérive uniquement les fonctions les plus simples, réservant aux plus compliquées une bonne dilution recursive de derrière les fagots, qui les rend de plus en plus faciles à traiter.

Ce programme a néammoins une petite limitation, il peut lui arriver de rendre des expressions surparenthésées. Mais cela ne saurait durer, nous poursuivons nos recherches sur le calcul symbolique.

Olivier Arbey (118)
Laurent Istria (3)
(Les fondus)

```
        LEX     'HALEX'
=id     EQU     #5D
FNRTN1  EQU     #0F216
HDFLT   EQU     #1B31B
POP1S   EQU     #0BD38
BSERR   EQU     #0939A
DRANGE  EQU     #1B076
        CON(2)  =id
        CON(2)  7
        CON(2)  9
        CON(5)  0
        NIBHEX  F
        REL(4)  1+TxTbSt
        REL(4)  MSGTBL
        REL(5)  POLHND
        CON(3)  (TxEn01)-(TxTbSt)
        REL(5)  Typecar
        CON(1)  #F
        CON(3)  (TxEn02)-(TxTbSt)
        REL(5)  Number
        CON(1)  #F
        CON(3)  (TxEn03)-(TxTbSt)
        REL(5)  Pos
        CON(1)  #F
TxTbSt
TxEn01  CON(1)  11
        NIBASC  'CHRTYP'
        CON(2)  7
TxEn02  CON(1)  9
        NIBASC  'ISNUM'
        CON(2)  8
TxEn03  CON(1)  7
        NIBASC  'MNOP'
        CON(2)  9
        NIBHEX  1FF

MSGTBL
        CON(2)  1       plus petit
        CON(2)  2       plus grand
* SL:V1 message bidon, Mais obligatoire.
        CON(2)  16
        CON(2)  2       Message # 2
        CON(1)  4
        NIBASC  'SL:V1'
        CON(1)  12

* Parenthesis omited

ePROM   EQU     #00001
        CON(2)  43
        CON(2)  ePROM   Message # 1
        CON(1)  10
        NIBASC  'Parenthe'
        NIBASC  'sis'
        CON(1)  6
        NIBASC  ' omited'
        CON(1)  12
        NIBHEX  FF
POLHND
        ?B=0    B       Interception du Poll VER$
        GOYES   Trt
        GONC    Rsxm
```

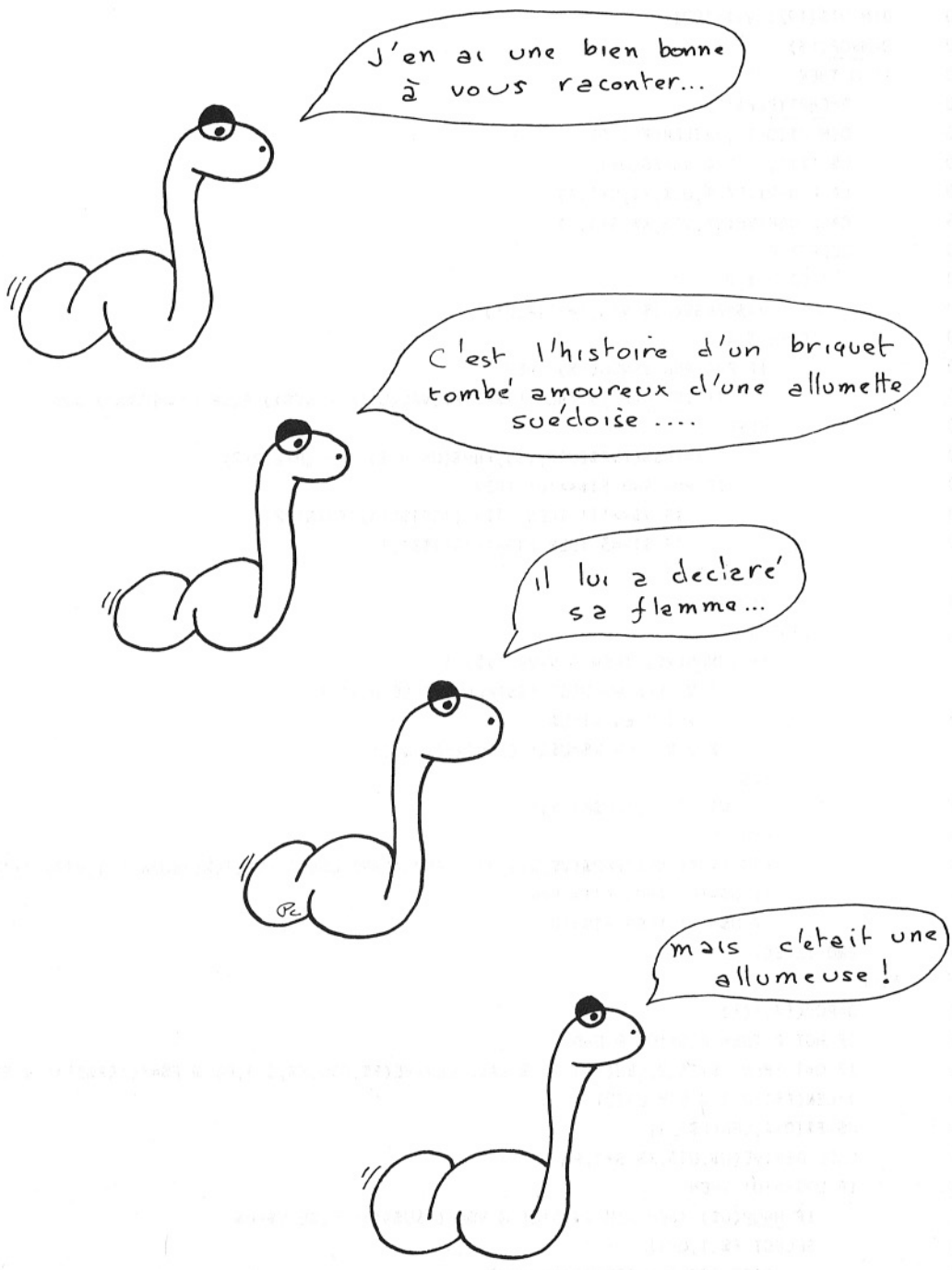
```
Trt     C=R3
        D1=C
        A=R2
        D1=D1-  14
        CD1EX
        ?C<A    A
        GOYES   Rsxm
        D1=C
        R3=C
        LCASC   ' CS:A02'  Calcul Symbolique
        P=      13
        DAT1=C  WP
Rsxm    RTNSXM

************************************************
*
* CHRTYP
*
* But: Implemente la fonction CHRTYP
*
* Entree:
*     - P = 0
*     - D1= Stack pointer
*     - D0= Program counter
*
* Sortie:
*     - D1 mis a jour
*     - sort par FNRTN1
*
* Appel: CARTYP
*
* Abime: A, C, D1, R1, P, S1, S2
*
* Niveau : 4
*
* Historique:
*     87/12/16 : L.I. Conception et codage
*     87/12/19 : L.I. Documentation
*
************************************************

        NIBHEX  411
Typecar GOSUB   store
        ?A#0    A
        GOYES   st
        P=      8       chaine vide
        GONC    Fin
st      D1=D1-  2       D1 pointe premier caractere
        A=DAT1  B       lit le caractere
        GOSUB   =CARTYP analyse du caractere
Fin     C=0     A
        C=P     0       place la valeur dans C
        A=C     A       place la valeur dans A
Sortie  GOSBVL  HDFLT   traduit en flottant
        CR1EX
        D1=C
        C=A     W
        GOVLNG  FNRTN1  retour a BASIC

************************************************
*
* store
*
* But: calcul de la longueur d'une chaine,
*      et sauvegarde de D1
*
* Entree:
*     - P = 0
*     - D1= stack pointer
*     - D0= program counter
*
* Sortie:
*     - A(A)= longueur chaine en quartet
*     - C(A)= longueur chaine en caracteres
*     - D1  = pointe le premier caractere
*     - R1  = D1 restaure pour la sortie
*
* Appel: rien
*
* Abime: A(A), C(A), D1, R1
*
* Niveau : 1
*
* Historique:
*     88/01/21 : L.I. Conception et codage
*     88/01/21 : L.I. Documentation
*
************************************************

store   GOSBVL  POP1S   lit la chaine
        CD1EX           C := ^ premier caractere
        C=C+A   A       debut de la chaine
        R1=C            stockage dans R1
        D1=C            dans D1
        C=0     W       pour le shift
        C=A     A       longueur en quartets
        CSRB            longueur en caracteres
        RTN

************************************************
*
* MNOP
*
* But: Implemente la fonction MNOP
*
* Entree:
*     - P = 0
*     - D1= Stack pointer
*     - D0= Program counter
*
* Sortie:
*     - D1 mis a jour
*     - sort par FNRTN1
*
* Appel: POP1S, CARTYP, HDFLT
*
* Abime: A,B(S),B(A),C,D(A),D1,P,R0-R2,S1-S3
```

```
*
* Niveaux : 4
*
* Historique:
*     87/12/18 : L.I. Conception et codage
*     87/12/19 : L.I. Documentation
*     87/12/19 : L.I. Bug corrigee: D decremente
*                     1 fois en entree
*     88/01/19 : L.I. nombre avec exposant traite
*
************************************************

        NIBHEX  411     une seule chaine
Pos     GOSUB   store
        R2=C
        R0=C
        ?A#0    A       si chaine vide
        GOYES   S0
        GOTO    F0
S0      D1=D1-  2       pointe premier caractere
        D=C     A       compteur caracteres dans D
        B=0     A       compteur parentheses a 0
        P=      15
        LCHEX   6
        A=C     P       A(15)=6
        ST=1    3       assure la validite
         *              du premier caractere
        D=D-1   A
Bcl     P=      0
        A=DAT1  B       boucle de traitement
         *              lecture du caractere
        LCASC   '('
        ?A#C    B
        GOYES   S1
        B=B+1   A       parenthese ouvrante
        GONC    NX      caractere suivant
S1      C=C+1   B
        ?A#C    B
        GOYES   S2
        B=B-1   A       parenthese fermante
        GONC    NX      caractere suivant
S2      ?B#0    A
        GOYES   NX      pas niveau 0 de parenthese
        GOSUB   =CARTYP
        C=P     15      C(15)= valeur du caractere
        ?C>A    S       priorite plus grande ?
        GOYES   NX      caractere suivant
        ?ST#0   3       premier caractere ?
        GOYES   stock   oui, pas de probleme
        B=C     S
        B=B-1   S
        B=B-1   S
        GOC     sgn     c'est un signe plus
        B=B-1   S
        GOC     sgn     c'est un signe moins
stock   A=C     S       A(15) := priorite
        C=D     A
        R0=C            position du car. dans R0
NX      D1=D1-  2       caractere suivant
        ST=0    3
        D=D-1   A       un caractere de moins
        GONC    Bcl     s'il en reste, on continue
        ?B=0    A       reste-t-il des parentheses ?
        GOYES   F0
        P=      0       P=0 en entree, et on charge
        LC(4)   (=id)*256+ePROM avec l'erreur 1
        GOVLNG  BSERR
F0      C=R0            position dans C
        A=R2            longueur de la chaine dans A
        A=A-C   A
        GOTO    Sortie
sgn     P=      3
        D1=D1+  2       2 caracteres precedents
        A=DAT1  WP      dans A(0-3)
        D1=D1-  2
        P=      0
        LCASC   'E'
        ?A#C    B       etait-ce un E ?
        GOYES   stock
        ASRC
        ASRC
        GOSUB   =CARTYP
        ?P=     10
        GOYES   NX      precede d'un chiffre ?
        GONC    stock


************************************************
*
* ISNUM
*
* But: Implemente la fonction ISNUM
*
* Entree:
*     - D1 = stack pointer
*     - D0 = PC
*     - P  = 0
*
* Sortie:
*     - D1 mis a jour
*     - sort par FNRTN1
*     - 0: la chaine n'est pas numerique pure
*     - 1: la chaine est un numerique pur
*
* Appel: POP1S, DRANGE, Nxt, lpnum
*
* Abime: A(A), C, D(A), P, R1
*
* Niveaux: 1
*
* Historique:
*     87/12/00 : A.O. & L.I. Conception et codage
*     88/01/09 : A.O. & L.I. Documentation
*     88/01/22 : L.I. traite, plusieurs signes
*
************************************************
```

```
        NIBHEX  411
Number  GOSUB   store
        ?A#0    A       chaine vide ?
        GOYES   anal    non
        GOTO    Fn0
anal    D1=D1-  2       pointe le premier caractere
        D=C     A       nombre de caracteres dans D
        P=      0
        D=D-1   A       1 en moins pour la boucle
        LCASC   '('
        B=0     A
        A=DAT1  B
bparL   ?A#C    B
        GOYES   trnum
        B=B+1   A
        D=D-1   A
        GOSUB   Nxt
        GOC     Fn0
        GONC    bparL
trnum   GOSUB   sgn?
        GOC     Fn0     chaine termine par operateur
        LCASC   '.'
        ?A#C    B       est-ce un point ?
        GOYES   num1    si non meme caractere
        GOSUB   Nxt     caractere suivant
        GOC     Fn0     non num
num1    GOSBVL  DRANGE  compris entre 0 et 9 ?
        GOC     Fn0     non num
        GOSUB   lpnum   on a un num,
         *              on cherche jusqu'ou
        GONC    Fn1     si bout de ligne -> numerique
        LCASC   '.'
        ?A#C    B       est-ce un point ?
        GOYES   spt     si non meme caractere
        GOSUB   Nxt     caractere suivant
        GOC     Fn1
spt     GOSBVL  DRANGE
        GOC     exp?
        GOSUB   lpnum
        GONC    Fn1
exp?    LCASC   'E'
        ?A#C    B
        GOYES   Fn0
        GOSUB   Nxt
        GOC     Fn0
        A=DAT1  B
        GOSUB   sgn?
        GOC     Fn0     un operateur termine la chaine
        GOSBVL  DRANGE
        GOC     Fn0
        GOSUB   lpnum
        GONC    Fn1
Fn0     P=      0
        GOTO    Fin
Fn1     C=B     A
        D=C     A
        P=      0
        LCASC   ')'
        D=D-1   A
        GOC     SR1
        A=DAT1  B
bparR   ?A#C    B
        GOYES   Fn0
        GOSUB   Nxt
        GOC     SR1
        GONC    bparR
SR1     P=      1       renvoie 1
        GOTO    Fin


*************************************************
*
* sgn?
*
* But: passer une suite de signe + et -
*
* Entree:
*     - A(B) = premier caractere a traiter
*     - D1   = pointe ce caractere
*     - D(A) = nombre de caracteres jusqu'au bout
*       de la chaine moins 1
*
* Sortie:
*     - D1 = pointe derriere le dernier operateur
*     - Cy = 1: un operateur termine la chaine
*     - Cy = 0: il y a un caractere derriere le
*                 dernier operateur
*               il n'y a pas d'operateur
*               dans ces deux cas, sortie valide
*
* Appel: Nxt
*
* Abime: C(1-0), Cy
*
* Niveau : 1
*
* Historique:
*     88/01/22 : L.I. Conception et codage
*     88/01/22 : L.I. Documentation
*
*************************************************

sgn?    LCASC   '-'
        ?A#C    B       est-ce un signe moins ?
        GOYES   pl1     non, est-ce un "+"
        GOSUB   Nxt     oui, caractere suivant
        RTNC
        GONC    sgn?    des fois qu'il y en
         *              ait plusieurs
pl1     LCASC   '+'
        ?A=C    B       est-ce un signe plus ?
        GOYES   nx
        RTNCC
nx      GOSUB   Nxt     si oui on passe au suivant
        RTNC
        GONC    sgn?    100 fois sur le metier...
```

```
************************************************
*
* lpnum
*
* But: lit une chaine de caracteres jusqu'a
*      epuisement des chiffres
*
* Entree:
*     - A(B) 1er caractere de la chaine
*     - D1 pointe ce caractere
*     - D(A) = nombre de caracteres jusqu'au bout
*       de la chaine moins 1
*
* Sortie:
*     - Cy=0 tous les caractere sont numeriques
*     - Cy=1 un caractere non numerique
*     - D1 pointe ce caractere non numerique
*
* Appel: DRANGE, Nxt
*
* Abime: D(A), A(B), D1, Cy, C(A)
*
* Niveau: 1
*
* Historique:
*     87/12/19 : A.O. & L.I. Conception et codage
*     88/01/09 : A.O. & L.I. Documentation
*
************************************************

lpnum   GOSBVL  DRANGE  dans l'intervalle 0-9 ?
        RTNC            Cy set si non
        GOSUB   Nxt     Caractere suivant
        GONC    lpnum   tant qu'il en reste...
        RTNCC           si on arrive au dernier
         *              on baisse Cy

************************************************
*
* Nxt
*
* But: Passe au caractere suivant en verifiant
*      qu'il en reste
*
* Entree:
*     - D1: pointe un des caracteres de la chaine
*     - D(A): nombre de caracteres moins 1
*
* Sortie:
*     - Cy=1 il ne reste plus de caractere
*     - Cy=0 il reste encore des caracteres
*
* Appel: Rien
*
* Abime: D(A), A(B), D1, Cy
*
* Niveau: 0
*
* Historique:
*     87/12/00 : A.O. & L.I. Conception et codage
*     88/01/09 : A.O. & L.I. Documentation
*
************************************************

Nxt     D1=D1-  2       caractere suivant
        A=DAT1  B
        D=D-1   A       mise a jour de D
        RTNC            en fin de ligne on leve Cy
        RTNCC           sinon baisse

************************************************
*
* CARTYP
*
* But: Trouver le type d'un caractere.
*
* Entree:
*     - A(B) = Code du caractere
*
* Sortie:
*     - P contient le code du caractere
*       definit comme suit
*       P = 0  : ( ou )
*       P = 1  : +
*       P = 2  : -
*       P = 3  : *
*       P = 4  : /
*       P = 5  : ^
*       P = 10 : Un nombre de 0 a 9
*       P = 12 : Une lettre de A a Z
*       P = 15 : Tout autre caractere
*
* Appel: CATCHR
*
* Abime: C, P, S1, S2
*
* Niveaux: 3
*
* Historique:
*     87/12/16 : A.O. & L.I. Conception et codage
*     87/12/19 : L.I. Documentation
*     87/12/19 : A.O. & L.I. Utilisation de CATCHR
*
************************************************

CATCHR  EQU     #03F70

=CARTYP
        P=      0
        LCASC   ')^/*-+(' operateurs et parentheses
bcl     ?A=C    B       operateur traite
        GOYES   fin     si egal, fin
        P=P+1           mise a jour de P
        CSR     W
        CSR     W       decalage operateur suivant
        ?P#     7       tous testes ?
```

```
        GOYES   bcl     si non on continue
        GOSBVL  CATCHR
        GONC    P15     ni alpha ni num si Cy=1
        P=      10
        ?ST=1   1       si numerique
        GOYES   fin
        P=      12
        GONC    fin     B.E.T.
P15     P=      15      sinon P vaut 15
fin     ?P#     6       si P vaut 6 ")"
        RTNYES
        P=      0       alors P a 0
        RTN
```



Programme "DERIVE" (calcul de dérivées symboliques, nécessite HALEX)

```
   - HP71 Le Derivateur 5.7 (Lex)
     Copyright (c) les Fondus 1987
  10 DIM F$[96],F1$[192]
  20 INPUT 'F(X)=? ',F$;F$
  30 CALL DERIVE(F$,F1$,'X',0,1)
  40 DISP "F'(X)=";F1$

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

  50 SUB DERIVE(F$,F1$,X$,S,S1)
  60     IF NOT POS(F$,X$) THEN F1$='0' @ END
  70     IF F$=X$ THEN F1$='1' @ END
  80     DIM U1$[192],V1$[192]
- 90     O=MNOP(F$)
 100     IF O THEN
- 110         P=CHRTYP(F$[O])
 120         DIM U$[O-1],V$[LEN(F$)-O]
 130         U$=F$[1,O-1] @ V$=F$[O+1]
 140         CALL DERIVE(U$,U1$,X$,S+1,P)
 150         CALL DERIVE(V$,V1$,X$,S+1,P)
 160         SELECT P
 170             CASE 1,2
 180                 F1$=FNS$(U1$,V1$,'+-'[P,P])
 190             CASE 3,4
- 200                 IF P=4 AND ISNUM(V$) THEN
- 210                     IF ISNUM(U1$) THEN F1$=STR$(VAL(U1$&'/'&V$)) ELSE F1$=U1$&'/'&V$
 220                 ELSE
 230                     F1$=FNS$(FNP$(U1$,V$),FNP$(U$,V1$),'+-'[P-2,P-2])
 240                     IF P=4 AND F1$<>'0' THEN
 250                         IF V$<>'1' THEN F1$='('&F1$&')'/'&V$&'^2'
 260                         IF S1>=3 THEN F1$='('&F1$&')'
 270                     END IF
 280                 END IF
 290             CASE 5
- 300                 IF ISNUM(V$) THEN @ V=VAL(V$)-1
 310                     IF V THEN W$=U$&'^'&STR$(V) ELSE W$='1'
 320                     IF V=1 THEN W$=U$
 330                     IF V<0 THEN W$=U$&'^('&STR$(V)&')'
 340                 ELSE
 350                     W$=U$&'^('&V$&'-1)'
 360                 END IF
 370                 F1$=FNS$(FNP$(FNP$(V$,W$),U1$),FNP$(FNP$(U$&'^'&V$,'LN('&U$&')'),V1$),'+')
 380                 IF U$='0' THEN F1$='NaN'
 390                 IF U$='1' THEN F1$='0'
 400         END SELECT
 410     ELSE
 420         Q=POS(F$,'(')
 430         IF NOT Q THEN F1$='0' @ END
 440         IF Q=1 THEN F$=F$[2,LEN(F$)-1] @ CALL DERIVE(F$,F1$,X$,S+1,P) @ F$='('&F$&')' @ END
 450         D=LEN(F$)-Q-1 @ DIM U$[D]
 460         U$=F$[Q+1,LEN(F$)-1]
 470         CALL DERIVE(U$,U1$,X$,S+1,P)
 480         IF U1$<>'0' THEN
- 490             IF MNOP(U$) THEN DIM V$[D+2] @ V$='('&U$&')' ELSE V$=U$
 500             SELECT F$[1,Q-1]
 510                 CASE 'COS' @ F1$='-SIN('&U$
```

```
520                 CASE 'SIN' @ F1$='COS('&U$
530                 CASE 'TAN' @ F1$='(1+TAN('&U$&')^2'
540                 CASE 'COSH' @ F1$='SINH('&U$
550                 CASE 'SINH' @ F1$='COSH('&U$
560                 CASE 'TANH' @ F1$='(1/COSH('&U$&')^2'
570                 CASE 'LOG','LN' @ F1$='1/('&U$
580                 CASE 'LOG10','LGT' @ F1$='.43429448190325182765/('&U$
590                 CASE 'EXP' @ F1$='EXP('&U$
600                 CASE 'SQR','SQRT' @ F1$='1/(2*SQR('&U$&')'
610                 CASE 'ACOS' @ F1$='-1/SQR(1-'&V$&'^2'
620                 CASE 'ASIN' @ F1$='1/SQR(1-'&V$&'^2'
630                 CASE 'ATAN' @ F1$='1/(1+'&V$&'^2'
640                 CASE 'ACOSH' @ F1$='1/SQR('&V$&'^2-1'
650                 CASE 'ASINH' @ F1$='1/SQR('&V$&'^2+1'
660                 CASE 'ATANH' @ F1$='1/(1-'&V$&'^2'
670                 CASE ELSE @ F1$=F$[1,Q-1]&"'("&U$
680             END SELECT
690             F1$=FNP$(F1$&')',U1$)
700         ELSE @ F1$='0'
710         END IF
720     END IF
730 END

740 DEF FNS$[192](U$,V$,S$)
750     IF ISNUM(U$) AND ISNUM(V$) THEN @ FNS$=STR$(VAL(U$&S$&V$))
760     ELSE
770         DIM M$[192]
780         IF S$='+' THEN
790             IF V$[1,1]='-' THEN M$=U$&V$ ELSE M$=U$&'+'&V$
800             IF U$=V$ THEN M$='2*'&U$ @ SFLAG 1
810             IF U$='0' THEN M$=V$ @ SFLAG 1
820             IF V$='0' THEN M$=U$ @ SFLAG 1
830         ELSE
840             IF V$[1,1]='-' THEN M$=U$&'+'&V$[2] ELSE M$=U$&'-'&V$
850             IF U$='0' THEN
860                 SFLAG 1
870                 IF V$[1,1]='-' THEN M$=V$[2] ELSE M$='-'&V$
880             END IF
890             IF V$='0' THEN M$=U$ @ SFLAG 1
900             IF U$=V$ THEN M$='0' @ SFLAG 1
910         END IF
920         IF S AND S1<>1 AND NOT FLAG(1,0) THEN FNS$='('&M$&')' ELSE FNS$=M$
930     END IF
940 END DEF

950 DEF FNP$[192](U$,V$)
960     U=0
970     IF U$<>'0' AND V$<>'0' THEN
980         IF U$[1,1]='-' THEN U$=U$[2] @ U=1
990         IF V$[1,1]='-' THEN V$=V$[2] @ U=1 EXOR U
1000        FNP$='-'[1,U]&U$&'*'&V$
1010        IF U$='1' THEN FNP$='-'[1,U]&V$
1020        IF V$='1' THEN FNP$='-'[1,U]&U$
1030    ELSE
1040        FNP$='0'
1050    END IF
1060 END DEF
```

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.

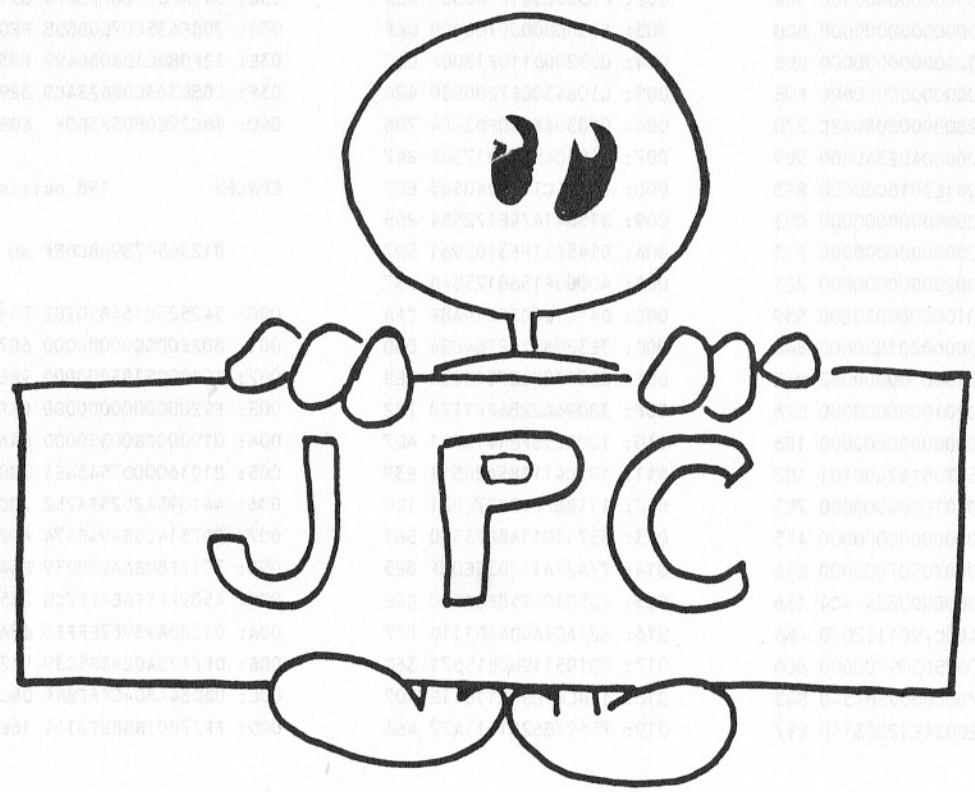
Vous trouverez donc le Lex CHARLEX nécessaire à la rédaction de votre article (voir "Ah ! Vous écrivez !"), ainsi que le Lex de programmation structurée de ce mois-ci.

```
CHARLEX

COMPLEX   COMPLEX   XWORD   94027        CONJ      XFN     94028
          IMPT      XFN     94029        MAG       XFN     94030
          REPT      XFN     94031

ERRLEX    EEND      XWORD   92001        ERROR     XWORD   92002
          WARNING   XWORD   92003

ACBATLEX  CLKSPEED  XFN     92099

HALEX     CHRTYP    XFN     93007        ISNUM     XFN     93008
          MNOP      XFN     93009
```

```
 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          624 octets

     0123456789ABCDEF sm

000: 34841425C4548502 35E
001: 802E000000000000 68D
002: 5E4001EFF0000000 9FD
003: FE00000000800001F D57
004: F31BF961400032BF 0EA
005: 38F14A11DB10AD23 484
006: 07D532BFB8FD7911 837
007: 11AD754D7A101743 BBA
008: 11014D1CB15D0000 F25
009: 71450375FF864834 2A2
00A: 5655581008355654 5F9
00B: 5810070507701724 93F
00C: 7700775070077517 C92
00D: 2077040708364545 FE0
00E: 4A30000A49724000 333
00F: 0808094A2C180814 69C
010: A464242008355455 9F6
011: 581000054C714000 D3C
012: 0C3142404C700832 098
013: 41414A70002078A0 3F0
014: 2F30000000000000 71B
015: 0000000000000000 A2B
016: 0000000000000000 D3B
017: 0000000000000000 04B
018: 0000000000000000 35B
019: 0000000000000000 66B
01A: 0000000000000000 97B
01B: 0000000000000000 C8B
01C: 0000000000000000 F9B
01D: 0000000000000000 2AB
01E: 0000000000000000 5BB
01F: 0000000000000000 8CB
020: 0000000000000000 BDB
021: 000000000000080C F06
022: 1A28080008080A2C 270
023: 180008040E340800 5B9
024: 08001E3018000000 8F3
025: 0000000000000000 C03
026: 0000000000000000 F13
027: 0000000000000000 223
028: 0201000000010200 539
029: 0000000201020000 84E
02A: 0001000100000002 B62
02B: 0102010000000000 E76
02C: 0000000000000000 186
02D: 045E755142400101 4D2
02E: 0101010000000000 7E5
02F: 0000000000000000 AF5
030: 0000070507000000 E18
031: 000000000383444C4 156
032: 44400D7901112D70 4B6
033: 050D750509700000 800
034: 0D70000000384540 B43
035: 4020014E322E3140 E97
036: 084E794142400000 1E7
037: 00000000002E4559 525
038: 3200000000000000 83A
039: 0000000000000026 B52
03A: 5556587008365556 EB1
03B: 5810083645464830 202
03C: 0832414248700024 543
03D: 5655587008345655 8A0
03E: 5810083446454830 BEF
03F: 0C3042414C700024 F44
040: 5556587008355654 2A1
041: 5810083546444830 5F0
042: 0C3142404C700025 946
043: 5455587008355455 CA0
044: 5810083544454830 FEE
045: 0C3140414C700875 350
046: 14141870000A4972 6A1
047: 40000E3159454E30 A01
048: 0C7A0F7949400024 D79
049: 5554587000084A71 0D5
04A: 40000C523A262D10 436
04B: 0424587458400875 78D
04C: 1415187000094A70 ADD
04D: 4000083544454830 E21
04E: 0C3140414C300C74 189
04F: 5655545000054C71 4E0
050: 40000           5D9

COMPLEX          501 octets

     0123456789ABCDEF sm

000: 34F4D405C4548502 37D
001: 802E000000000000 6AC
002: EE300E5B1F100000 A2D
003: FB30000000700000 D6F
004: 005200D110F1300F 0CC
005: C1064300F7200530 424
006: 0F0306F200FD34F4 7B6
007: D405C45485B1734F B47
008: 4E4A4C1794D40545 ED7
009: D15D41474E172554 256
00A: 0545F11FF3183961 5D7
00B: 40008F15681258FD 957
00C: D4101B0C8F214A8F CFA
00D: 3E320A71B2764024 06B
00E: 8E0138FD07813028 3EB
00F: 38096A2286FF1F70 782
010: 1000258F8A410001 AD7
011: 197C411185A20511 E39
012: 111B8FF53C07CB21 1DC
013: 55711011A8FF53C0 561
014: 77A27A112031E08F 8E5
015: 4058104258F8A410 C4E
016: 821AC1A4DA4D111D FF7
017: 80105119BCE15571 36D
018: 18BCE5EB0511011B 707
019: 7562786211111A77 A66
01A: 527A5211111B7942 DC9
01B: 7C4211011A7B328D 148
01C: 459E07FBFBCC7222 509
01D: 7712155772EF8F79 89E
01E: 6C077DF7702645F7 C3B
01F: 59F7BF18F9D3D011 FF3
020: 3AF671F1AF912210 388
021: BAF671E18FC14D07 746
022: FC18F7E3D079D18F B14
023: CA4C072B115577D7 EAF
024: FBCE8F796C07F6F7 296
025: F9170B18FCA4C061 640
026: EEAFA8F064A11CF1 A17
027: 371358BB80151703 D7F
028: 8DD4490B4000D400 0FE
029: 030E1F098F215508 479
02A: FFADA08F05CA0109 83C
02B: 8AC70E48508FD9DA C05
02C: 08F93EA046D8D84A FBC
02D: 808D782508DF8230 345
02E: 111130156716F717 69B
02F: F15676E4E7C20171 A2D
030: 157717F154718F15 DA5
031: 771547DB135693E7 131
032: 600AF240E137D713 4B3
033: 51B088F2156418E1 832
034: 4613416FB46B4640 BAD
035: 08D6CC7181175808 F33
036: DC32F0811777051F 2C0
037: 118BCE78EE31E08F 684
038: 4058159D8FC8CB04 A2A
039: 1C61BD8117740AF2 DC0
03A: 5C31184638117430 11A
03B: AF6592057D308F7E 4C4
03C: 3D0AF67F20793070 851
03D: 208F435C07E008D8 BE0
03E: 32F08DC1DB08D499 F93
03F: C08D363C08D234C0 329
040: 8DC29E08D5F3D0F  6BB

ERRLEX           198 octets

     0123456789ABCDEF sm

000: 542525C454850202 358
001: 802E000000000000 687
002: 19100C5103000000 9BE
003: F920000000000000 CEF
004: 019000CB0035000D 04A
005: 81016000D75454E4 3B0
006: 44109542525F4252 70C
007: 0D751425E494E474 A92
008: 301FF8DE6A208D39 E44
009: 4509FFFFDEFFF7CB 255
00A: 0228DA93902EFFF6 60A
00B: DFFF75A02A8F5C39 9E2
00C: 08D84A804CFFF8BF DBC
00D: FF77801BBB8F2144 16E
```

```
00E: 8FFD0908F81EF043 528
00F: 28F4CEF039025454 88A
010: E44415C91698FCBE C75
011: F08FA87914428FFF 037
012: 980100119A064229 38C
013: 0EA18FB51905318F 729
014: 92D708F4E02184D1 ABA
015: BBB8F2146694F8F6 E78
016: 81F08FDF8E0AC08F 24E
017: 322B18FFACE0AF2A 61F
018: BEBF4BF4BF4AF78F A26
019: 2D2B1AFEAFFAFE8F E34
01A: 2D2B1AFBF2F2AE60 204
01B: 1                236

ACBATLEX        136 octets

     0123456789ABCDEF sm

000: 1434241445C45485 35D
001: 802E000000000000 68C
002: 41100C5363600000 9CC
003: F710000000000000 CFA
004: 0E1000FF34C4B435 088
005: 05545444361FF007 3F4
006: 1800420D1114D010 743
007: 41B8F2E2808F14A1 ADE
008: 4E902AF14E21A0EA E8D
009: 0EB3514A9667FD0B 23A
00A: 3514A966AE808011 5B5
00B: 48ACFA20C5AF0D4F 98B
00C: 0C5C0D23131EAF08 D28
00D: FB13B1AF67E208D6 0E4
00E: 12F0132BF0BF0BF0 48B
00F: BF0BF0133BF0BF0B 854
010: F0BF0BF0104114BF BFD
011: 4BF4BF4BF4BF4133 FC8
012: BF4BF4BF4BF4BF41 3B5
013: 3203F            4C8

HALEX           433 octets

     0123456789ABCDEF sm

000: 8414C45485020202 354
001: 802E000000000000 683
002: 66300D5709000000 9CB
003: F9200E400B800000 D2F
004: 00C000FF00CD100F 0BC
005: C1040100FB348425 427
006: 4595057099435E45 796
007: 5D4807D4E4F40590 B2B
008: 1FF10200120435C4 E91
009: A36513CB210A0516 209
00A: 2756E64786563796 585
00B: 37602F6D69647564 906
00C: 6CFF9695052311B1 C97
00D: 351121CD1378B6E1 01C
00E: 13510B3D230314A3 380
00F: 3534022D15510041 6C8
010: 173308AC70285C01 A3C
011: C114B7822D280C0D DCF
012: A8FB13B1129135AF 173
013: 68D612F08F83DB01 514
014: 37C2109135AF2D68 8A1
015: 1E014117DDF10A10 C26
016: 88AC6067801C1D7D FC9
017: 12F306A8A853CF20 360
018: 14B318296670E550 6D0
019: 4B6696670CD5338A A69
01A: DE27A9180CF9C212 E18
01B: 87341AC5A4DA4D4B 1D0
01C: 3A4D453ACADB1081 57D
01D: C1843CF56A8A9112 91B
01E: 03310D58DA939011 C8F
01F: 8112EA6A2F231711 00F
020: 5311C12031549668 369
021: B814814762189A1B 6EA
022: 56A4117E0F8AC606 A86
023: C901C1D720CF3182 E1B
024: D114B96601E5CF72 1B9
025: E04975FE7D904F63 569
026: 1E29669078C04F58 8F8
027: F670B145574A0545 C76
028: 31E29669077A0444 FEA
029: 8F670B1490738053 35F
02A: 3315496642768O4D 6C7
02B: 114B71404318F670 A2E
02C: B14907850580206A D98
02D: 4ED9D7203192CF44 13B
02E: 114B9665E7740450 4AD
02F: 53F21632E31D2966 82D
030: C07E204005FE31B2 BBA
031: 962400379104005A F0D
032: D8F670B140075005 284
033: 1F031C114BCF4000 607
034: 3203D82B2D2A2F2E 9A7
035: 592962720CBF6BF6 D50
036: 8870F8F07F305E02 0E7
037: A871902C5402F886 46D
038: 002001F          5DD
```



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 56, rue Jean-Jacques Rousseau, 75001 Paris.

La maquette de ce numéro a été préparée et réalisée par Janick Taillandier grâce à un système comprenant un HP71B, un lecteur de disquettes HP9114A, un HP9807A, deux HP9154 et une imprimante LaserJet.

Les dessins sont de Jean-Jacques Dhénin et Paul Courbis.

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

Veuillez adresser toute correspondance à :
PPC Paris, BP 604, 75028 Paris Cedex 01.

Imprimé par Copy-Express, 42 86 91 94.

ENGLISH SUMMARY

JPC 55 - JUNE 1988

We hope you enjoy the drawings in *JPC*. Paul Courbis's « little worms » have joined Jean-Jacques Dhénin's « nibs ». They all live together well !

We are happy to announce you that three discs containing all the material published in *JPC* for the HP-71 and HP-75 are now available from the Club. Just send us 75 FRF, and three 3"1/2 discs and we will send them back with a catalog.

In the HP-28C column, you will find on page 4 an article by Paul Courbis, describing a method for creating HP-28S assembly language programs. Then, on page 5, Sébastien Lalande gives two programs for the HP-28C in assembly language that provide the same capabilities as →LCD and LCD→ initially implemented on the HP-28S. The last article, by the same author on page 8, allows you to walk easily through the HP-28S directory structure.

The HP-75 article on page 10 is a new Lex by Jean-Yves Hervé. It allows you to list the variables used in a program.

The HP-71 columns is very important this month. It starts in Forth with Xavier Bille describing the way to use strings in this language.

An Italian friend, Gigi Filippini, has sent us a Lex file he wrote to provide complex number processing capabilities on the HP-71 without a Math Rom. It is worth studying, it is the best introduction we know about complex in assembly. Then, on page 18, Serge Vaudenay's Lex is very useful : it produces Basic errors or warnings that provides your programs with a better system interface. On page 19, Jean-Jacques Dhénin, is interested by the problem of a low battery detection on the HP-71, and provides a function returning the clock speed, which is tied to the power supply.

On page 22, Olivier Arbey and Laurent Istria publish a new symbolic differenciator. It is completely rewritten, using heavily the STRUC2 keywords, see the result of listed by PBLIST on page 32...

Until next month,

Happy Programming and JPC reading !