PRATIQUE DE L'INFORMATIQUE

BASIC

et le traitement de textes

gilbert quaneaux

sylvine bouvry

EYROLLES

BASIC
ET LE TRAITEMENT DE TEXTES

EN VENTE AUX ÉDITIONS EYROLLES

Dans la même collection

BASIC :

Cl. DE ROSSI — *Apprentissage rapide du Basic* (1982).
S. HIRSCH — *Le Basic facile par une méthode progressive* (1982).
J. LONGCHAMP — *Basic. Construction méthodique des programmes* (1981).
J. NEVISON — *L'art de bien programmer en Basic. Le petit livre du style* (1982).
A. PARKER et V. SILBEY — *Le Basic en gestion* (1982).
G. QUANEAUX — *Exercices de gestion en Basic* (1981).
G. QUANEAUX — *Basic et le traitement de textes (1982).*

COBOL :

Ch. BONNIN — *Le Cobol A.N.S. avec exercices et corrigés* (1982).
Ch. BONNIN — *Les extensions au Cobol A.N.S. avec exercices et corrigés* (1981).
Ch. BONNIN — *Exercices pratiques de programmation en Cobol A.N.S. 74* (1982).

PASCAL :

Ph. KRUCHTEN — *Le langage de programmation Pascal* (1982).

PL/1 :

Ch. BONNIN — *Pratique du PL/1 et programmation structurée* (1982).
Ch. BONNIN — *Les techniques avancées de programmation PL/1* (1982).

MICROPROCESSEURS ET MINIORDINATEURS :

A. BARNA et D.I. PORAT — *Initiation aux minicalculateurs et microprocesseurs* (1980)
D. GIROD et R. DUBOIS — *Au cœur des microprocesseurs* (1980).
R.J. TOCCI et L. LASKOWSKI — *Microprocesseurs et microordinateurs. Matériel et logiciel* (1981).
A.M. LISTER — *Principes fondamentaux des systèmes d'exploitation* (1982).

Chez le même éditeur :

Ch. BONNIN — *Basic* (série Mémentos Eyrolles) (1982).
J.P. LAMOITIER — *Le langage Basic et la nouvelle norme* (1982).
G. LADEVIE — *Votre gestion avec Basic sur microordinateur* (coll. Microordinateurs) (1982).
J.C. LARRÉCHÉ — *Le Basic. Une introduction à la programmation* (1981).
R. SCHOMBERG — *Le Basic universel* (coll. Microordinateurs) (1982).

PRATIQUE DE L'INFORMATIQUE

BASIC
ET LE TRAITEMENT DE TEXTES

par

Gilbert QUANEAUX

EYROLLES 61, Boulevard Saint-Germain — 75005 Paris
1983

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

AVERTISSEMENT

L'apparition, toute récente, du vocable « bureautique » a mis l'accent sur les applications basées sur le traitement des textes. Presque chaque jour, des matériels et des logiciels conçus pour la manipulation des données alphabétiques sont lancés sur le marché.

Dans la vie de l'Entreprise, il est en pratique *difficile de dissocier Informatique et Bureautique.* Une secrétaire tient des fichiers, tout comme un comptable ou un employé administratif. L'édition de contrats demande l'accès à des fichiers informatiques : « tarifs » et « clients ». Pour éditer des lettres de rappel à des clients distraits, il faut utiliser des données issues de traitement informatiques : facturation et comptabilité etc...

Il n'y a aucune différence de *nature* entre ces applications « bureautiques » et les applications « informatiques » : les unes et les autres utilisent des constantes et des variables, des fichiers et des instructions. Bien souvent, d'ailleurs, on constate que les premières sont plus simples à réaliser que les secondes. Les applications bureautiques n'ont été prises en considération par les informaticiens que tardivement non pas à cause de leur complexité, mais d'abord pour des raisons *psychologiques* (la « clientèle » des secrétariats n'a pas du tout la même mentalité que les comptables ou les gestionnaires), *structurelles* (la rigidité - toujours existante - des systèmes informatiques centralisés s'accommode mal des changements incessants affectant les taches « de bureau » et *financières* (le coût de développement et de maintenance apparaissait bien lourd sur les gros systèmes centralisés). L'apparition des « minis » puis des

« micros » d'une part et les *coûts croissants* de la main d'œuvre ont totalement modifié les données du problème. Il devient maintenant *possible* et *rentable* d'implanter sur chaque bureau un terminal permettant à chacun de traiter ses propres textes et de gérer ses fichiers personnels.

Le BASIC, langage privilégié des micros et minis systèmes, se devait d'offrir un ensemble efficace et complet de fonctions permettant la manipulation de données alphabétiques (appelées souvent « chaînes de caractères »). Des logiciels de traitement de textes écrits en BASIC sont d'ailleurs actuellement diffusés.

L'emploi de ces fonctions BASIC permet donc aux utilisateurs d'intégrer *dans les mêmes procédures* traitement de Textes et traitement de Données, Informatique et Bureautique. Dans certains cas, l'utilisation du BASIC peut éviter l'acquisition d'un système spécialisé dans le traitement de Textes.

Nous verrons, de plus, que ces fonctions BASIC permettent de *simplifier* l'écriture et la maintenance des programmes de gestion « classiques ».

L'objectif de cet ouvrage n'est pas d'exposer toutes les fonctions offertes par les différents fournisseurs de matériel et de logiciel mais de présenter au lecteur un BASIC « représentatif » (celui du système 34 d'IBM), d'en détailler les fonctions liées au traitement de Textes et de les illustrer par de nombreux exemples et exercices. Il est bien évident que le lecteur pourra trouver sur d'autres systèmes des opérateurs ou des fonctions analogues portant des noms différents et offrant des possibilités plus ou moins importantes que celles que nous présentons.

Nous avons divisé cet ouvrage en deux grandes parties :

— La première (chapitres 1 à 5) présente le « mode d'emploi » des fonctions BASIC et l'illustre par des exemples et exercices simples.

— La deuxième (chapitres 6 et 7) présente les fonctions générales de la Bureautique et montre par un exposé théorique illustré par des exercices comment réaliser en BASIC la programmation correspondant à ces fonctions.

TABLE DES MATIÈRES

1

DÉFINITION D'UNE CHAÎNE DE CARACTÈRES

1.1 CONSTANTES ET VARIABLES ALPHANUMÉRIQUES

Les variables et constantes alphanumériques sont utilisées depuis fort longtemps par le langage BASIC. Si elles sont peu importantes dans les problèmes de calcul scientifique, il est indispensable de pouvoir les manipuler aisément dans les applications de gestion. Les opérations qu'on est appelé à faire sur les zones alphanumériques étant différentes de celles portant sur les zones numériques, il est nécessaire de fournir à l'interpréteur BASIC un moyen de distinguer les unes des autres.

Une variable alphanumérique sera distinguée par le BASIC d'une variable numérique par *la présence d'un caractère « dollar » ($) en dernière position du nom* symbolique attribué à la variable. Ainsi ZONX désignera une zone numérique et l'interpréteur saura que les opérations arithmétiques sont permises sur celle-ci. Par contre, ZONX$ référencera une zone alphanumérique et ces mêmes opérations seront interdites sur cette zone.

La longueur du nom symbolique varie suivant le système BASIC utilisé. Dans celui qui nous a servi pour cet ouvrage, elle peut aller de deux caractères (A$, par exemple) à neuf (MATRICUL$, par exemple). Rappelons, de plus, qu'un nom symbolique doit commencer par une lettre majuscule (de A à Z).

Une constante alphanumérique est composée d'un ensemble de caractères compris entre deux guillemets (''). Ainsi, l'expression ''ALE-

XANDRE'' sert à spécifier la constante ALEXANDRE dans le programme. Lorsque le programmeur désire utiliser ce caractère guillemet à l'intérieur d'une constante, il devra l'indiquer sous forme de deux guillemets accolés (''''). Ainsi, l'expression MARQUE'''' RENAULT'''''' sera comprise par le programme comme la constante MARQUE''RENAULT''.

Dans la suite de cet ouvrage, nous utiliserons le vocable « chaîne de caractères » ou plus brièvement « chaîne » pour désigner une constante ou une variable alphanumérique. Ce vocable, très utilisé en informatique, reflète bien la réalité : une chaîne est un ensemble de caractères enchaînés les uns aux autres, chaque caractère constituant un maillon de la chaîne.

1.2 JEUX DE CARACTÈRES

Rappelons au lecteur qu'un caractère est stocké dans la mémoire de l'ordinateur sous forme d'un nombre binaire, en général de huit chiffres (Octet). La table donnant l'équivalent binaire d'un caractère représenté sous sa forme externe (imprimée, affichée sur un écran cathodique ou gravée sur un clavier de terminal) s'appelle « jeu de caractères » (en anglais : character set).

Les figures 1.1 et 1.2 représentent les deux jeux de caractères les plus usités : l'EBCDIC et l'ASCII. On remarquera que certains nombres n'ont pas de représentation graphique : ils sont utilisés comme «caractères de commande» pour déclencher des fonctions périphériques de l'ordinateur (impression, transmission, affichage, etc...).

1.3 CONTENU ET LONGUEUR D'UNE CHAÎNE DE CARACTÈRES

Comme dans le cas des zones numériques, le signe égal (=) sert à affecter la valeur de l'expression située à droite du signe égal à la variable dont le nom figure à gauche. Ainsi :

RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR
0	00	00000000		64	40	01000000		128	80	10000000		192	C0	11000000	é
1	01	00000001		65	41	01000001		129	81	10000001	a	193	C1	11000001	A
2	02	00000010		66	42	01000010		130	82	10000010	b	194	C2	11000010	B
3	03	00000011		67	43	01000011		131	83	10000011	c	195	C3	11000011	C
4	04	00000100		68	44	01000100		132	84	10000100	d	196	C4	11000100	D
5	05	00000101		69	45	01000101		133	85	10000101	e	197	C5	11000101	E
6	06	00000110		70	46	01000110		134	86	10000110	f	198	C6	11000110	F
7	07	00000111		71	47	01000111		135	87	10000111	g	199	C7	11000111	G
8	08	00001000		72	48	01001000		136	88	10001000	h	200	C8	11001000	H
9	09	00001001		73	49	01001001		137	89	10001001	i	201	C9	11001001	I
10	0A	00001010		74	4A	01001010	°	138	8A	10001010		202	CA	11001010	
11	0B	00001011		75	4B	01001011	.	139	8B	10001011		203	CB	11001011	
12	0C	00001100		76	4C	01001100	<	140	8C	10001100		204	CC	11001100	
13	0D	00001101		77	4D	01001101	(	141	8D	10001101		205	CD	11001101	
14	0E	00001110		78	4E	01001110	+	142	8E	10001110		206	CE	11001110	
15	0F	00001111		79	4F	01001111	!	143	8F	10001111		207	CF	11001111	
16	10	00010000		80	50	01010000	&	144	90	10010000		208	D0	11010000	è
17	11	00010001		81	51	01010001		145	91	10010001	j	209	D1	11010001	J
18	12	00010010		82	52	01010010		146	92	10010010	k	210	D2	11010010	K
19	13	00010011		83	53	01010011		147	93	10010011	l	211	D3	11010011	L
20	14	00010100		84	54	01010100		148	94	10010100	m	212	D4	11010100	M
21	15	00010101		85	55	01010101		149	95	10010101	n	213	D5	11010101	N
22	16	00010110		86	56	01010110		150	96	10010110	o	214	D6	11010110	O
23	17	00010111		87	57	01010111		151	97	10010111	p	215	D7	11010111	P
24	18	00011000		88	58	01011000		152	98	10011000	q	216	D8	11011000	Q
25	19	00011001		89	59	01011001		153	99	10011001	r	217	D9	11011001	R
26	1A	00011010		90	5A	01011010	§	154	9A	10011010		218	DA	11011010	
27	1B	00011011		91	5B	01011011	$	155	9B	10011011		219	DB	11011011	
28	1C	00011100		92	5C	01011100	*	156	9C	10011100		220	DC	11011100	
29	1D	00011101		93	5D	01011101	)	157	9D	10011101		221	DD	11011101	
30	1E	00011110		94	5E	01011110	;	158	9E	10011110		222	DE	11011110	
31	1F	00011111		95	5F	01011111	^	159	9F	10011111	▮	223	DF	11011111	
32	20	00100000		96	60	01100000	-	160	A0	10100000		224	E0	11100000	ç
33	21	00100001		97	61	01100001	/	161	A1	10100001	¨	225	E1	11100001	
34	22	00100010		98	62	01100010		162	A2	10100010	s	226	E2	11100010	S
35	23	00100011		99	63	01100011		163	A3	10100011	t	227	E3	11100011	T
36	24	00100100		100	64	01100100		164	A4	10100100	u	228	E4	11100100	U
37	25	00100101		101	65	01100101		165	A5	10100101	v	229	E5	11100101	V
38	26	00100110		102	66	01100110		166	A6	10100110	w	230	E6	11100110	W
39	27	00100111		103	67	01100111		167	A7	10100111	x	231	E7	11100111	X
40	28	00101000		104	68	01101000		168	A8	10101000	y	232	E8	11101000	Y
41	29	00101001		105	69	01101001		169	A9	10101001	z	233	E9	11101001	Z
42	2A	00101010		106	6A	01101010	ù	170	AA	10101010		234	EA	11101010	
43	2B	00101011		107	6B	01101011	,	171	AB	10101011		235	EB	11101011	
44	2C	00101100		108	6C	01101100	%	172	AC	10101100		236	EC	11101100	
45	2D	00101101		109	6D	01101101	_	173	AD	10101101		237	ED	11101101	
46	2E	00101110		110	6E	01101110	>	174	AE	10101110		238	EE	11101110	
47	2F	00101111		111	6F	01101111	?	175	AF	10101111		239	EF	11101111	
48	30	00110000		112	70	01110000		176	B0	10110000		240	F0	11110000	0
49	31	00110001		113	71	01110001		177	B1	10110001		241	F1	11110001	1
50	32	00110010		114	72	01110010		178	B2	10110010		242	F2	11110010	2
51	33	00110011		115	73	01110011		179	B3	10110011		243	F3	11110011	3
52	34	00110100		116	74	01110100		180	B4	10110100		244	F4	11110100	4
53	35	00110101		117	75	01110101		181	B5	10110101		245	F5	11110101	5
54	36	00110110		118	76	01110110		182	B6	10110110		246	F6	11110110	6
55	37	00110111		119	77	01110111		183	B7	10110111		247	F7	11110111	7
56	38	00111000		120	78	01111000		184	B8	10111000		248	F8	11111000	8
57	39	00111001		121	79	01111001	µ	185	B9	10111001		249	F9	11111001	9
58	3A	00111010		122	7A	01111010	:	186	BA	10111010		250	FA	11111010	
59	3B	00111011		123	7B	01111011	£	187	BB	10111011		251	FB	11111011	
60	3C	00111100		124	7C	01111100	à	188	BC	10111100		252	FC	11111100	
61	3D	00111101		125	7D	01111101	'	189	BD	10111101		253	FD	11111101	
62	3E	00111110		126	7E	01111110	=	190	BE	10111110		254	FE	11111110	
63	3F	00111111		127	7F	01111111	"	191	BF	10111111		255	FF	11111111	

Fig. 1.1. — *Jeu de caractères EBCDIC*

NO	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR
0	00	00000000		64	40	01000000	à	128	80	10000000		192	C0	11000000	
1	01	00000001		65	41	01000001	A	129	81	10000001		193	C1	11000001	
2	02	00000010		66	42	01000010	B	130	82	10000010		194	C2	11000010	
3	03	00000011		67	43	01000011	C	131	83	10000011		195	C3	11000011	
4	04	00000100		68	44	01000100	D	132	84	10000100		196	C4	11000100	
5	05	00000101		69	45	01000101	E	133	85	10000101		197	C5	11000101	
6	06	00000110		70	46	01000110	F	134	86	10000110		198	C6	11000110	
7	07	00000111		71	47	01000111	G	135	87	10000111		199	C7	11000111	
8	08	00001000		72	48	01001000	H	136	88	10001000		200	C8	11001000	
9	09	00001001		73	49	01001001	I	137	89	10001001		201	C9	11001001	
0	0A	00001010		74	4A	01001010	J	138	8A	10001010		202	CA	11001010	
1	0B	00001011		75	4B	01001011	K	139	8B	10001011		203	CB	11001011	
2	0C	00001100		76	4C	01001100	L	140	8C	10001100		204	CC	11001100	
3	0D	00001101		77	4D	01001101	M	141	8D	10001101		205	CD	11001101	
4	0E	00001110		78	4E	01001110	N	142	8E	10001110		206	CE	11001110	
5	0F	00001111		79	4F	01001111	O	143	8F	10001111		207	CF	11001111	
6	10	00010000		80	50	01010000	P	144	90	10010000		208	D0	11010000	
7	11	00010001		81	51	01010001	Q	145	91	10010001		209	D1	11010001	
8	12	00010010		82	52	01010010	R	146	92	10010010		210	D2	11010010	
9	13	00010011		83	53	01010011	S	147	93	10010011		211	D3	11010011	
0	14	00010100		84	54	01010100	T	148	94	10010100		212	D4	11010100	
1	15	00010101		85	55	01010101	U	149	95	10010101		213	D5	11010101	
2	16	00010110		86	56	01010110	V	150	96	10010110		214	D6	11010110	
3	17	00010111		87	57	01010111	W	151	97	10010111		215	D7	11010111	
4	18	00011000		88	58	01011000	X	152	98	10011000		216	D8	11011000	
5	19	00011001		89	59	01011001	Y	153	99	10011001		217	D9	11011001	
6	1A	00011010		90	5A	01011010	Z	154	9A	10011010		218	DA	11011010	
7	1B	00011011		91	5B	01011011	°	155	9B	10011011		219	DB	11011011	
8	1C	00011100		92	5C	01011100	ç	156	9C	10011100		220	DC	11011100	
9	1D	00011101		93	5D	01011101	§	157	9D	10011101		221	DD	11011101	
0	1E	00011110		94	5E	01011110	^	158	9E	10011110		222	DE	11011110	
1	1F	00011111		95	5F	01011111	_	159	9F	10011111		223	DF	11011111	
2	20	00100000		96	60	01100000	µ	160	A0	10100000		224	E0	11100000	
3	21	00100001	!	97	61	01100001	a	161	A1	10100001		225	E1	11100001	
4	22	00100010	"	98	62	01100010	b	162	A2	10100010		226	E2	11100010	
5	23	00100011	£	99	63	01100011	c	163	A3	10100011		227	E3	11100011	
6	24	00100100	$	100	64	01100100	d	164	A4	10100100		228	E4	11100100	
7	25	00100101	%	101	65	01100101	e	165	A5	10100101		229	E5	11100101	
8	26	00100110	&	102	66	01100110	f	166	A6	10100110		230	E6	11100110	
9	27	00100111	'	103	67	01100111	g	167	A7	10100111		231	E7	11100111	
0	28	00101000	(	104	68	01101000	h	168	A8	10101000		232	E8	11101000	
1	29	00101001	)	105	69	01101001	i	169	A9	10101001		233	E9	11101001	
2	2A	00101010	*	106	6A	01101010	j	170	AA	10101010		234	EA	11101010	
3	2B	00101011	+	107	6B	01101011	k	171	AB	10101011		235	EB	11101011	
4	2C	00101100	,	108	6C	01101100	l	172	AC	10101100		236	EC	11101100	
5	2D	00101101	-	109	6D	01101101	m	173	AD	10101101		237	ED	11101101	
6	2E	00101110	.	110	6E	01101110	n	174	AE	10101110		238	EE	11101110	
7	2F	00101111	/	111	6F	01101111	o	175	AF	10101111		239	EF	11101111	
8	30	00110000	0	112	70	01110000	p	176	B0	10110000		240	F0	11110000	
9	31	00110001	1	113	71	01110001	q	177	B1	10110001		241	F1	11110001	
0	32	00110010	2	114	72	01110010	r	178	B2	10110010		242	F2	11110010	
1	33	00110011	3	115	73	01110011	s	179	B3	10110011		243	F3	11110011	
2	34	00110100	4	116	74	01110100	t	180	B4	10110100		244	F4	11110100	
3	35	00110101	5	117	75	01110101	u	181	B5	10110101		245	F5	11110101	
4	36	00110110	6	118	76	01110110	v	182	B6	10110110		246	F6	11110110	
5	37	00110111	7	119	77	01110111	w	183	B7	10110111		247	F7	11110111	
6	38	00111000	8	120	78	01111000	x	184	B8	10111000		248	F8	11111000	
7	39	00111001	9	121	79	01111001	y	185	B9	10111001		249	F9	11111001	
8	3A	00111010	:	122	7A	01111010	z	186	BA	10111010		250	FA	11111010	
9	3B	00111011	;	123	7B	01111011	é	187	BB	10111011		251	FB	11111011	
0	3C	00111100	<	124	7C	01111100	ù	188	BC	10111100		252	FC	11111100	
1	3D	00111101	=	125	7D	01111101	è	189	BD	10111101		253	FD	11111101	
2	3E	00111110	>	126	7E	01111110	¨	190	BE	10111110		254	FE	11111110	
3	3F	00111111	?	127	7F	01111111		191	BF	10111111		255	FF	11111111	

Fig. 1.2. — *Jeu de caractères ASCII*

— L'instruction ZONA$ = ''Nom et Prénom'' placera la constante Nom et Prénom dans la chaîne appelée ZONA$

— L'instruction ZONX$ = ZONA$ affectera le contenu de la chaîne appelée ZONA$ à la chaîne appelée ZONX$.

Il y a cependant une différence très importante entre le traitement des zones numériques et celui des chaînes de caractères : *la longueur d'une zone numérique reste fixe* quelques soient les traitements effectués, *celle d'une chaîne varie* en fonction des opérations que le programme lui fait subir.

Nous allons illustrer ceci en exécutant le petit programme ci-dessous. Il utilise la fonction BASIC ''LEN'' qui sert à récupérer la longueur de la chaîne ou plus généralement de l'expression « caractère » figurant entre parenthèses. Par exemple, l'instruction L = LEN(A$) placera la longueur de la chaîne A$ dans la zone numérique L.

```
00010 ! E101 : Longueur d'une chaine de caractères
00020 ! ------------------------------------------
00030             PRINT "au début,A$ contient :",A$;",sa longueur est :";LEN(A$)
00040 A$="ABCDE"
00050             PRINT "après 40,A$ contient :";A$;",sa longueur est :",LEN(A$)
00060 A$="AZERTY  "
00070             PRINT "après 60,A$ contient :";A$;",sa longueur est :";LEN(A$)
00080 A$="W""CVB"""
00090             PRINT "après 80,A$ contient :";A$;",sa longueur est :";LEN(A$)
00100 B$=A$
00110             PRINT "après 100,B$ contient :";B$;",sa longueur est :";LEN(A$)
00120 A$=""
00130             PRINT "après 120,A$ contient :";A$;",sa longueur est :";LEN(A$)
00140 STOP
RUN
au début,A$ contient :,sa longueur est : 0
après 40,A$ contient :ABCDE,sa longueur est : 5
après 60,A$ contient :AZERTY  ,sa longueur est : 8
après 80,A$ contient :W"CVB",sa longueur est : 6
après 100,B$ contient :W"CVB",sa longueur est : 6
après 120,A$ contient :,sa longueur est : 0

BAS-5055 le programme s'est terminé à la ligne 140
```

Que constatons-nous en examinant les résultats affichés par ce programme ?

— au début du programme, avant toute exécution, il n'y a *rien* dans A$ et la longueur de cette chaîne est *zéro*.

— après exécution de l'instruction 40, la valeur de la constante ABCDE a été affectée à A$ et la longueur de A$ est devenue 5.

— après exécution de l'instruction 60, le contenu de A$ est AZERTY suivi de deux blancs et sa longueur est *devenue 8.* Les deux « blancs » font, en effet, partie de la chaîne A$.

— après exécution de 80,B$ contient W''CVB'' (puisque chaque paire de guillemets à l'intérieur d'une constante compte pour un seul guillemet).

— après exécution de 100 (affectation de B$ à A$), A$ a pris le contenu et la longueur de B$.

— après exécution de 120,A$ ne contient plus rien et sa longueur est NULLE. On dira alors qu'on a affaire à une *chaîne « nulle »* (« null string » en anglais).

En conclusion, il est important pour l'utilisateur de BASIC de ne jamais perdre de vue que le système entretient au fur et à mesure de l'exécution du programme, deux valeurs pour une chaîne donnée :

— son CONTENU
— sa LONGUEUR

D'autre part, il est possible d'indiquer au système la longueur maximale que peut prendre une chaîne donnée. Ceci se fait par une instruction DIM. Elle a la forme :

```
DIM Nom-de-chaîne-1*L1,Nom-de-chaîne-2*L2,etc...
```

L1, L2, etc... étant les longueurs maximales permises pour les chaînes 1, 2, etc...

Par exemple, l'instruction DIM ZONE$*24,ALPHA$*2 indiquera au programme que la longueur maximale de la chaîne ZONE$ est de 24 caractères et que celle de ALPHA$ est de 2.
Quand le nom d'une chaîne ne figure pas dans un ordre DIM, le système lui attribue une longueur maximale par défaut (elle est de 18 pour le système que nous avons utilisé).

Examinons ce qui se passe si la longueur réelle d'une chaîne devient, en cours d'exécution, supérieure à celle spécifiée par DIM ou prise par défaut. Pour ce faire, exécutons la séquence suivante :

```
00010 ! E102 : dépassement de capacité d'une chaine de caractères
00020 ! ----   ------------------------------------------------
00030 DIM A$*10
00040 A$="ABCDEF"
00050 A$="ABCDEFGHIJKLM"
00060 STOP
RUN

BAS-5046 ligne 50: dépassement de capacité pour zone caractère
```

Nous constatons que le système nous a sorti un message d'erreur indiquant que l'instruction 50 n'a pu être exécutée, car elle essayait de donner à la chaîne A$ une longueur supérieure (celle de la constante "ABCDEFGHIJKLM"-soit 13) à la longueur définie dans l'ordre DIM - soit 10.

Il est possible de tester cette condition d'erreur par un "ON" et, par exemple, de sortir un message d'erreur et d'arrêter le traitement. C'est ce que nous avons fait dans le programme listé ci-après. Le mot-clé SOFLOW indiqué après le ON signifie « dépassement de capacité sur une chaîne de caractères » (en anglais : String OverFLOw).

```
00010 ! E103 : dépassement de capacité d'une chaine de caractères
00020 ! ----   ------------------------------------------------
00030 !
00040 ON SOFLOW GOTO 140   ! détection de l'erreur et débranchement
00050 !
00060 DIM A$*10
00070 A$="ABCDEF"
00080 A$="ABCDEFGHIJKLM"
00090 STOP
00100 !
00110 ! Routine de traitement du dépassement de capacité
00120 !
00130   PRINT ""
00140   PRINT "Il y a eu une ERREUR à la ligne ";LINE
00150   PRINT ""
00160   PRINT "A$ contient maintenant :";A$;",sa longueur est :";LEN(A$)
00170   STOP
run
Il y a eu une ERREUR à la ligne  80

A$ contient maintenant :ABCDEF,sa longueur est : 6

BAS-5055 le programme s'est terminé à la ligne 170
```

Rappelons au lecteur que la fonction BASIC « LINE » permet de récupérer le n° de ligne de l'instruction qui a provoqué l'erreur.

Nous aurions pu également sortir le message et continuer le traitement. Dans le programme E105, le CONTINUE de l'instruction 170 va provoquer le retour du traitement à l'instruction, suivant celle où l'erreur s'est produite - 90 STOP dans notre cas.

```
00010 ! E105 : dépassement de capacité d'une chaine de caractères
00020 ! ----   ------------------------------------------------
00030 !
00040 ON SOFLOW GOTO 130   ! détection de l'erreur et débranchement
00050 !
00060 DIM A$*10
00070 A$="ABCDEF"
00080 A$="ABCDEFGHIJKLM"
00090 STOP
00100 !
00110 ! Routine de traitement du dépassement de capacité
00120 !
00130   PRINT " "
00140   PRINT "Il y a eu une erreur à la ligne ";LINE
00150   PRINT " "
00160   PRINT "A$ contient maintenant :";A$;",sa longueur est :";LEN(A$)
00170   CONTINUE
RUN

Il y a eu une erreur à la ligne  80

A$ contient maintenant :ABCDEF,sa longueur est : 6

 BAS-5055 le programme s'est terminé à la ligne 90
```

Il est également possible de demander au système d'ignorer l'erreur et de continuer le traitement. Il suffit d'indiquer le mot-clé IGNORE après ON SOFLOW. C'est ce que nous avons fait dans le programme E104 ci-dessous.

```
00010 ! E104 : dépassement de capacité d'une chaine de caractères
00020 ! ----   ------------------------------------------------
00030 !
00040 ON SOFLOW IGNORE  ! on ne tient pas compte de l'erreur
00050 !
00060 DIM A$*10
00070 A$="ABCDEF"
00080 A$="ABCDEFGHIJKLM"
00090 PRINT " "
00100 PRINT "A$ contient :";A$;",sa longueur est :";LEN(A$)
00110 STOP
run

A$ contient :ABCDEFGHIJ,sa longueur est : 10

 BAS-5055 le programme s'est terminé à la ligne 110
```

En regardant le résultat obtenu, on constate que le BASIC a mouvementé la constante ''ABCDEFGHIJKLM'' dans la zone A$ jusqu'à ce que celle-ci soit pleine, c'est-à-dire jusquà ce que sa longueur soit 10.

1.4 COMPARAISON DE DEUX CHAÎNES DE CARACTÈRES

Tout comme dans le cas de variables et de constantes numériques, la comparaison entre deux chaînes de caractères s'effectue par une instruction IF suivie de THEN et de ELSE. Cette instruction a la forme :

IF chaîne-1 opérateur chaîne-2 THEN A ELSE B

ou :

L'opérateur peut-être + (égal)
ou < > (non égal)
ou < (inférieur)
ou > (supérieur)
ou < = (inférieur ou égal)
ou > = (supérieur ou égal)

A et B peuvent être un n° de ligne, une étiquette symbolique ou une expression arithmétique ou caractère.

L'emploi de ELSE est optionnel : s'il est omis, le programme continue en séquence.

Voici quelques exemples d'écriture de cette instruction .

```
00010 ! E107 : exemple d'instructions IF ... THEN ... ELSE
00020 !
00030   IF A$<B$ THEN 100
00035 !
00040   IF A$=B$ THEN 110 ELSE 120
00045 !
00050   IF A$<B$ THEN IMPRI
00055 !
00060   IF A$<=B$ THEN IMPRI ELSE FIN
00065 !
00070   IF A$>=B$ THEN A$="1234"
00075 !
00080   IF A$<>B$ THEN A$="AZERTY" ELSE A$="WXCVB"
00085 !
00090   IF B$>"123456" THEN X=Y*2+.21 ELSE IMPRI
00095 !
00100   GOTO 110
00110   IMPRI:PRINT A$,B$
00120   FIN: STOP
```

Examinons de plus près le mécanisme de la comparaison de deux chaînes, A$ et B$ par exemple :

a) Le système compare le premier caractère A1 de A$ avec le premier caractère de B1 de B$.

b) Si le rang de A1 dans le jeu de caractère utilisé est inférieur à celui de B1, la comparaison donne A$ < B$ et la comparaison s'arrête.

c) Si le rang de A1 est supérieur à celui de B1, le résultat est : A$ < B$ et la comparaison s'arrête.

d) Si les rangs de A1 et B1 sont égaux, la comparaison reprend en a), en prenant les deux caractères suivants.

e) La comparaison peut continuer ainsi jusqu'à ce que la fin d'une chaîne ou des deux (si elles ont la même longueur) soit atteinte.
Dans ce cas :

— si les deux chaînes ont la même longueur, elles sont considérées comme égales.

— si leurs longueurs sont différentes, la plus courte des deux est prise comme étant inférieure à l'autre.

Vérifions ce dernier point sur l'exemple ci-dessous :

```
00010 ! E106 : comparaison de deux chaines de caractères
00020 ! ----   ----------------------------------------
00030 PRINT " "
00040   A$="ABCDE"
00050                 PRINT "A$ contient :";A$;", sa longueur est :";LEN(A$)
00060   B$="ABCDE  "
00070                 PRINT "B$ contient :";B$;", sa longueur est :";LEN(B$)
00080 !
00090 PRINT " "
00100 IF A$=B$ THEN PRINT " ---> A$ est égale à B$"
00110 IF A$>B$ THEN PRINT " ---> A$ est supérieure à B$"
00120 IF A$<B$ THEN PRINT " ---> A$ est inférieure à B$"
00130 STOP
run

A$ contient :ABCDE, sa longueur est : 5
B$ contient :ABCDE  , sa longueur est : 7

 ---> A$ est inférieure à B$

 BAS-5055 le programme s'est terminé à la ligne 130
```

Nous constatons que, bien que les deux chaînes A$ et B$ aient apparemment le même contenu, A$ est prise comme étant inférieure à B$ et B$ est plus longue que A$ à cause des deux blancs à droite.

En conclusion :

1. La recherche de l'égalité entre deux chaînes n'est valable *que si les deux chaînes ont la même longueur.*

2. Le résultat de la comparaison entre deux chaînes dépend du *jeu de caractères* utilisé.

1.5 INITIALISATION DE CHAÎNES DE CARACTÈRES PAR DATA ET READ

Les ordres DATA et READ peuvent servir à initialiser les chaînes de caractères aussi bien que les variables numériques. L'ordre DATA a la forme :

DATA constante-1,constante-2,constante-3,etc...

et place les valeurs alphanumériques ou caractères 1,2,3,etc. dans le fichier interne au programme.

L'ordre READ a la forme :

READ variable-1,variable-2,variable-3,etc...

Il charge dans les variables (numériques ou chaînes) les valeurs lues dans le fichier interne dans l'ordre où elles y ont été mises par DATA.

Plusieurs ordres READ et DATA peuvent être utilisés dans un même programme. Voici un exemple d'utilisation :

```
00010 ! E109 : exemple d'utilisation de DATA et READ
00020 ! ------  ------------------------------------
00030 DIM PRENOM$*25     ! la zone PRENOM$ a une longueur maxi de 25
00040 !
00050   DATA "DUVALLIER",17,23
00060   DATA -12.15," Gilbert,Léon,Paul "
00070   DATA 1,23
00080 !
00090   PRINT " "
00100   READ NOM$,S,D,W
00110   PRINT "NOM$ contient :",NOM$;",sa longueur est :",LEN(NOM$)
00120 !
00130   PRINT " "
00140   READ PRENOM$,X,Y
00150   PRINT "PRENOM$ contient :",PRENOM$,",sa longueur est :",LEN(PRENOM$)
00160 !
00170  STOP
run

NOM$ contient :DUVALLIER,sa longueur est : 9

PRENOM$ contient : Gilbert,Léon,Paul ,sa longueur est : 19

 BAS-5055 le programme s'est terminé à la ligne 170
```

REMARQUE 1 : L'ordre RESTORE sert à indiquer que la lecture par READ du fichier interne doit recommencer au début de ce fichier, c'est-à-dire à la première constante chargée par DATA.
L'exemple ci-dessous illustre l'emploi de RESTORE.

```
 00010 ! E110 : exemple d'utilisation de RESTORE
 00020 ! ----   ---------------------------------
 00030 !
 00040   DATA "Pierre","Paul","Jacques"
 00050 !
 00060   PRINT " "
 00070   READ XA$,XB$,XC$
 00080   PRINT "Après  70 : XA$ = ";XA$;",XB$ = ";XB$;",XC$ = ";XC$
 00090 !
 00100   PRINT " "
 00110   RESTORE
 00120   READ A1$,A2$,A3$
 00130   PRINT "Après 100 : A1$ = ";A1$;",A2$ = ";A2$;",A3$ = ";A3$
 00140   STOP
 run

Après  70 : XA$ = Pierre,XB$ = Paul,XC$ = Jacques

Après 100 : A1$ = Pierre,A2$ = Paul,A3$ = Jacques

 BAS-5055 le programme s'est terminé à la ligne 140
```

Dans ce programme :

— l'instruction 70 charge les chaînes XA$,XB$,XC$

— l'instruction 110 (RESTORE) repositionne le fichier interne sur le première constante chargée par DATA (''Pierre'').

— l'instruction 120 charge les chaînes A1$,A2$,A3$.

REMARQUE 2 : La condition d'erreur SOFLOW peut être détectée au cours d'une opération READ si la longueur de la constante spécifiée dans DATA est supérieure à celle de la chaîne réceptrice.

Exemple :

```
00010 ! E111 : Dépassement de capacité sur READ
00020 ! ----   ---------------------------------
00030 !
00040   DIM NOM$*10
00050 !
00060   DATA "CARAVANSERAIL"
00070 !
00080   READ NOM$
00090 !
run

BAS-5046 ligne 80: dépassement de capacité pour zone caractère
```

2

OPÉRATIONS D'ENTRÉE-SORTIE SUR LES CHAÎNES DE CARACTÈRES TABLEAUX DE CHAÎNES DE CARACTÈRES

2.1. ENTRÉE DE CHAÎNES DE CARACTÈRES PAR INPUT ET LINPUT

L'instruction INPUT du langage BASIC peut être utilisée pour entrer des chaînes de caractères soit à partir de fichiers en entrée non formattés, soit directement à partir du clavier. Cette instruction a deux formes :

```
INPUT variable-1,variable-2,variable-3,...
ou INPUT ''message de service'' : variable-1,variable-2,...
```

Variable-1,2,3,etc... sont des noms de variables numériques ou de chaînes de caractères. Le « message de service » est utilisable uniquement lorsqu'il s'agit d'une entrée à partir d'un terminal « clavier-écran ». Ce message s'affiche à l'écran et sert à guider l'opérateur.

Comme toute opération sur les chaînes, INPUT attribue à chaque chaîne une longueur. Voyons sur l'exemple ci-dessous comment, en pratique, l'instruction INPUT fonctionne :

```
 00010 ! E112 : exemple d'instruction INPUT
 00020 !
 00030   DIM NOMCLI$*20   ! NOMCLI$ a une longueur maxi de 20 caractères
 00040 !
 00050   INPUT "Entrer le code et le nom du client : ": CODCLI,NOMCLI$
 00060 !
 00070   PRINT "NOMCLI$ =";NOMCLI$;" longueur :";LEN(NOMCLI$)
 00080 !
 00090   GOTO 50
 RUN
Entrer le code et le nom du client :  145,DUPONT
NOMCLI$ =DUPONT longueur : 6
Entrer le code et le nom du client :  145,            DUPONT
NOMCLI$ =DUPONT longueur : 6
Entrer le code et le nom du client :  145,"   DUPONT    "
NOMCLI$ =   DUPONT    longueur : 12
Entrer le code et le nom du client :  145,"DUPONT"
NOMCLI$ =DUPONT longueur : 6
Entrer le code et le nom du client :  145,DUPONT DUVAL MARTINEAU
 BAS-7001 chaine de caractères invalide -- réentrer entièrement la ligne
```

Nous constatons que :

1. Les chaînes de caractères peuvent être entrées telles quelles ou entre guillemets.

2. *Si les guillemets sont utilisés :*
La chaîne de caractères prise en compte par BASIC va du caractère suivant le premier guillemet à celui précédant le second.

3. *Si les guillemets ne sont pas utilisés :*
La chaîne prise en compte va du premier caractère non blanc au dernier caractère non blanc de la valeur entrée.

4. Le système *refuse* toute zone dont la longueur est supérieure à celle indiquée par DIM (ou à 18, si DIM n'est pas spécifiée pour cette zone).

L'ordre INPUT ne permet pas d'entrer n'importe quelle chaîne. Nous pouvons le constater en exécutant le programme E112 précédent avec une entrée différente.

```
 00010 ! E112 : exemple d'instruction INPUT
 00020 !
 00030   DIM NOMCLI$*20   ! NOMCLI$ a une longueur maxi de 20 caractères
 00040 !
 00050   INPUT "Entrer le code et le nom du client : ": CODCLI,NOMCLI$
 00060 !
 00070   PRINT "NOMCLI$ =";NOMCLI$;" longueur :";LEN(NOMCLI$)
 00080 !
 00090   GOTO 50
 RUN
Entrer le code et le nom du client :  1147,DUVAL,MARTIN ET FILS
 BAS-7002 trop de données entrées-- réentrer entièrement la ligne
```

La zone que nous avons essayé d'entrer sans guillemets a été refusée par le système. En effet, les virgules incluses dans la zone frappée au clavier sont considérées par le système comme des séparateurs de zones. Celui-ci va considérer qu'il y a plus de zones frappées qu'il n'y en a dans la liste indiquée dans INPUT et sort donc un message d'erreur.

Pour pallier cet inconvénient, on a incorporé au langage BASIC l'instruction LINPUT. Celle-ci ressemble à INPUT, mais ne permet que l'entrée d'une seule chaîne. Elle s'écrit :

LINPUT nom-de-chaîne
ou LINPUT ''message de service'' : nom-de-chaîne

Analysons son fonctionnement sur le programme E114 ci-dessous :

```
 00010 ! E114 : exemple d'instruction LINPUT
 00020 !
 00030   DIM NOM$*30     ! NOM$ a une longueur maxi de 30 car.
 00040 !
 00050   LINPUT "entrer le nom du client : ": NOM$
 00060 !
 00070   PRINT "NOM$ =";NOM$;" longueur : ";LEN(NOM$)
 00080 !
 00090   GOTO 50
 RUN
entrer le nom du client :  DUPONT
NOM$ =DUPONT longueur :  6
entrer le nom du client :              DUPONT
NOM$ =          DUPONT longueur :  16
entrer le nom du client :  "  DUPONT  "
NOM$ ="  DUPONT  " longueur :  12
entrer le nom du client :  "DUPONT" "DUVAL"
NOM$ ="DUPONT" "DUVAL" longueur :  16
entrer le nom du client :  DUPONT,DUVAL,"X"
NOM$ =DUPONT,DUVAL,"X" longueur :  16
entrer le nom du client :
NOM$ = longueur :  0
entrer le nom du client :
```

Nous voyons que, sans utiliser les guillemets :

1. La chaîne saisie par le programme inclut les blancs entrés à gauche.

2. Il est possible d'entrer une chaîne de longueur nulle (ce que INPUT n'accepte pas).

2.2 IMPRESSION DE CHAÎNES DE CARACTÈRES PAR PRINT

Rappelons que l'ordre PRINT permet d'imprimer, d'afficher sur un écran ou d'écrire sur un fichier non formatté des zones et des constantes numériques ou caractères figurant dans la liste après le mot-clé PRINT.

Chaque élément de la liste peut être séparé du suivant par une virgule (,) ou un point-virgule (;).

```
PRINT (£nn) :  nom-1          , nom-2          , etc...
               constante-1    ; constante-2    ; etc...
```

La sortie obtenue est différente suivant le caractère séparateur employé. Nous allons le montrer en exécutant le programme suivant :

```
00010 ! E115 : exemple d'utilisation de l'instruction PRINT
00020 ! ~~~~   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00030 !
00040   DIM A$*20,B$*10
00050 !
00060   A$="ABCDEFGHIJKLMNOPQRST"
00070   B$="0123456789"
00080 !
00085   PRINT ""
00090   PRINT A$,B$,"abcdefghijklo"
00095   PRINT ""
00100   PRINT A$;B$;"abcdefghijklo"
00105   PRINT ""
00110   PRINT A$,B$;"abcdefghijklo"
RUN

ABCDEFGHIJKLMNOPQRST            0123456789      abcdefghijklo

ABCDEFGHIJKLMNOPQRST0123456789abcdefghijklo

ABCDEFGHIJKLMNOPQRST            0123456789abcdefghijklo

 BAS-5033 le programme s'est terminé à la ligne 99999/99
```

Nous constatons que :

1. si deux zones (variables ou constantes) sont séparées par un point-virgule, elles sont imprimées « collées » l'une à l'autre.

2. si une zone est séparée de la suivante par une virgule, la zone spécifiée avant la virgule sera imprimée en étant cadrée à gauche d'une zone de longueur fixe.

Cette longueur est multiple d'une longueur « standard » qui, dans le système utilisé, est de 16 par défaut. Ainsi, l'ordre PRINT de l'instruction 90 imprimera :

— la chaîne A$ (longueur 20) cadrée à gauche d'une zone de longueur 32 (2 fois 16)

— la chaîne B$ (longueur 10) et la constante caractère ''abcd...'' (longueur 13) cadrées dans deux zones de longueur 16 (une fois 16).

Il s'agit, bien entendu, des longueurs réelles et non de celles indiquées dans DIM.

La longueur standard de la zone d'impression peut être indiquée par le programmeur à l'aide de l'instruction OPTION PRTZO *l*, où *l* est la longueur standard désirée.

Vérifions-le en modifiant le programme précédent en indiquant une longueur standard de 8 (ligne 35) :

```
00010 ! E116 : exemple d'utilisation de l'instruction PRINT avec OPTION
00020 ! ------  ------------------------------------------------------
00030 !
00035   OPTION PRTZO 8
00040   DIM A$*20,B$*10
00050 !
00060   A$="ABCDEFGHIJKLMNOPQRST"
00070   B$="0123456789"
00080 !
00085   PRINT ""
00090   PRINT A$,B$,"abcdefghijklo"
00095   PRINT ""
00100   PRINT A$;B$;"abcdefghijklo"
00105   PRINT ""
00110   PRINT A$,B$;"abcdefghijklo"
RUN

ABCDEFGHIJKLMNOPQRST    0123456789      abcdefghijklo

ABCDEFGHIJKLMNOPQRST0123456789abcdefghijklo

ABCDEFGHIJKLMNOPQRST    0123456789abcdefghijklo

 BAS-5033 le programme s'est terminé à la ligne 99999/99
```

Nous constatons bien qu'à l'instruction 90, par exemple, A$ est imprimée dans une zone de 24 (3 fois 8).

2.3 ENTRÉES ET SORTIES FORMATTÉES DE CHAÎNES DE CARACTÈRES

En BASIC, ces entrées-sorties se font à l'aide d'instructions READ, WRITE,REWRITE,PRINT... USING. Le mot-clé USING se réfère à une instruction FORM décrivant le format de l'enregistrement à lire ou à écrire.

Dans ce format, la spécification C suivie d'un entier sert à indiquer au système la longueur de la zone en entrée ou en sortie suivant le cas. Voici quelques exemples d'utilisation de ces instructions :

1. Lecture d'un enregistrement.

```
50 READ £1,USING 60 : X,A$,Y
60 FORM N 3,C 12,N 5
```

La chaîne de caractère A$ est lue sur le fichier n° 1 et prend une longueur de 12 caractères.

2. Lecture d'un enregistrement

```
 50 DIM B$*30,C$*36
** ********************************
100 READ £,USING 110 : Q$,W$
110 FORM POS 10,C 25,POS 60,C 30
```

Les chaînes Q$ et W$ sont lues à partir des positions 10 et 60 de l'enregistrement, sur des longueurs de 25 et 30 caractères respectivement. Nous avons indiqué dans ce cas les longueurs maximales de Q$ et W$ dans un ordre DIM. Sans cette précaution, une erreur système SOFLOW (cf. § 1.3) se serait produite à la lecture. En effet, le système aurait tenté d'attribuer à Q$, par exemple, une longueur de 25 alors qu'en l'absence de DIM pour Q$, la longueur maxi par défaut est 18.

3. Impression d'une ligne

```
020 DIM NOM$*30,PRENOM$*20
*** ********************************
200 PRINT £5, USING 300 : ''Nom :'',NOM$,''prénom :'', PRENOM$

*** ********************************
300 FORM PO$ 10,C 7,C 35,C 10,C 20
```

Une ligne sera imprimée par l'instruction 200. Nous y trouverons à partir de la position 10 de la ligne et de gauche à droite :

— la constante ''Nom :'' sur 7 caractères (C 7)
— la valeur de NOM$ sur 35 caractères (C 35) bien que la longueur indiquée dans DIM soit 30
— la constante ''prénom :'' sur 10 caractères (C 10)
— la valeur de PRENOM$ sur 20 caractères.

En résumé :

Dans toutes les instructions d'entrée-sorties, les chaînes (variables ou constantes) sont complétées à droite par des blancs si la longueur indiquée après le ''C'' du FORM est plus grande que la longueur réelle de la chaîne.
Si l'inverse se produit (longueur après ''C'' inférieure à la longueur réelle), une erreur ''SOFLOW'' sera détectée.

Il est donc, en pratique, nécessaire de spécifier la longueur de chaque chaîne par une instruction DIM *si cette longueur dépasse la longueur maxi par défaut.*

2.4 TABLEAUX DE CHAÎNES DE CARACTÈRES

Tout comme pour les variables numériques, le BASIC permet de définir et d'utiliser des tableaux de chaînes de caractères. Les principes du traitement des tableaux de chaînes sont les suivants :

1. L'origine des indices (0 ou 1) est définie pour tout le programme et est valable pour tous les tableaux, numériques ou caractères, du programme :

- OPTION BASE 0 indique que cette origine est ZÉRO
- OPTION BASE 1 indique que cette origine est UN

Si aucune BASE n'est spécifiée, 0 est pris par défaut.

2. Le BASIC admet des tableaux à une dimension (vecteurs) ou deux dimensions (matrices). Les dimensions de chaque tableau et la longueur maximale de chaque élément sont indiqués dans une instruction DIM.
Celle-ci a la forme :

DIM nom-du-tableau(Lmax,Cmax)*lmax

Lmax est le N° de ligne maximal.
Cmax est le N° de colonne maximal (il est omis pour un tableau à une dimension).
lmax est la longueur maximale de chaque élément.

Rappelons que le nom du tableau doit se terminer par le caractère $.

Exemple 1.

```
10 OPTION BASE 0
20 DIM TA$(5)*12,TV$(10,2)*26
```

Ces instructions définissent :

— l'origine des indices en zéro
— un tableau de chaînes TA$ de 6 (5 + 1) lignes, la longueur maxi de chaque chaîne étant 12 car.
— un tableau de chaînes TV$ de 11 (10 + 1) lignes et 3 (2 + 1) colonnes, la longueur maxi de chaque élément étant de 26.

Exemple 2.

```
10 OPTION BASE 1
20 DIM SDW$(12,23)*205
```

Ces instructions définissent un tableau SDW$, de 12 lignes et 23 colonnes, chaque élément ayant une longueur maxi de 205 caractères. Dans cet ouvrage, nous utiliserons toujours l'origine des indices en 1, ce qui est le plus commode.

3. Un élément d'un tableau est repéré par son N° de ligne et éventuellement son N° de colonne :

Nom du tableau (N° de ligne, N° de colonne)

N° de ligne et N° de colonne sont des variables, des constantes ou plus généralement des expressions arithmétiques entières.

Exemples :

— GB$(5) désigne le cinquième élément du tableau GB$

— SXR$(3,6) désigne le sixième élément de la troisième ligne de SXR$.

— JOUR$(I + 9) désigne le I + 9e élément de JOUR$.

4. Le système entretient *individuellement la longueur de chaque élément d'un tableau* au cours des diverses opérations (affectation, entrées, sorties etc...).

5. Le mot clé MAT suivi d'un nom de tableau indique qu'une opération concerne tout le tableau.
Ainsi :

— l'instruction PRINT MAT DF$ indique que tous les éléments du tableau DF$ doivent être imprimés.

— L'instruction MAT W$ = (''XK45'') affectera la constante caractère ''XK45'' à tous les éléments du tableau W$.

2.5 EXEMPLES DE TRAITEMENT SUR LES TABLEAUX DE CHAÎNES

Plutôt que d'exposer d'une façon théorique le traitement des tableaux de chaînes de caractères, nous avons préféré commenter quelques programmes simples de manipulation de ces tableaux.

Tous ces exemples utilisent un tableau M$ destiné à contenir les noms des douze mois de l'année.

Exemple 1 : initialisation et impression du contenu d'un tableau, élément par élément.

```
00010 ! E120 : initialisation & impression d'un tableau,élément par élément
00020 !
00030   OPTION BASE 1                              ! l'origine des indices est en 1
00040   DIM M$(12)*9                               ! tableau de 12 éléments
00050                                              ! de 9 car. maxi
00060                                              ! initialisation des 12 éléments
00070   M$(1)="Janvier": M$(2)="Février": M$(3)="Mars": M$(4)="Avril"
00080   M$(5)="Mai": M$(6)="Juin": M$(7)="Juillet": M$(8)="Aout"
00090   M$(9)="Septembre":M$(10)="Octobre":M$(11)="Novembre"
00100   M$(12)="Décembre"
00110 !
00120   PRINT NEWPAGE                              ! remise à blanc de l'écran
00130   FOR I=1 TO 12                              ! début de la boucle d'impression
00140 !
00150   PRINT "l'élém.N°";I;" contient ";M$(I);" longueur ";LEN(M$(I))
00160 !
00170   NEXT I                                     ! fin de la boucle d'impression
```

```
 l'élém.N° 1  contient Janvier longueur  7
 l'élém.N° 2  contient Février longueur  7
 l'élém.N° 3  contient Mars longueur  4
 l'élém.N° 4  contient Avril longueur  5
 l'élém.N° 5  contient Mai longueur  3
 l'élém.N° 6  contient Juin longueur  4
 l'élém.N° 7  contient Juillet longueur  7
 l'élém.N° 8  contient Aout longueur  4
 l'élém.N° 9  contient Septembre longueur  9
 l'élém.N° 10  contient Octobre longueur  7
 l'élém.N° 11  contient Novembre longueur  8
 l'élém.N° 12  contient Décembre longueur  8
```

On notera la boucle d'impression (lignes 130 à 170) : chaque élément y est désigné par son indice I. La sortie sur écran permet de vérifier que chaque élément a bien sa longueur propre.

Exemple 2 : Initialisation par DATA et READ, impression par PRINT MAT.

```
00010 ! E121 : initialisation par DATA et READ,impression par PRINT MAT
00020 !
00030   OPTION BASE 1                          ! l'origine des indices est en 1
00040   DIM M$(12)*9                           ! tableau de 12 éléments
00050                                          !
00060                                          ! chargement du fichier interne
00070   DATA Janvier,Février,Mars,Avril,Mai,Juin
00080   DATA Juillet,Aout,Septembre,Octobre,Novembre,Décembre
00090 !
00100   READ MAT M$                            ! lecture fichier interne -> M$
00110   PRINT NEWPAGE                          ! remise à blanc de l'écran
00120   PRINT MAT M$                           ! impression non formattée de M$
```

Les ordres DATA des lignes 70 et 80 écrivent les noms de mois sur le fichier interne. L'instruction READ MAT de la ligne 100 charge le tableau M$ d'un seul coup. On aurait pu remplacer cette instruction par la séquence :

```
100 FOR I = 1 TO 12
101       READ M$(I)
102 NEXT I
```

Le PRINT MAT (ligne 120) est un ordre d'impression non formatté. D'une façon générale, cet ordre imprime ou affiche une ligne de tableau sur un ligne d'écran ou d'imprimante, chaque élément occupant une zone de longueur standard. La sortie continuera éventuellement sur la ligne suivante si la première ligne est pleine.

Exemple 3 : redimensionnement du tableau M$.

```
00010 ! E122 : initialisation par DATA et READ et redimensionnement
00020 !
00030   OPTION BASE 1                          ! l'origine des indices est en 1
00040   DIM M$(12)*9                           ! tableau de 12 éléments
00050                                          !
00060                                          ! chargement du fichier interne
00070   DATA Janvier,Février,Mars,Avril,Mai,Juin
00080   DATA Juillet,Aout,Septembre,Octobre,Novembre,Décembre
00090 !
00100   READ MAT M$                            ! lecture fichier interne -> M$
00110                                          !
00120   MAT M$=M$(4,3)                         ! redimensionnement de M$
00130                                          ! à 4 lignes et 3 colonnes
00140                                          !
00150   PRINT NEWPAGE                          ! remise à blanc de l'écran
00160   PRINT MAT M$                           ! impression non formattée de M$
```

```
Janvier         Février         Mars

Avril           Mai             Juin

Juillet         Aout            Septembre

Octobre         Novembre        Décembre
```

Le début de ce programme est identique au précédent. L'instruction 120, qui est une affectation précédée du mot-clé MAT, ne modifie ni le contenu, ni la longueur de chaque élément de M$. Par contre, elle va modifier ses dimensions : le (4,3) signifie que, désormais, M$ aura 4 lignes et 3 colonnes. On le vérifie par le PRINT MAT de la ligne 160 : M$ est bien imprimé sous forme d'une matrice 4 x 3.

Exemple 4 : impression d'un tableau par PRINT USING.

```
00010 ! E123 : impression d'un tableau par PRINT USING ... MAT
00020 !
00030   OPTION BASE 1                              ! l'origine des indices est en 1
00040   DIM M$(12)*9                               ! tableau de 12 éléments
00050                                              ! chargement du fichier interne
00060   DATA Janvier,Février,Mars,Avril,Mai,Juin
00070   DATA Juillet,Aout,Septembre,Octobre,Novembre,Décembre
00080 !
00090   READ MAT M$                                ! lecture fichier interne -> M$
00100   PRINT USING 110: MAT M$                    ! impression formattée de M$
00110   FORM POS 10,6*C 10
00120   PRINT ""                                   ! impression une ligne blanche
00130   PRINT USING 140: MAT M$                    ! impression formattée de M$
00140   FORM POS 10,5*C 12
RUN
        Janvier   Février   Mars      Avril     Mai       Juin
        Juillet   Aout      Septembre Octobre   Novembre  Décembre

        Janvier     Février     Mars        Avril       Mai
        Juin        Juillet     Aout        Septembre   Octobre
        Novembre    Décembre

BAS-5033 le programme s'est terminé à la ligne 99999/99
```

Nous avons imprimé M$ suivant deux formats différents :

— le premier (ligne 110) indique que la ligne d'impression commence en position 10 (POS 10) et comprend six zones d'impression de dix caractères chacune (6*C 10).

— le second (ligne 130) décrit une ligne d'impression commençant en 10 et composée de cinq zones de douze caractères.

L'analyse des sorties obtenues par les instructions 100 et 130 nous permet de comprendre le mécanisme de PRINT USING : le système remplit chaque zone du FORM avec les élements de M$. Lorsque toutes

les zones sont pleines, il sort la ligne et retourne au début du FORM et ceci jusqu'à ce que tous les éléments de M$ soient imprimés.

Exemple 5 : transfert de tableaux dans un autre.

```
00010 ! E124 : transfert d'un tableau dans un autre
00020 !
00030   OPTION BASE 1                          ! l'origine des indices est en 1
00040   DIM M$(12)*9,P$(6,2)*12                ! - M$ a 12 lignes ( lmax=9)
00045                                          !- P$ a 6 lig.& 2 col.(lmax=12)
00060                                          ! chargement du fichier interne
00070   DATA Janvier,Février,Mars,Avril,Mai,Juin
00080   DATA Juillet,Aout,Septembre,Octobre,Novembre,Décembre
00090 !
00100   READ MAT M$                            ! lecture fichier interne -> M$
00110   MAT P$=M$(3,4)     ! transfert de M$ dans P$ -> P$ a 3 lignes,4 colonnes
00120   PRINT MAT P$                           ! impression non formattée de P$
RUN

Janvier         Février         Mars            Avril

Mai             Juin            Juillet         Aout

Septembre       Octobre         Novembre        Décembre


BAS-5033 le programme s'est terminé à la ligne 99999/99
```

L'instruction de la ligne 110 sert tout d'abord à changer les dimensions de P$ (comme dans l'exemple 2 ci-dessus). Elles passent de 6 lignes et 2 colonnes indiquées dans DIM à 3 lignes et 4 colonnes indiquées à la ligne 110. Elle attribue ensuite le contenu et la longueur de chaque élément de M$ à l'élément correspondan tde P$.

Cette instruction 110 équivaut à :

P$(1,1)=M$(1) : etc... : P$(2,2)=M$(6) : etc... : P$(4,3)=M$(12)

Si, à la ligne 110 nous avions écrit simplement :

110 MAT P$ = M$ (sans indiquer de dimensions)

P$ aurait pris les mêmes dimensions que M$, c'est-à-dire 12 lignes et une colonne.

3

OPÉRATIONS ÉLÉMENTAIRES SUR LES CHAÎNES DE CARACTÈRES

Dans les deux chapitres précédents, nous avons appris (ou rappelé) au lecteur comment définir, initialiser les chaînes de caractères, comment les lire et les écrire sur un fichier. Les instructions que nous avons vues sont suffisantes pour programmer les applications classiques de gestion : tenue de stock, paie, comptabilité etc... où les zones ''caractère'' ont des longueurs FIXES, où les libellés occupent toujours la même place dans la facture ou le bulletin de paie.

Si maintenant nous voulons aborder les applications où on traite des mots de longueur VARIABLE - édition de lettres, génération de contrats, recherche de documentation, par exemple - nous constaterons rapidement que les opérations que nous avons vues jusqu'ici ne permettent de traiter ces applications qu'au prix d'une programmation plutôt complexe.

Essayons par exemple d'imaginer la programmation correspondant à l'opération la plus simple en traitement de texte : la « concaténation » c'est-à-dire l'assemblage bout à bout de deux chaînes pour en constituer une troisième. Il n'y a pas d'autre moyen que de considérer chaque chaîne comme un tableau d'éléments d'un caractère chacun et d'effectuer l'opération à l'aide de boucles FOR...NEXT. C'est ce qui est illustré par le programme ci-dessous :

```
00010 ! E300 : concaténation de deux chaines
00020 !
00030  DIM A$(20)*1,B$(20)*1,C$(40)*1      ! définition des tableaux
00040 !
00050  OPEN £1: "NAME=FICHIER"
00060 !
00070  READ £1,USING 80: MAT A$,MAT B$ EOF 210
00080  FORM 20*C 1,20*C 1
00090 !
00100  FOR I=1 TO 20                       ! transfert de A$ vers C$
00110      C$(I)=A$(I)
00120  NEXT I
00130 !
00140  FOR I=1 TO 20                       ! ajout de B$ dans C$
00150      C$(I+20)=A$(I)
00160 NEXT I
00170 !
00180 PRINT USING 190: MAT C$
00190 FORM 40*C 40
00200 GOTO 70
00210 STOP
```

Procéder de cette façon est, bien sûr, possible - c'est ce que font ceux qui programment le traitement de textes en RPG, par exemple -, mais en BASIC cela risque de conduire à des délais de programmation, des tailles de programme et des temps d'exécution importants, sinon prohibitifs.

C'est pourquoi les concepteurs du BASIC ont prévu *des opérateurs et fonctions « caractère »* qui jouent vis-à-vis du traitement de *texte* le même rôle que *les opérateurs et fonctions algébriques* vis-à-vis du *calcul.* Nous allons au cours de ce chapitre et des deux suivants passer en revue ces opérateurs et ces fonctions « caractère ».

3.1 OPÉRATEUR DE CONCATENATION

L'opérateur''&'' (ampersand en anglais) permet de concaténer, c'est-à-dire de mettre bout-à-bout deux chaînes de caractères.
L'opération :

A$ & B$

donne comme résultat une chaîne formée des contenus de A$ et B$ et ayant comme longueur l1 + l2, l1 et l2 étant les longueurs respectives de A$ et B$.

Le programme conversationnel ci-dessous permet de visualiser le fonctionnement de cet opérateur :

```
 00010 ! E301 : opérateur de concaténation
 00020 !
 00030   INPUT "Entrer les deux chaines à concaténer :": A$,B$
 00040 !
 00050   C$=A$&B$                          ! concaténation de A$ et B$ --> C$
 00060 !
 00070   PRINT "C$ contient ";C$;",sa longueur est ";LEN(C$)
 00080 !
 00090   GOTO 30
 RUN
Entrer les deux chaines à concaténer : ABC,123
C$ contient ABC123,sa longueur est  6
Entrer les deux chaines à concaténer : " ABC "," 123 "
C$ contient  ABC  123 ,sa longueur est  10
Entrer les deux chaines à concaténer : "",AZERTY
C$ contient AZERTY,sa longueur est  6
Entrer les deux chaines à concaténer : AZERTYUIOP,1234567890

 BAS-5046 ligne 50: dépassement de capacité pour zone caractère
```

On notera en fin de ce programme que la condition d'erreur SOFLOW (dépassement de capacité) peut être détectée.
Ajoutons que l'opérateur "&" peut être employé plusieurs fois dans une expression et peut servir à concaténer aussi bien des constantes que des variables « caractère ».

On peut également concaténer une chaîne sur elle-même : dans ce cas, le nom de la chaîne figurera à gauche et à droite du signe "=". Voici un exemple d'utilisation de cette possibilité.

```
 00010 ! E302 : constitution d'une chaine comprenant N fois les memes caracteres
 00020 !
 00030   DIM C$*255                       ! dimension maxi pour CAR$
 00040   INPUT "Entrer le caractere et le nombre N :": CAR$,N
 00050   IF N=0 THEN STOP
 00060 !
 00070   FOR I=1 TO N
 00080       C$=C$&CAR$
 00090   NEXT I
 00100 !
 00105   PRINT C$
 00110   C$=""                            ! remise à zéro de la chaine C$
 00130   GOTO 40
 RUN
Entrer le caractere et le nombre N : *,60
************************************************************
Entrer le caractere et le nombre N : /*/,20
/*//*//*//*//*//*//*//*//*//*//*//*//*//*//*//*//*//*/
Entrer le caractere et le nombre N : "AH ! ",10
AH ! AH ! AH ! AH ! AH ! AH ! AH ! AH ! AH ! AH !
Entrer le caractere et le nombre N : 0,0

 BAS-5055 le programme s'est terminé à la ligne 50/2
```

On remarquera l'instruction ligne 110 : elle affecte à C$ une chaîne de longueur nulle. Si elle avait été omise, le programme aurait continué à concaténer la nouvelle chaîne sur l'ancienne, jusqu'à avoir un dépassement de capacité.

3.2 OPÉRATIONS « SOUS-CHAÎNE »

L'opérateur sous-chaîne '':'' (en anglais : substring) permet d'extraire et de modifier des ensembles de caractères d'une chaîne donnée.

L'expression sous-chaîne a la forme :

```
A$(x:y)
```

ou A$ est une chaîne, *x* et *y* deux expressions arithmétiques positives et entières.

Voici quelques exemples d'expressions sous-chaînes :

```
A$(3:9)
WW$(X:Y)
XS$(X + 8:Y/6)
(A$&RS) (C-1:M)
```

Une expression sous-chaîne peut être employée de deux façons différentes :

— *à droite du signe égal,* elle sert à *extraire* un ensemble de caractères appartenant à une chaîne.

— *à gauche du signe égal,* elle sert à *modifier* une chaîne, c'est-à-dire à y remplacer des caractères par d'autres, à en supprimer ou à y insérer de nouveaux caractères.

Nous allons voir ces différentes opérations en détail.

3.3 EXTRACTION D'UNE SOUS-CHAÎNE

Dans ce cas, l'opérateur '':'' figure à droite du signe égal. L'expression :

A$(*x*:*y*)

extrait une sous-chaîne de longueur *y-x-1* contenant les caractères qui vont des positions *x* à *y* de A$. Voyons ce que cela donne sur le programme E303 qui suit :

```
 00010 ! E303 : extraction d'une sous-chaine d'une chaine
 00020 !
 00030   DIM CH$*32,SC$*32
 00040   CH$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 00050   INPUT "positions début & fin de la sous-chaine à extraire ?": DEB,FIN
 00060 !
 00070   SC$=CH$(DEB:FIN)     ! extraction sous-chaine depuis DEB jusqu'à FIN
 00080 !
 00090   PRINT " ---> la sous-chaine contient ";SC$;",sa longueur est ",LEN(SC
 00100   PRINT "": GOTO 50
 RUN
positions début & fin de la sous-chaine à extraire ? 5,8
 ---> la sous-chaine contient EFGH,sa longueur est  4

positions début & fin de la sous-chaine à extraire ? 20,20
 ---> la sous-chaine contient T,sa longueur est  1

positions début & fin de la sous-chaine à extraire ? 9,5
 ---> la sous-chaine contient ,sa longueur est  0

positions début & fin de la sous-chaine à extraire ? -1,20.56
 ---> la sous-chaine contient ABCDEFGHIJKLMNOPQRSTU,sa longueur est  21

positions début & fin de la sous-chaine à extraire ?
```

En parcourant les résultats obtenus, nous remarquons que :

— si *x* est supérieur à *y*, la chaîne extraite est de longueur nulle.

— si *x* ou *y* sont négatifs, ils sont pris égaux à 1 par le système.

— si *x* ou *y* ne sont pas entiers, ils sont arrondis à l'entier le plus proche.

3.4 INSERTION D'UNE CHAÎNE A L'INTÉRIEUR D'UNE AUTRE CHAÎNE

Pour cette opération et les suivantes, l'opérateur « sous-chaîne »

figure à gauche du signe égal. L'instruction utilisée pour l'insertion d'une chaîne dans une autre aura donc la forme suivante :

$$A\$(x:y) = B\$$$

où A$ est la chaîne à modifier, x la position de A$ avant laquelle on désire faire l'insertion, y un entier inférieur à x (la plupart du temps, on prend y égal à zéro) et B$ une expression caractère qui contient les caractères à insérer dans A$. Après exécution de cette instruction la longueur de la chaîne A$ est augmentée de celle de B$.

Nous avons illustré le fonctionnement de cette opération par le programme conversationnel ci-dessous :

```
00010 ! E304 : Insertion d'une chaine à l'intérieur d'une autre chaine
00020 !
00030  R$="123456789"
00040  PRINT NEWPAGE
00045  PRINT "R$ contient au départ ";R$: PRINT ""
00050 !
00060  INPUT "Entrer la chaine à insérer et la position d'insertion :": SC$,P
00070 !
00080  A$=R$                ! restauration de la valeur initiale de A$
00090  PRINT " --> avant insertion,A$ contient ";A$;",sa longueur est ";LEN(A$)
00100  A$(P:0)=SC$          ! insertion de SC$ à partir de la position P de A$
00110 !
00120  PRINT " --> après insertion,A$ contient ";A$;",sa longueur est ";LEN(A$)
00130  PRINT ""
00140 GOTO 60
```

```
R$ contient au départ 123456789

Entrer la chaine à insérer et la position d'insertion : AZER,5
 --> avant insertion,A$ contient 123456789,sa longueur est  9
 --> après insertion,A$ contient 1234AZER56789,sa longueur est  13

Entrer la chaine à insérer et la position d'insertion : ****,0
 --> avant insertion,A$ contient 123456789,sa longueur est  9
 --> après insertion,A$ contient ****123456789,sa longueur est  13

Entrer la chaine à insérer et la position d'insertion : +++++,20
 --> avant insertion,A$ contient 123456789,sa longueur est  9
 --> après insertion,A$ contient 123456789+++++,sa longueur est  14

Entrer la chaine à insérer et la position d'insertion : "     ",5
 --> avant insertion,A$ contient 123456789,sa longueur est  9
 --> après insertion,A$ contient 1234     56789,sa longueur est  14

Entrer la chaine à insérer et la position d'insertion : "   ",1
 --> avant insertion,A$ contient 123456789,sa longueur est  9
 --> après insertion,A$ contient    123456789,sa longueur est  12

Entrer la chaine à insérer et la position d'insertion :
```

En lisant les résultats obtenus, le lecteur peut voir que :

— si la position d'insertion *x* est inférieure ou égale à un, l'insertion se fait bien *avant* le premier caractère de la chaîne d'origine.

— si, par contre, *x* est supérieure à la longueur de la chaîne à modifier, l'insertion se fera *après* la dernière position de cette chaîne.

3.5 REMPLACEMENT D'UNE SOUS-CHAÎNE PAR UNE CHAÎNE

Il s'agit maintenant de remplacer un groupe de caractères appartenant à la chaîne à modifier, par un autre groupe formant une chaîne, les deux groupes n'ayant pas forcément la même longueur.

L'instruction de remplacement a exactement la même forme que l'instruction d'insertion vue au § précédent :

$$A\$(x:y) = B\$$$

A$ est la chaîne à modifier, B$ la chaîne ou l'expression « caractère » contenant la modification, *x* et *y* les positions de début et de fin, dans A$, des caractères à remplacer (*y* étant évidemment supérieure ou égal à *x*). La longueur de la chaîne après modification est égale à la longueur initiale de cette chaîne, augmentée de celle de la chaîne de remplacement et diminuée de celle de la sous-chaîne remplacée.

Voici un exemple de programme utilisant cette opération :

```
00010 ! E305 : Remplacememt d'une sous-chaine par une chaine
00020  R$="123456789"
00030  PRINT NEWPAGE        ! remise à blanc de l'écran
00035 !
00040  INPUT "Entrer les pos.début & fin et la chaine remplaçante :": D,F,SC$
00050 !
00060  A$=R$                ! restauration de la valeur initiale de A$
00070 !
00080  PRINT " --> avant remplact,A$ contient ";A$;",sa longueur est ",LEN(A$)
00090 !
00100  A$(D:F)=SC$          ! remplacement des positions D à F de A$ par SC$
00110 !
00120  PRINT " --> après remplact,A$ contient ";A$;",sa longueur est ";LEN(A$)
00130 !
00140  PRINT "": GOTO 40
```

```
Entrer les pos.début & fin et la chaine remplaçante : 2,2,***
 --> avant remplact,A$ contient 123456789,sa longueur est  9
 --> après remplact,A$ contient 1***3456789,sa longueur est  11

Entrer les pos.début & fin et la chaine remplaçante : 2,7,?
 --> avant remplact,A$ contient 123456789,sa longueur est  9
 --> après remplact,A$ contient 1?89,sa longueur est  4

Entrer les pos.début & fin et la chaine remplaçante : 3,4,"  "
 --> avant remplact,A$ contient 123456789,sa longueur est  9
 --> après remplact,A$ contient 12  56789,sa longueur est  9

Entrer les pos.début & fin et la chaine remplaçante : 0,0,ZZZZZ
 --> avant remplact,A$ contient 123456789,sa longueur est  9
 --> après remplact,A$ contient ZZZZZ123456789,sa longueur est  14

Entrer les pos.début & fin et la chaine remplaçante : 20,22,(((())))
 --> avant remplact,A$ contient 123456789,sa longueur est  9
 --> après remplact,A$ contient 123456789((((,sa longueur est  15

Entrer les pos.début & fin et la chaine remplaçante :
```

En lisant les résultats, le lecteur constatera notamment que si x et y sont nuls ou tous deux supérieurs à la longueur de A$, l'opération revient à une insertion en début ou en fin de chaîne.

3.6 SUPPRESSION D'UNE SOUS-CHAÎNE A L'INTÉRIEUR D'UNE CHAÎNE

Il s'agit tout simplement d'un cas particulier de l'opération de remplacement : pour supprimer un groupe de caractères ou sous-chaîne, il suffit de le remplacer par une chaîne de longueur nulle. l'instruction de suppression aura donc la forme :

A$(x:y) = '''' ou C$

C$ étant une chaîne ''nulle''.

où x et y sont les positions de début et de fin du groupe à supprimer.

L'exemple ci-dessous illustre cette opération.

```
00010 ! E306 : Suppression d'une sous-chaine
00020  R$="123456789"
00030  PRINT NEWPAGE       ! remise à blanc de l'écran
00035 !
00040  INPUT "Entrer les pos.début & fin des caractères à supprimer :": P1,P2
00050 !
00060  A$=R$               ! restauration de la valeur initiale de A$
00070 !
00080  PRINT " --> avant suppress.,A$ contient ";A$;",sa longueur est ";LEN(A$
00090 !
00100  A$(P1:P2)=SC$         ! Suppression des positions P1 à P2 de A$ par SC$
00110 !
00120  PRINT " --> après suppress.,A$ contient ";A$;",sa longueur est ";LEN(A$
00130 !
00140  PRINT "": GOTO 40
```

```
Entrer les pos.début & fin des caractères à supprimer : 2,5
 --> avant suppress.,A$ contient 123456789,sa longueur est  9
 --> après suppress.,A$ contient 16789,sa longueur est  5

Entrer les pos.début & fin des caractères à supprimer : 5,2
 --> avant suppress.,A$ contient 123456789,sa longueur est  9
 --> après suppress.,A$ contient 123456789,sa longueur est  9

Entrer les pos.début & fin des caractères à supprimer : 2,2
 --> avant suppress.,A$ contient 123456789,sa longueur est  9
 --> après suppress.,A$ contient 13456789,sa longueur est  8

Entrer les pos.début & fin des caractères à supprimer : 1,9
 --> avant suppress.,A$ contient 123456789,sa longueur est  9
 --> après suppress.,A$ contient ,sa longueur est  0

Entrer les pos.début & fin des caractères à supprimer :
```

3.7 CONVERSION D'UNE ZONE NUMÉRIQUE EN CHAÎNE DE CARACTÈRES ET VICE-VERSA

L'utilisateur est fréquemment amené à composer un texte avec des variables ou constantes « caractère » bien sûr, mais aussi avec des résultats de calcul. Les opérations que nous venons de voir (concaténation, insertion, etc...) ne prenant en compte que des chaînes, il faut convertir au préalable ces zones numériques en chaînes de caractères.

Inversement, on peut vouloir extraire d'un texte un chiffre à partir duquel on désire faire des calculs et il faut donc convertir une sous-chaîne en zone numérique. Les fonctions STR$ et VAL permettent de réaliser ces conversions.

La fonction STR$ (de l'anglais STRing) a la forme :

STR$(N)

Elle convertit l'expression numérique N en chaîne de caractères. Voyons sur le programme-exemple ci-dessous comment marche cette fonction :

```
00010 ! E307 : conversion d'une zone numérique en chaine
00020 !
00030   INPUT "Entrer le nombre N :": N
00040 !
00050   CH$=STR$(N)    ! conversion de N en chaine
00060 !
00070   PRINT "---> N converti en chaine contient ";N;" longueur :";LEN(CH$)
00080   PRINT "": GOTO 30
run
Entrer le nombre N : 12345
---> N converti en chaine contient  12345  longueur : 5

Entrer le nombre N : 000145
---> N converti en chaine contient  145  longueur : 3

Entrer le nombre N : -12.45
---> N converti en chaine contient -12.45  longueur : 6

Entrer le nombre N : 12345678
---> N converti en chaine contient  1.23457E+7  longueur : 10

Entrer le nombre N :
```

Nous constatons que :

— la fonction STR$ supprime les zéros non significatifs
— elle tient compte du signe et du point décimal
— elle édite la valeur numérique en notation exponentielle (E+...) si cette dernière dépasse la longueur « standard ».

La fonction inverse VAL (de l'anglais VALue) a la forme :

VAL(A$)

où A$ est une chaîne de caractères, ou plus généralement une expression caractère.

En lisant l'exemple ci-après, nous voyons que la fonction VAL fonctionne bien à l'inverse de CHR$. Notons toutefois que si nous essayons de convertir en zone numérique une chaîne comprenant des caractères non numériques, une erreur de conversion (CONV) se produit.

```
 00010 ! E308 : conversion d'une chaine en zone numérique
 00020 !
 00030   INPUT "Entrer la chaine CH$ :": CH$
 00040 !
 00050   N=VAL(CH$)    ! conversion de CH$ en zone numérique
 00060 !
 00070   PRINT "---> CH$ convertie en zone numérique contient : ",N
 00080   PRINT "": GOTO 30
 run
Entrer la chaine CH$ : 1234
---> CH$ convertie en zone numérique contient :  1234

Entrer la chaine CH$ : -12.456
---> CH$ convertie en zone numérique contient : -12.456

Entrer la chaine CH$ : +678
---> CH$ convertie en zone numérique contient :  678

Entrer la chaine CH$ : DF56

 BAS-5060 la ligne 50 contient une erreur de conversion de données
```

3.8 CONCLUSION : EXPRESSIONS « CARACTÈRE

Au cours de ce chapitre, nous avons étudié des opérateurs ''caractère'' - ''&'' et '':'' - et une fonction ''caractère'' - STR$. Ces opérateurs et fonctions permettent d'écrire, comme nous l'annoncions au début du présent chapitre, des expressions ''caractère'' : les règles d'écriture de ces expressions sont les mêmes que pour les expressions algébriques. Les parenthèses, en particulier y jouent le même rôle. Voici quelques exemples d'expressions ''caractère'' dont vous pourrez évaluer le résultat :

```
00010 ! E309 : EXEMPLES D'EXPRESSIONS "CARACTERE".
00020 !
00030         A$="0123456789"
00040         B$="ABCDEFGHIJ"
00050 !
00060         XA$=(B$(5:8)&STR$(12345)&A$(7:9))(2:11)
00070 !
00080         XB$=B$(5:11)&STR$(-12.998)(3:6)
00090 !
00100         XC$=B$
00110         XC$(VAL(A$(1:2)):VAL(A$(5:5)))="*"
00120 !
00130         XD$=A$(2:5)
00140         XD$(3:0)=STR$(VAL(A$(2:6))-152)&" WWW "
00150 !
00160         XE$=B$
00170         FOR I=1 TO 4
00180             XE$(I:I)="+++"
00190         NEXT I
00200 !
00210         XF$=A$
00220         FOR J=6 TO 2 STEP -1
00230             XF$(J:4)="**"
00240         NEXT J
00250 !
00260         PRINT NEWPAGE
00270         PRINT "XA$=";XA$: PRINT ""
00280         PRINT "XB$=";XB$: PRINT ""
00290         PRINT "XC$=";XC$: PRINT ""
00300         PRINT "XD$=";XD$: PRINT ""
00310         PRINT "XE$=";XE$: PRINT ""
00320         PRINT "XF$=";XF$: PRINT ""
```

```
XA$=FGH1234567

XB$=EFGHIJ2.99

XC$=ABCDEFGHIJ

XD$=1212193 WWW 34

XE$=+++++++++BCDEFGHIJ

XF$=0*****4**56789
```

EXERCICES

1. Édition d'une date en clair

On demande d'éditer en clair une date entrée au clavier sous forme numérique JJMMAA. Par exemple :

10578 sera éditée sous la forme 1er mai 1978
271145 sera éditée sous la forme 27 novembre 1945

Voici la liste du programme correspondant et un exemple de sorties au terminal.

```
00010 ! E311 : Edition d'une date en clair
00020 !
00030   OPTION BASE 1
00040 !
00050   DIM TMN$(12)*10              ! définition de la table des noms de mois
00060   DATA Janvier,Février,Mars,Avril,Mai,Juin,Juillet,Aout,Septembre
00070   DATA Octobre,Novembre,Décembre
00080   READ MAT TMN$                ! chargement de la table des noms de mois
00090 !
00100   PRINT NEWPAGE
00110 !
00120   LINPUT "Entrer la date : ": D$
00130   IF D$="" THEN STOP                          ! arret si longueur date nulle
00140   IF LEN(D$)=6 THEN SUITE                     ! test si date à 6 car.
00150   IF LEN(D$)=5 THEN SUITA                     ! test si date à 5.car.
00160 !
00170   PRINT D$,": date invalide"                  ! affich.message si long.erronée
00180   PRINT " "
00190   GOTO 120
00200 !
00210   SUITA: D$="0"&D$                            ! zéro à gauche si 5 de long.
00220   SUITE: J$=D$(1:2)                           ! extraction du jour
00230          IF J$(1:1)="0" THEN J$(1:1)=" "      ! remplacemt  zéro à gauche jour
00240          IF J$(1:2)=" 1" THEN J$="1er"        ! traitement premier du mois
00250          M=VAL(D$(3:4))                       ! extract./convers. mois en num.
00260          M$=TMN$(M)                           ! consultation tableau nom mois
00270          A$=" 19"&D$(5:6)                     ! composition année
00280          DCLAIR$=J$&" "&M$&A$                 ! composition date en clair
00290 !
00300   PRINT "la date en clair est : ";DCLAIR$
00310   PRINT "": GOTO 120
```

```
Entrer la date :  11281
la date en clair est : 1er Décembre 1981

Entrer la date :  011280
la date en clair est : 1er Décembre 1980

Entrer la date :  150182
la date en clair est : 15 Janvier 1982

Entrer la date :  111111
la date en clair est : 11 Novembre 1911

Entrer la date :  011200
la date en clair est : 1er Décembre 1900

Entrer la date :  1234
1234            : date invalide

Entrer la date :
```

Ce programme ne présente pas de difficultés particulières. On notera simplement l'emploi de la fonction VAL (ligne 250) pour convertir le mois qui va servir d'indice en numérique.

2. Édition d'une somme en toutes lettres.

Il s'agit d'un problème très ancien qui s'est posé au tout début de l'ère informatique lorsqu'il s'est agi d'imprimer des chèques, des traites etc...

par ordinateur et on ne le résolvait pratiquement qu'en langage machine ou en assembleur.

On entre au clavier du terminal une somme en CENTIMES de 11 chiffres maximum et on demande d'éditer cette somme en toutes lettres (majuscules), en respectant si possible la grammaire.

```
00010 ! E312 : Edition d'une somme en toutes lettres
00020 !
00030   OPTION BASE 0,LPREC                    ! origine des indices en ZERO
00040                                          ! et calcul en double précision
00050   DIM TC$*255,TF$*255,L$*255             ! chaines pour édition
00060 !
00070   DIM TU$(19)*8                          ! définition de la table des unités
00080   DATA "",UN,DEUX,TROIS,QUATRE,CINQ,SIX,SEPT,HUIT,NEUF,DIX,ONZE,DOUZE
00090   DATA TREIZE,QUATORZE,QUINZE,SEIZE,DIX SEPT,DIX HUIT,DIX NEUF
00100   READ MAT TU$                           ! chargement de la table des unités
00110 !
00120   DIM TD$(9)*20                          ! définition table des dizaines
00130   DATA "",DIX,VINGT,TRENTE,QUARANTE,CINQUANTE,SOIXANTE,SOIXANTE DIX
00140   DATA QUATRE VINGT,QUATRE VINGT DIX
00150   READ MAT TD$ EOF 160                   ! chargement table des dizaines
00160 !
00170   PRINT NEWPAGE                          ! remise à blanc écran
00180   INPUT "Entrer la somme à éditer en centimes : ": S
00190   IF S=0 THEN STOP                       ! arret si somme nulle
00200 !
00210   TC$="": TF$=""                         ! RAB des libellés cents et francs
00220   CT=REM(S,100)                          ! calcul des centimes
00230   R=INT(S/100)
00240   F=REM(R,1000)                          ! calcul des francs
00250   T=INT(R/1000)
00260   K=REM(T,1000)                          ! calcul des Kfrancs
00270   T=INT(T/1000)
00280   M=REM(T,1000)                          ! calcul des Mfrancs
00290 !
00300   IF CT=0 THEN 340                       ! centimes nuls
00310   CDU=CT: GOSUB 560                      ! éditions des centimes
00320   TC$=L$&" centime"                      !
00330   IF CT>1 THEN TC$=TC$&"s"               ! mise au pluriel
00340   IF R=0 THEN 500                        ! francs nuls
00350   CDU=F: GOSUB 560                       ! édition des francs
00360   TF$=L$                                 !
00370   IF K=0 THEN 410                        ! Kfrancs nuls
00380   CDU=K: GOSUB 560                       ! édition des Kfrancs
00390   IF K=1 THEN L$=""                      ! mille francs
00400   TF$=L$&" MILLE "&TF$                   !
00410   IF M=0 THEN 450                        ! Mfrancs nuls
00420   CDU=M: GOSUB 560                       ! édition des Mfrancs
00430   IF M>1 THEN S$=" MILLIONS " ELSE S$=" million "
00440   TF$=L$&S$&TF$                          !
00450   S$=" francs "
00460   IF R=1 THEN S$=" franc "               ! singulier pour 1 Franc
00470   IF K=0 AND F=0 THEN S$="de francs " ! il n'y a que des Mfrancs
00480   TF$=TF$&S$&TC$                         ! libellé final
00490   GOTO 520
00500   TF$=TC$                                ! libellé final ( francs nuls)
00510 !
00520   PRINT "somme en lettres : ";TF$
00530   PRINT "": GOTO 180
00540 !
```

```
00560 ! Sous-programme d'édition sur trois chiffres
00570 ! ( le libellé composé a partir de CDU est L$ )
00580 !
00590   L$=""
00600   IF CDU=0 THEN RETURN                    ! retour si nul
00610   DU=REM(CDU,100)                         ! calcul dizaines-unités
00620   C=INT(CDU/100)                          ! calcul chiffre des centaines
00630   IF DU>0 THEN 670                        !
00640   L$=TU$(C)&" CENTS"                      ! libellé si diz-unité nulles
00650   IF C=1 THEN L$="CENT "                  ! exception 100
00660   RETURN
00670   GOSUB 760                              ! appel s/p édition 2 chiffres
00680   IF C=0 THEN RETURN                      ! centaines nulle
00690   IF C>1 THEN 720                         !
00700   L$="CENT "&L$                           ! composition libellé si 1xx
00710   RETURN
00720   L$=TU$(C)&" CENT "&L$                   ! composition libellé
00730   RETURN
00740 !
00750 !
00760 ! Sous-programme d'édition sur deux chiffres
00770 ! ( le libellé composé  à partir de DU est L$ )
00780 !
00790   U=REM(DU,10)                            ! calcul du chiffre des unités
00800   D=INT(DU/10)                            ! calcul du chiffre des dizaines
00810   IF DU>19 THEN 840                       !
00820   L$=TU$(DU)                              ! libéllé pour chiffres 1 à 20
00830   RETURN                                  !
00840   IF U>0 THEN 880                         !
00850   L$=TD$(D)                               ! libellé pour chiffres terminé 0
00860   IF DU=80 THEN L$=L$&"S"                 ! exception pour 80
00870   RETURN
00880   IF U=1 THEN CL$=" et " ELSE CL$=" "     ! determination liaison diz.-unité
00890   IF DU=81 OR DU=91 THEN CL$=" "          ! exception 81 et 91
00900   IF D=7 OR D=9 THEN 930                  ! exception 7x et 9x
00910   L$=TD$(D)&CL$&TU$(U)                    ! libellés pour 3x,4x,5x,6x,8x
00920   RETURN
00930   L$=TD$(D-1)&CL$&TU$(U+10)               ! libellés pour 7x et 9x
00940   RETURN                                  ! **** fin du S/P *****
```

```
Entrer la somme à éditer en centimes :  12
somme en lettres : DOUZE centimes

Entrer la somme à éditer en centimes :  189
somme en lettres : UN franc QUATRE VINGT NEUF centimes

Entrer la somme à éditer en centimes :  11100001
somme en lettres : CENT ONZE MILLE  francs UN centime

Entrer la somme à éditer en centimes :  50600000104
somme en lettres : CINQ CENT SIX MILLIONS UN francs QUATRE centimes

Entrer la somme à éditer en centimes :  80200000000
somme en lettres : HUIT CENT DEUX MILLIONS de francs

Entrer la somme à éditer en centimes :  99999999999
somme en lettres :
NEUF CENT QUATRE VINGT DIX NEUF MILLIONS NEUF CENT QUATRE VINGT DIX NEUF MILLE
EUF CENT QUATRE VINGT DIX NEUF francs QUATRE VINGT DIX NEUF centimes

Entrer la somme à éditer en centimes :  1
somme en lettres : UN centime

Entrer la somme à éditer en centimes :
```

Ce programme est nettement plus complexe et plus long que le précédent car il doit prendre en compte tous les pluriels et les diverses exceptions aux règles de base. Pour simplifier la compréhension du programme, nous avons écrit deux sous-programmes, l'un appelant l'autre, pour éditer les nombres à deux et trois chiffres (lignes 560-730 et 740-940).

Pour simplifier la programmation, nous aurions pu charger et utiliser un tableau de chaînes correspondant aux libellés des nombres de 1 à 99, mais il nous a paru plus intéressant de programmer toutes les exceptions, une par une.

Ce programme est un bon exemple de l'emploi de l'opération de concaténation.

3. Génération d'un histogramme horizontal

Les programmes de gestion aboutissent bien souvent à des résultats qu'on désire visualiser sous forme d'histogramme, c'est-à-dire sous forme de segments dont la longueur ou la hauteur est proportionnelle à la valeur à représenter.

Pour cet exercice, on entre au clavier un mois (02/82, par exemple) et quatre chiffres représentant les ventes mensuelles de quatre articles A, B, C et D.

Après chaque entrée, on imprime le mois et une chaîne de 100 car. composée de quatre segments. Chaque segment représente les ventes d'un article et se compose d'un caractère répété p fois, p étant le pourcentage relatif de la vente de l'article considéré par rapport à la vente totale. On utilise le '' + '' pour l'article A, ''o'' pour B, le ''-'' pour C et le ''*'' pour D.

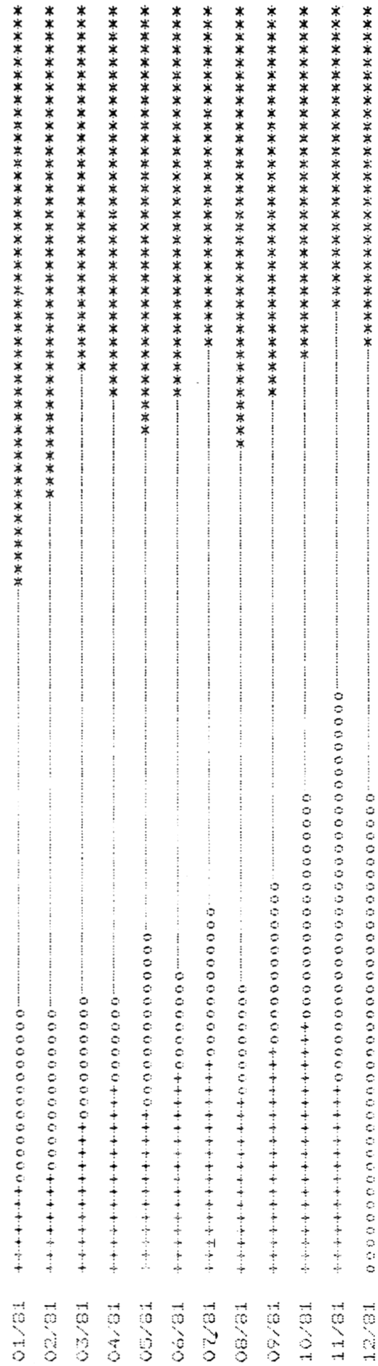
Par exemple si les ventes de A,B,C et D ont été de 50, 60, 80 et 10 recpectivement, cette chaîne se composera de 25 (50/2) ''+'', de 30''o'', de 40''-'' et de 5''*''.

Voici la liste du programme correspondant :

```
00010 ! E314 : Génération d'un histogramme horizontal
00020 !
00030   DIM LIGNE$*100
00040   PRINT NEWPAGE              ! remise à blanc de l'écran
00050   INPUT "Entrer le mois et les ventes de A,B,C,D :": M$,VA,VB,VC,VD
00060   IF M$="FIN" THEN STOP
00070 !
00080   VT=VA+VB+VC+VD             ! calcul des pourcentages relatifs
00090   PA=ROUND(VA*100/VT,0)
00100   PB=ROUND(VB*100/VT,0)
00110   PC=ROUND(VC*100/VT,0)
00120   PD=100-PA-PB-PC
00130 !
00140   LIGNE$=""                  ! chaine nulle dans ligne
00150 !
00160   FOR I=1 TO PA              ! boucle pour constituer segmemt corresp.à A
00170       LIGNE$=LIGNE$&"+"      ! concaténation d'un +
00180   NEXT I                     ! fin de boucle
00190   FOR I=1 TO PB              ! boucle pour constituer segmemt corresp.à B
00200       LIGNE$=LIGNE$&"o"      ! concaténation d'un -
00210   NEXT I                     ! fin de boucle
00220   FOR I=1 TO PC              ! boucle pour constituer segmemt corresp.à C
00230       LIGNE$=LIGNE$&"-"      ! concaténation d'un x
00240   NEXT I                     ! fin de boucle
00250   FOR I=1 TO PD              ! boucle pour constituer segmemt corresp.à D
00260       LIGNE$=LIGNE$&"*"      ! concaténation d'un *
00270   NEXT I                     ! fin de boucle
00280   PRINT £255,USING 290: M$,LIGNE$," "
00290   FORM C 5,POS 8,C 100,SKIP,C 1
00295   PRINT ""
00300   GOTO 50
```

Les entrées et l'histogramme édité :

```
Entrer le mois et les ventes de A,B,C,D : 01/81,12,23,56,78
Entrer le mois et les ventes de A,B,C,D : 02/81,15,25,78,75
Entrer le mois et les ventes de A,B,C,D : 03/81,21,18,85,51
Entrer le mois et les ventes de A,B,C,D : 04/81,31,15,99,65
Entrer le mois et les ventes de A,B,C,D : 05/81,32,35,100,87
Entrer le mois et les ventes de A,B,C,D : 06/81,44,22,125,89
Entrer le mois et les ventes de A,B,C,D : 07/81,78,56,202,123
Entrer le mois et les ventes de A,B,C,D : 08/81,99,66,302,256
Entrer le mois et les ventes de A,B,C,D : 09/81,120,88,252,201
Entrer le mois et les ventes de A,B,C,D : 10/81,58,52,99,78
Entrer le mois et les ventes de A,B,C,D : 11/81,42,89,85,71
Entrer le mois et les ventes de A,B,C,D : 12/81,0,45,41,32
```


01/81
02/81
03/81
04/81
05/81
06/81
07/81
08/81
09/81
10/81
11/81
12/81

Ce programme utilise des boucles FOR ... NEXT (instructions 160, 190, 220 et 250) pour constituer des chaînes de longueur proportionnelle à un entier. Rappelons au lecteur que la fonction BASIC « ROUND », employée lignes 90 à 110 permet l'arrondi à la valeur la plus proche (ici, à l'entier le plus proche).

4. Détection des mots dans un texte

Un texte se compose de mots séparés par des blancs et/ou des signes de ponctuation. Pour cet exercice, on lit à partir du clavier une ligne de texte (64 car. maxi). On demande de séparer chaque mot de la ligne du mot suivant par un''/'' en supprimant les blancs et les signes de ponctuation rencontrés. On considèrera qu'il peut y avoir plusieurs caractères séparateurs (blancs ou signes de ponctuation) à la suite.

La ligne modifiée sera affichée à l'écran du terminal.

La programme, les entrées et les sorties sur écran se trouvent ci-après:

```
00010 ! E315 : Détection des mots à l'intérieur d'une chaine de caractères
00020 !
00030   DIM CH$*128
00040 !
00050   TCS$=" ;,:.'""?()"              ! chargement chaine des car.séparateurs
00060  PRINT NEWPAGE
00070   LINPUT "Entrer la chaine à modifier :": CH$
00080  IF CH$="" THEN STOP             ! arret si chaine "nulle"
00090 !
00100   CH$="."&CH$&"."                 ! séparateur devant & derrière la chaine
00110   FOR I=1 TO LEN(CH$)             ! boucle remplacemnt separateurs par "/'
00120       FOR J=1 TO LEN(TCS$)        ! boucle de test si séparateur
00130           IF CH$(I:I)<>TCS$(J:J) THEN 160
00140           CH$(I:I)="/"            ! remplacement séparateur par "/"
00150           GOTO 170
00160       NEXT J                      ! fin de boucle de test
00170   NEXT I                          ! fin de boucle de remplacement
00180                                   !
00190                                   ! boucle de suppression des "/" contigus
00200   I=1                             ! initialisation indice
00210   IF CH$(I:I)<>"/" THEN 260       ! test si "/"
00220   IF CH$(I+1:I+1)<>"/" THEN 260   ! test si "/" dans position suivante
00230   PRINT "la chaine intermèd.contient :";CH$
00240   CH$(I+1:I+1)=""                 ! suppression du "/" pos.suivante
00250   IF I=LEN(CH$) THEN 300 ELSE 220 ! test si terminé
00260   IF I=LEN(CH$) THEN 300 ELSE 270 ! test si terminé
00270   I=I+1                           ! incrémentation indice
00280   GOTO 210                        ! retour suite boucle
00290                                   ! fin de la boucle de suppression
00300   PRINT ""
00310   PRINT "La chaine modifiée contient : ";CH$
00320   PRINT "": GOTO 70
```

```
Entrer la chaine à modifier : ERT;,,,,::RTY  ,1,2,,3

la chaine intermèd.contient :/ERT///////RTY///1/2//3/
la chaine intermèd.contient :/ERT//////RTY///1/2//3/
la chaine intermèd.contient :/ERT/////RTY///1/2//3/
la chaine intermèd.contient :/ERT////RTY///1/2//3/
la chaine intermèd.contient :/ERT///RTY///1/2//3/
la chaine intermèd.contient :/ERT//RTY///1/2//3/
la chaine intermèd.contient :/ERT/RTY///1/2//3/
la chaine intermèd.contient :/ERT/RTY//1/2//3/
la chaine intermèd.contient :/ERT/RTY/1/2//3/

La chaine modifiée contient : /ERT/RTY/1/2/3/

Entrer la chaine à modifier :
```

Cet exercice illustre bien une des fonctions de base du traitement de textes. La recherche de mots est utilisée aussi bien dans le traitement de documentation (recherche de mots-clés, indexation automatique, etc...) que dans la gestion des textes.

L'« isolation » de chaque mot se fait en deux temps :

a) On remplace chaque séparateur figurant dans la chaîne TCS$ par un ''/'' à l'aide de deux boucles FOR ... NEXT imbriquées (instructions 110 à 170). On notera qu'on place systématiquement un séparateur quelconque devant et derrière la chaîne à modifier (instruction 100).

b) On explore à nouveau la chaîne (instructions 200 à 280). A chaque fois qu'on trouve un ''/'', on regarde si le caractère suivant est également un ''/'' (lignes 210 et 220). Si oui, on supprime ce deuxième ''/'' (ligne 240) par insertion d'une chaîne « nulle » et ceci tant qu'on trouve un ''/'' en position I + 1 (ligne 250). Il est à remarquer que cette boucle de suppression n'aurait pu être programmée à l'aide d'un FOR I = 1 TO LEN(CH$) car la longueur de CH$ varie constamment au cours da la boucle. Pour visualiser cette évolution, nous avons affiché les valeurs intermédiaires de CH$ (voir sorties ci-dessus).

4

RECHERCHE DE CARACTÈRES DANS UNE CHAÎNE ET DANS UN TABLEAU

La recherche de caractères et de groupes de caractères est sans nul doute la fonction la plus importante du traitement de texte. L'exercice N° 4 du chapitre précédent nous montre que, même dans un cas simple, la programmation d'une telle recherche est plutôt laborieuse et exige un nombre non négligeable d'instructions si nous n'utilisons que les opérateurs et les fonctions vus au chapitre précédent.

Les fonctions que nous allons étudier dans ce chapitre vont nous permettre de programmer avec le minimum d'instructions de telles recherches.

4.1. FONCTION POS DE RECHERCHE

La fonction BASIC «POS» (de l'anglais POSition) a la forme suivante :

POS(A$,B$,*d*)

Cette fonction :

— explore la chaîne A$ à partir de la position *''d''* pour y trouver une sous-chaîne égale à B$.

— si elle trouve l'égalité, elle donne la position du premier caractère de la sous-chaîne dans A$.

— si elle ne trouve pas l'égalité, la fonction prend la valeur *zéro.*

Explicitons le fonctionnement de POS par le petit programme conversationnel ci-dessous.

```
00010 ! E401 : exemple de fonctionnement de la fonction POS
00020 !
00030 DIM CH$*64
00040 CH$="0102030405060708091011121314151617181920212223242526 27"
00050 PRINT NEWPAGE
00060 PRINT "CH$ contient ";CH$
00070 !
00080  PRINT ""  ! impression une ligne blanche
00090 !
00100  INPUT "Entrer la chaine à rechercher et la position de début :": R$,DEB
00110                          !-----------------------------------------!
00120   POSIT=POS(CH$,R$,DEB)  ! recherche de la position de R$ dans CH$  !
00130                          ! à partir de la position DEB              !
00140                          !-----------------------------------------!
00150  IF POSIT=0 THEN 160 ELSE 180  ! test pour savoir si R$ a été trouvée
00160 !
00170  PRINT R$," ne se trouve pas dans CH$": GOTO 80
00180 !
00190  PRINT R$," se trouve dans CH$ en position ";POSIT: GOTO 80
```

```
CH$ contient 0102030405060708091011121314151617181920212223242526 27

Entrer la chaine à rechercher et la position de début : 02,1
02 se trouve dans CH$ en position  3

Entrer la chaine à rechercher et la position de début : 02,15
02 se trouve dans CH$ en position  40

Entrer la chaine à rechercher et la position de début : 1112131,1
1112131 se trouve dans CH$ en position  21

Entrer la chaine à rechercher et la position de début : WXC,1
WXC ne se trouve pas dans CH$

Entrer la chaine à rechercher et la position de début : 0,1
0 se trouve dans CH$ en position  1

Entrer la chaine à rechercher et la position de début :
```

4.2 EXEMPLES D'APPLICATIONS

POS est l'une des fonctions les plus utiles du BASIC. La recherche de mots, de caractères séparateurs, etc... est en effet la fonction la plus

fréquemment utilisée pour le traitement de texte, pour la recherche documentaire, pour la gestion des fichiers et bien d'autres applications similaires. Indépendamment de ces applications typiquement « Texte », la fonction POS permet de simplifier considérablement la programmation des applications « classiques » de gestion. Elle permet notamment de réaliser la « consultation de tables », qui est l'une des opérations « de base » en gestion, en une seule instruction alors que normalement on emploie une boucle FOR ... NEXT. Voici quelques exemples d'utilisation de POS en programmation de gestion :

1. Contrôle de codification

Il s'agit simplement de contrôler l'existence d'un code par rapport à une table de référence. Pour cet exemple, le code à contrôler est entré au clavier.

```
 00010 ! E402 : Controle de codification
 00020 !
 00030   DIM TABREF$*64                  ! définition de la table de référence
 00040   TABREF$="WX3,CV4,XAB,X45,TTY"   ! et chargement
 00050 !
 00060   INPUT "Entrer le code à controler :": CODE$
 00070                                                    !---------------!
 00080   IF POS(TABREF$,CODE$,1)=0 THEN INCONNU ELSE TROUVE !  CODE$ est-il !
 00090                                                    !dans TABREF$ ? !
 00100                                                    ! NON -> INCONNU!
 00110                                                    ! OUI -> TROUVE !
 00120 !                                                  !---------------!
 00130   INCONNU: PRINT "Le code ";CODE$;" n'existe pas dans la table"
 00140            GOTO 60
 00150   TROUVE: PRINT "Le code ";CODE$;" est valide"
 00160           GOTO 60
 RUN
Entrer le code à controler : X45
Le code X45 est valide
Entrer le code à controler : TTY
Le code TTY est valide
Entrer le code à controler : GHJ
Le code GHJ n'existe pas dans la table
Entrer le code à controler :
```

Il s'agit de l'application la plus courante de POS : on explore une chaîne pour y rechercher un groupe de caractères et si on ne le trouve pas, on sort un message d'erreur. On notera dans cet exemple que les codes valides ont été *séparés* entre eux par un séparateur (une virgule). Si nous avions « collé » les codes les uns aux autres, le programme n'aurait pas donné des résultats corrects : les codes V4X ou 45T, par exemple, auraient été considérés à tort comme valides.

2. Consultation de table

Un tarif est entré en mémoire sous forme d'une chaîne de caractères. Cette chaîne est formée d'un certain nombre de postes (un poste par article), chaque poste étant séparé du suivant par un point-virgule. Chaque poste comprend un code article (1 à 5 car.) suivi de son prix unitaire en centimes (1 à 9 chiffres). Ces deux zones sont séparées par une virgule. Exemple :

E45,12;WW450,14500;X1,124578;

On entre au terminal le code article et une quantité et on demande de déterminer et d'afficher le prix total et le prix unitaire.

```
00010 ! E403 : Consultation de table
00020 !
00030  DIM TARIF$*128                                    ! déf.table tarif
00040  TARIF$="E45,12;WW350,14500;X1,124578;WW3,1245;"   ! et chargement
00050  PRINT NEWPAGE                                     ! écran à blanc
00060 !
00070  INPUT "Entrer le code article et la quantité :": CODAR$,QTE
00080 !
00090  I=POS(TARIF$,CODAR$&",",1)     ! calcul position de CODAR$ dans TARIF$
00100  IF I=0 THEN ERREUR             ! ---) ERREUR si CODAR$ non trouvé
00110  K=POS(TARIF$,",",I)            ! recherche du "," avant prix unitaire
00120  J=POS(TARIF$,";",I)            ! recherche du ";" aprés prix unitaire
00130  PU=VAL(TARIF$(K+1:J-1))        ! extraction et conversion numérique
00140  TOTAL=PU*QTE                   ! calcul du prix total
00150                                 ! affichage
00160  PRINT "Prix unitaire : ";PU;"   Prix total : ";TOTAL
00170  PRINT " "
00180  GOTO 70
00190                                 ! affichage message d'erreur
00200  ERREUR: PRINT CODAR$;" : code article inconnu "
00210  GOTO 170

       Entrer le code article et la quantité : E45,10
       Prix unitaire :  12    Prix total :  120

       Entrer le code article et la quantité : WW3,5
       Prix unitaire :  1245    Prix total :  6225

       Entrer le code article et la quantité : WW350,1
       Prix unitaire :  14500    Prix total :  14500

       Entrer le code article et la quantité : WWW4,56
       WWW4 : code article inconnu

       Entrer le code article et la quantité :
```

On notera dans cet exemple :

— l'instruction ligne 90 qui permet de rechercher le code article dans la table. Nous avons pris soin de concaténer la "," avec le code

article *ce qui évite de trouver à tort* dans la table un code ayant la même racine (WW 35, par exemple).

— les lignes 110, 120, 130 qui isolent, extraient et convertissent en numérique le prix unitaire. Ces trois instructions auraient pu être condensées en une seule expression « caractère ».

3. Débranchement en fonction d'un code enregistrement

Il s'agit, là encore, d'un cas fréquemment rencontré en informatique de gestion. On lit un enregistrement sur un fichier séquentiel. Chaque enregistrement est identifié par un code enregistrement (une lettre suivie de deux chiffres) et on demande de se débrancher à un sous-programme de traitement différent en fonction du code enregistrement. Certains codes ne donnent lieu à aucun traitement.

```
00010 ! E404 : débranchement en fonction d'un code enregistrement
00020 !
00030   DIM TABCE$*48                              ! définition et chargement
00040   TABCE$="E43,E44,E45,E67,F44,G45,H66,W00" ! de la table des codes
00050                                              ! enregistrement à traiter
00060   PRINT NEWPAGE
00070   PRINT ""
00080   INPUT "Code enregistrement ? ": CE$
00090 !
00100   IGOSUB=(POS(TABCE$,CE$,1)+3)/4             ! calcul de l'indice du GOSUB
00110                                              ! en fonc.pos de CE$ ds TABCE$
00120   IF IGOSUB<1 THEN GOTO 70                   ! pas de trait.pour ce code
00130   ON IGOSUB GOSUB 160,170,180,190,200,210,220,230
00140   GOTO 70
00150 !
00160   PRINT "Traitement des enreg.E43": RETURN ! traitement codes E43
00170   PRINT "Traitement des enreg.E44": RETURN ! traitement codes E44
00180   PRINT "Traitement des enreg.E45": RETURN ! traitement codes E45
00190   PRINT "Traitement des enreg.E67": RETURN ! traitement codes E67
00200   PRINT "Traitement des enreg.F44": RETURN ! traitement codes F44
00210   PRINT "Traitement des enreg.G45": RETURN ! traitement codes G45
00220   PRINT "Traitement des enreg.H66": RETURN ! traitement codes H66
00230   PRINT "Traitement des enreg.W00": RETURN ! traitement codes W00
```

```
Code enregistrement ?  E45
Traitement des enreg.E45

Code enregistrement ?  E44
Traitement des enreg.E44

Code enregistrement ?  B56

Code enregistrement ?  H66
Traitement des enreg.H66

Code enregistrement ?
```

Pour traiter cet exemple, nous avons constitué une table des codes enregistrements à traiter (TABCE$, lignes 30-40). Après chaque lecture d'un enregistrement, le programme explore par POS la table des codes et calcule l'indice du GOSUB de la ligne 130 : à la position 1 dans TABCE$, correspondra un indice égal à 1, à la position 4, un indice égal à 2 etc.. (ce calcul est réalisé par l'instruction ligne 100). Un tel type de programmation n'emploie pas de IF ... THEN ... ELSE et offre l'avantage d'aboutir à des programmes « structurés » (un module « directeur » appelant des sous-programmes). De tels programmes sont un peu plus longs à écrire que des programmes « classiques » mais bien plus faciles à tester et à modifier.

4. Transcodification d'un fichier

Il est parfois nécessaire, dans certaines applications, d'avoir à convertir un fichier créé avec un certain jeu de caractères (l'ASCII par exemple) en utilisant un jeu de caractères différent (l'EBCDIC par exemple).

Dans le programme ci-dessous, on entre une chaîne de caractères au terminal et on demande de convertir les minuscules de cette chaîne en majuscules et vice-versa.

```
 00010 ! E405 : Transcodification d'une chaine
 00020 !
 00030   DIM TA$*64,TB$*64,CH$*32
 00040   TA$="azertyuiopqsdfghjklwxcvbnmAZERTYUIOPQSDFGHJKLWXCVBNM"
 00050   TB$="AZERTYUIOPQSDFGHJKLWXCVBNMazertyuiopqsdfghjklwxcvbnm"
 00060 !
 00070   LINPUT "Entrer la chaine à transcodifier :": CH$
 00080 !
 00090   FOR I=1 TO LEN(CH$)              ! début de boucle de transcodification
 00100       P=POS(TA$,CH$(I:I),1)        ! calcul de la position dans TA$
 00110       IF P=0 THEN 130              ! le caractere n'est pas à remplacer
 00120       CH$(I:I)=TB$(P:P)            ! remplacement du caractere
 00130   NEXT I                           ! fin de boucle
 00140 !
 00150   PRINT "la chaine transcodifiée contient : ";CH$
 00160   PRINT "": GOTO 70
 RUN
Entrer la chaine à transcodifier : LE BASIC EST UN LANGAGE
la chaine transcodifiée contient : le basic est un langage

Entrer la chaine à transcodifier : NE AUX USA ( VERS 1965 )
la chaine transcodifiée contient : ne aux usa ( vers 1965 )

Entrer la chaine à transcodifier :
```

Ce programme utilise deux chaînes. La première -TA$- contient le jeu de caractères d'origine, la seconde -TB$- le jeu à utiliser pour la transcodification. On trouvera dans une position donnée de TB$ la traduction du caractère ayant la même position dans TA$ (dans cet exemple, les majuscules de TA$ sont « en face » des minuscules de TB$ et vice-versa).

Le fonctionnement du programme est très simple :

— il extrait, un par un, tous les caractères de la chaîne à transcoder (FOR ... NEXT lignes 90-130).

— il explore TA$ (POS ligne 100) avec le caractère extrait (POS ligne 100) et, si ce dernier figure dans TA$, il le remplace par le caractère de TB$ de même position (ligne 120).

4.3 FONCTION SREP$ DE RECHERCHE ET DE REMPLACEMENT

La fonction SREP$ (de l'anglais String REPlace) se présente sous la forme suivante :

SREP$(A$,*d*,X$,Y$)

où A$, X$, Y$ sont des chaînes de caractères et *d* une expression numérique entière.

Le traitement effectué par cette fonction est le suivant :

1. Le système explore la chaîne A$ à partir de la position *d* de cette chaîne.

2. à chaque fois qu'il trouve une sous-chaîne égale à X$, il la remplace par la chaîne Y$.

3. L'exploration une fois terminée, le résultat est placé dans la chaîne dont le nom figure à gauche du signe égal.

La chaîne A$ n'est pas modifiée sauf, bien sûr, si son nom figure à gauche du signe égal.

Les chaînes X$ et Y$ n'ont pas *obligatoirement la même longueur :* à chaque remplacement, la longueur de la chaîne d'origine est « *mise à*

jour », c'est-à-dire que le système lui additionne en valeur algébrique la différence entre la longueur de Y$ et celle de X$.

Voici un programme illustrant l'utilisation de cette fonction :

```
00010 ! E406 : fonction SREP$ de recherche et de remplacement
00020 !
00030   DIM CH$*128
00040 !
00050   PRINT NEWPAGE
00060   PRINT ""
00070   LINPUT "Entrer la chaine initiale :": CH$
00080 !
00090   INPUT "Entrer la chaine à remplacer et celle de remplacement :": A$,B
00100 !
00110   CH$=SREP$(CH$,1,A$,B$)  ! remplacement et stockage du résultat dans C
00120 !
00130   PRINT "CH$ contient maintenant :";CH$
00140   PRINT "": GOTO 90


        Entrer la chaine initiale : 123456789

        Entrer la chaine à remplacer et celle de remplacement : 5,*** NOM ***
        CH$ contient maintenant :1234*** NOM ***6789

        Entrer la chaine à remplacer et celle de remplacement : NOM,!
        CH$ contient maintenant :1234*** ! ***6789

        Entrer la chaine à remplacer et celle de remplacement : 7,***/***
        CH$ contient maintenant :1234*** ! ***6***/***89

        Entrer la chaine à remplacer et celle de remplacement : ***,""
        CH$ contient maintenant :1234 ! 6/89

        Entrer la chaine à remplacer et celle de remplacement : 34,22
        CH$ contient maintenant :1222 ! 6/89

        Entrer la chaine à remplacer et celle de remplacement : 2,--------
        CH$ contient maintenant :1---------------------- ! 6/89

        Entrer la chaine à remplacer et celle de remplacement :
```

Le lecteur imaginera facilement les nombreuses applications de cette fonction SREP$. Citons par exemple :

— remplacement d'abréviations par le mot entier et vice-versa

— remplacement de mots par leur synonyme

— suppression de mots non-significatifs dans un texte (dans ce cas, la chaîne de remplacement Y$ sera une chaîne de longueur nulle)

— transcodification de fichiers

— insertion de variables dans un texte fixe (la place de la variable est indiquée par un caractère spécial).

4.4 FONCTION SRCH DE RECHERCHE DANS UN TABLEAU

Cette fonction joue de même pour les tableaux de chaînes que POS vis-à-vis des chaînes. SRCH (de l'anglais SeaRCH) a la forme suivante :

SRCH(A$,B$,*i*)

où A$ est un tableau de caractères à une dimension (donc défini par une instruction DIM), B$ une chaîne et *i* une expression algébrique positive ou nulle et entière.

La fonction SRCH :

1. compare chaque élément du tableau A$ à B$ en partant de l'élément n° *i*.

2. quand elle trouve l'égalité, SRCH prend comme valeur le N° de l'élément sur lequel l'égalité a été trouvée et l'exploration s'arrête.

3. si aucune égalité n'a été trouvée, la fonction prend la *valeur - 1*.

Le programme ci-dessous illustre le fonctionnement de SRCH.

```
00010 ! E407 : Fonction SRCH de recherche dans un tableau
00020 !
00030   OPTION BASE 1
00040   DIM TM$(12)*10
00050   DATA JANVIER,FEVRIER,MARS,AVRIL,MAI,JUIN,JUILLET,AOUT,SEPTEMBRE,OCTOBRE
00060   DATA NOVEMBRE,DECEMBRE
00070   READ MAT TM$                    ! chargement du tableau
00080   PRINT NEWPAGE
00090 !
00100   INPUT "Entrer un nom de mois :": NOM$
00110 !
00120   P=SRCH(TM$,NOM$,1)              ! recherche de NOM$ dans le tableau TM$
00130   IF P<0 THEN NONTROU             ! --> NONTROU si non trouvé
00140 !
00150   PRINT NOM$;" se trouve dans l'élément N° ";P: PRINT "": GOTO 100
00160   NONTROU: PRINT NOM$;" non trouvé": PRINT "": GOTO 100


Entrer un nom de mois : JUIN
JUIN se trouve dans l'élément N°  6

Entrer un nom de mois : DECEMBRE
DECEMBRE se trouve dans l'élément N°  12

Entrer un nom de mois : JJUILLET
JJUILLET non trouvé

Entrer un nom de mois : JUILLET
JUILLET se trouve dans l'élément N°  7

Entrer un nom de mois :
```

L'application principale de la fonction SRCH est bien évidemment la consultation de table ou de tableau. Il est à remarquer que cette fonction est également applicable aux tableaux de valeurs numériques. La recherche se fait alors par comparaison de valeurs numériques, mais le fonctionnement est identique.

EXERCICES

1. Détection de mots dans un texte

On considère un texte composé de lignes de 64 caractères. Ce texte comprend des mots et des caractères séparateurs de mots qui sont des signes de ponctuation et des blancs. Il est demandé de :

— remplacer tous les caractères séparateurs par un séparateur unique : la barre de fraction ''/''. Lorsque deux caractères séparateurs se suivent (une virgule suivie d'un blanc, par exemple), ils seront remplacés par un seul ''/''.

— supprimer du texte tous les mots non significatifs tels que les prépositions, les adverbes, les diverses formes des verbes « avoir » et « être » etc... Ces mots seront contenus dans un tableau de 200 chaînes initialisé par DATA.

Par exemple, la ligne suivante :

LE LANGAGE BASIC PEUT,bEVIDEMMENT,bbETRE MODIFIE.

donnera :

LANGAGE/BASIC/MODIFIE/

en supposant que les mots « LE », « PEUT », « EVIDEMMENT »: et « ETRE » aient été entrés dans la table des mots non significatifs.
Pour simplifier, on admettra que tout le texte est en majuscules.

```
00010 ! E410 : Détection de mots dans un texte
00020 !
00030   OPTION BASE 1
00040   DIM MNS$(200)*8,CH$*128
00050   DATA ET,OU,NI,MAIS,OR,CAR,DONC,QUI,QUE,QUOI,DONT,OU,QU,QUAND,QUELQUE
00060   DATA QUELQU,JE,TU,IL,ELLE,NOUS,VOUS,ILS,ELLES,UN,UNE,UNS,UNES,EUX,ON
00070   DATA DU,DES,D,LA,LE,LES,MA,TA,SA,MON,TON,SON,NOTRE,VOTRE,LEUR,NOS,VOS
00080   DATA LEURS,CELUI,CELLE,CELLES,CEUX,CE,CES,CETTE,SIENNE,TIENNE,MIENNE
00090   DATA SIENNES,TIENNES,AVOIR,AI,AURAI,AS,A,AVONS,AVEZ,ONT,AVAIS,AVAIT
00100   DATA AVAIENT,AVIEZ,AVIONS,ETRE,SUIS,ES,EST,SOMMES,ETES,SONT,ETAIS
00110   DATA ETAIT,ETIONS,ETIEZ,ETAIENT,SOIT,ETE,SERA,SERAS,SERAI,SERONS
00120   DATA SEREZ,SERONT,SOIT,SOIE,SOYONS,SOYIEZ,SOIENT,DEUX,TROIS,QUATRE
00130   DATA CINQ,SIX,SEPT,HUIT,NEUF,DIX,DE,COMME,COMMENT,AVEC,SANS,PAR,AU
00135   DATA PARCE,DANS,SUR,SOUS,L
00140 !
00150   READ.MAT MNS$ EOF 170
00160 !
00170   TCS$=" ,;.:?'&!""()"
00180 !
00190   PRINT NEWPAGE
00200   LINPUT "Entrer la chaine à modifier :": CH$
00210   IF CH$="" THEN STOP
00220   CH$="."&CH$&"."
00230   FOR I=1 TO LEN(TCS$)
00240       CH$=SREP$(CH$,1,TCS$(I:I),"/")
00250   NEXT I
00260   IF POS(CH$,"//",1)=0 THEN FINI
00270   CH$=SREP$(CH$,1,"//","/")
00280   GOTO 260
00290   FINI: FOR I=1 TO 200
00300             IF MNS$(I)="" THEN SUIT2
00310             CH$=SREP$(CH$,1,"/"&MNS$(I)&"/","/")
00320         NEXT I
00330   SUIT2: PRINT "La chaine modifiée contient : ";CH$
00340          PRINT "": GOTO 200
```

```
Entrer la chaine à modifier : L OBJET DE NOD DESIRS

La chaine modifiée contient : /OBJET/NOD/DESIRS/

Entrer la chaine à modifier : L OBJET DE NOS DESIRS

La chaine modifiée contient : /OBJET/DESIRS/

Entrer la chaine à modifier : DEUX SOUS DE BON SENS

La chaine modifiée contient : /BON/SENS/

Entrer la chaine à modifier : LE BASIC ( NE AUX U.S.A,VERS 1965)

La chaine modifiée contient : /BASIC/NE/AUX/U/S/VERS/1965/

Entrer la chaine à modifier :
```

La structure de ce programme est très semblable à celle de l'exercice N° 4 du chapitre précédent mais le lecteur peut constater du premier coup d'œil que les fonctions POS et SREP$ nous ont permis d'économiqer pas mal d'instructions.

Ce programme comprend trois parties :

a) La boucle FOR ... NEXT lignes 230-250 qui prend les séparateurs un par un dans TCS et les remplacent par des ''/'' dans CH$(SREP$ ligne 240).

b) La boucle lignes 260-280 qui remplace les groupes ''//'' par des ''/'' (SREP$ ligne 270) jusqu'à ce qu'il n'y en ait plus dans CH$ (test par POS ligne 260).

c) La boucle FOR ... NEXT (lignes 290-320) qui prend chaque mot à supprimer dans MNS$ et le remplace (encadré par des ''/'' - pour éviter de détecter des parties de mots) par un seul ''/''.

Pour cette dernière partie, on aurait pu procéder différemment en isolant chaque mots de CH$ et en allant voir si ce mot figure dans MNS$ par SRCH. Cette variante comporte plus d'instructions mais est sans doute plus performante.

2. Cryptage du contenu d'un fichier

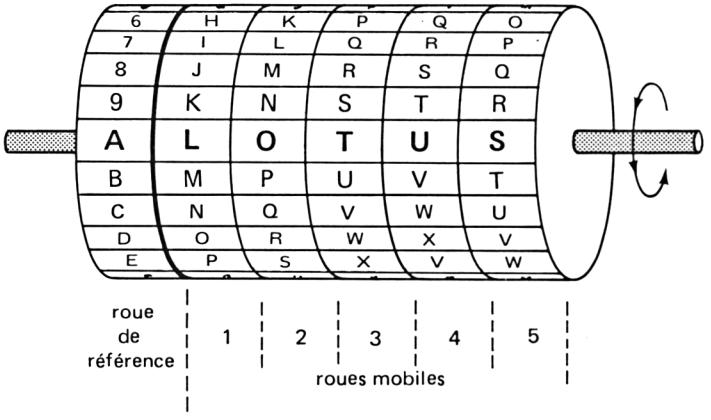
Pour des raisons de confidentialité, il est parfois nécessaire d'interdire à toute personne non autorisée la lecture du contenu d'un fichier informatique. La méthode la plus sûre (et également la plus ancienne) consiste à « crypter » ou à « coder » ce contenu, c'est-à-dire à remplacer les caractères qui le composent par d'autres caractères suivant une certaine formule.

Ce cryptage peut se faire soit par des unités spécialisées soit par programme. L'exercice qui va suivre illustre une méthode de cryptage simple à programmer à l'aide des fonctions BASIC de traitement de textes.

La méthode de codage proposée est utilisée depuis longtemps, en particulier dans les procédures de transmission militaires. Elle est basée sur l'utilisation d'un dispositif composé de six roues coaxiales. Chacune de ces roues porte sur sa tranche le jeu de caractères utilisé qui sera, pour cet exercice :

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9

La première de ces roues, qui sert de référence, est fixe par rapport à l'axe ; les cinq autres, numérotées de 1 à 5, sont mobiles.



Le principe de cryptage est le suivant :

a) on choisit une « clé » de cryptage qui est un mot de 5 caractères (faisant partie du jeu ci-dessus), par exemple le mot « LOTUS ».

b) on prend la première lettre de la clé — « L » et par rotation, on amène le « L » de la roue mobile n° 1 en face du « A » de la roue de référence. On prend ensuite la seconde lettre de la clé — « O » et on positionne le « O » de la roue mobile n° 2 en face du « A » de la roue fixe. On opère de la même façon avec les lettres « T », « U » et « S » de la clé et les roues mobiles n° 3, 4 et 5. Après cette opération, le mot-clé « LOTUS » se trouve en face du « A » de la roue de référence, comme représenté « à plat » ci-dessous :

référence	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9
roue N° 1	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K
roue N° 2	O	P	Q	R	S	T	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N
roue N° 3	T	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
roue N° 4	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
roue N° 5	S	T	U	V	W	X	Y	Z	b	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

← Sens de la rotation ←

c) pour crypter un enregistrement donné, on commence par le découper en groupes de cinq caractères. On prend ensuite le premier de ces groupes, le mot « CHOUX », par exemple. La première lettre de ce mot est « C » : en regard du « C » de la roue de référence, nous trouvons le « N » sur la roue mobile n° 1. Ce caractère sera le premier du mot crypté. De même, pour coder la deuxième lettre — « H », nous irons chercher en regard du « H » de la roue de référence, le « V » sur la roue mobile n° 2.

On procède de la même manière avec les « O », « U » et « X », remplacés par les «6 », « D » et « E » des roues mobiles n° 3, 4 et 5.

référence	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	ƀ	Ø	1	2	3	4	5	6	7	8	9
roue N° 1	L	M	N	O	P	Q	R		T	U	V	W	X	Y		ƀ	Ø	1	2	3		5	6		8	9	A	B	C	D	E	F	G	H	I	J	K
roue N° 2	O	P	Q	R	S	T	U	V	W	X	Y	Z	ƀ	Ø		2	3	4	5	6		8	9		B	C	D	E	F	G	H	I	J	K	L	M	N
roue N° 3	T	U	V	W	X	Y	Z	ƀ	Ø	1	2	3	4	5	6	7	8	9	A	B		D	E		G	H	I	J	K	L	M	N	O	P	Q	R	S
roue N° 4	U	V	W	X	Y	Z	ƀ	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		H	I	J	K	L	M	N	O	P	Q	R	S	T
roue N° 5	S	T	U	V	W	X	Y	Z	ƀ	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

Les autres groupes de cinq caractères seront traités comme le premier.

On demande donc de programmer le cryptage d'un fichier suivant cette méthode. Le fichier à crypter, en entrée, et le fichier crypté, en sortie, sont séquentiels et ont des enregistrements de 60 caractères.

Fig. 4.1 : Organigramme du programme «CRYPTAGE»

```
00010 ! E411  : Cryptage du contenu d'un fichier
00020 !
00030    OPTION BASE 1
00040                                                  ! ouverture des fichiers
00050    OPEN £1: "NAME=ACRYPT",INPUT                  ! 1.Fichier à crypter
00060    OPEN £2: "NAME=CRYPTE",OUTPUT                 ! 2.Fichier crypté
00070                                                  ! DEFINITION DES CHAINES
00080    DIM REF$*37                                   ! - roue de reférence
00090    DIM RM$(5)*37                                 ! - 5 roues mobiles
00100    DIM EAC$(12)*5                                ! - enreg.à crypter(5x12)
00110    DIM ECR$(12)*5                                ! - enreg.crypté   (5x12)
00120    DIM CC$*5                                     ! - clé de cryptage
00130                                                  !
00140 !-------------------------------------------------!
00150 ! saisie clé cryptage & positiont roues mobiles !
00160 !-------------------------------------------------!
00170    REF$="ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789" ! Initi.roue de reference
00180    INPUT "Entrer la clé de cryptage :": CC$      ! saisie clé de cryptage
00190                                                  ! BOUCLE DE POSITIONNT
00200    FOR I=1 TO 5                                  !
00210        N=POS(REF$,CC$(I:I),1)                    ! recherche position du
00220                                                  ! caract."I" ds reference
00230        IF N=0 THEN 180                           ! car.invalide -> saisie
00240                                                  !
00250        RM$(I)=REF$(N:37)&REF$(1:N-1)             !ROTATION ROUE MOBILE N°I
00260    NEXT I                                        !
00270 !-------------------------------------------------!
00280 ! lect.enr.à crypter,cryptage et écr.enr.crypté !
00290 !-------------------------------------------------!
00300    MAT ECR$=("")                                 !chaine nulle->enr.crypté
00310    READ £1,USING 320: MAT EAC$ EOF 480           ! lecture enreg.à crypter
00320    FORM 12*C 5
00330                                                  ! boucle 12 gpes de 5 car
00340    FOR J=1 TO 12                                 !
00350        FOR I=1 TO 5                              ! boucle 5 car du gpe J
00360                                                  !
00370            N=POS(REF$,EAC$(J)(I:I),1)            ! recherche position car.
00380                                                  ! I gpe J dans reference
00390            IF N=0 THEN 470                       ! enr.invalide -> ARRET
00400                                                  !
00410            ECR$(J)=ECR$(J)&RM$(I)(N:N)           ! mvt sur gpe J du car.
00420                                                  ! N de la roue mobile N°I
00430        NEXT I                                    ! car.suivant du gpe J
00440    NEXT J                                        ! gpe suivant de l'enreg.
00450    WRITE £2,USING 320: MAT ECR$                  ! écriture enreg.crypté
00460    GOTO 300                                      ! -> trait.enreg suivant
00470    STOP 1
00480    END                                           ! ** FIN DU PROGRAMME **
```

Dans ce programme, chaque roue est représentée par une chaîne (REF$ pour la roue de référence et le tableau RM$ pour les roues mobiles). Le traitement comprend deux parties principales :

a) La rotation des roues mobiles (lignes 200-260). Cette rotation se fait essentiellement par l'instruction 250 qui consiste à couper la chaîne REF$ en deux et à « recoller les morceaux » en les inversant.

b) Le cryptage de l'enregistrement qui se fait suivant un processus exposé au § 4.3 de ce chapitre (transcodification d'un fichier).

3. Traduction d'une table de décision

De nombreux analystes ont pris l'habitude de présenter les résultats de leurs analyses sour forme de tables de décision. Rappelons au lecteur qu'une « table de décision » comprend deux parties :

— une partie « Décisions ». Chaque ligne représente une condition à tester et chaque colonne un cas possible pour la condition.

— une partie « Actions » ou « Traitements ». Chaque ligne représente une action à effectuer. La présence d'une croix à l'intersection d'une ligne et d'une colonne indique que le traitement correspondant à la ligne est à faire dans le cas correspondant à la colonne.

La figure 4.2 représente une table de décision. On y voit, par exemple, que dans le cas N°8, on doit effectuer les calculs N°1, 3 et 5.

ANCIENNETÉ	< à 5 ans catégorie A				de 5 à 9 ans catégorie B				de 10 à 25 ans catégorie C				> à 25 ans catégorie D			
SEXE 1 = homme 2 = femme	Homme		Femme		Homme		Femme		Homme		Femme		Homme		Femme	
Statut Cadre Oui = codes 1,3,9	non	oui	non	oui	non	oui	non	oui	non	oui	non	oui	non	oui	non	oui
Cas n°	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Traitement n°	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Calcul Prime 1	X		X					X		X			X		X	X
Calcul Prime 2	X		X		X				X			X				
Calcul Prime 3		X		X		X	X	X			X		X	X		
Calcul Prime 4					X					X	X			X		
Calcul Prime 5			X	X			X	X				X			X	
Calcul Prime 6						X			X				X			X

Fig. 4.2 — Table de décision pour l'exercice n° 3

Un tel support d'analyse est un bon moyen de communication entre les analystes (ou les utilisateurs) et les programmeurs. Il permet aux premiers de s'assurer qu'aucun cas n'a été oublié et il évite aux seconds le dessin d'organigrammes complexes et l'emploi de IF ... THEN ... ELSE longs à écrire et à tester. Encore faut-il trouver une méthode pour passer simplement de la table de décision aux instructions du programme. Nous allons voir dans le présent exercice comment « traduire » une table de déci-

sion avec le minimum d'efforts et le maximum de fiabilité et ceci à l'aide des fonctions BASIC de traitement de textes.

Cet exercice porte sur un calcul de primes dans une paie : le programme doit lire le fichier du personnel, tester certains codes et calculer certaines primes. Les tests à effectuer et les calculs à faire sont spécifiés par la table de décision de la figure 4.1

Le code donnant le statut (cadre ou non-cadre) se trouve en position 5 du fichier : les statuts 1, 3 et 9 désignent des cadres ; les autres valeurs désignent des non-cadres.

L'ancienneté (0 à 40) se trouve en position 8 et 9 de l'enregistrement. Le code sexe (1 = Homme, 2 = femme) est en pos. 15.

Pour éviter d'alourdir le programme, on admet que les calculs de prime 1, 2, etc... sont des sous-programmes BASIC dont l'exécution est simulée par des messages à l'écran dans le programme ci-après. Dans le même esprit, nous avons remplacé la lecture du fichier par une entrée des données au clavier de façon à visualiser les entrées et les sorties.

```
00010 ! E412 : Traduction d'une table de décision
00020 !
00030   OPTION BASE 1
00040                                  !-----------------------------------!
00050   DIM TABDEC$(32)*6              ! définition et initialisation de la !
00060                                  ! table de décision :               !
00070   PRINT NEWPAGE                  ! une ligne DATA représente une     !
00080                                  ! colonne de la table de décision   !
00090                                  !-----------------------------------!
00100   DATA A1N,XX....
00110   DATA A1O,..X...
00120   DATA A2N,XX..X.
00130   DATA A2O,..X.X.
00140   DATA B1N,.X.X..
00150   DATA B1O,..X..X
00160   DATA B2N,..X.X.
00170   DATA B2O,X.X.X.
00180   DATA C1N,.X...X
00190   DATA C1O,X..X..
00200   DATA C2N,..XX..
00210   DATA C2O,.X..X.
00220   DATA D1N,X.X..X
00230   DATA D1O,..XX..
00240   DATA D2N,X...X.
00250   DATA D2O,X....X
00260                                  !-----------------------------------!
00270   READ MAT TABDEC$               ! chargement de la table de décision !
00280                                  ! - décisions --> N° élément impair  !
00290                                  ! - actions --> N° élément pairs     !
00300                                  !-----------------------------------!
00310   ST$="ONONNNNNO"                ! Chargement table cadre/non cadre  !
00320                                  ! en fonction du statut             !
00330                                  !-----------------------------------!
00340 !
00350   INPUT "Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9):": ANC,SEX,STA
00360                                  !-----------------------------------!
00370                                  ! partie DECISION de la table : une !
00380                                  ! colonne décision est mise dans R$ !
00390                                  !-----------------------------------!
00400   IF ANC<5 THEN D$="A"                 ! catégorie ancienneté A
00410   IF ANC>=5 AND ANC<=9 THEN D$="B"     ! ---------------------- B
00420   IF ANC>=10 AND ANC<=25 THEN D$="C"   ! ---------------------- C
00430   IF ANC>25 THEN D$="D"                ! ---------------------- D
00440   D$=D$&STR$(SEX)                      ! sexe --> D$
00450   D$=D$&ST$(STA:STA)                   ! cadre/non cadre -> D$
00460                                  !-----------------------------------!
00470                                  ! partie ACTION de la table : la    !
00480                                  ! colonne corersp. à D$ est mise    !
00490                                  ! dans A$ à partir de TABDEC$       !
00500                                  !-----------------------------------!
00510   N=SRCH(TABDEC$,D$,1)                 ! recherche D$ dans TABDEC$
00520   A$=TABDEC$(N+1)                      ! l'Action est ds l'élément + 1
00530   IF A$(1:1)="X" THEN GOSUB 600  !-----------------------------------!
00540   IF A$(2:2)="X" THEN GOSUB 610  ! exécution des sous-programmes en  !
00550   IF A$(3:3)="X" THEN GOSUB 620  ! fonction des "X" de la col.ACTION !
00560   IF A$(4:4)="X" THEN GOSUB 630  !                                   !
00570   IF A$(5:5)="X" THEN GOSUB 640  !                                   !
00580   IF A$(6:6)="X" THEN GOSUB 650  !-----------------------------------!
00590   GOTO 350
00600   PRINT "Calcul Prime 1": RETURN !-----------------------------------!
00610   PRINT "Calcul Prime 2": RETURN !                                   !
00620   PRINT "Calcul Prime 3": RETURN ! *** SOUS-PROGRAMMES DE CALCUL *** !
00630   PRINT "Calcul Prime 4": RETURN !                                   !
00640   PRINT "Calcul Prime 5": RETURN !                                   !
00650   PRINT "Calcul Prime 6": RETURN !-----------------------------------!
```

```
Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9): 31,2,9
Calcul Prime 1
Calcul Prime 6
Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9): 0,1,2
Calcul Prime 1
Calcul Prime 2
Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9): 12,2,3
Calcul Prime 2
Calcul Prime 5
Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9): 26,1,5
Calcul Prime 1
Calcul Prime 3
Calcul Prime 6
Entrer Ancienneté,sexe(1 ou 2),statut(1 à 9):
```

Le principe du programme est très simple :
chaque colonne de la table de décision est traduit par deux éléments adjacents d'un tableau de chaînes. Le premier (impair) correspond à la partie « Décision » et contient les valeurs des codes à tester mises bout-à-bout. Par exemple, l'élément N°3 du tableau qui correspond à la colonne 2 de la table de Décision contiendra ''A2N'' (ligne 120 du programme) ce qui veut dire :

ancienneté $<$ à 5 ans, femme, non-cadre

De même, l'élément suivant (pair) contiendra une suite de ''X'' et de ''.'', le ''X'' signifiant que le calcul est à effectuer.

Compte-tenu de ce principe, la structure du programme est facile à comprendre. Elle comprend trois parties :

a) la constitution et le chargement du tableau qui est fait par DATA (lignes 100-250 - une colonne de la table de décision est traduite par une ligne DATA).

b) la constitution de la chaîne D$ qui contient les valeurs déduites des codes entrés par INPUT (lignes 380-430). Ces valeurs sont obtenues soit par des IF (lignes 380-410) pour la catégorie d'ancienneté, soit par simple conversion en caractère pour le code « sexe », soit enfin par consultation de la table ST$ qui donne 0 ou N en fonction du statut.

c) la consultation du tableau avec D$ (par SRCH ligne 490) et l'appel des sous-programmes de calcul en fonction de chaque position de A$ (lignes 510-560).

Nous pensons que le lecteur sera frappé par la ressemblance entre la table de décision et le programme (en raffinant un peu, on aurait pu imprimer cette table à partir de TABDEC$). Soulignons que le programme ainsi obtenu est *facile à tester* (dans chaque cas, on visualise bien les calculs effectués) et *facile à modifier* (il suffira, dans beaucoup de cas, d'ajouter ou de retirer un ''X'' sans avoir à explorer une forêt de IF ... THEN).

5

FONCTIONS BASIC ANNEXES

Les fonctions BASIC étudiées dans ce chapitre sont moins importantes que la fonction POS, par exemple. Elles peuvent être facilement programmées à l'aide des opérateurs et des fonctions que nous avons jusqu'ici, mais leur emploi permet néanmoins de simplifier les programmes dans certains cas particuliers.

5.1 INSERTION ET SUPPRESSION DE BLANCS A GAUCHE ET A DROITE D'UNE CHAÎNE

Quatre fonctions LPAD$, RPAD$, LTRM$ et RTRM$ permettent d'insérer et de supprimer les caractères blancs à gauche et à droite d'une chaîne. Voici leurs caractéristiques :

a) Insertion de blancs à gauche d'une chaîne (LPAD$: Left PADding)

La fonction LPAD$(X$,*l*) ajoute des blancs à gauche de la chaîne X$ de façon à former une chaîne de longueur *l*, ce qui revient à dire qu'elle cadre X$ à droite d'une zone de longueur *l*. L'exemple ci-dessous illustre son utilisation :

```
)0010 ! E501 : insertion de blancs à gauche d'une chaine
)0020 !
)0030   DIM ZONE$*40                ! dimension de la zone de cadrage = 40 car.
)0040   INPUT "Entrer la chaine à cadrer à droite d'une zone de 40 car. ": CH$
)0050 !
)0060   ZONE$=LPAD$(CH$,40)         ! cadrage de CH$ à droite de ZONE$
)0070 !
)0080   PRINT ""
)0090   PRINT "ZONE$ contient : ";ZONE$;" longueur : ";LEN(ZONE$)
)0100   PRINT "": GOTO 40
RUN
ntrer la chaine à cadrer à droite d'une zone de 40 car.  *

)NE$ contient :                                        * longueur :  40

ntrer la chaine à cadrer à droite d'une zone de 40 car.  BERNARD

)NE$ contient :                                      BERNARD longueur :  40

ntrer la chaine à cadrer à droite d'une zone de 40 car.  "    BERTRAND    "

)NE$ contient :                                  BERTRAND     longueur :  40

ntrer la chaine à cadrer à droite d'une zone de 40 car.
```

b) Insertion de blancs à droite d'une chaîne (RPAD$: Right PADding).

La fonction RPAD$ est le pendant de LPAD$: elle permet de cadrer une chaîne X$ à droite d'une zone de longueur l. Voici un exemple d'utilisation :

```
00010 ! E502 : insertion de blancs à droite d'une chaine
00020 !
00030   DIM ZONE$*40                ! dimension de la zone de cadrage = 40 car.
00040   INPUT "Entrer la chaine à cadrer à gauche d'une zone de 40 car. ": CH$
00050 !
00060   ZONE$=RPAD$(CH$,40)         ! cadrage de CH$ à gauche de ZONE$
00070 !
00080   PRINT ""
00090   PRINT "ZONE$ contient : ";ZONE$;" longueur : ";LEN(ZONE$)
00100   PRINT "": GOTO 40
RUN
ntrer la chaine à cadrer à gauche d'une zone de 40 car.  *

ONE$ contient : *                                       longueur :  40

ntrer la chaine à cadrer à gauche d'une zone de 40 car.  BERNARD

ONE$ contient : BERNARD                                 longueur :  40

ntrer la chaine à cadrer à gauche d'une zone de 40 car.  "    BERNARD"

ONE$ contient :     BERNARD                             longueur :  40

ntrer la chaine à cadrer à gauche d'une zone de 40 car.
```

c) Suppression de blancs à gauche et à droite d'une chaîne.

Les fonctions LTRM$ (Left TRiM) et RTRM$ (Right TRiM) permettent de supprimer les blancs à gauche et à droite de la chaîne dont le nom figure comme argument. La longueur de la chaîne résultante est diminuée du nombre de blancs supprimés. L'exemple ci-dessous illustre le fonctionnement de ces deux fonctions :

```
 00010 ! E503 : suppression de blancs à gauche et à droite d'une zone
 00020 !
 00030   DIM Z$*64
 00040 !
 00050   INPUT "Entrer la chaine à modifier :": Z$
 00060   PRINT ""
 00070   PRINT "Z$ contient  ";Z$;" longueur ";LEN(Z$)
 00080   Z$=LTRM$(Z$)
 00090   PRINT ""
 00100   PRINT "après suppr.GAUCHE Z$ contient  ";Z$;" longueur ";LEN(Z$)
 REPLACE
 RUN
Entrer la chaine à modifier :"    LES FONCTIONS DU BASIC      "

Z$ contient      LES FONCTIONS DU BASIC       longueur  31

après suppr.GAUCHE Z$ contient  LES FONCTIONS DU BASIC       longueur  27

après suppr.DROITE Z$ contient  LES FONCTIONS DU BASIC longueur  22

 BAS-5055 le programme s'est terminé à la ligne 140
```

5.2 CONVERSION D'UNE CHAÎNE DE MAJUSCULES EN MINUSCULES ET VICE-VERSA

La fonction LWRC$ (en anglais : LoWeRCase) permet de convertir les majuscules existant dans une chaîne en minuscules.

Inversement, la fonction UPRC$ (UPpeRCase) convertit les minuscules rencontrées dans une chaîne en majuscules. Dans les deux cas, les caractères autres que les majuscules et les minuscules restent inchangés.

Nous avons illustré l'utilisation de ces deux fonctions par l'exemple suivant :

```
 00010 ! E505 : Conversion de majuscules en minuscules et vice-versa
 00020 !
 00030   DIM Z$*64
 00040 !
 00050   LINPUT "Entrer la chaine à convertir : ": Z$
 00055   PRINT ""
 00060   PRINT "avant conversion Z$ = ";Z$
 00070   Z$=LWRC$(Z$)
 00080   PRINT "après conversion en minuscules Z$ = ";Z$
 00090   Z$=UPRC$(Z$)
 00100   PRINT "après conversion en majuscules Z$ = ";Z$
 00110   STOP
 RUN
Entrer la chaine à convertir :  DEPUIS 1970, LE BASIC A EVOLUE !!!

avant conversion Z$ = DEPUIS 1970, LE BASIC A EVOLUE !!!
après conversion en minuscules Z$ = depuis 1970, le basic a evolue !!!
après conversion en majuscules Z$ = DEPUIS 1970, LE BASIC A EVOLUE !!!

 BAS-5055 le programme s'est terminé à la ligne 110
```

5.3 RÉPÉTITION D'UNE CHAÎNE DE CARACTÈRES

La fonction RPT$ (de l'anglais RePeTe) permet d'obtenir une chaîne composée d'une chaîne donnée répétée *n* fois. Cette fonction a la forme :

RPT$(A$,*n*)

Après exécution, la chaîne résultante contiendra la chaîne A$ concaténée *n* fois et sa longueur sera *n* fois celle de A$. Cette fonction est particulièrement utile lorsqu'il s'agit de constituer des chaînes de longueur proportionnelle à une valeur donnée : lignes horizontales ou verticales pour dessiner un cadre, lignes pour souligner un mot dans un texte, graphiques etc... Le programme ci-dessous illustre le fonctionnement de RPT$.

```
 00010 ! E506 : répétition d'une chaine de caractère
 00020 !
 00030   DIM CR$*128
 00040   INPUT "Entrer la chaine à répéter et le nb de fois : ": C$,N
 00050   CR$=RPT$(C$,N)
 00060   PRINT "résultat : ";CR$;" longueur : ";LEN(CR$)
 00070   PRINT "": GOTO 40
 RUN
Entrer la chaine à répéter et le nb de fois :  *,99
résultat :
**********************************************************************************
******************** longueur :  99

Entrer la chaine à répéter et le nb de fois :  + * +,7
résultat : + * ++ * ++ * ++ * ++ * ++ * ++ * + longueur :  35

Entrer la chaine à répéter et le nb de fois :  "  33  ",6
résultat :   33    33    33    33    33    33   longueur :  36

Entrer la chaine à répéter et le nb de fois :  "",999
résultat :  longueur :  0

Entrer la chaine à répéter et le nb de fois :
```

5.4 ÉDITION D'UNE ZONE NUMÉRIQUE

Dans la pratique, sur une facture, sur une lettre ou sur un tarif, par exemple, les zones numériques (montants, dates, etc...) sont éditées sous une certaine forme avec des signes + et —, des points ou des virgules, des symboles monétaires ($, F, £...) etc...

Cette forme d'édition à l'aide de « masques d'édition » est bien connue et est utilisée dans la plupart des langages dans les formats d'impression : le programmeur décrit l'image de la zone éditée par une spécification PIC (de l'anglais PICture).

La fonction BASIC CNVRT$ (de l'anglais CoNVeRT) utilise le même principe : elle permet de convertir une zone numérique en chaîne de caractère d'après la description figurant dans un masque d'édition.

Cette fonction a la forme suivante :

CNVRT$(M$,*z*)

ou MS décrit l'édition demandée et *z* est la zone numérique à éditer.

M$ est une chaîne de caractères qui peut avoir trois formes différentes :

A . M$ = ''N*x.y*''

La zone z est convertie en une chaîne de *x* caractères de long, avec un point décimal et *y* chiffres après ce point. La zone est cadrée à droite et les zéros non significatifs sont supprimés.

B . M$ = ''ZD *x.y*''

Le résultat est le même sauf que les zéros non significatifs ne sont pas supprimés.

C . M$ = ''PIC(masque)''

La zone est éditée suivant ce qui est spécifié dans le « masque ». Ce masque est une chaîne de caractères de contrôle qui ont chacun leur rôle ; ces caractères sont les suivants :

— **£ :** à l'édition, ce caractère est remplacé par un chiffre de la zone à éditer, même s'il s'agit d'un zéro non significatif.

— **Z :** à l'édition, ce caractère est remplacé par un chiffre de la zone à éditer ou par un blanc s'il s'agit d'un zéro non significatif.

— **$:** il s'agit du symbole « monétaire flottant ». les «$» sont remplacés par des chiffres de la zone à éditer ou des blancs. Seul le caractère précédant le premier chiffre significatif est remplacé par le symbole ''$''. Ce type d'édition est surtout usité aux U.S.A. où le signe ''$'' précède un montant, alors qu'en France le signe ''F'' le suit.

— ***** : les zéros non significatifs sont remplacés à l'édition par des astérisques. Cette possibilité est utilisée lorsqu'on veut « protéger » la zone précédant les chiffres significatifs d'un montant contre tout ajout « intempestif ».

— **+ ou —** : ils peuvent être utilisés de deux façons différentes :
placés à gauche du masque, ils se comportent comme des symboles flottants mais ne figurent dans la zone éditée que si celle-ci est ou positive ou négative.
placés à droite du masque, ils sont édités à l'endroit indiqué si la zone est positive (pour +) ou négative (pour —).

— **blanc ,/ ou , (virgule) :** ces caractères sont insérés dans la zone éditée si le caractère précédent n'est pas blanc.

— **. (point) :** le point est inséré entre le chiffre des unités et la première décimale de la zone à éditer.

— **CR, DB ou DR :** ces groupes de lettres sont placés à la fin du masque et n'apparaissent que si la zone à éditer est négative.

REMARQUES :

1. - Il est possible d'éditer une zone suivant les normes françaises (virgule décimale à la place du point décimal anglo-saxon) en utilisant l'OPTION INVP qui inverse le point et la virgule. Cette inversion se fait uniquement à l'édition et dans le masque les points et les virgules seront placés suivant les normes anglo-saxonnes.

2. - La fonction PIC$ permet de changer la valeur du symbole monétaire $. Il suffit d'écrire :

```
A$ = PIC$(''F'')
```

pour que le symbole monétaire devienne ''F'' (A$ est une zone quelconque qui ne joue ici aucun rôle). Néanmoins, le $ doit toujours figurer dans le masque et ne sera remplacé par le caractère modifié qu'à l'édition.

Le programme ci-dessous récapitule les principaux types de masques utilisés.

```
00010 ! E507 : exemples d'édition par CNVRT$
00020 !
00030   OPTION LPREC,INVP                          ! calcul en double précision
00040   A$=PIC$("F")                               ! symbole monétaire flottant = F
00050   PRINT £255: "ZONE NUMERIQUE","MASQUE","ZONE EDITEE"
00060   PRINT £255: ""
00070   INPUT "Entrer le masque et la zone à éditer : ": M$,Z
00080   ZE$=CNVRT$(M$,Z)
00090   PRINT £255: Z,M$,ZE$
00100   GOTO 70
```

ZONE NUMERIQUE	MASQUE	ZONE EDITEE
1456	PIC(£££ £££.££)	001 456,00
,14	PIC(£££ £££.££)	000 000,14
,09	PIC(ZZZ ZZZ.ZZ)	0,09
123,7	PIC(ZZZ ZZ£.££)	123,70
,08	PIC(ZZZ ZZ£.££)	0,08
1456	PIC(*** **£.££)	**1 456,00
1456	PIC($$$ $$£.££)	F1 456,00
-415,47	PIC(--- --£.££)	-415,47
-14147,1	PIC(ZZZ ZZ£.££ -)	14 147,10 -
12345,14	PIC(+++ ++£.££)	+12 345,14
120582	PIC(££/££/££)	12/05/82
1243,78	PIC(ZZZ ZZ£.££ CR)	1 243,78
-758,47	PIC(ZZZ ZZ£.££ CR)	758,47 CR

On remarquera, au vu des résultats obtenus, que, dans tous les cas, *la longueur de la zone éditée est égale à celle du masque.* Si, par exemple, on désire insérer dans un texte une zone éditée par masque, il faudra d'abord supprimer les blancs à gauche de cette zone (par LTRM$) de façon à éviter les blancs entre le mot précédent et la zone.

5.5 RANG D'UN CARACTÈRE DANS UN JEU DE CARACTÈRE

Nous avons défini dans le premier chapitre ce qu'était un jeu de caractère (voir page 2). Deux fonctions permettent en BASIC de se référer au jeu de caractère utilisé dans un programme : ce sont les fonctions ORD (de l'anglais ORDinal) et CHR$ (de l'anglais CHaRacter).

ORD(X$) donne le range (0 à 255) du caractère X$ dans le jeu de caractère utilisé. Voici un exemple :

```
 00010 ! E508 : fonction ORD
 00020 !
 00030   INPUT "Entrer un caractère :": CAR$
 00040   PRINT ""
 00050   PRINT "le caractère ";CAR$;" a le rang ";ORD(CAR$)
 00060   PRINT "": GOTO 30
 RUN
Entrer un caractère : A

le caractère A a le rang  193

Entrer un caractère : *

le caractère * a le rang  92

Entrer un caractère : %

le caractère % a le rang  108

Entrer un caractère : ì

le caractère ì a le rang  241

Entrer un caractère :
```

Inversement, la fonction CHR$(n) donne le caractère ayant le rang n dans le jeu de caractère comme le montre l'exemple ci-après.

Si le rang spécifié est supérieur à 255, la fonction prend comme rang le reste de la vision par 256 du chiffre indiqué.

```
 00010 ! E509 : fonction CHR$
 00020 !
 00030   INPUT "Entrer le rang :": RANG
 00040   PRINT "le caractère de rang ";RANG;" est ";CHR$(RANG)
 00050   PRINT "": GOTO 30
 RUN
Entrer le rang : 194
le caractère de rang  194  est B

Entrer le rang : 249
le caractère de rang  249  est 9

Entrer le rang : 456
le caractère de rang  456  est H

Entrer le rang : 2650
le caractère de rang  2650  est §

Entrer le rang : 79
le caractère de rang  79  est !

Entrer le rang :
```

5.6 TRI DES ÉLÉMENTS D'UN TABLEAU DE CHAÎNES

Il est souvent nécessaire de classer les données figurant dans un tableau de chaînes de caractères par ordre alphabétique. Les deux fonctions AIDX (tri d'un tableau en séquence croissante) et DIDX (tri d'un tableau en séquence décroissante) permettent de réaliser ces opérations de classement.

Les instructions de tri ont la forme :

MAT TT = AIDX (T$) *(l,c)*
ou DIDX

— T$ est le tableau de chaînes à trier qui doit être un vecteur, c'est-à-dire n'avoir qu'une seule dimension.

— TT est un tableau de zones numériques ayant le même nombre d'éléments que T$.

l et *c* sont des N° de lignes et de colonnes *facultatifs.*

Ces fonctions AIDX et DIDX effectuent les opérations suivantes :

a) Les éléments de T$ sont triés en séquence croissante (AIDX) ou décroissante (DIDX).

b) Les rangs des éléments de T$ sont placés ensuite dans TT dans l'ordre du tri. Le rang dans T$ de l'élément qui est le « plus bas » dans la séquence est mis dans le premier élément de TT, celui de l'élément de T$ qui lui est immédiatement supérieur dans le deuxième et ainsi de suite. *T$ n'est pas modifié par l'opération de tri.*

c) Si *l* et *c* ont été indiqués, le tableau TT est redimensionné suivant *l* et *c*. S'ils sont omis, TT prend les mêmes dimensions que T$.

Voici un exemple de programme utilisant le tri d'un tableau :

```
00010 ! E510 : tri d'un tableau
00020 !
00030   OPTION BASE 1
00040   DIM TM$(12)*10,TTM(12),TT$(12)
00050   DATA Janvier,Février,Mars,Avril,Mai,Juin,Juillet,Aout,Septembre
00060   DATA Octobre,Novembre,Décembre
00070   READ MAT TM$
00080 !
00090   PRINT NEWPAGE
00100   PRINT USING 110: MAT TM$
00110   FORM "TM$ contient : ",POS 20,6*C 10,POS 20,6*C 10
00120   PRINT "": PRINT "TRI DE TM$ EN ORDRE ALPHABETIQUE CROISSANT":PRINT ""
00130 !
00140   MAT TTM=AIDX(TM$)
00150   PRINT USING 160: MAT TTM
00160   FORM "TTM contient : ",12*N 3
00170 !
00180   FOR I=1 TO 12
00190       TT$(I)=TM$(TTM(I))
00200   NEXT I
00210   PRINT ""
00220   PRINT USING 230: MAT TT$
00230   FORM "TT$ contient : ",POS 20,6*C 10,POS 20,6*C 10
00240 !
00250   PRINT "": PRINT "TRI DE TM$ EN ORDRE ALPHABETIQUE DECROISSANT":PRINT ""
00260   MAT TTM=DIDX(TM$)
00270   PRINT USING 160: MAT TTM
00280   FORM "TTM contient : ",12*N 3
00290 !
00300   FOR I=1 TO 12
00310       TT$(I)=TM$(TTM(I))
00320   NEXT I
00330   PRINT ""
00340   PRINT USING 230: MAT TT$
```

```
TM$ contient :      Janvier   Février   Mars      Avril     Mai       Juin
                    Juillet   Aout      Septembre Octobre   Novembre  Décembre

TRI DE TM$ EN ORDRE ALPHABETIQUE CROISSANT

TTM contient :    8  4 12  2  1  7  6  5  3 11 10  9

TT$ contient :      Aout      Avril     Décembre  Février   Janvier   Juillet
                    Juin      Mai       Mars      Novembre  Octobre   Septembre

TRI DE TM$ EN ORDRE ALPHABETIQUE DECROISSANT

TTM contient :    9 10 11  3  5  6  7  1  2 12  4  8

TT$ contient :      Septembre Octobre   Novembre  Mars      Mai       Juin
                    Juillet   Janvier   Février   Décembre  Avril     Aout
```

Ce programme comprend cinq parties principales :

1. Le chargement du tableau des noms de mois TM$ (lignes 40 à 70).

2. Le tri croissant de TM$ (ligne 140) : les rangs des éléments de TM$ triés se trouvent dans TTM.

3. Le transfert de TM$ dans TS$ dans l'ordre indiqué par chaque poste de TTM (boucle lignes 180-200).

4. Le tri décroissant de TM$ (qui n'a pas changé) ligne 260.

5. Le transfert de TM$ dans TS$ (lignes 300-320).

EXERCICES :

1. Édition d'un histogramme

On entre au clavier 12 valeurs représentant les ventes d'un article pour les 12 mois de l'année et on demande d'afficher à l'écran un histogramme. Cet histogramme se compose de 12 lignes horizontales ; la longueur de chaque ligne est proportionnelle au pourcentage relatif de la vente considérée par rapport à la vente maximale de l'année et ceci à raison d'une position de l'écran pour 2 %. Par exemple, la vente maximale ayant été de 620, une vente de 255 sera représentée par une ligne de 21 ''*'' (255 x 100 / 620 x 2 = 21 arrondi à l'unité la plus proche). A la fin de chaque ligne, on reportera la valeur de la vente du mois considéré.
Voici le programme correspondant et un exemple d'histogramme.

```
00010 ! E511 : Edition d'un histogramme
00020 !
00030   OPTION BASE 1
00040   DIM L$*67,V(12),M$(12)*10
00050   DATA Janvier,Février,Mars,avril,Mai,Juin,Juillet,Aout,Septembre
00060   DATA Octobre,Novembre,Décembre
00070   READ MAT M$
00080 !
00090   PRINT NEWPAGE                          ! remise à blanc écran
00100 !
00110   INPUT "Ventes des 12 mois ? ": MAT V
00120 !
00130   M=0
00140   FOR I=1 TO 12                          ! boucle de détermination
00150       IF V(I)>M THEN M=V(I)              ! de la vente maximale
00160 !
00170   NEXT I
00180 !
00190   PRINT ""
00200                                          ! impression échelle horizontale
00210   L$="!"&RPT$(" ",49)&"1"
00220   PRINT USING 230: L$
00230   FORM POS 13,C 67
00240   L$="!    1    2    3    4    5    6    7    8    9    0 %"
00250   PRINT USING 230: L$
00260   L$="!"&RPT$("----0",10)
00270   PRINT USING 230: L$
00280                                          ! boucle d'impression
00290   FOR I=1 TO 12
00300       PC=ROUND(V(I)*50/M,0)              ! calcul des % divisés par 2
00310       L$="!"&RPT$("*",PC)&" "&STR$(V(I))! constitution ligne
00320       PRINT USING 330: M$(I),L$          ! impression ligne
00330       FORM C 12,C 67
00340   NEXT I
00350 !
00360   STOP
```

```
Ventes des 12 mois ?  456,511,658,789,910,1023,561,410,589,758,675,902

                !                                                    1
                !    1    2    3    4    5    6    7    8    9    0 %
                !----0----0----0----0----0----0----0----0----0----0
Janvier         !*********************-456
Février         !************************ 511
Mars            !*****************************-658
avril           !****************************************- 789
Mai             !*********************************************- 910
Juin            !**************************************************- 1023
Juillet         !***************************- 561
Aout            !********************- 410
Septembre       !******************************- 589
Octobre         !************************************- 758
Novembre        !*********************************- 675
Décembre        !*********************************************- 902
```

Ce programme utilise essentiellement RPT$ qui sert ici à générer une ligne de blancs (ligne 210), la ligne d'échelle (ligne 260) et une chaîne de longueur proportionnelle à une valeur (ligne 310).

2. Édition d'un jeu de caractères

On demande d'imprimer le jeu de caractère utilisé (dans notre cas il s'agit de l'EBCDIC) sous forme d'un tableau de 64 lignes et 4 colonnes. Chaque élément comprendra :

— le rang du caractère dans le jeu
— son équivalent hexadécimal
— son équivalent binaire
— sa forme imprimée (celle-ci n'existant pas toujours, elle sera blanche dans certains cas).

Voici le programme correspondant et la liste obtenue :

```
00010 ! E512 : Edition d'un jeu de caractères
00020 !
00030   OPEN £1: "PRINTER,RECL=198",OUTPUT,DISPLAY
00040 !
00050  DIM IMP$(255)*25,IM$*25,TIT$*110       ! définition du tableau à imprimer
00060                                          ! 1 élément = 1 caractère du jeu
00070  HE$="0123456789ABCDEF"                  ! table des chiffres hexadécimaux
00080                                          ! boucle de constitution tableau
00090  FOR R=0 TO 255                          ! le rang varie de 0 à 255
00100      IM$=CNVRT$("PIC(ZZ£)",R)            ! édition du rang
00110      G=REM(R,16)+1                       ! calcul du chiffre unités hexa
00120      D=INT(R/16)+1                       ! ------ -- ------- dizaine ---
00130      IM$=IM$&"    "&HE$(D:D)&HE$(G:G)    ! valeur hexa --> tableau
00140      Q=R: B$=""                          !
00150      FOR I=1 TO 8                        ! boucle conversion en binaire
00160          G=REM(Q,2)                      ! calcul unité binaire
00170          Q=INT(Q/2)                      !
00180          B$=STR$(G)&B$                   ! valeur binaire --> zone
00190      NEXT I
00200      IM$=IM$&"  "&B$&"    "&CHR$(R)      ! 3ZONE caractère --> zone
00210      IMP$(R)=IM$                         ! zone --> tableau
00220  NEXT R                                  !
00230 TIT$=RPT$("RANG  HEXA BINAIRE  CAR    ",4)
00240 PRINT £1,USING 250: TIT$                 ! impression du titre
00250 FORM C 110,SKIP 2
00260                                          !
00270 FOR I=0 TO 63                            ! boucle d'impression
00280     PRINT £1,USING 290: IMP$(I),IMP$(I+64),IMP$(I+128),IMP$(I+192)
00290     FORM 4*C 26
00300 NEXT I
00310 STOP
```

RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR	RANG	HEXA	BINAIRE	CAR
0	00	00000000		64	40	01000000		128	80	10000000		192	C0	11000000	é
1	01	00000001		65	41	01000001		129	81	10000001	a	193	C1	11000001	A
2	02	00000010		66	42	01000010		130	82	10000010	b	194	C2	11000010	B
3	03	00000011		67	43	01000011		131	83	10000011	c	195	C3	11000011	C
4	04	00000100		68	44	01000100		132	84	10000100	d	196	C4	11000100	D
5	05	00000101		69	45	01000101		133	85	10000101	e	197	C5	11000101	E
6	06	00000110		70	46	01000110		134	86	10000110	f	198	C6	11000110	F
7	07	00000111		71	47	01000111		135	87	10000111	g	199	C7	11000111	G
8	08	00001000		72	48	01001000		136	88	10001000	h	200	C8	11001000	H
9	09	00001001		73	49	01001001		137	89	10001001	i	201	C9	11001001	I
10	0A	00001010		74	4A	01001010	•	138	8A	10001010		202	CA	11001010	
11	0B	00001011		75	4B	01001011	,	139	8B	10001011		203	CB	11001011	
12	0C	00001100		76	4C	01001100	(	140	8C	10001100		204	CC	11001100	
13	0D	00001101		77	4D	01001101	(	141	8D	10001101		205	CD	11001101	
14	0E	00001110		78	4E	01001110	+	142	8E	10001110		206	CE	11001110	
15	0F	00001111		79	4F	01001111	!	143	8F	10001111		207	CF	11001111	
16	10	00010000		80	50	01010000	&	144	90	10010000		208	D0	11010000	è
17	11	00010001		81	51	01010001		145	91	10010001	j	209	D1	11010001	J
18	12	00010010		82	52	01010010		146	92	10010010	k	210	D2	11010010	K
19	13	00010011		83	53	01010011		147	93	10010011	l	211	D3	11010011	L
20	14	00010100		84	54	01010100		148	94	10010100	m	212	D4	11010100	M
21	15	00010101		85	55	01010101		149	95	10010101	n	213	D5	11010101	N
22	16	00010110		86	56	01010110		150	96	10010110	o	214	D6	11010110	O
23	17	00010111		87	57	01010111		151	97	10010111	p	215	D7	11010111	P
24	18	00011000		88	58	01011000		152	98	10011000	q	216	D8	11011000	Q
25	19	00011001		89	59	01011001		153	99	10011001	r	217	D9	11011001	R
26	1A	00011010		90	5A	01011010	§	154	9A	10011010		218	DA	11011010	
27	1B	00011011		91	5B	01011011	$	155	9B	10011011		219	DB	11011011	
28	1C	00011100		92	5C	01011100	*	156	9C	10011100		220	DC	11011100	
29	1D	00011101		93	5D	01011101	)	157	9D	10011101		221	DD	11011101	
30	1E	00011110		94	5E	01011110	;	158	9E	10011110		222	DE	11011110	
31	1F	00011111		95	5F	01011111	^	159	9F	10011111	▮	223	DF	11011111	
32	20	00100000		96	60	01100000	_	160	A0	10100000		224	E0	11100000	£
33	21	00100001		97	61	01100001	/	161	A1	10100001		225	E1	11100001	
34	22	00100010		98	62	01100010		162	A2	10100010	s	226	E2	11100010	S
35	23	00100011		99	63	01100011		163	A3	10100011	t	227	E3	11100011	T
36	24	00100100		100	64	01100100		164	A4	10100100	u	228	E4	11100100	U
37	25	00100101		101	65	01100101		165	A5	10100101	v	229	E5	11100101	V
38	26	00100110		102	66	01100110		166	A6	10100110	w	230	E6	11100110	W
39	27	00100111		103	67	01100111		167	A7	10100111	x	231	E7	11100111	X
40	28	00101000		104	68	01101000		168	A8	10101000	y	232	E8	11101000	Y
41	29	00101001		105	69	01101001		169	A9	10101001	z	233	E9	11101001	Z
42	2A	00101010		106	6A	01101010	ù	170	AA	10101010		234	EA	11101010	
43	2B	00101011		107	6B	01101011	,	171	AB	10101011		235	EB	11101011	
44	2C	00101100		108	6C	01101100	%	172	AC	10101100		236	EC	11101100	
45	2D	00101101		109	6D	01101101	_	173	AD	10101101		237	ED	11101101	
46	2E	00101110		110	6E	01101110	)	174	AE	10101110		238	EE	11101110	
47	2F	00101111		111	6F	01101111	?	175	AF	10101111		239	EF	11101111	
48	30	00110000		112	70	01110000		176	B0	10110000		240	F0	11110000	0
49	31	00110001		113	71	01110001		177	B1	10110001		241	F1	11110001	1
50	32	00110010		114	72	01110010		178	B2	10110010		242	F2	11110010	2
51	33	00110011		115	73	01110011		179	B3	10110011		243	F3	11110011	3
52	34	00110100		116	74	01110100		180	B4	10110100		244	F4	11110100	4
53	35	00110101		117	75	01110101		181	B5	10110101		245	F5	11110101	5
54	36	00110110		118	76	01110110		182	B6	10110110		246	F6	11110110	6
55	37	00110111		119	77	01110111		183	B7	10110111		247	F7	11110111	7
56	38	00111000		120	78	01111000		184	B8	10111000		248	F8	11111000	8
57	39	00111001		121	79	01111001	µ	185	B9	10111001		249	F9	11111001	9
58	3A	00111010		122	7A	01111010	:	186	BA	10111010		250	FA	11111010	
59	3B	00111011		123	7B	01111011	£	187	BB	10111011		251	FB	11111011	
60	3C	00111100		124	7C	01111100	à	188	BC	10111100		252	FC	11111100	
61	3D	00111101		125	7D	01111101	'	189	BD	10111101		253	FD	11111101	
62	3E	00111110		126	7E	01111110	=	190	BE	10111110		254	FE	11111110	
63	3F	00111111		127	7F	01111111	"	191	BF	10111111		255	FF	11111111	

La partie essentielle de ce programme est la boucle FOR ... NEXT des lignes 90 à 220. Dans cette boucle, le programme :

— convertit le rang en un nombre de deux chiffres hexadécimaux par division par 16 et consultation d'une table de conversion (lignes 100-120).

— convertit également ce rang en binaire par divisions successives par 2 (lignes 150-180). On aurait pu utiliser une table de conversion d'hexadécimale en binaire.

— « récupère » le caractère correspondant au rang par la fonction CHR$ (ligne 200).

La boucle des lignes 270 à 300 permet l'impression du tableau sur quatre colonnes.

3. Cryptage du contenu d'un fichier

Cet exercice a le même énoncé que l'exercice N° 2 du chapitre 4 à une différence près : le jeu de caractères utilisé n'est plus un jeu fictif comprenant les chiffres et les lettres de l'alphabet mais l'EBCDIC dont la liste figure dans l'exercice précédent.

```
00010 ! E531 : CRYPTAGE DU CONTENU D'UN FICHIER
00020 !
00030    OPTION BASE 1
00040                                              ! ouverture des fichiers
00050    OPEN £1: "NAME=ACRYPT",INPUT              ! 1.Fichier à crypter
00060    OPEN £2: "NAME=CRYPTE",OUTPUT             ! 2.Fichier crypté
00070                                              ! DEFINITION DES CHAINES
00080    DIM P(5)                                  !  position roues mobiles
00090                                              !   par rapport à refer.
00100    DIM EAC$(12)*5                            ! - enreg.à crypter(5x12)
00110    DIM ECR$(12)*5                            ! - enreg.crypté   (5x12)
00120    DIM CC$*5                                 ! - clé de cryptage
00130                                              !
00140 !-------------------------------------------!
00150 ! saisie clé cryptage & positiont roues mobiles !
00160 !-------------------------------------------!
00170    INPUT "Entrer la clé de cryptage :": CC$  ! saisie clé de cryptage
00180                                              ! BOUCLE DE POSITIONNT
00190    FOR I=1 TO 5                              !
00200                                              !  la position du car.
00210        P(I)=ORD(CC$(I:I))                    !  dans le jeu est la
00220                                              !  pos.de la roue mobile
00230    NEXT I                                    !
00240 !-------------------------------------------!
00250 ! lect.enr.à crypter,cryptage et écr.enr.crypté !
00260 !-------------------------------------------!
00270    MAT ECR$=("")                             !chaine nulle->enr.crypté
00280    READ £1,USING 290: MAT EAC$ EOF 420       ! lecture enreg.à crypter
00290    FORM 12*C 5
00300                                              ! boucle 12 gpes de 5 car
00310    FOR J=1 TO 12                             !
00320        FOR I=1 TO 5                          ! boucle 5 car du gpe J
00330                                              !
00340        PP=ORD(EAC$(J)(I:I))-P(I)+256         ! positiont caractère
00350        ECR$(J)=ECR$(J)&CHR$(PP)              ! caractère lu sur la
00360                                              ! roue mobile N°I
00370                                              !
00380        NEXT I                                ! car.suivant du gpe J
00390    NEXT J                                    ! gpe suivant de l'enreg.
00400    WRITE £2,USING 290: MAT ECR$              ! écriture enreg.crypté
00410    GOTO 270                                  ! -) trait.enreg suivant
00420    END                                       ! ** FIN DU PROGRAMME **
```

On voit du premier coup d'œil que l'emploi des fonctions ORD et CHR$ a permis de simplifier considérablement le programme de l'exercice N° 2 du chapitre précédent :

— le positionnement des roues est assuré par le chargement dans le tableau P des rangs des caractères de la clé de cryptage (fonction ORD ligne 210).

— le remplacement par le caractère lu sur la roue mobile est effectué par la ligne 240 (on a ajouté 256 pour avoir toujours un argument positif) et la ligne suivante qui va chercher dans le jeu EBCDIC le caractère transposé.

4. Constitution et édition d'un lexique

On entre au clavier chaque mot français du lexique et sa traduction anglaise (en majuscules, chaque mot ayant au plus vingt caractères). On demande d'éditer à partir de ces entrées :

— un lexique français-anglais
— un lexique anglais-français

Dans chaque lexique, le mot original sera imprimé en majuscules et sa traduction en minuscules (voir exemple ci-après).
On supposera que le lexique ne dépasse pas 200 mots dans chaque langue.

```
00010 ! E514 : Edition d'un lexique
00020 !
00030   OPTION BASE 1
00040   DIM FR$(200)*20,AN$(200)*20,TRI(200),MA$*20,MF$*20
00050 !
00060   PRINT NEWPAGE                          ! écran à blanc
00070                                          ! boucle d'entrée des mots
00080   FOR I=1 TO 200
00090       LINPUT "Entrer le mot en français ou FIN :": MF$
00100       IF MF$="FIN" THEN EDIT
00110       LINPUT "Entrer le mot en anglais :": MA$
00120       FR$(I)=LTRM$(MF$)                  ! cadrage à gauche et chargement
00130       AN$(I)=LTRM$(MA$)                  ! des mots dans les tableaux
00140   NEXT I                                 !
00150                                          !-----------------------!
00160   EDIT:MAT TRI=AIDX(FR$)                 ! tri des mots français !
00170                                          !-----------------------!
00180   PRINT £255,USING 190:                  ! impression du titre
00190   FORM X 10,"FRANCAIS",POS 31,"ANGLAIS",SKIP 3
00200                                          !-----------------------!
00210        FOR I=1 TO 200                    ! boucle d'édition FR-AN!
00220                                          !-----------------------!
00230            K=TRI(I)                      ! indice de l'élément trié
00240            MF$=FR$(K)                    ! mot en français
00250            IF MF$="" THEN 330            ! test si mot existant
00260            L=LEN(MF$)                    !
00270            MF$=RPAD$(MF$,20)             ! blancs à la fin du mot
00280            MF$=SREP$(MF$,L+1," ",".")    ! remplacement blanc par point
00290                                          ! à la fin du mot
00300            MA$=LWRC$(AN$(K))             ! mot anglais en minuscules
00310       PRINT £255,USING 320: MF$,MA$      ! impression d'une ligne
00320            FORM X 10,2*C 20,SKIP 2
00330       NEXT I
00340                                          !-----------------------!
00350   MAT TRI=AIDX(AN$)                      ! tri des mots anglais  !
00360                                          !-----------------------!
00370   PRINT £255: NEWPAGE                    !  saut de page
00380   PRINT £255,USING 390:                  ! impression du titre
00390   FORM X 10,"ANGLAIS",POS 31,"FRANCAIS",SKIP 3
00400                                          !-----------------------!
00410        FOR I=1 TO 200                    ! boucle d'édition AN-FR!
00420                                          !-----------------------!
00430            K=TRI(I)                      ! indice de l'élément trié
00440            MA$=AN$(K)                    ! mot en français
00450            IF MA$="" THEN 520            ! test si mot existant
00460            L=LEN(MA$)                    !
00470            MA$=RPAD$(MA$,20)             ! blancs à la fin du mot
00480            MA$=SREP$(MA$,L+1," ",".")    ! remplacement blanc par point
00490                                          ! à la fin du mot
00500            MF$=LWRC$(FR$(K))             ! mot français en minuscules
00510       PRINT £255,USING 320: MA$,MF$      ! impression d'une ligne
00520       NEXT I
00530   STOP
```

```
Entrer le mot en français ou FIN : CARACTERE
Entrer le mot en anglais : CHARACTER
Entrer le mot en français ou FIN : CHAINE
Entrer le mot en anglais : STRING
Entrer le mot en français ou FIN : SOUS-CHAINE
Entrer le mot en anglais : SUBSTRING
Entrer le mot en français ou FIN : CONCATENATION
Entrer le mot en anglais : CONCATENATION
Entrer le mot en français ou FIN : INSERER
Entrer le mot en anglais : INSERT(TO)
Entrer le mot en français ou FIN : TABLEAU
Entrer le mot en anglais : ARRAY
Entrer le mot en français ou FIN : FIN


FRANCAIS              ANGLAIS

CARACTERE...........character

CHAINE..............string

CONCATENATION.......concatenation

INSERER.............insert(to)

SOUS-CHAINE.........substring

TABLEAU.............array

ANGLAIS               FRANCAIS

ARRAY...............tableau

CHARACTER...........caractere

CONCATENATION.......concatenation

INSERT(TO)..........inserer

STRING..............chaine

SUBSTRING...........sous-chaine
```

Ce programme utilise surtout la fonction ''tri d'un tableau'' :

— les mots à trier sont entrés au clavier et chargés dans les tableaux FR$ et AN$ après avoir été alignés à gauche (ligne 110 et 120).

— les rangs des éléments du tableau trié sont mis dans TRI par la fonction AIDX de la ligne 180.

— la boucle d'impression lignes 210-330 aligne, convertit les éléments à éditer suivant l'ordre spécifié dans TRI. On remarquera l'emploi de la fonction SREP$ pour remplacer les blancs par des points (ligne 280),.

— l'édition de la deuxième partie du lexique est identique à la première (lignes 350-520).

5. Édition d'une lettre

On désire éditer une phrase d'un accusé de réception à un règlement. Le texte de cette phrase est le suivant :

Nous accusons réception de votre règlement du JJ/MM/AA d'un montant de XXX XXX,XX francs et nous avons le plaisir de vous adresser ci-joint un chèque de YY YYY,YY francs à titre de ristourne pour paiement comptant.

Le montant du règlement et sa date seront entrés au terminal, le montant de la ristourne sera égal à 2 % du montant du règlement ; la phrase sera affichée au terminal.

```
00010 ! E515 : Edition d'une lettre
00020   OPTION INVP                              ! inversion point et virgule
00030 !
00040   DIM TA$*255,TB$*255
00050 !
00060   TA$="Nous accusons réception de votre réglement du £1 d'un montant "
00070   TA$=TA$&"de £2 Francs et nous avons le plaisir de vous adresser "
00080   TA$=TA$&"ci-joint un chèque de £3 Francs à titre de ristourne pour"
00090   TA$=TA$&" paiement comptant"
00100 !
00110   PRINT NEWPAGE
00120   PRINT ""
00130   INPUT "=======>  Entrer montant et date du réglement :": M,D
00140 !
00150   D$=CNVRT$("pic(ff/ff/ff)",D)             ! édition de la date
00160   M$=CNVRT$("PIC(ZZZ ZZf.ff)",M)           ! édition du montant réglement
00170   R$=CNVRT$("PIC(ZZ ZZf.ff)",M*.02)        ! édition du montant ristourne
00180 !
00190   TB$=TA$                                  ! mouvement vers zone édition
00200   TB$=SREP$(TB$,1,"£1",D$)                 ! insertion de D$ dans TB$
00210   TB$=SREP$(TB$,1,"£2",LTRM$(M$))          ! insertion M$ avec sup.blancs
00220   TB$=SREP$(TB$,1,"£3",LTRM$(R$))          ! insertion R$ avec sup.blancs
00230 !
00240   PRINT ""
00250   PRINT TB$: GOTO 120
```

```
=======>  Entrer montant et date du réglement : 12358.02,120582

Nous accusons réception de votre réglement du 12/05/82 d'un montant de 12 358,00
 Francs et nous avons le plaisir de vous adresser ci-joint un chèque de 247,16 F
rancs à titre de ristourne pour paiement comptant

=======>  Entrer montant et date du réglement : 12.45,100582

Nous accusons réception de votre réglement du 10/05/82 d'un montant de 12,45 Fra
ncs et nous avons le plaisir de vous adresser ci-joint un chèque de 0,25 Francs
à titre de ristourne pour paiement comptant

=======>  Entrer montant et date du réglement : 986123,110582

Nous accusons réception de votre réglement du 11/05/82 d'un montant de 986 123,0
0 Francs et nous avons le plaisir de vous adresser ci-joint un chèque de 19 722,
50 Francs à titre de ristourne pour paiement comptant

=======>  Entrer montant et date du réglement :
```

Ce programme est une application directe de l'édition de zones numériques par CNVRT$ et PIC. Nous avons pris soin de supprimer les blancs à gauche des zones éditées (fonction LTRM$ des lignes 200, 210 et 220).

On remarquera la façon dont est indiqué l'endroit dans la phrase où il faut insérer les variables éditées : cet endroit est spécifié par le caractère £ suivi d'un chiffre.

L'insertion s'effectue alors à l'aide de la fonction SREPS qui remplace le ''£1'', par exemple, par la zone éditée (lignes 220-220). Cette technique est utilisée dans l'opération dite de « fusion texte-fichier » que nous allons voir au chapitre suivant.

6

LA BUREAUTIQUE ET LE LANGAGE BASIC

6.1 GÉNÉRALITÉS

Comme on le sait, la « Bureautique » est une discipline qui vise à automatiser les tâches manuelles dites « de bureau ». Dans une entreprise, ces tâches sont assurées non seulement par les secrétaires dont c'est l'objectif principal mais aussi à des degrés variables par chaque membre du personnel quel que soit son rang et sa fonction. Il ne faut d'ailleurs pas oublier les particuliers qui, en fonction des possibilités et des contraintes de la vie moderne, effectuent chez eux un certain nombre de ces travaux de bureau.

Les spécialistes ont analysé ces tâches et les ont classées en trois grandes rubriques que nous pouvons illustrer en prenant comme exemple le secrétariat d'un service commercial. Ces trois grandes familles sont les suivantes :

a) La gestion des textes

Ces textes sont de nature et de volume très variables : ce sont des lettres envoyées à des clients ou venant de ceux-ci, des communications internes à l'entreprise, des manuels de procédure ou des notices techniques. Gérer ces textes consiste non seulement à les frapper, ce qui constitue la partie la plus évidente de cette gestion mais aussi à les modifier, à

les classer et à les archiver. Il ne faut pas d'ailleurs oublier une pièce essentielle de cette gestion : la recherche d'un texte à partir de critères variés : la date d'émission, le nom de l'auteur ou celui du destinataire, les sujets traités etc...

b) La gestion des fichiers

Il ne s'agit pas là des fichiers « de base » de l'entreprise tels que le fichier du personnel, le tarif ou les comptes-clients, par exemple. Ces fichiers-là sont informatisés en général depuis longtemps mais nous voulons parler des innombrables « petits » fichiers tenus manuellement par chaque employé dans son coin et suivant ses propres normes. Ce peuvent être la liste du personnel assortie de diverses annotations utiles à la secrétaire, un carnet de rendez-vous ou le planning personnel d'un représentant, la liste des notaires ou des géomètres du secteur etc...

Ces fichiers ont des supports également très divers : carnets, répertoires, fiches cartonnées ou même feuilles volantes. Leur « informatisation » n'a, souvent, pas été jugée « rentable » ou possible : les volumes sont trop faibles, les données trop changeantes, les ordinateurs trop gros etc... etc... Par contre, il est envisageable de prendre en compte la gestion de tels fichiers à l'aide de micros ou mini-ordinateurs ou de systèmes de traitement de textes.

La gestion de ces fichiers est, en général, simple et peut se ramener à deux opérations principales :

— la création et la mise à jour d'enregistrements
— l'édition de listes :
ces éditions sont souvent précédées d'opérations de sélection et de tri d'enregistrements.

c) La diffusion des textes

Dans un certain nombre de cas, une secrétaire va taper une lettre ou une communication interne à l'intention d'un client donné ou d'un seul membre du personnel. Dans d'autres cas, par contre, elle adressera le *même texte* à un ensemble d'individus. Elle enverra, par exemple, une lettre d'invitation à tous les avoués du canton, une convocation à une

réunion pour tous les représentants d'un secteur ou une lettre de relance à tous les mauvais payeurs.

Ces textes, une fois imprimés, sont obtenus par rapprochement d'un texte, fixe mais comprenant des zones *variables* (le nom du destinataire, son adresse, la date du jour, le montant dû...) laissées « en blanc » et d'un fichier qui, lui, contient les valeurs à attribuer à ces zones variables. C'est d'ailleurs pourquoi on appelle généralement cette opération une « *fusion Texte-Fichier* ».

Ces opérations de fusion deviennent de plus en plus fréquentes au niveau de l'entreprise alors que les communications écrites individuelles se raréfient. Les campagnes de vente de masse, les opérations de « publipostage » (mailing en anglais) sont devenues monnaie courante. Contrairement aux fichiers évoqués au paragraphe précédent, ceux utilisés dans ces fusions sont la plupart du temps des extraits de fichiers informatiques internes (les fichiers du personnel, des clients, des effets impayés) ou nationaux (le fichier INSEE des entreprises). Des opérations de calcul simples (multiplication d'un prix unitaire par une quantité ou d'un montant par un taux de TVA) peuvent faire partie intégrante de ces traitements.
C'est d'ailleurs en analysant ces opérations de fusion Texte-Fichier que l'on se rend le plus compte que la frontière entre Informatique et Bureautique est bien imprécise sinon totalement artificielle. Un contrat de vente, par exemple, est-il un état informatique assez semblable à une facture, complété d'un certain nombre de constantes alphanumériques ou bien est-ce le résultat de la fusion de textes contractuels avec un fichier « clients » et un fichier « tarif » ? La réponse est si peu évidente que, pour ce problème, on peut trouver dans la même entreprise deux solutions différentes : l'une à base d'ordinateur et de programmes, l'autre utilisant une machine de traitement de textes.

Nous allons maintenant voir comment le langage BASIC avec ses instructions et surtout ses organisations de fichiers peut permettre ces trois grandes fonctions de la Bureautique : la gestion des Textes, celle des Fichiers et la fusion des Textes et des Fichiers.

6.2 GESTION DES TEXTES

Rappelons tout d'abord une évidence : contrairement à ce qui se passe avec une machine à écrire, le texte à imprimer ou plus généralement à gérer est saisi au clavier et stocké sur support magnétique (disque « souple » ou « dur », en général) dans un premier temps, puis relu à partir de ce support et imprimé dans un deuxième temps. Dans cette optique, un texte sera donc simplement un *fichier* et un fichier tout simple puisqu'il ne comporte pas de nombreuses « zones » numériques ou alphabétiques comme dans un fichier informatique « normal » mais *une seule zone* qui est ce que nous avons appelé jusqu'ici une *« chaîne de caractères »*.

Le premier problème que nous allons avoir à résoudre est donc de découper cette chaîne en *enregistrements de LONGUEUR FIXE,* la « longueur variable » n'étant pas, sauf exception, supportée par le BASIC. Pour déterminer ce découpage, on a le choix, a priori, entre deux méthodes diamétralement opposées (avec des variantes plus ou moins proches de l'une ou de l'autre de ces méthodes).

La première méthode est, peut-on dire, celle des éditeurs et des imprimeurs. Elle consiste à saisir le texte « au kilomètre » en ne mentionnant par des caractères spéciaux que les fins de paragraphe et les changements de page voulus par le rédacteur. Sans entrer dans les détails d'une technique par ailleurs fort complexe, on peut dire que tout le travail de mise en page se fait à l'édition. A ce moment seulement, l'utilisateur précise les divers paramètres : longueur de la ligne, espacement, position des marges, dessin des emplacements réservés à l'insertion de figures ou de photographies dans le texte etc... Cette méthode est « informatisée » depuis relativement longtemps soit par ordinateur alimenté à l'origine par cartes ou bandes perforées soit par des systèmes spécialisés comme les composeuses électroniques. Elle utilise maintenant des unités à clavier et écran cathodique comme en entrée, un ordinateur pour le traitement et une photo-composeuse en sortie.

La deuxième méthode est celle de nombreux de fabricants de matériels et de logiciels de traitement de textes. Elle est basée sur deux constatations : primo, dans une entreprise, le format des documents est standard et, par conséquent, la longueur d'une ligne est patiquement constante et secundo, l'utilisateur, que ce soit une dactylo, un représentant ou

un ingénieur n'est pas un professionnel de l'édition et désire voir apparaître sur l'écran de visualisation le texte tel qu'il sera imprimé sur le document final. Dans cette optique, certains constructeurs proposent d'ailleurs des écrans « pleine page ». Avec cette méthode, la plus grande partie de la mise en page est assurée par l'utilisateur au moment de la saisie du texte et cette mise en page est visualisée au terminal. Des touches de commande ou des caractères spéciaux permettent de simuler les margeurs et les crans de tabulation dont sont normalement munies les machines à écrire. Par contre, au moment de l'impression, l'utilisateur aura beaucoup moins de latitude : il pourra seulement choisir la position de la marge, l'espacement entre les lignes et le nombre de lignes par page imprimée. A ces détails près, le texte imprimé correspondra à ce que le rédacteur voyait sur son écran au moment de la saisie.

Le choix entre ces deux méthodes est, à notre avis, facile à faire. La première est celle choisie par les professionnels de la chose imprimée et elle répond aux nombreuses contraintes et à la complexité de leur métier. La programmation correspondante est compliquée surtout si on prend en compte les problèmes de performance du produit et elle dépasse à priori la compétence et les moyens de l'utilisateur du BASIC sur un mini ou un micro-système. La seconde méthode, elle, répond d'une façon acceptable aux besoins d'une entreprise dont l'activité principale n'est pas le traitement de textes. La programmation correspondante est tout à fait accessible par le Basic : elle est basée essentiellement sur la gestion d'un fichier dont chaque enregistrement représente une ligne du texte et aussi une ligne de l'écran cathodique du terminal. Certains utilisateurs considèrent d'ailleurs un texte comme un programme « source » et se servent, faute de mieux, des modules utilisés normalement pour la gestion des programmes COBOL, FORTRAN ou RPG pour saisir et imprimer leurs textes. Dans la suite de cet ouvrage, nous ne considèrerons que ce type de fichier où un enregistrement correspond à une ligne de texte.

La deuxième étape va consister à *choisir une organisation de fichier.* Comme on le sait, le BASIC en supporte trois :
L'organisation séquentielle, l'organisation séquentielle-indexée et l'organisation relative. Il convient d'ajouter parfois à ces trois organisations principales l'organisation dite « bibliothèque » (en anglais : library) qui est une variante de l'organisation séquentielle utilisée en principe pour la gestion des programmes « sources » et des procédures d'exécution des travaux.

A notre avis, l'utilisation des organisations permettant l'accès direct (indexée et relative) n'offre pas beaucoup d'intérêt par rapport à l'organisation séquentielle. Contrairement à un fichier « client » ou « article » où il est utile d'accéder à un enregistrement isolé, le fichier représentant un texte est utilisé *comme un tout* aussi bien au niveau de la saisie (cf. le paragraphe suivant) que de l'impression. De plus, il est en pratique nécessaire de renuméroter chaque ligne du texte après une modification de ce dernier et ceci implique, quelle que soit l'organisation choisie, une recopie complète du fichier correspondant. Sauf exception donc, un texte sera repésenté par un fichier séquentiel ou un « membre » d'une bibliothèque.

Nous allons maintenant examiner le problème suivant qui est celui de la *saisie* et de la *modification* des textes. Nous supposerons, comme c'est maintenant pratiquement toujours le cas, que cette saisie se fait à l'aide d'un terminal à clavier et à écran cathodique. Tout d'abord on voir que l'emploi d'un fichier intermédiaire de travail est nécessaire : le rédacteur du texte doit en effet pouvoir revenir à la situation initiale du fichier « texte » tant qu'il n'a pas donné son accord définitif à la saisie ou à la modification qu'il a effectuée. Compte tenu de cette contrainte, la saisie ou la modification d'un texte se déroulera en trois phases (voir fig. 6.1) :

1. Le programme recopie le fichier « texte » sur un fichier de travail si le texte existe déjà on crée un fichier de travail « vide » s'il s'agit de la première saisie du texte.

2. Le rédacteur entre ou modifie son texte ce qui se traduit par des ajouts, des modifications ou des suppressions d'enregistrements sur le fichier de travail.

3. Le rédacteur considère que la saisie ou les modifications faites sont bonnes et le fichier de travail est recopié sur un nouveau fichier qui représente la nouvelle version du texte ; l'ancien fichier texte et le fichier de travail sont supprimés.
Si le rédacteur désire ne pas tenir compte des modifications apportées, le fichier de travail est supprimé et le fichier texte initial est conservé.

Il est à remarquer que le fichier de travail, contrairement au fichier « texte », doit permettre *l'accès direct* car les ajouts, suppressions ou modifications d'enregistrements doivent se faire très rapidement durant cette phase de saisie *conversationnelle*. Pour ce fichier de travail, on

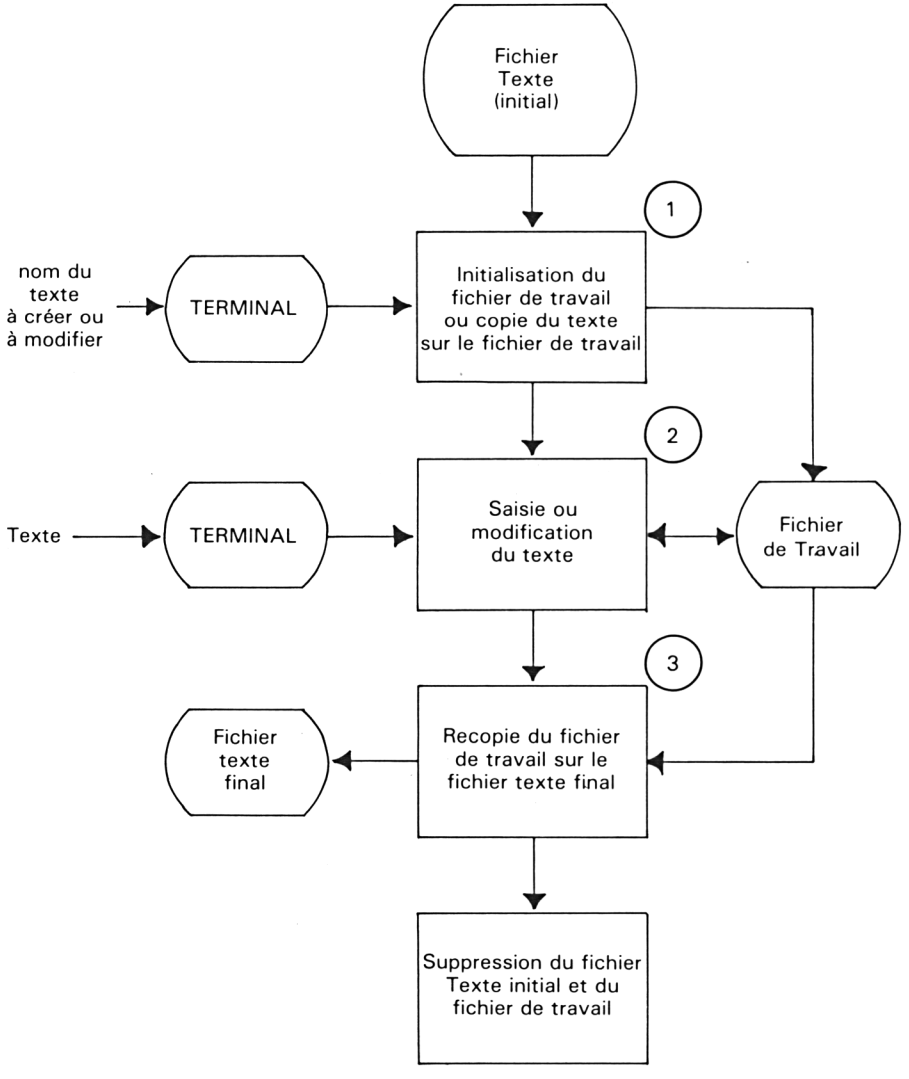


Fig. 6.1 - Saisie ou modification d'un texte

pourra choisir indifféremment l'organisation séquentielle indexée ou l'organisation relative. Dans le premier cas, le numéro de ligne constituera la « clé » de l'enregistrement correspondant à la ligne. Dans le second, ce numéro sera le numéro relatif de cet enregistrement. De toute façon, il sera obligatoire de numéroter les lignes du fichier de travail non pas de un en un mais de cent en cent (par exemple) de façon à permettre l'insertion de lignes entre deux lignes du texte initial.

Voyons maintenant un peu plus en détail quelles fonctions il faut prévoir dans ce programme de saisie et de modification de texte. Ces fonctions peuvent se regrouper en deux catégories :

1. La gestion des lignes du texte

Ces fonctions permettent à l'utilisateur de manipuler une ou plusieurs lignes du texte. Elles doivent assurer au minimum :

— la modification d'une ligne

— la suppression de lignes

— l'ajout de lignes blanches entre deux lignes existantes.

— le déplacement de lignes d'un endroit à l'autre du texte.

— la copie de lignes existantes dans un autre endroit du texte (les lignes existantes restant en place).

— la copie de lignes appartenant à un autre texte à un endroit donné

— l'indication par l'opérateur d'instructions relatives à l'impression du document final : saut de pages, espacements, « soulignage », etc...

La mémorisation de ces indications se fait souvent au niveau de l'enregistrement lui-même. Elle est faite par des « caractères de contrôle » prévus en début ou en fin de l'enregistrement représentant la ligne de texte concernée.

La programmation de ces fonctions en BASIC est simple et ne fait pas appel aux opérateurs et aux fonctions que nous avons vus dans les chapitres précédents. Il s'agit tout simplement *d'une gestion « classique » d'un fichier* simple puisque chaque enregistrement ne comporte qu'une zone, la ligne de texte et quelques caractères de contrôle.

2. La gestion des mots

Là, il n'existe pas de liste « minimale » de fonctions à prévoir. Son

contenu dépend essentiellement de l'imagination de chacun et des ressources disponibles pour la programmation de ces fonctions. Citons par exemple :

— l'insertion ou la suppression de mots au milieu d'un déplacement automatique des autres mots et coupures en fin de ligne.

— détection automatique des lignes comportant un mot donné

— remplacement d'un mot par un autre dans tout ou partie du texte

— utilisation d'abréviations.

La programmation de ces fonctions portant sur les mots fera, bien sûr, largement appel à toutes les instructions que nous avons vues depuis le début de cet ouvrage (POS, SREP$ etc...). Il ne faut pas cacher que la programmation de ces fonctions portant sur les mots est *beaucoup plus complexe* que celle de la gestion des lignes, le problème le plus épineux à résoudre étant celui des *coupures de mots en fin de ligne.* Avant de se lancer dans tout développement, il faudra bien examiner l'utilité de tèlle ou telle fonction demandée par les utilisateurs. Il faut, de plus, prendre en compte un facteur externe à la programmation qui est celui du fonctionnement du terminal : la plupart des terminaux comportent une unité de traitement microprogrammé et une mémoire interne (un « buffer ») et des touches de fonction du clavier permettent d'insérer ou de supprimer des caractères sur l'écran, sans communiquer avec l'ordinateur. Il nous paraît donc inutile de dupliquer au niveau du programme BASIC des fonctions assurées par les microprogrammes des terminaux.

Nous allons passer maintenant à l'étape suivante qui est celle de *l'impression des documents.* Contrairement à la saisie et si on admet que la longueur de la ligne d'impression est celle utilisée à la saisie, la programmation en est particulièrement simple : il s'agit d'un programme élémentaire de liste avec saut et comptage de pages en fonction des options indiquées à la saisie et éventuellement modifiées au moment de l'impression : type d'espacement, nombre de lignes dans une page, etc...

Le dernier sujet que nous allons évoquer concernant la gestion des Textes est celui de la *recherche d'un document,* lettre, mémo, article du manuel de procédure etc... Chacun de nous a en mémoire les scènes tragi-comiques où Chose demande à sa secrétaire de lui trouver la note de Truc à Machin (de fin mai ou plutôt de début juin) concernant - entre autre - la climatisation de la salle N° 3 et où ladite secrétaire se promène du « chrono » au « classement par sujet » sans résultat apparent.

· Là encore, il s'agit d'un problème - celui de la recherche documentaire - qui est connu et informatisé depuis longtemps au niveau des grandes entités : bibliothèques spécialisées, centres de documentation, « banques de données » etc...
Il n'est, bien sûr, pas question de mettre en place dans un secrétariat des bases de données sophistiquées mais le BASIC nous permet d'apporter une solution simple et satisfaisante, à condition de traiter des volumes d'information relativement faible (de l'ordre du millier de documents).

Tout le système est basé sur la constitution d'une *fiche descriptive de document.* Son contenu est variable suivant les cas mais on y trouvera au moins en ce qui concerne un secrétariat :

— la référence du document.

— un code de classement permettant de retrouver le document une fois classé.

— sa date d'émission

— le nom de son émetteur

— le nom du ou des destinataires le cas échéant

— un résumé du document.

Ce résumé peut se limiter à une liste de « mots-clés » ou bien se présenter sous forme d'un véritable texte.

La constitution de cette fiche n'implique pas forcément la rédaction d'un document : cette tâche peut être intégrée à la saisie du texte lui-même ou bien il suffit de porter quelques annotations en marge du document. Le contenu de la fiche est ensuite enregistré dans un fichier « descriptif ».

La recherche d'un document se fera bien évidemment par mots-clés : la personne intéressée pose sa question sous forme d'une liste de mots (auteur, destinataire, sujet etc...) liés entre eux par des « et » ou des « ou » logiques.

Le processus de recherche peut être plus ou moins élaboré :

— dans un premier temps, on peut se contenter de lire le fichier « descriptif » de bout en bout et d'analyser chaque enregistrement (à l'aide de la fonction POS, par exemple) pour voir si celui-ci répond à la question.

— si cette méthode très simple conduit à des temps de réponse trop longs, il faudra créer un fichier « *descriptif inversé* » donnant pour cha-

que mot-clé (y compris les noms des émetteurs et des destinataires) les références des documents (c'est-à-dire les clés des enregistrements) où ils se trouvent.

— on peut ensuite sophistiquer le système au niveau des mots-clés en prenant en compte les synonymes, les racines, les mots non-significatifs, les conjugaisons etc...

De toute façon, il ne faut pas perdre de vue qu'un système de recherche documentaire dans un secrétariat doit être *simple* et que l'on n'a pas affaire à des juristes ou à des chercheurs mais à des employés d'une entreprise recherchant des documents courts et de structures peu complexe.

6.3 LA GESTION DES FICHIERS

Nous avons défini au début de ce chapitre à quel genre de fichiers nous avons affaire : il s'agit de fichiers gérés par une personne ou un petit groupe de personnes. Ces fichiers peuvent être créés sur place ou peuvent provenir d'une extraction d'un fichier informatique et être complétés par les utilisateurs localement.

Formulons tout d'abord quelques remarques de bon sens à propos de ces fichiers :

— leur volume est *faible* et ils tiennent facilement sur un mini disque souple. Une secrétaire ne gère pas le fichier des 20 000 personnes de son entreprise mais celui des 32 personnes appartenant au département qu'elle supporte.

— leurs utilisateurs *ne sont pas des informaticiens :* ils sont assez peu sensibles aux beautés d'un dessin d'enregistrement, du décimal « packé » ou de la virgule flottante.

— la nature et la forme des données sont *très variables dans l'espace et dans le temps.* Un changement de la politique commerciale de l'entreprise peut amener un représentant à supprimer certaines données de ses fiches « client » de même qu'une secrétaire doit ajouter une colonne au cahier d'enregistrement des heures supplémentaires suite à une modification de la législation du travail. Or, les fichiers informatiques sont, on le

sait, particulièrement rebelles au changement et c'est d'ailleurs souvent la raison d'être de ces fichiers « parallèles » ou « pirates ».

— les contraintes de la codification des données n'enthousiasment en général pas les utilisateurs. Ces codifications datent souvent de l'ère de la carte perforée (où la « codif » était une nécessité technique) et ont été prorogées au fil des ans par les informaticiens bien que les systèmes d'aujourd'hui permettent dans bien des cas de s'en passer. Les utilisateurs préfèrent les *données en clair* ou, à la limite, les abréviations, les codes mnémoniques et les acronymes officiels ou officieux.

Ceci dit, la notion de fichier est particulièrement familière à l'informaticien puisque les fichiers sont à la base même de l'informatique. Les fichiers qui nous intéressent peuvent être, à notre sens, conçu de deux façons différentes :

1. Comme des fichiers informatiques « classiques » ayant un *format fixe,* chaque donnée occupant une zone allant de telle à telle position, étant numérique ou alphanumérique, etc...

2. *Comme des textes,* chaque enregistrement étant une chaîne de caractères formée de mots-clés suivis de la ou des valeurs correspondantes. C'est ce qu'on peut appeler un fichier en « *format libre* ».

Prenons un exemple très simple pour mieux expliciter ces deux conceptions fondamentalement différentes.

On désire créer un fichier du personnel d'un service à partir d'un imprimé ayant la forme suivante :

nom : DUVAL prénoms : GILBERT, LEON, PAUL
né le : 12/10/47 à BREST conjoint : MARTIN DENISE
prénom et date : MARC 5/10/69
de naissance : MARIE-CHRISTINE 21/5/72
des enfants : EVE 30/7/75

Si on choisit de constituer un fichier en format FIXE, tous les enregistrements auront *le même dessin,* celui-ci, par exemple :

positions 1 à 20 : NOM
positions 21 à 30 : PRÉNOM N°1
positions 31 à 40 : PRÉNOM N°2
positions 41 à 50 : PRÉNOM N°3

positions 51 à 57 : DATE DE NAISSANCE (JJMMAA)
positions 58 à 67 : LIEU DE NAISSANCE
positions 68 à 77 : NOM DU CONJOINT
positions 78 à 87 : PRÉNOM DU CONJOINT
positions 88 à 97 : PRÉNOM ENFANT N° 1
positions 98 à 103 : DATE DE NAISSANCE ENFANT N° 1

et ainsi de suite pour un certain nombre d'enfants...

Si maintenant nous choisissons de créer un fichier en format libre, l'enregistrement *n'aura pas de dessin bien défini* et se présentera sous forme d'une *chaîne de caractères.* En ce qui concerne l'exemple ci-dessus, cette chaîne peut être ainsi constituée :

nom:DUVAL;pré:GILBERT,LEON,PAUL,né l:12/10/47 à BRETS ; cjt:MARTIN DENISE ; enf : MARC 5/10/69, MARIE-CHRISTINE 21/5/72,EVE 30/7/75 ;

Comme le lecteur peut le constater, cet enregistrement est un reflet assez fidèle de ce que chacun peut lire sur la fiche d'origine. On y trouve :

— des mots-clés qui sont des abréviations du nom de la donnée (pré pour prénom, cjt pour conjoint etc...).

— les valeurs des données (DUVAL,LEON,MARC 5/10/69, etc...)

— des caractères délimiteurs permettant de séparer les mots-clés des valeurs (''','':'' et '';'' dans notre exemple)

Examinons maintenant quelles sont les implications du choix de l'une ou l'autre conception au niveau des organisations de fichiers et des programmes de mise à jour et d'exploitation (sélection, tri, liste, interrogation).

a) Fichiers en format fixe

L'organisation séquentielle indexée nous paraît la plus adaptée à ce genre de fichier : elle permet à la fois l'exploitation séquentielle et l'accès direct aux données. Le choix de la clé est souvent évident et le problème des clés « en double » (deux personnes ayant le même patronyme) est facilement soluble.

En ce qui concerne la mise à jour du fichier et son exploitation, remarquons tout d'abord qu'il est à peu près indispensable de créer et

d'utiliser un « *dictionnaire de données* » (Data Dictionnary en anglais). Ce dictionnaire de données donne pour chaque donnée, son nom symbolique (celui qui, en principe, est employé dans les programmes), sa nature, sa longueur et sa position dans l'enregistrement du fichier, sa description, les valeurs permises (s'il s'agit d'un code). Voici un exemple de dictionnaire de données correspondant à notre exemple de départ.

Nom	*Description*	*nature*	*longueur*	*pos. début*
NOM$	nom de l'employé	alpha.	20	1
PRE1$	prénom N° 1	alpha.	10	21
DATNAI	date de naissance	numer.	6	51

etc...

Ce dictionnaire de données peut n'être qu'une simple liste à laquelle l'utilisateur se réfère s'il veut savoir où trouver telle ou telle donnée dans un enregistrement. Mais, très souvent, ce dictionnaire est lui-même un fichier auquel les programmes d'exploitation peuvent accéder. C'est le cas de nombreux logiciels de gestion de fichiers qui, à partir d'un dictionnaire de données « génèrent » des modules de mise à jour, de liste et d'interrogation pour ces fichiers. Les programmes ainsi obtenus sont eux-mêmes des programmes en BASIC, en COBOL, en RPG etc... Ce processus est illustré par la figure 6.2.

Les fournisseurs de logiciels présentent de plus de ces modules standard des « langages d'interrogation » (Query Languages) permettant à l'utilisateur de programmer à l'aide d'instructions simples des interrogations personnalisées ou des listes « à la demande ». Il est à noter que le BASIC est en soit *un excellent langage d'interrogation* et que, de plus, il est la plupart du temps interprétatif ce qui élimine les compilations souvent longues des programmes écrits en COBOL ou en RPG, par exemple. Il n'est pas du tout irréaliste d'apprendre aux utilisateurs « de la base » le BASIC ou du moins ses instructions logiques et arithmétiques quitte à leur fournir des routines « squelette » de lecture des fichiers auxquelles il suffira d'incorporer les modules logiques et arithmétiques écrits par ces utilisateurs. En voici un exemple très simple :

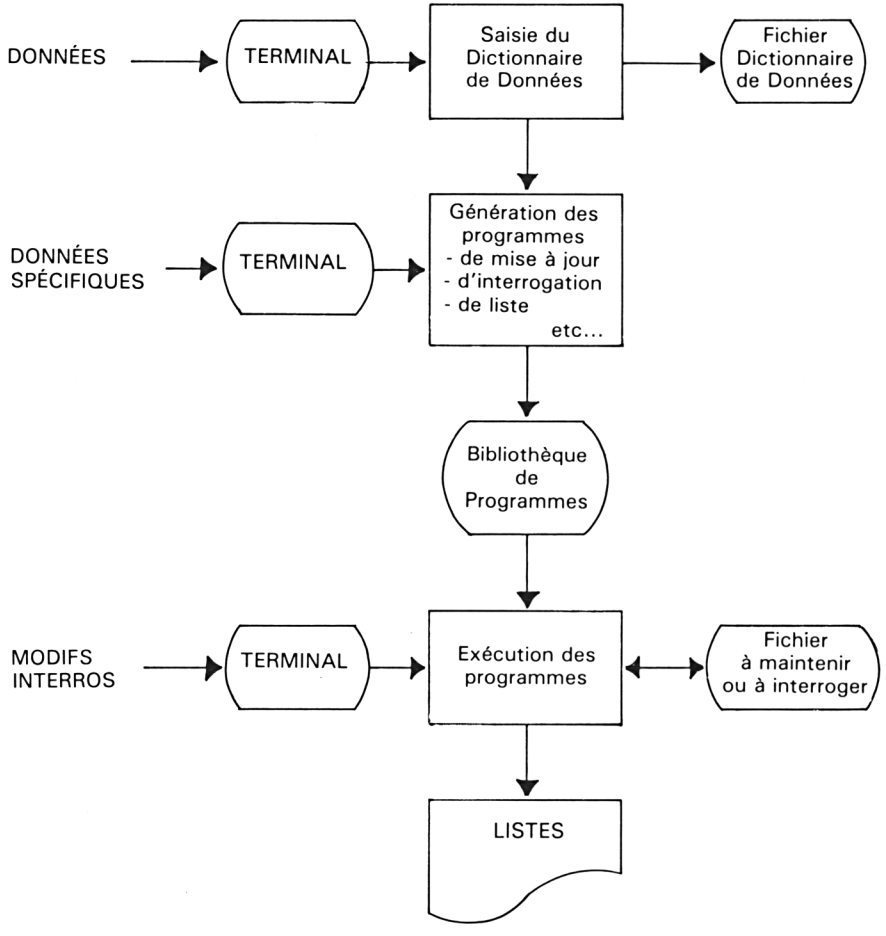


Fig. 6.2 - Génération de logiciels standard de gestion des fichiers

```
10 OPEN £ : ''NAME = CLIENT''
20 LEC: READ £,USING 30 : NOM$,A$,B$ EOF 70
30 GOTO TEST
40 GOTO LEC
50 PRT : PRINT NOMS
60 GOTO LEC
70 STOPO
```

routine squelette fournie par l'informaticien

```
100 TEST : IF A$ = ''112'' AND B$ = ''AB14'' THEN PRT
110 IF A$ = ''356'' OR B$ = ''XW12'' THEN PRT
120 GOTO LEC
```

module logique sélection écrit par l'utilisateur

La « fusion » entre la routine squelette et les modules logiques se fait automatiquement par l'intermédiaire des « commandes » BASIC du système.

Pour clore cette étude des fichiers en format fixe, essayons de voir quelles en sont, sinon les défauts, mais les limitations. Tout d'abord, ces fichiers supportent assez mal le changement :

— l'ajoût d'une donnée non prévue à l'origine demande, non seulement la modification du dictionnaire de données, mais la « régénération » des modules standard de gestion du fichier.

— La modification de longueur d'une zone est encore plus pesante : il devient nécessaire de recopier le fichier en déplaçant toute les zones.

De plus, la prise en compte des données « multiples » (plusieurs prénoms ou plusieurs enfants, dans notre exemple initial) n'est pas toujours évidente. Il est en effet obligatoire de prévoir un nombre maximal de « postes » pour ces données multiples et cela peut entraîner à la limite la création de plusieurs types d'enregistrement ce qui complique considérablement la gestion et l'exploitation des fichiers.

b) Les fichiers en format libre

Rappelons-le, un fichier en format libre est en réalité un fichier texte analogue à ceux que nous avons vus au paragraphe précédent mais où la longueur d'enregistrement est arbitraire et n'a rien à voir ni avec la largeur de la ligne d'impression ni avec celle de l'écran cathodique.

L'organisation de fichier est a priori séquentielle, chaque enregistrement étant repéré par un N° d'ordre en séquence. On peut, néanmoins

utiliser le séquentiel indéxé si une clé commode peut être trouvée : dans notre exemple, ce pourrait être, bien sûr, le nom (assorti des initiales des prénoms si on veut éviter les « doubles »). L'enregistrement serait légèrement différent : une donnée serait en format fixe : la clé, c'est-à-dire le nom, et les autres données en format libre. Rien n'interdit d'ailleurs de « panacher » ces deux types de format pour profiter des avantages de chacun d'eux. Les données du fichier réputées intangibles seront en format fixe alors que les données plus mouvantes seront en format libre.

L'organisation relative peut également être utilisée surtout lorsqu'aucune clé possible n'apparaît clairement. On accumule alors les enregistrements un par un dans l'ordre d'arrivée. Le fichier peut toujours être utilisé séquentiellement mais sa mise à jour sera facilitée si on connaît le numéro d'ordre de l'enregistrement à modifier.

Examinons maintenant les problèmes de mise à jour de ce type de fichier. Il n'y a pas besoin de recourir à un dictionnaire de données, il faut, au plus, une liste des mots-clés autorisés mais ils peuvent être judicieusement choisis de façon à éviter toute consultation trop fréquente de cette liste. Les opérations de création et de suppression d'enregistrements sont simples puisqu'elles sont liées à l'organisation utilisée. L'opération de modification peut sembler plus complexe mais en y regardant bien, on voit qu'elle se ramène à des recherches de caractères séparateurs ainsi qu'à des insertions, des suppressions ou des remplacements de sous-chaînes, choses que nous savons faire depuis les chapitres 3 et 4 de ce livre.

La recherche d'enregistrements d'après certains critères ne pose pas plus de problèmes si on utilise les fonctions BASIC de recherche de mots dans un texte. Un seul problème délicat peut se poser au niveau de l'exploitation du fichier : celui du tri car la totalité des programmes de tri sont conçus pour trier des fichiers en format fixe. S'il n'y a pas assez de place mémoire pour trier le fichier à l'aide des fonctions AIDX ou DIDX, il faudra créer des fichiers « index » en format fixe et les trier par l'utilitaire du système.

Si nous reprenons les conclusions du paragraphe précédent, nous pouvons dire que les fichiers en format variable permettent :

— l'ajout ou la modification de données dans les enregistrements *sans avoir à modifier* les données déjà présentes et *sans avoir à recopier* le fichier initial.

Il existe, bien sûr, une limite physique à la longueur de l'enregistrement mais il est possible de créer sans problème des enregistrements « suite » si la capacité de l'enregistrement initial s'avère insuffisante.

— la gestion des données multiples à l'intérieur d'un même enregistrement. Il n'y a pas de limite a priori au nombre de ces données si ce n'est la longueur de l'enregistrement, problème évoqué plus haut.

Par contre, il faut reconnaîatre que le nombre de logiciels disponibles pour le traitement de ce type de fichier est très limité.

Rappelons au lecteur que ces conclusions sont valables pour des fichiers de faible volume. Pour obtenir les mêmes possibilités de traitement sur les « grands » fichiers de l'entreprise, il faut recourir à des « bases de données » sophistiquées et nécessitant un investissement initial important.

6.4 LA FUSION DES TEXTES ET DES FICHIERS

Tous les programmeurs font ou ont fait de la « fusion Texte-Fichier » comme M. Jourdain faisait de la prose, c'est-à-dire sans le savoir. Une facture, un relevé de compte ou un bulletin de paie résultent d'une ou plusieurs opérations de fusion texte-fichier. Les textes sont ceux préimprimés sur le document ou ceux sortis par le programme ; les fichiers utilisés sont ceux de l'entreprise : le tarif, le fichier client ou les divers fichiers comptables. Ce type de fusion ne nous intéresse pas beaucoup : il représente l'écriture de plusieurs programmes et il s'agit d'opérations répétitives programmées par du personnel en général spécialisé.

Au niveau d'un secrétariat, le problème qui se pose est de reproduire *x* fois le même texte en changeant à chaque coup la valeur de quelques « variables » : le nom du destinataire, son adresse, une date etc... Il s'agit en général d'opérations beaucoup plus simples qu'une facturation ou une paie : diffusion d'une note à un service, envoi d'une lettre d'invitation à des clients ou convocation d'un certain nombre de personnes à une réunon de travail. Pour ce faire, on dispose (voir fig. 6-3) :

— d'un texte créé ou modifié pour la circonstance

— d'un fichier en format fixe ou libre. Il s'agira souvent d'un extrait de fichier obtenu par sélection et tri d'un fichier informatique

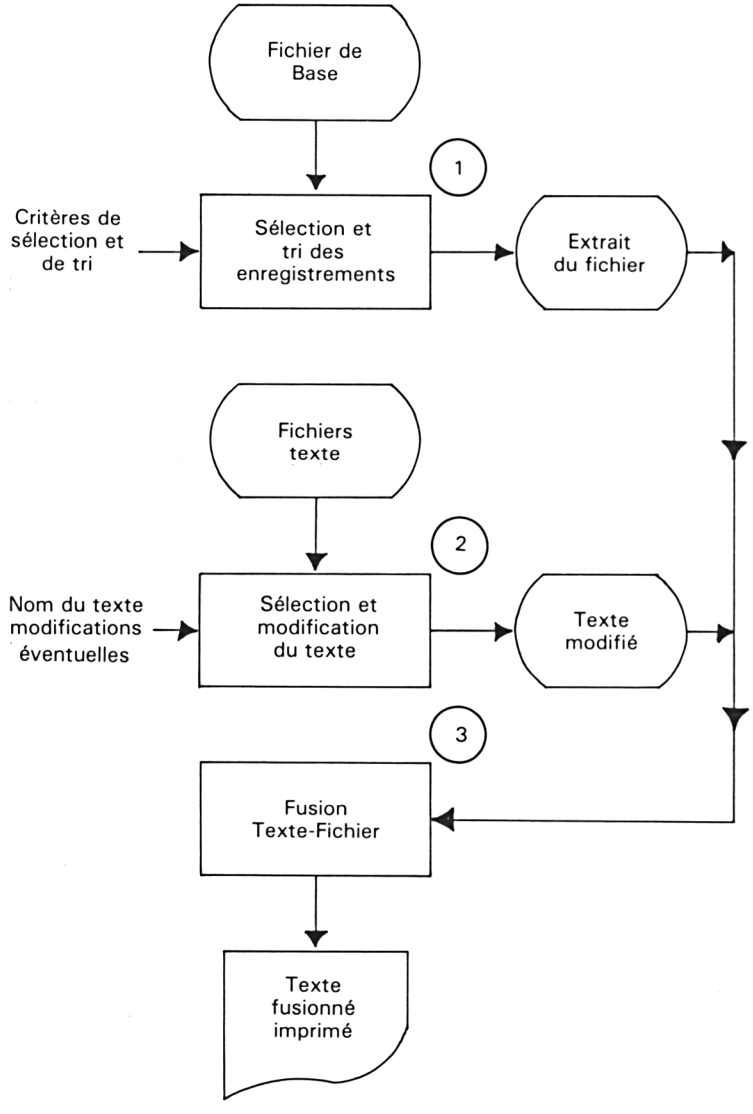


Fig. 6.3 - Fusion Texte-Fichier

· La logique de l'opération est simple et identique dans tous les cas ; elle est représentée figure 6.4. Elle consiste :

1. à lire le texte en mémoire sous forme d'un tableau
2. à lire un enregistrement du fichier
3. à mouvementer les données du fichier à l'intérieur du texte
4. à imprimer le texte
5. à recommencer la lecture en 2, jusqu'à la fin du fichier.

Cette opération peut faire l'objet d'un programme « squelette » mais toute la difficulté consiste à trouver une méthode pour *mouvementer (point 3) les données du fichier vers le texte* sans avoir à écrire un programme ou une partie de programme à chaque fois. Cette fois encore, deux possibilités sont offertes : le format « fixe » et le format « libre ».

En format fixe, on entrera dans le programme « squelette » une table indiquant pour chaque variable :

— ses positions de début et de fin dans l'enregistrement

— le numéro de la ligne de texte et la position dans cette ligne où la variable doit être placée.

Par exemple, l'élément de cette table figurant ci-dessous :

101 117 012 044

spécifiera que la variable se trouvant dans les positions 101 à 117 de l'enregistrement doit être imprimée sur la ligne n° 12 du texte à partir de la position 44 de cette ligne. Ceci suppose évidemment que la place nécessaire au milieu du texte ait été prévue au moment de la rédaction de ce texte.

On peut améliorer ce système en prévoyant, par exemple, un « code d'édition » permettant d'indiquer qu'une zone doit être, non seulement mouvementée, mais aussi éditée suivant un certain « masque » (dates, montants, etc...). Si une variable se trouve incluse au milieu d'un texte, il faudra penser à supprimer les blancs au milieu du texte définitif.

La deuxième méthode (format libre) consiste à indiquer, dans le corps du texte à éditer, la place des variables par un caractère spécial suivi du nom abrégé de la variable. De même, le fichier sera présenté en format libre : chaque enregistrement comprend le nom des variables

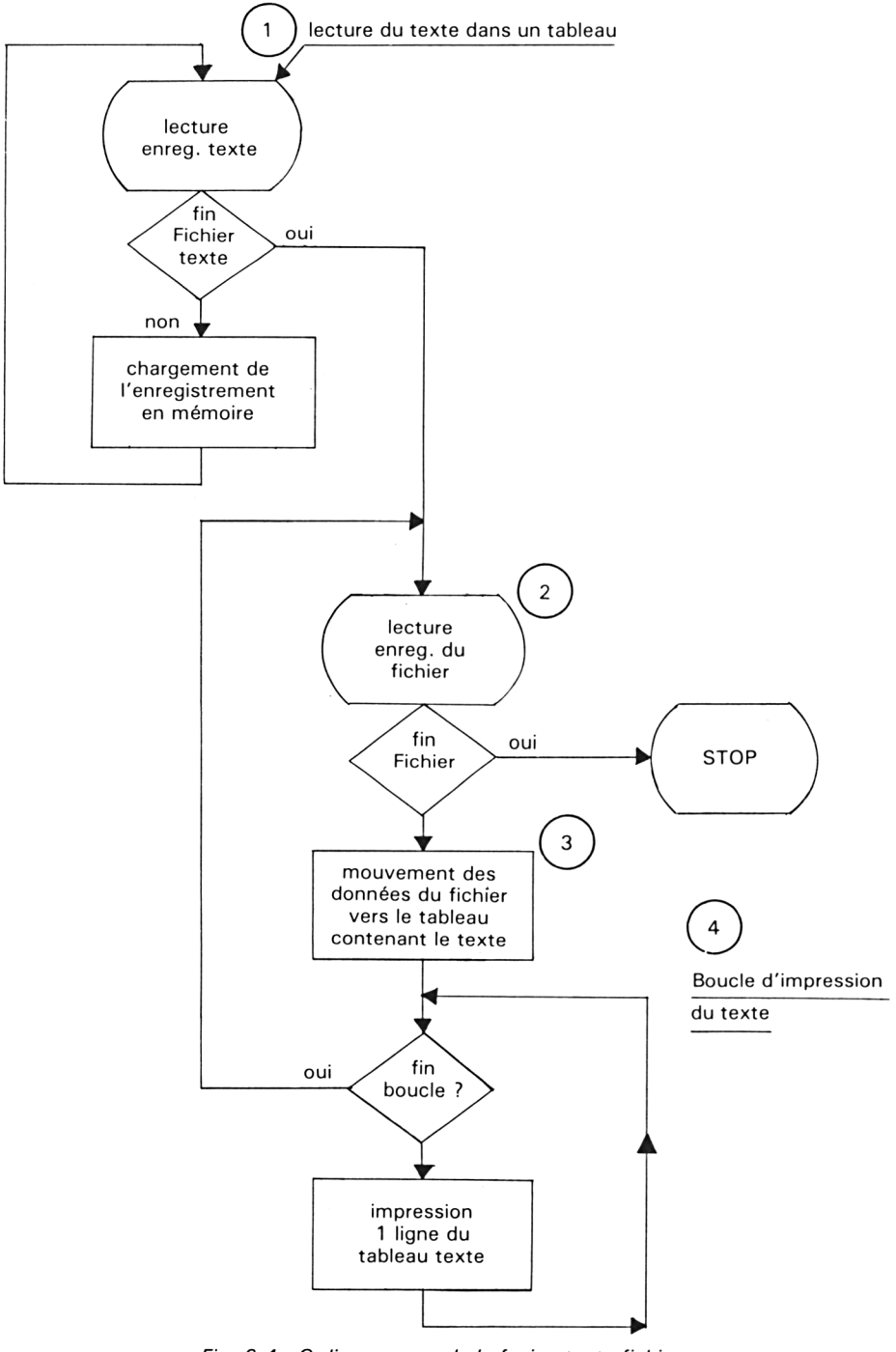


Fig. 6.4 - Ordinogramme de la fusion texte-fichier

suivi de leurs valeurs avec des caractères séparateurs. Voici un exemple d'application de cette méthode :

TEXTE A FUSIONNER :

*NOM Paris, le *DATE
*AD1
*AD2
*AD3
*AD4

*TITRE,

Nous vous prions de vous présenter à nos bureaux le *DATA pour y subir des tests psychotechniques.

FICHIER A FUSIONNER

Enreg. N° 1 :
*DATE = 12/05/82, *NOM = M.DUPONT, *AD1 = 36300 LE BLANC
*DATA = 25/05/82, *TITRE = Monsieur

Enreg. N° 2 :
*DATE = 12/05/82, *NOM = MELLE MARTY; *AD1 = RESIDENCE V. HUGO
*AD2 = BATIMENT C, *AD3 = 14 RUE ALFRED DE VIGNY,
*AD4 = 37000 TOURS; *DATA = 3/6/82, *TITRE = Mademoiselle

RESULTAT DE LA FUSION :

Texte N° 1

M. DUPONT **Paris, le 12/05/82**
36300 LE BLANC

Monsieur,

Nous vous prions de vous présenter à nos bureaux le 25/05/82 pour y subir des tests psychotechniques.

Texte N° 2

MELLE MARTY
RESIDENCE V. HUGO
BATIMENT C
14 RUE ALFRED DE VIGNY
37000 TOURS

Paris, le 12/05/82

Mademoiselle,

Nous vous prions de vous présenter à nos bureaux le 3/6/82 pour y subir des tests psychotechniques.

On notera que la présence de toutes les variables indiquées dans le texte n'est pas *nécessaire* dans chaque enregistrement : ainsi les variables *AD2, *AD3, *AD4 ont-elles été omises dans le premier enregistrement. Cette méthode nous paraît plus souple que la première car elle évite de calculer les positions dans une ligne et de remplir une table supplémentaire. Par contre, elle oblige à prévoir un fichier en format libre. La programmation en BASIC ne pose aucun problème. Le traitement consistera à :

— rechercher dans le texte le nom d'une variable (fonction POS)

— ce nom étant trouvé, rechercher dans l'enregistrement du fichier la valeur attribuée à cette variable (fonction POS).

— si cette valeur est trouvée, remplacer le nom de la variable dans le texte par cette valeur.

Ces méthodes sont applicables pour des opérations de fusion Texte-Fichier simples. Pour des opérations plus compliquées, il sera préférable d'écrire un programme adapté à chaque opération.

6.5 CONCLUSION

Nous pensons avoir, au cours de ce chapitre, montré au lecteur que les opérations prises en compte par la Bureautique sont parfaitement programmables à l'aide du BASIC, éventuellement complété par l'emploi éventuel de logiciels standard de gestion de fichier. L'utilisation du BASIC permet, de plus, de n'avoir qu'un *seul langage de program-*

mation aussi bien pour le traitement des applications de gestion que du traitement des textes. Dans la pratique, on s'aperçoit vite que les deux sont intimement liés et qu'il n'y a pas de différence fondamentale entre les deux types de traitement. La seule différence essentielle ne réside pas, à notre avis, dans les traitements eux-mêmes, mais dans les habitudes, la psychologie et le mode de pensée des « clients » respectifs. L'informaticien qui aborde donc pour la première fois les problèmes « bureautiques » aura donc tout intérêt à considérer le facteur humain au premier chef quite à faire passer au second plan les considérations techniques de langage, de système ou de performances. Il ne faut pas perdre de vue que ces applications bureautiques sont des applications simples si on les compare à toutes celles que l'informatique a permis de prendre en compte durant ces trente dernières années. De plus, les matériels que ce soit du point de vue de la mémoire, de la rapidité de calcul où des capacités sur disques s'avèrent en général sur-dimensionnés pour ces applications de bureautique. Le premier impératif est donc pour l'informaticien de « faire simple », étant bien entendu que cette simplicité est à considérer du point de vue... des utilisateurs.

7

EXERCICES RÉCAPITULATIFS

1. Indexation automatique d'un document

On demande de constituer automatiquement l'Index d'un document à partir de ce document lui-même supposé enregistré sous forme d'un fichier. Cet index se présente sous forme d'une liste donc chaque ligne comprend un mot suivi des numéros des pages où celui-ci figure, exemple :

LANGAGE : 12,45,78,105
LANGUE : 15, 48, 156

Le fichier « Texte » est composé d'enregistrements de 64 caractères de long et a le dessin suivant :

pos. 1 à 60 : une ligne de Texte
pos. 61 à 64 : le N° de la page.

Les mots (200 au maximum) devant figurer dans l'index sont entrés au début de traitement par le rédacteur de texte.
Celui-ci a la possibilité d'entrer, soit un mot en entier, soit une racine des mots à rechercher suivi d'un astérisque. Dans ce dernier cas, il faudra rechercher dans le texte tous les mots commerçant par cette racine. Par exemple, si le rédacteur entre la racine ''PROGRAMM*'' devront figurer dans l'index les pages où apparaissent les mots PROGRAMMES, PROGRAMMEURS, PROGRAMMATION, PROGRAMMÉES, etc... Un mot qui apparaîtrait plus d'une fois dans la même page ne devra figurer qu'une seule fois dans l'index. On supposera qu'un mot ne peut pas être « à cheval » sur deux lignes.

Voici le programme correspondant à ce traitement :

```
00010 ! E701 : INDEXATION AUTOMATIQUE D'UN DOCUMENT
00020 !
00030   OPTION BASE 1                                  ! origine des indices en 1
00040   DIM M$(200)*18                                 ! Def.tableau des mots
00050   DIM P$(200)*40                                 ! Def.tableau des pages
00060   DIM TRI(200)                                   ! Def.tableau de tri
00070 !
00080   FOR I=1 TO 200                                 ! BOUCLE DE SAISIE DES MOTS
00090       INPUT "Entrer un MOT ou FIN": MOT$
00100       IF MOT$="FIN" THEN 130                     ! test si fin de saisie
00110       M$(I)=MOT$                                 ! chargement mot
00120   NEXT I
00130 !
00140   DIM LS$*64,L$*82                               ! définition & initialisation
00150   LS$=" ,;:.?()!'"                               ! de la liste des séparateurs
00160 !
00170   OPEN #1: "NAME=TEXTE",INPUT                    ! ouverture fichier texte
00180   READ #1,USING 190: L$,PG EOF 400               ! lecture ligne de texte
00190   FORM C 60,N 5
00200   L$="."&L$&"."                                  ! séparat.avant & après ligne
00210 !
00220   FOR L=1 TO I                                   ! BOUCLE RECHERCHE DES MOTS
00230                                                  ! de l'index ds la ligne lue
00240       MOT$=M$(L)                                 !
00250       P=POS(MOT$,"*",1)                          ! test si * dans le mot
00260       IF P>0 THEN MOT$(P:P)=""                   ! suppress.* pour recherche
00270       Q=POS(L$,MOT$,1)                           ! recherche mot dans ligne
00280       IF Q=0 THEN 380                            ! mot non trouvé
00290       IF POS(LS$,L$(Q-1:Q-1),1)=0 THEN 380       ! test si separ.avant le mot
00300       IF P>0 THEN 360                            ! si * -> mot trouvé
00310       LO=Q+LEN(MOT$)                             ! calcul posit.après le mot
00320       IF POS(LS$,L$(LO:LO),1)=0 THEN 380         ! test si separ.après le mot
00330                                                  ! si le mot est trouvé -->
00340       PA$=","&STR$(PG)                           ! ajout virgule/convers.page
00350       IF POS(P$(L),PA$,1)>0 THEN 380             ! page déja dans la liste
00360       P$(L)=P$(L)&PA$                            ! ajout page dans la liste
00370                                                  ! l'élément du tableau pages
00380   NEXT L                                         ! fin boucle de recherche
00390   GOTO 180                                       ! ---> lect.ligne texte suiv.
00400 !
00410   MAT TRI=AIDX(M$)                               ! TRI DU TABLEAU DES MOTS
00420 !
00430   FOR L=1 TO 200                                 ! BOUCLE EDITION DE LA LISTE
00440       N=TRI(L)                                   !
00450       MOT$=M$(N)                                 ! extraction mot du tableau
00460       IF MOT$="" THEN 510                        ! ->mot suivant si long.nulle
00470       PA$=P$(N)                                  ! extraction ligne page
00480       IF PA$="" THEN 510                         ! mot non utilisé -> mot suiv
00490       PRINT #255,USING 500: MOT$,PA$(2:40)       ! impression ligne
00500       FORM C 18," : ",C 40
00510   NEXT L
```

Ce programme comprend quatre parties :

1. La saisie par l'utilisateur des mots faisant partie de l'Index (lignes 80 à 120).

2. La lecture du fichier «TEXTE», la recherche dans chaque ligne des mots de l'Index et la constitution des listes de pages (lignes 170 à 390). On remarquera qu'après avoir détecté dans une ligne un mot faisant partie

de l'index, on vérifie qu'il est *précédé et suivi* (sauf s'il s'agit d'une racine) *d'un séparateur* (lignes 290 à 320). Après chaque lecture, on a encadré la ligne de texte par un séparateur (ligne 200).

3. Le tri des mots de l'Index, le résultat se trouvant dans le tableau TRI (fonction AIDX, ligne 410).

4. L'édition de la liste suivant l'ordre contenu dans TRI (lignes 430 à 510). On n'imprime, ni les mots de longueur nulle (ceux qui n'ont pas été entrés), ni les listes de longueur nulle (celles correspondant à des mots non trouvés dans le texte). Ligne 490, on n'imprime pas la première position de PA$ car elle contient une virgule inutile.

2. Génération d'un Kwic-Index

Un KWIC-INDEX (KWIC = KeyWord In Context) est une liste d'ouvrages triée et présentée en séquence sur des mots - clés appartenant au titre de l'ouvrage. Une telle présentation permet de retrouver aisément un ouvrage dont on ignore le titre exact.

Le fichier « TITRES » des titres et des ouvrages est composé d'enregistrements de 80 caractères comportant :

— de 1 à 9 : le numéro de référence de l'ouvrage en bibliothèque
— de 10 à 80 ; les 71 caractères du titre de l'ouvrage.

Le traitement comprend trois étapes :

a) On met en évidence les mots-clés constituant un titre et on crée, par mot-clé détecté, un enregistrement ayant le format suivant :

— de 1 à 9 : le numéro de référence
— de 12 à 51 : les 40 caractères précédant le mot-clé
— de 54 à 103 : le mot-clé et la suite du titre.

Au cours de ce traitement, on élimine des mots-clés n'ayant pas de signification (et, ou, est, or, donc, etc...). Ces mots sont contenus dans une «liste noire» qui est sous la forme d'une zone alphabétique de 250 caractères.

b) Le fichier « KWIC » ainsi constitué est trié sur les positions 54 à 103 (mot-clé et la suite).

c) Le fichier trié «KWICTRI» est listé pour donner le KWIC-INDEX

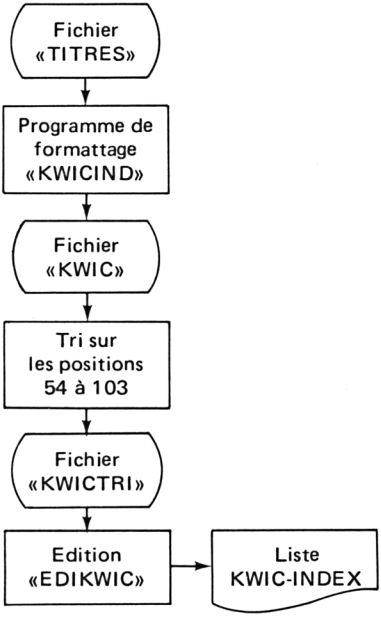


Fig. 7.1. — Organigramme de la constitution d'un KWIC - INDEX.

Par exemple, l'enregistrement du fichier «TITRES» suivant :

1 Référence de l'ouvrage 9	10 Titre de l'ouvrage (71 caractères) 80
C X 2 3 4 2 7 1 0	L A M O R T D U P E T I T C H E V A L

donnera lieu à écriture de trois enregistrements :

1 Référence 9	Blancs 12	40 caractères précédant le mot-clé 51	Blancs 54	Mot-clé et suite du titre (50 caractères) 103
C X 2 3 4 2 7 1 0		L A		M O R T D U P E T I T C H E V A L
C X 2 3 4 2 7 1 0		L A M O R T D U		P E T I T C H E V A L
C X 2 3 4 2 7 1 0		L A M O R T D U P E T I T		C H E V A L

Après tri sur le mot-clé et la suite, un extrait du fichier trié se présentera ainsi :

1 Référence 9	12	51	54 Mot-clé et suite du titre (argument de tri) 103
CX2342710	LA MORT DU PETIT		CHEVAL
AZ4492075		LE	CHEVAL D'ORGUEIL
TB4991042	ON ACHEVE BIEN	LES	CHEVAUX
CB4242097		LES	CHEVAUX DU SOLEIL

On supposera que les mots de chaque titre peuvent être séparés les uns des autres soit par un blanc, soit par un signe de ponctuation (,;.:'). Un mot clé aura une longueur maximale de 50 caractères.

La façon dont cet exercice a été programmé appelle quelques commentaires :

a) nous avons tout d'abord supprimé tous les blancs, sauf un, à droite de la zone titre (instruction 270), le blanc restant servant à repérer facilement la fin de cette zone.

b) nous avons ensuite placé 40 blancs à gauche de la zone «titre» de façon à faciliter le transfert de la partie précédent le mot clé (280).

c) nous avons stocké cette zone dans une zone de réserve et remplacé tous les caractères séparateurs (,.;:) par un seul : le «blanc» (instructions 340 à 360).

d) nous recherchons ensuite les mots clés dans la zone «titre» ainsi modifiée et nous faisons les mouvements *à partir de la zone de réserve* de façon à garder le titre d'origine.

```
00010 !   E702 : EDITION D'UN KWIC-INDEX
00020 !
00030                                                   ! Ouverture des fichiers
00040     OPEN £1: "NAME=TITRES",INPUT                  ! 1.Fic.Titres en entrée
00050     OPEN £2: "NAME=KWIC",OUTPUT                   ! 2.Fic. KWIC  en sortie
00060                                                   ! DEF.CHAINES CARACTERES
00070     DIM MC$*50                                    ! - mot-clé
00080     DIM TI$*112                                   ! - titre de l'ouvrage
00090     DIM TR$*112                                   ! - titre ( reserve )
00100     DIM P40$*40                                   ! - 40 caractéres
00110     DIM S50$*50                                   ! - mot-clé et la suite
00120     DIM LN$*250                                   ! - liste noire
00130     DIM CS$*6                                     ! - liste car.séparateurs
00140     DIM RF$*9                                     ! - réference ouvrage
00150                                                   ! INITIALISATION
00160     LN$="ET OU NI MAIS OR CAR DONC SI ALORS"      ! - de la liste noire
00170     LN$=LN$&"JE TU IL ELLE NOUS VOUS ILS ELLES" !
00180     LN$=LN$&"MON TON SON MA TA SA ME TE SE NOTRE VOTRE LEUR"
00190     LN$=LN$&"QUI QUE QUOI QU DONT DU DES DE LE LA LES Y A"
00200     CS$=",.:;' "                                  ! - liste car.séparateurs
00210 !
00220 !   lecture fic.TITRES et ecriture fic.KWIC
00230 !
00240     READ £1,USING 250: RF$,TI$ EOF 480            ! lecture fichier titres
00250     FORM C 9,C 40
00260     TI$=RTRM$(TI$)&" "                            ! suppr.bl.droite sauf 1
00270     TI$=RPT$(" ",40)&TI$                          ! insertion 40 b.à gauche
00280     TR$=TI$                                       ! mise en reserve de TI$
00290     L=LEN(TI$)                                    ! stockage longueur titre
00300                                                   !
00310     F=40                                          !
00320     D=F+1                                         ! recherche des mots
00330     FOR I=1 TO 6                                  !
00340         TI$=SREP$(TI$,1,CS$(I:I)," ")             ! remplacement separateur
00350     NEXT I                                        ! par des blancs dans TI$
00360                                                   !
00370     F=POS(TI$," ",D)                              ! rech.1 blanc dans TI$
00380     IF F=0 THEN STOP 1                            ! arret anormal
00390                                                   !
00400                                                   !
00410     IF POS(LN$,TI$(D:F-1),1)>0 THEN 470           ! mot-clé ds liste noire?
00420                                                   ! mvts->sortie à part.TR$
00430     P40$=TR$(D-40:D-1)                            !  - 40 car.préc.mot-clé
00440     S50$=TR$(D:90)                                !  - mot-clé et la suite
00450     WRITE £2,USING 460: RF$,P40$,S50$             ! ecriture fichier KWIC
00460     FORM C 9,X 2,C 40,X 2,C 50
00470     IF F=L THEN 230 ELSE 320                      ! recherche terminée ?
00480 END                                               ! ** FIN DU PROGRAMME **
```

3. Recherche documentaire par mots-clés

Une secrétaire tient à jour un fichier reflétant tous les documents émis et reçus par son secrétariat. Ce fichier comprend les données suivantes :

pos. 1 à 5 : le code de classement du document (6 chiffres)
pos. 6 à 11 : la date d'émission ou de réception du document sous la formle AAMMJJ (821208 par exemple).

pos. 12 à 31 : le nom de l'émetteur du document
pos. 32 à 51 : le nom du destinataire principal
pos. 51 à 113 : les mots-clés caractérisant le texte, séparés par des virgules.

Il est demandé d'écrire un programme permettant de retrouver les codes de classement des documents répondant à 5 critères :

— date comprise entre deux dates limites
— le ou les noms des émetteurs
— le ou les noms des destinataires
— deux listes de mots-clés : à l'intérieur de chaque liste, les mots sont liés par des conditions ''OU'' et les deux listes par la condition ''ET''.

Exemple : l'opérateur entre :

DATES LIMITES ? : 12/03/80, 30/08/81
EMETTEURS ? : JEAN,HECTOR,PIERRE
DESTINATAIRES ? : ROGER,JEAN
LISTE MOTS 1 ? : SYSTEME, ORDINATEUR
LISTE MOTS 2 ? : PARIS,MARSEILLE,ABIDJAN,PORT-GENTIL

Le programme devra rechercher et imprimer les enregistrements des documents :

— datés entre le 12/03/80 et le 30/08/81
— émis par JEAN ou par HECTOR ou par PIERRE
— destinés à ROGER ou à JEAN
— comprenant les mots-clés ''SYSTEME'' *ou* ''ORDINATEUR'' *et* les mots-clés ''PARIS'' *ou* ''MARSEILLE'' *ou* ''ABIDJAN'' *ou* ''PORT-GENTIL''.

Pour chaque document répondant aux critères indiqués, le programme imprimera le code de classement, la date, l'émetteur et le destinataire du document.

Si les critères DESTINATAIRE ET EMETTEUR sont par exemple, omis cela signifiera qu'il ne faut pas tenir compte, ni des destinataires, ni des émetteurs dans le processus de recherche. On considèrera que tous les mots-clés sont en majuscules et au singulier.

Chaque critère peut être omis sauf la date.

```
00010 ! E703 : RECHERCHE DOCUMENTAIRE PAR MOTS-CLES
00020 !
00030   OPTION BASE 1                                ! origine des indices en 1
00040   DIM EM$*20,DE$*20,MC$*64                     ! déf.zones du fichier
00050   DIM LE$*64,LD$*64,L1$*64,L2$*64              ! déf.listes entrées terminal
00060   DIM ML1$(10)*20,ML2$(10)*20                  ! tableaux des mots-clés listes
00070                                                ! ENTREE DES LISTES
00080   INPUT "Entrer les dates limites :": D$,F$
00090   LINPUT "Entrer liste des émetteurs:": LE$
00100   LINPUT "Entrer liste des destinataires :": LD$
00110   LINPUT "Entrer liste mots-clés N°1 :": L1$
00120   LINPUT "Entrer liste mots-clés N°2 :": L2$
00130                                                !
00140   D=VAL(D$(7:8)&D$(4:5)&D$(1:2))               ! convers.date début en AAMMJJ
00150   F=VAL(F$(7:8)&F$(4:5)&F$(1:2))               ! convers.date fin en AAMMJJ
00160   IF LE$="" THEN LE=1                          ! indicateur liste non entrée
00170   IF LD$="" THEN LD=1                          !
00180   IF L1$="" THEN L1=1                          !
00190   IF L2$="" THEN L2=1                          !
00200   V$=","                                       ! une "," après chaque liste
00210   LE$=LE$&V$: LD$=LD$&V$: L1$=L1$&V$: L2$=L2$&V$
00220                                                !
00230   IF L1=1 THEN 300                             ! STOCKAGE MOTS-CLES-) TABLEAU)
00240   PD=1                                         ! initialisation recherche
00250   PF=POS(L1$,V$,PD)                            ! recherche position virgule
00260   IF PF=0 THEN 310                             ! recherche terminée ?
00270   I=I+1                                        ! +1 -) indice
00280   ML1$(I)=L1$(PD:PF)                           ! Mot-clé -) tableau
00290   PD=PF+1: GOTO 250                            ! recherche mot suivant
00300   IF L2=1 THEN 370
00310   PD=1                                         ! initialisation recherche
00320   PF=POS(L2$,V$,PD)                            ! recherche position virgule
00330   IF PF=0 THEN 370                             ! recherche terminée ?
00340   J=J+1                                        ! +1 -) indice
00350   ML2$(J)=L2$(PD:PF)                           ! Mot-clé -) tableau
00360   PD=PF+1: GOTO 320                            ! recherche mot suivant
00370   OPEN £1: "NAME=DOCUMENT",INPUT               ! ouverture fichier document
00380                                                ! lecture d'un document
00390   READ £1,USING 400: CC,DD,EM$,DE$,MC$ EOF 640
00400   FORM ZD 5,ZD 6,C 20,C 20,C 63
00410   IF DD<D OR DD>F THEN 390                     ! test sur les dates
00420   IF LE=1 THEN 440                             ! test sur émétteur ?
00430   IF POS(LE$,EM$&V$,1)=0 THEN 390              ! émetteur dans la liste ?
00440   IF LD=1 THEN 460                             ! test sur destinataire ?
00450   IF POS(LD$,DD$&V$,1)=0 THEN 390              ! destinataire dans la liste
00460   IF L1=1 THEN 540                             ! test sur liste 1 ?
00470   MC$=MC$&V$                                   ! "," après liste mots-clés
00480                                                !
00490   FOR N=1 TO I                                 ! boucle recherche mot-clés
00500                                                ! du tableau 1 dans document
00510       IF POS(MC$,ML1$(N),1)>0 THEN 540         ! mot-clé trouvé ?
00520   NEXT N                                       !
00530   GOTO 390                                     ! si non trouvé -) lecture
00540   IF L2=1 THEN 620                             ! test sur liste 2 ?
00550                                                !
00560   FOR N=1 TO J                                 ! boucle recherche mot-clés
00570                                                ! du tableau 2 dans document
00580       IF POS(MC$,ML2$(N),1)>0 THEN 620         ! mot-clé trouvé ?
00590   NEXT N                                       !
00600   GOTO 390                                     ! si non trouvé -) lecture
00610                                                !
00620   PRINT £255: CC,DD,EM$,DE$                    ! IMPRESSION DOCUMENT TROUVE
00630   GOTO 390                                     ! -) lecture
00640 STOP
```

Ce programme comprend trois parties .

1. L'entrée par INPUT et LINPUT des critères de recherche (lignes 80-120).

2. La mise en forme de ces critères de façon à simplifier la recherche des documents (lignes 140-360). Cela consiste à :

— convertir les dates JJ/MM/AA en numérique AAMMJJ (lignes 140 et 150).

— positionner des indicateurs pour signaler au programme que tel ou tel critère n'a pas été entré (lignes 160-190).

— détecter les mots-clés des listes L1$ et L2$ et les placer, un par un, dans les tableaux ML1$ et ML2$ (lignes 210-360).

3. La lecture des documents et la recherche correspondant aux critères (lignes 390-620). Cette recherche se fait essentiellement par la fonction POS en tenant compte des indicateurs positionnés en 2 (lignes 420, 440, 460 et 540).

4. Liste à la demande d'un fichier en format fixe

Les services administratifs d'un constructeur d'ordinateurs gèrent un fichier « SYSTEMES » comprenant un enregistrement par système installé chez les clients de ce constructeur. Le « Dictionnaire de Données » correspondant à ce fichier est le suivant :

Nom abrégé de la donnée	*Description de la donnée*	*Positions dans le fichier*
CLI	nom du client	3-12
SYST	type de système	13-16
AGE	code de l'agence responsable	17-19
MAT	matricule de l'ingénieur respons.	20-24
AN	année d'installation	25-26
ACTI	code activité client	27-28
CPOS	code postal du lieu d'installation	29-34
VILLE	ville d'installation	35-54

Ce dictionnaire de données sera chargé dans le programme par DATA.

On demande de concevoir une application permettant au service Marketing de lister ce fichier suivant une certaine séquence et après avoir

sélecté les enregistrements comportant certaines valeurs pour certaines données.

Le processus de traitement se fera en trois étapes : (voir fig. 7.2).

1. SÉLECTION : l'opérateur entre les listes des valeurs pour les données à sélecter et la liste des données à faire figurer dans l'argument de tri-Ce programme crée un fichier intermédiaire en sortie « SYSTSELE » comportant :

de 1 à 54 : l'argument de tri
de 55 à 108 : l'enregistrement à trier

2. TRI : il s'agit d'un programme utilitaire standard qui trie systématiquement les 54 premières positions du fichier intermédiaire créé en 1.

3. LISTE : il s'agit également d'une liste standard des données en positions 55-108 du fichier intermédiaire trié.

On demande d'écrire le programme de SELECTION. Ce programme devra :

— sélecter les enregistrements et mouvementer chaque enregistrement sélecté dans le fichier en sortie

— constituer l'argument de tri et le placer également dans l'enregistrement en sortie.

Voici un exemple de ce qui peut être entré par un utilisateur pour ce programme :

```
TRI ? : AN,CPOS,AGE
SELECTION ? : AN = 78,79,80
SELECTION ? : SYST = 5422,8952
```

Ces entrées signifieront que le programme devra sélecter les enregistrements concernant les systèmes de type 5422 ou 8952 installés en 1978, 1979 ou 1980 et que l'argument de tri sera constitué de l'année, du code postal et du code de l'agence.

Voici le programme correspondant :

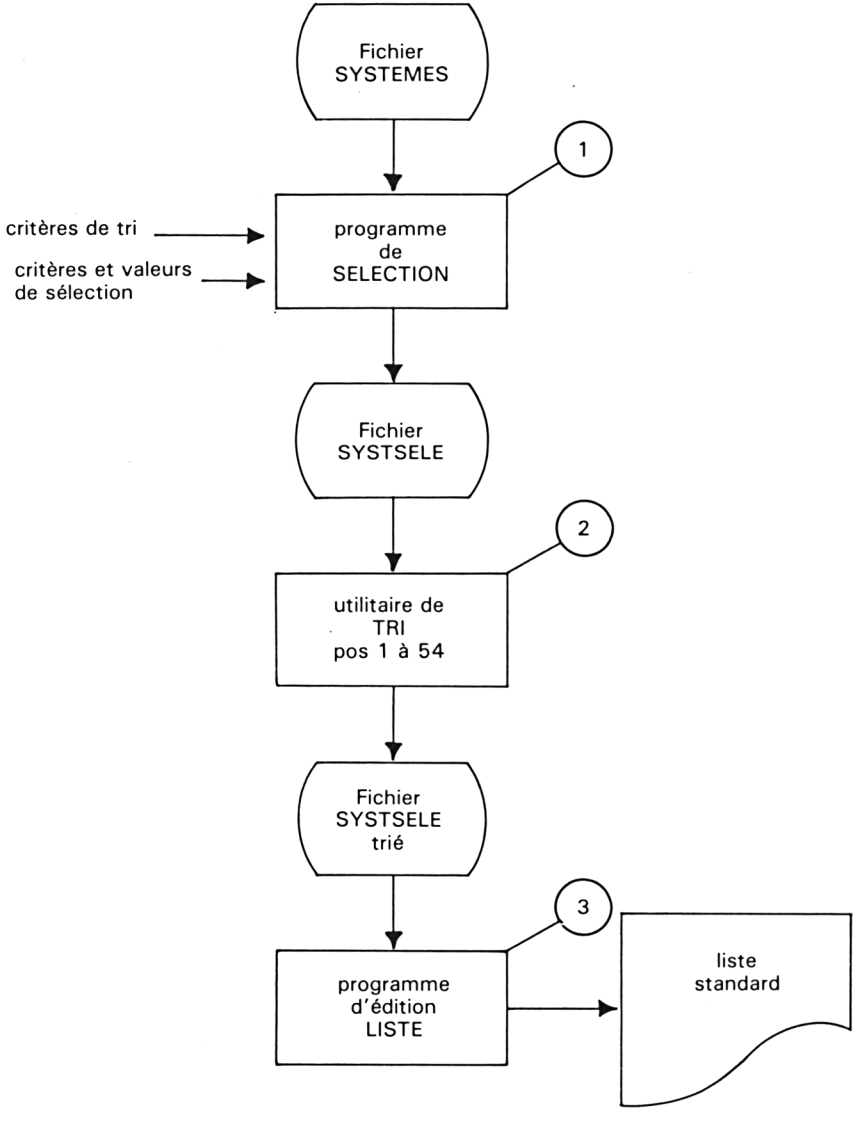


Fig. 7.2 - Liste à la demande d'un fichier en format fixe

```
00010 ! E704 : LISTE A LA DEMANDE D'UN FICHIER EN FORMAT FIXE
00020 !
00030   OPTION BASE 1                                ! origine des indices en 1
00040   DIM E$*52,ARG$*52                            ! déf.enreg.fichier & arg.tri
00050                                                ! DEFINITION DICTIONNAIRE
00060   DIM NOM$(8)*5,DEB(8),FIN(8)                  ! tabl.dictionnaire de données
00070   DIM SEL(8),TRI(8),L$(8)*64                   ! tabl.ind.sélect/tri & listes
00080                                                ! CHARGEMENT DICTIONNAIRE
00090   DATA CLI,SYST,AGE,MAT,AN,ACTI,CPOS,VILLE
00100   READ MAT NOM$                                ! - chargt des noms de zones
00110   DATA 3,13,17,20,25,27,29,35
00120   READ MAT DEB                                 ! - chargt posit.début de zone
00130   DATA 12,16,19,24,26,28,34,54
00140   READ MAT FIN                                 ! - chargt posit.fin de zone
00150                                                ! ENTREE DES PARAMETRES
00160   DIM ZE$*64                                   ! def.zone d'entrée au clavier
00170   LINPUT "TRI ?": ZE$                          ! entrée spécif. de tri
00180   ZE$=RTRM$(LTRM$(ZE$))&","                    ! suppression blancs
00190   D=1
00200   F=POS(ZE$,",",D)                             ! détection virgule
00210   IF F=0 THEN 280
00220   NOZ$=ZE$(D:F-1)                              ! extraction nom de zone
00230   I=SRCH(NOM$,NOZ$)                            ! recher.nom ds dictionnaire
00240   IF I>-1 THEN 260                             ! nom trouvé ?
00250   PRINT "Nom de zone invalide": GOTO 170       ! message erreur -> entrée
00260   K=K+1: TRI(I)=K                              ! positionnement indic.de tri
00270   D=F+1: GOTO 200                              ! -> recherche nom suivant
00280   LINPUT "SELECTION ? ": ZE$                   ! entrée sélection
00290   IF CMDKEY=7 THEN 400                         ! entrée terminée ?
00300   ZE$=RTRM$(LTRM$(ZE$))&","                    ! suppress.blancs et ajout ","
00310   P=POS(ZE$,"=",1)                             ! détection signe =
00320   NOZ$=ZE$(1:P-1)                              ! extraction nom de zones
00330   I=SRCH(NOM$,NOZ$)                            ! recher.nom ds dictionnaire
00340   IF I>-1 THEN 360                             ! nom trouvé ?
00350   GOTO 250                                     ! message d'erreur & -> entrée
00360   SEL(I)=1                                     ! posit.indic.de séléction
00370   L$(I)=ZE$(P+1:64)                            ! stockage liste
00380   GOTO 280                                     ! -> trait.séléction suivante
00390                                                !
00400   MAT TRI=AIDX(TRI)                            ! TRI DES INDICATEURS DE TRI
00410                                                !
00420                                                ! LECTURE,SELECTION & ECRITURE
00430   OPEN £1: "NAME=SYSTEMES",INPUT               ! ouverture fichier systèmes
00440   OPEN £2: "NAME=SYSTSELE",OUTPUT              !
00450   READ £1,USING 460: E$ EOF 620                ! lecture enreg.à sélecter
00460   FORM C 54
00470   FOR I=1 TO 8                                 ! BOUCLE DE SELECTION
00480       IF SEL(I)=0 THEN 550                     ! donnée à tester ?
00490       D$=E$(DEB(I):FIN(I))                     ! extraction valeur à tester
00500       IF POS(L$(I),D$,1)=0 THEN 450            ! VALEUR HORS LISTE -> lecture
00510   NEXT I                                       ! suite de la séléction
00520                                                ! cet enreg.est sélécté --->>
00530   ARG$=""                                      ! remise à blanc argument tri
00540   FOR N=1 TO 8                                 ! BOUCLE CONSTIT:ARGUMENT TRI
00550       I=TRI(N)                                 ! indice zone à mettre -> arg.
00560       IF I=0 THEN 600                          ! fin ?
00570       ARG$=ARG$&D$(DEB(I):FIN(I))              ! zone extraite ->argument tri
00580   NEXT I                                       ! suite boucle
00590                                                !
00600   WRITE £1,USING 610: ARG$,E$                  ! ecriture enreg.sélécté
00610   GOTO 450                                     ! -> lecture
00620   STOP                                         ! FIN DE TRAVAIL
```

Ce programme peut se décomposer en quatre modules :

1. Le chargement du « dictionnaire de données » (lignes 90-140) par DATA et READ.

2. L'entrée des paramètres (lignes 150-380). Au niveau de chaque donnée du dictionnaire à sélecter, le programme *positionne un indicateur* SEL *montrant que cette donnée est à sélecter* et place dans SEL$ la liste des valeurs à sélecter (lignes 370 et 380). De même, *un indicateur* TRI sert à désigner les données *devant figurer dans l'argument de tri :* si TRI = 1, la données sera en première position de l'argument, si TRI = 2, en deuxième, etc...

3. Le tri des indicateurs de tri (instruction AIDX ligne 400). Après cette opération, on retrouve dans TRI les numéros des données du dictionnaire *dans l'ordre où elles doivent figurer dans l'argument de tri.*

4. La lecture du fichier à sélecter et l'écriture du fichier SYSTSELE contenant les enregistrement sélectés (lignes 430 à 610). Ces opérations comprennent notamment :

— la lecture d'un enregistrement (ligne 450).

— la boucle de sélection qui, pour les données à sélecter (IF-ligne 480), va voir si sa valeur fait partie de la liste indiquée (lignes 490 et 500).

— La boucle de constitution de l'argument de tri (lignes 540-580) qui, pour les enregistrements sélectés, concatène les valeurs des zones (ligne 570) dans l'ordre indiqué dans TRI (ligne 550).

— l'écriture de l'enregistrement sélecté (ligne 600) composé de l'argument de tri et des données lues sur «SYSTEMES».

5. Création et mise à jour d'un fichier en format libre

Une société de services en Informatique désire créer un fichier PROSPECT en format libre. Ce fichier comprendra les données suivantes :

Pos.1 : indicateur de suppression
pos.2-11 : nom abrégé du prospect
pos.11-255 : texte décrivant les activités du client

Ce fichier est en séquentiel indéxé, sa clé étant le nom abrégé. Le texte est composé de mots-clés entre parenthèses et des valeurs correspondantes séparées par des virgules. Chaque ligne de texte se termine par une virgule. Les mots-clés autorisés sont les suivants :

PER : personnes contactées chez le prospect
APL : applications traitées
MAT : marques des matériels installés
LOG : logiciels utilisés
ACT : secteurs d'activité du prospect

Voici un exemple d'enregistrement de ce fichier :

Clé : STE SFAM
Texte : (ACT)AUTOMOBILE, (MAT)WANG,IBM,DEC (LOG)COBOL, CICS,BASIC, (APL)COMPTA,STOCKS,CALCUL, (PER) DUVAL,MARTIN,

Tous les mots-clés ne figurent pas obligatoirement dans un enregistrement donné.

On demande d'écrire :

1. un programme de création et de suppression d'enregistrement

2. un programme de mise à jour du fichier permettant l'ajout d'une valeur à une liste caractérisée par un mot clé, le remplacement d'une valeur par une autre et la suppression d'une valeur à l'intérieur d'une liste existante.

Exemple d'ajout :

— mot-clé : PER
— donnée à ajouter : MARC

Exemple de remplacement : — mot-clé : PER
— ancienne valeur : DUVAL
— nouvelle valeur : DUPONT

Exemple de suppression : — mot-clé : PER
— valeur à supprimer : MARTIN

Après exécution de ces trois mises à jour, l'enregistrement contiendra :

Clé : STE SFAM
Texte : (ACT)AUTOMOBILE, (MAT)WANG,IBM,DEC, (LOG)COBOL, CICS, BASIC, (APL)COMPTA,STOCKS,CALCUL, (PER) DUPONT,MARC,

Voici le premier programme suivi d'un exemple d'exécution :

```
00010 ! E705 : CREATION ET SUPPRESSION D'ENREG.DANS UN FICHIER EN FORMAT LIBRE
00020 !
00030   OPEN £1: "NAME=PROSPECT,RANDOM",KEYED      ! ouverture du fichier
00040   DIM LMC$*64,NA$*10,L$*64,E$*244
00050   ON SOFLOW GOTO 240                         ! test enreg. trop long
00060   LMC$="PER/APL/MAT/LOG/ACT"                 ! liste mots.clés valides
00070                                              ! TRAITEMENT DES ENTREES
00080   PRINT NEWPAGE                              ! entrée du type de mouvement
00090   INPUT "Entrer C pour créer ou S pour supprimer ou FIN :": CM$
00100   CAS=POS("CSFIN",CM$,1)                     ! test type mouvement
00110   ON CAS GOTO 130,130,120 NONE 230           ! débrancht fonction type mvt
00120   STOP                                       ! ****** fin de travail *****
00130   LINPUT "Entrer nom abrégé client:": NA$    ! entrée nom du client
00140   GOSUB 250                                  ! accès fichier PROSPECT
00150                                              !
00160   ON CAS GOTO 210,180,190,220                ! TRADUCTION TABLE DECISION
00170                                              !
00180   GOSUB 320: GOTO 90                         ! suppr.enreg.et entrée suivt
00190   GOSUB 370: GOTO 90                         ! saisie & création nouvel
00200                                              ! enregi.puis entrée suivante
00210   PRINT "Enreg.déja existant": GOTO 90       !
00220   PRINT "Enreg.non trouvé   ": GOTO 90       ! messages d'erreur et
00230   PRINT "Mouvement non valide": GOTO 90      ! entrée suivante
00240   PRINT "Enreg.trop long non crée": GOTO 90  !
00250                                              !-----------------------------!
00260                                              ! S/P lecture enreg.fichier  !
00270                                              !-----------------------------!
00280   NA$=RPAD$(NA$,10)                          ! cadrage clé à droite
00290   READ £1,KEY=NA$: NOKEY 310                 ! lect/test si enreg.existant
00300   RETURN                                     ! - cas enreg.trouvé
00310   CAS=CAS+2: RETURN                          ! - cas enreg.non trouvé
00320                                              !-----------------------------!
00330                                              ! S/P suppression d'un enreg!
00340                                              !-----------------------------!
00350   REWRITE £1,USING "FORM 'S'":               ! reécriture avec indicateur
00360   PRINT NA$;"Supprimé ": RETURN              ! de suppression
00370                                              !-----------------------------!
00380                                              ! S/P saisie et création    !
00390                                              !-----------------------------!
00400   E$=""                                      ! RAB enregistrement
00410   INPUT "Mot-clé ou CRE pour créer enreg. :": MC$
00420   IF MC$="CRE" THEN 510                      ! -> création enreg.saisi
00430   IF POS(LMC$,MC$,1)>0 THEN 450              ! test si mot-clé dans liste
00440   PRINT "mot-clé non valide ": GOTO 410      ! message d'erreur
00450   LINPUT "Entrer les valeurs : ": V$         ! saisie liste des valeurs
00460   P=POS(E$,MC$,1)                            ! mot-clé déja dans enreg. ?
00470   IF P=0 THEN 490
00480   E$(P+4:0)=V$&",": GOTO 410                 ! insert.valeur après mot-clé
00490   E$=E$&"("&MC$&")"&V$&","                   ! concaténation avec enreg.
00500   GOTO 410                                   ! -> saisie mot-clé suivant
00510   WRITE £1,USING 520: NA$,E$                 ! écriture enregistrement
00520   FORM POS 2,C 10,C 244
00530   PRINT "Enreg.crée: ";E$: RETURN          ! message et retour
```

```
Entrer C pour créer ou S pour supprimer ou FIN : C
Entrer nom abrégé client: STE SFAM
Mot-clé ou CRE pour créer enreg. : APL
Entrer les valeurs : STOCK,PAIE,CALCUL
Mot-clé ou CRE pour créer enreg. : MAT
Entrer les valeurs : IBM/34,DEC PDP-11
Mot-clé ou CRE pour créer enreg. : LOG
Entrer les valeurs : BASIC,GAP II
Mot-clé ou CRE pour créer enreg. : PER
Entrer les valeurs : PIERRE,PAUL,JACK
Mot-clé ou CRE pour créer enreg. : CRE
Enreg.crée:
(APL)STOCK,PAIE,CALCUL,(MAT)IBM/34,DEC PDP-11,(LOG)BASIC,GAP II,(PER)PIERRE,PAUL
,JACK,
Entrer C pour créer ou S pour supprimer ou FIN :
```

Ce programme est un programme de mise à jour classique d'un fichier en organisation séquentielle indéxée. Il est composé d'un *programme principal* (lignes 30-240) et de *trois sous-programmes* pour accéder au fichier (250-310), pour supprimer un enregistrement (350-360), pour saisir les données et créer un nouvel enregistrement (400-530). La liaison entre le programme principal et les sous-programmes est assurée par l'instruction 160 *qui ne fait que traduire la table de décision suivante :*

Code mouvement = ?	C	S	C	S
Enreg. trouvé ?	oui	oui	non	non
mess. "déjà existant"	X			
mess. "non trouvé"				X
création enreg.			X	
suppression enreg.		X		
lecture code suivant	X		X	X

traduit par CAS =	1	2	3	4
GOTO	200	180	190	220

La saisie et la création de l'enregistrement appelle quelques explications complémentaires :

— on entre un mot-clé (ligne 410).

— on saisit ensuite la liste des valeurs (450). Si le mot-clé a déjà été enregistré (460-470), on insère la chaîne des valeurs V$ après le mot existant (insertion ligne 480). Sinon, on concatène le mot-clé et les valeurs (avec les séparateurs) au bout de E$. Il ne faut pas, évidemment, oublier de remettre à zéro E$ avant chaque nouvelle saisie d'un enregistrement (ligne 400).

Voici le second programme suivi d'un exemple d'exécution :

```
00010 ! E706 : MODIFICATION D'ENREG.DANS UN FICHIER EN FORMAT LIBRE
00020 !
00030   OPEN #1: "NAME=PROSPECT,RANDOM",KEYED      ! ouverture du fichier
00040   DIM LMC$*64,NA$*10,L$*64,E$*244            ! def.zones caractères
00050   ON SOFLOW GOTO 410                         ! test enreg. trop long
00060   LMC$="(PER)(APL)(MAT)(LOG)(ACT)"           ! liste mots.clés valides
00070                                              ! TRAITEMENT DES ENTREES
00080   PRINT NEWPAGE                              ! - entrée nom du prospect
00090   INPUT "Entrer nom du prospect à modifier ou FIN :": NA$
00100   IF NA$="FIN" THEN STOP                     ! test fin de travail ->arret
00110   GOSUB 450                                  ! accès au fichier prospect
00120   ON CAS GOTO 140,350                        ! débrancht non trouvé/trouvé
00130                                              ! entrée type de mouvement
00140   PRINT "entrer A si ajout,S si suppression,M si modification"
00150   INPUT "ou MAJ pour écrire enreg.mis à jour :": CM$
00160   CAS=POS("ASMMAJ",CM$,1)                    ! test et débranchement
00170   ON CAS GOTO 180,240,290,420 NONE 360       ! en fonction type mouvement
00180                                              ! TRAITEMENT DES AJOUTS
00190 INPUT "Mot-clé & valeur à ajouter?": MC$,V$
00200   GOSUB 540                                  ! analyse enreg.à modifier
00210   ON CR GOTO 370,380,220,230                 ! debrancht selon code retour
00220   E$=E$&MC$&V$: GOTO 140                     ! mot clé non trouvé-> concat
00230   E$(C:D)=V$: GOTO 140                       ! mot clé trouvé ->insertion
00240                                              ! TRAITEMENT DES SUPPRESSIONS
00250 INPUT "Mot-clé & valeur à supprimer ?": MC$,V$
00260   GOSUB 540                                  ! analyse enreg.à modifier
00270   ON CR GOTO 280,380,390,400                 ! debrancht selon code retour
00280   E$(D:F)="": GOTO 140                       ! suppression valeur
00290                                              ! TRAITEMENT DS MODIFICATIONS
00300 INPUT "Mot-clé,ancienne valeur,nouvelle valeur ?": MC$,V$,N$
00310   GOSUB 540                                  ! analyse enreg.à modifier
00320   ON CR GOTO 330,380,390,400                 ! debrancht selon code retour
00330   E$(D:F)=N$&",": GOTO 140                   ! rempl.ancienne par nouvelle
00340                                              ! MESSAGES D'ERREUR
00350   PRINT "Enreg.non au fichier": GOTO 90      !
00360   PRINT "Type mouvt non valide": GOTO 140    !
00370   PRINT "valeur déja enregistrée": GOTO 140  !
00380   PRINT "mot-clé non valide": GOTO 140       !
00390   PRINT "mot-clé non trouvé": GOTO 140       !
00400   PRINT "valeur non trouvée": GOTO 140       !
00410   PRINT "enreg.trop long:fin trait": GOTO 140
00420                                              !
00430   REWRITE #1,USING 480: S$,E$
00440   PRINT "Enreg.mis à jour ";E$: GOTO 90      ! message & retour trait.svt
00450                                              ! S/P ACCES FICHIER PROSPECT
00460   NA$=RPAD$(NA$,10)                          ! cadrage clé à gauche
00470   READ #1,USING 480,KEY=NA$: S$,E$ NOKEY 530! lecture sur clé et test
00480   FORM C 1,POS 12,C 244
00490   IF S$="S" THEN 530                         ! test si enreg.supprimé
00500   E$=RTRM$(E$): L=LEN(E$)                    ! supp.blanc gauche E$
00510   PRINT E$                                   ! affichage enreg.à l'écran
00520   CAS=1: RETURN                              ! enreg.trouvé et retour
00530   CAS=2: RETURN                              ! enreg. non trouvé et retour
00540                                              ! S/P ANALYSE ENREGISTREMENT
00550   MC$="("&MC$&")": V$=V$&",": LV=LEN(V$)     ! car.séparateurs -> MC$ & V$
00560   IF POS(LMC$,MC$,1)=0 THEN 620              ! test si mot-clé valide
00570   B=POS(E$,MC$,1): IF B=0 THEN 630           ! test si mot-clé dans enreg.
00580   C=POS(E$&"(",",(",B+5)                     ! pos.déb.mot-clé suivt/fin
00590   D=POS(E$,V$,B): IF D=0 OR D>C THEN 640     ! test si valeur dans enreg.
00600   F=D+LV-1                                   ! cal.pos.fin valeur ds enreg
00610   CR=1: RETURN                               ! mot-clé et valeur trouvés
00620   CR=2: RETURN                               ! mot-clé non trouvé ds liste
00630   CR=3: RETURN                               ! mot-clé non trouvé ds enreg
00640   CR=4: RETURN                               ! valeur non trouvée ds enreg
```

```
Entrer nom du prospect à modifier ou FIN : STE SFAM
(APL)STOCK,PAIE,CALCUL,COMPTA.CLIENTS,(MAT)IBM/38,DEC PDP-11,(LOG)BASIC,GAP II,(
PER)PIERRE,PAUL,JACK,
entrer A si ajout,S si suppression,M si modification
ou MAJ pour écrire enreg.mis à jour : A
Mot-clé & valeur à ajouter? LOG,COBOL
entrer A si ajout,S si suppression,M si modification
ou MAJ pour écrire enreg.mis à jour : M
Mot-clé,ancienne valeur,nouvelle valeur ? PER,PAUL,JULES
entrer A si ajout,S si suppression,M si modification
ou MAJ pour écrire enreg.mis à jour : S
Mot-clé & valeur à supprimer ? APL,PAIE
entrer A si ajout,S si suppression,M si modification
ou MAJ pour écrire enreg.mis à jour : M
Mot-clé,ancienne valeur,nouvelle valeur ? PER,JACQUES,HENRI
valeur non trouvée
entrer A si ajout,S si suppression,M si modification
ou MAJ pour écrire enreg.mis à jour : MAJ
Enreg.mis à jour
(APL)STOCK,CALCUL,COMPTA.CLIENTS,(MAT)IBM/38,DEC PDP-11,(LOG)BASIC,GAP II,COBOL,
(PER)PIERRE,JULES,JACK,
Entrer nom du prospect à modifier ou FIN :
```

Ce programme ressemble au précédent. Il comporte un programme principal (lignes 30 à 450) et deux sous-programmes d'accès au fichier (lignes 460 à 530) et d'analyse de l'enregistrement (lignes 550 à 630). Ce dernier sous-programme détermine :

— la validité du mot-clé par rapport à la liste (560)

— l'existence et la position du mot-clé dans l'enregistrement (570)

— l'existence et les position de début et de fin de la valeur entrée (590). On notera, à ce propos, que la valeur doit *appartenir à la liste concernant le mot-clé entré* et non à une autre liste située « plus loin » dans le même texte.

Ce sous-programme renvoie au programme principal *un « code retour »* CR indiquant le résultat de ces traitements.
En fonction de ce code, ce programme principal se débranche, soit vers des traitements, soit vers des sorties de messages d'erreur, par des GOTO calculés (lignes 210, 270 et 320).
Le lecteur notera que ce programme ne supprime par les mots-clés, même s'ils ne sont suivis d'aucune valeur (par suite d'une suppression).

6. Édition en clair d'un fichier en format libre

La codification des données représente une charge importante lors de la constitution et de la mise à jour des fichiers. Les instructions et fonctions alphabétiques du BASIC permettent, dans certains cas, de s'affranchir des contraintes liées à la codification en employant des abréviations et des codes « mnémoniques ». Voici une application de ce type de codification.

Un fichier de véhicules d'occasion « OCCASE » se compose d'enregistrements ainsi constitués :

— de 1 à 9 : numéro minéralogique du véhicule
— de 10 à 64 : description abrégée du véhicule

La description abrégée est une chaîne de caractères composée de mots qui sont :

— soit des abréviations dont la signification figure dans une table
— soit des libellés complets.

Les mots sont séparés les uns des autres par des virgules, un point terminant l'enregistrement.

Ainsi l'enregistrement :

456XS36 CP, MER, 250SE, 1979, BA, TO, RC, 34000 KM

signifiera :

Coupé Mercédès 250SE 1979 Boîte Automatique Toit ouvrant Radiocassettes 34000KM.

La table des abréviations est constituée par un tableau de 6 chaînes de caractères. A l'intérieur de chaque chaîne se trouvent un certain nombre de postes : chaque poste comprend l'abréviation, suivi du signe « = » et de la traduction de l'abréviation. Il est séparé du poste précédent par un « / ».

Exemple :

chaîne 1 :
CI = CITROEN/PE = PEUGEOT/RE = RENAULT/MER = MERCEDES/etc...
chaîne 2 :
DI = Diesel/B5 = Boîte 5 vitesses/BA = Boîte Automatique/etc...

un «/» termine chaque chaîne.

L'édition se fera sur des lignes de 32 caractères maximum, la première ligne étant composée du numéro d'immatriculation. La coupure de ligne aura lieu en fin de mot, c'est-à-dire que le dernier caractère d'une ligne devra être soit un blanc, soit le dernier caractère d'un mot. On supposera que la description éditée fait 255 caractères au plus.

```
00010 ! E707 : EDITION D'UN FICHIER EN CLAIR
00020                                                   ! Ouverture des fichiers
00030     OPEN £1: "NAME=OCCASE",INPUT                  ! 1.Véhicules d'occasion
00040     OPEN £2: "PRINTER",OUTPUT,DISPLAY             ! 2.Imprimante
00050                                                   ! Définition zones alpha
00060     DIM NM$*9,DA$*55,MOT$*31,MOR$*32              ! -N°,des.abrégée & mot
00070     DIM TA$(5)*255                                ! -tab.traduction abrév.
00080     DIM DC$*255,LI$*32                            ! -descr.clair,lig.impr.
00090                                                   ! Init.table traduction
00100  TA$(0)="RE=RENAULT/PE=PEUGEOT/CI=CITROEN/TA=TALBOT/MA=MATRA/AL=ALPINE/"
00110  TA$(1)="ME=MERCEDES/OP=OPEL/FO=FORD/BM=B.M.W./VW=VOLKSWAGEN/TO=TOYOTA/"
00120  TA$(2)="CA=cabriolet/HT=cabriolet Hard-Top/C2=coupé 2 p./C3=coupé 3 p./"
00130  TA$(3)="B3=break 3 p./B5=break 5 p./TO=toit ouvrant/CC=Camping-Car/"
00140  TA$(4)="IN=injection/DI=diesel/TU=turbo/BA=boite auto/5V=boite 5 vit./"
00150  TA$(5)="RA=radio/RC=radio-cassettes/AR=att.remorque/PM=premiére main/"
00160 !
00170 !  LECTURE ET EDITION EN CLAIR DU FICHIER V.O.
00180 !
00190     READ £1,USING 200: NM$,DA$ EOF 620            ! lecture/test fin fich.
00200     FORM C 9,C 55
00210     DC$=""                                        ! Chaine nulle -> DC$
00220     PRINT £2,USING 230: NM$                       ! impression N°Immatric.
00230     FORM SKIP 2,C 10
00240     DA$=SREP$(DA$,1,".",",")                      ! remp.point par virgule
00250     DA$=RTRM$(DA$)                                ! suppr.blancs à droite
00260     L=LEN(DA$)                                    ! calcul longueur de DA$
00270     D=1                                           ! EXTRACTION DES MOTS
00280     F=POS(DA$,",",D)                              ! - recherche séparateur
00290     MOT$=DA$(D:F-1)                               ! - extraction du mot
00300     GOSUB 490                                     ! - appel S/P traduction
00310     DC$=DC$&MOT$&" "                              ! - mot à la fin de DC$
00320     IF F=L THEN 350                               ! - dernier mot traité ?
00330     D=F+1: GOTO 280                               ! - trait.mot suivant
00340                                                   ! IMPRESSION DES LIGNES
00350     IF LEN(DC$)<33 THEN 400                       ! - dernière ligne ?
00360     IF DC$(33:33)=" " THEN 430                    ! - coupure si fin mot ?
00370     FOR I=32 TO 1 STEP -1                         ! - recherche mot préced
00380         IF DC$(I:I)=" " THEN 440                  !
00390     NEXT I
00400     PRINT £2,USING 410: DC$                       ! - Impress.dernier mot
00410     FORM X 10,C 32
00420     GOTO 190                                      ! - Lecture enreg.suivt
00430     I=32                                          !
00440     LI$=DC$(1:I)                                  ! - extrac.ligne impress
00450     PRINT £2,USING 410: LI$                       ! - impression
00460     DC$(1:I)=""                                   ! - suppress. I premiers
00470     DC$=LTRM$(DC$)                                !  car. & cadrage gauche
00480     GOTO 350                                      ! - trait.ligne suivante
00490 !
00500 !       S/P de traduction des abréviations
00510 !
00520         MOR$=MOT$&"="                             ! init mot à rechercher
00530         FOR K=0 TO 5                              ! boucle de recherche
00540             Q=POS(TA$(K),MOR$,1)                  ! - recherche mot
00550             IF Q>0 THEN 580                       ! - mot trouvé ?
00560         NEXT K                                    !
00570         RETURN                                    ! - retour si non trouvé
00580         E=POS(TA$(K),"=",Q)                       ! - recherche signe "="
00590         G=POS(TA$(K),"/",E)                       ! - - - - - - - - - "/"
00600         MOT$=TA$(K)(E+1:G-1)                      ! - extract. mot traduit
00610         RETURN                                    !
00620 END                                               ! ** FIN DU PROGRAMME **
```

Ce programme se compose d'une boucle de traitement (lignes 190 à 480) qui :

1. lit un enregistrement (ligne 190)

2. le traduit en clair (lignes 240 à 310) en faisant appel à un sous-programme de traduction (lignes 500 à 610).

3. édite les lignes de texte (lignes 350 à 470).

Le lecteur remarquera la façon dont sont faites les coupures en fin de ligne d'impression : si la ligne ne se termine pas par un mot (test ligne 350), on explore « à reculons » la ligne jusqu'à la rencontre d'un blanc (boucle 370-390). Ce blanc trouvé, on sort la ligne d'impression (lignes 440 à 450) et on recommence l'opération avec la partie restant à traiter de la ligne de texte (lignes 460 et 470).

7. Fusion texte fichier en format fixe

On dispose :

— d'un texte représenté par un fichier « TEXTE » (une ligne de 64 car. par enreg.) de 50 enregistrements maximum.

— d'un fichier « CLIENT » dont la longueur d'enregistrement est 255.

En début du programme de fusion, l'utilisateur entre pour chaque zone concernée par la fusion :

— sa position de début dans l'enregistrement « fichier »
— sa position de fin dans l'enregistrement « fichier »
— le numéro de la ligne de texte où elle doit figurer
— sa position dans cette ligne
— un code d'édition

Ce code d'édition peut prendre trois valeurs :

''1'' : pas d'édition
''2'' : édition d'un montant en milliers de francs, francs et centimes avec symbole monétaire ''F''.
''3'' : édition d'une date AAMMJJ sous la forme JJ/MM/AA

On ne fera pas les coupures de mots en fin de ligne. La sortie des textes fusionnés se fera sur imprimante.

Le programme correspondant se trouve ci-après :

```
00010 ! E708 : FUSION TEXTE-FICHIER EN FORMAT FIXE
00020 !
00030   OPTION BASE 1                                  ! origine des indices en 1
00040   PP$=PIC$("F")                                  ! symbole monétaire = F
00050   DIM T$(50)*64                                  ! tableau contenant le TEXTE
00060   OPEN £1: "NAME=TEXTE",INPUT                    ! ouverture fichier TEXTE
00070   FOR I=1 TO 50                                  ! Boucle charg.tableau TEXTE
00080       READ £1,USING 90: T$(I) EOF 110            ! lecture ligne ds tableau
00090       FORM C 64
00100   NEXT I                                         ! fin boucle
00110 !
00120   DIM D(20),F(20),L(20),P(20),CE(20)              ! définition des tableaux
00130   DIM F$*255,Z$*255                              ! contenant début,fin,ligne,
00140                                                  ! position et code d'édition
00150                                                  !
00160   FOR I=1 TO 20                                  ! BOUCLE SAISIE DEFINITION
00170                                                  ! DES ZONES A MOUVEMENTER
00180       PRINT "Entrer pour chaque zone à éditer :"
00190       PRINT "Début,fin,N°ligne,position,code édition"
00200       PRINT " (0,0,0,0,0 si entrée terminée )"
00210       INPUT D(I),F(I),L(I),P(I),CE(I)
00220       IF P(I)=0 THEN 240
00230   NEXT I
00240 !
00250   OPEN £2: "NAME=CLIENT",INPUT                   ! ouverture FICHIER CLIENT
00260   READ £2,USING 270: F$ EOF 490                  ! lecture un enregistrement
00270   FORM C 255
00280   FOR I=1 TO 20                                  ! BOUCLE DE MOUVEMENT
00290                                                  ! DU FICHIER VERS LE TEXTE
00300       IF D(I)=0 THEN 430                         ! si terminé : ->impression
00310       Z$=F$(D(I):F(I))                           ! extraction zone
00320       ON CE(I) GOTO 390,330,360                  ! traitement code édition
00330       Z=VAL(Z$)                                  ! conversion en numérique
00340       Z$=CNVRT$("PIC($$$ $$£,££)",Z)             ! édition sv masque montant
00350       GOTO 390
00360       Z=VAL(Z$)                                  ! conversion en numérique
00370       Z$=CNVRT$("PIC(££/££/££)",Z)               ! édition svt masque date
00380                                                  !
00390       DZ=P(I): FZ=DZ+LEN(Z$)-1                   ! calcul pos.début et fin
00400       T$(L(I))(DZ:FZ)=Z$                         ! mouvement de la zone vers
00410                                                  ! la ligne de texte
00420   NEXT I
00430 !                                                ! IMPRESSION TEXTE FUSIONNE
00440   PRINT £255: NEWPAGE                            ! - saut de page
00450   PRINT £255,USING 460: MAT T$                   ! - impression tab.texte
00460   FORM C 64
00470   GOTO 260                                       ! retour lecture enreg.suiv
00480 !
00490   STOP
```

Ce programme est divisé en trois modules :

1. le chargement du texte dans le tableau de chaînes T$ (lignes 60 à 100).

2. la saisie et le stockage des caractéristiques des zones à mouvementer du fichier vers le texte (lignes 160 à 230).

3. la lecture du fichier « CLIENT », l'édition par masque et le mouvement des zones de chaque enregistrement vers les lignes de texte et

l'impression du texte lui-même (lignes 250 à 470). On remarquera que chaque mouvement *« écrase »* une partie de la ligne de texte concernée et qu'il est donc nécessaire de prévoir un nombre de blancs égaux à l'amplitude de la zone à mouvementer.

Signalons pour finir qu'il serait, en pratique, utile de prévoir un *contrôle de validité* des paramètres décrivant les zones à mouvementer. Nous ne l'avons pas fait pour éviter d'alourdir le programme.

8. Fusion Texte-Fichier en format libre

Le fichier « Texte » est le même que dans l'exercice précédent mais les endroits où les données doivent figurer sont indiqués par un mot-clé encadré par deux astérisques. On supposera que la place nécessaire à l'insertion des variables a été réservée au niveau de chaque ligne.

Le fichier « client » est en format libre. Chaque enregistrement de 255 caractères est composé de mots-clés également encadrés par deux astérisques suivi d'une valeur.

Si un mot-clé figurant dans le texte n'a pas de valeur correspondante dans le fichier, la ligne du texte ne sera pas imprimée.

Voici le programme correspondant :

```
00010 ! E709 : FUSION TEXTE-FICHIER EN FORMAT LIBRE
00020 !
00030   OPTION BASE 1                                   ! origine des indices en 1
00040   PP$=PIC$("F")                                   ! symbole monétaire = F
00050   DIM T$(50)*64,L$*64                             ! tableau contenant le TEXTE
00060   DIM M$(20)                                      ! def.tabl.mots-clés texte
00070   DIM L(20)                                       ! déf.tableau N°ligne texte
00080   OPEN £1: "NAME=TEXTE",INPUT                     ! ouverture fichier TEXTE
00090 !
00100   FOR I=1 TO 50                                   ! Boucle charg.tableau TEXTE
00110       READ £1,USING 120: L$ EOF 240               ! lecture ligne ds tableau
00120       FORM C 64
00130       T$(I)=RTRM$(L$)                             ! stockage ligne -> tableau
00140       PD=POS(L$,"*",1)                            ! détéction * debut mot-clé
00150       IF PD=0 THEN 210                            ! recherche terminée
00160       PF=POS(L$,"*",PD+1)                         ! détéction * fin mot-clé
00170       K=K+1                                       ! + 1 indice du tableau
00180       M$(K)=L$(PD:PF)                             ! mot-clé -> tableau
00190       L(K)=I                                      ! N°ligne de texte->tableau
00200       GOTO 210                                    !
00210   NEXT I                                          ! fin boucle
00220 !
00230   DIM S$(50)*64,F$*255                            ! def.tableau texte à sortir
00240   OPEN £2: "NAME=CLIENT",INPUT                    ! ouverture FICHIER CLIENT
00250   READ £2,USING 260: F$ EOF 520                   ! lecture un enregistrement
00260   FORM C 255
00270   F$=RTRM$(F$)&"*"                                ! * après l'enreg.du fichier
00280   MAT S$=T$                                       ! tabl.TEXTE -> tabl.SORTIE
00290   FOR N=1 TO K                                    ! BOUCLE RECHERCHE MOTS-CLES
00300                                                   ! ET REMPLACEMENT PAR VALEUR
00310       MC$=M$(N)                                   ! extraction mot-clé tableau
00320       PM=POS(F$,MC$,1)                            ! détect.mot-clé ds fichier
00330       LI=L(N)                                     ! N°Ligne à modifier
00340       IF PM=0 THEN 410                            ! non trouvé ->pas de sortie
00350       DV=PM+LEN(MC$)                              ! calcul pos.début valeur
00360       FV=POS(F$,"*",DV)-1                         ! calcul pos.fin   valeur
00370                                                   !
00380       S$(LI)=SREP$(S$(LI),1,MC$,F$(DV:FV))       ! REMPLAC.MOT-CLE PAR VALEUR
00390                                                   ! DANS LE TABLEAU A SORTIR
00400       GOTO 420                                    ! -> mot-clé suivant
00410       S$(LI)=""                                   ! ligne à ne pas imprimer
00420   NEXT N
00430 !                                                 ! IMPRESSION TEXTE FUSIONNE
00440   PRINT £255: NEWPAGE                             ! - saut de page
00450   FOR I=1 TO 50                                   ! boucle impr.tableau SORTIE
00460       IF S$(I)="" THEN 490                        ! test si ligne long.nulle
00470       PRINT £255,USING 480: S$(I)                 ! - impression ligne texte
00480       FORM C 64
00490   NEXT I
00500   GOTO 250                                        ! retour lecture enreg.suiv
00510 !
00520   STOP
```

Par rapport au précédent, ce programme-ci ne comprend que deux parties principales :

1. Le chargement du texte dans le tableau T$ (lignes 80 à 210). A chaque fois que le programme lit une ligne de texte, il en profite pour y rechercher les mots-clés (lignes 140-160). Chaque mot-clé est chargé

dans un poste du tableau MŞ (ligne 180) et le N° de la ligne de texte où il figure dans le poste correspondant du tableau L (ligne 190).

2. La lecture du fichier, le mouvement des zones de l'enregistrement vers le tableau texte et l'impression du texte modifié (lignes 240 à 500).

On remarquera que, à la différence de l'exercice précédent, les mouvements de zones ne se font pas par *« écrasement »* mais par *remplacement* du mot-clé par sa valeur extraite de l'enregistrement fichier. La longueur de cette valeur pouvant varier d'un enregistrement du fichier à l'autre, il est donc obligatoire de copier le tableau « texte » dans un tableau « sortie » qui, lui, sera modifié par les remplacements et imprimé. Cette copie se fait ligne 280 après chaque lecture du fichier « CLIENT ».

LEXIQUE

FRANÇAIS - ANGLAIS

Affecter (une valeur) ..... assign (to)
Afficher ..... display (to)
Aligner ..... justify (to)
Apostrophe ..... apostrophe
Astérisque ..... asterisk
Attribuer (une valeur) ..... set (to)
Barre de fraction (/) ..... slash
Bibliothèque ..... library
Blanc (caractère —) ..... blank
Boucle ..... loop
Centrer ..... center (to)
Chaîne (de caractères) ..... string (character —)
Chaîne (de long. nulle) ..... null string
Charger (un fichier) ..... load (to)
Classer ..... collate (to)
Clavier ..... keyboard
Clé (d'enregistrement) ..... key (record —)
Clé en double ..... duplicate key
Colonne ..... column
Commentaire ..... remark
Compléter ..... padd (to)
Concaténation ..... concatenation
Convertir ..... convert (to)

Créer ..... create (to)
Décaler, décalage ..... shift (to)
Dépanner (un programme) ..... debug (to)
Dépassement de capacité ..... overflow
Deux points ..... colon
Direct (accès -) ..... random (- access)
Ecran (cathodique) ..... screen (cathode ray -)
Enlever ..... remove (to)
Enregistrement ..... record
Erreur (de programme) ..... bug
Espacement ..... space
et «commercial» (&) ..... ampersand
Etiquette ..... label
Facteur ..... operand
Fichier ..... file
Fusionner ..... merge (to)
Guillemet ..... quotation mark
Image ..... picture
Imbriquer ..... nest (to)
Imprimante ..... printer
Imprimé ..... form (paper -)
Imprimer ..... print (to)
Insérer ..... insert (to)
Insertion ..... insertion
Interrompre ..... break (to)
Jeu (de caractères) ..... set (character -)
Ligne ..... line
Ligne (d'un tableau) ..... row (of an array)
Longueur ..... lenght
Machine à écrire ..... typewritter
Majuscule (lettre -) ..... uppercase (- letter)
Membre (d'une bibli.) ..... member (library -)
Message de service ..... prompt
Mettre à jour ..... update (to)
Mettre en fonction ..... enable (to)
Mettre hors fonction ..... disable (to)
Minuscule (lettre -) ..... lowercase (- letter)
Octet ..... byte
Parenthèse ..... parenthesis
Piste (sur un disque) ..... track

Point ..... period
Point-virgule ..... semicolon
Poste de travail ..... workstation
Recherche, rechercher ..... search (to)
Remplacement ..... replacement
Remplacer ..... replace (to)
Répéter ..... repete (to)
Restaurer (une valeur) ..... reset (to)
Retour chariot ..... carriage return
Saut, sauter (des lignes) ..... skip (to)
Sauter (de page) ..... jump (to)
Sauvegarder ..... save (to)
Séparateur (caractère -) ..... delimiter
Séquentiel ..... sequential
Séquentielle-indexée ..... indexed sequential
Souligner ..... underscore (to)
Sous-chaîne ..... substring
Sous-programme ..... subroutine
Suppression ..... deletion
Supprimer ..... delete (to)
Symbole monétaire ..... currency symbol
Tableau ..... array
Tabulateur ..... tab key
Terminal à écran ..... display station
Tiret ..... hyphen
Touche (sur un clavier) ..... key (keyboard -)
Touche de fonction ..... function key
Trier ..... sort (to)
Virgule ..... comma
Zone (de données) ..... field

ANGLAIS - FRANÇAIS

Ampersand ..... et «commercial» (&)
Apostrophe ..... apostrophe
Array ..... tableau
Assign (to) ..... affecter (une valeur)
Asterisk ..... astérisque
Blank ..... blanc (caractère -)
Break (to) ..... interrompre
Bug ..... erreur (de programme)
Byte ..... octet
Carriage return ..... retour chariot
Center ..... centrer
Collate (To) ..... classer
Colon ..... deux points
Column ..... colonne
Comma ..... virgule
Concatenation ..... concaténation
Convert (To) ..... convertir
Create (To) ..... créer
Currency symbol ..... symbole monétaire
Debut (To) ..... dépanner (un programme)
Delete (To) ..... supprimer
Deletion ..... suppression
Delimiter ..... séparateur (caractère -)
Disable (To) ..... mettre hors fonction
Display (To) ..... afficher
Display station ..... terminal a écran
Duplicate key ..... clé en double
Enable (To) ..... mettre en fonction
Field ..... zone (de données)
File ..... fichier
Forme (Paper -) ..... imprimé
Function key ..... touche de fonction
Hyphen ..... tiret
Indexed sequential ..... séquentielle-indexée

Insert (To) ..... insérer
Insertion ..... insertion
Jump (To) ..... sauter (de page)
Justify (To) ..... aligner
Key (Keyboard -) ..... touche (sur un clavier)
Key (Record -) ..... clé (d'enregistrement)
Keyboard ..... clavier
Label ..... étiquette
Lenght ..... longueur
Library ..... bibliothèque
Line ..... ligne
Load (To) ..... charger (un fichier)
Loop ..... boucle
Lowercase (- Letter) ..... minuscule (lettre -)
Member (Library) ..... membre (d'une bibliothèque)
Merge (To) ..... fusionner
Nest (To) ..... imbriquer
Null string ..... chaîne de longueur nulle
Operand ..... facteur
Overflow ..... dépassement de capacité
Padd (To) ..... compléter
Parenthesis ..... parenthèse
Period ..... point
Picture ..... image
Print (To) ..... imprimer
Printer ..... imprimante
Prompt ..... message de service
Quotation mark ..... guillemet
Random (- Access) ..... direct (acces -)
Record ..... enregistrement
Remark ..... commentaire
Remove (To) ..... enlever
Repete (To) ..... répéter
Replace (To) ..... remplacer
Replacement ..... remplacement
Reset (To) ..... restaurer (une valeur)
Row (of an array) ..... ligne (d'un tableau)
Save (To) ..... sauvegarder
Screen (cathode ray -) ..... écran cathodique
Search (To) ..... recherche, rechercher

Semicolon ..... point-virgule
Sequential ..... séquentiel
Set (Character -) ..... jeu (de caractères)
Set (To) ..... attribuer (une valeur)
Shift (To) ..... décaler, décalage
Skip (To) ..... saut, sauter (des lignes)
Slash ..... barre de fraction (/)
Sort (To) ..... trier
Space ..... espacement
String (Character -) ..... chaîne (de caractère)
Subroutine ..... sous-programme
Substring ..... sous-chaîne
Tab key ..... tabulateur
Track ..... piste (sur un disque)
Typewritter ..... machine à écrire
Underscore (To) ..... souligner
Update (To) ..... mettre à jour
Uppercase (- letter) ..... majuscule (lettre -)
Workstation ..... poste de travail

INDEX

L

M

O

P

R

S

T

U

V

Imprimerie de la Manutention - Mayenne
Dépôt légal : septembre 1982
N° d'éditeur : 3825

ASTIER - **La conduite de l'Apple II** (coll. Micro-ordinateurs)
Tome 1 : Le Basic de l'Apple II - 128 p., 1982
Tome 2 : Le système graphique et l'assembleur de l'Apple II - 120 p., 1982

LADEVIE - **Votre gestion avec Basic sur micro-ordinateur** - 152 p., 1982 (coll. Micro-ordinateurs)

LAMOITIER - **Le langage Basic et la nouvelle norme** - 232 p., 1982

LONCHAMP - **Basic.** Construction méthodique des programmes - 136 p., 1981 (coll. Pratique de l'Informatique)

PELLIER - **La conduite du TRS 80** - 120 p., 1982 (coll. Micro-ordinateurs)

PRADELS - **L'informatisation des entreprises.** Qualité. Productivité. Rentabilité des projets - 136 p., 1981

STROHMEIER - **Cobol 74. Approche systématique illustrée d'exemples** - 168., 1982

WEGNEZ - **Choix informatique et sécurité** - 176 p., 1980

EYROLLES

8760

EYROLLES

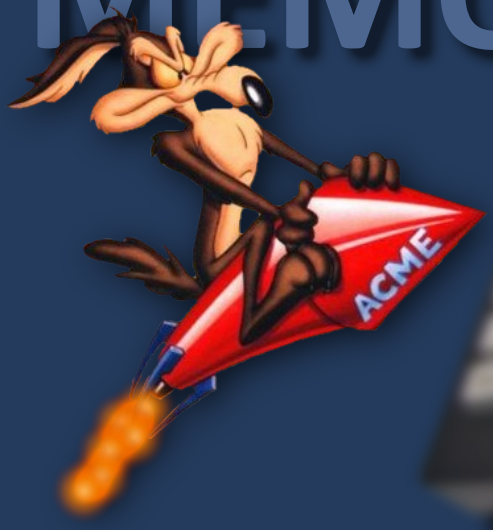
BASIC et le traitement de textes

g.quaneaux

Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/