JPC
JUILLET / AOUT 1993
NUMERO 88
Le numéro 40 F

A PROPOS DU CLUB

Le Bureau — Editorial — *1*

HP28

G. Toublanc — Le binôme de Newton (Acte II) — *4*
G. Toublanc — TIMED pour HP28S — *7*
J.F. Garnier — Points d'entrées compatibles HP48 (Acte II) — *7*
J.F. Garnier — Programmation System Rpl, correctif — *9*

HP48

G. Toublanc — Le binôme de Newton (Acte II bis) — *12*
G. Toublanc — Aides au désassemblage — *12*
G. Toublanc — Récupération des mnémoniques HP — *15*
G. Toublanc — REV : Nouvelle version — *15*

HP95

J. Belin — Les *Goodies Disks* HP95 — *18*
P. Guerin — Correction de Q.C.M. avec Lotus — *19*
J. Belin — Recherche de fichiers sur disque — *22*

Le coin des codes — *29*

EDITORIAL

Tout d'abord, nous avons le regret de vous annoncer que nous n'avons pas de nouvelle machine à vous présenter dans ce numéro... Et oui, on commencait à s'y habituer ! En attendant (au moins deux ou trois ans) que HP décide de sortir de nouvelles machines, il vous faudra vous contenter de celles qui existent actuellement !

Puisque nous parlons des nouvelles machines, vous devez savoir qu'elles sont disponibles depuis peu (aux USA ou chez un distributeur parisien bien connu...). Bien sûr certains d'entre nous ont déjà commencé à explorer leur possibilités et différences par rapport aux anciens modèles, mais il faudra attendre le prochain numéro pour commencer à voir des articles complets à ce sujet. Ceci explique peut-être aussi la relative finesse de la rubrique HP48.

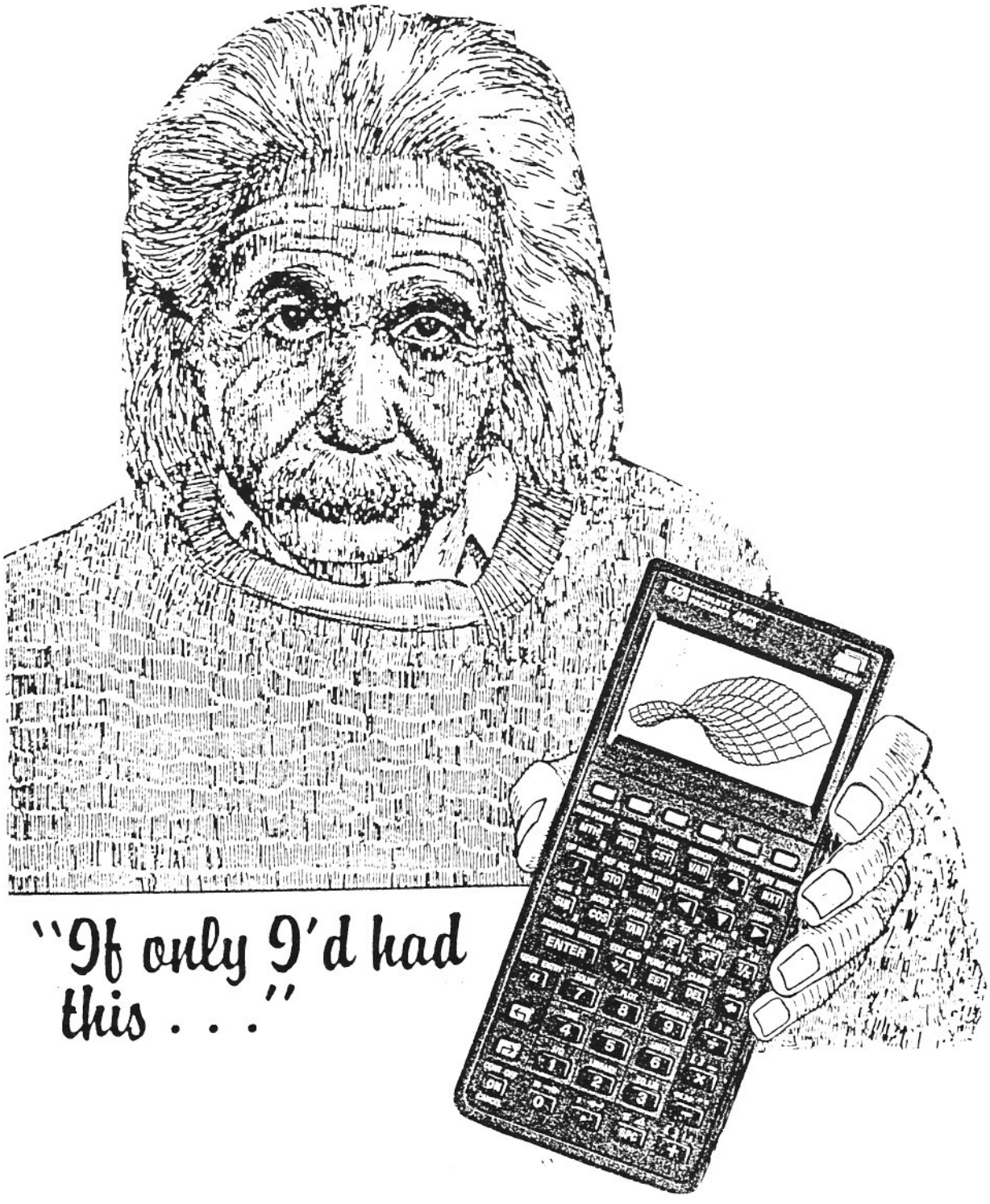
De nombreux utilisateurs de HP48 ont déjà eu le plaisir de recevoir nos fameuses disquettes vertes. Maintenant c'est enfin au tour des utilisateurs de HP95 d'avoir la possibilité d'obtenir des programmes freeware ou shareware pour leur machine préférée. Pourquoi cela a-t-il été si long ? Tout simplement parce que ces disquettes ont été préparées par le club, alors que celles destinées aux HP48 nous arrivaient toutes prêtes (merci jkh !). Il nous a donc fallu trier, tester, selectionner et compacter plusieurs centaines de programmes de toutes origines afin d'arriver à ces deux premières disquettes contenant ce que nous jugions le plus intéressant pour le HP95. Et bien sûr, cela ne s'est pas fait en deux jours, surtout que l'on devait entre temps écrire des articles pour le journal...

Les utilisateurs de HP95 sont bien servis en nouveautés ce mois-ci, car nous nous avons reçu notre premier article consacré à une feuille de calcul LOTUS. Nous espérons que cet article vous permettra de mieux maitriser cette partie souvent ignorée mais toujours intéressante de la machine, et vous inciter à nous montrer vos propres réalisations à ce sujet. Bien sûr, si nous devons continuer à passer des feuilles de calcul Lotus, nous pensons qu'il est préférable (dans l'esprit du club) de se limiter à celles qui sont optimisées pour l'affichage du HP95, comme celle que nous présentons dans ce numéro. N'oubliez pas aussi que d'autres applications internes (Le Solver de l'application CALC par exemple) peuvent permettre de créer des petites fonctions qui peuvent intéresser les autres adhérents !

Derniere nouveauté, vous trouverez à la fin du journal une nouvelle version des programmes d'entrées du *Coin des Codes* pour les HP28 et HP48. Sachez qu'il ne s'agit que d'une nouvelle présentation (sans modification majeure des algorithmes) ayant pour but de réduire le nombre de pages "constantes" dans le journal. Il n'y a donc pas obligation, pour les personnes ayant déjà une version précédente, de retaper une fois de plus ces programmes.

Nous espérons que vous profiterez des vacances pour prendre le temps de nous présenter vos petits trucs ou dernières réalisations sur votre machine préferée. Rapelez vous que si nous avons eu quelques retards pour boucler les derniers numéros, c'était souvent que parce qu'il ne nous manquait qu'un ou deux articles nous permettant d'obtenir une maquette acceptable...

En attendant le prochain numéro, ne prenez pas trop de coups de soleil !


"If only I'd had this . . ."

HP28

G. Toublanc	Le binôme de Newton (Acte II)	*4*
G. Toublanc	TIMED pour HP28S	*7*
J.F. Garnier	Points d'entrées compatibles HP48 (Acte II)	*7*
J.F. Garnier	Programmation System Rpl, correctif	*9*
	Le coin des codes	*29*

BINOME DE NEWTON

et retour aux sources

Les objectifs d'une publication comme JPC sont l'échange d'informations, d'idées et de travaux informatiques, mais d'abord pour ces derniers sur support papier avant le support magnétique ce qui peut donner une valeur pédagogique à ce journal de liaison. La rubrique *HP-28* a un rôle de premier plan à jouer dans ce domaine car le monde facile des cartes enfichables est refusé à nos HP-28.

Durant sa déjà longue vie, JPC a permis à plus d'un programmeur débutant de progresser grâce aux articles des prédécesseurs.

La vue d'une nouvelle signature est toujours pour moi un réel plaisir. Le travail d'Edmond Dougados sur les binômes de Newton m'a intéressé à plusieurs titres. En effet cela confirme le fait que tout le monde n'utilise pas les programmes universels mais volumineux, que des petits programmes spécifiques peuvent satisfaire certains, et enfin qu'il y a encore des possesseurs de HP-28 qui se lancent dans la création. Aussi le but de cet article est d'encourager l'auteur à nous concocter d'autres programmes et lui apporter l'aide de quelqu'un qui n'a que le mérite d'avoir fait ses débuts avant.

Rappelons le programme DEVL (*JPC 87 page 6*) se rapportant au développement du binôme de Newton et que nous allons essayer d'améliorer.

```
«
  1 FIX → a b n                                    @ 0
  «
    0 'POLY.' STO 0 'bc' STO -1 n
    FOR t t EVAL 'T' STO                           @ 1
      IF -1 t == THEN 0 'T' STO END                @ 2
      bc 'n-T' EVAL DUP 'bc' STO                   @ 3
      IF T 0 > THEN * DUP 'bc' STO END             @ 4
      T 1 + 'ABX' STO
      IF t -1 == THEN n / ELSE ABX FACT / END      @ 5
      IF t -1 == THEN 1 'r' STO                    @ 6
      ELSE t EVAL 2 + 'r' STO END                  @ 7
      'a^(n-r+1)' EVAL 'b^(r+1-2)' EVAL            @ 8
      * * POLY. SWAP + 'POLY.' STO
    NEXT POLY. DUP 'POLY.' PURGE
    { T bc ABX r } PURGE 2 ROLL DROP 2 ROLL DROP   @ 9
    SWAP DROP 1500 .5 BEEP
  »
»
```

1ière **étape** : quelques améliorations de forme.

- ligne 0 : `STD`
  au lieu de `1 FIX` car les coefficients sont entiers.
- ligne 1 : `EVAL` non nécessaire car *t* renvoie 1 réel et on ne stocke pas tout de suite *t*.
- ligne 2 : `DUP -1 ≠ * 'T' STO`
  au lieu de : `IF -1 t == THEN 0 'T' STO END`.
  Le résultat est le même.
- ligne 3 : `bc n T -`
  au lieu de : `bc 'n-T' EVAL DUP 'bc' STO`
  Formulation Rpl et non algébrique et on retarde le stockage.
- ligne 4 : `IF T THEN * END DUP 'bc' STO`
  au lieu de : `IF T 0 > THEN * DUP 'bc' STO END`
  T renvoie 0 ou 1 réel > 0 test possible.
- ligne 5 : `IF .. THEN .. ELSE .. END /`
  au lieu de : `IF .. THEN .. / ELSE .. / END`
- lignes 6 et 7 : `t 2 + 'r' STO`
  au lieu de :
  `IF t -1 == THEN 1 'r' STO ELSE t EVAL 2 + 'r' STO END`
  Le résultat est le même.
- ligne 8 : `'a' n r - 1 + ^`
  au lieu de : `'a^(n-r+1)'`
  ce qui est moins rapide et plus gourmand en octets.
  `'b' r 1 - ^` au lieu de `'b^(r+1-2)'`
- ligne 9 : `SWAP` au lieu de `2 ROLL`

Ces améliorations nous font passer d'un programme de 575 octets à 424 octets soit 35% en plus pour la version d'Edmond.

Voici le programme allégé.

```
« STD → a b n
  « 0 'POLY.' STO 0 'bc' STO -1 n
    FOR t t DUP -1 ≠ * 'T' STO bc n T -
      IF T THEN * END
      DUP 'bc' STO T 1 + 'ABX' STO
      IF t -1 == THEN n ELSE ABX FACT END
      / t 2 + 'r' STO 'a' n r - 1 + ^ EVAL 'b' r 1 -
      ^ EVAL * * POLY. SWAP + 'POLY.' STO
    NEXT POLY. 'POLY.' PURGE { T bc ABX r } PURGE
    SWAP DROP SWAP DROP 1500 .5 BEEP
  »
»
```

2ième **étape** : remplacement des variables globales par des variables locales.

Cette façon de faire évitera les interférences éventuelles avec les variables globales des autres programmes et d'avoir à purger ces variables.

D'autre part nous réduirons les noms à une seule lettre par mesure d'économie (si cela est moins

explicite, le résultat sera une économie toujours appréciable pour la mémoire de nos HP-28.

```
bc    → C
ABX   → M
POLY. → P
```

Jusqu'ici c'est toujours le programme d'Edmond mais perdant encore 79 octets pour ne garder que 345 octets :

```
« STD 0 0 0 DUP2 → a b n P C M T r
  « -1 n
    FOR t
      t DUP -1 ≠ * 'T' STO n T -
      IF T THEN C * END
      DUP 'C' STO T 1 + 'M' STO
      IF t -1 == THEN n ELSE M FACT END
      / t 1 + 'r' STO 'a' n r - ^ EVAL          @**
      'b' r ^ EVAL * * P SWAP + 'P' STO
    NEXT P 1500 .5 BEEP
  »
»
```

A la ligne ** une amélioration :
`t 1 +` au lieu de `t 2 +` ce qui permet de faire:
`'a' n r -` au lieu de `'a' n r - 1 +` et
`'b' r` au lieu de `'b' r 1 -`

3ième étape : abandon des variables locales mais travail avec la pile.

Nous abandonnons 7 variables locales et gardons seulement C. Le listing est moins explicite mais nous gagnons encore 107.5 octets ne gardant que 237.5 octets et le programme est 3 fois et demi plus rapide que l'original.

```
« STD 0 → C
  « 0 -1 3 PICK
    FOR t OVER t DUP 0 > * DUP2 -
      IF OVER THEN C * END
      DUP 'C' STO SWAP
      IF t 0 < THEN DROP OVER ELSE 1 + FACT END
      / t 1 + 7 PICK 4 ROLL 3 PICK - ^ 6 PICK
      ROT ^ * * +
    NEXT 4 ROLLD 3 DROPN 1500 .5 BEEP
  »
»
```

4ième étape : on efface tout et on recommence.

L'algorithme d'Edmond consiste à calculer les coefficients du binôme de Newton en utilisant le coefficient précédent. Nous pouvons utiliser la fonction COMB puisque chaque coefficient peut se calculer par C_n^p ce qui simplifie beaucoup les choses.

Encore 123.5 octets en moins pour tomber à 114 octets et un programme 5 fois plus court mais aussi plus de 5 fois plus rapide que l'original.

```
« STD 0 0 3 PICK
  FOR t OVER t COMB 5 PICK 4 PICK t - ^
    * 4 PICK t ^ * +
  NEXT 4 ROLLD 3 DROPN 1500 .5 BEEP
»
```

5ième étape : le luxe

Puisque notre programme est devenu léger nous pourrions lui en demander plus : calculer aussi $(A - B)^n$.

C'est ce que fait BINIOU ci-dessous.

Exemples :

```
1ère variable      : 'A' ← niveau 3
2ème variable      : 'B' ← niveau 2
le degré du binôme :  3  ← niveau 1

  BINIOU   →   'A^3+3(A^2*B)+3*(A*B^2)+B^3'
```

et avec '-B'

```
  BINIOU   →   'A^3-3(A^2*B)+3*(A*B^2)-B^3'
```

Voici le programme final pour HP-28. Avec ses commentaires, j'espère qu'il sera utile à quelques uns et peut-être permettra d'éclaircir certains points.

BINIOU
183.5 octets cksum # DABBh

Les commentaires partent des exemples:
$(A + B)^n$ ou $(A - B)^n$
donc au départ :
niveau 3 : 'A'
niveau 2 : 'B' ou '-B'
niveau 1 : n

```
« STD 1                  @ 1 pour le signe
 3 PICK                  @ duplique 'B' ou '-B'
 IF 1 OBGET { NEG } ==   @ si '-B'
 THEN ROT NEG 3 ROLLD    @ change '-B' en 'B'
 NEG                     @ et 1 en -1 pour le signe
 END SWAP                @ échange n et signe
 0                       @ initialise polynôme à 0
 0                       @ 0 départ de boucle
 3 PICK                  @ n pour fin boucle
 FOR t                   @ boucle de t = 0 à n
                         @ ici nous avons :
                         @*************************
                         @ niveau 5 : 'A'
                         @ niveau 4 : 'B' ou '-B'
```

```
                    @ niveau 3 :  signe
                    @ niveau 2 :  n
                    @ niveau 1 :  polynôme
                    @*************************
 OVER t COMB        @ n t → Cn^t
 4 PICK             @ signe
 t ^ *              @ Cn^t * (signe)^t
 6 PICK             @ 'A'
 4 PICK             @ n
 t - ^              @ n - t → 'A^(n-t)'
 6 PICK             @ 'B'
 t ^                @ 'B^t'
 * *                @ '±Cn^t*(A^(n-t)*B^t)'
 +                  @ polynôme + ce terme
NEXT                @ tour de boucle suivant
5 ROLLD             @ polynôme → niveau 5
4 DROPN 1500 .5 BEEP @ drope 4 premiers niveaux
»
```

Pour HP-28 et HP-48...

Pour terminer voici un résumé des idées exposées ci-dessus et qui peuvent aider les débutants qui ont quelques difficultés à produire des programmes optimisés en encombrement mémoire et vitesse :

- choisir pour les débuts un problème avec peu de paramètres. Le développement du binôme de Newton est un bon exemple.
- rechercher l'algorithme le plus simple en faisant appel aux fonctions les plus puissantes de la HP.
- si le problème est de nature mathématique, traiter dans un premier temps la programmation sous forme algébrique en suivant la ou les formules.
- pour débuter faire une programmation facile avec des noms de variables évocateurs pour bien suivre le déroulement du programme.
- donner la préférence aux variables locales, qu'il n'est pas besoin de purger lorsque le programme s'est déroulé, réservant les variables globales pour des sauvegardes définitives.
- avec cette ébauche vérifier le bon fonctionnement du programme.

A partir de maintenant noter lors de l'évolution de la programmation :

- l'encombrement mémoire pour chaque version avec `BYTES` de la HP-48 ou `BYTES` pour HP-28 (JPC-85 p9).
- la vitesse d'exécution avec `TIMED` pour HP-48 (JPC-82 page 26) ou `TIMED` pour HP-28 dans ce numéro de JPC.

Commencer l'optimisation:

- Supprimer les séquences de programmes dupliqués du genre `IF THEN ... toto ELSE ... toto END` en faisant `IF THEN ... ELSE ... END toto`.
- Si possible éviter les suites de mêmes tests en regroupant les différentes clauses à la suite du test :
  `IF bibi THEN toto1 END ... IF bibi THEN toto2 END`
  devrait être `IF bibi THEN toto1 ... toto2 END`.
- Tenir compte des valeurs possibles des variables lors des tests :
  `IF toto NOT THEN` au lieu de `IF toto == 0 THEN`
  `IF toto THEN` au lieu de `IF toto > 0 THEN`
  si *toto* ne peut prendre que des valeurs nulles ou positives
- Voir s'il est possible de remplacer des tests par des expressions :
  `toto DUP -1 0 > *` au lieu de :
  `IF toto == -1 THEN 0 ELSE toto END`
  si l'on sait que toto renvoie des valeurs > -2
- Eviter le stockage d'une valeur dans une variable et son rappel trop rapproché, utiliser à la place les mouvements de la pile.
- Passer de la programmation type *algébrique* à celle de type *Rpl* en notant sur une feuille les divers mouvements de la pile pour suivre l'évolution du programme.
- Eliminer progressivement les variables locales au bénéfice des mouvements de pile: SWAP, ROT etc.. en ne gardant que les seules variables locales rendant la programmation plus efficace que celle des mouvements de la pile.

Toutes ces modifications devraient se traduire par moins d'octets occupés et plus de rapidité.

Ces suggestions ne sont que quelques recettes et ne prétendent pas être la solution à tous les problèmes d'optimisation dont certains hpistes ont l'art de les résoudre. Dans ce domaine il n'est pas interdit qu'un lecteur de JPC lance un défi pour réaliser la meilleure optimisation d'un petit programme.

Exemples complémentaires d'optimisation en User Rpl :
- Chronique de notre ami Robert Pulluard tenue dans *Haute Performance* et dans son bulletin *48sxtant* dont la rubrique *Dr. Sexagésimus a un problème* est réservée aux problèmes d'optimisation.
- Article du signataire: *les différents modes de programmation. Le pour et le contre. (JPC 81 p 6).*

En conclusion, nous souhaitons recevoir beaucoup de petits programmes comme celui qui a provoqué cet article. Que ceux qui n'osent pas se lancer n'hésitent plus à nous envoyer leurs travaux: il y aura toujours quelqu'un à qui cela profitera.

Guy Toublanc (276)

BON TEMPS ET BONS PROGRAMMES HP-28S

Dans l'article de ce même *JPC, Binômes de Newton*, je faisais allusion aux problèmes d'optimisation de programmes. Pour tester l'efficacité d'une programmation il nous faut deux outils :

BYTES (*JPC 85 page 9*) et TIMED que je vous présente.

Ce dernier est pour HP-28S seulement car faisant appel au *System Rpl* pour des raisons de gain de de mémoire. C'est une transposition de ce que j'avais fait pour HP-48, aussi je vous renvoie à cette version (*JPC 82 page 26*) commentée pour la justification des mnémoniques employés. Ce programme donne des résultats plus précis et fiables qu'un chronométrage avec une montre.

TIMED
80.5 octets cksum # B933h

Syntaxe :

```
niveaux 2 3 ... : les arguments éventuels
niveau 1        : le programme ou son nom

résultat        : le temps d'exécution en secondes
```

Voici le fichier source :

```
ASSEMBLE
      con(5)    #02c67          * DOCOL
      con(5)    #075dc          * xDUP
      con(5)    #3e7f1          * ?TYPEIDNT
      con(5)    #3dc94          * IT
      con(5)    #0a1f7          * XEQRCL
      con(5)    #0a32b          * xMEM
      con(5)    #31a84          * WORDSIZE
      con(5)    #115dc          * SIXTYFOUR
      con(5)    #316f7          * dostws
      con(5)    #02a96          * DOLIST
      con(5)    #c9d86          * NULLAM
      con(5)    #c9d86          * NULLAM
      con(5)    #02f90          * SEMI
      con(5)    #059b5          * BIND
      con(5)    #011ca          * CLKTICKS
      con(5)    #3daba          * 2PUTLAM
      con(5)    #0a418          * xEVAL
      con(5)    #011ca          * CLKTICKS
      con(5)    #3d911          * 2GETLAM
      con(5)    #31913          * #-
      con(5)    #31e22          * #>%
      con(5)    #1a173          * %8192
      con(5)    #11d82          * %/
      con(5)    #02933          * DOREAL
      nibhex    8990000000003210 * .00375
      con(5)    #11d1d          * %-
      con(5)    #3d8d7          * 1GETLAM
      con(5)    #316f7          * dostws
      con(5)    #05973          * ABND
      con(5)    #02f90          * SEMI
```

Vous trouverez la liste des codes dans *Le coin des codes*.

Bonne optimisation.

Guy Toublanc (276)

POINTS D'ENTREES HP28 ACTE II

Voici la fin de la liste des points d'entrées équivalents entre la HP28 et la HP48. Rapelons qu'ils ne concernent que la HP28S version 2BB.

```
=xSTOSIGMA    EQU #0ABBC *  (user RPL)
=xCLSIGMA     EQU #0ABD6 *  (user RPL)
=xRCLSIGMA    EQU #0ABF0 *  (user RPL)
=xSIGMA+      EQU #0AC0A *  (user RPL)
=xSIGMA-      EQU #0AC38 *  (user RPL)
=xNSIGMA      EQU #0AC52 *  (user RPL)
=xCORR        EQU #0AC6C *  (user RPL)
=xCOV         EQU #0AC86 *  (user RPL)
=xMAXSIGMA    EQU #0ACA0 *  (user RPL)
=xMEAN        EQU #0ACBA *  (user RPL)
=xMINSIGMA    EQU #0ACD4 *  (user RPL)
=xSDEV        EQU #0ACEE *  (user RPL)
=xTOT         EQU #0AD08 *  (user RPL)
=xVAR         EQU #0AD22 *  (user RPL)
=xLR          EQU #0AD3C *  (user RPL)
=xPREDV       EQU #0AD56 *  (user RPL)
=xUTPC        EQU #0AD7A *  (user RPL)
=xUTPN        EQU #0AD9E *  (user RPL)
=xUTPF        EQU #0ADC2 *  (user RPL)
=xUTPT        EQU #0ADE6 *  (user RPL)
=xSCLSIGMA    EQU #0AE0A *  (user RPL)
=xDRWSIGMA    EQU #0AE24 *  (user RPL)
=xCOLSIGMA    EQU #0AE3E *  (user RPL)
=xSINV        EQU #0AE62 *  (user RPL)
=xSNEG        EQU #0AEF9 *  (user RPL)
=xSCONJ       EQU #0AF90 *  (user RPL)
=xSTO+        EQU #0B013 *  (user RPL)
=xSTO-        EQU #0B118 *  (user RPL)
=xSTO/        EQU #0B222 *  (user RPL)
=xSTO*        EQU #0B34F *  (user RPL)
=xEXGET       EQU #0B5C6 *  (user RPL)
```

```
=xEXSUB       EQU #0B5EA *  (user RPL)
=xOBSUB       EQU #0B622 *  (user RPL)
=xOBGET       EQU #0B646 *  (user RPL)
=xFORM        EQU #0B66A *  (user RPL)
=xCOLCT       EQU #0B6A2 *  (user RPL)
=xEXPAN       EQU #0B6DA *  (user RPL)
=xISOL        EQU #0B712 *  (user RPL)
=xQUAD        EQU #0B736 *  (user RPL)
=xSHOW        EQU #0B75A *  (user RPL)
=xTAYLR       EQU #0B788 *  (user RPL)
=xDER         EQU #0B7B6 *  (user RPL)
=xRCEQ        EQU #0B84D *  (user RPL)
=xSTEQ        EQU #0B871 *  (user RPL)
=xROOT        EQU #0B88B *  (user RPL)
=xINTEGRAL    EQU #0B8E1 *  (user RPL)
=xASR         EQU #0BB53 *  (user RPL)
=xRL          EQU #0BB77 *  (user RPL)
=xRLB         EQU #0BB9B *  (user RPL)
=xRR          EQU #0BBBF *  (user RPL)
=xRRB         EQU #0BBE3 *  (user RPL)
=xSL          EQU #0BC07 *  (user RPL)
=xSLB         EQU #0BC2B *  (user RPL)
=xSR          EQU #0BC4F *  (user RPL)
=xSRB         EQU #0BC73 *  (user RPL)
=xR>B         EQU #0BC97 *  (user RPL)
=xB>R         EQU #0BCBB *  (user RPL)
=xCONVERT     EQU #0BCDF *  (user RPL)
=xINDEP       EQU #0BD21 *  (user RPL)
=xPMIN        EQU #0BD45 *  (user RPL)
=xPMAX        EQU #0BD69 *  (user RPL)
=xAXES        EQU #0BD8D *  (user RPL)
=xCENTR       EQU #0BDB1 *  (user RPL)
=xRES         EQU #0BDD5 *  (user RPL)
=x*H          EQU #0BDF9 *  (user RPL)
=x*W          EQU #0BE1D *  (user RPL)
=xDRAW        EQU #0BE41 *  (user RPL)
=xPIXEL       EQU #0BE5B *  (user RPL)
=xDRAX        EQU #0BE7F *  (user RPL)
=xPR1         EQU #0BE99 *  (user RPL)
=xPRSTC       EQU #0BEB3 *  (user RPL)
=xPRST        EQU #0BECD *  (user RPL)
=xCR          EQU #0BEE7 *  (user RPL)
=xPRUSR       EQU #0BF01 *  (user RPL)
=xPRVAR       EQU #0BF1B *  (user RPL)
=xPRMD        EQU #0BF5D *  (user RPL)
=xPRLCD       EQU #0BF77 *  (user RPL)
=xCRDIR       EQU #0BF91 *  (user RPL)
=xPATH        EQU #0BFB5 *  (user RPL)
=xHOME        EQU #0BFCF *  (user RPL)
=xVARS        EQU #0BFE9 *  (user RPL)

=UNCOERCE2    EQU #0C1FB *  (prog)
=CK0NOLASTWD  EQU #0C361 *  (primitive)
=CK0          EQU #0C36A *  (primitive)
=CK3          EQU #0C3B0 *  (primitive)
=CK2NOLASTWD  EQU #0C3C4 *  (primitive)
=CK2          EQU #0C3CD *  (primitive)
=CK1NOLASTWD  EQU #0C3E1 *  (primitive)
=CK1          EQU #0C3EA *  (primitive)
=CKN          EQU #0C484 *  (primitive)
=SETSIZEERR   EQU #0C506 *  (primitive)
=UNCOERCE     EQU #0C610 *  (primitive)
=COMPEVAL     EQU #0C784 *  (primitive)
=Dispatch2    EQU #0C79C *  (primitive)
=Dispatch1    EQU #0C7A5 *  (primitive)
=COERCE       EQU #0C53B *  (primitive)
=TestUserFlag EQU #0C8DC *  (primitive)
=COERCEFLAG   EQU #0C9F4 *  (primitive)
=#*OVF        EQU #0CA29 *  (primitive)

=xIF          EQU #0E3E2 *  (user RPL)
=xTHEN        EQU #0E3F8 *  (user RPL)
=xELSE        EQU #0E459 *  (user RPL)
=xIFEND       EQU #0E479 *  (user RPL)
=xALG->       EQU #0E48F *  (user RPL)
=xWHILE       EQU #0E4D2 *  (user RPL)
=xREPEAT      EQU #0E4F2 *  (user RPL)
=xDO          EQU #0E535 *  (user RPL)
=xUNTIL       EQU #0E555 *  (user RPL)
=xSTART       EQU #0E56B *  (user RPL)
=xSTARTVAR    EQU #0E5B3 *  (user RPL)
=xNEXT        EQU #0E60F *  (user RPL)
=xSTEP        EQU #0E73E *  (user RPL)
=xIFERR       EQU #0E77A *  (user RPL)
=xHALT        EQU #0E842 *  (user RPL)
=xSILENT'     EQU #0E862 *  (user RPL)
=xRPN->       EQU #0E87D *  (user RPL)
=x>>ABND      EQU #0E9B5 *  (user RPL)
=x<<          EQU #0E9D0 *  (user RPL)
=x>>          EQU #0E9E6 *  (user RPL)
=xWHILEEND    EQU #0EA2D *  (user RPL)
=xENDDO       EQU #0EA4D *  (user RPL)
=xERRTHEN     EQU #0EA90 *  (user RPL)

=CKOATTNABORT EQU #0EAD9 *  (prog)
=palparse     EQU #0EAF7 *  (prog)
=DaDGNTc      EQU #0FDF0 *  (prog)
=nultrior     EQU #0FF21 *  (primitive)
=GetNextToken EQU #10D39 *  (prog)
=%%>%         EQU #11280 *  (primitive)
=%>%%         EQU #1129B *  (primitive)
=%0           EQU #112B4 *  (real)
=%1           EQU #112C9 *  (real)
=%2           EQU #112DE *  (real)
=POP1%        EQU #1160E *  (assembly)
=PUSH1%       EQU #117A9 *  (assembly)
=%MAX         EQU #11A01 *  (prog)
=%MIN         EQU #11A24 *  (prog)
=%0=          EQU #11A8B *  (primitive)
=%0<>         EQU #11AEF *  (prog)
=%<           EQU #11B97 *  (primitive)
=%>           EQU #11BB0 *  (primitive)
=%>=          EQU #11BC6 *  (primitive)
=%=           EQU #11BF2 *  (primitive)
=%/           EQU #11D82 *  (primitive)
=%IP          EQU #12474 *  (primitive)
```

```
=%FACT          EQU #125F6 *  (prog)
=SYNTAXERR      EQU #183A1 *  (primitive)
=ErrjmpC        EQU #1848d *  (assembly)
=SAFE@          EQU #1E737 *  (prog)
=DPRADIX?       EQU #1EE18 *  (prog)
=DO>STR         EQU #227BD *  (prog)
=DODISP         EQU #227DB *  (prog)
=DOCHR          EQU #229C3 *  (prog)
=DOSTR>         EQU #22A22 *  (prog)
=DOBEEP         EQU #22A45 *  (prog)
=setbeep        EQU #22AA7 *  (primitive)
=makebeep       EQU #22AD3 *  (assembly)
=nohalt         EQU #22D12 *  (locname)
=DOLCD>         EQU #25133 *  (prog)
=DO>LCD         EQU #2518D *  (prog)
=AND$           EQU #28215 *  (prog)
=OR$            EQU #28229 *  (prog)
=XOR$           EQU #2823D *  (prog)
=bitAND         EQU #31760 *  (primitive)
=bitOR          EQU #31771 *  (primitive)
=bitXOR         EQU #31782 *  (primitive)
=bitNOT         EQU #317AA *  (primitive)

=#>%            EQU #31E22 *  (primitive)
=%>#            EQU #31EC3 *  (primitive)
=cknumdsptch1   EQU #32B63 *  (prog)
=sscknum2       EQU #340A3 *  (prog)
=sncknum2       EQU #340BC *  (prog)
=nscknum2       EQU #340D5 *  (prog)
=top&           EQU #3AD91 *  (primitive)
=pshtop&        EQU #3AE39 *  (prog)
=psh            EQU #3B4F2 *  (primitive)
=ROTSWAP        EQU #3D57F *  (primitive)
=ROTDROP        EQU #3D5B9 *  (primitive)
=SWAPDROP       EQU #3D61C *  (primitive)
=4UNROLL        EQU #3D72C *  (primitive)
=1GETLAM        EQU #3D8D7 *  (primitive)
=2GETLAM        EQU #3D911 *  (primitive)
=3GETLAM        EQU #3D938 *  (primitive)
=casedrop       EQU #3DC14 *  (primitive)
=caseDROP       EQU #3DC4A *  (primitive)
=case           EQU #3DC5F *  (primitive)
=NOTcase        EQU #3DC85 *  (primitive)
=IT             EQU #3DC94 *  (primitive)
=popflag        EQU #3DCF9 *  (assembly)
=?SEMI          EQU #3DD32 *  (primitive)
='RSAVEWORD     EQU #3EB63 *  (primitive)
=tok'           EQU #3F1C6 *  (string)

=TEMPTOP        EQU #C005A =  (assembly)
=DSKTOP         EQU #C0064 =  (assembly)
=TEMPENV        EQU #C006E =  (assembly)
=TOUCHTAB       EQU #C0078 =  (assembly)
=UserFlags      EQU #C010F =  (assembly)
```

Jean François Garnier (242)

SYSTEM RPL

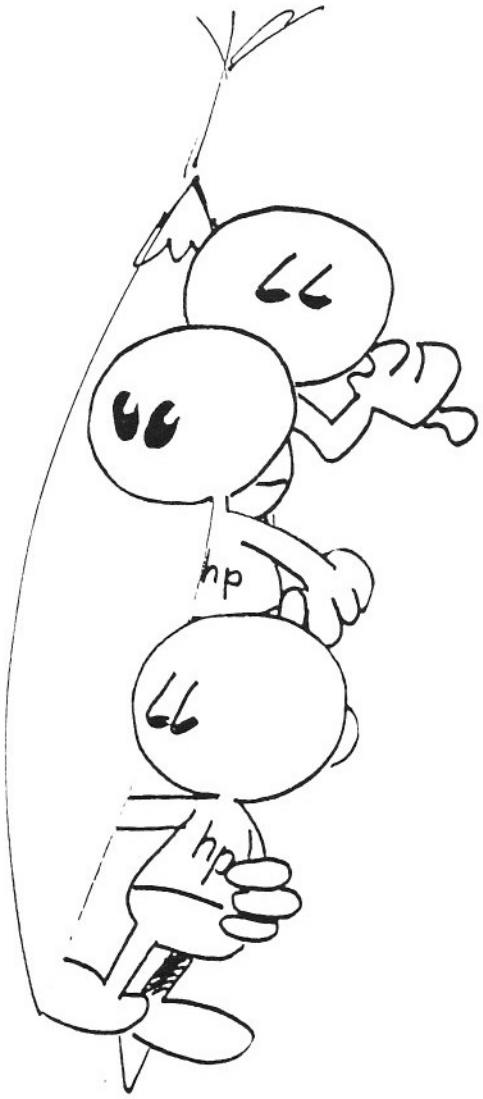
ACTE II

Dans mon article *Sytem Rpl sur HP28S* (JPC87), deux erreurs se sont glissées :

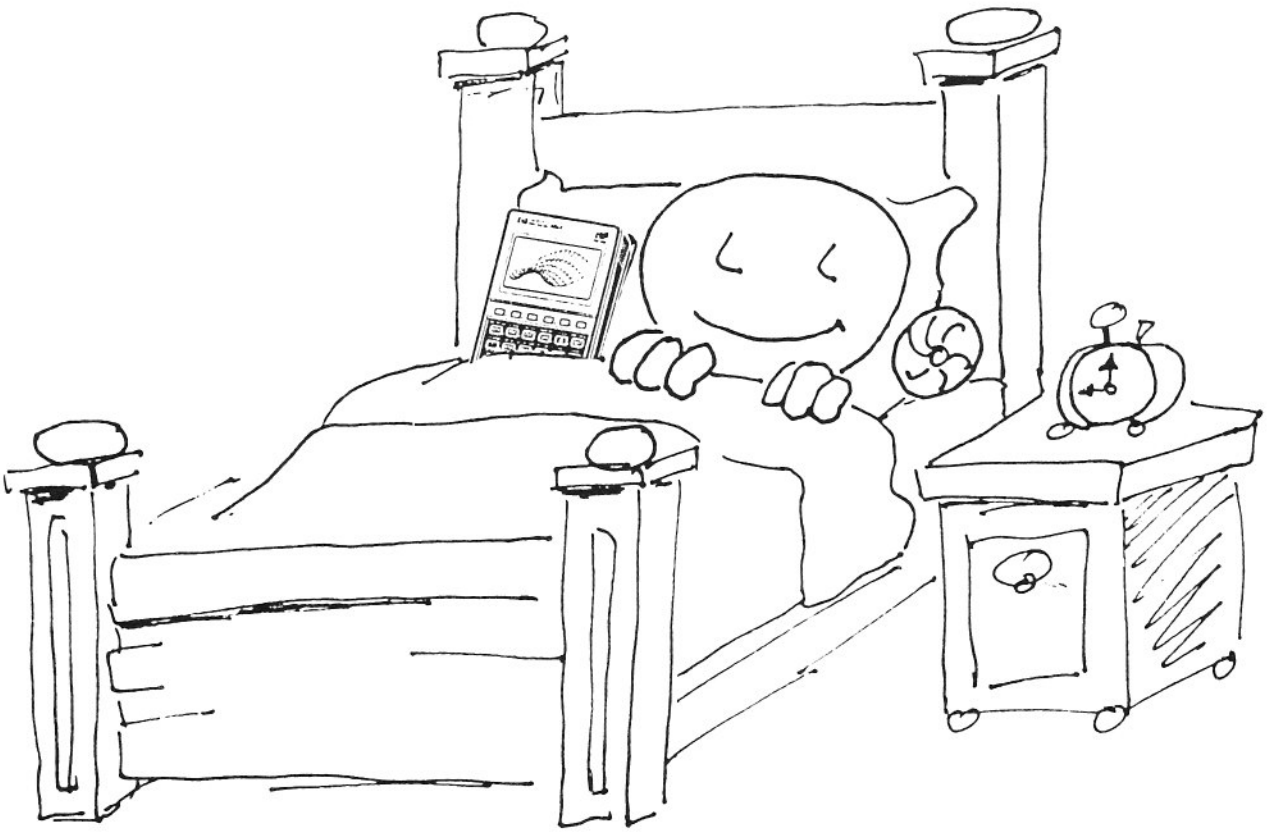
- La commande pour obtenir la version de la 28S n'est pas `#10h SYSEVAL`, mais `#Ah SYSEVAL`. le Syseval erronné a de grandes chances de provoquer un `MEMORY LOST`.

- Dans la routine `SB→B`, remplacer `B→R` par `R→B`.

Jean-Francois Garnier (242)



Je ne sais pas ce qu'a cette machine
mais elle est de plus en plus
affectueuse



HP48

G. Toublanc	Le binôme de Newton (Acte II bis)	*12*
G. Toublanc	Aides au désassemblage	*12*
G. Toublanc	Récupération des mnémoniques HP	*15*
G. Toublanc	REV : Nouvelle version	*15*
	Le coin des codes	*29*

BINOME DE NEWTON
ou les bonnes relations 28-48

Côté HP-28 vous trouverez dans l'article *binôme de Newton* des éléments communs aux deux types de machine. Si vous êtes intéressé par ce sujet ou seulement par les problèmes d'optimisation de programmes, vous êtes invité à vous reporter à l'article cité pour le mode d'emploi, les méthodes d'optimisation, et les commentaires complets du programme résultat.

Voici le listing concernant les HP-48. Juste le début diffère de celui pour HP-28 et est commenté.

BINIOU
176 octets cksum # FD1Dh

```
«
  STD 1                    @ 1 pour le signe
  3 PICK                   @ duplique 'B' ou '-B'
  IFERR OBJ→               @ si 'B' OBJ→ alors erreur
  THEN DROP                @ drop 'B'
  ELSE 3 DROPN             @ sinon drope résultat de
                           @ OBJ→ (3 niveaux)
  ROT NEG 3 ROLLD NEG      @ pour la suite voir
  END SWAP 0 0 3 PICK      @ rubrique HP-28
  FOR t  OVER t COMB 4 PICK
    t ^ *  6 PICK 4 PICK
    t - ^  6 PICK  t ^  * * +
  NEXT  5 ROLLD  4 DROPN 1500 .5 BEEP
»
```

Guy Toublanc (276)

DECOMPILATION ET SEGREGATION

Les programmes objet de cet article devraient grossir la rubrique *Tucs, astuces et petits utilitaires* car il sont courts et peuvent être utiles. Il font suite aux deux mini-programmes de cette rubrique du *JPC* de juin car ils nécessitent la librairie ←RPL→ de l'excellent *RPL Toolkit* de Detlef Mueller et R.H (goodies disk 8).

La décompilation d'un programme constitué à la fois de Rpl et d'assembleur nous donne avec RPL→ une chaîne listing des mnémoniques *Rpl* et pour l'assembleur CODE n ccc....ccc où *n* est un réel pour la longueur du code et *ccc...ccc* est la suite des codes hexadécimaux de la partie code. Pour obtenir le listing complet de désassemblage il faut éditer la chaîne, extraire les parties de codes hexa, avec chaque sous-chaîne de codes et réassembler avec ←RPL puis chaque objet code doit être désassemblé avec COD→ pour avoir le listing avec les mnémoniques assembleur. Ensuite il faut remplacer les blocs de codes hexa par les blocs listing assembleur dans le listing Rpl. Tout cela n'est pas simple et peut être source d'erreurs.

Aussi je vous propose deux programmes automatisant ce travail. D'autre part les listing des fichiers sources avec leurs commentaires constituent une initiation à la programmation en System Rpl pour ceux qui ont du mal à débuter.

Nous allons procéder en deux temps :

1- PROGDEC génère une chaîne listing avec les mnémoniques de la partie System Rpl ou User Rpl et insère des marques "CODE_i" pour signaler les blocs de code assembleur d'ordre *i*. Ces blocs sont stockés dans des variables 'Codei' correspondantes. Cette façon de faire permet de désassembler des programmes importants avec moins de problèmes de mémoire. En effet une chaîne de code hexa occupe deux fois plus d'octets que l'objet en mémoire. Maintenant il vous est possible de lire le listing de désassemblage Rpl et de repérer les éventuels blocs de code assembleur.Vous pouvez aussi, pour chaque bloc, désassembler avec COD→ les objets code contenus dans les variables 'Codei' pour les lister et les étudier individuellement.

2- Si le programme comprend des parties assembleur, ALLDEC permet d'avoir le listing complet Rpl et assembleur de désassemblage. Ce programme remplace dans le listing Rpl chaque marque "CODE_i" par le listing de désassemblage des objets code stockés dans les variables 'Codei'. A ce stade vous pouvez éditer le listing, éventuellement modifier et puis réassembler le tout avec →RPL.

Les commentaires devraient suffir à suivre la programmation. Pour les débutants je signale des différences avec les mêmes programmes écrits en User Rpl :

- Utilisation des *System Binary* (SB) à la place des réels ce qui est moins octivore et d'exécution plus rapide.

- Utilisation de points d'entrée uniques remplaçant plusieurs instructions *User Rpl* :

L'équivalent de :
`1 n START ... NEXT` ou `1 n FOR i ... NEXT`
est :
`n+1 1 DO ... LOOP`
avec 1 et n+1 en system binary
L'équivalent de :
`DO ... UNTIL ... END`
est:
`BEGIN ... UNTIL`
L'équivalent de :
`WHILE ... REPEAT ... END`
est:
`BEGIN ... WHILE ... REPEAT`

- Utilisation de variables locales de nom de longueur nulle d'où gain d'octets et qui se rappellent par `nGETLAM` ou se chargent par `nPUTLAM` d'exécution plus rapide et très économes en mémoire.
- L'appel à des programmes de librairies ou XLIB se fait par `ROMPTR m n` où *m* est le numéro de librairie en hexa et *n* est le numéro de *XLIB*.

Pour plus d'informations voyez le manuel de *RPLCOMP* (goodies disk 4) et aussi certaines informations trouvées dans les *Secrets de la HP48* de J.M.Ferrard dont le tome 1 est centré sur les objets avec les externals qui s'y rapportent tandis que le tome 2 le complète pour la programmation.

Je passe sous silence la 3ième édition d'un ouvrage pourtant célèbre mais qui pour le *System Rpl* est encore pratiquement nul.

Et pour terminer voici sucessivement les fichiers sources des deux programmes annoncés qui peuvent se compiler indifféremment par `RPLCOMP` de HP ou `→RPL` du *RPL Toolkit*.

PROGDEC
190 octets cksum # 7C34h

```
::                    ( début structure programme )
 CK1NOLASTWD        ( vérifie si 1 niveau de pile )
CODE
        A=DAT1 A              * adresse de l'objet
        GOSBVL =SAVPTR        * sauve les pointeurs
*                               pour PUSH#ALOOP
        GOVLNG =PUSH#ALOOP    * pousse l'adresse en
*               A(A) sur la pile et retour au RPL
ENDCODE
 SWAP                ( niveau 2 : adresse objet en )
           (  system binary     niveau 1 : objet )
 OCRC              ( BYTES interne sur un objet -> )
      ( longueur en quartets : 1 SB  et checksum )
 DROPOVER        ( drop sur le checksum puis over )
        ( pour dupliquer l'adresse  au niveau 1 )
#+-1       ( adresse objet + longueur objet - 1 )
                              ( = adresse fin en SB )
SWAP                        ( niv. 2 : adresse fin )
                         ( niveau 1 : adresse objet )
#>HXS                 ( conversion adresse objet : )
                  ( sytem binary -> integer binary )
ZERO                  ( initialisation avec SB 0 : )
        ( compteur de blocs de code  assembleur )
NULLLAM          ( nom nul pour variable locale : )
                                      ( adresse fin )
NULLLAM          ( nom nul pour variable locale : )
           ( adresse début -> adresse courante )
NULLLAM          ( nom nul pour variable locale : )
                    ( compteur de blocs assembleur )
THREE{}N             ( liste avec les 3 noms nuls )
          ( pour création des variables locales )
BIND         ( crée l'environnement des variables )
                                          ( locales )
NULL$              ( initialise la chaîne listing )
                          ( avec la chaîne nulle )
BEGIN    ( commence la boucle DO ... UNTIL.. END )
2GETLAM                ( 2ième variable locale -> )
                  ( adresse courante sur la pile )
ROMPTR 4D2 2    ( appelle DCADR de <-RPL->  pour )
           ( décompilation à l'adresse courante )
2PUTLAM           ( sauve l'adresse suivante dans )
                       ( la 2ième variable locale )
DUP                  ( duplique la chaîne courante )
                                ( de décompilation )
FIVE                          ( system binary <5h> )
1_#1-SUB$   ( fait un SUB de 1 à 4 sur la chaîne )
                     ( courante de décompilation )
$ "CODE"                ( génère la chaîne "CODE" )
HXS#HXS           ( compare "CODE" au sub 1 à 4 : )
     ( si chaînes différentes renvoie 1 sinon 0 )
%0=                   ( renvoie le booléen TRUE si )
                   ( le réel est nul sinon FALSE )
IT                                    ( IF THEN )
::                  ( si TRUE exécute ce programme )
 ROMPTR 4D2 1    ( appelle ->RPL de <-RPL-> pour )
                               ( compiler le code )
 $ "Code"              ( génère la chaîne "Code" )
 1GETLAM                ( 1ère variable locale : )
              ( compteur de blocs de code en SB )
 #1+DUP          ( actualise compteur et duplique )
                                   ( cette valeur )
 1PUTLAM         ( sauve le compteur dans la 1ère )
                               ( variable locale )
 UNCOERCE     ( conversion sytem binary <-> réel )
 DO>STR         ( conversion réel -> chaîne pour )
                        ( indice de bloc de code )
 SWAPOVER   ( niv. 3 : indice    niv. 2 : "Code" )
                               ( niv. 1 : indice )
 &$       ( concatène "Code" et la chaîne indice )
 $>ID                ( conversion chaîne -> nom )
 4ROLL            ( niveau 1 : la chaîne listing )
 $ "CODE_"             ( génère la chaîne "CODE" )
```

```
 &$    ( concatène la chaîne listing et "CODE_" )
 ROT        ( niv. 2 : chaîne listing + "CODE_" )
                              ( niv. 1 : indice )
;                          ( fin de ce programme )
 NEWLINE$&$         ( ajoute newline à la chaîne )
               ( courante ou à la chaîne indice  )
 &$                ( concatène la chaîne listing )
                              ( et cette chaîne )
2GETLAM                 ( 2ième variable locale : )
          ( adresse courante en entier binaire  )
HXS>#                   ( conversion binary integer )
                              ( -> system binary )
3GETLAM                 ( 3ième variable locale : )
                ( adresse de fin en sytem binary )
#>           ( comparaison entre 2 sytem binary : )
     ( renvoie le booleen TRUE si > sinon FALSE )
UNTIL                ( sortie de boucle si TRUE : )
               ( adresse courante > adresse fin )
1GETLAM                  ( 1ère variable locale : )
         ( compteur de blocs de code assembleur )
#0=?SKIP ( si zéro alors saute l'objet suivant: )
                              ( ici programme )
::                    ( début structure programme )
 1GETLAM              ( nombre n de blocs de code )
 #1+_ONE_DO (DO)          ( boucle de 1 à n + 1 : )
                              ( en fait 1 à n )
 UNROT ( niv. 3: chaîne listing  niv. 2: bloc i )
                               ( niv. 1 nom i )
 STO                ( stocke bloc i dans variable )
                             ( globale "Codei" )
 LOOP           ( fin de boucle si indice = n + 1 )
;                           ( fin de ce programme )
ABND                        ( fin d'environnement )
                          ( des variables locales )
;                    ( fin de structure programme )
```

ALLDEC

171.5 octets cksum # 7822h

```
::                ( début de structure programme )
 CK1NOLASTWD          ( vérifie s'il y a 1 niveau )
 CK&DISPATCH0          ( de pile puis si c'est un )
 THREE                  ( argument de type chaîne )
 ::   ( début de structure programme si arg. ok )
 DOSTD                                   ( STD )
 ONE         ( initialise position de recherche p )
 ZERO      ( initialise compteur de blocs code i )
 NULLLAM      ( 2ième variable locale de nom nul )
 NULLLAM       ( 1ère variable locale de nom nul )
 TWO()N       ( liste de ces 2 variables locales )
 BIND       ( crée l'environnement des var. loc. )
 BEGIN    ( début boucle while ... repeat .. end )
 $ "CODE_"              ( génère la chaîne "CODE_" )
 1GETLAM ( 1ère var. loc. : compteur de blocs i )
 #1+DUP     ( incrémente ce SB et le duplique    )
 1PUTLAM ( sauvegarde i+1 dans la 1ère var. loc.)
 #>$        ( conversion System Binary -> chaîne )
 &$              ( concatène pour faire "CODE_i" )
 2DUP      ( duplique chaine listing et "CODE_i" )
 2GETLAM ( 2e var. loc. -> position de recherche)
 POS$           ( recherche position de "CODE_i" )
 DUP            ( duplique position p ou 0 en SB )
 #0<>        ( différent de 0 ? -> TRUE ou FALSE )
 WHILE          ( si TRUE alors exécute la suite )
 ::                ( début programme à exécuter )
 DUP                       ( duplique position p )
 2PUTLAM  ( sauvegarde p dans la 2ieme var. loc.)
 SWAP               ( échange entre p et "CODE_i" )
 LEN$                ( longueur l de cette chaîne )
 OVER#+         ( niveau 2 : p    niveau 1 : p+l )
 3PICKSWAP   (niv.3:p  niv. 2:listing  niv.1:p+l)
 OVERLEN$       (niv. 3: listing    niv. 2: p+l )
                ( niv.1: longueur chaîne listing )
 SUB$           ( sub du listing de p+l à la fin )
 UNROT   ( niv.3:le sub  niv.2:listing  niv.3:p )
 1_#1-SUB$           ( sub du listing de 1 à p-1 )
  $ "CODE"             ( génère la chaîne "CODE" )
  &$                ( concatène ce sub et "CODE" )
  $ "Code"             ( génère la chaîne "Code" )
  1GETLAM  ( 1ère var. loc. -> compteur de blocs)
  #>$      ( conversion System Binary -> chaîne )
  &$             ( concatène pour faire "Codei" )
  $>ID                ( conversion chaîne en nom )
  XEQRCL                      ( rappelle 'Codei' )
  ROMPTR 4D2 3     (appelle COD→ : désass. Codei)
  &$      ( listing + le desassemblage de Codei )
  $ "ENDCODE"       ( génère la chaîne "ENDCODE" )
  &$SWAP        ( listing + "ENDCODE" et échange )
        ( avec la fin du listing après "CODE_i" )
  &$   ( concatène : la chaîne de désassemblage )
       ( de 'Codei' a remplacé "CODE_i" dans le )
                                  ( listing Rpl )
 ;           ( fin de structure de ce programme )
 REPEAT  ( on passe à la recherche de "CODE_i+1")
 2DROP  ( drop sur "CODE_i" et p avant le while )
 ABND   (abandonne l'environnement des var. loc.)
 ;         ( fin de structure du programme après )
           ( vérification du type de l'argument )
;         ( fin de structure du programme entier )
```

Les codes hexa se trouvent dans *le coin des codes*.

Guy Toublanc (276)



TRUCS, ASTUCES ET PETITS UTILITAIRES

Vous programmez en *System Rpl* ou *Assembleur* et vous avez une documentation qui vous donne des adresses diverses, par exemple celles que l'on trouve dans les très utiles *Secrets de la HP 48* de J.M. Ferrard. Vous voulez connaître les mnémoniques correspondants HP. Il vous faut :
- la librairie ←RPL→ de D.M (goodies disk 8).
- ASC→ : *JPC 79 page 11* ou équivalent.
- REV : *JPC 82 page 28* et actualisé dans ce *JPC*, ou équivalent.

Le petit programme MNEMO vous founira rapidement le mnémonique s'il existe, sinon l'adresse vous sera renvoyée dans une chaîne.

Exemples :

62035h : mnémonique dup;gn? des *Secrets.. 1* p 137
```
# 62035  MNEMO  →  "=DUPTYPEIDNT?"
```

F1C8h : mnémonique <>gn? des *Secrets.. 1* p 137
```
# F1C8   MNEMO  →  "#01FC8"
```
Pas de mnémonique HP donc point d'entrée non supporté

Le principe du programme est très simple : L'adresse est transformée en une chaîne de 5 caractères qui est inversée. On ajoute "D9D20" avant cette chaîne et "B2130" après, obtenant ainsi les codes hexa d'un programme. Ces codes sont assemblés par ASC→ puis ce programme est décompilé par RPL→. On fait un sub sur la chaîne générée pour obtenir le mnémonique. Ceci donne aussi bien les mnémoniques Rpl et Assembleur grâce à un défaut de RPL→ qui donne aussi les mnémoniques de l'assembleur ce qui n'est pas très logique mais fait ici notre bonheur.

MNEMO
123 octets cksum # A472h

```
« RCLF SWAP HEX 24
  STWS # 100000h +
  →STR 4 8 SUB REV
  "D9D20" SWAP + "B2130" +
  ASC→ RPL→ 4 OVER
  SIZE 2 - SUB SWAP
  STOF
»
```

Guy Toublanc (276)

CHOC EN RETOUR

Dans *JPC 82 page 28* j'avais publié une version de REV permettant d'inverser les caractères d'une chaîne mais ce programme a le défaut de modifier aussi la chaîne dans la variable dont elle est issue. Pour remédier à celà il suffit d'appeler la routine TOTEMPOB pour recréer la chaîne en mémoire.

Voici le nouveau fichier source fichier modifié et avec un gain de 2 octets grâce à l'utilisation de la routine GetStrLenStk. J'ai remplacé CK1&Dispatch qui s'utilise de préférence pour des programmes de librairies, par CK1NOLASTWD et CK&DISPATCH0 plus adaptés ici:

REV
58.5 octets cksum # 9069h

```
::
CK1NOLASTWD CK&DISPATCH0 str
:: TOTEMPOB
CODE
       gosbvl  =SAVPTR
       gosbvl  =GetStrLenStk
       ad1ex
       d1=a
       a=a+c   a
       a=a+c   a
       d0=a
       csrb.f  a
       b=c     a
       gonc    decr
loop   d0=d0-  2
       c=dat1  b
       a=dat0  b
       dat1=a  b
       dat0=c  b
       d1=d1+  2
decr   b=b-1   a
       gonc    loop
out    govlng  =GETPTRLOOP
ENDCODE
; ;
```

Vous trouverez la liste des codes dans *le coin des codes*

Guy Toublanc (276)


Le silence est d'or
Le temps c'est de l'argent
Je ferais mieux de me taire
et vous de ne plus me lire !

HP95

J. Belin	Les *Goodies Disks* HP95	*18*
P. Guerin	Correction de Q.C.M. avec Lotus	*19*
J. Belin	Recherche de fichiers sur disque	*22*
	Le coin des codes	*29*

QU'ELLE ETAIT VERTE MA DISQUETTE

Voici (enfin !) le première fournée de ce que nous pourrions appeler les "Goodies Disk HP95". Il s'agit d'une sélection personnelle de programmes totalement spécifiques au HP95 (applications System Manager, programmes graphiques ou sonores), ainsi que des utilitaires MS-DOS classiques, mais particulièrement intéressants en raison de leur très faible taille ou de leur bonne utilisation de l'écran du HP95.

Tous les programmes sont en Freeware (totalement gratuits) et en Shareware (une faible rétribution est demandée). Je vous encourage donc à lire les documentations et à respecter les souhaits des auteurs...

La grande majorité des programmes inclus dans ces disquettes sont d'origine américaine et sont donc fournis avec une documentation en anglais... Désolé pour les anglophobes, mais sachant que certains fichiers dépassent les 6000 lignes, j'ai préféré abdiquer tout de suite ! Notez aussi que certains fichiers (Lotus notament) nécessiteront soit que vous configuriez votre HP95 en Anglais, soit que vous les adaptiez. Dans tous les cas n'oubliez pas que si vous avez des problèmes d'installation ou d'utilisation, je suis à votre disposition pour vous aider.

Tous les fichiers ayant trait à un même programme ont été regroupés et compressés dans un fichier "ZIP". Ceci permet de mettre plus de programmes sur une même disquette, mais aussi de simplifier les échanges. D'autre part, la plupart des fichiers exécutables (COM et EXE) ont été compressés avec PKLITE, ce qui permettra de réduire leur taille dans les disques Ram du HP95.

Voici maintenant un court aperçu de ce qui est présent sur les disquettes :

Disquette n°1

Cette disquette est plutôt orientée sur les fichiers d'applications ou les extensions des applications intégrées.

Un directory APPT contient des programmes étendant les possibilités d'affichages du module Agenda, ainsi que des utilitaires de conversion de fichiers .ABK.

Pour ceux qui désirent effectuer des calculs informatiques (conversion de bases, opérations logiques...) j'ai inclus deux calculatrices System Manager, une en RPN l'autre en algébrique.

Hormis le célébrissime KERMIT, que j'ai inclus ici pour les quelques personnes qui ne l'auraient pas encore, vous trouverez d'autres programmes de communications dont un de très faible taille permettant de transférer des fichiers à 115 kbauds.

Les utilisateurs de Lotus trouveront aussi quelques feuilles de calcul concernant la gestion de compte bancaire, des calculs d'assurances, une application médicale...

Un directory PHONE est particulièrement fourni en utitaires de conversion des fichiers PBK vers d'autres programmes (SIDEKICK...) ou d'autres ordinateurs de poche (Psion..).

Pour les utilisateurs des applications intégrées, le programme le plus important de ces deux disquettes sera sûrement 95BUDDY. En effet, ce programme (probablement le plus connu sur le HP95) vous permettra de corriger tous les petits défauts qui rendent l'utilisation de la machine un peu irritante. Vous pourrez par exemple, en étant dans le FILER sélectionner un fichier, le visualiser en tapant ENTER, puis si vous le désirez, activer le MEMO et éditer le fichier en appuyant tout simplement une deuxième fois sur ENTER. Le programme offre ce genre de raccourcis clavier à toutes les applications internes, et bien d'autres choses encore...

Parmi les autres utilitaires, vous trouverez des programmes permettant de configurer les possibilités spécifiques du HP95 (Volume, Timeout, liaison série...), des programmes d'impression d'écran, des applications permettant de lancer simplement des applicationes Dos à partir du System Manager, des programmes graphiques, des utilitaires de visualisation de fichiers et une sélection des plus petits utilitaires DOS, pour ne pas encombrer la mémoire de votre HP95. Dont deux utilitaires de cryptage/décryptage de fichiers faisant moins de 700 octets à eux deux...

Dans toute compilation, il y a toujours des fichiers que l'on ne sait pas où classer. Celle ci n'échappe pas à la règle. J'ai donc choisi la solution de simplicité en créant un directory DIVERS. Celui ci contient donc quelques "bizareries", telles qu'un programme vous informant de la présence d'un "petit homme vert" dans votre HP95, ou des fichiers .PBK utilisant l'application de répertoire téléphonique comme gestionnaire de base de données pour donner les

coordonées géographiques des principaux aéroports mondiaux...

Disquette n°2

La deuxième disquette contient des éditeurs de textes et les applications, disons, moins sérieuses...

Des Echecs au Tetris de notre ami Yann Rousse (probablement le programme français le plus connu sur HP95) vous trouverez aussi de quoi vous amuser quelques bonnes heures. Pour ma part, je vous conseille SOKOBAN, un programme tactique très sympathique...

Pour tous ceux qui ne supportent pas l'éditeur intégré, vous en trouverez trois autres, très puissants et aux possibilités très différentes : VDE (plutôt orienté traitement de texte), OUTL95 (permettant de gérer des très gros mémos, avec une gestion de chapitres) et QED (très puissant et totalement paramétrable).

Si vous commandez ces deux disquettes, et même si vous possédez une carte mémoire, vous risquez de vous trouver très vite face au problème de l'encombrement de la mémoire de votre HP95. C'est pour cela que j'ai inclus ici les principaux programmes de compression de fichiers (PKZIP, LHARC, DIET...). La plupart d'entre eux permettent non seulement d'archiver les fichiers, mais aussi de compresser les exécutables de façon totalement transparente. Cependant, les taux de compression et les vitesses de traitement étant différentes, ce sera à vous de choisir lequel utiliser suivant vos impératifs.

De la sonnerie du téléphone à l'indicatif de la série télé "Twilight Zone", en passant par le désormais célèbre "I'll be back" du Terminator, vous aurez tout ce qu'il faut pour faire du bruit pendant vos réunions de travail ! Il est à noter que ces programmes ne fonctionneront pas sur le HP100LX, car il ne dispose pas du même "hardware" que le HP95.

Si vous désirez changer l'écran d'acceuil de votre machine, j'ai déniché quelques "Topcards", dont notre cher Albert Einstein...

Ces deux disquettes s'adressent d'abord aux utilisateurs de HP95. Cependant, il est fort probable que la prochaine soit plus orientée vers la programmation, avec peut-être des compilateurs de faible taille, mais surtout de nombreux sources et des informations concernant le développement des applications System Manager. Ce serait donc un peu l'equivalent de notre *Kit de développement HP48...*

Jacques Belin (123)

CHOIX MULTIPLES : UNE SEULE REPONSE ?

Depuis que dans certains examens et dans de nombreux cas, la pratique des interrogations par questions à choix multiples s'est répandue, on a vu se développer et se multiplier la variété des Q.C.M.

Certains sont de simples séries de questions dites "fermés", auxquelles on ne peut répondre que par oui ou non, ou ne pas répondre. Cette variété convient parfaitement aux enquêtes d'opinion publique. Le programme de correction automatique présenté ici peut être adapté à ce genre d'interrogations. Il suffit de convenir de donner le code 1 à une réponse affirmative, le code 2 à une réponse négative et le code 0 à l'absence de réponse pour pouvoir trier ensuite les réponses sur chacun de ces critères et "traiter" l'information ainsi collectée. Un tel tri pourrait d'ailleurs être automatisé.

Mais ce programme convient surtout à un véritable Q.C.M., car il peut analyser les réponses en les triant sur la base d'un choix de réponses à chaque question compris entre 1 et 9.

Traitons aujourd'hui, ce genre de questionnaires.

Le programme comprend 3 Macros. La première dénommée "\I", permet l'initialisation du système, en enregistrant le nombre de questionnaires à traiter, puis la réponse correcte à chaque question. Ces réponses sont enregistrées dans la colonne D. Le nombre de questions prévues est de 60, mais il est possible de l'utiliser pour des questionnaires moins longs, en enregistrant 0 pour chacune des questions supplémentaires au delà de la dernière question du questionnaire à traiter.

La seconde Macro dénommée "\D" permet la saisie des données et le calcul de la note obtenue pour l'ensemble des réponses. Pour chaque question, elle enregistre la réponse, la compare à la réponse correcte et attribue la note 2 s'il n'a pas été répondu à la question, la note 1 si la réponse est différente de celle de la réponse correcte et la note 0 dans le cas contraire.

Ces notes sont en réalité des notes négatives soustraites d'un total de 60 points (1 par question) dont chaque candidat est crédité. Au total, la note obtenue correspond bien au rapport entre le nombre de réponses justes et le nombre de questions. Il est ramené à 20 et arrondi à l'entier le plus proche. Toutefois, dans le cas où le nombre de questions du

questionnaire considéré est inférieur à 60, il convient de multiplier la note obtenue et affichée dans la cellule C71, par le rapport inverse entre le nombre de questions et 60.

On voit aisément l'intérêt du parti adopté dans l'écriture de ce petit programme qui est d'afficher toutes les données importantes dans le cadre des seules quatre premières colonnes standard que l'écran du "Jaguar" peut seulement contenir à la fois. Cela permet d'éviter les défilements latéraux toujours désagréables, sauf si l'on bloque à la fois les titres des lignes et des colonnes, ce qui réduit encore l'écran utile en le ramenant au mieux à 3 colonnes et quelques lignes. En prenant la précaution de bloquer les titres à la ligne 6, il est possible de visualiser à la fois pour chaque question la réponse exacte (colonne D), la réponse saisie (colonne B), la note attribuée à chaque réponse qui apparaît à la colonne C. Dans la partie du haut de l'écran réservée aux titres, apparaît colonne B, le nombre total de copies à corriger et colonne D, le numéro de la copie en cours de correction.

La troisième macro, dénommée "\R", permet d'établir la liste des notes obtenues par les différentes copies. Il suffit de l'appeler dès que la note de la copie traitée apparaît dans la cellule C71. Le nombre de copies à traiter à été volontairement limité à 20, afin d'économiser la mémoire du petit "Jaguar", mais il ne serait pas difficile de l'augmenter aisément, surtout si ce logiciel venait à être utilisé sur le tableur d'un véritable P.C..

A cet effet, un parti inhabituel a été pris. Celui d'écrire les Macros elles-même dans trois colonnes voisines, rendant possible la multiplication ou la réduction du nombre de lignes d'écriture en insérant de nouvelles lignes ou détruisant une partie de celles qui existent. Mais il s'agit là d'une opération délicate, susceptible d'être au moins partiellement automatisée, ce qui pourrait faire l'objet de l'écriture de nouvelles macros. Je me propose de vous en entretenir dans un prochain article.

Pour conclure celui-ci, je me bornerai à rappeler que les trois macros étant nommées, il suffit de taper à la fois `Alt` et la lettre nommant la macro pour démarrer le programme. Donc tapez `Alt+I` pour initialiser les réponses correctes. Lorsque toutes les réponses correctes ont été enregistrées, tapez `Alt+D` pour saisir les données du premier questionnaire à traiter. Enfin, lorsque toutes les données du questionnaire ont été saisies et que la note apparaît dans la cellule C71, tapez `Alt+R` si vous désirez établir une liste des résultats. Après l'enregistrement de chaque résultat dans une cellule de la colonne C, à la suite du précédent, tapez à nouveau `Alt+D` pour continuer la saisie des réponses.

Description de la feuille de calcul

NDLR : Si vous utilisez une version US du HP95, ou que vous avez configuré votre HP95 en mode 'English', il est nécessaire de traduire les commandes Lotus en Anglais. Pour cela utilisez les correspondances suivantes : `SI→IF`, `SOIT→LET`, `VA→GOTO`, `CONTENU→CONTENTS`, `SOMME→SUM`, `QUITTE→QUIT`, `B→D` et `D→R`.

Entrez les valeurs spécifiées dans les cellules suivantes :

Cellule A1 : `'Q.C.M.`
Cellule B1 : `'Réponse:`
Cellule D1 : `'Rép.Juste`
Cellule A3 : `'N. Copies`
Cellule B3 : `20`
Cellule C3 : `'Compteur`
Cellule D3 : `0`
Cellule A6 : `'Questions:`
Cellule C6 : `'N. Quest:`
Cellule D6 : `60`

Cellule A9 : `1`
Cellule A10 : `+A9+1`
Cellule A11 : `+A10+1`
. .
. .
Cellule A68 : `+A67+1`
(Méthode : Ecrire la cellule A10, puis la recopier avec '`/R`' sur les cellules A11 à A68)

Cellule B9 : `1`

Cellule C9 : `@SI(B9=D9;0;@SI(B9=0;2;1))`
Cellule C10 : `@SI(B10=D10;0;@SI(B10=0;2;1))`
Cellule C11 : `@SI(B11=D11;0;@SI(B11=0;2;1))`
. .
. .
Cellule C68 : `@SI(B68=D68;0;@SI(B68=0;2;1))`
(Même méthode que précédement)

Cellule D70 : `'o.k.`
Cellule A71 : `'Note:`
Cellule C71 : `(60-@SOMME(C9..C68))/3`
Cellule C73 : `'Résultats`
Cellule A74 : `'Devoir N°`

Puis spécifiez les cellules C74 à C93 en notation 'FIX 2', en se plaçant sur la cellule C74, puis en exécutant la séquence de touches '`CFF2 C74..C93`' ('`RFF2 C74..C93`' en version US). Faites la même opération pour la cellule C71.

Enfin, protégez les titres en vous placant en ligne 7 et en exécutant la séquence de touches '/fth' ('/wth' en version US).

Le texte des Macros

Remarques générales :

1- Il est préférable d'écrire chaque Macro dans une seule colonne.

2- Il est préférable également d'écrire une Macro directement, plutôt que d'utiliser le mode d'écriture "automatique" (alt F5), qui sépare chaque instruction en changeant de cellule, alors qu'il est possible de regrouper plusieurs instructions par cellule pour créer des séquences rendant plus intelligible l'ensemble du programme automatisé et facilitant la reproduction des séquences identiques par l'utilisation de l'instruction "recopie" (Menu ou / puis R (C en US), détermination de la cellule ou du champ (ensemble de plusieurs cellules) à recopier, détermination de la cellule ou du champ où l'instruction est recopiée. Pour cela il ne faut pas hésiter à créer des colonnes de plus de 9 caractères de largeur en utilisant l'instruction /FCD (/WCS en US) qui permet d'en modifier la largeur jusqu'à 254 caractères.

3- Chaque instruction simple est précédée du symbole "{" et suivie du symbole "}". Il en va de même des instructions complexes, auxquelles il convient d'ajouter à l'intérieur des {} les arguments séparés par le symbole séparateur ";". On le voit, il n'y a pas une très grande différence de technique entre la programmation "123" et la programmation en langage "Basic HP71" bien connu des membres du club.

4- Pour éviter toute interférence entre la zone des données de la feuille de calcul et celle du programme, il est préférable de placer cette dernière au dessous et à droite de la première.

5- Enfin dans l'écriture d'un programme composé de Macroinstructions il ne faut laisser aucune cellule vide entre deux instructions.

Macro \I *:*

Cellule G103 : {SOIT D3};+0}~
Cellule G104 : {VA}A9~{D}{D}{D}
Cellules G105 à G164 : {?}~{B}
Cellule G165 : {VA}D70~
Cellule G166 : {SOIT D70;"o.k."}~
Cellule G167 : {QUITE}~

Elle a été écrite dans la colonne G des cellules 103 à 165, portées à une largeur de 19 caractères. La cellule G103 contient les instructions qui ramènent le compteur à zéro. La cellule G104 contient les instructions positionnent le pointeur sur la première cellule qui doit recevoir des données. Sa fonction est quasiment équivalente à {VA}D9~, mais en repositionnant la colonne D à droite de l'affichage. La cellule G165 contient les instructions destinées à indiquer à l'utilisateur que la saisie des réponses type est achevée. La cellule suivante, la dernière du programme, contient l'instruction {QUITTE} mettant fin à l'exécution de la "Macro". Les cellules comprises entre la cellule G104 et G164 contiennent toutes les instructions {?}~{B}, permettant au programme de stocker interactivement les données saisies dans la cellule où le pointeur est positionné et ceci fait de se positionner sur la cellule de la ligne suivante.

Les autres "Macros" ont été écrites en s'inspirant de la même philosophie.

Cela donne:

Macro \D *:*

Cellule F103 : {SOIT D3;+D3+1}{SOIT D70;""}
Cellule F104 : {VA}A9~{D}
Cellules F105 à F164 : {?}~{B}
Cellule F165 : {B}{B}{D}

Cette macro est enregistrée dans la colonne F d'une largeur de 29 caractères.

La cellule F103 contient les instructions nécessaires pour positionner le pointeur, aprés avoir activé le compteur de copies, initialisé à zéro par la macro \I et effacé le message de fin d'initialisation inscrit dans la cellule D70.

La cellule F165 suspend l'exécution de la Macro, aprés avoir positionné le pointeur sur la cellule contenant la note obtenue par la copie qui vient d'être saisie.

Les autres cellules comprises entre F104 et F164 contiennent toutes les instructions {?}~{B} déjà rencontrées dans la Macro /I.

Macro \R *:*

NDLR : Le caractère '\' indique que ce qui est marqué sur la ligne suivante appartient à la même cellule.

Cellule H103 : {SOIT A75;A74}{SOIT A76;A74} \
{SOIT A77;A74}{SOIT A78;A74}
Cellule H104 : {SOIT A79;A74}{SOIT A80;A74} \
{SOIT A81;A74}{SOIT A82;A74}

```
Cellule H105 : {SOIT A83;A74}{SOIT A84;A74}   \
               {SOIT A85;A74}{SOIT A86;A74}
Cellule H106 : {SOIT A87;A74}{SOIT A88;A74}   \
               {SOIT A89;A74}{SOIT A90;A74}
Cellule H107 : {SOIT A91;A74}{SOIT A92;A74}   \
               {SOIT A93;A74}
Cellule H108 : {SI D3=0}{QUITE}
Cellule H109 : {SI D3=1}{CONTENU B74;D3}          \
               {CONTENU C74;C71}{VA}A74~{D}{D}    \
               {QUITTE}
Cellule H110 : {SI D3=2}{CONTENU B75;D3}          \
               {CONTENU C75;C71}{VA}A75~{D}{D}    \
               {QUITTE}
        .                .
        .                .
Cellule H128 : {SI D3=20}{CONTENU B93;D3}         \
               {CONTENU C93;C71}{VA}A93~{D}{D}    \
               {QUITTE}
Cellule H129 : {SI D3>20}{QUITTE}
```

Pour cette Macro, la structure est un peu différente et utilise une formule récurente qui peut être écrite sur une ligne, recopiée sur les suivantes en corrigeant les adresses des cellules concernées. Le résultat est le suivant: Colonne H portée à la largeur de 70 caractères. Les 4 premières lignes répètent au total 19 fois l'instruction de recopie de la cellule A74 contenant le libellé "Devoir N°:", dans les cellules des lignes suivantes soit de A75 à A93.

La ligne suivante (H108) invite la Macro à stoper la déroulement du programme si le compteur de copies est sur 0, et chacune des lignes suivantes (H109 à H128) permet au pointeur de se positionner sur la cellule de la colonne B correspondant au numéro de la copie et d'inscrire dans la cellule de la colonne C voisine la note obtenue par cette copie.

Chacune des lignes suivantes contient les mêmes instructions au numéro de ligne de la cellule qui est incrémenté d'une unité chaque fois pour aboutir à la ligne H128.

La dernière ligne du programme est destinée à empécher tout dépassement du nombre de copies.

Cela peut paraître compliqué. Il n'en est rien en réalité. L'écriture des "Macros" peut sembler difficile. Cela tient au vocabulaire des tableurs qui utilise fréquemment des mots longs. Mais le sens de ces mots est presque toujours évident et la compréhension de la syntaxe en est simplifiée.

Bonnes corrections.

Paul Guérin (353)

RECHERCHE FICHIER DESESPEREMENT...

Dans le précédent numéro, je vous ai présenté un petit utilitaire permettant de determiner avec précision l'encombrement d'un ou plusieurs directories. Ce mois-ci, je vous présente son petit frère qui vous permettra de retrouver un ou plusieurs fichiers égarés dans l'arborescence de votre disque dur. Comme pour DU, il s'agissait précédement d'un programme écrit en langage C, que j'ai converti en assembleur afin de réduire sa taille de façon très significative.

Notez avant tout qu'il ne brille pas par son originalité, car des utilitaires équivalents ont déjà été écrits à de nombreuses reprises. Ma version a cependant tenté de trouver un bon compromis entre les versions "puissantes" permettant entre autre de trouver des fichiers par rapport à la date, mais très gros (plus de 5000 octets) et ceux de très petite taille n'effectuant les recherches que par rapport au nom des fichiers. Vous en trouverez un exemple dans la disquette HP95 n°1, faisant moins de 512 octets WHERE.COM.

Appelé WHEREIS, sa syntaxe est la suivante :

```
whereis [-h] [-s] [-r] [-a] [-d répertoire_initial]
                            [fichier] [fichiers...]
```

Les options '-h', '-s', '-r' et '-a' indiquent que l'on veut limiter la recherche aux fichiers possédant les bits d'attribut correspondants armés :

h : fichiers cachés
s : fichiers système
r : fichiers protégés en écriture
a : fichiers modifiés

L'option '-d' permet d'indiquer que l'on veut restreindre la recherche à un répertoire donné ainsi que ses sous-répertoires. Si vous désirez n'effectuer la recherche qu'à partir du directory courant, il faudra entrer l'option '-d.'. Par défaut, la recherche est effectuée dans la totalité du disque courant.

Après ces options, il est possible de spécifier un ou plusieurs paramètres correspondant à des noms de fichiers ou de directories. Les *WildCards* sont bien entendu acceptés.

La recherche s'effectue pour tous les fichiers et pour tous les directories, même si il sont cachés.

L'affichage des fichiers se fait sous une forme sensiblement équivalente à celle de l'instruction DIR, avec en plus l'indication de leurs attributs. Malheureusement pour les utilisateurs de HP95, je n'ai pu réduire la ligne de texte qu'à 41 octets, ce qui fera perdre l'affichage du bit "archive" sur l'écran du Jaguar.

Exemples d'utilisation

Si nous voulons chercher tous les fichiers commencant par le caractère "_", sur le HP95, il nous suffit de taper :

```
whereis _*.*
```

Ce qui nous donnera ceci :

```
Directory :
 _CFLOW.WK1      4456 01/10/91 08:00 ..RA
 _STAT.WK1       5463 01/10/91 08:00 ..RA
 _SYS           <DIR>  01/10/91 05:18 H.R.
 _DAT           <DIR>  01/04/91 00:00 ....

Directory : \_SYS
 _1.PBK           512 01/10/91 08:00 ..RA

Directory : \_DAT
 _1.PBK           512 01/10/91 08:00 ....

       6 files found
```

Si (comme moi) vous avez l'habitude d'utiliser la commande xcopy /m pour effectuer vos sauvegardes, il est possible d'afficher seulement les fichiers crés ou modifiés depuis la nouvelle sauvegarde. Il suffit d'entrer la commande suivante :

```
whereis -a
```

Si vous ne désirez faire la même chose que pour certains fichiers, et sur un autre disque, vous pourrez, par exemple, taper :

```
whereis -a -dd: *.txt *.doc
```

Il existe cependant un petit défaut dans le programme. En effet, l'ordre suivant :

```
whereis -d\tmp whereis.* *.com
```

affichera :

```
Directory : \tmp
 WHEREIS.COM      1830 26/06/93 10:28 ...A
 WHEREIS.ASM     23439 26/06/93 10:28 ...A
 DU.COM           1532 04/06/93 13:02 ...A
 WHEREIS.COM      1830 26/06/93 10:28 ...A

       4 files found
```

Vous pouvez noter que WHEREIS.COM est affiché deux fois, ce qui peut être relativement gênant lorsque il y a beaucoup de fichiers. Il aurait été possible de tenir compte de ce problème (en stockant les noms de fichiers en mémoire puis en comparant les doublons), mais cela aurait trop augmenté la taille du programme. Il convient donc de connaitre l'existence de ce défaut, et en tenir compte lorsque on spécifie les paramètres.

Fonctionnement du programme

Comme je l'ai mentionné au début de cet article, ce programme est une variation directe du programme DU.COM et possède la même structure récursive. Je ne me bornerai donc qu'à expliquer les différences entre les deux programmes.

La différence principale réside dans le fait que l'affichage se fait à l'intérieur de la boucle de recherche des fichiers, alors que pour DU, il se faisait juste après. La seconde grande différence est que "*.*" était le seul paramètre de recherche possible dans DU, alors que pour whereis nous utilisons les paramètres de la ligne de commande. De plus, le nombre d'arguments étant variable, une boucle (pointée par optind) englobe la boucle de recherche des fichiers.

L'affichage des informations concernant le fichier est géré par le sous-programme d_entry. Il prend ses informations dans une zone mémoire de format équivalent à ce qui est retourné par les fonctions 4E et 4F du DOS. L'affichage de la date peut se faire aussi bien sous le format US (MM/JJ/AA) qu'Européen (JJ/MM/AA), à condition que la variable country ait été initialisé à l'aide des informations issues de la fonction 38h du DOS. Ce qui est fait au début du programme principal.

Vous trouverez les codes Hexa décimaux du programme dans le *Coin des Codes*. Notez qu'afin de diminuer votre travail, je l'ai compilé avec l'option '-m2' de TASM, puis compressé avec DIET, ce qui n'a normalement aucun intérêt car l'encombrement disque ne change pas. La taille du programme original est de 1830 octets.

Jacques Belin (123)

```
;*****************************************************************************
;*****************************************************************************
;**                          WHEREIS.ASM                                    **
;**                 Recherche de fichiers sur le disque                     **
;**                                                                         **
;**   Dernière modification : 30/06/93        Auteur : Jacques Belin        **
;**       Création                                                          **
;**                                                                         **
;*****************************************************************************
;*****************************************************************************
        DOSSEG
code    segment para 'CODE'
        assume cs:code,ds:code
        ORG     100h
start:  jmp     main                    ; première inst. exécutée

PATH_SIZ EQU    80                      ; taille maxi d'un chemin
DTA_SIZE EQU    48                      ; taille de la DTA

FA_RDNLY EQU    1                       ; codes d'attributs des fichiers
FA_HIDD  EQU    2
FA_SYST  EQU    4
FA_VOL   EQU    8
FA_DIR   EQU    16
FA_ARCH  EQU    32

        INCLUDE getopt.inc              ; routine d'analyse des options
                                        ; (sources parus dans JPC 87)

optstr  DB      "d:hsra",0              ; chaine des options
errflg  DB      0                       ; flag d'erreur
opti    DW      0                       ; sauvegarde de optind
nb_files DW     0                       ; nombre de fichiers trouvés
path_ini DB     PATH_SIZ dup (0)        ; directory initial
attr    DB      0                       ; attributs spécifiés en option
nodir   DB      0                       ; flag indiquant un paramètre "*.*"
found   DB      0                       ; flag indiquant si dir courant affiché

                                        ; Message d'erreur
usage   DB      "usage :  whereis [-h] [-s] [-r] [-a] [-d path] [file...]"
        DB      13,10,10,"$"

dirname DB      13,10,"Directory : "    ; affichage du directory courant
cur_path DB     PATH_SIZ dup (?)
cr      DB      13,10,0

total   DB      13,10,"  "              ; affichage du nombre de fichiers
total_2 DB      "xxxxxx files found",13,10,"$"

no_path DB      ".",0                   ; chaines diverses
back_sl DB      "\",0
wilds   DB      "\"
wilds2  DB      "*.*",0


main:   mov     bx,cs
        mov     ds,bx                   ; on assure DS=CS
        mov     es,bx                   ; on assure ES=CS

        mov     ah,4Ah                  ; on réjuste taille du bloc programmee
        mov     bx,4096/16
        int     21h
        mov     sp,4096                 ; on replace le pointeur de pile

        mov     ax,3800h                ; récupération du code de pays
        mov     dx,OFFSET path_ini      ; (pour d_entry)
        int     21h
        mov     bx,dx
        mov     ax,[bx]
        mov     [country],al
        mov     byte ptr [bx],0         ; réinitialisation path_ini à 0

        call    parse                   ; parsing dans argv[]

optloop: call   getopt                  ; analyse des options
        cmp     al,EOF                  ; fin d'analyse ?
        je      main_1                  ; oui, on passe à la suite du prgm
opth:   cmp     al,"h"                  ; si option "h"
        jne     opts
        or      byte ptr [attr],FA_HIDD ;on arme le bit correspondant dans attr
        jmp     nextopt                 ; ..et on continue l'analyse
opts:   cmp     al,"s"                  ; idem pour l'option "s"...
        jne     optr
        or      byte ptr [attr],FA_SYST
        jmp     nextopt
optr:   cmp     al,"r"                  ; idem pour l'option "r"...
        jne     opta
        or      byte ptr [attr],FA_RDNLY
        jmp     nextopt
opta:   cmp     al,"a"                  ; idem pour l'option "a"...
        jne     optd
        or      byte ptr [attr],FA_ARCH
        jmp     nextopt
optd:   cmp     al,"d"                  ; Si option "d"
        jne     opterr
        mov     di,[optarg]             ; copie de ce qui est pointé par
        mov     si,OFFSET path_ini      ;  optarg dans path_ini
        call    strcpy
        jmp     nextopt
opterr: cmp     al,"?"                  ; si erreur, on arme le flag
        jne     nextopt
        inc     byte ptr [errflg]
nextopt: jmp    optloop                 ; on boucle tant qu'on a pas EOF
        ;-------------

main_1: cmp     byte ptr [errflg],0     ; si une erreur a été détectée...
        je      main_2
        mov     dx,OFFSET usage         ; ...on affiche le message d'erreur...
        mov     ah,09h
        int     21h
        jmp     exit                    ; ...et on sort.
        ;----------

main_2: mov     ax,[optind]             ; argc == optind ? (pas de path ?)
        cmp     ax,[argc]
        je      main_3                  ; non
        mov     [opti],ax

main_3: mov     dx,OFFSET path_ini
        call    strlen
        cmp     cx,0
        je      main_4
        mov     bx,cx
        cmp     byte ptr [si+bx-1],"\" ; dernier caractère de path_ini="\" ?
        jne     main_4                  ; non, on continue
        mov     byte ptr [si+bx-1],0    ; oui, on l'efface

main_4: push    ds                      ; on fait pointer ES:DI sur path_ini
        pop     es
        mov     di,dx
        call    whereis                 ; APPEL A LA ROUTINE PRINCIPALE
```

```
        mov     ax,[nb_files]           ; AFFICHAGE DU NOMBRE DE FICHIERS
        xor     dx,dx
        mov     si,OFFSET total_2       ; conversion du nombre en chaine
        mov     bx,6
        call    ltoa
        mov     dx,OFFSET total         ; affichage de la chaine
        mov     ah,09h
        int     21h

exit:   mov     ah,4ch                  ; retour au DOS
        int     21h


;*************************************************
; WHEREIS : cherche un (ou plusieurs) fichiers dans
;           l'arborescence
; entrées : ES:DI adresse du nom du directory courant
;           (terminé par caractère 0)
;*************************************************
whereis PROC    NEAR
        push    ax
        push    bx
        push    cx
        push    dx
        push    bp
        push    es
        push    di
        push    ds
        push    si

        call    getdta                  ; sauvegarde de la DTA dans la pile
        push    ax

        mov     ah,48h
        mov     bx,((DTA_SIZE+PATH_SIZ)/16)+1 ; taille en paragraphes
        int     21h                     ; allocation d'un bloc de mémoire
        push    ax

        push    di                      ; sauvegarde paramètre d'entrée
        push    es
        mov     bp,sp                   ; on va se servir de BP comme indice
                                        ; pour les accès dans la pile

        ;à partir d'ici on a dans la pile :
        ; bp    =  ES niveau précédent
        ; bp+2  =  DI niveau précédent
        ; BP+4  =  segment du buffer courant
        ; BP+6  =  DTA précédente
        ;   .
        ;   .

        xor     bx,bx                   ; fait pointer nouvelle DTA sur buffer
        call    setdta

        mov     ax,cs:[opti]            ; on repositionne optind sur 1er param
        mov     cs:[optind],ax
        mov     byte ptr cs:[found],0   ; on désarme le flag 'found'

where_1: mov    ax,ss:[bp]              ; copie path courant après DTA ds buffer
        mov     es,ax
        mov     di,ss:[bp+2]
        mov     ax,ss:[bp+4]
        mov     ds,ax
        mov     si,DTA_SIZE
        call    strcpy

        push    cs                      ; on ajoute "\" à la fin
        pop     es
        mov     di,OFFSET back_sl
        call    strcat

        cmp     word ptr cs:[opti],0    ; y a t-il des paramètres ?
        jne     where_2                 ; oui, on va plus bas

        mov     di,OFFSET wilds2        ; sinon, on ajoute "*.*" à la fin
        call    strcat
        jmp     where_3
        ;------------
                                        ; on copie le param. pointé par optind
where_2: mov    ax,cs:[psp]
        mov     es,ax
        mov     bx,cs:[optind]          ; bx = optind
        shl     bx,1
        mov     di,OFFSET argv          ; si = argv
        mov     di,cs:[di+bx]           ; di ^ argv[optind]
        call    strcat

                                        ; RECHERCHE DES FICHIERS DU REPERTOIRE
where_3: mov    byte ptr cs:[nodir],0   ; on arme 'nodir' si param = "*.*"
        cmp     word ptr es:[di],".*"
        jne     where_4
        cmp     byte ptr es:[di+2],"*"
        jne     where_4
        mov     byte ptr cs:[nodir],1

where_4: mov    ah,4Eh                  ; cherche premier fichier du répertoire
        mov     dx,si
        mov     cx,FA_DIR+FA_HIDD+FA_SYST
        int     21h
where_5: jc     where_a                 ; si carry -> recherche terminée

        mov     bl,ds:[15h]             ; bl = attribut (sans bit Volume)
        and     bl,111111b
        mov     al,cs:[attr]            ; attr = 0 (pas de recherche sur option) ?

        cmp     al,0
        jne     where_6                 ; non, affichage suivant attr
        test    bl,FA_DIR               ; Si bit directory non armé
        jz      where_7                 ; ..on affiche
        cmp     byte ptr cs:[nodir],1   ; on n'affiche pas si param "*.*"..
        je      where_9
        cmp     byte ptr ds:[1Eh],"."   ; ..ou si nom directory = "." ou ".."
        je      where_9
        jmp     where_7

where_6: test   al,bl                   ; si aucun bit de bl correspond à al
        jz      where_9                 ;  ...on n'affiche pas
where_7: inc    word ptr cs:[nb_files]  ; inc. nombre de fichiers traités
        cmp     byte ptr cs:[found],0
        jne     where_8
        mov     byte ptr cs:[found],1
        push    ds

        mov     ax,ss:[bp]              ; ES:DI = paramètre d'entrée
        mov     es,ax
        mov     di,ss:[bp+2]
        push    cs
        pop     ds
        mov     si,OFFSET cur_path
        call    strcpy                  ; copie nom directory en fin de chaine
        push    cs                      ; on ajoute un CR à la fin
        pop     es
```

```
          mov   di,OFFSET cr
          call  strcat

          mov   bx,1                    ; affichage du texte dans 'stdout'
          mov   dx,OFFSET dirname
          call  fprintf
          pop   ds

where_8:  xor   si,si
          call  d_entry                 ; affichage de l'entrée

where_9:  mov   ah,4Fh                  ; cherche fichier suivant
          int   21h
          jmp   where_5                 ; et on boucle
          ;-------------

where_a:  cmp   word ptr cs:[opti],0    ; y a t-il des paramètres ?
          je    where_b                 ; non, on va plus bas
          inc   word ptr cs:[optind]    ; sinon, on incrémente optind...
          mov   ax,cs:[optind]          ; et on boucle tant qu'il y a des params
          cmp   cs:[argc],ax
          je    where_b
          jmp   where_1

                                        ; RECHERCHE DANS LES SOUS REPERTOIRES
where_b:  pop   es                      ; copie path courant après DTA ds buffer
          push  es
          mov   di,ss:[bp+2]
          mov   ax,ss:[bp+4]
          mov   ds,ax
          mov   si,DTA_SIZE
          call  strcpy

          push  cs                      ; on ajoute "\*.*" à la fin
          pop   es
          mov   di,OFFSET wilds
          call  strcat

          mov   dx,si
          mov   dx,DTA_SIZE             ; cherche premier fichier
          mov   ah,4Eh
          mov   cx,FA_DIR+FA_HIDD
          int   21h

where_c:  jc    where_e                 ; si carry -> terminé
          mov   al,ds:[15h]             ; si attribut fichier # directory...
          and   al,FA_DIR
          jz    where_d
          cmp   byte ptr ds:[1Eh],"."   ; ... et nom directory = "." ou ".."
          je    where_d                 ; ...on passe à l'entrée suivante

          mov   ax,ss:[bp]              ; ES:DI = paramètre d'entrée (récursif)
          mov   es,ax
          mov   di,ss:[bp+2]
          mov   si,DTA_SIZE             ; copie paramètre dans buffer
          call  strcpy

          push  cs                      ; on ajoute "\" au paramètre
          pop   es
          mov   di,OFFSET back_sl
          call  strcat

          push  ds                      ; on ajoute le nom de l'entrée
          pop   es
          mov   di,1Eh
          call  strcat

          mov   di,si                   ; ES:DI = nouveau paramètre
          call  whereis                 ; appel récursif

where_d:  mov   ah,4Fh                  ; cherche fichier suivant
          int   21h
          jmp   where_c                 ; et on boucle
          ;-------------

where_e:  pop   ax                      ; on depile les données inutiles
          pop   ax

          pop   es                      ; libération du buffer
          mov   ah,49h
          int   21h

          pop   ax                      ; reinitialisation de l'ancienne DTA
          xor   bx,bx
          call  setdta

          pop   si
          pop   ds
          pop   di
          pop   es
          pop   bp
          pop   dx
          pop   cx
          pop   bx
          pop   ax
          ret
whereis   ENDP


;************************************************
; D_ENTRY : affiche une entrée de directory
; entrées : DS:SI adresse de la DTA issue des fonctions
;           DOS 4E et 4F
; Note : il est conseillé de mettre a jour (par le
;        programme princpal, avec la fonction Dos 38h)
;        la variable "country"
;************************************************
country   DB    1                       ; format de date par défaut : Europe
d_filnm   DB    " "
filename  DB    "xxxxxxxx.xxx"
len_file  DB    " xxxxxxx "
date      DB    "jj/mm/aa hh:mm "
attribs   DB    "HSRA",13,10,0
dir_ent   DB    "  <DIR>  "

d_entry   PROC  NEAR
          pushf
          push  ax
          push  bx
          push  cx
          push  dx
          push  es
          push  di
          push  ds
          push  si
          push  bp

          push  cs                      ; ES <- CS
          pop   es
          push  si
          mov   bp,sp
```

```
                                        ; COPIE DU NOM DU FICHIER
        mov     di,OFFSET filename
        add     si,1eh                  ; si pointe sur le premier caractère
                                        ;   du nom
        mov     cx,13
        cld
d_ent_1: movsb
        dec     cx
        cmp     byte ptr ds:[si],0      ; fin de chaine ?
        jne     d_ent_1
d_ent_2: mov    byte ptr es:[di]," "    ; on complete avec des blancs
        inc     di
        loop    d_ent_2


        mov     si,ss:[bp]              ; AFFICHAGE LONGUEUR
        mov     di,OFFSET len_file
        mov     al,[si+15h]             ; lecture attribut
        test    al,FA_DIR               ; Est-ce une entrée directory ?
        jz      d_ent_3                 ; non, on va plus bas
        mov     cx,8                    ; sinon, copie de <DIR> à la place
        mov     si,OFFSET dir_ent       ;  de la longueur
        push    ds
        push    cs
        pop     ds
        rep     movsb
        pop     ds
        jmp     d_ent_4
d_ent_3: mov    si,ss:[bp]              ; conversion longueur en chaine
        mov     ax,ds:[si+1Ah]
        mov     dx,ds:[si+1Ch]
        mov     bx,8
        push    ds
        push    es
        pop     ds
        mov     si,di
        call    ltoa
        pop     ds
                                        ; AFFICHAGE DATE
d_ent_4: pop    si
        mov     di,OFFSET date
        mov     dx,[si+18h]             ; DTA.date -> dx
        mov     ax,dx                   ; numéro du jour dans ax
        and     ax,11111b
        mov     cl,5                    ; numéro du mois dans bx
        shr     dx,cl
        mov     bx,dx
        and     bx,1111b
        cmp     byte ptr cs:[country],1 ; format de date Europe ?
        je      d_ent_5                 ; oui, on va plus bas
        xchg    ax,bx                   ; sinon, on inverse mois et jour
d_ent_5: call   d_ent_a                 ; affichage "XX/../.."
        mov     ax,bx                   ; affichage "../XX/.."
        call    d_ent_a
        mov     cl,4                    ; affichage année
        shr     dx,cl
        mov     ax,dx
        add     ax,80
        call    d_ent_a


                                        ; AFFICHAGE HEURE
        mov     dx,[si+16h]             ; DTA.heure -> dx
        mov     cl,5                    ; on ne s'occupe pas des secondes
        shr     dx,cl
        mov     bx,dx                   ; nombre de minutes dans bx
        and     bx,111111b
        mov     cl,6                    ; nombre d'heures dans ax
        shr     dx,cl
        mov     ax,dx
        call    d_ent_a                 ; affichage des heures
        mov     ax,bx
        call    d_ent_a                 ; affichage des minutes

                                        ; AFFICHAGE DES ATTRIBUTS
        mov     al,[si+15h]             ; lecture attribut
        push    cs
        pop     ds
        mov     word ptr [di],".."      ; on initialise zone attributs à "...."
        mov     word ptr [di+2],".."
        test    al,FA_HIDD              ; si attribut "caché" armé..
        jz      d_ent_6
        mov     byte ptr [di],"H"       ; on remplace le point par "H"
d_ent_6: test   al,FA_SYST              ; idem pour attribut "système"
        jz      d_ent_7
        mov     byte ptr [di+1],"S"
d_ent_7: test   al,FA_RDNLY             ; idem pour attribut "read only"
        jz      d_ent_8
        mov     byte ptr [di+2],"R"
d_ent_8: test   al,FA_ARCH              ; idem pour attribut "archive"
        jz      d_ent_9
        mov     byte ptr [di+3],"A"

d_ent_9: mov    dx,OFFSET d_filnm       ; ...et on affiche la chaine !
        mov     bx,1
        call    fprintf

        pop     bp
        pop     si
        pop     ds
        pop     di
        pop     es
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        popf
        ret
                                        ; CONVERSION EN CHAINE ET DECALAGE
d_ent_a: aam                            ; conversion en BCD
        add     ax,3030h                ; conversion en caractères
        xchg    ah,al                   ; on inverse les digits
        mov     cs:[di],ax              ; écriture en mémoire
        add     di,3                    ; on déplace le pointeur sur suivants
        ret
d_entry  ENDP


;*************************************************
; STRCAT  : place une chaine à la fin d'une autre
; entrées : ES:DI à adresse de la chaine source
;           (terminée par caractère 0)
;           DS:SI à adresse de la chaine destination
;*************************************************
strcat  PROC    NEAR
        push    cx
        push    dx
        push    si

        mov     dx,si                   ; calcul longueur de chaine
        call    strlen
        add     si,cx                   ; on copie après le dernier caractère
```

```
        call    strcpy

strca_1: pop    si
        pop     dx
        pop     cx
        ret
strcat  ENDP


;*****************************************************
; GETDTA  : Donne l'adresse de la DTA
; entrées : Aucune
; sortie  : AX:BX = segment DTA
;*****************************************************
getdta  PROC    NEAR
        push    es

        mov     ah,2Fh                  ;  lecture adresse DTA
        int     21h
        mov     ax,es

        pop     es
        ret
getdta  ENDP


;*****************************************************
; SETDTA  : Positionne l'adresse de la DTA
; entrées : AX:BX = adresse DTA
; sortie  : aucune
;*****************************************************
setdta  PROC    NEAR
        push    ax
        push    bx
        push    dx
        push    ds

        mov     ds,ax                   ; on positionn les paramètres
        mov     dx,bx
        mov     ah,1Ah                  ; selection de la DTA
        int     21h

        pop     ds
        pop     dx
        pop     bx
        pop     ax
        ret
setdta  ENDP

;*****************************************************
; FPRINTF : affiche une chaine de caractères dans
;           le flux BX
; entrées : DS:DX = Chaine
;           BX = handle
; sortie  : Aucune
;*****************************************************
fprintf PROC    NEAR
        push    ax
        push    cx
        call    strlen                  ; calcul de la longueur de la chaine
        mov     ah,40h                  ; affichage
        int     21h
        pop     cx
        pop     ax
        ret
fprintf ENDP
```

```
;=============================================
; LTOA : Conversion d'un entier long en chaine
;        justifiée à droite
;        Pas de code de fin de chaine ('\0')
; Entrées: DX:AX = nombre à convertir
;          BX = nombre de digits total
;          DS:SI = Adresse du début de chaine
;=============================================
ltoa    PROC    NEAR
        push    bx
        push    cx
        push    di
        push    si
        push    bp

        push    dx                      ; on sauve le paramètre dans la pile
        push    ax                      ;  (pour simplifier les accès)
        mov     bp,sp

        add     si,bx                   ; calcul position du dernier caractère
        dec     si                      ;  de la chaine (digit poids faible)

        mov     cx,10                   ; diviseur (base)

ltoa_1: mov     ax,[bp+2]               ; on récupère le mot de poids fort
        sub     dx,dx                   ; la division ne se fait que sur AX
        div     cx                      ; division, le reste est en DX
        mov     [bp+2],ax               ; on sauve le quotient

        mov     ax,[bp]                 ; mot de poids faible
        div     cx                      ; division DX:AX/CX, le digit est en DX
        mov     [bp],ax                 ; sauve quotient

        add     dl,30h                  ; conversion du digit en ASCII
        mov     [si],dl                 ; écriture dans chaine
        dec     si
        dec     bx

        cmp     word ptr [bp],0         ; teste si fini
        jne     ltoa_1                  ; sinon, on boucle
        cmp     word ptr [bp+2],0
        jne     ltoa_1

        mov     cx,bx                   ; on remplit les digits non utilisés
ltoa_2: mov     byte ptr [si]," "       ; par des espaces
        dec     si
        loop    ltoa_2

        pop     ax
        pop     dx
        pop     bp
        pop     si
        pop     di
        pop     cx
        pop     bx
        ret
ltoa    ENDP

Code    ends
        END     start                   ; That'a All Folks !
```

LE COIN DES CODES

La compilation de certains programmes, tels ceux écrits en assembleur, nécessitent souvent un logiciel que ne possèdent pas tous nos lecteurs. Le *Coin des Codes* permet de résoudre ce problème.

Note importante :

Même si la présentation des listings est identique, le traitement de ceux-ci est différente suivant le programme d'entrée et la machine de destination. Chaque listing est prévu pour être entré sur sa machine de destination. Par exemple, ne tentez pas d'entrer un programme HP48 dans un fichier MS-DOS à l'aide du programme MAKEDOS. Vous obtiendrez un fichier que vous ne pourrez pas transférer dans la HP48.

Programmes HP48 et HP28S

Par rapport aux méthodes habituelles sur HP48, notre méthode effectuant un calcul local du checksum en fin de chaque ligne permet de faciliter la recherche d'erreurs, par rapport à une même recherche dans une chaîne de plusieurs centaines d'octets.

Par mesure de sécurité sauvegardez vos programmes et fichiers, éventuellement verrouillez vos cartes.

Tapez les deux programmes correspondant à votre machine:
- pour HP-48 : ASSCOD48 et INPUT48
- pour HP-28S : ASSCOD28 et INPUT28

(*attention aux parties communes et spécifiques HP-48 ou HP-28S et aux commentaires*)
Ceci avec la plus grande attention car leur mauvais fonctionnement peut entraîner un désordre fatal pour les objets contenus dans votre machine. La commande SPEED (JPC-85 page 12) peut être incluse au début de ASSCOD28. NEWLINE (↵) s'obtient sur HP-48 par [flèche bleue] puis [.].

ASSCOD48 996.5 octets
cksum # 5D1Bh

```
« RCLF HEX 64 STWS 1 SF "" 'tmpcod' STO
"nombre d'octets                          @ + NEWLINE
                                          @ NEWLINE
" 17 INPUT48 1 CF STR→ 2 * 16             @ INPUT28
DUP2 / IP 3 ROLLD MOD DUP2 0 > +          @ si HP-28S
3 ROLLD 1 + 4 ROLL # 0h DUP → n r f s o « 1 SWAP
FOR i
 DO "ligne " i 1 - R→B # 1000h + →STR 4 6 SUB + DUP
  "                                       @ NEWLINE
  chaine                                  @ + NEWLINE
  " + "---- ---- ---- ----" n i <
  IF THEN r DUP 4 / IP + SWAP OVER 1 SWAP OVER -
       SUB SWAP 18 + ELSE 38 END 'o' STO +
  "                                       @ NEWLINE
  " + 1 FS?C IF THEN ROT SWAP CLLCD 1 DISP HALT
             ELSE o INPUT48 END DUP       @ INPUT28
  WHILE DUP " " POS DUP                   @ si HP-28S
  REPEAT DUP2 1 SWAP 1 - SUB 3 ROLLD
   1 + 25 SUB IF DUP " " == THEN DROP ELSE + END
  END DROP 0 OVER SIZE 1 SWAP
  FOR j OVER j DUP SUB NUM j * + NEXT
  s + DUP # FFFh AND
  "                                       @ NEWLINE
  somme de controle                       @ + NEWLINE
   ---                                    @ + NEWLINE
  #" 6 ROLL SWAP + 34 INPUT48 STR→ == @ INPUT28
  IF THEN SWAP DROP 1                     @ si HP-28S
  ELSE DROP2 1000 1 BEEP 1 SF 0 END
 UNTIL END 's' STO
 WHILE DUP " " POS DUP
 REPEAT DUP2 1 SWAP 1 - SUB 3 ROLLD 1 + 19 SUB +
 END DROP 'tmpcod' DUP RCL ROT + SWAP STO
NEXT f STOF » tmpcod
@ * HP-48 * @@ ************** HP-28S ****************
"GROB 8 " @@ # 20204A04F3D02C67h # F80004F02C96040Ch
OVER SIZE @@ # 8DCC05081F804F27h # 313103190F818341h
2 / " "   @@ # 681808AE91B51391h # 45DC061171085168h
+ + SWAP  @@ # F3D3CEAA125E5D8Eh # 2F9004h 28 STWS
+ STR→    @@ #0 OR 64 STWS 1 7 START # 3B82h SYSEVAL
# 4017h   @@ NEXT # 20238h SYSEVAL # 4F3Dh SYSEVAL
SYSEVAL   @@ EVAL
# 56B6h   @@ toutes ces lignes marquées @@ peuvent
SYSEVAL   @@ être remplacées par ASC→ :
DROP      @@ soit pour HP-48 (JPC-79 page 11)
NEWOB     @@ ou pour HP-28S (JPC-85 page 8)
"fin" CLLCD 1 DISP »                      @ HP-28S dito
```

INPUT48 412.5 octets
cksum # 15C7h

INPUT28 228 octets
cksum # 9412h

```
« SWAP 1                                  @ partie commune
WHILE OVER CLLCD 1 DISP                   @ HP-48 et HP-28
REPEAT                                    @ puis séquence
 DO UNTIL KEY END                         @ HP-48 ou HP-28
@ ************** HP-48 ************ @@ ****** HP-28S ******
 IF DUP 51 == THEN DROP 0          @@ IF DUP "ENTER" ==
 ELSE IF DUP 55 == THEN DROP       @@ THEN DROP 0
   1 OVER SIZE 1 - SUB             @@ ELSE
  ELSE                             @@  IF DUP "BACK" ==
   CASE DUP 17 < THEN 54 + END     @@  THEN DROP 1 OVER
        DUP 66 < THEN 7 - END      @@    SIZE 1 - SUB
        DUP 76 < THEN 20 - END     @@  ELSE
        DUP 86 < THEN 33 - END     @@ suite voir HP-48
           92 == THEN 48 END       @@
                 46 END CHR        @@
  + DUP SIZE 3 PICK - 2 + 5 MOD NOT @ partie
  1 FC? AND IF THEN " " + END END 1 @ commune
 END END SWAP 60 SUB »                    @ HP-48 et HP-28
```

Donc à partir de maintenant pour tout assemblage de chaîne de codes procédez de manière suivante :

1- lancez le programme ASSCOD48 ou ASSCOD28
2- donnez le nombre d'octets (1/2 octet compris) puis validez avec ENTER.
3- tapez chaque ligne de codes, correspondant au numéro de ligne à 3 chiffres, sans les espaces et validez.
4- tapez la somme de contrôle et validez. S'il y a erreur la ligne de codes sera demandée à nouveau après émission d'un BEEP. L'appui sur EDIT fera apparaître la ligne des codes qui pourra être corrigée. Alors relancez avec CONT.
5- après l'affichage de "fin" stockez le programme assemblé dans la variable donnée en tête.
6- si tout s'est bien déroulé vous pouvez purger tmpcod qui contient la chaîne de codes.

Programmes MS-DOS

Afin d'être utilisé par tous, ce programme est destiné à être écrit en GWBASIC. Il devrait cependant être facile de le convertir pour un autre programme (QBASIC, Turbo BASIC...).

Mode d'emploi :

1- Lancer le programme : GWBASIC MAKEDOS.BAS
2- Entrer le nom du fichier destination.
3- Entrer la taille du fichier.
4- Entrer les listes de codes puis le checksum (en prenant soin d'entrer les codes héxadécimaux en majuscules). En cas d'erreur corriger la ligne, en prenant soin de placer le curseur après le dernier caractère avant de taper sur la touche d'entrée.
5- Une fois que toutes les lignes sont entrées, sortir du GWBASIC en exécutant la commande SYSTEM. Le nouveau programme est immédiatement disponible.

Note : La taille du fichier résultant peut être supérieure d'un octet à ce qui est affiché dans le listing. Cela n'est pas un problème.

Programme MAKEDOS.BAS

```
 10 INPUT "Nom du fichier : ",NOM$ : INPUT "Nombre d'octets : ",N : N=N*2
 20 OPEN NOM$ FOR OUTPUT AS #1 : CLOSE #1 : KILL NOM$
 30 OPEN "bin.tmp" FOR OUTPUT AS #1 : S=0 : P$="---- ---- ---- ----"
 40 NLINES=N\16 : LENLAST=(N MOD 16)+((N MOD 16)\5)
 50 IF (N MOD 16)=0 THEN NLINES=NLINES-1 : LENLAST=LEN(P$)
 60 FOR X=0 TO NLINES
 70   IF X=NLINES THEN P$=LEFT$(P$, LENLAST)
 80   C$=P$
 90   X2$="00"+HEX$(X) : PRINT RIGHT$(X2$,3);":";
100   Y=CSRLIN : LOCATE Y,6 : PRINT C$; : LOCATE Y,6 : INPUT "",C$ : Y=Y-1
110   LOCATE Y,27 : PRINT " sm = ---" : LOCATE Y,33 : INPUT "",D$
120   M=S
130   FOR Z=1 TO LEN(C$)
140     IF MID$(C$,Z,1)<>" " THEN M=(M+((Z-(Z\5))*ASC(MID$(C$,Z,1)))) MOD 4096
150   NEXT Z
160   D2$="00"+HEX$(M) : D2$=RIGHT$(D2$,3)
170   IF D2$<>D$ THEN PRINT "Erreur de somme" : BEEP : GOTO 90
180   FOR Z=1 TO LEN(C$) STEP 2
190     IF MID$(C$,Z,1)=" " THEN Z=Z-1 : GOTO 230
200     CH=ASC(MID$(C$,Z,1))-48 : IF CH>9 THEN CH=CH-7
210     CL=ASC(MID$(C$,Z+1,1))-48 : IF CL>9 THEN CL=CL-7
220     PRINT#1,CHR$((16*CH)+CL);
230   NEXT Z
240   S=M
250 NEXT X
260 CLOSE #1 : NAME "bin.tmp" AS NOM$ : END
```

```
TIMED28S                    (HP28)
# B933h              80.5 octets

     0123 4567 89AB CDEF   sm

000: 76C2 0CD5 701F 7E34   E3A
001: 9CD3 7F1A 0B23 A048   BEE
002: A13C D511 7F61 369A   9E2
003: 2068 D9C6 8D9C 09F2   924
004: 05B9 50AC 110A BAD3   843
005: 814A 0AC1 1011 9D33   51A
006: 1913 22E1 3371 A128   188
007: D113 3920 8990 0000   C85
008: 0000 3210 D1D1 17D8   9E8
009: D37F 6133 7950 09F2   6F8
00A: 0                     728


PROGDEC                     (HP48)
# 7C34h              190 octets

     0123 4567 89AB CDEF   sm

000: D9D2 02BA 81CC D206   E83
001: 1000 1438 FB97 608D   CAD
002: C753 0322 3044 9509   85B
003: 2036 8083 6322 30CC   56C
004: 950F EF30 03D4 303D   317
005: 4303 D43D C136 0D47   0C0
006: 0FD5 502A 1707 E316   DB8
007: 29E2 02D4 2000 F516   A30
008: 8813 0120 4095 236C   67C
009: 2A20 D000 034F 4445   2D4
00A: 4CE4 45B6 7A2C B916   167
00B: D9D2 029E 202D 4100   D9E
00C: C2A2 0D00 0034 F646   A47
00D: 566B 3169 0826 0E51   6E8
00E: 6FBD 8188 0410 8316   2FA
00F: 3915 051B 50BB F06C   1CE
010: 2A20 F000 034F 4445   E30
011: 4F53 9150 5923 0B21   A3C
012: 3019 1363 9150 7E31   68F
013: 630A 50E0 4163 8D30   34C
014: 8C17 06B3 16A3 336D   0E2
015: 9D20 6B31 6BD3 70CA   FB2
016: F067 2D70 4337 0B21   C0A
017: 3079 470B 2130        BF1


ALLDEC                      (HP48)
# 7822h            171.5 octets

     0123 4567 89AB CDEF   sm

000: D9D2 02BA 81D9 F81D   F73
001: 0040 D9D2 0707 619F   D12
002: F30F EF30 03D4 303D   AC6
003: 439B 1360 D470 2A17   7A4
004: 0C2A 20F0 0003 4F44   440
005: 454F 56B3 1690 8260   08B
006: E516 4E76 1391 50CA   E3C
007: 1307 E316 1B54 6881   AFB
008: 307C C30E E170 D9D2   9DA
009: 0881 300F 5163 2230   545
00A: 6365 0C27 36FD E262   38F
00B: 2650 3375 0CAF 0695   152
00C: 236C 2A20 F000 034F   E42
00D: 4445 4A03 9150 C2A2   B44
00E: 0D00 0034 F646 566B   8A6
00F: 3164 E761 3915 051B   543
010: 5018 B022 9E20 2D43   23A
011: 0039 150C 2A20 3100   DA9
012: 054E 4443 4F44 454A   B15
013: 6F36 3915 0B21 305E   7F4
014: 1708 5230 7947 0B21   420
015: 30B2 130              9B8


REV                         (HP48)
# 9069h             58.5 octets

     0123 4567 89AB CDEF   sm

000: D9D2 02BA 81D9 F81D   F73
001: 0040 D9D2 0756 60CC   D95
002: D208 4000 8FB9 7608   AE6
003: F4BF F213 3131 CACA   A0F
004: 1308 19F2 D554 1181   68B
005: 14F1 4A14 914C 171C   414
006: D5BE 8D34 150B 2130   079
007: B213 0                36E


ASCX                        (HP48)
# FC91h              88.5 octets

     0123 4567 89AB CDEF   sm

000: D9D2 0D9F 81D0 040D   DF7
001: 9D20 9023 6A21 70D9   B1A
002: D20B B726 FD55 02C2   916
003: 30A1 626A 1626 C1C1   681
004: 6322 30CC D202 5000   268
005: 8F14 6608 FB97 60CC   17D
006: D814 3130 1741 4313   C82
007: 1179 3193 15B1 9EA8   AD1
008: 0818 6861 5801 7116   665
009: 0CD5 4E8D 3415 0B9F   532
00A: 06B2 130B 2130 B213   125
00B: 0                     155


WHEREIS.COM              (MS-DOS)
                   1688 octets

     0123 4567 89AB CDEF   sm

000: 4D5A 9800 0400 0000   A84
001: 0200 880F FFFF E900   A3B
002: 8000 9D89 0000 0000   4E8
003: 1C00 0000 6469 6574   0D8
004: F99C EB09 0000 0A00   CF2
005: 0000 55F8 9C06 1E57   AC8
006: 5652 5153 500E FC8C   9C3
007: C82E 0106 4900 BA71   6C5
008: 0003 C28B D805 7800   38F
009: 8EDB 8EC0 33F6 33FF   35E
00A: B908 00F3 A54B 484A   193
00B: 79EE 8EC3 8ED8 BE66   222
00C: 00AD 8BE8 B210 EA7D   1F5
00D: 0671 0064 6C7A 203E   F88
00E: 05AA 2000 1707 3F5C   D0C
00F: E921 0301 0000 FEFC   B95
010: 871D F8F0 B883 E00E   A85
011: FA0F C02F FCFF 3F50   AD6
012: 5352 571E 56B4 62CD   990
013: 2189 1E85 01A1 FEFF   9F2
014: FD8E C0BB 8000 8BFB   99F
015: 4732 ED26 8A0F 83FF   997
016: C7F9 0074 4FB2 00BE   82B
017: 0501 83C6 02F0 05FF   69E
018: FF3C 2275 0980 F2FF   5BA
019: 26C6 0500 EB2B 3C20   365
01A: 75DF FD27 80FA FF74   3B7
01B: 22F1 263A 4501 7F74   0E8
01C: 7418 8BDF 4326 803F   EFF
01D: DB01 F9FF E30D 740B   E27
01E: 891C 4646 FF06 1747   B99
01F: E2EE F8C2 DB8B 0EF5   C50
020: 495E 1F5F 0722 5A5B   AB3
021: 58C3 8501 2DFB 3FDE   B56
022: FFFD 5351 5206 7CC7   97B
023: 0600 02FF F13B FFA1   987
024: DE3B 0602 0274 3775   59D
025: FD78 878B 1EF2 D1E3   50C
026: 8B38 FFFF 823A 0604   2FA
027: 0275 1E1E 061F 8BD7   228
028: BFC7 E8A1 001F 890E   09C
029: 086C 89A2 9D81 0702   D6E
02A: E606 96D5 E3FD EB5B   E61
02B: BA90 E86D 0083 6754   B4A
02C: FF81 8BDA 807F 013A   9B8
02D: 7532 833E D6E0 AEC3   978
02E: 22E0 B103 E889 3E92   77A
02F: 71F4 3B6D C77F 732B   6A3
030: 523D 2D74 25EB 2883   48D
031: C702 77A7 E7EB 1FCE   5A2
032: 7746 30D2 FF22 C6FC   59B
033: 4799 2DEB 0DB0 FFEB   74C
034: 0CC6 0691 3F37 FCB6   6B0
```

```
035: A0F8 3507 EFF1 5A59  5A9
036: 5BC3 5397 3774 0738  226
037: 3FF6 0774 0643 EBF4  0CD
038: BB39 8BD3 EA3B 8E33  033
039: C9E8 0441 EBB8 BFF7  0E2
03A: F050 5756 3B88 043C  E39
03B: FFF9 EE47 46EB F35E  E9E
03C: 5FFA 5FC1 643A 6873  CA1
03D: 7261 2148 FFFF 7573  B84
03E: 6167 6520 3A20 2077  752
03F: 6865 7265 6973 EFBB  703
040: 205B 2D68 5DFB 73FB  776
041: DDEE 72FB 61FB 6477  687
042: FF20 7061 74E7 6669  3D2
043: 6C65 2E2E 1FE3 2E5D  39D
044: 0D0A 0A24 FC44 69CC  316
045: 6374 7B07 6F72 79BE  23C
046: 66E1 313F A2FD 6878  121
047: 78D0 C5FE FE20 8863  FC3
048: 66C7 E36F 756E 64EA  F7F
049: 242E 005C FE2A 2EFF  082
04A: 732A 008C CB8E DB8E  1A4
04B: C3B4 4ABB D4DC 7F60  190
04C: BC00 10B8 0038 BA12  E93
04D: 03F0 E7F5 A98B 07A2  D54
04E: 95FF E7C6 0700 E83E  C0D
04F: FDE8 BEFD 3C14 FFF3  D1C
050: 463C 6875 0780 0E62  9E6
051: 0343 77FC 393C 73F5  89C
052: 04EB 2E3C 8EDF 72F5  8EF
053: 01EB 233C 61F5 F15F  806
054: 20EB 183C 6475 0C8B  67A
055: 18C7 FFBE B3E8 68FE  7D7
056: EB08 3C3F 7504 7FF0  670
057: FE06 0D03 EBB3 803E  518
058: FA61 3F6E 09BA 6503  2FB
059: B409 98EB 3FA1 DEDB  46D
05A: E740 7403 A3D8 FCA5  3D9
05B: 85E8 2AFE EDF8 98C3  485
05C: D980 78FF 5CCB C640  3FB
05D: E4FB FF1E 078B FAE8  500
05E: 1900 A110 0333 D25C  1F3
05F: EEBE 78BB 39E8 F577  196
060: 0702 BA02 04BF B44C  080
061: FCCB B59E 2055 1FFE  07C
062: F7E8 B802 50B4 48BB  F0E
063: 0900 ECFF C757 068B  E2D
064: EC33 DBE8 AF02 2EA1  D7C
065: A72E C8C5 A39A 2EA8  D56
066: 6486 8BB9 B946 000B  AFE
067: 8B7E 661F FF46 048E  A22
068: D8BE 30B6 CAFD 0E07  9DA
069: BF1D 04C1 81E8 6CDF  A0D
06A: 6DD6 2F3F 2208 BF20  824
06B: F25E 1B16 A42D C7DF  875
06C: 2E5E 6671 BFD8 F639  7E8
06D: E846 37CE B963 DA26  6E9
06E: 813D C1CF 9D75 0D2A  64E
06F: 7D02 2AE9 75FE EC01  5DA
070: B44E 8BD6 FDFF B916  683
071: 8472 6A8A 1E15 0080  30E
072: E33F 79EE 2EA0 F374  22E
073: 7516 1F7E F6C3 1074  FE9
074: 152E 06DB 7448 FC3F  FB6
075: F91E 002E 7441 EB04  D57
076: 84C3 743B 2E07 EF18  C18
077: 2AE6 6677 EE75 295E  B09
078: 011E 5DF0 6B0E 1FBE  B92
079: AFF0 37CE 60FF F8CC  D02
07A: 01BB 3166 FFBA 43E8  CBE
07B: EE01 1F33 F6AF 3BE8  CDA
07C: CB00 B44F C194 4BEE  CD2
07D: BD74 13B7 1D5E F9B3  C5C
07E: 2E39 B2E9 15FF BC03  BCF
07F: 07FE 18DA 47E2 FCB8  C5B
080: 1F26 8476 6010 D6BA  A7E
081: F053 B912 7744 5537  6DB
082: A056 2462 2AC7 0E6A  560
083: 2384 BB5D CEB0 CE1D  648
084: C0E6 526D F8E2 E878  5A0
085: E74A C6FE E892 FE88  61C
086: C78F 7A58 5807 B449  3D8
087: F758 9D4C 0169 995A  20A
088: 5D59 926D 2101 90F4  F14
089: 0EFE 2EFB F3FF 7F20  F91
08A: 6A6A 2F6D 6D2F 6161  DC9
08B: 2068 683A 4CB7 F720  C02
08C: 48ED 4161 B98F 633C  AA2
08D: 4449 523E F9AA 1A9C  A99
08E: 3E3F 558E F9D3 5649  93F
08F: BF76 06F3 FFBA 1EB9  A05
090: 0D00 FCA4 4980 C0F9  8CB
091: 79E6 2620 1FF9 8BFF  948
092: 3876 00BF 8206 8A44  6AC
093: 15A8 4F0E B908 7DFE  733
094: CBA1 061E DBF3 A4FB  7F1
095: 3F1F EB15 E58B 441A  6A9
096: 8B54 1CBB CC9E E941  69C
097: F7E8 D341 399F 5EBF  6EE
098: 8BFD 549D F9E7 C225  618
099: 1F00 B105 D3EA 2BAE  637
09A: 3983 E30F 9328 7E1E  4E0
09B: DF01 930E CD47 7FC3  45A
09C: E867 E604 E6C2 0550  14D
09D: C40E 1F5B EF54 16D7  075
09E: DFE5 3FB1 06E8 0EDD  0B6
09F: 3DDA 4100 8A66 E795  EC1
0A0: C705 46C7 459D 67E2  D1F
0A1: 2EA8 4603 6C48 A88C  BE3
0A2: F904 9848 53F0 EFA8  B55
0A3: B0F8 0252 A820 F89E  9EC
0A4: D303 41BA ED53 6F86  8EC
0A5: E844 005D 40FE 3F58  7A8
0A6: 9DC3 D40A 0530 3086  3F4
0A7: E02E 8905 3D90 FD03  215
0A8: C311 18D6 A5FD E3FB  2F8
0A9: 03F1 E849 FB5E DEC3  3E5
0AA: 06F9 33B4 2FB2 8CC0  2F1
0AB: 073C 7C9E E41E 4E22  1D4
0AC: 198D B41A F01F 39FE  1E8
0AD: 1150 51E8 16FB B440  FD9
0AE: F259 F43C 73D9 1E55  E4C
0AF: 5250 7E9E DD03 F340  C90
0B0: BFE7 7F52 022B D2F7  B5F
0B1: F189 6897 F948 F88B  ACA
0B2: FB37 C230 8814 4E4B  8B2
0B3: 837E 668F F8E1 FA02  808
0B4: FADB 8BCB C63F 1A04  738
0B5: 204E E2FA 585A 725F  653
0B6: B503 00FF D1ED FECA  84B
0B7: 7505 AD8B E8B2 10C3  6CA
0B8: E8F1 FFD0 D7E8 ECFF  939
0B9: 7214 B602 B103 E8E3  721
0BA: FF72 09E8 DEFF D0D7  829
0BB: D0E6 E2F2 2AFE B602  73B
0BC: B104 FEC6 E8CD FF72  829
0BD: 10E2 F7E8 C6FF 730D  802
0BE: FEC6 E8BF FF73 02FE  8A3
0BF: C68A CEEB 2AE8 B4FF  A6D
0C0: 7210 B103 B600 E8AB  8F1
0C1: FFD0 D6E2 F980 C609  76D
0C2: EBE7 AC8A C883 C111  5A3
0C3: EB0D B103 E895 FFD0  54E
0C4: D7E2 F9FE CFB1 0226  410
0C5: 8A01 AAE2 FAE8 84FF  51D
0C6: 7303 A4EB F8E8 7CFF  6B2
0C7: ACB7 FF8A D872 81E8  607
0C8: 72FF 72D6 3AFB 75DD  6B3
0C9: 5D0E 1FB9 0000 E31C  47D
0CA: BE07 00AD 0BC0 7809  24C
0CB: 03C5 8EC0 AD8B D8EB  3AA
0CC: 06D1 E0D1 F803 D826  19F
0CD: 012F E2E7 012E E006  F4C
0CE: 012E E606 585B 595A  D8B
0CF: 5E5F 1F07 9D73 13FA  C7D
0D0: 81C5 F0FF 8ED5 BC00  C32
0D1: 00FB 33ED 55EA 0001  9B2
0D2: F0FF 5DEA 0000 0000  58E
```

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.

PPC Paris est le représentant Français de **HEX** (**H**andhelds **E**uropean Clubs EXchange), la fédération des principaux clubs Européens d'utilisateurs de calculateurs HP.

La maquette de ce numéro a été préparée et réalisée par Jacques Belin et Asdin Aoufi.

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

Les informations et programmes parus dans ce journal sont publiées "Tels quels" et ne peuvent en aucun cas engager la responsabilité de Hewlett-Packard ou de PPC Paris. Hewlett-Packard se réserve le droit de ne pas répondre aux questions concernant le sujet de certains articles.

Les programmes publiés peuvent être utilisés librement. Cependant, ils ne peuvent être vendus ou fournis dans un ensemble commercialisé, sous quelque forme que ce soit, sans l'accord écrit de l'auteur ou de PPC Paris.

Directeur de la publication : Jacques Belin
Numéro ISSN : 0762 - 381X

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

Imprimé par Paris Copie, 4 rue Linné, 75005 Paris