JPC

OCTOBRE 1987
NUMERO 48

Le numéro 40 F

A PROPOS DU CLUB

P. David	Editorial	*1*
	PPC Paris se réunit	*2*
	Ah ! Vous écrivez !	*2*
	Courrier des lecteurs	*3*
	Courrier du coeur	*3*

HP28

M. Maupoux	Tracé de fonctions	*6*

HP41

M. Markov	Utilitaires de gestion de disques (acte I)	*8*

HP75

E. Gengoux	L'article est-il au catalogue ?	*14*

HP71

J. Baudier	L'assembleur du HP-71 (acte IV)	*16*
J. Elhay	Deux fonctions pour comparer	*22*
G. Toublanc	Choc en retour	*23*
O. Arbey	La guerre des noyaux (acte I)	*27*
	Le coin des Lhex	*34*

EDITORIAL

Chers amis,

L'Assemblée Générale annuelle du Club aura certainement lieu en Janvier 1988. Vous serez informé plus précisément sur le lieu et la date dans le prochain numéro, mais tenez-vous prêts. Cette assemblée sera certainement très intéressante, beaucoup de points importants seront discutés et décidés. Alors, ne la manquez pas !

Je suis très content de recevoir vos articles. L'apparition de nouvelles signatures dans le Journal est très positif, et montre bien l'activité des membres du Club. Souhaitons que les rubriques HP-41 et HP-75 suivent le mouvement...

En attendant de vous lire, bonne lecture !

Pierre David (37)

PPC PARIS SE REUNIT UNE FOIS PAR MOIS

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

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

Si donc cela vous intéresse, n'hésitez plus un seul instant, venez nous rejoindre tous les premiers samedis de chaque mois (sauf en période de vacances scolaires) au :
Centre de Jeunesse et de Loisirs Jean Verdier
11 rue de Lancry
75010 Paris
et en montant au deuxième étage, vous entendrez des éclats de rire et des discussions passionnées vers la salle 215. Attention, toutefois, de venir entre 16 et 19h.

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

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

Les dates des prochaines réunions sont :
Samedi 3 octobre 1987
Samedi 21 novembre 1987
Samedi 5 décembre 1987
Samedi 16 janvier 1988
Samedi 20 février 1988
Samedi 5 mars 1988
Samedi 16 avril 1988
Samedi 7 mai 1988
Samedi 4 juin 1988

Pierre David (37)

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 en fin de journal.

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)
```

COURRIER DES LECTEURS

Cher Trésorier,

Bien sûr que je ne veux pas rater un numéro de JPC ! Pour être franc, la plupart des articles sont bien au dessus de mon niveau (élémentaire) et pour l'instant, je me borne à développer une série de programmes de traitement de données, mais je vais progressivement me mettre à l'assembleur. Et là, tous les articles de JPC me seront alors très précieux. Longue vie au PPC Paris Chapter !

Amicalement,

François Dozol (318)

P.S. : le fait que je n'aie assisté qu'à une réunion (pour aller chercher le module JPCLEX) n'est pas une marque de désintérêt, mais résulte du fait que pour participer à ces réunions, il faut avoir un problème de même niveau que ceux qui passionnent les autres membres et avoir des choses à donner en échange, et j'en suis encore loin.

Réponse du Trésorier

Détrompez-vous ! Le Club a besoin de toutes les expériences. Et surtout de la vôtre ! Beaucoup de nouveaux membres se sentent un peu perdus à la lecture de JPC. Et vous qui êtes un « ancien » de PPC Paris, vous pouvez leur apporter votre expérience, vos idées et vos soucis. C'est cela la vocation de PPC Paris.

Si les mêmes personnes écrivent toujours des articles parfois incompréhensibles pour la majorité des membres, il ne va plus rester qu'un petit noyau au Club. Et je vous laisse imaginer la suite...

Par contre, faire un article, ce n'est pas grand-chose. Vous commencez par commenter un programme, ce qui est d'habitude le plus simple, et vous faites ensuite un mode d'emploi, en vous mettant à la place d'un autre et en vous disant : « si je lis JPC, et si je vois ce programme, saurais-je l'utiliser ? ». Vous mettez votre disquette, cassette ou carte magnétique dans une enveloppe, et vous nous l'envoyez. Et comme ça, vous n'aurez pas besoin d'imaginer ce qui arrivera au Club si...

Quant aux réunions, elles ne sont pas réservées à une « élite ». Beaucoup de nouveaux (ou futurs) membres y viennent, et trouvent une réponse à leurs questions, et apportent la vie nécessaire au Club.

Encore une fois : votre expérience a autant de valeur que celle des autres. Vos problèmes sont partagés par beaucoup de membres. Et il y en a beaucoup qui seraient heureux de vous lire dans JPC. A commencer par moi.

Et merci pour votre très gentille lettre.

Amicalement,

Janick Taillandier (246)

COURRIER DU COEUR

PPC-Paris
B.P. 604
75028 Paris Cedex 01

Vend :
Lecteurs de cartes magnétiques HP82400A neufs pour HP-71 (dans leur boîte, avec documentation et 5 cartes magnétiques) : 500 F seulement.

1 lot de cartes magnétiques pour HP-41, HP-67, HP-97 et même HP-65 : 100 F seulement.

Pierre David
33 Bd St Martin
75003 Paris
Tél : (1) 48 87 68 93

Vend :
Imprimante 80 colonnes HP82905B + papier + étiquettes : 2000 F.

Convertisseur HP82166A : 1000 F.

F. Lefebvre
Société Industrielle de Menuiseries
55 Haironville
55000 Bar Le Duc
Tél : 29 70 21 42

Vend :
HP-71 : 2500 F + Module éditeur de texte : 250 F.
Matériel acheté en 1986, livré en 1986.

Didier Gomiero
92 allée Modigliani
77190 Dammarie Les Lys

Vend :
HP-71 + Lecteur de cartes + 40 cartes + une pile de JPC + beaucoup de programme + 3 jeux de piles (batteries) : 3500 F.

Jean-Pierre Bondu
Domaine de Croix Marie
78121 Crespieres

Vend :
HP-71 + HP-IL : 3000 F, Module Math : 500 F, Lecteur de disquettes HP9114A : 3000 F.

HP28

M. Maupoux — Tracé de fonctions — *6*

TRACE DE FONCTIONS

Le but de cet article est de répondre à deux critiques fréquemment addressées au tracé de courbes sur HP-28C :
- comment empêcher qu'un programme ne s'arrête quand survient une erreur `Undefined Result`,
- comment accéder aux fonctions de contrôle du curseur : en particulier la digitalisation, après un tracé effectué par programme, tel qu'un tracé en coordonnées polaires.

Les deux cas seront illustrés par des exemples.

Le premier point est partiellement traité dans le *Manuel de Référence*, sous `IFTE` dans le menu `BRANCH`. L'expression `'IFTE(X<>0,SIN(X)/X,1)'` stockée dans `EQ` permet de tracer $\sin x / x$ sans erreur `Undefined Result` quand $x = 0$.

Le problème est plus complexe si vous ne savez pas d'avance pour quelles valeurs de x une erreur risque de se produire, ou même quelle erreur parmi plusieurs possibles. Certaines peuvent être ignorées en levant les flags appropriés, mais ceci n'est pas valable dans tous les cas.

Une solution est d'inclure la fonction algébrique dans une clause `IFERR` faisant partie d'un programme (ceci n'est pas possible dans une équation algébrique : c'est une commande et non une fonction).

Si un programme est écrit en utilisant des variables locales et la commande `PIXEL`, un autre problème voit le jour : c'est notre deuxième sujet.

Si vous voulez pouvoir déplacer le curseur après avoir tracé une courbe et digitaliser certains points, vous devez absolument utiliser `DRAW` au clavier depuis le menu `PLOT`. Quand `DRAW` est utilisé dans un programme, il n'a pas les mêmes fonctionalités, c'est pourquoi vous devez utiliser `CLLCD` ou `DRAX`.

Pour utiliser `DRAW` depuis le clavier, vous devez définir votre courbe dans EQ.

Pour conserver les possibilités de la fonction `DRAW` tout en utilisant un programme, vous devez garder à l'esprit un paragraphe du *Manuel de Référence* (section PLOT) : «si EQ contient un programme. [... il] obéit à la syntaxe d'une expression algébrique : il ne peut prendre aucun argument dans la pile et doit renvoyer un et un seul objet dans la pile».

Pour terminer, voici quelques exemples.

Interception d'erreur

Stockez dans EQ le programme suivant :

```
«
  IFERR
    F EVAL       F contient la foction à tracer
  THEN
    DROP2        enlève les arguments en erreur
    0            valeur par défaut
  END
»
```

Essayez avec F contenant `'SIN(X)/X'` et les valeurs par défaut de `PPAR` en mode radians.

Fonctions paramétriques

`EQ contient :`

```
«
  U EVAL         calcule la coordonnée horizontale
  V EVAL         calcule la coordonnée verticale
  R→C           combine les deux pour PIXEL
  PIXEL          trace le point
  A              retourne un objet sur la pile
»
```

Si A contient une constante, il y aura tracé d'une ligne horizontale. Essayez, par exemple, avec :

```
U: 'COS(X)+IP(2*X/3.14159)'
V: 'SIN(X)'
```

Courbes polaires et cartésiennes

Dans l'exemple précédent remplacez le `A` final par `F EVAL`.

Dans les trois cas, vous pouvez ajouter `36 CF` en début de programme et supprimer tous les `EVAL`. Vous pouvez utiliser `DRAW` et digitaliser ce que vous voulez ensuite. Bien entendu, vous pouvez mixer l'interception d'erreurs et le tracé de courbes paramétriques.

Michel Maupoux

HP41

M. Markov — Utilitaires de gestion de disques (acte I) *8*

UTILITAIRES DE GESTION DE DISQUES (ACTE I)

Le module HP-IL pour HP-41 a été programmé en pensant au lecteur de cassettes HP82161. A cette époque, l'implémentation du standard HP-IL / LIF était encore en cours de développement. Aussi, il n'est pas surprenant que certaines fonctions de gestion d'unités de stockage de masse (WRTP, CREATE, WRTS, WRTA, WRTK, WRTPV et NEWM) ne fonctionnent pas toujours avec des appareils plus récents.

En particulier, les commandes qui créent de nouvelles entrées dans le catalogue supposent que le support magnétique contient au plus 1FF secteurs (en hexadécimal). En pratique, ceci signifie que vous ne pouvez ajouter un fichier commencant au secteur 200, ou au dela, à moins que le catalogue ne contienne un fichier purgé de la taille adéquat.

NEWM permet de créer des catalogues ne pouvant contenir que 7 à 447 fichiers. Ce dernier chiffre vous donne le nombre maximum de fichiers de 1 secteur de long que vous pouvez stocker sur la cassette. Dans ces conditions, un disque 3,5 pouces pourrait contenir environ 2188 fichiers ! Bien entendu, des considérations telles que le temps d'accès aux fichiers, l'usure du support et le confort de l'utilisateur conduisent généralement à limiter la taille de catalogue des supports utilisés avec un HP-41 bien en dessous de la limite des 447 fichiers. Cependant il peut être utile pour certaines applications de disposer de très gros catalogues. Les techniques permettant de les utiliser seront décrites dans la deuxième partie de cet article.

Les utilitaires présentés ici utilisent le module *Extended IO* pour résoudre ces problèmes. La fonction XTOA (du module *Extended Functions*) peut être remplacée par son équivalent XTOAR. Mis à part ce module, vous n'avez besoin que d'un module HP-IL, que vous utiliseriez de toute manière pour accéder aux unités de stockage de masse.

Le premier utilitaire, NEWMED, peut être utilisé pour modifier la taille du catalogue d'un support qui vient juste d'être formatté. Le second programme améliore les fonctions qui ajoutent un fichier au support de masse, de telle manière que vous puissiez accéder à toute la capacité de stockage d'un support.

MODIFICATION DE TAILLE DU CATALOGUE

NEWMED est un programme indépendant parce que vous n'en avez besoin que lors de l'initialisation du support. Vous pouvez également utiliser ce programme à tout moment mais ceci risque de faire disparaître certains de vos fichiers. Je n'ai pas poursuivi d'études dans ce domaine : vous vous y aventurerez à vos risques et périls ! NEWMED ne doit être utilisé qu'après l'initialisation du support, sinon l'adresse du premier fichier ne correspondra pas à la taille du catalogue. Celui-ci risque alors de venir se superposer à vos premiers fichiers. Respectivement, si vous essayez d'écrire dans ces fichiers, vous risquez de corrompre le catalogue, si bien que DIR ne fonctionnera plus correctement. Diminuer la taille du catalogue est moins dangereux mais ne perdez pas de vue que vous ne pourrez probablement pas utiliser l'espace libéré par le catalogue.

NEWMED prend un nombre de fichiers dans le registre X, calcule le nombre de secteurs necéssaires pour le catalogue et modifie l'octet donnant la taille du catalogue dans le secteur 0 du support.

NEWMED utilise CLRDEV pour se positionner sur le secteur 0. Ainsi, les lignes 15 à 22 sont a peu près équivalentes à 19 XEQ 95 dans le second programme, c'est à dire aller à l'octet 00013_{16}. Le mode écriture partielle est activé par 6 DEVL, la nouvelle taille est enregistrée sur le support aux lignes 23 à 26. CLRDEV remet le lecteur dans un état connu, ce qui est toujours une bonne idée avec ces appareils.

LES PROGRAMMES D'ECRITURE

Ces programmes ont été conçus pour être aussi transparents que possible pour l'utilisateur. Les ordres en assembleur sont utilisés, si possible, pour minimiser autant que faire se peut le temps d'exécution et l'utilisation de la mémoire. Ces programmes s'utilisent exactement comme leur équivalent système.

Vous pourrez noter que je n'ai pas cherché à fournir une nouvelle commande WRTPV. Il y a à ceci plusieurs raisons : d'abord, les fichiers privés compliquent la vie lorsque vous voulez faire une copie de votre support. Ensuite, écrire cette nouvelle version de WRTPV demande 25 octets supplémentaires : ce programme est déjà bien assez long sans cela ! (382 octets).

C'est au moment de franchir la barre fatidique des 512 secteurs que vous aurez besoin de ce programme. Quand vous verrez apparaître MEDM FULL, il sera temps de le charger. En tout il utilise 20 % de la mémoire de la machine. Il peut y avoir des situations où ceci n'est

pas acceptable. Si tel est votre cas, vous pouvez créer des fichiers de données d'une taille convenable pour un usage ultérieur. Ces fichiers seront purgés quand vous voudrez inscrire un nouveau fichier sur le support. Une fois une entrée d'une taille convenable purgée du catalogue, les fonctions de base de la machine pourront effectuer normalement leur travail. Ce programme est, en fait, uniquement indispensable pour écrire de nouvelles entrées.

Ces fonctions ont été écrites pour minimiser l'espace reservé à un nouveau fichier. Ceci est très important car les fonctions système ne modifient pas cette valeur. Il est parfaitement plausible de stocker un fichier de 10 octets à la place d'un fichier de données de 20000 registres. L'espace perdu ne peut plus être récupéré à moins de purger le fichier. Ensuite, les ordres PACK du HP-71 ou du HP-75 constituent la meilleure solution à votre problème.

Un peu de théorie

La création d'une entrée de fichier valide est un processus relativement complexe. D'abord, vous devez parcourir le catalogue pour trouver soit une entrée correspondant à un fichier purgé de taille convenable, soit la première entrée non utilisée du catalogue. Une fois que ceci est fait, vous devez vous assurer qu'il ne s'agit pas de la toute dernière entrée du catalogue qui est réservée au système. Il faut ensuite calculer l'adresse de début du nouveau fichier. C'est, soit la taille du catalogue + 2 pour le premier fichier, soit la somme de l'adresse de début du dernier fichier et de sa longueur en secteurs. Il faut vérifier également qu'il y a la place sur le disque pour stocker le fichier. Ce dernier point est important en raison du problème de bouclage en mode lecture ou écriture continue qui pourait détruire votre catalogue. Après tout ceci, vous êtes prêt à écrire le nouveau fichier sur le support ou à sortir avec un message d'erreur approprié.

J'ai pu résoudre ces problèmes tout en économisant beaucoup de mémoire en étudiant soigneusement les conditions qui provoquent le message MEDM FULL. Les fonctions assembleur ont déjà vérifié l'existence d'un fichier purgé convenable et les pointeurs de l'unité de stockage sont laissés à l'adresse de la première entrée disponible plus 288 octets. Ainsi, il est possible d'éviter une longue recherche entrée par entrée. Dès que cette information est recupérée, il est facile d'obtenir l'information permettant de créer une entrée purgée fictive.

La longueur en secteurs est fixée pour les fichiers de statut (1 secteur) et les fichiers "write-all" (11 secteurs). La quasi-totalité des fichiers d'assignation de touches font 1 secteur. Mais, s'il y a plus de 65 définitions, 2 secteurs sont nécessaires. Vous pouvez soit traiter ce cas soit, comme je l'ai fait, supposer que ces fichiers ne feront pas plus de 1 secteur afin d'économiser la mémoire.

Cette solution simpliste ne peut être acceptée dans le cas des programmes dont la longueur peut varier de 1 à 9 secteurs. Vous devez donc dimensionner le fichier de manière convenable. Il serait idéal de pouvoir obtenir les données avec une fonction assembleur telle que RCLPTA. Ceci suppose que le programme soit en mémoire étendue, mais celle-ci peut ne pas être disponible. Une autre solution serait d'utiliser la fonction PPLNG du module CCD ou un équivalent. J'ai choisi une solution qui ne recourt qu'au module *Extended IO* : on commence par supposer que le programme fait 9 secteurs, on l'écrit sur le support, on obtient de l'entrée correspondante du catalogue la longueur du fichier en octets puis on met à jour la longueur en secteurs en fonction de cette valeur.
Les fonctions PPLNG et XTOAH du *CCD ROM* peuvent faire gagner au moins 30 octets.

Mode d'emploi

Les labels WRTP, WRTK, WRTS et WRTA doivent être présents en mémoire principale pour inhiber l'action des fonctions homonymes du module HP-IL. Vous pouvez utiliser les nouvelles fonctions exactement comme leurs équivalents standards. En fait vous n'utiliserez les nouvelles fonctions que si le système détecte une erreur. Un certain nombre de sécurités existent :
1 - Vous obtiendrez ADR ERR s'il n'y a pas de lecteur.
2 - Le message NO RESPONSE est émis si l'appareil est un lecteur de cassettes HP-82161.
3 - Le secteur du catalogue devant être utilisé est vérifié pour s'assurer que cela est possible (entrée purgée ou non utilisée).

J'ai fait très attention à rendre ce programme aussi insensible que possible aux problèmes. Vous devez pouvoir l'interrompre ou l'utiliser en mode trace (en mode AUTOIO) sans inconvénients. Cependant, si vous modifiez la pile ou le contenu des registres 0 à 7, je vous recommande d'exécuter CLRLOOP et de recommencer le programme.

Utilisation des flags

Les flags 25 et 34 sont effacés (flag d'erreur et indication du mode ADRON. Les autres flags sont inchangés.

Registres d'état

Le registre ALPHA est utilisé puis restauré. La pile est utilisée.

Utilisation des registres

00 : Longueur du fichier en secteurs
01 à 03 : Noms de fichiers du registre ALPHA
04 : Adresse de la nouvelle entrée
05 : Nombre maximum de secteurs sur le support
06 : Premier secteur pour le nouveau fichier
07 : Premier secteur pour le premier fichier

Lignes synthétiques

```
116 : 255, 68, 32, 32, 32, 32, 32, 32, 32, 32, 32,
      32, 0, 0, 0, 0.
121 : 243, 127, 0, 0.
163 : 242, 68, 0.
180 : 242, 68, 0.
```

Description détaillée

Les lignes 1 à 4, 27 à 31, 45 à 49, 54 à 58 et 64 à 67 répondent au même but : donner la possibilité aux commandes en assembleur de réaliser leur fonction. Si tout se passe bien, le programme se termine normalement sans modifier la pile ou les registres de données.

Les 4 lignes suivantes fournissent la taille par défaut du fichier, appellent le sous-programme qui crée une entrée fictive dans le catalogue de la bonne longueur et essaie à nouveau d'appeler la commande standard. A ce moment vous pouvez encore obtenir DIR FULL ou MEDM FULL. Si c'est le cas, le message n'est alors que trop vrai.

Le travail est terminé à moins qu'il ne s'agisse d'un programme. Les lignes 9 à 14 récupèrent la taille du programme à partir de l'entrée dans le catalogue : les registres X et Y contiennent cette taille dans un format "secteurs et octets". La plupart du temps, il suffit simplement de récupérer le premier octet et de l'incrémenter de 1. Le calcul exact (lignes 15 à 19) prend seulement 4 octets supplémentaires. Cette information est ensuite utilisée pour modifier la longueur en secteurs dans le catalogue (lignes 19 à 26).

La création de l'entrée fictive débute à la ligne 72. La longueur proposée pour le fichier est stockée dans le registre 0 pour utilisation ultérieure et le disque est selectionné comme unité primaire. Le nom de fichier est copié du registre ALPHA vers les registres 1 à 3. Une dernière vérification est faite pour s'assurer que l'appareil selectionné n'est pas une unité de cassettes : si tel est le cas vous obtenez le message NO RESPONSE lors de l'exécution de la fonction ID en ligne 82.

Il nous faut maintenant obtenir les valeurs de pointeurs du lecteur (lignes 83 à 88) et calculer le nouveau secteur de départ (lignes 89 à 95). Les lignes 96 à 99 obtiennent le plus grand numéro de secteur du support. Le DDT 7 devrait être implémenté par tous les appareils de stockage de masse qui ne sont pas équivalents au lecteur de cassettes. Ceci signifie que ce programme devrait fonctionner avec tous les appareils de stockage de masse.

Les lignes 100 à 105 lisent la taille du catalogue et calculent le secteur de départ du premier fichier ; celui-ci est stocké dans le registre 7 pour une utilisation ultérieure éventuelle.

Les lignes 106 à 113 déterminent le type du fichier correspondant à l'entrée que l'on veut reécrire. Si ce type n'est pas 0000_{16} ou $FFFF_{16}$ le fichier n'est ni purgé ni inutilisé : il est temps de sortir avant de faire une bétise.

Finalement, les lignes 114 à 142 construisent la fausse entrée de fichier purgé dans le registre ALPHA et l'envoie sur le support. Ceci se fait en positionnant d'abord le lecteur au début de l'entrée et en passant en mode écriture partielle (RCL 04 XEQ 95). Ensuite un nom de fichier et le type "fichier purgé" sont stockés dans le registre ALPHA à la ligne 116. Cette ligne synthétique est faite d'un D, de 10 espaces et de 4 octets nuls.

Les lignes 117 à 120 déterminent le bon secteur de début et et l'ajoutent à l'entrée toujours en construction dans le registre ALPHA. On calcule ensuite (lignes 121 à 130) la longueur du fichier en s'assurant qu'il y a assez de place sur le disque. Si la longueur proposée est trop importante, on gardera une longueur correspondant à la place encore disponible. Ce choix est guidé par les fichiers programmes qui sont généralement beaucoup moins gros que les 9 secteurs qui constituent la valeur d'essai.

C'est tout ce dont vous avez besoin pour une entrée fictive. Vous pouvez stocker l'entrée sur le disque (lignes 131 à 136), remettre à zéro l'appareil et restaurer le contenu du registre ALPHA pour la seconde tentative d'utilisation des fonctions standards.

Les sous-programmes

Les sous-programmes constituent le coeur de ce programme. Ils forment ensemble un éditeur de support de masse puissant. Soyez prudent si vous les utilisez : en fonction du contrôleur qui devra accéder aux fichiers, vous risquez de devoir reprendre certaines sommes de contrôle pour éviter de sérieux problèmes (les fichiers du HP-75 sont particulièrement sensibles à ce problème).

- Sous-programmes de conversion :
Le LBL 91 convertit une adresse entière en deux octets qui sont ajoutés au contenu du registre ALPHA.
Le LBL 92 lit 4 octets depuis le lecteur et convertit les 2 derniers octets en un entier retourné dans X.
Le LBL 93 effectue la même tâche mais ne garde que les deux derniers octets des 23 envoyés.

Ces trois sous-programmes vous permettent, entre autres avantages, de gérer l'adresse disque comme un seul nombre et non comme trois éléments séparés (piste, secteur et pointeur d'octets). L'espace adresse du disque HP-IL est représenté par trois octets. cependant, les catalogues actuellement existant ne font qu'une utilisation partielle de la piste logique 0. Ceci nous permet de nous interesser seulement aux deux derniers octets lors de la modification des entrées de catalogue.

- Sous-programme de lecture de données depuis le support :
Le LBL 94 est le coeur du transfert des données depuis le support, en commencant à une adresse donnée dans le registre X. Comme le mode DDT 1 est utilisé, l'utilisateur doit gérer lui-même les frontières de secteur : après avoir lu le 255ème octet de l'enregistrement, vous continuez avec l'octet 0 du même enregistrement. L'avantage essentiel de cette méthode est que vous pouvez positionner le pointeur d'octets selon vos besoins.

- Sous-programmes de stockage de données sur le support :
Le LBL 95 est utilisé pour stocker des données sur disque à partir de l'adresse donnée dans le registre X.
Le LBL 98 est utilisé en liaison avec le LBL 95 pour transférer les données envoyées au buffer 0 de l'appareil sur le support. Ce sous-programme continue dans le LBL 00 qui remet à zéro l'appareil et restaure le registre ALPHA.

Remarque générales

Le programme *HP-41 to HP-9114 Utility* (User's Library numéro de catalogue 41-09114-7) m'a fourni le cadre général qui m'a permis de réaliser cet ensemble d'utilitaires. Mes remerciements aux auteurs.

Une étude attentive des sources assembleur du module HP-IL m'a fourni les indications qui m'ont permis de réduire la taille du programme (de 903 à 382 octets), de l'accélerer substantiellement tout en ajoutant de nombreuses sécurités.

Si vous souhaitez disposer d'un WRTPV vous pouvez ajouter au début du programme les instructions suivantes :

```
LBL "WRTPV"  SF 25  WRTPV  FS?C 25  RTN  9  XEQ 01
WRTPV  GTO 09

LBL "WRTP"  SF 25 WRTP  FS?C 25  RTN 9  XEQ 01
WRTP  LBL 09  RCL 04 ...
```

Ceci vous permet d'utiliser le code commun de la manière la plus efficace.

Michael Markov (301)

Le programme de modification de taille du catalogue :

```
01*LBL "NEWMED"
8  /  1  +  ENTER^  INT  X≠Y?  ISG X  ""  16
FINDAID  SELECT  ID  CLRDEV  3  DEVL  19  OUTXB
R^  6  DEVL  X<>Y  OUTXB  8  DEVL  CLRDEV  END
```

Les utilitaires d'écriture de fichiers :

```
01*LBL "WRTP"
SF 25  WRTP  FS?C 25  RTN  9  XEQ 01  WRTP  RCL 04
28  +  XEQ 94  INXB  INXB  X≠0?  ISG Y  ""  X<>Y
STO 00  RCL 04  19  +  XEQ 95  RCL 00  OUTXB
GTO 98

27*LBL "CREATE"
SF 25  CREATE  FS?C 25  RTN  32  /  ENTER^  INT
X≠Y?  ISG X  ""  XEQ 01  RCL 00  32  *  CREATE
RTN

45*LBL "WRTS"
SF 25  WRTS  FS?C 25  RTN  1  XEQ 01  WRTS  RTN

54*LBL "WRTK"
SF 25  WRTK  FS?C 25  RTN  1  XEQ 01  WRTK  RTN

63*LBL "WRTA"
SF 25  WRTA  FS?C 25  RTN  11  XEQ 01  WRTA  RTN
```

```
72*LBL 01
STO 00  16  FINDAID  SELECT  ASTO 01  ASHF
ASTO 02  ASHF  ASTO 03  ID  3  DEVT  XEQ 93  288
-  STO 04  20  -  XEQ 94  XEQ 92  STO 06  XEQ 92
ST+ 06  7  DEVT  XEQ 92  STO 05  19  XEQ 94  INXB
2  +  STO 07  RCL 04  XEQ 94  12  XEQ 93  255  MOD
X≠0?  GTO 00  RCL 04  XEQ 95  "D          "
RCL 06  X=0?  RCL 07  XEQ 91  "⊢"  RCL 05
RCL 06  -  X<0?  0  RCL 00  X>Y?  X<>Y  XEQ 91  20
OUTAN

133*LBL 98
8  DEVL

136*LBL 00
CLRDEV  CLA   ARCL 01  ARCL 02  ARCL 03  RTN

143*LBL 91
STO Y  256  /  XTOA  X<> L  MOD  XTOA  RTN

152*LBL 92
4

154*LBL 93
INAN  ATOXR  ATOXR  256  *  +  RTN

162*LBL 94
"D"  XEQ 91  4  DEVL  SQRT  OUTAN  DEVT  5  DEVT
3  DEVL  ATOXR  OUTXB  SIGN  DEVT  RTN

179*LBL 95
"D"  XEQ 91  4  DEVL  SQRT  OUTAN  3  DEVL  ATOXR
OUTXB  6  DEVL  END
```

HP75

E. Gengoux — L'article est-il au catalogue ? — *14*

L'ARTICLE EST-IL AU CATALOGUE ?

Les heureux possesseurs de *VisiCALC* disposent d'une fonction géniale, INCAT(*fichier,type*), qui renseigne sur la présence ou l'absence d'un fichier et sur son type. Hélas, elle ne marche que sur des fichiers en mémoire et, pour avoir un équivalent pour disquette ou cassette, il faut utiliser un Lex (c'est MADLEX). Et, malheureusement, ça ne marche pas pour le Pod ou le PMS...

Reprenant une idée qui avait été développée par Richard Harvey pour le HP-71, voici un petit utilitaire Basic (151 octets), sous forme de SUB (donc Rom I/O obligatoire), que l'on pourra invoquer pour tester si un fichier HP-75 est ou non présent. Que ce soit dans un programme ou au clavier, faire par exemple :

```
CALL "INCAT"("TOTO:M2",T$)
```

Au retour, T$ contiendra soit la lettre caractérisant le type du fichier s'il existe (soit B, I, G, L, R, T, W ou ?), soit le chiffre 0 si le fichier n'est pas présent (ou que son spécificateur n'est pas le bon)...

Comment est-ce possible ? Simplement en demandant le CAT du fichier et en le capturant dans le buffer d'affichage, au moyen de la fonction BUF$ de la Rom I/O. Ce buffer fait 96 octets de longueur (soit 3 fois la fenêtre d'affichage du 75), et le type est en douzième position. Si le fichier n'est pas trouvé, CAT génère une erreur, exploitée par la ligne ON ERROR afin de ne pas arrêter le programme appelant (dans ce cas, T$='0'). Et c'est tout !

Ah, si ! Ne chargez *jamais* ce programme dans un PMS ou une Eprom : BUF$ et SUB font partie de ces instructions qui, exécutées à partir du PMS, créent des désastres. Voir à ce propos le cas de EXIT... En l'occurence, on obtient l'erreur Not enough memory et un plantage qui résiste à [SHIFT][CTL][CLR], et il faut alors retirer la batterie.

Avantage majeur de ce programme : il est beaucoup plus rapide que la méthode consistant à dérouler une boucle du style :

```
FOR I=1 TO INF
  C$=TCAT$(I,':M1')
  IF C$ ...
NEXT I
```

et, surtout, fait beaucoup moins souffrir la batterie de votre drive et les pistes où réside le catalogue...

```
10 SUB "INCAT"(F$,T$)
20 DIM B$[96],T$[1]
30 ON ERROR GOTO 60
40 CAT F$ @ B$=BUF$('L')
50 T$=B$[12,12] @ GOTO 70
60 OFF ERROR @ T$='0'
70 END
```

Eric Gengoux (108)

ASSEMBLEUR

J. Baudier	L'assembleur du HP-71 (acte IV)	*16*
J. Elhay	Deux fonctions pour comparer	*22*
G. Toublanc	Choc en retour	*23*

BASIC

O. Arbey	La guerre des noyaux (acte I)	*27*
O. Arbey	Programme "MARS"	*29*
X. Bille	Programme "BOOT" (voir JPC 47)	*31*

LE COIN DES LHEX

34

L'ASSEMBLEUR DU HP-71
(ACTE IV)

Comme je l'ai dit dans mon article du mois dernier, aujourd'hui nous allons parler des fichiers Lex. Dans un premier temps nous allons voir ce qu'est un fichier Lex, comment on le constitue, quel est son rôle. En deuxième partie, nous verrons comment il fonctionne au sein du HP-71. En troisième lieu, j'expliquerai quels sont les intérêts des Lex, quels sont les différents types qui peuvent exister. Enfin nous verrons en détail le chemin à suivre pour écrire un Lex et, au niveau pratique, comment apporter 5 nouvelles instructions au Basic du HP-71.

Qu'est ce qu'un Lex ?

Avant d'entrer plus en détail dans le sujet, voyons d'abord ce qu'est un Lex. Comme les fichiers Bin dont j'ai parlé dans les articles précédents, les Lex sont des programmes écrits en assembleur. Ainsi, comme ces derniers, ils permettent d'avoir en contrôle total des ressources du HP-71. Vous vous demandez alors ce qu'ils apportent de plus. Si vous lisez *JPC* régulièrement, vous avez sans doute remarqué l'abondance de fichiers Lex. Ceci a une raison.

Les fichiers Lex permettent d'apporter de nouvelles fonctions au Basic, ce qui fait leur attrait. Simplement en écrivant un Lex et en l'assemblant, ou plus simple encore, en le copiant d'une carte ou d'une disquette, on apporte une ou plusieurs instructions au Basic de base déjà bien riche. Par exemple, les modules additionnels de Rom comme les modules HP-IL, Math et le module JPC sont simplement de gros fichiers Lex qui contiennent plusieurs dizaines de nouvelles instructions. Lex veut dire en fait *Language EXtension* et on comprend maintenant pourquoi. En résumé, je dirais que les Lex sont des fichiers programmes qui permettent d'enrichir le vocabulaire du HP-71.

Pour que cela soit possible, il faut que les Lex soient pris en compte par le système d'exploitation du HP-71 et c'est ce qui se passe. De la même façon que les fichiers Bin, les Lex sont écrits en assembleur, mais le système d'exploitation les traite différemment.

Comment les Lex sont-ils pris en compte par le HP-71 ?

A chaque fois que l'on allume son HP-71, il se passe un tas de choses que l'utilisateur ne soupçonne pas. En particulier, une routine de configuration est exécutée. Le rôle de cette routine est de reconnaître l'environnement du HP-71, c'est à dire tous les modules de Ram ou de Rom qui sont connectés ainsi que tous les Lex présents en mémoire. Cette routine est très rapide, elle s'exécute entre le moment où l'on appuie sur [ON] et le moment où le curseur apparaît. Si l'environnemnt du HP-71 a beaucoup changé depuis le dernier allumage, alors la routine mettra plus de temps et ceci pourra être visible.

Pour aujourd'hui, je n'entrerai pas en détail sur le fonctionnement de cette routine de configuration. Ce qu'il faut savoir, c'est que chaque Lex contient avant le programme assembleur en lui-même, une partie décrivant ce qu'il contient (les nouvelles instructions qu'il apporte, leur nombre, etc...). J'appellerai cette partie le descripteur. A chaque configuration, ce descripteur est exploré et certaines informations sont rangées dans un buffer appelé *Lex Entry Buffer* ou bLEX. Ce buffer est très important. A chaque fois qu'une instruction est entrée au clavier ou exécutée à partir d'un programme Basic, le HP-71 consulte ce buffer pour savoir si cette instruction existe et où se trouve son emplacement en mémoire. Nous ne verrons pas en détail aujourd'hui la structure du bLEX. Ceci fera, j'espère, le sujet d'un prochain article.

Pourquoi écrire des Lex ?

Comme je l'ai dit dans un précédent article, les programmes écrits en assembleur permettent, d'une manière générale, de faire des choses difficiles à réaliser en Basic, par exemple. En effet, on a un contrôle total de l'environnement de la machine. On peut ainsi écrire pour les besoins d'un programme ou pour des raisons pratiques un petit Lex apportant une ou plusieurs instructions. On a ainsi l'avantage de la rapidité d'exécution mise au niveau du Basic. De plus, une nouvelle instruction peut rendre un programme beaucoup plus lisible, comme lors de l'utilisation de sous programmes, mais aussi plus court.

Structure d'un fichier Lex

Je sais que vous êtes pressés d'en savoir plus sur les Lex alors entrons tout de suite dans le vif du sujet. Comme pour l'écriture d'un Bin, il faut, lors de l'écriture d'un Lex, respecter une certaine structure dans le fichier source. Le fichier source suivant permet de voir à quoi ressemble un Lex d'une façon générale :

```
      LEX    'EXEMPLE'

      ID     #5C
      MSG    msglbl
      POLL   pollbl

      ENTRY  label1  <--
      CHAR   ?           |
       :                 |
      ENTRY  labeln      | <--
      CHAR   ?           |    |
                         |    |
      KEY    '?????' <--      |
      TOKEN  ?                |
       :                      |
      KEY    '?????'      <--
      TOKEN  ?

      ENDTXT

msglbl   :           <--
         :              |  Table de messages
         :           <--

pollbl   :           <--
         :              | Routine d'inteception
         :              |   de polls
         :           <--


      <infos>

label1   :           <--
         :              |
         :              |  Instruction 1
         :              |
      < fin >        <--

         :
         :
         :

      <infos>

labeln   :           <--
         :              |
         :              |  Instruction n
         :              |
      < fin >        <--

      END              fin du fichier source
```

Voyons maintenant en détail la signification de toutes ces directives destinées au programme d'assemblage.

LEX suivi d'un nom de fichier entre guillemets indique à l'assembleur de créer un fichier de type Lex.

Chaque instruction du Basic du HP-71 possède un code de deux octets qui permet de l'identifier. Le premier octet est appelé numéro d'id (à prononcer ayedi) pour identificateur et le deuxième token. En général, et comme cela était le cas pour la HP-41, le numéro d'id correspond à un numéro de module et le numéro de token à un numéro d'instruction dans le module. Par exemple la Rom système est composée de deux modules d'id 0 et 1, le module Forth / Assembleur a pour id 43 et l'HP-IL 255. On peut mettre ce que l'on veut comme numéro d'id et de token, pourvu qu'une fonction ou ordre déjà existant n'ait pas les mêmes id et token ce qui pourrait entraîner des problèmes. HP conseille d'utiliser les numéro d'id 5C à 5F en hexadécimal pour le développement de Lex. C'est ce que j'ai fait ici en indiquant id #5C en ligne 2.

La directive MSG msglbl indique que le Lex contient une table de définitions de messages d'erreurs placée à partir de l'étiquette *msglbl* et POLL pollbl indique qu'une routine d'interception de *poll* est placée dans notre Lex. Pour ceux qui se demandent ce qu'est un *poll*, je dirai que c'est une sorte d'interrogation qu'effectue le HP-71 en des occasions bien particulières aux Lex présents en mémoire. Ne vous inquiétez cependant pas, j'écrirai un article sur le fonctionnement des polls ainsi que sur les tables de messages.

Les quatre directives suivantes (ENTRY, CHAR, KEY, TOKEN) sont à répéter autant de fois qu'il y a d'instructions dans le Lex.

On place d'abord autant de paires ENTRY, CHAR que d'instructions. Puis le même nombre de paires KEY, TOKEN en faisant attention à respecter l'ordre.

La directive ENTRY indique le point d'entrée de l'instruction. Par exemple, ENTRY label1 indique que le code de la fonction se trouve à partir de l'étiquette *label1*.

CHAR suivi d'une valeur de 0 à F donne la valeur du *characterization nibble*. Ce quartet de caractérisation d'un point d'entrée indique son statut, à savoir quels sont les droits de l'instruction à laquelle il correspond. Mais nous allons voir ça plus loin avec un exemple.

KEY indique la suite de caractères qu'il faudra utiliser pour appeler notre fonction.

TOKEN suivi d'une valeur de 0 à FF indique que le numéro de token de notre instruction.

Enfin, ENDTXT indique la fin de la partie de description de notre fonction.

Toutes les informations situées avant ENDTXT servent à l'assembleur à fabriquer le descripteur du Lex dont j'ai parlé plus haut. Derrière ENDTXT sont placées les mnémoniques du programme qui réalise la ou les instructions.

label1 est le point d'entrée de la première instruction et *labeln* celui de la dernière. Comme pour des programmes écrits en Basic, on peut placer dans un Lex des sous-programmes qui peuvent être ainsi utilisés par plusieurs instructions.

Devant chaque point d'entrée d'une instruction, on se doit de placer ce que je que j'ai appelé *info*. En fait, il s'agit de quelques quartets contenant des informations concernant l'instruction que le HP-71 viendra lire. Je reviendrai plus loin sur ce point.

Pour ceux qui veulent en savoir plus sur la structure des Lex et en particulier sur le descripteur, je conseille la lecture du début du paragraphe 6.1 (Lex File Structure) des *IDS I* page 6-1. Vous pouvez égalament relire l'article de ce cher Pierre David dans le numéro 39 de *JPC* de novembre 1986 page 17.

Les fonctions

Il existe trois types d'instructions qu'un Lex peut apporter. On trouve en premier lieu les mots qui ne sont pas des instructions à proprement parler. Par exemple ON, OFF, TO, FLOW, VAR etc... sont des mots. Ensuite, on a les ordres appelés *statements* en anglais. Les ordres indiquent au HP-71 de faire quelque chose. Dans cette catégorie on trouve, par exemple, BEEP, COPY, DISP, PURGE etc... Les ordres sont faciles à distinguer : ils font quelque chose mais ne renvoient rien. Leur syntaxe varie beaucoup, elle peut être simple comme RETURN ou bien plus complexe comme CAT ou BEEP.

La troisième catégorie comprend les fonctions. Contrairement aux ordres, celles ci renvoient quelque chose : des valeurs numériques ou alphanumériques. La syntaxe des fonctions varie peu, une fonction peut ne pas avoir de paramètre et renvoyer un nombre comme DATE ou une chaîne comme DATE$. Elle peut aussi avoir un ou plusieurs paramètres comme MAX et MIN (qui renvoient des nombres) ou PEEK$ (qui renvoie une chaîne de caractères). Les paramètres eux aussi peuvent être de natures différentes : numérique ou alphanumérique. En règle générale, la syntaxe des fonctions est la suivante : les paramètres sont placés entre parenthèses et séparés par des virgules. Les fonctions qui renvoient de l'alphanumérique prennent un $ à la fin comme DATE$, PEEK$, DTH$, etc... De plus, sur le HP-71 les fonctions se comportent comme si il y avait un ordre DISP implicite devant. En effet, si l'on tape PI le HP-71 comprend DISP PI et renvoie 3.1415... Si l'on tape 10 PI lors de l'entrée d'un programme, lors de l'édition on aura bien 10 DISP PI.

Pour commencer, nous allons donc nous intéresser aux fonctions. En effet, celles ci sont beaucoup plus simples à écrire que les ordres car le HP-71 s'occupe de gérer la syntaxe ce qui n'est pas le cas avec les ordres. Nous verrons cela dans un prochain article.

Comme un exemple vaut mieux qu'un long discours, je vous propose le Lex suivant tiré du manuel de la Rom Forth / Assembleur page 54. Ce Lex contient la fonction ONE qui ne prend pas de paramètre et renvoie la valeur 1. Je vous propose donc de rentrer ce fichier source dans votre machine puis de lire les explications qui suivent avant d'assembler.

```
        LEX     'LEX1'

        ID      #5D
        MSG     0
        POLL    0

FNRTN1  EQU     #0F216

        ENTRY   FNCT
        CHAR    #F

        KEY     'ONE'
        TOKEN   1

        ENDTXT

        NIBHEX  00
FNCT    C=0     W
        P=      14

        LCHEX   1

        GOVLNG  FNRTN1

        END
```

Le source est déjà commenté dans le manuel mais nous allons pousser plus loin les explications.

La directive LEX 'LEX1',en début de fichier source, indique à l'assembleur de créer un fichier de type Lex ayant pour nom LEX1.

POLL 0 et MSG 0 indiquent l'absence de routine d'interception de poll et de table de messages dans ce Lex.

La ligne `FNRTN1 EQU #0F216` indique, comme nous l'avons vu dans un article passé, d'affecter la valeur hexadécimale 0F216 à l'identificateur FNRTN1. FNRTN1 est l'adresse dans la Rom système d'une routine. On dit que `FNRTN1` est un point d'entrée. Cette routine « FoNction ReTurN » se charge de l'affichage du nombre contenu dans l'accumulateur *C* et du retour de notre fonction.

La directive `ENTRY` indique le point d'entrée de notre fonction `ONE`.

`CHAR #F` donne la valeur F au *CHARacterization nibble*. Ce qui indique le statut de notre point d'entrée à savoir que notre instruction est une fonction.

`KEY` indique que la nouvelle fonction s'appelle `ONE`.

`TOKEN 1` indique que le numéro de token de notre instruction est 1.

Enfin `ENDTXT` indique la fin de la partie de description. Entrons maintenant dans notre fonction.

La directive `NIBHEX 00` indique à l'assembleur de générer deux quartets de valeur 0 qui seront placés juste avant le point d'entrée de notre fonction. Ce sont les fameux quartets d'information destinés au système d'exploitation. Dans le cas d'une fonction, le premier de ces deux quartets indique au HP-71 le nombre de paramètres minimum qu'accepte notre fonction et le deuxième le nombre maximum. Si ces deux quartets sont différents, c'est parce que la fonction accepte un ou plusieurs paramètres facultatifs. On s'aperçoit ainsi qu'une fonction peut en avoir au maximum 15 (c'est le nombre le plus grand que l'on peut faire tenir dans un quartet) ce qui suffit bien dans la plupart des cas ! Ici notre fonction ne prend donc aucun paramètre.

A chaque fois que l'on appelle la fonction `ONE`, que ce soit à partir du clavier ou dans un programme, le HP-71 vérifie que le nombre de paramètres que l'on donne correspond bien à ce qui est autorisé en regardant ces deux quartets. C'est ainsi que le HP-71 gère automatiquement le contrôle de syntaxe. S'il n'y a pas correspondance, le système renvoie le message `Invalid Expr.`

Notre fonction commence donc au label FNCT (pour FoNCTion). On commence d'abord par mettre à zéro le registre C en entier (W désigne le registre en entier). Deuxième instruction : on charge la valeur 14 dans le registre pointeur de 4 bits P. Troisième instruction : `LCHEX 1` (Load C HEXa) charge la valeur 1 dans C dans le quartet pointé par P. Le registre C contient alors : C = 0100000000000000. Je rappelle que la structure de C est la suivante :

```
-----------------------------------------------
15:14:13:12:11:10: 9: 8: 7: 6: 5: 4: 3: 2: 1: 0
-----------------------------------------------
                                      .XS.<-B->.
                                   .<----- A ---->.
<S>.<---------------- M ----------->.<-- X ->.
```

On a donc S=0 c'est à dire signe positif, M=1 et X=0. Le nombre contenu dans C est donc 1×10^0 soit 1. Il reste alors à envoyer cette valeur 1 à la routine FNRTN1 (qui se charge de l'affichage) par l'instruction `GOVLNG` (pour Go Very LoNG).

Nous pouvons donc maintenant assembler cette fonction en passant sous Forth et en appelant le programme d'assemblage par `" SOURCE1" ASSEMBLE` (en supposant que vous ayez appelé votre fichier source SOURCE1). Une fois l'assemblage terminé sans erreur, il faut se rappeler que notre Lex tout neuf n'a pas été encore configuré et que dans ces conditions, si l'on essaye d'exécuter la fonction `ONE`, on n'obtiendra en échange qu'un message `Excess Char.` Pour remédier à cela, il suffit d'éteindre sa machine et de la rallumer pour que notre Lex soit pris en compte par le système et configuré dans le bLEX. A partir de maintenant, on dispose d'une nouvelle fonction `ONE` qui renvoie 1. Génial, non ?

Les conditions lors de l'entrée dans un Lex

Une fois que le système d'exploitation a vérifié la syntaxe de la fonction, il passe la main à notre programme. Pour que tout fonctionne correctement, il faut cependant se plier à certaines règles. En particulier, au retour de notre programme, le HP-71 s'attend à retrouver dans certains registres des valeurs importantes. C'est notamment le cas pour D0 et D1 (les deux registres d'index) dont nous verrons l'usage dans un autre article. Par contre les registres A, B, C, D, R0..R3 sont disponibles pour faire ce que vous voudrez.

On se retrouve également avec P=0, mode hexadécimal et dans le cas d'une fonction, la partie signe du registre C (que l'on note C[S]) contient le nombre de paramètres que l'utilisateur a donné lors de l'appel.

LEX à deux fonctions

Nous allons continuer notre initiation avec l'écriture d'un deuxième Lex qui apportera deux nouvelles fonctions : `TWO` et `THREE` (deux et trois en anglais) pour continuer dans la suite logique de SOURCE1. Voici donc le listing du fichier source que j'ai appelé SOURCE2 :

```
        LEX     'LEX2'

        ID      #5C
        MSG     0
        POLL    0

FNRTN1  EQU     #0F216

        ENTRY   TWOe
        CHAR    #F
        ENTRY   THREEe
        CHAR    #F

        KEY     'TWO'
        TOKEN   2
        KEY      'THREE'
        TOKEN   3

        ENDTXT

        NIBHEX  00      Pas de paramètre pour TWO
TWOe    C=0     W
        P=      14
        LCHEX   2
        GOTO    fin

        NIBHEX  00      Pas de paramètre pour THREE
THREEe  C=0     W
        P=      14
        LCHEX   3
fin     GOVLNG  FNRTN1

        END
```

Etudions maintenant ce Lex. Les deux fonctions qu'il apporte ressemblent à ONE comme deux gouttes d'eau. On trouve bien les inséparables ENTRY et CHAR deux fois ainsi que leurs collègues KEY et TOKEN. La fonction TWO aura pour id 5C et pour TOKEN 2 et THREE aura le même id (forcément puisque que dans le même Lex) et 3 pour TOKEN.

Comme pour ONE, ces fonctions ne prennent aucun paramètre en entrée. On place donc NIBHEX 00 devant chacune pour l'indiquer.

Il reste alors à charger dans C[M] la bonne valeur avant de rendre la main à FNRTN1. Comme on appelle cette routine au sortir des deux fonctions on peut remplacer un GOVLNG FNRTN1 par GOTO fin. Ceci permet de gagner 3 quartets car l'instruction GOTO en prend 4 alors que GOVLNG est plus gourmande avec 7 quartets. Ceci s'explique par le fait que GOTO utilise une adresse relative (sur trois quartets) alors que GOVLNG utilise l'adresse absolue (donc sur 20 bits).

Il ne vous reste plus qu'à assembler ce Lex et à l'essayer (n'oubliez pas d'éteindre puis de rallumer votre HP-71) en tapant par exemple ONE+TWO+THREE ce qui doit faire 6 si je ne m'abuse. Facile comme 1, 2, 3 non ?

Remarque : En ce qui concerne les Lex que vous inventerez (ce vous ne manquerez pas de faire, j'espère), je vous conseille d'utiliser le numéro d'id 5E ce qui évitera les interférences avec les Lex que je proposerai (d'id 5C) et ceux d'HP (id 5D).

Représentation interne des nombres

Pour vous permettre de faire d'autres Lex, je vais parler maintenant plus en détail de la représentation interne des nombres dans un registre de 64 bits.

Les nombres (non complexes) peuvent s'écrire comme chacun le sait sous la forme $n \times 10^p$ où n est la mantisse et p la puissance de 10. Le nombre peut être, bien entendu, précédé d'un signe négatif. C'est sous cette forme que sont stockés les nombres dans les registres du microprocesseur du 71.

En fait, il existe deux représentations suivant que l'on veut avoir une mantisse de 12 digits de précision ou de 15. Cette dernière représentation nécessite alors deux registres pour contenir un nombre. Aujourd'hui, nous allons voir la première représentation. La structure est alors la suivante :

```
15                                  0
+-+-----------------------------+------+
|S| Mantisse                    |Exp.  |
+-+-----------------------------+------+
 1           12                    3    = 16
```

Le signe est donc sur un quartet, la mantisse sur 12 et l'exposant sur 3 ce qui donne :

```
15                                  0
+-+-----------------------------+------+
|S| M11 . . . . . . . . . . M0 |E2..E0|
+-+-----------------------------+------+
 1           12                    3    = 16
```

La mantisse est un nombre non signé précédé d'un quartet de signe, S, valant 0 pour un signe positif et 9 pour un nombre négatif. L'exposant quant à lui est représenté sous la forme d'un nombre en complément à 10. C'est à dire que pour représenter -5 par exemple, on aura E2..E0 = 1000 + (-5) soit 995. Il est donc assez facile de coder n'importe quel nombre dans un registre tel que C.

Pour finir...

Pour finir l'article de ce mois-ci, je vous propose deux autres petits Lex qui vous permettront de mieux comprendre la représentation des nombres.

Le premier contient une fonction HPI (Half-PI) qui renvoie $\pi/2$, le programme est simple. On utilise LCHEX pour charger dans C[M] la valeur désirée. Il ne faut pas oublier de mettre P à 4 avant de charger les 11 digits de facon à avoir le 1 de tête dans C[14] (c'est à dire M11). Je rappelle que LCHEX charge le nombre en commençant par les poids faibles dans le quartet de C pointé par P puis incrémente P et ainsi de suite. On aura donc bien C[M] = 15707963268.

Préalablement on a effacé C[W] ce qui met le signe à 0 (donc positif) ainsi que l'exposant. Voic donc le listing du fichier source HPILEXS (le S est pour indiquer que c'est un fichier source) :

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

FNRTN1  EQU     #0F216

        ENTRY   HPIe
        CHAR    #F
        KEY     'HPI'
        TOKEN   4

        ENDTXT

        NIBHEX  00
HPIe    C=0     W
        P=      4
        LCHEX   15707963268
        GOVLNG  FNRTN1

        END
```

Après avoir assemblé vous pouvez essayer la nouvelle fonction HPI en vérifiant que, par exemple, PI/2 = HPI donne bien 1. La précision peut poser ici des problèmes : en effet si PI/2 égale HPI, il n'en est pas de même pour PI et 2*HPI.

Le deuxième Lex, dont le fichier source PPCLEXS et imprimé ci-dessous, apporte une fonction MYPPC qui renvoie un nombre qui sera votre numéro de PPC. Ce Lex est donc à compléter par vous en fonction de votre numéro de membre. Pour l'exemple j'ai pris le mien, soit 192. On l'écrit sous la forme 1.92E2 et l'on doit se débrouiller pour avoir :

```
C[W]=0192000000000002
```

Pour faire cela, on dispose de plusieurs solutions :

On peut faire par exemple :

```
LCHEX 0192000000000002
```

ou bien, comme beaucoup de chiffres sont à zéro, effacer d'abord C[W] puis mettre C[14..12] = 192 et C[0] = 2. C'est ce que j'ai choisi de faire car on économisera 3 quartets par rapport à la solution précédente. On profite donc de P=0 pour charger en premier l'exposant. Il reste à pointer C[12] avec P et à charger 192 avant d'appeler FNRTN1 et le tour est joué. Je vous laisse donc adapter le fichier source suivant en fonction de votre numéro.

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

FNRTN1  EQU     #0F216

        ENTRY   MYPPCe
        CHAR    #F
        KEY     'MYPPC'
        TOKEN   5

        ENDTXT

        NIBHEX  00
MYPPCe  C=0     W

        LCHEX   2           L'exposant est 2

        P=      12
        LCHEX   192         Chargement de la mantisse

        GOVLNG  FNRTN1

        END
```

Il restera à compiler ce Lex et à l'essayer. Je vous donne rendez-vous au mois prochain pour un article avec lequel nous continuerons à étudier les fonctions, mais nous verrons comment leur passer des paramètres et faire des opérations dessus. A bientôt donc...

A partir de maintenant j'indiquerai en fin d'article où trouver, pour ceux que ça intéresse et qui ont les *I.D.S.*, des informations complémentaires sur le sujet de mon article. Je vous propose donc de consulter :

- Le paragraphe 6-1 page 6-1 sur la structure des fichiers Lex dans le volume I.
- Les conditions d'entrée de FNRTN1 dans le volume II page 8-3 ou volume III page 5 du module Execution controller (adresse #0F216).
- La structure du bLEX chapitre 12.9 volume I.

- Le format des nombres dans les registres paragraphe 13.2.1 du volume I.
- Dans les IDS III (module Functions) : Les points d'entrée de EPS (adresse #0B7A1), PI (adresse #0C000), INF (adresse #0B7AA) et MAXREAL (adresse #OB76E).
- Toujours dans les IDS III (module Flags, Traps and Modes) : les routines des fonctions INX, UNF, OVF, DVZ et IVL à partir de l'adresse #13883.
- Dans le numéro 39 de JPC (novembre 1986), l'article « Les débuts » de Pierre David et le chapitre Représentation interne des données de l'article de Jean-Jacques Dhénin, page 23.

Jacques Baudier (192)

NDLR : N'hésitez pas à encourager Jacques ; vous pouvez le contacter à l'adresse suivante :
4 impasse Daniel René
78800 Houilles

DEUX FONCTIONS POUR COMPARER

Le petit Lex que je vous propose fournit deux fonctions pour comparer des nombres. La différence avec les opérateurs de comparaison est que, ici, le test est fonction d'un nombre entier :

1 : <
2 : =
3 : ≤
4 : >
5 : < >
6 : ≥

Supposons, par exemple, que vous vouliez faire un programme de tri banalisé, qui sache trier en ordre ascendant ou descendant. Il suffit d'utiliser une de ces deux fonctions. Le tri pourra se faire simplement en demandant un nombre entre 1 et 6.

Fonction STEP

Syntaxe : STEP (*x*, *c*, *n*)
Renvoie *x* si le test est vrai, 0 sinon.
Par exemple, si on veut tester $x<c$, on fera TEST(x,c,1).

Fonction TEST

Syntaxe : TEST (*x*, *c*, *n*)
Renvoie 1 si le test est vrai, 0 sinon.
Par exemple, si on veut tester $x<>c$, on fera TEST(x,c,5).

Bonnes comparaisons...

Jack Elhay

```
        LEX     'TESTLEX'
* Fonctions de comparaisons numériques banalisées
* codage :
*   1 : <
*   2 : =
*   3 : <=
*   4 : >
*   5 : <>
*   6 : >=
        ID      #5C
        MSG     0
        POLL    0
        ENTRY   Step
        CHAR    #F
        ENTRY   Test
        CHAR    #F
        KEY     'STEP'    STEP ( X, C, n )
        TOKEN   20        Renvoie X si vrai, 0 sinon
        KEY     'TEST'    TEST ( X, C, n )
        TOKEN   21        Renvoie 1 si vrai, 0 sinon
        ENDTXT

ARGERR  EQU     #0BF19    Invalid Arg
RNDAHX  EQU     #136CB    dépile un réel en hexa A(A)
POP1R   EQU     #0E8FD    dépile réel, pas complexe
TST12A  EQU     #0D476    compare 2 réels A(W) et C(W)
FNRTN4  EQU     #0F238    sortie de la fonction

Argerr  GOVLNG  ARGERR

* 3 paramètres numériques obligatoires pour STEP
        NIBHEX  88833
Step    ST=0    0         Flag 0 := 0 pour STEP
        GOTO    start     Saute dans le code commun

* TEST a aussi 3 paramètres numériques obligatoires
        NIBHEX  88833
Test    ST=1    0         Flag 0 := 1 pour TEST
start   GOSBVL  RNDAHX    A(A) := n dépilé en hexa
        GONC    Argerr    mais n ne doit pas être <0
        LCHEX   #06       C(B) := 6 (limite sup.)
        ?A>C    B         n > 6 ?
        GOYES   Argerr    oui : erreur
        R0=A              non : ok, le sauver en R0
```

```
        D1=D1+ 16       D1 pointe sur C
        GOSBVL POP1R    récupère C
        R1=A            sauver C en R1
        D1=D1+ 16       D1 pointe sur X
        GOSBVL POP1R    récupère X
        R2=A            Sauve X en R2
        C=R0            récupère n dans C(B)
        P=C    0        Pointeur P vaut n
        C=R1            registre C := paramètre C
        CD1EX           D1 est bien positionné
        R3=C            aussi il faut le sauver
        CD1EX           et restaurer C pour le test
        GOSBVL TST12A   teste ! P = le test à faire
        C=0    W        efface C(W) pour la sortie
        GONC   OUT      Si Cy = 0, test échoué, 0
        ?ST=0  0        STEP
        GOYES  stpout   Oui : résultat = X
        P=     14       P est mis pour placer 1
        LC(1)  1        charge le 1
        P=     0        restaure le pointeur P
        GOTO OUT        Saute la prochaine ligne
stpout  C=R2            Place X (le résultat) en C(W)
OUT     A=R3            restaure D1 depuis R3
        D1=A            restaure D1
        GOSBVL FNRTN4   renvoie le résultat
        END
```

CHOC EN RETOUR

La dernière version de DIVILEX est parue dans JPC 38 (octobre 1986). Toutefois, des modifications sont intervenues depuis.

Les fonctions FPRM et NPRM ont été renommées en FPRIM et NPRIM respectivement.

La procédure d'interruption de la fonction PRIM (DIVILEX) pouvait poser des problèmes, aussi l'interruption des fonctions PRIM, FPRIM, NPRIM et PHI se fait maintenant par deux pressions sur [ATTN] comme cela m'a été demandé par Janick Taillandier.

La limite inférieure des paramètres pour FPRIM et NPRIM est maintenant zéro.

Une securité est ajoutée : toutes les fonctions de DIVILEX refusent les paramètres non entiers.

Guy Toublanc (276)

```
        LEX     'DIVILEX'
        ID      #E1
        MSG     msg
        POLL    0
t       EQU     82
ARGERR  EQU     #0BF19
ATNFLG  EQU     #2F442
BSERR   EQU     #0939A
CLRFRC  EQU     #0C6F4
DCHXW   EQU     #0ECDC
FLOAT   EQU     #1B322
FNRTN1  EQU     #0F216
IDIV    EQU     #0EC7B
MPY     EQU     #0ECBB
POP1R   EQU     #0E8FD
RJUST   EQU     #12AE2
SPLITA  EQU     #0C6BF
        ENTRY   FPR
        CHAR    #F
        ENTRY   NPR
        CHAR    #F
        ENTRY   PGD
        CHAR    #F
        ENTRY   PH
        CHAR    #F
        ENTRY   PPM
        CHAR    #F
        ENTRY   PRM
        CHAR    #F
        KEY     'FPRIM'
        TOKEN   t
        KEY     'NPRIM'
        TOKEN   1+t
        KEY     'PGCD'
        TOKEN   2+t
        KEY     'PHI'
        TOKEN   3+t
        KEY     'PPCM'
        TOKEN   4+t
        KEY     'PRIM'
        TOKEN   5+t
        ENDTXT

msg     CON(2)  15
        CON(2)  15
eInter  CON(2)  (endmsg)-(eInter)
        CON(2)  07
        CON(1)  7
        NIBASC  'Function'
        CON(1)  7
        NIBASC  ' Interru'
        CON(1)  3
        NIBASC  'pted'
        CON(1)  12
endmsg  NIBHEX  FF

        NIBHEX  811
PH      GOSUB   pop1n
```

```
        R0=A
        R2=A
        C=0    W
        C+P+1
        ?A=C   W
        GOYES  RESA
HI0     GOSUB  GTES
R0A     R0=A
        ?A#C   W
        GOYES  LO0
        R1=C
        GOSUB  PHI
        GONC   RESA
LO0     GOSUB  div
        C=R1
        ?B=0   W
        GOYES  R0A
        GOSUB  PHI
        GONC   HI0
PHI     A=R2
        C=R1
        GOSUB  div
        C=R1
        C=C-1  W
R2A     R2=A
dh      GOSBVL DCHXW
        SETDEC
        A=R2
        GOSBVL MPY
        R2=A
        RTN

RESC    A=C    W
RESA    GOSBVL FLOAT
        C=A    W
        ADOEX
        D0=(5) ATNFLG
        DAT0=A S
        D0=A
OUT     GOVLNG FNRTN1

        NIBHEX 88888888882A
PPM     ST=1   2
        GONC   pgd

        NIBHEX 88888888882A
PGD     ST=0   2
pgd     GOSUB  DCS
        R1=A
        GOSUB  pop1n
        GONC   pgc
LOOPG   R3=C
        GOSUB  pop1n
pgc     GOSUB  PGC
        ?D#0   S
        GOYES  LOOPG
        GONC   RESC
PGC     C=R1
        ?C#0   W
        GOYES  DIF0
        ACEX   W
DIF0    R3=A
        R2=C
EUCL    GOSUB  R1C
        ?B=0   W
        GOYES  PGCD
        A=R1
        GONC   EUCL

PGCD    C=R1
        ?ST=0  2
        RTNYES
        A=R3
        GOSUB  div
        R0=A
        A=R2
        GOSUB  E12
        ACEX   W
        GOSUB  div
        C=R0
        ?A<C   W
        GOYES  pop
        GOSUB  dh
        R1=A
        RTN

pop     C=RSTK
ERR     GOVLNG ARGERR

DCS     D=C    S
pop1n   GOSBVL POP1R
        D1=D1+ 16
        ?A=0   S
        GOYES  C11
        ST=1   0
C11     LCHEX  011
        ?A>C   X
        GOYES  pop
        GOSBVL SPLITA
        GOSBVL CLRFRC
        GONC   pop
        A=B    M
        D=D-1  S
        GOVLNG RJUST

E12     P=     1
        C=0    W
        C=P    12
        P=     0
        RTN
CONST   C=0    W
        LCHEX  2
        D=C    W
        LCHEX  10
        R4=C
        RTN
```

111

165

```
        NIBHEX 8812
FPR     ST=1   1
        GONC   prim

        NIBHEX 8822
NPR     ST=0   1
prim    ST=0   0
        GOSUB  DCS
        C=0    W
        ?ST=1  0
        GOYES  LIM2
        GOSUB  E12
LIM2    R2=C
        ?D=0   S
        GOYES  LIM1
        R2=A
        ST=0   0
        GOSUB  pop1n
LIM1    C=R2
        ?A<=C  W
        GOYES  N<n
        ST=1   0
N<n     C=0    W
        LCHEX  2
        ?ST=1  0
        GOYES  DESC
        ?A>=C  W
        GOYES  DESC
        A=C    W
DESC    R0=A
        A=R2
        ?ST=0  0
        GOYES  ASC
        ?A>=C  W
        GOYES  ASC
        R2=C
ASC     GOSUB  CONST
        A=0    W
        R3=A
        C=R0
LOOPF   R0=C
        A=R2
        ?ST=0  0
        GOYES  CRES
        ACEX   W
CRES    ?C>A   W
        GOYES  RESN
        GOSUB  NTES
        ?A#C   W
        GOYES  CONT
        ?ST=1  1
        GOYES  RESU
        C=R3
        C=C+1  W
        R3=C
CONT    C=R0
        ?ST=0  0
        GOYES  INCR
```

222

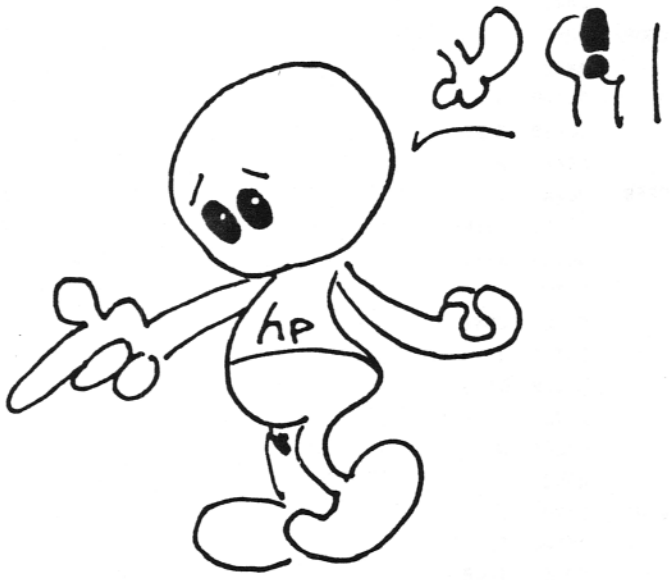
```
        C=C-1  W
        GONC   LOOPF
INCR    C=C+1  W
        GONC   LOOPF
RESN    A=R3
RESU    GOTO   RESA

R1C     R1=C
div     GOSBVL IDIV
        RTNCC

        NIBHEX 8812
PRM     GOSUB  DCS
        ?D=0   S
        GOYES  TVAL
        C=0    W
        LCHEX  999
        ?A>C   W
        GOYES  err
        C=C+1  A
        R1=C
        R0=A
        GOSUB  pop1n
        C=R1
        GOSUB  R2A
        A=R0
        A=A+C  W
TVAL    R0=A
        ?A#0   W
        GOYES  VAL
err     GOTO   ERR
VAL     GOSUB  GTES
        ?A#C   W
        GOYES  resc
        C=0    W
resc    GOTO   RESC

GTES    GOSUB  CONST
NTES    C=D    W
        GOSUB  TES0
        C=C+1  B
        GOSUB  TES0
        GOSUB  TES2
        GOSUB  TES2
        GOSUB  TES4
LOOP    GOSUB  TES2
        GOSUB  TES4
        GOSUB  TES2
        GOSUB  TES4
        GOSUB  TES6
        GOSUB  TES2
        GOSUB  TES6
        GOSUB  TES4
        GOSUB  TES2
        GOSUB  TES4
        GOSUB  TES6
        GOSUB  TES6
        GOSUB  TES2
```

277

```
        GOSUB   TES6
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES6
        GOSUB   TES4
        GOSUB   TES6
        GOSUB   TES8
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES4
        GOSUB   TES8
        GOSUB   TES6
        GOSUB   TES4
        GOSUB   TES6
        GOSUB   TES2
        GOSUB   TES4
        GOSUB   TES6
        GOSUB   TES2
        GOSUB   TES6
        GOSUB   TES6
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES4
        GOSUB   TES6
        GOSUB   TES2
        GOSUB   TES6
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES4
        GOSUB   TES2
        GOSUB   TES10
        GOSUB   TES2
        ?ST=0   12
        GOYES   PL10
        ADOEX
        DO=(5) ATNFLG
        A=DAT0 S
        ?A=0    S
        GOYES   DOA
        SETHEX
        A=A+1   S
        GOC     setdec
        DAT0=C S
        DO=A
        P=      0
        LC(4)   #E107
        GOVLNG BSERR

setdec SETDEC
DOA     DO=A
PL10    A=0     W
        P=      1
        A=A+1   P
        C=C+A   W
        A=R0
        GOSUB   R1C
```

334

```
        C=R1
        P=      7
        ?B=0    WP
        GOYES   rnp
        ?C>A    W
        GOYES   RESP
        GOTO    LOOP
RESP    C=R0
rnp     GOC     RNP
TES10   C=C+D   WP
TES8    C=C+D   WP
TES6    C=C+D   WP
TES4    C=C+D   WP
TES2    C=C+D   WP
TES0    R1=C
        A=R0
        P=      0
HI      ?C>=A   W
        GOYES   LO
        CSL     W
        P=P+1
        GONC    HI
LO      A=A-C   W
        GONC    LO
        A=A+C   W
        P=P-1
        GOC     LO1
        CSR     W
        GONC    LO
LO1     C=R1
        P=      7
        ?A#0    WP
        RTNYES
        C=RSTK
        C=R1
RNP     A=R0
        RTN
        END
```



372

LA GUERRE DES NOYAUX
(ACTE I)

Deux programmes informatiques, dans leur habitat naturel que constitue la mémoire d'un HP-71, se livrent une lutte sans merci, adresse par adresse. Parfois, ils partent à la rencontre de l'adversaire ; parfois, ils construisent un barrage de bombes numériques ; parfois ils se retranchent eux-mêmes dans une zone moins dangereuse où s'arrêtent un moment pour panser leurs blessures. Tel est le jeu appelé *Core War*.

Bien sûr, personne ne joue ! Une fois écrits par leurs auteurs, puis entrés en mémoire, les créateurs des programmes en lutte ne peuvent qu'observer leur machine et attendre passivement la mort ou la survie du produit qu'ils ont mis des heures à concevoir et à mettre au point. L'issue finale dépend uniquement de celui des deux programmes qui sera le premier touché dans une zone vulnérable.

Les programmes rivaux de Core War sont écrits dans un langage spécial, appelé *Redcode* par son auteur (A. K. Dewdney, publié dans la revue *Pour la science*), et qui fait partie des langages d'assemblage. L'ensemble *Core War* se compose donc de quatre éléments : une mémoire de 8000 mots, le langage d'assemblage *Redcode*, un programme superviseur appelé *MARS*, pour *Memory Array Redcode Simulator*, et l'ensemble des programmes combattants.

A l'origine, deux de ces programmes combattants sont introduits en mémoire à des emplacements choisis au hasard ; aucun d'eux ne sait où se trouve l'autre. Cependant, pour ce premier aperçu, le programme fourni pour HP-71 vous permettra de choisir vous-même les adresses de départ, afin de mieux contrôler vos premières expériences ; par la suite, une modification minime rétablira la règle originale prévue pour un véritable affrontement, en rétablissant un générateur aléatoire d'adresses.

Le superviseur *MARS* pilote le déroulement des programmes en temps partagé. Les deux programmes tournent chacun à leur tour : on exécute une instruction du premier, puis une instruction du second et ainsi de suite.

Le but est évidemment d'écrire un programme afin de détruire le programme adverse en démolissant ses instructions. Le combat prend fin lorsque le superviseur *MARS* trouve dans un programme une instruction qui n'est plus exécutable, ce qui est considéré être une conséquence de la guerre ; le programme inexécutable est déclaré vaincu.

Redcode comprend 9 instructions :

`MOV A B`
L'instruction transfère le contenu de l'adresse *A* à l'adresse *B*.

`ADD A B`
L'instruction ajoute le contenu de l'adresse *A* au contenu de l'adresse *B*.

`SUB A B`
L'instruction soustrait le contenu de l'adresse *A* du contenu de l'adresse *B*.

`JMP A`
L'instruction transfère l'exécution à l'adresse *A*.

`JMZ A B`
L'instruction transfère l'exécution à l'adresse *A* seulement si le contenu de l'adresse *B* est nul.

`JMG A B`
L'instruction transfère l'exécution à l'adresse *A* seulement si le contenu de l'adresse *B* est > 0.

`DJZ A B`
L'instruction retranche 1 du contenu de l'adresse *B* et saute à l'adresse *A* seulement si le résultat est nul.

`CMP A B`
L'instruction compare le contenu des adresses *A* et *B* et saute l'instruction suivante s'ils sont différents.

`DAT B`
Déclaration de donnée, *B* est la valeur de la donnée.

Cette dernière instruction est particulière car non exécutable par *MARS* : celui-ci déclare perdant le programme correspondant quand on lui demande de traiter une telle instruction. Elle peut servir de mémoire de travail pour conserver une donnée dont le programme aura besoin.

La mémoire a une structure circulaire : l'adresse 7999 (dans le cas d'une mémoire de 8000 mots) est suivie de l'adresse 0 : le superviseur *MARS* interprète toute adresse supérieure à 7999 en ne conservant que le reste de la division par 8000.

Précisons maintenant les particularités de l'adressage en *Redcode*. La méthode utilisée est celle de l'adressage relatif ; elle est employée pour qu'un programme de combat n'ait aucun moyen de connaître sa position absolue dans la mémoire.

Trois modes d'adressages sont possibles :

- direct (pas de préfixe) :
L'argument représente directement l'adresse relative.
Exemple : l'instruction `JMP -7` demande à *MARS* d'aller chercher la prochaine instruction à exécuter sept positions mémoire avant celle de l'instruction `JMP -7`.
De même, l'instruction `MOV 3 100` commande au superviseur *MARS* de progresser de trois positions mémoire, de lire le contenu de la case et de le recopier 100 positions au-delà de l'instruction `MOV` elle-même.

- indirect (préfixe @) :
L'argument représente une adresse qui elle contient l'adresse relative.
Exemple : l'instruction `MOV @3 100` qui recopie à l'adresse relative 100 (soit 515 dans cet exemple chiffré) le contenu de l'adresse relative (ici 413) obtenue à partir de l'adresse relative 3 (soit 418 - 5) :

```
412
413  DAT     22
414
415  MOV @3 100
416
417
418  DAT     -5
419
 :
514
515  DAT     22  <- résultat
516
```

- immédiat (préfixe #) :
L'argument n'est pas une adresse mais une quantité entière (donnée).
Exemple : `MOV #5 -1` recopie 5 à la position mémoire précédant l'instruction.

Voyons maintenant comment utiliser le programme `MARS71`.

Après un `[RUN]`, entrez la taille que vous désirez donner à la mémoire circulaire ; celle-ci sera comparée à celle disponible sur votre engin.

Entrez ensuite le nombre de programmes que vous désirez exécuter. Il est possible que vous ne vouliez suivre qu'un seul programme pour commencer : taper 1, sinon 2 pour mettre en scène un duel...

Vous devez ensuite entrer l'adresse de la première instruction de votre programme. Puis, c'est l'accès au compilateur qui vous permet d'entrer votre programme, ligne par ligne. Une ligne commence par la mnémonique et est suivie du ou des deux arguments, chaque zone étant séparée par au moins un espace.

Une fois le programme entièrement entré, tapez simplement `[ENDLINE]` pour revenir, soit à l'entrée du second programme, soit pour voir démarrer l'exécution du premier. Dans chaque cas, vous pouvez suivre le combat à l'affichage en lisant le numéro de la case mémoire en cours d'exécution, les drapeaux 1 et 2 vous identifiant à quel programme appartient l'instruction en cours.

Un utilitaire recherchera automatiquement la première instruction exécutable si votre programme débute par des instructions de réservation `DAT`. Attention, les champs adresse sont limités de -999 à 999...

Voici deux exemples de programmes de combat dont l'étude vous permettra de saisir la philosophie de ce type de combat :

Le plus court : **Imp**

```
MOV 0 1
```

Ce programme n'a d'autre finalité que de se reproduire et d'envahir ainsi toute la mémoire dans le but d'écraser son adversaire.

Le bombardier : **Dwarf**

```
DAT       -1   ; pointeur
ADD  #5   -1   ; progression du pointeur
MOV  #0   @-2  ; lâcher de la bombe
JMP  -2        ; retour à l'additionneur
```

Ce programme, lui, bombarde une case mémoire sur cinq avec des zéros : il faut savoir que zéro est le code-machine de l'instruction non exécutable `DAT`. Un programme qui tombe sur un zéro est donc contraint à l'arrêt...

Le mois prochain, nous étudierons plus en détail le programme `MARS71` et nous vous proposerons des combats un peu plus actifs. N'hésitez pas à nous faire part de toutes vos idées sur la question, et de vos avis sur un éventuel tournoi dans un cadre à préciser.

Bonne recherche et programmation,

Olivier Arbey (118)

Programme "MARS" (simulateur de Redcode)

```
  - MARS71
 20 DESTROY ALL @ OPTION BASE 0 @ CFLAG 1 @ CFLAG 2 @ CFLAG 7
  - Contrôle taille mémoire en fonction de la place disponible
 40 INPUT 'Taille mémoire ? ';M @ IF M*10>MEM*1.1 THEN 40
 50 DIM T(M)
 60 INPUT 'Nombre de prgms ? ';N7 @ IF N7=1 THEN SFLAG 6 ELSE CFLAG 6
  - Choix de l'adresse de la 1ère instruction du prgm 1
 80 INPUT "Adresse prgm 1 ? ";I @ I=MOD(I,M)
  - Saisie du programme 1
100 CALL COMPIL(T(),(I),M)
  - Recherche de la 1ère instruction exécutable, dans le cas où le prgm commence par des DAT
120 CALL DATSEEK(T(),I,M)
  - Exécution du prgm 1 si un seul prgm a été demandé
140 IF FLAG(6) THEN 220
  - Choix de l'adresse de la 1ère instruction du prgm 2
160 INPUT "Adresse prgm 2 ? ";J @ J=MOD(J,M)
  - Saisie du programme 2
180 CALL COMPIL(T(),(J),M)
  - Recherche de la 1ère instruction exécutable, dans le cas où le prgm commence par des DAT
200 CALL DATSEEK(T(),J,M)
  - Exécute une instruction du prgm 1
220 SFLAG 1 @ CFLAG 2 @ CALL EXEC(T(),I,M) @ IF FLAG(7) THEN END
  - Boucle si prgm 1 seul
240 IF FLAG(6) THEN 220
  - Exécute une instruction du prgm 2
260 CFLAG 1 @ SFLAG 2 @ CALL EXEC(T(),J,M) @ IF FLAG(7) THEN END
270 GOTO 220
  - Exécute l'instruction de rang I

============================================================
290 SUB EXEC(T(),I,M)
300 DISP I; @ CFLAG 5
310 N2=T(I)/100000000 @ N3=INT(N2)
320 IF N3=0 THEN 410
330 N2=10*FP(N2)
340 A0=INT(N2) @ N2=10*FP(N2)
350 B0=INT(N2) @ N2=1000*FP(N2)
360 A=INT(N2) @ B=1000*FP(N2)
370 ON N3 GOSUB 470,530,590,650,680,720,760,830
380 IF FLAG(5)=0 THEN I=I+1
390 I=MOD(I,M)
400 GOTO 870
  - Une instruction DAT a été rencontrée : FIN
420 DISP 'Fini pour ';
430 IF FLAG(1) THEN DISP '1' ELSE DISP '2'
440 SFLAG 7 @ BEEP
450 GOTO 870
  - MOV
470 CALL DONNEE(T(),A,A0,I,M)
480 N3=MOD(I+B,M)
490 IF B0=1 THEN T(N3)=A @ RETURN
500 IF B0=2 THEN T(MOD(I+B+T(N3),M))=A
510 RETURN
  - ADD
530 CALL DONNEE(T(),A,A0,I,M)
```

```
540 N3=MOD(I+B,M)
550 IF B0=1 THEN T(N3)=T(N3)+A @ RETURN
560 IF B0=2 THEN N4=MOD(I+B+T(N3),M) @ T(N4)=T(N4)+A
570 RETURN
  - SUB
590 CALL DONNEE(T(),A,A0,I,M)
600 N3=MOD(I+B,M)
610 IF B0=1 THEN T(N3)=T(N3)-A @ RETURN
620 IF B0=2 THEN N4=MOD(I+B+T(N3),M) @ T(N4)=T(N4)-A
630 RETURN
  - JMP
650 CALL ADR(T(),A,A0,I,M)
660 RETURN
  - JMZ
680 CALL DONNEE(T(),B,B0,I,M)
690 IF B=0 THEN CALL ADR(T(),A,A0,I,M)
700 RETURN
  - JMG
720 CALL DONNEE(T(),B,B0,I,M)
730 IF B>0 THEN CALL ADR(T(),A,A0,I,M)
740 RETURN
  - DJZ
760 A1=A @ A=1
770 GOSUB 600
780 A=A1
790 CALL DONNEE(T(),B,B0,I,M)
800 IF B=0 THEN CALL ADR(T(),A,A0,I,M)
810 RETURN
  - CMP
830 CALL DONNEE(T(),A,A0,I,M)
840 CALL DONNEE(T(),B,B0,I,M)
850 IF A#B THEN I=I+1
860 RETURN
870 END SUB

================================================================
  - Extraction de la donnée d'adresse A et de mode d'adressage A0

================================================================
890 SUB DONNEE(T(),A,A0,I,M)
900 IF A0=1 THEN A=T(MOD(I+A,M)) @ GOTO 920
910 IF A0=2 THEN A=T(MOD(I+A+T(MOD(I+A,M)),M))
920 END SUB

================================================================
  - Extraction de l'adresse pour une instruction de branchement

================================================================
940 SUB ADR(T(),A,A0,I,M)
950 SFLAG 5
960 IF A0=1 THEN I=I+A @ GOTO 980
970 IF A0=2 THEN I=I+A+T(MOD(I+A,M))
980 END SUB

================================================================
  - Recherche de la 1ère instruction exècutable

================================================================
```

```
1000 SUB DATSEEK(T(),I,M)
1010 IF T(MOD(I,M))<100000000 THEN I=I+1 @ GOTO 1010
1020 END SUB

================================================================
   - Compilateur de Redcode

================================================================
1040 SUB COMPIL(T(),I,M)
1050 INPUT "Op ? ";N$ @ N$=UPRC$(N$)
1060 IF N$='' THEN 1330
1070 NO$=N$[1,3] @ N$=N$[5]
1080 IF NO$='MOV' THEN N1=1
1090 IF NO$='ADD' THEN N1=2
1100 IF NO$='SUB' THEN N1=3
1110 IF NO$='JMP' THEN N1=4 @ N$=N$&' ' @ SFLAG 5 ELSE CFLAG 5
1120 IF NO$='JMZ' THEN N1=5
1130 IF NO$='JMG' THEN N1=6
1140 IF NO$='DJZ' THEN N1=7
1150 IF NO$='CMP' THEN N1=8
1160 IF NO$='DAT' THEN N1=MOD(VAL(N$),M) @ GOTO 1310
1170 N1=N1*10
1180 IF POS(N$,' ')=1 THEN N$=N$[2] @ GOTO 1180
1190 IF N$[1,1]='@' THEN N1=N1+2 @ N$=N$[2] @ GOTO 1210
1200 IF N$[1,1]='#' THEN N$=N$[2] @ GOTO 1210 ELSE N1=N1+1
1210 N1=N1*10000
1220 N3=POS(N$,' ')
1230 N1=N1+MOD(VAL(N$[1,N3]),M)
1240 IF FLAG(5) THEN N1=N1*1000 @ GOTO 1310
1250 N$=N$[N3+1]
1260 IF POS(N$,' ')=1 THEN N$=N$[2] @ GOTO 1260
1270 IF N$[1,1]='@' THEN N1=N1+2000 @ N$=N$[2] @ GOTO 1290
1280 IF N$[1,1]='#' THEN N$=N$[2] @ GOTO 1290 ELSE N1=N1+1000
1290 N1=N1*1000
1300 N1=N1+MOD(VAL(N$),M)
1310 T(I)=N1 @ I=I+1 @ I=MOD(I,M)
1320 GOTO 1050
1330 END SUB
```

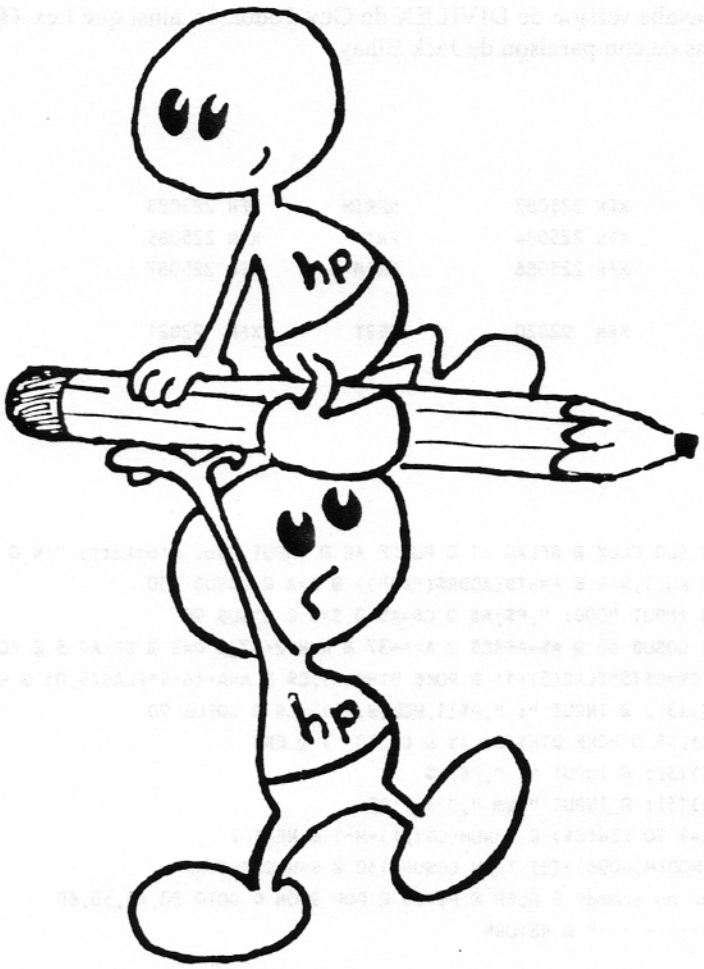
****

Programme "BOOT" (partie Basic du programme de X. Bille paru dans JPC 47)

- partie BASIC du programme de tracé de courbes
  auteur : Xavier Bille
  variables :
  c$ menu, c commande reconnue dans c$
  o$ choix utilisateur
  f$,x$,y$ expressions littérales de fonctions
  x,y abscisse et ordonnée
  a,b valeur d'un pixel (abscisse, ordonnée)
  1/a,1/b pas de calcul
  s paramètre de calcul
  x1,x2, val. min., val. max. en abscisse
  y1,y2 idem en ordonnée

```
    t1,t2 idem cas de courbes paramétriques
    d  pas de calcul du paramètre
    k paramètre indiquant le numéro de fenêtre à afficher
    k$ touche, h$ choix de la courbe
 30 DEFAULT OFF @ SFLAG -1 @ OPTION ROUND NEAR @ CFLAG MATH
 40 DISP "Graphic Function" @ C$="cl,gr,rn,sh,ot"
 50 DISP C$; @ INPUT " :",O$;O$ @ C=(POS(UPRC$(C$),O$[1,2])-1)/3
 60 IF C>=0 AND LEN(O$)=2 THEN O$ ELSE DISP "Command not found"
 70 GOTO 50
  - permet de sortir du programme.

============================================================
 80 'OT': DEFAULT EXTEND @ CFLAG MATH @ END
  - saisie de(s) la(les) fonction(s)
    tracé dans le repère

============================================================
 90 'GR': IF NOT FLAG(5) OR NOT (FLAG(1) OR FLAG(0)) THEN GOTO 'RN'
100 INTEGER N @ IF FLAG(1) AND NOT FLAG(0) THEN 160
110 LINPUT "F(x)=",F$;F$
120 DISP "Y = f(X) Curve "&CHR$(27)&'>'&CHR$(27)&'Q'
130 ON ERROR GOSUB 'ERR' @ S=1/A/(1+FLAG(4)) @ FOR X=X1 TO X2 STEP S
140 Y=VAL(F$) @ FORTHX "PLOT"
150 NEXT X @ GOTO 210
160 LINPUT "X(t)=",X$;X$ @ LINPUT "Y(t)=",Y$;Y$
170 DISP "Parametric Curve"&CHR$(27)&'>'&CHR$(27)&'Q'
180 ON ERROR GOSUB 'ERR' @ S=D/(1+FLAG(4)) @ FOR T=T1 TO T2 STEP S
190 X=VAL(X$) @ Y=VAL(Y$) @ FORTHX "PLOT"
200 NEXT T
210 OFF ERROR @ BEEP 2000,.01 @ DISP CHR$(27)&'<',N;" : Errors."
  - affichage de la fenètre

============================================================
220 'SH': INTEGER K @ DISP "Show"
230 FORTHX "COMPOSE",K @ GDISP FORTH$
240 IF NOT KEYDOWN THEN 240
250 IF KEYDOWN('#50') THEN K=K-1
260 IF K<0 THEN DISP "Up." @ K=0
270 IF KEYDOWN('#51') THEN K=K+1
280 IF K>30 THEN DISP "Down." @ K=30
290 K$=KEY$ @ IF K$='#162' THEN K=0 ELSE IF K$='#163' THEN K=30
300 IF K$#' ' THEN 230
310 DISP "Ok." @ GOTO 50
  - saisie des paramètres du repère

============================================================
320 'RN': DISP "Range" @ CFLAG 5,0,1 @ ON ERROR GOSUB 'ERR'
330 INPUT "Parm(P), Funct(F) ",H$;H$ @ H$=UPRC$(H$)
340 IF H$='F' THEN SFLAG 0 @ GOTO 360 ELSE IF H$#'P' THEN 330
350 SFLAG 1 @ INPUT "Tmin,Tmax,T : ",STR$(T1)&','&STR$(T2)&','&STR$(D);T1,T2,D
360 INPUT "Xmin,Xmax : ",STR$(X1)&','&STR$(X2);X1,X2 @ IF X1>=X2 THEN 360
370 A=131/(X2-X1) @ IF A=0 OR ABS(A)=INF THEN 360
380 INPUT "Ymin,Ymax : ",STR$(Y1)&','&STR$(Y2);Y1,Y2 @ IF Y1>=Y2 THEN 380
390 B=127/(Y2-Y1) @ IF B=0 OR ABS(B)=INF THEN 380
400 FORTHX "RANGE" @ SFLAG 5 @ OFF ERROR @ GOTO 420
  - néttoyage de la matrice-écran
    tracé des axes
```

```
================================================================
 410 'CL': DISP "Clear"
 420 FORTHX " CLEAR AXIS" @ GOTO 50
   - traitement des érreurs numériques

================================================================
 430 'ERR': IF ERRN=20 OR ERRN=31 OR ERRN=11 THEN DISP "Abort:";ERRM$ @ GOTO 'OT'
 440 IF FLAG(OVF) OR FLAG(UNF) OR FLAG(DVZ) THEN DISP "wrn:"&ERRM$;
 450 IF FLAG(IVL) THEN DISP "Cannot evalue ";
 460 N=N+1 @ Y=0 @ IF FLAG(1) THEN X=0
 470 IF N>15 THEN DISP "Too many errors, abort" @ GOTO 'OT'
 480 DISP CHR$(27)&'>'&CHR$(27)&'R' @ CFLAG MATH @ RETURN
   - initialiser les variables

================================================================
 490 'BOOT': CFLAG ALL @ DESTROY ALL @ RUN
```



LE COIN DES LHEX

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

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

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

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

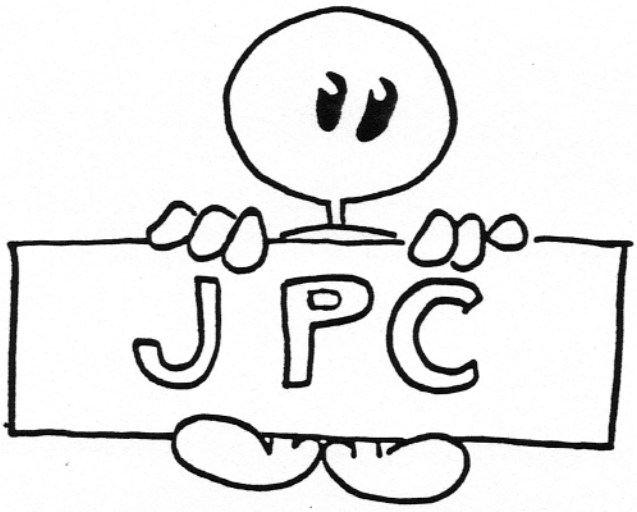
Si l'erreur "Erreur de somme" apparaît, vérifiez la ligne que vous avez introduite.

Vous trouverz donc le Lex CHARLEX nécessaire à la rédaction de votre article (voir "Ah ! Vous écrivez !"), la nouvelle version de DIVILEX de Guy Toublanc, ainsi que Lex TESTLEX contenant les deux fonctions de comparaison de Jack Elhay.

```
CHARLEX

DIVILEX   FPRIM      XFN 225082        NPRIM      XFN 225083
          PGCD       XFN 225084        PHI        XFN 225085
          PPCM       XFN 225086        PRIM       XFN 225087

TESTLEX   STEP       XFN  92020        TEST       XFN  92021
```

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

```
CHARLEX  ID#E1  624 octets

      0123456789ABCDEF sm

000: 34841425C4548502 35E
001: 802E008341029078 6B7
002: 5E4001E000000000 9FB
003: FE00000000800001F D55
004: F31BF961400032BF 0E8
005: 38F14A11DB10AD23 482
006: 07D532BFB8FD7911 835
007: 11AD754D7A101743 BB8
008: 11014D1CB15D0000 F23
009: 71450375FF864834 2A0
00A: 5655581008355654 5F7
00B: 5810002455565870 944
00C: 0026555658700836 C96
00D: 5556581008364545 FEC
00E: 4A30000A49724000 33F
00F: 0808094A2C180814 6A8
010: A464242008355455 A02
011: 581000054C714000 D48
012: 0C3142404C700832 0A4
013: 41414A70002078A0 3FC
014: 2F300000000000000 727
015: 0000000000000000 A37
016: 0000000000000000 D47
017: 0000000000000000 057
018: 0000000000000000 367
019: 0000000000000000 677
01A: 0000000000000000 987
01B: 0000000000000000 C97
01C: 0000000000000000 FA7
01D: 0000000000000000 2B7
01E: 0000000000000000 5C7
01F: 0000000000000000 8D7
020: 0000000000000000 BE7
021: 000000000000080C F12
022: 1A2808000808 0A2C 27C
023: 1B0008040E340800 5C5
024: 08001E3018000000 8FF
025: 0000000000000000 C0F
026: 0000000000000000 F1F
027: 0000000000000000 22F
028: 0201000000010200 545
029: 0000000201020000 85A
02A: 0001000100000002 B6E
02B: 0102010000000000 E82
02C: 0000000000000000 192
02D: 045E755142400101 4DE
02E: 0101010000000000 7F1
02F: 0000000000000000 B01
030: 0000070507000000 E24
031: 00000000083444C4 162
032: 44400D7901112D70 4C2
033: 050D750509700000 80C
034: 0D70000000384540 B4F
035: 4020014E322E3140 EA3
036: 084E794142400000 1F3
037: 000000000 2E4559 531
038: 3200000000000000 846
039: 0000000000000026 B5E
03A: 5556587008365556 EBD
03B: 5810083645464830 20E
03C: 0832414248700024 54F
03D: 5655587008345655 8AC
03E: 5810083446454830 BFB
03F: 0C3042414C700024 F50
040: 5556587008355654 2AD
041: 5810083546444830 5FC
042: 0C3142404C700025 952
043: 5455587008355455 CAC
044: 5810083544454830 FFA
045: 0C3140414C700875 35C
046: 14141870000A4972 6AD
047: 40000E3159454E30 A0D
048: 0C7A0F7949400024 D85
049: 5554587000084A71 0E1
04A: 40000C523A262D10 442
04B: 0424587458400875 799
04C: 1415187000094A70 AE9
04D: 4000083544454830 E2D
04E: 0C3140414C300C74 195
04F: 5655545000054C71 4EC
050: 40000             5E5

TESTLEX  ID#5C  104 octets

      0123456789ABCDEF sm

000: 45543545C4548502 362
001: 802E009341029078 6BC
002: 4D000C5415100000 A07
003: F0200000000000000 D2F
004: 043000FB0073000F 08E
005: 7354554054174554 3E2
006: 3545511FF8D91FB0 784
007: 888338406B008883 AF1
008: 38508FBC6315DD31 E8F
009: 609E64D10017F8FD 232
00A: F8E010117F8FDF8E 5FA
00B: 010211880D011913 942
00C: 710B1378F674D0AF CE1
00D: 2551860D02E30120 03D
00E: 660011A1131318F8 39C
00F: 32F0F             4C2
```



PPC PARIS CHAPTER

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 ? ______________________________
____
____

Je souhaite adhérer au club PPC Paris Chapter 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 Chapter,
par [ ] chèque bancaire, [ ] chèque postal ou [ ] mandat lettre

Eventuellement: je m'abonne à compter du |0|1|/|_|_|/19|_|_|

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

Le Journal JPC est le bulletin de liaison entre les membres de l'Association "PPC-PC", régie par la loi de 1901. Le Club est éditeur du JPC, et son siège social est au 56, rue Jean-Jacques Rouseau, 75001 Paris.

La maquette de ce numéro a été préparée et réalisée par Jacques Baudier, Pierre David, Jean-Jacques Dhénin et Janick Taillandier, grâce à un système comprenant un HP71B, deux lecteurs de disquettes HP9114A et une imprimante LaserJet.

Directeur de la publication : P. Guez
Numéro ISSN : 0762 - 381X

ENGLISH SUMMARY

JPC 48 - OCTOBER 1987

The letter in our *letter corner* on page 3 expresses the idea that it is necessary to have a very high technical skill to prepare an article for JPC. Clearly, this is not true : on the contrary if too many "high level" articles are published, there will be less members. This doesn't mean that there will be no more technical articles...

Next, on page 6, Michel Maupoux explains how to plot any function on the HP-28, even if an "undefined result" error occurs.

In the HP-41 column, Michael Markov presents us the first part of a very interesting series about the control of mass storage devices (especially the HP-9114A) from an HP-41 with Extended I/O Rom.

Eric Gengoux gives us for the HP-75, a small routine useful to test whether a file is present or not. This is a clever use of a function from the I/O Rom.

In the HP-71 column, you will find the fourth part of Jacques Baudier's introduction to assembly language programming on the HP-71. This regular column is appreciated by many members. This series is a lot of work.

Our Australian friend, Jack Elhay, presents us two small functions for the HP-71 he wrotes when he was a beginner. They should be interesting for anybody who wants to learn about assembly language.

The next article provides us with a new version of DIVILEX (first published last year).

On page 27, you will find the first article in a series by Olivier Arbey. It is inspired by an article by K. Dewdney in the French version of *Scientific American*. This is an interpreter of an hypothetical assembly language. The purpose of the game is to write two programs which try to kill the other. A program dies when it reaches a DAT instruction. Enjoy it !

Until next month,

Happy Programming