LIST

nº 4

LE JOURNAL DES AMATEURS DE PROGRAMMATION

NOVEMBRE 1984

GROS PLAN SUR PASCAL

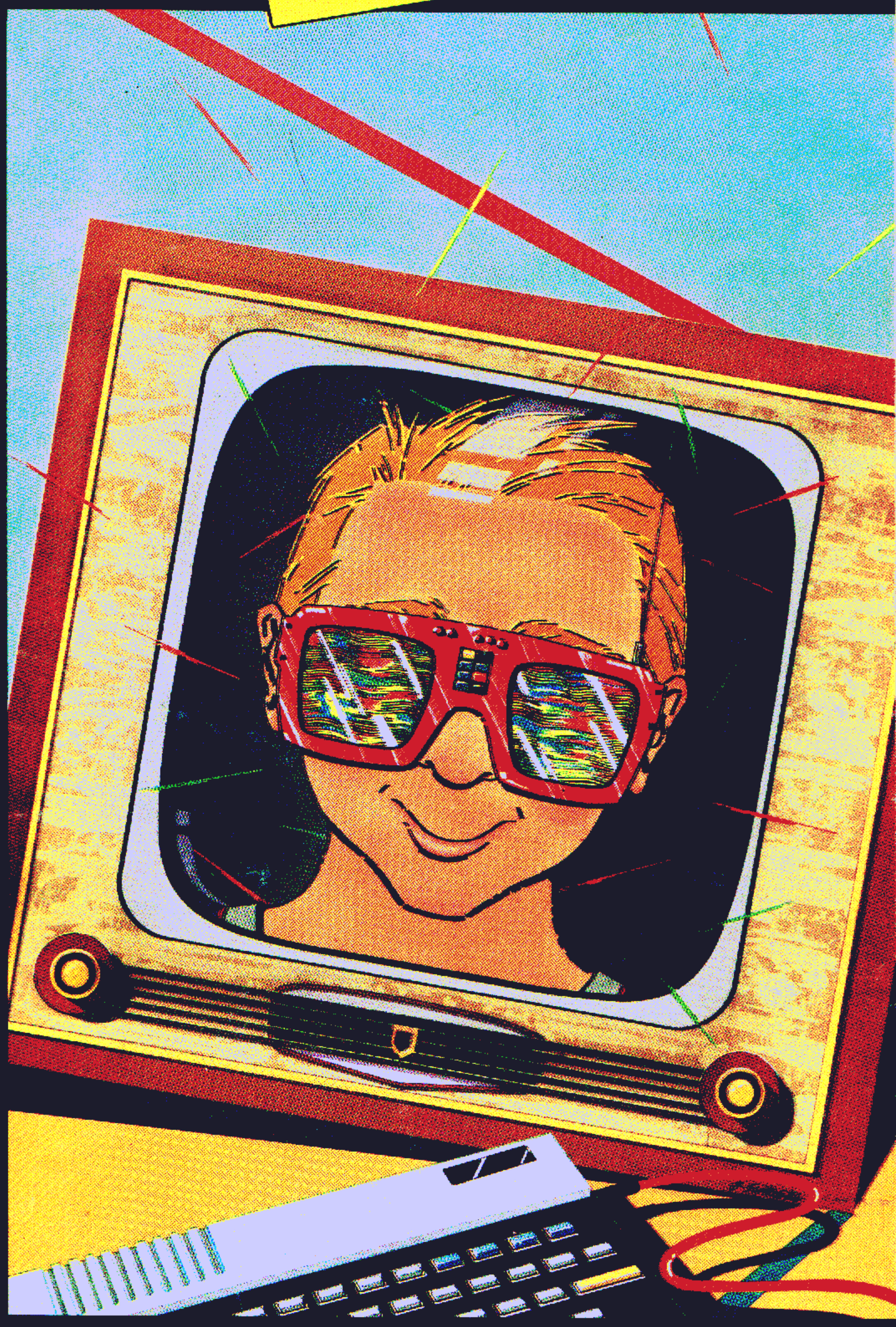
- Son histoire et sa famille
- Une procédure à petits pas
- Turbo Pascal : un compilateur qui décoiffe

SEIZE ORDINATEURS A L'ÉPREUVE

- Leur Basic soumis aux dix tests de LIST

DANS LA MÉMOIRE

- Compilateur et interpréteur, des alliés objectifs
- Basic du TRS-Coco2 et de l'Atari 800XL
- Assembleur sur TO7, pour programmer le 6809
- Récursivité : le Basic y vient



M 2712-4-20 F

L'Ordinateur de poche

Belgique : 154 FB - Canada : 2.95$C - Suisse : 6 FS

20 F

Chez Duriez : 15 micros portatifs + 9 domestiques

Super branchés (pas d'angoisse !)

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

Avez-vous vu les **300 prix**

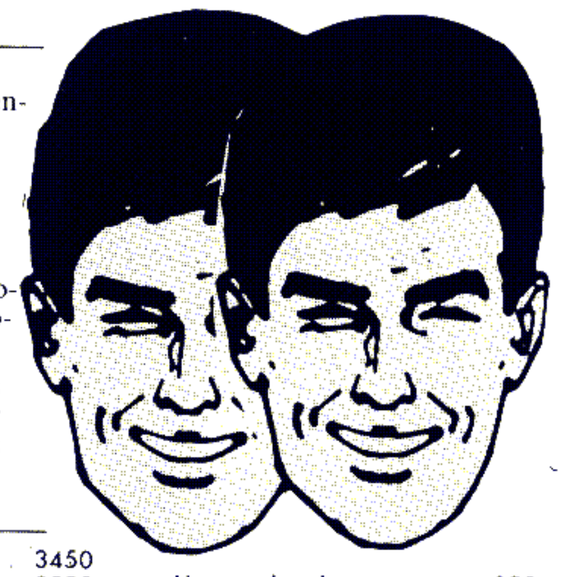
Charter© Duriez ?

valables jusqu'au 20 Novembre

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 2,10 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	1990
800 XL Péritel	2490
Magnéto	426
Lecteur de disquette	2754
Imprimante courrier	3229
Traceur 4 couleurs	854
Manette de jeu	120
Imprimante FP10	610
Fx 750	**1550**
FA 20	**1150**
Carte 4 Ko	**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

AMSTRAD

CPC 464 + moniteur vert	2990
CPC 464 + moniteur couleur	4490

COMMODORE

Commodore 64 Pal	2750
Commodore 64 Péritel	3450

PERIPHERIQUES VIC20 et C64

Lecteur de cassettes	465
Lecteur de disque 1541	3380
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)	588
Zoom Pascal (disq)	456
HES MON 64 (cart)	390
Professionnel	
HES Writer (cart)	329
Omnicalc (cart)	329
Stat 64 (cart)	490
Graph 64 (cart)	380
Multiplan (disq)	1100
Vizawriter (disq)	1355
Super Base 64 (disq)	1190
Educatif	
Turtle graphic (cart)	588
Paint brush (cart)	223
Sinthy 64 (K7)	326
Turtle Toyland (disq)	338
Coco (disq)	440
Jeux	
Choplifter (cart)	495
Lode Runner (cart)	495
Attack of Mutant Camel (cart)	384
Laser Zone (cart)	236
Gridrunner (cart)	170
Rootin tootin (cart)	236
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

PX 8	**10300**
Extension mémoire 60K	**3300**
Extension mémoire 120K	**4660**
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 41 CV	**2190**
HP 41 CX	2880
HP 71	**5100**
Extension mémoire 4K	**784**
Lecteur de cartes magnétiques	**1634**
Interface	**1318**
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

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	5200
M 10 mémoire 24K	**6990**
Traceur 4 coul.	2090
Secteur	98
Cordon imp. parallèle	199
Cordon imp. RS 232	498

ORIC ATMOS

Oric Atmos 48 K	2250
Cordon Péritel + alimentation 12 V	**95**
Traceur 4 coul. + cordon	1510
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	2065
Traceur 4 coul. CE 150	1990
PC 1500 A + CE 150	3990
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	1085
PC 1245	540
PC 1401	1060
PC 1260	1580
PC 1261	2065
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)	190
Othello (cart)	188
Mash (cart)	190
The Attack (cart)	134
Star Trek (cart)	190
Return to Pirate I (cart)	190
Tombstone City (cart)	188
Super Demon Attack (cart)	190
TI Invaders (cart)	188
Hopper (cart)	190
Mind Challenger (cart)	134
Burger Time (cart)	190

THOMSON

MO 5	**2387**
Lecteur de K7	**598**
TO7-70	**3486**
Lecteur K7	**690**
Extension 64K	**1055**
Contrôleur de communic.	850
Manettes jeux et son	580
Lecteur dis. avec cont.	**3596**
Memo Basic	480
Cordon imp.	290
Interface SECAM	530

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

★★★★★★★ Machines à écrire

- Photocopieurs
- Répondeurs téléphoniques
- Calculatrices
- Papeterie
- etc...

Demandez le nouveau **catalogue général Duriez** contre 3 timbres à 2,10 F.

☐ **Duriez,** 112 et 132 bld St-Germain 75006 Paris (M° Odéon, St-Michel)

★★★★★★★

Média Conseil

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 2,10 F.

☐ 1 Catalogue général Duriez (Calculatrices, Machines à écrire, Répondeurs, Photocopieurs, Classeurs, Dicteurs, Papeterie, etc...) contre 3 timbres à 2F10.

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

☐ 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.

Si changement de prix, je serai avisé avant expédition.

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

.....

.....

.....

Date et Signature .....

.....

LI 10 Octobre 84

PSI, 1er DE CORDÉE !

8 guides pratiques PSI

qui vous permettront d'aborder les utilisations spécifiques de votre ordinateur et vous aideront à comprendre des sujets parfois ardus, si vous avez déjà une certaine connaissance de l'informatique.

NOUVEAU

Pascal UCSD sur Apple II
par Jacques Rouault
et Patrice Girard
Tome 1 : *232 pages - 110,00 FF*
Tome 2 : *168 pages - 90,00 FF*
L'ordinateur Apple II, le langage Pascal et le système d'exploitation UCSD forment à eux seuls le plus petit ensemble de micro-informatique professionnelle.
Le tome 1 étudie ces trois éléments, les programmes de mise en route, les types et instructions Pascal UCSD. Le tome 2 aborde les instructions propres à l'Apple II (graphiques, musique, manettes...) ou permettant une extension de l'architesture de la programmation (structure en blocs, chainage...), les procédures de l'unité Applestuff et chainstuff, les manipulations d'octets, les problèmes de l'analyse. Des exercices clôturent le second tome, qui constitue un véritable support de cours pour la formation permanente.

Les bases de données sur Apple II
par Michel Keller
144 pages - 85,00 FF
L'objet de cet ouvrage est d'aider le lecteur à faire un choix parmi les nombreux logiciels existants sur Apple. Quatre de ces logiciels sont sélectionnés ici : PFS et PFS/Report - DB Master - CX BASE 200 - DBASE II. Pour chacun, on trouve une description détaillée du logiciel lui-même et de ses procédures de mise en route, de création de fichier, de saisie des données, de maintenance et d'édition. L'auteur termine cette étude par l'exposé des avantages et des inconvénients inhérents à chaque logiciel.

MS-DOS pas à pas NOUVEAU
par Alain Pinaud
120 pages - 80,00 FF
Le but de cet ouvrage est de faire découvrir au lecteur les commandes du système d'exploitation MS-DOS en les pratiquant. S'appuyant sur une méthode pédagogique pratique, il aborde les différentes commandes par niveaux de compréhension.
Après la définition du système d'exploitation, sont vues les commandes essentielles, les extensions de la version 2 de MS-DOS (introduite sous le nom de PC-DOS 2) et les logiciels d'accompagnement.
Le niveau de connaissances requises est réduit, le lecteur doit cependant être familiarisé avec les principaux termes touchant à l'exploitation de l'ordinateur.

CP/M pas à pas
par Alain Pinaud
128 pages - 80,00 FF
"CP/M pas à pas" s'adresse aux possesseurs d'ordinateur individuel munis de CP/M, désireux d'apprendre à utiliser ce système d'exploitation de disquette.

Lisp sur Apple II
par Nicole Bréaud-Pouliquen
112 pages - 80,00 FF
Description concrète et progressive de la programmation en langage LISP sur l'ordinateur Apple II, ce livre démystifie et met en évidence la puissance à l'expression de ce langage. De nombreux exercices et la présentation d'exemples complexes appliqués à la gestion des listes, l'analyse grammaticale et l'élaboration de dessins récursifs complètent cet exposé.

NOUVEAU

Basic système et langage machine du PC 1500/A
VOYAGE à l'intérieur du Sharp
par Jean-Christophe Krust
168 pages - 90,00 FF
Une première initiation au langage Basic est tout ce qui est nécessaire au lecteur pour aborder cet ouvrage sur l'ordinateur Sharp PC 1500/A ou Tandy PC 2. Tous les mécanismes secrets de la machine soit dévoilés pour permettre au lecteur d'utiliser toutes les possibilités du Sharp. Le débutant trouvera aussi tout ce qu'il faut pour faire ses premiers pas de programmeur en langage machine. L'ouvrage comporte : des informations sur la structure logicielle interne de l'ordinateur (codages, langage machine) et des programmes utilitaires mettant en œuvre ces informations à la fois comme méthodes de programmation et comme matière à travailler.

NOUVEAU

Le Basic des MO5 et TO7/70
par Gilles Blanchard
160 pages - 90,00 FF
Les deux premiers chapitres de cet ouvrage sur les ordinateurs MO5 et TO7/70 sont consacrés à la présentation et à la manipulation des claviers de chacun de ces matériels et de leurs périphériques. On passe ensuite à l'étude des variables numériques et alphanumériques. Le chapitre 4 traite des graphiques et de la couleur, pour lesquels le TO7/70 et le MO5 ont des capacités remarquables. Le lecteur apprend ensuite à écrire un programme, à le modifier puis à communiquer avec des organes extérieurs (imprimante, lecteur-enregistreur de programmes). Enfin, après une description des principales différences entre TO7/70 et MO5, le chapitre 8 donne le répertoire des fonctions et instructions du langage Basic sur ces ordinateurs. Des annexes utiles viennent compléter l'ouvrage (codes d'erreur, codes ASCII, fonctions des caractères spéciaux).

EGALEMENT CHEZ VOTRE LIBRAIRE ET EN BOUTIQUE SPECIALISEE

LGP 11

P.S.I. DIFFUSION
BP 86
77402 Lagny-S/Marne Cedex
FRANCE
Telephone (6) 006.44.35
P.S.I. BENELUX
5, avenue de la Ferme Rose
1180 Bruxelles
BELGIQUE
Telephone (2) 345.08.50
P.S.I. SUISSE
Case postale
Route neuve 1
1701 Fribourg
SUISSE
Tel. : (037) 23.18.28
CCP 17.56.84

Au Maroc
SMER DIFFUSION
3, rue Ghazza
Rabat
MAROC
Tél. : (7) 237.25

Au Canada
SCE Inc.
65, avenue Hillside
Montréal (Westmount)
Québec H3Z1W1 - CANADA
Tél. : (514) 935.13.14

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

DESIGNATION	NOMBRE	PRIX
	TOTAL	

par avion ajouter 8 FF (75 FB) par livre
Pour la SUISSE frais de port sur tout envoi : 1.50 FS

Paiement par chèque joint

Paiement en FF par carte bleue VISA (à P.S.I. DIFFUSION uniquement) paiement supérieur à 50 FF

Signature (obligatoire pour paiement par carte de crédit)

☐ Je désire recevoir le catalogue P.S.I. gratuit ;

N° _____ Date d'expiration _____

NOM _____ PRENOM _____

rue _____ n° _____

Code postal _____ Ville _____

Table de conversions en Francs belges et Francs suisses		
35 FF =	250 FB	12,20 FS
60 FF =	432 FB	19,10 FS
70 FF =	504 FB	22,20 FS
80 FF =	576 FB	25,30 FS
85 FF =	612 FB	26,90 FS
90 FF =	648 FB	28,40 FS
95 FF =	684 FB	29,90 FS
100 FF =	720 FB	31,50 FS
105 FF =	756 FB	33,00 FS
110 FF =	792 FB	34,60 FS
115 FF =	828 FB	36,10 FS
120 FF =	864 FB	37,60 FS
125 FF =	900 FB	39,10 FS
130 FF =	936 FB	40,60 FS
135 FF =	972 FB	42,20 FS
140 FF =	1.008 FB	43,70 FS
145 FF =	1.044 FB	45,20 FS
150 FF =	1.080 FB	46,70 FS
160 FF =	1.136 FB	49,70 FS
170 FF =	1.207 FB	52,60 FS
180 FF =	1.278 FB	55,60 FS
190 FF =	1.349 FB	58,60 FS
195 FF =	1.385 FB	60,00 FS
DISQUETTES		
195 FF =	1.500 FB	61,50 FS
295 FF =	2.275 FB	90,70 FS

AGRAPH

LIST
LE JOURNAL DES AMATEURS DE PROGRAMMATION

N° 4 - NOVEMBRE 1984

SOMMAIRE

RÉDACTION-RÉALISATION
Directeur de la rédaction : Bernard Savonet
Rédacteur en chef : Jean Baptiste Comiti
Responsable de rubrique : 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 : *Michel Arditti, Olivier Arbey, François J. Bayard, Robin Bois, Eric Boucher, Gilles Bransbourg, Viviane Bazin, Christian Boyer, Jean-Paul Carré, Thierry Chamoret, Raymonde Coudert, Geneviève Cuvelier, Jacques Deconchat, Robert Daguesse, Philippe François, Florence Gautier-Louette, Pierre Ladislas Gedo, Max Hagenburger, Trung Hua-Ngoc, Jean-Christophe Krust, Renée Koch, Jacques Labidurie, Alain Lavenir, Laurent Leclaire, Thierry Lévy-Abégnoli, Pierre Lison, Jean-Pierre Lalevée, Alain Mariatte, Alain Morel, Danièle Millet, Pierrick Moigneau, Claude Nowakowski, Yvon Pérès, Edouard Rencker, André Warusfel.*

Illustrations : *Boredom, Philippe Burel, Cannella, Catherine Dubreuil, Frapar, Paul Gendrot, Bernard Helme, Fabien Lacaf, Alain Mangin, Alain Prigent, José Santos, Nicolas Spinga, Eric Théocharidès.*

ÉDITION-PUBLICITÉ-PROMOTION
Éditeur : Jean-Pierre Nizard
Éditeur-adjoint : Jean-Daniel Belfond
Administration : Maryse Marti, assistée d'Anne Stolkowski
Publicité : Béatrice Ginoux Defermon, assistée de Nadine Schops

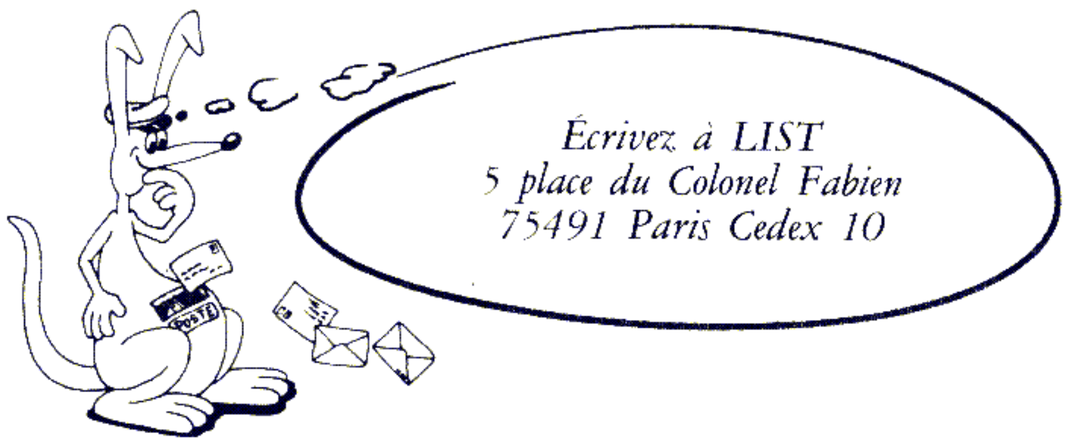
VENTES
Diffusion NMPP : Sophie Marnez
Abonnements : Muriel Watremez
assistée de Sylvie Trumel, Cécilia Mollicone et Dominique Loridan

Directeur de la publication : Jean-Luc Verhoye

LIST **est une publication du** tests

5 place du Colonel Fabien - 75491 Paris Cédex 10
Téléphone : (1) 240 22 01 - Télex : LORDI 215 105 F

A VOS CLAVIERS



Bienvenus

EN compensation d'un abonnement souscrit à la revue Trace, je compte désormais parmi vos lecteurs. Et j'en suis heureux.

Sachant qu'une revue est d'autant plus vivante que ses lecteurs y participent, je voudrais vous demander si lors d'un envoi de "trucs" ou programmes, une simple liste est suffisante, ou s'il vous faut une disquette les contenant ?

Jean-François Brandone
06 Carros

■ *Il est vrai que l'introduction au clavier d'un programme est parfois fastidieuse : merci de penser à nous. Mais dans la plupart des cas, lors de vos envois de "trucs" ou d'astuces, une simple liste, imprimée avec des caractères noirs de préférence, nous suffit.*

Interface trop chère

J'AIMERAIS savoir s'il est possible de relier le PB-700 à un magnétophone, sans acheter l'interface FA-4, trop coûteuse.

D'autre part, possédant la FA-3, interface pour le PB-100, je n'ai pas encore réussi à trouver un magnéto qui se connecte : quel matériel utiliser ?

Merci d'avance.

Christophe Chartier
44 Nantes

■ *L'interface FA-4 permet de relier le PB-700 à un magnétophone à cassette (ou à microcassette). Elle coûte 765 FF ttc. En lui ajoutant un cordon (500 FF), le PB-700 peut être relié à une imprimante de type Centronics.*

Une autre interface (en fait, un bloc interface-cassette/imprimante), la FA-10, existe pour le PB-700 : elle coûte environ 2 300 FF. Aucune autre possibilité de liaison entre le PB-700 et un magnétophone n'est possible actuellement.

Quant à l'interface FA-3, elle permet la liaison entre le PB-100 et tout magnétophone à cassette disposant de prises "jack" en sorties micro et écouteurs. De tels magnétophones se trouvent facilement, dans le commerce.

Programmer le Dragon

DÉBUTANT en informatique, je me suis penché, avec mon Dragon 32, sur le programme de Tic-tac-toe paru dans le numéro 2 de *LIST*.

J'ai observé qu'il fallait remplacer tous les LET du programme par des THEN (d'autres THEN ont été omis à la suite de IF). En outre, ne comprenant pas l'instruction DIM PO% (9841), j'ai simplement supprimé le signe "%" et ça n'a pas marché : DIM PO(9841) prend beaucoup trop de place en mémoire.

Enfin, le langage d'assemblage ou le langage-machine est-il abordable, et si oui, comment ?

Jean-Marie Monteil
31 Rieux

■ *Dans un programme Basic, la présence de IF n'entraîne pas forcément celle de THEN : celui-ci peut être "élidé". Ce n'est pas un oubli, mais une simplification.*

Quant au signe "%", il indique que la variable traitée est entière, et n'occupe donc que deux octets (au lieu de quatre, en simple précision). Ainsi, la mémoire n'est pas dépassée. Cette possibilité est accessible d'une autre façon, sur le Dragon : il faut déclarer la nature entière de la variable, en début de programme par DEF INT nom de variable.

La programmation en Assembleur — du 6809, sur le Dragon — est tout à fait "abordable". Par exemple, en lisant un ouvrage sur le 6809 (comme "Microprocesseurs : du 6800 au 6809, modes d'interfaçage" de Gérard Revellin, chez Dunod Informatique, 1981) pour apprendre, et en essayant un logiciel comme Dragbug (voir "les coups d'œil de LIST", page 41) pour programmer en Assembleur.

NSC 800, l'autre nom du Z80

CHAQUE mois, j'éprouve un immense plaisir à taper programmes et astuces sur ma "bécane". L'informatique est faite pour moi ! J'adore traquer les "Syntax Error", les "Illegal Fonction" ou autres messages d'erreur. Et une fois que le programme tourne, quelle récompense ! C'est ce qui fait que je souhaiterais aujourd'hui programmer en langage-machine sur le Canon X-07, dont le processeur est le NSC 800. Se programme-t-il comme le Z80 ?

François Fine
05 Briançon

JE possède un Canon X-07. Son microprocesseur, le NSC 800, est compatible avec le

INDEX DES ANNONCEURS

A VOS CLAVIERS

Z80. Je n'ai jamais fait que recopier des programmes utilisant le langage-machine, sans jamais rien y comprendre. Quelle bibliographie conseillez-vous pour apprendre le langage-machine du Canon ? Merci d'éclairer ma lanterne.

Frédéric Dalsace
50 Agon

■ *Le processeur du Canon X-07, le NSC 800, est dit "compatible" avec le Z80. En fait, il semblerait que ce soit le même processeur, sous deux appellations différentes. La programmation en langage-machine peut donc être abordée grâce à des ouvrages sur le Z80 : "Initiation au langage Assembleur" (avec des exemples sur 8080, 8085, Z80 et NSC 800) de Bernard Geoffrion et Henri Lilen, aux éditions Radio ; ou encore, "Programmer en Assembleur, illustré avec le jeu d'instruction du Z80", de Alain Pinaud, aux éditions du PSI.*

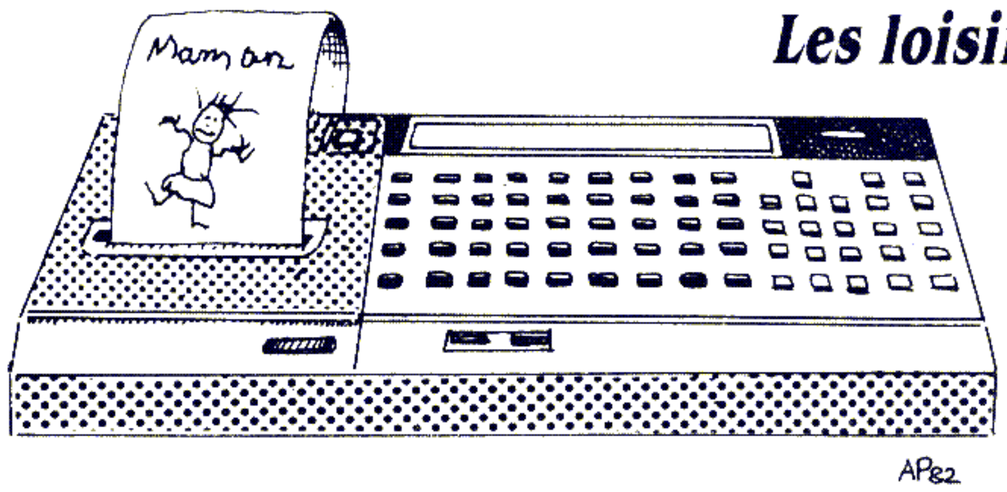
Les manuels en question

VOTRE nouvelle formule est un précieux complément aux notices fournies par les fabricants qui sont notoirement insuffisantes. Quand les distributeurs se rendront-ils compte que l'on voudrait exploiter toutes les possibilités de leurs appareils et pas seulement savoir les brancher ?

Longue vie à *LIST*.

Joël Palud
69 Lyon

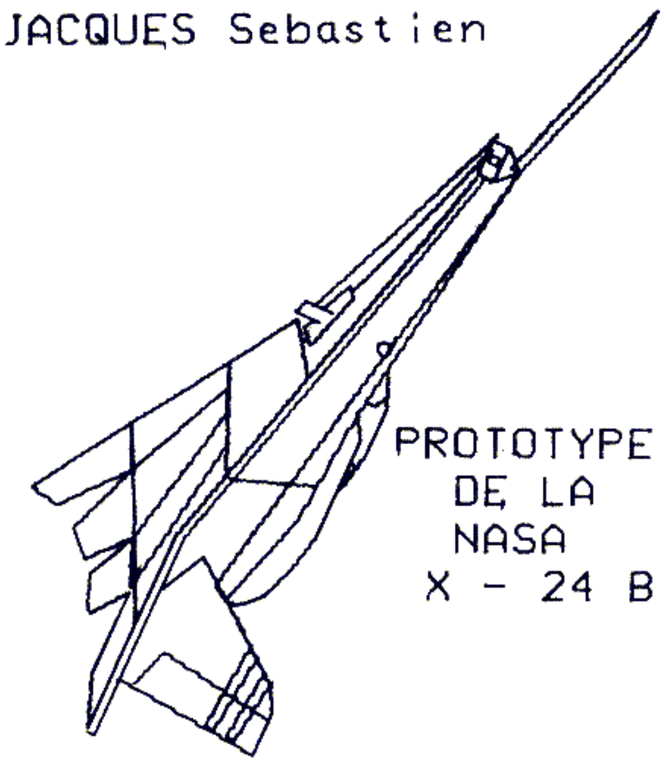
Les loisirs du PC-1500



SHARP PC-1500
TAKVORIAN

PORTRAIT

JACQUES Sebastien



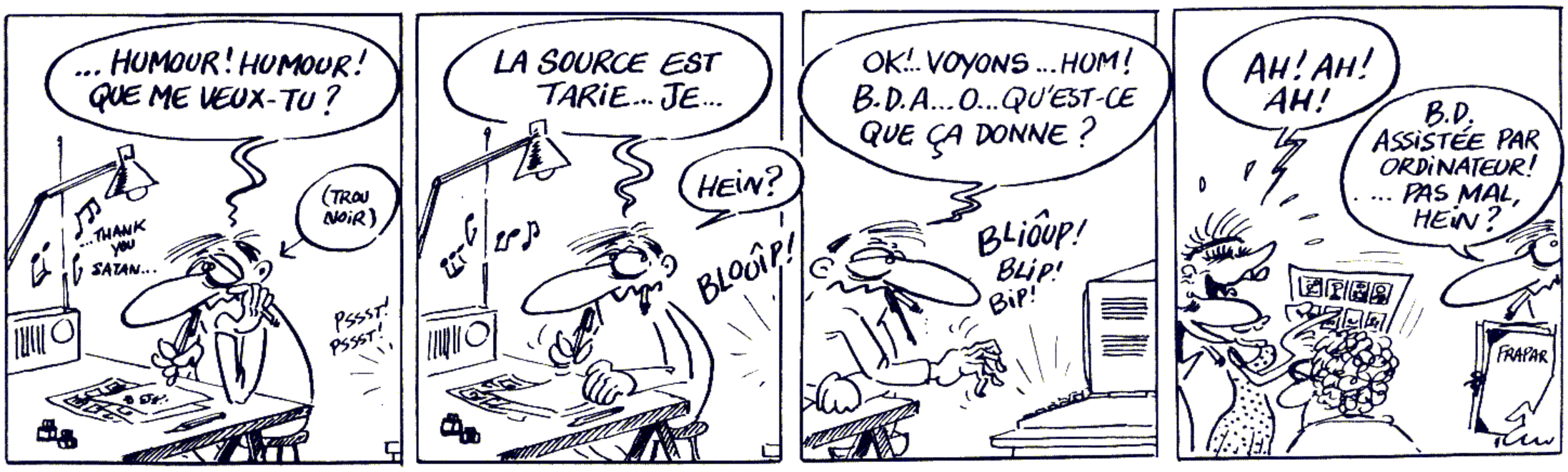
PROTOTYPE
DE LA
NASA
X - 24 B

LUCIEN

ECRIT ET DESSINE SUR
COMPUTER SHARP PC-1500 SENEE

© BY LUCE. & NIC. HERBER. 1983

Hervé Le Follic



LA GAZETTE DE LIST

UN LIVRE

L'assembleur du Commodore 64
Pratique Volume 2
Daniel-Jean David
Editions du PSI
Broché, 208 pages
Prix : 100 FF

NOUS voilà en terrain connu : « l'assembleur du Commodore 64 » n'est autre que la version revue et corrigée de l'édition originale parue en 1981, titrée « La pratique du PET/CBM-Volume 2 ».

Entre-temps, la rédaction et les schémas n'ont pratiquement pas été remaniés. Ce qui n'empêche nullement les nouveaux lecteurs d'y trouver matière à découverte. Les qualités pédagogiques sont intactes qui permettent aux néophytes comme aux programmeurs moyens un perfectionnement réel et l'acquisition des bases de la programmation en langage-machine sur C. 64.

Cinq chapitres bâtissent l'ouvrage. Le premier dessine une vue d'ensemble du jeu d'instructions et le mode de représentation des données en machine. Au fil des chapitres suivants, nous approfondissons l'étude des instructions tout en abordant le vaste dossier de la manipulation de l'assembleur-éditeur type du C. 64. Le chapitre de conclusion s'impose : quid des interactions avec Basic...

Un bon point pour les schémas : ils aident puissamment à la compréhension du texte.

Restent les annexes. Tableaux des instructions du processeur, liste des adresses stratégiques du système constituent un ensemble cohérent de bonne facture, émaillé d'exercices de difficulté croissante qui mèneront le lecteur à une excellente connaissance des manipulations de base. La solution (ou une solution) de chaque exercice se trouve dans les dernières pages du recueil.

Il s'agit là d'un outil d'apprentissage solide du langage-machine et de la programmation en Assembleur. JPL ■

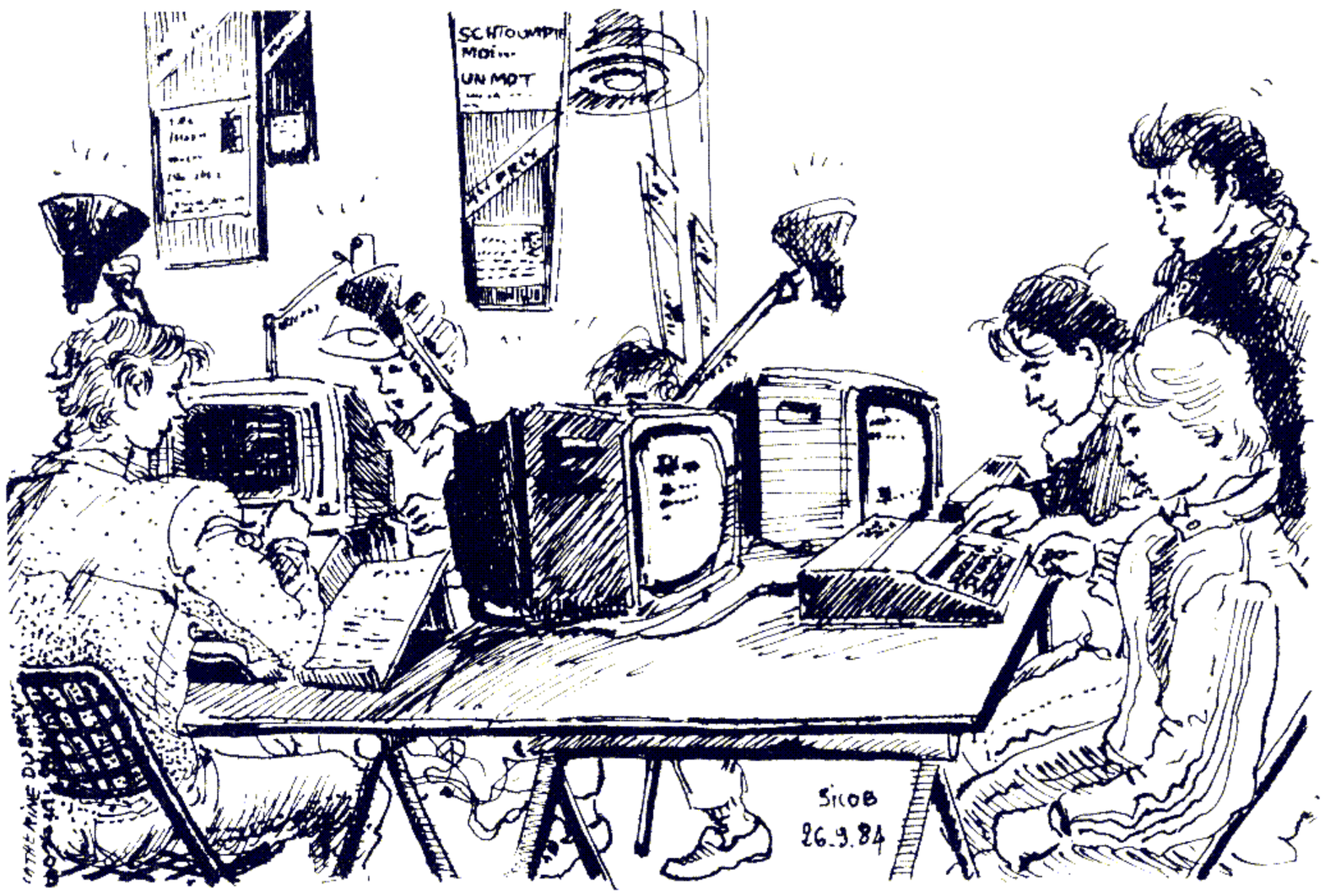
Forth, Logo et les autres

INFOGRAMES, la société lyonnaise récemment entrée sous la houlette de Vifi-Nathan annonce quatre nouveaux logiciels (deux utilitaires et deux langages), pour Oric Atmos et Commodore 64. Côté cœur,

Les logiciels du Festival

POUR ceux qui n'avaient pu se rendre à Villeneuve-les-Avignon en juillet dernier, les animateurs du Festival du Logiciel présentaient à l'occasion du Sicob les programmes primés.

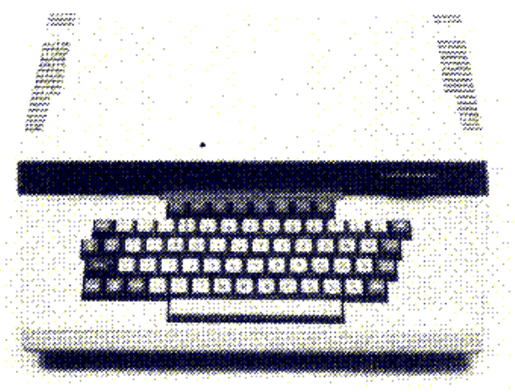
Le grand public a ainsi pu découvrir, du 19 au 30 septembre 1984, sur le parvis de La Défense, 40 logiciels choisis parmi 300 autres par les 8 000 visiteurs de ce festival ''estival''. ■



deux aides à la création graphique.

Images permet de créer, compiler et sauvegarder toute image sur Atmos, (en vue d'être utilisée au sein d'un programme Basic ou Assembleur) et *Gas Kit* combine sur Commodore 64, graphisme et musique. Côté initiation et apprentissage, *Logor* offrira aux propriétaires d'Atmos les joies infinies du langage Logo et *Forth* (aux mêmes utilisateurs) une adaptation du fameux Forth, (c'est sans surprise). ■

Einstein, la grosse tête



LES Anglais sont rusés. Profitant de la mythologie moderne et du nom sacré de l'un des plus grands scientifiques du siècle, ils lancent sur le marché un nouvel ordinateur baptisé Einstein.

De quoi faire rêver de nombreux consommateurs. Ratissant large (Einstein est destiné aux amateurs, hobbistes, joueurs et semi-professionnels), ce nouveau britannique s'avère effectivement prometteur : haute résolution graphique 256 × 192 points, 16 couleurs, 32 « sprites », 64 Ko de mémoire vive et 8 Ko de mémoire morte extensible à 32 Ko, il devrait être particulièrement apprécié des programmeurs.

Le Basic est extensible à 190 instructions et différents langages dont Cobol, Forth, Fortran, Logo et Pascal seront prochainement disponibles. Seul inconvénient, le prix 7 000 FF ttc (avec un lecteur de disquettes) qui met l'Einstein directement dans le haut de gamme familial. Un nouveau concurrent d'Apple ? ■

UN LIVRE

Comment programmer
Trois exemples pour une méthode
Jean-Claude Barbance
Éditions du PSI
Lagny, 1984
Broché, 216 pages
Prix : 100 FF

REEDITION d'un ouvrage paru en 1980, ce livre contient une description détaillée, étape par étape, de la méthode à suivre pour écrire des programmes faciles à maintenir. La première partie (une soixantaine de pages) décrit la démarche : comment définir les objectifs, les méthodes d'analyse, la programmation et la mise au point.

C'est bien écrit et facile à lire. L'auteur a choisi de s'adresser à des débutants, pour lesquels cette partie sera très précieuse. Cette description théorique est ensuite reprise de façon plus concrète avec trois exemples : écriture d'un nombre sous forme de mots, jeu du 421 et comptabilité personnelle.

Les trois exemples retenus illustrent assez bien les différents types de problèmes rencontrés, et ils sont écrits dans un Basic standard (TRS-80), mais l'utilisation du IF...THEN...ELSE, des variables entières N% et de plusieurs instructions par ligne posera parfois quelques problèmes d'adaptation au débutant. Heureusement, des organigrammes

très clairs devraient l'aider à s'y retrouver. C'est le programme de comptabilité familiale qui m'a semblé le plus intéressant (une annexe en donne d'ailleurs une version disquette). Un ouvrage qui pourra donc être lu avec profit par des débutants en programmation.

JD ■

A propos de l'interface parallèle pour C.64 et VIC 20

DANS le numéro 3 de *LIST*, nous signalions la découverte, à Bischheim près de Strasbourg, d'une interface parallèle pour Commodore 64 et Vic 20. Nous précisions même l'adresse de son distributeur. Mais pas son nom. Il s'appelle NEOL (4a, rue National, 67800 Bischheim).

L'oubli est réparé ! ■

DU CÔTÉ DES CLUBS

Deux clubs pour les programmeurs Sarthois

AU Mans, le Micro-Club Loisirs et Culture organise des stages de formation avec prêt d'appareil.

Ses membres seraient heureux d'accueillir des amateurs de programmation en Assembleur 6502.

Ils disposent d'Oric bien sûr, mais aussi de TRS-80 et de ZX 81.

Pour plus de renseignements, contactez :

M. Blondeau
Micro-Club Loisirs et Culture
6 allée du Spoutnik
72100 Le Mans

LE 6502 et le Z 80 sont également à l'honneur à Coulaines.

L'Association des Micro-Informaticiens Sarthois vous propose des ateliers d'initiation et de perfectionnement aux langages Logo, Forth, Basic et Assembleur (sur Apple, Oric et Sharp).

Une cotisation annuelle de 200 francs vous donne accès à ses activités ainsi qu'à la bibliothèque et à la programmathèque.

Chaque semaine, l'AMIS prend l'antenne. Son magazine hebdomadaire est diffusé par la Radio locale Le Mans FM104.

Si vous êtes intéressé, sachez qu'une permanence est tenue de 18 à 20 heures tous les mardis, à l'adresse suivante :

A M I S
1 rue de Moscou
72190 Coulaines
Tel. (43)81 83 63
(43)27 33 28

CARNET ROSE

Deux naissances à Paris

Destiné aux possesseurs de Canon X-07, le Club C7 est une association régie par la loi de 1901.

Elle propose à ses adhérents conseils techniques et stages, et met à leur disposition bibliothèque et programmathèque. Le Club édite aussi une gazette.

Une permanence téléphonique est assurée le mercredi matin au 371 22 20

Club C7
33 avenue Philippe Auguste
75011 Paris

LE CIA (lisez Club d'Informatique Avancée, ne pas confondre avec la CIA) ouvre ses portes aux jeunes de 14 à 18 ans (les plus âgés peuvent venir aussi !).

Possesseurs de TO7, ZX, C64, AppleII, TRS-80 ou autres, vous êtes invités à venir acquérir tous les samedis de 15 à 17 heures, les outils de base nécessaires à l'élaboration de logiciels de qualité (jeux, mais aussi tableurs, langages...).

Au programme, désassembleurs, compilateurs, interpréteurs.

Les jeunes membres du club bénéficieront des conseils avisés d'adultes expérimentés.

Si vous n'habitez pas Paris, vous pouvez aussi leur écrire.

MM. Jean Leflour et Christian Scherer
Club d'Informatique Avancée
Ecole Nationale Supérieure de Création Industrielle
48 rue Saint Sabin
75011 Paris

UN LIVRE

Boîte à outils pour Sinclair et Timex ZX 81, ZX Spectrum, Timex 1000, 1500 et 2000
Marcel Henrot
Editions SAPECA
Collection MegaO-Poche
Bruxelles, 1983
Broché, 128 pages
Prix : 35 FF

LA collection MegaO-Poche a été créée pour mettre à la disposition des utilisateurs de petite informatique des programmes qui leur permettent de résoudre de nombreux problèmes de la vie quotidienne. C'est en tout cas ainsi que ses éditeurs la présentent. J'avoue être un peu déçu par cet ouvrage consacré au ZX 81 et au Spectrum.

Peut-être suis-je l'une des rares personnes à ne pas rencontrer quotidiennement de problèmes avec la roulette russe, le tracé d'une ellipse, le codage de messages secrets, et j'en passe ! Toute plaisanterie mise à part, et bien qu'il me semble regrettable que l'aspect « vie quotidienne » ne soit pas effectivement davantage abordé, il s'agit d'un livre bien fait, assez éclectique et très agréable à utiliser.

Il propose 28 programmes, écrit pour le ZX 81 et adaptés au ZX Spectrum. Si l'originalité n'est pas le trait marquant de ces programmes, l'aspect éducatif du livre mérite en revanche d'être signalé : les programmes sont très clairement expliqués, les lignes intéressantes sont abondamment commentées et l'on trouve même la liste détaillée des variables utilisées.

Tout cela sous un format réellement de poche et pour un prix tout à fait abordable. Un livre clair et attrayant, à recommander au débutant.

BE ■

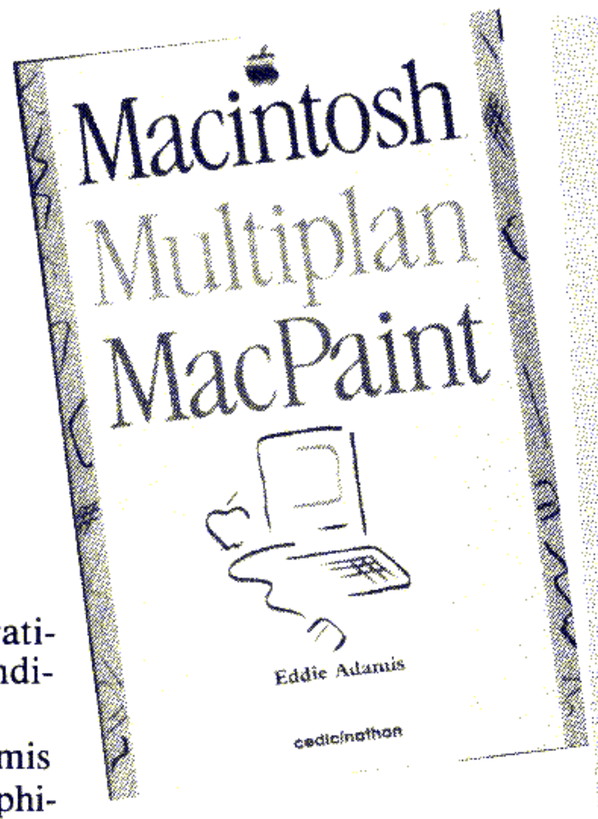
Macadam

NI cow-boy, ni revêtement routier, Macadam est un nouveau langage de programmation développé par une société française — **Régiciel** — à la demande de **CBS Electronics.** Macro-Assembleur destiné à l'ordinateur Adam, Macadam est, selon ses concepteurs, le fin du fin en matière de programmation : lecture et exécution ultra-rapides, haute résolution graphique, éditeur plein écran, gestion des entrées-sorties, multiplicité des modes d'assemblage, un outil efficace pour des jeux, traitements de texte et autres applications.

Pour les sceptiques, une démonstration des capacités de Macadam sera bientôt visible en boutique sous la forme d'un « Micro-Clip », logiciel de présentation de l'Adam. Une précision sur ce nouveau concept : « le micro-clip est à l'ordinateur ce que le vidéo-clip est à la vidéo ». En clair, de la pub joliment habillée. Macadam devrait être déjà disponible. Avis aux « microastes » (qui sont à la micro ce que les vidéastes sont à la vidéo). ■

UN LIVRE

Macintosh
Multiplan
Macpaint
Eddie Adamis
Editions Cédic-Nathan
Paris, 1984
Broché, 144 pages
Prix : 89 FF



IL y a mille façons de pratiquer l'informatique individuelle.

L'ouvrage d'Eddie Adamis s'inspire, quant à lui, d'une philosophie du type "l'ordinateur individuel, mais c'est facile" très particulière : celle que tente de promouvoir — avec semble-t-il un succès assez sympathique — Apple avec son "Système" Mac.

Mac, c'est bien entendu Macintosh que l'on déteste ou dont on tombe amoureux sans guère de nuances. Tout y est pensé pour que son utilisation soit simple (la bidouille n'a plus droit de cité), la souris fait tout pour vous.

Eddie Adamis, le plus francophone (et phile) des auteurs américains, bien connu pour ses "mots-clés du Basic" et ses ouvrages sur Apple III et IBM-PC, réunit ici une — très — brève introduction au Macintosh et deux longs chapitres consacrés à Multiplan et Macpaint.

Il n'existe pratiquement pas une page qui ne bénéficie, en guise de contrepoint, d'une copie d'écran : une idée astucieuse et un moyen agréable et précis de passer en revue la plupart des instructions de ces deux logiciels (un tableur célèbre et un programme de dessin à l'écran, en noir et blanc) ainsi que leur maniement concret.

Voilà un mode d'emploi très visuel qui donne envie de s'y mettre.

AW ■

Turbo Pascal : un logiciel sur CP/M ou MS/DOS

AVEC ses 100 000 exemplaires vendus en quelques mois, Turbo Pascal est le best-seller des compilateurs Pascal. Il faut dire que ses performances sont exceptionnelles.

Il s'agit d'un ensemble complet, possédant un éditeur, un chargeur, un éditeur de liens, un compilateur, un débogueur et même un tableur.

L'éditeur est compatible Word Star, et se révèle rapide, performant et agréable à utiliser. Le compilateur Pascal travaille à une vitesse foudroyante : de l'ordre de vingt lignes à la seconde dans des cas qui ne sont pas particulièrement favorables. Les Pascal M, JRT et MT+ sont largement battus, puisque ceux-ci sont environ dix fois plus lents.

Le débogueur, qui localise les erreurs dans le programme lui-même, est un des meilleurs qui soit. Il donne l'impression que l'on travaille avec un interpréteur Pascal.

Enfin, pour compléter le tout, un tableur, appelé MicroCalc, est fourni en langage source. Non compilé, il vous montrera comment un tel programme est conçu. Compilé, il vous rendra de bons services avec des performances très honorables.

Turbo Pascal est vraiment bien fait pour un prix particulièrement raisonnable : moins de 700 F. A ce prix-là, on risque de croire que ce n'est pas le meilleur compilateur Pascal.

TC ■

Math et matique vont encore en bateau

QUAND le vieux monstre refait surface dans les eaux tumultueuses de l'informatique pochéenne, cela fait « gloup ! ». Et Hewlett Packard est un artiste en matière de « gloup ! ».

De quoi s'agit-il ? Mais d'un nouveau module d'extension, monsieur. Cette fois-ci, c'est pour l'ordinateur HP-71 B — le fort en maths de la famille — et, avec ce petit air de « déjà vu », c'est encore une réussite.

A trente-cinq francs le Kilo-octet, en voici trente-deux intégralement consacrés aux mathématiques. Calcul matriciel, nombres imaginaires et recherches de racines n'ont désormais plus de secret : à chacun sa petite fonction. Sur la cinquantaine de ce module, on peut raisonnablement trouver chaussure à son pied.

FNROOT, justement, est une fonction capable de chercher les racines solutions d'une équation ayant jusqu'à cinq variables. Et, INTEGRAL calculera les intégrales multiples jusqu'à l'ordre cinq également. Quand on disait qu'un air de « déjà vu »...

PROOT (excusez du peu...) n'est pas une insulte mais une fonction destinée à la résolution d'équations polynômiales : trente-cinq minutes suffisent à régler le sort d'un polynôme de degré 100, « relativement rapidement » donc.

Les nombres complexes sont ici à la fête. Le module apporte une panoplie complète de fonctions arithmétiques et scientifiques. Le nombre complexe est traité comme un couple, (0,1) pour i, et les opérations s'effectuent entre couples : (0, 1) × (0,1) donne évidemment (-1,0)

Les matrices ne sont pas

en reste. Le module, ici, fait un peu oublier la pénurie d'instructions du HP-71 B en la matière. Là encore, on retrouve les fonctions arithmétiques de base accompagnées des inévitables calculs de déterminants et inverses, sans oublier la bonne idée de permettre la manipulation des matrices comme des nombres à part entière.

En trigonométrie, ô joie, on trouve les fonctions hyperboliques et la transformation de Fourier.

Bien d'autres fonctions sont maintenant disponibles — dont la très jolie fonction Gamma — qui vous aideront à passer encore des nuits blanches à manipuler vos chiffres chéris. N'oubliez pas non plus la leçon d'anglais donnée à titre gratuit : si 180 pages de manuel dans la langue de Shakespeare ne ravivent pas vos souvenirs scolaires, c'est à désespérer.

Enfin, à l'aide de ce puissant instrument de calcul scientifique qui multiplie très sérieusement la puissance (en ce domaine) du HP-71 B, on n'aura aucune peine à trouver une justification scientifique à la dépense des 1 140 FF nécessaires pour l'acquérir...

OA ■

PS : Si vous trouvez une solution élégante à ce menu problème, écrivez-moi SVP.

Commodore 64 en version RGB

EN attendant les nouveaux ordinateurs Commodore (Plus 4, 234 et autres), qui n'en finissent pas de finir d'arriver, on pourra se consoler avec une nouvelle version du talentueux C.64 : le C.64 RGB. Intégrée à l'unité centrale, la carte RGB donnera au cheval de bataille de Commodore une qualité de reproduction d'image meilleure et facilitera grandement le branchement TV, réalisé directement par la prise Péritel.

Quant aux infortunés possesseurs d'un téléviseur non-Péritel — ils sont nombreux — Commodore a annoncé qu'un nouvel adaptateur offrant une liaison Secam d'excellente qualité était disponible sous le nom de code d'« Oscar ». Des accessoires qui devraient faciliter la vie de nombreux utilisateurs. Il faut avouer, 18 mois après la sortie effective du Commodore 64, qu'il était temps. ■

Miami : les logiciels CNRS

LE laboratoire informatique du CNRS vient de publier une revue, gratuite, destinée au logiciel. Sous le titre aguicheur de Miami, ce magazine propose des essais logiciels et divers articles sur des thèmes tels que EAO (Enseignement Assisté par Ordinateur), image de synthèse, etc... Des comptes rendus de projets réalisés dans le club du CNRS seront également publiés. ■

UN PETIT TOUR CHEZ LE LIBRAIRE

Maîtrisez les interfaces de votre micro-ordinateur
Francis Saguez et Christian Andrieux
Editions Eyrolles
Paris, 1984
Broché, 144 pages
Prix : 85 FF

Guide de l'utilisateur
TRS-80 modèle 100
Orson Kellog
Editions Sybex
Paris, 1984
Broché, 106 pages
Prix : 78 FF

Pratique des MS-DOS et PC-DOS
Henri Lilen
Editions Radio
Paris, 1984
Broché, 126 pages
Prix : 90 FF

Programmer en LISP
Henri Farreny
Editions Masson
Collection ABC des langages
Paris, 1984
Broché, 120 pages
Prix : 68 FF

Le système d'exploitation MS-DOS version 1 et 2
Roger Politis et Bruno Vanryb
Editions Eyrolles
Paris, 1984
Broché, 216 pages
Prix : 120 FF

Guide de l'utilisateur Commodore 64
Joseph Kascmer
Editions Sybex
Paris, 1984
Broché, 160 pages
Prix : 78 FF

Basic, système et langage-machine
Jean-Christophe Krust
Editions du PSI
Lagny, 1984
Broché, 168 pages
Prix : 90 FF

LISP mode d'emploi
Christian Queinnec
Editions Eyrolles
Paris, 1984
Broché, 320 pages
Prix : 160 FF

CASSETTES ET DISQUETTES

Monams
Moniteur-assembleur
Cassette pour Oric1/Atmos
Edité par Micro-Compta 5
Distribué par VISMO
Prix : 95 FF

Logologic-1
Cassette pour C. 64/Vic20
Edité par No Man's Land
Distribué par Innelec
Prix : 120 FF

Oric-Calc
Cassette pour Oric
Distribué par Microprogrammes 5
Prix : 180 FF

Microdrive Forth 1-1
Deux cartouches pour ZX Spectrum
Edité par DDC/Sémaphore
Prix : 500 FF

Tasword deux
Cassette pour ZX Spectrum
Traitement de texte
Edité par DDC/Sémaphore
Prix : 255 FF

Logiciels pour Canon X-07

EXHAUSSÉS. Les utilisateurs du X-07 le seront bientôt dans leur quête du logiciel. **Logi'Stick**, société spécialisée dans la production de logiciels pour ordinateurs portables vient de sortir son catalogue Automne-Hiver avec quelque 15 titres différents. Au menu un Calc, une gestion de fichiers, des problèmes de maths

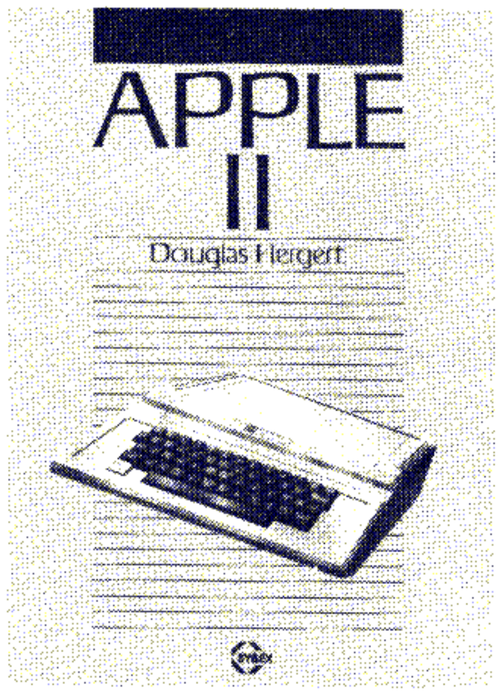
(niveaux 1ère , « Math Sup » et « Math Spé ») ainsi que deux aides à la programmation.

Aide est destiné à l'optimisation des programmes Basic et langage-machine (à noter au passage un moniteur Hexa-Décimal), et *Ass/desasse* réunit quatre programmes pour l'apprentissage des codes machines Z 80.

Ces logiciels tournent bien entendu sur écran 4 lignes (celui du X-07) mais sont également adaptés à l'affichage TV (pour ceux qui ont acquis l'interface Péritel de **Canon**). Ils sont disponibles chez **DDI**, Duplication et Diffusion Informatique, au prix de 150 FF chacun. ■

UN LIVRE

Guide du Basic Apple II
Douglas Hergert
Editions Sybex
Paris, 1984
Broché, 286 pages
Prix : 78 FF



CE guide du Basic de l'Apple II utilise une formule très classique ; quelque 153 mots y sont recensés, formant un véritable dictionnaire qui regroupe instructions, commandes et fonctions du langage avec, en prime, certains termes supplémentaires (comme Menu ou Notation scientifique).

Quelques reproches en passant : pour cette version française, où l'on prend soin de définir le mot ''programmeur'' (sic), nous aurions aimé pouvoir trouver quelques indications sur ce qu'est un ''slot''. En revanche, les détails mathématiques sur SIN ou LOG étaient peut-être superflus.

Au total, un ouvrage de référence bien commode pour ceux qui ont oublié comment écrire ''arc tangente'' ou les codes de la haute résolution en couleurs.

AW ■

Koala Pad
Un manuel en français

LA petite tablette graphique testée sur Commodore 64 dans *LIST 3*, peut être trouvée chez certains distributeurs comme **SPID** ou **Innelec**, avec un mode d'emploi en français. Le distributeur exclusif, **BIP**, la fournit avec un manuel en anglais !

SPID
39 rue Victor Massé
75009 Paris
Innelec
110 bis avenue du Général Leclerc
93506 Pantin Cedex
BIP
13 rue Duc
75018 Paris ■

Logiciel québécois pour Commodore 64

CHICONTINI et Sherbrooke sont les lieux de naissance d'un logiciel québécois de traitement de texte : *Traitex 64*. Conçu par des programmeurs, bien entendu canadiens, *Traitex* est un des premiers traitements de texte acceptant tous les accents de langue française y compris les ë, ï et ü. Nouvellement commercialisé aux Etats-Unis, il est vendu également en France sous le nom de « Virgule » (édité par Micro Application Software, pour un prix de 750 FF). Dans quelques mois, *Traitex 64* sera complété d'un programme de correction orthographique contenant plus de 125 000 mots. Affaire à suivre. ■

Sharp PC-2500 :
un super compatible

SHARP a annoncé le lancement d'un nouveau portatif (et non de poche), baptisé PC-2500. A son actif : un écran LCD de 4 lignes de 24 caractères avec une définition de 150 × 32, une imprimante-table traçante 4 couleurs et un Basic assez complet qui devrait permettre notamment certaines acrobaties graphiques.

Quant à la surprise, elle vient de l'entière compatibilité du PC-2500 avec les logiciels et programmes développés sur les autres machines Sharp. Le Basic devrait même permettre la réutilisation de données spécifiées sur un autre appareil.

Pour couronner le tout, des cartes mémoires seront disponibles en deux versions : 8 et 16 Ko. Le PC-2500 coûtera environ 3 500 FF et n'est disponible pour l'instant qu'au Japon. ■

Utilitaires Oric

FORT du principe : « on peut faire autre chose que des jeux d'arcade avec son Oric », **ARG Informatique** propose désormais divers utilitaires pour Oric 1 et son compère Atmos. *Super Copy Ecran* permettra aux créateurs et graphistes chevronnés de conserver intactes leurs œuvres en produisant une copie conforme de l'écran.

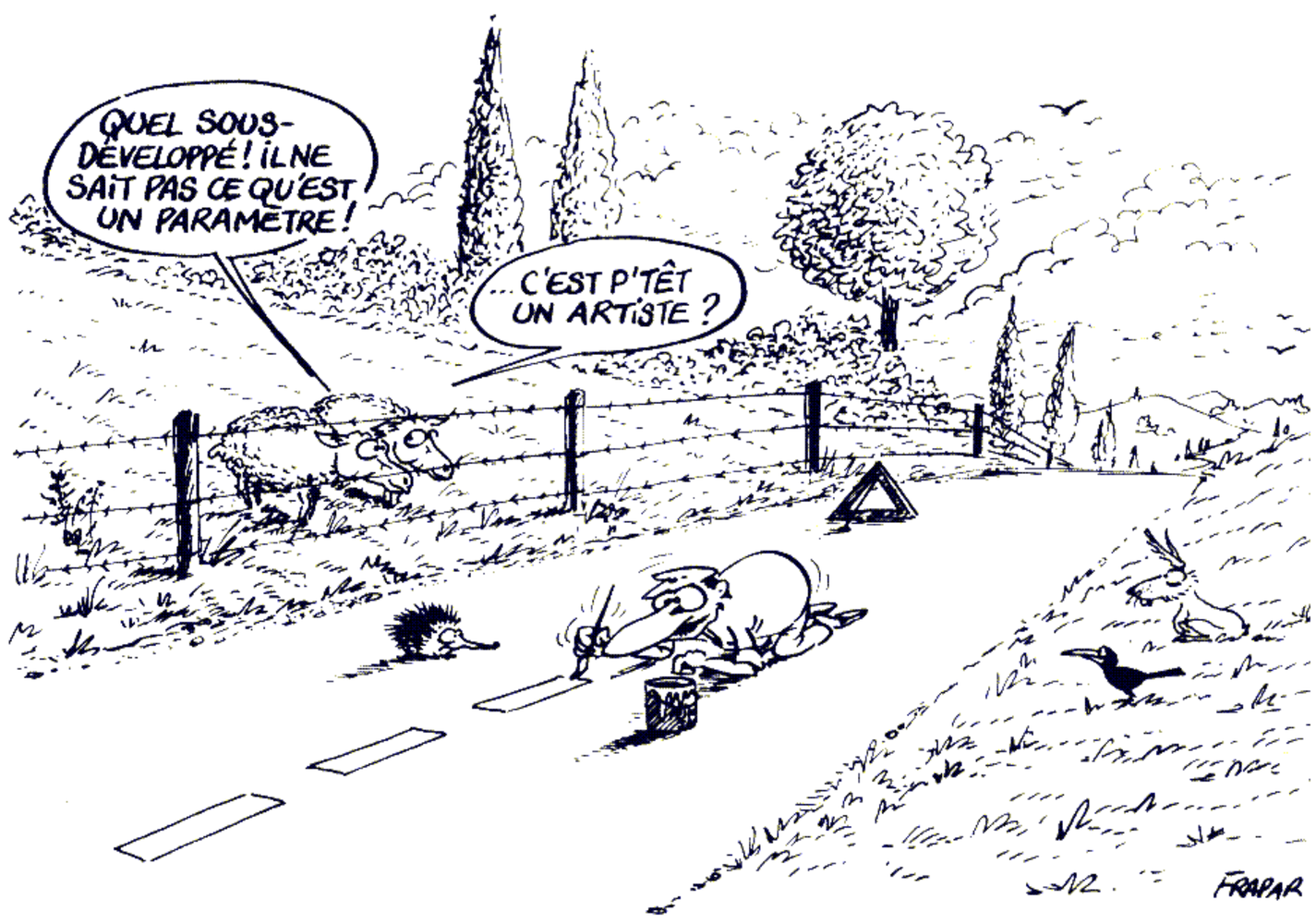
Haute, basse résolution, simple ou double densité, noir sur fond blanc ou blanc sur fond noir tout est permis même les caractères double hauteur. Utile mais n'oubliez pas l'imprimante.

Côté langage-machine *Oric Basic Plus 1* offre la possibilité de renuméroter un programme Basic (en entier ou en partie), d'en supprimer des pavés ou encore de « mélanger » plusieurs programmes. *Oric Basic Plus 1* est garanti « 100 % langage-machine ». Et si le Forth est une de vos lubies ou obsessions, *J'apprends l'Forth* devrait vous apporter la solution. ■

PARAMÉTRER POUR MIEUX TRACER

Où l'auteur se livre à un regrettable attendrissement sur ses erreurs de jeunesse. Comme s'il en était sorti !

PROGRAMMER le tracé d'un trait horizontal sur l'écran est une opération simple. Et plus on démêle les ficelles de la programmation, en particulier celles des paramètres, plus cette opération fait gagner du temps et de la place.



Pour séparer proprement différentes parties d'un programme il suffit de tracer, sur l'écran, un trait horizontal. Il peut apparaître avec une ligne de la forme : 100 PRINT ''---------- (etc., j'abrège) -----''.

Aujourd'hui, en frappant ces lignes, j'ai l'index suffisamment flemmard pour se trouver tout dolent et courbattu de frapper tant de tirets. Mais à l'époque, chaque fois qu'il fallait un trait, un PRINT ''----- convenait parfaitement. Et il tirait sur la mémoire, c'en était navrant !

Un grand pas a été franchi le jour où j'ai découvert l'usage du sous-programme : il a suffi de frapper la fastidieuse ligne une fois, du côté des 9000, de la faire suivre d'une instruction RETURN, et un simple GOSUB 9000 faisait le reste. Et puis, comme la machine peut se charger du répétitif (elle fait ça si bien), la ligne 9000 est devenue : 9000 FOR I=1 TO 39:PRINT ''-'';: NEXT I:PRINT: RETURN.

Si le compteur d'octets fou est à l'écoute, il aura remarqué que la ligne 100 en fait 5 (prise en charge, numéro compris) + 1 pour PRINT + 39 pour les tirets + 2 pour les guillemets, soit 48. La ligne 9000 ''newlook'' fait 5 + 25 signes ou mots-clés, soit 30, et même 25 si l'on supprime les espaces : on n'est pas loin des 50 % d'économie.

On pouvait même adapter sans dou-

leur le programme à d'autres largeurs d'écran, surtout en déclarant au début du programme LE=39:REM LARGEUR DE L'ECRAN et en mettant en 9000 : FOR I=1 TO LE... Même que si on voulait sortir quelque chose sur l'imprimante, c'était aussi bigrement commode.

Du côté des professionnels

Et puis, au fil des siècles (je ne fais pas mon âge), le procédé s'est affiné. En fait j'ai simplement piqué l'idée dans le programme d'un autre. C'est d'ailleurs l'intérêt (le seul) de rentrer des programmes au clavier : on apprend des tours de mains pas possibles, surtout si on est un as de la fautte de frape commme vorte sreviteur, parce que pour trouver l'erreur, on est obligé de comprendre comment ça marche. Bref, le sous-programme en 9000 a été remplacé par une initialisation : 100 T$="": FOR I=1 TO LE:T$=T$+"-":NEXT I et là, un PRINT T$ (3 octets contre 5 pour GOSUB 9000) tirait le trait.

Un petit regard vers des programmes écrits par des professionnels nous apprend qu'ils paramètrent abondamment. Prenons l'exemple d'un menu de fichier :

Voici ce qu'on peut en faire si l'on essaie d'éliminer les constantes :

```
100 LE=39:REM LARGEUR D'ECRAN
110 DATA "SORTIE DE TOUTES LES FICHES", "SELECTION DE FICHES"
120 DATA "ENTREE DE FICHES", "MODIFICATION DE FICHES"
130 DATA "SUPPRESSION DE FICHES", "TRI CROISSANT", "TRI DECROISSANT"
140 DATA "ENREGISTREMENT DES FICHES", "FIN DES OPERATIONS"
150 DIM N$(9):FOR I=1 TO 9:READ N$(I):NEXT I
160 T$="":FOR I=1 TO LE:T$=T$+"-":NEXT I
```

Les initialisations paraissent longuettes ? Voyons le menu:

```
500 PRINT CHR$(147);TAB(18);"[RVS]MENU[OFF]":PRINT T$
510 FOR I=1 TO 9:PRINT"[RVS]";I;"[OFF]";N$(I):PRINT T$:NEXT I
520 PRINT "VOTRE CHOIX ?"
```

Et toc ! Mais on gagne encore à avoir mis les différentes options en tableau, car le choix sera un nombre compris entre 1 et 9, baptisé N, par exemple. Et chaque option commencera par un PRINT CHR$(147);N$(N):PRINT T$. Ainsi on aura un écran effacé, le rappel de l'option et un trait horizontal.

Dans le cours d'un programme, il est aussi possible de gagner pas mal de place en évitant la formule CHR$(13) pour les retours-chariot ou CHR$(34) pour les guillemets en déclarant au début : C$=CHR$(13):G$=CHR$(34)

Où l'on peut vraiment s'amuser

Reprenons le cas d'un menu. Un petit paquet de lignes courtes à affi-

```
500 PRINT CHR$(147);
510 PRINT TAB(18);"[RVS]MENU[OFF]"
520 PRINT "----------"
530 PRINT "[RVS]1[OFF] SORTIE DE TOUTES LES FICHES"
540 PRINT "----------"
550 PRINT "[RVS]2[OFF] SELECTION DES FICHES"
560 PRINT "----------"
570 PRINT "[RVS]3[OFF] ENTREE DE FICHES"
580 PRINT "----------"
590 PRINT "[RVS]4[OFF] MODIFICATION DE FICHES"
600 PRINT "----------"
610 PRINT "[RVS]5[OFF] SUPPRESSION DE FICHES"
620 PRINT "----------"
630 PRINT "[RVS]6[OFF] TRI CROISSANT"
640 PRINT "----------"
650 PRINT "[RVS]7[OFF] TRI DECROISSANT"
660 PRINT "----------"
670 PRINT "[RVS]8[OFF] ENREGISTREMENT DES FICHES"
680 PRINT "----------"
690 PRINT "[RVS]9[OFF] FIN DES OPERATIONS"
700 PRINT "----------"
710 PRINT "VOTRE CHOIX ?"
```

PARAMÉTRER POUR MIEUX TRACER

cher, ça prend de la place dans un programme, à cause des instructions PRINT:

```
1000 PRINT''SORTIE'' :PRINT
     ''SELEC'': PRINT''ENTREE''
     : PRINT''MODIF''
     :PRINT''TRI''
1010 PRINT''SUPPR'' :PRINT
     ''ECRIT'' :PRINT ''FIN''
```

Ces lignes peuvent devenir :

```
100  C$=CHR$( 13)
1000 PRINT''SORTIE'';C$;
     ''SELEC'';C$;''ENTREE''; C$;
     ''MODIF''; C$; ''TRI'';C$;
     ''SUPPR''
1010 PRINT''ECRIT'';C$;''FIN''
```

Et en supprimant les points-virgules :

```
1000 PRINT ''SORTIE''C$
     ''SELEC'' C$''ENTREE''
     C$''MODIF'' C$''TRI'' C$
     ''SUPPR''C$ ''ECRIT''
     C$''FIN''
```

Si l'on a lu (comme plus haut) dans N$(X) les différents articles du menu pour qu'ils puissent aussi servir de tête de chapitres, on peut encore, en début de programme mettre :

```
200  MN$=CHR$(147):FOR I=1 TO
     9:MN$ = MN$ + N$(I) + C$;
     NEXT I
```

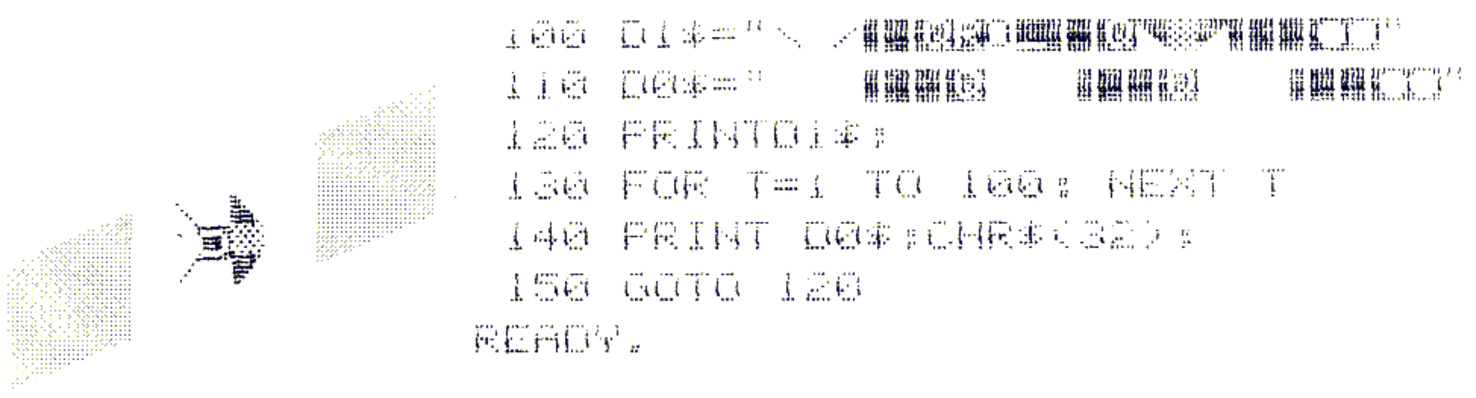
Alors, un simple PRINT MN$ affiche le menu complet !

Mais là où l'on peut vraiment s'amuser, c'est avec les blocs graphiques sur les machines qui disposent de caractères de mouvement de curseur :

```
100  D1$=''[RVS][3 ESPACES][3
     CRSR GAUCHE][1 CRSR
     BAS][3 ESPACES][3 CRSR
     GAUCHE][1 CRSR BAS][3
     ESPACES][3 CRSR
     GAUCHE][2 CRSR
     HAUT][RVS OFF]''
110  D0$=MID$(D1$,2,LEN(D1$(-1))
```

Vaisseau spatial à l'écran

On vient de constituer un carré de 3 × 3 en ''vidéo inversée'', et un carré blanc qui pourra effacer le premier. L'intérêt est que, chaque fois qu'on voudra appeler l'un ou l'autre, un petit PRINT D1$ ou PRINT D0$ suffira.



Ajoutons donc pour voir :

```
120 PRINT D1$;
130 GET R$:IF R$='''' THEN 130
140 PRINT D0$;CHR$(32);
150 GOTO 120
```

Un carré traverse gentiment l'écran pas à pas chaque fois que l'on enfonce une touche.

On corse ? Corsons ! Sur un Commodore, on met à la place de la ligne 100 :

```
100 D1$=''[SHIFT M][ESPACE]
    [SHIFT N][3 CRSR GAUCHE]
    [1 CRSR BAS][RVS]O[RVS
    OFF][3 CRSR GAUCHE][1
    CRSR BAS][SHIFT FLECHE
    GAUCHE][SHIFT &][SHIFT
    PARENTHESE FERMEE][3
    CRSR GAUCHE][2 CRSR
    HAUT]''
```

A la place de la ligne 110 :

```
110 D0$=''[3 ESPACES][3 CRSR
    GAUCHE][1 CRSR BAS][3
    ESPACES]
    [3 CRSR GAUCHE][1 CRSR
    BAS][3 ESPACES][3 CRSR
    GAUCHE][2 CRSR HAUT]''
```

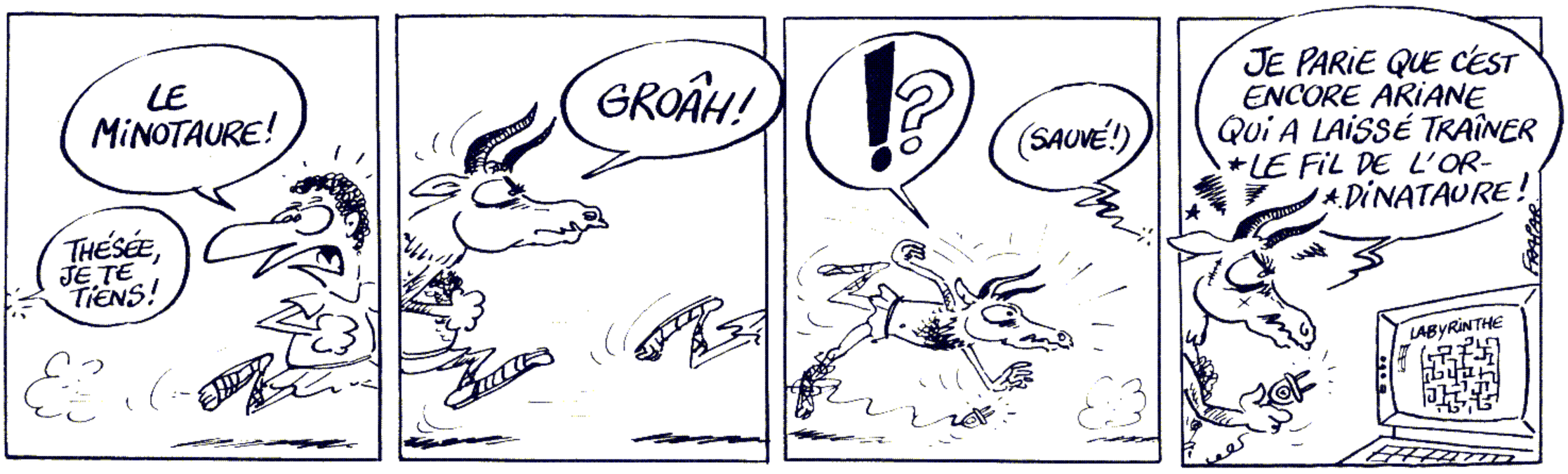
Et à la place de la ligne 130 :

```
130 FOR T=1 TO 100:NEXT T
```

Et voilà un petit vaisseau spatial qui traverse et retraverse l'écran. Il donne bien envie d'écrire une suite au programme pour essayer de communiquer avec les extra-terrestres qui l'habitent.

Comme quoi rien de tel que la contrainte des paramètres pour ouvrir la porte au rêve, et même au rêve économique !

François J. BAYARD



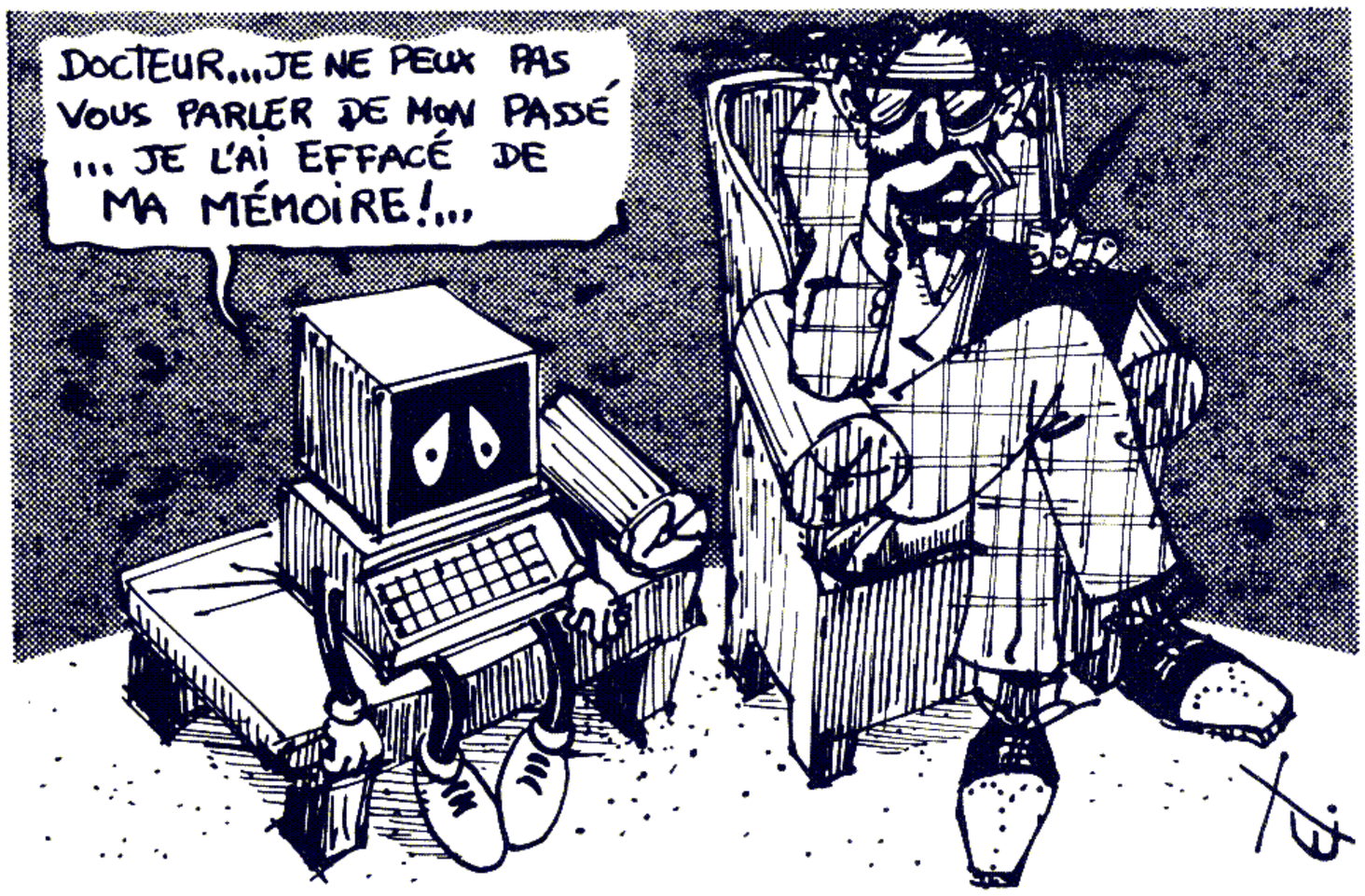
ANALYSE D'UN RAMASSE-POUSSIÈRE

TOUT programmeur rencontre un jour le fléau des mémoires vives, fléau avec lequel il a maille à partir : cette « calamité » nécessaire est plus connue sous le nom de « garbage collection ». Elle consiste à nettoyer systématiquement des zones de mémoire encombrées.

Les utilisateurs de la HP-41 C sont habitués au nettoyage systématique des zones encombrées : une fonction PACK permet de récupérer des octets inutilisés au sein d'un programme.

Malheureusement, la plupart des ordinateurs déclenchent ces grandes manœuvres quand bon leur semble... Le moment venu, celui qui est absorbé par sa tâche, interrompt toutes ses autres activités pendant le temps nécessaire. Ce délai, indécelable dans la plupart des cas, peut atteindre plusieurs minutes lorsqu'on déplace un grand nombre de chaînes en mémoire (essayez le programme de démonstration de la page suivante pour vous en convaincre) : l'ordinateur présente tous les symptômes du plantage et le programme tourne au ralenti (fini le tri rapide).

Afin de dominer la situation et de tirer le meilleur parti de *votre* machine, une enquête de mœurs s'impose.



Conseils de première urgence

Les chaînes alphabétiques n'occupent pas toujours des zones fixes en mémoire : les chaînes que l'on assemble à l'aide d'instructions du type CHR$, MID$, STR$... s'inscrivent dans une zone de travail généralement située au sommet de la mémoire vive. Retenez maintenant ceci : plus vous conservez de chaînes en mémoire et plus le temps imparti au « nettoyage » est important. L'ordinateur ne tient pas compte, en inspectant la mémoire, des chaînes dont vous vous débarrassez.

Sur certaines machines, les délais peuvent varier comme le carré du nombre de chaînes présentes en mémoire (vérifiez-le en modifiant la boucle FOR... NEXT dans l'exemple).

Cependant les algorithmes de « salubrité » étant des plus complexes, il reste à déterminer le comportement exact de *votre* ordinateur dans les cas extrêmes : il pourra, par exemple, être amené à effectuer cette tâche plus souvent et plus vite quand la mémoire sature ; ou bien, plus souvent et... très

PROGRAMME TRÈS UTILE

ANALYSE D'UN RAMASSE-POUSSIÈRE

lentement à chaque fois qu'un pointeur déclenche l'alarme.

Il n'existe pas à ce jour de remède miracle contre ces phénomènes incontrôlés, sinon de chercher à les prévoir ou à les provoquer. Voici donc quelques conseils de première urgence.

Tout d'abord, bien diagnostiquer le mal : élaborez un programme qui vous permettra de découvrir la zone de mémoire susceptible de subir le nettoyage (utilisez une boucle qui agrandit une chaîne et un test PEEK qui remonte la mémoire).

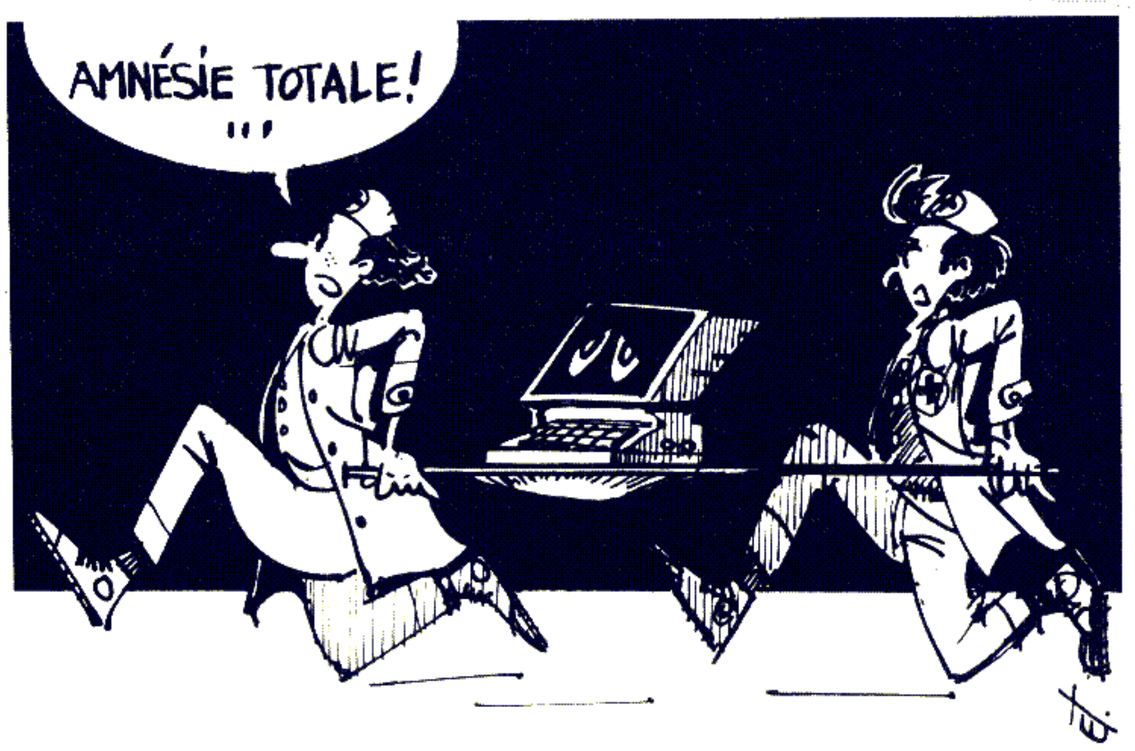
Lorsqu'un de vos programmes souffrira de « garbage collectionite aiguë », une courte enquête sur les lieux du crime vous permettra de retrouver la routine coupable. Bien souvent, il s'agit d'une routine qui déplace de longues chaînes (tri, tableur, gestion de fichiers). N'oubliez pas qu'une routine du genre :

```
10 A$ = INKEY$
20 IF A$ = "" THEN 20
30 IF A$ <> CHR$ (13) THEN STOP
40 B$ = B$ + A$
50 GOTO 10
```

peut provoquer un « nettoyage » par encombrement de la mémoire vive restante.

Écrire un bon programme revient à respecter certaines règles d'hygiène au sein de la mémoire de l'ordinateur. Bien que les notices soient le plus souvent muettes à ce sujet, nous pouvons énoncer quelques règles précieuses.

Il faut éviter à tout prix de déplacer des chaînes alphabétiques en mémoire ; on utilisera des tables



Programme de démonstration

```
100 ' Programme pour rencontrer un ramassage de poubelles
110 DEFINT I-N
120 '
130 ' Commençons par créer des chaines
140 DIM A$(640)
150 FOR I=1 TO 640
160 A$(I)=CHR$(46)          ' Code du point
170 NEXT
180 '
190 ' Puis créons notre chaine de 'remplissage'
200 B$="AA"
210 FOR I=2 TO 6
220 B$=B$+B$
230 NEXT
240 ' La longueur de b$ est maintenant 2^6 soit 64
250 B$=B$+B$+B$+B$   'et maintenant 192 caractères
260 '
270 ' ... et démarrons le remplissage de la mémoire
280 PRINT"Attention, on y va...."
290 T=TIME
300 K=0             ' Attention, c'est un INT
310 ' on boucle...
320 IF K>32000 THEN PRINT"Pas arrivé au bout de 32000 boucles, on arrete":STOP
330 B$=B$+""        ' pas méchant, non ?
340 T1=TIME
350 IF T1-T<=2 THEN T=T1:K=K+1:GOTO 310
360 PRINT"Il y a eu ramassage de poubelles au bout de ";K;"exécutions"
370 PRINT"Temps pour le ramassage:";T1-T;"secondes (+ ou - 2s)"
380 PRINT"mais ce n'est pas fini...."
390 T=TIME
400 X=FRE(".")
410 PRINT"Le ramassage prend encore ";TIME-T;"secondes !"
```

A l'exécution du programme, le ramassage a lieu

```
Attention, on y va....
Il y a eu ramassage de poubelles au bout de  272 exécutions
Temps pour le ramassage: 59 secondes (+ ou - 2s)
mais ce n'est pas fini....
Le ramassage prend encore  59 secondes !
```

En remplaçant la ligne 160 par 160 A$ (I) = ".", il n'y a plus de ramassage

```
Attention, on y va....
Pas arrivé au bout de 32000 boucles, on arrete
Break in 320
```

Délais, délais

Selon le modèle ou la version d'une machine, le mode de ramassage peut varier considérablement, et même échapper aux investigations. Cependant tout délai observé supérieur à 1 seconde mérite d'être pris en considération : en bouclant 10 000 fois, le même programme passerait près de 3 heures en pure perte !

En pratique, les délais correspondent à des ramassages partiels proportionnels au remplissage de la mémoire, ponctués en fin de cycle par un ramassage plus long et plus complet.

d'index mobiles (variables numériques) qui définiront des équivalences entre une liste de messages et une liste de nombres (exemple : 1 rouge, 2 noir, 3 bleu, etc.).

Une visite médicale

Quand le transfert est inévitable, penser à affecter une chaîne nulle aux variables désaffectées pour faire de la place.

Ne pas travailler sur des listes ou des tableaux surdimensionnés : il est toujours possible de les sectionner en sous-groupes plus petits sauvegardés sur disquette ; avant de charger un nouveau bloc, on peut initialiser les tableaux de chaînes (avec DIM, par exemple) et forcer le grand « nettoyage » par un FREE (ou FRE (0) ou SIZE) au moment où un minimum de chaînes est affecté (ne pas oublier le message « PATIENTEZ SVP »). L'opération est d'autant plus rapide.

Quoi qu'il en soit, prenez-vous par la main et faites passer une bonne visite médicale à votre machine. Lorsque vous dominerez bien ses réactions vis-à-vis du ramassage des poussières, vous n'aurez plus qu'à concevoir un squelette de logiciel armé contre tous les risques connus (gestion de la mémoire sur disquette, tableaux indexés numériquement ; un signal d'alarme sous la forme du pointeur de la zone mémoire utilisée pour la construction des chaînes, vous permettra de détecter l'imminence du ramassage).

La fonction TIME

LA fonction TIME met en évidence le phénomène de manière automatique. Certains préféreront observer celui-ci « manuellement », en plaçant une instruction sonore (BEEP ou SOUND) avant le GOTO de la ligne 350 du programme de démonstration, ou de la ligne 320 du programme de test. Celui-ci jouera le rôle d'un métronome, les irrégularités du « bip » refléteront l'importance et la fréquence du ramassage, qui pourra être évalué avec un chronomètre pendant les longues pauses. D'une manière générale, le principe devrait être appliqué dans tout programme pour suivre son évolution à l'aide de messages sonores ou de texte.

Programme de test

```
100 '   Programme pour tester le "ramassage de poubelles"
110 '                            (garbage collecting)
120 DEFINT I-N
130 B$=" ###### "
140 T1=0:T2=0:T3=0:T4=0:T5=0 'On définit les variables pour avoir les memes FRE
150 PRINT:PRINT " Nb ch  Rempl.  FRE(0)         FRE('X')  Poub!   ERASE":PRINT '
160 N=10
170 ' Boucle
180 PRINT USING B$;N;
190 T1=TIME
200 DIM A$(N)
210 FOR I=1 TO N
220 A$(I)="."+CHR$(46)
230 NEXT
240 T2=TIME:PRINT USING B$;T2-T1;
250 PRINT USING B$;FRE(0);
260 T3=TIME:PRINT USING B$;T3-T2;
270 PRINT USING B$;FRE("X");
280 T4=TIME:PRINT USING B$;T4-T3;
290 ERASE A$
300 PRINT USING B$;FRE("X");
310 T5=TIME:PRINT USING B$;T5-T4
320 N=2*N:IF N<1000 THEN 170
```

Résultats du programme de test

Nb ch	Rempl.	FRE(0)		FRE('X')	Poub!	ERASE	
10	0	55524	0	55534	1	55596	0
20	0	55464	0	55484	0	55596	0
40	1	55344	0	55384	0	55596	0
80	1	55104	0	55184	1	55596	0
160	1	54624	1	54784	4	55596	0
320	2	53664	0	53984	15	55596	0
640	5	51744	0	52384	59	55596	0

En remplaçant la ligne 220 par 220 A$ (I) = ''. .'', les résultats sont différents

Nb ch	Rempl.	FRE(0)		FRE('X')	Poub!	ERASE	
10	0	55560	0	55560	0	55602	0
20	0	55530	0	55530	0	55602	0
40	1	55470	0	55470	0	55602	0
80	0	55350	0	55350	1	55602	0
160	1	55110	0	55110	0	55602	0
320	2	54630	0	54630	0	55602	0
640	3	53670	0	53670	0	55602	0

Sur ce Basic Microsoft, le ramassage n'apparaît qu'avec FRE (X$) ; sur d'autres, il apparaît aussi avec FRE (X).

Qu'advient-il des lutins ?

Précisons enfin un point particulier : l'ordinateur qui collecte les octets dispersés simule parfaitement la catalepsie sans espoir de rémission. Cependant on peut espérer dans la plupart des cas un juste retour à la normale.

Qu'advient-il des ordinateurs qui gèrent des *lutins* ou qui vocalisent pendant le nettoyage de printemps ? Deux cas de figure se présentent : mes expériences personnelles montrent que si les lutins poursuivent leur chemin quand le reste du programme s'arrête, on court à la catastrophe. Malheureusement si le logiciel pense à immobiliser les lutins pendant la durée nécessaire, rien n'est prévu pour les remettre en mouvement correctement ! Les aberrations résultantes sont facilement contrôlables par logiciel, heureusement.

Désormais, de nouvelles méthodes de programmation risquent fort de s'imposer : nous nous sommes laissés dire qu'un programmeur averti en valait deux.

Michel ARDITTI

PASCAL ET SA FAMILLE

UNE étude chronologique des langages informatiques commence nécessairement par les trois ancêtres : le langage-machine, Fortran et Cobol. Ensuite, le choix devient bien plus difficile. Tenter de dresser une liste est même impossible ; tout au plus est-il pensable de présenter brièvement quelques familles importantes, ou des cas particuliers intéressants (comme les langages "français", LSE, Prolog ou Ada). Et, puisqu'il faut trancher, c'est la famille Pascal, dont l'influence est sans cesse grandissante, qui est présentée ici : elle réunit les langages Algol, Pascal et C.

Le Pascal est le plus connu de toute une série de constructions a priori, censées aujourd'hui satisfaire à un nouveau principe de base : la structuration des programmes, objectif typiquement pédagogique (au sens large du terme), favorisant leur lecture et leur mise au point. La réussite en est claire : il est par exemple tout à fait possible que ce soit un « Pascal-like » qui remplace le bon vieux Basic sur les machines personnelles de demain. Mais visitons d'abord un passé déjà un peu ancien...

Il y eut d'abord Algol, antérieur même à Cobol, mis au point en Europe dans un environnement universitaire. Une excellente source est, une fois de plus, l'ouvrage fondamental de René Moreau « Ainsi naquit l'informatique » (Dunod), mais aussi les manuels qui étaient consacrés à Algol (il faut employer cet imparfait, car il n'est pratiquement plus en course aujourd'hui). Une première réunion, animée principalement par F.L. Bauer de Munich, se tint à Zurich en 1958. Y travaillèrent notamment Backus, Katz, Perlis, Rutishauser, que rejoignirent plus tard par exemple le mathématicien français Bernard Vauquois (né en 1929, professeur à Grenoble, spécialiste en particulier du traitement automatique des langues), le néerlandais Adrian Van Wijngaarden (créateur des ordinateurs Arra — à l'origine des machines Philips — et professeur d'Edsger W. Dijkstra) et le danois Peter Naur au moment de la transformation d'Algol 58 en la version Algol 60, la plus célèbre et la première réellement « universelle ».

Des qualités reconnues

Le nom initial d'Algol était IAL (International Algebraic Language), ce qui rappelle bien son origine scientifique. De fait, il fut tout de suite clair qu'il ne pouvait que difficilement traiter de problèmes de gestion, aux énormes fichiers. En particulier les procédures d'entrée et de sortie de données, capitales pour ce type d'applications, étaient mal définies et peu efficaces. Il connut donc une carrière commerciale assez médiocre.

Les compilateurs Algol furent surtout le fait de constructeurs européens (sauf Burroughs), et les clients principaux étaient universitaires, moins fortunés que les groupes industriels et commerciaux !

Le sommet de la carrière d'Algol se situe peut-être en 1968, au moment de la publication de sa version la plus évoluée. Elle est notamment due au talent de Donald E. Knuth de l'Université californienne de Stanford, célèbre pour son œuvre capitale « The Art of Computer Programming » — noter l'emploi du mot « Art », là où Wirth, par exemple, voudrait mettre

« science » —, en sept volumes (quatre à paraître), véritable bible professionnelle.

Voici, inspiré d'un traité d'« Analyse numérique linéaire », un programme Algol calculant la somme des valeurs absolues des éléments A(I,J) d'une matrice carrée d'ordre N (pour plus de précision, voir le livre de Noël Gastinel aux Editions Hermann) :

```
procedure norme (A,N) ; reel tableau A ; entier N ; reel S

    debut entier I, J ; S : = 0
        pour I : = 1 pas 1 jusqua N faire
        pour J : = 1 pas 1 jusqua N faire
        S : = S + ABS (A(I,J)) ;
        aller a SORTIE
    fin

SORTIE : NORME : = S
fin ;
```

Il faudrait naturellement ajouter l'entrée des nombres N et des coefficients de la matrice.

Les qualités d'Algol sont reconnues aujourd'hui : il comportait notamment des nouveautés exceptionnelles, le recours systématique à des « procédures »avec variables dites locales — une même lettre peut figurer avec deux sens distincts dans le corps même du programme et dans un calcul annexe —, la récursivité (possibilité de faire fonctionner par un appel direct par exemple un sous-programme à l'intérieur de lui-même), et surtout la première facilité sérieuse de « structuration » par l'emploi de « blocs », délimités par les célèbres BEGIN et END, s'enchâssant au besoin les uns dans les autres. Mais ses défauts sont clairs : lourdeur indéniable, difficulté d'apprentissage surtout chez des débutants non mathématiciens. Et le succès actuel de Pascal, qui lui doit tant, est une sorte de revanche quasi-posthume pour Algol.

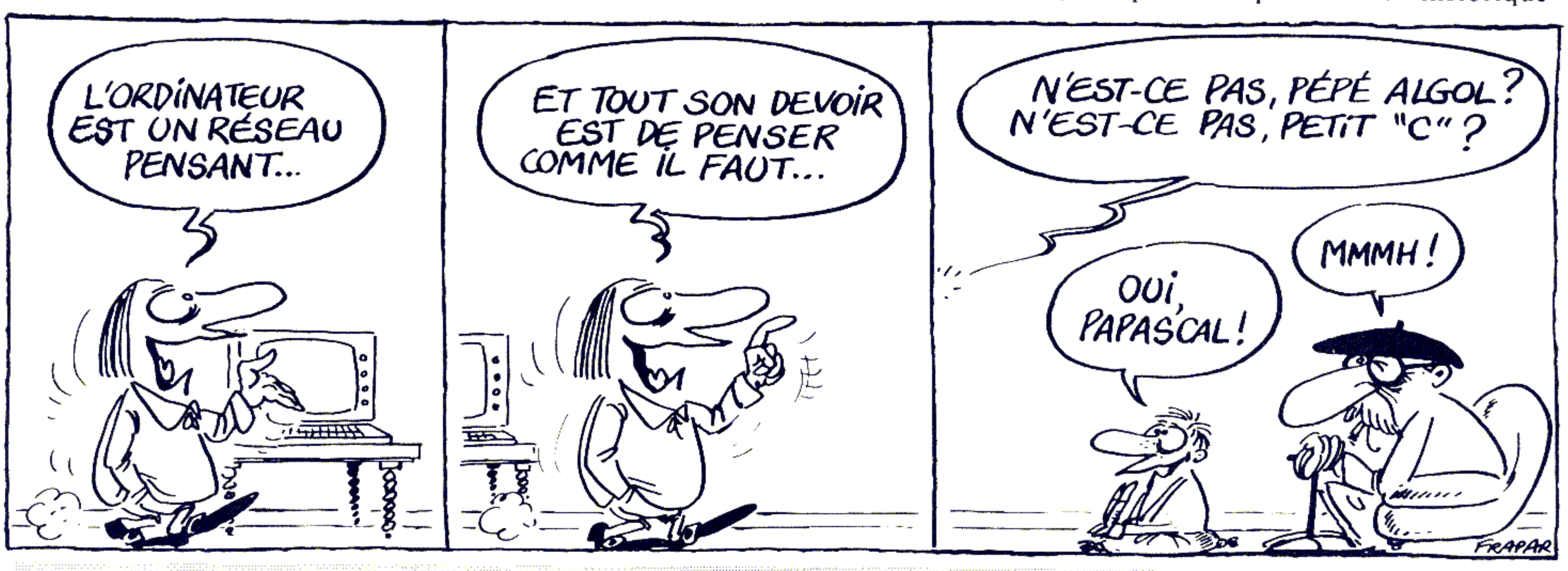
Le langage Pascal est beaucoup mieux connu que son ancêtre direct. Il a justement hérité des propriétés citées ci-dessus (blocs BEGIN/END, récursivité, procédures, donc bonnes possibilités de lisibilité et de maintenance). De plus, les entrées et sorties sont mieux gérées. Mais il n'a pas su non plus échapper à une rigidité assez pénible : même une application très simple exige une longue liste. Le traitement des fichiers est peu efficace par rapport à Cobol (on peut tout de même faire aujourd'hui de la gestion sur ordinateur avec certaines variantes de Pascal spécialement adaptées). La grande infériorité sur le Basic — qu'il dépasse sur de nombreux autres points et qu'il remplacera peut-être un jour dans une version plus souple — est son manque d'interactivité. Mais le bilan est très largement satisfaisant.

Un exemple issu du livre « Premières leçons de programmation » de Jean Arsac (Cedic-Nathan) montre bien la filiation ; il calcule la racine carrée d'un entier par défaut :

```
PROGRAM RACINE

VAR N, R, V, T, INTEGER

BEGIN
WRITE ('N = ');READ(N);
WHILE N>=0 DO
    BEGIN
    R:=0;T:=1;N:=1;
    WHILE T<=N DO
        BEGIN
        V:=V+2;
        T:=T+V;
        R:=R+1
        END;

    WRITELN('RACINE(',N,') = ',R);
    WRITE('N = ');READ(N)
    END;
END.
```

On pourrait encore ajouter de nombreuses lignes de commentaires, éclairant par exemple les égalités fondant l'algorithme très simple utilisé ici.

Le créateur de Pascal voulait notamment obtenir vers 1968 « un langage souple et puissant que l'on puisse implanter assez efficacement sur la plupart des ordinateurs » (in le « Rapport révisé » de 1973, traduit chez Eyrolles). Il cite comme sources explicites Algol 60 et sa variante Algol W. Le professeur Niklaus Wirth travaille à Zürich (Eidgenössische Technische Hochschule): son premier compilateur fut achevé en 1970 sur un CDC 6000 (Control Data) en langage-machine (voir par exemple un bref historique



PASCAL ET SA FAMILLE

dans le numéro de février 1983 de la revue « Pascalissime »). Il utilisa ensuite un intermédiaire universel (le *P-code)* permettant de passer plus commodement à des machines réelles à partir d'un modèle idéal (la *P-machine)*, ce qui garantit une certaine portabilité.

Un langage universel ?

Les versions sur micro-ordinateurs sont souvent dites « UCSD » (Université de Californie à San Diego), du nom de la version de Kenneth Bowles qui, tout en créant des extensions intéressantes, avait su optimiser la place nécessaire au compilateur originel, trop gourmand (près de 100 Ko). Apple, dès 1979, proposait un UCSD ; Tandy eut d'abord un Tiny Pascal (tiny = ténu), puis diffusa le Pascal Alcor.

Le langage a été normalisé en 1973 par l'ISO. En tête d'un programme doivent figurer de nombreuses déclarations (notamment des types de variables). Cette gêne — c'est ainsi que le ressent le débutant — correspond très bien aux vues pédagogiques de Wirth, qui voulait contraindre le programmeur à plus de rigueur en l'empêchant de pianoter à toute vitesse n'importe quoi sur son clavier. On sait que le GOTO est quasiment absent de Pascal : certains ont eu le plus grand mal, au bout de quelques mois, à dénouer les fils du ballet des indicateurs trop bien imbriqués sur lequel l'auteur avait fait reposer un algorithme trop « astucieux ».

Pascal aurait pu devenir le langage « universel » recherché, par exemple, à partir de 1975, par le Département de la Défense américain (processus qui aboutit, comme on le sait, à la naissance du langage Ada, auquel sera consacré tout un prochain article). Mais il n'atteignait à la rigueur souhaitée que d'une manière un peu réductrice, qui le rendait par exemple assez mal approprié à l'exécution des tâches en temps réel. Aussi fut-il abandonné, à regret, mais en précisant que le prototype finalement retenu devrait s'inspirer de lui.

D'ailleurs, son père lui-même, comme tous les vrais informaticiens de grande classe, fut très vite lucide sur les défauts de son enfant, et il chercha à y remédier (seuls les disciples sans originalité se fixent avec obstination sur des positions rappelant parfois les anathèmes des guerres de religion). Bien que né seulement en 1934, Niklaus Wirth avait, déjà avant Pascal, une grande expérience de la construction de langages : ingénieur électronicien de l'ETH, il avait pu travailler à Stanford sur PL 360, puis Algol W ; il recommença vers 1975 en créant Modula, profondément transformé entre 1977 et 1980 en Modula II. Il existe d'ailleurs aujourd'hui aux États-Unis, par exemple, un « Journal of Pascal, Ada & Modula-2 »...

Un autre successeur de Pascal est sans doute le langage C, très en vogue aujourd'hui car lié au système Unix d'AT&T. Il est né en 1972 dans les laboratoires de la célèbre Bell Telephone à Murray Hill (New Jersey), d'une équipe dirigée par Dennis Ritchie. Le chapitre 1 de l'excellente introduction de Claude Nowakowski (« Programmer en C », PSI 1984) et la traduction du livre de base de Ritchie et Kernighan (Editions Masson) nous apprennent qu'il descend d'une longue chaîne algolique, par l'intermédiaire de CPL (Strachey, 1965) BCPL (Richard, 1969) et de « B » de Ken Thompson en 1970...

Blaise Pascal, génie scientifique

Langage très structuré comme Pascal, très original, par certains côtés proche d'un assembleur, C est promis à un grand avenir : il est puissant, efficace, utilisable en gestion et en calcul, implémentable sur des machines de plus en plus nombreuses.

Au départ, il était plutôt défini pour aider à la conception de compilateurs...

La littérature disponible à son sujet est heureusement en pleine extension et permet donc de se faire une idée de ses caractéristiques.

Toute la production des langages modernes est donc, comme on le voit, née de Fortran et d'Algol, mais surtout fortement marquée par l'œuvre principale de Wirth : sans aucun doute, il a bien fait de placer publiquement son travail sous l'invocation de Blaise Pascal, esprit remarquable et universel, lié à l'informatique par tant de côtés (première machine à calculer à connaître une fabrication en petite série avec prospectus publicitaire et représentant de commerce attitré — Roberval, au demeurant professeur au Collège de France... —, travaux mathématiques admirables en combinatoire, calcul différentiel et intégral et probabilités, texte sur l'« esprit géométrique »). L'influence de Pascal ne saurait se limiter aux *Pensées* ou aux *Provinciales*, par lesquelles on est initié à son œuvre au lycée : très bientôt, peut-être par le biais de leurs ordinateurs, les collégiens finiront par savoir que c'était aussi un génie scientifique...

André WARUSFEL

UN BONHEUR DE TABLE

Pouvoir sortir sur une imprimante
une table de conversion décimal-hexa et retour,
pour que chacun puisse en avoir un exemplaire,
c'est pas ça, le bonheur ?

Il existe, c'est vrai, des esprits matheux qui jonglent dans leur tête, s'amusent avec les bases et convertissent de décimal en hexa comme d'autres font les règlements de la Sécurité Sociale, c'est-à-dire visiblement en pensant à autre chose. Rendons-leur hommage. Il existe aussi, ce n'est pas moins vrai, d'excellents petits programmes de conversion décimal-hexadécimal. Ils n'ont qu'un défaut : ils ne sont jamais là quand on a besoin d'eux. Un défaut à réparer !

Laissant les convertisseurs à ceux qui ont un œil tourné vers l'avenir (d'où ce léger strabisme qui fait leur charme), je me rabats régulièrement, à l'ancienne, sur l'écrit-papier : il y a, dans un de mes premiers livres d'informatique, une table de conversion. Et lorsqu'il tombe par terre, c'est là qu'il s'ouvre, avec la crasse vénérable, les ronds de fonds de verres et les coins cornés, toute la chaude patine des pages complices cent fois relues. Comme je ne peux tout de même pas le prêter à tout le monde, je vous propose un programme qui sort sur imprimante une table identique à celle de mon livre. Il est écrit pour n'importe quel Commodore muni, ne serait-ce que temporairement, d'une imprimante, mais son adaptation à d'autres matériels ne doit poser aucun problème.

A l'origine, il devait servir dans le cadre d'un club, permettant à chacun de convertir gaiement sans avoir à se battre à quinze sur un même livre. Pourtant, depuis, chez moi, j'ai collé des tables partout : dans tous mes classeurs, au mur, sous mon ordinateur, pour ne pas avoir à chercher. Mais trêve de détails intimes, allons droit au pratique : comment s'en sert-on ?

D'hexadécimal en décimal, ça dépend si le nombre à traduire comporte deux chiffres ou plus. S'il comporte deux chiffres, pas de problème, on le cherche dans la colonne du milieu (voir page suivante), celle marquée HEXA P.FA, et on a la réponse dans la colonne de gauche, celle marquée DEC. C'est ainsi qu'à gauche de 9F, écrit 009F, on lit 159.

Si le nombre hexadécimal comporte quatre chiffres, on le coupe en deux par le milieu. $5BD4, par exemple, sera donc tronçonné en 5B d'une part et D4 d'autre part. La moitié gauche est appelée « octet de poids fort » et la moitié droite « octet de poids faible ».

On commence par chercher 5B dans la colonne du milieu, mais cette fois-ci il faut regarder ce qu'il y a en face dans la colonne de droite marquée HEXA P.FOR. On lit 23296, et on recopie ce nombre sur un bout de papier, sur une

tablette de cire, ou au doigt sur l'écran (si on a une tartine beurrée à la main). Reste à trouver D4, l'octet de poids faible. Colonne de gauche, on lit en face : 212. Une rapide addition donne 23508 et on ne gagne rien si on a trouvé la bonne réponse.

Si le nombre hexadécimal comporte trois chiffres, on force un zéro en tête et l'on est ramené au problème précédent.

Quant à la conversion dans l'autre sens (un thème, quoi !), si le nombre est inférieur à 256, c'est simple, on le cherche en colonne de gauche, et on a la réponse au milieu. Sinon, on cherche dans la colonne de droite le nombre inférieur le plus proche et... il vaut mieux prendre un exemple. Soit un nombre entre 256 et 65536. Au hasard, 1984. On parcourt d'un index agile la colonne de droite. 2048, c'est trop... le nombre immédiatement inférieur est 1792. En face de 1792, colonne du milieu, on trouve 07 (0007, en fait) et on prend note. Reste alors à se livrer à une soustraction : 1984 – 1792 = 192, à chercher 192, colonne de gauche, on lit au milieu C0 (00C0, en fait). On raboute les deux moitiés, on met un dollar en tête pour bien montrer qu'il s'agit d'hexa, et on obtient $07C0. Nous sommes en l'an de grâce $07C0. Tiens, pour l'an prochain, j'enverrai des cartes de vœux avec un gros $07C1 dessus, ça en surprendra plus d'un.

Pour ceux qui veulent adapter ce programme à leur machine, je signale ceci :

• CHR$(32) (ligne 110 du programme pour Commodore), c'est tout simplement un espace ;

• GET (ligne 190) attend un caractère au clavier (il correspond au INKEY$ de certains Basic) ; moyennant une petite transformation, il peut être remplacé par un INPUT ;

• les caractères de contrôle de curseur (entre crochets) sont seulement là pour faire joli ;

• pour sortir sur imprimante, on ouvre un fichier (ligne 220) et l'on écrit dessus (PRINT # 4,), puis on le referme poliment (ligne 320). Essayez LPRINT...

• enfin, le signe "%" (ligne 350) désigne un nombre entier ; au lieu de D% = D, on peut faire X = INT(D).

Notons au passage l'astuce du sous-programme de conversion (elle n'est pas de moi) : au lieu de mettre en DATA les chiffres et les lettres de 0 à F, on utilise le fait que "0" c'est CHR$(48) et que CHR$(57) c'est "9". Donc CHR$(48 + N) rend compte des chiffres de 0 à 9. Mais si N vaut 10, alors le chiffre hexa est "A", c'est à dire CHR$ (65), et non CHR$ (57). Qu'à cela ne tienne ! Il est si simple de se servir des booléens dont parlait Robin Bois dans le numéro 1 de *LIST* : lorsque N est inférieur à 9, l'expression (N>9) est fausse, donc vaut 0 ; lorsque N est supérieur à 9, l'expression (N>9) est vraie, donc vaut − 1. Si l'on multiplie l'expression par 7, (N>9)*7 vaut 0*7, soit 0, si N est inférieur à 9 ; et − 1*7 soit −7, si N est supérieur à 9. Par conséquent, (48 + N −(N>9) *7), tant que N est inférieur à 9, vaut 48 + N − 0 soit 48 + N et sort un caractère numérique entre "0" et "9". Mais quand N vaut 10, (48 + N − (N > 9) *7), vaut 48 + N − (− 1) * 7 soit 48 + 10 − (− 7) = 65, et sort "A".

Evidemment, pour le ZX 81, le problème ne se pose pas, puiqu'il a un code à part, où "0" = CHR$ (28) et où les lettres suivent les chiffres ; "A" = CHR$ (38). Il suffit donc de demander CHR$(28 + N). D'ailleurs, sur ZX, une expression vraie vaut 1, et non −1, alors... pour l'exemple, la transposition a été faite. Mais, seulement pour l'exemple !

Hextable (prononcer Ex-taibeule)
Programme pour ZX81 et imprimante
Auteur François J. Bayard


```
100 REM * HEXTABLE * FJB 1984 *
110 LET A$=CHR$ 0
120 LET B$=A$+A$+A$
130 LET C$=A$+B$
140 LET D$="DEC HEXA  HEXA "
150 LET E$="     P.FA P.FOR "
160 CLS
170 PRINT "HEXTABLE"
180 PRINT ,,,,"CE PROGRAMME SOR
T SUR IMPRIMANTE"
190 PRINT ,,"UNE TABLE DE CONVE
RSION"
200 PRINT ,,"DECIMAL/HEXA SUR D
EUX COLONNES."
210 PRINT ,,,,"FRAPPER ■ SI L I
MPRIMANTE"
220 PRINT ,,"EST PRETE."
230 IF INKEY$<>"I" THEN GOTO 23
0
240 PRINT ,,,,"IMPRESSION EN CO
URS"
250 FAST
260 LPRINT D$;A$;D$
270 LPRINT E$;A$;E$
280 FOR I=1 TO 126
290 IF I/33=INT (I/33) THEN LPR
INT ,,D$;A$;D$,E$;A$;E$
300 FOR J=0 TO 1
310 LET D=I+126*J
320 LET K=D
330 LET X$=B$+STR$ K
340 LPRINT X$(LEN X$-2 TO );A$;
350 GOSUB 470
360 LPRINT A$;
370 LET X$=C$+STR$ (K*256)
380 LPRINT X$(LEN X$-4 TO );A$;
A$;
390 NEXT J
400 LPRINT
410 NEXT I
420 SLOW
430 CLS
440 PRINT AT 11,12;"TERMINE"
450 STOP
460 REM *** DECIMAL>HEXA ***
470 LET D=D/4096
480 FOR H=1 TO 4
490 LPRINT CHR$ (28+INT D);
500 LET D=16*(D-INT D)
510 NEXT H
520 RETURN
```

Hextable (prononcer Ex-taibeule)
Programme pour Commodore et imprimante
Auteur François J. Bayard


```
100 REM *** HEXTABLE *** FJB 1984 ***
110 A$=CHR$(32):B$=A$+A$+A$:C$=A$+B$
120 D$=B$+" DEC HEXA  HEXA "
130 E$=B$+"     P.FA P.FOR "
140 PRINT"[CLR][RVS]HEXTABLE[DOWN][DOWN
    ]"
150 PRINT"     CE PROGRAMME SORT SUR IM
    PRIMANTE"
160 PRINT"[DOWN]UNE TABLE DE CONVERSION
     DECIMAL/HEXA"
170 PRINT"[DOWN]SUR QUATRE COLONNES."
180 PRINT"[DOWN][DOWN]FRAPPER [RVS]I[RV
    SO] SI L'IMPRIMANTE EST PRETE."
190 GET R$:IF R$="" THEN 190
200 IF R$<>"I" THEN 330
210 PRINT"[DOWN][RVS]IMPRESSION EN COUR
    S[RVSO]"
220 OPEN4,4
230 PRINT#4,D$;D$;D$;D$
240 PRINT#4,E$;E$;E$;E$
250 PRINT#4
260 FOR I=1 TO 64:PRINT#4,B$;
270 FOR J=0 TO 3:D=I+64*J:K=D
280 PRINT#4,A$;RIGHT$(B$+STR$(K),3);
290 PRINT#4,A$;:GOSUB 350:PRINT#4,A$;
300 PRINT#4,RIGHT$(C$+STR$(K*256),5);C$
    ;
310 NEXT J:PRINT#4:NEXT I
320 CLOSE 4
330 PRINT"[CLR][RVS]TERMINE[RVSO].":END

340 REM ********* DECIMAL>HEXA *********
350 D=D/4096:FORH=1TO4:D%=D:PRINT#4,CHR
    $(48+D%-(D%>9)*7);:D=16*(D-D%):NEXT:
    RETURN
READY.
```

Extraits de la table de conversion décimal-hexadécimal

DEC	HEXA P.FA	HEXA P.FOR
1	0001	256
2	0002	512
3	0003	768
4	0004	1024
5	0005	1280
6	0006	1536
7	0007	1792
8	0008	2048
9	0009	2304
89	0059	22784
90	005A	23040
91	005B	23296
92	005C	23552
93	005D	23808
94	005E	24064
155	0098	39680
156	009C	39936
157	009D	40192
158	009E	40448
159	009F	40704
160	00A0	40960
187	00BB	47872
188	00BC	48128
189	00BD	48384
190	00BE	48640
191	00BF	48896
192	00C0	49152
193	00C1	49408
207	00CF	52992
208	00D0	53248
209	00D1	53504
210	00D2	53760
211	00D3	54016
212	00D4	54272
213	00D5	54528
214	00D6	54784
215	00D7	55040
216	00D8	55296

(exemple avec ZX81 et imprimante)

François J. BAYARD

EH BIEN, COMPILEZ MAINTENANT

UN langage de programmation n'est pas directement compris par un processeur. Celui-ci ne peut exécuter que ses codes. Une traduction s'impose. Elle peut avoir lieu de deux manières : soit globale, grâce à un compilateur, soit instruction par instruction, grâce à un interpréteur.

La possibilité d'utiliser un langage symbolique de haut niveau (en mode interactif) a contribué fortement au succès des micro-ordinateurs. Le langage Basic, développé initialement sur les gros systèmes (travaillant en temps partagé), a été universellement adapté aux ordinateurs individuels.

Il faut voir derrière tout langage un volumineux programme de traduction qui permet de passer d'un texte plus ou moins symbolique (« programme-source »), à une liste de codes directement exécutables par le processeur (« programme-objet »).

Les compilateurs génèrent, à partir du code-source, un code intermédiaire qui est ensuite transformé en un programme directement exécutable. Les interpréteurs travaillent exclusivement sur le texte-source d'origine : les instructions sont décodées une par une au cours de chaque exécution. Les compilateurs offrent donc une grande rapidité d'exécution, et les interpréteurs la possibilité de suivre pas à pas le déroulement du programme (ce qui facilite considérablement la mise au point).

Aujourd'hui, le programme complexe qui permet au processeur de travailler en Basic est logé dans une mémoire électronique (un minuscule boîtier de plastique noir placé sous le capot de l'engin) : on imagine mal le degré d'élaboration qu'atteint ce genre de programme (Basic est inspiré du Fortran, autre langage symbolique dont l'élaboration du compilateur a demandé un travail énorme, près de 18 années-homme vers 1956). On peut s'en faire une bonne idée en désassemblant le Basic, ou tout autre programme de ce genre (interpréteur APL, compilateur Fortran, etc.).

Et ce domaine n'est pas interdit à l'amateur. Un coup d'œil sur la presse non professionnelle le prouve : des raisons très variées et parfois curieuses ont conduit des amateurs à écrire des interpréteurs Basic, Logo, Pilot... et des compilateurs Pascal, C, Ada (!). Certains ont été jusqu'à créer des langages spécialisés (Basex,...).

Il est impossible ici de détailler par le menu le fonctionnement d'un compilateur ou d'un interpréteur. On doit se contenter de quelques notions importantes.

P-machine, l'ordinateur fait idée

L'exemple du compilateur Tiny-Pascal peut les présenter : facilement accessible, il met bien en évidence le principe de traduction-exécution. Le programme-source (voir exemple page suivante) est converti en « P-code » (ce code intermédiaire est le langage de programmation d'une machine fictive dite P-machine). Il est ensuite exécuté à partir d'un interpréteur simple, mais efficace : il conserve des possibilités d'aide à la mise au point (trace, point d'arrêt, reprise,...). L'avantage de ce principe est de se limiter à l'écriture d'un unique compilateur standardisé (programme long). Seul l'interpréteur (programme court) est spécifique d'un système donné. D'autre part, dans certaines versions, le programme en « P-code » peut lui-même être compilé en code-machine (dit « langage-machine »).

Un compilateur peut être écrit dans presque n'importe quel langage. S'il

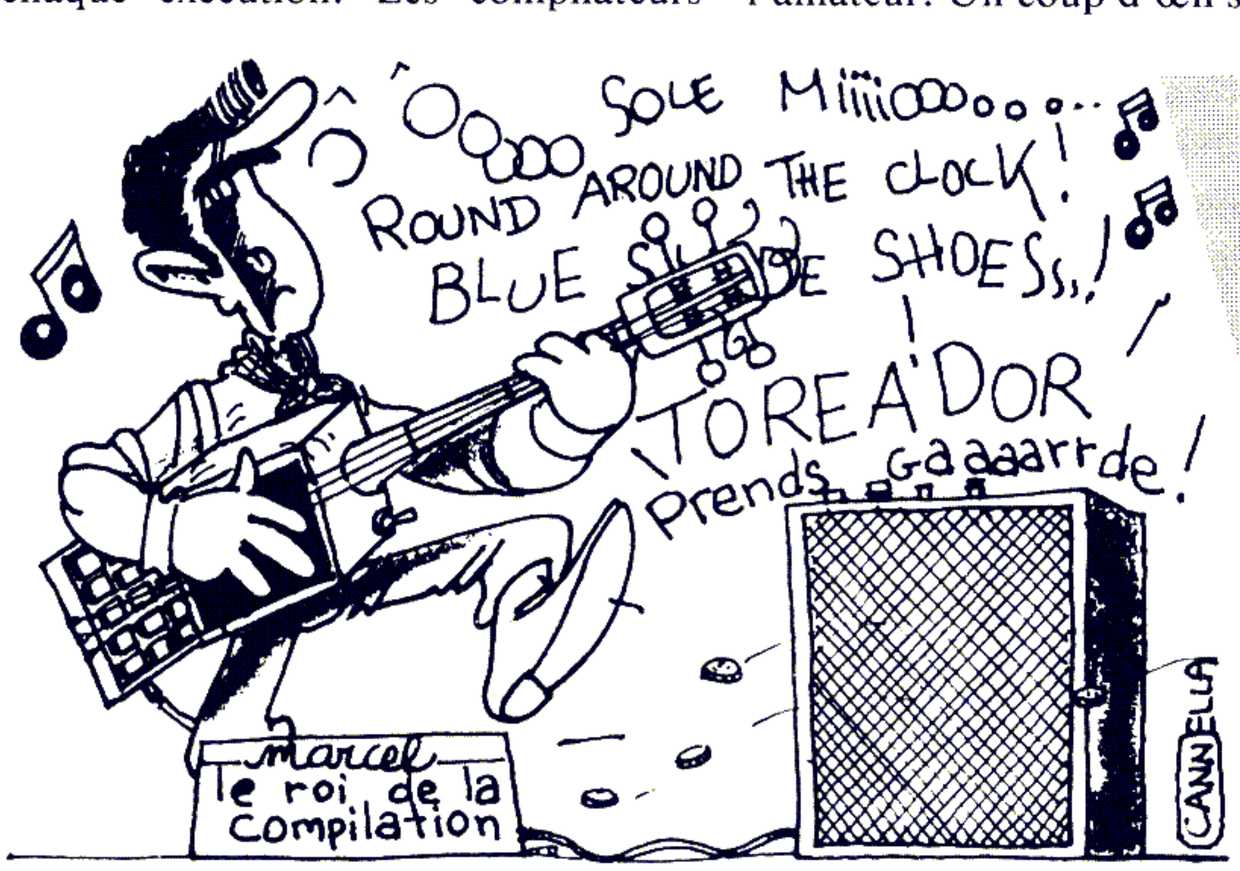
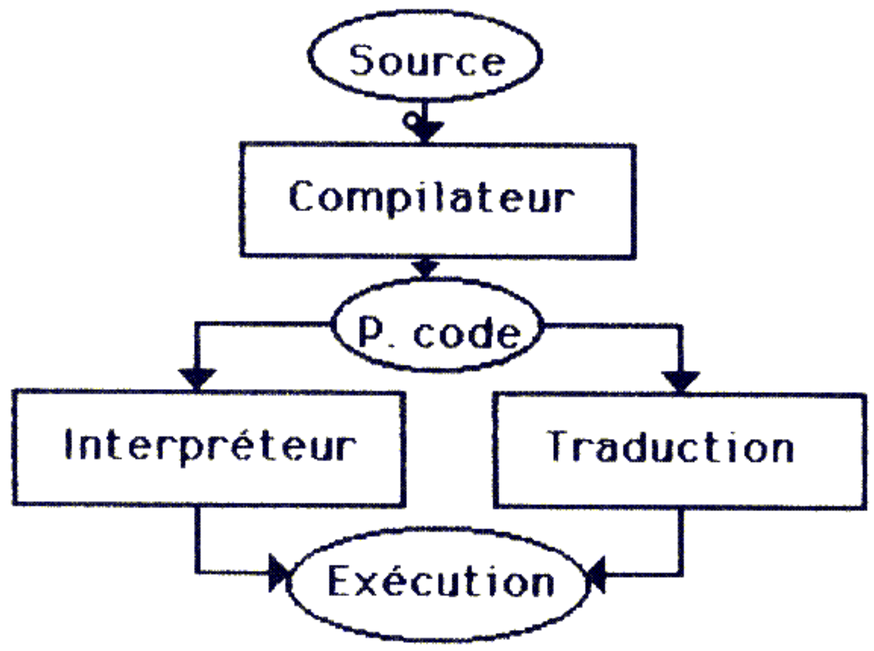


Figure 1
Traduction en deux étapes



Exemple de traduction-exécution avec Tiny-Pascal

Programme-source en Tiny-Pascal

```
VAR I : INTEGER ;
BEGIN
      FOR I := 1 TO 10000 DO
END.
```

Programme en P-code

0 JMP 0 3	Sauter à l'instruction 3	P = 3
3 INT 0 4	Augmenter le pointeur de pile de 4	T = T + 4
6 LIT 0 1	Empiler la constante 1	T = T + 1;S[T] = 1
9 STO 0 19	Stocker (en position 19)	S[base(0) + 19] = S[T];T = T—1
12 LIT 39 16	Empiler la constante 10000	T = T + 1;S[T] = 10000
15 CPY0	Copier sommet de pile	T = T + 1;S[T] = S[T—1]
16 LOD 0 19	Charger (sommet de pile)	T = T + 1;S[T] = S[base(0) + 19]
19 GEQ0	Test « supérieur ou égal »	S[T] = S[T] > = S[T + 1]
20 JPC0 0 33	Si sommet de pile vrai, sauter en 33	Si S[T] vrai, P = 33; T = T—1
23 LOD 0 19	Charger	T = T + 1;S[T] = S[base(0) + 19]
26 INC0	Incrémenter pointeur sommet de pile	T = T + 1
27 STO 0 19	Stocker (en position 19)	S[base(0) + 19] = S[T];T = T—1
30 JMP 0 15	Sauter à l'instruction 15	P = 15
33 DECT	Décrémenter pointeur sommet de pile	T = T—1
34 RET0	Retour 0 = fin de programme	

S[T] : pile de données
T : pointeur de pile
P : pointeur de programmes

existe en deux versions (une en Tiny-Pascal et l'autre en Basic, par exemple) on peut se livrer au contrôle suivant : compiler le compilateur version Tiny-Pascal en P-code à l'aide de la version Basic ; compiler le compilateur version Tiny-Pascal au moyen du compilateur obtenu en P-code. Les deux codes obtenus doivent être rigoureusement identiques.

Cela dit, le Tiny-Pascal, sous-ensemble du Pascal, convient très bien à l'écriture de compilateurs. Sa souplesse permet d'ajouter des éléments ou de les compléter par améliorations successives, jusqu'au but final. Avec le compilateur Tiny-Pascal, la traduction se fait en deux étapes.

La machine fictive ou P-machine comprend quatre registres de 16 bits :
- T registre du sommet de la pile de travail (pointeur) ;
- B registre d'adresse de base pour le calcul des adresses dans la pile (pointeur) ;
- P registre d'adresse du programme (pointeur) ;
- I registre d'instruction.

La mémoire de la P-machine comprend deux zones :
- la zone des « P-codes », programme à exécuter (zone fixe) PRG ;
- la zone de travail, la pile des données S (zone variable) qui contient toutes les variables de la procédure active et certaines adresses.

Le compilateur ne détermine aucune adresse absolue : celle-ci est calculée au moment de l'exécution (allocation dynamique). La pile comprend un ensemble de blocs, correspondant aux procédures actives. A la base de chaque bloc, trois emplacements SL, DL et RA permettent le chaînage : SL (Static Link) pour la hiérarchie des procédures, DL (Dynamic Link) pour retrouver la base précédente à la fin de chaque procédure et RA (Return Adress) pour localiser en zone PRG (P-code), l'instruction à exécuter après l'appel de procédure.

Le registre de base B pointe toujours l'emplacement de départ du segment de bloc de données sur la pile.

Pour mieux comprendre ce mécanisme subtil, il suffit d'examiner certains ordres de cette P-machine (voir l'encadré « certains ordres de la P-machine »).

Les interpréteurs sont basés sur le même principe : décodage d'une instruction, exécution, passage à l'instruction suivante. On peut donc progresser pas à pas, avec des arrêts à volonté : toute chose bien agréable et efficace dans la mise au point rapide des programmes.

La technique du code intermédiaire, type « P-code », est classique, et les codes retenus sont très variés. Autre exemple : l'un des nombreux compilateurs C génère un code d'assemblage directement utilisable, comme source, par un macro-assembleur.

Avant de voir comment le texte-source est transformé en P-code, c'est-

Certains ordres de la P-machine

- CALL L,A : appel du sous-programme situé à l'adresse A, de niveau L. Les actions sont :
  S[T + 1] = BASE (L)
  S[T + 2] = B
  S[T + 3] = P
  B = T + 1
  P = A (adresse de SP)

  BASE (L) est la fonction qui permet de remonter à la base correspondant au niveau d'appel L.
  FUNCTION BASE (L) : B1 = B
  Tant que L > 0
  B1 = S [B1]
  L = L—1
  BASE = B1
- RETURN : fin de procédure, retour. Les actions sont :
  T = B – 1
  B = S[T + 2]
  P = S[T + 3] (retour)
- ADD : addition. Les actions sont :
  T = T – 1
  S[T] = S[T] + S[T + 1]
- LOAD L,D : chargement variable niveau L, déplacement D. Actions :
  T = T + 1
  S[T] = S[BASE (L) + D]
- STORE L,D : sauvegarde variable niveau L, déplacement D. Actions :
  S[BASE (L) + D] = S[T]
  T = T – 1
- JUMP A : saut à l'adresse A. Action :
  P = A

à-dire comment se fait la compilation proprement dite, il faut définir parfaitement le langage.

La description formelle de sa syntaxe se fait couramment au moyen de diagrammes (Conway, bien connu des utilisateurs de Pascal).

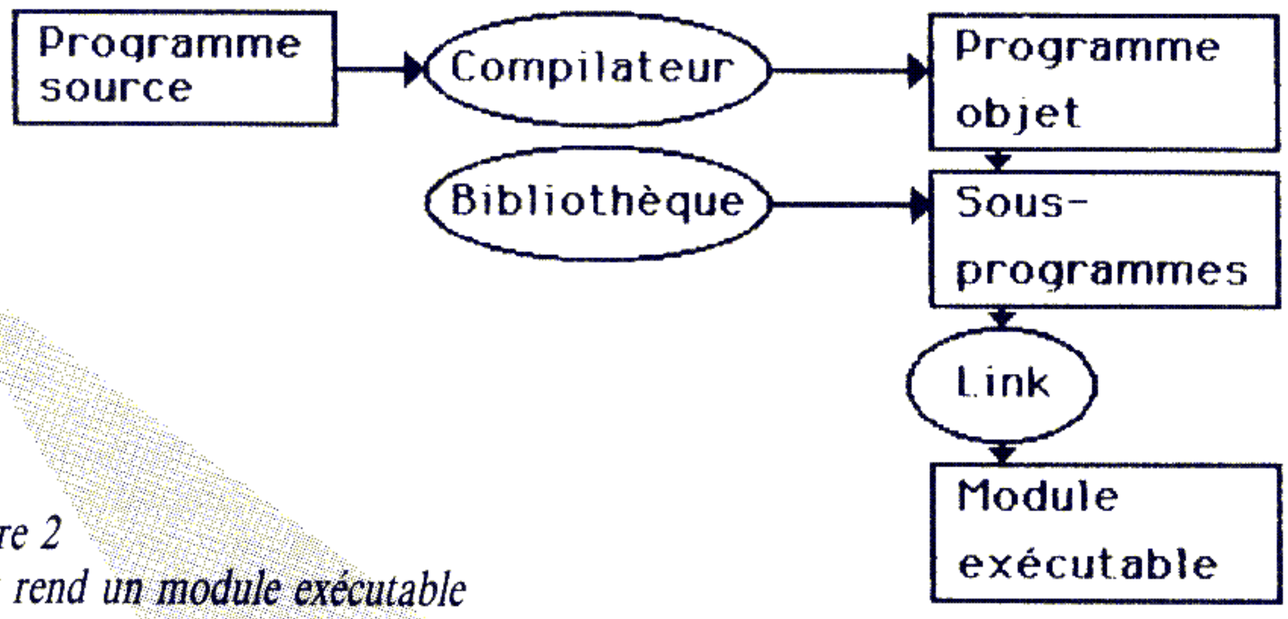


Figure 2
Link rend un module exécutable

Ils présentent deux avantages : d'une part, ils définissent parfaitement la syntaxe, d'autre part, ils conduisent l'écriture du compilateur suivant quelques règles élémentaires. Leur configuration exprime aussi les propriétés du langage associé à l'algorithme d'analyse syntaxique (théorie des langages). Ce langage doit obéir à des règles rigoureuses (récursivité, réduction des phrases,...).

Attention : Vitesse limitée

Le compilateur comprend donc des routines d'analyse et de génération de codes, des tables de variables et de fonctions, des routines de tri et de recherche dans les tables, etc. L'étude de la liste d'un compilateur est un exercice enrichissant. On y trouve une grande variété d'algorithmes avec, en prime, récursivité, optimisation, etc.

Ces notions sont très succinctes, et tous les compilateurs ne sont pas réalisés selon ce principe. Certains génèrent, comme objet intermédiaire, un code relogeable. Le programme, ou chaque partie du programme, est traduit dans un code proche du langage-machine, donc presque utilisable, mais de nombreuses adresses ne sont pas figées, et le code est translatable. Un programme spécifique (LINK) met bout à bout les parties de programme (figure 2), y compris les sous-programmes de bibliothèque, et calcule à chaque étape les adresses absolues du code exécutable.

A titre d'exemple, le module-objet généré par le compilateur Fortran Microsoft comprend deux types de codes, absolus et relogeables, sous forme de chaînes binaires :

- premier bit à 0 : absolu. Les huit bits suivants sont chargés comme octet absolu ;
- premier bit à 1, lire les deux bits suivants :

00 code spécial (nom de programme, module de librairie, références internes ou externes, longueur du programme, bases pour les calculs d'adresses, fin de programme...) ;

01 programme (relatif), chargement des 16 bits suivants après ajout de la base du programme ;

10 donnée (relatif) ;

11 zone des variables communes (relatif).

Cette opération prend souvent beaucoup de temps sur les petits systèmes à cause des recherches des sous-programmes de bibliothèque système (librairie) ; en contre partie, les programmes absolus qui en résultent sont très performants pour les calculs à répétition (tris, recherches...).

Les tests de vitesse donnent quelques indications, mais il faut être prudent !

Signalons enfin deux langages de programmation qui méritent une attention particulière : APL et Forth. Tous deux sont très bien adaptés aux micro-ordinateurs. Ils restent méconnus à cause du « phénomène » Basic : la force de l'habitude... Les performances de ces langages interprétés ou « semi-compilés » ne sont pas prises en considération comme elles le devraient (encombrement mémoire faible et vitesse de traitement élevée).

Les passionnés de programmation ou les utilisateurs désireux d'entreprendre des travaux plus sérieux doivent examiner les différents langages symboliques car les possibilités du Basic sont limitées (même si, théoriquement, il permet de traiter tous les problèmes). Il n'est pas toujours facile de choisir le plus court chemin entre l'algorithme et le programme exécutable...

Par exemple, la programmation de jeux a de nombreux adeptes, mais les algorithmes sont parfois de très haut niveau et ils exigent l'emploi du langage-machine : saisie spéciale sur clavier, animation rapide, graphisme détaillé, effets sonores, synthèse vocale, conversion analogique-digitale (manettes à potentiomètres), calculs mathématiques, etc.

Trois solutions sont envisageables pour une telle programmation : faire appel à l'assembleur, utiliser un compilateur, programmer en Forth.

Si l'on s'en tient aux jeux, les sous-ensembles de langage de haut niveau peuvent parfois donner des résultats surprenants (Tiny-Pascal, Tiny C,...). Leur coût est nettement moins élevé que celui des logiciels professionnels, l'encombrement mémoire est réduit, les temps de compilation sont plus courts, le codé généré est plus efficace.

Il reste à savoir si les nouveaux langages de programmation iront dans ce sens : peut-être des (bonnes) surprises à attendre...

Claude NOWAKOWSKI

Bibliographie

A compiler generator - WM Mc Keeman, JJ Horning, DB Wortman - *Prentice Hall, 1970**

Algorithms + Data structure = Programs - N Wirth - *Prentice Hall, 1976**

A small C Compiler for 8080's - R Cain - *Dr Dobb's Journal n° 45, 1979**

Basex - P Warme - *Byte, 1979**

Compiler construction - FL Bauer, J Eickel - *Springer Verlag, 1975*

Compiler construction for digital computer - D Gries - *John Wiley, 1971*

Formalisation des notions de machine et de programme - L Nollin - *Gauthier Villars, 1969**

Pascal sur TRS 80 - C Nowakowski - *PSI, 1983*

Pascal, the language and its implementation - DW Barron - *John Wiley, 1981**

Small C Compiler V2 - JE Hendrix - *Dr Dobb's Journal n° 74-75, 1983**

Techniques de compilation - FRA Hopgood - *Dunod, 1970*

The Byte Book of Pascal - BW Liffick - *Byte magazine, 1979**

Threaded Interpretive Languages - RG Loeliger - *Byte Book, 1981**

** avec liste complète de compilateur ou interpréteur.*

LE BASIC DE L'ATARI 800 XL

Si l'on se souvient qu'Atari fut le pionnier de la console de jeux vidéos et des programmes d'arcades, on ne doit pas être trop étonné par le Basic du 800 XL, une petite « bombe » dans le domaine des sons et des graphiques : des couleurs, des demi-teintes, de la musique presque comme un synthétiseur. Facile à apprendre pour un néophyte, sa richesse cachée enthousiasmera les vieux routiers de la programmation.

Quelques lignes suffiront à décrire une machine qui se fond dans la production actuelle. Un coffret lesté, de quarante centimètres de large, facile à poser sur les genoux (ou une table), un clavier mécanique traditionnel au toucher ferme, un connecteur de cartouches de programmes et des prises d'extensions munies d'un standard propre à Atari font que cet appareil se situe déjà au-dessus de la moyenne.

Un net avantage

Il s'inclut dans la gamme des domestiques et a été pensé en fonction du banal téléviseur de la maison. Aucun risque de brûler le tube cathodique si vous le laissez branché toute la nuit : dès que l'on cesse de toucher au clavier, sous contrôle d'une horloge interne, l'affichage change périodiquement de couleur, même en cours d'exécution d'un programme. Un regret, toutefois : seule la version PAL permet une pleine utilisation des couleurs. Il est donc préférable de se munir d'un téléviseur multi-standard.

Le clavier permet la frappe en majuscules, en minuscules (à l'aide de l'habituelle touche SHIFT), et comporte 29 caractères semi-graphiques (par pression simultanée de la touche « CONTROL » et d'une touche alphabétique). Ces caractères très utiles pour améliorer la présentation (encadrements, petits symboles...) n'ont hélas aucun rapport avec ceux retenus par les systèmes télétex (Télétel, Antiope, etc.). Les fervents du traitement de texte pourront substituer à ces caractères graphiques, un jeu de minuscules accentuées à l'européenne, à l'aide du simple ordre 'POKE 756,204' qui modifie l'adresse mémoire à laquelle débute la table de définition du générateur de caractères. L'affichage se fait sur 24 lignes de 40 caractères, mais seules les majuscules sont utilisables pour écrire les ordres Basic.

L'éditeur permet la saisie d'une ligne en positionnant le curseur en n'importe quel endroit de celle-ci. Quatre touches le déplacent dans les autres directions, par pression simultanée sur la touche « CONTROL ». Malheureusement, l'insertion et l'effacement de caractères dans une ligne nécessitent également cette double manipulation et elles doivent être répétées pour chaque nouveau caractère introduit (ou effacé).

Avec cet outil l'écriture de programmes en Basic ne pose plus guère de problèmes, d'autant qu'il est possible d'introduire la plupart des ordres et commandes sous une forme abrégée limitée à un ou deux caractères suivis d'un point. Ainsi, G. peut être tapé à la place de GOTO. Toutefois la longueur d'une ligne Basic reste limitée à 114 caractères, soit trois lignes d'écran.

La première caractéristique qui frappe le concepteur de programmes est la correction instantanée des erreurs de syntaxe. Nul besoin d'attendre le 'RUN' suivi du sempiternel 'SYNTAX ERROR' pour savoir pourquoi et où l'écriture est défaillante. On le saura dès le 'RETURN' qui marque la fin d'une ligne d'instructions. Sur ce point-là, le Basic Atari prend un avantage certain sur son concurrent Microsoft et les débutants (ou les maladroits) apprécieront.

En fait, dès qu'une ligne est introduite à partir du clavier, l'Atari en fait l'analyse : le texte est stocké et la syntaxe des instructions est vérifiée immédiatement. En plus, chaque fois qu'il rencontre un nom de variable numérique, l'Atari le compare à une liste des variables précédentes ; s'il n'y figure pas déjà, il l'ajoute et réserve en mémoire un certain nombre d'octets pour y ranger ses valeurs futures.

Cette particularité unique permet à tout instant de connaître le contenu d'une variable numérique. Contrairement aux autres Basic, la modification d'une ligne ne remet pas à zéro la liste des variables, pas plus que leurs valeurs. Cette mise en réserve des données qui permet, par exemple, de vérifier un test

Atari 800 XL et Atari 600 XL

LE Basic de l'Atari 600 XL est le même que celui du 800 XL. La présentation du Basic faite ici est donc valable pour les deux ordinateurs. Ce qui les différencie, c'est essentiellement la taille de la mémoire vive : elle est de 16 Ko pour le 600 XL et de 64 Ko pour le 800 XL.

Il semblerait qu'Atari veuille arrêter la fabrication du 600 XL (fin 1984) et son prix actuel est d'environ 1 600 FF ttc (le 800 XL valant environ 2 500 FF ttc).

ou encore le domaine de validité d'une fonction, est particulièrement appréciable pendant la mise au point d'un programme.

Les chaînes se déchaînent

L'Atari ne connaît que deux types de variables : numérique ou alphanumériques. Les premières sont automatiquement gérées par le Basic, qui réserve huit octets pour stocker chaque valeur, plus un octet par lettre qui en compose le nom. Virtuellement, il n'y a aucune limitation (hormis la taille de la mémoire) pour choisir les noms des variables. En outre, TY et TYPE représentent deux variables distinctes.

Le Basic Atari ne fait aucune distinction entre variables entières ou réelles : la précision est de 9 chiffres significatifs. Pour les calculs mathématiques, l'Atari fait mieux que ses concurrents et permet de traiter les valeurs numériques jusqu'à 9.999999999 E + 97, au lieu des habituels 10 E + 38.

Les tableaux de valeurs (ou matrices) sont autorisés, mais leur déclaration est obligatoire, même si les indices utilisés ne dépassent pas 10. Curieusement, le fait de dimensionner un tableau n'initialise pas les valeurs à 0. Après tout ordre DIM, il est prudent de réaliser manuellement cette mise à zéro, par exemple à l'aide d'une boucle FOR...NEXT.

Une remarque encore sur les bizarreries qui émaillent ce Basic non standard : il est impossible de réaliser la saisie d'un élément de tableau à l'aide de l'ordre INPUT. Il faudra utiliser une variable intermédiaire. Ainsi la succession d'ordres INPUT X : N(0) = X sera nécessaire pour remplacer le banal INPUT N(0).

La particularité essentielle de l'Atari provient de la gestion de ses chaînes de caractères. Si la plupart des Basic gèrent des chaînes entre 0 et 255 caractères, Atari ignore cette limitation et on peut ranger des kilomètres de texte sous un seul nom de variable. Une contrainte toutefois : il faut avoir prédimensionné auparavant ces variables à l'aide d'un ordre DIM.

Si, par accident, on dépasse le nombre prévu, les caractères excédentaires sont ignorés, une troncature se faisant automatiquement, sans apparition de message d'erreur. En revanche, il est impossible de créer des tableaux : ils sont réservés aux seules variables numériques. Pour en simuler, il faut créer une chaîne unique et la subdiviser en tronçons d'égale longueur grâce à des opérateurs de troncature.

Ici, nul besoin d'utiliser les classiques MID$, RIGHT$ ou LEFT$: une syntaxe unique, très souple permet d'exécuter toutes ces fonctions de traitement de chaînes avec un seul ordre. Les opérations de concaténation se font sans appel du symbole d'addition et sont un jeu d'enfant. Seul, peut-être, l'équivalent de la forme RIGHT$ est plus délicat à manipuler.

Les habituelles fonctions, ASC, STR$, LEN et VAL sont présentes. Mais attention, la forme VAL ne s'applique qu'aux caractères numériques (VAL (« A ») se solde par un message d'erreur).

Les messages d'erreur sont succincts et se limitent à un code sur deux chiffres. A chacun de les gérer, à l'aide de l'ordre TRAP qui les intercepte sans interrompre le programme. Judicieusement, un ordre POP permet de sortir prématurément d'un sous-programme (quoiqu'il soit possible de le boucler sur lui-même sans trop de risques, à l'exception d'une hypothétique saturation de la mémoire).

La structure d'un programme en Basic Atari est, en fait, plus simple que sur la plupart des autres machines, car il est possible de mettre des labels qui repèrent en clair les parties du programme, sans devoir abuser des remarques. Il est tout de même plus simple d'appeler un sous-programme de musique à l'aide d'une instruction GOSUB MUSIQUE qu'avec l'hermétique forme GOSUB 10000.

Exit les caricatures

Cette « labellisation » est possible pour les formes GOTO, GOSUB, ON GOTO et ON GOSUB. Même les lignes de DATA offrent cette possibilité, avec en plus, le pouvoir de choisir sélectivement leur lecture, par un READ suivi d'un numéro de ligne, ou par un label.

Au chapitre des regrets, les tests conditionnels se limitent au banal IF-THEN. Ici, point de ELSE ni de WHILE-WEND.

On retrouve les habituelles fonctions arithmétiques et trigonométriques. Celles qui travaillent sur des angles le font en degrés ou en radians. Comme souvent, la fonction tangente est absente et il faut la reconstituer à l'aide de SIN et COS. En revanche, la fonction réciproque ATN (pour arctangente) est au menu, ainsi que les logarithmes décimaux et népériens. Les tests logiques

LE BASIC DE L'ATARI 800 XL

sont limités à leur plus simple expression avec AND et OR. Quant aux valeurs booléennes, elles sont représentées par +1 pour vrai et 0 pour faux.

Quant au dessin, l'Atari ne se limite pas aux couleurs : il introduit la notion de teinte. L'ordre SETCOLOR choisit non seulement la couleur (par exemple le rouge), mais ajoute un paramètre de luminance (qui varie de très vif à pâle). Finies les couleurs trop vives, caricatures de la réalité, il devient possible de jouer sur les demi-teintes. Il est dommage que cette palette se limite en Basic à un maximum de cinq couleurs différentes contenues dans cinq registres mémoire de 708 à 712.

Le « sixième périphérique »

En mode texte normal, on utilise une couleur de fond, une couleur de bordure et une couleur d'impression des textes, soit trois couleurs. Il existe deux autres modes texte introduits par les ordres GRAPHICS 1 ou GRAPHICS 2 qui eux, utilisent les cinq couleurs. Dans ces deux modes, les caractères sont dilatés, réduisant le nombre de colonnes et de lignes disponibles. La couleur du caractère est déterminée par son impression en majuscule (ou en minuscule), et son code ASCII supérieur (ou non) à 128.

La syntaxe des impressions est alors différente du banal PRINT. Tout se passe comme si l'écran était un périphérique particulier (le sixième) auquel on envoie un caractère avec l'ordre 'PRINT # 6;'. Si cette notion de périphérique est courante dans le cas d'échanges avec une imprimante, une cassette ou un disque, elle est inhabituelle dans le domaine des impressions.

Il existe 13 modes graphiques différents (GRAPHICS 3 à 15), avec un maximum de définition de 320 points sur 192 en deux couleurs. Bien entendu, plus la définition augmente, plus la mémoire disponible diminue (8138 octets occupés par les modes graphiques les plus poussés). Les ordres de dessin se limitent au simple point (PLOT) et au trait (DRAWTO), la couleur étant sélectionnée parmi cinq, à l'aide de l'ordre COLOR. Etrangement, le manuel passe sous silence le mode 11 qui permet les dessins avec 16 couleurs différentes sans possibilité de teinte, et avec une résolution de 80×192.

Les sons ne sont pas en reste et quatre générateurs indépendants sont programmables sous Basic. L'instruction SOUND permet de déterminer, pour chaque voix, la fréquence du son, le niveau sonore et un degré de distorsion. Il ne s'agit pas encore d'un vrai synthétiseur, mais il faut avouer que les effets obtenus sont plus que flatteurs.

Si les ordres de dessins sont succincts, il est possible de faire des remplissages de périmètre par une méthode détournée considérant l'écran comme un périphérique (à l'aide de l'ordre XIO).

Fiche technique de l'Atari 800 XL

Constructeur : *Atari*
Prix public : *2500 FF ttc*
Mémoire vive : *64 Koctets*
Mémoire morte : *24 Koctets*
Langage : *Basic Atari*
Variables : *neuf chiffres significatifs pour les variables numériques ; longueur des variables alphanumériques sans autre limitation que la place mémoire*
Modes : *deux modes texte ; treize modes graphiques de 320×192 points en deux couleurs, à 80×192 points en seize couleurs.*

Dans le même ordre d'idées, il est possible de saisir un caractère « à la volée ». Il faut ouvrir le périphérique clavier par un ordre OPEN # 1,4,0, "K:" suivi d'un GET # 1,A. Si l'analyse de la valeur de la variable A renseigne sur la touche pressée, Atari n'a pas choisi la voie de la normalisation. Il utilise une codification particulière, ignorant ASCII : 1 est codé 31, 2=30, 3=26, 4=24, 5=29, etc.

Il est possible de saisir un caractère à l'écran ou de connaître la couleur d'un point à l'aide de la fonction LOCATE. Mais attention en l'utilisant : le risque de détruire ce caractère (ou ce point) est grand, et il faut absolument le remettre par un PRINT (ou un PLOT).

De charmants parasites

Une vilaine « bogue » a été rencontrée par hasard : sous certaines conditions, l'ordre LOCATE perturbe le contenu de la dernière chaîne de caractères manipulée par le Basic. Pour éviter ce problème, il faut utiliser une variable de sauvegarde S$, en faisant S$=A$; puis, exécuter LOCATE, et enfin, rétablir A$ par A$=S$.

Quant au curseur baladeur qui perturbe l'écran, un simple 'POKE 752,1' se charge de l'éliminer. Dommage que le Basic ne possède pas d'ordre spécifique pour cela, pas plus qu'il n'y a de CLS pour effacer l'écran (en fait, il faut faire un PRINT CHR$(125), bien compliqué).

Mentionnons également le *scroll* inverse : en imprimant un caractère en colonne 39, on provoque un décalage vers le bas de toutes les lignes situées en dessous. Ce phénomène, éventuellement intéressant pour certains jeux, est tout à fait parasite en usage normal.

Il faut louer Atari d'avoir fait des choix judicieux pour l'utilisateur néophyte. En dépit de quelques « originalités » parfois agaçantes, il est plus simple d'apprendre le Basic Atari que les versions (plus complexes) de Microsoft. Si certaines possibilités sont cachées, elles ne sont pas forcément utiles au débutant mais il est regrettable que le manuel livré avec l'appareil ne les révèle pas. L'apprenti programmeur prendra énormément de plaisir à manipuler les ordres simples dévoilés au grand jour, tandis que l'amateur averti appréciera la puissance cachée sous le capot.

Alain LAVENIR

LA CYCLOÏDE : PIÈCE QUI ROULE...

LA représentation d'un objet en mouvement, par un dessin ou par la pensée, est moins évidente qu'on ne le pense. On s'en fait souvent une idée fausse qui nous mène, tout naturellement, à un paradoxe. Pour s'en sortir, il reste l'expérience. Les travaux pratiques deviennent obligatoires.

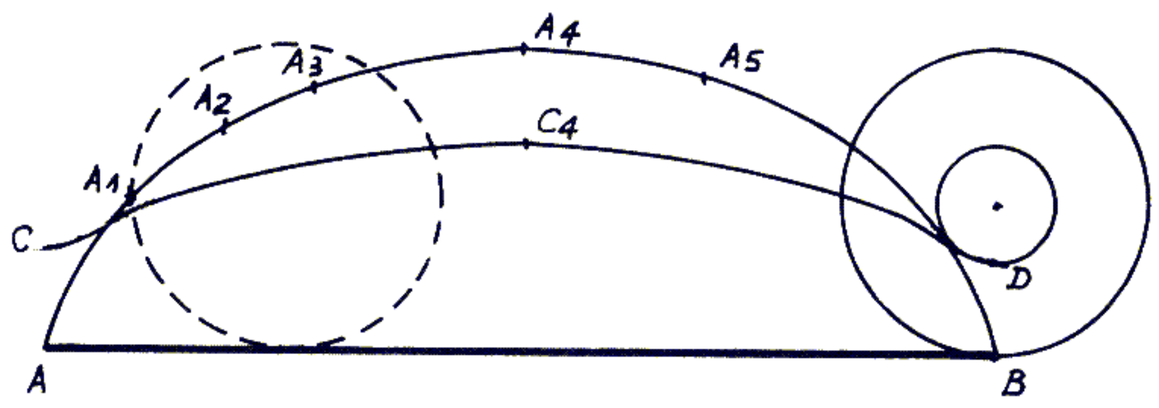
Essayons d'imaginer une pièce de monnaie, ronde, qui roule sans glisser sur une table. Ça semble facile.

Intéressons-nous alors à un point de cette pièce, le point A qui, au départ, est en contact avec la table. Lorsque la pièce a fait un tour complet, le point A se retrouve en B (figure 1).

La distance de A à B est donc égale à la circonférence de la pièce.

Si l'on s'intéresse à un autre point de cette pièce, le point C, on peut observer qu'il se retrouvera en D, après un tour complet. Mais, le cercle de rayon OC (O étant le centre de la pièce), plus petit que le cercle de rayon OA, a effectué un tour complet lui aussi.

Figure 2 : A doit aller plus vite que C, son trajet est plus long

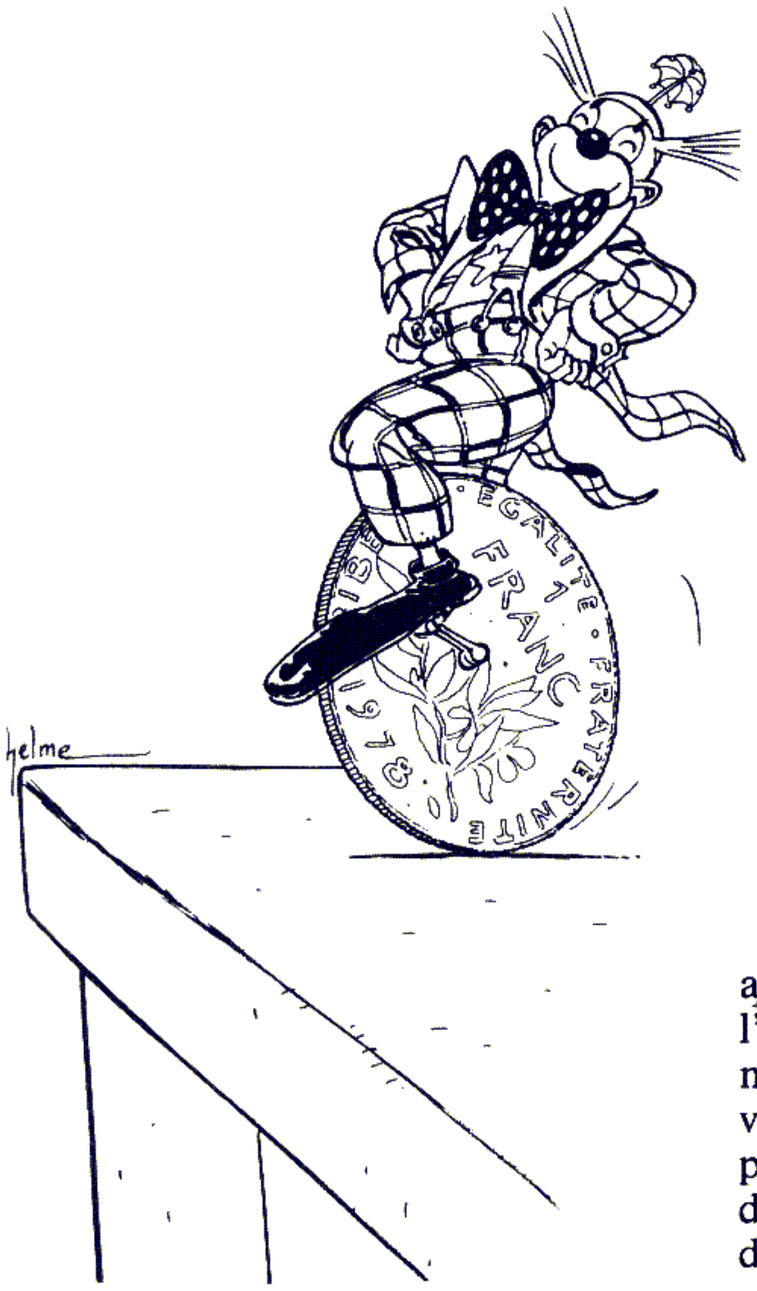


Comme la distance de A à B, apparemment égale à la circonférence de la pièce, est égale à la distance de C à D, elle-même semblant égale à la circonférence du petit cercle, on en arrive à une conclusion intéressante : le périmètre du cercle de rayon OA est le même que celui du cercle de rayon OC ! Pour comprendre où est l' « erreur » (car il semble bien qu'il y en ait une !), tentons des travaux pratiques : ils permettent de mieux voir le mouvement. Jusque-là, ce mouvement n'a été qu'imaginé et l'on s'est peut-être trompé.

Figure 1 : de A à B, la pièce a fait un tour complet

Reprenons notre pièce et traçons un petit point noir sur sa circonférence. Puis, faisons-la rouler sans glisser et observons l'évolution du point noir sur de petits intervalles (figure 2).

Le mouvement nous apparaît concrètement. Dans un premier temps, le point A arrive en A_1. Puis, toujours en montant, il parcourt un morceau de courbe jusqu'à A_2, etc. Il arrive alors en A_4, diamétralement opposé à sa position d'origine, puis passe en A_5 en descendant. Enfin, il atteint B. Le point A a donc voyagé, il n'a pas effectué un simple « tour de pièce »,



il a parcouru un trajet plus long : ce trajet s'appelle une cycloïde.

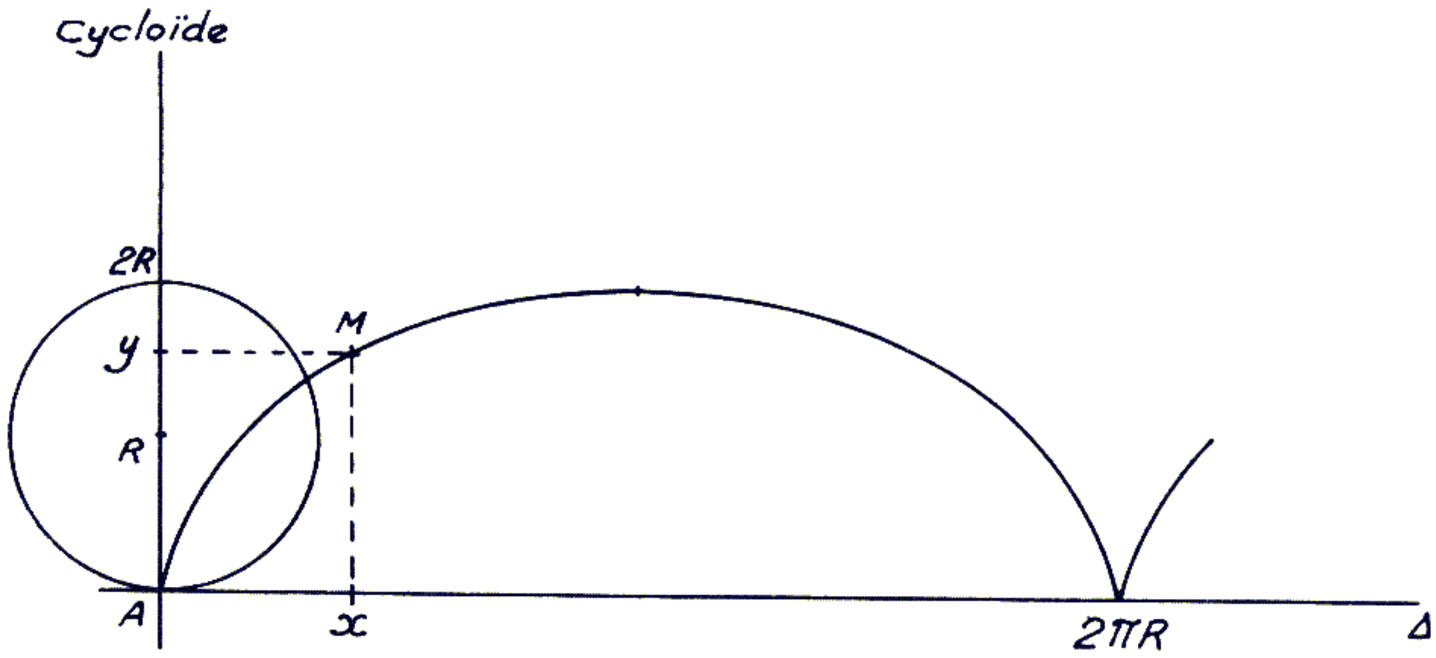
Par définition, une cycloïde est la courbe engendrée par le point d'un cercle qui roule sans glisser sur une droite fixe.

Lorsque la pièce fait un tour complet, A se retrouve bien en B, et C en D. Mais A et C ont parcouru des trajets différents à des vitesses différentes selon l'instant. Ainsi, pendant que A passait de A à A_4, C passait de C à C_4. Donc, pendant ce temps-là, A allait plus vite que C. C'est là que l'imagination nous trompe : elle ne nous permet pas de représenter la vitesse ou le temps sur un dessin. Le paradoxe auquel on avait abouti était donc dû à une mauvaise visualisation du mouvement.

Résolu par les plus grands

Une autre représentation de ce mouvement qui allie les travaux pratiques à l'abstrait, est celle des équations paramétriques. Accompagnées d'une représentation graphique, ces équations permettent une bonne description du phénomène.

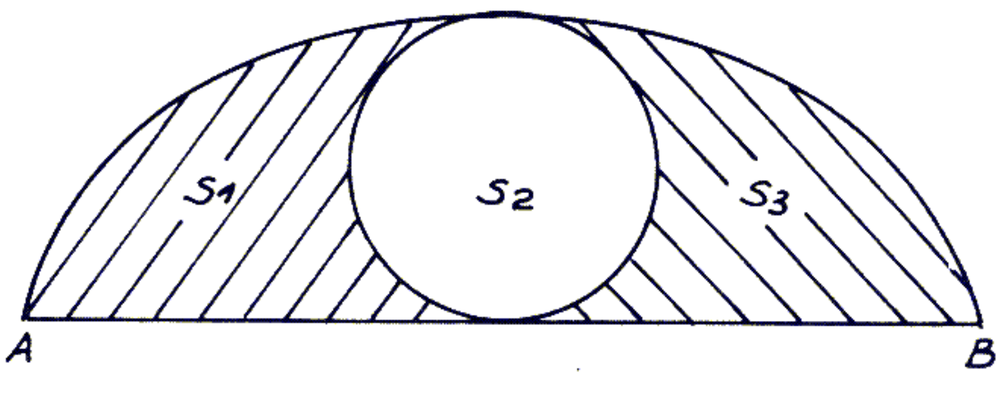
Figure 3 : un point M (x, y) de la cycloïde où x = R (t-sin t) et y = R (1-cos t)



Dans un repère tel que l'axe des abscisses est représenté par la droite fixe (Δ) et l'axe des ordonnées par la droite perpendiculaire à (Δ) et passant par A, chaque point M de la cycloïde aura, à chaque instant t, des coordonnées x et y définies par : x = R (t-sin t) et y = R (1-cos t) où R est le rayon de la pièce (figure 3). Les problèmes posés par la cycloïde ont été résolus par les plus grands mathématiciens : Galilée, Descartes, Roberval, Pascal...

Un des résultats les plus intéressants est que l'aire comprise entre la courbe et la base est égale au triple de l'aire du cercle d'origine (figure 4). Quant à la longueur de la cycloïde, de A à B, elle est égale à huit fois le rayon du cercle.

Figure 4 : les aires S_1, S_2 et S_3 sont égales



Les amateurs de programmation et de tracés de courbes pourront vivre les travaux pratiques d'une autre façon : les équations d'une cycloïde peuvent évidemment être programmées. Et les résultats obtenus alors seront plus exacts que ceux obtenus par l'imagination...

Anne-Sophie DREYFUS

En route vers Villapuce

Grâce aux bienfaits de la fée « cyclo », vous êtes transformé en puce et vous voyagez, selon votre choix, sur le bord extérieur ou au centre d'une roue de vélo. Pour vous aider à choisir, ce petit programme vous permet de comparer les distances parcourues selon votre position sur la roue.

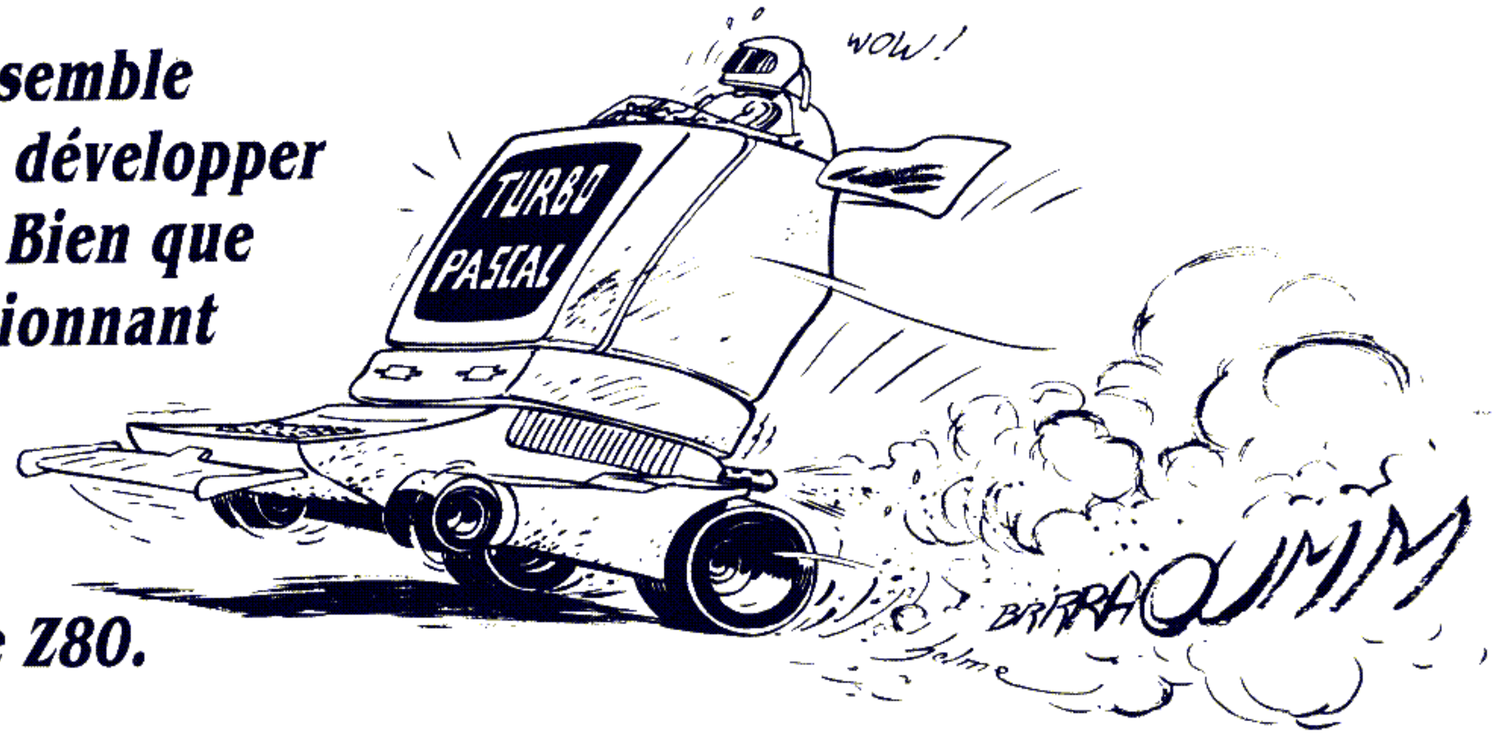
Ainsi, partant de Paris, vous désirez vous rendre à Villapuce. La distance officielle est de 421 km et le rayon de la roue sur laquelle vous voyagez est de 1 m (par exemple !).

L'entrée de ces données permet à votre machine de vous annoncer que, placé au centre, vous parcourrez bien 421 km mais, installé sur l'extérieur de la roue, vous parcourrez... 536 km. L'écart est donc de 115 km. Une puce avertie...

```
10: INPUT "DISTA
    NCE EN KM ?"
    ;A
20: INPUT "RAYON
     ROUE EN M ?
    ";B
30: P=2*π*B
40: N=E3*A/P
50: E=8*B*N*E-3:
    E=INT E
60: PRINT "PARCO
    URS CENTRE "
    ;A;" KM"
70: PRINT "...ET
     EXT. ROUE "
    ;E;" KM"
80: E=E-A:PRINT
    "ECART= ";E;
    " KM"
90: GOTO 10
```

TURBO PASCAL
UN COMPILATEUR PASCAL

TURBO PASCAL est un ensemble logiciel permettant de développer des programmes en Pascal. Bien que destiné aux machines fonctionnant sous CP/M ou MS/DOS, il donnera une bonne raison à ceux qui disposent d'un Apple II d'acheter une carte Z80.



Pour qui utilise de façon régulière un compilateur Pascal, les performances de *Turbo Pascal,* telles qu'elles sont décrites dans la publicité des revues américaines, paraissent à peine croyables.

Il a donc fallu l'essayer, bien entendu sans lire auparavant la documentation. Après avoir inséré la disquette et exécuté *Turbo Pascal,* on se trouve devant le menu général. Celui-ci représente bien la nature du système : il est simple, mais complet et performant. Il propose d'éditer, de compiler, d'exécuter, de sauver un programme, ou de modifier les options du compilateur.

Des performances à faire pâlir

Un seul appui sur la touche E, et on se retrouve alors immédiatement dans l'éditeur. En effet, comme la totalité du système se trouve dans les 64 Ko de la mémoire, les temps de réponse sont extrêmement rapides.

Il ne reste plus alors qu'à apprendre les différentes fonctions qui permettent de se positionner sur le caractère, le mot ou la ligne suivante, d'effacer un caractère, un mot ou une ligne, de déplacer un paragraphe, de rechercher une variable ou de remplacer son nom par un autre dans tout le texte.

Certaines commandes rendent l'écriture des programmes plus aisée. C'est le cas par exemple de l'« indentation » des lignes, c'est-à-dire leur position par rapport à la gauche de l'écran.

Cet éditeur, avec au total ses trente-huit commandes disponibles, est complet. Il est également agréable à utiliser.

En effet, toutes les commandes provoquent une interruption au niveau de l'affichage de l'écran. C'est ainsi que lorsqu'une commande est entrée, il n'est pas nécessaire d'attendre que l'écran soit complètement réaffiché pour appeler la fonction suivante. L'affichage est alors interrompu, et la commande est exécutée. Cela permet de travailler rapidement, certains affichages intermédiaires n'étant pas effectués.

Enfin, si la largeur de l'écran est toujours de quatre-vingts caractères, il est possible de faire un déroulement (*scrolling*) horizontal du texte, qui peut alors être édité avec une largeur de cent vingt-six caractères.

L'appel du compilateur déclenche soit la traduction du programme présent en mémoire, soit celle du fichier de travail. C'est sans doute ici que *Turbo Pascal* est le plus impressionnant, puisqu'un programme d'une centaine de lignes est compilé en moins d'une seconde. Cela est dû au fait que tous les traitements sont effectués en mémoire. Des performances que peuvent envier de nombreux ordinateurs.

Bien entendu, pour des programmes un peu longs, et si l'on ne dispose que d'une mémoire restreinte (de l'ordre de 64 Ko), il devient nécessaire de modifier les options du compilateur, afin que le texte et le code ne soient pas dans leur intégralité en mémoire. Dans ce cas, le programme et le code sont lus ou écrits sur le disque par petits segments.

Le manuel fourni avec *Turbo Pascal* est assez bien fait, et, sur ses 266 pages, plus de 100 sont consacrées à ce langage. Dans sa version française, comme dans sa version anglaise, il pourra constituer une bonne introduction. Le Pascal est conforme à celui défini par Kathleen Jensen et Niklaus Wirth, dans le sens où toutes les possibilités du langage sont présentes, malgré la petite taille du compilateur. Il se distingue toutefois par des différences et des extensions.

Au niveau des différences entre *Turbo Pascal* et le Pascal standard, il faut noter la question des variables dynamiques et les entrées/sorties sur les fichiers structurés.

Pour les variables dynamiques, c'est-à-dire celles qui sont créées au

cours de l'exécution d'un programme, la procédure DISPOSE du Pascal standard n'est pas fournie, mais est remplacée par les routines MARK et RELEASE. Cette différence n'est pas très gênante, puisqu'elle est analogue à celle du Pascal UCSD qui est assez répandu.

La lecture et l'écriture d'informations structurées, qui se font en Pascal standard par les procédures GET et PUT s'effectuent ici de la même façon que pour les fichiers de textes, c'est-à-dire par READ et WRITE. Cette option est bonne dans le sens où elle rend le langage plus homogène, mais présente un gros inconvénient, puisque les programmes écrits en *Turbo Pascal* doivent être modifiés pour être traduits par un autre compilateur.

Outre ces différences, de nombreuses extensions, très puissantes, sont apportées au langage. Celles-ci sont moins gênantes que les différences par rapport au langage standard, puisqu'il est toujours possible de ne pas les utiliser si l'on désire programmer en Pascal « pur ». Certaines de ces extensions ont un intérêt qui n'est pas évident. Par exemple, il est possible d'utiliser les opérations logiques sur des entiers, celles-ci étant réalisées sur chacun des bits des nombres. L'opérateur NOT, appliqué à un entier, inverse par exemple tous ses bits : NOT 0 est égal à -1. De même, 12 AND 22 est égal à 4, puisque ces deux valeurs ont toutes deux le bit 2 (qui représente 2 ^ 2 = 4) positionné à un.

Même si ce genre d'extension peut de temps à autre être utile, de nombreuses erreurs de programmation risquent de passer ainsi inaperçues au compilateur. De plus, leur utilisation facilite la programmation « à la façon Basic », puisque l'on peut écrire des formules telles que a : = (b=c) + (2 OR (b < c) qui ne brillent pas spécialement par leur clarté.

D'autres extensions sont heureusement plus utiles. *Turbo Pascal* permet de déclarer des variables de type chaîne de caractères, comme tous les bons compilateurs. Les valeurs hexadécimales et les caractères de contrôle peuvent être également utilisés dans les programmes.

Mais surtout, il est possible de définir des constantes structurées. Ainsi, nous pouvons par exemple déclarer :

```
type  etat_imprimante          =  (eteinte, hors_ligne, libre, occupee) ;
      message_imp              =  array [etat_imprimante] of string [41] ;
const msg_imp : message_imp    =  ('Veuillez mettre l''imprimante sous tension',
                                   'Veuillez mettre l''imprimante en ligne',
                                   'L''imprimante est disponible',
                                   'L''imprimante est en cours d''utilisation') ;
```

Il devient alors possible, dans un programme, d'utiliser le tableau MSG_IMP qui représente une constante mémorisant les différents messages. De la même façon, on peut définir des ensembles ou des articles constants.

Une autre amélioration apportée par *Turbo Pascal* se situe dans l'instruction CASE ... OF qui peut recevoir, pour chacun des cas possibles, un intervalle de valeurs. Le texte suivant est par exemple accepté par le compilateur :

```
case annee of
   min..1983 : begin
                  (* instruction *)
               end ;
   1984      : (* instruction * ) ;
   1985..1999: begin
                  (* instruction *)
               end ;
else
   write ('Erreur : annee incorrecte.')
end ;
```

En ce qui concerne les types définis par énumération de valeurs, comme : *type* hemisphere = (nord,sud,est, ouest) ; la fonction ORD définie en Pascal standard permet d'obtenir la représentation numérique d'une telle valeur. Ainsi, ord(ouest) est égal à 3. *Turbo Pascal* dispose de la fonction inverse, qui est absente du Pascal standard. Cette fonction a pour identificateur le nom du type associé : hemisphere(3) est ainsi égal à ouest.

Le manuel en anglais du Turbo Pascal

Le logiciel en quelques lignes

Nom : *Turbo Pascal*
Ordinateur : *testé ici sur Apple II, Turbo Pascal s'adapte aux systèmes MS/DOS, PC/DOS, CP/M*
Forme : *disquette*
Edité par : *Borland International*
Distribué par : *Fraciel (Tours)*
Prix public : *750 FF avec manuel en français ; on peut le trouver à 590 FF chez Mnémodyne (Paris) avec un manuel en anglais*
Principale orientation : *compilateur Pascal*
Autres orientations : *édition, sauvegarde des programmes, modification des options du compilateur*

Se tromper un vrai plaisir

Pour donner un exemple concret de ce qu'il est possible de faire avec *Turbo Pascal*, le système est accompagné d'une seconde disquette sur laquelle est enregistré un petit tableur, appelé *MicroCalc*. Celui-ci est fourni dans sa version source, c'est-à-dire qu'il peut être listé et modifié. Il donne un bon exemple de ce qu'il est possible de faire, en un peu plus de 1000 lignes Pascal, tout en étant parfaitement utilisable.

Les erreurs de syntaxe ou d'exécution deviennent presque un plaisir avec *Turbo Pascal* : elles sont signalées en clair par le compilateur, sans qu'il soit nécessaire de se reporter à une table de codes d'erreur. De plus, le curseur est automatiquement positionné dans l'éditeur à l'endroit où se situe l'erreur de syntaxe.

En ce qui concerne les erreurs d'exécution, le système se comporte de la même façon, c'est-à-dire que le curseur est positionné à la ligne où l'erreur s'est produite. Il existe très peu de systèmes qui disposent de cette facilité de mise au point. Avec la rapidité de compilation, on a vraiment l'impression de travailler, pour des programmes de quelques centaines de lignes, avec un interpréteur Pascal.

Voilà de quoi convaincre ceux qui portent un petit intérêt à Pascal de ne plus travailler seulement en Basic.

Thierry CHAMORET

DRAGBUG
MONITEUR-DÉSASSEMBLEUR POUR DRAGON 32

LE logiciel Dragbug réunit, sur une même cassette, un moniteur et un désassembleur pour Dragon 32. Des bases de programmation en langage-machine sont indispensables afin de mieux apprécier les qualités de ce logiciel. Il coûte environ 170 FF ttc.

Dragbug, ce diminutif évocateur suggère une concision de bon aloi dans la présentation du logiciel. Cette simplicité extrême rassure : le mode d'emploi est tout entier compris dans un petit dépliant de référence facile à mémoriser. Pas de mystère, après une lecture rapide, on est prêt pour le départ.

Bien évidemment, il est nécessaire d'apprendre les bases de la programmation en langage-machine avant d'utiliser *Dragbug*. Un manuel d'initiation au processeur 6809 ou même au 6800 conviendra parfaitement, et la perspective de concevoir des programmes cent fois plus rapides et occupant moins de place en mémoire, est une motivation excellente.

Cela dit, le moniteur de *Dragbug* sera une aide précieuse pour vos premiers essais. Il propose une dizaine de commandes qui dédramatisent le langage-machine. On les appelle par leur initiale : P (comme PRINT) active ou déconnecte une imprimante éventuelle qui recopie l'écran (l'imprimante n'est pas indispensable, mais permet de gagner du temps pendant le « débuggage »). Q, pour QUITTER, retourne au Basic ; fort heureusement on peut repasser au moniteur instantanément grâce à une formule magique. Les autres commandes sont M pour visualiser huit adresses en Mémoire, ou E (Editer) pour inspecter une adresse à la fois et la modifier le cas échéant.

Au standard Motorola

Toutes les entrées/sorties du système sont en hexadécimal, mais la clarté n'en est pas compromise. La commande « offset » (O) effectue le calcul automatique des sauts, ce qui permet le choix de sauts « longs » ou « courts ». On exécute un programme en tapant G suivi de l'adresse du point d'entrée ; on peut désormais traquer la bogue en insérant des points d'arrêt (breakpoints) au sein du programme. Quand il les rencontre, il s'interrompt et affiche le contenu de ses registres internes (A, B, X, Y, C et R). Ceci peut être obtenu d'ailleurs en fin de programme avec la commande R.

Pour parfaire la mise au point, C permet de recopier des blocs de mémoire dans différentes zones, ce qui remplace les fonctions « insert/delete » qui font défaut.

Mais *Dragbug* est aussi un désassembleur au standard Motorola, ce qui est très important : à défaut d'avaler et de digérer des commandes Assembleur (tant pis...), il est capable de les reconnaître à partir d'une analyse sur le binaire. Cette facilité permet de déchiffrer à peu près toutes les routines existantes pour 6809, et de s'en inspirer largement.

Si *Dragbug* n'est pas le fantastique éditeur-assembleur-désassembleur que l'on pouvait espérer, il contient le nécessaire de programmation en langage-machine le plus indispensable, et le plus agréable à mettre en œuvre pour le débutant.

Eric BOUCHER

Le logiciel en quelques lignes

Nom : *Dragbug*
Ordinateur : *Dragon 32*
Forme : *cassette*
Edité par : *Personal Software Services*
Distribué par : *Innelec*
Prix public : *170 FF ttc*
Principales orientations : *moniteur, désassembleur*

ASSEMBLEUR
POUR TO7 ET TO7/70

MUNI de la cartouche Assembleur, le TO7 (comme le TO7/70) offre la possibilité de programmer en...Assembleur. Ce logiciel est écrit par Microsoft, édité par TO TEK et coûte 890 FF ttc.

Le logiciel *Assembleur* rend la programmation en langage-machine (voir encadré *Langage-machine et langage Assembleur)* accessible aux utilisateurs du TO7 ou du TO7/70. Bien conçu dans l'ensemble, il permet de tirer un bon parti des ressources de la machine. A condition d'avoir déjà quelques notions d'Assembleur (le langage). En effet, la notice d'emploi qui accompagne la cartouche est assez succincte. Elle supportera d'être complétée par un ouvrage sur le processeur du TO7, le 6809.

Ce « mode d'emploi » ne décrit que les commandes du logiciel et, sauf une table des codes ASCII, il ne comporte aucun tableau des mnémoniques. C'est un petit peu comme si un Basic était livré avec une notice indiquant les utilisations des touches de curseur, la signification de LOAD et de SAVE, et ne parlait ni des mots-clés du langage, ni de sa syntaxe !

Huit options au menu

Le menu principal du logiciel comporte huit options. Cinq concernent la gestion des fichiers :

- DIRECTORY permet de visualiser le catalogue de la disquette ;
- COPY recopie un fichier existant sous un autre nom ;
- RENAME renomme un fichier ;
- KILL détruit un fichier ;
- FORMAT « formate » une disquette.

Une sixième option sert au choix de la largeur de l'impression (40 ou 80 colonnes). Les sorties sur imprimante thermique, en 40 colonnes, consomment énormément de papier. De plus, les lignes dont la longueur dépasse 22 caractères (les lignes de commentaires, par exemple) sortent en deux fois, rendant peu claire la liste du programme.

Il reste deux options, les principales : l'éditeur-assembleur et le moniteur.

Puisque l'assembleur n'est autre qu'un programme traducteur, l'éditeur crée un programme-source et l'« édite » (au sens anglais du terme). C'est un programme d'aide à l'écriture qui comprend diverses possibilités de modification, d'insertion, d'écrasement, de copie, etc.

Une ligne Assembleur comporte quatre zones : étiquette, code opération (COP), opérande, commentaire. Ces zones sont de longueur fixe et l'on passe à la suivante par la barre d'espacement. L'espace s'obtient par CTRLS.

Simple à utiliser

Différentes touches (décrites dans le mode d'emploi) déplacent le curseur d'un caractère, d'une ligne ou d'une page ; vont directement au début ou à la fin du programme ; insèrent, suppriment ou copient un caractère ou une ligne. La commande Z définit une zone de programme dans laquelle on pourra effectuer des traitements particuliers :

Langage-machine et langage Assembleur

UN ordinateur n'est capable de comprendre (et donc d'exécuter directement) que des instructions en langage-machine, c'est à dire en code binaire. Par exemple, la suite des trois octets 10110110 00010010 00110100 (B6 12 34, en hexadécimal) signifie : mettre dans l'accumulateur (qui est un registre interne du microprocesseur) le contenu de l'octet d'adresse $1234, en hexadécimal.

Il est facile de se rendre compte que l'écriture directe d'un programme en langage-machine est une tâche pour le moins fastidieuse. Une simple addition de deux nombres entiers inférieurs à 256 (donc, tenant sur un octet) demande l'écriture (et la frappe) de 72 nombres binaires ! Pour remédier à cela, les constructeurs ont créé un langage dit symbolique, qui est appelé langage d'assemblage ou Assembleur. Dans ce langage, chaque code-opération est représenté par un code mnémonique. Ainsi, l'instruction précédente s'écrira : LDA $1234 (LDA = LoaD Accumulator). Un tel langage est dit "direct" car chaque instruction symbolique correspond à une et une seule instruction machine. L'utilisation de l'Assembleur nécessite évidemment un programme de traduction, qui est lui aussi appelé *assembleur* (dans ce cas, on l'écrira sans majuscule) ! Le programme original est le programme-source et le résultat qui est donc le code directement exécutable, est le programme-objet.

Chaque microprocesseur, ayant son propre jeu d'instructions machine, aura donc son assembleur. On parle ainsi de l'Assembleur du Z80, du 6502, du 6800, du 6809 (c'est le cas du TO7), etc.

Le logiciel en quelques lignes

Nom : *Assembleur*
Ordinateur : *TO7 et TO7/70*
Forme : *cartouche*
Edité par : *TO TEK*
Distribué par : *les revendeurs THOMSON*
Prix public : *890 FF ttc*
Principales orientations : *éditeur-assembleur et moniteur*
Autres orientations : *gestion de fichiers, option d'impression.*

par exemple, rechercher les occurences d'une chaîne, la remplacer par une autre, imprimer une partie du programme, etc.

L'assembleur est très simple à utiliser : il suffit de taper A. Bien sûr, quelques options viennent compliquer (et compléter) la tâche, permettant d'avoir ou non une sortie sur papier, de visualiser ou non la liste de la table des symboles (il manque une table des références croisées), de sortir le code-objet sur disquette, cassette ou en mémoire, etc.

La liste des fonctions d'assemblage est assez complète (INCLUD est très utile pour gérer une bibliothèque de routines).

Les amateurs apprécieront

Quant au moniteur, il est aussi intéressant que l'éditeur-assembleur. Il permet en effet d'examiner et de modifier le contenu de la mémoire sous différentes formes : en ASCII, en numérique (décimal ou hexadécimal), en mnémonique (le moniteur sert alors de désassembleur). Une commande affiche les registres et une autre transforme le TO7 en calculatrice. Les amateurs de DUMP hexadécimal apprécieront cette possibilité.

Certaines commandes de « débogage », pour la mise au point, sont disponibles. Et les commandes de contrôle sont identiques à celles de l'éditeur-assembleur (LOAD, SAVE, VERIFY, PRINT, etc.).

Assembleur est donc un logiciel très réussi dont le prix (890 FF ttc), bien que justifié par sa qualité, risque de décourager certains candidats.

Jacques LABIDURIE

CHOPIN ET GÉNÉCAR

CRÉATION MUSICALE ET GÉNÉRATEUR DE CARACTÈRES POUR LYNX

LES éditions Eyrolles s'intéressent aussi au Lynx (48 Ko ou 96 Ko). Dans la collection Logilivre, elles proposent une cassette composée de deux logiciels élaborés pour cet ordinateur, Chopin et Génécar. Le premier offre la possibilité de créer des mélodies, le second de générer des caractères.

Une même cassette renferme deux logiciels : une création musicale, *Chopin*, et un générateur de caractères, *Génécar*. Ces deux logiciels sont enregistrés pour les deux versions de Lynx, 48 Ko et 96 Ko, et sont extraits de l'ouvrage « Tout savoir sur Lynx ».

La qualité de la présentation et les instructions en français en rendent l'emploi facile. Un court programme d'introduction sert de menu et permet de vérifier rapidement le bon réglage du magnétophone.

Génécar est un utilitaire intéressant qui permet de définir facilement des caractères, de modifier ceux qui existent déjà, sans même avoir recours à une feuille de papier quadrillée. Les caractères sauvegardés sur cassette pourront être utilisés ultérieurement, ce qui épargnera au programmeur les fastidieuses corvées de DATA.

Le logiciel *Chopin* est plus attrayant : il transforme l'ordinateur en un petit orgue électronique, huit touches numériques donnant un accès direct aux notes. Plusieurs possibilités appréciables sont offertes par un menu : le choix de l'octave, la durée des notes, l'obtention des dièses, le tout au tempo désiré, et modifiable.

La qualité du son est assez bonne pour un appareil qui n'est pas doté d'un véritable synthétiseur.

Une portée musicale classique affichée à l'écran, contrôle la saisie des notes d'une mélodie. Les possesseurs du Lynx 48 Ko pourront mémoriser 50 notes au maximum, contre 500 avec le Lynx 96 Ko !

D'une utilisation simple et agréable, ce programme de création — et d'initiation — met la musique à la « portée » de tous.

Le logiciel en quelques lignes

Nom : *Création musicale et générateur de caractères (Chopin et Génécar)*
Ordinateur : *Lynx 48 Ko et 96 Ko*
Forme : *cassette*
Edité et distribué par : *Eyrolles*
Prix public : *120 FF ttc*
Principales orientations : *création de mélodies et génération de caractères*

On peut donc se féliciter de voir des logiciels de qualité pour le Lynx, et espérer que d'autres programmes aussi bons vont suivre.

Yvon PÉRÈS

MODULO-CI, MODULO-LA

Jean-Christophe KRUST

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 succéderont : des programmes toujours plus courts, plus rapides...
Et les records tomberont !

Savez-vous planter un chou ? Moi non plus. Et trouver avec une HP-41C le reste de la division entière de 8^{152} par 3 ? Là, on peut s'entendre...

Le défi de Misez p'tit, lancé dans *LIST* n° 2, a suscité de nombreuses lettres où la fonction MOD, qu'il s'agissait d'étendre aux très grands nombres, fut réellement mise "à toutes les sauces". L'article de Robert Pulluard (*LIST* n° 3) sur cette fonction y est certainement pour beaucoup.

Si le programme-défi lancé fut, souvent largement, relevé... l'auteur en a considérablement accru les performances et ses routines de calcul sont les plus économes et les plus rapides.

Ici, de toute évidence, il n'y a pas une, mais plusieurs solutions : soit on recherche la vitesse d'exécution, et l'on doit alors sacrifier des octets de la mémoire (plus d'instructions et de registres employés), soit l'économie est le souci premier et... la vitesse est vraiment dépassée.

Le texte du défi tient en quelques mots : considérons un grand nombre, 5^{125} par exemple. Quel est le reste de sa division entière par 3 ? C'est 2. On le vérifiera aisément en tapant : 5 ENTER 125 Y^x 3 MOD.

Mais, s'agissant de très grands nombres, comme 8^{152}, il est très (!) difficile de procéder ainsi pour obtenir 8^{152} modulo 3... Il faut réaliser un programme d'extension de la fonction MOD aux très grands nombres. Les différents paramètres sont introduits au départ dans la pile opérationnelle (8, 152 et 3 dans l'exemple) et le résultat s'y trouve à l'arrivée (ici 1).

Las, laissons donc l'auteur du défi et, ô miracle, du programme gagnant en exposer la philosophie et les subtilités logicielles.

A la Mod de chez nous

Nous voici donc en présence d'un thème illustrant la maxime préférée des optimiseurs : *"le mieux peut-il être l'ami du bien ?"*.

En effet, on peut tout d'abord imaginer la routine suivante de 13 octets (sans compter ni le LBL de tête, ni le END final) :

```
LBL^T MODULO1
SIGN
LBL 00
RCL Y
ST*Y
X<>L
MOD
DSE Z
GTO 00
END
```

L'entrée des données s'effectue simplement : pour calculer a^b modulo c,

taper b ENTER a ENTER c, et le résultat se trouve en X en fin d'exécution.

Le défaut du programme réside dans sa (relative) lenteur d'exécution :
23^9 modulo 18 = 17 en 2 secondes,
5^{125} modulo 3 = 2 en 25 secondes et
8^{152} modulo 3 = 1 en 31 secondes.

Et pourtant, nous voyons qu'un programme plus sophistiqué apporterait plus rapidement une solution car :
$5 \equiv 2 \pmod 3$,
$5^2 \equiv 1 \pmod 3$,
$5^3 \equiv 2 \pmod 3$,
$5^4 \equiv 1 \pmod 3$, etc.

Cela signifie que 5 modulo 3 = 2 puis 5^2 modulo 3 = 1, etc. Tous les 5^n où n est impair ont pour reste 2 (dans la division par 3) tandis que lorsque n est pair le reste est 1. Immédiatement, on peut déduire que $5^{1234567890}$ modulo 3 = 1 car l'exposant de 5 est pair. Il apparaît donc qu'*il existe une période des congruences* ! Trivial, mais fondamental.

Autre exemple, avec 17^n modulo 15 :
$17 \equiv 2 \pmod{15}$,
$17^2 \equiv 4 \pmod{15}$,
$17^3 \equiv 8 \pmod{15}$,
$17^4 \equiv 1 \pmod{15}$,
$17^5 \equiv 2 \pmod{15}$, etc.

Ici, la suite des restes successifs est 2, 4, 8, 1. On voit immédiatement que 17^{417} modulo 15 vaut 2 car 417 est impair (c'est soit 2, soit 8) et vaut 104*4 (période des restes) + 1.

Modulo
Programme pour HP-41C
Auteur Gilles Bransbourg
Copyright *LIST* et l'auteur

```
01♦LBL "MODUL02"
02 X<>Y
03 2
04 -
05 1 E3
06 /
07 STO T
08 RDN
09 MOD
10 STO 00
11♦LBL 00
12 LASTX
13 STO T
14 RDN
15 RCL 00
16 *
17 R↑
18 MOD
19 ISG Y
20 GTO 00
21 RTN
22♦LBL 00
23 STO IND Y
24 RDN
25 ENTER↑
26 INT
27 1
28 -
29 1 E3
30 /
31 X<>Y
32 R↑
33 STO L
34 RDN
35 RCL IND X
36♦LBL 01
37 RCL IND Z
38 X=Y?
39 GTO 02
40 RDN
41 ISG Z
42 GTO 01
43 GTO 00
44♦LBL 02
45 *
46 RDN
47 -
48 CHS
49 LASTX
50 FRC
51 1 E3
52 *
53 1
54 +
55 X<>Y
56 INT
57 MOD
58 +
59 RCL IND X
60 END
```

Cependant, un problème se pose : quand débute la période ? Par exemple, avec 46^n modulo 28,
$46 \equiv 18 \pmod{28}$,
$46^2 \equiv 16 \pmod{28}$,
$46^3 \equiv 8 \pmod{28}$,
$46^4 \equiv 4 \pmod{28}$,
$46^5 \equiv 16 \pmod{28}$, etc.

La suite est 16, 8, 4 mais elle débute à la puissance 2 et non plus 1 comme précédemment.

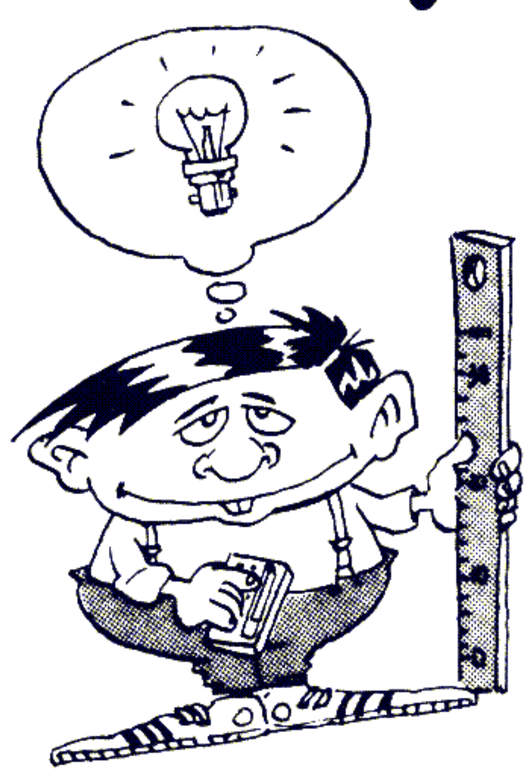
Une seule méthode nous permet de déterminer le début de la période et d'en connaître les valeurs : stocker tous les résultats intermédiaires et les comparer à la dernière puissance calculée : cela occupe un nombre... indéterminé, a priori, de registres de mémoire.

Nous obtenons ainsi le programme MODULO2, reproduit ci-contre, énorme (relativement) consommateur d'octets puisqu'il en compte 74, sans préjuger du nombre de registres employés mais les temps d'exécution sont spectaculaires pour les grands nombres :
23^9 modulo 18 = 17 en 8 secondes,
5^{125} modulo 3 = 2 en 3 secondes,
8^{152} modulo 3 = 1 en 2,5 secondes,
et $5^{1234567890}$ mod 3 = 1 en 1 seconde.

Cette fois-ci (foin des économies !) l'ordre d'introduction des données est "naturellement" a, b, c.

Gilles BRANSBOURG

QUI DIT MIEUX ?



OSERAIS-je affronter les spécialistes de cette Polonaise perverse inversée dont l'ingéniosité me laisse, parfois, pantois ? Oui, en lançant un défi non exprès taillé pour les registres de la dame 41C...

A vos méninges d'abord ! Trouvez le bon algorithme, la solution du problème avant de songer à programmer !

Prenez un triangle, quelconque. Connaissant les coordonnées X, Y de chacun de ses trois sommets, calculez les coordonnées du centre de gravité du triangle. Facile, non ?

Imaginez une aiguille pointée vers le ciel, un triangle de carton et l'exercice suivant : posez, en équilibre, le triangle sur l'aiguille. Un seul point du triangle correspond à cet équilibre : le centre de gravité.

Mon programme occupe 17 octets, aucune mémoire et travaille... vite. Faites mieux, mais surtout, pensez à trouver le bon algorithme.

Pierre LISON

LE BASIC DU COCO 2

Sous un boîtier tout blanc, le Basic du Colour Computer 2, surnommé Coco 2, est riche en fonctions graphiques qui doivent malheureusement rester indépendantes du texte. Et les couleurs sont bien là, mais seulement quatre à la fois.

Le nouveau Tandy Colour Computer 2, surnommé Coco 2, a changé de boîtier par rapport à son prédécesseur, Coco 1 : il est blanc, et comporte un vrai clavier AZERTY.

D'entrée de jeu, on regrette l'emplacement de la touche d'effacement d'écran (CLEAR), située juste à côté de la touche ENTER. Cela peut entraîner quelques crises de nerfs... D'après le manuel d'utilisation, SHIFT 0 fait passer en mode minuscule : hélas, trois fois hélas, on n'obtient toujours que des majuscules, mais en vidéo inversée ! Les minuscules ne semblent donc pas exister sur le Coco 2. Pour en finir, provisoirement, avec le chapitre des regrets, on peut déplorer l'absence de répétition automatique des touches.

A la mise sous tension, un agréable affichage noir sur fond vert nous informe que le Basic est un Tandy sous licence Microsoft. Une partie des ordres Basic sera donc probablement très proche du standard Microsoft.

L'écran comporte seulement 16 lignes de 32 caractères. Et Tandy reste fidèle à sa tradition : pas d'éditeur d'écran, d'où l'obligation de se plier à la gymnastique acrobatique de l'ordre EDIT. La mise au point des programmes est toutefois facilitée par la présence de RENUM, DEL, TRON et TROFF. Lors d'un LIST, le déroulement des lignes peut être momentanément suspendu par l'appui sur SHIFT @, puis repris par l'appui sur une touche quelconque.

Les noms de variables ne devront pas excéder un ou deux caractères : une lettre, une lettre suivie d'un chiffre, ou deux lettres. Ce Basic fait encore partie de la vieille génération ! Les réels sont stockés sur cinq octets : la mantisse affichée a 9 chiffres, alors que l'exposant doit être compris entre −38 et +38. On regrette ici l'absence d'entiers, qui permettent pourtant d'accélérer nettement certains traitements, et de réels double précision pour les applications scientifiques.

Musical et ludique

Les chaînes alphanumériques sont stockées dans une zone spéciale. A la mise sous tension, 200 octets sont réservés pour les chaînes et on peut à volonté augmenter ou diminuer la taille de cette zone par CLEAR n qui réserve n octets ; cela au détriment de la mémoire programme et des variables réelles.

Les tableaux peuvent être multidimensionnés. Quant aux nombres, il est possible de les écrire en décimal, bien sûr, mais aussi en octal grâce à & ou &O, et en hexadécimal grâce à &H. A noter l'intéressante fonction HEX$ qui effectue la conversion de décimal en hexadécimal.

Les instructions du Basic classique sont là : IF-THEN-ELSE, ON-GOTO, ON-GOSUB, READ-DATA-RESTORE, DEF FN, PRINT USING. On déplorera toutefois l'absence de gestion des erreurs.

Les fonctions scientifiques, et les manipulations de chaînes de caractères sont complètes. On retrouve pour ces dernières les sympathiques STRING$ et INSTR: répétition et recherche de chaînes sont donc facilitées. En outre, MID$ peut se situer à gauche du signe « = » pour modifier commodément le contenu des chaînes.

La variable TIMER joue un rôle d'horloge et on peut l'interroger ou la modifier : elle varie de 0 à 65535 en « environ 18 minutes » (selon le manuel), puis retourne à 0. Elle est donc incrémentée en « environ 1,6 centième de seconde » : on aurait préféré une autre correspondance avec les secondes...

L'instruction SOUND *h*, *d* permet de jouer de la musique : *h* représente la hauteur du son et *d*, la durée. L'instruction PLAY est plus commode : elle permet de jouer des notes selon la notation anglo-saxonne. C'est ainsi que la gamme complète se joue avec la commande PLAY "O2CDEFGABO3C" où O effectue un changement d'octave parmi les cinq admises, et où C, D, E,... représentent respectivement do, ré, mi,...

La durée, le tempo ou le volume sont modifiables par un L, un T ou un V. Il est également possible de rappeler des sous-chaînes par X. Ainsi A$ = "CDEFG" : PLAY" XA$; XA$;" jouera deux fois de suite « do, ré, mi, fa, sol ». Le dièse ou le bémol s'obtiennent en faisant suivre la note par les signes « # » ou « — ».

Le branchement de deux manettes de jeu a été prévu, et c'est la fonction JOYSTK qui permet de connaître leur position.

Le Coco 2 se révèle ainsi bien adapté aux jeux : grâce à PLAY et à JOYSTK, mais aussi grâce à ses possibilités graphiques couleurs. En effet, ce Tandy dispose de huit couleurs, plus le noir.

On pourra obtenir des graphiques grossiers sur la page de texte avec SET (x, y, c) qui allume le point de coordonnées x, y avec la couleur c. Les points ainsi allumés sont assez gros : ils ont la taille d'un quart de caractère. Mais leur définition n'est alors que de 64×32. En outre, les quatre points d'un même caractère ne peuvent être allumés qu'avec une même couleur. Un point est éteint grâce à RESET (x, y).

La grande force du Coco 2

La faible définition, et les deux seuls ordres SET et RESET, rendront le dessin sur la page de texte assez difficile. On préférera les vraies pages graphiques où la définition varie de 128×96 à 256×192, et où les ordres de tracé de lignes et de cercles existent.

Ce qui fait la grande originalité, et la grande force, du Coco 2 est la possibilité qu'il offre de disposer de plusieurs pages graphiques affichables séparément.

La mémoire vive de cette version du Tandy possède 32 Koctets, dont un Ko est réservé par le système et 0,5 Ko par la mémoire écran de texte : il reste donc 30,5 Ko à l'utilisateur.

L'ordre PCLEAR n réserve n pages graphiques de 1,5 Ko : une page au minimum et huit au maximum. La mémoire réellement disponible à l'utilisateur pour son programme et ses variables peut donc varier de 29 Ko avec PCLEAR 1 à 18,5 Ko avec PCLEAR 8. A la mise sous tension, le Coco 2 se réserve quatre pages, et un PRINT MEM confirme que l'on dispose d'environ 24,5 Ko.

Cinq modes graphiques sont disponibles (voir tableau des Modes graphiques du Coco 2) avec l'instruction PMODE *m*, *p* où *m* représente le mode et *p* la page affichée. Par exemple, en mode 4 (la plus haute résolution disponible), après avoir fait PCLEAR 8 et rempli les deux dessins des pages 1 à 4 et 5 à 8, on pourra les faire afficher alternativement et rapidement avec une ligne de programme telle que : 100 PMODE 4,1 : PMODE 4,5 : GOTO 100. On pourra ainsi concevoir des dessins animés, le passage à l'écran d'un graphique à l'autre étant immédiat. Et avec PCOPY p_1

Liste des mots-clés du Basic du Coco 2

Ordres sur 1 octet

	8	9	A	B	C	D - E - F
0	FOR	RETURN	SOUND	AND	PCLEAR	
1	GO	STOP	AUDIO	OR	COLOR	
2	REM	POKE	EXEC	>	CIRCLE	
3		CONT	SKIPF	=	PAINT	
4	ELSE	LIST	TAB(	<	GET	
5	IF	CLEAR	TO	DEL	PUT	
6	DATA	NEW	SUB	EDIT	DRAW	
7	PRINT	CLOAD	THEN	TRON	PCOPY	
8	ON	CSAVE	NOT	TROFF	PMODE	
9	INPUT	OPEN	STEP	DEF	PLAY	
A	END	CLOSE	OFF	LET	DLOAD	
B	NEXT	LLIST	+	LINE	RENUM	
C	DIM	SET	—	PCLS	FN	
D	READ	RESET	*	PSET	USING	
E	RUN	CLS	/	PRESET		
F	RESTORE	MOTOR	↑	SCREEN		

Fonctions sur 2 octets (le premier octet étant toujours FF)

	8	9	A	B à F
0	SGN	MID$	PPOINT	
1	INT	POINT	STRING$	
2	ABS	INKEY$		
3	USR	MEM		
4	RND	ATN		
5	SIN	COS		
6	PEEK	TAN		
7	LEN	EXP		
8	STR$	FIX		
9	VAL	LOG		
A	ASC	POS		
B	CHR$	SQR		
C	EOF	HEX$		
D	JOYSTK	VARPTR		
E	LEFT$	INSTR		
F	RIGHT$	TIMER		

Modes graphiques du Coco 2

PMODE	Définition	Nombre de couleurs disponibles	Nombre de pages-mémoire utilisées pour un dessin	Nombre de dessins maxi mémorisables (en PCLEAR 8)
0	128× 96	2	1	8
1	128× 96	4	2	4
2	128×192	2	2	4
3	128×192	4	4	2
4	256×192	2	4	2

TO p_2 qui copie la page p_1 dans la page p_2, l'outil devient parfait.

Il est impossible d'obtenir simultanément les huit couleurs à l'écran : elles ne sont disponibles que par deux ou par quatre. Mais on choisit la gamme des couleurs avec SCREEN *t*, *g* : *t* est le type de l'affichage (0 pour texte, 1 pour graphique), et *g* la gamme de couleurs (0 ou 1).

Par exemple, pour les modes 1 et 3, les quatre couleurs affichées sont, pour la gamme 0, vert - jaune - bleu - rouge, et pour la gamme 1, beige - turquoise - pourpre - orange.

Dans les modes 0, 2 et 4, les deux couleurs sont noir - vert (gamme 0) ou noir - beige (gamme 1) : en réalité, il n'y a qu'une seule couleur, un point étant allumé (vert ou beige selon la gamme), ou éteint (noir). En mode 256×192, le Coco 2 porte donc bien mal le nom de « Colour ».

Le choix entre *texte* et *graphique* imposé par l'ordre SCREEN dévoile même son gros défaut : l'impossibilité de mêler sur le même écran du texte et des dessins. Les instructions PRINT agissent uniquement dans la page *texte,* et les instructions PSET et LINE uniquement dans la page *graphique.*

Les ordres graphiques sont nombreux : PCLS c efface l'écran avec la couleur c ; COLOR *cf*, *ce* choisit la couleur du fond (*cf*) et celle de l'écriture (*ce*) ; PSET (x, y, c) allume un point et PRESET (x, y) l'efface ; PPOINT (x, y) retourne la couleur du point. Quel que soit le mode choisi, l'origine (point de coordonnées 0 et 0) est en haut à gauche de l'écran et les coordonnées du point diamétralement opposé valent toujours 251 et 191.

C'est ainsi que Tandy a résolu les problèmes de conversion : il n'est plus nécessaire de recalculer les coordonnées des points lorsqu'on change de mode.

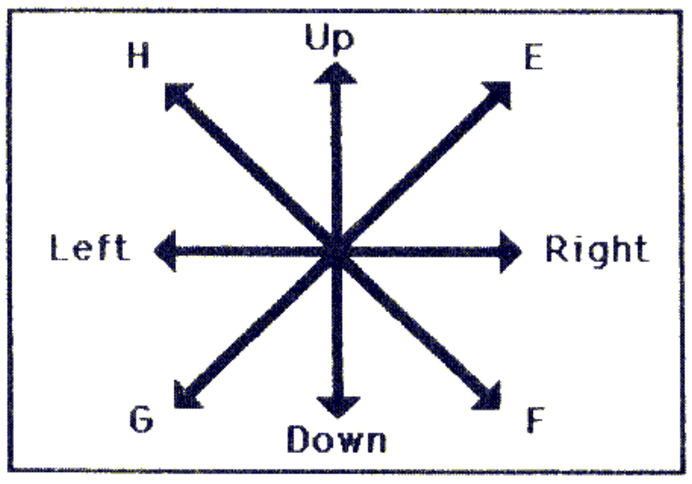
Les instructions LINE et CIRCLE offrent la possibilité de tracer des lignes, des rectangles, des rectangles pleins, des cercles, des arcs de cercles, des ellipses, des arcs d'ellipses. PAINT (x, y), c_1, c_2 peint en couleur c_1 une zone partant du point (x, y) jusqu'aux limites en couleur c_2.

Tandy a eu l'idée intéressante et originale de permettre les recopies partielles d'écran. Par exemple, le programme :

```
10 DIM T (40, 20)
20 GET (10, 10) — (50,30), T
30 PUT (100, 60) — (140, 80), T
```

recopie l'image contenue dans un rectangle de 40 points de large sur 20 points de haut dans une autre zone de l'écran. Ceci accélère nettement les tracés à caractère répétitif.

Directions de l'ordre DRAW



L'étude du graphisme du Coco 2 doit être complétée par l'ordre DRAW qui permet d'obtenir des tracés de plusieurs lignes avec un seul ordre. Par exemple, un carré peut être tracé grâce à la ligne DRAW"M10,10; R50;D50 ;L50;U50" dans laquelle M déplace le pointeur en (10, 10) et R, D, L et U tracent des lignes de 50 points vers la droite (R comme right, en anglais), le bas (D comme down), la gauche (L comme left) et le haut (U comme up). Ces tracés étant en mouvement relatif, la dernière ligne ramène bien au point initial. E, F, G et H tracent des lignes en diagonale. Mais on peut également décaler les huit directions grâce à A. Par exemple après A1, elles seront décalées de 90 degrés et un R (right) opérera alors comme un U (up).

C permet de modifier la couleur du tracé et S de changer d'échelle. Une sous-chaîne sera exécutée grâce à X, comme dans l'ordre PLAY. L'ordre DRAW est donc très ingénieux, capable de faire des tracés assez complexes avec un minimum de directives.

Fiche technique du Coco 2

Nom : *Colour Computer 2, TRS-80 couleurs 2 ou Coco 2*
Constructeur et distributeur : *Tandy*
Mémoire vive : *16 Koctets de base, extension possible jusqu'à 64 Koctets*
Mémoire morte : *16 Koctets*
Langages : *Basic Microsoft, Assembleur du 6809*
Prix public : *2 500 FF ttc dans sa version de base ; 3 000 FF ttc avec un Basic étendu ; 3 900 FF ttc dans la version 64 Ko et un Basic étendu*
Variables : *réels stockés sur cinq octets (mantisse affichée sur 9 chiffres et exposant compris entre −38 et +38)*

Pour les fanas du 6809

Les fanatiques du 6809 pourront programmer en langage-machine grâce à PEEK, POKE, VARPTR, EXEC, ainsi que CLOADM et CSAVEM. L'ordre CLEAR *n, a* (à ne pas confondre avec CLEAR suivi d'un seul argument, pour les chaînes alphanumériques) réserve *n* octets à partir de l'adresse *a*, en vue du langage-machine. Outre EXEC, on peut appeler des programmes 6809 par USR qui permet, en plus, le passage de paramètres. Il faut auparavant définir le point d'entrée par DEF USR.

Les programmes sont facilement stockés sur cassette avec CLOAD, CSAVE et SKIPF. Les fichiers de données sont possibles avec OPEN-CLOSE, PRINT#, INPUT#, et EOF. Les instructions MOTOR ON et OFF commandent le moteur du magnétophone. Quant à AUDIO ON ou OFF, elles dirigent, ou non, le son de la cassette sur le téléviseur.

En conclusion, le point fort de Coco 2 est la richesse de ses ordres graphiques. Mais ses faibles commodités d'édition, son impossibilité de mixer texte et image (sauf en mode 64×32) et ses seules quatre couleurs sont de graves inconvénients face à certains ordinateurs concurrents, comme... le MO5, pour ne pas le citer.

Christian BOYER

CONVERSION DE CHAÎNE EN ENTIER

IL arrive souvent qu'un nombre soit mémorisé sous forme de chaîne. Mais alors, il n'est plus considéré comme un entier. Il faut donc le convertir. Grâce à une procédure Pascal, la conversion a lieu de telle sorte que le nombre soit stocké dans une variable de type entier.

Une procédure Pascal est chargée de convertir un nombre mémorisé dans une chaîne en un entier stocké dans une variable de type entier.

Elle est déclarée par :

```
procedure CONVENTIER (VALCHA : string ; var VALENT : ENTIER) ;
```

Le premier paramètre VALCHA, correspond à la VALeur, transmise sous forme de CHAîne, qu'il est nécessaire de convertir. Le second paramètre contient, après l'appel de cette procédure, la VALeur sous forme ENTière. Ce type ENTIER, pourra être INTEGER, si la capacité des variables de ce type suffit au problème à résoudre.

N'oubliez pas de reconvertir

Il faut noter que cette procédure ne permet que de convertir des chaînes mémorisant des nombres entiers. Toutefois, la conversion d'une chaîne comportant une virgule ou un point décimal est possible. Dans ce cas, le séparateur décimal est ignoré. Par exemple, dans un programme où sont traités des montants, les conversions sont effectuées, mais il faut savoir alors que les valeurs retournées correspondent à des centimes. Dans ce cas, il est nécessaire d'envoyer à la procédure une chaîne correctement formatée, c'est-à-dire comportant toujours le même nombre de chiffres après la virgule.

Le principal intérêt de cette procédure est de permettre à un programme d'effectuer la saisie des nombres sous forme de chaînes, et de ne pas être interrompu par un message d'erreur dans le cas d'une mauvaise entrée. Par exemple, dans un programme, la demande d'une référence numérique valant de 1 à 999 s'effectuera de la façon suivante :

```
VALABLE : = [1..999] ;
repeat
   WRITE('Reference de l''article  (<return> pour sortir) : ') ;
   READLN(CHAINE) ;
   if CHAINE <>''
      then
         begin
            CONVENTIER(CHAINE, REFERENCE) ;
            if not (REFERENCE in VALABLE)
               then
                  WRITELN('Erreur : cette reference est erronee')
         end
until (CHAINE = '') or (REFERENCE in VALABLE) ;
```

Le début du traitement que réalise la procédure (page suivante) consiste à initialiser deux variables. La première, VALENT, est mise à zéro : elle est destinée à mémoriser le nombre provenant de la chaîne. La seconde, appelée NEGATIF, est initialisée à la valeur FAUX, et indique si l'entier VALENT est positif ou négatif.

Cette procédure réalise ensuite une boucle sur chaque caractère de la chaîne qui a été passé en paramètre. Celui en cours de traitement est mémorisé dans la variable CAR. Cela permet d'accélérer l'exécution de la procédure et de réduire la longueur de son code. En effet, l'accès à une variable simple, tel qu'un caractère, est facile et rapide, alors qu'une recherche dans un tableau, tel qu'une chaîne, nécessite la

```
procedure CONVENTIER (VALCHA : string ; var VALENT : ENTIER) ;
var     I          : integer ;
        CAR        : char ;
        NEGATIF : boolean ;
begin
   VALENT : = 0 ;
   NEGATIF : = FALSE ;
   for I: = 1 to LENGTH (VALCHA) do
    begin
         CAR : = VALCHA [I] ;
         if CAR in ['0'..'9']
            then
               VALENT : = 10 * VALENT + ORD (CAR) — ORD ('0')
            else
               if CAR in ['+','–']
                  then
                     NEGATIF : = CAR = '—'
    end ;
   if NEGATIF then VALENT : = — VALENT
end ;
```

recherche de l'indice de l'élément, le calcul de son adresse, et un test pour vérifier s'il ne se situe pas au-delà de la longueur de la chaîne.

Lorsque le caractère est un chiffre, il intervient dans l'évaluation du nombre. Celui-ci est multiplié par 10, ce qui a pour effet de décaler tous les chiffres qui le composent vers la gauche. La valeur numérique du caractère lui est ajoutée, et se situe donc à la position des unités.

Lorsque le caractère est un signe, l'expression logique « le signe est négatif » est mémorisée dans la variable NEGATIF.

A la fin du traitement, le nombre est pris égal à son opposé si le signe *moins* a été détecté dans la chaîne, c'est-à-dire si NEGATIF est vrai.

Thierry CHAMORET

BALISTIQUE SUR PB-700

QUESTION DE MOUVEMENT

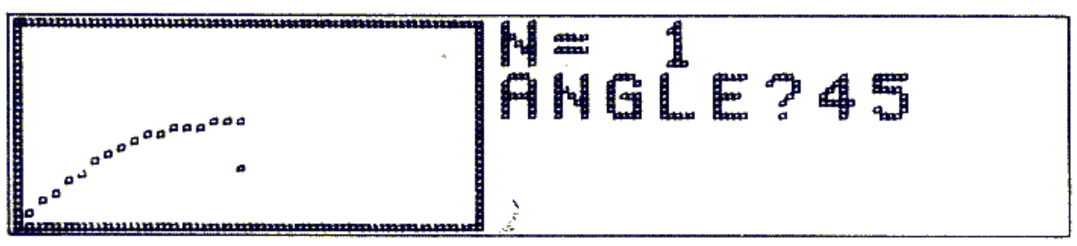
Il était une fois une équation. Devenue programme, elle transforma un phénomène physique peu connu, la balistique, en jeu... Ce n'est pas un conte de fées, mais une démarche intéressante — peut-être même un premier pas vers l'Enseignement Assisté par Ordinateur — suivie ici par le PB-700.

Posez à quelqu'un qui n'est pas très familiarisé avec les notions de mécanique la colle suivante : comment représenter la trajectoire d'un projectile lancé obliquement ou horizontalement, en négligeant la résistance de l'air.

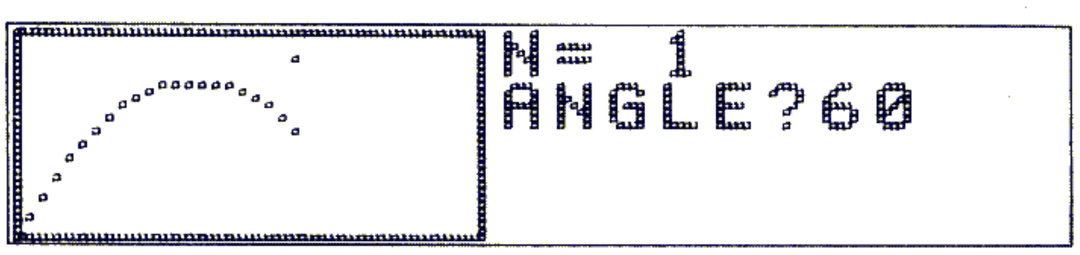
Votre interlocuteur vous tracera probablement une trajectoire oblique ou horizontale qui s'incurve progressivement pour devenir verticale... Ce petit test montre que l'intuition est souvent trompeuse et qu'en matière de balistique (puisque c'est de cela qu'il

Figure 1
Exécution du programme 1. Le petit point représente la cible

a. L'angle choisi (45 degrés) est trop important.



b. Quel que soit l'angle initial, la cible reste inaccessible. (Cas où l'équation de la trajectoire n'admet pas de racine correspondant aux coordonnées de la cible).



s'agit), bien des gens en sont restés à des concepts dépassés.

Les deux jeux proposés ici sont des variations sur ce thème de la balistique. Ils mettent à profit l'écran "haute résolution" du PB-700.

Dans le premier jeu, il s'agit d'atteindre une cible en ajustant correctement l'angle de tir d'un projectile ; celui-ci part du coin inférieur gauche de l'écran.

Trois niveaux de difficulté sont proposés (ligne 30 du programme 1) de 1 à 3. Selon le niveau choisi, la cible est soit immobile (niveau 1), soit en mouvement ; dans ce dernier cas, elle se

Programme 1

```
 10 REM ---BALISTIQUE---
 15 REM   *(JEU No 1)*
 20 REM ================
 25 CLS :N=1:G=10:V=25
 30 INPUT "NIVEAU DE DIFFICULTE(1-2-3)
    ";D:CLS
 40 GOSUB 200
 45 IF N=6 THEN 100
 50 DRAW(X0,Y0)
 60 GOSUB 300
 70 GOSUB 400
 75 IF INT(Y+.5)=Y0 THEN 100
 80 IF D=1 THEN 45
 85 DRAWC(X0,Y0)
 90 X0=X0-4:IF D=3 THEN Y0=Y0-1
 95 IF X0*Y0>0 THEN 50
100 BEEP :BEEP :BEEP 1
110 GOTO 20
200 REM"CADRE ET CIBLE"
210 DRAW(0,0)-(70,0)-(70,31)-(0,31)-(0
    ,0)
220 Y0=ROUND(RND*25,-1)+3
230 X0=ROUND(RND*34,-1)+16
240 IF X0 MOD 2=1 THEN X0=X0+1
250 RETURN
300 REM"ANGLE DE TIR"
310 LOCATE 9,0:PRINT "N=";N
320 LOCATE 9,1:PRINT "ANGLE=     "
330 LOCATE 14,1:INPUT A
340 RETURN
400 REM"TRAJECTOIRE"
410 N=N+1:X1=0:Y1=0
420 B=G/2/V/V/COS(A)/COS(A):C=TAN(A):P
    =C/B
430 IF X0<P THEN P=X0
440 IF A>70 THEN I=1 ELSE I=2
450 FOR X=0 TO P STEP I
460 Y=B*X*X-C*X+31
470 DRAWC(X1,Y1):DRAW(X,Y):X1=X:Y1=Y
480 NEXT X
490 DRAWC(X1,Y1):BEEP
495 RETURN
```

Programme 2

```
 10 REM ---BALISTIQUE---
 11 REM   *(JEU No 2)*
 12 REM   ============
 15 G=10:K=.5:N=1:X1=0:Y1=0
 20 GOSUB 100
 30 GOSUB 200
 40 GOSUB 300
 50 IF ABS(X0-P0)<1 THEN 95
 55 IF R*P<8 THEN 30
 60 P1=P0+K*P:A=A/K
 65 IF P1<80 THEN 80
 70 P1=80:R=0
 80 GOSUB 400
 90 GOTO 50
 95 BEEP :BEEP :BEEP 1:GOTO 10
100 REM"CADRE ET CIBLE"
110 CLS :DRAW(0,0)-(0,31)-(80,31)-(80,
    0)-(0,0)
120 X0=ROUND(RND*68,-1)+10
130 DRAWC(X0-1,31)-(X0+1,31)
140 DRAW(X0-2,30):DRAW(X0+2,30)
150 RETURN
200 REM "VITESSE"
210 LOCATE 11,0:PRINT "N=";N
220 LOCATE 11,1:PRINT "V=     "
230 LOCATE 13,1:INPUT V
240 R=1:N=N+1
250 RETURN
300 REM"PREMIERE CHUTE"
310 P0=V*SQR(62/G):P=2*P0:A=.5*G/V/V:B
    =SQR(62*G)/V
320 FOR X=0 TO P0 STEP 2
325 DRAWC(X1,Y1):Y=A*X*X
330 DRAW(X,Y):X1=X:Y1=Y
340 NEXT X
350 BEEP
355 DRAWC(X1,Y1)
360 RETURN
400 REM"REBOND"
410 FOR X=P0+1 TO P1
420 DRAWC(X1,Y1)
430 Z=X-P0:Y=A*Z*Z-B*Z+31
440 DRAW(X,Y):X1=X:Y1=Y
450 NEXT X
460 BEEP :P0=P1:P=K*P
480 RETURN
```

QUESTION DE MOUVEMENT

rapproche de vous en vol horizontal (niveau 2) ou en gagnant de l'altitude (niveau 3).

Dans la première hypothèse, le nombre d'essais est limité à cinq. Pour les deux autres niveaux, on a tout le loisir d'ajuster le tir jusqu'au moment où la cible sort de l'espace de contrôle, ou encore, éventualité plus dramatique, où elle se trouve à la verticale du poste de tir...

Le deuxième jeu est beaucoup moins stressant : il faut loger une bille dans un trou creusé dans le sol, en la lançant à la vitesse convenable. Et même si elle n'a pas été lancée suffisamment fort, il reste encore une chance de gagner ! En effet, dans ce cas, elle rebondit en arrivant au sol, et après plusieurs sauts, elle peut très bien tomber dans le but.

Quelques aspects théoriques doivent être abordés, à l'intention des curieux. Pour le premier jeu, l'équation de la trajectoire du projectile, qui sera appelée *équation 1*, s'écrit :

$z = (-g/2v^2\cos^2\alpha) + (\tan\alpha)x$

où g représente l'intensité de la pesanteur (10 m/s²), v la vitesse d'éjection du projectile (25 m/s), et α l'angle de tir.

La ''portée'' s'obtient en faisant $z = 0$ dans l'*équation 1*, ce qui donne :

$P = (v^2 \sin 2\alpha)/g$

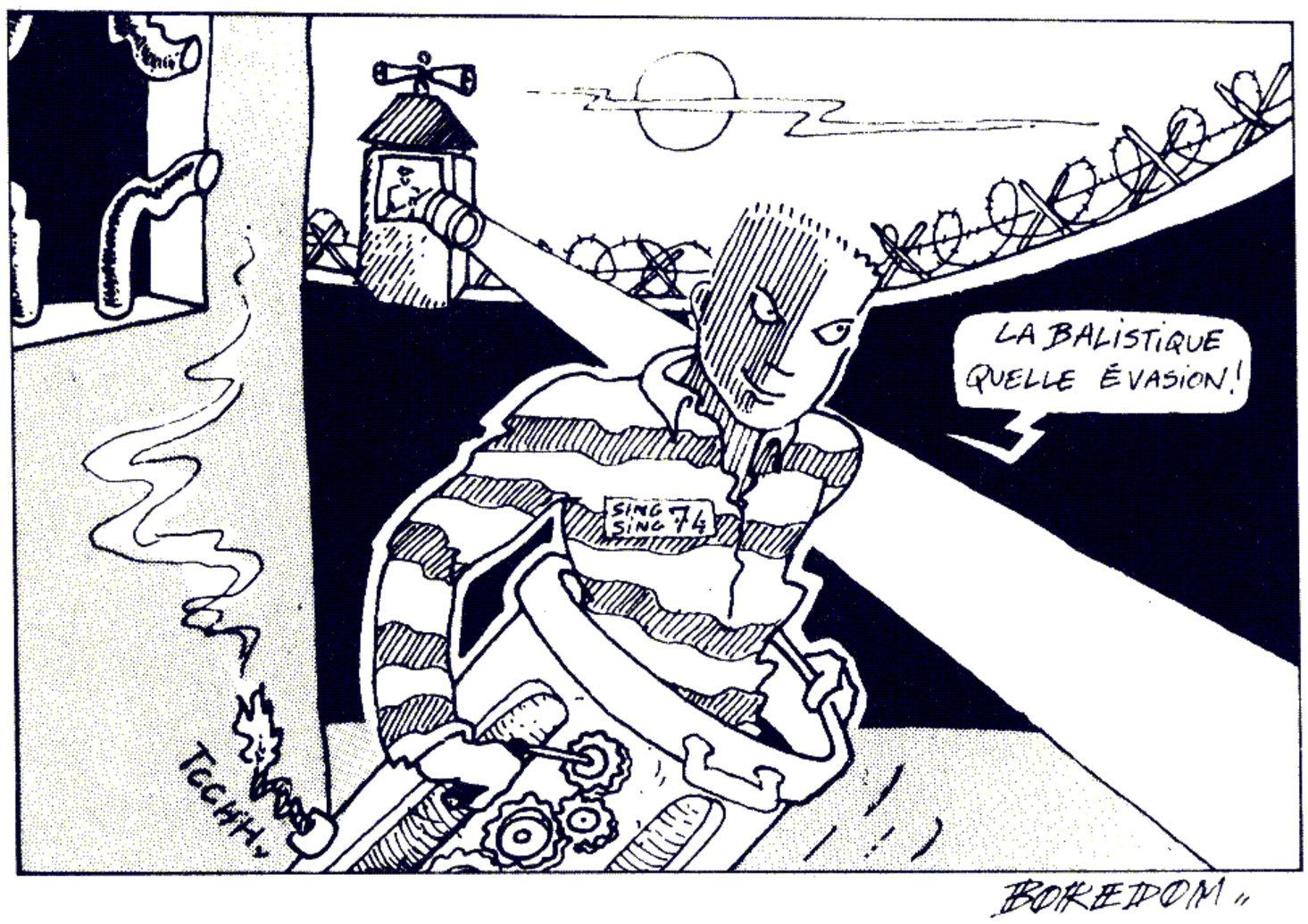
On voit que P est maximum quand $\sin 2\alpha = 1$, c'est-à-dire quand $\alpha = 45$degrés.

Au cours des parties, on observe que certaines positions de la cible la mettent hors d'atteinte du projectile, quel que soit l'angle choisi (voir figure 1b). Ceci s'explique en faisant intervenir la notion de ''parabole de sûreté''. L'*équation 1*, en effet, s'écrit encore :

$\tan^2\alpha - 2(v^2/gx)\tan\alpha + ((2v^2/gx^2)z + 1) = 0$

Il s'agit alors d'une équation du second degré en ''tanα'', et trois cas peuvent se présenter selon le signe du discriminant :

- s'il est positif, la cible de coordonnées (x,z) est atteinte par deux angles de tir (les racines de cette équation) ;
- s'il est négatif, la cible est inaccessible, l'équation n'ayant pas de racines réelles ;
- s'il est nul, il n'existe qu'un seul angle de tir pour atteindre la cible.



Avant et après

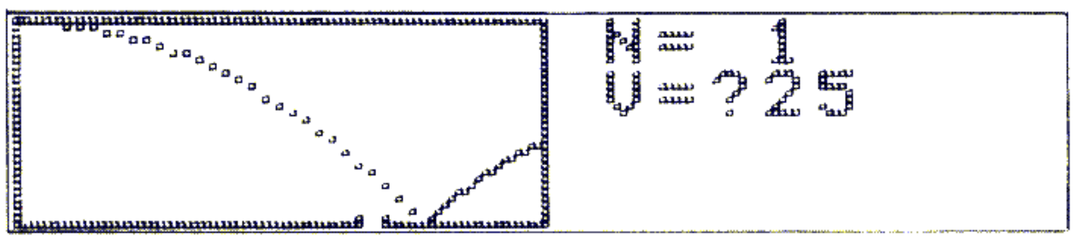
Le dernier cas (discriminant nul) permet d'écrire l'équation de la ''parabole de sûreté'', dans le plan. Pour le second jeu, à chaque rebond, la bille perd une partie de son énergie cinétique. Le coefficient K (ligne 15 du programme 2) représente le rapport des énergies cinétiques avant et après chaque contact de la bille avec le sol. Théoriquement, le nombre de rebonds est infini, mais dans la pratique, la bille s'arrête assez rapidement.

Un autre jeu consiste à calculer les distances totales parcourues par la bille dans les deux sens, vertical et horizontal !

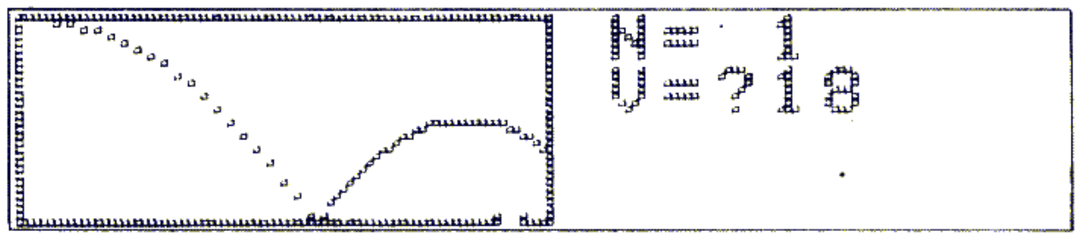
Trung HUA-NGOC

Figure 2
Exécution du programme 2.
La cible est le petit espace à la base du rectangle, qui représente un trou dans le sol

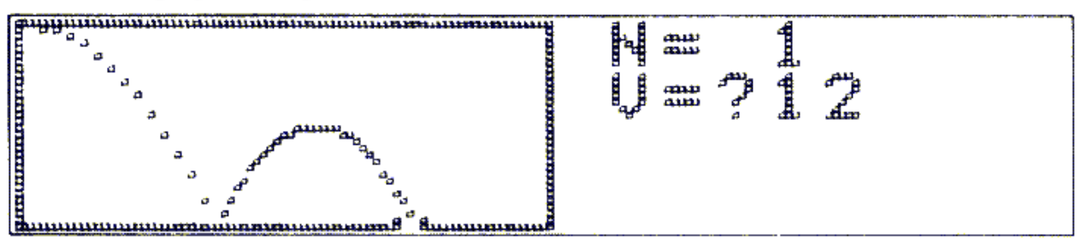
a. La vitesse initiale est trop forte



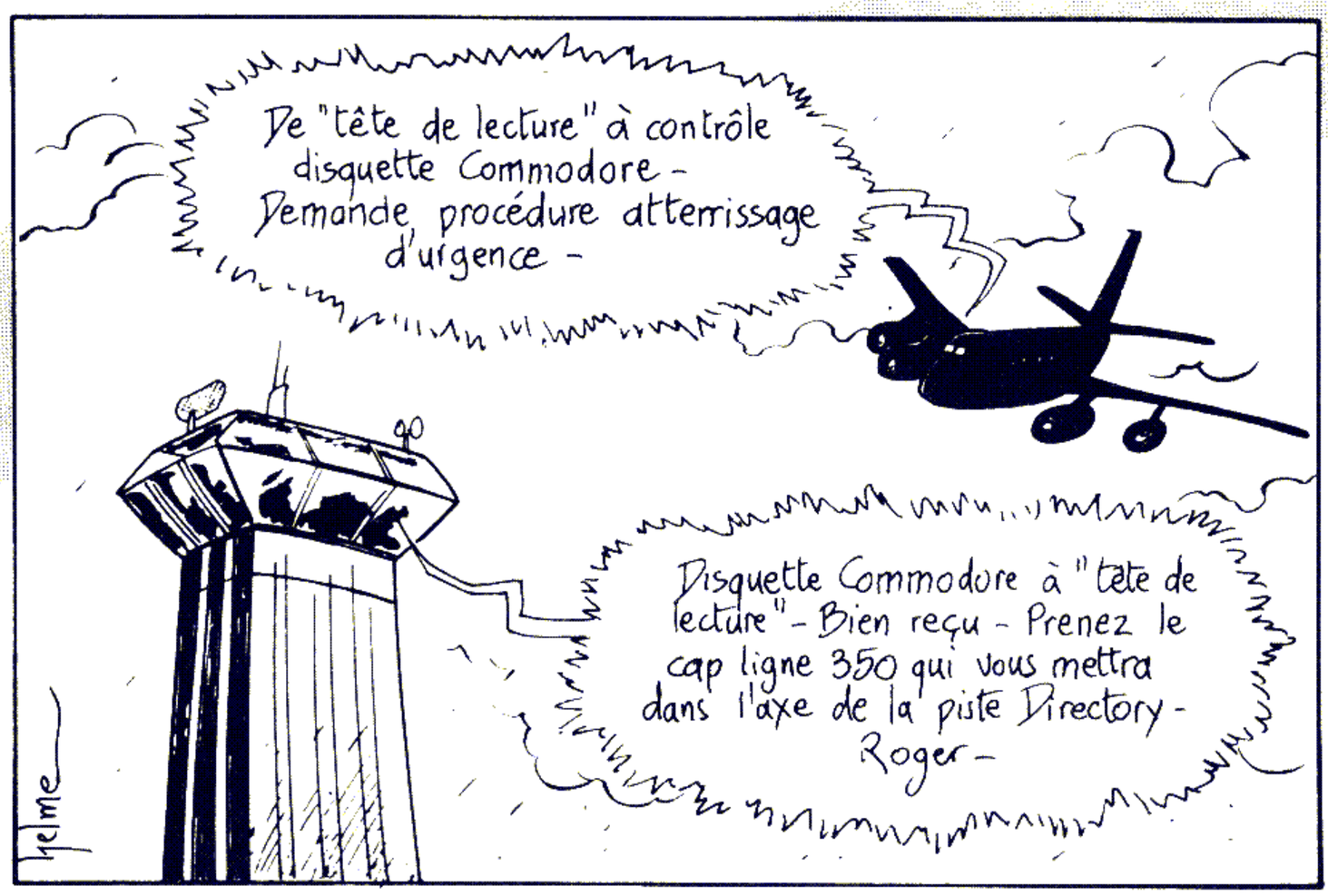
b. La vitesse initiale est trop faible



c. C'est gagné... grâce au rebond



SUR LA PISTE DIRECTORY



LA « gestion-manipulation » des disquettes Commodore fait souvent pénétrer dans des techniques trop rarement utilisées. Il est vrai qu'elles ne sont pas toujours faciles d'accès, bien que leurs applications soient nombreuses. La description de la piste « directory », piste centrale d'une disquette Commodore, va permettre ici une approche sans effort.

Chaque disquette Commodore est découpée, lors du formatage, en une série de pistes. L'une d'elle porte le numéro 18 sur l'échelle des numéros de piste, qui s'étend de 1 à 35, et occupe donc une position médiane sur la zone magnétique des disquettes : c'est la piste « directory ».

Sa situation présente l'énorme avantage de limiter les déplacements de la tête de lecture/écriture, qui sont grands dévoreurs de temps, et nuisent par conséquent aux performances ! Cette piste est découpée en 19 secteurs (numérotés de 0 à 18). Le secteur 0 est particulier, les 18 secteurs restants sont ceux qui nous intéressent ici.

La figure 1 fournit une base de travail intéressante, sous la forme d'une image hexadécimale du contenu de la piste 18, secteur 1 d'une disquette quelconque.

Cette image rassemble 256 octets formant un « block » (bloc, en français !). A l'extrême droite, figure la traduction ASCII des octets du bloc ; les codes non alphanumériques étant traduits par de simples points.

On y reconnaît sans effort les titres

Figure 1
Le premier bloc de la piste

```
PISTE 18 SECTEUR 1
  0 > 00 FF 82 11 00 31 20 50 41 53 54 41 47 41 A0 A0  : .....1 PASTAGA..
 16 > A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 16 00  : ................
 32 > 00 00 81 13 00 32 44 49 52 43 4F 4D 50 A0 A0 A0  : .....2DIRCOMP...
 48 > A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 05 00  : ................
 64 > 00 00 84 13 03 31 44 49 52 43 4F 4D 50 A0 A0 A0  : .....1DIRCOMP...
 80 > A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 07 00  : ................
 96 > 00 00 00 13 07 45 4C 49 4D 49 4E 45 A0 A0 A0 A0  : .....ELIMINE....
112 > A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 4C 00  : ..............L.
128 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
144 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
160 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
176 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
192 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
208 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
224 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
240 > 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  : ................
```

SUR LA PISTE DIRECTORY

des fichiers présents sur la disquette et qui apparaissent par un LOAD ''$'', 8 (figure 2). Les autres octets du bloc contiennent des informations utiles. Trente octets sont consacrés à la description de chaque fichier, ce qui limite par conséquent le nombre de titres à 8 au maximum pour chaque bloc de la piste 18.

Figure 2
Le catalogue normal et... incomplet

```
0 "ESSAIS 64/2      " 62 2A
22    "1 PASTAGA"         PRG
5     "2DIRCOMP"          SEQ
7     "1DIRCOMP"          REL
630 BLOCKS FREE.
```

Les données dont on dispose pour chaque titre sont les suivantes :

• octet 1 : type de fichier ; il en existe quatre principaux : SEQ, PRG, USR, REL ; un cinquième peut s'y adjoindre : le fichier DEL (*delete,* détruire en français) ;

• octet 2 : numéro de la piste où commence le fichier ;

• octet 3 : numéro du secteur sur la piste en question ;

• octets 4 à 19 : codes ASCII du titre du fichier ; les derniers étant des A0 (160 en décimal, espaces shiftés), si le titre a une longueur inférieure à 16 caractères ;

• octets 20 à 28 : ils sont presque toujours à 00, sauf cas particuliers (fichiers REL, par exemple) ;

• octets 29 et 30 : nombre total de blocs occupés par le fichier.

En créant un programme « catalogue étendu » (page ci-contre), plus précis que le catalogue habituel, on fait réapparaître cet ensemble d'informations. Et pour en tirer le meilleur profit, il faut agir avec méthode et suivre les explications du programme.

Tel qu'il est livré ici, le programme affiche les informations attendues sur l'écran de la machine. Et pour obtenir ces résultats sur l'imprimante, rien de plus facile. Il suffit d'ajouter les lignes suivantes au programme :

```
220 OPEN 4,4
830 CLOSE 4
```

Sans omettre, naturellement, de modifier tous les 'PRINT' à partir de la ligne 330 (incluse), en 'PRINT# 4,'.

La figure 3 présente le résultat obtenu sur imprimante à partir d'une disquette d'essai grâce à cet utilitaire. N'est-ce pas mieux que le traditionnel catalogue ?

Figure 3
Le catalogue étendu

```
TITRE: ESSAIS 64/2        ID: 62

FIC #P  #S      TITRE          #B
PRG 17   0  " 1 PASTAGA       " 22
SEQ 19   0  " 2DIRCOMP        "  5
REL 19   3  " 1DIRCOMP        "  7
DEL 19   7  " ELIMINE         " 76
554 SECTEURS LIBRES
```

Pour son utilisation, deux idées essentielles sont à retenir.

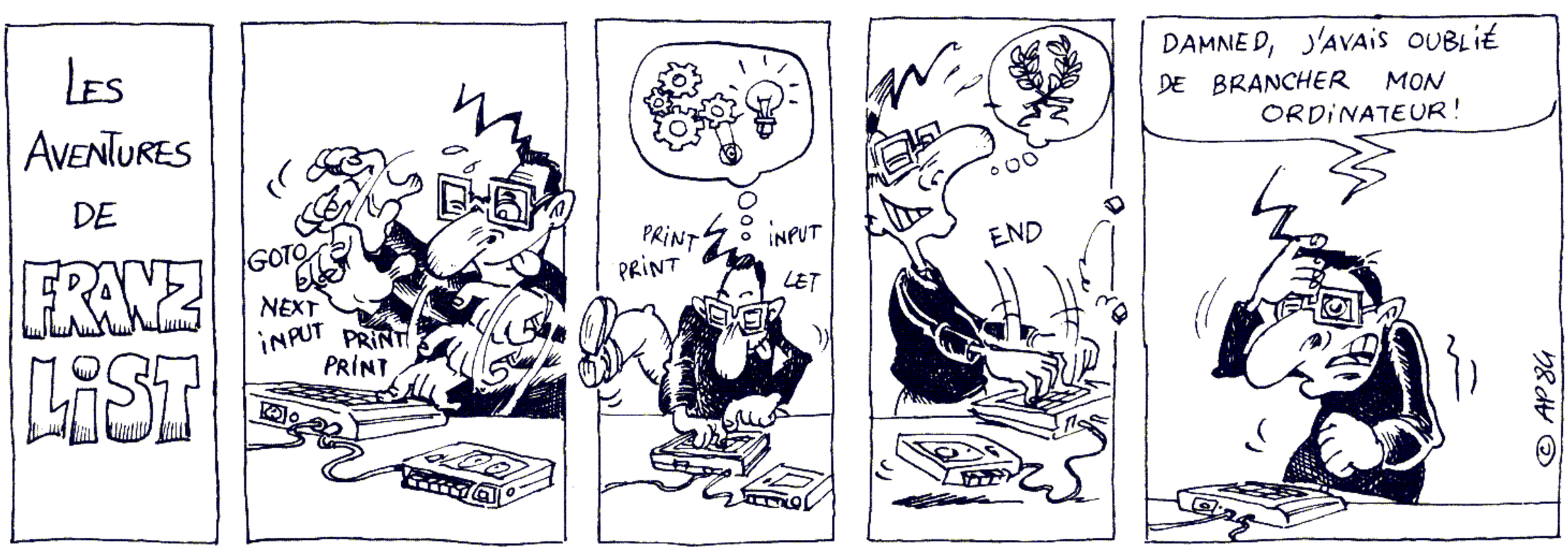
Si, par inattention ou erreur de manipulation, un important fichier a été détruit... et si par malheur, l'étendue du désastre n'est découverte que plus tard, le programme proposé est d'un secours remarquable. Il permet de savoir ce qu'il est advenu du fichier perdu, et s'il est récupérable. C'est un premier pas franchi vers la réparation d'une catastrophe accablante !

Prenez vos précautions

L'utilisation fréquente de la commande SAVE ''@0 : titre'', 8 entraîne parfois une mésaventure, probablement due à une bogue du SED (Système d'Exploitation des Disquettes) Commodore. De fait, quand on essaye de charger le programme en question, c'est un autre qui est mis en mémoire... Le programme essentiel est, par conséquent, perdu. Or, si l'on a pris la précaution de conserver une trace de la piste et du secteur où commence le fichier — et cela, grâce au programme *catalogue étendu* — on s'en tirera généralement sans problème !

D'autres idées d'utilisations peuvent sans doute être trouvées et mises en pratique. Dans un prochain article, un sixième type de fichiers sera dévoilé. En attendant, la ligne 500 du programme mérite de l'attention : elle contient la réponse à l'énigme qu'elle pose !

Jean-Pierre LALEVÉE



Catalogue étendu
Programme pour Commodore 64 et lecteur de disquette
Auteur Jean-Pierre Lalevée
Copyright *LIST* et l'auteur

```
100 REM ****************************
110 REM * UTILITAIRE DISQUE NO.2 *
120 REM *    CATALOGUE ETENDU    *
130 REM ****************************
140 :
150 PRINT"■","■ CATALOGUE ETENDU "
160 PRINT"■LECTURE COMPLETE DU CATALOGUE."
170 PRINT"RECONNAISSANCE DES FICHIERS DETRUITS."
180 :
190 C0$=CHR$(0)
200 DIM A$(4):FOR I=0 TO 4:READ A$(I):NEXT
210 DATA DEL,SEQ,PRG,USR,REL:REM TYPES DE FICHIERS
220 :
230 REM ****************** INITIALISATIONS DISQUE **********************
240 OPEN 15,8,15,"I0":GOSUB 880
250 :
260 OPEN 2,8,2,"#":GOSUB 880
270 :
280 REM ************************ TITRE ET ID ***************************
290 PRINT#15,"U1"2;0;18;0:GOSUB 880
300 PRINT#15,"B-P"2;144:GOSUB 880
310 :
320 FOR I=0 TO 19:GET#2,A$:F$=F$+A$:NEXT I
330 PRINT"■■■ TITRE: ■";LEFT$(F$,16);"■ ID: ■";RIGHT$(F$,2)
340 :
350 REM ************************  CATALOGUE  ***************************
360 PRINT"■■ FIC #P  #S         TITRE         #B "
370 P=18:S=1
380 PRINT#15,"U1"2;0;P;S:GOSUB 880
390 PRINT#15,"B-P"2;0:GOSUB 880
400 :
410 REM ++++++++++++++++++ EMPLACEMENT SUIVANT ++++++++++++++++++++++
420 GET#2,P$:P=ASC(P$+C0$):REM PISTE
430 GET#2,S$:S=ASC(S$+C0$):REM SECTEUR
440 :
450 FOR J=0 TO 7:REM IL Y A 8 TITRES MAXI PAR BLOC
460 :
470 REM +++++++++++++++++++++ TYPE DU FICHIER ++++++++++++++++++++++++
480 PRINT#15,"B-P"2;J*32+2
490 GET#2,X$:X=ASC(X$+C0$) AND 15:TF$=A$(X)
500 IF ASC(X$+C0$) AND 64 THEN TF$=LEFT$(TF$,2)+"*":REM MYSTERE !!!
510 :
520 REM ++++++++++++++++++++++ PISTE, SECTEUR ++++++++++++++++++++++++
530 GET#2,N$:D=ASC(N$+C0$)
540 IF D=0 THEN 770:REM PLUS DE TITRES ?
550 PRINT " ";TF$;
560 IF D<10 THEN PRINT" ";
570 PRINT D;
580 GET#2,N$:D=ASC(N$+C0$)
590 IF D<10 THEN PRINT" ";
600 PRINT D;
610 :
620 REM ++++++++++++++++++++++ TITRE FICHIER ++++++++++++++++++++++++
630 PRINT"' ";
640 FOR I=1 TO 16
650 GET#2,A$:PRINT A$;
660 NEXT I
670 PRINT"'";
680 :
690 REM ++++++++++++++++++++++ NB DE SECTEURS ++++++++++++++++++++++++
700 PRINT#15,"B-P"2;J*32+30
710 GET#2,L$,H$:L=ASC(L$+C0$):H=ASC(H$+C0$)
720 D=H*256+L
730 IF D<10 THEN PRINT" ";
740 PRINT D
750 NS=NS+D
760 :
770 NEXT J
780 :
790 IF P THEN 380:REM SUITE ?
800 :
810 REM ++++++++++++++++++++ FIN DU DIRECTORY ++++++++++++++++++++++++
820 PRINT"■";664-NS;"SECTEURS LIBRES"
830 :
840 CLOSE 2:CLOSE 15
850 GOTO 920
860 :
870 REM ********************* S/P ERREURS DISQUE ***********************
880 INPUT#15,E1,E$,E3,E4
890 IF E1>20 THEN PRINT"■ERREUR DISQUE :":PRINT E1","E$","E3","E4
900 RETURN
910 :
920 END
READY.
```

Explications du programme

Lignes 190 à 210 : initialisations diverses, avec en particulier mémorisation des différents types de fichiers qu'il est possible de rencontrer.

Lignes 230 à 260 : ouverture du canal de commande, et du canal de transfert de données. Une précaution a été prise : vérifier que tout se passe sans problème, grâce aux appels du sous-programme de lecture des erreurs.

Lignes 280 à 330 : on a besoin de connaître le titre et le numéro d'identification de la disquette ; ces renseignements se trouvent sur le secteur 0 ; on fait donc une entorse à la décision de ne pas toucher à ce secteur pour l'instant...

A partir de la ligne 350, commence la lecture du catalogue proprement dit. Tous les renseignements détectés dans l'ordre de leur rencontre seront écrits : le type de fichier, les numéros de piste et de secteur de début du fichier, le titre, et enfin le nombre de blocs occupés.

Ligne 370 : la première lecture doit se faire sur la piste 18, secteur 0 (début du catalogue : TITRE et ID).

Ligne 380 : dans le tampon numéro 2, le bloc concerné est mis en mémoire, pour pouvoir être lu par la suite, soit 256 octets.

Ligne 390 : le pointeur qui permettra d'effectuer la lecture à partir de l'octet 0 est positionné.

Lignes 420 et 430 : les numéros de piste, puis de secteur du bloc à lire après celui-là sont extraits successivement. La suite doit être repétée huit fois, puisque huit titres au maximum sont disponibles dans chaque bloc.

Lignes 480 et 490 : le pointeur est positionné sur l'octet concerné qui représente le type du fichier ; et il est ramassé au passage. TF$ est la variable du type.

Ligne 500 : cette ligne est un peu spéciale ! Il s'agit ici du sixième type de fichier qui fera l'objet d'une prochaine étude...

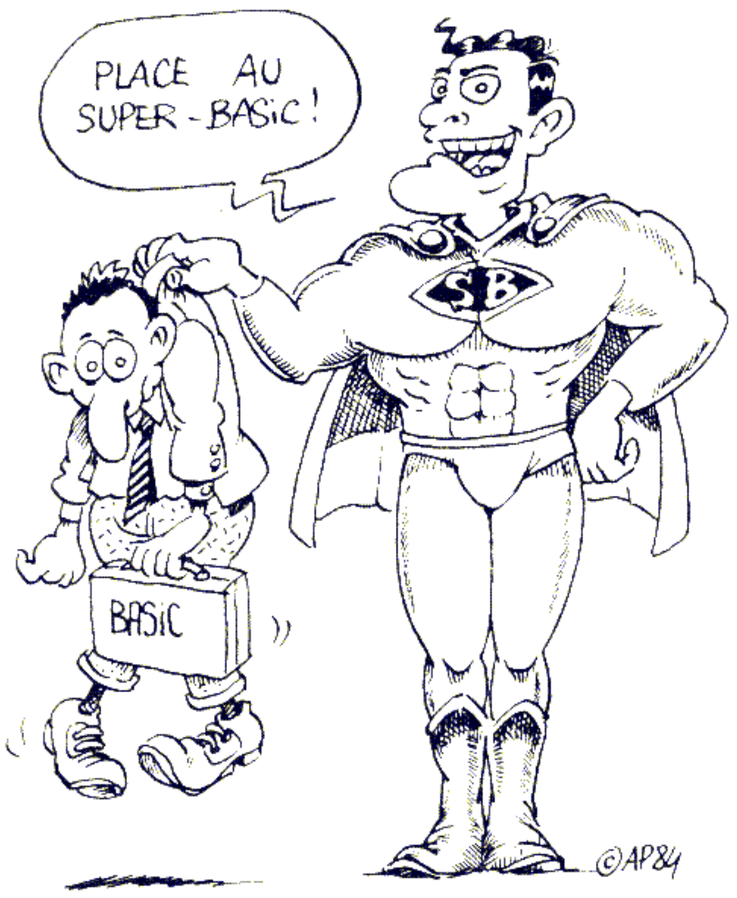
Lignes 530 à 600 : les deux octets suivants sont récupérés, puisqu'ils contiennent un numéro de piste et de secteur dont on a besoin. Mais attention, si le numéro de piste est 0, cela signifie que tous les titres ont été lus : en ce cas, le travail sera très vite terminé (ligne 540).

Lignes 620 à 670 : voici le tour des 16 octets ASCII du titre qui sont affichés sans autre forme de procès.

Lignes 700 à 750 : enfin, le pointeur est positionné un peu plus loin, sur les octets qui contiennent le nombre de blocs occupés par le fichier. Cette valeur est comptabilisée dans le compteur des blocs en ligne 750.

Lignes 820 à 850 : le travail achevé, le nombre de secteurs restés réellement libres est affiché, puis les canaux qui ont rempli leur office sont refermés.

LE BASIC RÉCURSIF



CERTAINS reprochent au Basic standard d'être trop lent, difficile à structurer, pauvre en procédures, etc. Peut-être trouveront-ils des solutions satisfaisantes à ces problèmes avec le Basic récursif.

La guerre des langages est impitoyable ; elle dépasse en horreur la lutte sans pitié que se livraient, si l'on en croit Gulliver, les petits-boutistes et gros-boutistes sur la meilleure façon de déguster un œuf à la coque !

Basic est, on le sait, à la fois triomphant en informatique par son extrême diffusion, et abondamment méprisé et attaqué par les « puristes ». Les arguments sont toujours les mêmes (et ils ne manquent pas toujours de force) : lenteur insupportable, rarement compilable, structuration difficile, pauvreté des procédures souvent réduites au seul GOSUB. Par exemple, Basic n'est pas récursif, alors que Pascal et le LSE (Langage Symbolique d'Enseignement), tous deux fils d'Algol, ont ce luxe — également inconnu des Assembleurs ou des grands ancêtres Fortran et Cobol.

Dialectes d'un langage populaire

S'il existait bien déjà des « dialectes » récursifs de Basic, ce n'était guère que sur des machines très professionnelles, comme le HP-9816 de Hewlett-Packard. Or en voici une implantation sur un matériel de plus grande diffusion : le remarquable HP-71B (présenté dans le numéro 2 de *LIST*, page 41), sans doute promis à une grande notoriété, en particulier à cause de la puissance de son langage.

On annonce par ailleurs une version récursive d'un super-Basic, appelé **Sioux**, à venir dans quelques mois pour les Thomson TO7 et TO7/70 (quelques échos, parfois maladroits, de cette introduction étaient déjà parus dans la presse au printemps, sous le nom de code initial EXEL).

En fait, l'origine du Sioux est assez ancienne. Développé au Laboratoire Central de Recherches de Thomson CSF, il est issu de travaux universitaires français, notamment autour de Messieurs Arsac, Nolin, Ruggiu et Vasseur, aux alentours de 1974, pour concevoir un sur-langage pour APL (et aujourd'hui, Basic) destiné notamment à en accroître les possibilités de structuration. Il sera sans doute disponible début 1985 sous la forme d'une cartouche de 16 Ko.

L'arrivée de ces Basic prive LSE et, dans une moindre mesure, Pascal, d'une partie de leurs arguments contre la " naïveté " du plus populaire des langages. Rappelons en deux mots ce dont il s'agit : c'est une vieille histoire, qui remonte à Descartes et à sa méthode de décomposition d'un problème complexe en problèmes plus faciles.

Supposons que nous ayons à construire un programme — c'est-à-dire un algorithme qui transforme un ensemble A de données en un ensemble R de résultats — tel que le passage de A à R, noté comme en mathématiques par une relation du type R = F (A), nécessite un grand nombre d'étapes intermédiaires. Très fréquemment, l'obtention de F(A) exige que l'on traite des résultats partiels F (A'), F (A'')... à partir de données moins riches que A.

L'exemple le plus simple est celui du calcul d'une suite u(n) définie par récurrence : on connait u(0) et une règle qui donne explicitement u(n + 1) en fonction de u(n). Une boucle Basic ordinaire permet le traitement d'une telle récurrence (FOR I = 0 TO N...). On parle alors de procédure itérative. Par exemple, pour calculer la somme des cubes des entiers de 0 à N pour laquelle u(0) = 0 et u(n + 1) = u (n) + $(n + 1)^3$, on peut se référer au programme 1.

Un référendum sur les référendums

La situation est parfois bien plus complexe. Voici quelques exemples tout à fait classiques puisés dans les cours de programmation. Le premier concerne le " Tri rapide " de Hoare (Quicksort en anglais), découvert en 1962 ; une version Basic " ordinaire " de ce tri figure dans le programme 2.

Il s'agit de ranger en ordre croissant N nombres réels, mis dans un tableau (T(1), T(2),..., T(N)). La lecture du programme est des plus pénibles — c'est un excellent exercice que de l'analyser complètement — mais son efficacité, pour N au moins égal à 50, est bien supérieure en moyenne à celle des tris plus classiques. En réalité, tout lecteur qui voudra prendre le temps de comprendre l'algorithme s'apercevra

qu'il s'agit en fait de ramener le problème du tri d'une liste de N nombres à celui de plusieurs listes plus courtes, jusqu'à obtenir des sous-listes de 1 ou 2 nombres pour lequel le problème est enfantin (1).

Une telle attitude est dite récursive. Au lieu d'aller du simple vers le compliqué, comme dans la technique itérative, on se tient un raisonnement du type : trier une liste de 48 nombres, je ne sais pas faire ; mais je peux la couper en deux sous-listes de, disons 17 et 31 nombres, telles que toute valeur du premier ensemble soit inférieure à toute valeur du second, je n'aurai plus alors qu'à trier des listes plus courtes qui, à leur tour, pourront elles-mêmes être scindées, etc.

On cherche à décomposer le problème initial en tranches moins complexes, allant donc cette fois-ci du compliqué vers le simple. N'importe quel livre sur Pascal contient des versions récursives du Quicksort rédigées dans ce langage, bien plus faciles à lire que notre programme Basic.

Si l'on voulait imaginer une illustration de la récursivité en politique, voici une suggestion — bien peu sérieuse — issue de la technique classique des « référendums sur les référendums » : on commence par un référendum sur une question quelconque ; on propose ensuite, chaque mois, un référendum portant sur la question suivante : « Etes-vous d'accord avec la majorité qui s'est dégagée au cours du référendum du mois précédent ? »

Le score du référendum R(n) aurait alors une définition visiblement récursive, car il ne pourrait être calculé qu'en fonction des scores précédents aux dates 0, 1, 2... jusqu'à (n-1) ; sa signification objective risquerait d'être assez confusément perçue par l'homme de la rue...

Voici un dernier exemple, moins farfelu, imaginé par John Mc Carthy (né en 1927, il est bien connu comme créateur du langage Lisp spécialisé en intelligence artificielle). Il s'agit d'étudier les suites u(n), s'il en existe, telles que u(n) = n-10 si n est au moins égal à 101, et que u(n) = u(m) avec m = u(n + 11) dans les autres cas (on peut montrer qu'il y en a une et une seule, définie par la formule très simple : u(n) = max (91,n-10)).

Le calcul de u(n) en Basic ordinaire est possible, même sans utiliser la relation ci-dessus : le programme 3 donne une réalisation possible de l'algorithme, qui suit la définition de Mc Carthy. Elle n'est pas très difficile à comprendre, mais le rôle joué par l'index I n'est pas immédiat à interpréter a priori.

Alors, pour ou contre ?

En fait la bonne présentation de cette suite, telle qu'elle a été écrite par son inventeur, est évidemment récursive. Voici le programme HP-71 B correspondant :

```
10 INPUT N @ DISP FNU (N)
20 DEF FNU (N) @ IF N > 100 THEN
   FNU = N-10 ELSE FNU = FNU
   (FNU (N + 11))
```

L'équivalent en Sioux sur Thomson TO7 de la ligne 20 sera également très court, quelque chose comme :

```
DEF F(U;N) : SI N > 100 ALORS
U = N-10 SINON U = F (F(N + 11)) IS
FIN
```

Bien entendu, une telle comparaison entre les listes récursives et non récursives est très trompeuse. C'est parce que la suite de Mc Carthy est l'un des deux ou trois exemples (avec le calcul de n! ou des coefficients du binôme) qui sont « étudiés pour », qu'est si net l'avantage au profit de la récursivité. Pour un programme pris « au hasard », le gain serait bien plus minime.

Notons qu'on démontre que tout programme où intervient explicitement une récursivité peut être transformé en programme itératif, généralement beaucoup moins lisible, mais souvent plus rapide. Alors, pour ou contre la récursivité ?

André WARUSFEL

(1) On pourra trouver une description complète de ce programme dans le numéro 49 de l'Ordinateur Individuel *(page 194) ; on en lira une autre version dans* « Programmes Basic pour scientifiques et ingénieurs » *d'Alan Miller, aux éditions Sybex (page 148), ou dans* LIST *n° 3 (page 52).*

Programme 1
Programme Basic itératif calculant la somme U(N) des cubes des entiers inférieurs ou égaux à N

```
10 INPUT "N=";N:U=0
20 FOR I=0 TO N
30     U=I*I*I+U
40 NEXT
50 PRINT U
60 END
```

Programme 2
Programme Basic (non récursif) de Quicksort (tri rapide de Hoare)

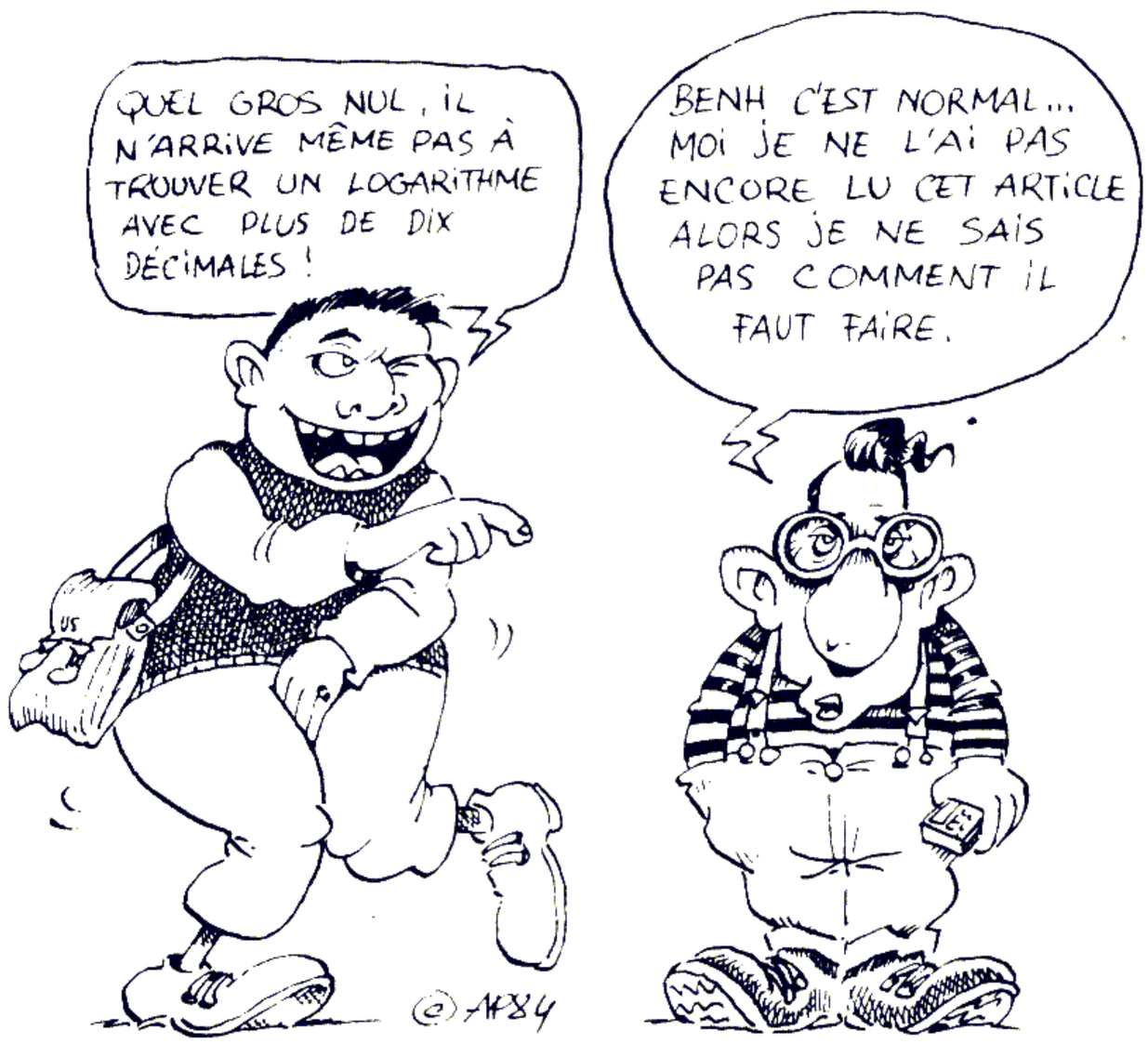
```
10 INPUT "N=";N:DIM T(N),G(N/2),D(N/2)
20 FOR I=1 TO N
30     PRINT "T(";:PRINT I;:PRINT ")";
40     INPUT T(I)
50 NEXT
60 P=1:G(1)=1:D(1)=N
70 G=G(P):D=D(P):P=P-1
80 I=G:J=D:X=(T(G)+T(D))/2
90 IF T(I)<X THEN I=I+1:GOTO 90
100 IF T(J)>X THEN J=J-1:GOTO 100
110 IF I<=J THEN T=T(I):T(I)=T(J):T(J)=T:I=I+1:J=J-1
120 IF I<=J THEN 90
130 IF I<D THEN P=P+1:G(P)=I:D(P)=D
140 IF G<J THEN D=J:GOTO 80
150 IF P<>0 THEN 70
160 FOR I=1 TO N
170     PRINT T(I);:PRINT ";";
180 NEXT:END
```

Programme 3
Programme Basic (non récursif) pour calculer les valeurs de la suite de Mc Carthy

```
10 I=1: INPUT "N=";N
20 IF N>100 THEN 40
30 N=N+11:I=I+1:GOTO 20
40 N=N-10:I=I-1
50 IF I=0 THEN PRINT N:END
60 GOTO 20
```

VINGT DÉCIMALES POUR UN LOG

C'EST un programme écrit en Basic standard qui permet d'obtenir les logarithmes avec vingt décimales. Pour cela, il fait appel à la multiprécision, aux mémoires tournantes, au "pilotage" et même aux développements en série.



Par rapport aux anciennes tables de logarithmes à 5 ou 7 décimales, la fonction LOG des ordinateurs, ou même des calculatrices, apporte aujourd'hui facilité, confort, sécurité et précision.

De merveilleuses petites machines fournissent instantanément les logarithmes naturels ou décimaux avec généralement 10 décimales, voire 10 chiffres significatifs (il ne faut pas en effet confondre ces deux notions : par exemple 0,031 est exprimé avec 3 décimales, mais seulement 2 chiffres significatifs).

Mais... plus on en a, plus on en veut, c'est bien connu. Rechercher des logarithmes avec une précision meilleure que 10 décimales ne constitue pas seulement un excellent entraînement sportif, mais devient une quasi-nécessité dans certains calculs mettant en jeu des fonctions exponentielles. Pourquoi le calcul des factorielles par la formule de Stirling est-il si précaire dès que l'on atteint des nombres élevés ? Pourquoi cette méthode appliquée à l'évaluation de la factorielle de 50 000 000 par exemple, ne permet-elle d'exprimer raisonnablement le résultat qu'avec deux chiffres significatifs ? C'est parce que les logarithmes ne sont indiqués par la plupart des ordinateurs qu'avec 10 chiffres significatifs.

C'est la raison pour laquelle il fallait un programme pour donner les logarithmes décimaux avec 20 décimales. Il met en jeu plusieurs techniques, dont la plus connue est la multiprécision, et quelques autres, plus confidentielles (les usagers du distorseur van vogtien seront heureux de les découvrir).

Ce programme est écrit en Basic standard. Pour s'y retrouver facile-

ment, les mémoires ont été assignées par ordre alphabétique, avec des exceptions :

- N est réservé à la variable principale (nombre dont on veut connaître le logarithme) ;
- les variables V, W, X et Y sont affectées aux constantes.

On s'est fixé pour but de calculer les logarithmes décimaux des nombres compris entre 1 et 10 (borne supérieure exclue) avec 20 décimales, le nombre introduit pouvant comporter au plus 10 chiffres significatifs, comme par exemple 6,789 314 159.

Pour obtenir 20 décimales exactes, il a fallu naturellement travailler avec plus de 20 décimales afin de disposer d'une bonne sécurité quant à l'exactitude de la vingtième et dernière décimale. Tous les calculs ont donc été effectués en double ou triple précision, avec 24 décimales, et arrondissement final à 20 décimales, d'où une incertitude de l'ordre de 10^{-4} sur le dernier chiffre affiché. Soit dit en passant, c'est bien meilleur que la précision offerte par un ordinateur qui exécute les calculs avec 12 chiffres et en affiche 10, avec une marge d'erreur de 10^{-2} quant au dernier chiffre.

Le calcul des logarithmes effectué par le programme est réalisé par développement en série : il consiste à additionner les termes d'une suite dont les éléments successifs sont de plus en plus petits, et deviennent plus ou moins rapidement négligeables.

Le nombre pilote

La formule retenue est bien entendu celle qui converge le plus rapidement, autrement dit celle qui nécessite de retenir le minimum de termes, c'est-à-dire :
$\log x = 2 \times 0{,}43429\ldots \times [((x-1)/(x+1)) + (1/3)\ ((x-1)/(x+1))^3 + (1/5)\ ((x-1)/(x+1))^5 + \ldots]$ avec $0{,}43429\ldots = 1 / \ln 10$, ln désignant les logarith-

Vingt décimales pour un log
Programme en Basic standard
Auteur Pierre Ladislas Gédo
Copyright *LIST* et l'auteur

```
10 REM  ----------
11 REM  LOGARITHMES
12 REM  ----------
20 INPUT "N = ";N
30 IF N<1 OR N>10 THEN 20
40 V=1E10
50 W=1E15
100 A=INT((1+10*LOG(N)/LOG(10))/3)
102 X=10*A-3*INT(10*(LOG(N)/LOG(10)))
110 B=2^X
111 C=B/10^INT((LOG(B)/LOG(10)))
112 Y=INT((LOG(N/C)/LOG(10))/(LOG(1.04+.5)/LOG(10)))
120 D=1.04^Y
130 E=0
140 IF Y=5 THEN E=4/V
150 IF Y=6 THEN E=4.96/V
160 F=C*D
170 G=C*D-F+C*E
200 H=(N-F-G)/(N+F+G)
210 I=(1E-7)*(INT(H*(1E7)+.5))
212 J=(1E-12)*(INT(H*(1E12)+.5))-I
214 K=H-I-J
220 L=(1E-4)*(INT((N+F)*(1E4)+.5))
222 M=N+F-L
230 Z=N-F-I*L-I*M-J*L-G-G*H-J*M-K*L-K*M
232 O=Z/(N+F+G)
240 P=Z/(N+F+G)-O
250 Q=INT(O*W)/W
252 R=INT(K*W)/W
254 S=INT(J*V)/V
260 A=I+S
262 B=J-S+Q+R
264 C=(O-Q)+(K-R)+P
300 E=INT(I*I*(1E14)/3)/(1E14)
302 F=I*I-3*E
310 G=INT(E*(1E9)+.5)/(1E9)
312 L=E-G
320 M=INT((I+3*J)*(1E7)+.5)/(1E7)
322 P=I+3*J-M
330 Q=G*M
332 R=G*P+L*M
340 O=INT(Q*V)/V
342 S=INT(Q*W)/W
344 T=INT(R*W)/W
350 U=H*H*K+H*H*O+J*J*I+J*J*J/3+F*I/3+F*J+L*P
360 E=S-O+T
362 F=Q-S+R-T+U
400 G=INT(A*(1E4))/(1E4)
402 H=INT(B*(1E14))/(1E14)
404 I=A-G+H
406 J=B-H+C
409 Z=G*G*G*G
410 K=INT(Z*G*W/5)/W
412 M=Z*G/5-K
420 O=INT(Z*I*W)/W
422 P=Z*I-O
430 Q=2*G*G*G*I*I
432 R=INT(Q*W)/W
434 S=Q-R
440 T=2*G*G*Z+G*I*Z+Z*I*I/5+G*G*G*G*J+S*G*G*G*I*J
450 G=K+O+R
455 H=M+P+S+T
500 J=A+B+C
510 K=SGN(A+B+C)*ABS(A+B+C)^7
520 I=K/7+J*J*K/9+J*J*J*K/11
600 J=A+O
610 K=B+E+G
620 L=C+F+H+I
700 A=(1E-5)*INT(J*1E5)
707 B=J+-A
710 C=.86858
711 O=8.9638*(1E-6)
712 E=6.503*(1E-12)
713 F=6.5530226*(1E-16)
720 G=C+O
722 H=O=O+E
723 I=E+F
730 M=AO+BC
732 O=INT(M*V)/V
734 P=M-O
736 Q=INT((B*O+C*K)*W)/W
738 R=B*O+C*K-Q
740 S=INT((E*J+F*J+H*K+G*L)*W)/W
743 O=A*C+O
744 P=P+Q+S
746 Q=E*J+F*J+H*K+G*L-S+R
800 A=LOG(2-1/V)/LOG(10)
802 B=63981/W
804 C=.1942137389/W
810 O=LOG(1.04-1/V)
812 E=98780/W
814 F=.3548477218/W
820 G=(C*X+F*Y+Q)*V*V
822 H=INT(G+.5)/V*V
824 I=B*X+E*Y+P-INT(B*X+E*Y+P)
830 M=(INT(I*V))/V
832 R=A*X+O*Y+O
834 R=R-INT(R)+M
840 S=I+H-M
900 R=R*V
902 S=S*V*V
910 PRINT R,S
920 GOTO 20
```

VINGT DÉCIMALES POUR UN LOG

mes naturels (cette formule qui donne log x sera appelée *équation 1*).

La série converge certes rapidement, et d'autant plus rapidement que la variable (x-1)/(x+1) est voisine de zéro (par exemple pour x = 2, il suffit de retenir 10 termes de la suite pour obtenir log 2 avec 10 décimales). Mais il est possible d'accélérer la convergence par une autre procédure.

En effet, il est beaucoup plus facile de calculer le logarithme d'un nombre à partir d'un autre nombre très voisin qu'à partir de rien. Ainsi, au lieu de calculer le logarithme du nombre n, on calcule la différence des logarithmes de deux nombres voisins : n et n'. Ce dernier est appelé « nombre pilote » : sa structure est très simple, et son logarithme est parfaitement connu. Par exemple, si n = 2,03, on fixe n' = 2. Le logarithme de 2 est calculé une fois pour toutes avec 25 décimales. Il ne reste plus alors qu'à trouver la valeur de Δ = log 2,03 - log 2 et ajouter log 2 à Δ pour obtenir le logarithme de 2,03.

Bon résultat avec cinq termes

Sachant que log n - log n' = log (n/n'), on remplace x par n/n' dans l'*équation 1* : log (n/n') = (2/ln 10) × [(n - n') / (n + n') + (1/3) ((n-n')/(n+n'))^3 + (1/5) ((n-n')/(n+n'))^5 + ...]. Et en posant z = (n-n')/(n+n'), on obtient l'*équation 2* : Δ = (2/ln 10) × (z + (z^3/3) + (z^5/5) + ...). Le résultat recherché, log n, s'obtient alors par l'équation : log n = log n' + Δ.

Cette fois-ci, la variable z est très voisine de zéro et la série de l'*équation 2* converge encore plus rapidement que celle de l'*équation 1*. En l'occurence, pour n = 2,03 et n' = 2, on a z = 0,007 444 168 734 49, et il suffit de retenir seulement deux termes de l'*équation 2* pour obtenir Δ, soit log 2,03 avec 10 décimales, et de retenir cinq termes pour obtenir le résultat avec 20 décimales exactes.

Tout le problème consiste à trouver rapidement le nombre pilote n' en fonction de n, tout en veillant à ce que le logarithme de n' soit aisément calculable. Pour y arriver, on procède en deux étapes.

La série de Renard

Une première opération classe grossièrement toutes les valeurs possibles de n en 10 groupes (voir figure 1). A chacun de ces groupes est associé un nombre c, égal à une puissance de 2, désignée par X, positive ou négative (en ne retenant que la mantisse de c exprimée en notation scientifique). Les groupes ainsi définis correspondent approximativement à une progression géométrique de raison $10^{0,1}$ = 1,259. On notera que cette classification est très proche de la série dite de Renard, base de la normalisation internationale.

Une deuxième opération subdivise chacun des groupes précédemment définis en six intervalles plus fins, correspondant à une progression géométrique de raison 1,04, telle que n' = c × 1,04^Y. La figure 2 représente cette opération pour l'intervalle c = 2.

Analyse du programme

Le programme est subdivisé en dix parties, dont les huit premières sont consacrées au calcul de Δ.

Lignes 20 à 50 : initialisation.

La ligne 30 rejette les entrées non valables : comme pour une table de logarithmes ordinaire, seuls sont acceptés les nombres compris entre 1 et 10, borne supérieure exclue.

Lignes 100 à 170 : calcul du nombre pilote n'.

C'est aux lignes 100 à 110 que s'effectue le calcul du nombre pilote dont la valeur est n' = F + G. Le stockage sur deux mémoires est obligatoire, car n' peut comporter jusqu'à 14 chiffres significatifs.

Lignes 200 à 264 : calcul de z = (n-n')/(n+n').

Cette partie du programme, de même que toutes les suivantes jusqu'à celle portant les numéros de ligne 700 à 746, obéit à certains principes. Ainsi, le calcul s'effectue en multiprécision. Les résultats intermédiaires sont stockés en trois mémoires correspondant à trois tranches :

- les dix premières décimales pour la tranche 1,
- les cinq décimales suivantes pour la tranche 2,
- les décimales après la quinzième pour la tranche 3.

Les tranches relatives au résultat d'un calcul peuvent être de signes différents. Par ailleurs, afin d'économiser de précieux octets, les retenues d'une tranche sur la précédente ne sont pas effectuées horizontalement au fil des résultats intermédiaires, ce qui fait que, par exemple, la tranche 3 peut exceptionnellement déborder sur la quinzième décimale, et non commencer à la seizième. Ce n'est pas gênant, car les retenues sont effectuées verticalement, en fin de calcul.

Pour le calcul de z, on a : z = A + B + C ; A = tranche 1, B = tranche 2, C = tranche 3.

Lignes 300 à 362 : calcul de z^3/3.

z^3/3 = D + E + F ; D = tranche 1, E = tranche 2, F = tranche 3.

Lignes 400 à 455 : calcul de z^5/5.

z^5/5 = G + H. La tranche 1 disparaît, et on a : G = tranche 2, H = tranche 3.

Figure 1
Classement des yaleurs possibles de n

n	1	1,259	1,585	1,995	2,512	3,162	3,981	5,012	6,310	7,943	10
log n	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
c	1	1,25	1,6	2	2,5	3,2	4	5	6,4	8	

Figure 2
Un découpage plus fin, en six intervalles

n	1,995	$2\times1{,}04^{0,5}$ 2,040	$2\times1{,}04^{1,5}$ 2,121	$2\times1{,}04^{2,5}$ 2,206	$2\times1{,}04^{3,5}$ 2,294	$2\times1{,}04^{4,5}$ 2,386	$2\times1{,}04^{5,5}$ 2,481	2,512
log n	0,3							0,4
n'	2	$2\times1{,}04$	$2\times1{,}04^2$	$2\times1{,}04^3$	$2\times1{,}04^4$	$2\times1{,}04^5$	$2\times1{,}04^6$	

Lignes 500 à 520 : calcul de $x^7/7 + z^9/9 + z^{11}/11$.

Calcul en simple précision, car les tranches 1 et 2 disparaissent et le résultat est stocké en mémoire I.

L'écriture de la ligne 510 est particulière : on aurait été tenté d'écrire K = (A + B + C) ∧ 7, mais A + B + C peut être négatif. Et certaines machines refusent d'élever à quelque puissance que ce soit un nombre négatif. On se serait exposé dans ce cas à un message d'erreur. L'expression retenue est donc équivalente à K = (A + B + C) ∧ 7.

Lignes 600 à 620 : calcul de $z + z^3/3 + \ldots + z^{11}/11$.

Le résultat est stocké de la façon suivante : D = tranche 1, E = tranche 2, F = tranche 3.

Lignes 700 à 746 : multiplication en multiprécision.

Calcul de $\Delta = (2/\ln 10) \times (z + z^3/3 + \ldots + z^{11}/11)$, Δ = O + P + Q, O = tranche 1, P = tranche 2, Q = tranche 3.

Les lignes 710 à 713 stockent la valeur de 2 / ln 10 en quatre tranches (C, D, E, F) totalisant 23 décimales.

Lignes 800 à 840 : calcul de log n = log n' + Δ.

Les lignes 800 à 804 stockent la valeur de log 2 en trois tranches (A, B, C) totalisant 25 décimales ; les lignes 810 à 814 stockent la valeur de log 1,04 en trois tranches (D, E, F) totalisant également 25 décimales. Les valeurs précises, avec 20 ou 25 décimales, des constantes particulières telles que 1/ln 10, log 2, figurent dans les *Tables Numériques Universelles* de Marcel Boll (éditions Dunod).

Lignes 900 à 910 : affichage.

La version de base présente le résultat en deux tranches de dix chiffres (séparés par un point sans imprimante, et sur deux lignes si l'imprimante est connectée). Selon la convention habituelle, si une tranche comporte moins de 10 chiffres, compléter à gauche par le nombre de zéros nécessaires.

Exemple : pour n = 1,2, l'afficheur indique : 791812460. 4762482772 ce qui se traduit par log 1,2 = 0,07918124604762482772.

La variante 1 est réservée à l'utilisation sans imprimante. Elle affiche le résultat en quatre tranches de cinq chiffres ; dans notre exemple : 7918.12460.47624.82772.

La variante 2 est réservée à l'utilisation avec imprimante. Comme la variante 1, elle imprime le résultat en quatre tranches de cinq chiffres, mais sur quatre lignes.

Dans le cas des variantes 1 et 2, si une tranche comporte moins de 5 chiffres, compléter à gauche par le nombre de zéros nécessaires.

L'espace compris entre 1 et 10 est ainsi subdivisé en 60 micro-intervalles et autant de nombres pilotes.

Quelques points doivent être soulignés :

- le pilotage de n vers le nombre pilote le plus proche (par excès ou par défaut) est effectué automatiquement par le programme ;
- le nombre pilote n' associé à n est exclusivement un multiple de 2 et de 1,04, c'est-à-dire de la forme $n' = 2^X \times 1{,}04^Y \times 10^K$ (X compris entre −3 et 6, Y compris entre 0 et 6, K indifférent), expression dont le logarithme peut être calculé par addition de multiples de log 2 et de log 1,04 que l'on calcule une fois pour toutes avec la précision voulue ;
- le nombre n ne diffère du nombre pilote que de 2 % dans le pire des cas (exactement de $\sqrt{1{,}04}$ %), cette différence pouvant être positive ou négative ; ainsi, les six premiers termes de l'*équation 2* suffiront pour donner Δ avec 25 décimales exactes (dans le cas le plus défavorable, le terme est de l'ordre de 10^{-28}).

Pour avoir une précision encore plus grande, il suffit de rappeler la variable G : elle indique, après la virgule, les décimales qui ont été omises après la vingtième pour des raisons de sécurité... Bien entendu, elles ne sont pas garanties, mais les deux ou trois premières ont toutes les chances d'être exactes. Ceci pour le cas où vingt décimales ne suffiraient pas !

Pierre Ladislas GEDO

DIALOGUE AVEC LOGO

COMME avec d'autres langages, en Logo, l'entrée des données se fait à partir des périphériques et, en particulier, à partir du clavier. Mais ici, il n'est pas toujours nécessaire de passer par l'intermédiaire d'une variable. Un véritable dialogue s'instaure alors, à condition de ne pas commettre d'erreurs.

Le langage Logo dialogue avec le clavier par l'intermédiaire d'un « tampon » : dès qu'une information demandée a été acquise, une valeur est retournée à la procédure en cours d'exécution. Lorsque Logo rencontre une primitive de dialogue, le traitement est suspendu jusqu'à l'obtention de l'information désirée. Le nom de la primitive précise la nature de l'information : LISCAR pour un caractère, LISMOT pour un mot, LISLISTE ou LISLIGNE pour une liste.

Pas d'intermédiaire

Pour LISCAR, le contenu du tampon est retourné dès la frappe d'un **caractère ; et pour les autres primitives**, il est retourné dès la frappe du « retour-chariot ». Il convient alors d'indiquer quoi faire avec l'information retournée. Par exemple, on peut l'affecter à une variable comme dans un langage classique. Mais on peut aussi, éventuellement, la traiter directement :

```
POUR DIALOGUE
DONNE ''REPONSE LISCAR
SI :REPONSE = ''A ALORS...
```

Ou bien :

```
POUR DIALOGUE
SI LISCAR = ''A ALORS...
```

Mais il est toujours possible de rencontrer des « bogues », de petites erreurs qui interrompent le bon déroulement d'un programme. Ainsi, en utilisant directement le contenu du tampon sans variable intermédiaire, ce dernier ne sera traité qu'une seule fois :

```
POUR DIALOGUE
SI LISCAR = ''A ALORS...
SI LISCAR = ''B ALORS...
```

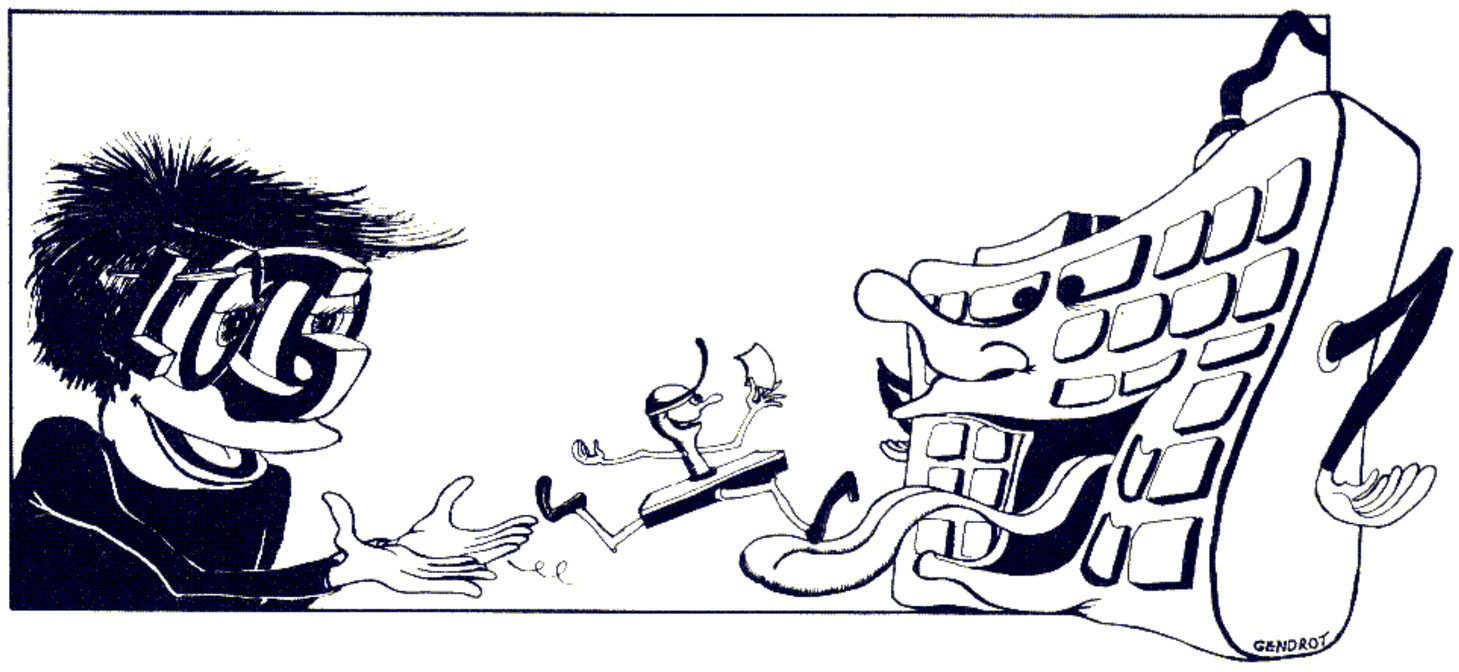
La troisième ligne ne sera exécutée que si l'on tape à nouveau un caractère. Sous cette forme, il est donc impossible de comparer successivement à A et à B, un caractère tapé.

Les primitives LISCAR, LISMOT, LISLISTE suspendent le traitement et attendent une information en provenance du clavier. Pour comparer un caractère lu à plusieurs, il est nécessaire de passer par une variable intermédiaire :

```
POUR DIALOGUE
DONNE ''REP LISCAR
SI :REP = ''A ALORS...
SI :REP = ''B ALORS...
```

La nature de l'information obtenue est fixée par la primitive. Ainsi LISMOT retourne un MOT. C'est pourquoi cette primitive n'est implantée que dans les versions capables de distinguer deux types de caractères « espace » : le séparateur de mots et le caractère ordinaire.

Par exemple, ''CHAUVE SOURIS n'est pas, en principe, un mot Logo. Il le deviendra si l'on crée un caractère spécial (en général, CTRL Q) qui indique que le caractère suivant n'est pas un séparateur. Ainsi, ''CHAUVE\SOURIS devient un mot. Pour LISMOT, le



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.

caractère de fin d'information est donc le retour-chariot.

Il ne faut pas oublier que l'information obtenue par LISMOT est un mot. Donc Logo introduit automatiquement des caractères spéciaux CTRL Q devant les séparateurs. Par exemple, 3 + 5 sera lu comme ''3\ \ + \ 5

Erreurs dues à la primitive LISLISTE	
Faux	**Juste**
POUR AVANCER AVANCE LISLISTE	POUR AVANCER AVANCE PREMIER LISLISTE
POUR COMPARER AFR [TAPER OUI OU NON] SI LISLISTE = ''OUI	POUR COMPARER AFR [TAPER OUI OU NON] SI LISLISTE = [OUI]
POUR NOMMER AFR [TAPER LE NOM CHOISI] DEFINIS PREMIER LISLISTE :PROC	POUR NOMMER AFR [TAPER LE NOM CHOISI] DONNE "N PREMIER LISLISTE SI NON :N = "DONNE :N :PROC
Pour la procédure NOMMER, il était nécessaire de donner à l'utilisateur la possibilité de ne pas conserver son travail. Il a donc fallu créer une variable intermédiaire, contenant un mot, qui teste une liste éventuellement vide.	

La confusion est facile

On doit alors être sûr de récupérer un mot pour que l'analyse du tampon soit simple.

La primitive la plus couramment utilisée est LISLISTE. L'application des primitives de traitement des listes au contenu du tampon doit donc être faite avec attention. Les « bogues » sont nombreuses car il est facile de confondre listes, mots et nombres. Et LISLISTE retourne une liste (voir encadré ci-dessus).

Les « bonnes » versions Logo ont une primitive de pas à pas. Les « meilleures » permettent une intervention sur l'instruction qui va s'exécuter si une bogue est perçue auparavant. Car sans bogue, l'exécution de telles procédures ne servirait plus à rien !

Robert DAGUESSE

Les procédures A, B, C, D et E

Cinq procédures à méditer étaient proposées dans le numéro 2 de *LIST*.

```
POUR A :P
COPIEDEF MOT ". :P :P
DONNE "T TEXTE :P
DONNE "V (LISTE PREMIER
:T)
DONNE "V (LISTE :V
(PHRASE [ ENTREE DANS ] :P))
B PREMIER :T
C SAUFPREMIER :T
DEFINIS :P :V
FIN
POUR B :W
SI :W = [ ] STOP
DONNE "V LISTE :V (LISTE
(AFR [ LA VALEUR DE ] PREMIER
:W "EST) (MOT ": PREMIER :W))
B SAUFPREMIER :W
FIN
POUR C :X
SI :C = [ ] STOP
DONNE "V LISTE :V (LISTE
"AFR PREMIER :X)
DONNE "V LISTE :V D
DONNE "V LISTE :V PREMIER
:X
C SAUFPREMIER :X
FIN
POUR D
DONNE "Z LISLISTE
RETOURNE "
FIN
POUR E :Q
COPIEDEF :Q MOT ". :Q
EFFACE MOT ". :Q
FIN
```

Les quatre premières procédures A, **B, C et D simulent un pas à pas.**

Dans A, COPIEDEF crée une procédure dont le nom est celui de la procédure à exécuter pas à pas (il est précédé d'un point). Cette procédure sera la sauvegarde de la précédente, maintenant modifiée. Par exemple, A "CARRE **crée la procédure-copie .CARRE**

Par TEXTE, la procédure d'origine est alors transformée en une liste nommée T. La première liste de T est celle des paramètres. Le but consiste donc à créer une nouvelle procédure par l'intermédiaire de V, variable globale, qui imprimera les commentaires nécessaires à l'exécution de chaque ligne et attendra un retour-chariot tapé au clavier pour exécuter la ligne suivante, grâce à la procédure D.

Par exemple :

```
POUR ANGLE :L :N
AVANCE :L
DROITE :N
AVANCE :L
FIN
```

Les procédures A, B, C et D créeront la procédure :

```
POUR ANGLE :L :N
AFFICHE [ENTREE DANS
ANGLE]
(AFFICHE [LA VALEUR DE :L
EST] :L)
(AFFICHE [LA VALEUR DE :N
EST] :N)
AFFICHE [AVANCE :L]
```

Appel de D pour attente d'un retour-chariot :

```
AVANCE :L
AFFICHE [DROITE :N]
```

Appel de D :

```
DROITE :N
AFFICHE [AVANCE :L]
```

Appel de D :

```
AVANCE :L
FIN
```

L'exécution donnera :

```
A "ANGLE
ANGLE 10 60
ENTREE DANS LA PROCE-
DURE ANGLE
LA VALEUR DE :L EST 10
LA VALEUR DE :N EST 60
AVANCE :L
```

Attente d'un retour-chariot.

Dès qu'il est frappé, la ligne affichée s'exécute et la suivante apparaît :

```
DROITE :N
```

Attente d'un retour-chariot pour exécuter cette ligne. Et ainsi de suite jusqu'à la fin.

C'est la primitive LISLISTE qui permet l'attente d'un retour-chariot et renvoie un caractère vide qui n'a aucune influence. Ceci est une astuce pour créer une exécution conditionnelle soumise à une frappe. Seul le caractère retour-chariot fait continuer l'exécution de la procédure.

Les procédures B et C ont été créées pour séparer les paramètres et les instructions. **C'est pourquoi leurs paramè**tres respectifs sont la première liste de T et tout sauf la première liste de T.

Alors que B utilise des ordres d'affichage directs, C introduit ces ordres sous forme de mots, AFR étant dans ce cas précédé de guillemets. B et C sont dites récursives.

Quant à la procédure E, elle transforme à nouveau la procédure initiale en ce qu'elle était et détruit la copie. C'est là que se termine le pas à pas.

MESURER LE BASIC

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.

Les dix tests présentés ici permettent de se faire une première idée de la rapidité de son ordinateur. Mais certaines adaptations peuvent être nécessaires, en particulier pour la lecture, l'écriture de fichiers (tests 9 et 10) ou les tracés graphiques (test 8).

Certains ordinateurs ne possèdent pas d'instructions graphiques sur leur modèle de base (Commodore 64, Vic 20, TRS-80 Modèle 1,...). L'exécution du test 8 par l'utilisation d'un programme Basic aurait pulvérisé des records de lenteur.

Par curiosité, sur le C.64, ce test a été effectué en utilisant la cartouche *Tool 64*. Le nombre de tracés a été réduit à 1 000 (au lieu de 10 000). Le temps d'exécution, 731 secondes pour 10 000, laisse penser que l'optimisation reste encore à faire...

Le nombre de points de la résolution graphique a, lui aussi, son importance. Ainsi, sur le DAI 48K le test 8 passe de 2 518 secondes pour une résolution de 336×256 points, à 706 secondes pour une résolution de 72×65 points ; l'Electron passe de 1 276 secondes en 640×256 points, à 299 secondes en 160×256 points ; le Coco 2 passe de 880 secondes en 256×192 points à 507 secondes en 128×96 points. En revanche, le QX-10 reste à 67 secondes que la résolution soit de 320×200 points ou de 640×400 points.

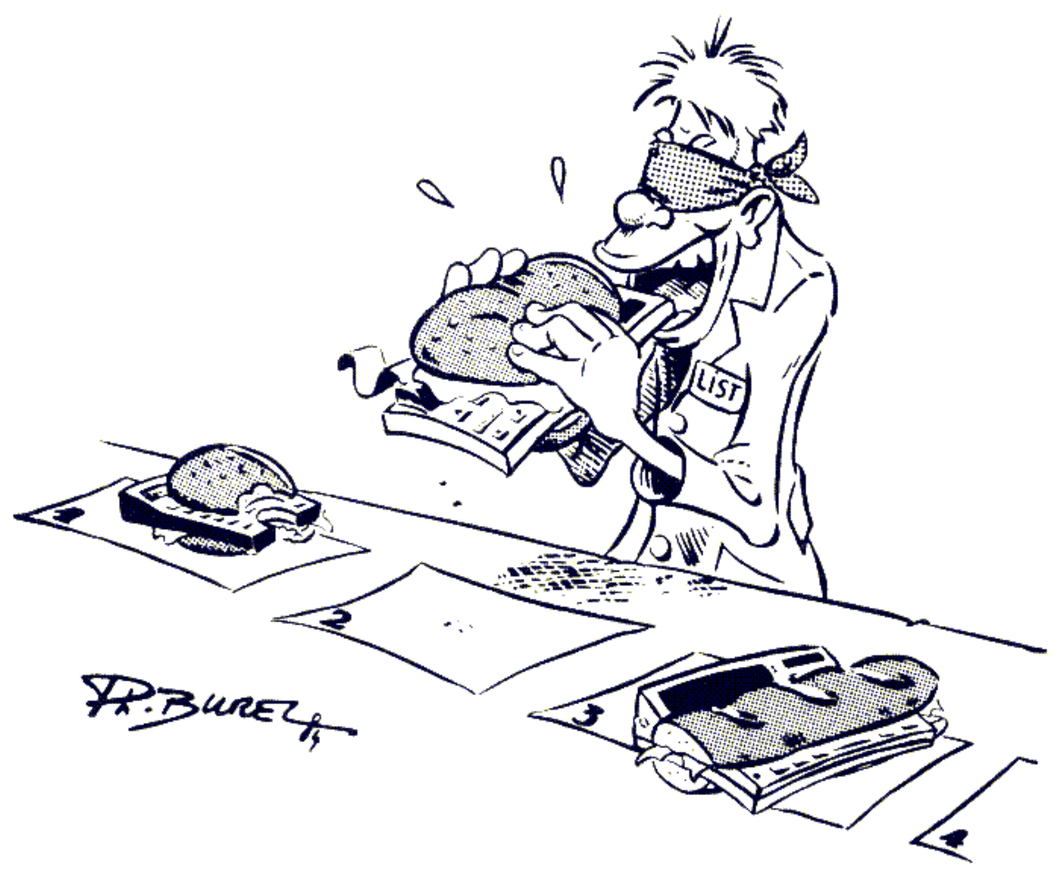
Les boucles FOR...NEXT sont présentées ici sous la forme FOR...NEXT I. Or, la présence du nom de la variable (I, ici) après NEXT a pour effet de modifier sensiblement les temps d'exécution des boucles sur les ordinateurs qui acceptent les deux formes NEXT ou NEXT I. C'est le cas par exemple du Commodore 64, du QX-10, du Vic 20, etc. Et le gain de temps est important quand le nom de la variable n'apparaît pas après NEXT.

Il reste que les comparaisons sont difficiles à faire. Mais certaines parentés existent. Ainsi, MO5 et Coco 2 disposent d'un Basic Microsoft sur processeur 6 809 ; BBC et Electron sont de la même famille, ainsi que Commodore 64 et Vic 20.

Quant au MSX testé ici, le Yeno, il devrait donner les mêmes résultats que ses « confrères » MSX : le Basic est « standardisé ».

Il ne faut pas oublier que ces tests ne peuvent donner qu'une *première* idée des qualités de son ordinateur : gagner en rapidité peut parfois se traduire par la perte de sophistication.

LIST



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) ;
      « LISTEST » ; 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
```

Résultats des tests de 16 Basic

Ordinateurs (Classés par ordre alphabétique)	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Test 6**	**Test 7**	**Test 8**	**Test 9**	**Test 10**
Apple IIe	13	43	65	136	93	479 (9 chiffres)	130	484 (280 × 160)	404 (disquette)	710 (disquette)
Atari 600 XL	22	88	78	271	204	2 040 (9 chiffres)	225	1 766 (320 × 192)		
BBC	7	15	32	41	57	405 (9 chiffres)	114	239 (160 × 256)	1 944 (cassette)	1 944 (cassette)
Colour computer 2	14	47	69	173	116	598 (9 chiffres)	168	880 (256 × 192)	998 (cassette)	998 (cassette)
Commodore 64	15	44	75	155	105	295 (9 chiffres)	200	731 (320 × 200)	394 (disquette)	323 (disquette)
DAI 48K	5	30	21	87	87	799 (6 chiffres)	427	2 518 (336 × 256)	10 500 (disquette)	9 500 (disquette)
Electron	9	20	43	55	64	578 (9 chiffres)	191	299 (160 × 256)		
HP-71 B	95	206	97	266	262	823 (15 chiffres)	1 146		420 (mémoire)	200 (mémoire)
HX-20	25	71	152	209	179	558 (9 chiffres)	1 973	1 234 (120 × 32)	2 460 (cassette)	2 400 (cassette)
MO 5	16	45	87	135	122	376 (6 chiffres)	295	1 430 (320 × 200)	940 (cassette)	940 (cassette)
QX-10	21	56	85	146	99	305 (16 chiffres)	499	67 (320 × 200)	132 (disquette)	126 (disquette)
TO 7	18	46	100	138	124	417 (6 chiffres)	222	1 037 (320 × 200)		
TRS-80 Modèle 1	27	86	111	213	166	516 (7 chiffres)	263			
Vic 20	12	37	63	130	88	445 (9 chiffres)	173		369 (disquette)	310 (disquette)
Yeno MSX	20	41	60	108	185	1 937 (16 chiffres)	139	1 165 (256 × 192)	1 863 (cassette)	1 863 (cassette)
ZX Spectrum	42	106	115	195	132	1 180 (8 chiffres)	243	610 (256 × 176)		

Les temps sont exprimés en secondes.

BASIC APPELLE LANGAGE-MACHINE

LES langages évolués de l'Apple, qu'ils soient interprétés (Basic Applesoft) ou compilés (Pascal UCSD) fournissent un excellent environnement pour l'obtention de programmes-sources clairs, lisibles et, parfois structurés. Cependant, certaines applications nécessitent une extrême rapidité d'exécution, ou un accès direct à des parties du système normalement interdites ou difficilement accessibles. Il est alors indispensable d'écrire des sous-programmes en Assembleur qui seront appelés à partir du langage évolué. Avec Basic Applesoft, un certain « bricolage » est de rigueur.

L'environnement Applesoft est beaucoup moins « confortable » que celui du Pascal : pas d'éditeur, pas d'assembleur (le « mini-assembleur » des premiers Apple II n'est plus disponible avec l'Apple II+ muni de l'« auto-start »), et encore moins d'éditeur de liens (« linker ») ! Avant toutes choses, nous allons donc nous trouver dans l'obligation d'acquérir un bon assembleur (Big Mac/Merlin, S-C Macro-Assembleur, Lisa 2.5 ou DOS Toolkit, par ordre de préférence). Ceci étant réalisé, travaillons sur un exemple. On fait silence, on prend ses cahiers...

Soit à « nettoyer » l'écran « TEXTE ». On pourrait naturellement écrire le programme Applesoft qui suit :

```
10 FOR I=1 TO 24
20 PRINT
30 NEXT I
```

Evidemment, cela n'est pas follement rapide. Il est de loin préférable d'utiliser HOME, l'un des innombrables sous-programmes écrits par Steve Wozniak — un des créateurs de l'interpréteur et du Moniteur Apple.

Pour éviter les interférences

Or, 10 CALL −936 appelle une partie de code machine du moniteur appelé HOME et situé à l'adresse $FC58. Alors pourquoi pas un CALL 64600 ? En vérité, cela est parfaitement possible et le CALL −936 est une survivance des temps déjà anciens où le seul langage disponible sur l'Apple était l'Integer Basic qui ne pouvait manipuler que des nombres compris entre −32768 et 32767. La valeur −936 se réfère donc à un déplacement à partir du sommet de la mémoire (un nombre positif se référant à la base) : 64 Ko = 64 * 1024 octets = 65 536 octets, et 65 536 − 936 = 64 600 = $FC 58.

Oui, mais Steve Wozniak a cherché à faire court plutôt que rapide, enfin relativement... On peut faire mieux, mais plus long, c'est l'éternel problème (programme 1).

Bon, mais où placer ce programme 1 pour éviter les interférences avec le DOS ou le programme Basic résident ?

Il n'est pas question de le placer en page zéro : elle est presque entièrement

utilisée par les sous-programmes du Moniteur, de l'interpréteur Applesoft et du DOS. Plusieurs solutions sont possibles.

• Le placer dans la « carte langage » (extension mémoire 16 Ko). Par des PEEK ou des POKE dans la zone 49280-49295 ($C080-$C08F), on peut contrôler l'emplacement mémoire (carte langage ou Apple) sur lequel on lira, ou écrira, dans la zone $D000-$FFFF. L'avantage est naturellement de disposer de 12 à 16 Ko supplémentaires, mais la procédure est assez délicate et reste du domaine du programmeur averti.

• Le placer dans l'espace « utilisateur » de la page trois. On dispose d'un emplacement de 207 octets dans la page de $300 à $3CF (768-975) ce qui permettra d'implanter de courts programmes (tout le monde utilise cette zone d'où l'abondance des CALL 768 ($300) que l'on rencontre dans les programmes Basic).

• Le placer dans l'espace libre. On implante le programme au-dessous du HIMEM et au-dessus du programme Basic, puis on le protège en modifiant le HIMEM.

Il existe naturellement bien d'autres emplacements possibles : dans un « buffer » du DOS, entre le DOS et ses « buffers », à la fin du programme Basic, dans la page deux, etc. Mais ces méthodes sont vraiment trop particulières et trop compliquées à mettre en œuvre pour être explicitées ici.

L'algorithme de Lucas

La troisième méthode est de loin la plus simple à utiliser. Dans l'exemple précédent, notre Apple étant de 48 Ko, son HIMEM (plus haute mémoire disponible) sera, avec le DOS en place, de $9600 (38400). Le programme précédent étant de 32 ($20) octets devra donc être implanté en $9600-$20 = $95E0 (38368) et le HIMEM modifié dans le programme Applesoft (sinon, les chaînes de caractères étant stockées à partir du HIMEM, le programme serait détruit lors du premier stockage). On réassemble le programme avec une origine appropriée ($95E0) et on sauve sous le nom HOME :

```
5 PRINT CHR$(4)« BLOAD HOME »
10 HIMEM : 38368
20 CALL 38368 : REM OU CALL
   - 27168
```

Les premiers Apple étaient fournis

Programme 1

```
**************************
*                        *
* NETTOYAGE RAPIDE DE    *
*    L'ECRAN TEXTE       *
*                        *
**************************
*

        LDA CARACT
        LDY #$77
ETIQ    STA $400,Y : lignes 0  8 16
        STA $500,Y : lignes 2 10 18
        STA $600,Y : lignes 4 12 20
        STA $700,Y : lignes 6 14 22
        STA $480,Y : lignes 1  9 17
        STA $580,Y : lignes 3 11 19
        STA $680,Y : lignes 5 13 21
        STA $780,Y : lignes 7 15 23
        DEY
        BPL ETIQ
        RTS
CARACT  ASC " "
```

Programme 2

```
**********
* GENERE *
**********
*
* génére un son dans le haut-parleur de l'Apple
*
* P%=0..255 Période
* D%=0..255 Durée
*
WAIT EQU $FCA8 ; routine d'attente
DATA EQU $D995 ; déplace le pointeur TXTPTR
*              ; en fin d'instruction
*
*
GENERE    LDA PERIOD
          BNE TONE
*
* une période égale à zéro génére un son blanc
*
          LDA DURATN
          JSR WAIT
          JMP DATA
*
TONE      LDA $C030 ; excite le haut-parleur
TONE1     DEY
          BNE TONE2
          DEC DURATN
          BEQ TONE3
TONE2     DEX
          BNE TONE1
          LDX PERIOD
          JMP TONE
TONE3     JMP DATA
*
PERIOD    DS 1 ; P%
DURATN    DS 1 ; D%
*
```

Programme 3

```
**********************************
*                                *
*  PASSAGE DES PARAMETRES EN     *
* FIXANT LEUR PLACE EN MEMOIRE   *
*                                *
**********************************
*
*
VARTAB EQU $69 ; pointeur début table variables
*

        LDY #$03
        LDA (VARTAB),Y
        STA DURATN
        LDY #$08
        LDA (VARTAB),Y
        STA PERIOD
*
        JMP GENERE
```

BASIC APPELLE LANGAGE-MACHINE

avec un manuel technique qui contenait notamment l'algorithme de Lucas permettant de générer des sons de période et de durée variables. Cet algorithme étant paramétré par deux variables (durée et période), le problème réside dans le passage de ces deux paramètres au programme machine. Rien n'est prévu à cet effet (programme 2).

Une des méthodes les plus sûres

En fait il existe au moins cinq façons d'opérer.

• Par l'utilisation de la fonction USR car USR (expression) passe « expression » à un sous-programme en langage-machine. L'argument « expression » est évalué et placé dans l'accumulateur des flottants (adresse $0A à $0C) et un saut à l'adresse $0A est alors effectué (cette zone doit donc contenir l'adresse effective du programme assembleur considéré). L'expression étant mise automatiquement sous forme de réel, il nous faudra repasser en entier le cas échéant, et on ne peut passer qu'un paramètre à la fois. USR n'est donc pas toujours très pratique.

• Par des POKE directs des valeurs des paramètres en mémoire : 10 INPUT « DUREE » ; D :INPUT « PERIODE » ; P : POKE 805, P : POKE 806, D : CALL 768 : REM $325 = 805 ET $326 = 806 (adresse de PERIOD et DURATN)

Cette méthode est une des plus utilisées et des plus sûres. Elle nécessite toutefois une certaine gymnastique lors du passage de nombres entiers plus grands que 255 (exigeant deux octets) ou de nombres réels.

• Le passage des paramètres via des variables « Basic » dont on a fixé les emplacements en mémoire :
0 P% = 0 : D% = 0 : REM LES VARIABLES P ET D FIGURERONT EN PREMIERE POSITION DANS LA TABLE DES VARIABLES
100 INPUT « DUREE » ; D% : INPUT « PERIODE » ; P% : CALL 768. A la condition d'ajouter au programme machine l'en-tête du programme 3.

Expliquons-nous : P% et D% étant les deux premières variables à figurer dans la table des variables (dont le début est indiqué par VARTAB $69-$6A), on atteindra les valeurs numériques de P% et D% par un adressage post-indexé : 3 pour P% et 8 pour D% (chaque variable, entière ou réelle, occupant cinq octets).

Cette méthode, quoique plus intéressante que la première, reste un peu primitive si l'on utilise de nombreux sous-programmes machine (et donc de nombreux paramètres à passer).

• L'utilisation de routines de recherche des variables de l'Applesoft.

Quand Applesoft rencontre l'ampersand

Cette méthode ressemble à la précédente mais l'utilisation des sous-programmes internes permet d'éviter d'avoir à donner une place fixe aux variables-paramètres (programme 4).

L'appel se fait alors simplement par :
100 INPUT P%, D% : CALL 768

Cette façon d'opérer est plus élégante que la précédente puisque le programme va chercher la valeur de la variable où elle se trouve (et peut même créer la variable dans la table des variables si elle n'existait pas auparavant). Mais ce n'est pas encore l'idéal puisque les variables paramètres sont fixées une fois pour toutes.

• L'utilisation de l'« ampersand » : lorsque l'Applesoft rencontre le signe « & » (autrement dit, l'« ampersand ») l'interpréteur opère un saut à l'adresse $3F5 qui lui indique alors l'adresse à laquelle il doit se rendre :
03F5 4C 00 03 JMP $300 ; $300 = 768 (programme 5).

De cette façon, on pourra obliger l'interpréteur à se dérouter vers l'adresse de départ du programme et à interpréter tout ce qui suivra le signe « & » :
10 CALL 768:REM ACTUALISATION DU & A UN JMP $300
20 INPUT P%,D%
30 & T (P%,D%)

Ainsi, on pourra passer les paramètres par valeur ou par variable et ce, avec les variables de notre choix.

Cette dernière méthode est de très loin la meilleure. Pour s'en convaincre, comparer POKE 805,255 :POKE 806,255: CALL 768 avec & T (P%,D%).

Son seul inconvénient est de nécessiter la présence d'un programme auxilliaire destiné à « gérer » les diverses commandes de l'« ampersand » dans le cas de programmes multiples. Un moindre mal...

Philippe FRANÇOIS

Programme 4

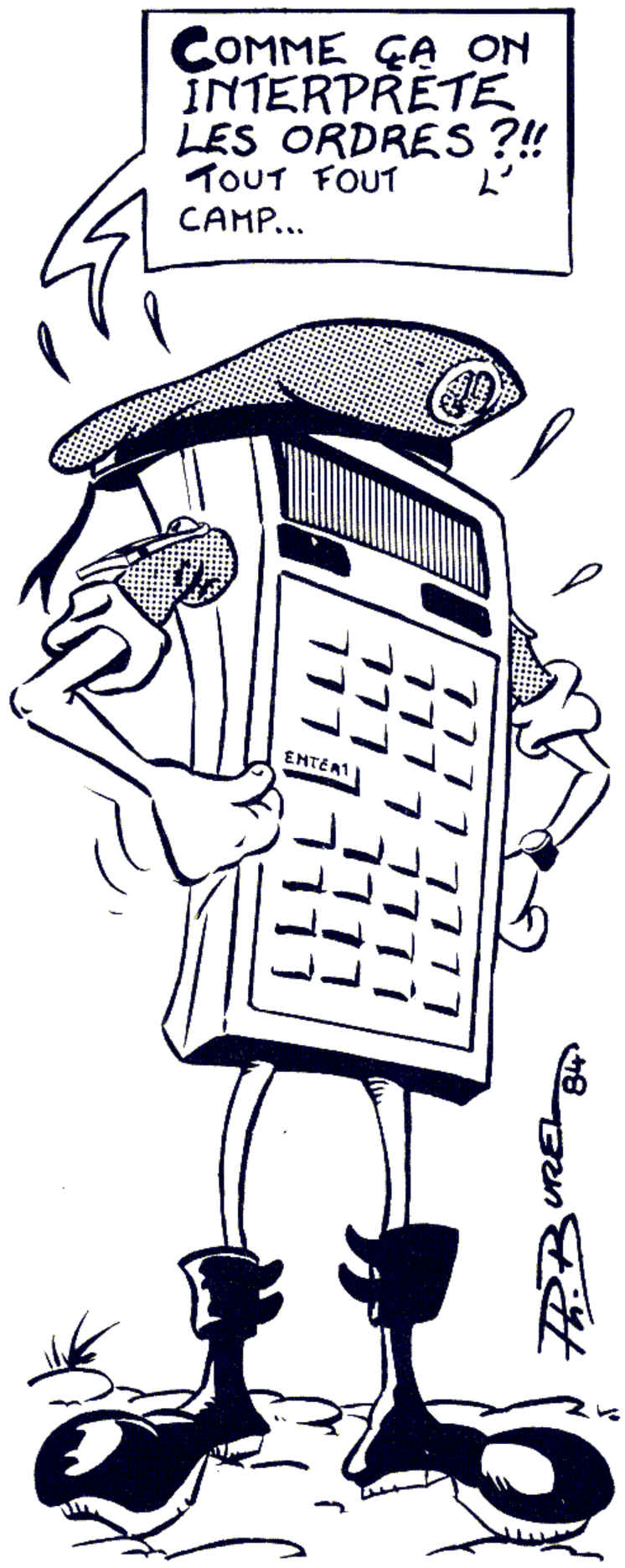
```
*****************************************
*                                       *
*   PASSAGES DES PARAMETRES PAR         *
* ROUTINES APPLESOFT DE RECHERCHE       *
*          DE VARIABLES                 *
*                                       *
*****************************************
*
TXTPTR    EQU $B8
PTRGET    EQU $DFE3
*
*
*
          JSR FINDP
          LDY #$01
          LDA (TXTPTR),Y
          STA PERIOD
          JSR FINDD
          LDY #$01
          LDA (TXTPTR),Y
          STA DURATN
*
          JMP GENERE
*
FINDP     JSR FIND
          ASC 'D%'
          HEX 00
*
FINDD     JSR FIND
          ASC 'P%'
          HEX 00
*
FIND      PLA
          STA TXTPTR
          PLA
          STA TXTPTR+1
          INC TXTPTR
          BNE SUITE
          INC TXTPTR+1
SUITE     JSR PTRGET
          STA TXTPTR
          STY TXTPTR+1
          RTS
```

Programme 5

```
*****************************************
*                                       *
*   PASSAGES DES PARAMETRES PAR         *
* ROUTINES APPLESOFT DE RECHERCHE       *
*          DE VARIABLES                 *
*                                       *
*****************************************
*
TXTPTR    EQU $B8
PTRGET    EQU $DFE3
CHRGET    EQU $B1
FRMNUM    EQU $DD67
GETADR    EQU $E752
AMPV      EQU $3F5
SYNTAX    EQU $DEC9
CHKOPN    EQU $DEBB
CHKCLS    EQU $DEB8
CHKCOM    EQU $DEBE
*
*
*         Mise à jour de l'ampersand
*
          LDA #>DEPART
          STA AMPV+2
          LDA #<DEPART
          STA AMPV+1
          RTS
*
DEPART    CMP #'T' ; on vérifie la présence du "T"
          BEQ DEP1
          JMP SYNTAX      ; "SYNTAX ERROR"
*
DEP1
          JSR CHRGET      ; on fait avancer le pointeur
          JSR CHKOPN      ; on vérifie la présence de "("
          JSR FRMNUM      ; récupération de la variable
          JSR GETADR      ; et de son adresse
          LDY £$01
          LDA (TXTPTR),Y
          STA PERIOD
          JSR CHKCOM      ; on vérifie la présence de ","
          JSR FRMNUM
          JSR GETADR
          LDY #$01
          LDA (TXTPTR),Y
          STA DURATN
          JSR CHKCLS      ; on vérifie la présence de ")"

          JMP GENERE
```

PRENEZ LA BONNE CORRESPONDANCE

UNE HP-41 C, comme tout ordinateur, possède trois composantes essentielles : un clavier, un écran et de la mémoire. Lorsqu'on fait l'étude de sa structure intime, on donne les ordres au clavier et on consulte un résultat sur l'afficheur. Et ce que l'on voit est toujours intercepté par la HP-41 C avant l'affichage.



Parfois, de simples codes de la HP-41 C sont visualisés comme s'il s'agissait de fonctions, normales ou non, d'un programme.

Une petite expérience va en faire la démonstration. Effectuons un Memory Lost (presser ←, allumer la HP) puis la séquence de pressions de touches récapitulée dans le tableau ci-contre.

Qu'avons-nous fait ? En employant la "bogue" (erreur de programmation) fondatrice de la programmation synthétique :
(PRGM) CAT 1 R/S (ALPHA) ← nous avons positionné la HP sur une zone n'ayant aucun rapport avec la zone des programmes. Mieux, nous nous sommes placés sur les registres d'état examinés théoriquement dans *LIST* 2. Nous consultions les registres comme s'il s'agissait d'un programme, nous en avons modifié les contenus en programmant, simplement.

Une seconde expérience va illustrer plus encore ce principe. Après avoir

Faire	Lire	
(PRGM)	00	REG 46
CAT 1 immédiatement suivi de R/S	.END.	REG 46
(ALPHA)		
←	4094	RCL 01
(ALPHA)		
GTO.001, attendre	01	CL Σ
SST	02	LBL 01
SST	03	RCL 12
←	02	LBL 01
STO 12	03	STO 12
Les indicateurs 2, 3 et 4 s'affichent		
RCL 12	04	RCL 12
STO 12	05	STO 12
STO 12	06	STO 12
L'indicateur USER s'affiche		
(PRGM)	0.0000	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0 NULL 00	1 LBL 00 01	2 LBL 01 02	3 LBL 02 03	4 LBL 03 04	5 LBL 04 05	6 LBL 05 06	7 LBL 06 07	8 LBL 07 08	9 LBL 08 09	10 LBL 09 10	11 LBL 10 11	12 LBL 11 12	13 LBL 12 13	14 LBL 13 14	15 LBL 14 15	0
1	16 0 16	17 1 17	18 2 18	19 3 19	20 4 20	21 5 21	22 6 22	23 7 23	24 8 24	25 9 25	26 . 26	27 EEX 27	28 CHS 28	29 GTO∝ 29	30 XEQ ∝ 30	31 BIDON 31	1
2	32 RCL 00 32	33 RCL 01 33	34 RCL 02 34	35 RCL 03 35	36 RCL 04 36	37 RCL 05 37	38 RCL 06 38	39 RCL 07 39	40 RCL 08 40	41 RCL 09 41	42 RCL 10 42	43 RCL 11 43	44 RCL 12 44	45 RCL 13 45	46 RCL 14 46	47 RCL 15 47	2
3	48 STO 00 48	49 STO 01 49	50 STO 02 50	51 STO 03 51	52 STO 04 52	53 STO 05 53	54 STO 06 54	55 STO 07 55	56 STO 08 56	57 STO 09 57	58 STO 10 58	59 STO 11 59	60 STO 12 60	61 STO 13 61	62 STO 14 62	63 STO 15 63	3
4	64 + 64	65 − 65	66 * 66	67 / 67	68 X<Y? 68	69 X>Y? 69	70 X≤Y? 70	71 Σ + 71	72 Σ − 72	73 HMS+ 73	74 HMS− 74	75 MOD 75	76 % 76	77 %CH 77	78 P−R 78	79 R−P 79	4
5	80 LN 80	81 X^2 81	82 SQRT 82	83 Y^X 83	84 CHS 84	85 e^X 85	86 LOG 86	87 10^X 87	88 e^{X-1} 88	89 SIN 89	90 COS 90	91 TAN 91	92 ASIN 92	93 ACOS 93	94 ATAN 94	95 DEC 95	5
6	96 1/X 96	97 ABS 97	98 FACT 98	99 X≠0? 99	100 X>0? 00 100	101 LN1+X 01 101	102 X<0? A 102	103 X=0? B 103	104 INT C 104	105 FRC D 105	106 D−R E 106	107 R−D F 107	108 HMS G 108	109 HR H 109	110 RND I 110	111 OCT J 111	6
7	112 CLΣ T	113 X<>Y Z	114 PI Y	115 CLST X	116 R↑ L	117 RDN M [	118 LASTX N \	119 CLX O]	120 X=Y? P↑	121 X≠Y? Q —	122 SIGN ⊢ ⊤	123 X≤0? a	124 MEAN b	125 SDEV c	126 AVIEW d	127 CLD e	7
8	128 DEG 00	129 RAD 01	130 GRAD 02	131 ENTER 03	132 STOP 04	133 RTN 05	134 BEEP 06	135 CLA 07	136 ASHF 08	137 PSE 09	138 CLRG 10	139 AOFF 11	140 AON 12	141 OFF 13	142 PROMPT 14	143 ADV 15	8
9	144 RCL 16	145 STO 17	146 STO+ 18	147 STO− 19	148 STO* 20	149 STO/ 21	150 ISG 22	151 DSE 23	152 VIEW 24	153 ΣREG 25	154 ASTO 26	155 ARCL 27	156 FIX 28	157 SCI 29	158 ENG 30	159 TONE 31	9
A	160 XROM 32	161 XROM 33	162 XROM 34	163 XROM 35	164 XROM 36	165 XROM 37	166 XROM 38	167 XROM 39	168 SF 40	169 CF 41	170 FS?C 42	171 FC?C 43	172 FS? 44	173 FC? 45	174 GTO IND XEQ IND 46	175 BIDON 47	A
B	176 BIDON 48	177 GTO 00 49	178 GTO 01 50	179 GTO 02 51	180 GTO 03 52	181 GTO 04 53	182 GTO 05 54	183 GTO 06 55	184 GTO 07 56	185 GTO 08 57	186 GTO 09 58	187 GTO 10 59	188 GTO 11 60	189 GTO 12 61	190 GTO 13 62	191 GTO 14 63	B
C	192 GLOBAL 64	193 GLOBAL 65	194 GLOBAL 66	195 GLOBAL 67	196 GLOBAL 68	197 GLOBAL 69	198 GLOBAL 70	199 GLOBAL 71	200 GLOBAL 72	201 GLOBAL 73	202 GLOBAL 74	203 GLOBAL 75	204 GLOBAL 76	205 GLOBAL 77	206 X<> 78	207 LBL 79	C
D	208 GTO 80	209 GTO 81	210 GTO 82	211 GTO 83	212 GTO 84	213 GTO 85	214 GTO 86	215 GTO 87	216 GTO 88	217 GTO 89	218 GTO 90	219 GTO 91	220 GTO 92	221 GTO 93	222 GTO 94	223 GTO 95	D
E	224 XEQ 96	225 XEQ 97	226 XEQ 98	227 XEQ 99	228 XEQ 00 100	229 XEQ 01 101	230 XEQ A 102	231 XEQ B 103	232 XEQ C 104	233 XEQ D 105	234 XEQ E 106	235 XEQ F 107	236 XEQ G 108	237 XEQ H 109	238 XEQ I 110	239 XEQ J 111	E
F	240 TEXT 0 T	241 TEXT 1 Z	242 TEXT 2 Y	243 TEXT 3 X	244 TEXT 4 L	245 TEXT 5 M [	246 TEXT 6 N \	247 TEXT 7 O]	248 TEXT 8 P↑	249 TEXT 9 Q —	250 TEXT 10 ⊢ ⊤	251 TEXT 11 a	252 TEXT 12 b	253 TEXT 13 c	254 TEXT 14 d	255 TEXT 15 e	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

LA BONNE CORRESPONDANCE

rangé en alpha la suite des lettres : ABB DEFGHIJKL, soit les 12 premières lettres à l'exception du C remplacé par B, exécutons la séquence de la figure 1.

Figure 1

Faire	Lire
(PRGM)	06 STO 12
SST	07 LBL 03
SST,... jusqu'à lire	23 —
SST	24 *
SST	25 *
←	24 *
/	25 /
(PRGM)	0.0000

Si l'on regarde maintenant le registre alpha, on constate que le second B est devenu C. Le registre alpha étant composé de 3, 4 registres d'état (*LIST* n° 1), nous avons pu le consulter : lettre A = "-", B = "*", C = "/", etc., et le modifier directement.

La première leçon est donc la suivante : on peut accéder à n'importe quel endroit de la mémoire, son contenu nous est livré *interprété par la HP-41 sous la forme d'une fonction* — il en existe de bien étonnantes : XROM, eG0BEEP, etc. — et, enfin, on introduit un nouveau contenu comme on modifierait un simple programme.

La difficulté est donc d'interpréter correctement ce qu'on lit — à quel code correspond quelle fonction — et de parvenir à introduire n'importe quel code en mémoire — à quelle fonction correspond quel code. Pour ce faire, on dispose d'un outil très simple : la table de correspondance entre les fonctions et les codes (1).

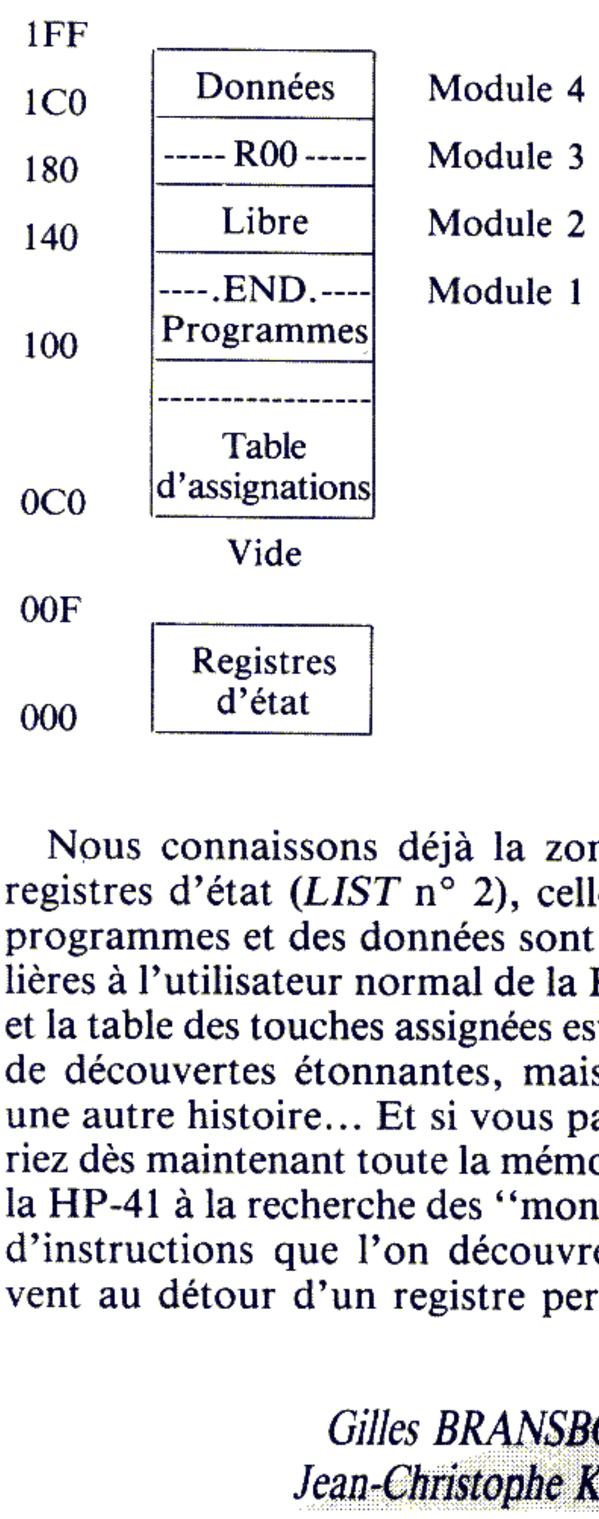
Cette table est reproduite page précédente dans son intégralité. On en examinera ultérieurement les finesses.

Quand, quoi, où ?

Savoir "quoi" n'est pas tout, il faut aussi savoir "où ?". Une seconde table, plutôt une carte géographique, permet de connaître les différentes zones dans lesquelles on peut travailler. Un code quelconque, par exemple STO 12 (code 60 ou 3C en hexadécimal), n'a évidemment pas les mêmes signification et fonction selon qu'il se trouve dans un programme ou dans un registre d'état.

La mémoire est vaste : 320 registres de 7 octets chacun. On s'y repère en hexadécimal en numérotant chaque registre, en lui donnant une adresse en mémoire. Le plus "bas" est le registre 0C0 (192 en décimal) et le plus grand 1FF (511 en décimal), ceci pour une HP-41 CV avec 320 registres (512-192 = 320). Le tout est organisé selon la carte de la figure 2.

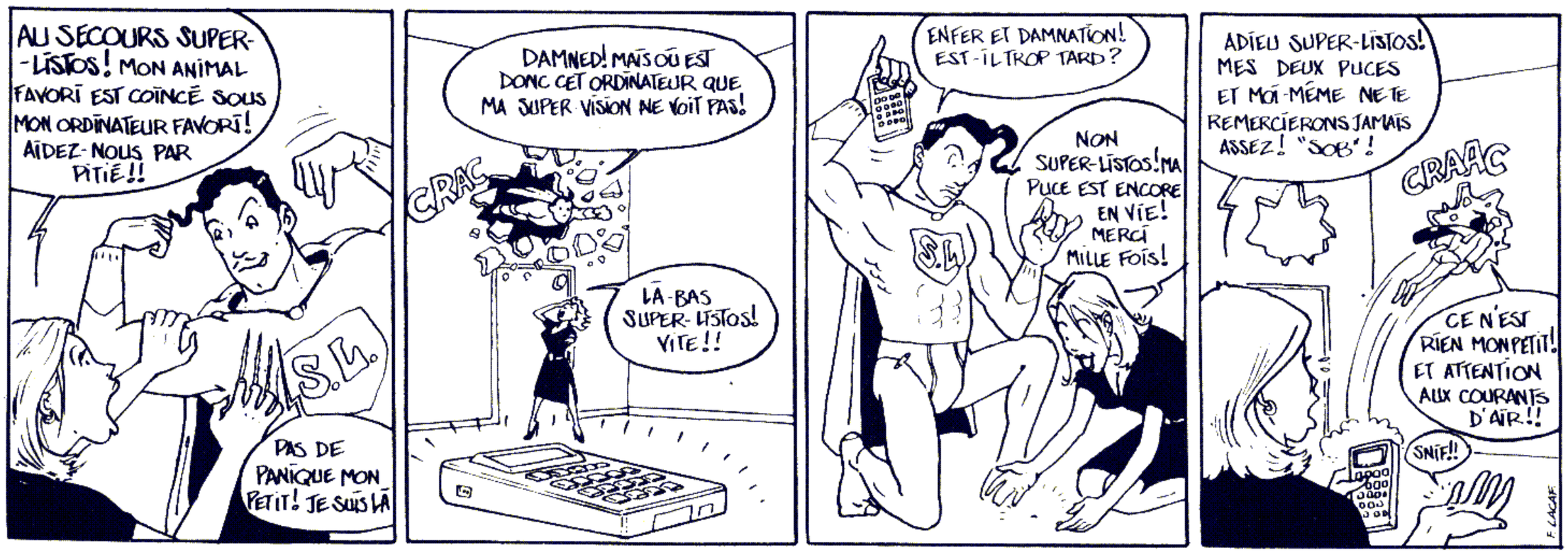
Figure 2
Carte des zones de mémoire de la HP-41

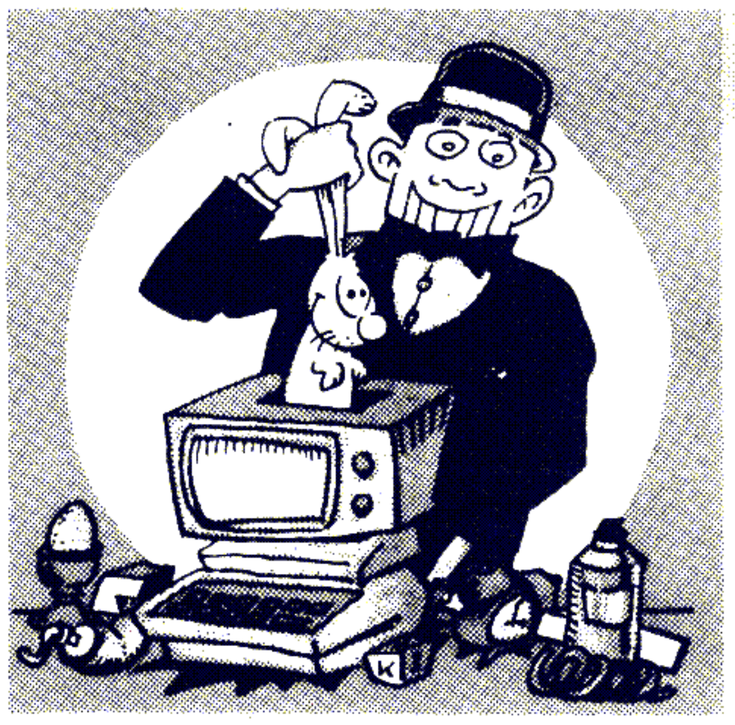


Nous connaissons déjà la zone des registres d'état (*LIST* n° 2), celles des programmes et des données sont familières à l'utilisateur normal de la HP-41 et la table des touches assignées est riche de découvertes étonnantes, mais c'est une autre histoire... Et si vous parcouriez dès maintenant toute la mémoire de la HP-41 à la recherche des "monstres" d'instructions que l'on découvre souvent au détour d'un registre perdu ?

(1) Sources : l'Ordinateur Individuel *nos 21 à 28 et n° 31,* Synthetic Programming on the HP-41 C *de WC Wickes et dont la traduction française est éditée par les éditions du Cagire.*

Gilles BRANSBOURG
Jean-Christophe KRUST





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.

LES TRUCS INDISPENSABLES

Les utilisateurs du C.64 sont souvent pris au dépourvu lorsqu'ils veulent faire exécuter par leur machine certaines actions spécifiques qui ne sont possibles qu'au prix d'une grande complexité du programme, ou même tout à fait impossibles...

Les quelques "trucs" rassemblés ici sont destinés à rendre réalisables bien des projets, et à faciliter la vie de tous les programmeurs du C.64. Vous pouvez en tester certains directement au clavier, mais ils sont essentiellement utilisables dans les programmes.

POKE 650,127... supprime la répétition automatique des touches.
POKE 650,128... établit la répétition pour toutes les touches.
POKE 650,0... rétablit la situation normale (répétition des touches CRSR).
POKE 808,239... supprime l'effet de la touche STOP.
POKE 788,52... idem.
POKE 808,1... rétablit le fonctionnement de la touche STOP.
POKE 788,49... idem.
POKE 808,225... supprime l'effet de STOP/RESTORE, avec pour effet secondaire l'empêchement de LISTer le programme présent en mémoire.
POKE 808,1... rétablit le fonctionnement normal.
POKE 792,34:POKE 793,253... provoque la perte de contrôle de l'ordinateur en cas de STOP/RESTORE.
POKE 657,128... supprime l'effet de SHIFT/COMMODORE (passage majuscules-minuscules).
PRINT CHR$(8)... idem.
POKE 657,0... rétablit la situation normale.
PRINT CHR$(9)... idem.
POKE 198,0: WAIT 198,1... attend la pression sur une touche.
WAIT 197,64... attend le relâchement de toutes les touches.
WAIT 653,1... attend la pression sur SHIFT.
WAIT 653,1,1... attend le relâchement de SHIFT.

LIST

WAIT 653,2... attend la pression sur COMMODORE.
WAIT 653,4... attend la pression sur CTRL.
WAIT 203,4... attend la pression sur F1.
WAIT 203,5... attend la pression sur F2.
PRINT PEEK (203)... rend 64 si aucune touche n'est appuyée, sinon rend la valeur de cette touche.
WAIT 203, XX... attend la pression sur la touche qui provoquera l'apparition de la valeur XX à l'adresse 203. La touche associée à cette valeur est connue grâce à l'instruction précédente (PRINT PEEK (203)).
POKE 819,245... supprime la fonction SAVE.
POKE 819,203... retourne à la normale.
POKE 775,200... supprime la fonction LIST.
POKE 775,167... rétablit la fonction.
POKE 808,225:POKE 818,32... supprime à la fois LIST et SAVE.
POKE 808,237:POKE 818,85... rétablit les deux fonctions.
PRINT PEEK (43)+256*PEEK (44)... permet de connaître l'adresse de début du programme Basic.
PRINT PEEK (174)+256*PEEK (175)... donne l'adresse de fin du programme qui vient d'être chargé en mémoire par LOAD.
PRINT PEEK (152)... donne le nombre de fichiers ouverts à un instant donné.
PRINT CHR$(14)... fait passer en mode minuscule.
PRINT CHR$(142)... fait passer en mode majuscule.
POKE 204,0... force le clignotement du curseur (pour un GET par exemple : POKE 204,0:POKE 198,0:WAIT 198,1:GET A$:POKE 204,1).
POKE 204,1... rétablit le fonctionnement habituel du curseur.
POKE 207,1... empêche le clignotement du curseur lors d'un INPUT.
PEEK(1) AND 16... rend 0 si le moteur du magnétophone est en marche, et 16 s'il est arrêté.
POKE 646,XX... établit la couleur du texte affiché à l'écran (valeurs possibles pour XX : 0 à 15 inclus).
POKE 41,8:POKE 2048,0:NEW... permet d'éditer sur Commodore 3000/4000/8000 un programme du C.64, chargé après avoir tapé cette ligne de trois instructions.
POKE 53265,11:LOAD suivi de POKE 53265,27... permet d'utiliser un lecteur 1540 normal (lecteur VIC 20).

Enfin, POKE 214,12:PRINT:PRINT « texte »... affiche le texte sur la douzième ligne de l'écran.

Faites bon usage de tous ces trucs, et, pourquoi pas, faites-nous part de vos propres découvertes...

Robin BOIS

PC-1211 ET PC-1251

CHASSER L'ERREUR

LE mot DEBUG apparaît dans la table des codes du PC-1251 et dans le Basic du PC-1211. Mais il semble n'avoir aucun effet sur mon PC-1251. Or, debug *peut être traduit par* anti-erreurs. *Il doit donc être bien utile. Qu'en est-il ?*

Laurent LECLAIRE

Décidément, même le nom des commandes n'est pas standard : ce que Sharp nomme DEBUG, d'autres l'appellent TRACE (ou TRON...).

Cette commande est particulièrement utile à la mise au point de programmes : elle permet d'en suivre le cheminement ligne par ligne (et ainsi d'en détecter les erreurs). Ceci est vrai sur le PC-1211. Mais si le mot DEBUG apparaît bien dans la mémoire du PC-1251, il ne semble lui être d'aucune utilité. A moins que l'on puisse découvrir comment le rendre actif...

ALICE

RÉALISER SES RÊVES

Alice est un ordinateur intéressant, plutôt rapide et d'un prix abordable, mais son Basic est assez incomplet. Pour rendre les programmes plus lisibles et leur écriture plus aisée, il faut user d'astuces, parfois même très élémentaires. Elles permettront de combler les quelques lacunes de la machine.

Ainsi, l'utilisation de sous-programmes spécifiques, placés en début de liste et commentés par des REMarques, est une habitude à prendre. En leur réservant les cent premières lignes, le programme principal peut être appelé par la ligne 1 et débuter par une REM en ligne 100.

Il sera prudent, pour éviter des problèmes avec les boucles ou les variables, de faire une liste des variables appelantes, c'est-à-dire celles qui vont être utilisées avant l'appel du sous-programme, et de convenir d'écrire avec des lettres doublées (II, JJ, ...) toutes celles qui seront utilisées à l'intérieur des sous-programmes.

Par exemple, pour disposer d'une temporisation, il suffit d'introduire

une ligne du type 20 FOR II = 1 TO T : NEXT T : RETURN qui pourrait être appelée par ...T = 150 : GOSUB 20 (la valeur de T variant en fonction de la durée souhaitée pour cette temporisation).

Autre hypothèse, vous voulez réaliser un rectangle de couleur ; utilisez le sous-programme suivant où X et Y représentent la position de début du dessin (coin supérieur gauche), H et V respectivement la longueur et la largeur et C, la couleur.

```
40 FOR II = Y TO Y + V - 1
41 FOR JJ = X TO X + H - 1
42 PRINT @ 32*II + JJ,
   MID$(".....",C,1);
43 NEXT JJ
44 NEXT II
45 RETURN
```

L'appel de ces six lignes se fera par ...X = 5:Y = 7:H = 12:V = 4:C = 3: GOSUB 40

Dans MID$(".....",C,1), la variable chaîne (entre guillemets) est constituée en fait des couleurs disponibles, obtenues en frappant Shift Q dans la couleur désignée. Et le point-virgule au bout de l'expression à afficher est indispensable si l'on veut éviter que le reste de la ligne ne se colore en vert.

Toujours au chapitre des couleurs, voici un sous-programme (à appeler par ...GOSUB 80) qui change la couleur de l'écran et celle du texte :

```
80 J = ABS(64 - J)
81 POKE 49151,J
82 RETURN
```

Chaque appel de ces lignes provoque un changement d'affichage. Pour J = 0, les caractères sont en noir sur fond vert, pour J = 64 ils sont rouges sur fond orangé. Les affichages des couleurs ne sont pas modifiés.

Un sous-programme d'attente de caractère est souvent très utile. Appelé par ...GOSUB 60, il peut être écrit :

```
60 IF INKEY$ = "" THEN 60
61 RETURN
```

Il est possible d'utiliser le caractère ainsi récupéré en modifiant la ligne 61 comme suit :

```
61 A$ = INKEY$:RETURN
```

Le caractère sera dès lors retourné dans le programme principal par la variable A$.

Il existe bien d'autres cas où l'utilisation de sous-programmes spécifiques se révélera très pratique. Citons au hasard :

```
10 REM Sous-programme d'identifi-
   cation d'une réponse OUI/NON.
20 REM Sous-programme permet-
   tant d'émettre des sons définis par
   des lettres (A pour LA, B pour SI,
   etc.).
30 REM Affichage à un emplace-
   ment X,Y de l'écran.
40 REM Identification d'un carac-
   tère aux emplacements X,Y de
   l'écran.
```

Cette énumération peut sans doute donner des idées... Laissons à chacun le soin de les réaliser.

Jacques DECONCHAT

TO7-MO5

JOUEZ LA COULEUR

Pour modifier les attributs de couleurs d'une chaîne (caractères et fond de l'écran) sur TO7 ou MO5, la manipulation est simple. Il suffit d'un CHR$(27) suivi d'un caractère ou du code ASCII de celui-ci (voir encadré ci-dessous).

La syntaxe généralisée de cette méthode s'écrit : PRINT CHR$(27) + CHR$(64 + CC) + CHR$(27) + CHR$(80 + CF) + « Chaîne à afficher dans les dites couleurs » + ... où CC est le code de la couleur des caractères et CF celui de la couleur de l'écran.

Arlequin
Programme pour TO 7 et MO 5
Auteur Jean-Paul Carré
Copyright *LIST* et l'auteur

```
10 INPUT "ENTREZ VOTRE
   CHAINE" ; A$
20 C$ = ""
30 FOR I = 1 TO LEN (A$)
40 C$ = C$ + CHR$(27) +
   CHR$(64 + RND * 7) + CHR$(27)
   + CHR$(80 + RND * 7) +
   MID$(A$,I,1)
50 NEXT I
60 PRINT C$
70 GOTO 10
```

Les quelques lignes du programme Arlequin donneront une petite idée de ce que l'on peut faire.

Cette façon de procéder présente, sur l'utilisation de l'instruction COLOR, l'avantage de pouvoir changer les couleurs au sein d'une même chaîne de caractères avec un seul PRINT.

Pensez au casse-tête posé par les PRINT USING "% %" !

Jean-Paul CARRÉ

Codes des couleurs				Codes ASCII	
CHR$(27) + "@"	=	caractères	noirs	@	64
CHR$(27) + "A"	=		rouges	A	65
" + "B"	=		verts	B	66
" + "C"	=		jaunes	C	67
" + "D"	=		bleus	D	68
" + "E"	=		magentas	E	69
" + "F"	=		cyans	F	70
" + "G"	=		blancs	G	71
CHR$(27) + "P"	=	fond	noir	P	80
" + "Q"	=		rouge	Q	81
" + "R"	=		vert	R	82
" + "S"	=		jaune	S	83
" + "T"	=		bleu	T	84
" + "U"	=		magenta	U	85
" + "V"	=		cyan	V	86
" + "W"	=		blanc	W	87

TRS-80

ENREGISTRER ET RAPPELER UN ÉCRAN GRAPHIQUE

Si, comme moi, vous utilisez souvent une carte haute résolution pour dessiner, sur l'écran de votre ordinateur, des figures complexes — courbes, réseaux tracés sur une surface, etc. —, vous appréciez sûrement la possibilité de pouvoir les sauvegarder sur disquette de façon à les retrouver à la demande, en tirer des copies sur papier, etc.

Le TRS-80 Modèle III peut disposer d'un excellent écran de 240 × 640 points, soit 153 600 pixels (la même carte existe pour les modèles II et IV). Une instruction (GSAVE) permet d'enregistrer le travail, qui est récupéré s'il le faut par la commande GLOAD. Malheureusement on ne peut pas les utiliser sous Basic G (Basic graphique de Tandy), mais uniquement sous TRS-DOS (le système d'exploitation). Si on rappelle un écran par GLOAD et si l'on veut ensuite le modifier par exemple, on doit repasser en Basic G... ce qui a pour effet immédiat de vider la mémoire graphique spéciale de 32 Ko...

Une idée naturelle serait de mettre dans des mémoires l'état (ON ou OFF) de chaque pixel ; mais cela signifierait que l'on dispose de quelque 150 Ko. Même 48 + 32 Ko ne suffisent pas. C'est donc exclu. Heureusement, le Basic G possède une arme puissante : le couple (GET, PUT) qui enregistre l'état d'une portion rectangulaire de l'écran (une « fenêtre ») et permet d'en recopier un double exact à n'importe quel endroit (commode pour créer des motifs géométriques !).

Grâce à ces instructions, deux programmes Basic G permettent de créer et d'exploiter, sur une disquette enfichée dans le second lecteur et ayant au moins 101 granules libres (pour 301 enregistrements), un fichier séquentiel "TOTO:1", contenant toute l'information graphique de l'écran. Le temps d'exécution de chacun de ces deux programmes (sauvegarde et chargement de l'écran haute résolution) est assez long : un peu moins de quatre minutes.

Il suffirait certainement d'écrire avec attention, en Assembleur Z 80, des routines d'écriture et de lecture du fichier TOTO sur disquette pour pouvoir insérer, au sein de ces programmes Basic G, des modules en langage-machine. Ceux-ci ramèneraient les temps d'exécution à quelques secondes. Si un « mordu » veut se livrer à ce travail désagréable, il peut nous envoyer ses codes !

La ligne 80 du second programme peut être modifiée de façon à afficher sur l'écran l'image vidéo inversée de ce qui a été enregistré (les TRS-80 n'ont malheureusement pas d'instruction spécialisée pour le faire) : il suffit de remplacer PSET par PRESET.

André WARUSFEL

Programme Basic G d'enregistrement d'un écran graphique sur disquette

```
10 DIM A%(9601)
20 GET(0,0)-(639,239),A%
30 OPEN"O", 1, "TOTO:1"
40 FOR I%=0 TO 9601
50     PRINT #1, A%(I%)
60 NEXT
70 CLOSE
80 END
```

Programme Basic G de rappel d'un écran graphique enregistré

```
10 DIM A%(9601)
20 SCREEN 0
30 OPEN"I", 1, "TOTO:1"
40 FOR I%=0 TO 9601
50     INPUT #1, A%(I%)
60 NEXT
70 CLOSE
80 PUT(0,0),A%,PSET
90 END
```

ORIC

PROGRAMMER POUR MIEUX BOUTER

Sans mémoires de masse (disquettes, cassettes, ...), un ordinateur n'est pas grand-chose. En effet, à la mise sous tension, sa mémoire centrale est généralement vierge (certains ordinateurs, en particulier de poche, disposent eux d'une mémoire continue). Par contre, les mémoires de masse contiennent des fichiers et des programmes, mais aussi des « programmes-système » utiles au fonctionnement de l'ordinateur et de ses périphériques.

Un premier problème se pose alors : comment faire passer les « programmes-système » nécessaires, de la mémoire de masse à la mémoire centrale alors que ce sont eux, justement, qui assurent ce passage ? La solution consiste à introduire un « micro-programme » qui chargera un « chargeur », qui lui-même chargera le « système ». On appelle cette opération, le « boutage » (*booting*, en anglais) : elle consiste à « bouter » le système dès la mise sous tension.

Quant à l'initialisation de ce système, elle se fait grâce à un programme d'« auto-démarrage », l'*Autostart*. Il recherche la présence d'un périphérique de mémoire de masse et charge le « système d'exploitation » qui s'y trouve. Cette opération est réalisée suivant des standards propres à l'ordinateur. Il était donc facile d'y ajouter, par convention, la désignation d'un programme « démarré automatiquement », une fois le système initialisé.

Ainsi, après quelques secondes de mouvements sur les disques, on est directement dans le programme utili-

LIST

le journal des amateurs de programmation

Si programmer un ordinateur est devenu pour vous un loisir, un plaisir... une passion, sachez que **LIST** a été créé pour vous. **LIST** vous aide à tirer davantage de votre matériel, à vous perfectionner dans la conception des programmes qui "tourneront" sur votre machine.
LIST vous initie aux langages informatiques et sélectionne les meilleurs livres pour progresser. **LIST** vous informe de l'actualité et vous fournit trucs, astuces et idées pour mieux programmer...
Pour être sûr de ne rater aucun numéro et pour recevoir **LIST** chez vous, **abonnez-vous** !

FAITES 40F D'ECONOMIE !

LIST, LE PLAISIR DE PROGRAMMER

20F chez votre marchand de journaux

BULLETIN D'ABONNEMENT

à retourner à **LIST**
(service Abonnements)
5, place du Colonel-Fabien
75491 Paris Cedex 10

Nom : _____
Adresse : _____
Ville : _____
Code postal : ☐☐☐☐☐ Pays : _____

Veuillez m'abonner pour 10 numéros au prix avantageux de **160 F*** au lieu de 200 F. Je fais ainsi une économie de 40 F sur le prix de vente au numéro.
Je joins mon règlement indispensable libellé à l'ordre de **LIST**.

* Belgique : **1330 FB** ; Suisse : **50 FS** ; Canada : **30 $ C** ; autres pays : **210 FF**.
Par avion : Afrique francophone : **245 FF** ; Amérique, autre Afrique, Océanie : **305 FF** ; Asie : **355 FF**.
Belgique : Soumillon, 28, av. Massenet, 1190 Bruxelles.
Versement Société Générale 2100405 835-39.
Suisse : 19, route du Grand-Mont, CH 1052, Le Mont-sur-Lausanne, versement Caisse d'Epargne et de Crédit, 10-2418 Le Mont CH 1052, compte courant n° 650 156-7.
Canada : LMPI, 9345, rue de Meaux, St Léonard (Québec), H 1R 3H3, Canada.
Autres pays : 5, place du Colonel-Fabien, 75491 Paris Cedex 10.

Boutage
Programme pour Oric
Auteur Max Hagenburger


```
0 REM ** BOOTUP **
10 CLS
20 POKE 49,40 'bogue ...
30 DOKE 18,48000
40 PRINT "microdisque ESSAIS du 01/10/84 de Max "
50 PRINT :PRINT
60 PRINT "'A' programme A
70 PRINT "'E' ECRAN graphique
80 PRINT "'S' SQUASH
90 PRINT "'esc' liste du microdisque & abandon
100 PRINT :PRINT "choix : ";:GET C$
110 IF C$="A" THEN !LOAD"PROG.A"
120 IF C$="E" THEN PRINT "ECRAN":!ECRAN
130 IF C$="S" THEN PRINT "SQUASH":!SQUASH
140 IF C$=CHR$(27) THEN PRINT :!DIR
150 NEW
```

sateur, la première page ou le premier menu.

Sur Apple, par exemple, le programme désigné par cette méthode de l'Autostart s'appelle *Hello* (le nom lui est attribué à l'initialisation de la disquette). Il peut lui-même lancer un autre programme.

Sur Oric muni d'un lecteur-enregistreur de microdisque, certaines conditions doivent être remplies : le microdisque doit être autonome (maître) et contenir un programme se nommant "BOOTUP.COM" chargé en mode automatique.

Pour créer un microdisque autonome (appelé ESSAIS, par exemple), il faut le formater et y charger le système (DOS V1.1) de la manière suivante :

! FORMAT 0
Disc name ? ESSAIS
Load ...
(formatage du microdisque ESSAIS)

Puis, avec le « disque-système » dans le lecteur (C pour un seul lecteur) :

! COPY "SYSTEM.DOS" TO 0,C
Load...
(recopie du système sur le microdisque ESSAIS)

Le programme de boutage (voir ci-dessus) crée un programme de menu et de chargement pour le microdisque, à condition que les programmes aient été sauvegardés comme ceci :

! SAVE "ECRAN.COM",AUTO
! SAVE "SQUASH.COM",AUTO

Mais avant tout essai, il devra être sauvegardé comme suit :

! SAVE "BOOTUP.COM",AUTO

Les noms de programmes appelés sous cette forme (un point d'exclamation et un nom) et directement exécutés, comme ! ECRAN par exemple, ne doivent pas contenir de mot réservé du Basic ; ainsi, LOGO (log) ou CHIFFRE (if,fre) doivent être évités.

TO BOOT : (tou boute) v. tr. – Pour aller sur la lune avec des chaussures à lacets il faut tirer très fort sur les lacets et sur le pied jusqu'à ce que le pied touche le sol et recommencer avec l'autre pied.

Dès la mise sous tension de l'Oric (ou un Reset), l'auto-démarrage du microdisque sera actionné sans aucune commande du système. Et le menu apparaîtra à l'écran. C'est vraiment facile de bouter !

Max HAGENBURGER

PC-1500

QUAND LES CONVERSIONS PASSENT PAR L'ÉCRAN

L'ordinateur Sharp PC-1500 calcule aussi bien en notation hexadécimale que décimale. Il suffit d'employer le signe & avant un nombre pour faire la conversion : &48 ENTER donne bien 72. Mais essayez donc de programmer cette conversion en n'employant que le Basic... Et pourtant il existe un truc tout simple qui utilise deux instructions classiques du Basic : GPRINT et POINT.

L'instruction GPRINT qui allume une colonne de l'écran (le manuel le précise) accepte les arguments hexadécimaux exprimés « en toutes lettres » :

A$ = "3A"
GPRINT A$
est correct syntaxiquement.

L'instruction POINT, qui consulte l'état de l'écran, retourne, elle, un nombre décimal. Ainsi, programmer simplement :

10 : GPRINT 5 : A = POINT 0
rangera la valeur 5 dans la variable A.

De même (et c'est ce qui importe ici) :

10 : GPRINT "3A" : A = POINT 0
mettra dans A la valeur de &3A convertie en décimal : 58.

Enfin ce petit programme :

10 : INPUT A$
20 : GPRINT A$
30 : A = POINT 0
réalisera par programme la conversion hexadécimal-décimal souhaitée pour des nombres compris entre 0 et 255 (&00 à &FF).

Alain MOREL

DES ATTRACTEURS ÉTRANGES

Il y a plusieurs années, la revue « Pour la science » publiait une récréation mathématique sur les attracteurs étranges. Voici une rapide description du phénomène : on commence par tracer une parabole d'équation $y = 4Lx(1-x)$, puis la droite d'équation $y = x$ (c'est-à-dire une diagonale à 45 degrés).

L'intersection de ces deux graphes est le « point d'attraction ». Il faut ensuite trouver une fonction qui place les points de sa courbe représentative alternativement sur la parabole et sur la diagonale, soit f(f(x)).

On obtient des « marches d'escalier » allant de l'origine au point d'attraction. Jusque-là, rien d'anormal direz-vous ; justement, c'est ici qu'apparaît le phénomène et que d'étranges choses se passent... Le système se met à osciller autour d'un ou plusieurs centres de gravité, de façon stable ou non, selon la valeur du paramètre L de la fonction et celle de x.

Si les matheux s'expliquent le phénomène, les autres pourront introduire dans leur machine le programme 1 afin d'admirer le spectacle.

Les lignes 90 à 110 tracent les axes et la droite d'équation $y = x$ (ligne 110) ; les lignes 130 à 170, la parabole.

Il faut ensuite effectuer une itération récursive (si, si !) : la ligne 200 évalue la fonction. Puis, la marche d'escalier est tracée (lignes 220 et 230). La valeur de Y, calculée au tour précédent, est alors placée dans X (ligne 240). On retourne à la ligne 200 pour une nouvelle évaluation de la fonction. Et ainsi de suite, mais... pas jusqu'à l'infini car, comme nous le disions précédemment, le système prend un équilibre plus ou moins complexe autour du « point d'attraction ».

Chacun peut trouver empiriquement les plus belles figures en donnant au paramètre L (ligne 120) des valeurs différentes. Pour obtenir les effets les plus spectaculaires, il suffit de faire varier L de 0, 7 à 0,999. Les plus téméraires peuvent envisager plusieurs dérivations en cascade f(f(f(x))). Grand frisson garanti, mais arrangez-vous pour que votre production ne sorte pas de l'écran : réduisez le pas de 200 à 100, et même plus encore.

Spécimen d'attracteur étrange, photographié par le mathématicien Michel Hénon au cours d'une expédition dans une parabole en juin 1976.

Programme 1

```
10    REM ***********************************************
20    REM ***  Fn(x): Y=4*L*x*(1-x)                     ***
30    REM ***  x est la "graine".L est le "curseur"***
40    REM ***  recherche de l'attracteur et iteration*
50    REM ***********************************************
60    REM
70    REM FN     Y=4*l*x*(1-x)
80    PRINT CHR$(12):MODE 6A:COLORG 0 12 2 8
90    DRAW 0,0 0,YMAX 8
100   DRAW 0,0 XMAX,0 8
110   DRAW 0,0 200,200 8
120   L=0.7
130   FOR X=0.0 TO 1.0 STEP 1.0/200.0
140   Y=4.0*L*X*(1.0-X)
150   XE=X*200.0:YE=Y*200.0
160   DOT XE,YE 12
170   NEXT
180   REM
190   Y0=0.0:X=4E-2
200   Y=4.0*L*X*(1.0-X)
210   XE=X*200.0:YE=Y*200.0:YE0=Y0*200.0
220   DRAW XE,YE0 XE,YE 2
230   DRAW XE,YE YE,YE 2
240   X=Y:Y0=Y
250   GOTO 200
260   GOTO 260
*
```

Programme 2

```
10    REM
20    REM ***********************************************
30    REM *** attracteur etrange de Michel HENON ***
40    REM *** (c) ALAIN MARIATTE  &    LIST       ***
50    REM ***********************************************
60    REM
70    PRINT CHR$(12):MODE 6A:COLORG 0 1 2 3
80    X=0.0:Y=0.0:A=7.0/5.0:B=3.0/10.0
90    C=Y-A*X*X+1.0
100   D=B*X
110   XE=C*100.0+XMAX/2.0:YE=D*100.0+YMAX/2.0
120   DOT XE,YE 3
130   X=C:Y=D
140   GOTO 90
*
```

Le principe est le même que précédemment et le résultat est une sorte de « boomerang » qui se parfait à l'infini. L'analyse mathématique montre que la forme se compose d'une infinité de sous-formes semblables, chaque sous-élément pouvant à son tour se décomposer en une infinité de « sous-sous-éléments ». Un hologramme, quoi !

Alain MARIATTE

LES JEUX ET CASSE-TÊTE INFORMATIQUES

de 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...

10

Gagner des microsecondes

L'initialisation d'un tableau est une opération très courante en informatique.
Par exemple, pour remettre à zéro un tableau de 100 éléments, appelé T, on peut écrire :

```
10 I = 0
20 I = I+1
30 T(I) = 0
40 if I < 100 then 20
```

Cet extrait de programme manque toutefois de structure. On peut encore l'écrire de la manière suivante :

```
10 for I = 1 to 100
20 T(I) = 0
30 next I
```

S'il gagne en lisibilité, ce programme n'est pas encore le plus rapide. Ainsi, sur l'ordinateur que nous utilisons, les temps (en microsecondes, c'est-à-dire en millionièmes de seconde) d'exécution de chacune des instructions sont représentés dans le tableau ci-dessous.

Le premier des deux programmes s'exécutera donc en un temps égal à : 1 fois la ligne 10 + 100 fois la ligne 20 + 100 fois la ligne 30 + 100 fois la ligne 40. Soit 460 + 71 000 + 88 000 + 45 000 = 204 460 microsecondes. Ce qui correspond environ à 0,2 seconde.

De la même façon, le second programme mettra : 100 fois la ligne 10 + 100 fois la ligne 20. Soit 85 000 + 88 000 = 173 000 microsecondes.

Comme c'est souvent le cas, le pro-

Dénomination de l'instruction	Exemples d'instruction	Temps d'exécution
Affectation d'une constante à une variable	I = 0 ou I = 1	460
Incrémentation d'une variable	I = I + 1	710
Affectation d'une constante à l'élément d'un tableau	T(I) = 0 ou T(I) = 1	880
Affectation d'une constante et calcul de l'indice de l'élément d'un tableau	T (I+1) = 0	1100
Affectation de la valeur d'un élément à un autre élément d'un tableau	T(I) = T(1)	1280
Comparaison d'une valeur	if I < 100 then/else	450
Passage sur une boucle for... next	for I=1 to N step P/next I	850 par itération

* *Les solutions des jeux proposés ici paraîtront dans le prochain numéro de* LIST.

gramme le plus facile à comprendre est le plus rapide à l'exécution. Ce dernier a en effet permis de gagner 204 460 - 173 000, soit 31 460 microsecondes.

Toutefois, ce temps de calcul peut encore être amélioré, pour ne pas dépasser 150 000 microsecondes. En utilisant seulement les instructions dont les temps sont indiqués dans le tableau précédent, comment allez-vous programmer cette initialisation afin de ne pas dépasser ces 150 000 microsecondes ?

11

Trouver les mots

Voici dix définitions de mots utilisés plus ou moins couramment en informatique. Essayez de trouver les mots qui se rapportent à ces définitions.

1. Représentation graphique du cheminement d'un programme.

2. Logiciel de traduction d'un langage évolué en langage-machine.

3. Suite d'instructions d'un programme qui doivent être exécutées de façon répétitive.

4. Partie d'un ordinateur qui permet de réaliser des opérations élémentaires.

5. Unité de vitesse de transmission d'informations.

6. Dispositif qui élabore un signal en réaction à une donnée physique.

7. Logiciel conçu pour l'enseignement assisté par ordinateur.

8. Suite d'étapes nécessaires à la résolution d'un problème donné.

9. Dispositif matériel où sont stockées des informations.

10. Action de rechercher les erreurs d'un programme.

12

L'aléatoire encadré

La plupart des ordinateurs disposent de la fonction RND (RaNDom) qui retourne un nombre pseudo-aléatoire entre 0 (inclus) et 1 (exclu).

Ce type de fonction est très utile pour toutes les simulations et les jeux qui peuvent être programmés. L'intervalle de 0 à 1 n'est toutefois pas toujours celui que l'on désire avoir.

Vous trouverez certainement une formule générale permettant d'obtenir, à partir de cette fonction RND, un nombre aléatoire compris entre A (inclus) et B (inclus). A partir de cette formule, vous pourrez en déduire une autre permettant, par exemple, de lancer un dé.

SOLUTIONS DU NUMÉRO PRÉCÉDENT

Les solutions présentées ici répondent aux problèmes posés dans le précédent numéro de LIST. Ce ne sont pas forcément les meilleures !

▸▸▸7

Minimum et maximum

La fonction MIN (A, B) peut être remplacée par l'expression : (A + B)/2 − (ABS(A − B))/2 où ABS (X) retourne la valeur absolue de X.

Cette expression présente l'avantage de ne faire apparaître aucun test de comparaison. De même, MAX (A, B) peut être remplacé par : (A + B)/2 + (ABS(A − B))/2.

▸▸▸8

L'affectation, cette incomprise

L'exécution de l'instruction A := A s'effectue en deux temps :

- lecture de la valeur du A situé à droite du symbole d'affectation ;
- stockage de cette valeur à l'adresse du A situé à gauche du symbole d'affectation.

Le premier A de l'instruction représente donc une *adresse,* alors que le second représente une *valeur*.

▸▸▸9

Sigles et abréviations

La liste ci-dessous donne la signification de chaque abréviation (ou sigle), souvent rencontrée en informatique :

BIT	: Binary digIT (chiffre binaire)
CP/M	: Control Program/Monitor (programme de contrôle/moniteur)
ASCII	: American Standard Code for Information Interchange (code standard américain pour l'échange d'informations)
BS	: BackSpace (espace arrière)
SOS	: Silicon On Sapphire (silicium sur saphir)
CRT	: Cathode Ray Tube (tube à rayon cathodique)
ALGOL	: ALGOrithmic Language (langage algorithmique)
SSII	: Société de Services et d'Ingénierie en Informatique

Imprimerie SIMA, rue des Epinettes, ZI de Torcy Sud, 77200 Torcy. Printed in France. Directeur de la publication : Jean-Luc Verhoye.
Dépôt légal octobre 1984. Diffusion NMPP. Numéro de commission paritaire : 63617