LIST

n°1

LE JOURNAL DES AMATEURS DE PROGRAMMATION

JUILLET-AOÛT 1984

A l'essai : le Basic du nouveau Thomson MO5

- Coup d'œil sur trois logiciels : Compactor pour TO7 Basic étendu du TI 99/4A Tool pour Commodore 64
- Ordinateur de poche, ordinateur de table : évaluez le Basic de votre machine avec les 10 tests de LIST
- En avant pour Forth !
- Histoire des langages de programmation : les débuts
- Calculatrices, ordinateurs de poche, ordinateurs domestiques : un trésor d'idées pour mieux programmer

M 2712 - 1 - 20 F

l'Ordinateur de poche

Belgique : 166 FB - Canada : 2.95$C - Suisse : 6 FS

20 F

FAITES CIRCULER...

S'IL y a mille et une manières de programmer, la meilleure et la plus agréable reste certainement de le faire par goût. Dans ce domaine, en effet, l'amateur est un privilégié.

La petite informatique a connu ces dernières années un succès et des perfectionnements tels que l'on trouve aujourd'hui, à des prix abordables, des ordinateurs (de poche ou de table) dont on n'aurait pas osé rêver il y a dix ou vingt ans. L'informatique n'est plus l'apanage des spécialistes, et la programmation devient un loisir d'une richesse exceptionnelle.

Contrairement au professionnel, l'amateur n'a aucune obligation de résultat, aucun projet prédéfini par quelqu'un d'autre que lui-même, aucun délai à respecter. Son activité s'exerce donc sans contrainte, comme s'il s'agissait d'un travail de recherche fondamentale. Son souci n'est pas avant tout de se faciliter la tâche coûte que coûte, d'éviter systématiquement les difficultés, mais plutôt de les affronter, et de les surmonter. Chez l'amateur isolé, combien de trouvailles dorment dans des tiroirs ?

Souvent, on se pose des problèmes délicats pour la seule satisfaction de les résoudre du mieux possible. Que l'on soit débutant ou non, ce qui compte, c'est le plaisir de la découverte, même si l'on procède un peu à l'aveuglette. Bien sûr, on peut réinventer la roue. Il n'empêche, cette belle liberté permet d'explorer en dehors des sentiers battus. Programmeur du dimanche, programmeur du samedi, oui... mais c'est le dimanche souvent que se font les découvertes. Encore faut-il les faire connaître.

Par ailleurs, la qualité de la programmation n'est pas nécessairement liée au matériel utilisé. On peut, sur un PC-1211, découvrir « le » truc qui rendra de grands services aux utilisateurs de TO7, d'Oric, de Spectrum... Quelle que soit la machine pratiquée, nous avons une mine d'idées à creuser. Qui peut se targuer d'avoir fait le tour d'une « simple » calculatrice programmable ? Personne. Faites circuler, faites circuler vos idées.

Voilà pourquoi LIST vous engage dès maintenant à prendre votre plus belle plume (ou votre plus belle bille). Langage-machine, assembleurs, Pascal, Logo, Forth, Basic, notation polonaise inverse ou algébrique, etc., trouvailles d'intérêt général ou destinées à un matériel particulier, algorithmes originaux, nous attendons dès maintenant votre courrier : critiques, suggestions, propositions d'articles...

Le dialogue entre un journal et ses lecteurs ne peut s'établir que si les rôles s'inversent de temps à autre, c'est-à-dire si les lecteurs écrivent et si la rédaction consacre une partie de son temps à lire les lettres qui lui sont adressées.

C'est en nous écrivant donc que vous contribuerez à ce que LIST corresponde encore mieux à ce que vous attendez de votre journal.

LIST

LIST — LE JOURNAL DES AMATEURS DE PROGRAMMATION

N° 1 - JUILLET-AOÛT 1984

SOMMAIRE

Ce numéro contient en encart des bulletins d'abonnement paginés 19, 20, 85 et 86.

RÉDACTION-RÉALISATION
Directeur de la rédaction : Bernard Savonet
Rédacteur en chef : Jean Baptiste Comiti
Rédaction : Anne-Sophie Dreyfus
Conception graphique et secrétariat de rédaction : Eliane Gueylard
Assistante de rédaction : Maryse Gros
Administration : Marie-Hélène Muniz

Ont collaboré à ce numéro : Antoine, Olivier Arbey, Michel Arditti, Claude Balan, François J. Bayard, Catherine Bellamy, Viviane Bazin, Frédéric Blondiau, Robin Bois, Pierre Borenstein, Bernard . Bouvier, Christian Boyer, Gilles Bransbourg, Jean-Pierre Brunerie, Eric Buhr, Jean-Paul Carré, Henri Casal, Thierry Chamoret, Sylvain Clément, Raymonde Coudert, Geneviève Cuvelier, Robert Daguesse, Jacques Deconchat, Philippe François, Pierre-Ladislas Gedo, Béatrice Ginoux Defermon, Max Hagenburger, Renée Koch, Jean-Christophe Krust, Xavier de La Tullaye, Jean-Pierre Lalevée, Bernard Lambey, Raoul Lebastard, Patrick Leclercq, Jean-Charles Lemasson, Thierry Levy-Abegnoli, Alain Mariatte, Pierrick Moigneau, Thierry Muller, Claude Nowakowski, Yvon Pérès, Jean-Jacques Robert, David Rousseau, Benoît Thonnart, André Turlure, André Warusfel, Marc Wisniewsky.

Illustrations : Frapar, Philippe Burel, Antoine Chereau, Dominique Cuesta, Bernard Helme, Philippe Mairesse, Alain Mangin, Hervé Marly, Jacques Mesen, Alain Mirial, Alain Prigent, Nestor Salas, Nicolas Spinga, Elisabeth Tournier.

ÉDITION-PUBLICITÉ-PROMOTION
Éditeur : Jean-Pierre Nizard
Éditeur-adjoint : Jean-Daniel Belfond
Administration : Maryse Marti, assitée de Anne Stolkowski
Publicité : Colette Sauvant, assistée de Nadine Schops

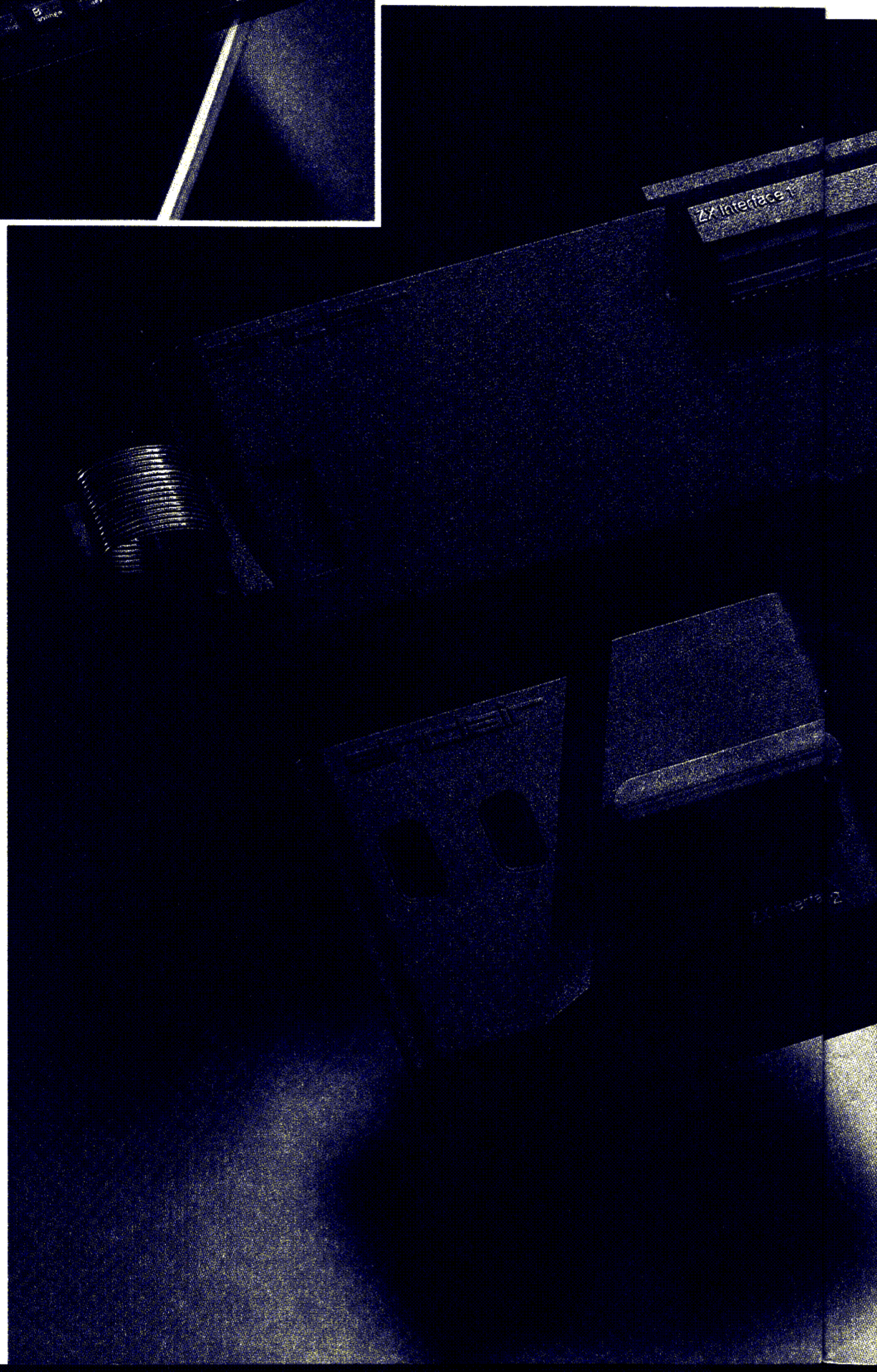
VENTES
Diffusion NMPP : Sophie Marnez
Abonnements : Muriel Watremez assistée de Sylvie Trumel, Cécilia Mollicone et Dominique Loridan

Directeur de la publication : Jean-Luc Verhoye

LIST est une publication du
5 place du Colonel Fabien
75491 PARIS CEDEX 10
Téléphone : (1) 240 22 01
Télex : LORDI 215 105 F

groupe tests

SINCLAIR s'impose par la passion des Sinclairistes. Ils sont 2 millions dans le monde à avoir découvert Sinclair. Les revues et les nombreux clubs en sont l'écho.



Fiche technique du ZX SPECTRUM

Unité centrale
Microprocesseur Z 80 A, 3,25 MHz.
RAM 16 K ou 48 K.
ROM 16 K.

Clavier
40 touches avec répétition automatique et témoin sonore. Système d'entrée de toutes les fonctions par mots-clefs.

Affichage
31 x 24 caractères, majuscules ou minuscules. Haute définition graphique 256 x 192 (49 152 points adressables individuellement).

Générateur de caractères
ASCII étendu (matrice 8 x 8). 21 caractères programmables. Possibilité de redéfinition de l'ensemble des caractères.

Couleurs et sons
8 couleurs. Haut-parleur intégré 130 demi-tons (10 octaves). Amplification par prise micro.

Langages
Basic intégré, Pascal, Assembleur et Forth en option.

Interface magnétophone
Vitesse de transmission : 1500 bauds. Sauvegarde de pages mémoire et tableaux séparés. Fonctions VERIFY et MERGE.

Ecran
Raccordement sur prise antenne pour récepteur PAL ou prise PERITEL pour récepteur SECAM.

Nous sommes à votre disposition pour toute information au 359.72.50.
Magasins d'exposition-vente :
Paris - 11 rue Lincoln 75008 (M° George V)
Lyon - 10 quai Tilsitt 69002 (M° Bellecour)
Marseille - 5 rue St-Saëns 13001 (M° Vieux-Port).

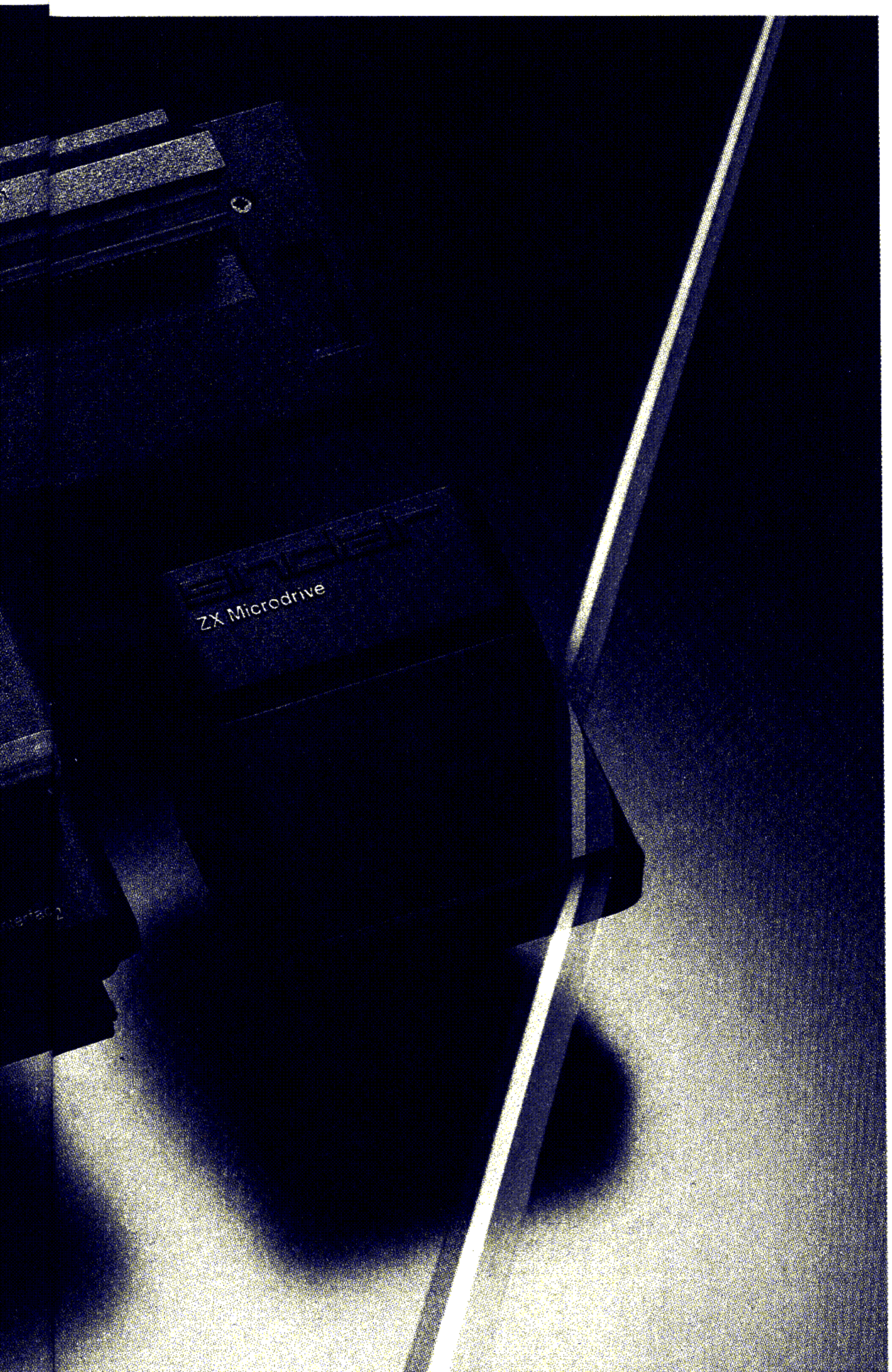
LIST - PAGE 6

Sinclair s'impose.

Sinclair s'impose par la richesse unique de sa gamme de logiciels et de par sa bibliographie incomparable.

Sinclair s'impose par sa capacité d'innovation et son souci de la perfection, à des prix abordables par tous.

Les 3 nouveaux périphériques du ZX SPECTRUM en sont la preuve. Découvrez-les d'urgence.



Le Microdrive ZX

Une prouesse technologique dans le domaine de la mémoire. Chaque microdrive utilise des bandes sans fin interchangeables, d'une capacité de 85 K octets. L'accès à la mémoire s'effectue en un temps record. Ainsi, un programme de 48 K octets se charge en 9 secondes. 8 microdrives peuvent être connectés au SPECTRUM, qui dispose alors d'une capacité de 680 K octets en ligne. C'est incomparable.

L'Interface ZX 1

Une extension qui transforme votre micro en géant. Elle permet, outre le raccordement des microdrives, de gérer des fichiers et de brancher des imprimantes de format courant. De plus, elle autorise l'établissement d'un réseau de communication à vitesse élevée, pouvant regrouper 64 SPECTRUM. Et toujours à un prix Sinclair.

L'Interface ZX 2

Avec elle, le plaisir est total. Elle lit instantanément les nouvelles cartouches ROM de jeu et permet le branchement simultané de 2 manettes de jeu.

Ce nouveau périphérique peut se brancher directement sur le micro-ordinateur ou sur l'interface ZX 1.

Le ZX SPECTRUM constitue alors un incomparable système informatique. Sinclair s'impose.

Présent au Sicob Printemps Stand 344 et à Micro-Expo Stand N 10.

sinclair

la micro-ordination

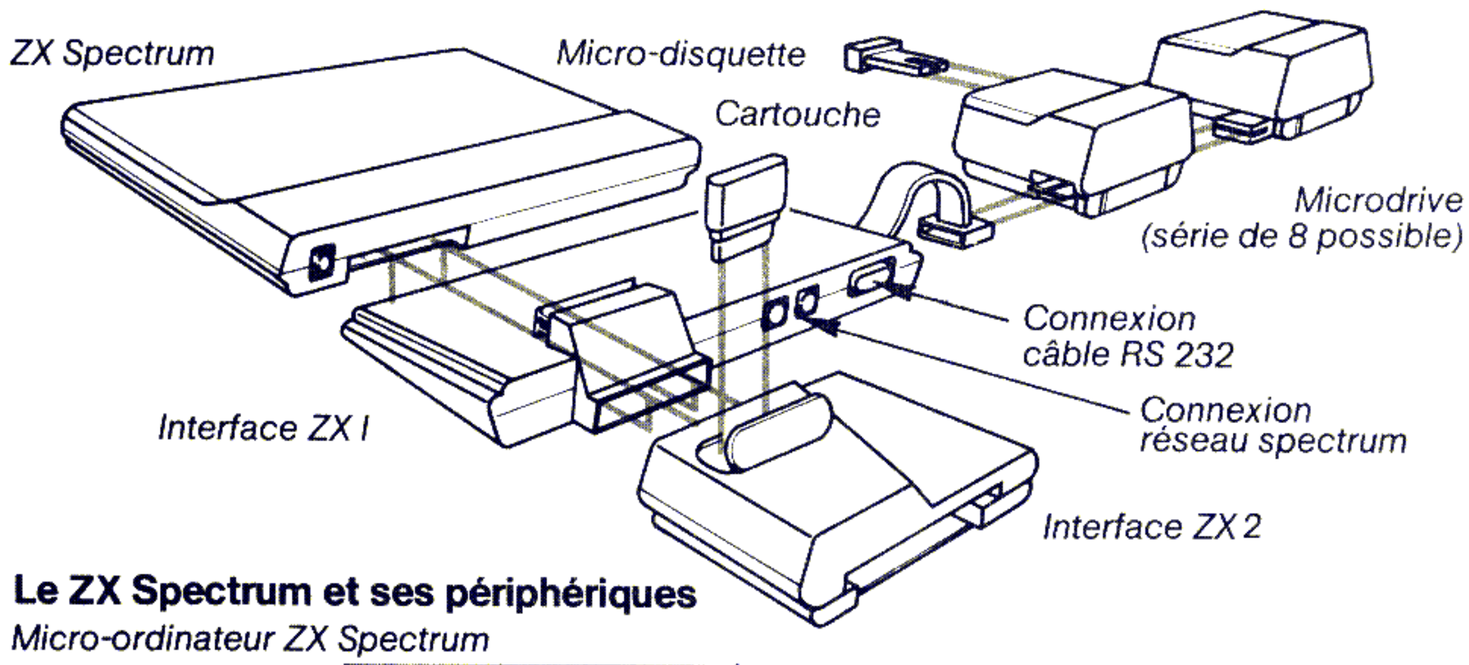
Bon de commande au verso.

ZX Spectrum. Un incomparable système informatique.

CRÉATEURS CONSEILS

Bon de commande

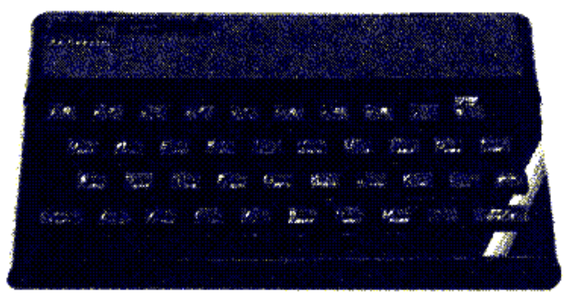
A retourner à Direco International - 30, avenue de Messine - 75008 Paris.



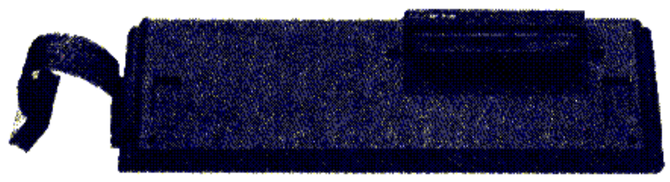
Le ZX Spectrum et ses périphériques

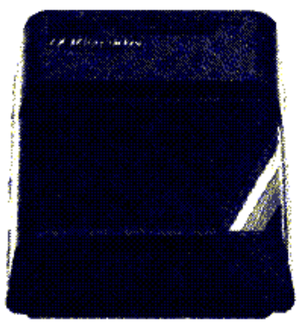
Micro-ordinateur ZX Spectrum

16 K RAM PAL	1490 F x		SS 01
48 K RAM PAL	1965 F x		SS 02
16 K RAM Péritel	1850 F x		SS 03
48 K RAM Péritel	2325 F x		SS 04



Interface ZX 1	895 F x		SS 05
Câble RS 232	235 F x		SS 06





Microdrive ZX

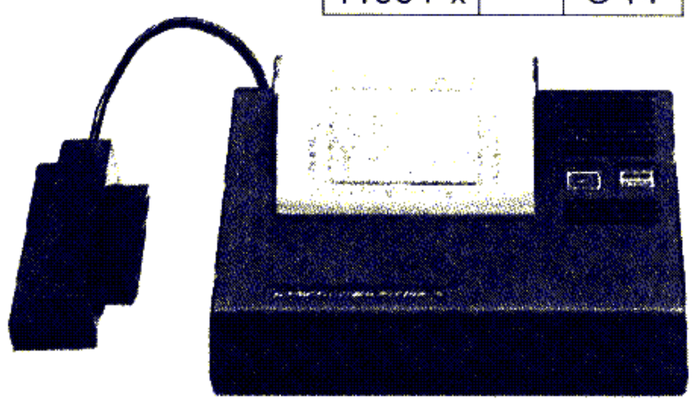
940 F x		SS 07

Boîte de 4 microdisquettes vierges

316 F x		SS 09

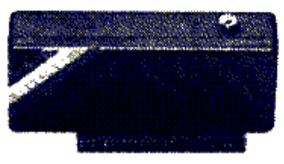
Imprimante Alphacom 32

1190 F x		C 14



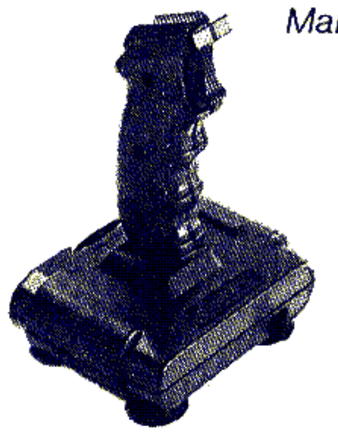
Boîte de 5 rouleaux de papier

150 F x		P 02



Modulateur noir et blanc

190 F x		CS 04

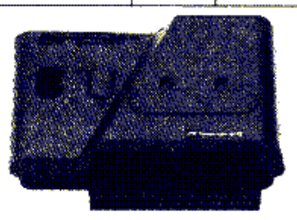


Manette de jeux Quickshot

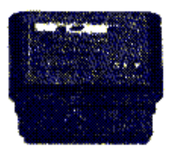
140 F x		C 15

Interface ZX 2

351 F x		SS 10



Les logiciels-cartouches

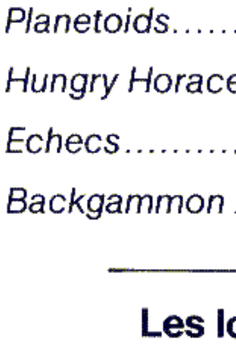


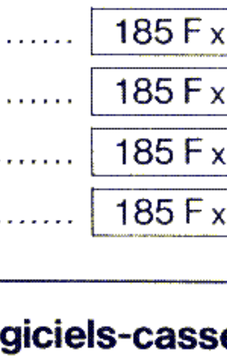
Pssst !	185 F x		RS 01
Jet Pac	185 F x		RS 02
Cookie	185 F x		RS 03
Trans Am	185 F x		RS 04
Space Raiders	185 F x		RS 05
Planetoids.....	185 F x		RS 06
Hungry Horace.....	185 F x		RS 07
Echecs.....	185 F x		RS 09
Backgammon	185 F x		RS 10

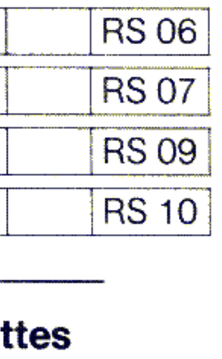
Les logiciels-cassettes

JEUX DE RÉFLEXION

Cobalt (simul. de vol)	95 F x		JS 01
Echecs.....	115 F x		JS 15
Othello.....	75 F x		JS 02
Manager.....	140 F x		JS 16







UTILITAIRES

Pascal.....	260 F x		US 01
ZX Trans.....	95 F x		US 03
Devpac (Ass/Desass)	160 F x		US 02

JEUX D'ARCADES

Jumping Jack.....	95 F x		JS 17
Zoom.....	95 F x		JS 18
Alchemist	95 F x		JS 23
Mined-Out.....	86 F x		JS 05
Androïdes.....	75 F x		JS 07

GESTION

Direction financière.	120 F x		GS 01
Gestion de fichier ...	115 F x		GS 02

TOTAL : ..... F

Indiquez dans chaque case la quantité commandée. Effectuez le calcul du total et inscrivez le résultat dans la case TOTAL.

Votre commande vous sera adressée sous 3 semaines.

Je paie par : chèque bancaire ..... ☐

CCP..... ☐

établi à l'ordre de Direco International, joint au présent bon de commande.
(aucun chèque n'est encaissé avant l'expédition du matériel).

contre-remboursement* ☐

** Contre-remboursement taxe PTT (14,20 F) pour toute commande de moins de 2000 F. Au-delà, barème Sernam.*

Nom _____

Prénom _____

Adresse _____

_____

Code postal ☐☐☐☐☐ Tél. : _____

Signature (pour les moins de 18 ans, signature de l'un des parents):

LLF1

Au cas où je ne serais pas entièrement satisfait, je suis libre de vous retourner le matériel dans les 15 jours. Vous me rembourserez alors entièrement.

sinclair

la micro-ordination

LE LASER 200

UN MICRO ORDINATEUR COULEUR SECAM

VRAIMENT TRÈS ÉTONNANT.

1490 F TTC

Microprocesseur Z 80 A • Langage Microsoft Basic • Affichage direct antenne télé SECAM • Clavier 45 touches pleine écriture, + clef d'entrée, + graphismes, + bip sonore anti-erreurs... • Texte + graphismes mixables 9 couleurs • Edition et correction plein écran • Son incorporé • Toutes options : extension + 16 K + 64 K, interface imprimante, imprimante, stylo optique, manettes, jeux, modem, disquettes...

eva w

VIDEO TECHNOLOGIE FRANCE

19, rue Luisant - 91310 Montlhéry
Tél. (6)901.93.40
Télex SIGMA 180114

BON DE COMMANDE

A retourner à : VIDEO TECHNOLOGIE - 19, rue Luisant - 91310 Montlhéry
Tél. (6)901.93.40 - Télex SIGMA 180114

Je désire recevoir :
LASER 200 SECAM comprenant :
Le LASER 200 avec son modulateur SECAM incorporé se branchant directement sur l'antenne du téléviseur.
+ Câble de liaison fiches jack pour lecteur de K7
+ Câble de liaison micro/télé ou moniteur
+ Livre technique (150 pages) de BASIC
+ Livret d'exercices
+ Manuel de mise en route
+ Cassette de démonstration en français
+ Garantie
..... 1.490 F TTC

EXTENSION-PERIPHERIQUES-INTERFACES LASER 200

Extension mémoire 16K	590 F TTC
Extension mémoire 64K	1.190 F TTC
Lecteur préréglé de cassettes type DR 10	570 F TTC
Paire de manettes de jeux avec son interface	320 F TTC
Interface d'imprimante "Centronic parallele"	320 F TTC
Imprimante 4 couleurs papier standard	2.190 F TTC
Interface disquette (en préparation)	N.C.
Stylo optique (en préparation)	N.C.

LOGICIELS LASER 200
Cassettes avec programmes 4K ou 16K ... 79 F TTC
(Voir liste détaillée constamment augmentée)

TOTAL DE MA COMMANDE :

Je choisis de payer le total de ma commande :
☐ Au comptant, par CCP, chèque bancaire, ou mandat, à l'ordre de VIDEO TECHNOLOGIE FRANCE
☐ Contre-remboursement au transporteur, moyennant une taxe de 60 F.

Nom
Prénom
N°
Rue
Ville
Code Postal

Signature

L1

Liste de plus de 100 revendeurs, sur simple demande

Pour votre ordinateur, Boîtes à outils MégaO Poche.

Mine de rien, une mine d'idées.

La collection MégaO Poche met à votre disposition des petits programmes tout prêts qui vous permettront de résoudre les nombreux problèmes de la vie quotidienne, de simplifier votre comptabilité, votre tableau de bord, la tenue de vos fichiers etc...

MegaO POCHE

Actuellement 12 titres :
Boîte à outils pour PC 1500
Boîte à outils pour FX 702 P
Boîte à outils pour Sinclair
Boîte à outils pour PC 1251
Boîte à outils pour PB 100 et FX 802 P, TRS-80 PC-4

SPECIAL VACANCES

NOUVEAU

Boîte à outils pour le navigateur de plaisance
Boîte à outils pour ORIC - Tome 1
Boîte à outils pour ORIC - Tome 2
Boîte à outils pour Commodore 64
Boîte à outils pour TI-99/4A
Micro gestionnaire pour Sinclair
Micro compta pour Sinclair

La collection MégaO Poche est diffusée par :
P.S.I. DIFFUSION B.P. 86 77402 Lagny-sur-Marne Cedex
Tél. : (6) 006.44.35 Télex : PSIDIF 600978 F

Envoyer ce bon accompagné de votre règlement à P.S.I. DIFFUSION ou pour la Belgique et le Luxembourg à P.S.I. BENELUX ou pour la Suisse à P.S.I. SUISSE.

P.S.I.

P.S.I. DIFFUSION BP 86 - 77402 Lagny-S/Marne Cedex FRANCE
Téléphone (6) 006.44.35

P.S.I. BENELUX
5, avenue de la Ferme Rose
1180 Bruxelles BELGIQUE
Téléphone (2) 345.08.50

En SUISSE
P.S.I. Suisse
Route Neuve 1
1700 Fribourg
Tél. : (037) 23.18.28
CCP 17-56-84

Prix		
35 FF	250 FB	12,20 FS
Frais de port par envoi		
10 FF	50 FB	1,50 FS
par avion, ajouter 8 FF (75 FB) par livre		

NOM _____

ADRESSE _____

_____

Code postal |_|_|_|_|_| Ville _____

☐ *Paiement par chèque joint* ☐ *Paiement en FF par carte bleue VISA (à P.S.I. DIFFUSION uniquement) (montants supérieurs à 50,00 FF exclusivement)*

_____

N° |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_| *Date d'expiration* |_|_|_|

Signature (obligatoire pour paiement par carte de crédit)

DESIGNATION	PRIX
TOTAL	

POURQUOI DIABLE PROGRAMMEZ-VOUS ?

CALCULATRICES programmables, ordinateurs de poche ou de table, c'est apparemment le même virus ou (qui sait ?) le même démon. Une fois qu'il s'est manifesté, on change de loisirs. On délaisse ses distractions favorites pour se consacrer à la programmation. Que s'est-il passé ?

Nous sommes des dizaines de milliers à pratiquer la programmation.
Et le virus est contagieux... Pourquoi pianoter des heures durant (la nuit parfois) sur le clavier d'une machine ?
Vous lirez ici différentes réponses à cette question.
Elles proviennent de gens
qui, pour la plupart, ne sont pas informaticiens de profession.
Certains sont jeunes, d'autres moins.
Ce qu'ils ont en commun : l'amour de la programmation.
Comme vous le verrez, quelques-unes des raisons qu'ils avouent concordent, mais l'ensemble est très varié.

Peut-être vous reconnaîtrez-vous en partie, ici ou là, dans ces courtes réponses ? Suffisent-elles, même rapidement, pour faire le tour de la question ? Probablement pas.

Il est possible que vous aimiez la programmation pour d'autres raisons encore. Dans ce cas, si cela vous dit, écrivez à LIST. Essayez d'expliquer à votre tour "pourquoi diable programmez-vous ?"
Quel démon vous a saisi ?

JBC

UNE FOIS POUR TOUTES

MON travail dans un laboratoire photo m'amène à manipuler des ordinateurs très "sérieux" : contrôle de qualité, facturation, statistiques... Ceux-là, pour l'instant, pas question de les programmer moi-même !

En revanche, pour un tas de petits calculs jusque-là assez pénibles, mon PC-1251 fait parfaitement l'affaire : il me permet ainsi de traiter facilement calculs d'horaires, statistiques sur différents films ou contrôles d'échantillons de production.

Le plus gros intérêt de la programmation apparaît alors : il suffit de se creuser la cervelle une fois pour toutes et de créer un programme. Quand ce programme tourne rond, c'est fini, plus besoin de réfléchir... en tout cas à ce problème-là !

Pierrick MOIGNEAU

JE PASSE POUR QUELQU'UN...

LE plus souvent, je mets au point des formules difficiles à comprendre. Et j'ai un très bon prétexte pour cela, je peux expliquer qu'elles sont courtes, qu'elles permettent de remplacer cinq lignes d'instructions, qu'elles contribuent à l'optimisation et à la simplification des programmes.

Et comme personne autour de moi n'arrive à refaire mon raisonnement pour reconstituer la formule obtenue, et comme chacun constate qu'elle fonctionne bien, je passe pour quelqu'un qui s'y connaît bien ! C'est pour cela que je programme.

Oui, je sais que mes formules alambiquées vont à l'encontre de la programmation structurée dont, justement, l'objet est de concevoir des programmes simples à comprendre. Mais c'est tellement agréable de montrer sa supériorité technique...

Thierry MACTHORE

UN JOUR, MON MÉTIER

AU début, c'était un simple passe-temps pour m'occuper entre deux cours. Des camarades de classe s'y sont mis à leur tour, et une sorte de compétition s'est instaurée entre nous. L'esprit de défi s'y est mis, puissante incitation s'il en est. Tout ce que mes camarades parvenaient à faire, il fallait que je le fasse à mon tour.

Ensuite, le côté "outil de travail" a prévalu. Ce n'était plus dès lors vraiment un jeu. Maintenant, j'essaye de toucher à tout ce qui se rattache à la programmation, par curiosité, bien sûr, mais aussi dans l'espoir d'en faire un jour, peut-être, mon métier.

Olivier ARBEY

LOGIQUE ET INTUITION

IL fallait bien qu'un jour quelqu'un me demande pourquoi je programme... Croyez-le ou non : c'est la première fois que l'on me pose la question. On m'avait souvent demandé comment je programme. On m'avait parfois demandé sur quel matériel je programme, mais pourquoi je programme, jamais.

La réponse est très simple. Elle est même évidente pour moi : c'est tout bêtement par goût de la logique !

Comment décrire la situation ? A ma gauche, une machine à qui l'on a appris des processus simples qu'elle peut enchaîner très vite pour passer de A à Z. A ma droite, mes cellules grises qui doivent se freiner pour ne pas sauter à pieds joints de A à Z. Car ces cellules grises sont capables de ce je-ne-sais-quoi, de cette intuition qui fait complètement défaut à la machine. Le combat entre ces deux adversaires me contraint à reconstituer tout ce que la machine ne fait pas.

C'est là que l'intuition peut

POURQUOI DIABLE PROGRAMMEZ-VOUS ?

apporter un "plus". En somme, la programmation, c'est pour moi un défi lancé à mes cellules grises.

Bernard LAMBEY

UN JEU DE CONSTRUCTION

ENSEIGNANT la programmation depuis bientôt quinze ans, je programme pour mon métier. Mais en vérité, je programme aussi pour mon plaisir, car programmer est un jeu de construction, une recherche de rigueur, et parfois un défi à la logique, cette logique que l'on retrouve dans la structure des langages.

Ce qui me plaît surtout, c'est de retrouver les structures simples et naturelles qui conviennent le mieux à l'utilisateur installé devant un clavier et un écran. C'est là, me semble-t-il, que se situe l'une des difficultés majeures de la programmation. C'est là que se fait le lien entre l'ordinateur et l'homme.

Max HAGENBURGER

DES PROGRAMMES SUR MESURE

COMME tous les gens curieux, je ne suis jamais satisfait : plus on découvre, et plus on a envie de découvrir. Dès qu'une étude est terminée, il faut avancer, il faut entreprendre la suivante. La petite informatique, et particulièrement la programmation est un domaine inépuisable pour qui veut apprendre, comprendre, initier, créer...

Ce sont les applications techniques et scientifiques qui m'intéressent le plus. Le domaine est immense. On se rend compte très vite que ce l'on étudie est une infime partie du sujet. La modestie est de rigueur...

Ecrire de petits programmes permet d'analyser un problème et d'appréhender ce qui se passe dans tous les cas de figure, car il faut que l'analyse soit complète. Les programmes prêts à l'emploi sont souvent lourds, mal commodes, et ils ne conviennent pas toujours pour certains cas originaux. Programmer soi-même, c'est faire du "sur mesure".

Mais les petits programmes ne sont pas tout. Il y a aussi le projet sérieux et suffisamment passionnant pour vous occuper quelques mois. Dans le domaine qui m'intéresse (la conception assistée par ordinateur), malgré les développements des logiciels, les travaux à entreprendre sont aussi nombreux que variés.

Les amateurs, c'est-à-dire ceux et celles qui ne sont pas programmeurs de profession, joueront, à mon avis, un rôle de plus en plus important dans les années à venir. Leur contribution, déjà, n'est pas négligeable. Et il est tout à fait possible que certains de leurs travaux dépassent ceux qui sont développés pas des professionnels.

Claude NOWAKOWSKI

AVANT TOUT LA CURIOSITÉ

MON premier programme (HP-65, 1976) n'était pas professionnel. Seule la curiosité m'avait poussé à emprunter cette machine à un ami ingénieur. Après 12 heures de souffrance, je réussis à calculer automatiquement le plus grand de deux nombres... La curiosité reste la première motivation des heures que je passe devant mes TRS 80, Apple IIc, Sharp PC-1500, HP-15C ou Casio 702 P. Faire le tour des possibilités de tel matériel, essayer de contrer ses déficiences par des astuces (personnelles, copiées dans des revues, piquées à des amis), comprendre telle obscurité des langages-machine, comparer des systèmes différents...

Viennent ensuite les raisons professionnelles : programmes mathématiques (calculs d'intégrales, résolution d'équations différentielles, tracés de courbes) en rapport avec mon métier de professeur et/ou d'auteur de problèmes, mais aussi programmes de "vulgarisation" en vue de la rédaction d'articles pour cette revue et ses cousines, de livres, etc.

Il m'est arrivé de programmer à la demande pour aider un ami à résoudre un problème (le dernier : calculer l'aire d'un polygone donné par ses sommets, pour un architecte). Cette activité exceptionnelle est parfois très stimulante, comme tout défi.

André WARUSFEL

LE BÉBÉ PLEURE...

Comme il est doux de programmer
Quand autour de moi tout s'agite...
Faut-il m'en plaindre ou m'en blâmer ?
Ai-je des torts ou du mérite ?

Les yeux rivés sur mon écran
Depuis si longtemps que j'en louche,
Je programme un jeu délirant
Et mes doigts vont de touche en touche.

Le bébé pleure, et je poursuis
L'œuvre toujours inachevée,
Sans plus trop savoir où j'en suis.
La vaisselle n'est pas lavée,

Ma tendre épouse n'a rien dit
Ce soir encore. Et je programme...
Et je programme... (en érudit)
Un jeu qui peut plaire à ma dame !

Robin BOIS

ET PUIS UN JOUR...

GRAND ciel bleu, pas l'ombre d'un nuage ; une légère brise d'alizé rafraîchit doucement le mouillage... Et quel mouillage : pratiquement entourée de toute part d'îles basses ébouriffées de cocotiers et cernées de sable étincelant, une piscine naturelle ovale émaillée de tous les bleus de la création, bleu turquoise, bleu d'outremer, bleu roi, bleu profond. De temps en temps une mouette curieuse ou une tortue oisive viennent tourner autour de mon bateau à l'ancre, et se demandent bien pourquoi je ne sors pas profiter du soleil, de l'eau limpide, des plages dorées...

Non, pas pour l'instant. Le soleil, c'est pour plus tard. Pour l'instant, GOSUB, PEEK, POKE et CLOAD. Excusez-moi, à l'heure qu'il est, seul à l'ancre, loin de toute civilisation, j'ai le nez plongé dans mon ordinateur de poche et... je programme !

Si je vous disais que c'est par goût pour la bande dessinée que je suis venu, ou plutôt revenu, à l'informatique ?

1964. Les ordinateurs ont fait du chemin depuis... A l'époque, ils balbutiaient leurs premiers mots de Basic ou de Fortran. Et ce n'est qu'avec l'âge qu'ils ont vu diminuer leur taille : celui de l'Université de Princeton, USA, où j'avais failli faire un stage cette année-là, occupait plusieurs étages d'un grand bâtiment.

Aujourd'hui, grâce aux circuits intégrés, il pourrait sans doute tenir dans le coin d'un bureau. Mais voilà, alors que je terminais mes études d'ingénieur à l'Ecole Centrale (où, bien sûr, n'existait encore aucune spécialisation "informatique"), ma vie avait pris un fameux tournant, et le projet de stage d'un an sur l'ordinateur de Princeton avait été bien vite balayé, remplacé par des tournées, des Olympia, des émissions de télévision et de radio. Bref, j'étais devenu un chanteur populaire, et pendant plus de quinze ans, je ne fus qu'un témoin tout à fait passif de la révolution informatique qui envahissait tous les domaines.

Cette révolution aurait pu m'atteindre, pourtant, par le biais de la musique : d'année en année, les "claviers" faisaient de plus en plus appel à l'électronique ou à la programmation. Si piano et orgue gardaient leur place, on voyait apparaître chaque jour des synthétiseurs recomposant les sons à partir de leurs composantes

élémentaires. Les enregistrements sont devenus numériques, et les musiciens qui "jouent des claviers" débarquent en studio entourés de véritables pyramides d'instruments superposés, où chaque son, imitation d'un son réel, création ou élucubration du musicien, doit être programmé... Michel Magne, grand spécialiste en la matière, auteur de nombreuses musiques de films, m'avouait qu'il prenait peur parfois, quand certains de ses instruments se remettaient à jouer tout seuls la nuit, pataugeant dans l'aléatoire et synthétisant à tous vents !

Moi, je ne sais pas jouer des claviers, ma guitare est restée acoustique, et pour l'informatique, je n'étais qu'une ligne de DATA dans les ordinateurs des "Big Brothers" qui nous immatriculent (quand l'ordinateur n'avait pas, comme celui de la Sécurité Sociale, perdu purement et simplement ma trace, affirmant que je n'existais pas !).

Et puis un jour... c'était, je m'en souviens très bien, dans une librairie de Pointe-à-Pitre, Guadeloupe, où j'avais fait escale en venant du Brésil. Heureux de retrouver des publications en français, j'y achetais une brassée de ces magazines de bandes dessinées, *Fluide Glacial, l'Echo des Savanes, A Suivre,* qui font mes délices, quand mon attention fut attirée par la couverture d'un mensuel que je ne connaissais pas. Elle était signée par Gibrat, un illustrateur dont j'aime le style...

Magazine de "Bédé" ? J'ouvre... pas du tout... qu'est-ce que c'est que ça ? Effaré, je découvre, page après page, des listes de chiffres, de symboles, de phrases absolument incompréhensibles. Piqué au vif, je feuillette et rien, rien, je n'y pige rien ! Çà, c'est trop fort ! Ce magazine, ça a certainement une grande diffusion, pour qu'on le trouve même dans cette petite librairie perdue. Ils sont donc des dizaines de milliers à comprendre tout ça !... Et je suis censé ne pas être plus c... qu'un autre. Alors quoi ?...

Rageur, j'achète le magazine, et pendant trois jours mon équipière a pu me voir lire et relire, de bas en haut, de haut en bas et en diagonale, des pages entières auxquelles je ne comprenais toujours rien !

Moi, vous me connaissez, je n'allais pas m'avouer vaincu... Rentrant passer quelques semaines en France au printemps 83, j'avais pris la ferme décision de me renseigner sur ces langages obscurs... L'ami chez qui j'allais passer quelques jours, le directeur des chantiers *Prometa,* constructeur de mon bateau, venait de s'acheter un PC-1500...

A partir de là, vous devinez le scénario : nuits blanches où l'on s'étonne de voir le ciel s'éclaircir alors que l'on n'a pas encore très bien compris la différence entre une virgule et un point virgule, entre un GOSUB et un GOTO, joie devant le premier programme qui "tourne", colère quand apparaît pour la centième fois "ERROR 1 IN 150"... Et toujours l'envie de penser que c'est la machine qui a quelque chose qui ne va pas... Non, elle ne se trompe jamais, c'est toi qui t'es planté dans l'affectation de tes variables !

Conquis par le PC-1500, j'en ai embarqué un à bord de mon voilier, où je recharge l'accu de l'imprimante à partir du 12 V de mes batteries. Bien sûr, sitôt que j'ai pu à peu près maîtriser le langage, je me suis intéressé aux programmes de navigation de Lucien Strebler publiés au fil des nombreux numéros de l'*Ordinateur de poche.* Je m'en suis inspiré, je les ai adaptés pour le PC-1500 (qui a l'avantage de comporter une horloge-calendrier permanent, bien pratique pour tous les programmes de navigation).

J'ai développé les programmes et je me sers maintenant en navi-

Après l'Ecole Centrale,
la chanson , la navigation, etc.,
Antoine a découvert...
la programmation !

gation de mon "grand" programme de 9000 pas qui me permet de faire le point sur le soleil, la lune, les planètes et 40 étoiles, de faire en trois couleurs les tracés de navigation, de tenir mon estime, de calculer la distance à n'importe quel point du monde...

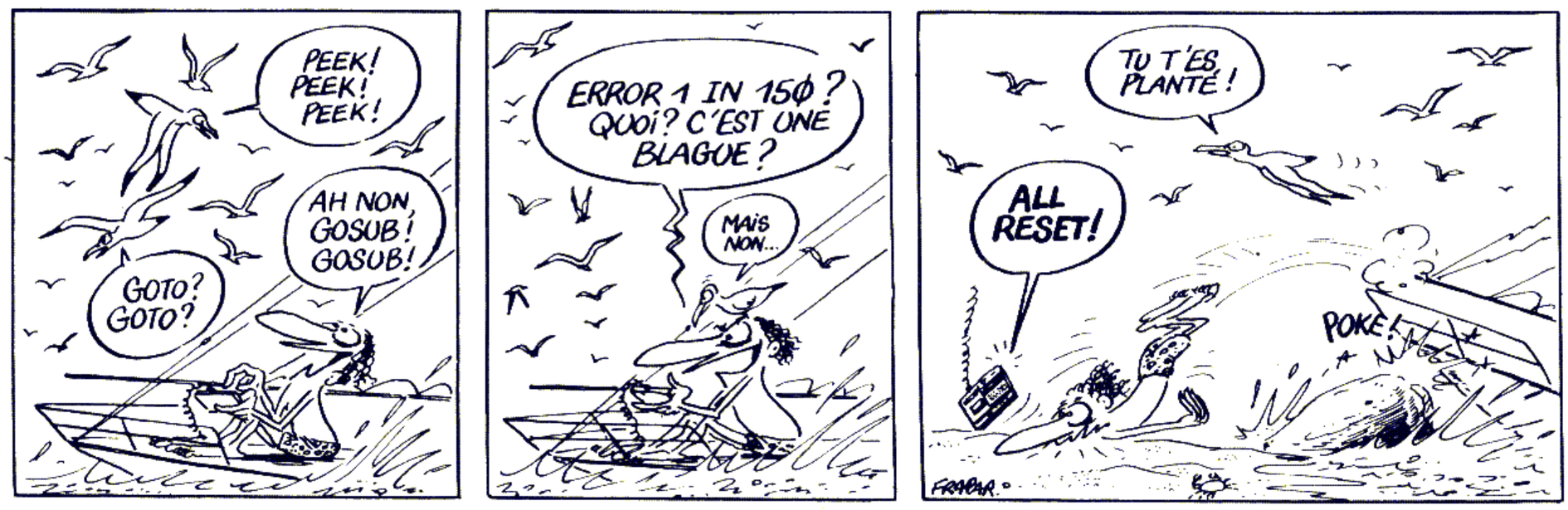
Puis j'ai découvert avec plaisir que le service hydrographique de la Marine venait de publier un livre permettant de calculer sur PC-1500, entre autres, les heures et les hauteurs des marées un peu partout dans le monde. J'ai injecté ce programme à mon PC-1500 et hop là, je peux vous dire à quelle heure je vais pouvoir rentrer dans telle petite baie peu profonde où je suis sûr d'être seul à l'ancre ! Jeux, exercices variés, dessins, je peux faire dessiner à l'imprimante un pavillon anglais, américain (avec les étoiles !) ou japonais (quand il dessine ce dernier, on croit voir perler une petite larme de nostalgie au coin de la touche Shift... nostalgie du pays natal ?).

Plus récemment, je me suis mis au langage-machine, et si je m'y perds encore entre AEX et DRL, SJP et BCS, je passe tout de même de sacrés bons moments... jusqu'à ce qu'un POKE malencontreux m'oblige à un ALL RESET salvateur !

Mes projets ? Ne le dites pas à mon PC-1500, mais j'envisage de le compléter prochainement par son grand-frère, le PC-5000, de Sharp aussi, qui me semble avoir tous les avantages d'un ordinateur de table (grande mémoire, possibilité de stocker de nombreuses données, ce qui me sera utile pour écrire livres et articles grâce au traitement de texte incorporé, et imprimante qualité courrier), sous l'encombrement d'une petite machine à écrire portable, et fonctionnant sur 12 V avec une faible consommation, donc utilisable à bord de mon bateau. J'attends avec impatience qu'il soit disponible avec le clavier Azerty.

Cela dit, je vous quitte pour finir de mettre au point mon programme de "trouveur d'étoiles", et puis j'essaierai quand même d'aller faire un petit tour au soleil. Quel virus, l'informatique : l'autre jour, je me suis surpris à esquisser une subroutine en la traçant du doigt sur le sable de la plage !

ANTOINE



CASIO PB 100

LE BASIC PAS SORCIER

PB 100 : UN ORDINATEUR DE POCHE ET LA METHODE VIVANTE POUR DIALOGUER AVEC LUI. "Apprenez par la Pratique", enfin une méthode simple pour s'initier à la programmation ! Avec des exemples amusants, des exercices faciles et même des jeux... Progressivement, en vous servant de votre ordinateur personnel PB 100 (800 octets), les instructions préprogrammées en Basic, le clavier ASCCI avec 114 caractères différents, le traitement de chaînes de caractères, les boucles, les sauts, les tests, etc. n'ont plus de secret pour vous. Vous avez tellement fait de progrès que vous y ajoutez un module RAM qui porte la capacité de mémoire à 1800 octets, une imprimante et un interface pour stocker vos programmes sur un magnétophone à cassettes. Et puis, vous serez membre du Club Casio qui est là pour vous aider. En vente dans les papeteries et magasins spécialisés. *Distributeur exclusif: Ets Noblet Paris.*

CASIO

CA COMPTE

Brains-Navarre

LA GAZETTE DE LIST

Dictionnaire du Basic
Encyclopédie du langage Basic
David A. Lien
Traduit de l'américain par Yves Leclerc
Éditions du P.S.I.
Lagny, 1983
Broché, 448 pages
Prix : 195 FF

Ecrire ce livre était une gageure. Bien entendu, on ne doit pas s'attendre à y trouver tout-tout-tout sur le Basic et ses différents patois (ce langage évolue trop vite), et pourtant l'ouvrage est excellent. On l'utilisera plus comme un dictionnaire que comme une encyclopédie.

Après une introduction sur la programmation, rapide mais de bon conseil, suivent environ quatre cents fiches décrivant chacune une fonction Basic : sa définition, un court programme-test aidant au diagnostic, une autre routine pour simuler cette fonction sur les ordinateurs qui en sont dépourvus et enfin la liste des instructions Basic à consulter pour mieux utiliser cette fonction. Il est à chaque fois précisé si l'on a affaire à une commande, un opérateur, une déclaration ou une fonction Basic, et si celle-ci possède d'autres noms courants.

Malheureusement, ce point n'est pas complet pour les ordinateurs domestiques européens de type Spectrum, ORIC, DAI (ou même pour le TI 99/4A).

On trouve en fin de livre un index regroupant cinq cents fonctions Basic, mais pas de tableau croisé. Cet ouvrage volumineux permet d'avoir un bon aperçu du Basic tel qu'on le pratique aux USA (le livre est traduit de l'américain) et de disposer d'un outil agréable de traduction de programmes. Mais l'utilisateur d'un seul ordinateur préférera le plus souvent se reporter à la notice de son appareil.

MA □

INDEX DES ANNONCEURS

Ce sont des copains

Dans *Votre Ordinateur*, le premier « Cahier de plaisirs de vacances en informatique »

Ils ne nous parlaient plus. Si nous abordions leur table au restaurant, ils s'enfermaient dans le silence. Et, plus le printemps avançait, plus leur teint affichait la pâleur des nuits de travail !

Mais que nous concoctait donc l'équipe de Votre Ordinateur ? Un numéro d'été bien spécial, une petite bombe : le numéro 6 de *Votre Ordinateur* comportera un cahier de 64 pages supplémentaires intitulé « Cahier de plaisirs de vacances en informatique ».

Ce cahier est destiné aux « cancres » en informatique ; à ceux qui, s'ils n'y connaissent rien, n'en sont pas moins intéressés par le sujet. En vingt leçons, ils devraient acquérir une approche générale du propos.

De plus, en juillet, le soir à 21 h sur France-Inter, la rédaction de *Votre Ordinateur* racontera avec Jacques Pradel (dans le cadre de l'émission *Adrénaline*) « Silicium blues », à savoir la grande et palpitante histoire de l'informatique !

Ouvrez bien vos oreilles : un concours doté chaque semaine d'un bon de 4 000 FF pour acheter l'ordinateur de votre choix dans la boutique de votre goût sera proposé aux auditeurs-lecteurs d'*Adrénaline - Votre Ordinateur*. □

Une interface-cassette pour le PB-700

Casio propose une autre interface-cassette pour son PB-700 : la FA-4. Elle n'est pas dotée de l'imprimante de la FA-3, mais conserve des dimensions relativement imposantes.

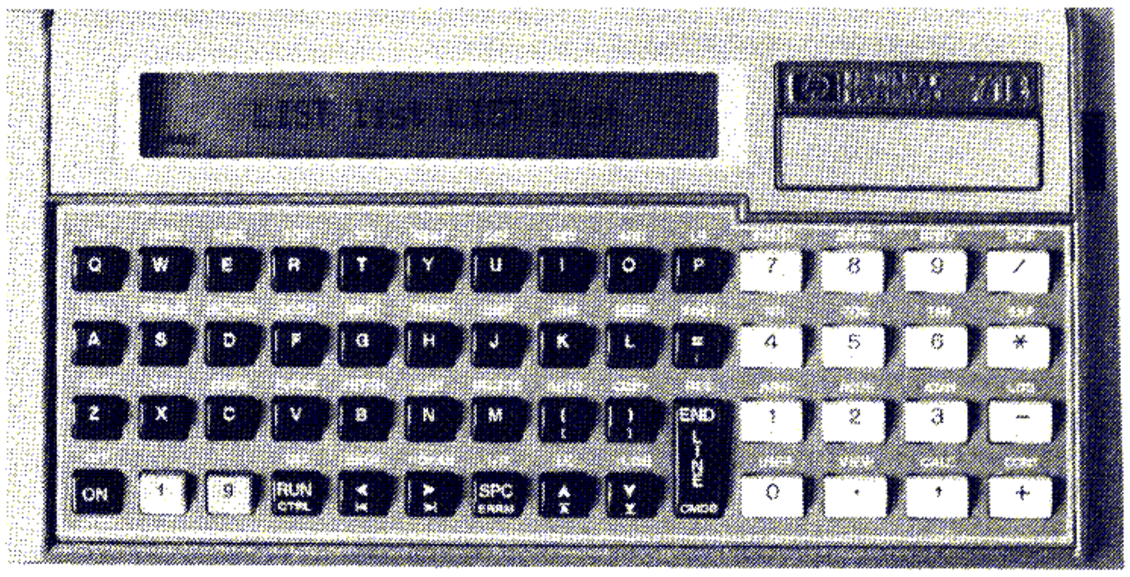
En contrepartie, le prix a largement diminué (environ 750 FF), et le dos de la FA-4 comporte une sortie parallèle pour imprimante... Rappelons que l'interface-cassette permet également de relier deux PB-700 pour un transfert de données.

MA □

Faites-vous connaître...

Si vous faites partie d'un club d'informatique, sans but lucratif, où l'on apprend et pratique la programmation et si vous recherchez de nouveaux adhérents, signalez-nous votre existence. En lisant votre adresse dans ces colonnes, beaucoup de nos lecteurs seront contents d'apprendre que votre club n'est pas trop loin de chez eux. □

HP-71 B : le nouveau poquette de Hewlett-Packard



Du concentré d'ordinateur... Jugez-en vous-même : 17,5 Ko de mémoire vive, un Basic étendu (64 Ko de mémoire morte, et un mode CALCul, le tout dans un volume très modeste (19 × 8,7 × 1,2 cm).

En regard, le prix : environ 5 600 FF ttc. Le HP-71 B n'est donc pas un poquette destiné aux lycéens. Le clavier (QWERTY) est une réussite : touches noires et blanches, caractères en bleu, ocre, noir et blanc. Les différentes fonctions ont été judicieusement regroupées par zones utiles (statistiques, fichier, programme, etc.). Avec un peu d'entraînement, on s'y retrouve bien.

A l'affichage, 22 caractères (8 × 132 points adressables par colonnes seulement) sur une « ligne » virtuelle de 96 caractères.

Le Basic du HP-71 B est particulièrement performant pour un ordinateur de poche ; il rendrait bien des points à certains ordinateurs de table. A titre d'exemple, notons qu'il est possible de redéfinir tout le clavier sauf deux touches et que les calculs se font sur quinze chiffres avec douze affichés, sauf si l'on opte pour la demi-précision ou pour les variables entières !

La notion de variables locales donne accès à la récursivité et l'utilisateur peut créer sa propre police de 128 caractères. Les GOSUB et autres FOR/NEXT ne sont limités que par la mémoire vive, et les fonctions PEEK$ et POKE attendent les programmeurs en langage-machine (le manuel, toutefois, n'est pas encore commercialisé). Le traitement des erreurs est très élaboré, y compris celui des erreurs mathématiques : cela peut aller jusqu'à identifier le signe de zéro ou jongler avec l'infini !

La présence de 128 indicateurs binaires (une moitié pour le système, l'autre à la discrétion de l'utilisateur) permet d'autres acrobaties. Une fonction BEEP, paramétrable en fréquence et durée avec deux niveaux d'intensité, sonorise l'ensemble. Une petite indication sur la vitesse d'exécution : le HP-71 B travaille pendant 9 secondes et 27 centièmes pour venir à bout d'une boucle FOR 1 = 1 TO 1000 @ NEXT 1. Ce n'est donc pas un bolide.

Les fonctions statistiques traitent jusqu'à 15 variables d'un coup (2 sur la HP-41). Une horloge, un calendrier et quelques fonctions subsidiaires permettent, entre autres applications, de faire démarrer des programmes à l'heure dite. Parmi les gadgets appréciés : STARTUP exécute des instructions dès l'allumage, LOCK contrôle l'utilisation du poquette avec un mot de passe, et CONTRAST règle l'afficheur par programme... Cela étant, on regrettera, pour certaines applications, que les tableaux soient limités à deux dimensions et que leurs noms, ainsi que ceux des variables aient au plus deux caractères. Dommage...

La mémoire vive (17,5 Ko dont 1 environ est utilisé par l'ordinateur) se répartit en deux zones. La première, la principale, peut être perdue au cours d'un MEMORY LOST ou INIT/3. La seconde est indépendante : on la crée par blocs de 4 096 octets, et elle est protégée contre les accidents de ce type. Les programmes qui y sont stockés sont directement exécutés par simple appel de leur nom. Quant aux fichiers de données, ils sont de trois types : DATA, TEXT (échange avec d'autres ordinateurs de table HP) et SDATA, spécialement prévu pour les échanges avec la HP-41.

Si les quelques 240 instructions de ce Basic ne suffisent pas, il sera par la suite possible d'en ajouter grâce aux modules de mémoire morte BIN et LEX (extension de langage), contenant des programmes en langage-machine. Quatre petits logements situés sur la face avant du poquette permettent en effet de brancher des modules de mémoire vive ou morte. A l'arrière de la machine, on trouve le connecteur prévu pour la boucle HP-IL et, sur la platine, une petite trappe a été ménagée pour recevoir un lecteur de cartes magnétiques.

Parmi les facilités qui donnent un petit « plus » au HP-71, retenons la pile d'instructions, contenant les cinq dernières « fonctions » demandées par l'utilisateur. Ces « fonctions » sont, selon les cas, la dernière ligne de programme entrée, le dernier calcul demandé ou même une série complète de calculs. Ainsi, une expression telle que PI + 4 * COS (3 + 3 * FACT(X) − 16 * LOG(312) n'occupe qu'un seul niveau de la pile. Pour remplacer FACT(X) par FACT(H), un petit coup sur Λ puis sur > pour placer le curseur sur X, on tape H, puis END LINE et l'expression est entièrement réévaluée ! Et ce n'est qu'un exemple de ce que l'on obtient avec le mode CALC et la pile d'instructions.

L'alimentation électrique est assurée par quatre petites piles de 1,5 V, mais la connexion sur le secteur est possible (en option).

Les périphériques disponibles, mis à part le lecteur de cartes dont le logement est prévu, sont très nombreux puisqu'ils comprennent d'emblée la gamme HP-IL (imprimantes, interfaces vidéo, lecteur de cassette, etc.). Un module Forth/Assembleur est annoncé, qui devrait considérablement augmenter la puissance, déjà non négligeable, du HP-71 B.

La documentation fournie impressionne par son volume : plus de 760 pages, partagées en un manuel d'utilisation (300 pages), un manuel de références (400 pages) et un aide-mémoire (60 pages) qui se glisse dans la poche prévue à cet effet à l'intérieur de la housse.

Comme il est indiqué dans le manuel d'utilisation, le HP-71 ne s'adresse pas aux novices en informatique : c'est un engin plutôt professionnel. **OA** □

■ UN LIVRE

Programmer sur calculatrices et ordinateurs de poche
Michel Appert et Bertrand-René de Fraguier
Editions Dunod
Paris, 1983
Broché, 102 pages
Prix : 57 FF

AOS, RPN ou BASIC

Le pratiguide est un ouvrage d'initiation à la programmation des ordinateurs de poche. Parallèlement, des applications présentent une comparaison complète des trois langages-types : AOS sur TI-57, RPN sur HP-15C, Basic sur PC-1211/1212.

Le livre a donc une vocation double : la première partie développe le principe de fonctionnement des calculatrices et des ordinateurs en général, puis expose les bases de la programmation ; la suite de l'ouvrage

est consacrée à l'examen de douze programmes et de leur fonctionnement sur chacune des machines. Bien sûr, il n'est pas nécessaire de les posséder toutes les trois, au contraire : la présentation comparative peut souvent aider le lecteur à déterminer les capacités de l'ordinateur de poche dont il a besoin.

Le mérite des auteurs réside d'abord dans la clarté et la concision du premier chapitre : celui-ci a pour objet de définir par le menu ce que peut faire un ordinateur, quel qu'il soit, lorsqu'il effectue une opération donnée.

Les différents langages, types de mémoires et périphériques sont démythifiés en quelques pages qui comptent pour beaucoup par la suite lorsqu'il s'agit d'écrire "l'algorithme" : on copie le fonctionnement de la machine.

Ce passage important à la rédaction du programme est détaillé au chapitre suivant, ainsi que l'ensemble des instructions des ordinateurs de poche. A part les différences de notations, beaucoup d'instructions sont communes aux trois, comme le montrent les exemples choisis. Quelle notation faut-il préférer ? Restons impartial ! Chacune a ses adeptes, celle des auteurs est peut-être RPN, mais les machines sont toutes examinées avec le même soin, fort heureusement ; l'idéal est de les bien connaître, toutes...

Si les exemples qui viennent ensuite semblent peu originaux, la manière dont ils sont traités rehausse leur valeur. Six jeux — très classiques — permettent de saisir les ficelles de la programmation, tandis que cinq programmes mathématiques utilisent ces ficelles avec brio : plus que le programme, c'est la méthode qui prime. Les programmes de calcul intégral, de statistiques, de décomposition en facteurs premiers sont très optimisés et donnent d'excellents résultats, à la différence des programmes habituels : les méthodes simples, logiques, sont meilleures...

Derrière ce livre présenté intelligemment, on aurait aimé trouver une liste d'exercices pour mieux maîtriser certains aspects de la programmation : manipulation des mémoires, etc. Mais ce n'est pas le but de cet ouvrage, limité aux petites machines. Nous reprendrons la conclusion des auteurs : "Travaillez, prenez de la peine", car c'est bien la meilleure école. La programmation est-elle autre chose qu'une réflexion intellectuelle qu'il faut maîtriser ?...

MA □

Votre premier programme Basic
Rodnay Zaks
Editions Sybex
Paris, 1984
Broché, 208 pages
Prix : 98 FF

Les ouvrages d'initiation au Basic sont légion. Mais il en est peu qui savent faire un judicieux dosage des explications. Certains sont simples mais restent enfantins. D'autres sont plus complets, mais sont sévères et peu accessibles. *Votre premier programme Basic* a su trouver un bon équilibre. Il est vraiment destiné aux débutants en informatique. C'est un excellent bouquin à acheter si la notice de votre ordinateur se contente de décrire des instructions, sans vraiment expliquer la logique du langage Basic et les principes de la programmation.

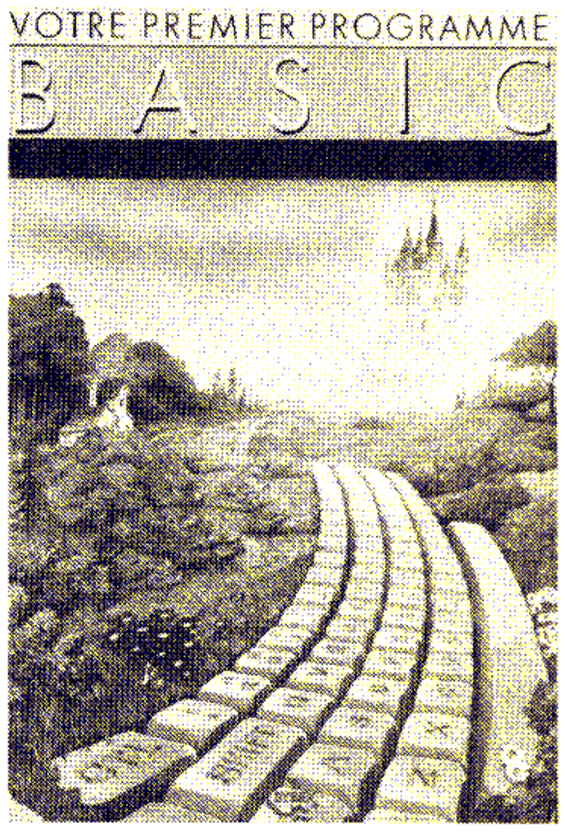
La progression, de chapitre en chapitre, est bien dosée. Cela commence avec une description de l'ordinateur et du langage Basic. Puis on attaque les calculs, les variables, l'écriture d'un programme, les tests et les répétitions. Un chapitre est réservé à la conception des programmes : algorithme, organigramme, etc.

On peut regretter que le livre n'aborde pas certaines instructions qui désarment souvent le débutant, comme le traitement des chaînes de caractères, le couple READ-DATA ou l'exploitation des tableaux. Mais il est destiné à l'initiation d'un Basic minimum et il guide bien le novice dans ses premiers pas.

La présentation en est agréable, aérée et mise en valeur par l'utilisation de deux couleurs : noir pour le texte, rouge pour les titres et les représentations d'écran. Des illustrations récréatives renforcent le texte.

Enfin, la couverture du livre est une incontestable réussite. Elle a d'ailleurs été primée, comme la plus belle couverture dans la catégorie enseignement, au dernier Salon du Livre.

XdLT □



Du nouveau pour le Spectrum

L'interface ZX1 et les lecteurs de microcartouches

Longtemps annoncés, longtemps attendus par les « Sinclairistes », les « microdrives » sont enfin disponibles en France.

Lors de la sortie du Spectrum, une mémoire de masse était annoncée dans le manuel. Cette mémoire de masse était intitulée « microdisque ». De disque il n'y a pas, car il s'agit d'une bande magnétique sans fin dans un petit étui plastique de 4,5 cm sur 3,5 cm pour une hauteur de 7 mm. Mais avant tout, il est nécessaire de parler de l'interface ZX1.

Cette interface est enfichée sur le connecteur du Spectrum et repose sous celui-ci. Le clavier se trouve alors légèrement incliné, offrant une position plus ergonomique. Trois nouvelles fonctions étendent alors les possibilités de l'ordinateur.

La première est celle qui était attendue : pouvoir disposer d'une mémoire de masse d'accès rapide grâce aux cartouches de bandes sans fin.

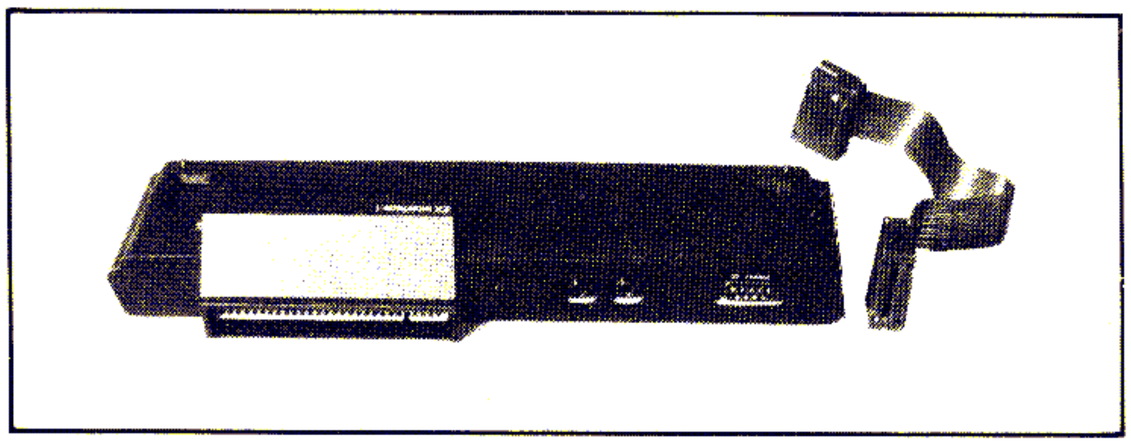
La seconde est une interface RS232 ouvrant le Spectrum sur un environnement informatique non spécifiquement Sinclair. Cette interface-série permet la connexion à des imprimantes, des modems, d'autres ordinateurs.

Les entrées et sorties réalisées par cette voie sont facilement mises en œuvre par des instructions du Basic étendu, apportées par l'interface ZX1. Deux types de canaux peuvent être ouverts sur l'interface RS232, le canal « t » qui gère des caractères définis sur 7 bits, et le canal « b » qui prend en compte le huitième bit des caractères. Sur ces deux canaux, par un simple POKE sur une variable système, on définit le débit de transmission, débit standard compris entre 50 et 19 200 bauds, ou débit non standard.

La troisième fonction permet d'établir un réseau entre 2 à 64 Spectrum à l'aide de simples câbles jack-jack. Il est probable que l'on pourra aussi connecter, grâce à ce réseau, le futur produit Sinclair, le Quantum Leap ou QL (toujours pas disponible en France). De même qu'avec l'interface RS232, les transmissions sont aisément établies par l'intermédiaire d'instructions du Basic étendu.

Le réseau permet ainsi de faire communiquer plusieurs ordinateurs, de faire profiter l'ensemble des interlocuteurs d'une imprimante, ou de programmes stockés sur la cartouche de tel ou tel Spectrum. Une fonction très intéressante donc, pouvant être mise à profit, par exemple, dans une classe

Vue arrière sur l'interface ZX1 du Spectrum



d'école, comme il est montré dans la notice d'emploi. Chaque ordinateur doit alors être muni d'une interface ZX1.

Venons-en enfin aux lecteurs de cartouches ! Un minimum de 85 Ko stockés sur un mince ruban magnétique. Un temps moyen d'accès à un fichier, programme ou bloc d'octets, de 3,5 secondes. Mais c'est un temps moyen ! On charge 48 Ko en 9 secondes !

Les commandes de sauvegarde, chargement, vérification et fusion de l'interface-cassette sont aussi disponibles.

Outre ces commandes, on peut ouvrir des fichiers pour y stocker des données de façon séquentielle. Les données alphanumériques et numériques peuvent être mixées dans le fichier. Une fois les données enregistrées, le fichier doit être fermé. Mais il ne peut plus, alors, être ouvert qu'en lecture, ce qui est un sérieux inconvénient. En effet pour modifier une donnée, il faut ouvrir un nouveau fichier et reprendre une par une les données pour les y recopier toutes. Alors seulement, on peut effacer l'ancien. Cette limitation entraîne une petite gymnastique de programmation dont on se serait bien dispensé...

Tout comme avec un lecteur de vraies disquettes, on peut obtenir un catalogue des noms de fichiers, de programmes, de blocs d'octets... Une procédure intéressante a été prévue. A la suite d'une réinitialisation du Spectrum, à la mise sous tension, ou après un NEW, si l'un des programmes stockés sur cartouche porte le nom « run », on obtient son chargement en tapant simplement la commande RUN du Basic, validée par ENTER. On peut ainsi, sur chaque cartouche, mettre un programme « run » qui affichera le menu de la cartouche et chargera le programme choisi.

Physiquement, le lecteur de cartouches se présente sous la forme d'une boîte (9 cm sur 8,5 cm et 4 cm de haut). Sur sa face avant se trouvent l'ouverture, où l'on insère la cartouche, et une petite diode rouge qui s'allume lors de la mise en route du moteur. Le lecteur est relié par un court fil en nappe. Il est aussi possible de le solidariser avec sept autres lecteurs en série.

Au niveau de la fiabilité, il est encore un peu tôt pour se prononcer. Néanmoins, les cartouches semblent relativement fragiles, et elles ne sont pas données (75 FF). Un signe de vieillissement est l'augmentation du temps de chargement d'un programme. Le programme est lu et relu tant que les sommes de contrôle ne sont pas correctes.

Sinclair a modifié son interface ZX1 plusieurs fois ; celle que j'ai essayée est une quatrième version. Il est probable qu'elle comporte encore quelques petites bogues. J'ai dû parfois couper l'alimentation alors qu'une cartouche était dans le lecteur, ce qui est fortement déconseillé. Une nouvelle version, revue et corrigée, est sans doute en cours.

Quoi qu'il en soit, l'agrément apporté par cette mémoire de masse prévaut largement sur ces petits ennuis épisodiques. Et l'on doit tenir compte aussi du prix, nettement moins élevé que celui d'un lecteur de disquette conventionnel. L'interface ZX1 vaut 895 FF, le lecteur de cartouche 940 FF, et le câble RS232 235 FF (ttc). Distributeur : Direco International.

BT □

Langage-machine et PC-1250/1251

Le manuel de référence du langage-machine pour PC-1250/1251, édité par **Sharp**, est arrivé. Cet ouvrage de cent pages, au format 21 × 29,7, est rédigé en anglais.

On y trouvera bien sûr la description des 115 instructions du langage-machine (près de soixante pages leur sont consacrées). Mais on trouvera aussi une foule d'autres renseignements sur les PC de la série 12, le processeur qui les équipe, la carte de la mémoire, la représentation des variables, etc.

Cet ouvrage, agréable à consulter, est un manuel de référence ; il ne suffira donc pas pour une première initiation au langage-machine.

On devrait pouvoir se le procurer (comme le manuel correspondant pour PC-1500) auprès du club des *Sharpentiers* pour le prix de 150 FF, port non compris.

Pour tout renseignement, écrire au club :
SBM (Sharpentier)
151-153, avenue Jean Jaurès
93307 Aubervilliers Cedex □

Un petit tour chez le libraire

Tout savoir sur Laser 200-210
Bertrand Ravel
Editions Eyrolles
Paris, 1984
Broché, 100 pages
Prix : 80 FF

Programmer en Forth
Alain Pinaud
Editions du P.S.I.
Lagny, 1983
Broché, 160 pages
Prix : 82 FF

Applications du 6502
Rodnay Zaks
Editions Sybex
Paris, 1984
Broché, 288 pages
Prix : 105 FF

Pratique du micro-ordinateur Alice
Henri Lilen
Editions Radio
Paris, 1984
Broché, 158 pages
Prix : 100 FF

Dictionnaire Marabout de la micro-informatique
Ilya Virgatchik
Editions Marabout
Paris, 1984
Broché, 246 pages
Format poche
Prix : 24 FF

Pour mieux utiliser le Commodore 64 Méthodes pratiques
Jacques Boisgontier
Editions du P.S.I.
Lagny, 1984
Broché, 160 pages
Prix : 100 FF

La programmation synthétique de la HP-41
W.C. Wickes
Traduit de l'américain
par Luc Mathieu
Editions du Cagire
Toulouse, 1984
Broché, 94 pages
Prix : 100 FF □

JEUX ET CASSE-TETE INFORMATIQUES

par Thierry CHAMORET

LES jeux et casse-tête qui vous sont proposés dans cette rubrique ont plusieurs aspects. Tout d'abord, ils peuvent être pris sous l'angle ludique, c'est-à-dire qu'il s'agit de jeux, de petits problèmes plus ou moins faciles à résoudre. Ils ont également un aspect pratique. Ils permettent en effet à chacun d'exercer son agilité logique. Et il n'est pas nécessaire, pour trouver la solution, d'avoir un ordinateur sous la main...
Les réponses paraîtront dans le prochain LIST.

1

Remontons un peu dans le temps...

Vous disposez (simple supposition) d'un ordinateur possédant les cinq opérations entières suivantes :

+	addition
–	soustraction
*	multiplication
div	division entière (seule la partie entière est conservée, c'est-à-dire que 10 div 3 = 3)
mod	modulo (reste de la division entière, 10 mod 3 = 1).

Une variable, appelée *Mois,* mémorise le numéro d'un mois du calendrier, c'est-à-dire une valeur entre 1 et 12, et une autre variable, *Annee,* le numéro de l'année (avec 2 ou 4 chiffres). Nous cherchons à revenir en arrière d'un mois, en calculant la nouvelle valeur prise par *Mois* et éventuellement par *Annee.* Cela peut s'effectuer de la façon suivante :

```
SI    Mois = 1 ALORS
      Mois = 12
      Annee = Annee – 1
SI    NON
      Mois = Mois – 1
FIN   SI
```

Saurez-vous, sans utiliser de test de comparaison (<, >, =, <>), remplacer les instructions ci-dessus par deux expressions arithmétiques fournissant les nouveaux numéros du mois et de l'année ?

2

Pas un bit de plus

Vous savez qu'avec un bit, il est possible de stocker deux états différents, représentés par les chiffres 0 et 1. Avec deux bits, quatre valeurs différentes peuvent être mémorisées. Elles sont représentées par les motifs binaires suivants :

Numéro de la valeur	Motif binaire
0	0 0
1	0 1
2	1 0
3	1 1

Plus généralement, il existe une formule indiquant le nombre N de valeurs qui peuvent être représentées en fonction du nombre B de bits disponibles. Cette formule est la suivante :
$N = 2^B$ avec $B>0$.

Ainsi, avec B égal à 2, nous trouvons 4, qui est égal à la valeur indiquée précédemment.

La question que nous soumettons à votre réflexion est, à l'inverse : pour mémoriser N états différents, quel est le nombre minimum B de bits nécessaires ? D'une façon analogue, la valeur de B peut être exprimée avec une formule qui est fonction de N. La réponse à cette question nécessite quelques connaissances en mathématiques (fonction logarithme).

3

I et J, mais pas K

Vous connaissez l'algorithme classique, très utilisé dans les tris, permettant d'échanger le contenu de deux variables I et J. On utilise une variable auxiliaire K, pour stocker provisoirement le contenu de I. L'extrait de programme suivant effectue l'échange des valeurs de I et de J.

```
K = I    I = J    J = K
```

Le langage-machine de certains ordinateurs comporte une instruction qui effectue cette opération ; elle est appelée généralement SWAP (permuter). Mais le processeur utilise en fait un registre interne qui joue le rôle de la variable K. Arriverez-vous à trouver un algorithme effectuant cet échange de valeurs, et cela sans utiliser de variable auxiliaire ?

La solution est simple... quand on la connaît. Pour vous aider, sachez toutefois qu'il n'y a aucun trucage, que trois lignes sont nécessaires, comme ci-dessus, et que l'addition et la soustraction sont utilisées. ■

PROGRAMMATION STRUCTURÉE :

UNE QUESTION DE MÉTHODE

COMMENT programmer ? Quelle méthode utiliser ? Le propos de cet article, et de ceux qui suivront dans les prochains LIST, est de vous aider à trouver VOTRE réponse personnelle à cette question. En effet, peu importe la méthode, pourvu que vous en suiviez une. Seront donc exposées plusieurs méthodes qui, toutes, présentent des avantages et des inconvénients. A vous de prendre dans l'une ou l'autre ce qui vous convient ou ce qui correspond à votre problème.

Ce premier article a trait à une méthode dérivée de la programmation structurée, méthode qui pourrait être appliquée, par exemple, à la résolution du problème suivant : « apprendre le vocabulaire anglais en classe de 6e », mais ce sera pour une prochaine fois.

50 % de REMs ? Au fou !

Quel que soit l'ordinateur que vous utilisez (qu'il s'agisse d'un Apple, d'un ZX Spectrum, d'un Alice, etc.), quel que soit le langage que vous utilisez (Basic, Cobol, Assembleur, etc.) et quelle que soit la méthode que vous utilisez (Warnier, Coriq, etc.), l'acte de programmer va se traduire ainsi : vous vous assierez devant votre clavier et votre écran et vous entrerez alors effectivement le code de votre programme. Cette phase, vous le savez, est la plus passionnante et la plus fascinante de la programmation ; mais pour être menée à bien d'une part, et rapidement d'autre part, quelques précautions s'imposent.

Oubliez un instant les limites de votre ordinateur et tous vos soucis d'optimisation, et essayez d'appliquer les quelques conseils suivants.

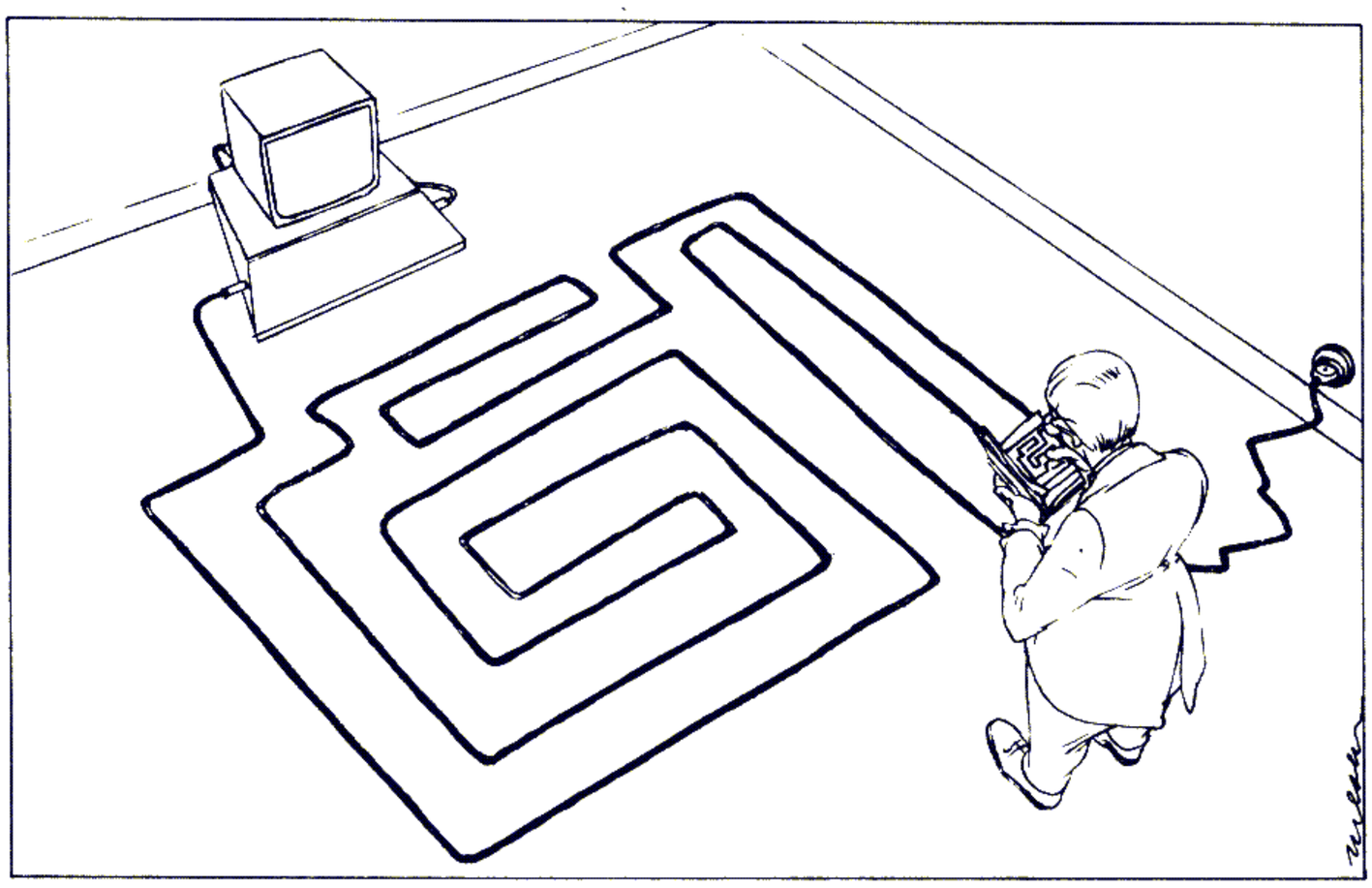
Un programme est toujours constitué d'ordres et de variables. Plus rarement de commentaires. L'idéal serait que votre programme, quel que soit sa taille, comporte au minimum 50 % de commentaires. « A fou !, me direz-vous, et ma place mémoire disponible ? Et mon temps d'exécution ? » Je reviendrai plus tard sur les outils à utiliser pour résoudre ces deux problèmes mais, ce qui importe, c'est que *la documentation du programme soit incluse dans le corps même du programme.* Il est alors inutile de rechercher une hypothétique documentation externe au programme et classée dans on ne sait plus quel tiroir ou quel dossier. Un seul et même support, qu'il s'agisse d'une cassette ou d'une disquette, renferme programme et documentation.

Allons un peu plus loin. Quand vous utiliserez ce programme, comment ferez-vous ? Là aussi, il ne faut pas rechercher ailleurs la réponse et *tout programme doit comprendre son propre mode d'emploi et son guide d'utilisation.* La réponse à la question « comment ça marche ? » doit, au sens propre, vous sauter aux yeux. Tout écran affiché doit comprendre son propre mode d'emploi et vos actions possibles.

Cette phase de commentaires, qu'ils soient internes (documentation) ou externes (mode d'emploi) est certes fastidieuse, mais le gain de temps qui en résulte et surtout le confort d'utilisation sont irremplaçables. Un programme « non confortable » ne sera pas utilisé.

Revenons à un niveau plus simple, celui des ordres et des variables et convenons de n'utiliser qu'*un seul ordre par ligne* et *une seule fonction par variable.*

N'utiliser qu'un seul ordre par ligne peut être gênant si l'on manque de



place, mais cela facilite le repérage des branchements (dans tous les cas), la recherche des erreurs (la ligne fautive ne présente plus d'ambiguïtés) et, bien sûr, la lisibilité. Ceci permet également de réaliser une liste de programme qui présente les indentations significatives du niveau d'imbrication de la séquence d'instructions (boucles contrôlées par la machine FOR...NEXT ou par vous-même IF...THEN...GOTO).

La priorité : être lisible

De la même manière, et pour les mêmes raisons (repérage, recherche des erreurs et lisibilité), il ne faut utiliser une variable que pour une seule fonction. Vous êtes alors à même de penser à lui donner une valeur initiale, à la faire évoluer et à contrôler sa valeur finale. Ceci vaut même (surtout ?) pour les simples variables de contrôle de boucles. Leurs noms doivent être également significatifs de leurs fonctions (ceci dans les limites de l'interpréteur ou du compilateur dont vous disposez).

Revenons un instant sur les boucles contrôlées précédentes. La quasi-totalité des langages comporte un ordre de boucle (FOR...NEXT en Basic) et la *totalité* des langages comportent les ordres de test (IF...THEN du Basic) et de débranchement inconditionnel (GOTO en Basic). *Programmez vous-même vos ordres de boucle.*

La séquence :

```
100 FOR I=a TO b STEP c
200 NEXT I
300 REM suite
```

peut parfaitement être remplacée par :

```
 90 I=a-c
100 I=I+c
110 IF I > b THEN 300
200 GOTO 100
300 REM suite
```

Compte non tenu de la vitesse d'exécution (la deuxième séquence est plus rapide) et de la disponibilité (tous les langages ne comportent pas l'équivalent de FOR...NEXT et, a fortiori, STEP), *utiliser un vocabulaire minimum* facilite la portabilité des programmes, leur mise en œuvre (le vocabulaire reste simple) et la relecture (inutile de se poser de longues questions sur ce que peut faire telle option de tel langage sur telle machine alors que, justement, vous ne vous souvenez plus de l'endroit où se trouve le manuel d'utilisation...).

Tous les conseils qui précèdent ne relèvent pas d'un souci d'optimisation mais de *lisibilité avant toute autre chose*. L'optimisation est forcément dépendante de l'ordinateur sur lequel vous travaillez et vous savez fort bien la vitesse à laquelle évoluent les capacités des ordinateurs. Il n'apparaît, à l'évidence, pas opportun de se lier à tel ou tel état de la technique. Si votre programme ne va pas assez vite, il existe des compilateurs qui vous procureront un gain de temps sans commune mesure avec ce que vous auriez pu grapiller en réorganisant vos sous-programmes ou en jouant de telle ou telle astuce.

« D'accord pour la vitesse, me direz-vous, mais que faire par rapport à la taille mémoire qui m'est comptée ? ». C'est là où l'organisation de votre programme va jouer. Premier principe : votre programme doit n'être qu'une suite de sous-programmes, chacun d'eux ayant une fonction bien définie. *A une fonction doit correspondre un sous-programme.* Considérez chacun d'entre eux comme une boîte noire : vous savez ce que vous en attendez (en sortie) et vous savez ce que vous y mettez (en entrée). *Vous contrôlerez* chacune de ces boîtes noires non en fonction de ce qu'elles contiennent, mais *en fonction de ce qui en sort*. Que vous importe que deux calculs successifs à l'intérieur d'une boîte soient faux, pourvu que le résultat final soit juste.

Chacun de ces sous-programmes doit avoir une taille moyenne de 40 lignes *(deux écrans au maximum)*. Au-delà, on ne peut pas l'examiner sans utiliser un support externe — liste manuelle ou sur imprimante — dont nous avons évité l'emploi jusqu'à présent.

Dans le cas de sous-programmes généraux — que vous réutilisez d'un programme à l'autre (pour contrôler une date par exemple, ou obtenir le quantième d'un jour donné) — essayez de leur affecter toujours les mêmes numéros de ligne. Même si vous ne disposez pas d'un éditeur puissant capable de réintégrer ces petites boîtes successives dans un nouveau programme, vous prendrez l'habitude de toujours appeler telle fonction à telle adresse. La programmation s'apparente alors au Meccano et *votre ensemble de sous-programmes est votre boîte à outils.*

Il ne nous reste plus qu'à procéder à l'assemblage et, pour ce faire, nous allons utiliser une méthode dérivée de la programmation structurée.

Sauf si vous êtes un inconditionnel du mouvement perpétuel, vous conviendrez que le plus important pour un programme est qu'il s'arrête un jour. Nous allons donc nous en assurer en priorité.

```
 10 GOSUB 100
 20 IF A=1 THEN 50
 30 GOSUB 200
 40 GOTO 20
 50 GOSUB 300
 60 END
100 REM début
110 A=0
190 RETURN
200 REM milieu
210 A=1
290 RETURN
300 REM fin
390 RETURN
```

Bien agencer les boîtes noires

La séquence précédente ne fait strictement rien si ce n'est s'arrêter chaque fois après une exécution correcte. Ce qui était le but recherché. Elle appelle quelques commentaires :

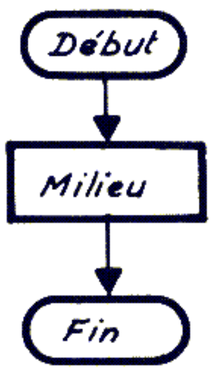
- on peut l'utiliser quel que soit le problème que l'on traite et, en ce qui me concerne, *tous les programmes* que j'écris comportent ces ordres (à l'exception de la ligne 210 que l'on verra plus loin) ;
- le corps principal du programme se réduit aux six premières lignes dont le seul rôle est de contrôler l'arrêt du traitement. *Tout le reste est une suite de sous-programmes ;*
- à un niveau donné, on retrouve toujours un sous-programme de *fin de traitement* (ici 300-390) et, donc, un sous-programme de *début de traitement* (ici 100-190) qui correspondent respectivement à la sortie et à l'entrée de la boîte noire de *traitement proprement dit* (ici 200-290) ;
- certaines de ces boîtes noires peuvent être vides (ici la fin de traitement) ; en tout état de cause, elles sont toujours présentes ;
- ce qui vaut pour le traitement (les boîtes noires de fin, début et traitement) vaut *pour les variables.* La seule variable présente ici, A, comporte un *contrôle de fin* (ligne 20 IF A=1 THEN 50), donc elle comporte une *valeur de début* (ligne 110 A=0) et un *traitement* entre les deux (ici ligne 210 A=1).

On essaiera donc de décomposer tout problème en pensant à ce que doit effectuer le programme (que veut-on obtenir ?), à son début (quelles données puis-je fournir ?) et à son milieu (comment passe-t-on de l'un à l'autre ?).

Deux remarques : la fin est évidem-

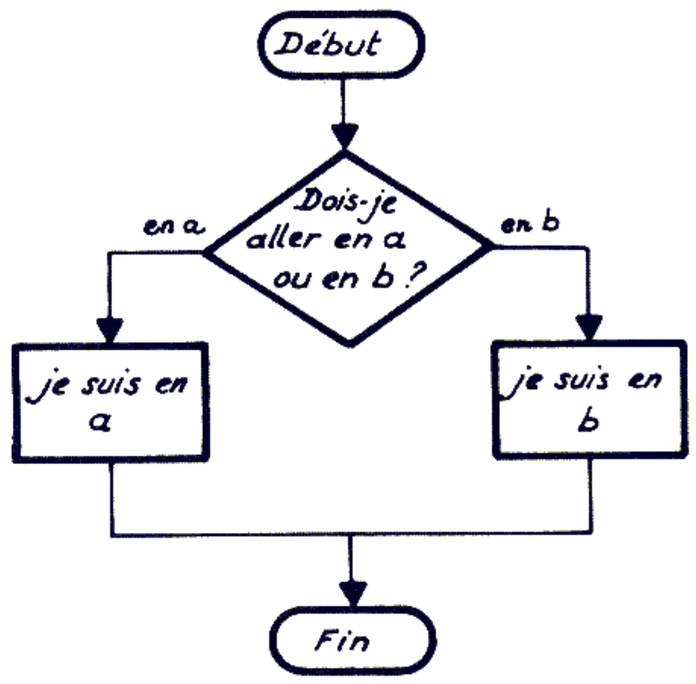
ment la partie la plus importante. Si l'on sait ce que l'on veut, on a une chance de l'obtenir ; la réciproque n'est pas vraie. Déterminer ce point avec précision est toujours le plus délicat.

La deuxième remarque concerne le milieu : si l'on s'aperçoit qu'il va dépasser la limite fixée précédemment (deux écrans), on le décompose à son tour en sa fin, son début et son milieu. Et ainsi de suite. On peut donc toujours représenter un problème sous la forme suivante :

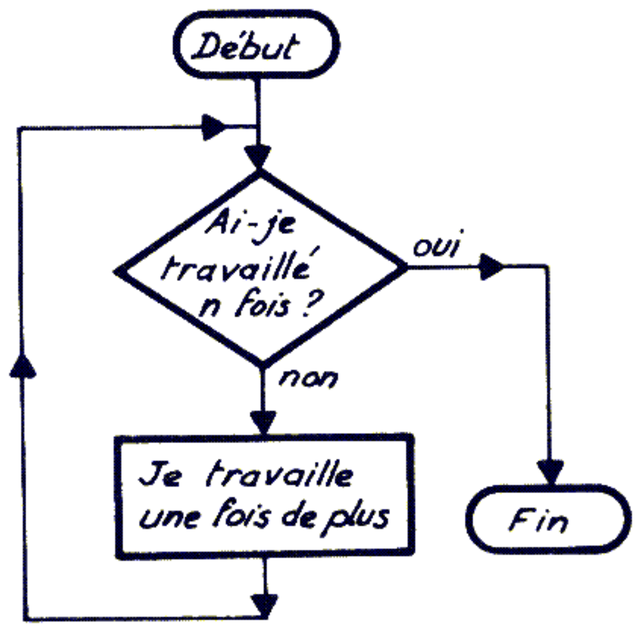


S'il le faut, on décomposera un rectangle en trois parties, fin, début et milieu, quitte à décomposer l'un ou l'autre des rectangles en ses propres fin, début et milieu.

Par souci de lisibilité et de cohérence (une fonction par boîte), deux autres cas généraux peuvent se présenter : l'alternance et la répétition. L'alternance est le choix entre deux options mutuellement exclusives. Elle se présente comme suit.



Quant à la répétition, elle consiste à effectuer un même traitement n fois :



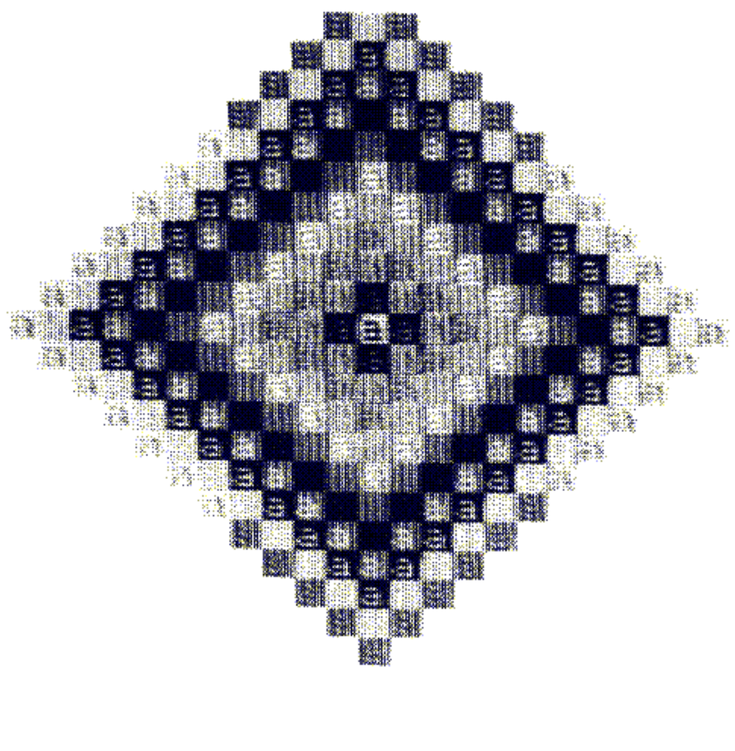
Ce dernier schéma peut être formalisé dans le code du programme. Ceci correspondrait à tester à la ligne 20 de notre exemple IF A=N THEN 50 et à remplacer la ligne 210 par A=A+1. On peut préférer l'intégrer dans le traitement concerné. Dans notre exemple, cela conduirait à rajouter une ligne 105 B=0, une ligne 205 B=B+1 et à modifier la ligne 210 en 210 IF B=N THEN A=1. Je préfère personnellement cette dernière solution, A n'ayant jamais que deux valeurs 0 — on continue — ou 1 — on s'arrête.

Nous verrons dans un prochain article le cas concret de construction d'un répétiteur de vocabulaire. Le programmeur aura retrouvé les sources de diverses fautes de programmation (que celui qui n'a jamais oublié d'initialiser une variable...). De façon plus pratique, on construit un programme morceau par morceau (est-ce que la fin me convient bien ?) en faisant évoluer simultanément le programme et sa validation. Utiliser la programmation comme outil de modélisation est le meilleur moyen de programmer effectivement vite et bien. N'hésitez pas à essayer de valider le plus tôt possible ce que vous avez développé. Plus la remise en cause est tardive et plus vous regretterez les heures perdues antérieurement.

Jean-Pierre BRUNERIE

QUAND LE HASARD SE MET A LA PEINTURE ABSTRAITE

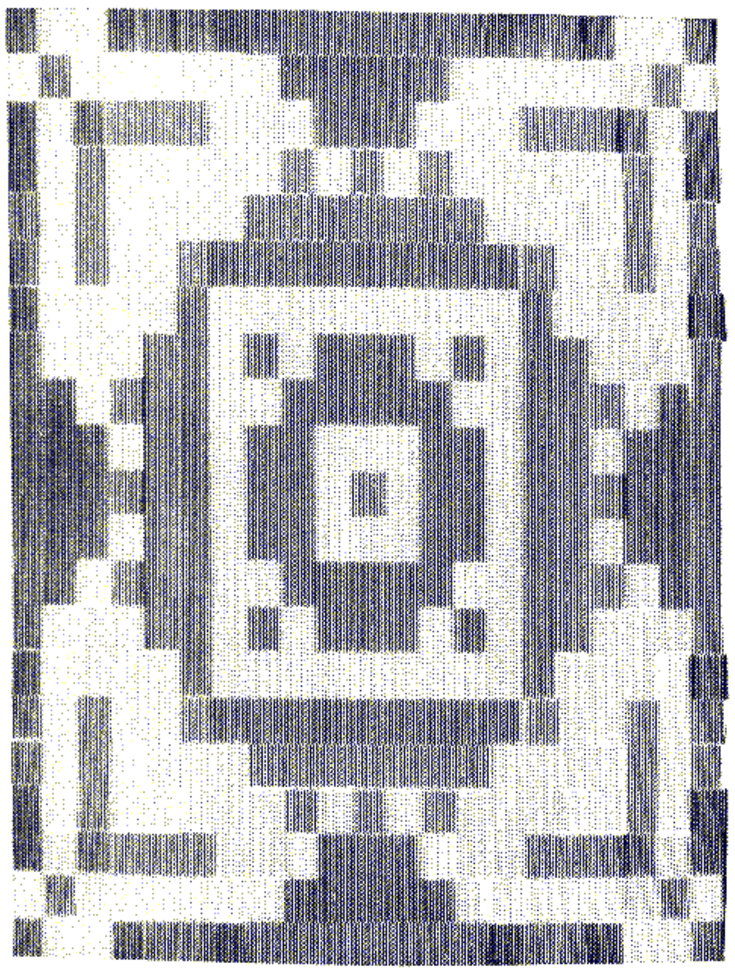
L'ÉCRAN d'un téléviseur éteint est d'un triste... Si votre ordinateur n'est pas occupé, laissez-lui le soin de créer pour vous des tableaux animés. Vous serez sans doute étonné par certaines de ses compositions.



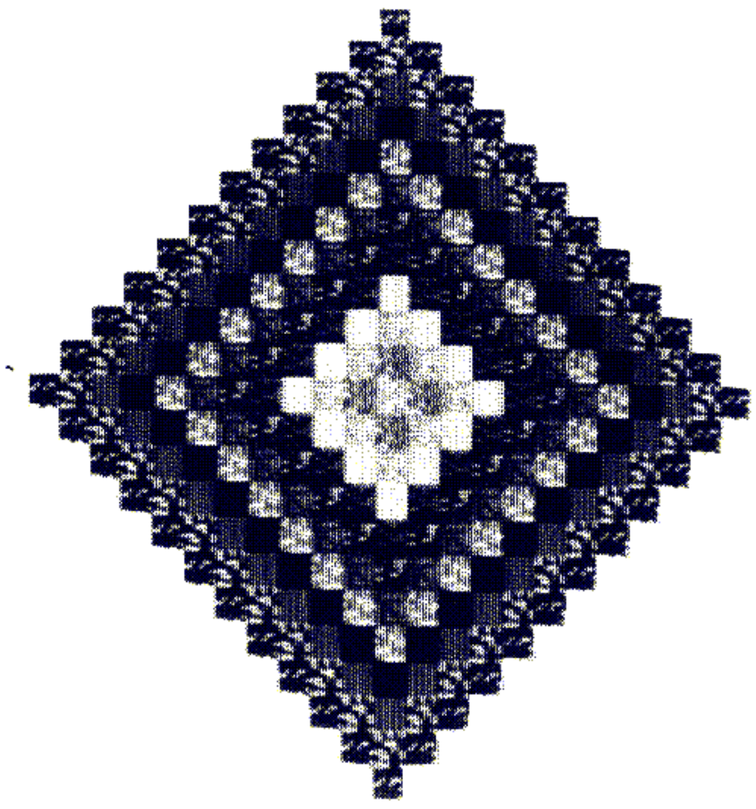
Carrés colorés et animés sur le Spectrum (programme n° 3)

Les courts programmes qui vous sont proposés ici pour TO7, TRS modèle 1, Lynx, Spectrum, vous donneront peut-être l'idée de développer à votre tour des logiciels qui ne sont destinés qu'au plaisir des yeux. Nous l'espérons.

Pour la plupart, une fois qu'ils sont lancés, rien ne les arrête. Ils affichent (en couleur, sauf sur le TRS) des compositions abstraites qui évoluent de seconde en seconde. Chacun de ces programmes est un point de départ autour duquel vous pourrez broder à votre guise.



Le Lynx propose un tapis persan...



Contraste et couleurs : le TO7 est agréable à suivre

Architecture, tissages, etc.

Quelques suggestions, si vous êtes en mal d'inspiration ; faites dessiner à votre ordinateur :

- des motifs de napperons,
- du pied-de-poule,
- du prince de galles,
- du tissu écossais,
- des rosaces de cathédrales,
- de faux Vasarely,
- des roues de paons,
- des plans de citadelles, de villes, etc.

Et n'oubliez pas de nous envoyer les programmes dont vous serez le plus fier, surtout s'ils sont très courts.

A titre d'exercice, vous pouvez vous

QUAND LE HASARD SE MET A LA PEINTURE ABSTRAITE

entraîner dans un premier temps sur le sujet suivant : certaines enseignes de pharmaciens sont des variations lumineuses sur le thème de la croix verte, avec clignotements multiples et autres effets d'optique. Observez-les et tâchez d'obtenir que votre ordinateur fasse aussi bien, ou mieux. Vous pouvez aussi inventer des carottes de buraliste, des fontaines lumineuses, des aquariums où se promènent d'étranges poissons chatoyants... Et vous aurez sans doute beaucoup d'autres idées.

LIST

SUR ZX SPECTRUM ET TO7

L'écran haute résolution du ZX Spectrum est géré par la machine d'une façon très particulière : en effet, les cases-mémoire correspondant à l'affichage ne sont pas consécutives, mais utilisent un arrangement spécial, qui tient compte en fait du balayage télévision (voir le manuel à ce sujet). De plus, les couleurs de fond, d'encre, le clignotement, la surbrillance sont gérés par un certain nombre de cases-mémoire, situées très exactement juste après cette mémoire d'écran.

Le premier programme proposé, outre le fait qu'il réalise des tableaux abstraits, permet de bien se rendre compte de cette organisation particulière. La mémoire d'écran va de l'adresse 16384 à l'adresse 22527 et le fichier des attributs va de 22528 à 23296.

Le deuxième programme réalise le même type de tableaux abstraits mais, cette fois-ci, directement à partir des instructions Basic Sinclair. Un caractère quelconque est d'abord emmagasiné en caractère graphique prédéfini (il faudra taper "GRAPHICS A" dans le PRINT), puis il est affiché sur l'écran. La couleur de l'encre, la couleur du fond, le clignotement et la surbrillance sont choisis de façon aléatoire. Le résultat est, comme précédemment, un tableau très inattendu que l'on peut modifier à volonté en pressant une touche quelconque. Une application spectaculaire de ce programme à la réalisation d'un kaléidoscope est ensuite proposée, ainsi qu'une adaptation du même programme pour l'ordinateur TO7.

Jacques DECONCHAT

SUR LYNX 48 KO

Le programme dans ses grandes lignes

100 : permet le chaînage avec un autre programme qui, sans cela, pourrait altérer les caractères normaux
120 et 150 : commande multiple de l'affichage par VDU
160 : initialisation des variables V et H
180 et 220 : changement de caractères selon la valeur 1 ou 2 du drapeau H
190 et 200 : début des boucles d'affichage
210 : permet de doubler ou de tripler le même caractère
240 à 270 : affichage du caractère à huit reprises
280 : scrutation du clavier
290 : permet de figer l'évolution du dessin
300 à 330 : si les touches appropriées sont pressées, le drapeau H change de valeur ; un bip signale que la commande a été enregistrée
340 et 350 : fin des boucles L et C

Mode d'utilisation

RUN	: construction du kaléidoscope
SPACE	: arrêt sur l'image
← et →	: changement du caractère utilisé
RETURN	: maintien du caractère

Animation graphique de l'écran
Programmes pour ZX Spectrum
Auteur Jacques Deconchat

Programme n° 1

```
10 FOR I=16384 TO 23295
20 POKE I,INT (RND*255)
30 NEXT I
```

Programme n° 2

```
 10 REM KALEIDOSCOPE
 30 FOR J=0 TO 7
 50 POKE USR "A"+J,INT (RND*255
)
 60 NEXT J
 80 LET F=INT (RND*7): LET G=IN
T (RND*7)
 90 PRINT INK F; PAPER G; FLASH
INT (RND*2); BRIGHT INT (RND*2)
"A";
100 GO TO 30
```

Programme n° 3

```
 10 REM KALEIDOSCOPE
 20 FOR K=0 TO 11
 30 FOR J=0 TO 7
 50 POKE USR "A"+J,INT (RND*255
)
 60 NEXT J
 80 LET F=INT (RND*7): LET G=IN
T (RND*7)
 90 LET R=INT (RND*2): LET S=IN
T (RND*2)
 110 FOR I=0 TO K
 120 PRINT AT 10-I,15-K+I; INK F
; PAPER G; FLASH R; BRIGHT S;"A"
 130 PRINT AT 10+I,15-K+I; INK F
; PAPER G; FLASH R; BRIGHT S;"A"
 140 PRINT AT 10+I,15+K-I; INK F
; PAPER G; FLASH R; BRIGHT S;"A"
 150 PRINT AT 10-I,15+K-I; INK F
; PAPER G; FLASH R; BRIGHT S;"A"
 160 NEXT I
 170 NEXT K
 180 GO TO 10
```

Kaléidoscope
Programme pour Lynx 48 Ko
Auteur Yvon Pérès
Copyright *LIST* et l'auteur

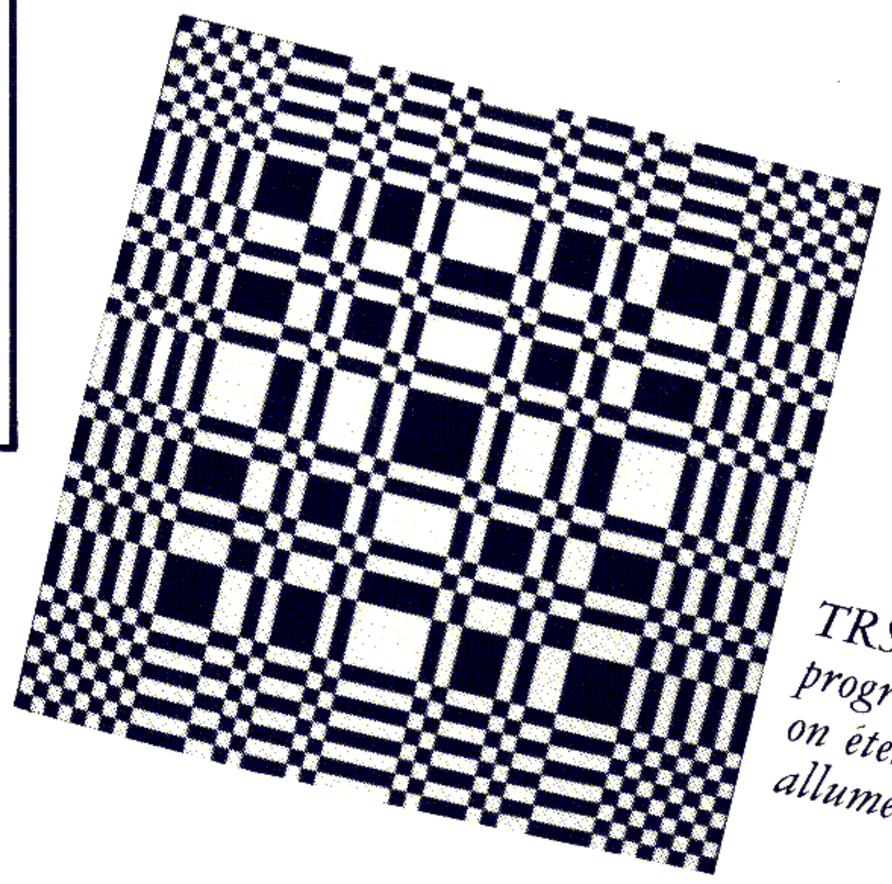
```
100 DPOKE GRAPHIC,&01D4
110 CLS
120 VDU 1,2,2,5,24
130 PRINT  @ 40,50;"    KALEIDOSCOPE    ";

140 PAUSE 15000
150 VDU 2,0,25,4
160 LET V=239.H=0
170 LABEL DEBUT
180 IF H=2 THEN  LET V=RAND(31)+225
190 FOR L=0 TO 10
200   FOR C=L TO 10
210     IF RAND(4)>RAND(4) THEN  GOTO 24
0
220     IF H=1 THEN  LET V=RAND(31)+225
230     INK RAND(6)+1
240     PRINT  @ C*3+33,(L+2)*10;CHR$(V)
;  @ 93-C*3,(L+2)*10;CHR$(V);
250     PRINT  @ C*3+33,220-L*10;CHR$(V)
;  @ 93-C*3,220-L*10;CHR$(V);
260 PRINT  @ L*3+33,(C+2)*10;CHR$(V);  @
93-L*3,(C+2)*10;CHR$(V);
270 PRINT  @ L*3+33,220-C*10;CHR$(V);  @
93-L*3,220-C*10;CHR$(V);
280 LET G=KEYN
290 IF G=32 THEN  LET G=GETN
300 IF G=13 THEN  LET H=0
310 IF G=22 THEN  LET H=1
320 IF G=12 THEN  LET H=2
330 IF G=13 OR G=22 OR G=12 THEN  BEEP 1
50,50,63
340 NEXT C
350 NEXT L
360 GOTO  LABEL DEBUT
```

En pressant les touches ← ou →, on obtiendra des caractères d'impression différents. Si l'on désire un fond de couleur aléatoire, on écrira, en ligne 150, VDU 2, RAND (7), 25, 4.

On peut également jouer sur la grandeur du dessin en remplaçant (ligne 190) le 0 de FOR L = 0 par un chiffre compris entre 0 et 5.

Kaléidoscope
Programme pour TO7
Auteur Jacques Deconchat
Copyright *LIST* et l'auteur

```
10 REM KALEIDOSCOPE
20 IF INKEY$="" THEN Y=RND:GOTO 20
30 CLEAR,,1:CLS
40 FOR K=0 TO 11
50 DEFGR$(0)=INT(RND*255),INT(RND*255),I
NT(RND*255),INT(RND*255),INT(RND*255),IN
T(RND*255),INT(RND*255),INT(RND*255)
60 F=INT(RND*7):G=INT(RND*7)
70 FOR I=0 TO K
80 COLOR F,G:LOCATE 20-K+I,12-I,0:PRINT
GR$(0)
90 COLOR F,G:LOCATE 20-K+I,12+I,0:PRINT
GR$(0)
100 COLOR F,G:LOCATE 20+K-I,12+I,0:PRINT
 GR$(0)
110 COLOR F,G:LOCATE 20+K-I,12-I,0:PRINT
 GR$(0)
120 NEXT I
130 NEXT K
140 GOTO 40
```



TRS 80, programme n° 2 : on éteint le point allumé et vice versa

Kaléidoscope
Programmes pour TRS 80 Modèle 1
Auteur Claude Balan
Copyright *LIST* et l'auteur

Programme n° 1

```
10 REM *** KALEIDOSCOPE ***
20 CLS
30 LET A = RND (47) - 1
40 LET H = RND (0)
50 IF H > 0.5 THEN GOTO 200
60 REM *** TRAITS BLANCS ***
70 FOR I = 0 TO 93
80     SET (I,A)
90     SET (I,46-A)
100 NEXT I
110 LET A = A + A
120 FOR I = 0 TO 46
130     SET (A,I)
140     SET (A+1,I)
150     SET (92-A,I)
160     SET (92-A+1,I)
170 NEXT I
180 GOTO 30
190 REM *** TRAITS NOIRS ***
200 FOR I = 0 TO 93
210     RESET (I,A)
220     RESET (I,46-A)
230 NEXT I
240 LET A = A + A
250 FOR I = 0 TO 46
260     RESET (A,I)
270     RESET (A+1,I)
280     RESET (92-A,I)
290     RESET (92-A+1,I)
300 NEXT I
310 REM *** ON RECOMMENCE ***
320 GOTO 30
```

SUR TRS 80 MODÈLE 1

Programme n° 2

```
10 REM *** KALEIDOSCOPE ***
20 CLS
30 LET A = RND (47) - 1
40 REM *** TRACES HORIZONTAUX ****
50 FOR I = 0 TO 93
60     IF POINT (I,A) THEN RESET (I,A) ELSE SET (I,A)
70     IF POINT (I,46-A) THEN RESET (I,46-A) ELSE SET (I,46-A)
80 NEXT I
90 REM *** TRACES VERTICAUX ***
100 LET A = A + A
110 FOR I = 0 TO 46
120     IF POINT (A,I) THEN RESET (A,I) ELSE SET (A,I)
130     IF POINT (A+1,I) THEN RESET (A+1,I) ELSE SET (A+1,I)
140     IF POINT (92-A,I) THEN RESET (92-A,I) ELSE SET (92-A,I)
150     IF POINT (92-A+1,I) THEN RESET (92-A+1,I) ELSE SET (92-A+1,I)
160 NEXT I
170 REM *** ON REPART POUR UN TOUR ***
180 GOTO 30
```

MISEZ P'TIT : OPTIMISEZ !

Si jongler avec la pile opérationnelle de votre HP-41 C, traquer la milliseconde perdue et rogner le moindre octet est votre pain quotidien... Ou si, à l'inverse, vous échappe parfois un peu de la subtile recherche des programmes en Notation Polonaise Inverse...

Alors, voici qui doit vous intéresser.

En matière de programmation, est-on jamais certain d'avoir fait aussi bien que possible ?

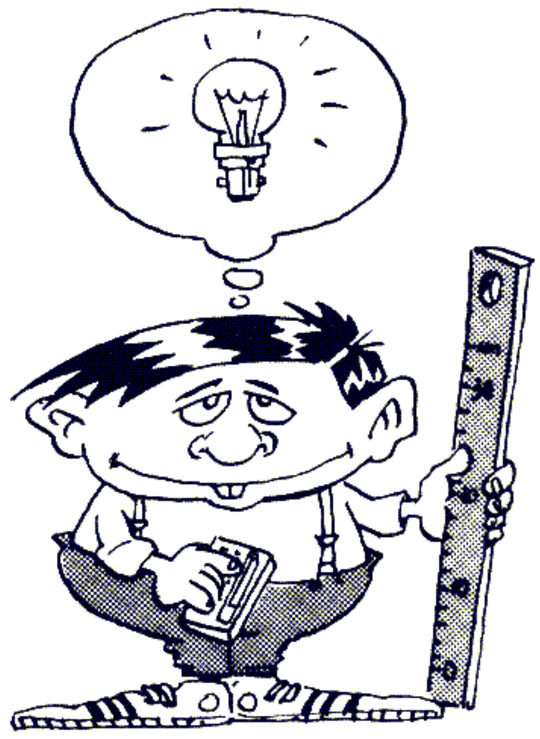
Le mieux pourrait-il être l'ami du bien ?

Dans cette rubrique, les défis se succèderont : des programmes toujours plus courts, plus rapides... Et les records tomberont !

Optimiser un programme, c'est en trouver la meilleure expression possible : la plus courte en nombre de pas de programme mais surtout en encombrement de la mémoire (en « octets » ; il existe des fonctions de 1, 2, 3 octets et plus, le détail en est donné à l'annexe D du manuel de la HP-41C). C'est obtenir le résultat recherché le plus rapidement possible ; c'est, enfin, consommer le moins de registres de mémoire possible.

Dans l'art d'optimiser, ici, peu importe en définitive le thème du programme - il aura intérêt à être très élémentaire - pourvu que l'on ait l'ivresse et la joie de la recherche et souvent aussi de la découverte du meilleur algorithme de résolution du problème, de la plus élégante astuce.

La philosophie de « Misez p'tit » est simple. Un lecteur lance un défi à tous les autres lecteurs : « *Voici les caractéristiques de mon programme, faites mieux si cela est possible !* »



Un principe cher à cette rubrique : les programmes réalisés sont « portables », utilisables partout et à tout moment avec l'un quelconque de tous vos autres programmes. Donc, ils ne doivent employer, lorsque cela est possible, aucune des mémoires normales R00 à Rnn, mais seulement les registres X, Y, Z, T et LASTX de la pile opérationnelle. Ils peuvent ainsi ultérieurement devenir de simples sous-programmes, presque comme de nouvelles fonctions.

On ne devra donc pas user non plus, dans les programmes optimisés, d'instructions d'arrêt (STOP,...) ou de demande de paramètres (PROMPT,...). Toutes les données nécessaires auront été fournies dans la pile opérationnelle avant l'exécution, et tous les résultats s'y trouveront ensuite.

Les critères d'une bonne optimisation (vitesse, taille mémoire) sont globalement complémentaires : plus c'est court, plus c'est rapide ! Mais là où les limites de l'optimisation sont proches, près de l'algorithme parfait, le mieux tend inversement à devenir l'ennemi du bien. En arrachant la suprême optimisation sur un point, on risque fort de perdre beaucoup sur un autre !

A vos marques, prêts ?

Gilles Bransbourg propose, ce mois-ci, deux programmes optimisés. L'un, « coefficients binômiaux » est un modèle du genre (mais si vous trouvez mieux...), l'autre, « simplifications de fractions » (Voir "Qui dit mieux ?") est bien le défi du mois. Son auteur en décrit les performances sans toutefois en révéler pour le moment tous les détails.

Il est souvent plus facile d'optimiser un défi que d'en lancer un, car il faut trouver un thème de qualité : pas trop classique (quoique... NDLR), *ni trop court, ni trop long ; aux multiples possibilités d'optimisation et pas encore tout à fait parfait. Il faut bien que l'on puisse relever le défi et qu'il suscite la bataille !*

Loi binômiale
Programme pour HP-41
Auteur Gilles Bransbourg


```
01 LBL"BIN
02 SIGN
03 LASTX
04 ST-Y
05 R↑
06 Y↑X
07 X<>Y
08 LASTX
09 FACT
10 ST/ Z
11 X<>L
12 R↑
13 ST-Y
14 FACT
15 RDN
16 CHS
17 Y↑X
18 LASTX
19 FACT
20 /
21 *
22 *
23 END
```

Pile	T	Z	Y	X	LAST X
Départ	?	k	n	p	?
SIGN LAST X ST-Y	k	n	(1-p)	p	—
R↑	n	(1-p)	p	k	—
y^x	n	n	(1-p)	p^k	k
X<>Y	n	n	p^k	(1-p)	k
LAST X FACT ST/Z	n	p^k	(1-p)	k !	k
X<>L	n	p^k/k !	(1-p)	k	—
R↑	p^k/k !	(1-p)	k	n	—
ST-Y	p^k/k !	(1-p)	k-n	n	—
FACT RDN	n !	p^k/k !	(1-p)	k-n	—
CHS y^x	n !	n !	p^k/k !	$(1-p)^{n-k}$	n-k
LAST X FACT	n !	p^k/k !	$(1-p)^{n-k}$	(n-k) !	—
/	n !	n !	p^k/k !	$\frac{(1-p)^{n-k}}{(n-k)!}$	—
*	n !	n !	n !	$\frac{p^k\ (1-p)^{n-k}}{k!\ (n-k)!}$	—
*	n !	n !	n !	$\frac{n!\ p^k\ (1-p)^{n-k}}{k!\ (n-k)!}$	—

Note : La présence d'un tiret dans la case du registre LAST X indique un contenu sans importance.

C'est ainsi que naquit ma première idée de défi : calculer les coefficients de la loi binômiale, bien (?) connue de générations de lycéens (au moins de réputation !). Le calcul en lui-même est fort simple et se résume ainsi : on se donne deux nombres entiers positifs n et k vérifiant la relation n ⩾ k (c-à-d n est toujours plus grand ou égal à k). On se donne un dernier nombre p compris entre 0 et 1 (en général 1/x où x est simplement un entier positif). La formule à calculer est :

$(n\ !/(n-k)\ !\ \times\ k\ !) \times (1-p)^{n-k} \times p^k$

ou encore $(n\ !\ \times\ (1-p)^{n-k} \times p^k)/((n-k)\ !\ \times\ k\ !)$

Il n'est pas nécessaire de connaître l'utilité de la formule, sa signification « profonde », pour la calculer et offrir ainsi le prétexte à une optimisation drastique. Le signe « ! » signifie « factorielle » et correspond à la fonction FACT de la HP-41C. Le reste n'est que division, multiplication, soustraction et élévation à la puissance (voir Y↑X).

Mon programme optimisé est reproduit ci-dessus avec le LBL" BIN. Il ne consomme que 25 octets de la mémoire (sans compter ni le LBL de tête, ni le END final) et n'utilise que les registres de la pile opérationnelle. Le calcul avec n=5, k=2 et p=1/4 soit 0,263671875... est réalisé en 1,3323 seconde.

L'introduction des paramètres de départ s'effectue ainsi : k ENTER n ENTER p, et XEQ la galère ! (XEQ « BIN » convient aussi). Le résultat est en X.

J'ai choisi de garder n et k dans la pile, de calculer plus tard n-k et d'effectuer successivement p↑k et la factorielle k !, puis de même (1-p) ↑ (n-k) et (n-k) !, en récupérant à chaque fois dans LASTX, k puis (n-k).

Le résultat doit tomber pile

Le novice en optimisation gagnera à bien étudier ce programme. Son principe de travail est simple, il suffit pour le suivre de bien examiner, au fur et à mesure des instructions, le mouvement de la pile opérationnelle et les résultats (voir le tableau ci-dessus).

Il est vraiment très difficile d'optimiser encore, mais ne parlons pas... A vous de répondre !

Jean-Christophe KRUST

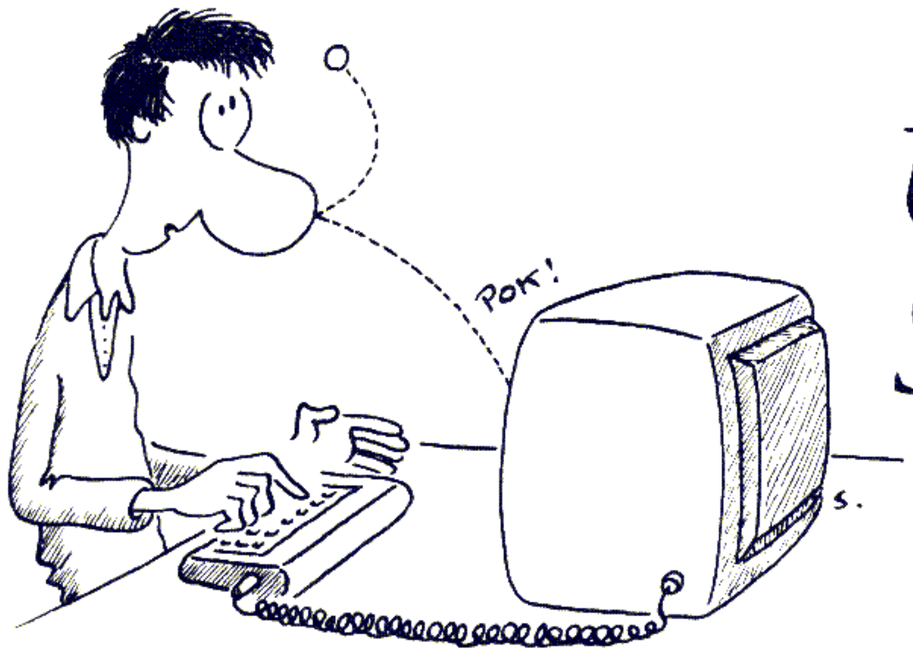
QUI DIT MIEUX ?

Dissertation sur le thème classique des simplifications de fractions. Peu de choses à dire, on introduit la fraction à simplifier : numérateur ENTER dénominateur, et le programme doit retourner, dans le même ordre, le numérateur et le dénominateur de la fraction simplifiée.

Mon programme occupe 15 octets en 12 pas de programme (sans compter ni le label de tête (LBL FRAC) ni le END final. Il trouve que 127/381 vaut 1/3 en 0,2262 seconde et même que π/1 « vaut » 1.570.796.327/500.000.000 en 1,4552 seconde.

Gilles BRANSBOURG

COMMENT J'AI PROGRAMMÉ



UN « SQUASH » SUR MON ORIC

TOUT jeu est défini par un ensemble de règles très précises. C'est sans doute pourquoi les jeux se prêtent si bien à la programmation. D'ailleurs, de ce point de vue, la programmation est aussi un jeu où il est impossible de tricher, mais où chacun a son style, son tour de main, ses recettes...

Jeu de squash
Programme pour Oric-1
Auteur Max Hagenburger

Une raquette, une balle, trois murs, en un mot un « squash », voilà ce que je vous propose de faire entrer dans votre Oric, en vous indiquant la manière dont je m'y suis pris.

En ce qui me concerne (et je ne suis pas le seul), je commence toujours par inscrire au début de chaque programme une ligne de REMarque qui me renseignera plus tard sur le sujet dudit programme. C'est bête comme chou, mais c'est tellement commode...

Y aura-t-il ou non des rebonds ?

La méthode que je pratique consiste à écrire ensuite la structure générale du programme sans me préoccuper du détail de chaque chapitre. Dans le cas du jeu de « squash », cela donne :

• Toute partie se dispute en plusieurs balles, et donc **au moins** en une, d'où la ligne 20 du programme : REPEAT : GOSUB 40 : UNTIL DERNIERE. Ici, GOSUB 40 équivaut au traitement d'une balle, et la variable DERNIERE est une variable « booléenne ». En fait, DERNIERE = (NBALLES > 10).

• Pour chaque balle, il peut y avoir **ou non** des renvois par la raquette. Il n'est pas question d'utiliser ici REPEAT... UNTIL. En effet, rien ne dit qu'il y aura au moins un coup de raquette bien placé. En ligne 50, on inscrit donc GOSUB 70 : IF RENVOI THEN 50. Le GOSUB 70 appelle le sous-programme traitant le renvoi d'une balle.

• De la même façon, après chaque renvoi par la raquette, il se produit **ou non** des rebonds sur les trois murs de l'espace de jeu. Cela nous donne, à la ligne 80, GOSUB 400 : IF REBOND THEN 80, le sous-programme qui débute en 400 traitant le déplacement de la balle entre les rebonds.

Pour achever l'ossature de notre programme, nous devons encore prévoir :

• Le début de la partie (dessin du terrain de jeu) et la fin (affichage du score).

• Le lancement d'une balle et le passage à la balle suivante.

• En cas de renvoi d'une balle, l'affichage du nouveau score (gain de 10 points en cas de succès).

On peut alors écrire :

```
100 REM début du programme (chapitre 100)
195 RETURN
```

```
0 REM ** squash **
5 REM une partie de squash:
10 GOSUB 100 'les balles:
20 REPEAT :GOSUB 40 :UNTIL DERNIERE
30 GOSUB 700 'fin
35 REM-----------
39 REM une balle:
40 GOSUB 200 'renvois raquette:
50 GOSUB 70 :IF RENVOI THEN 50
60 GOSUB 600 :RETURN
65 REM-----------
69 REM un renvoi de raquette:
70 GOSUB 300 'rebonds sur bords:
80 GOSUB 400 :IF REBOND THEN 80
90 GOSUB 500 :RETURN
99 REM===========
100 REM initialisations
105 CLS :PRINT CHR$(20)
110 LORES 0 :INK 5 :POKE 618,10
120 R$="#" :B$=" " :C$=CHR$(96)
125 K=10 :L=K+24 :H=21
130 PLOT K,0,C$+" Max"
135 PLOT K+18,0,"SQUASH"
140 FOR I=K TO L-1:PLOT I,1,"-" :NEXT
150 FOR I=K TO L-1:PLOT I,H,"-" :NEXT
160 FOR I=2 TO H-1:PLOT L,I,"I" :NEXT
170 R=11 :IF KEY$="" THEN 170
175 FOR I=R-1 TO R+1:PLOT K,I,A$:NEXT
180 :GOSUB 1000 'Raquette
185 SCORE=0
190 NBALS=1
195 RETURN
199 REM-----------
200 REM debut une balle
210 X=17+K
220 Y=INT(RND(1)*8+8)
230 DX=-1
240 DY=RND(1)*2-1
250 PLOT X,Y,B$
```

200 REM début d'une balle (chapitre 200)
295 RETURN

Même chose pour 300, 400, 500, 600 et 700, mais en 795 on écrira END et non pas RETURN.

Ce « squelette » de programme marche tout seul ! Si l'on demande RUN, il est même impossible de l'arrêter sans débrancher l'ordinateur ou ordonner CTRL C. Pourquoi ? Mais parce que la variable DERNIERE ne prend jamais la valeur « vraie ». Ajoutons les deux lignes suivantes :

680 NBALLES = NBALLES + 1
690 DERNIERE = (NBALLES > 10)

et le programme (qui ne fait toujours rien) s'arrêtera de lui-même à point nommé, à moins qu'il ne détecte une erreur que l'on s'empressera de corriger.

Au chapitre débutant en ligne 200, on détermine le point de départ de la balle :

210 X = 2 : Y = 10 : DX = 1 : DY = 1

Si l'on ajoute +1 à X, on va à droite, +1 à Y, on va vers le bas, −1 à X, on va à gauche, et −1 à Y, on va vers le haut.

On peut maintenant s'occuper tranquillement du point central du programme qui commence en ligne 400 : comment déplacer une balle sur l'écran.

On calcule d'abord la nouvelle position de la balle :

XX = X + DX et YY = Y + DY

(X, Y = ancienne position et XX, YY = nouvelle position). On efface alors l'ancienne position **avant** d'afficher la nouvelle, ce qui rend mieux l'illusion du mouvement continu :

PLOT X, Y, " " : PLOT XX, YY, "@"

Et on recommence, dans les limites du jeu sur l'écran (lignes 410 à 450). Cette répétition utilise le branchement conditionnel IF ... THEN 410, plus rapide que REPEAT : GOSUB 410 : UNTIL X + DX < 2 ... (la vitesse prime ici). Quand on a atteint les limites du terrain de jeu, on inverse le sens de déplacement (DX = −DX et DY = −DY).

La raquette doit rester dans l'écran

Autre point important de ce chapitre : comment faire bouger la raquette **en même temps** que la balle. C'est à la ligne 440 que GOSUB 1000 appelle le sous-programme gérant les mouvements de la raquette. Si l'on appuie sur les touches ↓ ou ↑, on la fait descendre ou monter. La variable R correspond à la position du milieu de la raquette et RR à la position qu'elle va prendre. On aurait pu programmer :

1005 R$ = KEY$
1010 IF R$ = CHR$ (10) THEN RR = R + 2
1020 IF R$ = CHR$ (11) THEN RR = R − 2

Mais avec une telle solution, le mouvement risque d'être saccadé, car il dépend de la fonction de répétition automatique des touches. On a donc intérêt à remplacer l'instruction KEY$ par un test sur le contenu d'une adresse-mémoire qui change de valeur quand on appuie sur une touche. Cela nous conduit à la formule IF PEEK (520) = 180 THEN RR = R + 2 et IF PEEK (520) = 156 THEN RR = R − 2 (lignes 1010 et 1020). L'octet 520 prend en effet la valeur 180 ou 156 selon que la touche ↓ ou ↑ est enfoncée.

Après avoir contrôlé que la raquette reste dans les limites de l'écran, on la déplace aux lignes 1060 et 1070.

Le plus important est fait. Il ne reste qu'à compléter chaque chapitre du programme. Ainsi, en passant en LORES 0, on obtient un fond noir (plus agréable à l'œil). On donne de l'effet à la balle quand elle rebondit sur la raquette avec l'expression (ligne 530) DY = DY/2 − (R − Y). On fait compter les points, on rend sonores les renvois de la balle...

Et dix autres petites choses qui font que, progressivement, le programme prend sa forme définitive.

Max HAGENBURGER

```
260 PLOT K,H+1,CHR$(2)+"balle"+STR$(NBALS)
295 RETURN
299 REM------------
300 REM debut renvoi raquette
320 PLOT K+15,H+1,"score"+STR$(SCORE)
395 RETURN
399 REM------------
400 REM deplacemt balle entre rebonds
410 : XX=X+DX :YY=Y+DY
420 : PLOT X,Y," " :PLOT XX,YY,C$
430 : X=XX :Y=YY
440 : GOSUB 1000 'Raquette
450 IF X+DX>K AND X+DX<L AND Y+DY=>2 AND Y+DY<H THEN 410  :REM un rebond:
470 IF Y+DY<2 OR Y+DY=>H THEN DY=-DY
480 IF X+DX=>L THEN DX=SGN(DX)*-2
490 REBOND = (X+DX>K)
495 RETURN
499 REM------------
500 REM fin renvoi raquette
510 ECRAN=SCRN(K,Y)
520 PLOT K,Y," " :PLOT X,Y," "
530 DX=SGN(DX)*-2 :DY=DY/2-(R-Y)
540 IF ABS(DY)>2 THEN DY=SGN(DY)*2
550 RENVOI=(ECRAN=ASC("#"))
560 IF RENVOI THEN PING :SCORE=SCORE+10 :PLOT K,Y,A$
595 RETURN
599 REM------------
600 REM fin d'une balle
620 EXPLODE
630 WAIT 100
680 NBALS= NBALS+1
690 DERNIERE = (NBALS>10)
695 RETURN
699 REM------------
700 REM fin du programme
760 CLS :INK 0
770 POKE 618,3
780 PRINT :PRINT "SCORE="SCORE
790 PRINT CHR$(20)
795 END
799 REM============
1000 REM raquette
1010 IF PEEK(520)=180 THEN RR=R+2
1020 IF PEEK(520)=156 THEN RR=R-2
1030 IF RR < 3 THEN RR=3
1040 IF RR >H-2 THEN RR=H-2
1060 IF RR>R THEN PLOT K,RR,A$:PLOT K,R-1,B$:PLOT K,RR+1,A$:PLOT K,R,B$
1070 IF RR<R THEN PLOT K,RR,A$:PLOT K,R+1,B$:PLOT K,RR-1,A$:PLOT K,R,B$
1080 R=RR
1095 RETURN
1099 REM================
```

TOOL POUR COMMODORE 64

TOOL est un logiciel destiné à améliorer les performances du Commodore 64. En apportant de nouvelles instructions au Basic d'origine, il dope l'ordinateur et simplifie la tâche du programmeur. Cette cartouche qui se connecte à l'arrière du C. 64 coûte... 8 Ko de mémoire vive et environ 640 FF ttc.

Le Commodore 64 est décidément une bien curieuse machine. Il peut tout faire, ou presque. Mais il est souvent incapable de faire les choses simplement. La seule programmation d'une page écran vraiment présentable passe par une armada de mouvements du curseur un peu décourageante quand on a eu l'occasion de travailler sur d'autres matériels. Et que dire de l'accès à la haute résolution, de la programmation des « sprites » et autres joyeusetés sonores ? Tout cela n'est pas vraiment à la portée du débutant.

Alors, bien sûr, les aides à la programmation fleurissent. Mais pas « l'Aide » universelle, celle qui d'un coup rendrait tout plus simple, plus facile, plus « accueillant ». Des multitudes d'aides, en fait, orientées vers telle ou telle des possibilités de l'appareil, à choisir en fonction de ses désirs ou de ses besoins. Je cite, pêle-mêle : *Tool 64* (gestion de l'écran et graphisme haute résolution), *Master* (gestion de l'écran et des disquettes), *Graf 64* (dessin), *Synthy 64* (musique), etc. Certaines de ces aides se présentent sous forme d'une cartouche, d'autres sous forme d'une disquette. J'ai choisi d'essayer l'une des plus répandues, et certainement l'une des plus remarquables : *Tool 64,* autrement (et en toute modestie) dit « the Tool », l'Outil.

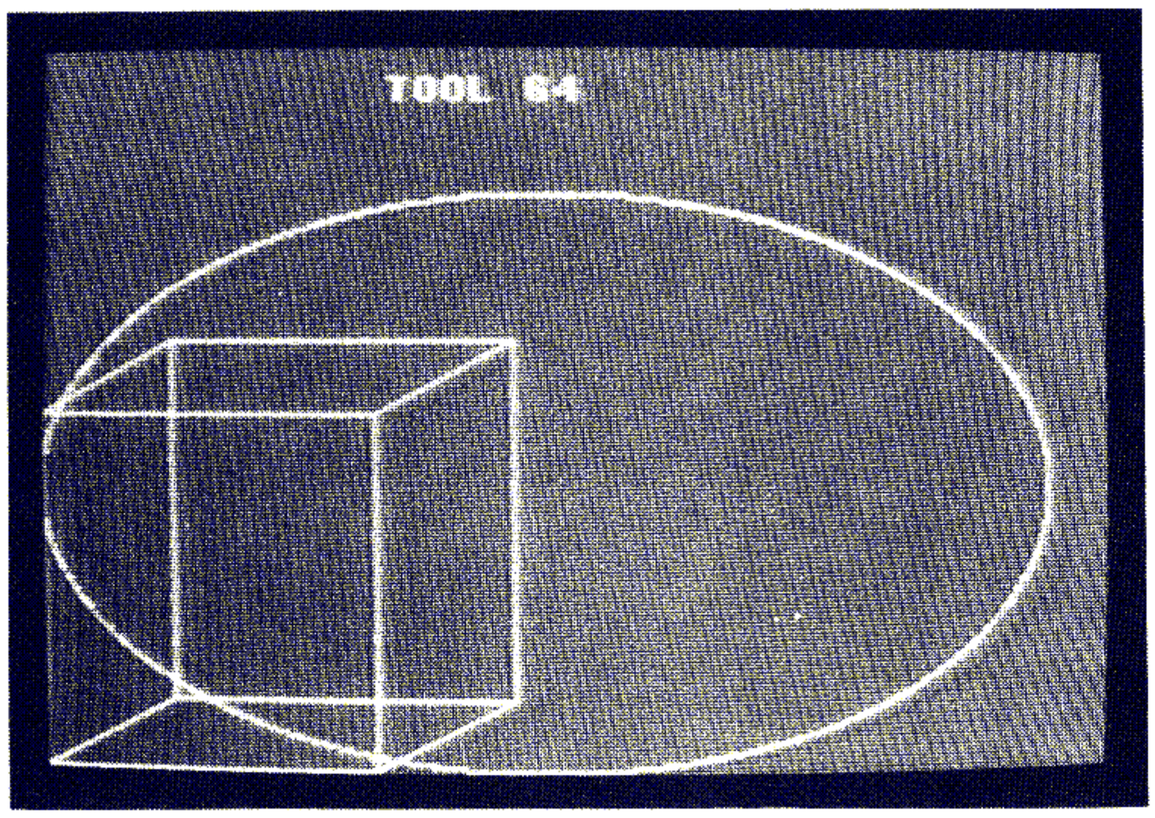
Tool 64, écrit par ***Micro Application*** et diffusé par ***Procep,*** se présente physiquement comme un petit classeur rigide, bien décoré. Le dos de la couverture contient, dans une poche plastique, la cartouche destinée à être enfichée dans le connecteur arrière du C. 64 (aucune erreur n'est possible). Le classeur contient également une documentation rédigée en français, très complète et très claire, qui présente et commente les principales caractéristiques du logiciel : une cinquantaine de pages, imprimées seulement d'un côté, et classées par thèmes.

Quand on y a goûté...

La lecture des rubriques permet de se rendre compte très vite que *Tool* simplifiera la vie du programmeur essentiellement dans trois domaines : la génération d'écrans, la haute résolution graphique, et l'aide à la programmation. Quelques autres facilités non négligeables sont offertes en complément, et un appendice permet de retrouver rapidement le nom des instructions supplémentaires, leur syntaxe et leur fonction. Chaque page de l'ouvrage présente une nouvelle instruction en donnant, de façon sommaire mais efficace, sa forme, sa syntaxe, son rôle et un exemple ou deux d'utilisation. Tout cela est très clair, et permet d'utiliser très rapidement les possibilités du logiciel.

A vrai dire, on se demande même, quand on a découvert un C. 64 muni de la cartouche *Tool,* comment on avait pu jusqu'alors se passer d'un tel

```
LIST

10 GRAPHIC:SCLEAR
20 A=0:B=20:C=40:D=100
30 MOVE A,A
40 DRAW A,D,1:DRAW D,D,1:DRAW D,A,1:DRAW
 A,A,1
50 DRAW C,B,1:DRAWC+D,B,1:DRAWC+D,B+D,1
60 DRAW C,B+D,1:DRAWA,D,1
70 MOVED A:DRAWD+C,B,1
80 MOVE C,B:DRAW C,B+D,1
90 MOVED D:DRAWD+C,D+B,1
100 DISPLAY "TOOL 64",2,14
120 MOVE 140,140
130 FOR I=0 TO 6.28 STEP 0.005
140 PLOT150*(1 +COS(I)), 80*(1+SIN(I)),1
150 NEXT I
160 CARGET ""
READY.
15 COLOR 4,1,20,40,1
RUN

READY.
```



Tool 64 est riche en fonctions graphiques

Haute résolution, en couleurs ou en noir et blanc

outil ! Vous connaissez certainement le refrain : « Ah, si j'avais connu plus tôt... ». Toujours est-il que, dès les premières secondes, on se rend compte que quelque chose a changé : l'affichage est en noir sur fond blanc, et non plus en bleu foncé sur fond bleu clair, comme c'est le cas normalement. De plus, la présence de Tool est immédiatement signalée par un petit message : « The Tool 64 / (c) Micro Application / 30719 bytes free » (au lieu des 38911 annoncés normalement) : le prix à payer en mémoire pour bénéficier des facilités offertes est donc de 8 Ko ! Reste à savoir, bien sûr, si le jeu en vaut la chandelle.

Encore ouverte sur la couleur

L'affichage, d'abord : TLINE et TCOL pour dessiner des lignes horizontales ou verticales, en n'importe quel point de l'écran texte (soit 24 lignes de 40 colonnes) et de n'importe quelle longueur. CLEAR, pour effacer n'importe quelle portion d'écran. OUT, pour afficher un texte à partir de la ligne et de la colonne désirées ; ainsi, OUT « BONJOUR », 5, 8 provoquera l'affichage du mot BONJOUR à la huitième colonne de la cinquième ligne. La commande REV permet de changer la couleur (fond ou caractères) d'une fenêtre de l'écran. SCROLL fait défiler tout ou partie de l'écran dans la direction désirée. Toutes ces instructions sont bien entendu suivies d'un certain nombre de paramètres grâce auxquels on indique les lignes, les colonnes, la longueur et la largeur des fenêtres concernées.

Quelle aisance dans la saisie

A cet ensemble d'instructions, déjà intéressant en lui-même, s'en ajoutent d'autres qui permettent l'acquisition de données et utilisent les concepts de page écran et de masque de saisie, avec des zones associées, en entrée ou en sortie. Ces fonctions, très puissantes, sont obtenues ici par des instructions faciles à employer : on définit une zone (parmi 128 possibles, pour une seule page écran) ; on effectue une saisie dans cette zone, et on transfère le résultat de cette saisie dans la variable désirée.

La définition d'une zone se fait au moyen de l'instruction DECZ, suivie d'un numéro d'identification, de la ligne et de la colonne d'origine, de la longueur (jusqu'à 255 caractères) et du type de contrôle effectué dans la zone. On peut saisir des données numériques, des textes, et définir la ou les touches de sortie de zone. Toutes les possibilités d'édition du C. 64 restent valables dans cette zone (déplacement du curseur, insertion, effacement...).

Si l'on dépasse la longueur prévue lors de la saisie, on en est averti par

LES COUPS D'OEIL DE LIST
TOOL POUR COMMODORE 64

une inversion vidéo : le curseur revient en début de zone et l'on peut recommencer la saisie, s'il y a lieu. La zone ainsi définie sera concrètement utilisée par une instruction REQZ *n* (qui « réquisitionne » la zone numéro *n*) : le curseur sera dès lors positionné au début de la zone définie par un DECZ *n*. Une troisième instruction sera utilisée pour transférer les données provenant de la zone *n* dans une variable alphanumérique au choix de l'utilisateur.

Toujours dans les instructions d'acquisition de données, on notera la présence d'une intéressante instruction notée CARGET, qui améliore considérablement le célèbre GET A$: on peut attendre un caractère, parmi ceux que l'on a présélectionnés, et ceci à un endroit quelconque de l'écran. Ainsi CARGET « abc » suspend l'exécution du programme tant que l'une des touches a, b, ou c n'a pas été frappée. La touche choisie sera d'ailleurs reconnue et enregistrée sous deux formes : un numéro d'ordre d'une part (1 pour a, 2 pour b et 3 pour c) stocké dans la variable réservée OK, et d'autre part la valeur ASCII du caractère choisi, emmagasinée dans ZO (qui est la deuxième variable réservée utilisée par *The Tool*).

Les pages d'écran, si l'on dispose d'une unité de disquette, sont sauvegardées, effacées ou chargées au moyen des trois instructions SSAVE, SLOAD, et SCLEAR.

L'art à l'écran

Passons maintenant aux graphismes. Le système *The Tool* permet au C. 64 de fonctionner sous deux modes : le mode Texte et le mode Graphique. Les instructions valent pour les deux modes. Cela dit, le système différencie et mémorise séparément l'écran graphique et l'écran texte. Le passage de l'un à l'autre se fait très simplement en tapant TEXT ou GRAPHIC, selon le cas. De plus, l'instruction SSAVE, que nous avons déjà rencontrée, sauvegarde la page courante (texte ou graphique), qui sera effacée par SCLEAR.

Les instructions graphiques permettent de travailler sur un écran haute résolution de 200 × 320 points. L'origine est logiquement placée en bas à gauche. MOVE positionne le curseur, PLOT allume ou éteint un point, DRAW trace ou efface une ligne, POINT teste l'état d'un point et DISPLAY permet l'affichage d'un texte dans l'écran graphique. La couleur est définie par blocs de 8 fois 8 points. Tout cela est assez complet et facilitera beaucoup la réalisation d'affichages de qualité.

Avez-vous fait le tour de *Tool* ? Bien sûr que non. Parmi les autres instructions intéressantes, notons :

- RENU qui renumérote les lignes d'un programme Basic ;
- DUMP qui affiche la liste des variables utilisées et leurs valeurs après une exécution ;
- ERROR qui indique la ligne où la machine détecte une bogue ;
- FIND qui retrouve la ou les occurrences d'une chaîne de caractères, dans un programme Basic ;
- TRACE, enfin, pour l'exécution pas à pas d'un programme.

Autant d'aides à la mise au point des programmes.

A l'usage, *The Tool* m'est apparu comme un complément de qualité, tout à fait utile pour qui veut programmer des applications faisant appel à de nombreuses saisies et à des écrans multiples. Le prix (environ 640 FF) paraît très raisonnable en regard des avantages apportés. Incontestablement, ce logiciel rend le Commodore 64 beaucoup plus agréable à utiliser. Est-ce un compliment pour cet ordinateur ?

Jacques DECONCHAT

LES COUPS D'OEIL DE LIST
LE BASIC ÉTENDU DU TI 99/4A

LE Basic étendu (en anglais TI Extended Basic) se présente sous la forme d'une cartouche de mémoire morte. C'est un ensemble de fonctions sophistiquées et pas courantes qui facilitent la gestion de l'écran et l'écriture des programmes. Comme on va le voir, on y trouve bon nombre d'ingrédients très utiles.

L'attrait principal du Basic étendu réside sans aucun doute dans la manipulation des lutins graphiques. Rappelons qu'il s'agit de symboles graphiques mobiles doués d'autonomie dans leurs déplacements. Ils se superposent au contenu de l'écran sans l'effacer. Cela permet toutes sortes d'animations de sujets au travers d'un décor qui reste fixe : *Pac Man, Startrek* — ou textes. Il s'agit toujours de déplacer des images en deux temps trois mouvements.

C'est évidemment l'aspect le plus spectaculaire du Basic étendu. Imaginez non pas 8, 12 ou encore 16, mais bien 28 lutins de 16 couleurs différentes, dessinés en haute résolution et circulant sans saccades en même temps sur l'écran ! Chacun d'eux est défini par son nom, le code ASCII affecté à sa définition, sa couleur, ses coordonnées de départ ainsi qu'un secteur vitesse (entre −128 et +127) atteignant un écran par seconde (instruction CALL SPRITE). Ajoutez à cela des fonctions complexes telles que POSITION qui peut indiquer la position absolue de 22 lutins simultanément dans autant de variables.

Le manuel et la cartouche du Basic étendu

DISTANCE évalue les positions relatives de deux lutins (ou bien d'un lutin et d'un caractère) tandis que COINC détecte la coïncidence de deux lutins (ou d'un lutin avec un caractère) en un même emplacement. En effet, les lutins sont dotés de numéros prioritaires pour l'affichage en cas de recouvrement ou chevauchement. MOTION modifie la vitesse d'un lutin (*sprite* anglais) en cours de déplacement, et PATTERN sa forme, sans affecter les autres paramètres : l'illusion du mouvement devient parfaite. LOCATE transfère immédiatement un lutin d'un point à un autre de l'écran, et DELSPRITE le retire de la circulation. Non content de cela, le Basic étendu possède une fonction « zoom » : MAGNIFY agrandit jusqu'à trois fois un caractère de base ; en composant des lutins de taille maximum, on arrive facilement à animer tout l'écran avec de grands mobiles complexes.

Un tout petit programme est lancé et...

```
>LIST
 100 CALL CLEAR
 110 CALL CHAR(96,"1898FF3D3C
 3CE404")
 120 DISPLAY AT(8,8):"BONJOUR
 "
 130 CALL SPRITE(#1,96,5,92,1
 24)
 140 CALL SPRITE(#2,96,5,45,4
 5)
 150 CALL MAGNIFY(2)
 160 FOR I=1 TO 1000
 170 NEXT I
>
```

Autour de ces nouvelles macro-instructions graphiques viennent se greffer des fonctions améliorées du TI Basic : DISPLAY AT est l'équivalent de PRINT AT qui débute l'affichage aux coordonnées indiquées. Existent aussi PRINT USING et DISPLAY USING suivis obligatoirement d'un numéro de ligne où figurent des indications de format (IMAGE). Plusieurs formats peuvent être définis au sein d'un même programme et sélectionnés séparément. ACCEPT AT est un développement du classique INPUT, il utilise la liste de paramètres suivante :

- coordonnées du curseur,
- activation du clavier alphabétique ou numérique,
- validation limitée aux touches désignées,
- bip sonore,
- effacement de l'écran avant message,
- variable.

Messages d'erreur plus explicites

Cette richesse déroutante dans les options permet de tirer le meilleur parti de séquences préprogrammées dans la majorité des situations. IF THEN ELSE en est la preuve, qui revêt la structure IF *condition* THEN *instruction* ELSE *instruction,* où *instruction* peut être différent de GOTO.

Parmi les autres améliorations de taille, citons les trois suivantes : des messages d'erreur plus complets qu'en TI Basic, et surtout plus explicites, la possibilité de plusieurs instructions sur une même ligne ; et enfin un meilleur manuel d'emploi. Nous reviendrons sur ce dernier.

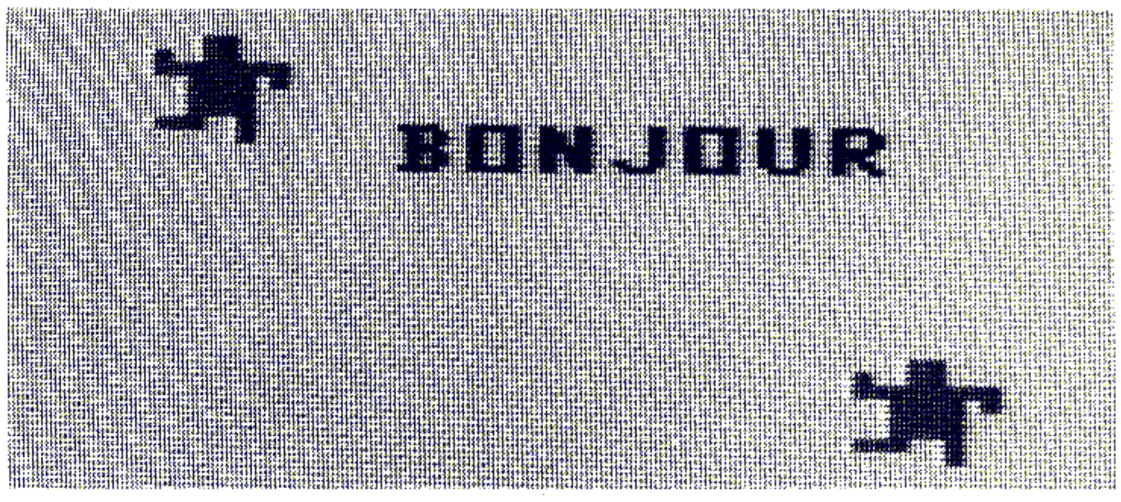
Au chapitre des fonctions très utiles, citons ON ERROR GOTO/GOSUB, secondée par ERR qui contient le dernier message d'erreur enregistré. ON WARNING permet de ne pas tenir compte des messages système — ou mieux, de les remplacer par les vôtres. SIZE est l'équivalent du FREE qui rappelle le nombre d'octets disponibles en mémoire. La commande RPT$ est une des plus puissantes du genre : elle répète une chaîne alphanumérique dans une instruction autant de fois qu'il est spécifié, et au besoin en répétant l'instruction.

Les fonctions de ce type sont assez complexes à utiliser, mais elles sont bien pratiques : une lecture attentive du manuel s'impose. Celui-ci est bien conçu, ce qui est étonnant quand on se souvient de celui du TI Basic. Les ordres Basic y sont classés par ordre alphabétique dans des fiches explicatives illustrées d'exemples, tandis qu'une introduction structurée rétablit un ordre de lecture correct pour le novice. Les tables et index divers n'ont pas été négligés et sont repris dans une dizaine d'appendices finaux.

Continuons notre description avec les fonctions booléennes AND, OR et NOT, puis MAX et MIN ; les fonctions de gestion de fichier sont décrites dans le détail, mais de manière assez obscure : signalons rapidement un MERGE pour fusionner deux programmes, REC pour localiser un enregistrement, et LINPUT pour saisir une ligne de texte complète.

Quant aux fonctions LOAD, LINK, PEEK et INIT de l'assembleur, ainsi que SAY et SPGET, leur utilisation nécessite les extensions assembleur et synthétiseur de parole.

Nous avons bien sûr gardé le meilleur pour la fin : il s'agit du contrôle



... deux sympathiques lutins nous saluent !

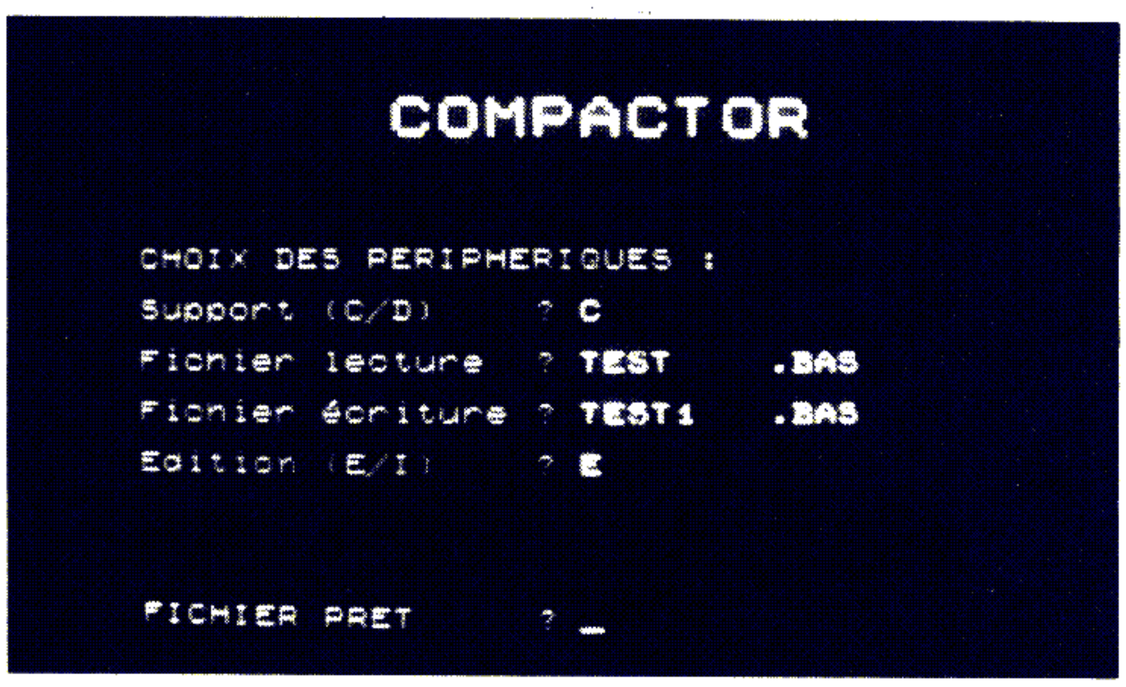
de procédures de type « Pascal » , SUB « texte » permet de nommer une routine limitée par SUB END, et exécutable par CALL « texte ».

A la différence du Pascal, il n'y a pas passage de variables, car les variables concernées doivent être déclarées en début de routine et conservent leur valeur courante. Cette caractéristique prometteuse est particulièrement adaptée à l'initiation au Basic et à l'algorithmique. Voilà qui ajoute encore à l'intérêt de ce module qui vaut environ 500 FF ttc (distribué par Texas Instruments).

Michel ARDITTI

COMPACTOR POUR TO 7

ÉDITÉ par TO TEK (prononcer Té-ô-tek), Compactor est un utilitaire – praticiel – en cartouche, destiné aux utilisateurs du TO 7. Il sera utile à tous ceux qui souffrent du problème classique de manque d'espace mémoire, ou plus simplement à ceux qui veulent optimiser leurs programmes.



Compactor informé fera bien son travail

Compactor comporte cinq options de travail : une renumérotation automatique, une table de références, une suppression des commentaires, une suppression des espaces inutiles et un compactage.

La première option, *renumérotation*, permet une renumérotation des lignes du programme ou d'une partie du programme seulement. Elle corrige aussi les branchements comme GOTO ou GOSUB. Les paramètres à fournir sont le numéro de la première ligne, le nouveau numéro à lui attribuer et le pas de numérotation.

La rencontre d'un problème en cours de programme entraîne un message d'erreur.

Cette option est utile : elle améliore la présentation des programmes et leur lisibilité. Elle permet en outre d'insérer des lignes oubliées, entre deux lignes consécutives (une ligne 125 entre les lignes 120 et 130, par exemple).

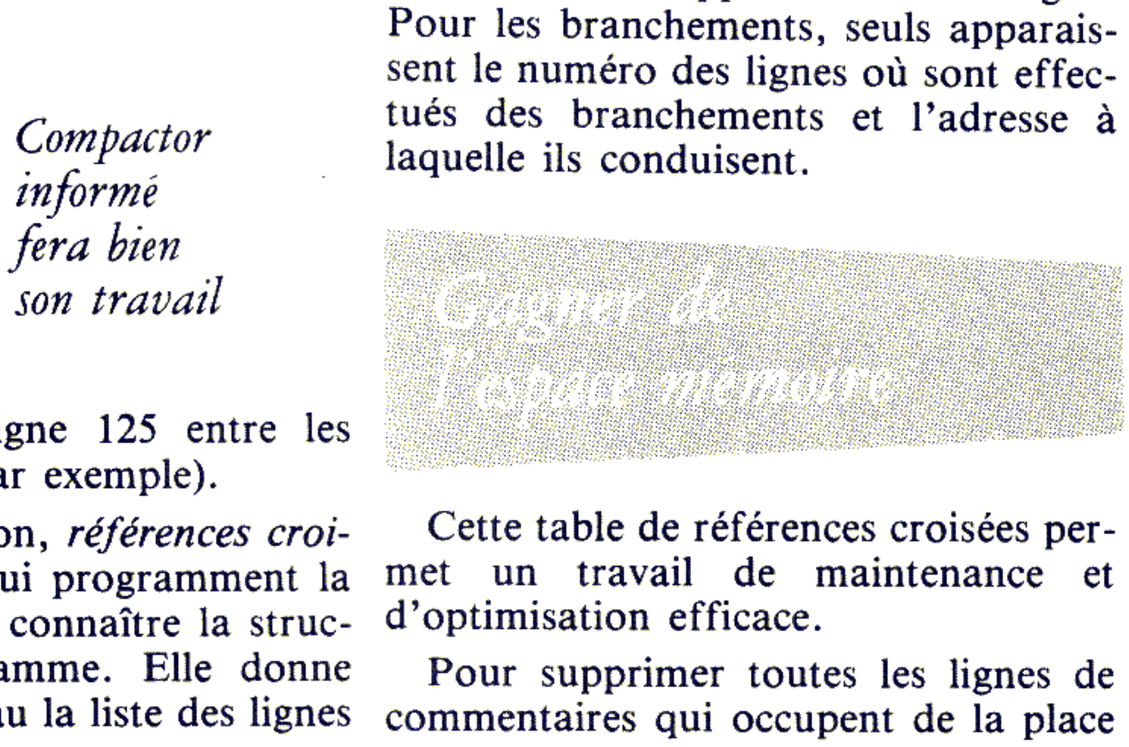
La deuxième option, *références croisées*, offre à ceux qui programment la possibilité de mieux connaître la structure de leur programme. Elle donne sous forme de tableau la liste des lignes où apparaissent les variables et les lignes comportant des branchements.

Ce tableau peut être obtenu sur l'écran ou sur l'imprimante. Pour les variables, il comporte leur nom, le numéro des lignes où elles apparaissent et le nombre d'apparitions sur la ligne. Pour les branchements, seuls apparaissent le numéro des lignes où sont effectués des branchements et l'adresse à laquelle ils conduisent.

Gagnez de l'espace mémoire

Cette table de références croisées permet un travail de maintenance et d'optimisation efficace.

Pour supprimer toutes les lignes de commentaires qui occupent de la place



Le menu de Compactor propose cinq options

```
100 REM PROGRAMME TEST
105 INPUT"VOTRE CHOIX ";REPON$
110 ON VAL(REPON$) GOSUB 200,300
123 GOTO 105
190 :
200 REM 1ER CAS
210 GOSUB 500
215 RETURN
300 REM 2EME CAS= BOUCLE VIDE
304 CMPT1=0
305 CMPT1=CMPT1+1
306 FOR COMPTEUR=0 TO 200
307 REM LIGNE INUTILE
310 NEXT COMPTEUR
311 IF CMPT1<10 THEN GOTO 305
315 PRINT"TERMINE"
320 RETURN
500 REM S/P
565 PRINT "VOUS AVEZ CHOISI: ";REPON$
600 RETURN
650 REM LIGNE CONSERVEE APRES COMPACTAGE
```

Un programme test en exemple : il occupe 377 octets.

```
100 REM PROGRAMME TEST
105 INPUT"VOTRE CHOIX ";REPON$
110 ON VAL(REPON$) GOSUB 125,140
115 GOTO 105
120 :
125 REM 1ER CAS
130 GOSUB 185
135 RETURN
140 REM 2EME CAS= BOUCLE VIDE
145 CMPT1=0
150 CMPT1=CMPT1+1
155 FOR COMPTEUR=0 TO 200
160 REM LIGNE INUTILE
165 NEXT COMPTEUR
170 IF CMPT1<10 THEN GOTO 150
175 PRINT"TERMINE"
180 RETURN
185 REM S/P
190 PRINT "VOUS AVEZ CHOISI: ";REPON$
195 RETURN
200 REM LIGNE CONSERVEE APRES COMPACTAGE
```

Sous l'action de la première option, le programme est renuméroté.

en mémoire mais n'interviennent pas dans le déroulement du programme, la troisième option, *suppression des rem,* est indispensable.

Et si le programme est abondamment commenté, quel gain de place mémoire ! De plus, l'exécution du programme en est plus rapide. Mais il est trop tard pour chercher à comprendre ce qui se passe dans le programme.

Avec la quatrième option, *suppression des blancs,* seuls les espaces inutiles du programme seront supprimés. Là encore, on gagne de la place en mémoire et l'exécution du programme devient plus rapide. Si les commentaires du programme sont intacts, l'absence des blancs rend la relecture des lignes de Basic moins aisée.

La dernière option, *compactage,* est l'option majeure de Compactor ! Elle effectue un travail complet de « nettoyage » du programme avec renumérotation, suppression des commentaires, suppression des espaces inutiles, réduction des THEN-GOTO, réduction des noms de variables, concaténation des lignes.

Cette option permet un gain particulièrement important de place mémoire. Mais une fois soumis à ce compactage, le programme ne peut généralement plus être édité ou modifié facilement.

Cette opération de compactage n'équivaut pas à une véritable compilation. Il s'agit en fait d'une optimisation intéressante, tant par l'occupation minimum de la mémoire que par la vitesse d'exécution du programme (encore que le gain en vitesse ne soit pas énorme).

Son principal défaut est de rendre les programmes pratiquement non modifiables — bien qu'il ne s'agisse pas, notons-le, d'une véritable compilation. Il ne faudra donc compacter que des programmes parfaitement au point...

Un équipement minimum nécessaire

Compactor est utilisable avec l'équipement minimum que constituent l'unité centrale et son magnétophone.

Toutefois, un lecteur de disquette et une imprimante simplifieront sensiblement le travail. En effet, l'édition sur papier de la table des références croisées est d'un emploi plus pratique qu'une édition sur l'écran...

Qu'il soit lu sur cassette ou sur disquette, le programme modifié est réécrit sur le même périphérique. Attention donc aux erreurs de manipulation pouvant entraîner la perte du programme original.

Il est tout à fait conseillé d'utiliser Compactor sur une copie, et de conserver l'original en archives.

L'utilisation de Compactor satisfera ceux qui aiment programmer sur leur TO7. Ils pourront enfin renuméroter, gagner en mémoire et en vitesse d'exécution, compacter un programme. Un plaisir qui leur coûtera environ 490 FF ttc.

Jean-Pierre LALEVÉE

```
 1 INPUT"VOTRE CHOIX ";R0$:ONVAL(R0$)GOSUB2,3:GOTO1
 2 GOSUB6:RETURN
 3 C0=0
 4 C0=C0+1:FORC1=0TO200:NEXTC1:IFC0<10THEN4
 5 PRINT"TERMINE":RETURN
 6 PRINT"VOUS AVEZ CHOISI: ";R0$:RETURN
 7 REM
```

Après avoir subi le compactage (option 5), le programme n'occupe plus que 154 octets !

LANGAGE-MACHINE ET ASSEMBLEUR

PEU importe que vous trouviez, vous, que se nourrir tous les jours de Gourmet-Foies-Volailles est totalement insipide alors qu'il y a les frites et les fonds d'artichaut Saint-Louis. C'est à votre chat siamois de vous imposer son goût en ce qui concerne le contenu de son plat, et non à vous de lui dicter le vôtre. Ainsi vous ne vibrez qu'avec Basic ou Pascal, mais votre processeur Z 80 se moque de vos INPUT et BEGIN comme de sa première bogue. Son régal à lui, chacun le sait, c'est bien entendu le langage-machine (ou binaire).

Voici un court exemple de ce dont se nourrit en fait un processeur :

```
00100001  10100011  01110010
01000110  10010111  00100011
10111110  00110000  00000001
01111110  00010000  11111001
00110010  11110010  01111010
```

Les cent vingt chiffres écrits ci-dessus, séparés par des espaces correspondant à la localisation des instructions dans l'ordinateur, ne sont pas écrits au hasard. C'est le programme qui consiste à prendre le plus grand d'une série de nombres placés dans les mémoires de la machine. Nous reviendrons à lui un peu plus tard.

Le premier programmeur de l'histoire, la célèbre comtesse Augusta Ada Lovelace, fille de Byron, utilisait déjà en principe ce genre de langage (cf. ses mémoires scientifiques de 1842 sur sa contribution à la « machine analytique » de Charles Babbage). Mais on sait que l'ordinateur théorique sur lequel elle travaillait n'a jamais été construit. Il vaut donc mieux remonter aux véritables débuts de l'informatique. Les pionniers des années quarante n'avaient en fait que deux moyens de se faire obéir :

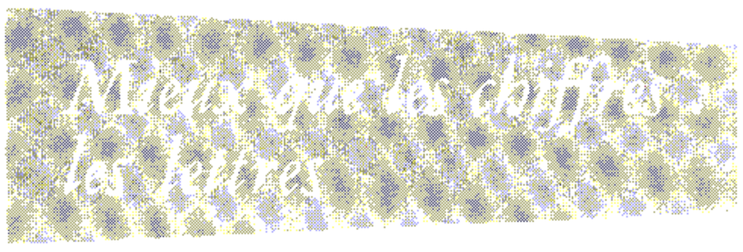
- ou bien câbler les machines de manière convenable avant le début de chaque application, par exemple en ouvrant et fermant toute une batterie d'interrupteurs,
- ou bien écrire de véritables programmes placés à l'intérieur même des mémoires (c'est le coup de génie de Von Neumann), qui sont effectivement des listes de 0 et de 1 et constituent des codes binaires en langage-machine.

Bien entendu utiliser de tels symboles est un supplice, et les erreurs d'écriture sont infiniment probables (sans compter que la relecture est presque impossible). Très vite on s'est préoccupé d'établir des moyens automatiques d'écriture de ces fameuses listes. Ce furent les premiers assembleurs. Aujourd'hui trop de gens confondent, à tort, assembleur et binaire. L'assembleur constitue déjà un très gros progrès sur le langage-machine. En fait le programmeur est amené à utiliser sur son clavier ce que l'on appelle des mnémoniques qui seront transformées directement en zéros et en uns par un programme, écrit une fois pour toutes, qui s'appelle lui aussi « assembleur ». De ce point de vue-là, il n'y a donc pas de différences fondamentales entre langage évolué et assembleur, dans la mesure où tous deux devront subir ensuite une traduction, « interprétation » ou « compilation », heureusement automatique, pour être utilisable par l'unité centrale.

Plus court et plus simple

Une première simplification consistait bien sûr à traiter les 10111110 comme des nombres écrits en binaire, que l'on peut mémoriser sous leur forme usuelle (c'est-à-dire décimale) ; c'est ainsi que le programme ci-dessus est encore équivalent à la suite des quinze nombres 33 163 114 70 151 35 190 48 1 126 16 249 50 242 et 122 : par exemple le

nombre binaire 10111110 = 190 en décimal. Pour plus de commodité, on utilise en fait l'écriture hexadécimale de base seize et non dix. Les chiffres en sont, outre 0 1 2 3 4 5 6 7 8 9, les signes A (=10), B (=11), C (=12), D (=13), E (=14) et F (=15). Voilà donc la liste complète en hexa de notre programme Z 80 de calcul du plus grand terme d'une suite de nombres :
21 A3 72 46 97 23 BE 30 01 7E 10 F9 32 F2 7A. Ainsi, par exemple, 10111110 = 190 (déc) = BE (hexa).



Mais cette simplification est juste un raccourcissement ; elle ne donne aucune indication qui puisse aider un peu à la compréhension des instructions. En effet 10111110, qui signifie « comparer le contenu du registre dont le numéro d'ordre est contenu dans le registre HL à celui de l'accumulateur A, et mémoriser un symbole de retenue si et seulement si la valeur en A était strictement inférieure », sera peut-être un peu plus facile à reconnaître de loin si on l'écrit sous la forme symbolique CP (HL). Au moins les lettres CP, par exemple, évoquent-elles bien une comparaison, alors que 10111110 n'a guère de chances de dire quoi que ce soit à personne...

C'est donc là l'intérêt de l'assembleur. Ecrivons une fois de plus notre programme machine avec ces codes un peu moins hermétiques (?). Cela donne :

```
LD HL,72A3H
LD B,(HL)
SUB A
INC HL
CP (HL)
JR NC,L2
LD A,(HL)
DJNZ L1
LD (7AF2H),A
```

où l'on peut lire LD pour « load » (= charger), SUB pour « soustraire », CP pour « comparer », A, HL... étant des noms de registres (mémoires spécialisées), 72A3 une adresse en hexadécimal (d'où la finale H), etc. Même si un profane ne voit pas très bien l'étendue du gain, quelques heures de pratique le persuaderaient facilement de la différence.

Dans la pratique, un programme écrit en assembleur se présente de la manière suivante :

```
8000                00100          ORG     8000H
8000 21A372         00110 TRS      LD      HL,72A3H
8003 46             00120          LD      B, (HL)
8004 97             00130          SUB     A
8005 23             00140 L1       INC     HL
8006 BE             00150          CP      (HL)
8007 3001           00160          JR      NC,L2
8009 7E             00170          LD      A, (HL)
800A 10F9           00180 L2       DJNZ    L1
800C 32F27A         00190          LD      (7AF2H), A
8000                00200          END     TRS
00000 Total Errors
L2   800A
L1   8005
TRS  8000
```

Inutile de vérifier que ce programme est sans erreur, c'est écrit !

Les deux colonnes de gauche sont mises au point automatiquement par le programme spécial de traduction (souvent écrit par le constructeur de la machine et appelé « éditeur assembleur » ; il figure généralement sur une disquette ou une bande magnétique), mais seulement à la fin de l'assemblage. Elles constituent ce que l'on appelle d'un nom pompeux : le code-objet. Elles sont évidemment fonction de ce qu'a réellement écrit le programmeur ; celui-ci a d'abord enclenché un processus de numérotage des lignes, ici de 100 à 200, puis frappé sur son clavier les contenus des autres colonnes situées à droite sur la figure, qui constituent le code-source : de la source est né un objet (curieux langage...).

La première ligne fixe une origine (ORG), c'est-à-dire détermine l'adresse physique de la mémoire de l'ordinateur où sera situé le début du programme lui-même. Les sigles TRS, L1, L2 sont des « labels », noms symboliques un peu analogues à des variables en Basic, librement choisis par l'auteur ; ce ne sont donc pas des mnémoniques du langage. Tout ce qui suit la ligne END TRS (fin du programme TRS) a été également calculé par l'éditeur assembleur, sans intervention humaine. On voit qu'il y a eu notamment un contrôle (« 00000 Total Errors »). Ce contrôle ne porte pas sur le contenu logique du programme, mais seulement sur la bonne conformité de son écriture, ce qui est déjà un gage correct d'exactitude. Reconnaissons que l'aspect extérieur d'un programme en assembleur peut dérouter un profane, qui n'y retrouve pas la simplicité d'une liste de Basic ou de Pascal.

Si imparfait que paraisse ce système, il s'apprend assez vite en réalité, et c'est le seul possible lorsqu'il s'agit par

LANGAGE-MACHINE ET ASSEMBLEUR

exemple de programmer des jeux graphiques rapides ou, plus généralement, lorsque les problèmes de vitesse d'exécution sont primordiaux. Ici, comme avec une calculette de poche HP ou Texas, les mémoires qui doivent accueillir telle information sont appelées effectivement par leur nom (ou par un adressage « indirect », également présent sur les calculettes haut de gamme, où le nom — l'adresse — est contenu dans une mémoire dont on vous donne le nom, comme dans un jeu de piste). C'est pour cela que l'ordinateur peut aller très vite.

Les inventeurs de l'assembleur

L'assembleur est assez ancien ; il existe depuis 1950 environ. Parmi ses inventeurs figurent des personnages prestigieux : chez Univac on peut citer Grace Hopper, officier de la Marine américaine, née en 1907 (eh oui, Mrs Hopper est du sexe féminin) et toujours en activité en 1984 avec le grade de Commodore (sic) d'après notre confrère *Time Magazine ;* John Backus, auteur de Speedcode (premier langage d'assemblage d'IBM) qui fit également parler de lui par la suite, notamment avec Fortran. (J'emprunte ces renseignements au merveilleux livre *Ainsi naquit l'informatique* de René Moreau, paru en 1982 chez Dunod.)

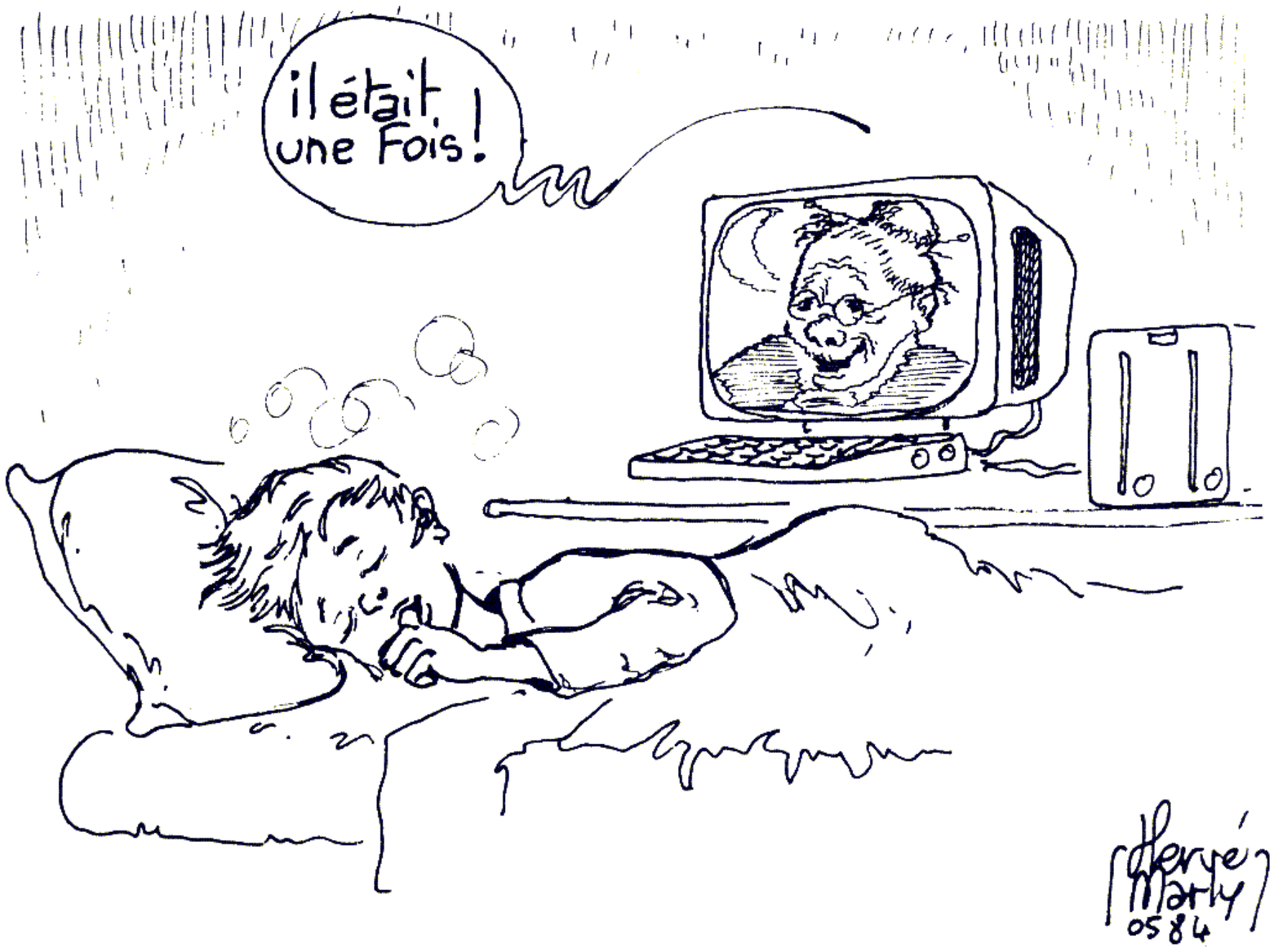
Bien entendu, il n'y a pas d'assembleur standard, mais presque un assembleur par ordinateur. Même s'ils ont des structures analogues, chaque langage-machine est très dépendant du matériel : il est normal qu'il en soit de même des assembleurs qui y donnent accès.

Prenons un exemple récent. Dès la commercialisation du PC-1500 de Sharp, son langage-machine commençait déjà à être exploré par des fanatiques astucieux — voir les articles de notre confrère *L'Ordinateur Individuel* et, bien sûr, ceux de *l'Ordinateur de poche.* Le constructeur n'avait donné aucune indication dans ses manuels ; aussi les premiers articles publiés et le premier programme éditeur assembleur conçu et commercialisé pour le PC-1500 utilisèrent-ils les codes mnémoniques du Z 80 (ceux de l'exemple donné plus haut). Lorsque Sharp s'est enfin décidé à diffuser en avril dernier son propre livre de programmation machine, il employa d'autres mnémoniques, ce qui ne simplifie rien ! Pour tout arranger, le langage-machine du PC-1251 est encore différent...

Une transparence bien opaque

Le tout premier programme assembleur ne pouvait, naturellement, qu'être écrit lui-même en langage-machine. Mais depuis ce temps, cette ineffable corvée peut être évitée et les nouveaux assembleurs sont eux-mêmes rédigés en assembleur, voire en langages évolués comme Pascal. Dans la pratique donc, seules des applications très spéciales se font encore en 0 et en 1. On peut pratiquement dire que le langage-machine est mort, même s'il continue à être le seul effectivement utilisé dans les entrailles de tous les ordinateurs du monde, grands et petits !

C'est un trait typique de l'informatique, qui n'est qu'un gigantesque jeu d'illusions où la réalité est sans cesse occultée par des masques. On dit qu'il y a « transparence » au niveau de l'utilisateur. A la programmation réelle en langage binaire, au compromis encore relativement proche des circuits que constitue le recours à l'assembleur, s'est souvent substitué l'usage des langages de haut niveau, vaguement cousins de l'anglais de base. Enfin, dans un but de vulgarisation complète, on voit de plus en plus l'utilisateur « final » renoncer à toute programmation et recourir à divers artifices (souris, écrans tactiles...) pour mettre en œuvre des logiciels tout faits et ignorer à jamais les subtilités des zéros et des uns ; qui s'en plaindrait si cela doit faire encore baisser les coûts des matériels ? Mais il restera heureusement toujours des gens assez intéressés pour créer leurs propres programmes. LIST est là pour les y aider... et leur procurer de nombreuses heures de passion !



André WARUSFEL

FAITES LE TOUR DE VOS BOUCLES FOR...NEXT

COMME tout autre langage, Basic a ses patois. Une même instruction ne réagit pas toujours de façon identique d'une machine à l'autre. Cela mérite une petite enquête. Voici quelques idées pour découvrir par vous-même la vraie nature d'un Basic étonnant : celui de votre ordinateur.

Vous connaissez la boucle FOR...TO...NEXT pour l'avoir utilisée sans compter dans tous vos programmes : elle fait partie des fonctions de tous les Basics. Nous vous proposons d'achever convenablement les présentations : préparez-vous à rencontrer un mutant et installez-vous devant votre clavier avec un crayon et un bloc de papier.

La boucle FOR...NEXT possède généralement la syntaxe suivante :

```
FOR A = B TO C STEP D
... programme à répéter (C-B/D) fois
NEXT A
```

ce qui signifie que la variable A va prendre successivement toutes les valeurs de D en D qui seront comprises dans l'intervalle (B, C). Entrez maintenant le programme qui nous servira de test :

```
10  B = 100
20  C = 10
30  D = 1
40  E = 1
100 FOR A = B TO C STEP D
200 E = E + 1
300 NEXT A
400 PRINT A ; B ; C ; D ; E
500 END
```

Vérifions ensuite la vraie signification de FOR-NEXT. Il peut s'agir de *faire jusqu'à ce que (condition),* ou bien encore de *tant que (condition faire).*

Si la condition n'est jamais vérifiée, la boucle sera exécutée au moins une fois dans le premier cas, et jamais dans le second.

Faites RUN pour en avoir le cœur net. Si E = 2, votre FOR-NEXT est de type « jusqu'à » ; si E = 1, c'est un « tant que ».

Mais alors, combien de boucles effectue réellement un :

```
FOR A = 1 TO 10 STEP 1
NEXT A
```

Le résultat peut être 9, 10, ou 11. Attention, la valeur de A ne représente peut-être pas le bon nombre, mais la dernière valeur calculée avant le test qui termine la boucle, c'est-à-dire 10, 11 ou 12 ! A vous de jouer :

```
10 B = 1
RUN
```

Consignez bien le résultat sur vos tablettes. Et enchaînons avec des manipulations hardies.

Le mot STEP qui désigne la valeur du « pas d'incrémentation » est facultatif : si vous l'omettez, la valeur du saut est obligatoirement 1. Certains Basics acceptent des STEP −0,5 négatifs (quand B > C) et décimaux ; vérifiez ces possibilités :

```
10 B = 20
30 D = — 0,5
RUN
```

En cas de succès, réjouissez-vous, c'est fort utile en trigonométrie. Exécutez alors une boucle entre 0 et PI :

```
FOR A = 0 TO 3,1416 STEP 0,0001
NEXT A
```

Sinon vous userez d'artifices :

```
FOR A = 0 TO 31416 (STEP 1)
NEXT A
```

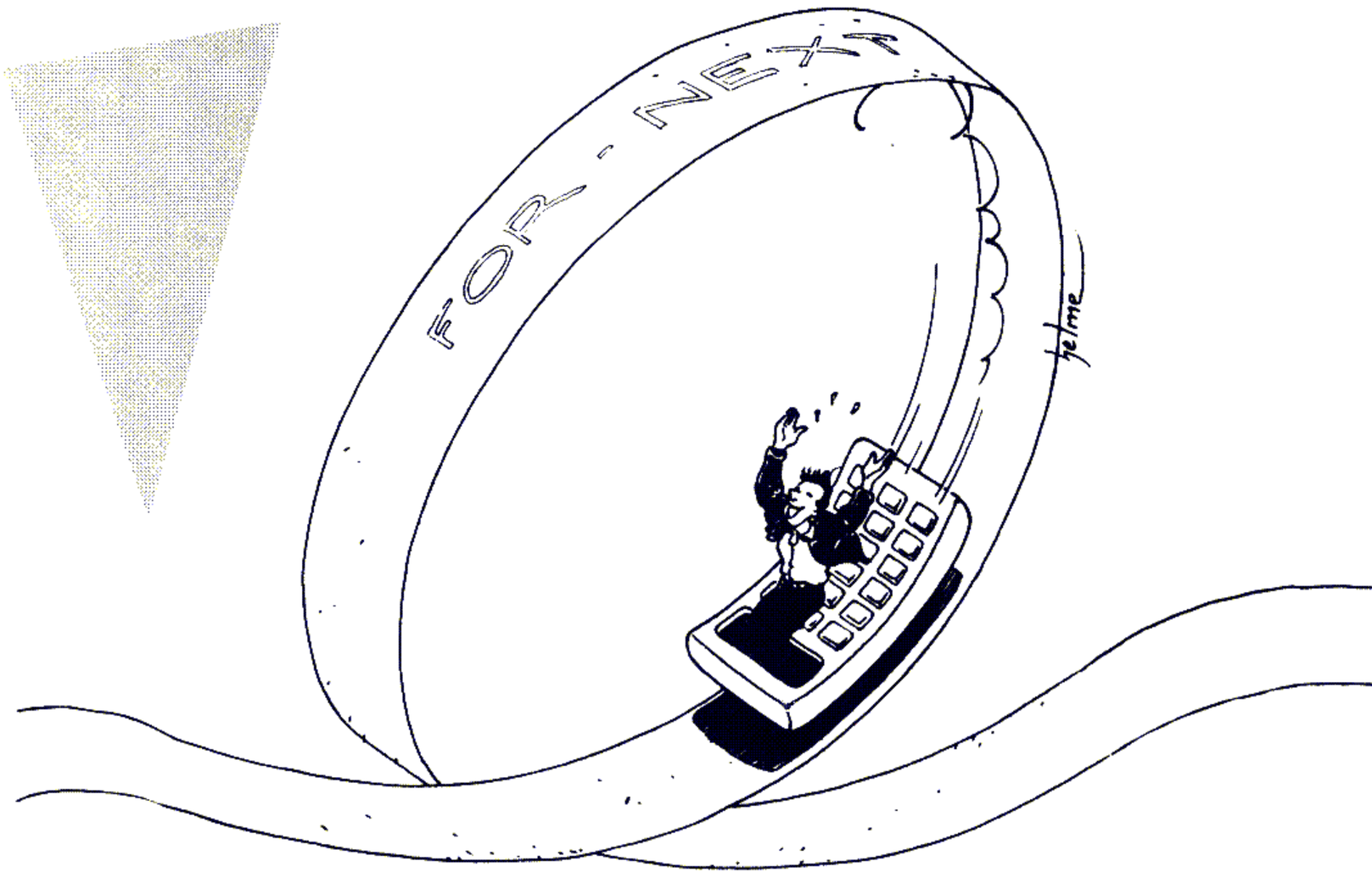
à condition que votre ordinateur accepte de calculer autant de boucles. Essayez

```
10 B = — 31416
20 C = 0
30 D = 100
RUN
```

qui devrait vous aider si vous êtes limité à C < 1000 ou C < 32000 : le plus souvent, seule la valeur d'arrivée, C, est limitée mais pas celle de départ, B.

Enfin on obtient un STEP décimal de cette façon :

```
10  B = 0
20  C = 3,1416
30  D = 0,0001
100 FOR A = B TO C
150 A = A + D — 1
RUN
```



A la ligne 150, nous modifions la valeur de A à notre guise ; en est-il de même des autres paramètres ? Pouvons-nous modifier C, la valeur d'arrivée ?

```
10  B = 1
20  C = 100
30  D = 1
150 C = 50
```

Si E est proche de 100, la modification n'a pas été prise en compte, bien que C = 50 soit confirmé. (Si E = 50, écrivez-nous pour décrire les réactions de votre matériel.)

Les boucles imbriquées

De même essayons de modifier D. La plupart des Basics « compilent » la boucle sur la variable A, c'est-à-dire que les paramètres sont lus puis codés une fois pour toutes à la première boucle. Les transformations portant sur d'autres variables que A ne modifient que rarement le nombre final de boucles. Bonne chasse !

Il reste à étudier les boucles imbriquées, elles sont du type :

```
100 FOR A = B TO C (STEP D)
200 FOR F = G TO H (STEP I)
300 NEXT F
400 NEXT A
```

Essayez le programme en ajoutant les lignes :

```
10  B = 1
20  C = 10
30  G = 1
40  H = 5
250 PRINT F ;
350 PRINT A
```

qui vous montreront ce qui se passe.

Il est parfois possible de compacter les pas 300 et 400 en NEXT F, A. (Mais on ne peut plus intercaler le PRINT.)

Plus simplement encore, un NEXT tout seul peut résumer la situation sur certaines machines :

```
10 FOR I = 1 TO 10
20 FOR J = 1 TO 4
   ...
30 IF I < 10 THEN NEXT
```

Cette disposition dite de « contrôle de boucles » est une précaution utilisée dans le cas où l'on ne connaît pas le comportement exact d'une boucle Basic. Mais vous saurez vous en dispenser, n'est-ce pas ?

Michel ARDITTI

TRI INTERNE EN SUIVEZ LA PRO

LES algorithmes de tri performants ne manquent pas. Pourtant quand il s'agit d'écrire une routine classant des données, les programmeurs sont pris au dépourvu, ou utilisent des méthodes lentes. Pour les amateurs du langage Pascal, voici un tri simple à programmer et rapide.

On appelle tri le réarrangement d'articles précédemment classés dans un ordre différent. Ce réarrangement s'effectue à partir de clefs sur lesquelles est définie une relation d'ordre permettant de comparer deux articles et donc de savoir quel est celui qui doit être placé avant l'autre.

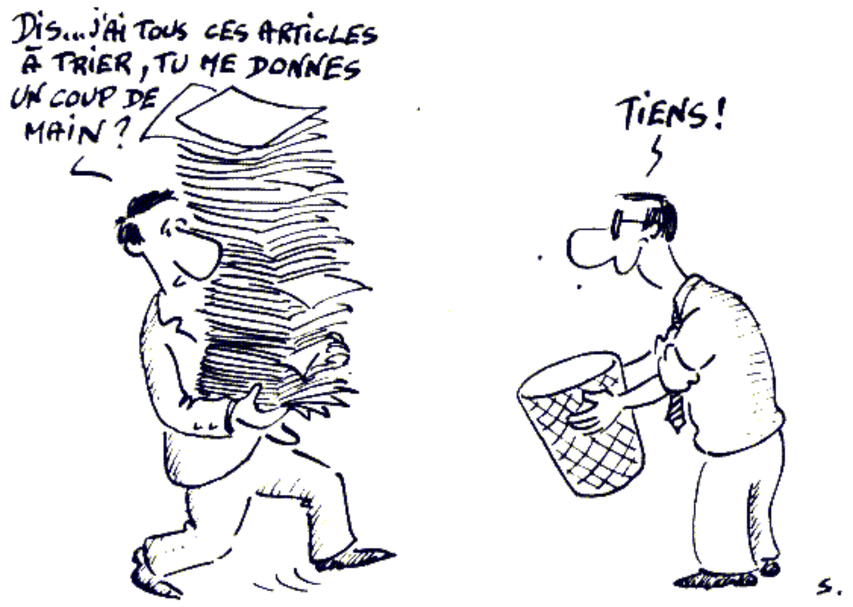
Les tris se répartissent en deux grandes catégories. Les premiers, les tris *internes,* consistent à classer des articles qui sont tous présents dans la mémoire centrale de l'ordinateur. Les tris dits *externes,* au contraire, manipulent des listes d'articles présentes par exemple sur disques ou bandes magnétiques et qui ne peuvent être logées en mémoire centrale. Un bon tri interne n'est jamais un bon tri externe car les buts ne sont pas les mêmes. Ainsi, un tri interne doit échanger le plus petit nombre d'articles, alors qu'un tri externe doit effectuer le minimum d'entrées/sorties, opérations très longues sur les périphériques (disques ou bandes magnétiques).

La procédure Pascal proposée ici est un tri interne. Elle ne doit donc être utilisée que sur des données présentes en mémoire centrale. Les articles à trier doivent avoir la structure suivante :

```
type UN_ARTICLE = record
                    CLEF      : TYPE_CLEF;
                    DONNEES   : TYPE_DONNEES
                  end ;
```

La variable CLEF, comme son nom l'indique, est ce que l'on a coutume d'appeler la clef ou le critère de tri. C'est sur cette donnée que s'effectueront les comparaisons qui permettront de savoir si un article est plus petit qu'un autre. La variable DONNEES, qui peut être une liste de variables, mémorise les autres informations de l'article. Les articles à trier sont stockés dans un tableau déclaré, par exemple :

```
type ARTICLES : array (1..MAXART) of UN_ARTICLE ;
var  ARTICLE  : ARTICLES ;
```



N PASCAL : O CÉDURE

La constante MAXART désigne le nombre maximal d'articles qui peuvent être présents dans la variable ARTICLE. Une variable NBRART indique le nombre d'articles à trier. Un tri interne ne pourra donc être utilisé que dans le cas où NBRART < = MAXART. Dans le cas contraire, on aura recours à un tri externe travaillant directement sur le fichier où sont mémorisés les articles. Ainsi, le programme de choix du tri à mettre en œuvre s'écrit :

```
if   NBRART < = MAXART
     then
       TRI__INTERNE (NBRART, ARTICLE)
     else
       TRI__EXTERNE (NBRART, 'fichier') ;
```

Comme nous le constatons, les deux paramètres de la procédure de tri interne sont d'une part le nombre d'articles effectivement présents dans le tableau et d'autre part le tableau ARTICLE, qui contient les articles à trier. Après l'appel de la procédure de tri, ARTICLE contiendra les mêmes informations que celles qui lui ont été communiquées, mais dans l'ordre des clefs. Voici la procédure :

```
Procedure TRI__INTERNE__DE__SHELL (NBRART : INTEGER ; var ARTICLE : ARTICLES) ;

var     I, DECALAGE : INTEGER ;
        ECHANGE     : BOOLEAN ;
        TEMPORAIRE : UN__ARTICLE ;
begin
   DECALAGE : = NBRART ;
   repeat
      DECALAGE : = DECALAGE div 2 ;
      repeat
         ECHANGE : = FALSE ;
         for I : = 1 to NBRART – DECALAGE do
            if ARTICLE [I]. CLEF>ARTICLE [I + DECALAGE]. CLEF
               then
                  begin
                     TEMPORAIRE : = ARTICLE [I] ;
                     ARTICLE [I] : = ARTICLE [I + DECALAGE] ;
                     ARTICLE [I + DECALAGE] : = TEMPORAIRE ;
                     ECHANGE : = TRUE
                  end
      until not ECHANGE
   until DECALAGE <= 1
end ;
```

Tri interne en Basic

En Basic, la procédure doit être légèrement modifiée. Les articles à trier peuvent se représenter sous la forme de deux tableaux appelés Clef $ et Donnee $. Un entier Nbrart indique Le nombre d'éléments utilisés dans ces deux tableaux. Le sous-programme de tri peut alors s'écrire :

```
9000 REM  TRI INTERNE DE SHELL
9010 REM  --------------------
9020 REM  Utilise les variables I, Echange, Decalage et Temp$
9030          LET Decalage=Nbrart
9040          LET Decalage=INT(Decalage/2)
9050            LET Echange=0
9060              FOR I=1 TO Nbrart-Decalage
9070                IF Clef$(I)<=Clef$(I+Decalage) THEN GOTO 9150
9080                  LET Temp$=Clef$(I)
9090                  LET Clef$(I)=Clef$(I+Decalage)
9100                  LET Clef$(I+Decalage)=Temp$
9110                  LET Temp$=Donnee$(I)
9120                  LET Donnee$(I)=Donnee$(I+Decalage)
9130                  LET Donnee$(I+Decalage)=Temp$
9140                  LET Echange=1
9150              NEXT I
9160            IF Echange=1 THEN GOTO 9050
9170          IF Decalage>1 THEN GOTO 9040
9180          RETURN
```

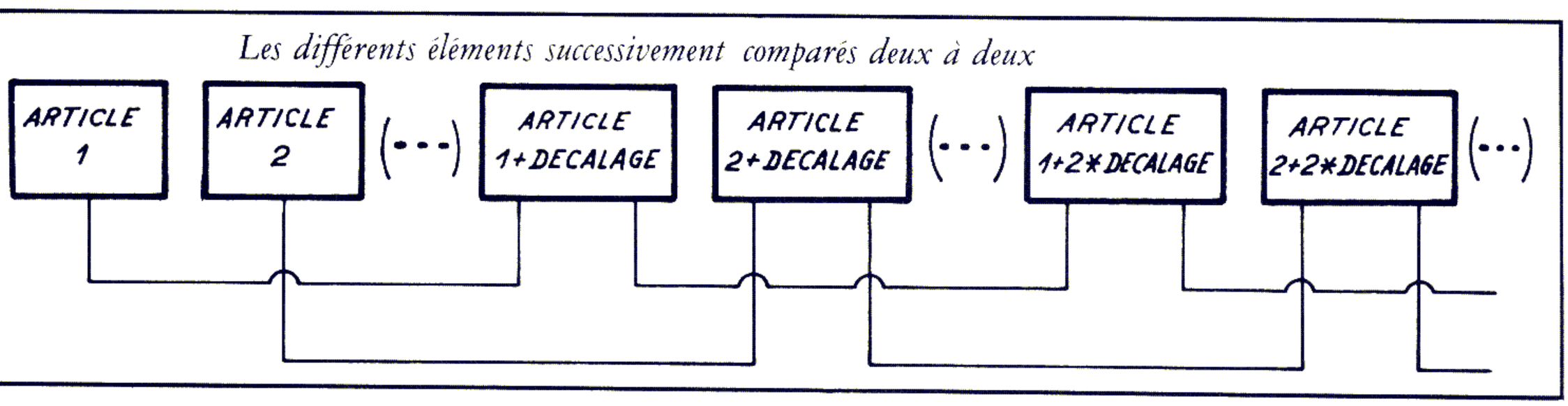
Cette méthode de tri (inventée en 1959 par D.L. Shell) est une amélioration des différentes techniques de tri par insertions. Ici, au lieu de comparer un élément par rapport à son voisin immédiat, il est comparé à celui qui est à une distance égale à DECALAGE. Cet algorithme trie des séries d'articles éloignés les uns des autres. Un schéma, au bas de la page, illustre ce procédé.

Comme nous le constatons, cette technique de tri de sous-listes permet de faire faire de grands sauts aux éléments mal placés. Ainsi, un élément de clef très faible situé à la fin du tableau « remontera » très vite par bonds égaux à la valeur de DECALAGE.

Cette procédure vous permettra de disposer à tout moment d'un tri à la fois court et rapide en exécution.

Thierry CHAMORET

Les différents éléments successivement comparés deux à deux



ORDINATEUR DE POCHE, O
TOUTE MÉDAILLE A

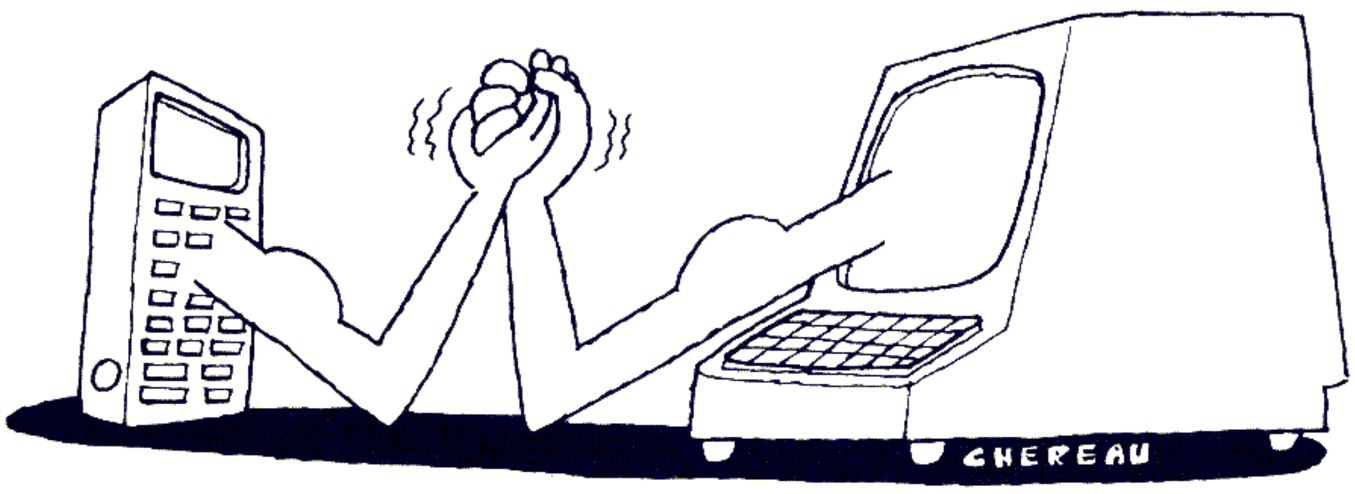
LES ordinateurs de poche et les ordinateurs de table ont des caractéristiques bien différentes. Faut-il en conclure que les uns finiront par supplanter les autres ? Ce n'est pas sûr du tout. Ces deux catégories de machines ne sont pas concurrentes, elles sont complémentaires.

Il y a quelques années, la situation était bien nette. En dehors des cas où ils constituaient des outils de calcul choisis en raison de leur faible encombrement, les ordinateurs de poche étaient les machines des débutants. On achetait une calculatrice programmable « pour voir », pour ne pas engager trop de frais dans une expérience incertaine.

La machine étant transportable, elle suivait son maître (bien souvent un élève en fait) dans tous ses déplacements, à l'école, au lycée, au bureau parfois, en vacances... Et le maître apprenait la programmation, vite et bien. Les moyens limités du matériel contraignaient alors à une démarche assez rigoureuse.

La situation qui vient d'être décrite remonte à quatre ou cinq ans. Aujourd'hui, le paysage de l'informatique individuelle a beaucoup changé. Cela a débuté avec des machines telles que le ZX 80/81, le TI 99, le Vic 20... En arrivant sur le marché à des prix nettement plus bas que les héros antiques, ces nouvelles machines étaient à la portée de l'amateur peu fortuné.

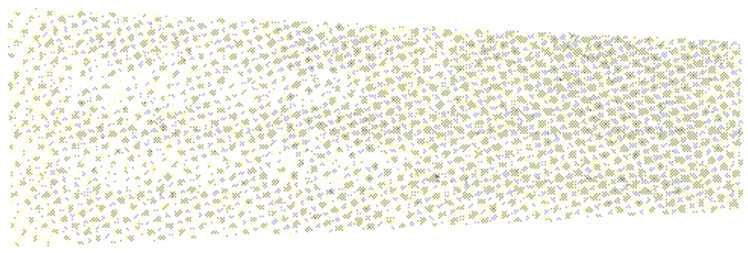
Dès lors, la calculatrice programmable n'était plus le point de passage quasiment obligatoire pour accéder à l'informatique. Elle a donc évolué vers une nouvelle spécificité. Le langage Basic a rapidement gagné du terrain et il s'est mis à ressembler de plus en plus au Basic des « grands ». Dans plusieurs cas, les machines de poche ont pris un peu de poids pour se doter d'un clavier plus confortable et d'un afficheur multiligne et l'on a vu apparaître des périphériques sophistiqués. Aujourd'hui, on peut toujours acheter un ordinateur de poche par souci d'économie mais on peut aussi choisir tel ou tel modèle parce qu'il présente des caractéristiques que l'on ne retrouve pas sur les machines de table.



Ensuite, le programmeur devenu plus expérimenté commençait généralement à rêver d'une machine plus grosse qui allait occasionner de sérieux dommages à sa tirelire. Un ordinateur de table moyen pour amateur coûtait alors le prix d'une petite machine professionnelle d'aujourd'hui. Les héros de ces temps reculés s'appelaient Pet, TRS 80 Modèle 1, Apple II, Nascom...

Penchons-nous un instant sur les différences existant entre les ordinateurs de table et de poche. Quelle influence ces différences ont-elles sur la programmation ?

Une première constatation crève les yeux : les ordinateurs de poche sont petits et, par excellence, portatifs. Des piles ou des batteries rechargeables leur assurent une totale indépendance. A l'inverse, les ordinateurs de table ne servent à rien sans la prise de courant à partir de laquelle il faut souvent tisser une belle toile d'araignée. Avantage décisif des ordinateurs de poche dans ce domaine. Même si l'ordinateur de table est portatif, on ne pourra pas l'utiliser n'importe où. En métro, en voiture, en train ou en avion, il n'est ni plus ni moins qu'un bagage inerte.



Contrepartie de sa petite taille et de son aptitude à voyager, l'ordinateur de poche est équipé d'un clavier minuscule qui rend la dactylographie assez délicate. Mais, dimensions mises à part, ces claviers sont dans l'ensemble très convenables. Or, ce n'est pas toujours le cas des ordinateurs de table bas de gamme. Ils sont grands, oui, mais la frappe sur certains claviers sensitifs ou à touches « guimauve » (les touches à la gomme) est assez pénible. Heureusement, ces deux types de claviers commencent à s'améliorer ou à disparaître tout simplement au profit de claviers mécaniques qui sont beaucoup plus confortables.

Si l'on regarde maintenant du côté de l'affichage, les modèles de table sont incontestablement supérieurs. L'écran couleur d'un moniteur ou d'un téléviseur est autrement plus attrayant, plus spectaculaire que le petit afficheur de poche. Si l'informatique « familiale » connaît un tel succès, elle le doit en grande partie à la qualité de ces effets graphiques. Sans ces effets, la plupart des programmes de jeu, par exemple, perdraient beaucoup de leur intérêt.

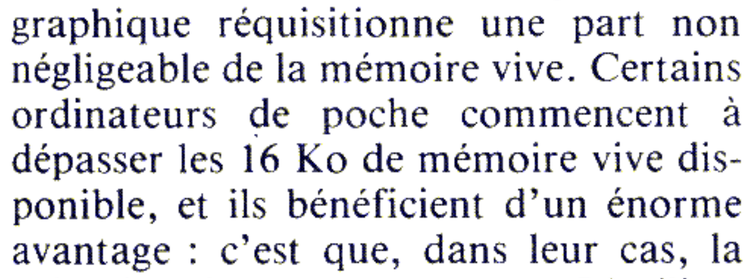
Depuis peu, les ordinateurs de poche progressent dans ce domaine avec des écrans graphiques d'une ou plusieurs

LIST - PAGE 52

E, ORDINATEUR DE TABLE, E A SON REVERS...

lignes. Mais le noir et blanc reste de rigueur, et l'on est encore loin de la finesse d'image que peut rendre un écran de télévision.

La capacité de la mémoire doit, elle aussi, être prise en considération. Pour un ordinateur familial, 64 Ko n'ont rien d'exceptionnel aujourd'hui. Encore faut-il tenir compte de l'espace éventuellement occupé par un langage résidant en mémoire morte, et savoir que l'écran graphique réquisitionne une part non négligeable de la mémoire vive. Certains ordinateurs de poche commencent à dépasser les 16 Ko de mémoire vive disponible, et ils bénéficient d'un énorme avantage : c'est que, dans leur cas, la mémoire vive est permanente. L'utilisateur ne doit pas recharger son programme (sauf s'il en change) chaque fois qu'il éteint sa machine. On trouve même des modules de mémoire vive interchangeables qui conservent programmes et données même lorsqu'ils sont retirés de la machine. C'est un système de mémoire de masse assez coûteux, mais très astucieux.



Malgré cela, c'est tout de même dans le domaine de la mémoire de masse que les ordinateurs de poche sont les moins bien lotis. Il n'existe toujours pas d'unités de disquettes prévues pour ces machines. Les techniques actuelles permettraient pourtant d'adapter des lecteurs de micro-disquettes à ce type de matériel, mais aucun constructeur ne s'est encore lancé dans cette voie. Le meilleur système en service est sans doute le lecteur de cartouches magnétiques de la HP-41, mais il est onéreux. Les autres ordinateurs de poche fonctionnent avec l'inévitable lecteur-enregistreur de cassettes (ou de micro-cassettes), et ce système est lent.

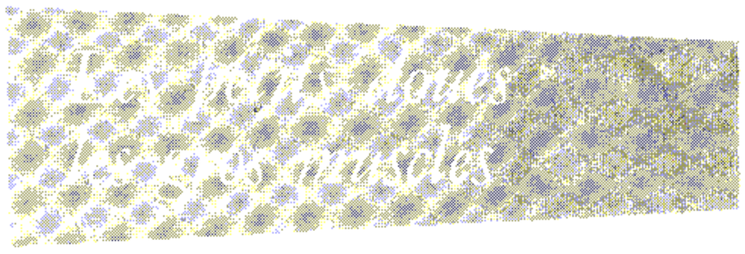
Du côté des ordinateurs à vocation familiale, on commence à voir sur certains matériels des unités de disquettes 13 cm (ou 9 cm, disquettes rigides de conception Sony). Mais les prix font hésiter car ces périphériques sont aussi chers, sinon plus, que l'unité centrale... Et pourtant, dès que l'on a goûté un peu

aux avantages des disquettes, il est difficile de retourner au lent déroulement des cassettes. La firme britannique Sinclair, avec son récent lecteur de cartouches magnétiques, pourrait bien offrir un compromis acceptable entre performance et prix.

Les langages de programmation ont longtemps départagé les deux types d'ordinateurs que nous distingons. Les premières machines de poche, c'est-à-dire les calculatrices programmables, étaient toutes dotées d'un « langage-machine spécialisé », très commode pour traiter des séquences de calculs, mais manquant de souplesse quand il s'agissait de jongler avec les structures évoluées de la programmation (tests et boucles). C'est en 1980, avec le PC-1211 de Sharp, qu'est apparu le premier Basic de poche. Le début d'une révolution. Un Basic restreint, mais bon calculateur, malgré sa lenteur.

Depuis, la situation s'est remarquablement améliorée : on trouve des poquettes qui, sans être hors de prix, sont des ordinateurs incomparablement plus puissants que les monstres des années cinquante. Plusieurs d'entre eux présentent en outre un avantage incontestable par rapport à beaucoup de machines de table actuelles. Quand il s'agit d'éditer ou de corriger un programme, ils offrent des solutions d'une qualité exceptionnelle. Les ordinateurs de table s'efforcent de suivre, mais beaucoup sont encore tributaires de commandes d'édition archaïques.

Jusqu'à ces derniers temps, si vous désiriez tâter sérieusement de l'assembleur ou d'un langage évolué autre que le Basic, il fallait recourir aux machines fixes. Plusieurs de ces machines disposent de Pascal, de Forth (un langage qui semble avoir sérieusement le vent en poupe). On a même vu récemment un langage C sur le Dragon 64. Là aussi, les choses sont en train d'évoluer : on commence à voir apparaître sur certains poquettes d'autres langages que le Basic. C'est une excellente chose. Bien sûr, il est plus intéressant de maîtriser un langage, un seul, que d'en connaître vaguement plusieurs. Mais on ne doit pas négliger l'aspect didactique de la chose, et mieux vaut avoir l'embarras du choix que de n'avoir aucun choix du tout !



Ainsi donc, pour l'instant, avantages et inconvénients dans les deux camps. Les ordinateurs de poche sont doués pour les maths, leur mémoire vive est continue, et ils sont portatifs. Du côté de l'affichage, de la vitesse d'exécution, de la mémoire de masse, ce n'est pas ce qui se fait de mieux. Si les ordinateurs de table, en revanche, sont souvent plus musclés, ils sont inutilisables sans le secours de l'EDF.

Des machines réunissant les avantages de ces deux catégories sont-elles du domaine du rêve ? Pas vraiment. Des ordinateurs comme le TRS Modèle 100 ou l'Epson PX-8 semblent ouvrir une voie médiane. Offrant des écrans multilignes, un clavier mécanique standard, une mémoire vive étendue, ils sont trop volumineux pour la poche, mais ils demeurent portatifs et autonomes. Le prix actuel de ces ordinateurs ne les destine pas à un marché grand public. Mais dans un ou deux ans qu'en sera-t-il ? Qui peut jurer que l'on ne trouvera pas des ordinateurs similaires et très bon marché ?

Mis à part ces hybrides, les poquettes d'un côté et les ordinateurs de table de l'autre sont (comme par le passé) des outils complémentaires.

Xavier de LA TULLAYE
Jean-Charles LEMASSON

INSPECTONS LE SPECTRUM

Le ZX Spectrum dit bonjour à ... LIST !

SI votre progression en informatique est liée à celle de la production britannique Sinclair, vous avez débuté avec un ZX 80, puis vous êtes passé au ZX 81. Peut-être ne connaissez-vous pas encore, ou depuis peu, son successeur, le ZX Spectrum ? Il y a beaucoup à dire sur cet ordinateur...

La mémoire morte du Spectrum est de 16 Ko et ce diable a pris des couleurs (8 en l'occurrence). Son processeur a adopté un autre circuit pour assurer la gestion de l'affichage (plus de modes SLOW ni FAST) et son cœur (l'horloge interne) bat un peu plus vite que celui du ZX 81.

Si le Spectrum paraît bien différent du petit frère qui a fait la réputation de la famille, il a néanmoins reçu la même éducation et le fond reste le même : même éditeur, même calculateur en virgule flottante, etc. En examinant la mémoire morte du Spectrum, on retrouve l'essentiel des routines. A tel point que les concepteurs du logiciel de base ont laissé une partie d'une routine du ZX 81 qui n'est pas utilisée par le Spectrum.

Toutes ces ressemblances nous conduiront à faire référence au ZX 81, et ce pour mieux faire ressortir les différences.

Le Basic du Spectrum

Depuis l'origine, Sinclair s'est nettement démarqué des autres Basics. Sans reprendre, une à une, les instructions de ce langage, nous allons en signaler certaines dont le comportement particulier peut avoir un intérêt en programmation. Commençons par FOR-NEXT.

La variable servant de compteur peut prendre n'importe quelle valeur, entière ou non. Il en est de même pour le pas d'incrément qui peut être négatif ou positif. Si la valeur du pas n'est pas précisée, elle est implicitement de +1. Mais que se passe-t-il si l'on ne peut obtenir la valeur finale du compteur à partir de la valeur initiale au moyen du pas implicitement ou explicitement précisé et à quoi va conduire l'exécution de ce programme ?

```
10 FOR N=100 TO 1
20 PRINT N
30 LET X=1234
40 LET A$="JE SUIS PASSE PAR ICI"
50 BEEP 10,10
60 NEXT N
```

Surprise ! Il s'est exécuté à une vitesse

exceptionnellement rapide, sans rien afficher, sans se faire entendre. Il a cependant terminé son exécution par le message bien connu chez nous : OK !

Si maintenant vous demandez l'affichage de la variable numérique X ou de la variable alphanumérique A$, il répond abruptement « variable non trouvée ».

Une boucle contournée

Comment expliquer cela ? Le pas était implicitement de +1 ; il était donc impossible de partir de la valeur initiale 100 pour aller à 1 en ajoutant 1 à chaque boucle. L'ensemble des lignes de la boucle a donc été totalement ignoré. En ajoutant une ligne 70 PRINT N, nous constaterons que la valeur du compteur de boucle est restée telle qu'au début du programme. La possibilité d'ignorer toutes les lignes d'une boucle est une excellente chose, surtout lorsque la valeur initiale, la valeur finale, ou le pas, sont des valeurs calculées au préalable par le programme et se traduisent donc sous forme de variables, par exemple FOR N=A TO B STEP C.

Tout comme avec la boucle FOR... NEXT, il est possible, avec IF... THEN, de sauter tout un paragraphe lorsqu'une condition n'est pas remplie. Ceci tient à la façon dont fonctionne le THEN et à la possibilité d'écrire plusieurs instructions dans une même ligne (net progrès par rapport au ZX 81). Le programme :

```
10 INPUT X
20 IF X=1 THEN LET A=1 : LET
   B=2 : LET C=3 : GOTO 40
30 LET A=100 : LET B=99 : LET
   C=98
40 PRINT A, B, C
```

permet de ne pas effectuer le test trois fois pour affecter la valeur désirée aux variables A, B et C en fonction de la valeur de X. Ainsi, quand X=1, on a A=1, B=2, C=3. Dans les autres cas, A=100, B=99 et C=98.

Nouvelles sur le Spectrum par rapport au ZX 81, les instructions IN et OUT offrent la possibilité d'une communication bidirectionnelle entre l'ordinateur et son environnement. Avec un générateur de sons, une poignée de jeu, une imprimante, ou tout autre périphérique, ces instructions d'entrée/sortie permettent de communiquer directement en Basic, sans avoir à recourir au langage-machine.

La petite imprimante ZX Printer de Sinclair est reliée au port d'entrée/sortie 251. Le moteur de l'imprimante est mis en marche quand on envoie sur ce port la valeur 0 ; il est arrêté avec la valeur 4. OUT 251,0 fait défiler le papier. OUT 251,4 arrête l'imprimante. Les lignes :

```
10 LPRINT "IMPRESSION"
20 OUT 251,0 : PAUSE 150 : OUT
   251,4
```

font avancer le papier jusqu'à ce que le message « impression » soit visible. Une valeur différente pour la durée de la PAUSE permet de moduler la longueur de papier à dérouler.

Avant de passer à d'autres charmes de ce petit monstre noir réhaussé d'un arc-en-ciel, il faut citer la puissance de la fonction VAL. Après avoir construit de toutes pièces une expression (correcte syntaxiquement en langage Sinclair) dans une chaîne de caractères, vous pouvez en demander l'évaluation.

Prenons un exemple simple pour illustrer cette puissance :

```
10 LET X=1234 : LET Y=5678 :
   LET Z=-90
20 LET A$="X + 45 * Y -" +
   STR$ PI + "/Z"
30 PRINT A$ ; "=" ; VAL A$
```

Une fois que vous aurez bien maîtrisé cette fonction, vous en apprécierez certainement l'utilité en programmation.

Le fichier d'affichage

Contrairement à ce qui se produit sur le ZX 81, l'espace de mémoire où sont stockées les informations affichées à l'écran du Spectrum n'est pas mouvant ni dépendant de la longueur du programme. Cet espace va des adresses mémoire 16384 à 23295 et comporte quatre zones dont deux pour les fichiers d'affichage :

- le fichier d'affichage des 22 lignes (ou 176 minilignes) réservées à l'utilisateur ;
- le fichier d'affichage des 2 lignes de compte rendu (ou 16 minilignes) normalement non accessibles.

Ces deux fichiers occupent 6144 octets, chaque octet servant à définir huit points d'encre (ou pixels) sur l'écran. Le point est le plus petit élément que l'on puisse afficher (tel qu'on le réalise en faisant PLOT X,Y). Il y a donc 6144 × 8 pixels, soit au total 49152 pixels.

Correspondant à ces deux fichiers, deux autres espaces de mémoire conservent les attributs de couleur des points affichés à l'écran. Cependant, il n'y a que 768 octets pour définir la couleur des 49152 pixels. Un octet de la zone des attributs définit la couleur de l'encre, du papier et le caractère « flashant » ou « brighté » d'une cellule regroupant 64 pixels.

Pour des raisons de simplification matérielle, les fichiers d'affichage des pixels d'encre n'ont pas une structure linéaire. Vous l'avez certainement remarqué lorsque vous avez chargé à partir d'une bande une copie d'écran.

Cet écran est partagé en trois parties, chacune d'elles étant elle-même partagée en huit rangées de huit minilignes.

Retrouver le bon ordre

Si l'on veut remettre en ordre un fichier d'affichage, on remarque que les adresses ne progressent pas de façon linéaire. Ce programme vous le montrera :

```
10 LET A=175
20 FOR X=16384 TO 22527 STEP
   2048
30 FOR Y=X TO X+255 STEP 32
40 FOR Z=Y TO Y+2047 STEP 256
50 PRINT#0 ; AT 1,0 ; "LIGNE
   NO:";"#" AND SGN A=-1 ;
   ABS A, "ADRESSE:" ; Z
60 FOR T=0 TO 31
70 POKE Z+T, 255
80 NEXT T
90 PAUSE 0
100 LET A=A-1
110 NEXT Z
120 NEXT Y
130 NEXT X
140 PAUSE 0
```

A l'exécution, le programme noircit une à une les 192 minilignes de l'écran en donnant le numéro de la miniligne concernée et l'adresse du premier groupe de huit points (1 octet) de cette miniligne. Les 16 minilignes du bas sont notées avec #, soit #1 à #16.

La ligne 70, en mettant à 255 chaque octet, noircit donc huit points en une fois. La variable T réitère le processus pour les 32 colonnes de 8 points.

La variable Z s'occupe d'une rangée de huit minilignes, la variable Y de huit rangées et la variable X des trois parties de l'écran.

Je vous propose une formule permettant de retrouver l'adresse dans le fichier d'affichage d'un point affiché à l'écran. Si ce point a pour abscisse X et pour ordonnée Y (0 à 255 pour X, 0 à 175 pour Y), voici ce que cela donne :

INSPECTONS LE SPECTRUM

ADRESSE (X,Y) = 18848 − 256 * Y + 2016 * INT (Y/8) + 1792 * INT ((21-INT (Y/8))/8) + INT (X/8).

Voyez ce qui se passe à l'exécution du programme suivant :

```
10 FOR N=27 TO 87 STEP 20
20 CIRCLE 127, 87, N
30 NEXT N
40 FOR Y=0 TO 175
50 LET Z=18848 − 256 * Y + 2016 *
   INT (Y/8) + 1792 * INT ((21 −
   INT (Y/8))/8)
60 FOR X=0 TO 31
70 POKE Z + X, 255 − PEEK (Z+X)
80 NEXT X
90 NEXT Y
```

La ligne 70 intervertit la couleur des points entre celle du papier et celle de l'encre.

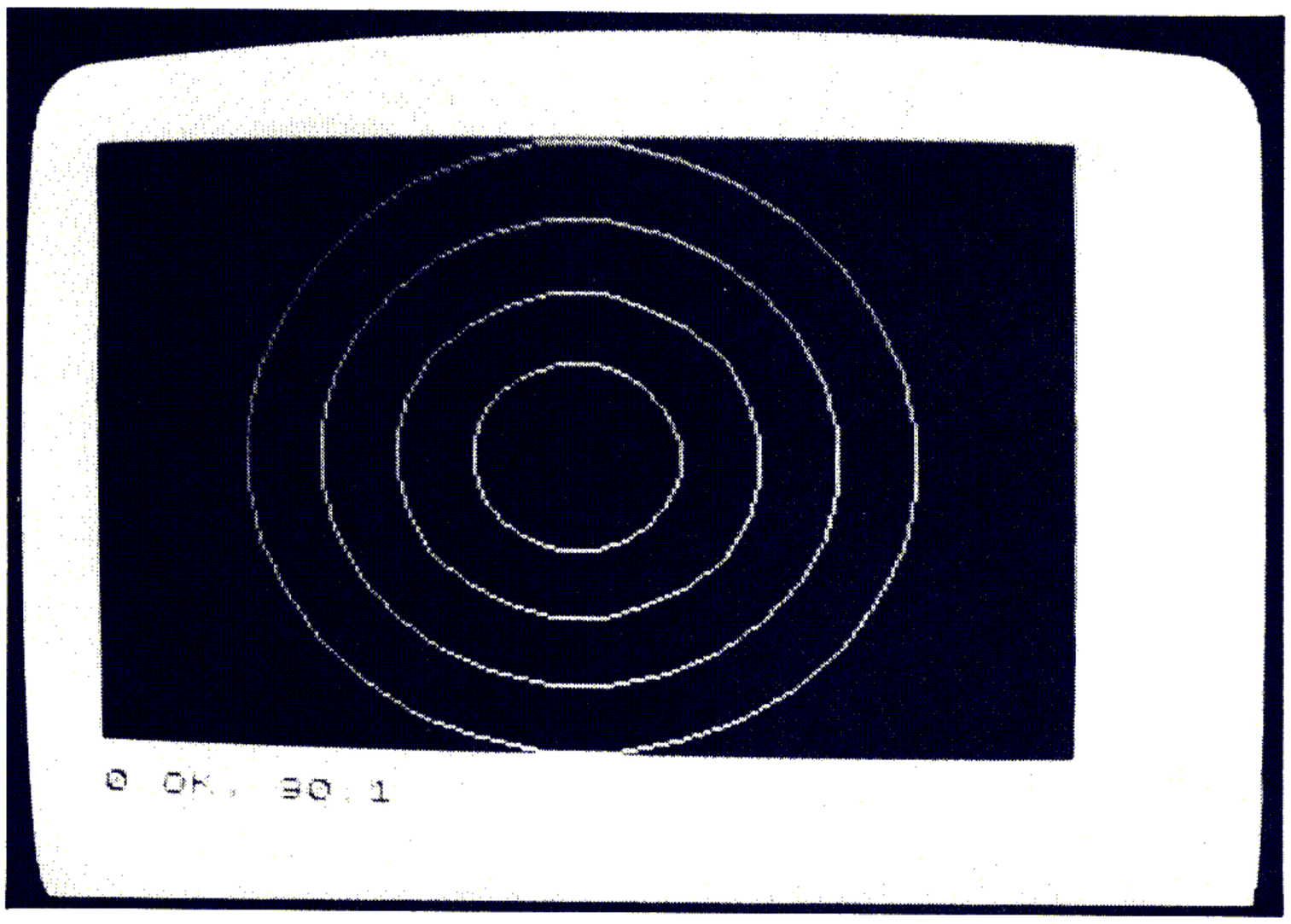
La formule permettant de retrouver les attributs de couleur d'une ligne et d'une colonne est plus simple car le stockage dans la zone des attributs se fait ligne par ligne. La voici : ATTR (ligne, colonne) = 22528 + 32 * ligne + colonne. Le programme ci-dessous fait « flasher » une cellule qui ne « flashe » pas et inversement. Il peut utiliser la formule ou la fonction ATTR.

```
10 INPUT L,C
20 LET A=22528 + 32 * L+C
30 POKE A, PEEK A+128
40 GOTO 10
```

ou bien

```
10 INPUT L,C
20 LET A=22528 + 32 * L+C
30 POKE A, ATTR (L,C) + 128
40 GOTO 10
```

Des cercles aussi ronds que possible



Les variables système du Spectrum

L'ordinateur se réserve une partie de la mémoire vive comme bloc-notes. Il mémorise ainsi certaines caractéristiques du système : interpréteur Basic, programme...

Cette zone mémoire s'étend des adresses 23552 à 23733. Nous allons reprendre ici certaines de ces variables et donner quelques exemples d'utilisation pour montrer qu'elles peuvent être utiles.

La variable système CHARS, tout d'abord, est un pointeur désignant l'emplacement de la matrice des caractères. Elle pointe vers le début de la zone de mémoire morte où est stockée la matrice des caractères dont le code va de 32 (pour l'espace) à 127 (pour le signe du copyright). Cette variable est codée sur deux octets. On obtient l'adresse qu'elle détient en faisant : PRINT PEEK (CHARS) + 256 * PEEK (CHARS + 1) soit PRINT PEEK (23606) + 256 * PEEK (23607)

Le résultat est normalement 15360. Or, il est dit dans le manuel de programmation qu'il faut ajouter 256 pour obtenir l'adresse du jeu de caractères. L'adresse en mémoire morte du début de la matrice des caractères est donc 15616. Chaque caractère est représenté sur huit octets. Ainsi, dans le cas de l'espace, les huit octets sont stockés de l'adresse 15616 à 15623.

Pour illustrer ceci, le programme ci-dessous reprend un à un les octets de la matrice d'un caractère dont vous introduisez le code et les « POKE » sur l'écran (écran dont nous venons de préciser la structure particulière).

```
10 INPUT "CODE DU CARAC-
   TERE:" ; C
20 FOR N=0 TO 7 : POKE 16384 +
   N * 256, PEEK (15616 + (C−32)
   * 8 + N) : NEXT N
30 GOTO 10
```

Traverser pour l'exemple

Maintenant, amusons-nous un peu avec cette variable système en exécutant ce programme :

```
10 CLEAR 31831
20 FOR N=0 TO 263 : POKE 31832
   + N, PEEK(15616+N) : NEXT N
30 FOR N=0 TO 207 : POKE 32096
   + N, PEEK(16136+N) : NEXT N
40 FOR N=0 TO 47 : POKE 32304
   + N, PEEK(16088+N) : NEXT N
50 FOR N=0 TO 207 : POKE 32352
   + N, PEEK(15880+N) : NEXT N
60 FOR N=0 TO 39 : POKE 32560
   + N, PEEK(16344+N) : NEXT N
70 POKE 23606,88 : POKE 23607,123
80 LIST
```

L'exécution prend un certain temps

La liste du programme après sa propre exécution

```
   10 clear 31831
   20 for N=0 to 263: poke 31832+
N,peek (15616+N): next N
   30 for N=0 to 207: poke 32096+
N,peek (16136+N): next N
   40 for N=0 to 47: poke 32304+N
,peek (16088+N): next N
   50 for N=0 to 207: poke 32352+
N,peek (15840+N): next N
   60 for N=0 to 39: poke 32560+N
,peek (16344+N): next N
   70 poke 23606,88: poke 23607,1
23
   80 list

0 ok, 80:1
```

car il y a recopie des 768 octets, correspondant aux 96 caractères, dans un espace mémoire préalablement réservé (par la ligne 10) juste en dessous des caractères graphiques (sur un Spectrum 16 Ko de mémoire vive). Cependant, on a inversé les minuscules avec les majuscules. La ligne 70 modifie la variable système pour qu'elle pointe alors sur le nouveau jeu de caractères.

Conclusion : vous pouvez, grâce à cette méthode, redéfinir l'ensemble des caractères, et non pas seulement les caractères graphiques.

D'une adresse à l'autre

Examinons maintenant d'autres variables système. Pour connaître la longueur d'un programme en octets, il suffit de faire en mode commande : PRINT PEEK 23627 − PEEK 23635 + 256 * (PEEK 23628 − PEEK 23636). On utilise ici les variables système PROG et VARS qui pointent respectivement vers l'adresse de début de la zone programme et vers l'adresse de début de la zone des variables. On voit sur la carte mémoire (page 189 du manuel de programmation Sinclair) que le programme se termine juste avant le début de la zone des variables.

De même, en utilisant les variables système VARS et E-LINE, il est possible de connaître la taille de l'espace mémoire occupé par les variables en faisant : PRINT PEEK 23641 — PEEK 23627 + 256 * (PEEK 23642 — PEEK 23628) — 1. Nous avons retranché 1 car la zone des variables se termine par un marqueur de fin (code 128).

Pour connaître la valeur que vous avez affectée à RAMTOP par l'instruction CLEAR (comme dans le programme précédent), vous utiliserez la variable système RAMTOP PRINT PEEK 23730 + 256 * PEEK 23731. RAMTOP ne représente pas le dernier octet de la mémoire vive mais le dernier octet que le système Basic peut utiliser.

Tous les octets au-delà de cette adresse peuvent être utilisés pour y loger soit du langage-machine, soit une nouvelle matrice de caractères, soit tout autre chose, mais ceci sans interférer avec l'exécution normale du Basic. Par contre, la variable P-RAMT contient l'adresse du dernier octet existant réellement sur votre Spectrum. Vous pouvez vérifier que PRINT PEEK 23730 + 256 * PEEK 23731 affichera 32767 pour un Spectrum 16 Ko et 65535 pour un Spectrum 48 Ko.

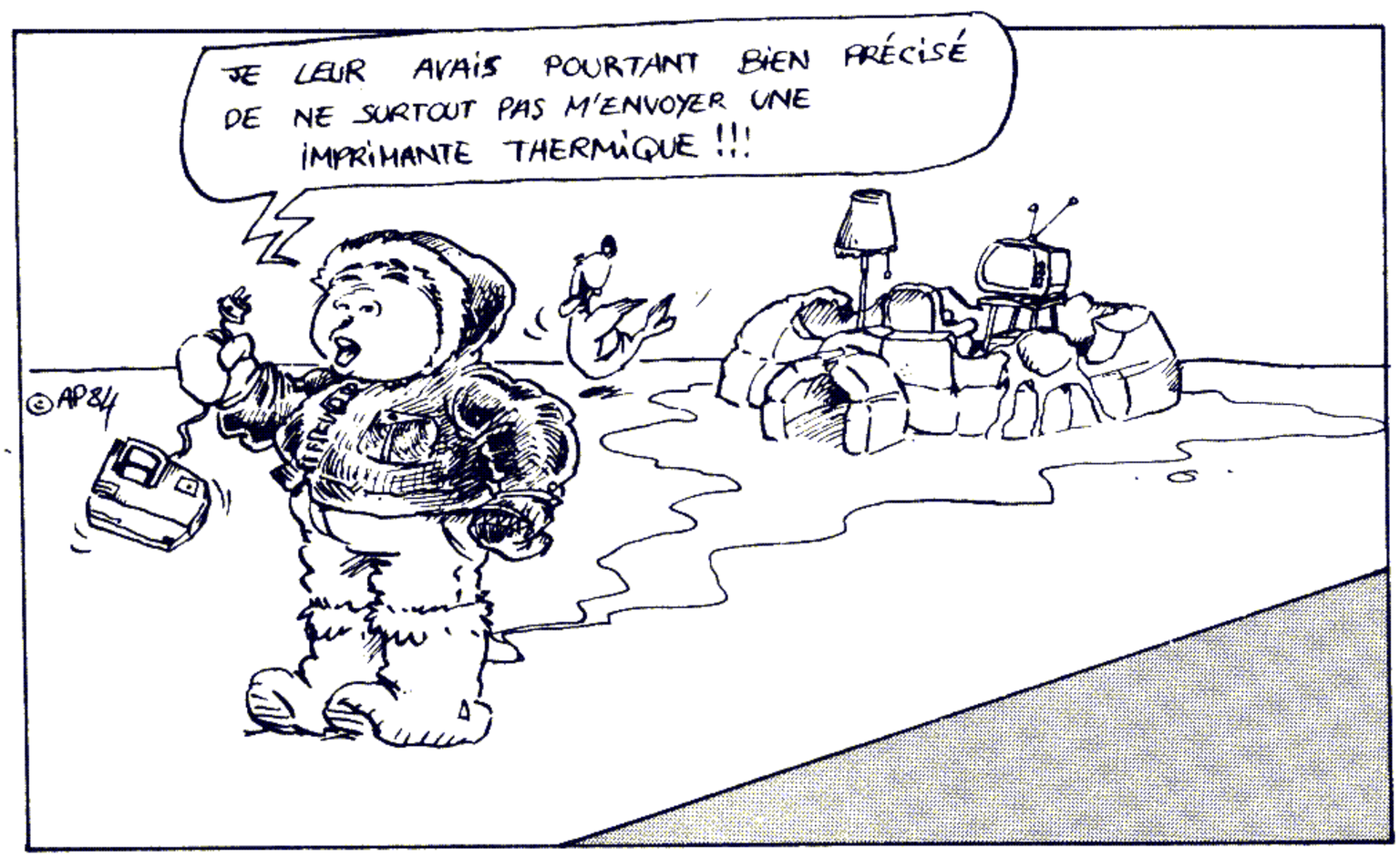
Enfin, l'adresse des caractères graphiques définis par l'utilisateur est dans la variable système UDG *(User Defined Graphics)*. Le stockage de la matrice de ces caractères est fait de la même manière que pour les caractères habituels. Cette adresse peut être obtenue soit par PRINT PEEK 23675 + 256 * PEEK 23676, soit par PRINT USR"a".

Parlons pour finir, et brièvement car nous aurons l'occasion d'y revenir, de la possibilité d'écrire en langage-machine.

La possibilité de redéfinir la partition de la mémoire, de charger des blocs d'octets à partir d'une cassette ou d'un microdrive, sont des avantages certains par rapport au bon vieux ZX 81. Le processeur, n'ayant plus à gérer l'affichage à l'écran, nous restitue quelques-uns de ses registres dont les registres indexés...

Mais ce sera pour une prochaine fois. Et, pour l'instant, continuez l'exploration de votre Spectrum. On n'est pas prêt d'en avoir fait le tour.

Benoît THONNART



Chez Duriez : 15 micros portatifs + 9 domestiques

bons pour le Service

ATARI, CANON, CASIO, COMMODORE, HEWLETT PACKARD, ORIC, SHARP, SINCLAIR, THOMPSON.

Avez-vous vu les **300 prix**

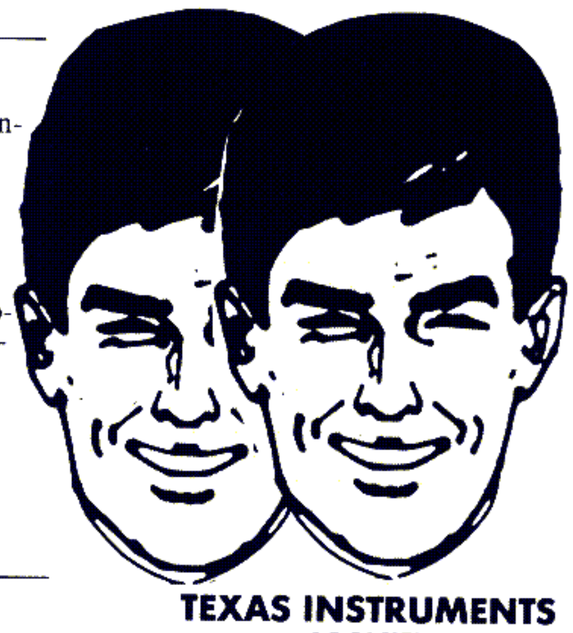
Charter© Duriez ?

AU CŒUR DU QUARTIER LATIN, Duriez vend en magasin et par poste à prix charter.©

Il publie régulièrement bancs d'essai et Catalogues, condensés de caractéristiques techniques précises, sans délayage publicitaire, complétés par des appréciations et des tests Duriez sans complaisance.

Ce banc d'essai est gratuit en magasin, ou envoyé par poste contre 3 timbres à 2 frs.

POUR CHOISIR, pensez 2 fois.
1° Les performances de l'appareil ?
2° Les performances des programmes disponibles ?
Duriez fait des sélections pour vous éviter des regrets. Vous êtes tranquille.



ATARI

600 XL Péritel	2500
800 XL Péritel	3500
Magnéto	890
Lecteur de disquette	3690
Imprimante courrier	3490
Traceur 4 couleurs	2590
Manette de jeu	120

LOGICIEL

Assembleur éditeur (cart)	778
Defender (cart)	382
Qlx (cart)	382
Galaxian (cart)	382
Dig dug (cart)	382
Graph II (disq)	154
Attaque (disq)	98
Culture physique (disq)	98
Générateur sonores (disq)	295
Limonade (disq)	98
Atlas (disq)	154
Catalogue de disq. (disq)	154
Budget familial (disq)	154
Editeur de son (disq)	295
Question réponses (disq)	205
Génerat. de caract. (disq)	295
Gestion de données (disq)	205
Agenda électron. (disq)	98
Caverne de Mars (disq)	363
Descente à ski (K7)	98
Contrôle nucléaire (K7)	154
Black jack (K7)	98
Initiation à la Progr. (K7)	258
Limonade (K7)	98
Génerat. de caract. (K7)	295
Ministre de l'énergie (K7)	205
Jeu du royaume (K7)	205
Graph (K7)	205
Cassette Graph	155
Cassette Stat	285
Disquette File	419
Livre Manuel Library	214

CANON

X07 mémoire 8K	2170
Traceur 4 coul. X710	1850
X07 + X710	3900
Extension 8K	750
Carte mém. 4K XM100	412
Carte mémoire 8K XM101	850
Cordon magnéto	65
Coupleur optique	470
Inter. RS232 + cordon	725
Cordons imp. parallèle	295
Secteur	82
Carte Fichiers	530
Carte Graphique	530
Cassette Stat	298
Cassette Graph	298
Cassette Text	298

CASIO

PB 700	1660
Traceur 4 coul. FA10	2280
PB 700 + FA10	3900
Extension 4KO R4	427
Magnéto intégré CM1	850
Interface FA4	865
Fx 702P	1095
Interface magnéto FA2	280
Imprimante FP10	610
Fx 802P	990
PB 100	675
Interface magnéto FA3	285
Imprimante FP12	600
FP 200	2990
Extension 8K	623
Cordon magnéto	85
Traceur 4 coul.	2280
Lecteur de disquettes	4430
Clavier numérique	512
Secteur	225
Cordon impri. parallèle	390
Extension CETL (ROM)	809

COMMODORE

Commodore 64 Pal	2790
Commodore 64 Secam	3650
Commodore 64 Péritel	3750
Commodore VIC 20 Pal	1550
Commodore VC 20 Secam	2100
Extension mémoire 3K	295
Extension mémoire 8K	416
Extension mémoire 16K	665

PERIPHERIQUES VIC20 et C64

Lecteur de cassettes	465
Lecteur de disque 1541	3600
Imprim. 50 cps MPS801	2690
Traceur 4 couleurs	1995
Interface RS232C	345
Manette de jeu	120
Crayon lumineux	475

LOGICIEL VIC 20

Super expender	430
Programmer's aid	350
Screen Master	415
VIC Forth	800

LOGICIEL C64

Utilitaire	
TOOL 64 (cart)	640
Master (disq)	950
64 Forth (cart)	659
Zoom Pascal (disq)	456
HES MON 64 (cart)	440
Professionnel	
HES Writer (cart)	498
Omnicalc (cart)	587
Stat 64 (cart)	490
Graph 64 (cart)	380
Multiplan (disq)	990
Vizawriter (disq)	1355
Super Base 64 (disq)	1600
Educatif	
Turtle graphic (cart)	659
Point brush (cart)	223
Sinthy 64 (K7)	326
Turtle Toyland (disq)	365
Coco (disq)	440
Jeux	
Choplifter (cart)	495
Lode Runner (cart)	495
Attack of Mutant Camel (cart)	384
Lazer Zone (cart)	328
Gridrunner (cart)	328
Rootin tootin (cart)	365
Omegarace (cart)	215
Space rescue (disq)	495
Speed Bingo (cart)	215
Clowns (cart)	215
Kickman (cart)	215
Sea Wolf (cart)	215
Jupiter Lander (cart)	215
Radar rat race (cart)	215
Echec Grand Master (K7)	305
Kong (K7)	125
Scramble (K7)	125
Motor Mania (K7)	165
M TNT (cart)	329
Benji (cart)	236
The Pit (cart)	329

EPSON

HX 20	5800
Magnéto intégré	1100
Extension 16K	1200
Modem + cordon	1755
Cassette Intext	780

HEWLETT-PACKARD

HP 11C	810
HP 15C	1235
HP 12C	1235
HP 16C	1235
HP 41C	1765
HP 41 CV	2540
HP 41 CX	2880
Lecteur de cartes	1850
Lecteur optique	1190
Imprimante 82 143	3690
Accus rechargeables	390
Chargeur	155
40 cartes magnétiques	239
Papier therm. noir (6b.)	120
Mémoire quadruple	809
Module X fonction	809
Module temps	809
Module mémoire tampon	809
HP 71	5100

PERIPHERIQUES HPIL

Module HPIL pour HP41	1348
Lecteur de cassette digit.	4770
Imprim. thermique HPIL	4770
Interface TV	3450
Interface moniteur	2290
10 mini cassettes digit.	990

OLIVETTI

M10 mémoire 8K	5990
Traceur 4 coul.	2090
Secteur	98
Cordon imp. parallèle	199
Cordon imp. RS 232	498

ORIC ATMOS

Oric Atmos 48 K	2350
Traceur 4 coul. + cordon	1800
Cordon magnéto (jack)	45
Cordon imp. parallèle	150
Modulateur noir et blanc	210
Modulateur coul. SECAM	530
Lecteur de disquettes 3''	3600
disquette 3''	69
Aigle d'or (K7)	180
Categoic (K7)	95
Xenon (K7)	120
Zorgon (K7)	120
Hobit (K7)	249
Forth (K7)	180
Anglais Assimil (K7)	440
Author (K7)	187
Oric Calc (K7)	187
Poly Fichier	180

SHARP

PC 1500 A	1980
Traceur 4 coul. CE 150	1990
PC 1500 A + CE 150	3900
Extension 8K CE 155	790
Ext. 8K Protégée CE 159	1000
Ext. 16K Protégée CE 161	1700
Interf. RS232/Parallèle	1990
Cable imp. parallèle	480
Clavier sensitif	1265
PC 1251	1215
PC 1245	750
PC 1401	1290
Interface magnéto	169
Imprimante CE 126P	790
Imp. + magnéto CE 125	1695

SINCLAIR

ZX 81	580
Extension 16K	360
Spectrum 48K Péritel	2325
Spectrum 48K Pal	1965
Interface Péritel	360

TEXAS INSTRUMENTS LOGICIEL

Jawbreaker II (cart)	250
Othello (cart)	188
Mash (cart)	250
The Attack (cart)	134
Star Trek (cart)	250
Return to Pirate I. (cart)	250
Tombstone City (cart)	188
Super Demon Attack (cart)	250
TI Invaders (cart)	188
Hopper (cart)	250
Mind Challenger (cart)	134
Burger Time (cart)	250

THOMSON

TO 7	2440
Lecteur K7	690
Extension 16K	690
Contrôleur de communic.	850
Manettes jeux et son	580
Lecteur dis. avec cont.	3800
Memo Basic	480
Imprimante thermique	2500
Imprimante impact	2500
Cordon imp. thermique	250
Cordon imp. impact	290
Interface SECAM	530
PROMOTION	
TO7 + Memo Basic + Magneto + Manettes de jeux et son + Initiation Basic	3990

LOGICIELS TO7

Basic vol. 1	195
Basic vol. 2	195
Basic vol. 3	195
Basic vol. 4	195
Basic vol. 5	195
Basic vol. 6	195
Atomium	350
Echo	260
Survivor	350
Logicod	295
Gemini	260
Crypto	295
Motus	295
Tridi	260
Trap	375
Pictor	495
Melodia	495
Sauterelle	125
Compléments et mult.	120
Carré magique	175
Horloge	125
Encadrement	120
Carotte malicieuse	175
Diététique	175
Allemand vol. 1	195
Allemand vol. 2	195
Mots croisés 1	195
Mots croisés 2	195
Budget familial	450
Carnet d'adresses	480
Gérer vos fichiers	525
Ronde des chiffres	125
Noix de coco	145
Carte de France	145
Mots en fleurs	185
Bibliothèque	490
Coktail 1	95
Coktail 2	95
Coktail 3	95
Calculatrice	360
Agenda	490
Portefeuille boursier	580
Mélimélo	437
Clé des champs	170
Quest (ROM)	325
Quest histoire géographie	66
Quest sport	66
Quest sciences	66
Signes dans l'espace	175
Système métrique	150
Pickman	120
Stock car	120
Yams	179
Loto	128
Ronde des formes	148

Je commande à Duriez : 132, Bd St-Germain, 75006 Paris.

☐ 1 Catalogue Duriez "Micros" (essais comparatifs des 20 micro-ordinateurs les plus vendus chez Duriez) contre 3 timbres à 2 F.

☐ Le(s) article(s) entouré(s) sur cette page photocopiée (ou cités ci-dessous).

Si changement de prix, je serai avisé avant expédition.

☐ Ci-joint chèque de .....

..... F y compris Port et Emballage 40 F.

☐ Je paierai à réception (Contre-Remboursement) moyennant un supplément de 30 F + 40 F Port et Emballage.

J'aurai le droit, si non satisfait, de renvoyer sous 8 jours le(s) appareil(s) modules, cassettes ou ouvrages Duriez, qui me remboursera la somme ci-dessus, (sauf suppl. 30 F du C. Rb), port et emballage.

Mes Nom, Prénoms, Adresse (N°, Rue, Code, Ville) :

.....

.....

.....

Date et Signature .....

.....

Juil. 84 Média Conseil

la puissance

DRAGON

Data Ltd.

DRAGON 64 Prix : 3 600 F T.T.C. adaptateur PERITEL en option

Microprocesseur	6809 E
Mémoire	64 K RAM 16 K ROM (41 K avec 4 pages graphiques)
Ports	RS 232. 2 manettes, 1 cassette, 1 parallèle centronic
Sorties	Connecteur 40 lignes 6809 - PAL ou Peritel/UHF (son + vidéo) - 1 moniteur composite
Clavier	53 touches machine à écrire avec autorépétition
Affichage	Noir sur vert 16 x 32 - 24 x 51 sous OS 9 curseur bleu en mode 48 K
Graphique	16 x 32 9 couleurs - 32 x 64 9 couleurs 128 x 96 2 sets 2 couleurs - 128 x 96 2 sets 4 couleurs 256 x 192 2 sets 2 couleurs
Son	Par télé ou amplificateur
Basic	Microsoft® couleur étendu

LOGICIELS D 64 K de 750 à 1 250 F T.T.C.

OS 9	Système d'exploitation multitaches, multifonctions
Pascal	Langage P implemanté complet
C	Langage compilé linkable
Dynacale	Tableur professionnel
Stylograph	Traitement de textes + dictionnaire + liaison fichiers
RMS	Base de donnée
Basic 09	Basic structuré complet modulable
Flex	Système d'exploitation le plus répandu dans le monde

EXIGEZ LA CARTE DE GARANTIE DU REVENDEUR

BON DE COMMANDE ET DEMANDE DE DOCUMENTATION

Je désire recevoir : Une documentation ☐
Dragon 32 PERITEL ☐ UHF ☐
Dragon 64 PAL/Moniteur ☐ PERITEL (650 F) ☐
Lecteur de disquette ☐

ci-joint : Chèque bancaire ☐ Mandat ☐ Date :
Contre remboursement ☐ Signature
Frais à ma charge

NOM ..... Prénom .....

Adresse .....

Code postal ..... Ville .....

GOAL COMPUTER 15, rue de Saint-Quentin, 75010 PARIS

LE BASIC DU THOMSON MO5

EN version de base, le nouvel ordinateur de Thomson est doté de 32 Ko de mémoire vive pour l'utilisateur et de 16 Ko supplémentaires pour la gestion de l'écran. C'est bien. Restent 16 Ko de mémoire morte. Et c'est là que se trouve ce qui nous intéresse le plus : le Basic, conçu par Microsoft. Bravo ! C'est un très bon Basic pour une machine de cette catégorie. Le MO5 coûte 2 390 FF ttc.

La petite famille des ordinateurs familiaux français vient de s'agrandir avec l'arrivée du dernier Thomson, le MO5. Dans sa coque plastique de couleur noire, le MO5 a l'air sobre, sérieux, peut-être même un peu austère. Le clavier AZERTY (avec minuscules accentuées) sacrifie à la mode des touches en gomme, et cela très probablement pour des raisons d'économie. A l'usage toutefois, la frappe se révèle presque aussi rapide que sur un clavier mécanique ; un bip sonore, discret, confirme chaque appui.

Toutes les touches sont grises, sauf SHIFT (jaune) et BASIC (noire). En maintenant cette dernière enfoncée pendant la frappe d'une autre touche, on affiche directement un mot clé du Basic. La plupart des touches comportent donc au moins trois indications : la fonction normale, inscrite en blanc, la fonction « shiftée », en jaune, et la fonction Basic, en noir. Cinq des touches de la rangée supérieure ont même une fonction supplémentaire. Grâce à la touche ACC, elles donnent également accès aux principaux caractères accentués du français (à, ù, é, è) ainsi qu'au c cédille (ç).

Six connexions ont été prévues. Sur le flanc droit de l'appareil, deux prises DIN permettent de brancher le crayon optique et le magnétophone à cassette (périphériques qui sont tous les deux en option). A l'arrière, on trouve une fiche Jack pour l'alimentation électrique de l'ordinateur, un cordon Péritel destiné au téléviseur et un connecteur « bus » pour de futures extensions telles que manettes de jeux ou disquettes... Une trappe enfin, sur le haut du MO5, permet d'enficher une cartouche de 4 Ko de mémoire morte. Thomson prévoit donc de diffuser des programmes sur cassettes, mais aussi en cartouches.

L'écran contient 25 lignes de 40 caractères. Si l'on programme, une ligne de Basic peut s'étendre sur 255 caractères et par conséquent occuper jusqu'à 7 lignes d'écran. L'éditeur, autrement dit l'ensemble des moyens dont on dispose pour corriger un programme, est très bien conçu : on déplace le curseur à sa guise sur la page affichée et l'on utilise les touches EFF (effacement) et INS (insertion). Un petit regret toutefois : cette dernière commande ne fait pas vraiment passer en mode d'insertion. Il faut donc appuyer autant de fois sur INS que l'on prévoit de caractères à rajouter.

On dispose encore de CNT-X qui efface tout ce qui se trouve à la droite du curseur jusqu'à la fin de la ligne de Basic et de DELETE $x-y$ qui efface toutes les lignes comprises entre la ligne n°x et la ligne n°y. Comme pour l'ordre LIST, toutes les combinaisons sont possibles :

- DELETE x (x variant de 0 à 63999) détruit la ligne n° x ;
- DELETE x— détruit toutes les lignes depuis x jusqu'à la fin du programme ;
- DELETE —x détruit le début du programme jusqu'à la ligne x ;
- DELETE—, enfin, efface tout.

Si les programmeurs peuvent regretter l'absence d'une commande renumérotant automatiquement les lignes d'un programme, ils apprécieront les classiques TRON et TROFF (exécution pas à pas).

Le Basic proprement dit occupe 16 Ko de mémoire morte et gère les 48 Ko de mémoire vive qui sont découpés en deux tranches : 32 Ko directement disponibles pour l'utilisateur et 16 Ko de mémoire graphique.

Dès la mise sous tension, le copyright de Microsoft nous indique que le Basic n'a pas été créé en France. Avec PRINT FRE (0), on apprend que 31008 octets sont à la disposition du programmeur, et FRE (X$) précise que 300

Occupation mémoire du MO5

adresse	taille	rôle
0000-1FFF	8 K	graphique couleur (en fait, 2 pages de 8 K sont à ce même emplacement, soit 16 K de mémoire vive graphique)
2000-9FFF	32 K	mémoire vive disponible pour le programme Basic et les données
A000-A7FF	2 K	prévu pour extensions (disquettes, ...)
A800-AFFF	2 K	libre
B000-BFFF	4 K	prévu pour cartouches mémoire morte
C000-FFFF	16 K	mémoire morte moniteur et Basic
Le MO5 de base comporte donc 16 K mémoire morte et 48 K mémoire vive répartis en 32 K réels et 16 K graphiques que l'on peut exploiter grâce à la fonction POINT.		

octets ont été d'office réservés pour les chaînes de caractères.

L'ordre CLEAR *x,y,z* permet de jouer avec toute cette mémoire vive : il réserve *x* octets pour les chaînes alphanumériques et 8 fois *z* octets pour la définition de *z* caractères graphiques (nous reviendrons sur ce point). Par ailleurs, le même ordre CLEAR contraint le Basic à n'utiliser que des mémoires dont l'adresse est inférieure ou égale à *y*, ce qui est très pratique si l'on veut stocker des programmes écrits en langage-machine. Quand on branche le MO5, on a $x = 300$, $y =$ &H9FFF et $z = 0$. Quant à l'ordre CLEAR utilisé sans argument, il a bien le rôle attendu d'effacer toutes les variables.

Le nouveau Thomson distingue trois types de variables. Tout d'abord, les variables entières (valeur comprise entre —32768 et 32767) qui sont stockées sur deux octets. Les variables en simple précision sont, elles, stockées sur quatre octets, trois pour la mantisse (soit sept chiffres significatifs dont seuls les six premiers sont affichés) et un pour l'exposant. Ce deuxième type de variables peut donc traiter des valeurs comprises entre 2^{127} et 2^{-128}, ou, ce qui revient au même, entre $1{,}701 \times 10^{38}$ et $2{,}938 \times 10^{-39}$.

Restent, bien entendu, les variables alphanumériques. Le tout forme un ensemble cohérent. Certains utilisateurs regretteront cependant l'absence de véritables variables en double précision : les six chiffres de mantisse seront insuffisants pour certaines applications scientifiques. On y remédiera en programmant soi-même les routines adéquates...

Le nom des variables, en revanche, nous réserve une bonne surprise. Au lieu des deux seuls caractères auxquels nous ont habitués beaucoup de Basics, nous pouvons avoir ici des variables dont les noms ont quinze caractères de long... Mais, comme le rappelle justement le manuel, gare aux mots réservés du Basic ! C'est ainsi que la variable FORMIDABLE est interdite, car elle contient l'instruction FOR, et le MO5 y perd son Basic.

Le suffixe qui termine le nom d'une variable en indique le type : un % pour les entières, un ! pour les variables en simple précision et un $ pour les chaînes de caractères. En fait, le ! n'est guère employé puisque les variables numériques sont toutes en simple précision, sauf si le programmeur en a décidé autrement. Mais, comme avec presque tous les Basics de Microsoft, on peut forcer le type des variables commençant par la même lettre. Ainsi, avec DEFINT I-K, on décide que toutes les variables commençant par les lettres I, J et K seront des variables entières. Le suffixe % devient, dans leur cas, inutile. Les instructions correspondantes sont DEFSNG et DEFSTR pour les variables en simple précision et les variables alphanumériques.

De bonnes surprises

Autre surprise agréable : les tableaux n'ont aucune limitation, sauf, bien entendu, celles qui tiennent à la mémoire vive disponible. A titre indicatif, nous avons pu créer un tableau (absurde, mais c'était pour les besoins

Liste des mots clefs Basic du MO5

Ordres sur 1 octet

	8	9	A	B	C	D	E - F
0	END	TRON	MOTOR		USR	IMP	
1	FOR	TROFF	SKIPF	NEW	ERL	MOD	
2	NEXT	DEFSTR	EXEC	SAVE	ERR	@	
3	DATA	DEFINT	BEEP	LOAD	OFF	>	
4	DIM	DEFSNG	COLOR	MERGE	THEN	=	
5	READ		LINE	OPEN	NOT	<	
6		ON	BOX	CLOSE	STEP		
7	GO	TUNE		INPEN	+		
8	RUN	ERROR	ATTRB	PEN	–		
9	IF	RESUME	DEF	PLAY	*		
A	RESTORE	AUTO	POKE	TAB(	/		
B	RETURN	DELETE	PRINT	TO	↑		
C	REM	LOCATE	CONT	SUB	AND		
D	'	CLS	LIST	FN	OR		
E	STOP	CONSOLE	CLEAR	SPC(	XOR		
F	ELSE	PSET	DOS	USING	EQV		

Fonctions sur 2 octets
(le 1er octet étant toujours FF)

	0	1	2	3-F
0	SGN	EOF	INKEY$	
1	INT	CINT	INPUT	
2	ABS		CSRLIN	
3	FRE		POINT	
4	SQR	FIX	SCREEN	
5	LOG	HEX$	POS	
6	EXP		PTRIG	
7	COS	STICK		
8	SIN	STRIG		
9	TAN	GR$		
A	PEEK	LEFT$		
B	LEN	RIGHT$		
C	STR$	MID$		
D	VAL	INSTR		
E	ASC	VARPTR		
F	CHR$	RND		

Ces tableaux ont été obtenus grâce à un programme simpliste 10 REM COUCOU, puis par des POKE et LIST successifs.

Pour le 1er tableau, par POKE & H25A8,i

Pour le 2e tableau, par POKE & H25A8,255 : POKE & H25A9,i

LE BASIC DU THOMSON MO5

de l'exemple) ne comportant pas moins de 124 dimensions : DIM A (0,0,0,0, ..., 0,0) ! La 125e dimension demeure inaccessible, une ligne de Basic n'admettant que 255 caractères.

Une petite remarque amusante au passage : en recherchant quel était le plus grand tableau à une dimension exploitable sur le MO5, je suis tombé sur une fantaisie de l'ordinateur. Avec DIM A (50000), il répond ERROR 6 (« nombre trop grand dans un calcul »). Avec DIM A (10000), il préfère ERROR 9 (« indice en dehors des limites »). Puis avec DIM A (8000), cela devient ERROR 7 (« dépassement de capacité en mémoire centrale »). Cocasse. Il suffit de le savoir.

L'instruction DIM est facultative pour les petits tableaux d'une à trois dimensions, chacune des dimensions pouvant comporter un maximum de dix éléments. Ainsi, quand le programme Basic rencontre un A (2,5,7) sans qu'il y ait eu auparavant dimensionnement de A, aucune erreur ne survient. En fait, un DIM A (10,10,10) implicite est automatiquement réalisé. On peut donc se permettre quelques étourderies.

On ne s'étonnera pas de retrouver sur le MO5 les instructions classiques des Basics Microsoft, IF-THEN-ELSE, ON-GOTO, ON-GOSUB, READ-DATA-RESTORE, et une gestion complète des erreurs, avec ON ERROR GOTO, ERR, ERL, et RESUME. Moins banal : en ce qui concerne GOSUB, le nombre d'imbrications n'est pas limité, sauf toujours par la mémoire vive. Sachant que chaque GOSUB consomme sept octets, il est possible, en théorie du moins, de programmer plus de 4000 imbrications !

A côté des opérations courantes (+, —, *, / et élévation à une puissance), on trouve également @ et MOD qui permettent respectivement d'obtenir le quotient et le reste des divisions entières. La ligne I = 100 : J = 12 : PRINT I @ J ; I MOD J affiche 8 et 4. On retrouve par ailleurs les autres fonctions mathématiques usuelles (trigonométrie, logarithme, exponentielle...). Le RND produit des nombres aléatoires et, en plus de l'instruction INT (A), le MO5 connaît CINT (A) et FIX (A). Ces trois dernières instructions retournent, dans l'ordre, la partie entière de

A, l'entier le plus proche de A, et A tronqué de sa partie décimale. Par ailleurs, en plus des opérations logiques AND, OR, NOT et XOR, on trouve IMP et EQV : implication et équivalence. Les programmeurs ne s'en plaindront certainement pas...

Les opérations sur les chaînes de caractères s'effectuent au moyen des classiques LEFT$, MID$, RIGHT$ et LEN, ASC, CHR$, VAL et STR$. Pour les entrées à partir du clavier, l'utilisateur dispose des instructions INPUT, LINEINPUT et INKEY$. Mais INPUT$ (I) attend l'appui sur I touches sans contrôle à l'écran.

Le MO5 est également musicien. Avec l'ordre PLAY, on peut tout d'abord jouer une mélodie simple en faisant, par exemple, PLAY «DOREMI »... Chaque note est codée sur deux lettres, le sol se retrouvant donc tronqué en « SO ». Quant aux altérations, on les obtient tout simplement en faisant suivre la note à jouer du signe du dièse (#) ou de la lettre b (pour bémol). Le P représente, selon les cas, une pause, un silence, un soupir. On peut également faire varier quatre autres paramètres :

- durée d'une note (entre 1 et 96) ;
- octave (cinq octaves sont disponibles, une flûte à bec n'en offre pas autant) ;
- tempo du morceau ;
- attaque (256 possibilités).

Que l'on ne s'y trompe pas : cela ne fait pas du MO5 un synthétiseur. Mais il reste que grâce à cet ordre PLAY, facile à mettre en œuvre et disponible en version de base, il est à lui seul un bel instrument de musique pour débutants.

Du côté du langage-machine, tout est permis grâce aux PEEK, POKE, EXEC, VARPTR, SAVEM et LOADM. Ces deux dernières commandes permettant la sauvegarde et la lecture sur cassettes de programmes LM.

« Après la musique, les dessins »

Autre atout du MO5 : les graphismes. La machine travaille avec 16 couleurs et sur 320 × 200 points. Sauf erreur de ma part, cela nous conduit à un total de 64000 points. Il était exclu que la couleur de chaque point soit définie individuellement. Je vous laisse le soin de calculer la mémoire vive qu'il aurait fallu pour que la chose soit possible (n'oubliez pas que la palette comporte 16 couleurs...).

En fait, sur le MO5, ces points sont regroupés par lignes de huit. La couleur de chacune de ces minilignes peut être celle du fond ou celle de l'écriture. Comptez deux octets pour définir chaque groupe de huit points... Et voilà comment 16 Ko de mémoire vive sont occupés dans le MO5. Cette mémoire d'écran est, répétons-le, entièrement indépendante des 32 Ko réservés à l'utilisateur.

La configuration générale de l'écran est déterminée au moyen de SCREEN *x,y,z* qui spécifie, dans l'ordre, la couleur de l'écriture, celle du fond et celle du pourtour de l'écran.

Lorsque l'interface d'incrustation du MO5 sera disponible, l'ordre SCREEN

permettra aussi de définir, sur l'écran d'un téléviseur (et même pendant la réception d'une émission) un rectangle réservé à l'ordinateur. Imaginez qu'une chaîne de télévision diffuse alors une série de cours d'initiation à l'informatique. Imaginez... le professeur en haut du petit écran et les exercices de chacun dans une petite fenêtre du téléviseur.

Si l'on écrit maintenant — en utilisant les parenthèses — SCREEN (*x,y*), cela prend une tout autre signification que le SCREEN *x,y* (sans parenthèses) décrit plus haut : l'ordinateur retourne en effet le code ASCII du caractère affiché en colonne *x* et en ligne *y*. Selon les cas, SCREEN sera donc commande ou fonction. Il aurait mieux valu éviter le risque de confusion. Dommage.

L'ordre CONSOLE *x,y,z,t* permet de définir à sa guise la fenêtre d'écriture.

Les périphériques du MO5 sans la télévision !

On peut ainsi décider que l'affichage ne s'effectuera plus qu'entre les lignes *x* et *y*. Le paramètre *z* permet de changer les couleurs courantes. Quant à *t*, s'il vaut zéro, c'est le déroulement normal, assez brutal en fait. Si *t* vaut 1, le déroulement s'effectue huit fois plus lentement, en mode graphique, bien plus régulier et plus agréable. Si *t* vaut 2 enfin, il n'y a plus de déroulement à l'affichage : lorsque l'écran est plein, l'affichage recommence en haut de l'écran en prenant la place de ce qui s'y trouve déjà inscrit.

Avec ATTRB, on change la taille des caractères : double largeur, double hauteur ou les deux à la fois. L'affichage des messages et des résultats passe par l'inévitable PRINT et ses acolytes TAB, SPC, USING... On dispose en outre de CSRLIN et de POS qui retournent la ligne et la colonne où se trouve le curseur. On peut, inversement, choisir la position du curseur et décider s'il sera clignotant ou invisible : c'est la fonction de LOCATE.

Pour définir ses propres caractères, on utilisera DEFGR$ (i) après avoir demandé un CLEAR,, i+1. En effet, le MO5 ne s'est réservé que les 128 premiers codes ASCII. Les 128 suivants sont disponibles à condition qu'ils aient été définis par GR$ (i) ou CHR$ (128+i). Chaque définition d'un nouveau caractère consomme huit octets de mémoire vive.

Les quatre ordres PSET, LINE, BOX et BOXF permettent d'allumer un point, de tracer une ligne, de dessiner le contour d'un rectangle ou de le remplir. On peut même choisir une autre couleur que celle qui a été définie avec SCREEN sans pour autant affecter la couleur d'écriture des caractères. Enfin, et cela n'est pas à négliger, il est possible de tester l'état d'un point grâce à POINT.

En résumé, les qualités graphiques du MO5 sont remarquables, même si l'on doit regretter l'absence de l'instruction CIRCLE.

Trois accessoires, en option, ont été prévus dans le Basic d'origine : le crayon optique, la manette de jeu et le magnétophone à cassette. Ainsi le réglage du crayon se fait par TUNE, alors qu'INPEN et INPUTPEN retournent la position de l'écran pointée par le crayon (INPEN n'attend pas l'appui sur l'écran). L'instruction PTRIG détecte si la pointe du crayon est enfoncée ou non. L'ensemble paraît fiable et très précis.

Concernant la manette de jeu, STICK retourne sa position (de 0 à 8) et STRIG teste si la touche « action » est enfoncée. Il s'agit des mêmes manettes que celles du TO 7, le contrôleur se branchant sur le bus, à l'arrière de l'ordinateur.

Décollement au bon Basic

Enfin, ce que Thomson nomme un LEP est un lecteur-enregistreur de programmes, autrement dit un magnétophone dédié à cet usage spécifique. Il ne fonctionne qu'avec le MO5 et il est le seul, apparemment, à convenir. Votre bon vieux magnétophone ne vous sera donc d'aucun secours. Dommage...

Outre les LOAD, SAVE, MERGE, MOTORON et MOTOROFF, on trouve un très utile SKIPF qui permet d'enjamber un ou plusieurs fichiers. A cette batterie d'instructions s'ajoutent OPEN, CLOSE, PRINT#, INPUT#, LINEINPUT# et EOF (gestion des fichiers de données).

Si l'on a enregistré un programme en lui donnant un nom (SAVE « nom », P), il se trouve protégé : les ordres tels que LIST, PEEK, POKE, EXEC deviennent inopérants.

On annonce des disquettes 5 pouces 1/4, mais de capacité modeste : 80 Ko. Le système d'exploitation des disquettes (en abrégé SED, ou DOS en anglais) apportera quelques mots supplémentaires au Basic. On en trouve la liste dans celle des mots réservés : FN, HEX$, USR et DOS.

Le MO5 est vendu accompagné d'un guide dont l'auteur n'est autre qu'André Déledicq ; vous connaissez peut-être ses articles et ses ouvrages scientifiques ? Il reste ici fidèle à sa réputation : très agréable à lire. Ce guide du MO5 est partagé en deux parties à peu près égales : « Leçons » et « Fiches de référence ». Il devrait convenir tant aux néophytes qu'aux programmeurs avertis.

En conclusion, le point fort du MO5 reste son Basic, riche et souple et très habile pour les graphiques. Son prix (2 390 FF ttc) le place en concurrent direct du ZX Spectrum. A l'essai, la vitesse d'exécution du MO5 s'avère très honorable, comme on pourra en juger aux pages 87 et 88 avec les dix tests de LIST. Ce petit ordinateur français (Basic made in U.S.A. tout de même) devrait connaître un beau succès.

Christian BOYER

PB-700

UN PROGRAMME BASIC SOUS LA LOUPE

VOICI un jeu de cartes inspiré d'un divertissement très populaire qui fait fureur auprès des auditeurs de RTL (le PokeRTL). Le programme correspondant met en œuvre tout à la fois un tableau de variables à deux dimensions, des caractères graphiques, des nombres aléatoires, un positionnement sur les quatre lignes de l'écran... Tout y passe, ou presque !

Imaginez la scène : un candidat, un animateur, un jeu de cinquante-deux cartes. On commence par tirer une carte dans le jeu, puis on demande au courageux candidat si la suivante sera supérieure ou inférieure à celle-ci, compte tenu de l'ordre des cartes : 2, 3, 4, 5, 6, 7, 8, 9, 10, Valet, Dame, Roi, As.

La carte en question est alors tirée. L'animateur ménage ses effets : "Cher Monsieur, vous avez dit supérieure au huit de carreau, et c'est... c'est... un deux de trèfle !"

Les jeux sont ouverts

Puis le jeu continue, mais cette fois, à partir de la carte qui vient d'être tirée. Dans ce cas précis, si nous avons effectivement un deux de trèfle, l'effet de surprise est réduit à néant : comme il s'agit de la plus petite carte, la suivante lui sera forcément supérieure ; et si, par hasard, elle lui était égale, le coup ne compterait pas et il faudrait en tirer une nouvelle.

Chaque fois que le candidat a parié dans le bon sens, il gagne 10 000 F, mais si par malheur il se trompe, il perd tout ce qu'il avait gagné !

Ce jeu doit beaucoup au hasard, mais le joueur dispose quand même d'un élément d'appréciation : les cartes tirées ne sont pas remises dans le jeu, ce qui permet d'évaluer les chances de sortie des grosses ou des petites valeurs, en fonction de celles déjà passées. Dans le programme correspondant à ce jeu, nous nous limiterons à 10 tirages de cartes en plus de la première, pour simplifier.

Ce programme suit le plan décrit par l'organigramme général : la séquence pari-tirage-résultats va donc revenir dix fois de suite. Il faudra créer une boucle pour elle. De même, l'opération "tirage d'une carte" va intervenir une première fois pour la carte de départ, puis revenir dix fois : pour ne pas nous répéter, nous devrons placer ce tirage dans un sous-programme.

Commençons à approfondir chacune des phases de cet organigramme.

La formation du jeu de cartes va se traduire par un tableau à deux dimensions dans la mémoire de l'appareil : dimension 1, valeur des cartes, et dimension 2, "couleur" (pique, cœur, carreau, trèfle). Comment réaliser un tel tableau ? Voyons tout d'abord ces quatre "couleurs" de cartes ; nous n'avons qu'un écran noir et blanc, mais le PB-700 possède les graphismes nécessaires pour les cartes à jouer : ils sont obtenus avec la fonction CHR$ suivie du code ASCII approprié (232 pour un pique, 233 pour un cœur, 234 pour un carreau et 235 pour un trèfle).

Quant aux valeurs de cartes, nous allons simplement les disposer sur une ligne de DATA, dans laquelle une instruction READ va venir "piocher" quatre fois de suite, avec un RESTORE chaque fois. Au final, notre jeu sera créé par ces deux boucles emboîtées (lignes 50 et 70).

N'oublions pas Restore

Nous pouvons alors commencer à écrire la première partie du programme qui va des lignes 20 à 110.

A la ligne 20, le dimensionnement des tableaux est indispensable dès lors que nous utilisons des variables qui comportent un ou deux indices. La longueur des variables E$ est ici précisée : 10 caractères sont nécessaires pour le plus long des affichages, "VALET DE...". La variable B$(A) représente la couleur de nos cartes. Comme A prend successivement les valeurs 0, 1, 2 et 3, l'expression de la ligne 60, CHR$(232+A) nous donnera bien les quatre symboles : pique, cœur, carreau ou trèfle.

A la ligne 80, nous lisons les valeurs des cartes tirées des DATA de la ligne 30 ; et à la ligne 90 nous regroupons tous ces éléments par une concaténation : valeur + "de" + couleur.

Les lignes 100 et 110 contiennent les NEXT correspondant aux FOR et un RESTORE qui, entre chaque couleur, remet le pointeur de données au début des DATA.

Supérieur ou inférieur ?
Programme pour PB-700

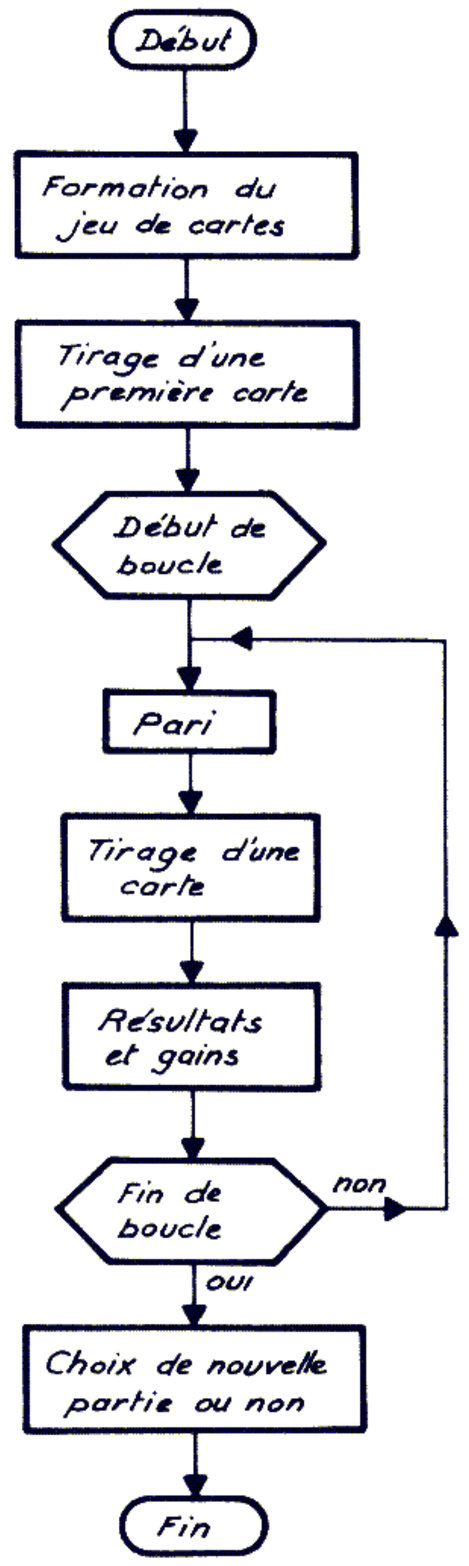
Auteur Pierrick Moigneau

Programme principal

```
 1 REM *-- Superieur ou inferieur --*
 5 CLEAR :CLS
10 PRINT "* SUPERIEUR":PRINT TAB(9);"
   ou":PRINT TAB(9);"INFERIEUR *";
20 DIM B$(3),C$(12),E$(3,12)*10,G(2)
25 REM *-- Valeur des cartes --*
30 DATA2,3,4,5,6,7,8,9,10,VALET,DAME,
   ROI,AS
40 REM *-- Couleurs des cartes --*
50 FOR A=0 TO 3
60 B$(A)=CHR$(232+A)
65 REM *-- Formation du jeu --*
70 FOR D=0 TO 12
80 READ C$(D)
90 E$(A,D)=C$(D)+" de "+B$(A)
100 NEXT D
110 RESTORE :NEXT A
120 REM *-- Tirage d une carte --*
130 I=1:CLS :GOSUB 500
140 FOR K=1 TO 10
150 REM *-- Affichage de la carte --*
155 LOCATE 4,0:PRINT NN$;"      "
160 PRINT "SUPR(S) OU INFR(I)?
                "
170 H$=INKEY$:IF H$="" THEN 170
180 REM *-- Tirage carte suivante --*
190 I=2:GOSUB 500
200 LOCATE 4,0:PRINT "
                        "
210 LOCATE 4,2:PRINT NN$;"      "
220 REM *-- Recherche du resultat --*
230 IF G(2)=G(1) THEN 190
240 IF G(2)>G(1) THEN IF H$="S" THEN 3
    00
250 IF G(2)<G(1) THEN IF H$="I" THEN 3
    00
260 REM *-- Si c est perdu... --*
270 J=0:BEEP :BEEP 1
280 PG$="PERDU!. ":GOSUB 700:GOSUB 600
    :NEXT K:GOTO 400
290 REM *-- Si c est gagne..  --*
300 J=J+10000:BEEP 1:PG$="GAGNE!. ":GO
    SUB 700:GOSUB 600:NEXT K
390 REM *-- Resultat final --*
400 CLS :PRINT "...VOUS AVEZ GAGNE:";U
    SING"######";TAB(5);J;" Fr":GOSUB
    600
410 PRINT "*VOULEZ VOUS REJOUER(Y/N)?"
    ;
440 REM *-- Retour au debut --*
450 L$=INKEY$:IF L$="" THEN 450 ELSE I
    F L$="Y" THEN 5: ELSE END
```

Sous-programmes

```
490 REM *--Ss-prgm/Tirage d une carte
    --*
500 F=INT(RND*4):G(I)=INT(RND*13):IF E
    $(F,G(I))="" THEN 500
510 NN$=E$(F,G(I))
520 REM *-- Elimination de la carte ve
    nant d etre tiree --*
530 E$(F,G(I))="":RETURN
590 REM *-- Ss-prgm/Boucle de temporis
    ation --*
600 FOR M=1 TO 100:NEXT M:RETURN
690 REM *-- Ss-prgm/Affichage des gain
    s --*
700 G(1)=G(2)
710 LOCATE 0,3:PRINT CHR$(254);PG$;CHR
    $(254);"F ";J;TAB(18);CHR$(254);
720 RETURN
```



Organigramme général

Cette portion de programme peut très bien fonctionner individuellement, ce qui permet de vérifier qu'il n'y a pas d'erreur. En tapant sur le clavier E$ suivi de deux indices, nous devons obtenir avec ENTER l'affichage d'une carte.

Par exemple, E$(3,2) ENTER doit faire apparaître 4 de♣.

Les remarques seront de mise

Passons à présent au sous-programme "tirage d'une carte" ; rappelons que la structure sous-programme va permettre de ne l'écrire qu'une fois, alors qu'il y aura onze tirages de cartes au total.

Chaque tirage se fera en déterminant deux nombres aléatoires : le premier, compris entre 0 et 3 sera la couleur de la carte tirée, et le deuxième, compris entre 0 et 12, sera sa valeur. Ces nombres seront utilisés comme indices pour localiser la carte dans le tableau.

Il faudra ensuite vérifier que cette carte n'a pas déjà été tirée. Si c'est le cas, la case de cette carte dans le tableau est vide et il faut effectuer un autre tirage. Quand la carte est présente dans le tableau, il faut l'en éliminer tout de suite, justement pour qu'elle ne puisse pas être tirée une seconde fois. Il suffira pour cela de placer une chaîne vide ('' '') dans sa case. Au préalable, comme nous avons besoin de cette carte pour la suite du jeu, nous aurons pris soin de l'affecter à une variable intermédiaire, dont le contenu changera donc à chaque tirage. Le sous-programme qui effectue toutes ces opérations est inscrit aux lignes 490 à 530.

N'oubliez pas les REM SVP ! Elles ne vous tendent pas leurs casquettes, mais elles ont bel et bien un rôle de guide... Nous devons garder à l'esprit qu'un programme doit pouvoir être compris par une autre personne que son concepteur.

Notons les nouvelles variables qui arrivent :
F = nombre aléatoire, indice de la couleur de la carte tirée ;
G(I) = nombre aléatoire, indice de la valeur de la carte tirée ;
I = indice permettant dans la suite du jeu de savoir dans quel ordre les cartes ont été tirées ;
NN$ = variable-tampon contenant la désignation de la carte tirée.

L'appel de ce sous-programme se fera pour la première carte depuis la ligne 130 : I = 1 : CLS : GOSUB 500.

Il faut maintenant afficher la carte qui vient d'être tirée, en utilisant la variable-tampon NN$. Arrive alors la question fatidique : supérieure ou inférieure ?

Cette partie du programme ne pose pas de problème de raisonnement, mais nous devons réfléchir à la disposition de nos différents éléments sur les quatre lignes de l'afficheur :
- première ligne : affichage de la carte préalablement tirée ;
- deuxième ligne : question « supérieure ou inférieure » ;
- troisième ligne : emplacement réservé à la carte qui va sortir ;
- quatrième ligne : affichage des résultats et des gains.

Pour caser la question « supérieure ou inférieure » sur une seule ligne, en précisant ce que doit répondre l'utilisateur, il faut abréger : « SUPR (S) OU INFR (I) ? » (car nous n'avons que 20 caractères par ligne). La réponse sera

une lettre seule : S ou I, ce qui permettra de recevoir celle-ci dans un INKEY$. Solution, les lignes 140 à 170 du programme.

A la ligne 140, nous trouvons le démarrage de la boucle qui effectuera 10 fois le cycle pari-tirage-résultats. Notons nos deux nouvelles variables : K, variable-compteur de boucle et H$, variable destinée à recevoir le pari « S » ou « I ».

Peu importe la couleur

Pour tirer une autre carte, il suffit de rappeler notre sous-programme qui commence à la ligne 500. Nous donnerons simplement la valeur « 2 » à l'indice I, ce qui va nous permettre de comparer G (1), valeur de la carte précédemment tirée, et G (2), valeur de la carte actuelle. Nous ne nous occuperons pas de la couleur de ces cartes, puisqu'elle n'intervient pas dans le jeu.

Nous avons donc en main les éléments suivants : deux valeurs de cartes, G (1) et G (2), et un pari, sous la forme « S » ou « I ». Si les deux cartes ont la même valeur, par exemple deux dames, le coup ne compte pas et il faut retirer une autre carte. Si G (1) est supérieure à G (2) et que nous avons parié « S », c'est gagné ; de même, si le pari est « I » avec G (1) inférieure à G (2).

En dehors de ces deux situations (cartes égales ou pari gagné), les autres cas ne peuvent correspondre qu'à des paris perdus : il est donc inutile de les faire figurer dans le programme. Tout cela nous conduit aux lignes 180 à 250.

A la ligne 200, une longue suite d'espaces sert à effacer la première carte et la question « SUPR (S) ou INFR (I) ? », de manière à rendre plus clair l'affichage. La carte qui sort est affichée par la ligne 210.

Il nous reste à voir ce qui se passe si le pari est gagné. Nous allons placer cette section du programme à partir de la ligne 300. Ainsi dans le cas d'un pari perdu, c'est la ligne venant immédiatement après la 250 qui sera exécutée.

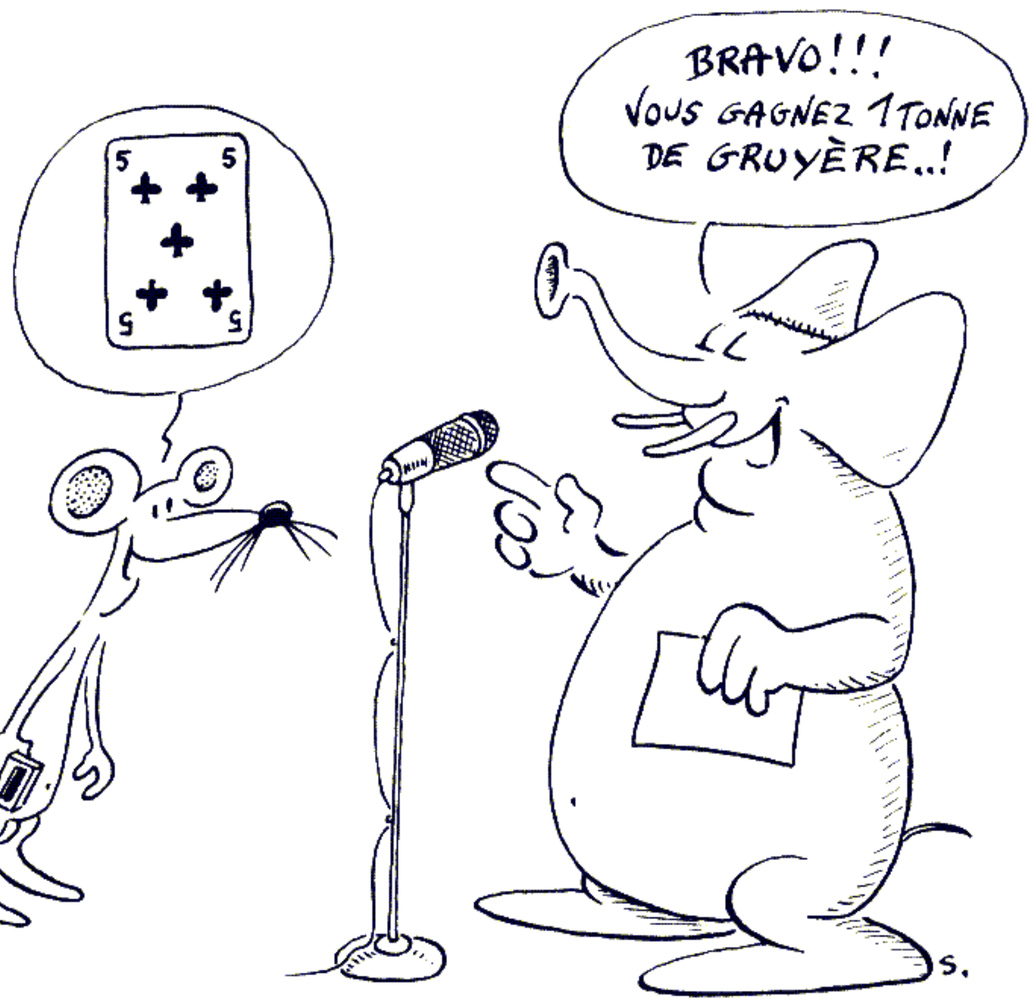
L'affichage des résultats comportera deux phases :

- affichage d'une chaîne de caractères qui sera, selon le cas, « GAGNE ! » ou « PERDU ! » ;
- modification des gains obtenus par le joueur ; nous savons que si le pari est gagné, ces gains doivent être augmentés de 10 000 F, alors que s'il est perdu, ils doivent revenir au niveau zéro.

Donc, quel que soit le résultat, nous avons un mot et un gain à afficher ; ce dernier pouvant être compris entre 0 et 100 000 F. De plus, la carte qui vient d'être tirée doit servir de référence pour le prochain pari : il est donc nécessaire que sa valeur, représentée jusque-là par G(2), soit maintenant affectée à G (1).

Nous allons placer ces trois éléments dans un sous-programme, qui interviendra dès que le mot et les gains à afficher auront été définis.

Pour que le déroulement de notre jeu soit plus clair, il sera bon de laisser s'écouler quelques instants avant le tirage suivant. Pour cela, une boucle de temporisation fera l'affaire. Nous la placerons également dans un sous-programme pour qu'elle puisse nous resservir.



Ultime fignolage : un signal sonore différent retentira en fonction des résultats.

Ce sont les lignes 260 à 300, complétées par les sous-programmes 600 et 700 qui affichent les résultats. Deux nouvelles variables apparaissent : PG$, qui contient « GAGNE ! » ou « PERDU ! » et J, qui est le montant des gains.

Le sous-programme « affichage des gains » (lignes 690 à 720) n'appelle guère de commentaires, pas plus que celui des lignes 590 et 600 qui ne contient qu'une boucle vide.

Pour rejouer : Y comme Yes, OK ?

Avons-nous terminé cette fois ? Non, car il reste à savoir ce qui va se passer quand notre boucle de 10 tirages successifs sera terminée : les lignes 280 et 300 nous enverront alors en 400, où sera affiché le résultat final. Après une petite temporisation (toujours celle de la ligne 600), la question se posera de savoir si l'on veut rejouer une partie. Si c'est le cas, il suffira de « sauter » au début du programme : IF L$= ''Y'' THEN 5 (ligne 450).

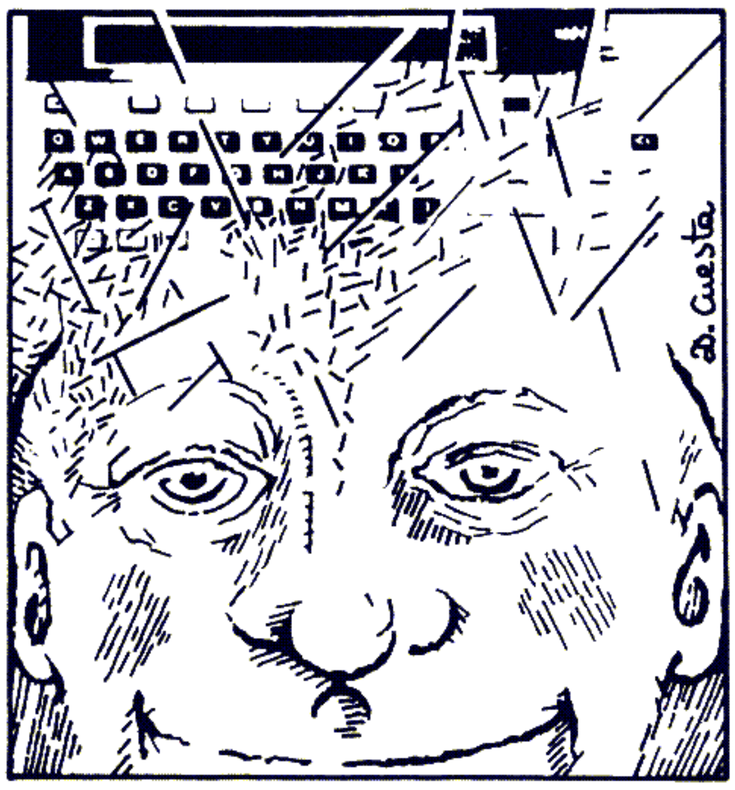
Pour gagner du temps, nous avons quelque peu brûlé les étapes lors de la phase « conception » ; en effet, on ne s'aperçoit pas tout de suite de certains petits problèmes tels que : affichages, espaces, etc... Il faut passer par une phase « finition » où, sachant que la partie principale du programme est en ordre de marche, on peut se permettre de régler les ultimes détails : confection des titres, fignolages divers, réglage des temporisations, et renumérotage des lignes, pour qu'elles soient toutes espacées de 5 en 5 ou de 10 en 10 (pour les perfectionnistes...).

Revenons à notre programme. Il est bel et bien terminé cette fois, et il ne reste plus qu'à passer à l'archivage : sauvegarde sur cassette, et classement des documents nécessaires à une utilisation ou une modification future. A savoir : une liste complète, un tableau des variables et un descriptif rapide de la façon d'utiliser le programme.

Vous avez maintenant décortiqué, tapé, « débogué », sauvegardé, archivé tout ce qui concerne ce programme ? Alors, il est temps de commencer à jouer...

Pierrick MOIGNEAU

ET LA LUMIÈRE FUT



A l'impossible nul n'est tenu ? Que nenni, voici un programme pour le PC-1500 qui liste ses congénères et est même capable d'en commander l'exécution d'un RUN bien senti. S'il faut le voir pour le croire, le comprendre est une autre histoire...

Quand on a différents programmes dans la mémoire du PC-1500, il faut quelquefois avoir la "grosse tête" pour se souvenir des emplacements de chacun d'entre eux et, notamment, de leurs numéros de ligne respectifs.

Bien sûr, quand un programme est nommé à l'aide d'une étiquette, par exemple 1110:''RESOLUTION D/EQUATION'':..., il suffit alors *en mode PRO* de taper LIST "RESOLUTION D/EQUATION" pour y accéder. De même, *en mode RUN,* taper RUN "RESOLUTION D/EQUATION" suffit à l'exécuter.

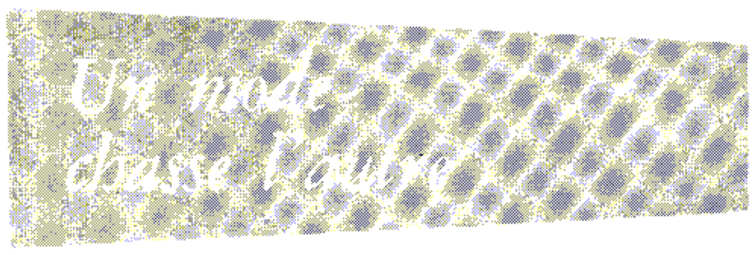
Mais tout cela est d'un classique !

L'ordinateur se débrouille très bien tout seul : lister un programme quand on tape DEF L, et même l'exécuter à DEF A. Le plus étonnant c'est que l'on utilise les instructions LIST et RUN dans le programme utilitaire alors que le manuel est formel : "LIST et RUN ne sont pas programmables". Effectivement, essayez et vous aurez une belle erreur !

Le programme reproduit ci-contre (Utilitaire, avant les POKEs) occupe seulement 140 petits octets. Il se tape normalement mais doit être modifié ensuite par exécution des instructions suivantes :

```
N = PEEK &7865*256 + PEEK &
7866
POKE N+96,0
POKE N+118,0
POKE N+133,0
```

Cette petite gymnastique donne au programme une drôle d'allure. Trois des lignes portent le numéro 0 (Utilitaire, après les POKEs).



Programmez maintenant vos propres programmes, ceux qu'il vous plaît de taper, en n'omettant pas, bien sûr, de les nommer à l'aide d'une étiquette alphabétique.

Pour lister l'un de ces programmes, en mode RUN (eh, oui), tapez son nom puis faites DEF L : l'ordinateur se retrouve en mode PRO juste sur la ligne recherchée.

Mieux encore, pour exécuter le programme voulu, tapez ce nom et DEF A. Par la suite, il est inutile de réécrire le nom pour re-lister ou ré-exécuter le programme. En outre, si vous l'oubliez, tapez DEF A ou DEF L en maintenant pressée la touche A ou L : le nom est visualisé mais, bien sûr, pas d'exécution ni de listage.

Un détail : un point d'interrogation signifie que le nom donné n'existe pas comme étiquette d'un programme en mémoire.

Voilà 9 lignes de programme dont on ne saurait plus se passer pour faire la lumière sur la mémoire de l'ordinateur.

Quant à la lumière sur le mode de fonctionnement de cet utilitaire, c'est encore une fois une histoire de pointeurs...

Frédéric BLONDIAU

Utilitaire
Programme pour PC-1500
Auteur Frédéric Blondiau
Copyright *LIST* et l'auteur

Avant les POKEs

```
1:"A"AREAD M$
2:"L"AREAD M$:ON
  ERROR GOTO 8:
  IF ASC M$LET L
  $=M$
3:WAIT 9:PRINT L
  $:RESTORE L$:
  IF ASC INKEY$
  CALL &CB9A
4:IF PEEK &7B0F=
  158THEN 6
5:RUN L$
6:POKE &704F,36
7:LIST L$
8:CURSOR 9
9:PRINT "?
```

Après les POKEs

```
1:"A"AREAD M$
2:"L"AREAD M$:ON
  ERROR GOTO 8:
  IF ASC M$LET L
  $=M$
3:WAIT 9:PRINT L
  $:RESTORE L$:
  IF ASC INKEY$
  CALL &CB9A
4:IF PEEK &7B0F=
  158THEN 6
0:RUN L$
6:POKE &704F,36
0:LIST L$
8:CURSOR 9
0:PRINT "?
```

VIVENT LES PAS, LES PARAS, VIVENT LES PARAMÈTRES !

Il y a ceux qui paramètrent
et il y a les autres.
(attribué, peut-être à tort,
à Mao Tsé-Toung)

LA simplicité a tout pour plaire.
Mais attention à ne pas se laisser séduire trop vite ! S'il est très simple d'utiliser des constantes, on a souvent avantage à leur préférer les variables.
D'ailleurs, est-ce vraiment plus compliqué ?

Paramétrer, ça veut dire utiliser une variable à la place d'une constante. Ou si vous préférez, au lieu d'écrire :

```
10 PRINT « BONJOUR »
```

écrire :

```
10 LET A$ = « BONJOUR »
20 PRINT A$
```

au lieu d'écrire :

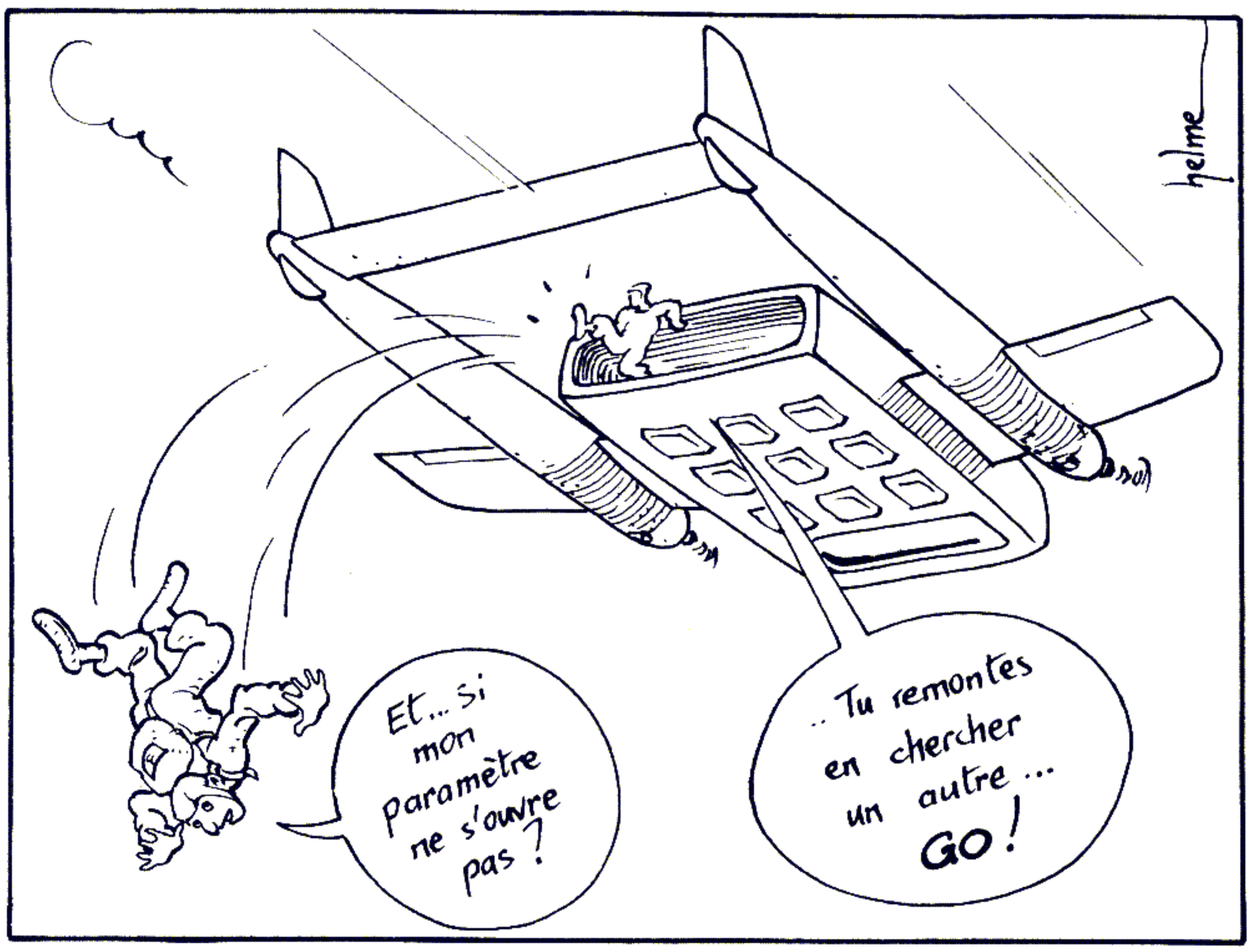
```
10 LET A=Y*40+X
```

écrire :

```
10 LET N=40
20 LET A=Y*N+X
```

Paramétrez, paramétrez, vous dis-je !

Vous entendez d'ici le débutant faire remarquer que ça ne sert à rien qu'à compliquer les choses, et puis d'abord pourquoi faire deux lignes quand une suffit, hein ? Ne vous gaussez pas du débutant. Il a raison. Il a raison si son rêve est d'écrire un programme simple et concis grâce auquel un ordinateur affiche à l'écran le message « BONJOUR » et s'en tienne là. Mais il se trouve que vos ambitions et les miennes vont parfois un brin au-delà. Et s'il s'agit d'un projet un peu plus élaboré, un peu plus complexe et un peu moins nunuche, alors, on s'aperçoit que la simplicité n'est pas toujours la meilleure solution. Cela dit, je le répète, ne riez pas trop du débutant (que j'ai moi-même été, comme vous).



Une des premières raisons de paramétrer est de gagner de la place en mémoire. Cette pauvre Alice n'a pas une très grosse tête, un ZX 81 tout nu, ça ne pèse pas lourd, et pour peu que la fantaisie vous prenne d'attaquer un « LISTinge » par un DIM A$(5000, 5000), carrément, sans prévenir, même sur une bébête beaucoup plus grosse, vous risquez d'avoir envie par la suite de ne pas gaspiller trop d'octets en programme.

Sur le ZX précité, par exemple, les constantes numériques sont de redoutables croqueuses d'octets, puisqu'elles pompent 1 octet par caractère ASCII, plus six octets de codage. La ligne : 10 LET A=16514 va donc occuper 5 octets pour frais de ligne dont 2 octets forfaitaires pour le numéro (que ce soit 1 ou 9999), +1 pour LET, +1 pour A, +1 pour =, +1 pour 1, +1 pour 6, +1 pour 5, +1 pour 1, +1 pour 4, +6 pour le codage, soit 19 octets.

Bon, 19 octets de fichus, dont 11 pour la constante numérique 16514. Une supposition que j'aie à utiliser ce nombre cinq fois dans un programme. Vous ne direz pas que j'abuse, hein ?

Cinq fois, ce n'est pas énorme. Cinq fois 11, ça fait 55. Donc, si j'utilise cinq fois la constante 16514, j'utilise 55 octets. Si je consacre une ligne à en faire une variable, je perds 19 octets, certes, mais après, 16514 est désigné par la seule lettre A qui consomme en tout et pour tout UN octet. Faites le calcul : 19+5 = 24 au lieu de 55. Ça vaut le coup, non ?

Une autre bonne raison de paramétrer les nombres est la vitesse d'exécution. On peut tenter un test avec le programme simplet suivant :

```
10 REM PROGRAMME SIMPLET
20 LET A=0
30 FOR I=1 TO 100
40 LET A=A+123456
50 NEXT I
60 PRINT « STOP »
70 STOP
```

Se munir d'un chronomètre, ou utiliser sur sa belle montre à quartz la fonction chrono dont on ne s'est pas servi depuis la semaine où elle était neuve. Frapper RUN, et appuyer sur RETURN/ENTER/NEWLINE en même temps qu'on lance le chrono. Arrêter le chrono à l'apparition du message STOP. (Si vous avez une bécane à horloge interne utilisable sur variable réservée vous pouvez utiliser le programme ci-contre, où TI est une variable réservée au temps en soixantièmes de seconde, dits aussi « jiffies ».)

Recommencer l'expérience en changeant le 100 de la ligne 30 en 1000, puis en 10 000 et consigner chaque fois les résultats avec soin et avec un crayon.

On peut alors modifier le programme :

```
10 PROGRAMME SIMPLET MODIFIE
20 LET A = 0
25 LET B = 123456
30 FOR I = 1 TO 100
40 LET A = A + B
50 NEXT I
60 PRINT « STOP »
70 STOP
```

Refaire l'expérience et comparer les temps. Allez-y, faites-le, je ne vous dis rien de plus pour l'instant. Affaire à suivre...

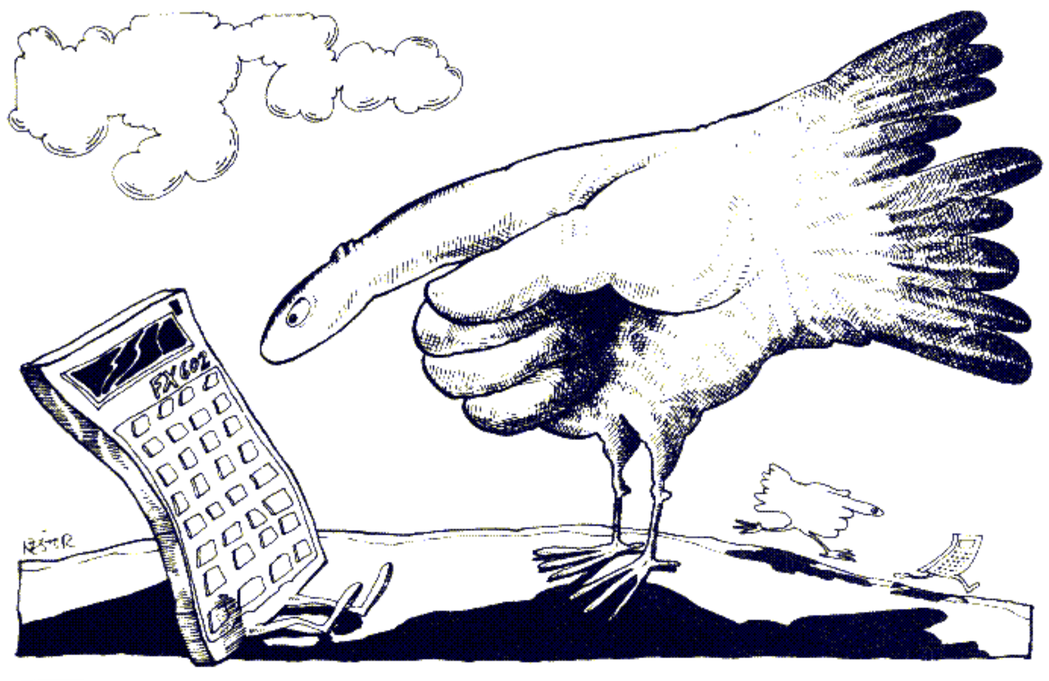
François J. BAYARD

Programme de démonstration
sur Commodore

Auteur François J. Bayard

TI est une variable réservée liée à l'horloge interne qui représente le temps écoulé depuis l'allumage de l'ordinateur, temps exprimé en « jiffies » ou soixantièmes de seconde. Ça ne doit pas être trop sorcier de l'adapter à d'autres bécanes si l'on a accès à une horloge. Dans le cas contraire, vous serez votre propre chronométreur ; il n'y a pas de honte à ça, allez allez ! Au fait, avez-vous remarqué que les messages (lignes 130-160) étaient sous forme de variables ?

```
100 rem ******** paraboucles *********
110 rem
120 nt=100:rem nombre de tours
130 nc$="nombre en constante"
140 nv$="nombre en variable"
150 b1$="boucles de 1 ligne  "
160 b3$="boucles de 3 lignes "
170 print chr$(147):rem efface l'ecran
180 rem
190 rem ---------- nombre en constante
200 rem ..................... 3 lignes
210 print nc$
220 print
230 print nt;b3$;
240 t=ti
250     for i=1 to nt
260     a=a+123456
270     next i
280 tt=ti-t:gosub 550
290 rem ...................... 1 ligne
300 print nt;b1$;
310 t=ti
320    for i=1 to nt:a=a+123456:next i
330 tt=ti-t:gosub 550
340 rem ---------- nombre en variable
350 rem ..................... 3 lignes
360 print
370 print nv$
380 print
390 print nt;b3$;
400 t=ti:b=123456
410     for i=1 to nt
420     a=a+b
430     next i
440 tt=ti-t:gosub 550
450 rem ...................... 1 ligne
460 print nt;b1$;
470 t=ti
480     for i=1 to nt:a=a+b:next i
490 tt=ti-t:gosub 550
500 print
510 rem
520 nt=nt*10:if nt<10↑7 then 210
530 end
540 rem ======== chronometrage =======
550 s=int(tt/60)
560 dd=int((tt-s*60)*60/100)
570 print mid$(str$(s),2);".";
580 print mid$(str$(dd)+"0",2,2)
590 return

ready.
```



ÉTIQUETONS LES ANGLES DE LA FX-602 P

L'UNITÉ d'angles avec laquelle vous travaillez apparaît à l'écran de la FX-602 P grâce aux indicateurs DEG, RAD et GRA.
Ces petits mots vont servir, ici, à animer... un jeu.

Sur la FX-602 P, ce jeu *sous tous les angles,* n'est pas toujours facile à pratiquer : il faut savoir intervenir au bon moment, ni trop tôt, ni trop tard.

La partie débute par l'apparition d'un compteur (il ira de 20 à 0) et d'un indicateur de mode angulaire, DEG, RAD ou GRA. Pour marquer un point, il faut appuyer sur 4, 5 ou 6 selon l'indicateur : 4 pour DEG, 5 pour RAD, 6 pour GRA. Et l'appui doit se faire rapidement, sinon le programme n'en tient pas compte.

Une astuce spécifique à la 602

Les erreurs ne font rien gagner, elles ne font rien perdre non plus. A la fin de la partie, le score est affiché suivi du record.

Le programme est simple et court. Il utilise une astuce spécifique à la FX-602 P : la présence d'un même label (ici LBL 4) deux fois dans la liste. C'était indispensable, les dix labels, 0 à 9, étant tous utilisés.

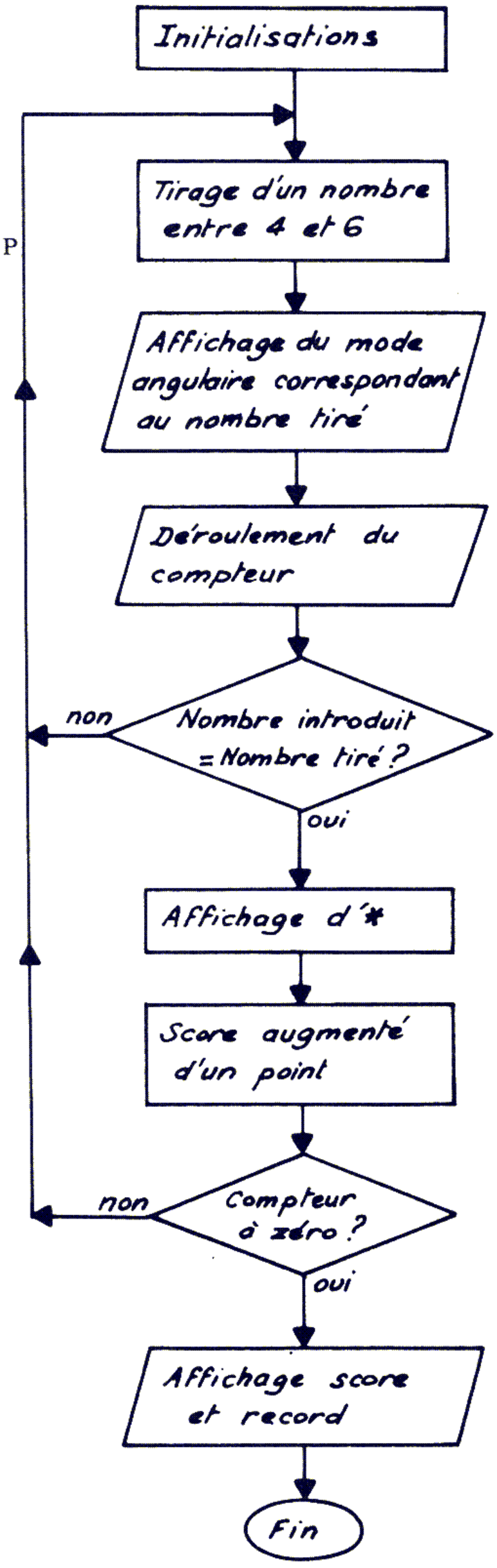
Le programme ne confond pas ces deux étiquettes. Le premier LBL 4 de la liste est atteint par l'instruction IND qui le précède. Quant au second LBL 4, il est atteint par l'instruction GOTO 4. En effet, dès que le programme rencontre cette dernière, il remonte la liste en marche arrière jusqu'à ce qu'il trouve l'étiquette LBL 4 (s'il ne la trouvait pas, il redescendrait et irait la chercher après le GOTO).

Et maintenant, entraînez-vous à surveiller les indicateurs...

Jean-Jacques ROBERT

Sous tous les angles
Programme pour FX-602 P
Auteur Jean-Jacques Robert
Copyright *LIST* et l'auteur

```
AC Min02
20 Min00
LBL1
4 MinF
RAN# x 10 = INT
Min01 x≥F GOTO0
GOTO1
LBL0
7 MinF
MR01 x≥F GOTO1
MR01 MinF IND GOTO1
LBL4
DEG
GOTO2
LBL5
RAD
GOTO2
LBL6
GRA
LBL2
MR00
PAUSE x=F GOTO3
DSZ GOTO1
LBL4
MR19 MinF
MR02 x≥F Min19
"Score: AR02  Record
: AR19 "
HLT
LBL3
4 MinF
MR01 x=F GOTO7
5 MinF
MR01 x=F GOTO8
"*"
GOTO9
LBL8
"* "
GOTO9
LBL7
"*  "
LBL9
1 M+02
DSZ GOTO1
GOTO4
```



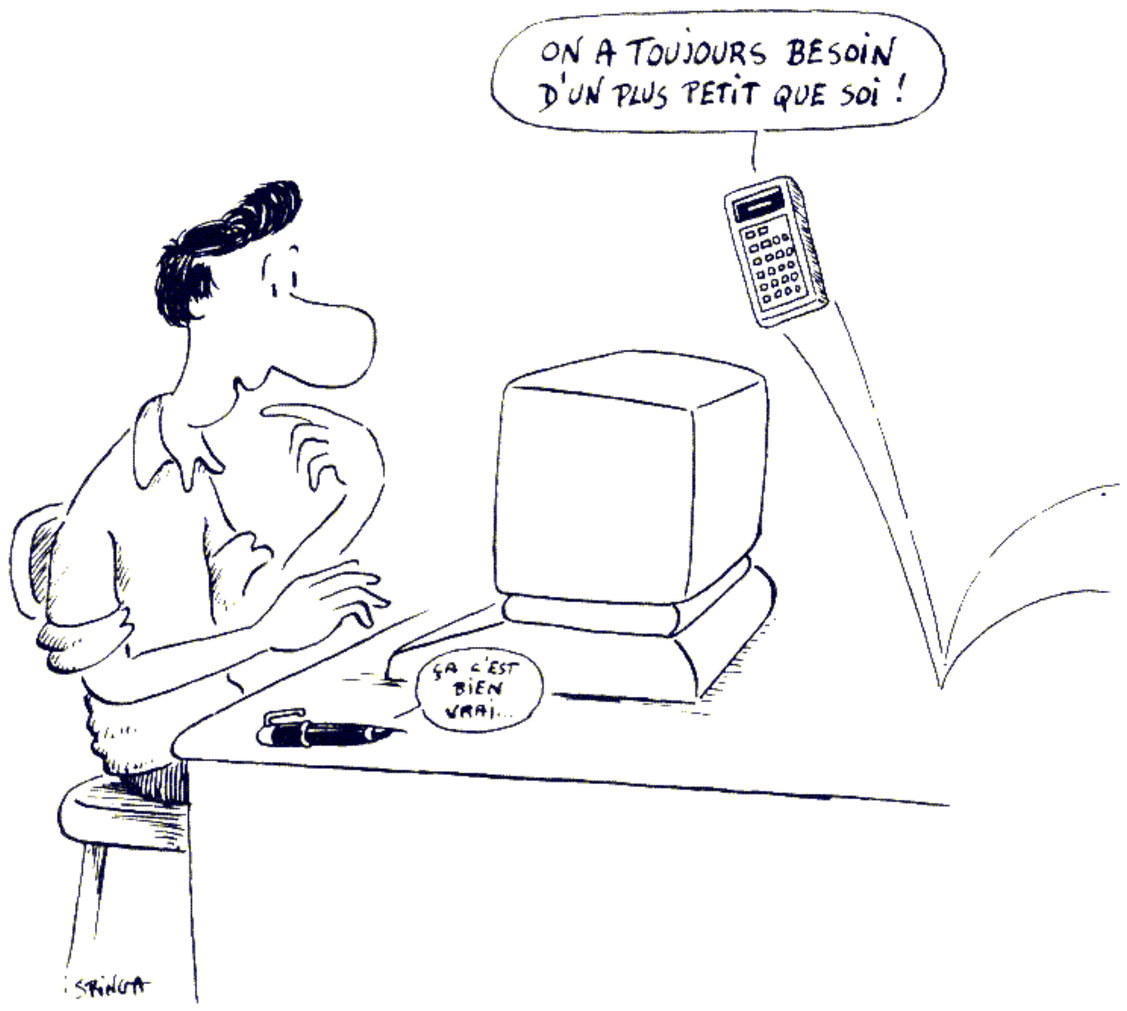
METTEZ UNE CALCULETTE DANS VOTRE ORDINATEUR

LES machines de table sont souvent très mal équipées pour les opérations au clavier. Voici quelques courts programmes qui vous faciliteront la vie si vous devez effectuer des calculs à la chaîne.

Votre "gros" ordinateur fait-il partie de ceux qui ne peuvent pas réaliser simplement un exploit tel que le calcul de 2 + 2 ? Si oui, vous devez frapper d'abord l'instruction PRINT (ou un équivalent abrégé tel que le point d'interrogation), puis 2 + 2, et presser sur la touche RETURN ou ENTER. Le résultat, 4, s'affiche. Très bien.

Très bien... Et si maintenant vous voulez savoir combien font 4 — 1 ? Vous devez repartir à zéro : PRINT 4 — 1, etc. Les calculettes font beaucoup mieux : elles permettent d'enchaîner les calculs. Avec ces petites machines, on tape simplement 2 + 2 =, et 4 s'affiche ; et si l'on demande alors — 1 =, on obtient 3 ; et ainsi de suite. C'est extrêmement commode pour tenir à jour le solde d'un compte chèque, par exemple.

Il suffit de quelques lignes de Basic pour rendre votre ordinateur aussi pratique qu'une calculette. Vous trouverez ici quelques-uns de ces courts programmes pour TRS Modèle 1, Dragon 32/64, TI-99, TO7.



On peut toujours ficeler

Si votre machine n'est pas dans le lot, vous n'aurez aucune difficulté pour adapter l'un ou l'autre de ces programmes. Mais ce n'est qu'un point de départ. Et si les quatre opérations ne vous suffisent pas, il ne tient qu'à vous d'y ajouter l'élévation à une puissance, le calcul d'une racine carrée, les fonctions trigonométriques, et tout ce qu'il vous plaira.

Vous pouvez aussi programmer la possibilité de reprendre le dernier calcul ou choisir d'opérer en notation polonaise inverse.

LIST

CALCULETTE DANS VOTRE ORDINATEUR

Calculs en chaîne
Programme pour TO7
Auteur Jacques Deconchat


```
10 REM CALCULATRICE QUATRE OPERATIONS PO
UR TO 7
20 CLS:SCREEN 0,1,2:ATTRB 0,1
30 A=0:B$="":F=1:P=10:C=0
40 A$=INKEY$:IF A$="" THEN 40
50 LOCATE C,P,0:C=C+1
60 IF A$="=" OR A$="+" OR A$="-" OR A$="
/" OR A$="*" THEN 100
70 PRINT A$
80 B$=B$+A$
90 GOTO 40
100 REM CALCULS PARTIELS ET AFFICHAGE FI
NAL
110 B=VAL(B$)
120 IF F=1 THEN A=A+B
130 IF F=2 THEN A=A-B
140 IF F=3 THEN A=A*B
150 IF F=4 THEN A=A/B
160 F=-(A$="+")-2*(A$="-")-3*(A$="*")-4*
(A$="/")
170 LOCATE C,P:PRINT A$
180 IF A$="=" THEN LOCATE C+1,P:PRINT A:
END
190 B$="":C=C+2
200 GOTO 40
```

Calculs en chaîne
Programmes pour TI-99, Dragon 32/64
et TRS 80 Modèle 1
Auteur Xavier de La Tullaye

Sur TRS 80 Modèle 1

```
40 CLS
50 DEFDBL A,C,R
60 REM *** PREMIER TERME
70 INPUT A
80 IF A = 0 THEN GOTO 70
90 REM *** QUELLE OPERATION : + - * / ?
100 INPUT "OPERATION "; B$
110 IF VAL (B$) <> 0 THEN GOTO 100
120 IF B$ = "+" LET O = 1 : GOTO 180
130 IF B$ = "-" LET O = 2 : GOTO 180
140 IF B$ = "*" LET O = 3 : GOTO 180
150 IF B$ = "/" LET O = 4 : GOTO 180
160 REM *** ON PREND SEULEMENT + - * ET /
170 GOTO 100
180 REM *** DEUXIEME TERME
190 INPUT C
200 REM *** ON REJETTE LE ZERO
210 IF C = 0 THEN GOTO 190
220 REM *** CALCUL DU RESULTAT
230 IF O=1 THEN LET R = A + C
240 IF O=2 THEN LET R = A - C
250 IF O=3 THEN LET R = A * C
260 IF O=4 THEN LET R = A / C
270 REM *** AFFICHAGE DE L'OPERATION
280 PRINT A;" ";B$;" ";C;" -----> :";R
290 REM *** R DEVIENT LE PREMIER TERME
300 LET A = R
310 REM *** ON ENCHAINE
320 GOTO 100
```

Sur TI-99

```
140 REM
150 CALL CLEAR
160 REM
200 REM  PREMIER TERME
210 INPUT A
220 IF A=0 THEN 210
230 REM
300 REM  CHOIX OPERATION
310 INPUT "OPERATION ":B$
400 REM   DEUXIEME TERME
410 INPUT C
420 IF C=0 THEN 410
430 REM
500 REM  CALCUL RESULTAT
510 IF B$="+" THEN 550
520 IF B$="-" THEN 570
530 IF B$="*" THEN 590
540 IF B$="/" THEN 610
550 R=A+C
560 GOTO 700
570 R=A-C
580 GOTO 700
590 R=A*C
600 GOTO 700
610 R=A/C
700 REM  AFFICHAGE RESULTAT
710 PRINT A;" ";B$;" ";C;" -
--->  ";R
800 REM  OPERATION SUIVANTE
810 INPUT "GARDER R COMME PR
EMIER TERME (O/N)":REP$
820 IF REP$<>"O" THEN 850
830 A=R
840 GOTO 300
850 GOTO 200
>
```

Sur Dragon 32/64

```
150 CLS : REM EFFACEMENT ECRAN
160 REM
200 REM PREMIER TERME
210 INPUT A
220 IF A=0 THEN 210
230 REM
300 REM CHOIX OPERATION
310 INPUT "OPERATION ";B$
400 REM DEUXIEME TERME
410 INPUT C
420 IF C=0 THEN 410
430 REM
500 REM CALCUL RESULTAT
510 IF B$="+" THEN 550
520 IF B$="-" THEN 560
530 IF B$="*" THEN 570
540 IF B$="/" THEN 580
550 R=A+C : GOTO 600
560 R=A-C : GOTO 600
570 R=A*C : GOTO 600
580 R=A/C
600 REM AFFICHAGE RESULTAT
610 PRINT A;" ";B$;" ";C;" --->
";R
700 REM OPERATION SUIVANTE
710 INPUT "GARDER R COMME PREMIE
R TERME (O/N)":RE$
720 IF RE$<>"O" THEN 750
730 A=R
740 GOTO 300
750 GOTO 200
OK
```

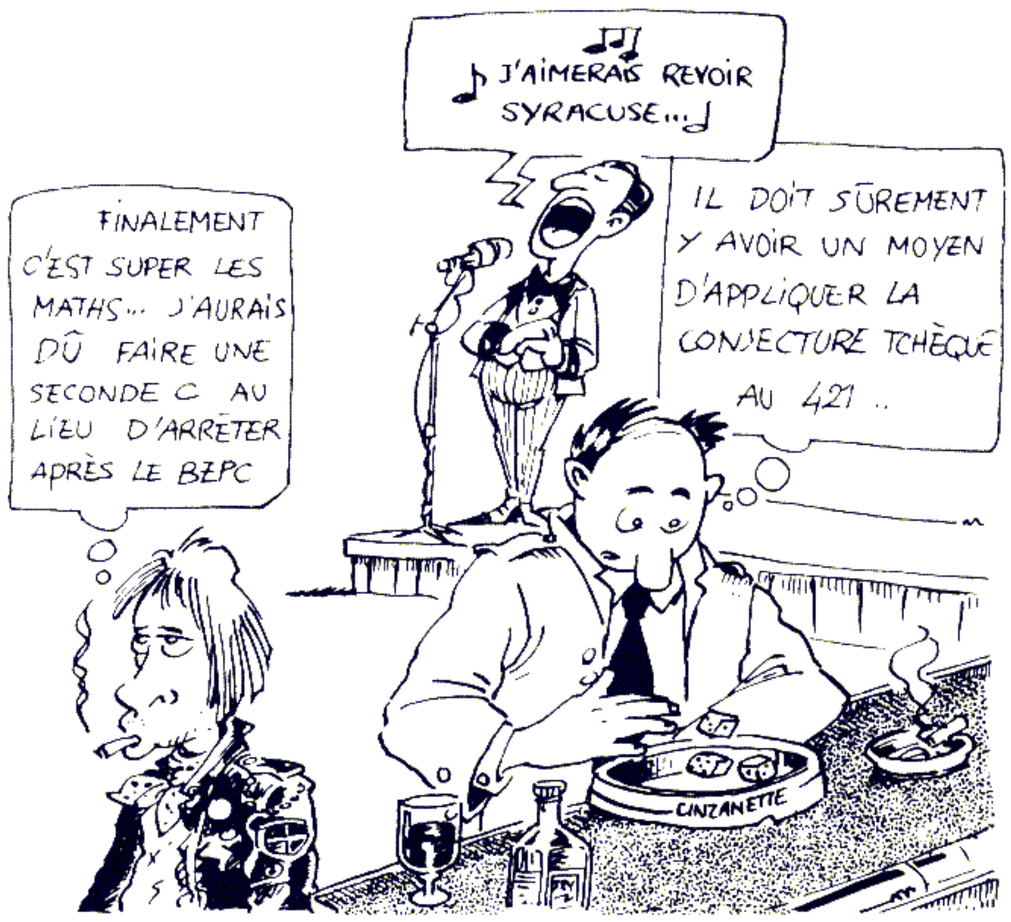
PROBLÈME DE SYRACUSE OU CONJECTURE TCHÈQUE

TOUJOURS PAS DE SOLUTION

SI votre ordinateur est doté d'un Z80, voici une bonne occasion de découvrir l'assembleur ou le langage-machine. Mais peut-être pratiquez-vous déjà cette façon de programmer ? Dans ce cas, c'est l'application proposée ici qui pourrait vous retenir. Il s'agit d'un des plus beaux casse-tête de l'arithmétique.

Prenez un nombre entier quelconque. S'il est pair, divisez-le par deux. S'il est impair, multipliez-le par trois, et ajoutez un. Avec le nouvel entier obtenu, renouvelez l'opération. Quel que soit l'entier de départ et après une suite d'opérations plus ou moins longue, on arrive toujours à passer par la valeur 1 (qui elle-même boucle indéfiniment en donnant 4, puis 2, puis 1, puis 4, puis 2, puis 1, etc.). Par exemple, en partant avec 3, cela donne : 10, 5, 16, 8, 4, 2 et 1.

Tous les chemins mènent à 1



Ce problème, aux allures fort simples, a été mis à la mode à l'Université de Syracuse aux Etats-Unis en 1950. Il a été repris de nombreuses fois (1) mais n'a toujours pas été résolu. A l'heure actuelle, la seule chose sûre est que tous les entiers de 1 à 10^{12} finissent par retomber sur 1 : la propriété « semble » donc vraie, aucun contre-exemple n'ayant été trouvé.

En examinant la suite donnée par 3, nous dirons que la distance de 3 vaut 7, puisque 3 demande 7 itérations avant d'arriver à 1. Il est facile de faire un programme Basic calculant la distance en fonction de l'entier de départ, mais les résultats se font souvent attendre très longtemps...

D'où l'idée d'écrire un tel programme en assembleur (ici Z80) pour gagner en vitesse. Il a été conçu de façon à être facilement implanté sur tout ordinateur construit autour de ce processeur.

Le nombre obtenu après 3 n + 1 ne peut être que pair, n étant impair dans ce cas. Il est donc destiné à être divisé par 2, d'où le CALL DIV derrière le CALL MUL (dans le programme principal) sans test de parité entre les deux.

Au départ, le nombre initial est stocké sur 15 octets, plus un octet indiquant le nombre d'octets significatifs (donc inférieur ou égal à 15) en zone NB1. Un appel de SYRAC calcule alors la distance dans DIST. Puisqu'en Z80, les programmes sont difficilement relogeables, les octets à corriger sont soulignés.

La règle d'adoption est simple : selon les adresses MEV libres de votre système, implantez la routine machine à une adresse multiple de 256, soit &Hxx00. Cela se fait simplement en remplaçant par &Hxx tous les octets soulignés dans la liste. Et cela marchera sur votre machine !

Vous êtes maintenant capable de savoir en moins d'une seconde que la distance de 63 728 127 vaut 949. Et vous découvrirez des choses étranges, par exemple que tous les entiers de 57 346 à 57 370 ont la même distance valant 78 !

Il est possible de modifier et d'améliorer le programme. Voici quelques suggestions. Si vous désirez pouvoir étudier des nombres plus grands encore, remplacez LG EQU 15 par LG EQU 255. Vous pourrez alors aller jusqu'à 10^{614} au lieu de 10^{36}. D'autre part, il serait intéressant d'obtenir le nombre maximum atteint pendant le cheminement vers 1. Ainsi l'entier 26 623, de distance 307, ne monte-t-il pas jusqu'à 106 358 020 !

Pourquoi, enfin, ne pas chercher les

(1) Notamment, dans la revue « Pour la Science » de mai 1984 et dans « Le Petit Archimède » (édité par l'ADCS, association pour le développement de la culture scientifique, BP 022 - 80002 Amiens Cedex) de juin 1983 et de novembre 1983. Ce problème est également connu sous le nom de conjecture tchèque.

```
                ;
                ; ***** Probleme de Syracuse
                ; ***** Auteur    : Christian BOYER
                ; ***** Copyright : LIST et l'auteur
                ;
LG:     EQU   15
LGC:    EQU   4
;
NB1:    DEFS  LG                ; Entier de depart
LG1:    DEFS  1
;
NB2:    DEFS  LG
LG2:    DEFS  1
;
NB3:    DEFS  LG
LG3:    DEFS  1
;
DEPASS: DEFS  1                 ; Depassement
DIST:   DEFS  LGC               ; Distance
;
; ---- Programme principal
;
SYRAC:  CALL  INIT              ; Initialisation
SYRAC1: CALL  PAIR              ; NB2 pair ?
        JR    Z,SYRAC2          ; Si oui, on saute MUL
        CALL  MUL               ; NB2=3*NB2+1
        JR    C,PROBL
SYRAC2: CALL  DIV               ; NB2=NB2/2
        JR    C,PROBL
        CALL  FINI              ; NB2=?1
        JR    NZ,SYRAC1         ; On boucle tant que
        RET                     ;   different de 1.
;
PROBL:  LD    A,1               ; Si probleme, DEPASS=1
        LD    (DEPASS),A
        RET
;
; ---- Initialisation
;
INIT:   LD    DE,NB2            ; NB2=NB1
        LD    HL,NB1
        LD    BC,LG+1
        LDIR
;
        LD    HL,DIST           ; DIST=0
        LD    B,LGC
        XOR   A
INIT1:  LD    (HL),A
        INC   HL
        DJNZ  INIT1
        LD    (DEPASS),A        ; DEPASS=0
        RET
;
; ---- NB2 est-il pair ?
;
PAIR:   LD    A,(LG2-1)
        AND   1
        RET
;
        SKP   H
;
; ---- NB2=3*NB2+1 et DIST=DIST+1
;
MUL:    LD    A,(LG2)           ; NB3=NB2
        LD    C,A
        LD    B,0
        INC   BC
        LD    DE,LG3-1
        LD    HL,LG2-1
        LDDR
;
        LD    B,A               ; NB2=2*NB2
        LD    HL,LG2-1
        OR    A
MUL1:   RL    (HL)
        DEC   HL
        DJNZ  MUL1
        JR    NC,MUL2
;
        LD    A,1               ; NB2 allonge
        LD    (HL),A
        LD    A,(LG2)
        INC   A
        LD    (LG2),A
        CP    LG
        CCF
        RET   C
;
MUL2:   LD    A,(LG2)           ; NB2=NB2+NB3+1
        LD    B,A
        LD    DE,LG2-1
        LD    HL,LG3-1
        SCF
MUL3:   LD    A,(DE)
        ADC   A,(HL)
        LD    (DE),A
        DEC   DE
        DEC   HL
        DJNZ  MUL3
        JR    NC,MUL4
;
        LD    A,1               ; NB2 allonge
        LD    (DE),A
        LD    A,(LG2)
        INC   A
        LD    (LG2),A
        CP    LG
        CCF
        RET   C
;
MUL4:   CALL  MAJDIS            ; DIST=DIST+1
        RET
;
        SKP   H
;
; ---- NB2=NB2/2 et DIST=DIST+1
;
DIV:    LD    HL,LG2            ; Recherche 1er octet
        LD    C,(HL)            ;   non nul de NB2.
        LD    B,0
        OR    A
        SBC   HL,BC
;
        LD    A,(HL)            ; 1er octet = 1 ?
        CP    1
        JR    NZ,DIV1
;
        LD    A,C               ; Si oui, modif LG2.
        DEC   A
        LD    (LG2),A
;
DIV1:   LD    B,C               ; NB2=NB2/2
        OR    A
DIV2:   RR    (HL)
        INC   HL
        DJNZ  DIV2
;
        CALL  MAJDIS            ; DIST=DIST+1
        RET
;
; ---- DIST=DIST+1
;
MAJDIS: LD    HL,DIST+LGC
        DEC   HL
        SCF
        LD    B,LGC
MAJ1:   LD    A,(HL)
        ADC   A,0
        LD    (HL),A
        RET   NC
        DEC   HL
        DJNZ  MAJ1
        RET
;
; ---- Est-ce que NB2 = 1 ?
;
FINI:   LD    A,(LG2)
        CP    1
        RET   NZ
        LD    A,(LG2-1)
        CP    1
        RET
;
        END
```

La liste du programme en assembleur du Z80

Si vous ne possédez pas d'assembleur, le langage-machine peut être introduit par DATA avec le programme Basic ci-dessous, éventuellement adapté aux particularités de votre ordinateur.

```
5 AD=&HOOOO
10 REM *** Chargement langage machine
20 FOR I=&H35 TO &HFA
30 READ J:POKE AD+I,J
40 NEXT I
50 DATA &HCD,&H53,&HOO,&HCD,&H6C,&HOO,.....
51 DATA ..........................................
58 DATA ..........................................
59 DATA .....&H3A,&H1E,&HOO,&HFE,&HO1,&HC9
60 REM *** Introduction du nombre
70 FOR I=0 TO 14:POKE AD+I,0:NEXT I
80 INPUT "Nombre ";N
90 L=0
100 P=INT(N/256):POKE AD+14-L,N-256*P
110 N=P:L=L+1:IF N<>0 THEN 100
120 POKE AD+15,L
130 REM *** Calcul et resultat
140 CALL(AD+&H35):D=0
150 IF PEEK(AD+&H30)=1 THEN PRINT"Depassement":GOTO 60
160 FOR I=&H31 TO &H34:D=256*D+PEEK(AD+I):NEXT I
170 PRINT"Distance ";D
180 GOTO 60
```

records de distance. Cela commence ainsi :

entier de départ	distance
1	0
2	1
3	7
6	8
7	16
9	19
18	20
25	23
27	111
54	112
73	113
97	118

On peut, sans difficulté, modifier le programme en assembleur pour qu'il effectue la recherche systématique de ces records. Et peut-être trouverez-vous même le fameux « contre-exemple » ne convergeant pas vers 1, et qui vous rendrait célèbre !

Christian BOYER

LOGO N'EST PAS UN LANGAGE ENFANTIN

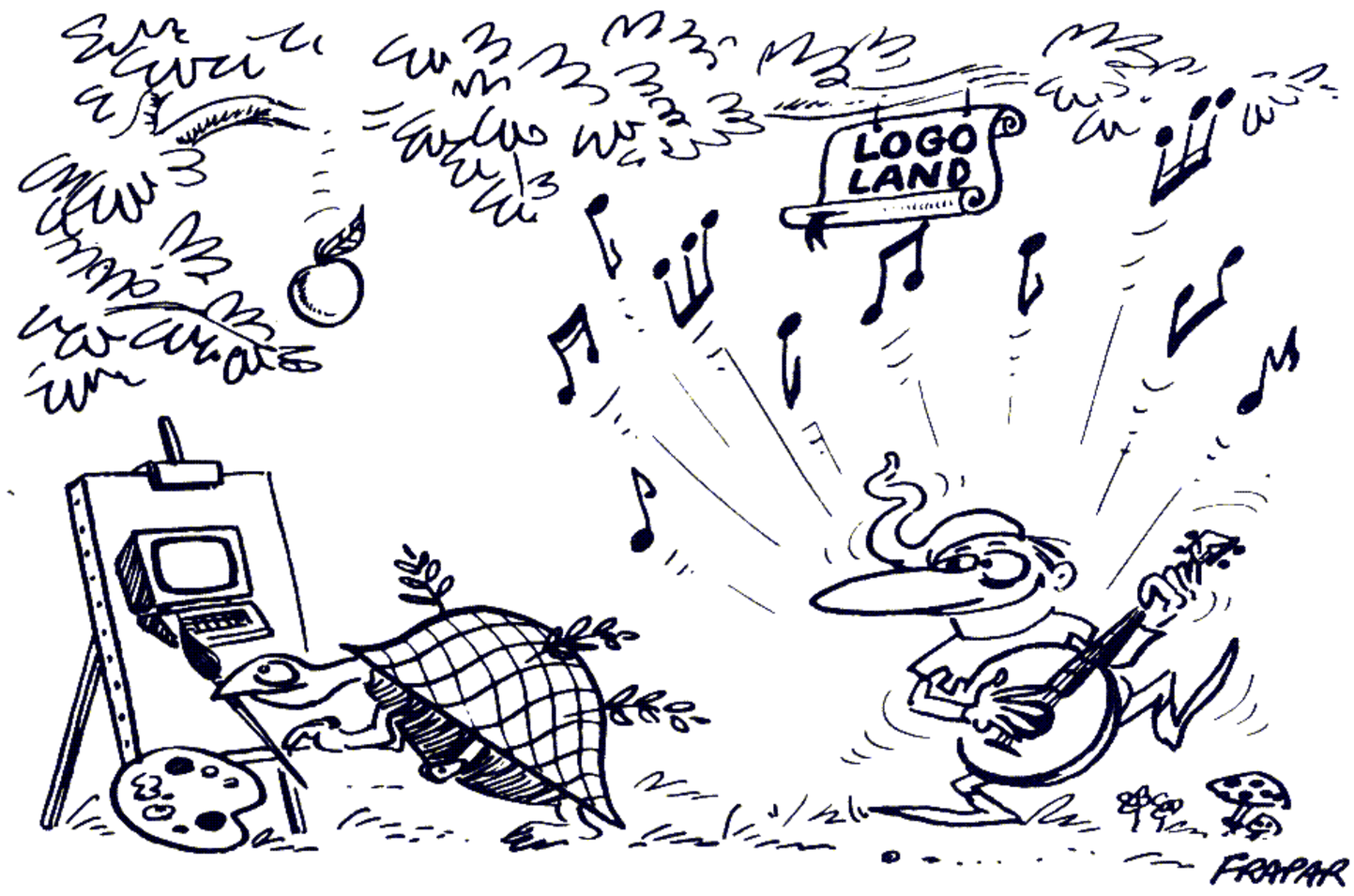
A Logo est associée la notion de "micro-mondes" dont le plus célèbre est lié aux commandes d'un curseur graphique nommé "tortue". Mais il y a aussi le micro-monde des nombres, des mots, de la tortue dynamique qui obéit aux lois de Newton, de la mécanique orbitale ou plus simplement de la musique et des lutins... Même si Logo est accessible à de très jeunes enfants, c'est un langage puissant.

Beaucoup d'articles sur Logo présentent ce langage comme un outil de communication destiné aux enfants et aux enseignants dans un contexte pédagogique. Les élèves construisent leurs connaissances, pas obligatoirement informatiques, en "éduquant" une machine bêtement logique. Oui, Logo a été conçu pour cela. Mais Logo est aussi un remarquable langage de programmation. On peut adopter, grâce à lui, des modes de raisonnement différents de ceux auxquels nous habituent des langages comme Basic ou LSE.

On appelle chaque action par son nom

En Basic, par exemple, la résolution de votre problème passe le plus souvent par l'écriture d'un programme monolithique dont chaque ligne doit être numérotée. Un ensemble de lignes exécuté fréquemment peut être groupé en sous-programme, donnant aux instructions GOTO et GOSUB un rôle privilégié dans la structuration de votre programme. Dans un langage de type Logo, chaque action est décrite sous forme de procédures indépendantes, possédant un nom. Un programme est un enchaînement d'appels à des procédures clairement identifiées. Le GOSUB 500 devient par exemple ENTRER.NOMBRE.

Notre objectif n'est pas d'aborder le B.A.-BA du langage Logo, que vous trouverez par ailleurs dans d'autres articles ou dans le manuel accompagnant chaque version. Ce manuel est en

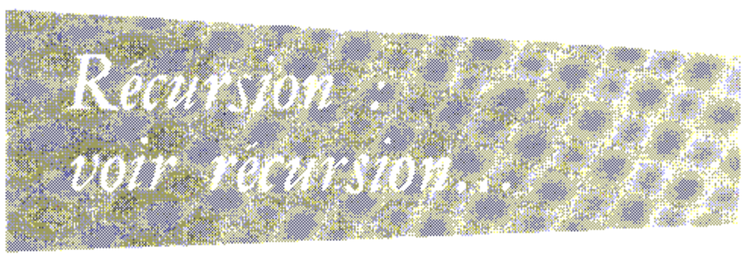


RAPPEL DE LA SYNTAXE LOGO

LOGO est un langage procédural. Les procédures disponibles à l'initialisation sont appelées **primitives.** Celles que vous créez sont nommées **procédures.** Une procédure commence par le mot POUR et se termine par FIN.

En Logo, un nombre est écrit tel quel, éventuellement précédé d'un signe. Un mot est toujours précédé de guillemets (on ne referme pas les guillemets à la fin du mot). Une liste est encadrée de crochets carrés []. Les noms de variables, qui ne sont pas liés à leur contenu, sont des mots. Le contenu de la variable "A est :A.

général bien fait, et bien traduit pour les versions importées. Dès ce premier numéro de LIST, nous nous proposons d'analyser des modes de raisonnement et des programmes de façon à ce que chacun puisse exploiter au mieux les richesses de l'un des premiers langages de type "intelligence artificielle" accessible sur les ordinateurs individuels.



Il est toujours possible d'appliquer à Logo les schémas de pensée classiques liés aux langages de programmation de type Basic. Le résultat : des programmes longs avec beaucoup de variables inutiles, qui n'exploitent pas la récursion ou le passage des paramètres par noms ou valeurs. Mais arrêtons-nous justement sur la récursion. Nous verrons une autre fois d'autres caractéristiques de Logo.

La définition la plus courte de la récursion est sans doute celle donnée par certains dépliants publicitaires d'origine québécoise. Récursion : voir Récursion... On peut aussi la définir comme l'art de recommencer la même chose, éventuellement dans des conditions différentes. Une procédure récursive contient toujours au moins un appel à elle-même :

```
POUR ECRIRE.SANS.FIN
AFFICHE [J'AIME LOGO]
ECRIRE.SANS.FIN
FIN
```

Procédure que d'autres auraient écrite, sans récursion :

```
10 PRINT "J'AIME BASIC"
20 GOTO 10
```

En fait, ECRIRE.SANS.FIN utilise la récursion dite "terminale" sans changer les conditions initiales. C'est une itération sans grand intérêt. Voici un autre cas de récursion terminale :

```
POUR DOUBLER :A
AFFICHE :A
DOUBLER MOT :A :A
FIN
```

La ligne DOUBLER MOT :A :A rappelle la procédure DOUBLER en prenant comme paramètre le résultat de l'instruction MOT :A :A qui suppose que le contenu de la variable locale "A est un nombre ou un mot, et concatène ce contenu à lui-même. Nous n'avons pas besoin de créer de variable "A, ni de lui affecter à chaque appel le résultat de la concaténation.

Un programmeur "classique" aurait écrit par exemple :

```
POUR DOUBLER :A
AFFICHE :A
DONNE "A MOT :A :A
DOUBLER :A
FIN
```

Ce qui est correct, mais inutilement long. Faisons un essai :

```
DOUBLER "TUT
TUT
TUTTUT
TUTTUTTUTTUT
TUTTUTTUTTUTTUTTUTTUTTUTTUT
...
```

L'exécution de la procédure ne s'arrête jamais. Il faut donc se méfier des procédures récursives, et **toujours** introduire un test d'arrêt, en général comme première instruction. Par exemple : SI COMPTE :A > 7 ALORS STOP

La récursion non terminale est un appel à la procédure elle-même avant sa fin. Pour bien la comprendre, il vous faut maîtriser les deux exemples que voici.

Exemple n° 1 :

```
POUR ALIGNER :M
SI VIDE? :M ALORS STOP
AFFICHE PREMIER :M
ALIGNER SAUFPREMIER :M
FIN
```

Exemple n° 2 :

```
POUR ALIGNER1 :M
SI VIDE? :M ALORS STOP
ALIGNER1 SAUFPREMIER :M
AFFICHE PREMIER :M
FIN
```

Dans le deuxième exemple, ALIGNER1, les deux dernières lignes de ALIGNER sont interverties, ce qui nous donne à l'écran :

Exemple n° 1	Exemple n° 2
M	T
O	O
T	M

Le tableau ci-dessous explique la différence des résultats obtenus avec les deux programmes.

Les deux procédures pas à pas

```
ALIGNER "MOT
SI VIDE? " MOT ALORS STOP
AFFICHE PREMIER "MOT
M
ALIGNER "OT
SI VIDE? "OT ALORS STOP
AFFICHE PREMIER "OT
O
ALIGNER "T
SI VIDE? "T ALORS STOP
AFFICHE PREMIER "T
T
ALIGNER "
SI VIDE? " ALORS STOP
Terminé
```

```
ALIGNER1 "MOT
SI VIDE? "MOT ALORS STOP
ALIGNER1 "OT
SI VIDE? "OT ALORS STOP
ALIGNER1 "T
SI VIDE? "T ALORS STOP
ALIGNER1 "
SI VIDE? " ALORS STOP
AFFICHE PREMIER "T
T
AFFICHE PREMIER "OT
O
AFFICHE PREMIER "MOT
M
Terminé
```

Imaginons maintenant la procédure ALIGNER1 ainsi modifiée :

```
POUR ALIGNER1 :M
SI PREMIER :M = :M ALORS STOP
DONNE "M SAUFPREMIER :M
ALIGNER1 :M
AFFICHE PREMIER :M
FIN
```

La première ligne diffère de la précédente en ce sens que la procédure s'arrête lorsque le paramètre ne contient plus qu'un seul caractère qui, en principe, ne doit pas s'afficher. Elle contient aussi une affectation qui

remplace le contenu de "M par son ancien contenu amputé de son premier caractère. La nouvelle procédure ne semble pas très différente de l'ancienne. Et pourtant, exécutons-la :

```
ALIGNER1 "CHAT
T
T
T
```

Aviez-vous prévu ce résultat ? C'est pourtant très simple. Dans le premier cas, l'affectation DONNE "M SAUFPREMIER :M est implicite dans l'appel ALIGNER1 SAUFPREMIER :M, mais le paramètre est une **variable locale** créée à chaque appel, et elle est indépendante de la variable locale "M de la procédure appelante, bien que son nom soit identique. La variable "M a donc une identité et une valeur dans chaque appel, ce qui permet d'exécuter correctement les affichages lorsque le dernier appel entraîne l'exécution de la primitive STOP.

T, T et T : la queue du chat

Par contre, dans le second cas, la primitive DONNE entraîne pour la variable "M un statut de **variable globale.** Sa dernière valeur étant "T, elle la conserve pour toutes les procédures appelantes, qui sont au nombre de trois : ALIGNER1 "CHAT, ALIGNER1 "HAT et ALIGNER1 "AT. C'est pourquoi nous avons obtenu l'affichage de trois T.

Cet exemple n'est pas neutre car, si vous avez pratiqué d'autres langages de programmation, par exemple Basic, vous garderez une tendance à vouloir affecter des valeurs à toutes vos variables, et vous obtiendrez parfois, en Logo, des résultats "bizarres" du style factorielle 5 est égale à 0.

Quatre petits problèmes

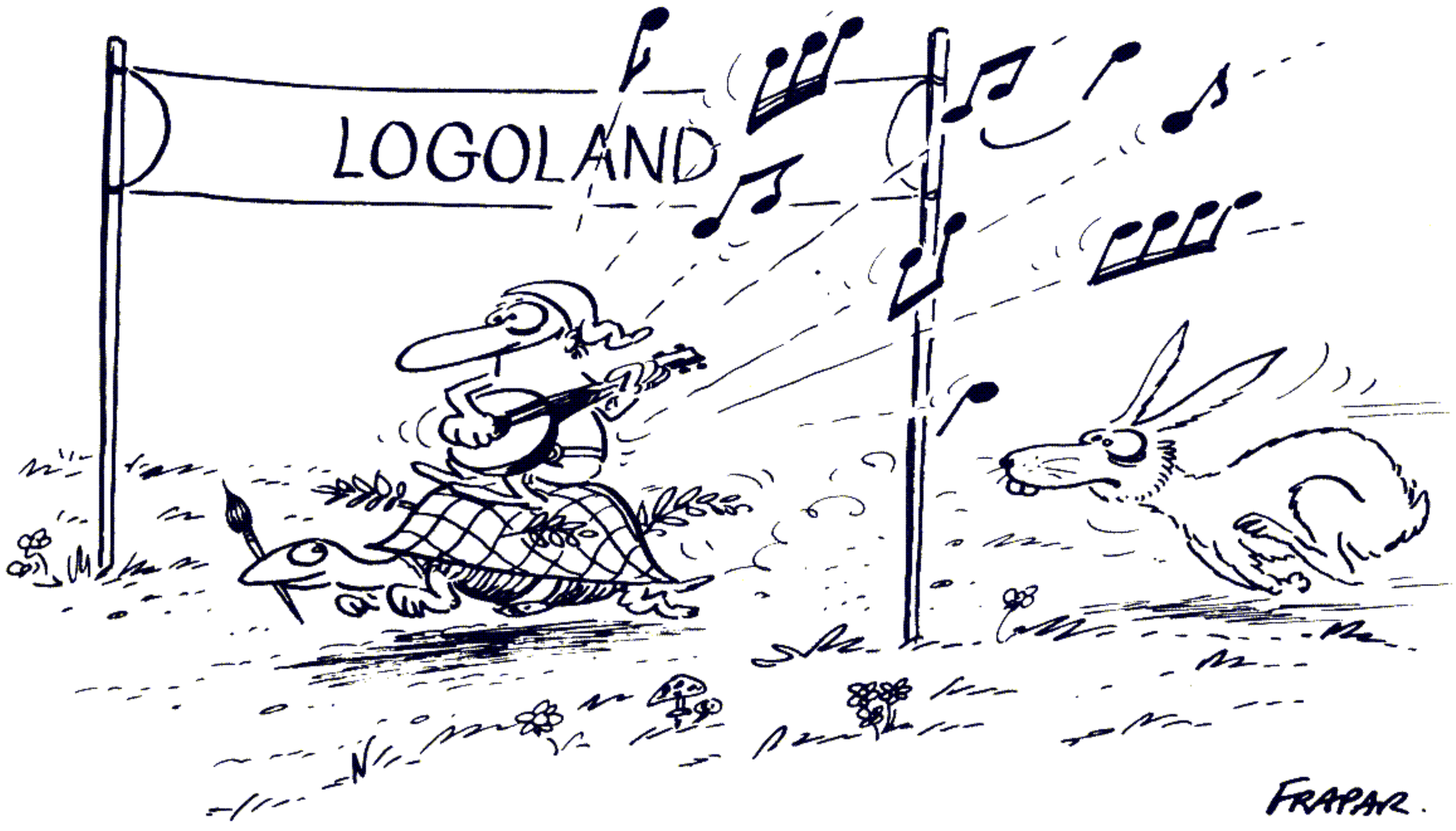
Le raisonnement inverse est également vrai, car lorsque vous aurez compris la puissance de la notion de paramètre, vous vous étonnerez des résultats produits par des procédures qui sont censées modifier des données, et qui les conservent intactes après traitement. Attention : distinguez bien ce qui est local, recréé à chaque appel et inconnu en dehors de la procédure qui le traite, de ce qui est global. En Logo, il faut s'habituer à ce qu'un même nom de variable puisse représenter des choses différentes, et à ce que plusieurs noms différents représentent la même chose. Le tout est de bien contrôler qui fait quoi, à quel moment, sur quel type d'informations.

Avant d'examiner ensemble les petits problèmes que vous n'arrivez pas à maîtriser et qui pourraient faire l'objet de thèmes d'études dans les prochains numéros, voici quatre procédures à méditer. Bien sûr, il vous suffit de les taper pour comprendre ce qu'elles font. Malgré les initialisations nécessaires, elles ne contiennent aucune affectation. (Et j'espère bien que certains d'entre vous me soumettront leurs propres procédures.)

```
POUR DESSIN
AVANCE 2 DROITE 5
SI CAP = 0 ALORS STOP
DESSIN
AVANCE 10 GAUCHE 5
FIN

POUR AA :L1
SI :L1 = [ ] ALORS AFFICHE
:L1 STOP
BB :L1[ ]
FIN

POUR BB :L2 :L3
SI :L2 = [ ] ALORS AFFICHE
:L3 NIVEAUSUP
CC :L2 :L3 [ ]
FIN

POUR CC :L4 :L5 :L6
SI :L5 = [ ] BB SAUFPREMIER
:L4 PHRASE PREMIER :L4 :L5
SI PREMIER :L4 > DERNIER
:L5 ALORS BB SAUFPREMIER :L4
(PHRASE :L5 PREMIER :L4 :L6)
SINON CC :L4 SAUFDERNIER :L5
DERNIER :L5 :L6
FIN
```

Il y a deux choses à savoir ici. La première est que la primitive NIVEAUSUP permet de revenir de n'importe quel niveau de récursion sans repasser par les niveaux intermédiaires (essayez STOP à la place, et vous apprécierez la différence). Seconde remarque : ces procédures sont écrites en Edi Logo ; adaptez-les à votre version. Sachant cela, pouvez-vous décrire l'action de la procédure AA ? Pour l'une des trois procédures, une ligne n'est pas nécessaire. Laquelle ? Enfin, il reste une petite bogue. Sauriez-vous la trouver et la corriger ?

Robert DAGUESSE



COMMENT DÉTERRER 8 KO DE MÉMOIRE MORTE

(PC-1251)

SHARP a toujours annoncé que le PC-1251 était doté de 24 Ko de mémoire morte. Vous connaissez peut-être les 16 Ko que l'on trouve aux adresses 16384 à 32767. Mais comment dénicher les 8 Ko manquants ? L'instruction PEEK reste en effet inopérante...

Dans le récent manuel du langage-machine PC-1251 (1), le constructeur indique que 96 octets de mémoire vive (MEV) et 8 Ko de mémoire morte (MEM) sont physiquement inscrits à l'intérieur même du processeur SC-61860. Malheureusement, on ne peut pas lire cette mémoire morte en utilisant PEEK. L'instruction LDD du langage-machine reste elle aussi sans effet, le registre DP ne pointant que vers la mémoire externe au processeur. La situation paraît donc bloquée.

Un indice bien timide

Et pourtant, dans le tableau des instructions (page 94 du manuel de référence), on trouve la curieuse « DATA » — à ne pas confondre avec le DATA du Basic. Cette instruction est codée 53 en système décimal, soit &35 en hexadécimal. Dans la suite de ce texte, nous emploierons le préfixe & pour signaler qu'une valeur est exprimée dans cette dernière notation. Mais revenons à la fameuse instruction DATA dont le manuel, bizarrement, ne dit rien.

En fait, DATA permet de transférer un bloc de I + 1 octets vers la mémoire vive (MEV) interne au processeur, et cela de la façon suivante :

(P) ← (B, A)
P ← P + 1
B, A ← B, A + 1
I ← I — 1

Cette procédure se répète jusqu'à ce que I ait pris la valeur &FF. Or il s'avère que DATA permet de tout transférer : l'octet pointé par les registres B et A du processeur peut appartenir à la MEM externe **ou** interne. Nous sommes sauvés ! C'est probablement la seule instruction donnant accès à la MEM interne. L'appel de sous-programmes se fait simplement par les codes &E0nn à &FFnn qui sont des CAL00nn à CAL1Fnn (notez bien CAL avec un seul L).

Bien entendu, une fois l'octet demandé transféré dans la MEV interne du processeur, il faut encore pouvoir le transférer dans la MEV externe. Alors seulement, un PEEK permettra de le lire. La seule façon de faire est apparemment d'utiliser « EXWD » qui échange un bloc de I + 1 octets entre ces deux zones de mémoire vive. Le tout débouche sur le programme suivant :

```
C100   03  xx        LIB  xx      ;l'octet à lire se
       02  yy        LIA  yy      ;situe en xxyy
       A0            LP   20      ;20 pour ne pas perturber la pile
       00  00        LII  00      ;bloc d'un octet
       35            DATA         ;(P) ← (B, A)
C108   10  00  FF     LIDP C0FF    ;résultat en C0FF
       A0            LP   20      ;on relit en 20
       00  00        LII  00      ;bloc d'un octet
       19            EXWD         ;(DP) ↔ (P)
       37            RTN          ;fin
```

D'où le programme Basic :

```
10 POKE &C100, &03, &00, &02,
   &00, &A0, &00, &00, &35
20 POKE &C108, &10, &C0, &FF,
   &A0, &00, &00, &19, &37
30 INPUT « ADRESSE ? » ;A
40 X = INT (A/256) : Y = A -
   256 * X
50 POKE &C101, X : POKE &C103,
   Y
60 CALL &C100
70 PRINT « PEEK » ; A ; « = » ;
   PEEK &C0FF
80 GOTO 30
```

Grâce à ce programme, vous découvrirez que la mémoire morte du système commence ainsi :

```
0000  4E  A0  WAIT  A0  ;78,160
0002  02  01  LIA   01  ;2,1
0004  12  5F  LIP   5F  ;18,95
0006  DB      EXAM      ;219
0007  DF      OUTC      ;223
```

Suivent 8184 octets que vous pouvez désassembler si vous avez l'âme d'un explorateur...

Bernard BOUVIER

(1) Se reporter à la page 28 du présent numéro

LA BOÎTE A MALICES...

PRENEZ un programme et ôtez-en très soigneusement toutes les astuces, des plus élémentaires aux plus subtiles. Vous êtes certain de n'en avoir laissé passer aucune ? Bien. Que reste-t-il ? Rien, ou peut-être une bogue ou deux (tout le monde peut se tromper). En fait, tout programme n'est qu'une suite d'astuces. Dans les pages qui suivent, vous en trouverez un grand nombre. Certaines sont de portée très générale. D'autres ne valent que pour un matériel particulier. Mais dans tous les cas, vous aurez intérêt à être curieux, à fouiner dans la boîte à malices. Même s'il ne s'agit pas de votre machine, vous trouverez souvent des idées à reprendre. Par ailleurs vous avez sans doute vos propres recettes, vos façons de faire... Si ce ne sont pas des secrets que vous cherchez à conserver jalousement, faites en part au journal. Celles qui nous paraîtront les plus intéressantes enrichiront à leur tour la boîte à malices. Tous les lecteurs pourront ainsi en profiter.

LIST

FX-702 P

COMMENT « EFFACER » UN BLANC

Sur le FX-702P, une ligne telle que : 10 A = 8 : PRT CSR 0 ; A aura pour effet d'imprimer le chiffre 8 sur la deuxième matrice de l'écran.

La première matrice est réservée au signe et celui-ci n'apparaît que s'il est négatif.

Un moyen de faire imprimer 8 sur la première matrice est :
10 A=8 : PRT CSR 0 ; # ; A

Ce moyen n'est pas applicable sur le PB-100.

Sylvain CLÉMENT

ALICE

QUELQUES FACÉTIES

Alice dispose, en version de base, de huit couleurs, qui sont en fait considérées comme des caractères graphiques particuliers. Les codes 128 à 255 sont ainsi réservés aux caractères graphiques, regroupés par couleur. Ces caractères peuvent donc être obtenus soit par un CHR$(n), soit directement dans un PRINT ou dans une chaîne de caractères. Mais, dans ce dernier cas, ils n'apparaîtront pas lors du listage d'un programme. On peut contourner cet inconvénient de deux façons au moins.

Première méthode (un peu lourde) : ajouter à chaque ligne faisant appel à un affichage graphique en couleur une remarque précisant les touches utilisées.

Ainsi, on écrira : B$ = "888" : REM couleur 8, Shift Z, puis Shift R et Shift S. Ou bien encore PRINT "464" : REM couleur 4, Shift Q, couleur 6, Shift Q, couleur 4, Shift Q.

Deuxième méthode, beaucoup plus simple à mettre en œuvre : on prendra soin, en début de programme, de définir les variables alphanumériques dont on aura besoin par la suite pour faire des dessins.

On écrira par exemple : A$ =

CHR$(255) : B$ = CHR$(239), etc. Pour utiliser le graphisme, il suffira d'écrire : PRINT A$ B$ A$. On peut remarquer au passage que le ";" (point-virgule) n'est en fait pas nécessaire, ce qui allège beaucoup l'écriture d'un programme.

L'extension de mémoire vive 16 Ko devrait être disponible très prochainement. Elle s'enfiche derrière l'appareil (à la manière des extensions du ZX 81), et elle sera livrée avec un manuel "Aller plus loin avec Alice". Dans les particularités intéressantes offertes par cette extension, figurent la possibilité de faire du graphisme haute résolution, l'amélioration des sauvegardes sur cassette et des possibilités de programmes en assembleur.

A propos de graphisme haute résolution, avez-vous déjà essayé de faire un POKE à une adresse comprise entre 32768 et 49151 ; essayez par exemple POKE 32768,34 ou une autre valeur ; vous obtiendrez des affichages très curieux ; un POKE 32768,0 vous permettra heureusement de tout remettre en ordre.

De la même façon, des POKE dans les quatre premiers octets provoquent des réactions bizarres ; essayez par exemple POKE 0,23 ou POKE 1,32 et regardez ce que deviennent les touches du clavier. Plutôt curieux, non ? Qui pourra nous donner les raisons de ces comportements fantaisistes ?

Jacques DECONCHAT

CANON X-07

UNE TABLE DES MATIÈRES

Voici un utilitaire qui affiche à l'écran du X-07, par pages successives, la liste des programmes en mémoire et permet de lancer l'exécution du programme retenu par simple pression sur une touche.

Les touches ▲ et ▼ servent à changer de pages (le nombre total de pages, ainsi que le numéro de la page courante sont toujours indiqués en haut de l'écran). Pour lancer un programme, il suffit d'appuyer sur le chiffre qui lui est associé à l'affichage. Pour quitter ce programme, on frappe la touche 'Q', comme "quitter".

Patrick LECLERCQ

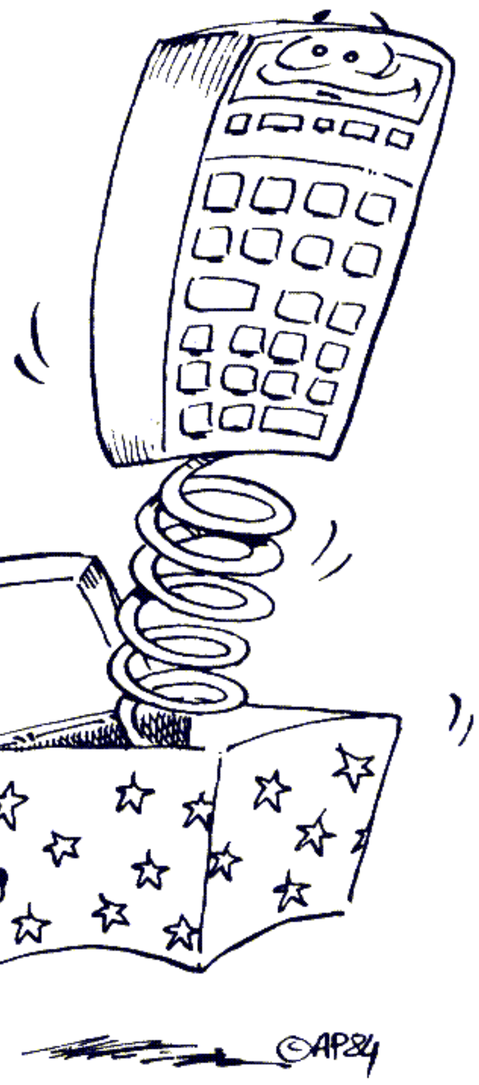


Table des matières
Utilitaire pour Canon X-07
Auteur Patrick Leclercq


```
12 DEFFNP(X)=PEEK(X)+PEEK(X+1)*256
15 CLS:DIMT$(20):'Nb max. de fichiers
17 PRINT"Une seconde SVP"
20 GOSUB1000:NE=X:'LECTURE DES NOMS DEFI
CHIERS
30 NP=INT(NE/6):IFNEMOD6<>0THEN NP=NP+1
40 GOSUB3000:'Affichage de la page coura
nte
50 GOSUB5000:'Attente des ordres et exec
ution
60 GOTO40
1000 A=FNP(528):X=1:T$(0)="Basic"
1010 IFPEEK(A)=0THENRETURN:' Fin de la z
one fichier
1020 E=A+6:IFPEEK(E)<>80THEN1090ELSEI=A:
A$="":F=0:'Test sur l'extension
1040 N=PEEK(I):IFN=32ORI=ETHEN1080
1050 A$=A$+CHR$(N):I=I+1:GOTO1040
1080 T$(X)=A$:X=X+1
1090 A=A+FNP(A+7):GOTO1010:'Lecture de l
'adresse du fichier suivant
3000 REM AFFICHAGE DES NOMS
3020 CLS:X=PC+1:PRINTTAB(6);"Page ";:PRI
NTUSING"#";X;
3030 PRINT"/";:PRINTUSING"#";NP
3040 FORL=0TO2:FORZ=0TO1:IF(6*PC+2*L+Z)=
NETHENL=2:Z=1ELSEGOSUB6000
3050 NEXT:NEXT:RETURN
5000 F=0
5005 A$=INKEY$:IFA$=""THEN5005ELSEA=ASC(
A$):IFPC=0ANDA=48THENCLS:END
5010 IFA>47ANDA<55THENA$=T$(6*PC+A-48):R
UNA$
5020 IFA=30ANDPC<>0THENPC=PC-1:F=1
5030 IFA=31ANDPC+1<>NPTHENPC=PC+1:F=1
5040 IFA$="Q"THENCLS:END
5050 IF F=1THEN RETURN ELSE 5005
6000 REM AFFICHAGE D'UN NOM DE PROGRAMME
6010 LOCATEZ*10,L+1:PRINTT$(PC*6+L*2+Z);
6020 LOCATEZ*10+6,L+1:X=2*L+Z:PRINT"<";:
PRINTUSING"#";X;:PRINT">";:RETURN
```

TI-57 LCD

UNE PAUSE A DURÉE VARIABLE

La fonction Pause de la TI-57 LCD est bien sûr très utile mais elle a l'inconvénient de ne pas être réglable. Impossible de choisir pendant combien de temps l'affichage doit persister.

Imaginons que l'on veuille programmer une horloge pour cette machine et faire afficher heures et minutes pendant 60 secondes, sans interruption : la fonction Pause n'est là d'aucun secours.

Une solution simple, certainement adaptable à d'autres poquettes, consiste à réaliser une boucle vide à l'aide de Dsz. Et la longueur de la boucle est réglable...

Ainsi, le petit programme :

3
5
2
STO 0
1
STO + 1
RCL 1
LBL 0
Dsz
GTO 0
RST

permettra de voir la machine compter avec des pauses d'une minute.

Raoul LEBASTARD

PC-1500

AU BOUT DU ROULEAU ?

L'imprimante CE-150 de l'ordinateur Sharp PC-1500 ne possède pas de touche spéciale pour rembobiner le papier dévidé. Seule l'instruction LF – n permet de remonter de n lignes en arrière. Mais pas plus de 10 cm ! Un dispositif de comptage interne y veille.

A ce jour, une seule méthode existait pour rembobiner malgré tout : LF-23 (en taille 2), OFF qui remet à zéro ce compteur de centimètres et ON pour recommencer. Il faut ajouter à ces

manœuvres le temps perdu à regarder l'imprimante faire valser ses stylos (initialisation) et à débiter cinq lignes de papier blanc ! Il faut recommencer 20 fois l'opération pour rembobiner 2 mètres de papier seulement.

Tout ceci est très dangereux pour la machine étant donné l'état induit sur les nerfs du propriétaire... Si, comme moi, vous êtes au bout du rouleau, le petit programme suivant supprime d'un coup nos problèmes d'hyper-irritabilité en autorisant le recul « illimité » du papier :

```
10:''K'':TEXT:CSIZE 2:IF
INKEY$<>'' ''POKE
31204,0,1:LF—23:GOTO ''K
20:POKE 31204,0,1
```

Faire DEF K *et garder cette touche enfoncée.* Tant que vous pressez, le papier recule. Relâchez et c'est fini. C'est le POKE 31204,0,1 qui remet le compteur à zéro, le reste du programme est sans surprise, classique. La ligne 20 est facultative.

David ROUSSEAU

APPLE II

LE CHAÎNAGE DES PROGRAMMES APPLESOFT

Il existe plusieurs moyens de chaîner des programmes en Applesoft : le plus simple consiste à terminer le premier programme par l'appel du second. Cela marche parfaitement mais présente l'inconvénient de perdre toutes les données du premier programme.

La deuxième solution est d'employer CHAIN, un utilitaire Apple en langage machine qui figure sur la disquette DOS MASTER 3.3. Pour chaîner PROGRAMME1 avec PROGRAMME2 sans perdre les valeurs des variables, il faut que CHAIN figure sur la même disquette que PROGRAMME1 et PROGRAMME2 (on transférera CHAIN avec FID, un autre utilitaire Apple). Il faudra alors insérer les deux lignes suivantes :

... PRINT CHR$(4) ; ''BLOAD CHAIN,. A520
... CALL 520 ''PROGRAMME2''

Malheureusement, si ce procédé marche fort bien, il est en revanche très lent à l'exécution lorsque PROGRAMME1 est long ou comporte beaucoup de données.

Il existe une troisième façon de faire beaucoup plus efficace mais qui nécessite un minimum de précautions. Rappelons tout d'abord quelques généralités sur la façon dont l'Applesoft gère ses variables et ses programmes :

- les adresses du début et de la fin du programme sont stockées dans les pointeurs $67-$68 et $AF-$B0 ;
- les adresses du début et de la fin du stockage des données sont en $69-$70 et $6D-$6E et les adresses du début et de la fin du stockage des chaînes de caractères en $6F-$70 et $73-$74.

Un RUN remettant ces pointeurs à zéro, une solution évidente consiste à stocker les valeurs de ces pointeurs dans une zone protégée et à faire un RUN du deuxième programme dont la première instruction sera naturellement de faire appel à un sous-programme régénérant ces pointeurs. Il faudra aussi naturellement que les deux programmes aient la même longueur apparente. On cherchera la plus grande longueur (par un CALL-151 et un examen du pointeur correspondant $AF-$B0) puis on fixera la longueur du plus petit par un LOMEM correspondant.

Une dernière précaution : si, dans le programme, une chaîne de caractères est initialisée par une instruction du type A$ = ''TOTO'', ''TOTO'' ne sera pas stockée dans la partie haute de la mémoire comme les chaînes habituelles, mais le pointeur de chaîne indiquera l'adresse effective de ''TOTO'' (en plein programme Applesoft) et, lorsque le deuxième programme aura remplacé le premier, ce pointeur pointera sur n'importe quoi ! La solution consiste à remplacer ce genre d'initialisation par A$ = MID$ (''TOTO'',1) qui force ''TOTO'' à aller se placer dans la bonne zone.

On trouvera ci-dessus un exemple de chaînage réalisé de cette façon ainsi que le programme machine correspondant.

Philippe FRANÇOIS

Exemple de chaînage

Programme 1

```
5  PRINT  CHR$ (4)"BLOAD CHAIN"
10 A$ =  MID$ ("TOTO",1): REM force le stockage de "TOTO" en haut
memoire
20 A = 10.01
30 A% = 12
40  PRINT "A$=";A$: PRINT "A=";A: PRINT "A%=";A%
50  CALL 768: REM  Sauvetage des pointeurs
60  PRINT  CHR$ (4)"RUN PROGRAMME2"
```

Programme 2

```
10  LOMEM: 2208
20  CALL 791: REM  adresse du programme de regeneration
40  PRINT "A$=";A$: PRINT "A=";A: PRINT "A%=";A%
```

A l'exécution, on obtient :

```
RUN

A$=TOTO
A=10.01
A%=12

A$=TOTO
A=10.01
A%=12
```

Programme 3

Langage-machine (Applesoft)
Auteur Philippe François
Copyright *LIST* et l'auteur

```
          ORG $300
          OBJ $300
*
DEBPROG EQU $AF
*
          LDX #$0F
          LDA DEBPROG+1
          STA STOCK,X
          DEX
          LDA DEBPROG
          STA STOCK,X
          DEX
LABEL1    LDA $67,X
          STA STOCK,X
          DEX
          BPL LABEL1
          RTS
*
          LDX #$0F
          LDA STOCK,X
          STA DEBPROG+1
          DEX
          LDA STOCK,X
          STA DEBPROG
          DEX
LABEL2    LDA STOCK,X
          STA $67,X
          DEX
          BPL LABEL2
          RTS
*
STOCK     HEX 00000000000000000
          HEX 00000000000000000
```

PC-1211

CONVERTISSEUR DÉCIMAL BINAIRE

Il vous est sûrement arrivé d'avoir à décomposer un nombre en éléments binaires. Le nombre prend alors la forme $N = 2^{x1} + 2^{x2} + 2^{x3} + \ldots + 2^{xn}$

Si N = 70 par exemple, on a : $N = 2^1 + 2^2 + 2^6$ soit égal à 2 + 4 + 64 = 70

Le programme de conversion établi ici pour PC-1211/PC-1212/PC-1 (de préférence avec imprimante) permet de résoudre des variantes de ce problème, en particulier le classement des puissances de deux.

Tout d'abord, deux petits utilitaires indépendants du programme (lignes 500-550 et 600-650) affichent les valeurs successives des exposants de 2, en mode DEF et l'imprimante branchée : par SHFT C(C pour « croissant »), dans l'ordre croissant — dans notre exemple 1, 2, 6 ; par SHFT D(D pour « décroissant »), dans l'ordre décroissant — dans notre exemple 6,2,1.

Chacun de ces deux utilitaires repose sur un algorithme spécifique : le premier est très connu, le second, relatif aux exposants décroissants, un peu moins. Nous ne les reproduisons pas, les lignes des programmes décrivant clairement les processus.

La première partie du programme (lignes 10 à 430) exploite ces deux utilitaires pour imprimer sous forme de tableau non seulement les valeurs des exposants de 2 (appelées X), mais aussi les valeurs des nombres 2^x (appelées Y), dont la somme est naturellement égale à N (dans notre exemple 2, 4 et 64).

L'initialisation par SHFT A sort les résultats dans l'ordre croissant, l'initialisation par SHFT Z, dans l'ordre décroissant.

Dans les deux cas, la liste imprimée rappelle d'abord la valeur de N, définit les positions de X et de Y, imprime les valeurs de X et de Y, totalise les Y et introduit une séparation en vue du calcul suivant.

Remarquez aux lignes 610 et 620 l'introduction d'une constante +0,5. Elle est destinée à éliminer les erreurs d'arrondi de la machine. Sans cette précaution, vous trouvez en mode calcul (et naturellement également en mode programme) des résultats... faux. Par exemple, pour M = 16, X = INT (LOG M/LOG 2) = 3, alors que l'on devrait trouver 4, puisque $16 = 2^4$; pour X = 4, Y = INT (2ΛX) = 15, alors que l'on devrait trouver 16, puisque $2^4 = 16$.

En écrivant X = INT (LOG (M + .5) / LOG 2) et Y = INT (2ΛX + .5), comme aux lignes 610 et 620, vous obtenez les valeurs correctes.

Le plus court programme n'est donc pas forcément le meilleur.

Pierre Ladislas GEDO

Convertisseur décimal binaire
Programme pour PC-1211, 1212/PC-1 avec imprimante
Auteur Pierre Ladislas Gedo

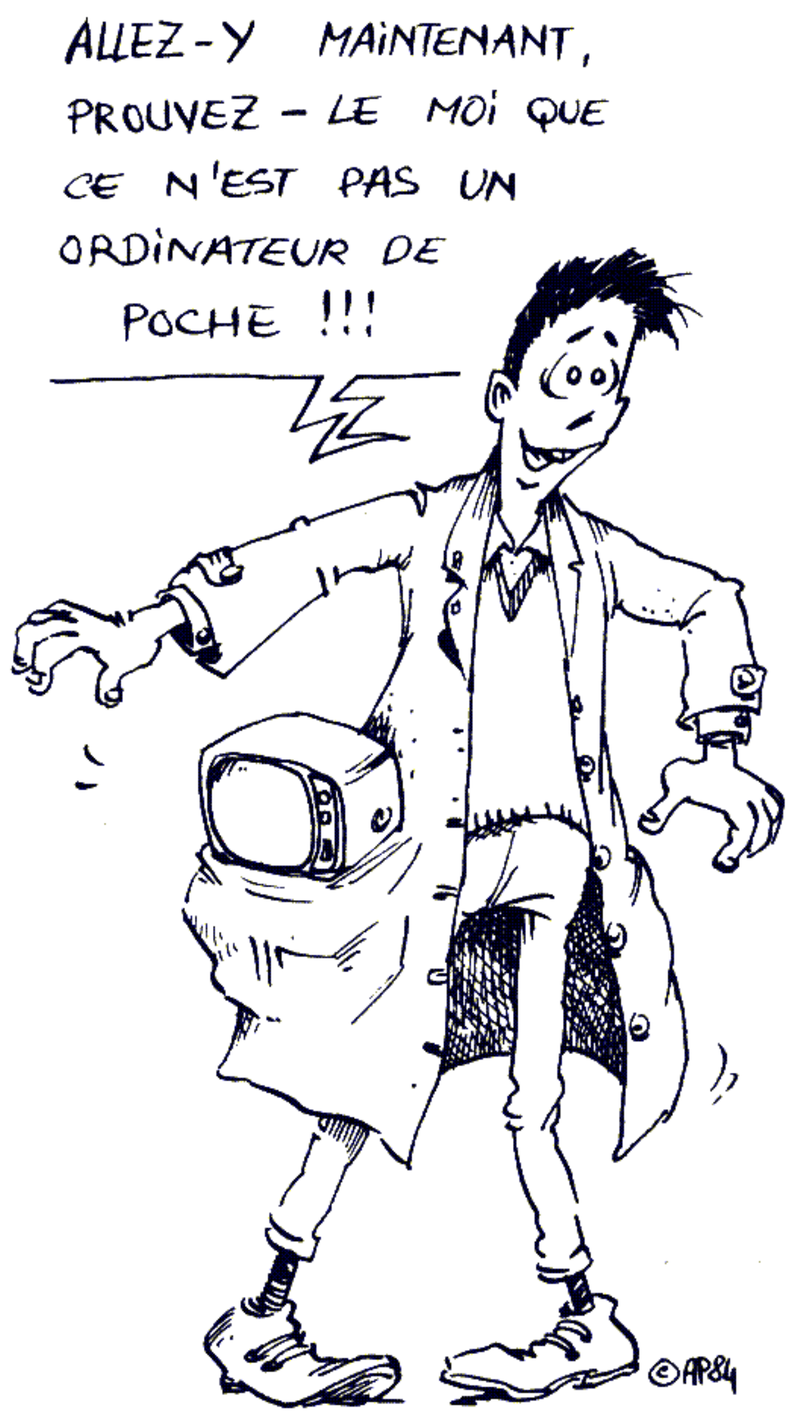

```
10:"A"INPUT " ---> N=";N
20:M=N:X=0
30:GOSUB 200
40:Q=INT (M/2):IF M=2*QLET M=Q:X=X+1:GOTO 40
50:Y=INT (2^X+.5)
60:GOSUB 300
70:IF M>1LET M=M-1:GOTO 40
80:GOSUB 400
110:"Z"INPUT " <--- N=";N
120:M=N
130:GOSUB 200
140:X=INT (LOG (M+.5)/LOG 2)
150:Y=INT (2^X+.5):M=M-Y
160:GOSUB 300
170:IF M>0GOTO 140
180:GOSUB 400
200:A$="":B$=" ":C$="  ":D$="   ":E$="    ":F$="     "
210:PRINT C$;"N=";N
220:PRINT "   =SOMME Y=2^X"
230:PRINT " ----------"
240:PRINT "  X  Y"
250:PRINT " --  ----------"
260:RETURN
300:J=10-INT LOG Y
310:G$=A$:H$=A$
320:IF X<10LET G$=B$
330:IF Y<ESLET H$=F$:J=J-5
340:I$=A$(J)
350:PRINT B$;G$;X;B$;H$;I$;Y
360:RETURN
400:PRINT "  ----------"
410:PRINT N
420:PRINT " **********"
430:END
500:"C"INPUT N:M=N:X=0
510:Q=INT (M/2)
520:IF M=2*QLET M=Q:X=X+1:GOTO 510
530:PRINT X
540:IF M>1LET M=2*Q:GOTO 510
550:END
600:"D"INPUT N:M=N
610:X=INT (LOG (M+.5)/LOG 2)
620:Y=INT (2^X+.5):M=M-Y
630:PRINT X
640:IF M>0GOTO 610
650:END
```

```
N=151874.
 =SOMME Y=2^X
----------
 X        Y
--  ----------
 1.         2.
 6.        64.
 8.       256.
12.      4096.
14.     16384.
17.    131072.
     ----------
    151874.
**********
```

Un nombre décomposé dans l'ordre croissant (SHFT A)...

... et dans l'ordre décroissant (SHFT Z)

```
N=151874.
 =SOMME Y=2^X
----------
 X        Y
--  ----------
17.    131072.
14.     16384.
12.      4096.
 8.       256.
 6.        64.
 1.         2.
     ----------
    151874.
**********
```



PB-100

UNE NOUVELLE FONCTION

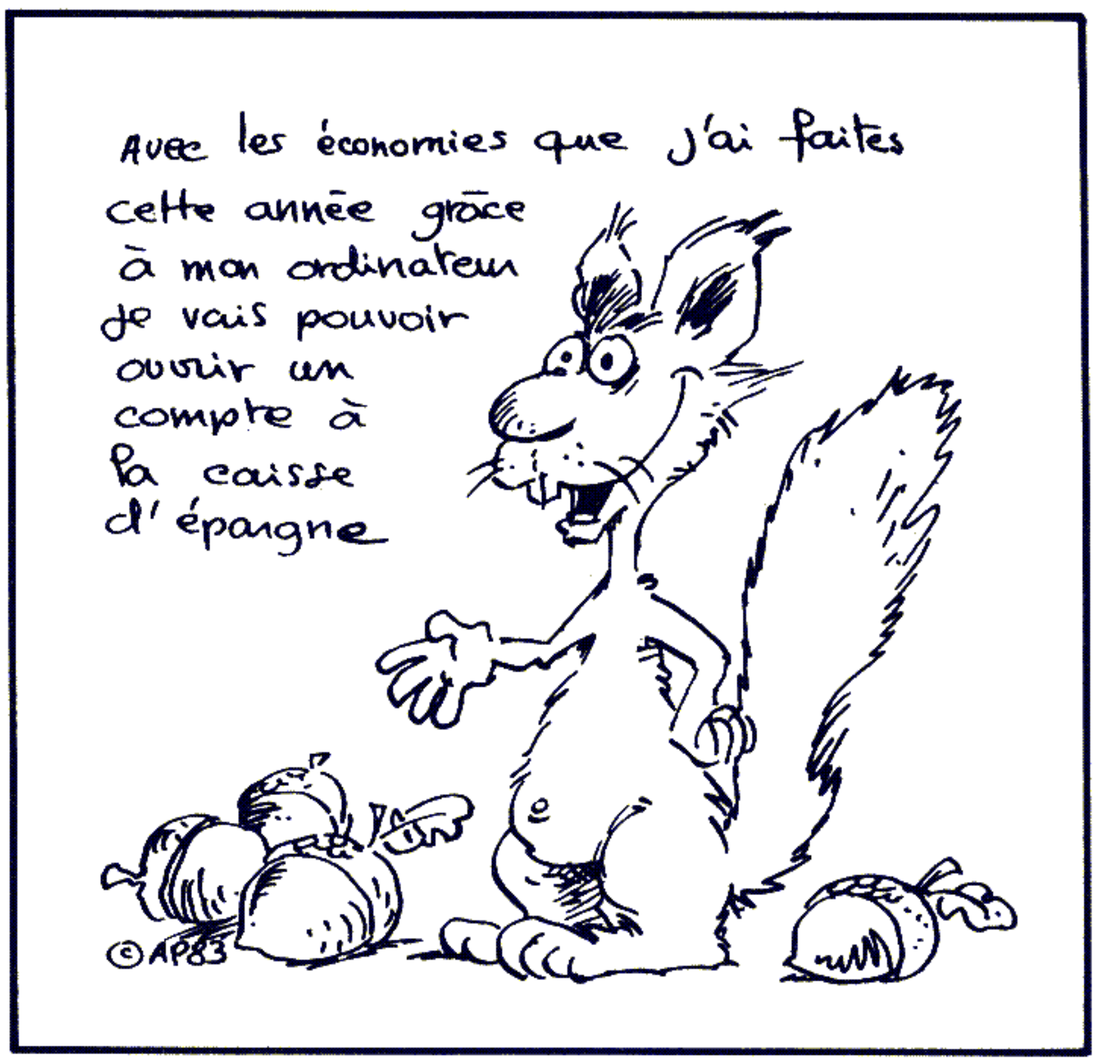
La fonction STR$ du Basic standard transforme une valeur numérique en une chaîne de caractères. A première vue, cette fonction peut sembler inutile. Des lignes telles que 10A = 123 : PRINT STR$(A) et 20A = 123 : PRINT A donnent exactement le même résultat.

Cependant, l'utilité de STR$ est inestimable dans les traitements et l'édition des résultats. Certaines fonctions comme LEN, MID,... ne peuvent être appliquées qu'à des variables de caractères. La transformation de A en STR$(A) devient alors très intéressante.

Mais tous les Basics ne possèdent pas cette fonction. En particulier, elle est absente du PB-100. Il faut donc la simuler. C'est ce que font les quelques lignes du programme ci-dessus. Dans cet exemple, la ligne 1 est à elle seule le programme principal et les lignes 4 à 9 le sous-programme simulant la fonction STR$.

```
1 INPUT Y:GOSUB 4
  :PRINT $:GOTO 1
4 $="0123456789"
5 FOR Z=9 TO 0 ST
  EP -1:Y=Y/10
6 A$(Z)=MID(1+10*
  FRAC Y,1):Y=INT
   Y
7 IF Y*0;NEXT Z
8 $="":Y=Z:FOR Z=
  Y TO 9:$=$+A$(Z
  ):NEXT Z
9 RETURN
```

André TURLURE



TO7

LE TEMPS DE LA « PAUSE »

L'instruction PAUSE n'existant pas sur TO7, il faut l'inventer !

Une première solution consiste à utiliser l'instruction INPUTWAIT dont la syntaxe est « INPUTWAIT L;T,X », ce qui signifie « exécuter la ligne L si après T secondes la variable X n'a pas reçu de valeur ».

Cette instruction a l'avantage de donner une temporisation directement exprimée en secondes, mais cette solution comporte quelques inconvénients. Ainsi, un magnifique point d'interrogation s'affichera pendant toute la durée de l'attente. Et si, par malheur, vous entrez une donnée dans X, le programme poursuivra à la ligne suivante alors que vous vouliez poursuivre en ligne L.

Une deuxième solution consiste à faire tourner une boucle vide comme suit : FOR I = 1 TO T : NEXT I. La temporisation sera proportionnelle à la valeur de T. Sur le TO7, il faut compter environ 6 secondes pour T = 5000.

La troisième solution est d'utiliser une routine en langage-machine utilisant le TIMER du TO7. Je vous en propose une. Quand vous l'aurez implantée en mémoire, il vous suffira de donner la durée de la pause en dixièmes de seconde par POKE 24804, T où T exprime la temporisation désirée en dixièmes de

ZX 81

ET ÇA DÉFILE, ET ÇA DÉFILE...

Pendant la programmation, à l'introduction d'une ligne, le ZX 81 décale le fichier d'affichage pour créer la place de la nouvelle ligne de programme. Puis il affiche le programme en faisant défiler la liste si l'affichage dépasse lécran, et cela jusqu'à l'apparition de la ligne complète. Cet affichage à déroulement fait perdre beaucoup de temps. Or, un CLS puis LIST n° ligne n'empêche pas cette action.

Pour que l'affichage ne considère que les dernières lignes introduites, il faut, au début du programme, garder un n° de ligne libre (2, 3,..., 9 au maximum) et dès que l'affichage remplit l'écran, au lieu de donner son numéro normal à la ligne suivante, on lui octroie un n° réservé auparavant. Après NEWLINE, on édite la ligne et on lui rend son n° normal. L'ordinateur n'affichera plus que la dernière ligne de programme et la précédente. On recommence dès que l'écran est plein. Il ne faut surtout pas oublier d'effacer la ligne réservée quand le programme est terminé.

Exemple, si la ligne réservée est la n° 2 :

ligne normale : 500 GOSUB 1 000

ligne entrée : 002 GOSUB 1000 puis NEWLINE

L'écran liste le programme depuis le début.

On édite la ligne :

modification de ligne : 500 GOSUB 1000 puis NEWLINE

l'ordinateur affiche la ligne précédente (490) et la ligne 500.

L'introduction d'une nouvelle ligne ne fera apparaître le programme qu'à partir de la ligne 500.

Henri CASAL

seconde. Vous pourrez alors dans un programme Basic quelconque lancer la temporisation par EXEC 24805.

Voici le programme Basic qui vous permettra d'implanter la routine de Pause en mémoire :

```
100 FOR ADR=24805 TO 24820
110       READ OCTET
120       POKE ADR, OCTET
130 NEXT ADR
140 DATA 125, 231, 198, 39, 251, 125,
    231, 198, 38, 251, 106, 140, 242, 38,
    241, 57
```

Cette routine occupe seize octets qui normalement ne sont pas utilisés.

Seule restriction : la Pause ainsi obtenue ne peut être supérieure à 25 secondes et demie.

Jean-Paul CARRÉ

PC-1500

UNE QUESTION SUR @ $ ET @

A quoi peuvent bien servir les « variables de tableau mémoire fixe » *dont j'ai trouvé trace en parcourant les codes d'erreur du manuel du PC-1500 (ERROR 20, page 153 précisément). Nulle explication dans le manuel et je n'ai rien trouvé de précis en expérimentant.*

Jean-Pierre PICARD
76 - LE HAVRE

Cela mérite en effet une explication. Le manuel Sharp du PC-1500 est parfois étrangement discret sur certaines des possibilités du Basic de l'ordinateur... et c'est un bon manuel, comparé à certains autres.

En fait, il faut se reporter à la page 71 pour apprendre ce qu'est un tableau de variables. Ce signe @ permet l'utilisation des 26 variables A à Z comme s'il s'agissait d'un tableau (matrice à une dimension de taille 26). Techniquement la variable A est rigoureusement équivalente à @ (1), B à @ (2), C à @ (3), ..., Z à @ (26). De même, il existe cette possibilité pour les variables alphabétiques A$ à Z$: @$ (1) à @$ (26).

On le voit donc, seul le nom change : A peut se nommer @ (1). Pas sorcier, mais quel intérêt ? Essayez :

```
10 : FOR A = 2 TO 26
20 : INPUT "NOMBRE ?";@(A)
30 : NEXT A
40 : END
```

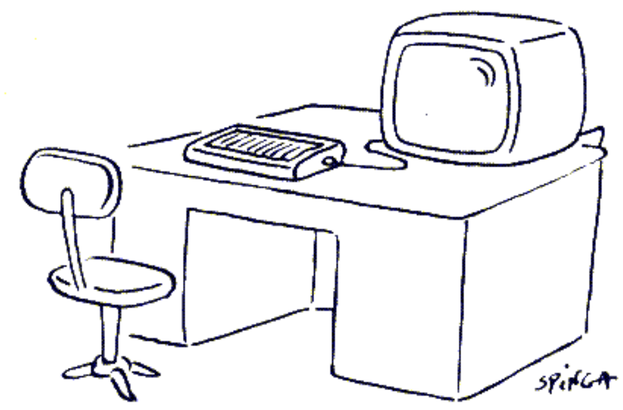
RUN, et ce petit programme demande successivement 25 nombres qui seront rangés automatiquement dans les variables B à Z (c'est-à-dire @ (A) pour A valant successivement 2 à 26).

Tout l'intérêt de ce tableau de variables est donc de permettre l'utilisation indirecte (par l'intermédiaire d'une variable, ici A) de toutes les variables dites « fixes » A à Z et A$ à Z$.

Jean-Christophe KRUST

LOGIQUE EN BASIC

POUR le Basic aussi, ce qui est vrai est vrai, ce qui est faux est faux. Mais où sont le vrai et le faux ?



Bien que n'étant pas reconnus en tant que tels par la plupart des Basics disponibles sur les ordinateurs, les booléens peuvent faciliter l'exécution de certains programmes, et servir efficacement dans des algorithmes délicats...

Qu'est-ce qu'un booléen ? C'est une variable susceptible de prendre deux valeurs : VRAI ou FAUX, à l'exclusion de toute autre valeur. Les langages structurés (PASCAL, LSE, LOGO...) manipulent les booléens comme ils le font d'autres types de variables. Basic, lui, ne connaît que des « pseudo-booléens » qui revêtent une forme numérique particulière permettant bien des astuces de programmation.

Un essai pour commencer

Pour connaître les deux valeurs booléennes utilisées par Basic, vous pouvez tout simplement taper au clavier de votre machine favorite : PRINT 5=5, et la réponse sera −1 (ou 1 selon la machine utilisée) ; PRINT 8>12 aura pour résultat l'affichage de 0.

Si vous faites quelques essais avec des valeurs différentes, vous constaterez qu'il n'existe que deux réponses possibles, qui dépendent de la proposition :

- si celle-ci est vraie, la réponse est −1 ou 1 ;
- si elle est fausse, la réponse est 0.

Le Basic reconnait donc le vrai du faux, mais ne sait pas les désigner par les mots « true » (vrai) et « false » (faux) comme d'autres langages plus évolués. Ses « pseudo-booléens » sont donc plutôt des opérateurs de relation...

Et ils sont simples à utiliser. Ainsi, il est parfaitement possible d'intégrer des opérateurs de relation dans les calculs numériques les plus traditionnels. Essayez par exemple : PRINT (8=8)*3 et vous obtiendrez −3 (ou 3). Mieux encore, essayez : A=8 : B=9 : PRINT (A>B)*10.

Ici, la réponse est 0, puisque A est inférieur à B.

Cette propriété peut vous permettre d'économiser beaucoup de place en mémoire, et de gagner de la vitesse à l'exécution de certaines parties de programme. L'exemple qui suit illustre ce gain. Il affiche le double d'une valeur entrée au clavier, si celle-ci dépasse 5, et affiche 0 dans le cas contraire.

```
1 — Programme classique
 5 B=0
10 INPUT A
20 IF A>5 THEN B=A*2
30 PRINT B
40 GOTO 5

 2 — Programme plus « sioux »
10 INPUT A
20 PRINT ABS (A>5)*A*2
30 GOTO 10
```

Si votre ordinateur produit 1 quand la relation est vraie, il est possible de supprimer le ABS dans la ligne 20. Il est même possible d'intégrer un opérateur booléen dans une ligne DEF du programme. Encore faut-il que votre Basic possède l'instruction DEF permettant de définir une fonction.

L'exemple ci-dessus devient alors :

```
 5 DEF FNF (X) = ABS (X>5)*X*2
10 INPUT A
20 PRINT FNF (A)
30 GOTO 10
```

Une application particulière

Dans les expressions IF...THEN du Basic, il est tout à fait habituel d'utiliser les opérateurs de relation. Dans une ligne comme : IF A<>0 THEN PRINT « DIFFERENT », l'ordinateur évalue l'expression proposée. Si le résultat du test est vrai, la valeur rendue est −1 (ou 1), et la machine exécute la fin de la ligne. Sinon, elle passe à la ligne suivante... Il existe toutefois une faille dans la logique de l'ordinateur, qui rend possible des lignes telles que : IF A THEN PRINT « DIFFE RENT ».

En effet, la variable A n'étant comparée avec rien, le Basic la prend comme le résultat de l'évaluation de l'expression. Certains Basics considèrent que la valeur booléenne de A est 0 (c'est-à-dire « faux ») dès que A est inférieur ou égal à 0, d'autres seulement quand A est égal à 0. Ils passent alors à la ligne suivante.

Aussi, il est parfaitement possible de remplacer dans vos programmes tous les « <>0 » ou « >0 », selon le Basic, par... rien du tout !

L'économie en octets peut être ainsi appréciable (ce que vous constaterez facilement en contrôlant dans vos programmes la fréquence d'apparition de cette formule) et l'exécution du programme pourra être accélérée de façon spectaculaire.

Lors de l'évaluation d'un calcul faisant intervenir les pseudo-booléens, la priorité va aux calculs numériques. Ils sont effectués avant les calculs logiques. L'utilisation des parenthèses est donc primordiale dans certaines évaluations : PRINT 30=15+15 rend la valeur 1 ou −1, parce que l'addition est effectuée d'abord ; PRINT (30=15)+15 rend 15 qui est la somme de 0 (faux) et de 15.

Le champ d'application des pseudo-booléens est pratiquement illimité. Seule l'imagination peut faire défaut... Certains ne manqueront pas de leur reprocher de compliquer la maintenance des programmes en les rendant plus difficiles à comprendre. Mais toute médaille a son revers !

Robin BOIS

LES DIX TESTS DE LIST

POUR évaluer en partie les performances d'un ordinateur donné, et plus précisément les qualités de son Basic, nous avons retenu (provisoirement) dix tests. Ils permettent de mesurer la vitesse avec laquelle la machine exécute ses appels de sous-programmes et diverses instructions de traitement de chaînes de caractères, de calculs arithmétiques, d'opérations sur les tableaux de variables, etc.

Cette batterie de tests n'est évidemment pas parfaite, mais elle permet de se faire une première idée. Pour certains ordinateurs, il faudra procéder à des adaptations. Rien ne dit d'ailleurs qu'à l'avenir nous ne serons pas conduits à modifier l'un ou l'autre de ces tests pour obtenir un outil permettant une mesure plus précise, plus complète. Vos suggestions sont les bienvenues.

LIST

Chaque test est constitué d'une boucle parcourue dix mille fois, y compris le test 3 (voir la description des tests page suivante) où la boucle est répartie en trois FOR-NEXT imbriqués.

Le premier d'ailleurs consiste en une simple boucle vide, et donne une première idée de la rapidité du Basic. Grâce à lui vous pouvez le plus souvent déterminer assez précisément la durée des opérations qui font l'objet des autres tests. Une soustraction suffit : durée du test moins durée des 10 000 boucles. Attention cependant : on a vu récemment des Basics particulièrement rusés qui (peut-être dans l'optique de ce style de test) sautent l'exécution réelle de la boucle lorsqu'un NEXT suit directement son FOR. Le temps devient voisin de zéro, et le test n'a bien sûr plus aucune signification !

Le deuxième test contient une imbrication de deux sous-programmes. A première vue, il semblerait qu'un simple GOSUB-RETURN ait suffit, mais il s'avère que certains Basics font leur premier GOSUB extrêmement rapidement et ralentissent dès qu'ils rencontrent une imbrication avec un deuxième GOSUB. Et rares sont les vrais programmes ne comportant aucune imbrication...

Dix mille tours par boucle

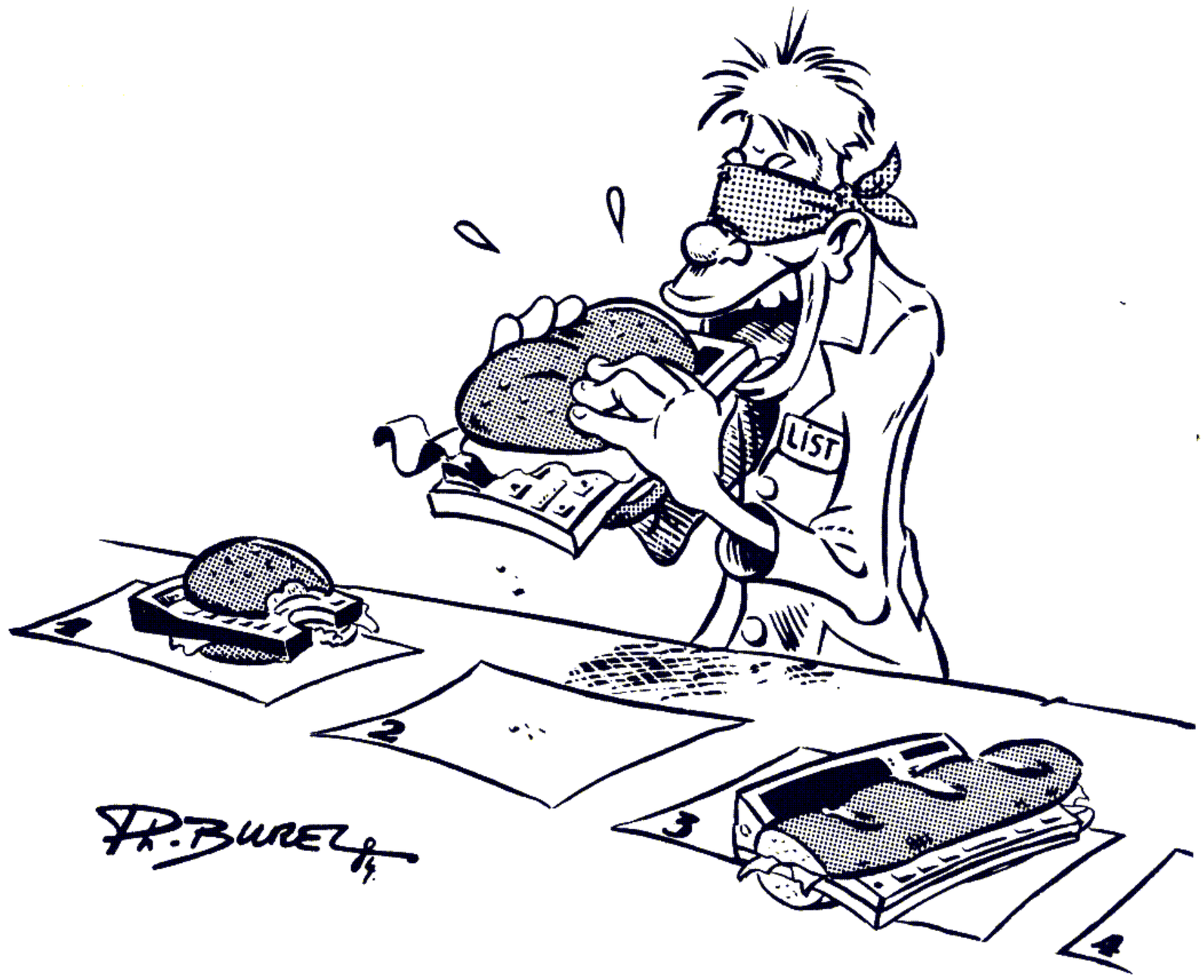
La rapidité d'accès à des variables faisant partie de tableaux à deux dimensions est mesurée par le test 3. Pour que la clause des « dix mille » soit respectée, nous avons été conduits à découper en trois FOR-NEXT. Nous aurions pu utiliser seulement deux bou-

Les dix tests appliqués au MO5 et au Spectrum

Test	Thomson MO5	Sinclair Spectrum
1 - Boucle vide	16	42
2 - Sous-programmes	45	106
3 - Matrice	87	115
4 - Chaînes de caractères	135	195
5 - Arithmétique	122	132
6 - Calcul scientifique	376 (6 chiffres)	1 180 (8 chiffres)
7 - Affichage	295	243
8 - Tracé graphique	1 430 (320 × 200 pts)	610 (256 × 176 pts)
9 - Ecriture fichier	940 (cassette)	
10 - Lecture fichier	940 (cassette)	

(tous les temps sont exprimés en secondes)

LES DIX TESTS DE LIST



cles de 1 à 100, dites-vous ? Mais peu de petits ordinateurs acceptent à l'heure actuelle des DIM A(100,100).

Les quatrième, cinquième et sixième tests tournent respectivement autour de la manipulation des chaînes de caractères, des calculs arithmétiques et des calculs scientifiques. Selon l'utilisation de votre machine, l'un de ces tests sera plus important que les autres. Il est à noter que dans le test 6 nous avons choisi deux fonctions parmi les plus utilisées : sinus (les temps mis par les autres fonctions trigonométriques sont du même ordre) et logarithme (les calculs de puissance utilisent les log). Comme ce test favorise les Basics calculant avec peu de chiffres significatifs, il n'a de sens que si l'on tient également compte de la précision des calculs.

Test 1 - Boucle vide

```
10    FOR I = 1 TO 10000
20    NEXT I
30    END
```

Test 2 - Sous-programmes

```
10    FOR I = 1 TO 10000
15    GOSUB 100
20    NEXT I
30    END
100   GOSUB 110
110   RETURN
```

Test 3 - Matrice

```
 5    DIM A(10,10)
10    FOR I = 1 TO 10
12    FOR J = 1 TO 10
13    FOR K = 1 TO 100
15    A(I,J) = K
17    NEXT K
18    NEXT J
20    NEXT I
30    END
```

Test 4 - Opérations sur les chaînes de caractères

```
 5    A$ = « LISTEST »
10    FOR I = 1 TO 10000
15    B$ = LEFT$(A$,2) + MID$
      (A$,3,3) + RIGHT$(A$,2)
20    NEXT I
30    END
```

Test 5 - Arithmétique

```
10    FOR I = 1 TO 10000
15    J = I*7 + 3/I
20    NEXT I
30    END
```

Test 6 - Calcul scientifique

```
10    FOR I = 1 TO 10000
15    J = SIN (LOG(I))
20    NEXT I
30    END
```

Test 7 - Affichage

```
10    FOR I = 1 TO 10000
15    PRINT CHR$(11) ; « LIS
      TEST » ; I
20    NEXT I
30    END
```

Test 8 - Tracé d'une ligne graphique

```
10    FOR I = 1 TO 10000
15    LINE (0,0) – (319,199)
20    NEXT I
30    END
```

Test 9 - Ecriture de fichiers

```
 5    A$ = « LISTEST »
 6    OPEN « O », #1, « FICHIER »
10    FOR I = 1 TO 10000
15    PRINT #1, A$
20    NEXT I
25    CLOSE
30    END
```

Test 10 - Lecture de fichiers

```
 6    OPEN « I », #1, « FICHIER »
10    FOR I = 1 TO 10000
15    INPUT #1, A$
20    NEXT I
25    CLOSE
30    END
```

L'écriture des quatre derniers tests peut varier selon les Basics. En voici la description :

• Test 7. Pour éviter qu'il ne mesure la durée de déroulement sur l'écran (en anglais : *scrolling*) plutôt que celle de l'affichage, nous écrivons dix mille fois au même endroit. Ainsi le rôle du CHR$(11) du MO5 est simplement de remonter le curseur d'une ligne vers le haut. Ce caractère de contrôle peut varier selon les machines.

• Test 8. La commande graphique la plus employée étant le tracé de ligne, ce test dessine la diagonale allant du haut gauche au bas droit de l'écran. Là aussi, il nous faut préciser la définition de l'écran puisque les machines à faible résolution seront favorisées.

• Tests 9 et 10. Ces écritures et lectures de fichiers séquentiels sont importantes pour bien des applications. Le fichier à manipuler est gros : il comporte en effet 10 000 × (7 caractères + caractère de fin de chaîne), soit un total de 80 Ko. Il nous faut bien sûr préciser s'il est sur disquette ou cassette. Quel débit moyen en déduisons-nous pour les fichiers sur cassette du MO5 ? Nous obtenons (80 000 × 8)/940, soit environ 680 bits par seconde. La différence avec les 120 bauds annoncés par le constructeur s'explique par le découpage en blocs de 256 octets, l'intervalle entre les blocs n'étant pas négligeable.

Christian BOYER

FORTH, UN LANGAGE ÉPATANT

CET article est le premier d'une série destinée à vous familiariser avec un langage curieusement peu connu en France. L'auteur est un passionné du Forth, et il espère bien vous faire partager son enthousiasme. Si vous avez des suggestions ou des critiques à lui adresser, n'hésitez pas : à vos claviers !

Pour moi, tout a commencé dans une boutique de Marseille, où j'étais venu fureter, glaner des informations, de la « doc ». J'avais alors décidé d'acheter un ordinateur, mais je ne connaissais à peu près rien à l'informatique. Je savais seulement que les ordinateurs consomment de l'électricité, qu'ils sont dotés d'une mémoire et que l'on peut leur demander beaucoup de choses à condition de savoir s'y prendre...

A quarante-huit ans, quand on voit autour de soi beaucoup de jeunes se passionner pour une activité très récente et très prometteuse, on a le choix entre deux attitudes : se réfugier dans une critique désabusée ou se lancer à la découverte. Si l'on adopte la seconde solution, on doit oublier son amour propre qui risque d'être malmené pendant quelque temps... C'est ce que j'avais décidé de faire.

Le décor Forth

Dans cette boutique de Marseille, le vendeur auquel j'allais avoir affaire était compétent et il souriait, mais sans suffisance. J'étais un client assez intéressé pour être « intéressant ». Il me montra ce que la maison pouvait faire pour moi, c'est-à-dire me vendre un ordinateur avec une ou deux unités de disquette et quelques logiciels. Le tout, bien entendu, pour un prix très avantageux ! Coup de chance, il y avait justement une promotion sur un fruit anglais entamé...

Poliment, je pris quelque recul en avertissant que j'allais réfléchir, que je ne voulais pas me décider sur un coup de tête. Et c'est à ce moment-là que survint l'événement qui allait déterminer la suite de mon avenir informatique. Un autre client, plus jeune que moi, et dont l'aspect m'inspirait confiance, me demanda :

— Vous ne connaissez rien à l'informatique, si j'ai bien compris ?

— Rien de rien. Mais je suis plein de bonne volonté...

— Me permettez-vous un conseil ?

— Je vous écoute.

— Laissez tomber le Basic. Choisissez le Forth, d'entrée de jeu. Le Forth...

— Comment cela ?

— Mais oui, le Forth. C'est un langage extraordinaire. Il est à peu près dix fois plus rapide que le Basic et cinq à six fois moins gourmand en mémoire. Il a été créé il y a une dizaine d'années par un scientifique américain qui n'était pas satisfait de ce qu'il pouvait faire avec les autres langages.

— Fichtre ! Mais ça doit être beaucoup plus difficile que le Basic ?

— Pas du tout. Si c'était le cas, je ne vous donnerais pas ce conseil.

— Excusez-moi, sans être indiscret, vous êtes de la maison ?

— Pas du tout : je suis biologiste et je travaille dans la recherche.

Cinq minutes plus tard, au bar-tabac du coin, il continuait :

— Croyez-moi, si les gens savaient ce qu'est le Forth, ce qu'il peut leur apporter, ils laisseraient souvent le Basic de côté...

— Ma parole, mais vous ne jurez que par Forth !

— Il y a de ça. Je programmais en Fortran, en Cobol et en Basic. Eh bien, progressivement, je délaisse ces langages pour le Forth que j'ai découvert depuis peu. Voilà pourquoi je vous recommande de débuter avec le Forth. En fin de compte, c'est aussi facile que le Basic, mise à part une petite difficulté sur laquelle je reviendrai. Forth est rapide, il est proche de la machine avec laquelle vous allez dialoguer, et il consomme peu de mémoire. De plus, il est construit autour d'un ensemble de mots avec lesquels vous pouvez définir votre propre vocabulaire.

— Sans comprendre tout ce que vous me dites, je suis assez séduit. Mais quelle machine choisir ? Je n'ai aucune intention de me ruiner, vous savez ?

— Il n'est pas question de vous ruiner. Beaucoup d'ordinateurs destinés aux particuliers ont un Basic résident, c'est-à-dire inscrit en mémoire morte. Mais ces mêmes machines ont très souvent un Forth en option, soit sur cassette, soit sur disquette et parfois même en cartouche de mémoire morte. Il existe aussi quelques ordinateurs dotés d'origine d'un Forth en mémoire morte.

— Lesquels ?

— Le Jupiter Ace, par exemple, qui est anglais, très astucieux et bon marché. Et puis l'Hector HRX, français, plus cher, mais remarquable...

FORTH, UN LANGAGE ÉPATANT

— Il y en a un qui a votre préférence ?

— J'utilise les deux.

— Bien. J'y vois un peu plus clair. Mais tout à l'heure, vous m'avez parlé d'une petite difficulté...

— Ah, c'est vrai, j'oubliais. Comment fait-on une addition, selon vous ?

— Vous êtes sérieux ?

— Mais bien sûr.

— Eh bien, selon moi, deux plus deux, cela fait quatre. C'est toujours valable, n'est-ce pas ?

— Oui, mais en Forth, cette addition se fait et s'écrit « deux, espace, deux, espace, plus, et enfin point ». C'est une habitude à prendre...

— Aïe ! Je n'y comprends plus rien du tout. Qu'est-ce que vous voulez dire ?

— Exactement la même chose que deux plus deux font quatre. Seulement les machines Forth fonctionnent selon la notation polonaise inverse. Elles ne sont pas les seules. Les utilisateurs des calculatrices Hewlett-Packard se sont habitués à cette façon d'écrire une opération et la plupart d'entre eux l'estiment excellente. Mais indépendamment de cela, cette notation particulière est très importante, car elle explique en grande partie pourquoi le Forth est si rapide.

Cet entretien m'a trotté dans la tête plusieurs jours durant, et je me suis finalement décidé à me mettre au Forth. Je ne l'ai pas regretté. Pourquoi ne feriez-vous pas comme moi ? Au risque d'insister un peu, je vous dirais encore que je me suis mis au Forth récemment, et que mon avantage sur vous n'est donc pas bien grand. Je vous dirais aussi que je ne suis plus un gamin. Si vous avez la chance de l'être encore, vous progresserez plus vite que moi. Sinon, vous êtes dans mon cas, et vous n'avez pas de temps à perdre. Alors, on y va ? On regarde le Forth d'un peu plus près ? Oui, OK ?

Ainsi débuta l'expérience

Eh bien, c'est déjà fait : nous venons de commencer, car OUI ou OK (selon les machines) sont deux des mots Forth que vous rencontrerez le plus souvent. Ce n'est pas une exclusivité de Forth, mais dans ce langage qui dialogue en permanence avec l'utilisateur, ils sont chargés de sens. Ce n'est pas un OK de pure forme : la machine vous signale ainsi que ce que vous venez de faire est licite, exécutable.

Forth chargé depuis sa cassette ou sa disquette (s'il n'est pas résident en mémoire morte), l'ordinateur vous renvoie donc OK. Nous allons nous livrer à une première expérience. Mais auparavant, une précision s'impose pour mieux comprendre la suite. A chaque fois que vous rencontrerez le signe "°", cela signifiera que l'on doit frapper la barre d'espacement. Quand vous aurez bien compris l'importance de l'espace en Forth, nous nous dispenserons de ce signe. Mais il faut dès à présent se pénétrer de l'idée que l'espace est un mot du Forth.

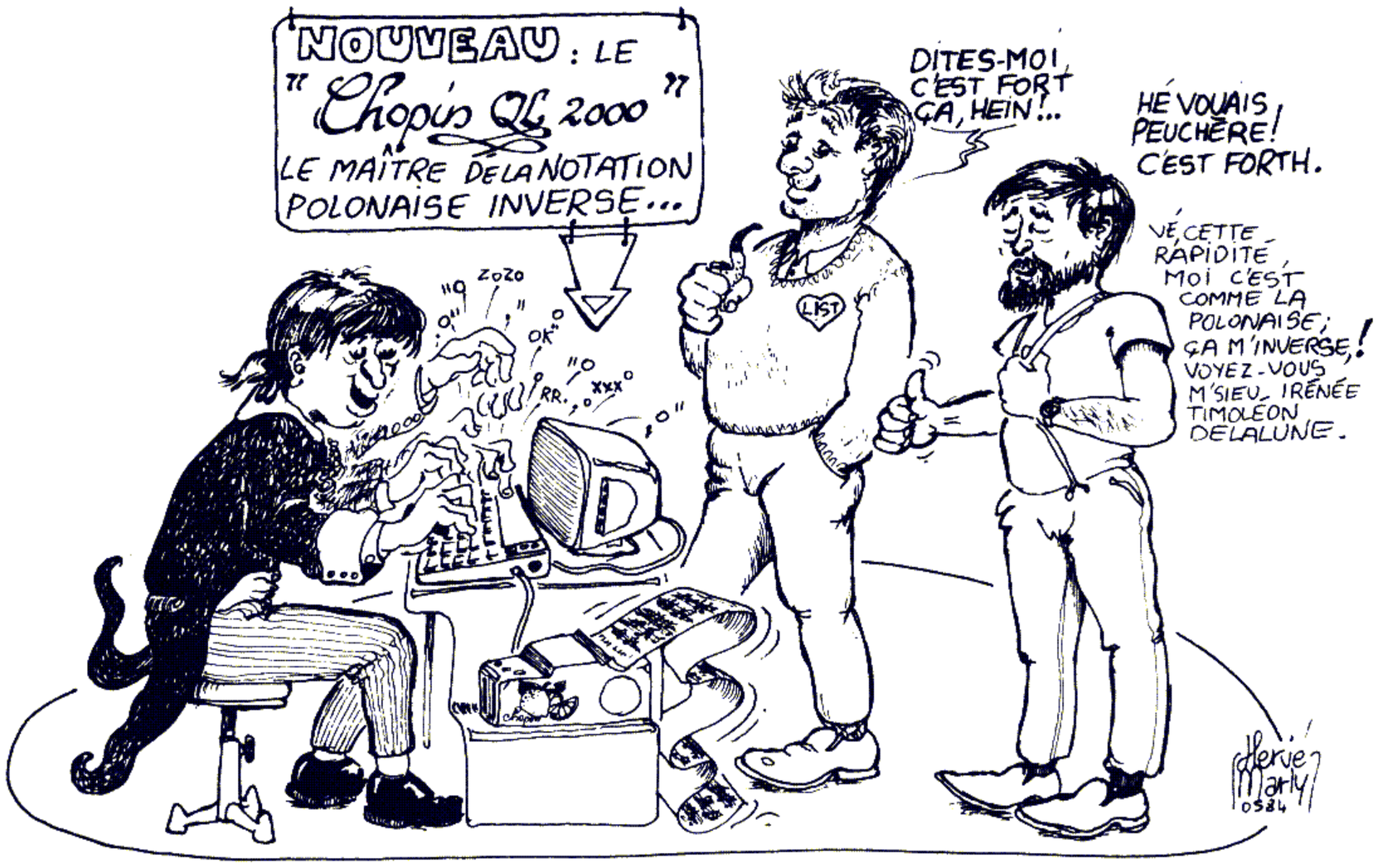
Vous frappez donc au clavier :
:°SALUT°."°BONJOUR°LES° AMIS°"°;RR

Le RR final représente — vous l'aviez sans doute deviné — un appui sur la touche RETURN ou ENTER, selon la machine. Une fois RR exécuté, vous retrouvez à l'écran OK et le curseur. Maintenant, vous frappez simplement SALUT°RR

La machine vous renvoie BONJOUR LES AMIS OK, et vous retrouvez votre précieux curseur (dont la perte dans les entrailles de la machine ne présage jamais rien de bon !). Prenons les signes un par un.

: (les deux-points). Mot Forth qui débute une nouvelle définition du dictionnaire et l'ajoute en tête de tous les mots que connaît déjà votre machine.

° (l'espace). Joue le rôle de délimiteur et indique à l'ordinateur quand un mot est terminé, prêt pour l'exécution.



Comme un mot Forth peut comporter un ou plusieurs signes, il faut bien fournir au processeur un repère pour séparer les mots.

SALUT° Voilà la suite de caractères dont vous voulez faire un nouveau mot Forth. La situation est analogue à celle de tout dictionnaire (Larousse, Littré, Robert, etc.). Vous venez de créer une tête de rubrique, et sa définition va suivre. Remarquez, bien entendu, l'espace qui termine le mot.

.‘‘° Cette fois-ci, il n'y a pas d'espace entre le point (.) et les guillemets (‘‘), vous l'avez bien noté. Ce mot Forth s'appelle ‘‘point-guillemets’’. Il donne à la machine l'ordre d'afficher sur l'écran la chaîne de caractères qui suit l'espace. Ce mot est donc composé de trois signes, le point, les guillemets et l'espace. J'insiste ici, car la majorité des difficultés rencontrées par les personnes qui débutent en Forth vient de l'oubli de l'espace à la fin de chaque mot (croyez-en mon expérience personnelle...).

BONJOUR°LES°AMIS° Rien à signaler, c'est la suite de caractères à afficher.

‘‘° Fin de la suite de caractères. Vous avez compris que le point-guillemets (.‘‘) appelle obligatoirement un guillemets-espace (‘‘°) à l'autre bout de la chaîne : il faut bien que ce message ait une fin !

; Vous noterez que, pour une fois, il n'y a pas d'espace entre le point-virgule et le RR. Cela s'explique facilement, puisque le point-virgule signifie précisément ‘‘fin de la définition’’.

Notre premier programme Forth est terminé et, détail qui a une certaine importance (non ?), il marche !

Les premiers seront les derniers

Frappons maintenant au clavier un autre mot que nous allons souvent utiliser. Il s'agit de VLIST. Avec VLIST, on fait afficher par l'ordinateur la totalité des mots qui constituent le dictionnaire. Ce sont les derniers entrés qui apparaissent en premier. On voit donc s'afficher : SALUT, puis FORTH, etc. Vous avez donc bien enrichi le dictionnaire d'un nouveau mot (SALUT), et à chaque fois que vous entrerez SALUT, votre machine vous retournera BONJOUR LES AMIS. Vérifions-le en créant deux autres mots. Imaginons que votre nom soit Irénée Timoléon Delalune, et frappons au clavier :

:°SALUT°CR°.‘‘°BONJOUR°MON° CHER°’’°CR°;RR

:°XXX°.‘‘°IRENEE°TIMOLEON°DE LALUNE°’’;RR

Et pendant que nous sommes en veine de création, ajoutons :

:°ZOZO°CLS°SALUT°XXX°;RR

Remarquons tout de suite que nous avons employé deux mots nouveaux :

CR c'est-à-dire passage à la ligne ou retour-chariot (en anglais *Carriage Return,* d'où le CR) ;

CLS qui efface tout ce qui est affiché à l'écran (abréviation de l'anglais *Clear Screen*).

Nous sommes prêts pour faire exécuter ZOZO°RR. Docilement, votre ordinateur vous renvoie :

BONJOUR MON CHER
IRENEE TIMOLEON DELALUNE
OK

Faites vos mots...

Regardons d'un peu plus près ce que nous avons demandé à Forth. Nous avons créé trois mots SALUT, XXX et ZOZO. Mais les choses ont évolué rapidement. Pour définir le troisième mot, nous avons utilisé les deux autres et il a suffi de rentrer le troisième (ZOZO) pour que l'ordinateur aille rechercher dans le dictionnaire la signification de CLS, SALUT, XXX, qui définissent ZOZO. Tout s'est passé de la même façon que lorsque vous tombez sur un mot dont vous ignorez le sens. Vous vous reportez à votre dictionnaire favori, et si, dans la définition du mot, vous trouvez de nouveau un terme difficile, vous allez chercher à sa rubrique dans le même dictionnaire. Et ainsi de suite... Forth procède exactement comme vous et moi.

Que donne VLIST maintenant ? Eh bien il affiche successivement ZOZO, XXX, SALUT, SALUT, FORTH, etc. Et ne nous étonnons surtout pas de trouver deux fois de suite le mot SALUT. C'est que le premier n'a pas été oublié par l'ordinateur. Quand nous demandons l'exécution de ZOZO, la machine va chercher le dernier SALUT entré, celui qui vous salue, vous, en particulier. Et si elle procède ainsi, c'est parce que ce SALUT est au sommet de la ‘‘pile’’, un concept sur lequel nous ne manquerons pas de revenir en détail une prochaine fois.

Nous aurons beau nous obstiner en tapant SALUT°RR, nous obtiendrons systématiquement : BONJOUR MON CHER OK. Si nous voulons retrouver le BONJOUR LES AMIS OK du départ, il ne nous reste plus qu'à demander FORGET°SALUT°RR... Mais n'allons pas trop vite. Il faut en effet que vous sachiez bien ce qui se produit quand on emploie FORGET (en français : oublie). Cet ordre fait oublier par la machine le mot désigné (ici le dernier SALUT entré au dictionnaire), mais aussi tous les mots qui ont été définis par la suite. Ainsi, dans notre exemple, FORGET°SALUT°RR fera perdre SALUT, mais aussi XXX et ZOZO.

Avant d'utiliser FORGET, il est prudent de vérifier par VLIST quels sont les mots qui vont être oubliés, rayés du dictionnaire. Souvent, il est intéressant de définir un nouveau mot, mais il arrive aussi que les limites imposées par la mémoire vive disponible obligent à ‘‘nettoyer’’ le dictionnaire des mots devenus inutiles.

Dans notre cas, réintroduire le dernier SALUT, XXX et ZOZO ne demande que peu de temps. Nous allons donc passer par le redoutable FORGET°SALUT °RR. Voilà qui est fait. Demandons maintenant SALUT°RR, et nous obtenons bien l'affichage de BONJOUR LES AMIS OK, comme au début.

Si nous faisons la contre-épreuve en entrant ZOZO°RR, nous ne serons pas surpris de voir s'afficher un message d'erreur dont la teneur variera selon la machine utilisée. L'ordinateur a recherché ZOZO dans son dictionnaire et ne l'a pas trouvé ; on lui a donc demandé l'impossible. Pas de surprise non plus du côté de VLIST°RR qui affiche désormais SALUT, FORTH, etc.

... la machine a les siens

Le mot FORTH qui apparaît grâce à VLIST est le premier du dictionnaire permanent de la machine. Cette partie du dictionnaire est à tout point de vue essentielle : elle regroupe les primitives et les définitions de base. Nous avons déjà rencontré plusieurs de ces primitives : les deux-points, le point-guillemets, les guillemets, le point-virgule, CR, CLS, FORGET et VLIST.

Alors, le Forth, est-ce si compliqué que cela ? Pour le prochain article, je vous réserve un petit programme de derrière les fagots qui vous donnera, je l'espère, une haute idée des possibilités que ce langage de programmation vous offre...

Bernard LAMBEY

VISITE GUIDÉE DANS LES RECOINS DE LA HP-41C

LA PROGRAMMATION SYNTHÉTIQUE

UNE « bogue », en jargon informatique, est une erreur de programmation.
Les programmeurs de chez HP ont mis grand soin à faire de la HP-41C une machine sophistiquée, évolutive, sérieuse...
Mais **la** *bogue est là, heureuse, énorme et subtile.*
Loin de diminuer l'intérêt du calculateur, elle en multiplie les attraits.
La porte s'ouvre grande sur un monde merveilleux où tout est possible.

La programmation synthétique n'est pas une nouveauté. On la doit à des explorateurs passionnés (1), mais tous ses secrets n'en ont pas encore été dévoilés. D'abord superbement ignorée par Hewlett-Packard, cette nouvelle programmation est maintenant reconnue, elle a fait toutes ses preuves.

Nous, nous débutons ici en exposant clairement le principe fondateur du « synthétisme » : le Cric, et en découvrant quelques-unes de ses applications.

Que le grand Cric me croque !

« Chargement dans le Registre alpha d'Instructions Codées », ou Cric pour les intimes, tel est le nom de la procédure de création d'instructions synthétiques. C'est aussi le « sésame », la clef du jardin caché. Sans attendre, créons-le en suivant les instructions de l'encadré ci-contre.

C'est fait ? Le Cric est donc maintenant assigné à la touche Σ+. Nous l'avons créé de toutes pièces, le Cric n'est pas programmable, seulement exécutable en mode de calcul. Si vous possédez le lecteur de cartes magnétiques, sauvegardez cette assignation à l'aide de WSTS. Au passage, notons qu'une seule face de carte est nécessaire : enregistrer deux fois la face 1, relire en l'introduisant puis en pressant la touche ← pour terminer l'opération.

Le Cric est comme un couteau qui permet de découper en tranches des instructions normales et, avec ces morceaux, tel un jeu d'assemblage, d'en construire de nouvelles : synthétiques.

Assignation du Cric à la touche Σ+

Avec une HP-41C nue de tout périphérique et module spécial (TIME autorisé), provoquer un Memory Lost (éteindre la HP, presser ←, puis ON et relâcher ←).

Faire :	**Lire :**
ASN (α) BEEP (α) Σ+	ASN BEEP 11
ASN (α) PACK (α) Σ–	ASN PACK –11
(PRGM)	00 REG 45
ENTER	01 ENTER
GTO..	00 REG 45
CAT 1 immédiatement suivi de R/S	END
(ALPHA)	
←	4094 ENTER
BST (attendre)	4093 ENTER
BST	4092 0
BST	4091 LBL 00
BST	4090 BEEP
←	4089 LBL 03
←	4088 LBL 08
C (en mode alpha)	4089 ᵀC
(PRGM) GTO..	0.0000

Le Cric est maintenant assigné à la touche Σ+. Jusqu'à NULL, en mode USER, presser cette touche et lire XROM 05,03.

Rien n'est vraiment créé, toutes les instructions synthétiques existent déjà dans les circuits de la machine, mais comme elles ne sont pas normalement disponibles pour l'utilisateur, il faut les retrouver par des voies détournées. Pour le comprendre, examinons ce qu'est une instruction pour la HP-41C.

Dans la mémoire, les instructions ne sont pas écrites en toutes lettres mais codées. Ainsi LOG correspond au code 86 et SIN au code 89. Il y a des instructions à un seul code telles celles-là mais il en existe aussi à deux codes : VIEW X, TONE 9 ou RCL 16, etc.

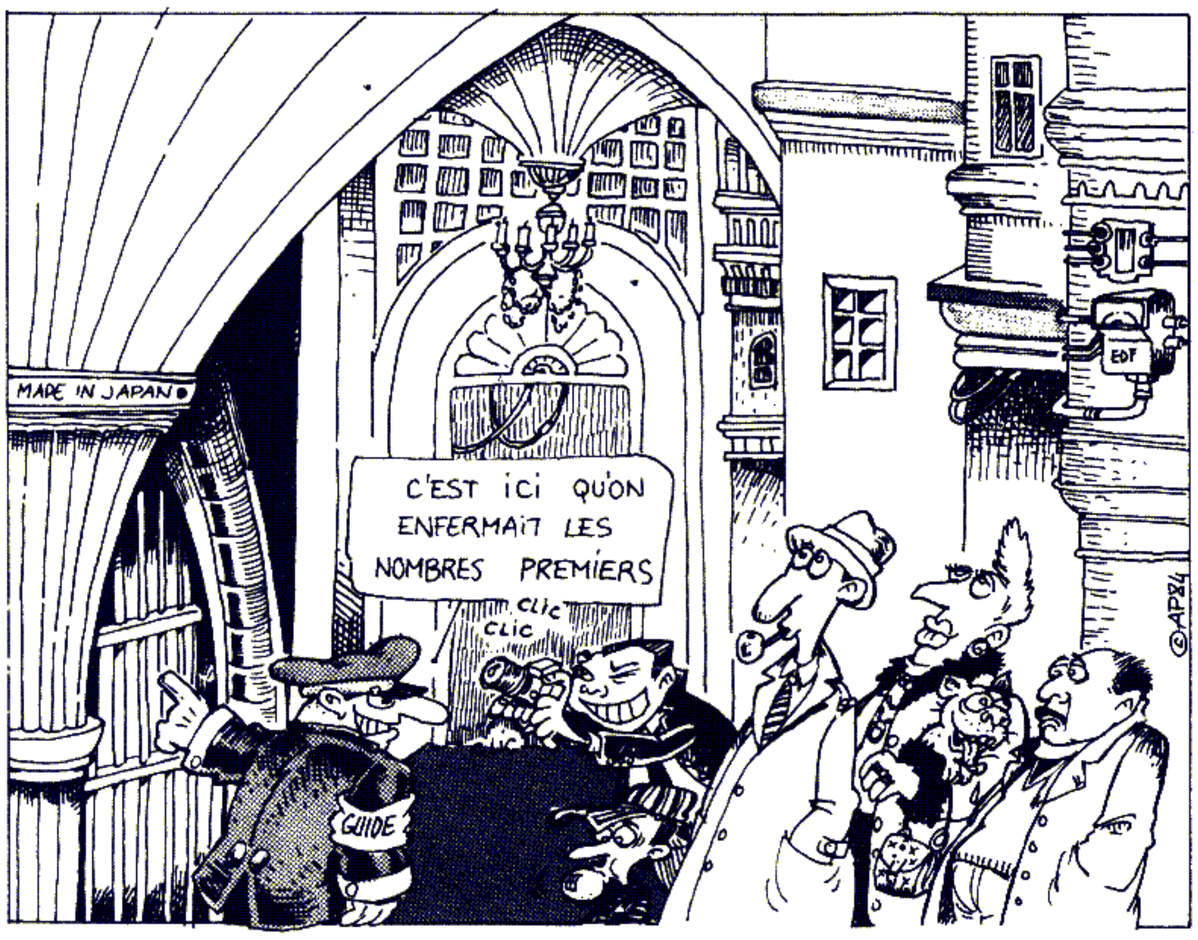
Dans ce cas, le premier code correspond au *corps* de l'instruction (RCL, TONE, VIEW, etc.) et le second code correspond lui à l'*argument* (X, 9, 16, etc.).

Et pourtant, elles existent

Si l'utilisateur avait la possibilité de programmer directement ces codes, comme en Basic avec la fonction POKE, le Cric n'aurait aucune utilité sinon pour les esthètes. Mais la HP rétorque imperturbablement NON EXISTENT lorsqu'on tape des fonctions telles RCL M ou d'autres fonctions synthétiques : elle ne les reconnaît pas.

Sachant qu'un code peut prendre une valeur comprise entre 0 et 255, il y a donc 256 corps d'instructions théoriquement possibles (moins en fait car

(1) Synthetic Programming on the HP-41C *de W.C. WICKES et dans* L'Ordinateur Individuel *n^os 24 à 28 et n° 31, les articles de* Philippe Descamps *et* Jean-Jacques Dhénin.



des lettres A, B, ..., et chiffres demandent aussi leur codage). Comme il y a des instructions de deux codes (on dira aussi : octets), voire plus, c'est à un minimum de 256 × 256 codages possibles de deux octets, soit 65536, que nous avons affaire ! La HP « officielle » est loin du compte !

Sans attendre, réalisons l'administration de la preuve. La HP placée en mode USER (le Cric est à Σ+ et PACK à Σ-), programmons :

```
01 1
02 RCL IND 16
03 RCL IND 16
04 RDN
```

BST deux fois pour se positionner sur le premier des RCL IND 16. En mode calcul exclusivement, exécuter le Cric. De retour en PRGM, effacer le RCL IND 16, SST, effacer RCL 00, SST : l'instruction synthétique RCL M est programmée !

Laissons le détail technique de la manipulation pour un prochain article, et contentons-nous d'en examiner la logique. On programmera toujours :

```
1
RCL IND 16
RCL IND préfixe
suffixe
```

où le préfixe détermine le *corps* de l'instruction à créer (16 pour RCL) et le suffixe détermine son *argument* (RDN pour M). Que 16 — en fait IND 16 — se transforme en RCL et RDN en M ne nous étonne plus car nous savons que ce qui importe n'est pas l'orthographe mais le code. Or, IND 16 possède le code 144 de même que RCL, RDN vaut 117 de même que l'argument M. C'est la position du code (première ou deuxième) dans l'instruction qui détermine sa fonction :

```
144 144 donne RCL, IND 16
145 144 donne STO, IND 16
144 117 donne RCL, M
117     donne RDN
etc.
```

Vous noterez que 144—128 = 16. En fait si IND 16 vaut 144 c'est parce que IND ajoute toujours 128 : ainsi IND 16 vaut 128 + 16, soit 144.

L'encadré ci-dessous donne une première liste des préfixes et suffixes intéressants ; notons-en les plus spectaculaires. Le préfixe 31 donne la fonction TONE. Rien d'extraordinaire... Mais il apparaît que 128 codes différents peuvent servir d'argument à cette fonction, générant ainsi autant de sons différents, du plus court au plus long, graves ou aigus ! (Préfixe 31, suffixes LBL 00 à LBL 14, 0 à 9, •, EEX CHS, RCL 00 à RCL 15, STO 00 à STO 15, +—*/ X<Y? X>Y? X<=Y? Σ+ Σ— HMS+ HMS— MOD % %CH P—R R—P LN X↑2 SQRT Y↑X CHS E↑X LOG 10↑X E↑X—1 SIN COS TAN ASIN ACOS ATAN DEC 1/X ABS FACT X≠0? X>0? LN1+X X<0? INT FRC D—R R—D HMS HR RND OCT CLΣ X<>Y PI CLST R↑ RDN LASTX CLX X=Y? X≠Y? SIGN X<=0? MEAN SDEV AVIEW CLD.)

Quelques exemples d'instructions synthétiques

Préfixe IND	Corps généré	Suffixe	Argument généré
16	RCL	RDN	M
17	STO	LASTX	N
18	ST+	CLX	O
19	ST—	X=Y?	P
20	ST*	X≠Y?	Q
21	ST/	SIGN	⊢
22	ISG	X<=0?	a
23	DSE	MEAN	b
24	VIEW	SDEV	c
25	ΣREG	AVIEW	d
26	ASTO	CLD	e
27	ARCL		
28	FIX		
29	SCI		
30	ENG		
31	TONE		

Nous découvrons aussi 11 registres nouveaux de la HP-41C : M, N, O, P, Q, ⊢, a, b, c, d et e. Ce sont les registres internes et la programmation synthétique est essentiellement l'art de leur manipulation (RCL, calcul, STO). Nous n'examinerons maintenant dans le détail que les quatre premiers qui servent aux manipulations alphabétiques. Ils sont sans danger pour la santé intellectuelle de la machine.

Ce n'est pas le cas des suivants qui permettent d'ordinaire à la HP de conserver des informations essentielles à son fonctionnement (variables du système) : leur manipulation inconsidérée peut la plonger dans un profond sommeil. Il devient nécessaire parfois de la laisser sans piles durant 24 heures... Alors n'y touchons pas aujourd'hui.

Le registre alpha

Un total de 24 caractères peut être conservé dans le registre alpha. Ce super-registre est en fait composé des registres internes M, N, O et P de la manière suivante :

Des caractères introduits dans le registre alpha seront rangés à la queue leu leu d'abord dans M puis N, O et P. Chaque nouveau caractère pousse les anciens vers la gauche à la manière d'une pile opérationnelle de 24 petits registres. Ainsi écrire LONGUE VIE A LIST donnera dans les registres internes :

et, par exemple, exécuter maintenant la fonction RCL M ramènerait en X les codes de la chaîne de caractères « A LIST ».

Ce qu'il importe de savoir c'est que ces registres alpha peuvent être employés tout à fait normalement par exemple pour économiser la pile opérationnelle ou des registres classiques (une mine d'astuces pour l'optimisation des programmes).

Enfin, avec STO, on découvrira que la HP-41C possède bien plus de caractères affichables que ne le dit son constructeur : de quoi agrémenter les jeux de pendus de petits bonshommes et de potences !

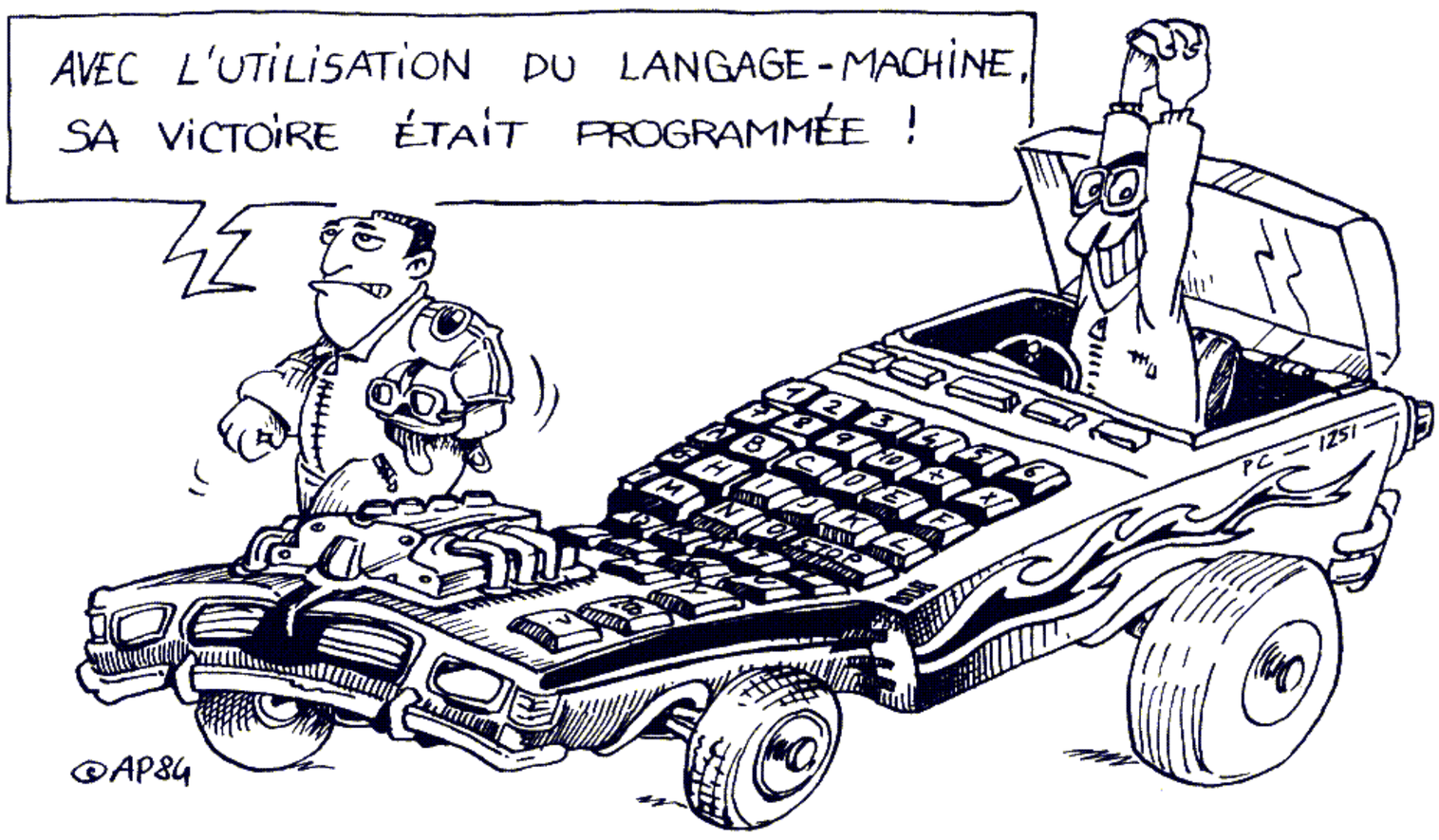
Mais, patience...

Jean-Christophe KRUST

NETTEMENT PLUS RAPIDE LE LANGAGE-MACHINE

(PC-1251)

ON le sait, le langage-machine est beaucoup plus rapide que le Basic. On le sait, oui. Mais tant que l'on ne l'a pas constaté « de visu », on ne se doute pas à quel point c'est vrai. Faites l'expérience sur votre poquette. Il y a gros à parier que vous ne serez pas déçu...



La dernière version du manuel français qui est fourni lors de l'achat d'un PC-1251 donne des indications précieuses si l'on veut que, pendant l'exécution d'un programme, l'afficheur du poquette ne soit pas aveugle.

On y apprend, entre autres choses, que les 24 matrices de l'afficheur (5 colonnes de 7 points pour chacune d'entre elles) sont divisées en deux zones distinctes.

Que de points à l'écran

La première de ces deux zones comprend les douze premières matrices d'affichage, soit un total de 60 colonnes (12 × 5 = 60) qui sont mémorisées aux adresses 63488 à 63547 (en hexadécimal, &F800 à &F83B).

Pour les 12 matrices suivantes, on doit compter à rebours : la dernière colonne de l'afficheur est en 63552 (&F840), et la première colonne de la treizième matrice est en 63611 (&F87B). Au total, c'est donc en 120 octets de mémoire vive que se trouve codé ce que le 1251 peut afficher.

Dans la même version du manuel, on apprend qu'il est possible de maintenir un message affiché pendant que le poquette exécute un programme. On obtient cette persistance de l'affichage avec un CALL &11E0 et l'on y met fin avec un CALL &11E5. Ainsi, pour remplir l'écran de votre poquette d'un même motif, vous n'avez qu'à essayer ce programme en Basic :

```
10 INPUT « MOTIF ? » ;A
20 CALL &11E0
30 FOR I = 0 TO &3B : POKE
   &F800+I,A : NEXT I
40 FOR I = 0 TO &3B : POKE
   &F840+I,A : NEXT I
50 GOTO 50
```

Le Basic prend son temps...

A la question « MOTIF ? », répondez par un nombre compris entre 1 et 127, et observez le résultat : il faut une douzaine de secondes pour que l'écran affiche ce que vous lui avez demandé... Accessoirement, on remarque que le Basic utilise pour ses besoins propres les octets &F800, et &F840 à &F847. Les colonnes correspondant à ces mémoires affichent en effet autre chose que prévu.

... mais pas le langage-machine

Essayons maintenant d'obtenir le même résultat en langage-machine. Pour cela, nous utiliserons FILD dont la fonction est de remplir un bloc de I + 1 octets à partir de l'octet pointé par DP, ces I + 1 octets prenant la valeur contenue dans l'accumulateur A. L'équivalent des lignes 30 à 40 du programme Basic précédent donne en LM (langage-machine) la liste de la page suivante.

```
02  FF      LIA   FF      ;A = FF
00  3B      LII   3B      ;I = 3B
10  F800    LIDP  F800    ;DP = F800
1F          FILD          ;(DP) = A,(DP + 1) = A, (...), (DP + I) = A
00  3B      LII   3B      ;I = 3B
10  F840    LIDP  F840    ;DP = F840
1F          FILD          ;(DP) = A,(DP + 1) = A, (...), (DP + I) = A
37          RTN           ;FIN
```

D'où le programme que je vous invite maintenant à entrer au clavier :

```
10 POKE &C100, &02, &FF, &00, &3B, &10, &F8, &00, &1F
20 POKE &C108, &00, &3B, &10, &F8, &40, &1F, &37
30 INPUT « MOTIF ? » ;A : POKE &C101, A
40 CALL &11E0
50 CALL &C100
60 GOTO 60
```

Le CALL &C100 de la ligne 50 mène au même résultat que les lignes 30 et 40 du programme Basic. Mais vous avez sans doute remarqué une « petite » différence, j'en suis sûr. Les choses se passent si vite maintenant que vous n'êtes plus capable de chronométrer la vitesse avec laquelle votre programme est exécuté. C'est devenu fulgurant.

Remplacez maintenant la ligne 30 par 30 POKE &C101, RND 127 ; et la ligne 60 par 60 GOTO 30. Exécutez le nouveau programme : le PC-1251 vous montre à l'afficheur des phénomènes auxquels vous n'étiez pas habitué.

Bernard BOUVIER

POUR ENCHAÎNER VOS PROGRAMMES

SUR COMMODORE

SI votre programme est trop long pour tenir dans la mémoire de votre ordinateur, le remède est simple. Découpez-le en plusieurs morceaux. Le premier morceau appellera le deuxième, et ainsi de suite. Bien sûr, il y a quelques précautions à prendre... Quand on les connaît, tout se passe bien.

A quoi bon chaîner les programmes ? En fait, les raisons ne manquent pas, mais j'en vois deux qui paraissent très pratiques.

LOAD passe le relais

En premier lieu, c'est grâce à cette technique que l'on pourra, à partir d'un programme de « menu », appeler le programme présent sur une disquette. Autre application, bien plus importante : c'est le chaînage qui permettra de faire exécuter par la machine des programmes beaucoup trop longs pour la mémoire disponible. Encore faut-il, dans ce dernier cas, que le passage d'une portion de programme à la suivante n'entraîne pas la perte des variables...

L'instruction CHAIN, en tant que telle, ne figure pas dans le Basic des machines Commodore, mais peu importe. Le manuel de l'utilisateur est très discret sur cette question, il indique seulement qu'un LOAD placé dans un programme provoque le chargement et l'exécution du programme spécifié, sans perte des variables. Ce n'est pas vrai dans tous les cas, et l'exemple que nous allons développer nous fournira sur ce point des indications très utiles. Cet exemple, je vous invite à le taper sans attendre au clavier de votre machine favorite.

Premier temps de la manœuvre, mettre en mémoire et sauvegarder sur disquette les quatre lignes que voici :

```
10 REM PROGRAMME 1
20 REM CE PROGRAMME EST LE
   PLUS LONG
30 INPUT « VARIABLE Z » ; Z
40 LOAD « PROGRAMME 2 », 8
```

Vous devez maintenant entrer au clavier un second programme (plus court, remarquez-le, que le premier) et le sauvegarder aussi sur disquette sous le nom de « PROGRAMME 2 ».

```
10 REM PROGRAMME 2
20 PRINT « Z CONTIENT
   ENCORE » ; Z
30 END
```

Cela fait, rappelez maintenant le programme n° 1 (LOAD « PROGRAMME 1 », 8), et lancez-en l'exécution. La ligne 30 vous demande d'attribuer une valeur à la variable Z, puis la ligne 40 charge automatiquement le programme 2 et, sans rien vous demander, le fait s'exécuter. Vous constatez alors que le contenu de la variable Z n'a pas été modifié. Pourtant, le programme en mémoire a changé entretemps : vous pouvez le vérifier avec l'ordre LIST.

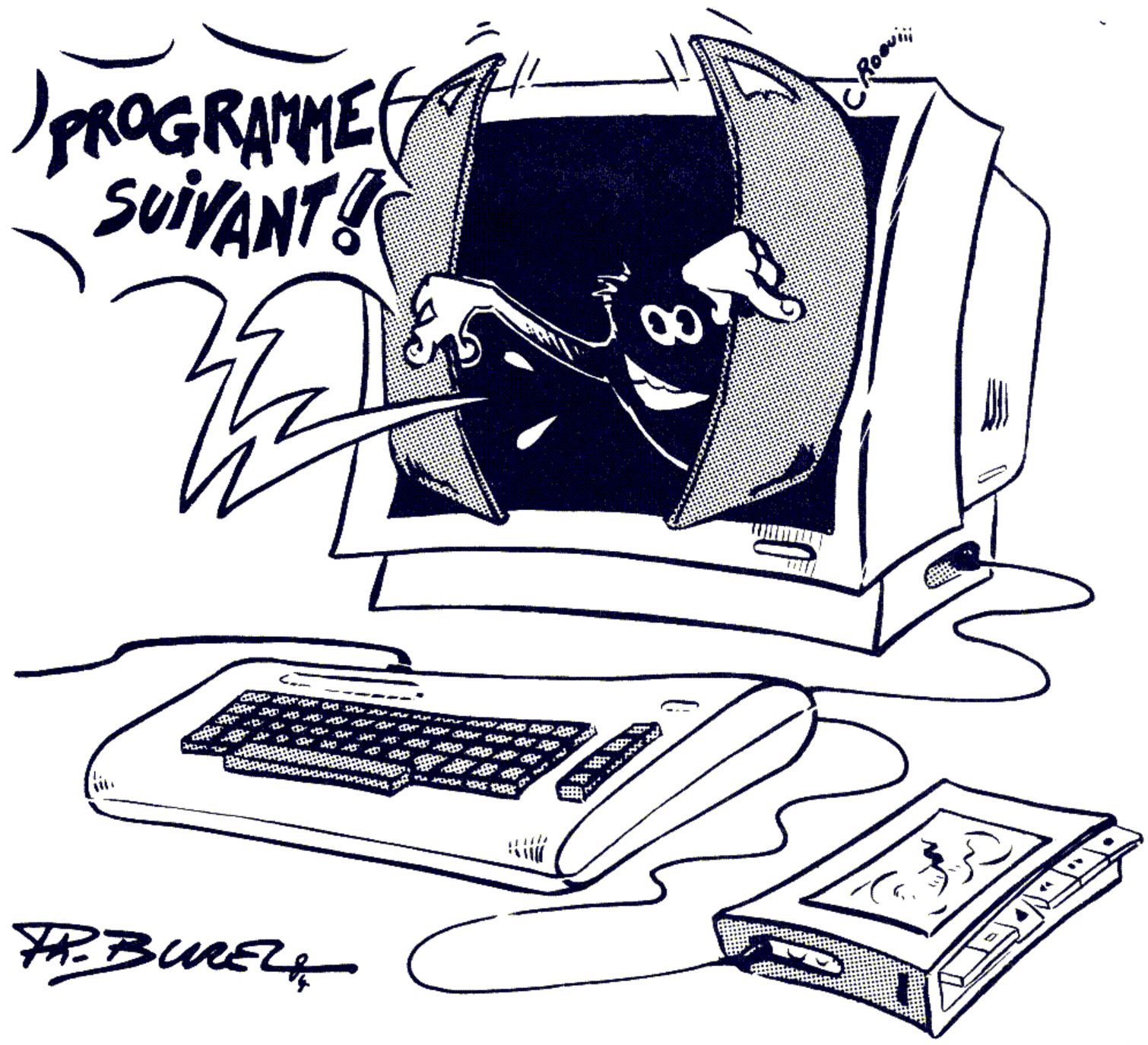
Tout paraît donc fonctionner à merveille... Observons tout de même ce qui se produit quand le programme n° 2

POUR ENCHAÎNER VOS PROGRAMMES SUR COMMODORE

est plus long que le programme appelant. Pour ce faire, ajoutons au programme n° 2 une ou deux très longues lignes de REM, sauvegardons-le à la place du précédent en utilisant la commande SAVE « @ : PROGRAMME 2 », 8

Relançons maintenant le programme n° 1 pour constater (eh oui, malheureusement) que le contenu de la variable Z est définitivement perdu. Ce phénomène s'explique assez facilement : le pointeur indiquant la fin du programme Basic et le début de la zone où sont mémorisées les variables a été mis à jour au moment du premier LOAD, mais il n'a pas été modifié quand le second programme a été chargé.

Le programme appelé est donc chargé en mémoire, mais il peut recouvrir les variables et, par conséquent, les détruire. Pire : le programme appelé ne s'exécute pas en entier pour la même raison. Essayez LIST, pour voir...



Et les chaînes... de caractères ?

Dans tous les cas, un autre problème se pose si des constantes alphanumériques ont été définies dans le programme appelant. Exemple :
25 CH$ = « CHAINE »

Ici, un pointeur indique l'emplacement de la chaîne dans la mémoire de l'ordinateur. Mais ce pointeur n'est pas, lui non plus, modifié lors du chargement suivant, alors qu'à l'emplacement de la chaîne se trouve tout autre chose ! Pour éluder ce problème, il suffit de conserver la chaîne dans la partie de mémoire vive réservée aux variables alphanumériques. On y parvient, tout simplement, en formulant ainsi la ligne de définition :
25 CH$ = «» + « CHAINE »

Pour des raisons analogues, les définitions de fonctions telles que :
25 DEF FNZ(X) = X↑2 + SIN(X)
provoquent souvent des difficultés. Dans ce cas, la solution la plus simple et la plus sûre consiste à redéfinir la fonction dans le programme appelé.

Il arrive très fréquemment que le programme « appelant » soit plus court que le programme « appelé ». Cela se produit, par exemple, quand le premier programme est un menu. Vous pouvez alors procéder de la façon suivante :

- chargez l'un après l'autre tous les programmes par LOAD ;
- pour chaque programme, notez la valeur obtenue en demandant à la machine PRINT PEEK (46), PEEK (45)(1) ;
- les deux valeurs que vous obtiendrez représentent le contenu du pointeur de fin de programme Basic ; l'important est de retenir la valeur maximale obtenue, autrement dit celle du programme le plus long ;
- cette valeur une fois connue, ajoutez au programme n° 1 la ligne suivante(1) : POKE 46,(X):POKE 45, (Y):CLR

Dans cette ligne, X représente la valeur de PEEK(46), et Y la valeur de PEEK(45) ; ce sont les valeurs obtenues avec le plus long des programmes à chaîner. N'oubliez pas le CLR qui établit les valeurs des pointeurs associés.

(1) Sur CBM 3000/4000/8000, on remplacera les adresses 46 et 45 par 43 et 42.

Attention aux modifications !

Une remarque pour finir. Si vous procédez comme je vous l'ai indiqué, et si vous voulez modifier par la suite l'un des programmes qui seront chaînés, retapez-le en entier au clavier et reprenez le processus que nous venons de décrire. C'est indispensable si le programme que vous avez modifié est devenu le plus long de tous.

Une prochaine fois, nous étudierons une autre technique de chaînage, moins élégante il est vrai, mais couramment utilisée sur toute la gamme Commodore.

Jean-Pierre LALEVÉE

VOTRE ORDINATEUR

LE MAGAZINE DE L'INFORMATIQUE A LA MAISON

ISSN 0752-2363

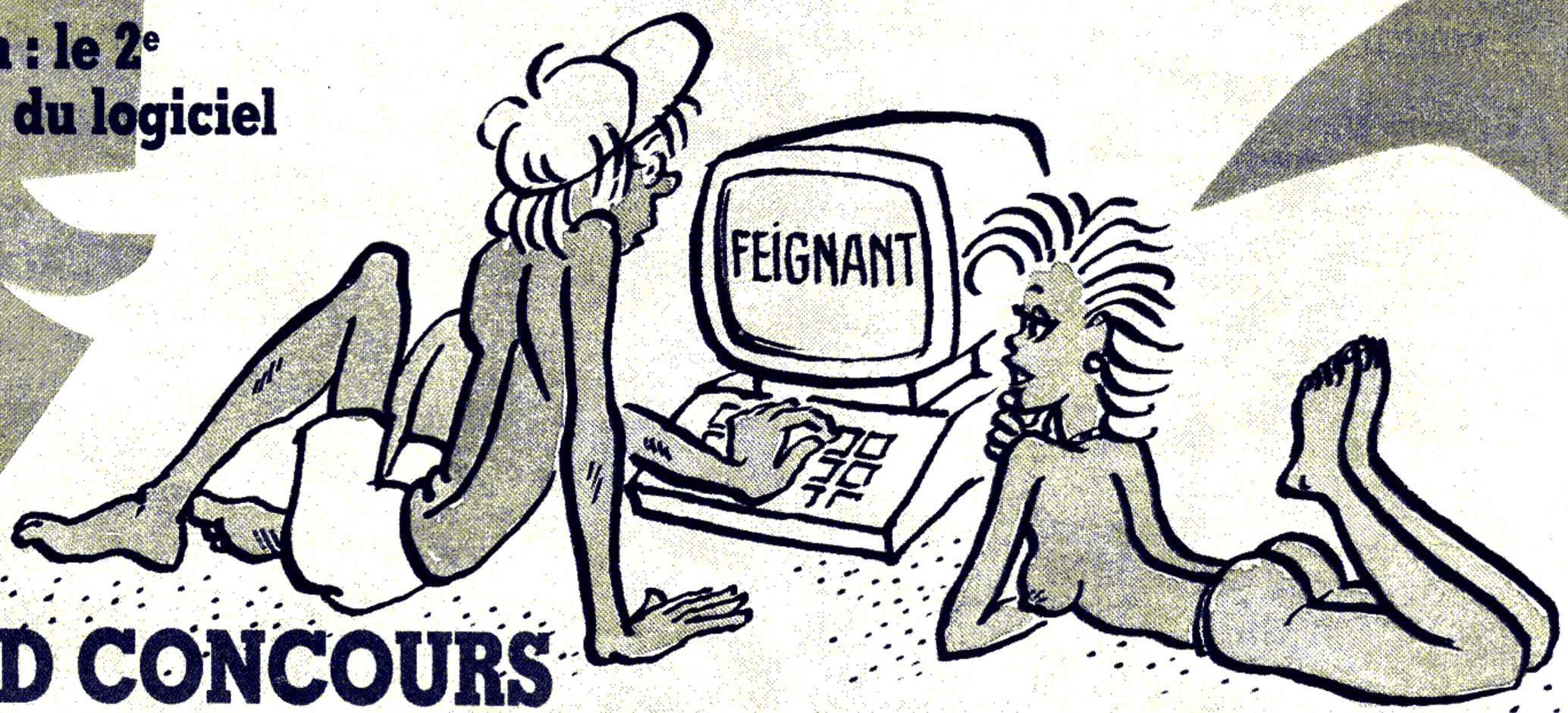
N°6

EN VENTE DANS TOUS LES KIOSQUES

INITIEZ-VOUS EN JOUANT !

supplément de 64 pages
cahier de vacances

- à l'essai : Casio PB700, Electron, MO5 les ordinateurs autonomes
- basic, logo les fiches programmes
- Avignon : le 2e Festival du logiciel



Cabu

GRAND CONCOURS
GAGNEZ DES ORDINATEURS
AVEC *France inter* ET VOTRE ORDINATEUR

M 2867 - 6 SP - 25 F JUILLET-AOÛT 84 - 25 F BELGIQUE 207 FB - CANADA 3,65 $ C - SUISSE 7,5 FS