JPC

OCTOBRE 1988

NUMERO 58

Le numéro 40 F

A PROPOS DU CLUB

HP28

HP75

HP71

EDITORIAL

Chers membres,

L'appel lancé le mois dernier semble avoir été entendu. Des articles nous sont parvenus, et nous avons pu fabriquer ce *JPC* sans problème. Mais il y aura d'autres *JPC*, et vous devez continuer à nous envoyer vos oeuvres. Rappelez-vous que votre journal n'est composé que de ce que vous y mettez...

L'appel lancé en juillet pour la reprise de la *Bibliothèque de Genève* a lui aussi été entendu. Thomas Rouyer s'en est chargé. Si vous souhaitez en profiter, mettez-vous très rapidement en contact avec Thomas pour définir avec lui ce que vous pourrez faire de ces programmes pour HP-41 :

Thomas Rouyer
398 avenue Maurice Dauvergne
77350 Le Mée sur Seine

En revanche, et c'est inquiétant, la décision du Bureau de ne pas se représenter à la prochaine Assemblée Générale n'a pas suscité beaucoup de réactions. Une seule personne s'est manifestée pour faire partie du futur Bureau. **Ce n'est pas suffisant. Le Club a besoin de vous.** Je vous rappelle que la date limite est le 15 novembre. J'attends donc vos candidatures.

Je compte sur vous.

Pierre David (37)

20 + 22 + 27 - 28 = 41 ?

Depuis les dernières annonces d'HP en matière de calculatrices, vous vous sentez un peu perdu. Laquelle fait quoi ? Quelles sont leurs caractéristiques ? Combien valent-elles ?

Pour répondre à ces (angoissantes) questions, je vous propose un petit tour d'horizon de la gamme HP telle que vous pouvez la trouver chez votre distributeur favori. Les prix mentionnés sont approximatifs.

HP-20S

Le HP-20S est le bas de gamme actuel. Pour 400 F, vous disposez d'une calculatrice (oserais-je dire *calculette* ?) fonctionnant en notation algébrique. Une seule ligne de 12 caractères et le *look* d'une calculatrice classique. Elle est programmable et offre 99 pas et 10 registres. Les fonctions statistiques de base sont présentes. Notez que le calcul d'intégrales et la résolution numérique d'équations se font par des programmes intégrés. En bref, une bonne calculatrice de bas de gamme.

HP-22S

Toujours le même aspect vertical en un seul volet, toujours en notion algébrique, toujours avec une seule ligne d'affichage mais plus agréable, tel est le HP-22S pour 550 F. Le clavier dispose en jeu secondaire de touches alphabétiques. La résolution numérique d'équations est maintenant une fonction de base et non plus un programme intégré. En revanche, l'intégration numérique n'est plus disponible. Notons que ce calculateur dispose de fonctions financières. Le tout est programmable et dispose de 371 octets de mémoire utilisable.

HP-32S

Le HP-32S est similaire au HP-22S, mais fonctionne en notation RPN. L'intégration numérique est présente, mais pas les calculs financiers. Le prix est de 700 F pour une mémoire utilisateur de 390 octets, dont 260 au maximum peuvent être utilisées en programmation.

HP-27S

Vous le connaissez depuis longtemps maintenant. Le HP-27S a le même format que les HP-20, HP-22 et HP-32 mais dispose, lui, de deux lignes d'affichage. C'est le premier modèle de la gamme à posséder une interface infra-rouge permettant de le relier à une imprimante. Le prix est de 1000 F.

HP-28S

Je pense qu'il n'y a plus besoin de présenter le HP-28S. Son prix (approximatif) est de 2250 F).

Les oubliés

N'oubliez pas que les HP-11C, HP-12C, HP15C, HP-16C et HP-41 se vendent aussi toujours très bien.

J'ai omis dans ce bref tour d'horizon les calculateurs financiers HP-17B et HP-19B. J'espère que vous ne m'en voudrez pas trop...

J'ai également omis le HP-71B. Il est arrêté, mais le support continuera encore pendant 5 ans.

Ragots, bruits et rumeurs

Des rumeurs de plus en plus insistantes annoncent la sortie future du remplaçant du HP-41. En fait, on pense à un HP-28 musclé avec des capacités d'entrées/sorties. On pense également que le HP-41 ne sera pas retiré de sitôt, grâce aux ventes qui continuent à l'étonnement de tous.
On avance pour la sortie de cette éventuelle machine l'été 1989, marquant ainsi le dixième anniversaire du HP-41 et le cinquantième de HP.

Pierre David (37)

PLUS LOIN AVEC LA HP-27 ET SOLVE

Le manuel *Technical Applications* (#00027-90045), disponible en France depuis peu, pourrait passer inaperçu, eu égard à son petit format et au fait qu'il est en anglais... Ce serait dommage, car il introduit des instructions qui ne sont pas décrites dans le manuel utilisateur de la HP-27, et celles-ci augmentent de façon intéressante les possibilités du menu SOLVE de cette machine (ainsi que de la HP-17B).

Les deux nouvelles instructions

Elles sont décrites dans la première partie du manuel, intitulée *Advanced Solver Techniques*, et ont noms L (pour LET) et G (pour GET). Elles permettent de définir des variables intermédiaires, c'est à dire des expressions complètes qui pourront être réutilisées à plusieurs reprises par la suite, avec invocation par leur nom (donc, sans avoir à les réécrire

explicitement). En outre, ces variables pourront alors être masquées, c'est à dire ne pas apparaître dans les menus.

Mais ce n'est pas tout : certes, il est intéressant d'alléger menus et expressions, mais ces instructions permettent bien d'autres choses...

Par exemple, la récursion : on utilise la valeur précédemment calculée de la variable intermédiaire comme entrée dans l'itération suivante... Ou bien, on étend les possibilités de la fonction SIGMA, vers les boucles indéfinies...

Leur syntaxe

`L( <nomvar> : <expression> )`

`G( <nomvar> )`

Elles ne peuvent être utilisées que dans les équations du menu SOLVE (en aucun cas directement au clavier).

Le manuel explique en détail les différences entre variables formelles qui apparaissent dans les menus et les variables intermédiaires qui n'y apparaissent pas... Donc, dont la valeur est déterminée au moment où l'expression L() est évaluée, et qu'on ne peut invoquer que par G() car, sinon, on créerait une autre variable, formelle celle-là !

Quelques exemples

L'expression suivante permet de voir l'avantage de gain de place et de clarté :

```
A=B+C-LN(B+C)+X^(B+C)
```

qui peut également s'écrire :

```
A=L(D:B+C)-LN(G(D))+X^G(D)
```

On dira que l'exemple n'a guère d'intérêt, sorti de la mise en facteur de B+C. Mais voyons le second exemple :

```
A=2*A+B et A=2*G(A)+B
```

Quelle différence ? Dans la seconde expression, on est en pleine récursion ,on réinjecte en effet la précédente valeur de A... On peut imaginer les applications directes partout où il est question de développements en série, de convergence de suites récurrentes, etc. Notez que, dans cet exemple, la valeur initiale de A devant être rentrée explicitement pour lancer la récursion, on n'utilise pas L(A:...).

Les boucles indéfinies

Telle qu'elle est décrite dans le manuel, l'instruction de contrôle de boucle SIGMA, dont la syntaxe est :

`SIGMA(<compteur> : <Cinit> : <Cfin> : <pas>)`

ne permet d'envisager que des boucles définies, c'est-à-dire pour lesquelles la sortie de boucle se fait lorsque le compteur atteint ou dépasse la valeur finale, fixe.

Or, le manuel d'application décrit comment on peut simuler une structure DO...UNTIL ou DO...WHILE, en intervenant sur la valeur de *<Cfin>*, chose impossible jusqu'ici. Ou même en modifiant les autres paramètres de SIGMA... Je ne vous en dis pas davantage, car sinon, je vais finir par pomper tout le bouquin !

Autres richesses cachées

Le manuel indique aussi comment réorganiser les menus et l'ordre des variables, comment créer des menus à choix multiples au moyen de clauses IF, et bien d'autres choses... D'ailleurs, les exemples donnés en seconde partie, même s'ils ne vous paraissent pas directement utiles, sont instructifs et méritent d'être analysés en détail.

Ainsi, ce manuel montre, beaucoup mieux que le manuel de base, comment on peut avoir plusieurs équations dans le menu SOLVE (désignées par leur nom, et accessibles comme les éléments d'une liste, au moyen des deux touches fléchées), et repasser des valeurs de l'une à l'autre. Voici un exemple, combinant choix et équations multiples, passations de valeurs, etc. :

```
CONVL1: IF S(NM) OR S(KM):NMx1.8525-KM:MILx1.62-KM)

CONVL2: IF(S(M) OR S(FT):Mx3.281-FT:INx25.4-MM)
```

CONVL1 permet les passages de km en Nautiques ou en Statute Miles ; CONVL2 donne les conversions de pieds en mm ou en pouces. Pour passer des Miles terrestres aux pieds, on se servira de l'intermédiaire métrique : km dans CONVL1, multipliés par 1000, puis réinjectés par [M] dans le menu de CONVL2. C'est presque aussi bien qu'avec la HP-28 !

En conclusion

Même si votre domaine d'intérêt n'est pas la résolution de triangles (très bel exemple de choix multiples, soit dit en passant) ou les opérations sur des nombres complexes, ce petit manuel vaut l'achat. Son anglais n'est pas difficile, il est très clair, et constitue un complément indispensable si l'on veut aller un peu plus loin.

D'une façon générale, d'ailleurs, ayant eu entre les mains d'autres manuels d'application, je les ai trouvés attractifs, clairs et bien faits... Ainsi, dans celui qui s'intitule *Marketing and Sales*, on trouvera un moyen simple de simuler les fonctions IRR et NPV sur flux inégaux (ces flux étant tout simplement placés dans une liste STAT, donc modifiables individuellement sans aucune difficulté), fonctions dont l'absence dans le menu TVM de ma HP-27S m'avait vraiment manqué.

Ne vous intoxiquez quand-même pas trop aux SOLVEants!

Eric Gengoux (108)

COURRIER DU COEUR

Sébastien Lalande
12 rue de Seine
78920 Croissy sur Seine
Tél : (1) 39 76 27 41

Vend :
HP28C version 1BB, 66 Ko, accélérée, avec connecteur in/out, cable HP28-Apple II+, manuels d'origine ainsi que quatre manuels complémentaires : Algebra & college maths, Vectors & matrices, Probability & statistics, calculus. Le tout en parfait état. Prix : 4000 FF.

Thomas Rouyer
398 avenue Maurice Dauvergne
77350 Le Mée sur Seine

Vend :
PC-1500A Sharp, état neuf (20/07/88) : valeur 1700 FF, vendu 1200 FF.

Pierre Bourgot
2 square Castiglione
78150 Le Chesnay

Vend :
Lecteur de cassettes HP82161A : 1500 FF, imprimante thermique HP82162A : 1000 FF, convertisseur HPIL HP82166A : 1000 FF.
Livres *Au fond de la HP-41* de Jean-Daniel Dodin : 50 FF, *Autour de la boucle* de Janick Taillandier : 50 FF.

Hervé Thévenon
10 rue Coypel
75013 Paris
Tél : (1) 43 36 03 57 (9 à 20h en semaine)

Vend :
HP-28C, état neuf : 1200 FF avec housse et deux mois de garantie.

Fabien Navarro
12 rue Victor Hugo
91100 Corbeil-Essonnes
Tél : () 64 96 51 77 (après 17h)

Vend :
HP-28C : 1200 FF.

Ariel Chemouny
2, rue de la Buttes aux Bergers
95470 St Witz
Tél : 34 68 39 73

Vend :
HP-71 : 2900 FF, module HPIL : 700 FF, module Translator Pac : 390 FF.
HP-41CV : 990 FF, module HPIL : 700 FF, lecteur de codes barres : 680 FF, MLDL 8K Rom 24 Eprom avec Eprom Eramco : prix à débattre. Module XFonctions : 310 FF, module horloge : 310 FF, module Stress Analysis : 175 FF, 40 cartes magnétiques : 40 FF.
Interface HPIL/HPIB (HP82169) : 1900 FF, lecteur de cassettes (HP82161A) : 2700 FF, cassettes digitales : 50 FF pièce, table traçante HP7475 HPIB : prix à débattre, et cables HPIL 5 mètres : 50 FF.

HP28

T. J. Affinito

Système 411 Version 3 *6*

SYSTEME 411 VERSION 3

Cet article de gestion d'arborescence pour HP-28S nous est parvenu des U.S.A. par le réseau UNIX, reliant plusieurs milliers d'ordinateurs de par le monde. Thomas Affinito, son auteur, a donné l'autorisation aux Clubs de le publier. Paul Courbis l'a donc traduit, et c'est avec plaisir que nous vous le présentons ce mois-ci.

INTRODUCTION

Une mémoire à structure de stockage arborescente peut fournir une méthode agréable pour organiser la toujours croissante myriade de variables qu'un utilisateur actif crée pour son usage personnel. La plupart du temps, l'utilisateur passe trop de temps à utiliser ORDER, à monter et descendre les branches de l'arborescence pour chercher à retrouver à quel endroit sont situées certaines variables. Mon premier projet de programmation HP28S était de dompter cette hiérarchie à l'aide d'un système basé au sommet de l'arborescence de manière à libérer l'utilisateur des commandes transversales.

Mon idée était de mettre en oeuvre un concept d'espace de travail (comme en APL) avec des propriétés spécifiques :

1- l'utilisateur doit disposer de commandes simples pour créer (BUILD) et détruire (CRUSH) automatiquement un espace de travail.

2- l'espace de travail doit se comporter comme un banc d'essai pour le développement et les essais de nouveaux programmes, de nouvelles idées.

3- les programmes achevés et les variables doivent pouvoir être supprimés du menu USER (HIDE) et rangés dans un menu parent dans lequel il sera toujours accessible ; de plus les programmes, les variables et, plus généralement, les utilitaires pourront toujours être repoussés dans un menu parent plus élevé qui sera visible depuis tous les espaces de travail créés (STALL, abréviation de *install*).

4- les commandes du système principal peuvent être placées dans le menu *custom* pour en faciliter l'usage ; de plus, l'utilisateur peut créer des labels spécifiques qui seront toujours chargés quand l'espace de travail particulier sera activé (SLAB) ; les labels généraux (présents pour tous les espaces de travail) sont aussi possibles (GLAB) ; ainsi l'utilisateur peut-il cacher (HIDE) ou installer (STALL) tous ses programmes, et nommer de manière locale ou globale (SLAB ou GLAB) les applications principales, ce qui laissera tout en dehors de l'espace USER, gardera tous les utilitaires opaques et donnera une certaine transparence aux applications.

5- l'utilisateur peut passer d'un espace de travail à un autre avec des commandes simples ; les passages transversaux sont totalement éliminés.

6- le système ne prend pas trop de mémoire.

Le système à été minutieusement débogué et est plein d'utilitaires intéressants ; il est facilement personnalisable et occupe à peine 2 Ko (un faible prix à payer avec la 28S). Voici le système que j'utilise pour toute ma programmation. Je serai intéressé par tous les commentaires, toutes les critiques et tous les compléments à ce système...

VERSION 3 ?

Après avoir programmé sur la 28C pendant 6 mois, j'ai tressailli quand est sortie la 28S et je l'ai immédiatement achetée. 411 fut créé pour diriger mon style d'écriture de programmes. Après avoir lu le livre de Wlodek, j'ai réalisé que je devais placer mon système dans un sous-menu du menu HOME pour pouvoir conserver PEEK et POKE dans le menu principal (*). Ainsi naquit la version 2.

Il y a trois jours, j'ai acheté le livre de Bill Wickes, *HP 28 Insights* que je recommande vivement. C'est une extension des manuels, avec d'intéressantes suggestions... Certains de ces programmes étaient similaires aux miens : de manière à rendre 411 plus accessible, j'ai changé certains de mes programmes pour reprendre ses noms, ses algorithmes à la place des miens. A savoir :

UP page 83 de son livre ainsi que dans le manuel HP. Mon programme UP laissait dans la pile le chemin qu'il venait de quitter... Ainsi mes programmes avaient des paires de UP et de DOWN (qui exécutait le chemin dans la pile) quand j'effectuais des chemins transversaux.

DOPATH page 84. Mon programme faisant cela s'appelait DOWN mais j'ai changé le nom pour être en accord avec Bill ; Mon algorithme n'est pas le même... Le mien utilise des utilitaires non-inclus dans son livre et est plus général car il exécute une liste ou un nom.

Purge et Clusr. J'ai un seul programme (en fait une chaîne) qui fait un nettoyage complet d'un menu (appelé XCL pour *eXtra CLear*) Le Purge de Bill fait la même chose, ainsi ai-je utilisé ses noms et algorithmes... Avec l'exception que Clusr est placé dans une chaîne d'objets. J'ai fait cela comme protection ; avoir une touche qui peut vider toute la

mémoire est trop dangereux ! J'ai souvent perdu des choses lors de l'écriture de XCL ! Ainsi il faut faire Clusr STR→ pour lancer ce programme, ce qui ne peut être fait accidentellement par quelqu'un qui tâtonne...

QUE FAIRE ?

La fin de cet article contient tous les listings de programmes et les valeurs des variables par défaut. Il n'y a qu'à les taper ! Pour prévenir toute confusion, il faut savoir que :

- Dans le menu HOME, on stocke le programme ON411, et on crée le sous-menu →411 ; ainsi le menu HOME est vide pour accueillir les programmes devant résider en haut de Ram pour les programmes de Wlodek(*).

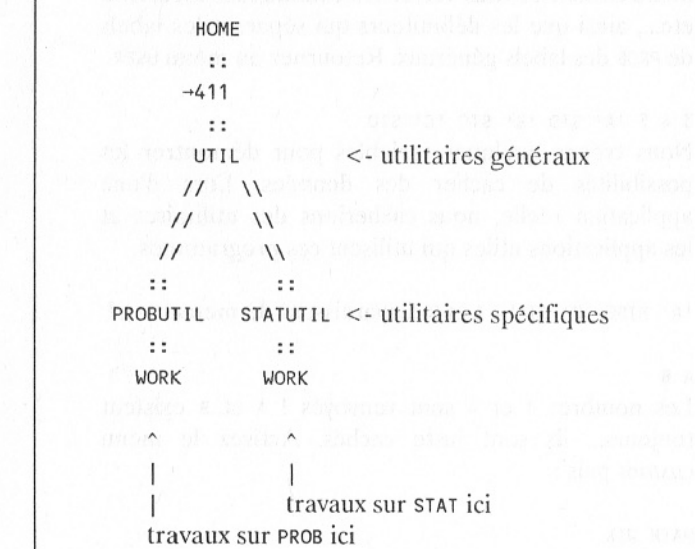
- Dans le menu →411 vont toutes les autres choses mais vous devez créer un sous-menu appelé ici UTIL et y stocker GU.LS.

- Les "" qui apparaissent dans certains programmes ne sont que deux guillemets que j'utilise comme séparateurs dans les menus *custom*.

- Les variables sont listées dans l'ordre dans lequel elles apparaissent dans mon espace USER : des commandes les plus importantes aux moins... Pour obtenir le même ordre : entrer les variables dans l'ordre inverse...

QUE FAIT LE PROGRAMME ?

Supposons que vous ayez créé les deux espaces de travail PROB et STAT (la manière de les créer est décrite dans la section suivante). 411 crée l'arborescence suivante pour vous :



411 automatise la création, la destruction, et les déplacement transversaux de cette structure. Il automatise aussi les mouvements de programmes et d'objets achevés dans les aires d'utilitaires spécifiques ou dans la zone d'utilitaires généraux. Il contrôle aussi les changements spécifiques dans le menu *custom* pour que les applications générales et spécifiques puissent rester transparentes en étant nommées dans ce menu.

Note : Ce type de structure est la même que celle suggérée par Wickes à la page 77 de son livre. Comme je n'ai fait que feuilleter le livre, j'ai eu peur qu'il contienne des programmes faisant la même chose que 411, rendant mes efforts inutiles. Fort heureusement, tel n'est pas le cas ! Ainsi j'espère que cet article sera un supplément et une illustration des idées présentées dans son livre, et qu'il aidera tous les utilisateurs à diriger leur arborescence d'une manière approuvée par le Maître...

COMMENT L'UTILISER ?

Pour commencer : MAIN, ON411

MAIN crée un menu *custom* de tous les espaces de travail, suivi par les commandes EXIT, BUILD et CRUSH. Depuis ce menu, on peut aller dans n'importe quel espace de travail en pressant le nom de cet espace... On peut aussi utiliser les trois commandes précitées.

MAIN est aussi valable dans le menu *custom* lorsqu'on est dans un espace de travail : presser MAIN est généralement fait quand on a fini d'utiliser un espace de travail et que l'on veut retourner au menu principal pour voir les autres espaces où l'on peut se rendre, ou pour utiliser BUILD, CRUSH ou EXIT...

Si vous êtes dans le menu HOME, activer 411 par ON411 vous enverra dans le sous-menu approprié et exécutera MAIN pour vous ; après être sorti de 411 grâce à EXIT et être retourné au menu HOME, ON411 est placé dans le menu *custom* par commodité.

Quitter 411 **:** EXIT

EXIT vous renvoie au menu HOME et affiche un menu *custom* contenant ON411.

Quitter un espace de travail : MAIN, WOFF

MAIN vous ramène au menu principal. WOFF (*Workspace OFF*) est utilisé à la fin du travail pour revenir au menu principal, nettoyer la pile et ré-initialiser les drapeaux à votre convenance.

Construire un espace de travail : BUILD

Taper le nom de votre espace, puis BUILD. Après quoi le menu de la commande MAIN sera activé, avec un nom pour votre nouvel espace. Retourner au menu MAIN par le menu *custom* en pressant MAIN.

Détruire un vieil espace : CRUSH

Taper le nom d'un espace de travail existant, puis CRUSH. Toutes les variables et sous-menus associés à cet espace seront détruits et le nom de l'espace sera retiré du menu *custom*.

Cacher des variables : HIDE, STALL, SEEK

Pour avoir des variables exécutables dans votre espace de travail, mais invisibles dans le menu USER, mettre tous leurs noms dans une liste et exécuter HIDE depuis votre espace de travail.
Pour avoir des variables exécutables depuis tous les espaces : utiliser STALL au lieu de HIDE.
Pour retrouver des variables cachées par HIDE ou STALL, utiliser une liste de noms de variables puis SEEK. SEEK range les variables récupérées dans le menu courant mais ne change aucun label spécifique ou local : vous devez utiliser LABOUT.

Contrôler le menu custom : SLAB, GLAB, LABOUT

SLAB prend une liste et ajoute les objets qu'elle contient à la liste spécifique de commandes. Quand vous êtes dans cet espace de travail, vous verrez toutes les commandes entrées par SLAB en première position dans le menu *custom*. Ceci est généralement utilisé ainsi : imaginons que j'ai un programme d'application spécifique appelé APP qui utilise les utilitaires spécifiques U1 U2 et U3. Je l'ai débogué et il est prêt à être utilisé : on le cache du menu USER par {APP U1 U2 U3} HIDE, puis on rend APP visible par {APP} SLAB (vous pouvez aussi taper 'APP' SLAB. HIDE, STALL, SEEK, SLAB, GLAB et LABOUT peuvent prendre un nom au lieu d'une liste à un élément. MOVE, DELL et VL→S aussi).
GLAB fonctionne de manière similaire pour une liste de commande globale qui affectera tous les espaces de travail.
LABOUT enlève une liste des listes de commandes générales et spécifiques.

Déplacer des variables : MOVE

Taper une liste de noms de variables, puis le nom de l'espace de travail où les envoyer puis MOVE. L'avantage de cette séquence sur la séquence { *<liste de nom de variables>* } VL→S 'nom d'un autre espace' EVAL S→VR est que la seconde séquence entraînera l'exécution de SWON et WON qui peuvent être modifiées pour faire des choses étranges (comme RESET) ce qui pourrait vider la pile. Note : MOVE ne détruit pas les copies originales des variables.

EXEMPLES ILLUSTRES

Vous avez tapé 411... Allons-y !

HOME

Nous sommes au niveau supérieur, ainsi si nous voulons taper PEEK et POKE, nous le pouvons. Si vous n'êtes pas familier de ce genre de chose dites adieu au menu HOME car vous n'aurez plus besoin d'y revenir(*).

ON411

Nous sommes à présent dans le système. Il n'y a pas encore d'espace de travail, mais les trois commandes sont présentes à côté du délimiteur qui sépare les noms des espaces de travail de ceux des commandes. Créons un espace de travail : PROB qui contiendra tous mes programmes de probabilité et de travail, ML pour quelques trucs en langage machine, TEST pour les brouillons et WIX pour les programmes du livre de Bill Wickes.

'PROB' BUILD 'ML' BUILD 'WIX' BUILD 'TEST' BUILD
Après avoir effectué ces commandes, les noms des espaces de travail apparaîtront dans le menu *custom*. Faisons un peu de probabilités...

PROB
Nous sommes soudainement placés dans le menu USER vide de notre nouvel espace de travail. Activez le menu *custom* et vous verrez les commandes HIDE, MAIN etc..., ainsi que les délimiteurs qui séparent les labels de PROB des labels généraux. Retournez au menu USER.

3 4 5 'A' STO 'B' STO 'C' STO
Nous créons quelques variables pour démontrer les possibilités de cacher des données. Lors d'une application réelle, nous cacherions des utilitaires et les applications utiles qui utilisent ces programmes.

'A' HIDE 'B' STALL A et B disparaissent du menu USER !

A B
Les nombres 1 et 4 sont renvoyés ! A et B existent toujours... ils sont juste cachés. Activez le menu *custom* puis :

MAIN WIX

Une nouvelle fois un menu vide nous fait face... Nous sommes dans l'espace de travail WIX...

```
A B C
```

A renvoie A car l'espace WIX ne peut pas voir A qui a été caché à l'aide de HIDE par PROB ; de même WIX ne peut voir 'C' qui existe dans PROB. Par contre WIX peut accéder à B qui a été caché comme objet général par STALL. Nous avons appris trois choses : 1) les espaces de travail sont séparés et indépendants les uns des autres, 2) HIDE cache les objets qui restent spécifiques à leur espace de travail, 3) STALL les cache tout en les rendant exécutables par tous les espaces de travail. Retournons à PROB par la voie rapide :

```
PROB
```

Le menu user PROB réapparaît. Maintenant supposons que A et B soient des applications importantes et que nous voulons les exécuter par une touche depuis PROB.

```
{A B} SLAB
```

Le menu *custom* est activé et vous voyez A et B apparaître sur les touches. Ils apparaîtront toujours comme touche dans PROB jusqu'à ce que vous les enleviez ou que vous les détruisiez PROB par CRUSH.

```
WIX
```

Allez au menu *custom* et vous ne trouverez ni la touche A, ni la touche B... SLAB n'effectue que des appellations spécifiques à un espace de travail.

```
'B' LABOUT
```

Le menu *custom* est activé et le nom B a disparu. La variable existe toujours.

```
'B' GLAB
```

Remarquez comme la position de B a changé... Il est dans les variables globales.

```
WIX
```

Allez au menu *custom*. B y est ! Les labels globaux apparaissent dans tous les espaces de travail. Toute chose pouvant être mise dans une liste peut devenir un label. Il faut remarquer que le fait que B soit un label global ne garantit pas le fait que B soit une variable globale. L'affectation de labels (GLAB, SLAB, LABOUT) est indépendante du masquage des variables (HIDE, STALL, SEEK). Nettoyons à présent la mémoire :

```
'B' LABOUT 'B' SEEK
```

Ceci supprime le label global et l'objet global, qui continuerait à exister si on s'était contenté de détruire l'espace de travail spécifique. C'est une bonne chose qu'ils demeurent... nous ne les ramenons à un espace spécifique que pour faire la démonstration de la destruction d'une telle variable.

```
WOFF 'PROB' CRUSH 'WIX' CRUSH 'TEST' CRUSH 'ML' CRUSH
```

La meilleure façon d'apprendre à se servir de 411 est de créer d'autres exemples et de jouer avec, par vous-même. Il est aisé d'apprendre à utiliser 411 si vous en prenez le temps, temps que vous regagnerez grâce aux possibilités d'organisation qu'il fournit...

NOTES ET RESUMES

Après avoir appris les commandes, vous vous êtes sans doute rendu compte qu'il était plus facile de taper les commandes que de passer par le menu *custom*. Ce menu *custom* est là pour l'exécution en une touche des utilitaires que vous avez sélectionnés.

Pour aller d'un espace de travail à un autre, il n'est nul besoin de passer par MAIN ou par WOFF... Il suffit de taper le nom de cet espace et vous y êtes... Le menu MAIN est un aide-mémoire pour se souvenir de la liste de ces noms.

Toutes les commandes fonctionneront toujours si vous avez créé un sous-menu dans votre zone de travail, ou un sous-menu dans votre sous-menu, etc. Le seul problème est que si vous cachez, par HIDE ou par STALL, des variables qui ne sont pas dans le menu courant, dans le menu de travail ou dans le menu des utilitaires spécifiques, alors elles seront cachées correctement mais leur copie originale continuera à exister dans le sous-menu où elles ont été créées. Ce problème ne se produira jamais si vous utilisez correctement cet utilitaire (le seul besoin de sous-menu est pour le stockage simultané de solutions obtenues par SOLVER d'une équation donnée.).

BUILD et CRUSH vous ramènent au menu principal. Ces commandes sont supposées être peu fréquentes... Les espaces de travail sont prévus pour que l'on y travaille...

Pour montrer les possibilités de 411 à vos amis, créez l'espace de travail DEMO où vous pourrez faire tous vos essais... Après quoi il sera facile de tout détruire par WOFF 'DEMO' CRUSH!

UTILITAIRES UTILES

BOOP beep d'erreur standard.

S→N effectue une conversion chaîne→nom ou algebraic (si le contenu de la chaîne le permet)

O→S Prend un objet délimité et le renvoie dans une chaîne sans les délimiteurs... Fonctionne pour les noms, les listes, les vecteurs.

Purge détruit toute variable... Même un sous-menu non vide.

Clusr détruit toutes les variables dans le menu courant, y compris les sous menus non vides. Il doit être exécuté avec STR→.

DELL prend une liste ou un nom au niveau 2, et une liste ou un nom au niveau 1 et supprime toute apparition des objets du niveau 2 de la liste du niveau 1. Rien n'est renvoyé dans la pile.

RESET vide la pile et redonne les valeurs par défaut aux drapeaux.

VL→S prend une liste de variables (ou un nom) et rappelle chacune des variables dans la pile, le contenu étant suivi par le nom. A la fin, le nombre de variables rappelées est placé dans la pile.

S→VR prend la sortie de VL→S et le transforme en variables dans le menu courant. La combinaison de VL→S et S→VR rend facile le déplacement de programmes depuis le menu HOME dans un espace de travail et vice versa... bien que ceci soit rare.

UP passe au menu père.

DOPATH prend une liste de noms et les exécute. Ceci est généralement utilisé pour revenir à un menu dont la position a été sauvée dans la pile grâce à la commande PATH.

PERSONNALISATION DU SYSTEME

Je m'excuse : j'aurai dû commenter mes programmes mais je voulais poster ceci aussi vite que possible, ayant peur de commencer une version 4... Mes programmes ne sont pas faits que d'astuces, et vous n'êtes pas obligés de les comprendre pour les utiliser. Voici quelques solutions simples pour personnaliser 411 :

Pour changer le comportement en entrée de tout espace de travail : modifiez WON (*Workspace ON*) : tout ce que fait WON pour le moment est de créer le menu *custom* pour l'espace de travail. Si on veut ajouter un beep à l'entrée il suffit de remplacer WON par « UCML BOOP ».

Pour changer le comportement en entrée d'un espace précis : modifier SWON (WON spécifique) qui est situé dans le menu utilitaires au dessus de la zone de travail. Le SWON standard ne fait qu'appeler WON. Imaginons que vous êtes dans l'espace PROB et que vous vouliez effectuer un RESET : il faut changer SWON en « WON RESET » (faire UP pour pouvoir modifier SWON).

Si vous n'aimez pas les "" comme délimiteurs de menus : enlevez les "" ainsi que les + qui les suivent dans MAIN et ACML.

Il y a des commandes très utiles en sortie (EXIT) : modifiez la liste dans EXIT pour inclure les commandes que vous désirez (par exemple ON411).

Si l'état standard des drapeaux est différent du mien : mettez le calculateur dans l'état souhaité puis RCLF 'FVFLG' STO (dans →411). RESET et WOFF mettront dorénavant la machine dans cet état standard.

Vous voulez que vos programmes modifient le menu *custom* puis le remettent plus tard à son état standard : il suffit de mettre dans votre programme la séquence CM.M MENU.

Accéder directement au menu des labels : la variable SU.LS (dans le menu père du menu WORK, ou menu de travail) contient ces labels. GU.LS (dans le menu UTIL) contient la liste des labels globaux.

411

Menu HOME

→411 directory

ON411 active 411
« HOME →411 MAIN »

Menu →411

UTIL directory contenant GU.LS

GU.LS general utility list
{}

WK.LS liste des espaces de travail
{}

CM.LS liste des commandes du menu MAIN
{ EXIT BUILD CRUSH }

CU.LS liste des commandes utilitaires
{ MAIN WOFF "" HIDE SLAB "" STALL GLAB "" SEEK LABOUT "" MOVE }

FVFLG état standard des drapeaux

```
#8081FFD40000000h
```

MAIN aller au menu principal

```
« HOME →411 WK.LS "" + CM.LS + MENU »
```

BUILD créer un espace de travail
(entrée : un nom)

```
« HOME →411 DUP O→S "UTIL" + → n n.s
  « IF WK.LS n POS
    THEN "Name In Use" 1 DISP BOOP
    ELSE 'WK.LS' DUP EVAL n + SWAP STO
      "«UTIL " N.S + " WORK SWON»" +
      STR→ n STO UTIL n.s DUP S→N
      CRDIR « WON » SWON STO {} SU.LS
      STO MAIN
    END
  »
»
```

CRUSH détruit un espace de travail
(entrée : un nom)

```
« HOME →411 DUP O→S "UTIL" + → n n.s
  « IF WK.LS n POS
    THEN n PURGE n 'WK.LS' DELL UTIL
      n.s S→N Purge MAIN
    END
  »
»
```

EXIT quitter 411

```
« HOME { "" "" ON411 "" "" "" } MENU 23 MENU »
```

HIDE cacher le variables de la liste
(entrée : liste de noms)

```
« → ls
  « PATH ls VL→S ls PURGE WORK
    ls PURGE UP S→VR DOPATH
  »
»
```

STALL cacher des variables générales
(entrée : liste de noms)

```
« → ls
  « PATH ls VL→S ls PURGE WORK ls PURGE
    UP ls PURGE UTIL S→VR  DOPATH
  »
»
```

SEEK retrouver des variables
(entrée : liste de noms)

```
« → ls
  « ls VL→S PATH WORK ls PURGE UP ls PURGE
    UTIL ls PURGE DOPATH S→VR
  »
»
```

SLAB mettre une liste dans un menu spécifique
(entrée : liste)

```
« SU.LS + PATH SWAP WORK UP 'SU.LS' STO
  DOPATH UCML
»
```

GLAB mettre une liste dans un menu général
(entrée : liste)

```
« GU.LS + PATH SWAP UTIL 'GU.LS' STO
  DOPATH UCML
»
```

LABOUT retirer les labels de la liste des menus spécifiques et généraux.
(entrée : liste)

```
« → ls
  « PATH WORK UP ls 'SU.LS' DELL
    UTIL ls 'GU.LS' DELL DOPATH UCML
  »
»
```

MOVE déplacer les variables de la liste dans l'espace de travail spécifié.
(entrée : liste des noms, nom espace)

```
« → nm
  « PATH SWAP VL→S UTIL nm O→S
    "UTIL" + STR→ WORK S→VR DOPATH
  »
»
```

WON activation de l'espace de travail

```
« UCML 23 MENU »
```

WOFF désactivation de l'espace de travail

```
« RESET MAIN »
```

UCML crée le menu *custom*

```
« PATH WORK UP SU.LS "" + GU.LS + "" +
  CU.LS + DUP 'CM.M' STO MENU DOPATH
»
```

UP aller au menu père

```
« PATH DUP SIZE 1 - 1 MAX GET EVAL »
```

DOPATH exécute une liste de noms
(entrée : liste de nom ou un nom)

```
« O→S STR→ »
```

VL→S liste de variables -> variable dans la pile
(entrée : liste de noms)
(sortie : obj, nom, obj ... nom, obj, nom, #obj)

```
« {} + → ls
  « 1 ls SIZE
    FOR j ls j GET DUP RCL SWAP NEXT
    ls SIZE
  »
»
```

S→VR pile de variables --> variables
(entrée :obj,nom,obj...nom,obj,nom,#obj)

```
« 1 SWAP START STO NEXT »
```

RESET vide la pile, initialise les modes

```
« CLEAR FVFLG STOF »
```

DELL enlève des objets d'une liste donnée par son nom
(entrée : {obj obj...obj } 'nom liste'
ou : obj 'nom liste')

```
« SWAP {} + → lsnm erls
  « 1 erls SIZE
    FOR j
      erls j GET lsnm EVAL
      WHILE
        DUP 3 PICK POS DUP
      REPEAT
        SWAP LIST→ DUP DUP 3 + ROLL - 2
        + ROLL DROP 1 - →LIST
      END
      DROP lsnm STO DROP
    NEXT
  »
»
```

Purge détruit toutes les variables
(entrée : nom)

```
« 31 SF IFERR PURGE
        THEN DUP EVAL Clusr STR→ UP PURGE
        END
»
```

Clusr vide un menu entier

```
"VARS LIST→ 1 SWAP START Purge NEXT"
```

O→S objet délimité --> chaîne
(exemples : 'nom' --> "nom" ou {liste} --> "liste")

```
« →STR DUP SIZE 1 - 2 SWAP SUB»
```

S→N chaîne --> nom
(entrée : chaîne)

```
« "'" SWAP + STR→ »
```

BOOP son d'erreur

```
« 1400 .07 BEEP »
```

J'espère que vous apprécierez cet utilitaire autant que moi...

Thomas J. Affinito
traduction Paul Courbis (392)

(*) : Note de la Rédaction : les restrictions citées par Thomas ne sont pas applicables avec les programmes publiés par Paul Courbis. Voir les anciens numéros de JPC, et plus particulièrement JPC 51 et JPC 56.

HP75

J.Y. Hervé

TRIEZ VOS TABLEAUX

Ce fichier Lex de 239 octets permet le tri d'un tableau de variables numériques à une ou deux dimensions. Il fournit un nouveau mot-clé directement utilisable dans un programme Basic, dont la syntaxe est :

```
TRITAB V(I),N  ou
TRITAB V(I,J),N
```

N représente le nombre d'éléments à trier dans le tableau numérique lui-même, en partant de l'élément V(I) s'il s'agit d'un tableau à une dimension, ou de V(I,J) s'il s'agit d'un tableau à deux dimensions.

Dans sa version actuelle, TRILEX comporte encore un certain nombre de lacunes : en particulier, il n'effectue pas le tri de tableaux déclarés en SHORT, mais seulement de ceux déclarés en REAL ou en INTEGER. Il n'effectue également pour l'instant que des tris ascendants. Ces améliorations seront développées ultérieurement.

Vérification du nombre d'éléments

Le problème majeur reste pour l'instant celui du contrôle de la validité de la valeur donnée à N. En effet, dans la présente version du Lex, aucun contrôle de la valeur de N n'est effectué, si bien qu'une variable peut tout aussi bien être triée sur un nombre d'éléments beaucoup plus grand que son dimensionnement déclaré, ce qui entraîne un débordement en dehors des limites du tableau et les ravages que l'on imagine facilement en Ram...

Vérification de la variable de départ

La variable V(I) ou V(I,J) par laquelle va débuter le tri est vérifiée à plusieurs reprises dans le Lex :

- Si cette variable ne correspond pas à un tableau numérique à une ou deux dimensions, l'erreur 87 "Bad expression" est générée. Cette même erreur apparaît également si la syntaxe de TRITAB n'est pas respectée.

- Le Lex vérifie également le type de la variable : le tri sera orienté vers les sous-programmes spécifiques à chacun des deux types REAL et INTEGER. Dans la présente version, un tableau déclaré SHORT générera l'erreur 89 "Bad parameter".

- D'autre part, l'erreur 27 "Invalid subscript" apparaîtra si le ou les indices fournis sont en dehors des limites du tableau.

Précautions d'utilisation

Pour utiliser cette version de TRILEX, il apparaît donc préférable de prendre systématiquement les précautions suivantes :

- Vérifier l'OPTION BASE choisie, surtout lorsqu'on utilise des tableaux à deux dimensions ;
- Vérifier si le tableau est correctement dimensionné ;
- S'assurer de l'affectation des valeurs pour chaque élément à trier, sachant qu'un élément non affecté sera assimilé à zéro et trié en conséquence ;
- Vérifier enfin que le nombre N d'éléments à trier à partir de l'élément de départ n'entraînera pas de dépassement hors des limites du tableau. Pour un tableau à une seule dimension, il faudra :

```
N ≤ DIM - I + 1
```

et pour un tableau à deux dimensions, il faudra :

```
N ≤ (DIM-I+1)*(DIM-OPTBAS+1)+OPTBAS-J
```

Performances du Lex

En remplissant les tableaux avec des valeurs aléatoires, on obtient les performances moyennes suivantes :

Nombre d'él. à trier	Tableau INTEGER	Tableau REAL
100	0.5 s	3 s
200	2 s	12 s
300	5 s	26 s
400	8 s	47 s
500	12 s	73 s

Pour un tableau INTEGER, le Lex emploie un tri à bulles classique ; la durée maximale du tri sera atteinte lorsque tous les éléments seront déjà classés en ordre décroissant.

Pour les tableaux REAL, le Lex utilise également un tri à bulles, mais en intervenant à trois niveaux successifs : il teste d'abord le signe du nombre, puis son exposant, et enfin sa mantisse. La durée maximale sera, là également, atteinte lorsque tous les éléments seront déjà classés avant le tri.

Le tableau suivant montre les résultats obtenus en secondes pour trier 100 éléments, selon leur ordre initial de classement :

	Ordre croissant	Ordre aléatoire	Ordre décroissant
INTEGER	0,37	0,53	0,72
REAL	4,52	2,87	3,33

Le programme de *benchmark* utilisé pour faire ce test était le suivant :

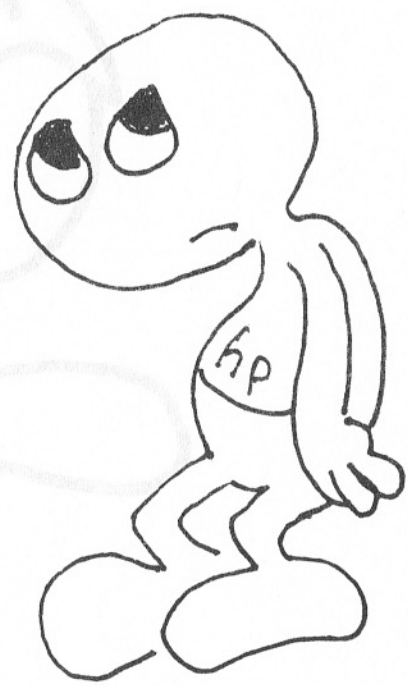
```
 10 OPTION BASE 1
 20 DIM V(500)
 30 INPUT "N=","100";N
 40 FOR I=1 TO N
 50   V(I)=RND*1000-500
 60 NEXT I
 70 T=TIME
 80 TRITAB V(1),N
 90 T=TIME-T @ DISP T
100 FOR I=1 TO N @ PRINT V(I); @ NEXT I
```

Jean-Yves Hervé (450)


```
TRILEX   L 239  Bytes    ROM ID: 120

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

  1  58 B2 DD 00 8D 4C 4D 7A B6 A6 54 52  8E
  2  49 4C 45 58 20 20 78 00 0A 00 12 00  08
  3  0C 00 19 00 1C 00 49 00 1D 00 FF FF  A7
  4  54 52 49 54 41 C2 FF FF 61 17 9E 6D  CC
  5  A9 B4 78 00 06 E5 4C E4 CE A3 0C 4C  BE
  6  C8 02 F6 13 CE BC 16 F9 0E CE 66 10  C3
  7  CE 70 0E F9 06 6C 06 E3 0A E5 9E 6C  9E
  8  06 E3 CE 99 4C 57 A1 CE 8B 3E 12 A3  E5
  9  61 0A E3 54 CE 12 1F 99 66 86 98 C8  8B
 10  04 F7 25 C8 06 F7 77 52 8B F4 1C 10  5E
 11  A3 65 14 E1 56 A1 50 8B F4 F1 6D 16  3D
 12  E1 25 C1 F5 F5 14 E7 65 16 E7 E1 14  0A
 13  E1 F0 EB 9E 52 8B F4 FB 10 A3 70 14  64
 14  E1 56 A1 CE D0 3E 38 A3 98 50 8B F4  FC
 15  EB 70 16 E1 CE D0 3E 68 38 A1 CE DF  23
 16  3C 5A 84 5B 84 1A C0 F4 E7 F6 20 90  5A
 17  F7 0D 5A 1E A1 5C A3 1A A1 62 3A A1  19
 18  6A 32 A1 5E 1C C1 F4 0B F6 CE 67 87  2F
 19  6F 87 61 29 C1 F5 C5 70 14 E7 78 16  F9
 20  E7 E1 14 E1 F0 BA CE 99 4C 59 9E     18
 21  7309
```




hp

FORTH

BASIC

LA RECETTE DES SPAGHETTI
OU
L'ART DU SYSTEME EXPERT

Les systèmes experts sont à l'intelligence artificielle ce que le sable est au mortier, une partie indissociable du tout. Le but de cette présentation est de montrer comment est construit un logiciel pompeusement baptisé **générateur de systèmes experts**. Basé sur un moteur de type zéro, diagnostiqueur, monotone avec mécanisme de chaînage arrière en profondeur d'abord et détectant les bases incohérentes (ouf !), ce petit programme a pour vocation de « raisonner habilement à propos de tâches dont on pense qu'elles requièrent une expertise humaine considérable » (dixit les textes officiels).

Mais que viennent faire les nouilles dans cette histoire ? Tout simplement, les pâtes fraîches à l'italienne sont au centre d'une expertise et fournissent un bon exemple de *base de connaissance* à un système expert.

Système Expert, qu'est-ce ?

La vocation de ce type de programme s'exprime fort mal dans la définition officielle. En fait, elle est de remplacer ou d'assister l'Homme dans des domaines où existe une expertise humaine :
- insuffisamment structurée pour constituer une méthode précise, sûre, complète et directement transposable sur ordinateur ;
- sujette à révision ou complément au fil de l'expérience dans les domaines des sciences de la vie (diagnostics et traitements médicaux par exemple), des sciences humaines et sociales (orientation scolaire...).

Dans tout les cas, il y a une formalisation minimum du savoir sous forme d'unités de connaissance ou *règles*. Ces règles représentent des classes de situations particulières et fournissent une série de raisonnements lorsque combinées.

Quelques exemples de règles ? En voici, en voila :
- **si** animal vole **et si** animal pond des oeufs **alors** animal est oiseau.
- **si** sel **et si** semoule **et si** jaune d'oeuf **et si** eau **alors** pâtes fraîches.

La phrase « animal vole » est appelée *fait* et la règle combine des faits pour induire un nouveau fait (la conclusion de la règle est « animal est oiseau »).

Les fondements du Système Expert

Il ne s'agit pas ici de reproduire l'historique du système expert sachant qu'on la retrouve dans de nombreux bouquins, mais de montrer l'idée simple sous-jacente au procédé. La méthode de raisonnement d'un système expert est des plus rudimentaires, elle se nomme *modus ponens* :
- si la proposition A est vraie ;
- si la proposition B se déduit de A ;
- alors la proposition B est vraie.

Cette règle de déduction est à la base des meilleures histoires de Poirot ou Holmes, soit dit en passant.

Le monde du système expert est le plus souvent celui de la logique des propositions. Il suppose que A ou B ont pour seules alternatives d'être *vrai* ou *faux*. Cette contrainte s'appelle *l'hypothèse du monde clos*. Cependant certains systèmes expert sont capable de gérer des faits proportionnellement vrais (A vrai à 60% de probabilité, par exemple).

Le Système Expert est une colle à 3 composants

La *base de faits*, la *base de règles* et le *moteur d'inférence* sont les parties constitutives du système expert. Le terme « générateur de système expert » regroupe un moteur et un ensemble permettant la construction de différentes bases de règles.

La base de faits que l'on nomme parfois *mémoire à court terme* regroupe l'ensemble des faits connus (ceux donnés initialement plus ceux déduits par processus) du système expert à l'instant t.

La base de règles ou *mémoire à long terme* contient l'ensemble des règles de déduction qui serviront à induire de nouvelles connaissances à partir de celles acquises par le passé. C'est en somme la concentration de l'expertise humaine qui est regroupée dans la base de règles. Le moteur d'inférence manipule et confronte faits et règles afin d'enchaîner les déductions et construire des raisonnements.

Un moteur est de *type 0* s'il ne traite que des objets de la logique des propositions ; pour comparaison le langage Prolog contient un moteur de *type 1* capable de travailler sur des prédicats qui sont en gros des propositions dont la valeur dépend des variables les composant.

Un moteur d'inférence est dit *monotone* s'il rajoute tous les faits déduits à la base de faits sans jamais en retirer aucun.

AH ! VOUS ECRIVEZ

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

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

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

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

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

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

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

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

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

Jean-Jacques Dhénin (177)

```
DEF KEY 'fW', CHR$(197);      (é)
DEF KEY 'fE', CHR$(193);      (ê)
DEF KEY 'fR', CHR$(201);      (è)
DEF KEY 'fY', CHR$(203);      (ù)
DEF KEY 'fU', CHR$(195);      (û)
DEF KEY 'fI', CHR$(209);      (î)
DEF KEY 'fO', CHR$(194);      (ô)
DEF KEY 'f/', CHR$(92);       (\)
DEF KEY 'fA', CHR$(192);      (â)
DEF KEY 'fS', CHR$(200);      (à)
DEF KEY 'fD', CHR$(205);      (ë)
DEF KEY 'fJ', CHR$(207);      (ü)
DEF KEY 'fK', CHR$(221);      (ï)
DEF KEY 'f*', CHR$(124);      (|)
DEF KEY 'fC', CHR$(181);      (ç)
```

PPC PARIS SE REUNIT
UNE FOIS PAR MOIS

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

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

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

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

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

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

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

Les dates des prochaines réunions sont :
Samedi 1er octobre 1988
Samedi 5 novembre 1988
Samedi 3 décembre 1988
Samedi 7 janvier 1989
Samedi 4 février 1989
Samedi 4 mars 1989
Samedi 6 mai 1989
Samedi 3 juin 1989

Pierre David (37)

NOUS EN AVONS

La coopérative du Club vous propose :

- de **lecteurs de cartes** magnétiques pour HP-71, neufs, dans leur boîte d'origine, avec 5 cartes magnétiques, pour 500 F (port compris),
- des **anciens numéros** de JPC, au prix de 40 F + 7,40 F de frais d'affranchissement,
- d'une **année complète** de numéros de JPC (février à janvier) pour 300 F (offre spéciale) port compris,
- des **I.D.S.** du module Forth / Assembleur (listing interne commenté par HP) pour 250 F (port compris),
- des **VASM** pour HP-41 (listings des Roms internes commenté par HP) pour 300 F (port compris),
- de **manuels de service** du HP-41 au prix de 75 F (port compris),
- de **manuels de service** du HP-75 au prix de 75 F (port compris).

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

Si vous souhaitez des renseignements complémentaires, n'hésitez pas à nous contacter.

VOUS EN VOULEZ

Nom :
Prénom :
No de membre :
Adresse :

Commande :

	Qté		Prix Unitaire		Prix Total
lecteur de cartes pour HP-71		x	500	FF	
anciens numéros de JPC		x	47,40	FF	
année complète de JPC		x	300	FF	
I.D.S. du module Forth		x	250	FF	
Programmathèque HP-71 (joindre 3 disquettes)		x	75	FF	
Manuel de service pour HP-41		x	75	FF	
Manuel de service pour HP-75		x	75	FF	
JPC Rom + Manuel		x	600	FF	
JPC Rom + Manuel + vos propres programmes		x	800	FF	
Actualisation Eprom		x	150	FF	
			Total		FF

Préciser éventuellement les
numéros de JPC commandés :

PPC PARIS

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

BULLETIN D'ADHESION

Nom ______________________________

Prénom ______________________ Date de naissance __/__/__

Adresse ______________________________

____

____

____

Code Postal _____ Ville ______________________

Téléphone Domicile ______________ Bureau ______________

Profession ______________________________

Intérêts ______________________________

____

Matériel HP en votre possession ______________________________

____

Autre matériel informatique ______________________________

____

Comment avez-vous connu PPC Paris ? ______________________________

____

Que recherchez-vous au sein de PPC Paris ? ______________________________

____

____

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

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

Date __/__/19__

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

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

Paiement à l'ordre de "PPC Paris" par chèque bancaire ou virement postal (CCP No 18 823 40 C à Paris). Ne pas utiliser d'Eurochèques.

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

Il existe deux grandes méthodes de déduction, éventuellement combinables pour donner le chaînage mixte, qui sont le chaînage avant et le chaînage arrière. La première rassemble les faits qu'elle possède et les essaye dans toutes les règles jusqu'à ce qu'une des règles soit complète et permette de déduire un nouveau fait. Ce nouveau fait est ajouté à la base de faits s'il n'y est pas déjà, puis le processus est réitéré. On s'arrête lorsque plus aucune règle ne donne de déduction. Le chaînage arrière quant à lui, se donne un but à démontrer. Pour cela, il va *remonter* la chaîne de déductions qui y conduit. Le terme de *profondeur* vient de la manière dont est explorée la base de règles (voir l'explication du paragraphe *comment construire sa base de règles*).

Au boulot !

A présent que les présentations sont faites, penchons-nous de plus près sur la chose en commençant par la représentation des connaissances. Le moteur d'inférence viendra en fin, non pas à cause sa complexité (bien qu'il y ait un peu de récursivité à gérer) mais parce qu'il est un maillon dépendant de tous les autres au dessus de lui, par priorité décroissante.

Création d'une règle, codage des données

La base de faits est à la réflexion, la représentation d'une partie de la base de règles. Soit la règle :
- **si** animal a des poils **et si** animal donne lait **alors** animal est mammifère

Cette règle comprend une prémisse composée de deux hypothèses et une conclusion. La conclusion se compose d'un fait unique et dans le logiciel proposé, les règles pourront avoir de un à sept faits d'hypothèses mais un seul fait de conclusion (pour simplifier on dira une conclusion et une à sept hypothèses). Pour la suite de l'exposé, remarquons que l'ordinateur (bête et méchant) n'attache aucune espèce de sémantique aux objets-faits qu'il manipule.

Dans un soucis de minimalité, il a été décidé de coder l'ensemble des chaînes alphanumériques représentant les faits dans un tableau. Ainsi une règle sera la juxtaposition de références aux enregistrements du tableau de type *string-array*. De même, dans la base de faits, « animal a des poils » sera remplacé par le fait numéro 24.

Pour résumer, une règle se compose de 17 quartets et découpée comme suit :

C H0 H1 H2 H3 H4 H5 H6 D

où C est la conclusion (1 octet), Hi les hypothèses (1 octet chaque) et D pour l'option diagnostique de la règle (1 quartet). Cette option sert à indiquer que la règle est le point final d'un raisonnement. Dans la base de règles donnée ci-après, le signe @ à la fin de la règle marque l'option diagnostique. Chaque fait de la base de faits tient sur 3 quartets, un octet référence au tableau, un quartet marquant la valeur de vérité attachée (vrai ou faux).

Gestion des bases, dialogues avec l'utilisateur

La gestion des bases de règles et de faits est des plus simplistes : destruction, insertion, affichage sont les seules commandes disponibles. Toutes ces commandes sont construites autour d'un système d'entrée/sortie qui se veut proche de l'INPUT du Basic. Les phrases entrées au clavier sont converties en minuscules et les espaces inutiles sont supprimés, ceci limite les erreurs d'introduction.

Il a été prévu un ramasse-miettes (*garbage collector*) afin de supprimer du tableau des chaînes les phrases-faits qui n'apparaissent dans aucune règle et éviter ainsi tout engorgement du système. Le ramasse-miettes est particulièrement utile lorsque la base de règles subit de nombreuses modifications structurelles.

Enfin, pour être agréable, l'ensemble du programme renvoie ses messages en français. Les mots Forth le constituant ont eux aussi une consonnance hexagonale !

Comment faut-il construire sa base de règles ?

La mise au point d'une bonne base de règles est une partie complexe et passionnante à la fois. Plus que sur les performances du moteur d'inférence, un système expert se juge à la qualité de la connaissance renfermée. Avant d'introduire l'explication du mode de travail du moteur, découvrons ensemble un exemple de base de règles : **les pâtes fraîches à l'italienne**. Cette petite merveille est l'oeuvre de deux camarades de promotion (nous sommes élève-ingénieurs à l'Ecole Supérieure d'Informatique, Electronique et Automatique), Messieurs S. Labati et F. Jallut, lesquels m'ont gentiment autorisé à reproduire leur travail.

Cette base propose des raisonnements qui n'ont pas moins de trois niveaux d'exploration : les pâtes, les ingrédients annexes, les sauces, les plats.
Les signes @ en fin de certaines règles indiquent des diagnostics possibles : ce système expert a pour tâche de reconnaître un plat de pâtes.

A présent, voici quelques éléments de fonctionnement du moteur d'inférence, soit par exemple à reconnaître le plat *spaghetti carbonara*. Si le système expert peut prouver qu'il est en présence de spaghetti et de sauce carbonara (règle 36), c'est gagné. Les règles servent à décomposer le problème en plusieurs problèmes de nature plus simple. Pour avoir des spaghetti, il faut avoir des pâtes fraîches (règle 4). On a des pâtes fraîches lorsqu'on réunit du *sel*, de la *semoule*, de *l'eau* et des *oeufs* (règle 2). A ce niveau, il n'y a plus de décomposition possible ; 2 solutions : le fait *oeuf* existe dans la base de faits avec la mention vrai (présent) ou faux (absent), alors le moteur peut poursuivre son raisonnement, ou bien la base de faits ne contient pas l'information et dans un premier temps on va explorer d'autres possibilités (cas de faits décomposables par les règles) et enfin, si l'information n'existe toujours pas, dans un deuxième temps on questionne l'utilisateur. Supposons que la base de faits contienne les faits nécessaires à la constitution de pâtes fraîches, alors le fait *pâtes fraîches* est ajouté à la base, puis le fait *spaghetti* (par la loi du *modus ponens*). Donc en cherchant à décomposer le fait *spaghetti carbonara*, le moteur a réussi à prouver que le fait *spaghetti* est vrai. Il va de la manière décrite ci-dessus, tenter de prouver l'existence du fait *sauce carbonara*.

Afin de comprendre le détail du fonctionnement, je vous propose de faire *tourner* le moteur pour différentes entrées.

Revenons un instant sur le fait *sel*, il est capital : si le moteur ne peut prouver que le fait *sel* est vrai (donc qu'il y a du sel), il ne sera jamais capable de prouver qu'il y a des pâtes fraîches dans le plat à reconnaître. En effet pour que la partie conclusion d'une règle soit établie, il faut que l'ensemble des hypothèses soit vérifié. Pour cette histoire de sel, pas de sel alors pas de pâtes et il sera impossible au système de déduire quoi que ce soit. C'est ici le point faible de l'expertise, mais Messieurs Jallut et Labati ont eû de l'ingéniosité dans le choix de leur sujet et c'est là que réside l'intérêt.

Conclusion : cet exemple montre qu'il n'est pas facile de codifier un sujet d'expertise.

Le moteur d'inférence

Typiquement, le moteur va *remonter* les règles exactement comme il a été exposé au paragraphe précédent. Un des « plus » du moteur est un certain mécanisme anti *base-incohérente*. Une telle base contient 3 règles du genre :
- si A alors B ;
- si B alors C ;
- si C alors A .

A, B, C ne peuvent pas donner un raisonnement qui tiennent debout car ils sont interdépendants, on tourne en rond. Pour empêcher le programme de planter, il suffit de noter le numéro de la règle qui est examinée. Si ce numéro intervient dans le futur alors c'est qu'on repasse au même endroit et donc incohérence.

D'autre part, ce moteur d'inférence est un logiciel récursif, usant de la récursivité indirecte, et la pile de données (*data stack*) sert à la sauvegarde de l'environnement entre chaque appel. La programmation des appels récursifs passe par l'instruction Forth EXECUTE, appliquée au CFA du mot appelé ; l'appelant étant situé avant l'appelé dans le dictionnaire.

Voila, c'est ici que j'achève l'explication « technique ». Certains développements pouvant mener fort loin (trop pour moi !) ont été volontairement éludés, le but de l'article reste la découverte du système expert. Sachez cependant que l'ensemble du logiciel peut s'améliorer et c'est avec un plaisir à peine dissimulé par un voile de fausse modestie que je répondrai à vos questions !

MODE D'EMPLOI

Manipulation de la base de règles

Les ordres manipulant les règles se trouvent dans le vocabulaire BAREG.

RAZREGLES (--)
Efface la base de règles et la base de faits. Cet ordre est sans appel, alors méfiance !

AJREGLE (--)
Ajout d'une nouvelle règle. Il y a une conclusion mais de une à six hypothèses possibles.

La contenance de la base de règles est de 50 règles numérotées de 0 à 49. Chaque fait possède au plus 25 caractères. Exemple de règle : **si** animal vole **et si** animal pond oeufs **alors** animal est oiseau

affichage programme :

```
Conc. : animal est oiseau [ENDLINE]
Hyp#0. : animal vole [ENDLINE]
Hyp#1. : animal pond oeufs [ENDLINE]
Hyp#2 : [ENDLINE]
```

Le fait d'avoir entré la chaîne vide démarre le processus de codage. Entrer une chaîne vide comme conclusion de règle annule le processus.

INSREGLE (n --)

Insère une règle en position précédent *n*. L'usage est celui de AJREGLE.

DETREGLE (n --)

Détruit, après confirmation, la règle *n*.

TOUREGLE (--)

Edite des règles avec leur numéro d'enregistrement. Une pression sur une touche autorise le passage de la conclusion aux hypothèses, puis de règle en règle.

LISREGLE (n --)

Edite la règle *n*. L'usage est identique à TOUTREGLE.

RM% (--)

Ramasse-miettes. A utiliser lorsque la base a subi des modifications ; l'ordre permet de récupérer la place gaspillée. Pour toute case récupérée un + s'affichera à la place d'un *.

Manipulation de la base de faits

L'ensemble des ordres se trouve dans le vocabulaire FORTH. Les ordres sont duals de ceux avancés pour les règles.

RAZFAITS (--)
Remet à zéro la base de faits.

AJFAIT (--)
Rajoute un fait à la fin de la base. Tous les faits entrés de cette manière seront *vrai*

DETFAIT (n --)
Détruit, après confirmation, le fait numéro *n*.

TOUTFAIT (--)
Edite la base de fait avec le numéro d'enregistrement et la valeur de vérité de chaque fait.

LISFAIT (n --)
Edite le fait *n* . L'usage est identique à TOUFAIT.

Moteur d'inférence

Deux mots situés dans le vocabulaire FORTH.

MOTEUR (--)

Partie du moteur chargée de montrer un fait que l'on entre au clavier. Il est important de noter que la base de faits peut être initialisée indépendamment, le moteur s'en servira. Tous les faits déduits au cours de l'opération ainsi que ceux qui auront fait l'objet d'une question à l'utilisateur seront ajoutés aux faits initialement installés dans la base de faits.

Pour comprendre l'affichage : le moteur affiche à l'écran le fait qu'il essaye d'établir. On peut dès lors suivre la progression des raisonnements. Lorsque le moteur pose une question concernant un fait, il affiche en partie droite de l'écran : "<- V,F,P". Trois réponses possibles qui sont : V pour *vrai*, F pour *faux* et P pour *pourquoi*. Dans le dernier cas, le moteur édite à l'aide de LISREGLE la règle dont il essaye de démontrer la conclusion, puis le moteur repose sa question aux trois choix. La réponse du moteur en fin d'exploration est de la forme "XXX est établi α", où XXX est le fait entré au début et α est la valeur de vérité associée. L'ensemble des faits déduits peut être listé par TOUFAIT.

DIAGNOSE (--)
Cet ordre va tenter d'établir l'une des conclusions parmi les règles dont on dit qu'elles servent au diagnostic. La base de fait est remise à zéro au début de l'opération. DIAGNOSE s'arrête lorsqu'un des diagnostics a été établi ou par épuisement des possibilités. Le maniement de DIAGNOSE est similaire à celui de MOTEUR en ce qui concerne le choix « V,F,P » .

Remarque : La Forthram doit être étendue à 8000 octets pour supporter l'ensemble du logiciel.

Xavier Bille (203)

```
.( Générateur de Systèmes Experts)

( auteur : Bille [203]                                    )
( ******************* déclaration des variables ** )
( constante indiquant le nombre maximum de phrases )
( dans le système                                         )
60 CONSTANT MXPH
( constante indiquant le nb maxi. de règles du SE  )
50 CONSTANT MXRE
( constante indiquant le nb maxi. de faits du SE   )
50 CONSTANT MXFA
( tableau contenant les phrases-faits du SE        )
25 MXPH STRING-ARRAY PHRASES
( pointeur sur la dernière+1 case occupée du       )
( tableau des phrases                                     )
CREATE PH@ 1 C,
```

```
( structure des règles                                    )
: TABLE
  CREATE 17 * NALLOT
  DOES> SWAP 17 * + ;

( mot retournant l'adresse de la i-ème hypothèse          )
: HYPO                                    ( ad -- ad' )
  2* + 2+ ;

( retourne l'adresse du quartet diagnostique              )
: DIAG                                    ( ad -- ad')
  16 + ;

( structure des faits                                     )
: TABLE3
  CREATE 3 * NALLOT
  DOES> SWAP 3 * + ;

( les règles                                              )
MXRE TABLE REGLES
( pointeur sur la dernière+1 règle                        )
CREATE RE@ 0 C,
( variable intermédiaire pour le codage des règles        )
CREATE UNEREG 17 NALLOT
( les faits                                               )
MXFA TABLE3 FAITS
( pointeur sur le dernier+1 fait                          )
CREATE FA@ 0 C,
( ******************** utilitaires *************** )
( identique à +! mais pour les octets                     )
: C+!                                     ( o ad -- )
  DUP C@ ROT + SWAP C! ;

( convertit une chaîne en minuscule                       )
: LWRC$                                   ( str -- str')
  DUP 64 > OVER 91 < AND 32 * - ;

( affichage avec scrolling                                )
: DISP                                    ( str -- )
  " DISP FORTH$@SCROLL1" BASICX ;

( ABORT en préservant l'environnement vocabulaire         )
: SOFTABORT                               ( fl str -- )
  ROT
  IF TYPE SP! QUIT ELSE 2DROP THEN ;

( demande de confirmation                                 )
( renvoie -1 pour oui, 0 sinon                            )
: O/N?                                    ( -- fl)
  ." Sur et certain (O/N)? " KEY
  LWRC$ 111 = DUP
  IF ." oui" ELSE ." non" THEN ;

( teste que n < valeur rangée en adr                      )
: AGIR?                                   ( n adr -- n)
  C@ OVER > OVER 0< NOT
  AND NOT " Impossible d'agir." SOFTABORT ;
```

```
( interface d'entrée au clavier                           )
( transforme en minuscule, enlève les blancs de           )
( tête et de queue, limite à un blanc séparateur          )
( entre les mots                                          )
: >KBOARD                                 ( -- str )
  PAD 96 OVER C! 2+ 2+
  DUP 0 OVER
  27 EMIT 62 EMIT
  EXPECT96 OVER
  BEGIN
    DUP C@ BL =
  WHILE 2+ REPEAT       ( enlève les blancs de tête )
  BEGIN
    DUP 4N@
  WHILE     ( enlève les blancs supplémentaires )
    DUP C@              ( dans la séparation de mots )
    LWRC$ DUP BL -
    IF 2SWAP DROP
      >R R@ C! R> 2+ 0
    ELSE 2SWAP
      IF -1 ROT DROP
      ELSE >R R@ C! R> 2+ -1
      THEN
    THEN
    ROT 2+
  REPEAT
  DROP 2*
  +         ( enlève le blanc de queue qui a survécu )
  OVER - 2/
  2DUP SWAP 2- C! ;


( retourne n le numéro d'enregistrement dans le           )
( tableau des phrases si chaîne est répertoriée           )
: EST-PH                  ( str -- 0 str) ( str -- -1 n)
  0 DUP
  BEGIN
    NOT OVER PH@ C@ < AND
  WHILE 1+ DUP
    2OVER ROT PHRASES S=
  REPEAT
  DUP PH@ C@ = NOT DUP
  IF 2SWAP 2DROP ELSE SWAP DROP THEN ;


( teste et retourne lorsqu'il existe, le numéro           )
( d'enregistrement du fait n' dans base de faits          )
: EST-FA                  ( n -- 0 n) ( n -- -1 n')
  0 0
  BEGIN
    NOT OVER FA@ C@ < AND
  WHILE
    2DUP FAITS C@ =
    SWAP OVER NOT - SWAP
  REPEAT
  DUP FA@ C@ < DUP >R
  IF SWAP THEN
  DROP R>  ;
```

```
( essaye d'insérer la phrase représentée par n      )
( dans la base de faits en y codant sa valeur de    )
( vérité [fl=0,-1]                                  )
( retourne -1 si ok., 0 si base de faits saturée    )
: NOUVFA                              ( n fl -- fl')
  FA@ C@ DUP MXFA = NOT DUP >R
  IF FAITS ROT OVER C! 2+ N!
    1 FA@ C+!
  ELSE 2DROP DROP CR
    ." Base de faits saturée"
  THEN R> ;


( saisie d'un fait au clavier et son insertion en   )
( fin de la base de faits                           )
: AJFAIT                                    ( -- )
  ." Fa.: " >KBOARD
  EST-PH NOT " Phrase Inconnue" SOFTABORT
  EST-FA
  IF ." Déjà enregistré No" .
  ELSE -1 NOUVFA
    IF ." Fait No" FA@ C@ 1- . THEN
  THEN ;


( détruit un fait de la base de faits               )
( demande confirmation                              )
: DETFAIT                                   ( n -- )
  FA@ AGIR? O/N?
  IF DUP 1+ FAITS
   OVER FAITS ROT
   FA@ C@ SWAP - 3 * NMOVE
   -1 FA@ C+!
   ." ,fini."
  ELSE DROP THEN ;

( liste le fait n en donnant l'énoncé et la valeur  )
( de vérité                                         )
: LISFAIT                                   ( n -- )
  FA@ C@ 0= " Base de faits vide." SOFTABORT
  DUP 0< NOT NEGATE *
  FA@ C@ 1- OVER - DUP 0> NOT * -
  FAITS DUP C@ PHRASES
  ." Fa.: " DISP
  ." Valeur: " 2+ N@
  IF ." vrai" ELSE ." faux" THEN
  SPACE ;

( liste la totalité de la base de faits             )
: TOUFAIT                                   ( -- )
  FA@ C@ 0 DO
    ." No" I DUP . 127 EMIT SPACE
    LISFAIT " " DISP
  LOOP ;

( remet à zéro la base de faits                     )
: RAZFAITS                                  ( -- )
  0 FA@ C! ;
```

```
( ******************** vocabulaire des règles **** )
VOCABULARY BAREG IMMEDIATE
BAREG DEFINITIONS

( enregistre une nouvelle phrase et lui attribue    )
( un numéro                                         )
: NOUVPH                                  ( str -- n)
  1
  BEGIN
    DUP PHRASES SWAP DROP
    SWAP OVER 0> -
    DUP MXPH > " Plus de place." SOFTABORT
    SWAP 0=
  UNTIL
  DUP >R PHRASES S! R>
  DUP PH@ C@ =
  IF DUP 1+ PH@ C! THEN ;

( utilitaire pour afficher un message               )
: CO/HY                                     ( n -- )
  DUP 0<
  IF ." Conc" DROP ELSE ." Hyp#" . THEN
  ." : " ;

( pour installer une nouvelle règle                 )
( saisie au clavier et codage dans variable UNEREG  )
( 1 conclusion et 7 hypothèses maximum              )
: >>REGLE                                   ( -- )
  -1 UNEREG
  7 -1 DO
    OVER
    IF I CO/HY
      >KBOARD DUP
      IF EST-PH NOT
        IF NOUVPH THEN
      ELSE 2DROP
        I 0< " Abandon." SOFTABORT
        I 0>
        IF SWAP NOT SWAP 0
        ELSE ." Non permis," -1 THEN
      THEN
    ELSE 0 THEN
( fin de règle détectée, on complète avec des 0     )
    2DUP SWAP I HYPO C!
    0< NOT NEGATE +LOOP
  ." Diagnostique (O/N)? " KEY
  LWRC$ DUP EMIT
  111 = OVER DIAG N!
  2DROP ;

( pour insérer une règle avant celle de No n        )
: INSREGLE                                  ( n -- )
  DUP 0<
  RE@ C@ DUP MXRE = ROT OR
  " Base de règles saturée" SOFTABORT
  2DUP >
  IF SWAP THEN
  DROP >>REGLE
```

```
  DUP REGLES
( on va créer un trou dans la base de règles      )
( sauf si on insère en fin de base de règles      )
  OVER DUP 1+ REGLES
  RE@ C@ ROT OVER - 17 *
  SWAP 1+ MXRE = NOT * NMOVE>
  UNEREG OVER REGLES 17 NMOVE
  CR ." Règle No" .
  1 RE@ C+! ;

( ajoute une règle à la fin de la base de règles  )
: AJREGLE                                  ( -- )
  RE@ C@ INSREGLE ;

( efface les bases de règles et de faits          )
( le tableau des phrases est réinitialisé         )
: RAZREGLES                                ( -- )
  MXPH 1+ 1 DO
    NULL$ I PHRASES S!
  LOOP
  1 PH@ C! 0 RE@ C! RAZFAITS ;

( pour détruire la règle No n de la base de règles )
( demande la confirmation                         )
: DETREGLE
  RE@ AGIR? O/N?
  IF DUP 1+ REGLES
    OVER REGLES ROT
    RE@ C@ SWAP - 17 * NMOVE
    -1 RE@ C+!
    ." ,fini." RAZFAITS
  ELSE DROP
  THEN ;

( édition de la règle No n                        )
( affichage se rapprochant de fourni pour la      )
( saisie au clavier                               )
: LISREGLE                                 ( n -- )
  RE@ C@ 0= " Base de règles vide." SOFTABORT
  DUP 0< NOT NEGATE *
  RE@ C@ 1- OVER - DUP 0> NOT * -
  REGLES
  7 -1 DO
    DUP I HYPO C@ ?DUP
    IF I CO/HY
      PHRASES DISP
      KEY DROP CR
    THEN
  LOOP
  DIAG N@
  IF " {diagnostique}" DISP THEN ;

( édition de toute la base de règles              )
: TOUREGLE                                 ( -- )
  RE@ C@ 0 DO
    ." No" I DUP . 127 EMIT SPACE
    LISREGLE
  LOOP ;

( ramasse-miettes pour récupérer la place         )
( gaspillée dans le tableau des phrases           )
: RM%                                      ( -- )
  ." Ramasse-Miettes: "
  RE@ C@ 0= PH@ C@ DUP 1 = ROT OR
  IF DROP RAZREGLES EXIT THEN
( boucle pour chacune des phrases du système      )
  1 DO I PHRASES SWAP DROP
    IF I
( on essaye de retrouver la phrase dans une règle )
      RE@ C@ 0 DO I REGLES
        7 -1 DO
          2DUP I HYPO C@ =
          IF SWAP DROP 0 SWAP LEAVE THEN
        LOOP
        DROP DUP 0= IF LEAVE THEN
      LOOP DUP
      IF NULL$ I PHRASES S! THEN
( si intervention + affiché, * sinon              )
      0= 43 + EMIT
    THEN
  LOOP
( réajustement des pointeurs                      )
  1 PH@ C@ 1- DO
    I PHRASES SWAP DROP 0=
    IF -1 PH@ C+! ELSE LEAVE THEN
    -1 +LOOP
  RAZFAITS ;


( ******************** moteur d'inférence ******** )
FORTH DEFINITIONS
CREATE STA@ 0 C,


( ***** mots pour le mécanisme anti-incohérence ** )
( sauvegarde de n dans une pile                   )
( contenance maxi. : 50 éléments                  )
: PUSH                                     ( n -- )
  VARID @ ?DUP
  IF FINDBF STA@ C@ +
  ELSE 100 MAKEBF NOT " MSG$(24)" BASIC$ SOFTABORT
    VARID !
  THEN
  C! 2 STA@ C+! ;


( dépilement d'une valeur                         )
( renvoie -1 si la pile est vide,                 )
( 0 si elle n'existe plus                         )
: POP                                ( -- n) ( -- fl)
  VARID @ 0> STA@ C@ AND
  IF -2 STA@ C+!
    VARID @ FINDBF STA@ C@ + C@
  ELSE
    VARID @ KILLBF
    0 STA@ C! 0 VARID !
  THEN ;
```

```
( teste l'appartenance de n à la pile                    )
( renvoie -1 si appartient                               )
: IN-STA                                    ( n -- fl)
  VARID @ FINDBF DUP STA@ C@ + 0
  BEGIN
    >R 2DUP < R> NOT AND
  WHILE
    2- DUP C@
    2OVER DROP =
  REPEAT
  SWAP DROP C@ OVER = STA@ C@ 0> AND ;

( ********** moteur d'inférence proprement dit ** )
( recherche n, partie conclusion d'une règle à     )
( partir de r-cur, règle courante                  )
( en cas d'échec renvoie 0 et r-cur=re@            )
( si succès, r-cur pointe sur la règle satisfiant  )
( la contrainte                                    )
: CONCLU?                      ( n r-cu -- fl r-cu' n)
  0
  BEGIN
    NOT OVER RE@ C@ < AND
  WHILE
    2DUP REGLES C@ =
    SWAP OVER NOT · SWAP
  REPEAT
  DUP RE@ C@ < ;

( poser une question relative au fait n            )
( renvoie la valeur de vérité attribuée par        )
( l'utilisateur                                    )
: DEMANDE                                   ( n -- fl)
  BEGIN
    "  ← V,F,P? " DISP
    KEY LWRC$
    DUP 102 = OVER 118 = OR
    IF 118 = 2* 1+ -1 ROT DROP
    ELSE CR 112 =
      IF STA@ C@ 0>
       IF POP DUP PUSH
         ." Règle No" DUP . ." -> "
         BAREG LISREGLE FORTH
         ." répondre: "
       ELSE ." ne peut conclure "
       THEN
     THEN
     DUP PHRASES TYPE 0
    THEN
  UNTIL ;

( variable utilisée en récursivité                 )
VARIABLE ETA@

( dépile les k buts empilés dans la data-stack     )
( α signifie une valeur quelconque                 )
: ABANDON                 ( n1 n2 ... nk α k -- 1 0)
  SWAP DROP
  BEGIN
    DUP 0>
  WHILE
    >R DROP R> 1-
  REPEAT 1 ;

( partie du moteur qui tente de vérifier les       )
( prémisses de la règle r-cu                       )
( renvoie 0 : on ne sait pas, -1 : vrai, 1 : faux  )
: CHAINAR                          ( r-cu -- fl r-cu)
  IN-STA
  IF 0 STA@ C!
    POP "  # Base Incohérente #" SOFTABORT
  THEN
( sauvegarde du No de la règle pour le mécanisme   )
( anti-bouclage                                    )
  DUP PUSH REGLES 0
( empile dans la data-stack celles des hypothèses  )
( qui ne sont pas dans la base de faits            )
  0 6 DO
    OVER I HYPO C@ EST-FA
    IF FAITS 2+ N@ 0=
( si une des hypoth. de la règle est fausse alors  )
( la règle ne peut donner de conclusion, renvoie 1 )
      IF ABANDON LEAVE THEN
    ELSE ?DUP
      IF ROT ROT 1+ THEN
    THEN
    -1 +LOOP
( si tout est ok [pas abandon], on va essayer      )
( d'établir la valeur de vérité des hypothèses     )
( de la règle                                      )
  SWAP 0> ?DUP
( on poursuit les opérations tant que ETABLIR      )
( renvoie vrai                                     )
  IF BEGIN
      OVER 0> OVER 0< AND
    WHILE
      ROT ETA@ @ EXECUTE AND
      SWAP 1- SWAP
    REPEAT
    DUP 0< NOT
    IF SWAP OVER >R ABANDON DROP R> THEN
    SWAP DROP
  THEN
  >R POP R> ;

( drapeau qui inhibe les questions à l'utilisateur )
CREATE DEM? 1 NALLOT

( cherche à établir que le fait n est vrai [-1] ou )
( faux [1] et à l'extrême, « sans opinion » [0]    )
: ETABLIR                                   ( n -- fl)
  DEM? N@
  IF CR ." Cour.: " DUP PHRASES TYPE THEN
  EST-FA
  IF FAITS 2+ N@ 0> 2* 1+ EXIT THEN
  0 CONCLU?
  IF 1
```

```
   BEGIN
( attention : appel récursif indirect          )
      ROT ROT CHAINAR
      >R ROT R> 2DUP >
      IF SWAP THEN
      DROP DUP 0<
      IF DUP ELSE >R 1+ CONCLU? NOT R> SWAP THEN
    UNTIL
  ELSE DEM? N@
    IF OVER DEMANDE ELSE 0 THEN
  THEN
  ROT ROT DROP
  DEM? N@
  IF OVER 0< NOUVFA NOT
    IF ." #blocage" THEN
  ELSE DROP THEN ;
( ETA@ contient le cfa de ETABLIR afin que ce  )
( dernier puisse être appelé par CHAINAR       )
' ETABLIR ETA@ !

( saisir un fait au clavier et de lui attribuer  )
( une valeur de vérité                           )
( 2 passages : le premier tente d'établir sans   )
( poser de questions, seulement en utilisant la  )
( base de faits , le second est activé lorsqu'il y )
( a échec, le moteur d'inférence se permet de    )
( poser des questions                            )
: MOTEUR                                  ( -- )
  ." But: " >KBOARD EST-PH
  NOT " Phrase inconnue" SOFTABORT
  0 DEM? N!
  DUP ETABLIR ?DUP 0=
  IF 1 DEM? N! ETABLIR ELSE SWAP DROP THEN
  CR ." But établi: " 0<
  IF ." vrai" ELSE ." faux" THEN
  SPACE ;

( essaye d'établir un diagnostic, en posant des  )
( questions, cela va de soi...                   )
: DIAGNOSE                                ( -- )
  RAZFAITS 0
  RE@ C@ DUP 0= " Pas de règles" SOFTABORT
  1 DEM? N!
  0 DO
    I REGLES DIAG N@
    IF I REGLES C@ ETABLIR 0<
      IF NOT I REGLES C@ SWAP LEAVE THEN
    THEN
  LOOP CR
  IF ." DIAGNOSTIQUE -> " PHRASES TYPE
  ELSE ." Je ne sais pas"
  THEN 2 SPACES ;
```

EXEMPLE DE BASE DE REGLES

LES PATES FRAICHES A L'ITALIENNE

(Labati-Jallut-Bille)

1) **si** oeuf **alors** jaune d'oeuf
2) **si** sel **et** semoule **et** oeuf **et** eau **alors** pâtes fraîches
3) **si** sel **et** farine **et** oeuf **et** eau **alors** pâtes fraîches
4) **si** pâtes fraîches **alors** spaghetti
5) **si** pâtes fraîches **alors** tagliatelle
6) **si** pâtes fraîches **alors** cannelloni
7) **si** pâtes fraîches **alors** lasagnes
8) **si** pâtes fraîches **alors** macaroni
9) **si** pâtes fraîches **alors** tortellini
10) **si** tomates **et** thym **et** laurier **et** sel **et** poivre **alors** sauce tomate
11) **si** jambon **et** veau **alors** farce viande
12) **si** épinard **et** ricotte **alors** farce verte
13) **si** pâtes fraîches **et** épinard **alors** pâtes vertes
14) **si** pâtes fraîches **et** tomate **alors** pâtes rouges
15) **si** lasagnes **et** pâtes vertes **alors** lasagnes vertes
16) **si** sauce tomate **et** farce viande **alors** sauce bolognese
17) **si** crème **et** lard **et** jaune d'oeuf **alors** sauce carbonara
18) **si** moules **et** calmars **et** crème **alors** sauce marinière
19) **si** huile d'olive **et** basilic **et** pecorino **alors** sauce basilic
20) **si** sauce basilic **et** ail **et** marjolaine **et** persil **et** épinard **alors** sauce pistou
21) **si** pâtes fraîches **et** sauce pistou **alors** pâtes au pistou @
22) **si** farine **et** lait **et** beurre **alors** sauce béchamel
23) **si** pâtes fraîches **et** farce viande **et** sauce tomate **alors** ravioli à la tomate @
24) **si** pâtes fraîches **et** farce viande **alors** tortellini
25) **si** pâtes fraîches **et** farce viande **et** gratiné **alors** cannelloni @
26) **si** pâtes fraîche **et** farce verte **alors** tortellini @
27) **si** lasagne **e** sauce tomate **et** farce viande **et** cèpes **et** parmesan **et** sauce béchamel
alors lasagnes vertes au four @
28) **si** pâtes fraîches **et** sauce marinière **alors** pâtes marinières @
si beurre **et** pomme de terre **et** farine **et** jaune d'oeuf **alors** gnocchi
30) **si** beurre **et** semoule **et** jaune d'oeuf **alors** gnocchi @
31) **si** escalope de veau **et** chapelure **e** oeuf **alors** escalope panée
32) **si** escalope panée **et** citron **et** pâtes fraîches **alors** escalope milanese @
33) **si** escalope panée **et** jambon **et** parmesan **alors** saltimboca @
34) **si** spaghetti **et** sauce bolognese **alors** spaghetti bolognese @
35) **si** tagliatelle **et** sauce bolognese **alors** tagliatelle bolognese @
36) **si** spaghetti **et** sauce carbonara **alors** spaghetti carbonara @
37) **si** tagliatelle **et** sauce carbonara **alors** tagliatelle carbonara @

TETE DE LEX !

Le programme WRITHEAD a été écrit pour résoudre l'une des tâches les moins passionnantes auxquelles doivent faire face les programmeurs en assembleur du HP-71 écrivant des fichiers Lex contenant des douzaines de mots-clé : la préparation des en-têtes du fichier Lex, incluant la *text table*, la *main table*, la *speed table* et le traitement du *poll* VER$.

WRITHEAD sera particulièrement utile toutes les fois que vos tokens ne seront pas triés dans un ordre bien précis, parce qu'il éliminera le risque d'inversion de l'ordre d'instructions variées, et le danger associé à de telles erreurs.

WRITHEAD et cette documentation sont placées par les présentes dans le domaine public pour une utilisation non commerciale.

WRITHEAD surmonte les problèmes mentionnés plus haut en créant automatiquement des fichiers de code source Lex pour HP-71. Il commence en posant quelques questions, et selon les réponses fournies, démarre la création de l'en-tête désiré. Au moment voulu, il vous demandera de fournir les informations sur les mots-clé que vous souhaitez inclure dans votre fichier Lex. Cette information peut être lue depuis un fichier d'affectation de tokens approprié, s'il y en a un de disponible. Sinon, l'information pourra être ou bien ajoutée à un fichier d'affectations de tokens déjà existant, ou bien stockée dans un fichier qui sera automatiquement créé. Le programme vous attendra jusqu'à ce que toutes les informations voulues soient fournies.

L'en-tête résultant se trouvera dans un fichier texte nommé ASMHEAD. A ce niveau, tout ce que vous aurez à faire pour obtenir un chouette fichier Lex est d'ajouter votre code et de lancer l'Assembleur... Si le code pour chaque mot-clé a été testé individuellement avec succès, le seul problème est alors vraisemblablement la duplication de labels, ce qui peut être facilement corrigé avec l'éditeur de texte approprié.

Les avantages des grands fichiers Lex

Un fichier Lex monolithique a plusieurs avantages sur une collection de fichiers plus petits qui ont été chaînés ensembles pour fournir le même jeu de mots-clé. La réduction potentielle en utilisation de mémoire peut justifier les efforts et le temps supplémentaire nécessaires pour préparer le fichier source, même si vous utilisez le module Forth Assembler, plus lent. Ainsi, le fichier Lex monolithique vous aidera significativement à améliorer les performances de votre HP-71 au niveau du clavier et du temps d'exécution.

La performance améliorée (réponse plus rapide), est due aux faits suivants :

1) le système d'exploitation doit passer le contrôle à un seul *poll handler*, au contraire d'un *poll handler* pour chaque fichier Lex chaîné ;

2) le système d'exploitation peut trouver les points d'entrée des mots-clé plus rapidement si il a à traiter seulement un ou deux item dans le buffer bLEX (adresse de fichier Lex) et

3) la recherche à travers les *text tables* est considérablement accélérée avec une *speed table*.

Il est important de comprendre que le temps gagné est celui passé à chercher chaque Lex individuellement et à déterminer les adresses des points d'entrée du code à rechercher ou à examiner. Cela prend autant de tant pour un petit Lex que pour un grand. J'ai découvert que combiner de nombreux petits Lex dans un Lex moins petit a le même impact qu'une accélération de mon HP-71.

Le fichier d'affectation de tokens

Le fichier d'affectation de tokens est un fichier texte standard du HP-71 qui contient en format libre la description des mots-clé. C'est le coeur du problème puisque toutes les autres parties de l'en-tête de fichier Lex sont fixées, suivant la structure rigide requise pour tous les fichiers Lex.

Chaque ligne doit fournir les informations suivantes pour chaque mot-clé donné, séparé par un ou plusieurs espaces :

1) le numéro de token, en hexa
2) le quartet de caractérisation, également en hexa
3) le label du point d'entrée
4) le mot-clé lui-même

Toute information ultérieure sera ignorée.

Exemple tiré de MMTOKENS :

```
  :
  :
B9 F ATBIN$ ATBIN$
BA F BTA$   BTA$
BB F BTD    BTD
BC F CHR2$  CHR2$
```

```
BD F DTBIN$ DTBIN$
BE F INT$   INT$
BF F NUMR   NUMR
C0 F NUMX   NUMX
C1 F BINCMP BINCOMP$
C2 D chirp  CHIRP
C3 F COUNT  COUNT
C4 F cursor CUR$
C5 F Start  DT2DOT$
C6 F ESC    ESCP$
:  : :      :
:  : :      le mot-clé, comme vous l'épelez
:  :  -----le label pour le point d'entrée
:  :        du mot-clé
:   -------le quartet de caractérisation (CHAR #F)
 ----------le numéro de token en hexa
```

Note : si WRITHEAD crée pour la première fois un fichier d'affectation de tokens, l'information est entrée séquentiellement par numéro de token. Ceci minimise la vraisemblance d'affectation de tokens, de labels ou de mots-clé dupliqués. L'information est entrée en colonne 1, 4, 6 et 13. Ce format n'est pas impératif (vous avez seulement besoin d'un espace entre les entrées) mais il a l'avantage important de permettre un tri plus facile du fichier par numéro de token, label de point d'entrée ou mot-clé avec le sous-programme QSORTF de Tapani Tarvainen, qui est partie prenante de WRITHEAD.

WRITHEAD fournit un minimum vital de test de la validité des contenus du fichier d'affectation. On considère que l'utilisateur apportera des informations correctes.

Utilisation de WRITHEAD

Avant de créer l'en-tête désiré, un certain nombre de questions sont posées. Leur utilité et les réponses attendues sont indiquées ci-dessous.

1) `Compatible with ASSEMBLE ?(Y/N)`

La réponse par défaut est "N". Ce qui signifie que le format d'en-tête par défaut sera compatible avec l'assembleur de la série HP-200 (Iapetus) et avec SASM de Hand Held Products(*). Si vous souhaitez utiliser l'en-tête avec ASSEMBLE du module Forth/Assembler, vous devez répondre avec un "Y". Les différences principales sont :
a) les pseudo-ops LEX, ID... ne sont pas reconnus, et
b) des anti-slash sont utilisés pour délimiter les chaînes au lieu de simples quotes.
Voir les IDS du module HP-IL si vous souhaitez jeter un oeil sur un exemple de code compatible avec la série 200.

2) Ensuite, un titre sera demandé pour votre Lex. Il sera utilisé pour sélectionner un nom pour les fichiers Lex produits et pour documenter le code source. Notez que les informations concernant la date sont prises du système du HP-71. Vous devrez utiliser SETDATE pour être sûr que l'information entrée dans votre documentation est correcte.

3) Vous serez ensuite interrogé sur l'identificateur de VER$ pour votre Lex. La valeur par défaut fournie est `" DIR:a"`. Noter le blanc en début. Cet espace doit être fourni pour séparer votre réponse des réponses des autres fichiers Lex.

4) Enfin, nous approchons du point crucial, démarrant avec l'Id du Lex. La valeur par défaut est #5E, le dernier Id de scratch, que j'utilise pour mes allocations de ressources personnelles. A ce point, il vous sera demandé le nom de votre document d'allocation de ressources personnelles. Le défaut fourni est MMTOKENS, où MM représente mes initiales... libre à vous d'y substituer tout autre nom de fichier que vous préféreriez.

Gardez à l'esprit que vous devez fournir un nom de fichier même si vous souhaitez entrer les données à la main. C'est là que les données seront stockées pour un usage futur.

Ensuite, il vous sera demandé de fournir le numéro de token du premier et du dernier mot-clé à inclure dans le fichier Lex que vous être actuellement en train d'écrire. Si l'information, ou toute partie de celle-ci, manque, il vous sera demandé de la fournir depuis le clavier.

5) Une fois les données précédentes entrées, il vous sera demandé si vous voulez inclure une *speed table* avec votre Lex, et si également si une *message table* est à ajouter.

Selon qu'une *message table* est à ajouter ou non, WRITHEAD inclut simplement un `REL(4) MSGTBL` ou un `CON(4) 0` en fonction de votre réponse.

Vous avez maintenant achevé votre partage du travail et c'est à votre HP-71 de devenir très occupé. Il y a pas mal de tris à effectuer correctement pour être sûr que les entrées dans les *text* et *main tables* sont entrées dans le bon ordre, ce qui est assez long. Quand le voyant PRGM s'éteindra, vous pourrez observer le résultat en listant ASMHEAD.

Deux points avant de vous quitter :
1) Une *speed table* prend quelques 78 quartets de mémoire. C'est beaucoup pour de petits fichiers Lex. C'est un gaspillage énorme à moins d'être sûr que votre Lex inclut suffisamment de mots-clé pour que le

temps passé à analyser la *speed table* soit significativement inférieur à celui pris pour parcourir la *text table* en entier.
2) Il vous est rappelé que WRITHEAD suppose que les informations que vous lui fournissez sont valides. WRITHEAD effectue seulement un minimum de vérifications pour éliminer les entrées invalides. Vérifier la validité de vos données est dans votre intérêt.

WRITHEAD peut être amélioré de nombreuses manières : il a encore à être optimisé à la fois en vitesse et en taille du code. J'aimerais ajouter une routine qui pourrait générer une *message table*. Egalement, dans cette version, j'ai évité l'emploi de mots-clé non disponibles dans le système HP-71 + HPIL + Forth Assembler. D'autres mots-clé, et TRIM$ en particulier (du Lex STRINGLX de la HP User's Library), pourraient être utilisés avec grand avantage. Cela rendrait, j'espère, le programme immédiatement utilisable à tous les utilisateurs.

J'espère que vous trouverez ce programme utile et qu'il vous aidera à gagner du temps.

Michael Markov (301)
Traduction Olivier Arbey (118)

(*) : Note de la Rédaction : N'oubliez pas que le Club diffuse l'assembleur de Pierre David et Janick Taillandier qui est bien bien plus rapide et plus puissant que ces produits.

Programme "WRITHEAD" (génération d'en-tête de fichier Lex. Nécessite module Forth/Assembler)

```
- WRITe HEADer - utilities for assembly programmers  3/17/1986
  For the HP-71..  by Michael Markov

  Modified 4/15/87 to support the creation of headers containing
  up to 255 token assignments. Previous version allowed a maximum
  of only 99 tokens.

  The purpose of this program is to write a LEX file "shell", usually
  using a resource allocation file. However, if the file does not exist,
  it is created. Also, if the file exists but there are missing entries,
  they will be added to the file.
  The file name ASMHEAD is arbitrarily allocated to the file in which
  the shell is to be stored. Any existing file of the same name will be
  purged.

  WRITHEAD also purges & creates files ASMTKN & ASMWORDS. These files
  will usually be specially formatted / sorted subsets of resource
  files such as JRBTOKENS or MMTOKENS.

  The Pseudo-ops LEX, KEY, ENTRY, ID, TOKEN and CHAR are avoided to
  provide compatibility with the Iapetus Saturn assembler, and to allow
  token ordering (Main Table) that does not necessarily follow the TEXT
  table ordering. (Default options). However, the first prompt allows
  you to write a LEX file "shell" that is compatible with the FORTH /
  ASSEMBLER ROM orignal ASSEMBLE word, or with the enhanced SASM word
  provided by JRB's FTHFIX5A to fix the 5 known ASSEMBLE bugs.

  See J.R. Baker's FTHFIX5B  for FORTH word SASM, which will allow you
  to assemble the resulting Iapetus compatible source code file header
  with an HP-71.

  This program assumes that there will always be a poll handler, if only
  the VER$ handler, which is created by WRITHEAD...

  The MSG table is NOT writen by WRITHEAD! I may try in the future, but
  what I know about message tables is just about nothing...

  This program is NOT optimized for either speed or memory useage -- be
  happy that it works. Hopefully, future versions will be an improvement.

400 DESTROY ALL @ STD @ OPTION BASE 0 @ DIM B9$[70],C$[120],S$[70]
410 SFLAG -1 @ PURGE ASMHEAD @ CFLAG -1 @ CREATE TEXT ASMHEAD
420 ASSIGN #1 TO ASMHEAD @ RESTORE #1
430 INPUT "Compatible with ASSEMBLE ?(Y/N) ","N";Y$
440 C$="FILENAME ver. <"&DATE$&">"
450 LC OFF @ INPUT 'TITLE ? ',C$;C$
451 IF C$[1,1]=" " THEN C$=C$[2] @ GOTO 451
452 IF C$[LEN(C$)]=" " THEN C$=C$[1,LEN(C$)-1] @ GOTO 452
460 B$="        " @ B9$="**********" @ B9$=B9$&B9$&B9$ @ B9$=B9$&B9$
470 P=UPRC$(Y$)="N" @ IF P THEN D=92 ELSE D=39
480 V$='" DIR:a"' @ LINPUT 'VER$ ? ',V$;V$ @ V=LEN(V$) @ V$=V$[2,V-1]
490 F=POS(C$," ") @ IF NOT F THEN F=9
500 F=MIN(F,9) @ F$=C$[1,F] @ F$[9]="" @ F9$=F$
501 IF F9$[1,1]=" " THEN F9$=F9$[2] @ GOTO 501
502 IF F9$[LEN(F9$)]=" " THEN F9$=F9$[1,LEN(F9$)-1] @ GOTO 502
```

```
510 IF NOT P THEN PRINT #1;B$&"LEX   '"&F9$&"'"
520 PRINT #1;B$&"TITLE "&C$
530 IF NOT P THEN 630 ! Pseudo-op LEX is used to create the file header
540 PRINT #1;"*" @ PRINT #1;B9$
550 PRINT #1;"*        FILE HEADER BLOCK"
560 PRINT #1;"*"
570 PRINT #1;B$&"NIBASC   "&CHR$(92)&F$&CHR$(92)
580 PRINT #1;B$&"NIBHEX   802E      LEX type"
590 PRINT #1;B$&"CON(2)   0         flags=0"
600 PRINT #1;B$&"NIBHEX   0000      Mn/Hr"
610 PRINT #1;B$&"NIBHEX   000078    Dy/Mo/Yr"
620 PRINT #1;B$&"REL(5)   =FILEND"
630 PRINT #1;"*" @ PRINT #1;B9$
640 PRINT #1;"*        LEX  HEADER BLOCK"
650 PRINT #1;"*"
660 INPUT "LEX ID#(HEX) ","5E";I$
661 IF I$[1,1]=" " THEN I$=I$[2] @ GOTO 661
662 IF I$[LEN(I$)]=" " THEN I$=I$[1,LEN(I$)-1] @ GOTO 662
670 ON ERROR GOTO 660 @ I=HTD(I$)
680 OFF ERROR @ I$=DTH$(I)[4] @ IF I>255 THEN DISP "Invalid Arg"; @ GOTO 660
690 INPUT "Assigment file? ","MMTOKENS";F$
691 IF F$[1,1]=" " THEN F$=F$[2] @ GOTO 691
692 IF F$[LEN(F$)]=" " THEN F$=F$[1,LEN(F$)-1] @ GOTO 692
700 PRINT #1;B$&"CON(2)   #"&I$&"      LEX ID, see "&F$&" for assignment"
710 INPUT "LOW TOKEN #(HEX) ","01";I$
711 IF I$[1,1]=" " THEN I$=I$[2] @ GOTO 711
712 IF I$[LEN(I$)]=" " THEN I$=I$[1,LEN(I$)-1] @ GOTO 712
720 ON ERROR GOTO 710 @ L=HTD(I$)
730 OFF ERROR @ IF L>255 THEN DISP "Invalid Arg "; @ GOTO 710
740 INPUT "HIGH TOKEN #(HEX) ",I$;I$
741 IF I$[1,1]=" " THEN I$=I$[2] @ GOTO 741
742 IF I$[LEN(I$)]=" " THEN I$=I$[1,LEN(I$)-1] @ GOTO 742
750 ON ERROR GOTO 740 @ H=HTD(I$)
760 OFF ERROR @ IF H>255 THEN DISP "Invalid Arg "; @ GOTO 740
770 IF L>H THEN DISP "LOW TOKEN > HIGH TOKEN "; @ GOTO 710
780 PRINT #1;B$&"CON(2)   #"&DTH$(L)[4]&"      Low Token"
790 PRINT #1;B$&"CON(2)   #"&DTH$(H)[4]&"      High Token"
800 PRINT #1;B$&"CON(5)   0        No link"
  - Linked LEXes not supported at this time.
    It could be implemented, but it would be easier to edit ASMHEAD to do
    the job, and use WRITHEAD twice...
840 INPUT "Speed Table? (Y/N) ","N";Y$
850 S=UPRC$(Y$)="Y" @ IF NOT S THEN PRINT #1;B$&"NIBHEX   F        No speed table"
860 INPUT "Message table? (Y/N) ","N";Y$
870 M=UPRC$(Y$)="Y"
  - up to this point we have been collecting information
     L= Low token H= High token S#0 =>write speed table
     M#0 =>write MSGTBL offset
     D= ASCII string delimiter
     Now, we are ready to process information that is saved in a suitably
     formatted resource allocation table.
     If the file is not found in RAM, look for it on :tape(1)
     If the file is still not found, then key-in the required information.

    The format for the resource allocation table is free format. However,
    the information MUST be entered in the following order:

    Token #(hex), characterization nib (hex 0=>F), ENTRY label, KEYWORD, other.
```

```
    Each entry must be separated by at least one space.
    Any line that starts with '*' will be ignored.
    There is some minimal testing for invalid data, but I assume the user
    is not deliberatly trying to screw-up WRITHEAD. Ideally, the information
    should be stored using the following IMAGE: "#,2A,X,A,X,6A,X,8A,X,??A"

    Since the main purpose of token assignment tables is to avoid duplicate
    ID & Token use, I assume the job was done properly and therefore do not
    test for such errors. YOU HAVE BEEN WARNED.

1150 DIM A$(H-L+1)[21] ! we will only load "must" info
1160 ON ERROR GOTO 1180
1170 Y$=ADDR$(F$) @ GOTO 1360
1180 ON ERROR GOTO 1200
1190 COPY F$&":tape" @ GOTO 1360
1200 OFF ERROR
  - Darn, we have to create the file!
1220 CREATE TEXT F$ @ ASSIGN #2 TO F$ @ RESTORE #2
1230 FOR J=L TO H @ WINDOW 1,22
1240     DISP "Token #"&DTH$(J)[4] @ WINDOW 11,22
1250     A$(J-L+1)[1,2]=DTH$(J)[4]
1260     ON ERROR GOTO 1260 @ INPUT "CHAR ","F";C$
1270     C=HTD(C$) @ IF C>15 THEN 1260 ELSE A$(J-L)[4]=DTH$(C)[5] @ OFF ERROR @ E$=""
1280     INPUT "ENTRY ",E$;E$
1281     IF E$[1,1]=" " THEN E$=E$[2] @ GOTO 1281
1282     IF E$[LEN(E$)]=" " THEN E$=E$[1,LEN(E$)-1] @ GOTO 1282
1285     IF LEN(E$)>6 OR NOT LEN(E$) THEN 1280
1290     IF POS(E$," ") OR E$[1,1]='*' OR E$[1,1]="(" THEN 1280
1300     E$[7]="" @ A$(J-L)[6]=E$
1310     K$="" @ LC OFF
1320     INPUT "Keyword ",K$;K$ @ K$=UPRC$(K$)
1321     IF K$[1,1]=" " THEN K$=K$[2] @ GOTO 1321
1322     IF K$[LEN(K$)]=" " THEN K$=K$[1,LEN(K$)-1] @ GOTO 1322
1325     IF LEN(K$)<2 OR LEN(K$)>8 THEN 1320
1330     IF POS(K$," ") THEN 1320
1340     OFF ERROR @ A$(J-L)[13]=K$
1350 NEXT J @ PRINT #2;A$ @ WINDOW 1,22 @ GOTO 1670
1360 ASSIGN #2 TO F$ @ RESTORE #2 @ K=1
  -
    the desired entries are copied into a string array, and reformatted
    to ease further processing. Do minimal error trapping.
    LABEL spelling is NOT checked -- use LABEL program for that job.
1410 FOR J=0 TO FILESZR(F$)-1 @ READ #2;C$
1411     IF C$[1,1]=" " THEN C$=C$[2] @ GOTO 1411
1412     IF C$[LEN(C$)]=" " THEN C$=C$[1,LEN(C$)-1] @ GOTO 1412
1415     E$="Invalid Token "
1420     IF C$[1,1]="*" THEN 1600 ! Ignore comments
1430     P=POS(C$," ") @ IF NOT P OR P>3 THEN DISP "Invalid line ";J+1 @ GOTO 1600
1440     ON ERROR GOTO 1590 @ T=HTD(C$[1,P-1]) @ OFF ERROR
1450     IF T<L OR T>H THEN 1600 ! looking for a specific range of tokens...
  -      Now, check the characterization nib
1461     C$=C$[P]
1462     IF C$[1,1]=" " THEN C$=C$[2] @ GOTO 1462
1463     IF C$[LEN(C$)]=" " THEN C$=C$[1,LEN(C$)-1] @ GOTO 1463
1470     E$="Invalid CHAR line " @ IF C$[2,2]<>" " THEN 1590
1480     ON ERROR GOTO 1590 @ C=HTD(C$[1,1]) @ OFF ERROR
1490     IF C>15 THEN 1590
```

```
1500     E$="Invalid ENTRY line "
1501     C$=C$[3]
1502     IF C$[1,1]=" " THEN C$=C$[2] @ GOTO 1502
1503     IF C$[LEN(C$)]=" " THEN C$=C$[1,LEN(C$)-1] @ GOTO 1503
1510     P=POS(C$," ")
1520     IF NOT P OR P>7 THEN 1590
1530     S$=C$[1,P-1] @ C$=C$[P+1]
1531     IF C$[1,1]=" " THEN C$=C$[2] @ GOTO 1531
1532     IF C$[LEN(C$)]=" " THEN C$=C$[1,LEN(C$)-1] @ GOTO 1532
1535     S$[7]=" "
1540     E$="Invalid keyword " @ P=POS(C$," ")
1550     IF LEN(C$)<2 OR P>9 THEN 1590
1560     IF P THEN C$=C$[1,P-1] @ C$[9]=""
1570     A$(K)=DTH$(T)[4]&" "&DTH$(C)[5]&" "&S$&C$ @ K=K+1
1580     GOTO 1600
1590     DISP E$;J+1
1600     OFF ERROR @ NEXT J
   -
     You now have all the available data in array a$. What's missing?
     Let's sort by token #.

     >>> In the event of missing entries, the end of the array is filled
     >>> with a null strings. Must handle.
1670 SFLAG -1 @ PURGE ASMTKN @ PURGE ASMWORDS @ CFLAG -1 @ RESTORE #2,9999
1680 CREATE TEXT ASMTKN @ ASSIGN #3 TO ASMTKN @ RESTORE #3
1681 PRINT #3;"" @ FOR J=L TO H @ IF A$(J-L+1)<>"" THEN PRINT #3;A$(J-L+1)
1682 NEXT J
1690 F=FILESZR("ASMTKN")-1 @ IF F THEN CALL QSORTF(#3,1,F,1)
1691 Z2$=CHR$(27)&">" @ Z1$=CHR$(27)&"<"
1692 FOR J=L TO H
1693     READ #3,J-L+1;C$
1694     IF HTD(C$[1,2])=J THEN 1699 ELSE DISP "No data on token "&DTH$(J)[4]
1695     C$=Z1$&DTH$(J)[4]&" "&Z2$&"F"&Z1$&" "&Z2$&"ENTRY "&Z1$&" "&Z2$&"KEYWORD "&Z1$&" "
1696     LINPUT "Token #",C$;C$ @ C$[16]=""
1697     C$=DTH$(J)[4]&" "&C$[1,1]&" "&C$[2,7]&" "&C$[8]
1698     INSERT #3,J-L+1;C$ @ PRINT #2;C$
1699 NEXT J
   -
1710 COPY ASMTKN TO ASMWORDS @ ASSIGN #4 TO ASMWORDS @ RESTORE #4
1720 F=FILESZR("ASMWORDS")-1 @ IF F THEN CALL QSORTF(#4,1,F,13)
   -

     Since the longest keyword is 8 long, provide for up to 8 swap-backs
1760 FOR J1=1 TO 8 @ FOR J=1 TO FILESZR("ASMWORDS")-2
1770         READ #4,J;Q$ @ READ #4,J+1;Q1$
1780         IF Q$[13,14]<>Q1$[13,14] THEN 1840
1790         P=POS(Q$[13],"$") @ IF NOT P THEN P=POS(Q$[13]," ")
1800         IF NOT P THEN P=9
1810         IF POS(Q1$[13],Q$[13,P+11])<>1 THEN 1840
1811         Q9$=Q$[13]
1812         IF Q9$[1,1]=" " THEN Q9$=Q9$[2] @ GOTO 1812
1813         IF Q9$[LEN(Q9$)]=" " THEN Q9$=Q9$[1,LEN(Q9$)-1] @ GOTO 1813
1816         Q8$=Q1$[13]
1817         IF Q8$[1,1]=" " THEN Q8$=Q8$[2] @ GOTO 1817
1818         IF Q8$[LEN(Q8$)]=" " THEN Q8$=Q8$[1,LEN(Q8$)-1] @ GOTO 1818
1820         IF Q9$=Q8$&"$" THEN 1840
1830         DELETE #4,J+1 @ INSERT #4,J;Q1$
1840     NEXT J @ NEXT J1
```

```
-
    We now have two files ready. ASMTKN is sorted in the order
    required for the MAIN table, and ASMWORDS is sorted in the order
    required by both the TEXT table and the SPEED table.

    We are now ready to build the rest of ASMHEAD.

    SPEED table. If S=0, we are already done.
1930 IF NOT S THEN 2070 ! go finish-up the LEX header block...
1940 PRINT #1;"*" @ PRINT #1;"*       Speed Table" @ PRINT #1;"*"
1950 PRINT #1;B$&"NIBHEX   0        Speed table exists"
1960 S$=B$&"CON(3) (TxTbEn)-(TxTbSt) " @ K=1 @ K1=FILESZR("ASMWORDS")
1970 FOR J=65 TO 90
1980     IF K<K1 THEN READ #4,K;C$ ELSE C$[13,13]=CHR$(255)
1990     K$=DTH$(K)[4]
2000     IF J<NUM(C$[13]) THEN PRINT #1;S$&CHR$(J) @ GOTO 2040
2010     IF J=NUM(C$[13]) AND K<K1 THEN PRINT #1;S$[1,18]&"En"&K$&")-(TxTbSt) "&CHR$(J) @ GOTO 2030
2020     IF J>NUM(C$[13]) THEN K=K+1 @ GOTO 1980
2030     K=K+1
2040 NEXT J
2050 PRINT #1;B$&"NIBHEX   0        Speed table exists"
2060 PRINT #1;"*" @ PRINT #1;"*"
2070 PRINT #1;B$&"CON(4)   (TxTbSt)+1-(*)  Offset to TEXT Table"
2080 IF NOT M THEN PRINT #1;B$&"CON(4)   0                  No message table"
2090 IF M THEN PRINT #1;B$&"REL(4)   MSGTBL          Offset to message table"
2100 PRINT #1;B$&"REL(5)   POLHND        Offset to poll handler"
2110 PRINT #1;"*"
2120 PRINT #1;B9$
2130 PRINT #1;"*"
2140 PRINT #1;"*       STITLE  Main Table" @ PRINT #1;"*"
   - Now, you will see why I allowed the proliferation of apparently
    duplicate files. This sneaky approach to building the main table
    is both fast and (or so I think) easy to implement...

2190 FOR J=L TO H @ READ #3,J-L+1;C$
2200     P=SEARCH(C$,1,1,9999,4) ! You could error test with IF NOT P, but no need.
2210     P$=DTH$(IP(P))[4]
2220     PRINT #1;B$&"CON(3) (TxEn"&P$&")-(TxTbSt)"
2230     PRINT #1;B$&"REL(5)  "&C$[6,11]
2240     PRINT #1;B$&"NIBHEX  "&C$[4,4] @ PRINT #1;"*"
2250 NEXT J @ PRINT #1;B9$ @ PRINT #1;"*       STITLE TEXT TABLE"
2260 PRINT #1;"*" @ PRINT #1;"TxTbSt" @ P$="01"
2270 FOR J=L TO H @ READ #4,J-L+1;C$
2280     PRINT #1;"TxEn"&P$
2290     P=J-L+2 @ P$=DTH$(P)[4]
2300     IF J=H THEN PRINT #1;B$&"CON(1) (TxTbEn)-(*)-4"
2310     IF J<H THEN PRINT #1;B$&"CON(1) (TxEn"&P$&")-(*)-4"
2311     Q9$=C$[13]
2312     IF Q9$[1,1]=" " THEN Q9$=Q9$[2] @ GOTO 2312
2313     IF Q9$[LEN(Q9$)]=" " THEN Q9$=Q9$[1,LEN(Q9$)-1] @ GOTO 2313
2320     PRINT #1;B$&"NIBASC "&CHR$(D)&Q9$&CHR$(D)
2330     PRINT #1;B$&"CON(2) #"&C$[1,2]
2340     PRINT #1;"*"
2350 NEXT J @ PRINT #1;"TxTbEn" @ PRINT #1;B$&"NIBHEX 1FF        Text table terminator"
2360 PRINT #1;B9$ @ PRINT #1;"*       STITLE POLL HANDLER"
2370 PRINT #1;"*" @ PRINT #1;"POLHND"
2380 PRINT #1;B$&"?B=0   B        VER poll?"
2390 PRINT #1;B$&"GOYES  VER$P    Yes"
```

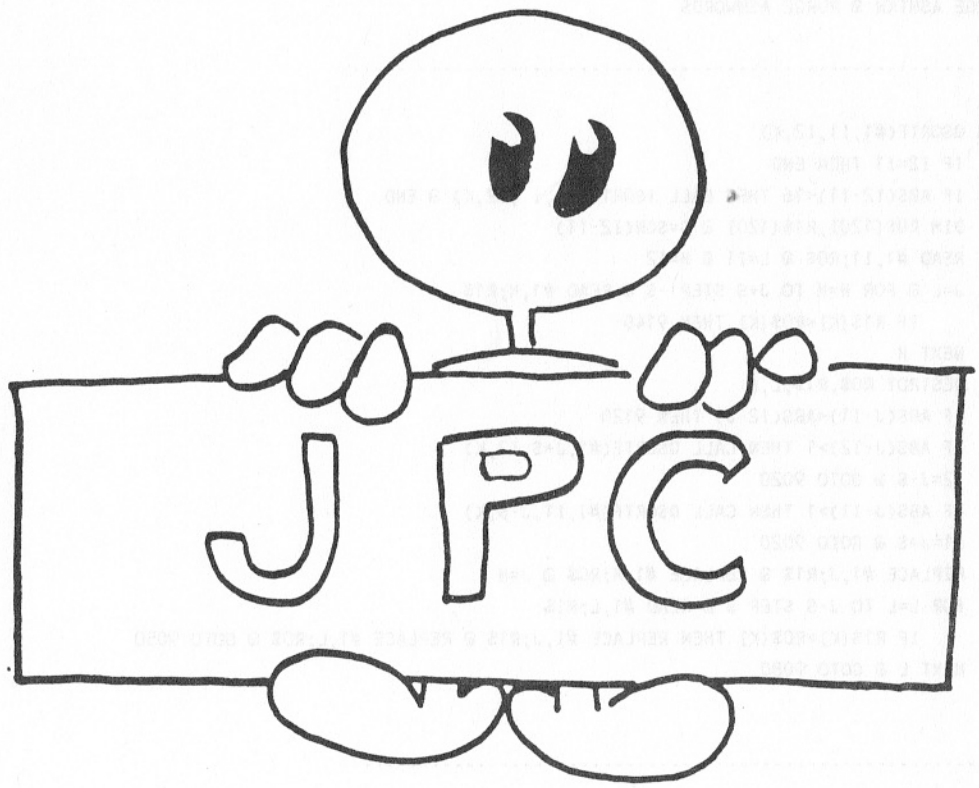
```
2400 PRINT #1;B$&"RTNSXM          No, exit XM=1, Cy=0"
2410 PRINT #1;"VER$P   A=R2            Fetch AVMEMS"
2420 PRINT #1;B$&"C=R3            Fetch stack pointer"
2430 PRINT #1;B$&"D1=C"
2440 PRINT #1;B$&"D1=D1- (VER$en)-(VER$st)-2  allow for our reply"
2450 PRINT #1;B$&"CD1EX"
2460 PRINT #1;B$&"?A>C   A        not enough memory?"
2470 PRINT #1;B$&"GOYES  VER$Pe   exit not handled Cy=1"
2480 PRINT #1;B$&"D1=C            restore stack pointer"
2490 PRINT #1;B$&"R3=C            and save for next file"
2500 PRINT #1;"VER$st"
2510 PRINT #1;B$&"LCASC   "&CHR$(D)&V$&CHR$(D)
2520 PRINT #1;"VER$en"
2530 PRINT #1;B$&"DAT1=C (VER$en)-(VER$st)-2  Write our reply"
2540 PRINT #1;"VER$Pe"
2550 PRINT #1;B$&"RTNSXM          exit"
2551 PRINT #1;"*"
2552 PRINT #1;B9$ @ PRINT #1;"*"
2553 ASSIGN #1 TO * @ ASSIGN #2 TO * @ ASSIGN #3 TO * @ ASSIGN #4 TO *
2554 PURGE ASMTKN @ PURGE ASMWORDS

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

9000 SUB QSORTF(#1,I1,I2,K)
9010     IF I2=I1 THEN END
9020     IF ABS(I2-I1)<16 THEN CALL ISORTF(#1,I1,I2,K) @ END
9030     DIM R0$[120],R1$[120] @ S=SGN(I2-I1)
9040     READ #1,I1;R0$ @ L=I1 @ H=I2
9050     J=L @ FOR H=H TO J+S STEP -S @ READ #1,H;R1$
9060         IF R1$[K]<R0$[K] THEN 9140
9070     NEXT H
9080     DESTROY R0$,R1$,L,H
9090     IF ABS(J-I1)<ABS(I2-J) THEN 9120
9100     IF ABS(J-I2)>1 THEN CALL QSORTF(#1,J+S,I2,K)
9110     I2=J-S @ GOTO 9020
9120     IF ABS(J-I1)>1 THEN CALL QSORTF(#1,I1,J-S,K)
9130     I1=J+S @ GOTO 9020
9140     REPLACE #1,J;R1$ @ REPLACE #1,H;R0$ @ J=H
9150     FOR L=L TO J-S STEP S @ READ #1,L;R1$
9160         IF R1$[K]>R0$[K] THEN REPLACE #1,J;R1$ @ REPLACE #1,L;R0$ @ GOTO 9050
9170     NEXT L @ GOTO 9080
   -

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

9190 SUB ISORTF(#1,I1,I2,K)
9200     S=SGN(I2-I1) @ IF NOT S THEN END
9210     DIM R1$[120],R2$[120],R3$[120] @ READ #1,I1;R1$
9220     FOR I=I1 TO I2-S STEP S @ READ #1,I+S;R2$
9230         IF R2$[K]>=R1$[K] THEN R1$=R2$ @ GOTO 9280
9240         DELETE #1,I+S
9250         FOR J=I-S TO I1 STEP -S @ READ #1,J;R3$ @ IF R3$[K]<=R2$[K] THEN 9270
9260         NEXT J
9270         INSERT #1,J+S;R2$
9280     NEXT I
9290 END
```


JPC

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

La maquette de ce numéro a été préparée et réalisée par Pierre David et Jean-Jacques Dhénin grâce à un système comprenant un HP71B, un lecteur de disquettes HP9114A, deux HP9807A, deux HP9154 et une imprimante LaserJet.

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

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

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

Imprimé par Copy-Express, 42 86 91 94.

ENGLISH SUMMARY
JPC 58 - OCTOBER 1988

The French part of the HP User's Library formerly in Geneva has been transfered to one of our members and we expect now it will be possible get copies of programs.

Pierre David this issue of *JPC* with a short presentation of the new range of HP calculators. Then, Eric Gengoux reviews the *Technical Application Manual* for the HP-27S. It is interesting to note that two SOLVE functions are used (L for Let and G for Get) which are not described in the User's Manual.

The HP-28C columns is a translation of an article by Thomas J. Affinito which we received from the Unix network. It is an extended workspace manager. It allows you to switch between various environment contexts.

Then, Jean-Yves Hervé gives a new HP-75 Lex file to sort integer or real arrays.

The HP-71 column begins with an article which will please our Italian friends. It describes a expert system built in Forth. As an example, a system is given to recognize *paste fresche*.

Then, Michael Markov gives us a program to build Lex file headers from a short and easy to enter description file.

Until next month,

Happy Programming and JPC reading !