# LOGICIELS EXELVISION

**GUPPY :** Le drame du petit poisson gobeur de graines, perdu dans le labyrinthe et poursuivi par de terribles pieuvres. Classique ? Malheureux, attendez donc la fin pour voir !

**WIZORD :** Un démon terrifiant, des Muzords sataniques, de grotesques Bozords, un champ de mines, des Kazords fous, un château imprenable... vous ne saurez plus où donner du canon et de la manette.

**VIRUS :** Eh oui ! Vous avez attrapé des virus ! Votre cas est désespéré. Avec vos malheureuses seringues, vous aurez le plus grand mal à éliminer les virus et les Super-Virus. En fait, vous êtes pratiquement condamné !

**TENNIS :** Le **vrai** tennis avec l'arbitre qui annonce les points et les fautes, le public qui crie, les ramasseurs de balles, le bruit des balles, les revers, les montées au filet, les lobs, les amorties, les aces... Un graphisme extraordinaire, l'ambiance sonore... le vrai tennis quoi !

**CAPITAINE MENKAR :** Si vous n'avez jamais conduit un vrai vaisseau spatial auparavant, vous aurez peut-être un peu de mal au début avec les cadrans, les témoins et les jauges de votre vaisseau. Sans compter les pirates extra-terrestres, les mines, les quantums d'Hyper-Energie... Heureusement, Else la navigatrice et Xor le robot vous aideront de la voix pour traverser l'Hyper-Espace sans dommages... enfin presque.

**IMAGIX :** Un jeu éducatif graphique qui passionnera les enfants et les adultes. Grâce à un nombre quasiment illimité de combinaisons, de formes et de couleurs, IMAGIX permet de réaliser des images stylisées et de merveilleux tableaux animés.

exelvision
Immeuble Ophira, Place Joseph Bermond 06560 VALBONNE
Tél. (93) 74.41.40 - Télex 461 355 F

Fabriqué en France

**L'informatique facile à vivre**

Logiciel pour TO7, TO8

# EXELBASIC

exelvision

# LE GUIDE DE L'EXL 100

Christian TAVERNIER

# SOMMAIRE

Comment lire ce manuel 7

**MODE D'EMPLOI DE L'EXL 100**

Déballage et préparation 11
Installation 12
Mise sous tension 14
Ce qu'il ne faut pas faire 15
Suivez le guide 16
En cas de problème 17
Connexion d'un magnétophone à cassettes 18

**INITIATION A LA PROGRAMMATION**

**Chapitre 1 :** Partons à l'aventure 23
- Vous ne risquez rien 23
- Qu'est-ce qu'un ordinateur? 23
- Programme et langage Basic 24
- Le Basic 25

**Chapitre 2 :** Dialogue avec EXL 100 26
- Le clavier 26
- L'écran 31
- Corrigez vos fautes 31
- Encore un mot 33

**Chapitre 3 :** Notre première instruction 33
- Mode direct et mode programmé 33
- L'instruction PRINT 34
- Les quatre opérations 35
- Un PRINT facultatif 36
- Nombres entiers et décimaux 36
- Quelques définitions 38

**Chapitre 4 :** Des chiffres et des lettres 38
- Définissons des variables 38
- Les commandes CALL CLEANUP et NEW 40
- LET, une instruction qui n'existe plus 41
- Un signe égal qui ne signifie pas toujours égal 41
- Utilisons les variables 42
- Des variables qui varient 43
- Les chaînes de caractères 43
- Les tableaux 45

**Chapitre 5 :** Notre premier programme 47
- L'écriture d'un programme 43
- Quelques règles à respecter 48
- Listing et commande LIST 49
- Numérotation de lignes automatique 50
- Corrigeons encore nos fautes 51
- Changez les numéros des lignes 52
- L'exécution de notre programme 53

**Chapitre 6 :** Jonglez avec les chaînes de caractères 54
- Une curieuse addition 54
- La fonction LEN 55
- Ne mélangeons pas les variables 56
- La fonction VAL 56
- La fonction NUMERIC 57
- La fonction STR$ 57
- Un peu de gymnastique 58
- Codes et caractères 58
- Des caractères cachés 59
- Coupons les chaînes en morceaux 60
- Répétez une chaîne à l'infini 60
- Si vous cherchez votre chaîne 61

**Chapitre 7 :** Discutez avec votre EXL 100 61
- Les secrets de l'instruction PRINT 61
- Affichez des textes où vous voulez 64
- Lorsque l'EXL 100 vous interroge 65
- Des INPUT spéciaux 67

**Chapitre 8 :** Les boucles 69
- Des lignes qui se répètent toutes seules 69
- Imbriquons des boucles 72
- Des boucles de taille variable 73

**Chapitre 9 :** Branchements et sous-programme 74
- Perturbons le déroulement d'un programme 74
- Comment interrompre un programme 75
- Les sous-programmes 76
- Les sauts calculés 78
- L'instruction CALL et les sous-programmes particuliers 79

**Chapitre 10 :** Des sons et des couleurs 80
- Commentez vos programmes 80
- Vous allez en voir de toutes les couleurs 81
- Perdons du temps 82
- Arrêtons-nous 82
- Un ordinateur qui a de la voix 82



**Chapitre 11 :** Lorsque l'ordinateur prend des décisions 83
- Les opérateurs de comparaison 83
- L'instruction IF THEN ELSE 85
- Encore des opérateurs 86

**Chapitre 12 :** Les fonctions scientifiques 88
- La constante PI 88
- Fonctions trigonométriques 89
- Fonctions trigonométriques inverses 90
- Les fonctions logarithme et exponentielle 90
- Les fonctions scientifiques diverses 90
- Hasard et informatique 91

**Chapitre 13 :** L'EXL 100 dessinateur 93
- Le mode haute résolution 93
- Allumez des points 95
- Une seule instruction pour une droite 96
- A cheval entre graphique et alphanumérique 97
- Définissez vos propres caractères 99

**Chapitre 14 :** Utilisation d'un magnétophone 100
- Un magnétophone pour quoi faire? 100
- La commande SAVE 101
- La commande LOAD 101
- Si ça ne marche pas 102
- Quelques conseils 104
- On peut aussi mettre des données sur cassettes 105

**Chapitre 15 :** Pour programmer comme un « pro » 108
- READ, DATA et RESTORE 108
- Exploitez vos erreurs 110
- Les points d'arrêt 111
- Pour utiliser les points d'arrêt à autre chose 112

**Chapitre 16 :** Fabriquez vos propres instructions 113
- Les instructions qui manquent 113
- Définissez vos propres instructions 114
- Définissez vos propres touches de fonctions 116
- Le langage machine 117
- Lisez dans la mémoire 118
- Écrivez dans la mémoire 119

**Chapitre 17 :** Pour finir 119
- A vous de jouer 119
- Les extensions 120

FICHES DE RÉFÉRENCE 123

ANNEXES

Annexe 1 : Liste des instructions de l'EXL 100 209
Annexe 2 : Signification des messages d'erreur 210
Annexe 3 : Priorités des opérateurs 213
Annexe 4 : Code ASCII et caractères spéciaux 214
Annexe 5 : Grille d'adressage de l'écran alphamérique et graphique 216
Annexe 6 : Chaînes de génération de sons pour CALL SPEECH 217
Annexe 7 : Tableau de conversion décimal - hexadécimal - binaire 223
Annexe 8 : Cartographie mémoire de l'EXL 100 230
Annexe 9 : Brochages des prises de l'EXL 100 231
Annexe 10 : Schéma synoptique de l'EXL 100 233



# COMMENT LIRE CE MANUEL

Le manuel que vous découvrez maintenant a été écrit pour que vous puissiez tirer le meilleur parti de votre micro ordinateur EXL 100 quelles que soient vos connaissances initiales en micro informatique.

Il est divisé en quatre parties :
- Une première partie qui est le mode d'emploi de l'EXL 100. Elle vous explique comment le raccorder, le mettre en marche et vous donne quelques conseils d'utilisation. Que vous connaissiez ou non la micro informatique, vous devez lire cette partie.
- Une deuxième partie intitulée initiation à la programmation qui a été écrite pour vous qui ne connaissez pas ou peu la micro informatique. Tout ce qu'il faut savoir de la programmation vous y est expliqué simplement et progressivement. Si vous connaissez déjà le Basic, vous pouvez survoler rapidement cette partie pour voir les particularités de l'EXL 100.
- Une troisième partie qui comporte les fiches de référence des instructions. C'est cette partie que vous utiliserez lorsque vous connaîtrez bien votre micro ordinateur pour savoir avec précision ce que fait telle ou telle instruction et comment elle s'utilise.
- Une quatrième partie enfin contient les annexes. Vous y trouverez les messages d'erreurs, les codes de caractères, une grille de repérage de l'écran, des tables de conversion et une foule d'informations utiles.

Les fiches de référence et les annexes peuvent être lues et feuilletées dans n'importe quel ordre en fonction de vos besoins; la deuxième partie du manuel doit, par contre, être lue dans l'ordre des chapitres car ceux-ci ont été écrits de façon à respecter une progression logique dans la présentation de l'EXL 100. Tous les mots nouveaux sont imprimés en majuscule lors de leur première apparition afin que vous puissiez plus facilement les reconnaître et les retenir.

Nous vous souhaitons de joyeuses heures de loisirs instructifs avec votre micro ordinateur EXL 100 et vous invitons, sans plus tarder, à lire la première partie de ce manuel.

1ère partie

# MODE D'EMPLOI DE L'EXL 100

# MODE D'EMPLOI DE l'EXL 100

**Déballage et préparation :**

Dans la boîte de votre EXL 100 vous devez trouver le matériel suivant :
- Le micro ordinateur EXL 100 lui-même
- Le clavier sans câble de raccordement
- La "cassette" de ROM Basic qui est un boîtier plat de la taille d'une petite boîte d'allumettes
- Un câble péritélévision
- Un cordon secteur
- Une grille en plastique transparente avec des découpes identiques aux touches du clavier
- Le manuel que vous êtes en train de lire
- Une carte de garantie

Si un seul de ces éléments est absent, signalez le sans plus tarder à votre revendeur.

Pour pouvoir utiliser votre EXL 100, il vous faut en outre le matériel suivant :
- Une pile 9 volts ALCALINE pour le clavier
- Un récepteur de télévision couleur muni d'une prise péritélévision. Cette prise est facilement identifiable ; c'est un gros rectangle avec un angle déformé, muni de 21 contacts et situé en face arrière du téléviseur. De plus, cette prise est obligatoirement présente sur tous les récepteurs vendus en France depuis fin 1980. Si votre récepteur ne comporte pas une telle prise, il faut vous procurer un adaptateur spécial comportant un codeur SECAM et un modulateur UHF. Consultez votre revendeur pour cela.
- Un magnétophone à cassettes ordinaire sera également nécessaire dès que vous aurez commencé à écrire des programmes. Ce peut être un modèle quelconque ; il faut seulement qu'il soit en bon état de fonctionnement et que ses têtes soient propres.

- un câble de raccordement pour le magnétophone muni à une extrémité d'une prise DIN à 3 ou 5 pôles et à l'autre extrémité de prises correspondant à votre magnétophone. Des explications plus détaillées vous sont fournies ci-après à ce sujet.

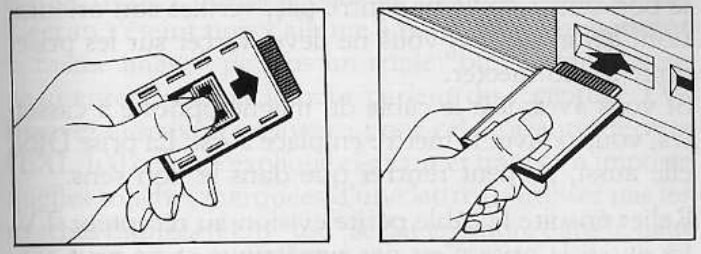
## Installation :

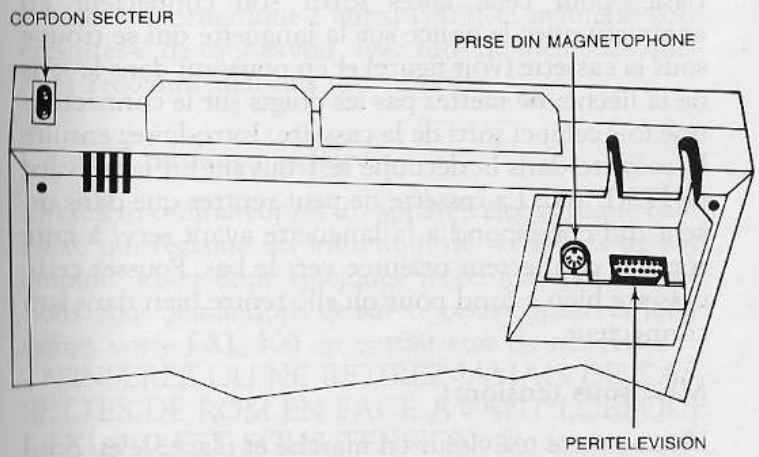
Votre micro ordinateur doit être installé près du récepteur TV auquel il doit être raccordé. Vous pouvez le placer sous le récepteur si celui-ci est un modèle portable (écran de taille inférieure à 36 cm). Si votre récepteur est de taille plus importante, il est préférable de ne pas placer l'EXL 100 en dessous car le poids du téléviseur pourrait déformer le boîtier de l'ordinateur.
L'orientation du boîtier de l'EXL 100 doit être telle que sa face avant soit dirigée approximativement vers l'endroit où vous allez vous installer avec le clavier.
Le clavier utilise une liaison infrarouge (même principe que les télécommandes de télévisions ou de magnétoscopes) qui porte jusqu'à environ 4 mètres de distance dans des conditions d'éclairage normales. Si la lumière ambiante est très forte, cette portée peut être un peu réduite.
Les infra rouges se réfléchissent assez bien sur les objets, ce qui permet de ne pas devoir orienter exactement le clavier vers l'EXL 100 ; par contre : il ne faut pas qu'il y ait de gros obstacles entre l'EXL 100 et son clavier ; les infra rouges ne sont pas des rayons X ; ils ne peuvent pas passer au travers de la matière ! Pour cela, prenez soin de ne pas masquer avec vos doigts ou autre chose la fenêtre translucide située à la verticale des touches oranges du clavier car c'est par là que passe le faisceau infra rouge.

Comme ce clavier n'a pas de fil de liaison, il faut l'alimenter. Il utilise une pile de 9 volts classique. Choisissez une pile de type ALCALINE (Mallory Duracel, UCAR, etc...) qui est le seul type ne risquant pas de couler et offrant une durée de vie suffisante.
Mettez la pile en place dans le compartiment se trouvant sous le clavier. Il s'ouvre avec une pièce de mon-



**MISE EN PLACE DE LA CASSETTE ROM BASIC**






**DISPOSITION DES CONNECTEURS**

naie. La pile ne peut se brancher que dans le bon sens ; ne forcez donc pas sur son connecteur. En refermant son compartiment, veillez à ne pas coincer les fils avec la trappe en plastique car cela pourrait les couper.

Côté EXL 100, mettez en place le cordon secteur (voir figure 1) puis le câble péritélévision. Attention, la prise utilisée n'est pas symétrique et ne peut rentrer que dans le bon sens ; si elle ne rentre pas, vérifiez son orientation. En aucun cas vous ne devez forcer sur les prises pour les connecter.
Si vous avez déjà le câble du magnétophone à cassettes, vous pouvez le mettre en place aussi. La prise DIN, elle aussi, ne peut rentrer que dans le bon sens.

Reliez ensuite le câble péritélévision au récepteur TV. Là aussi, la prise n'est pas symétrique et ne peut rentrer que dans le bon sens sur le poste.

Vous devez ensuite mettre en place la cassette ROM Basic; pour cela, faites sortir son connecteur en appuyant avec le pouce sur la languette qui se trouve sous la cassette (voir figure) et en poussant dans le sens de la flèche; ne mettez pas les doigts sur le connecteur une fois celui-ci sorti de la cassette. Introduisez ensuite la cassette dans la découpe se trouvant en face avant de l'EXL 100; La cassette ne peut rentrer que dans un sens qui correspond à la languette ayant servi à faire sortir le connecteur orientée vers le bas. Poussez cette cassette bien à fond pour qu'elle rentre bien dans son connecteur.

**Mise sous tension :**

Mettez votre téléviseur en marche et placez-le en position péritélévision (appelée position "audiovisuel" sur certains appareils); montez un peu le volume sonore. Mettez en marche l'EXL 100 en appuyant sur le poussoir orange de la face avant. Un triple "bip" se fait entendre dans le haut-parleur du récepteur TV (si vous avez suffisamment monté le volume); deux voyants rouges s'allument aux deux extrémités de la fenêtre récepteur infra rouge et une seconde ou deux après apparaît une image.



L'écran doit être de couleur bleu ciel, avec une bande bleue foncée tout en haut sur laquelle est affiché le mot RADIAN en jaune. Un carré noir est présent sur la dernière ligne en bas à gauche de l'écran.
Réglez le contraste, les couleurs et la luminosité de votre téléviseur à votre goût.
Pour vérifier que tout se passe bien, appuyez sur le poussoir de RESET qui est le poussoir blanc de la face avant. L'écran s'éteint puis s'allume à nouveau en présentant la même image ; de plus un triple "bip" se fait à nouveau entendre dans le haut parleur du récepteur TV. Essayez alors votre clavier ; pour cela, orientez-le vers l'EXL 100 comme expliqué ci-avant et frappez n'importe quelles touches marquées d'une lettre (n'utilisez pas les touches oranges !) Un "toc" se fait entendre dans le haut parleur du récepteur TV à chaque frappe correcte et la lettre frappée apparaît sur la dernière ligne de l'écran.

Vous êtes prêt pour le grand saut dans le monde de la micro informatique ; auparavant, et même si vous êtes pressé de tout savoir, lisez tout de même les quelques recommandations qui suivent.

**Ce qu'il ne faut pas faire :**

Un micro ordinateur est un appareil électronique complexe qui réclame un minimum de soins lors de son emploi. Voici donc quelques recommandations que nous vous demandons de suivre pour conserver longtemps votre EXL 100 en parfait état de marche :
- N'INSEREZ OU NE RETIREZ JAMAIS DE CASSETTES DE ROM EN FACE AVANT LORSQUE L'EXL 100 EST SOUS TENSION. Vous risquez de détruire la cassette et certains circuits internes de l'appareil.
- Lorsque vous avez retiré une cassette de ROM, faites rentrer son connecteur à l'intérieur au moyen de la languette qui se trouve dans la cassette. Ne mettez pas les doigts sur ce connecteur car il est sensible à l'électricité statique.
- Lorsque vous branchez et débranchez les câbles, tirez sur les prises elles-mêmes et non sur les câbles.

- N'exposez pas votre EXL 100 ou son clavier à des températures excessives. En particulier, si vous les transportez, ne les placez jamais dans des endroits très chauds tels que la plage arrière d'une voiture.
- Le clavier ne consomme du courant que lorsqu'on appuie sur les touches ; pour cela, rangez le toujours, touches vers le haut et veillez à ce qu'aucun objet ne puisse appuyer sur les touches car vous déchargeriez ainsi sa pile en quelques heures seulement.

**Suivez le guide :**

Vous avez sans doute remarqué un certain nombre de choses dont nous n'avons pas parlé sur votre EXL 100. En face arrière, tout d'abord, se trouvent deux trappes en plastique qui si vous les enlevez, donnent accès à des connecteurs au fond d'un logement. Ces emplacements sont réservés à des modules d'extension tels que interface imprimante, modem, lecteurs de disquettes, etc ... Ils ne nous intéressent donc pas pour l'instant.
En face avant, la partie gris clair est un tiroir que vous pouvez ouvrir en pressant à fond dessus ( un ressort le fait alors sortir). Ce tiroir permet de ranger deux manettes de jeux ou joysticks fournis en option. Pour le refermer appuyez simplement sur le tiroir jusqu'à ce qu'il se bloque.
Attention, si vous avez posé sur votre EXL 100 un récepteur TV lourd, il se peut que vous n'arriviez pas à ouvrir le tiroir car le poids du poste peut déformer le boîtier de l'ordinateur.
La fente de la face avant où nous avons mis la cassette de ROM Basic peut recevoir des cassettes ROM de programmes pour l'EXL 100 ; votre revendeur est à même de vous présenter ces produits. Pratiquement, et quel que soit le programme, il se présente dans une cassette comme celle que vous venez de mettre en place.
La fenêtre rouge ou apparaissent les deux voyants cache le récepteur infra rouge de l'EXL 100 ; il ne faut pas la masquer car vous ne pourriez plus utiliser votre clavier.



**En cas de problème :**

Si vous prenez les précautions expliquées ci-avant, il est peu probable que vous ayez des problèmes avec votre EXL 100 ; si tel était le cas, voici quelques petites vérifications à faire avant de rapporter l'appareil à votre revendeur.

• Lorsque vous appuyez sur le poussoir orange, rien ne se passe et les voyants de la face avant restent éteints :
- Vérifier s'il y a du courant sur la prise où est branché l'EXL 100 en y mettant un autre appareil par exemple.
- Si oui, vérifiez si le cordon secteur de l'EXL 100 ne s'est pas débranché.
- Si non, vérifiez s'il n'est pas coupé en l'échangeant avec celui d'un autre appareil (c'est un cordon standard très répandu).
- Si aucune de ces vérifications ne donne de résultat, votre appareil est sans doute en panne. Rapportez-le à votre revendeur avec son clavier et sa cassette de ROM Basic.
EN AUCUN CAS, VOUS NE DEVEZ ENVOYER L'APPAREIL DIRECTEMENT A EXELVISION.

• Les voyants s'allument bien lorsque vous mettez en marche mais vous n'avez ni image ni son :
- Vérifiez si le cordon péritélévision ne s'est pas débranché côté EXL 100 et côté télévision.
- Vérifiez si la télévision est bien en position péritélévision (ou audiovisuel).
- Si tout semble correct, votre EXL 100 est sans doute en panne. Rapportez-le alors à votre revendeur comme indiqué ci-dessus.

• Les voyants s'allument mais au lieu de l'image prévue, vous avez un petit papillon de couleur et EXELVISION sur l'écran :
- Vous avez tout simplement oublié de mettre en place une cassette de ROM en face avant ; éteignez l'appareil, mettez en une en place et tout rentrera dans l'ordre.

• Les voyants s'allument, vous avez bien l'image prévue mais rien ne se passe lorsque vous utilisez le clavier :
- Vérifier la pile du clavier ; en cas de doute changez-la.

- Vérifiez l'état de la fenêtre de l'émetteur infra rouge du clavier et de celle du récepteur infra rouge de l'EXL 100.
- Rapprochez EXL 100 et clavier et essayez à nouveau.
- Si aucune de ces opérations ne donne de résultat, rapportez le tout à votre revendeur comme expliqué ci-avant.

Rassurez-vous, ces problèmes sont très rares (sauf l'usure de la pile du clavier !) et il est fort peu probable que vous ayez à faire ces essais.

Un dernier point reste à signaler : les infra rouges utilisés par l'EXL 100 n'ont rien à voir avec ceux des télécommandes des récepteurs TV et ils n'ont en principe aucune influence l'un sur l'autre. Toutefois, s'il arrivait que lorsque vous frappez sur le clavier de votre EXL 100, cela agisse sur la télévision, il faudrait masquer le récepteur infra rouge de celle-ci avec un petit morceau de papier ou de ruban adhésif opaque.

### Connexion d'un magnétophone à cassettes :

Comme nous l'avons dit, n'importe quel magnétophone peut être utilisé avec l'EXL 100, il faut seulement qu'il soit en bon état et que ses têtes soient propres.
Pour le raccorder, il faut un cordon muni d'une prise DIN à 3 ou 5 pôles côté EXL 100 et muni :
- de la même prise DIN à l'autre extrémité si votre magnétophone utilise des prises de ce type.
- de deux prises jacks à l'autre extrémité si votre magnétophone utilise des prises de ce type.

De tels cordons existent dans tous les grands magasins et chez tous les revendeurs radio-TV. Si vous êtes un peu bricoleur, ou si un de vos amis l'est, vous pouvez faire ce câble vous-même, car le brochage de la prise DIN de l'EXL 100 est indiqué en annexe 9 en fin de ce manuel.

Côté magnétophone, vous devez utiliser la prise entrée/sortie "ligne" si votre appareil est équipé de prises DIN et les prises "micro" et "écouteur ou haut parleur extérieur" si votre appareil est équipé de prises jacks.



Sur les cordons DIN - jacks, la prise jack à mettre dans la prise "micro" du magnétophone est repérée en rouge.

En cas de doute, consultez votre revendeur qui sera à même de vous conseiller.

Si vous ne possédez pas de magnétophone à cassettes, nous vous recommandons les appareils : Continental Edison MC 8226, Carrefour MG 1102 ou Radialva RM 2420 qui ont donné toute satisfaction lors de nos essais.

### En avant !

Tout est prêt, votre EXL 100 vous surveille de ses deux yeux rouges, le clavier attend vos ordres alors allez-y, tournez cette page et faites connaissance avec ce monde passionnant qu'est l'informatique ...

2ème partie

# INITIATION A LA PROGRAMMATION

CHAPITRE 1

# Partons à l'aventure

## Vous ne risquez rien

A partir de ce chapitre nous allons découvrir l'EXL 100 mais, malgré tout le soin apporté à la rédaction de ce manuel, certaines choses resteront peut être dans l'ombre. Si tel est le cas ou si vous vous posez des questions du type "qu'est ce qui se passerait si je faisais cela ?" n'hésitez pas à essayer. Une fois que votre ordinateur est branché comme expliqué dans le chapitre précédent, vous ne risquez rien ni lui non plus. Vous pouvez frapper ce que vous voulez pour voir ce qui se passe sans aucune restriction. Si ce que vous frappez est vraiment trop incompréhensible pour l'ordinateur, il se "plantera" comme l'on dit dans le jargon des programmeurs c'est-à-dire qu'il sera perdu et ne saura plus quoi faire. Si tel est le cas, il suffit d'appuyer sur le poussoir RESET qui est le poussoir blanc situé en face avant de votre ordinateur pour que tout rentre dans l'ordre. Ce poussoir met l'EXL 100 dans un état particulier que l'on appelle l'état INITIAL, et ce sera le premier point important à retenir.

Toute action sur le poussoir de RESET (poussoir blanc de la face avant) ramène l'EXL 100 à l'état initial. Tout programme éventuellement contenu en mémoire est perdu après un RESET.

Ceci étant vu, sachez aussi que ce qui suit se comprend beaucoup mieux en le lisant à côté de l'ordinateur et en frappant à son clavier au fur et à mesure les exemples que nous donnons.

## Qu'est-ce qu'un ordinateur ?

Rassurez-vous, nous n'allons pas vous faire un cours

sur l'intérieur de l'EXL 100 ; nous allons simplement vous dire en deux mots ce qu'est un ordinateur et pourquoi il faut connaître un certain nombre de choses pour l'utiliser.

Un ordinateur est une machine électronique capable d'effectuer des opérations très rapidement (dans l'EXL 100 une addition entre deux nombres à deux chiffres prend 5 millionièmes de seconde). Ces opérations peuvent être des opérations arithmétiques telles que addition, soustraction, multiplication ou division, des opérations dites scientifiques, telles que calculs de racines carrées, de sinus et de cosinus d'angles, etc ... mais aussi des opérations plus curieuses travaillant sur des mots du langage courant ; un ordinateur peut ainsi additionner "bonjour" et "comment allez-vous ?".
Tout cela est très bien et permet de comprendre pourquoi les ordinateurs savent faire de très nombreuses choses différentes, malheureusement ils ont un gros défaut : il faut leur avoir indiqué au moins une fois quelles sont les opérations à faire pour réaliser un travail déterminé. Cette indication ne peut être faite que par un être humain et constitue ce que l'on appelle UN PROGRAMME. Lorsqu'un programme a été écrit, un ordinateur saura le lire et réaliser les opérations qu'il contient autant de fois que vous le désirez et ce, à une vitesse impressionnante. Le fait de donner un programme à un ordinateur pour qu'il accomplisse les opérations qui y sont contenues s'appelle faire EXECUTER le programme. On dit aussi lorsque l'on est du métier, faire TOURNER un programme.
Un ordinateur est donc une machine capable d'exécuter une suite d'opérations quelconques contenues dans un programme.

**Programme et langage Basic :**

Un programme, nous l'avons dit, est une suite d'opérations que devra effectuer l'ordinateur, mais pour que la machine comprenne ce que l'on veut qu'elle fasse, il faut écrire ce programme d'une certaine façon et selon des règles bien précises. De plus, comme il existe de très



nombreuses façons d'écrire des opérations, un certain nombre de mots ont été définis une fois pour toutes, et l'ordinateur ne comprend que ceux-ci. Ces mots sont les MOTS CLES ou INSTRUCTIONS du LANGAGE de programmation.

De même qu'il existe de par le monde plusieurs langues différentes, il existe dans le monde des ordinateurs des langages de programmation différents, et chaque langage a un nom particulier, tel que : BASIC, FORTRAN, ALGOL, PASCAL, etc ... Rassurez-vous, il n'est pas utile de connaître tous ces langages pour pouvoir utiliser votre EXL 100 et nous limiterons au basic.
Il vous faut encore savoir une chose très intéressante : ces langages sont indépendants (dans une large mesure) des ordinateurs et de leurs marques ou de leurs pays de fabrication. Ainsi, le Basic que vous utilisez sur votre EXL 100 est le même que vous utiliseriez sur un gros ordinateur IBM venant des USA ou sur un calculateur Siemens fabriqué en Allemagne. Ces langages sont donc universels et une fois que vous en avez appris un sur un appareil, ce que vous avez appris peut servir pour ce même langage utilisé sur n'importe quel autre appareil.

**Le Basic**

Le Basic a été créé en 1964 aux Etats-Unis et était à l'origine un langage pour débutants. Son nom l'indique puisqu'il signifie Beginners All purpose Symbolic Instruction Code ce qui veut dire à peu près : codage d'instructions à usage général pour les débutants.
Depuis cette époque le Basic a bien évolué et c'est maintenant un langage de programmation puissant et souple. Il a cependant su garder sa simplicité d'emploi initial comme nous allons le voir.
Le Basic se compose d'un certain nombre de mots clés standards définis depuis l'origine de ce langage mais comporte aussi quelques mots clés qui peuvent varier d'un ordinateur à un autre en raison des possibilités particulières de l'appareil. Ainsi, sur les micro ordinateurs qui ne travaillent qu'en noir et blanc ne trouverez-vous pas l'équivalent de l'instruction COLOR de l'EXL

100 ; cette instruction sert en effet à définir des couleurs ce qui est parfaitement inutile sur un appareil noir et blanc.

CHAPITRE 2

# Dialoguez avec votre EXL 100

## Le clavier :

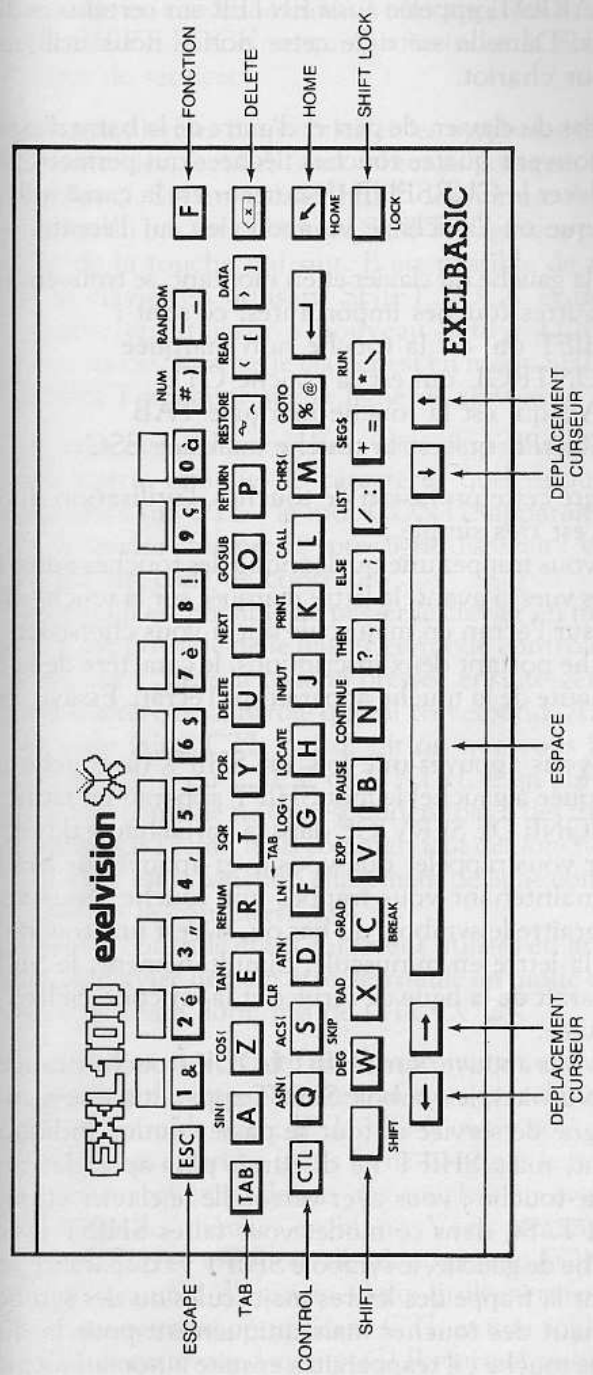
Pour donner des ordres à votre ordinateur vous disposez d'un clavier. Mise à part l'absence de fil pour le relier à votre ordinateur puisqu'il dispose d'une liaison par infra rouges, ce clavier ressemble à un clavier de machine à écrire classique. Seules, quelques touches supplémentaires autorisent des fonctions propres à un ordinateur et qui n'auraient rien à faire sur une machine à écrire, tandis que certaines autres touches propres à une machine à écrire n'existent pas ici.

Examinez la figure ci-jointe qui représente le clavier et vous permet de repérer certaines touches particulières par les noms que nous leur donnerons dans toute la suite de ce manuel. Ces noms ne sont pas choisis au hasard et correspondent à ceux habituellement utilisés en informatique.

Sur la droite du clavier, les quatre touches oranges s'appellent respectivement et de haut en bas :
- FONCTION pour la touche F
- DEL ou DELETE pour la touche repérée par un X dans une flèche
- HOME pour la touche repérée par une flèche oblique
- SHIFT LOCK pour la dernière touche.

A côté de la touche HOME, se trouve une touche plus grande que les autres portant une flèche coudée. Cette touche est très importante : c'est la touche RETOUR





CHARIOT, appelée aussi ENTER sur certains ordinateurs. Dans la suite de cette notice nous utiliserons retour chariot.

En bas du clavier, de part et d'autre de la barre d'espace, se trouvent quatre touches fléchées qui permettent de déplacer le CURSEUR (le curseur est la carré noir qui indique où l'affichage va avoir lieu sur l'écran).

Sur la gauche du clavier et en montant, se trouvent quatre autres touches importantes, ce sont :
- SHIFT qui est la touche non marquée
- CONTROL qui est la touche CTL
- TAB qui est la touche marquée TAB
- ESCAPE qui est la touche marquée ESC.

malgré cette profusion de touches, l'utilisation du clavier est très simple :
- Si vous frappez une quelconque des touches autre que celles vues ci-avant, la lettre marquée sur la touche apparaît sur l'écran en majuscule ou, si vous choisissez une touche portant deux inscriptions, le caractère de la partie haute de la touche apparaît sur l'écran. Essayez pour voir.
- Si vous appuyez une fois sur SHIFT (la touche non marquée à gauche) le mot SHIFT apparaît en jaune sur la LIGNE DE SERVICE dans la partie haute de l'écran pour vous rappeler que vous avez appuyé sur SHIFT. Si, maintenant vous frappez une touche, vous voyez apparaître le symbole du bas ou, si c'est une touche lettre, la lettre en minuscule. Simultanément, le SHIFT disparaît de la ligne de service et la fonction SHIFT est annulée.
- Si vous appuyez sur SHIFT LOCK (touche orange en bas à droite), le symbole SHIFT apparaît à nouveau sur la ligne de service et tout se passe comme indiqué ci-avant, mais SHIFT ne disparaît plus après la frappe d'une touche ; vous avez verrouillé le clavier en mode SHIFT. Si, dans ce mode, vous faites SHIFT avec la touche de gauche, le symbole SHIFT va disparaître autorisant la frappe des lettres majuscules ou des symboles du haut des touches mais uniquement pour la durée d'une touche ; il réapparaîtra ensuite automatiquement.



Pour revenir en mode normal, il faut appuyer à nouveau sur SHIFT LOCK ce qui fait éteindre le SHIFT de la ligne de service.

en résumé, les lettres majuscules et les symboles de la partie haute des touches sont obtenus lorsque l'on frappe celles-ci. Si l'on utilise SHIFT, les minuscules et les symboles du bas sont obtenus. SHIFT n'agit que pour la frappe de la touche qui suit. Il est possible de verrouiller le clavier en utilisant SHIFT LOCK et de le déverrouiller en utilisant à nouveau SHIFT LOCK. Dans tous les cas, lorsque le clavier est en mode SHIFT, le mot SHIFT apparaît sur la ligne de service.

Si vous avez essayé les touches, vous devez avoir une ligne en partie remplie de caractères quelconques ; appuyez alors sur CTL ; le mot CONTL apparaît sur la ligne de service ; frappez E, votre ligne disparaît ! Vous avez utilisé un code de contrôle.
De même qu'il est possible de passer le clavier en mode SHIFT, il est possible de le passer en mode contrôle en appuyant sur CTL ; la touche pressée ensuite génère alors le caractère de contrôle qui lui correspond. Ainsi, lorsque vous faites CTL puis E dit on que vous avez frappé contrôle E ou CTL E ou CNTRL E en abrégé. Ces codes de contrôle ne correspondent pas à des caractères visibles sur l'écran mais commandent des fonctions. Ainsi, CTL E est la fonction effacement de ligne comme vous avez pu le constater.
Les codes de contrôle étant rarement utilisés en grand nombre, le clavier ne peut être verrouillé en mode contrôle et il n'existe donc pas de CTL LOCK

Pour poursuivre notre visite du clavier, mettez en place sur celui-ci la grille transparente que vous avez dû trouver dans la boîte de votre EXL 100. Vous y voyez de nombreux mots étranges (mais ils ne vont pas le rester longtemps) marqués au-dessus des touches en orange et au-dessous des touches en blanc. Vous remarquez aussi que CTL est entourée de blanc et que FONCTION est entourée d'orange. Cette grille permet de savoir ce qui se passe en utilisant CTL et une touche. Ainsi, CTL E correspond au mot CLR marqué en blanc

sous E; CLR signifie clear ce qui veut dire effacer. Nous verrons dans un prochain chapitre le rôle de INS (CTL A), — TAB (CTL T) et BREAK (CTL C).

La touche FONCTION (marquée F à droite du clavier) permet d'avoir accès aux mots inscrits en orange au-dessus des touches. Faites F ; le mot FONCT apparaît sur la ligne de service ; faites alors K et vous voyez apparaître le mot PRINT sur l'écran; simultanément, FONCT disparaît de la ligne de service comme pour SHIFT et CTL.

La touche FONCTION permet de frapper les mots clés ou instructions du Basic en deux opérations seulement (F puis la touche correspondant au mot clé désiré).

Essayez le clavier tant que vous voulez mais ne vous étonnez pas si vous voyez apparaître des messages mystérieux du style Err XX ou XX est un numéro. En faisant vos essais, il peut arriver que vous donniez des ordres incohérents à l'EXL 100 qui vous l'indique alors, mais comme indiqué au chapitre 1 ; vous ne risquez rien !

Remarquez que chaque fois que vous frappez sur une touche, un "bip" sonore est émis. Il vous indique que la touche a bien été prise en compte par l'ordinateur et évite de faire trop de fautes de frappe.

La grande touche ne fait afficher aucun caractère visible mais si vous l'actionnez vous allez voir le curseur se déplacer d'une position. Cette touche sert à afficher un ESPACE ou un BLANC.

Remarquez aussi que si vous maintenez une touche autre que les touches particulières vues ci-avant enfoncée, le caractère qu'elle génère se répète automatiquement tant que vous maintenez le doigt sur la touche.

Cette présentation du clavier étant faite, nous allons peu à peu ne plus vous indiquer exactement quelles touches frapper afin que cela devienne pour vous évident. Lorsque nous écrirons frappez =, il faudra savoir qu'il faut pour cela faire SHIFT puis la touche + =. C'est



un peu pénible au début, mais cela permet de connaître le clavier très vite.

**L'écran :**

L'écran est l'organe principal dont dispose l'EXL 100 pour dialoguer avec vous.

Cet écran est divisé en 24 Lignes de 40 caractères. Si vous avez pianoté sur le clavier vous avez du remarquer que pour l'instant nous n'écrivions que sur la ligne la plus basse de l'écran. Par contre, si l'on frappe plus de 40 caractères (essayez), la ligne pleine remonte d'une position et l'on continue à écrire sur la ligne suivante qui est de nouveau la ligne la plus basse. Ce phénomène peut se reproduire autant de fois que vous le désirez jusqu'à ce que l'écran soit plein.

Le fait que l'écran remonte automatiquement d'une ligne lorsque vous arrivez à la fin de la dernière ligne s'appelle le SCROLLING ou le SCROLL de l'écran.

Pour l'instant, votre ordinateur s'est contenté d'afficher sur l'écran les caractères que vous avez frappés ; ce n'est pas terrible. La raison en est simple, vous ne lui avez pas encore donné d'ordre. En effet, vous pouvez frapper ce que vous voulez sur le clavier, l'ordinateur l'affiche à l'écran sans chercher à comprendre tant que vous n'appuyez pas sur la touche retour chariot qui lui ordonne alors d'interpréter ce que vous venez de frapper.

Appuyez sur RESET puis frappez n'importe quoi et faites retour chariot. Si vous avez vraiment frappé n'importe quoi, un message Err XX où XX est un numéro va apparaître en bas de l'écran vous signalant que l'EXL 100 n'a pas compris ce que vous vouliez.

Notez bien que l'EXL 100 n'interprète ce que vous avez frappé qu'après un retour chariot.

**Corrigez vos fautes :**

Commencez par faire RESET ce qui libèrera votre écran de la ligne incompréhensible que vous venez de frapper.

Les informaticiens ne sont pas des dactylographes, même après de longues années de pratique, et ils font souvent, comme vous sans doute, des fautes de frappe. Votre EXL 100 est capable de corriger un certain nombre de ces fautes dès leur frappe.

A tout instant, pendant que vous êtes en train de frapper une ligne de caractères, vous pouvez actionner les flèches horizontales de déplacement du curseur pour amener celui-ci n'importe où sur la ligne et continuer à frapper à partir de la position ainsi obtenue. Si vous ramenez le curseur en arrière, vous pouvez ainsi frapper par dessus ce que vous venez d'écrire et corriger les fautes que vous avez pu faire.

Si vous avez frappé des caractères en trop comme dans EXXXL 100 par exemple, il suffit de ramener le curseur à gauche du ou des caractères à effacer. Chaque appui sur la touche DEL (orange avec une flèche et un X à droite du clavier) fait effacer un caractère situé à droite du curseur et décale tout le reste de la ligne. Essayez avec EXXXXL 100 de faire effacer les X en trop. Lorsque l'effacement est fini, vous pouvez ramener le curseur où vous voulez avec les flèches et continuer la frappe.

Mais votre EXL 100 peut faire encore plus fort. Supposons que vous ayez oublié un caractère dans ce que vous avez frappé ; soit EL 100 par exemple. Ramenez le curseur sur le L au moyen de la flèche vers la gauche puis faites une commande INS (CTL puis touche A comme indiqué sur la grille transparente) ; vous êtes passé en mode INSERTION. Frappez alors X ; celui-ci apparaît à gauche du curseur et le reste de la ligne se décale. Vous pouvez alors frapper autant de caractères que vous voulez ; ils apparaîtront à gauche du curseur et feront décaler le reste de la ligne. Lorsque vous en avez assez, amenez le curseur où vous voulez avec les flèches et continuez la frappe.

Si, maintenant, vous avez frappé une ligne qui contient tellement de fautes que vous ne voulez pas la corriger, il vous suffit de faire une commande CLR (CTL puis



touche E, déjà vue au paragraphe précédent) pour effacer toute la ligne.

En résumé, vous pouvez déplacer le curseur au sein d'une ligne avec les flèches horizontales, vous pouvez effacer des caractères à droite du curseur avec la touche DELETE et en insérer à gauche du curseur avec la commande INS. Vous pouvez effacer toute une ligne avec la commande CLR.

### Encore un mot :

Comme un ordinateur peut manipuler des chiffres et des lettres et que la lettre O et le chiffre 0 se ressemblent, il a fallu trouver un moyen de les différencier facilement lors de l'écriture. Pour cette raison, la lettre O est représentée normalement par un cercle ou un ovale (selon les machines) alors que le chiffre 0 est représenté par un cercle ou un ovale barré en biais. Cette représentation est universelle en informatique et n'est pas une particularité de l'EXL 100.

## CHAPITRE 3

# Notre première instruction

### Mode direct et mode programmé :

Ainsi que nous vous l'avons expliqué dans le chapitre 1, un ordinateur est capable d'effectuer un certain nombre d'instructions contenues dans un programme et c'est là sa fonction première mais il est également possible d'utiliser un ordinateur en mode DIRECT ou encore en mode CALCULATEUR pour lui faire exécuter immédiatement des opérations. Dans ce cas, votre ordi-

nateur se comporte presque comme une calculatrice ordinaire et c'est par cela que nous allons commencer.

Pour que l'ordinateur puisse savoir si vous voulez l'utiliser en mode programmé ou en mode direct, il faut le lui indiquer de la façon suivante :
- Toute ligne qui commence par un numéro de ligne c'est-à-dire par un nombre compris ente 1 et 32766 suivi par un espace puis par un mot clé du Basic est considéré comme faisant partie d'un programme. Elle n'est pas exécutée immédiatement.
- Toute ligne qui commence par une instruction Basic est exécutée aussitôt en mode direct.

## L'instruction PRINT

La première instruction que nous allons étudier est PRINT ; elle indique à l'ordinateur qu'il doit afficher quelque chose sur l'écran (en anglais print veut dire imprimer). Elle dispose de très nombreuses possibilités que nous verrons tout à l'heure mais pour l'instant nous allons l'utiliser sous sa forme la plus simple.

Faites un RESET pour remettre votre écran à l'état initial et frappez PRINT 2. Pour cela, vous pouvez frapper successivement les lettres P, R, I, N et T puis un espace et ensuite le chiffre 2 mais vous pouvez aussi faire afficher d'un seul coup le mot PRINT suivi d'un espace en faisant FONCTION K puisque PRINT est inscrit au-dessus de la touche K. Ceci est valable pour tous les mots clés du Basic inscrits au-dessus des touches.

Pour frapper un mot clé ou instruction du Basic, vous pouvez frapper séparément les lettres qui le constituent ou utiliser les mots clés inscrits au-dessus de certaines touches. Dans ce dernier cas, le mot clé s'affiche sur l'écran suivi d'un espace et d'une parenthèse ouverte éventuelle si celle-ci est obligatoire après le mot choisi.

Revenons à notre PRINT 2 ; après l'avoir frappé, faites retour chariot puisque c'est cela qui demande à l'ordinateur d'interpréter ce que l'on a frappé au clavier (revoyez le chapitre 2 si nécessaire). Le chiffre 2 apparaît sous la ligne que vous venez de frapper précédé par un espace.



L'ordinateur a donc bien exécuté l'ordre PRINT 2 qui veut dire afficher 2.

Ecrivons maintenant PRINT 2 + 3, sans faire de reset préalable. Votre ligne apparaît sous la ligne contenant le 2 qu'elle fait remonter d'un écran. Vous pouvez mettre des espaces entre 2, + et 3 ou ne pas en mettre, l'ordinateur les ignore ; pour lui 2 + 3 est la même chose que 2+3. Dès que vous faites retour chariot, l'ordinateur interprète l'ordre et affiche le résultat de 2+3 soit 5 sous la ligne PRINT 2+3.

## Les quatre opérations

Notre ordinateur ne sait pas faire que des additions et dispose, bien sûr, des quatre opérations : addition (notée +), soustraction (notée -), multiplication (notée *) et division (notée /). Attention donc aux signes de représentation de la multiplication et de la division.
Vous pouvez essayer toutes les opérations de votre choix en n'oubliant pas de les faire précéder de l'instruction PRINT qui en fait afficher le résultat :

- PRINT 2 * 3 fait afficher 6
- PRINT 8 /2 fait afficher 4
- PRINT 2 * 3 + 4 - 6 fait afficher 4, etc...

Comme sur une calculatrice, vous pouvez enchaîner des opérations sur une même ligne (voir le dernier exemple ci-dessus). Dans ce cas, votre ordinateur respecte des PRIORITES entre les opérations ainsi les multiplications et divisions seront toujours exécutées avant les additions et soustractions.
Par exemple PRINT 2 * 3 + 4 * 5 fera afficher 26 puisque l'on exécutera en premier le 2 * 3 et le 4 * 5 et que l'on ajoutera ensuite les résultats.

Pour modifier ces priorités, vous pouvez, comme en arithmétique, utiliser des parenthèses. Dans ce cas, le contenu des parenthèses est toujours effectué le premier. Par exemple PRINT 2 * (3 + 5) donnera 16 alors que PRINT 2 * 3 + 5 aurait donné 11.
Votre ordinateur sait donc utiliser les quatre opérations

+, -, * et /. Multiplication et division sont prioritaires sur addition et soustraction sauf si l'on utilise des parenthèses pour modifier ces règles.

## Un PRINT facultatif :

Comme il est assez pénible de frapper PRINT à chaque fois que l'on veut avoir le résultat d'un calcul, l'EXL 100 permet de se passer de PRINT en mode direct. Ainsi, si vous faites :

2 * 3

vous verrez apparaître 6 et le curseur se positionnera juste après ce 6 vous permettant ainsi d'enchaîner un autre calcul. Si vous ne voulez pas, il suffit de faire un CLR (..... CTL E bien sûr !).

Cette possibilité d'enchaînement des calculs va plus loin ; en effet, supposons que vous ayez fait 2 * 3 ce qui vous donne 6 avec le curseur juste après et que vous vous rendiez compte que vous voulez calculer l'inverse de 6 (1 divisé par 6). Il est inutile d'effacer la ligne puis de frapper à nouveau 1/6. Utilisez la flèche à gauche pour amener le curseur sur le 6 ; puis faites INS (CTL A) et insérez 1/ puis faites un retour chariot : le résultat de 1/6 apparaît alors.

Bien sûr, avec cet exemple, cela a peu d'intérêt mais si le résultat obtenu est très long (l'EXL 100 peut manipuler des nombres de 10 chiffres) et si le calcul oublié "avant" le résultat est complexe, cette possibilité est intéressante. Elle n'est pas nouvelle car si vous êtes observateur, vous avez du reconnaître une partie de ce que nous avons expliqué au chapitre précédent dans "corrigez vos fautes".

## Nombres entiers et décimaux :

Dans les exemples précédents, nous n'avons montré que des nombres entiers mais votre ordinateur peut manipuler aussi des nombres décimaux et même des nombres en notation scientifique. Il peut aussi travailler avec des nombres positifs ou négatifs.
Les nombres négatifs sont écrits précédés d'un signe



moins (-) alors que les nombres positifs sont écrits seuls et non pas précédés du signe plus.

L'écriture des nombres décimaux se fait à la méthode américaine c'est-à-dire que la virgule est remplacée par un point : ainsi 2,3 doit être écrit 2.3

Les zéros précédant un nombre ou placés après le dernier chiffre après la virgule sont ignorés et automatiquement enlevés par l'ordinateur. Ces zéros sont appelés zéros NON SIGNIFICATIFS.
Essayez PRINT 000023, vous verrez afficher : 23 ; de même PRINT 2.340000 fera afficher 2.34.

Essayez PRINT 1234567000000, votre ordinateur vous répondra alors 1.234567E+12 ce qui signifie 1.234567 que multiplie 10 à la puissance 12 (c'est-à-dire que multiplie le nombre constitué par un 1 suivi de 12 zéros). Si vous avez besoin de ce type de notation, vous devez en principe avoir les connaissances mathématiques s'y rapportant. Il vous suffit donc de savoir que le "10 puissance" est représenté ici par un E suivi par le signe de l'exposant puis par l'exposant lui-même.
Votre EXL 100 peut manipuler des nombres en notation scientifique compris entre 9E+127 et 1E-128. Au-dessus de 9E+127 il y a erreur de dépassement de capacité alors qu'en dessous de 1E-128, l'ordinateur considère que votre nombre est nul tellement il est petit.

Lors de l'écriture d'un nombre en notation scientifique, le signe de l'exposant peut être omis s'il est positif ; ainsi, 1E6 a le même sens que 1E+6

Votre ordinateur passe tout seul dans le mode de notation qui lui convient le mieux ; ainsi, si vous faites PRINT 1E6, il affichera 1000000 car cela lui est possible sans dépasser les 10 chiffres qu'il sait manipuler. Par contre si vous frappez 123456789900 il passera automatiquement en notation scientifique car ce nombre fait plus de dix chiffres. Un arrondi aura lieu automatiquement car la partie significative d'un nombre en notation scientifique ne comporte que 7 chiffres.

**Quelques définitions :**

Pour la suite de nos explications et pour que vous compreniez le langage des informaticiens que vous pourrez ensuite lire dans d'autres ouvrages, sachez que les nombres entiers sont appelés nombre entiers (c'est original !). Les nombres décimaux sont appelés nombres décimaux ou nombres réels.
Les nombres en notation scientifique sont appelés nombres en notation scientifique ; la partie du nombre précédant la lettre E est la MANTISSE et le chiffre et le signe qui suivent la lette E consituent l'EXPOSANT.

Les symboles des quatre opérations sont appelés les OPERATEURS ARITHMETIQUES et un ensemble de chiffres, de parenthèses et d'opérateurs constitue ce que l'on appelle une EXPRESSION NUMERIQUE car, dans ce cas, elle ne contient que des nombres. Ainsi, 2 * 3 + (4 - 5) est une expression numérique.

## CHAPITRE 4

# Des chiffres et des lettres

**Définissons des variables :**

Dans le chapitre précédent, nous avons manipulé des nombres sous diverses formes mais qui ont toujours été exprimés directement. Un ordinateur sait aussi travailler autrement et peut manipuler des nombres en faisant référence à des noms que vous leur avez donnés.
Essayez de frapper les lignes suivantes :

POMME = 25 suivi de retour chariot bien sûr, puis PRINT POMME, vous voyez alors s'afficher 25 en bas de l'écran.

L'explication est simple, votre première ligne a créé dans



la mémoire de l'ordinateur, une VARIABLE appelée POMME et lui a affecté, grâce au signe égal, la valeur 25. Lorsque vous avez demandé grâce au PRINT de la ligne suivante d'afficher la variable POMME, l'ordinateur est allé chercher dans sa mémoire la valeur que vous aviez donnée à POMME et vous l'a indiquée.
Si maintenant vous refaites POMME = 30, la valeur 25 précédente va être perdue et POMME se verra affecter la valeur 30.

A un instant donné, une variable ne peut représenter qu'un nombre donné et un seul.

Si, par contre, vous avez besoin de plusieurs variables, vous pouvez très bien en définir autant que vous voulez et leur affecter les valeurs que vous voulez. Essayez pour cela de frapper les lignes qui suivent (toujours terminées par retour chariot bien entendu) :

POMME = 25
BANANE = 40
POIRE = 2.345

PRINT POIRE doit vous faire afficher 2.345 alors que PRINT BANANE vous ferait afficher 40 et PRINT POMME 25. Vous avez défini trois variables dont POMME, BANANE et POIRE sont les noms. Ces variables peuvent maintenant être utilisées autant de fois que vous voulez ; elles sont dans la mémoire de l'ordinateur.

Pour changer la valeur donnée à une variable, il suffit de lui affecter une nouvelle valeur en écrivant à nouveau VARIABLE = NOMBRE. A partir de cet instant, la variable prend la valeur spécifiée.

La mémoire de l'ordinateur est un circuit électronique et, pour qu'elle fonctionne, il faut qu'elle soit alimentée. Cela veut dire que si vous débranchez, même un court instant, votre EXL 100, sa mémoire va se vider et que les noms des variables et les valeurs que vous leur aviez données seront perdus. Faites l'essai en éteignant votre EXL 100 puis, après avoir remis en marche en frappant PRINT POMME par exemple :

Vous allez voir s'afficher 0 ; l'EXL 100 a "oublié" la valeur précédemment affectée à la variable POMME.

Le fait d'utiliser la touche RESET (poussoir blanc de la face avant) a aussi pour effet d'effacer de la mémoire de l'EXL 100 toutes les variables que vous avez pu définir. Pour vous en convaincre donnez à POMME une valeur comme ci-avant puis faites PRINT POMME pour voir s'afficher celle-ci. Faites alors un RESET et constatez que le PRINT POMME donne 0 comme après un arrêt de l'appareil.

**Les commandes CALL CLEANUP et NEW :**

Dans certains cas il peut être utile d'effacer des noms de variables de la mémoire de l'ordinateur ; comme il n'est pas possible de l'éteindre pour faire cela, car on effacerait ainsi toute sa mémoire et, donc, le programme qu'elle contient, il existe une commande qui sait faire cela ; c'est la commande CALL CLEANUP.

Essayez de faire les opérations suivantes :

POMME = 10
BANANE = 30
PRINT POMME fait afficher 10 et PRINT BANANE fait afficher 30 ; cela vous le saviez déjà !

Faites maintenant CALL CLEANUP (toujours suivi de retour chariot bien sûr) puis essayez à nouveau PRINT POMME ou PRINT BANANE ; vous obtenez 0 preuve que les définitions de variables que vous aviez faites ont été effacées par la commande CALL CLEANUP.

La commande CALL CLEANUP efface de la mémoire toutes les variables précédemment définies.

Il existe également un autre moyen d'effacer des variables de la mémoire de l'EXL 100, c'est en utilisant la commande NEW. Cette commande est normalement utilisée pour vider la mémoire de l'EXL 100 de tout ce qu'elle contient et est utilisée avant de commencer à écrire un programme. Comme elle vide la mémoire, elle efface aussi les variables que celles-ci pouvait contenir.



Il ne faut pas confondre CALL CLEANUP et NEW. CALL CLEANUP n'efface de la mémoire que les variables précédemment définies alors que NEW efface le contenu de la mémoire.

**LET, une instruction qui n'existe plus :**

Si l'on veut être rigoureux et suivre les règles initiales du langage Basic, on ne doit pas affecter de valeur à une variable comme nous l'avons fait en écrivant VARIABLE = NOMBRE mais l'on doit faire normalement :
LET VARIABLE = NOMBRE. Ainsi aurions-nous du écrire :
LET POMME = 25 par exemple.

Comme beaucoup de micro ordinateurs modernes, l'EXL 100 admet que l'on n'écrive pas le LET et que l'on se contente de marquer VARIABLE = NOMBRE. Mais si cela vous amuse vous pouvez utiliser le LET, il comprendra aussi.
D'autre part, l'EXL 100 admet une frappe encore plus réduite dans certains cas ; en effet, si vous voulez affecter une même valeur à plusieurs variables vous pouvez écrire :
VARIABLE 1, VARIABLE 2, etc ... = NOMBRE.
Toutes les variables se verront affecter la valeur NOMBRE.
Ainsi :
POMME, POIRE, BANANE = 30 affecte 30 à chacune des variables comme vous pouvez le constater en faisant ensuite un PRINT POMME, un PRINT POIRE et un PRINT BANANE.

**Un signe égal qui ne signifie pas toujours égal :**

Un ordinateur aime l'ordre ! Si en arithmétique on peut écrire A = 2 ou 2 = A, pour un ordinateur cela peut être totalement différent. En effet, lorsque vous écrivez POMME = 25 cela signifie : je définis la variable POMME et je lui donne la valeur 25. L'ordinateur comprend toujours un signe égal dans ce sens et si vous aviez écrit 25 = POMME il n'aurait pas aimé car cela vou-

drait dire : je définis la variable 25 et je lui donne la valeur POMME ce qui ne veut rien dire.

Les noms de variables que nous avons utilisés jusqu'à maintenant ont été quelconques ; en fait, votre EXL 100 accepte beaucoup de choses à condition de respecter quelques règles élémentaires :
- Un nom de variable doit commencer par une lettre, le tiret de soulignement (-) ou le symbole a commercial. (@)
- Un nom de variable ne doit comporter que des lettres, des chiffres, un tiret de soulignement et le symbole a commercial. (@)
- Un nom de variable ne doit pas être un mot clé du Basic mais peut en contenir un.
- Un nom de variable doit comporter au maximum 15 caractères.
- Dans un même programme, il ne peut y avoir plus de 95 noms de variables utilisés simultanément.

Ainsi : BONJOUR, POMME, X, A9B, XYZ sont des noms de variables corrects alors que ABS (identique au mot clé ABS), 9AB (commence par un chiffre) ou A?*B (comporte autre chose que des chiffres ou des lettres) ne sont pas des noms de variables corrects.

## Utilisons les variables :

Une fois que nous avons défini des variables, nous pouvons les utiliser dans des expressions numériques comme celles que nous avons vues dans le précédent chapitre.

Une variable, une fois définie, s'utilise comme le nombre qu'elle représente.

Ainsi, vous pouvez faire :

TOTO = 6.50
TITI = 3.40
PRINT TOTO + TITI et vous verrez s'afficher 9.90.

Comme les variables représentent des nombres, vous pouvez utiliser les quatre opérations vues au chapitre précédent et faire toutes les combinaisons que vous



souhaitez ; ainsi BANANE * 2 + POMME * A9B - BONJOUR + 7 / EXL sera compris et calculé par l'ordinateur sous réserve que les variables BANANE, POMME, A9B, BONJOUR et EXL aient été préalablement définies.

## Des variables qui varient :

Nous allons maintenant essayer quelque chose d'étrange ; frappez : A = 2, vous affectez ainsi à la variable A la valeur 2.
PRINT A vous fait afficher 2 ce qui confirme ce que nous venons de dire.
Ecrivez maintenant :
A = A + 1 puis, après le retour chariot traditionnel, faites un autre PRINT A ; vous devez lire 3. Que s'est-il passé ?
La ligne A = A + 1 a changé la valeur affectée à A de la façon suivante ; elle veut dire : je donne à la variable A la valeur A + 1 ; comme A avait déjà été définie et contenait 2, A vaut maintenant 3. Ce processus peut être répété à l'infini et est très souvent utilisé dans de nombreux programmes.

## Les chaînes de caractères :

Une possibilité très intéressante des ordinateurs est qu'ils peuvent manipuler autre chose que des nombres. Pour vous en persuader essayez de frapper la ligne suivante : PRINT "BONJOUR" n'oubliez pas les guillemets (SHIFT 3). Après le retour chariot traditionnel, vous allez voir s'afficher BONJOUR en bas de votre écran, Vous venez d'utiliser une CHAINE DE CARACTERES. Une chaîne de caractères est n'importe quel assemblage de caractères compris entre deux guillemets ; un avant le premier caractère de la chaîne et un après le dernier.
Alors que dans les expressions numériques votre EXL 100 ignorait les espaces (nous vous avons dit que 2 + 3 était pour lui la même chose que 2+3), dans une chaîne de caractères chaque espace compte

comme un caractère. Si vous faites PRINT "EXL 100" vous aurez en bas de l'écran EXL 100. Les espaces ont été pris en compte.

Ces chaînes de caractères sont très utiles car elles permettent de faire "parler" votre ordinateur. Dans des programmes, c'est grâce à elles que vous voyez des messages apparaître sur l'écran ; ainsi lorsqu'un ordinateur qui exécute un programme vous dit :
QUEL EST VOTRE NOM ? c'est que dans le programme qu'il exécute se trouve la ligne suivante :
PRINT "QUEL EST VOTRE NOM ?"

Il n'y a qu'une précaution à prendre au niveau des chaînes de caractères et elle concerne les guillemets. Ceux-ci servent à indiquer les extrémités des chaînes de caractères, ils ne doivent pas être contenus dans une chaîne. Pour tourner la difficulté, la règle suivante est utilisée : si un guillemet doit faire partie d'une chaîne de caractères, il doit être écrit en double. Ainsi, pour définir la chaîne suivante :

L'INFORMATIQUE "C'EST LE PIED" AVEC L'EXL 100
vous devrez écrire :
"L'INFORMATIQUE ""C'EST LE PIED"" AVEC L'EXL 100".

Pour pouvoir manipuler les chaînes de caractères avec souplesse, il est possible de leur donner des noms comme ce que nous avons expliqué ci-avant pour les nombres. Cependant, pour ne pas que l'ordinateur confonde ces noms avec ceux des variables "nombres", ils doivent respecter les règles suivantes :
- Un nom de variable chaîne de caractères est toujours suivi du symbole dollar ($).
- Un nom de variable chaîne de caractères doit respecter toutes les contraintes exposées ci-avant pour les variables numériques.

Ainsi, BONJOUR$, POMME$, X$,A9B$, etc ... sont des noms de variables chaîne de caractères corrects alors que ABS$ (comporte le mot clé ABS), 9AB$ (commence par un chiffre), A?*B$ (comporte autre chose que des



lettres ou des chiffres), TITI (ne se termine pas par dollar) et ABCDEFGHIKLMNOP$ (comporte plus de 15 caractères) ne sont pas des noms de variables chaînes de caractères corrects.

Pour affecter des valeurs à des variables chaînes de caractères, il suffit de faire comme pour les nombres vus ci-avant ; essayez :
A$ = "BONJOUR"
PRINT A$ et vous verrez s'afficher BONJOUR en bas de l'écran.

**Les tableaux :**

Il arrive bien souvent que de simples variables telles que celles que nous venons de voir ne suffisent pas. On peut alors faire appel à des TABLEAUX c'est-à-dire à des ensembles de variables repérés par un même nom et par un indice différent.
Examinez la liste de valeurs suivantes :

15 25 32 0 1 23 12 8 1000 1234

Vous pouvez donner un nom à chacune mais, vous pouvez aussi considérer que ces variables font partie d'un tableau qui, dans cet exemple, comporte 10 éléments. Si l'on appelle ce tableau A, 15 sera A(0), 25 sera A(1), .... et 1234 sera A(9). A est le nom du tableau ou encore le nom de la VARIABLE INDICEE. Le chiffre contenu entre parenthèses est l'INDICE ; c'est en fait lui qui repère la position de la variable dans le tableau, sachant que les indices commencent à 0.

Une variable indicée ou tableau peut recevoir n'importe quel nom respectant les contraintes relatives aux noms de variables exposées ci-avant. De plus, un nom de variable indicée ne doit pas exister ailleurs dans un programme comme nom de variable simple contrairement à certains Basic. Ainsi, si le tableau MAT(I) est défini, il ne devra pas y avoir de variable MAT ailleurs dans le programme.

Pour que l'ordinateur puisse s'y retrouver, les variables indicées doivent être définies avant toute utilisation au

moyen de l'instruction DIM. Cette instruction précise leurs dimensions et permet à l'ordinateur de prévoir de la place dans sa mémoire pour ranger tous les éléments qu'elles vont comporter.
DIM s'utilise sous la forme DIM X (n1, n2, n3) ou X est le nom de la variable indicée et où n1, n2 et n3 sont les dimensions du tableau. En effet, l'EXL 100 peut manipuler des tableaux à plusieurs dimensions.

Un tableau à deux dimensions se présente sous la forme suivante :

| | | | |
|---|---|---|---|
| 2 | 45 | 27 | 1233 |
| A(0,0) | A(0,1) | A(0,2) | A(0,3) |
| 124 | 0 | 456 | 678 |
| A(1,0) | A(1,1) | A(1,2) | A(1,3) |

Nous avons marqué sous chaque valeur le repérage de sa position au moyen des indices de la variable ; ainsi A(1,2) est-il 456 dans cet exemple.

Pour vérifier cette possibilité de manipuler des tableaux, essayez de faire les opérations suivantes :

DIM A (1,3) qui définit un tableau de la taille de celui montré en exemple.

Vous pouvez alors entrer un par un les éléments du tableau sous la forme :

A(0,0) = 2
A(0,1) = 45
... etc ...

Lorsque c'est fini, vous pouvez rappeler n'importe quel élément du tableau et le visualiser par un PRINT A(I,J) où I et J sont les indices de l'élément que vous voulez afficher.

Vérifier l'utilité de l'instruction DIM en essayant de rentrer un élément de trop dans le tableau ; par exemple : A(1,5) = 123; vous obtenez un message Err 32 en bas d'écran qui signifie débordement. En effet DIM a précisé que le tableau ne contiendrait que 2 lignes de 4 éléments ; il est donc exclu d'y mettre plus de choses (5 éléments par ligne dans ce cas).



Pour vous éviter des oublis fâcheux, l'EXL 100 est capable d'accepter des tableaux qui comportent jusqu'à 11 éléments sans définition préalable par un DIM.

Enfin, et nous en resterons là pour les tableaux, il est interdit, dans un programme de redéfinir la taille d'un tableau plusieurs fois de suite.

CHAPITRE 5

# Notre premier programme

## L'écriture d'un programme

Nous avons vu au chapitre 3 qu'un ordinateur pouvait fonctionner dans deux modes différents : le mode direct que nous avons utilisé jusqu'à maintenant et le mode programmé. La seule différence d'écriture entre ces deux modes est qu'une ligne, en mode programmé, doit commencer par un nombre qui est son NUMERO DE LIGNE.
Faites l'expérience suivante :

10 POMME = 25, dès que vous avez frappé retour chariot, et contrairement à ce qui se passait jusqu'à maintenant, le curseur se positionne sur la ligne suivante et l'EXL 100 attend. Il a rangé la ligne que vous venez de frapper dans sa mémoire et il attend soit un ordre, soit une nouvelle ligne.
Faites alors ;

20 PRINT POMME ; le même phénomène se produit et cette ligne se place en bas de l'écran sous la ligne n° 10.
Ces deux lignes constituent un programme qui est stocké en mémoire dans l'ordinateur. Tant que vous ne coupez pas le courant, que vous n'utilisez pas la commande

NEW ou que vous n'appuyez pas sur le poussoir RESET, le programme reste dans la mémoire.

Avant de le faire exécuter, nous allons voir quelques particularités de l'écriture d'un programme.
Les numéros de lignes tout d'abord doivent être compris entre 1 et 32766 mais on les fait en général se suivre de 10 en 10 (10, 20,30, etc ...), pourquoi ? Tout simplement parce qu'il arrive souvent, lorsque vous écrivez un programme, que vous vous rendiez compte que vous avez oublié une ou plusieurs lignes au beau milieu de celui-ci. Il est alors possible de les ajouter en leur donnant un numéro compris entre les numéros des deux lignes situées de part et d'autre ; en voici un exemple.
Supposons que, dans notre programme précédent, nous voulions ajouter avant le PRINT la ligne BANANE = 30. Il suffirait de faire :
15 BANANE = 30. Cette ligne apparaît à la suite des autres mais si vous faites maintenant LIST (que nous décrivons quelques lignes plus loin) vous voyez s'afficher :
10 POMME = 25
15 BANANE = 30
20 PRINT POMME.
La ligne n° 15 a automatiquement été insérée entre 10 et 20. Il en aurait été de même pour tout numéro de ligne compris entre 10 et 20. Et c'est là une chose fondamentale :

Quel que soit l'ordre dans lequel on les frappe, les lignes d'un programme sont toujours classées et exécutées dans l'ordre de numéro croissant.

Vous comprenez alors pourquoi l'on fait évoluer les numéros de 10 en 10 ; Celà permet en cas d'oubli de rajouter jusqu'à 9 lignes entre deux lignes quelconques.

**Quelques règles à respecter :**

Nous vous avons dit dans un précédent chapitre que l'ordinateur ignorait les espaces (autres que ceux contenus dans une chaîne de caractères) puisque 2 + 3 était pareil que 2+3.



Pour l'écriture des lignes de programme cette règle est valable mais avec quelques restrictions. En effet :
- Il faut au moins un espace entre un numéro de ligne et l'instruction qui suit.
- Il faut au moins un espace entre une instruction et la donnée qui la suit sauf si cette donnée est comprise entre parenthèses.

Remarquez que, pour vous faciliter la vie, les mots clés du Basic accessibles par la frappe de la touche FONCTION et d'une autre touche (PRINT avec FONCTION K par exemple) sont automatiquement affichés sur l'écran suivis de l'espace obligatoire. C'est donc une bonne habitude à prendre que d'utiliser FONCTION et les touches de mots clés plutôt que de frapper ceux-ci caractère par caractère ; en plus cela va beaucoup plus vite dès que l'on écrit des programmes un peu longs. La frappe de 100 lignes de PRINT en mode caractère par caractère nécessite la frappe de 600 touches (5 pour P R I N T et l'espace qui suit) alors qu'avec FONCTION il n'en faut que 200 (FONCTION et K) !

Comme sur de nombreux Basic, l'EXL 100 vous permet d'écrire plusieurs instructions sur la même ligne. Elles doivent être séparées les unes des autres par deux points (:) et le nombre total de caractères de la ligne de programme ainsi constituée ne doit pas dépasser 80 caractères ; c'est en effet une limite pour l'EXL 100.

La taille maximum d'une ligne ne peut dépasser 80 caractères soient 2 lignes d'écran.

**Listing et commande LIST :**

Nous verrons tout à l'heure que, lorsque l'on a frappé un très long programme, il peut être nécessaire de faire afficher celui-ci sur l'écran à partir de tel ou tel numéro de ligne ou tout entier. L'ensemble des lignes d'un programme affichées sur un écran ou sur le papier d'une imprimante constitue le LISTING du programme.
Votre ordinateur vous permet de faire apparaître ce listing sur l'écran au moyen de la commande LIST.
Faites LIST suivi de retour chariot, votre programme réapparaît à partir du bas de l'écran. il

était déjà, puisque nous venions juste de l'écrire, mais vous constatez bien qu'il s'y réécrit.

Faites maintenant LIST 10, vous voyez s'afficher la ligne 10 : LIST 20 ferait s'afficher la ligne 20 et ainsi de suite. Vous pouvez aussi faire LIST 20 - et vous verrez apparaître les lignes 20 et 30 car LIST N - fait afficher de la ligne N à la fin du programme. De même, vous pouvez faire LIST - 20 et vous verrez les lignes 10, 15 et 20 car LIST - N fait afficher les lignes du début à la ligne N. Ce n'est pas compliqué et c'est pratique pour les programmes très longs dont on ne veut voir qu'un morceau.

Si votre programme est trop long pour tenir dans l'écran (ce qui n'est pas encore le cas), le listing se poursuit de la façon suivante, les lignes disparaissent en haut de l'écran au fur et à mesure que les nouvelles lignes apparaissent en bas ; c'est le SCROLL (chapitre 2). Si, lors d'un tel défilement vous voulez suspendre le listing pour lire un groupe de ligne à tête reposée, il suffit de frapper n'importe quelle touche (sauf les touches spéciales). Pour faire continuer le listing, il suffit de frapper à nouveau n'importe quelle touche.

**Numérotation de lignes automatique :**

Lorsque l'on écrit un programme, il devient vite fastidieux d'écrire à chaque fois le numéro de ligne et de penser à augmenter celui-ci de 10 unités. Pour vous faciliter la tâche, votre EXL 100 dispose d'une fonction de numérotation de lignes automatique. Pour l'essayer, effacez votre programme actuel par une commande .....NEW (très bien si vous vous en êtes souvenu.). Faites alors NUMBER ou NUM ou FONCTION # ; vous voyez apparaître 100 suivi d'un espace ; frappez alors notre programme de tout à l'heure commençant par POMME = 25. Dès la frappe du retour chariot, 110 apparaît à la ligne suivante et vous pouvez continuer la frappe de façon à avoir le programme suivant l'écran :



```
100 POMME = 25
110 BANANE = 25
120 POIRE = 30
130 PRINT POMME
```

remarquez combien la frappe est rapide avec les touches de fonction ; la ligne 130 nécessite 15 touches en frappe "normale" et seulement 7 en utilisant la numérotation automatique et FONCTION K pour PRINT.

Lorsque cette frappe est finie, le curseur se trouve ligne 140 ; pour arrêter la numérotation il suffit alors de frapper un retour chariot tout seul et le curseur saute à la ligne suivante sur laquelle aucun numéro ne s'affiche : vous êtes sorti de la numérotation automatique.

Telle qu'elle vient d'être utilisée, la fonction de numérotation de lignes part de 100 et augmente les numéros de 10 en 10 mais si cela ne vous plaît pas vous pouvez aussi faire :
NUM ou NUMBER M, N et à ce moment-là vous commencerez votre numérotation à M en augmentant les numéros N par N.

**Corrigeons encore nos fautes :**

Nous avons vu au chapitre 2 comment corriger nos fautes de frappe. Tout ce qui est écrit dans ce chapitre est valable aussi bien pour le mode direct que pour le mode programmé pour ce qui est de la correction des fautes au fur et à mesure de la frappe d'une ligne.
En mode programmé, il est de plus, possible de corriger des fautes après la frappe de plusieurs lignes de programmes. Pour cela, il suffit d'utiliser les flèches de déplacement du curseur vers le haut ou vers le bas pour amener celui-ci sur la ligne voulue.
Si vous avez suivi nos essais, vous devez avoir le programme précédent sur l'écran ; frappez alors la flèche dirigée vers le haut ; tout l'écran descend et le curseur se retrouve successivement sur les différentes lignes de votre programme. Lorsqu'il est sur une ligne, vous pouvez faire toutes les corrections expliquées au chapitre 2 ; elles seront immédiatement prises en compte à une seule exception près : la commande CLR (CTL E) ne

fonctionne pas dans ce mode ; elle efface bien la ligne de l'écran mais non du programme ; pour cela, il faut utiliser la commande DEL ou DELETE.

Pour effacer une ou plusieurs lignes d'un programme, il suffit de frapper DEL ou DELETE ou FONCTION U suivit du ou des numéros de lignes à effacer. S'il y a plusieurs numéros, il faut les séparer par des virgules. Essayez DEL 110 puis faites LIST, vous verrez que la ligne 110 a disparu de votre programme.

Les possibilités de correction permettent aussi quelques facilités d'écriture ; en effet, en déplaçant le curseur il est possible de l'amener sur le numéro de ligne et, donc, de modifier celui-ci. Si vous faites cela vous répétez ainsi la ligne concernée sous un autre numéro. Pour le vérifier faites les opérations suivantes :

- Amenez le curseur sur la ligne 100 et modifiez le 100 en 105
- Faites des retours chariot pour sortir de ce mode de correction et faites ensuite un LIST.
- Vous constatez que la ligne 100 est toujours présente et qu'il existe maintenant une ligne 105 qui lui est identique.
- Cette possibilité est très intéressante lorsque vous avez de longues lignes très peu différentes à écrire dans un programme.

Effacez cette ligne 105 par un DEL 105 pour ne pas perturber la suite de notre exposé.

**Changez les numéros de lignes :**

Nous venons d'effacer une ligne du programme ce qui fait que la numérotation de celle-ci est devenue irrégulière puisqu'elle fait 100, 120, 130. Cela n'est pas gênant et peut très bien rester comme cela, mais vous devez savoir que votre EXL 100 peut refaire seul la numérotation des lignes d'un programme.

Faites : RENUMBER ou REN ou FONCTION R ; puis faites une commande LIST ; vos lignes partent toujours de 100 mais évoluent à nouveau de 10 en 10.

Ajoutez alors une ligne entre 100 et 110 en faisant 105 BANANE = 25 puis faites à nouveau REN puis LIST ;



votre programme commence toujours en 100 mais les numéros de lignes évoluent à nouveau de 10 en 10.

La commande RENUMBER permet de refaire la numérotation des lignes d'un programme. Les numéros de lignes qui apparaissent dans certaines instructions sont automatiquement pris en compte.

Comme NUM vue ci-avant, REN permet de préciser un numéro de ligne de début et une valeur d'incrémentation des numéros ; ainsi REN 500, 100 fait refaire la numérotation des lignes en partant de 500 et augmentant de 100 à chaque ligne.

**L'exécution de notre programme :**

Après ce que nous venons d'expliquer, il ne nous reste plus qu'à apprendre comment faire exécuter un programme ; pour cela, effacez le programme se trouvant en mémoire (avec un NEW mais vous l'aviez déjà fait sans hésiter) puis frappez le petit programme suivant :

```
100 POMME = 25
110 PRINT POMME
120 BANANE = 30
130 PRINT BANANE
```

Vérifiez que vous n'avez pas fait de faute de frappe, sinon corrigez- les en utilisant les possibilités que nous venons de voir.

Pour lancer l'exécution du programme, il vous suffit alors de frapper RUN suivi de retour chariot. Vous voyez alors apparaître 25 qui correspond à l'exécution des lignes 100 et 110 puis, sur la ligne suivante, 30 qui correspond aux lignes 120 et 130.

La commande RUN, utilisée seule, fait exécuter le programme contenu en mémoire à partir d'un numéro de ligne le plus bas et jusqu'à :

- la ligne contenant l'instruction STOP ou END s'il y en a une,
- la dernière ligne de programme si aucun STOP n'existe.

Parfois, il peut être utile de faire exécuter un programme à partir d'une ligne autre que la première ; pour cela,

il est possible de faire RUN suivi par la numéro de ligne où vous voulez commencer l'exécution.

Essayez RUN 120, vous ne voyez s'afficher que 30 qui correspond à l'exécution des lignes 120 et 130 de votre programme.

Dernière chose importante à retenir concernant la commande RUN, avant de commencer l'exécution d'un programme, elle efface toutes les variables éventuellement contenues en mémoire ; exactement comme si vous utilisiez la commande CALL CLEANUP vue au chapitre 4. Pour le vérifier, faites RUN 110 ; POMME est inconnue de l'ordinateur puisque la commande RUN l'a effacée de la mémoire et que vous n'avez pas exécuté la ligne 100 pour la définir à nouveau ; les résultats 0 et 30 sont donc affichés.

## CHAPITRE 6

# Jonglez avec les chaînes de caractères

### Une curieuse addition :

Le Basic permet de réaliser un certain nombre de fonctions avec les variables chaînes de caractères. Nous allons vous les présenter dans ce chapitre et pour commencer nous allons faire une bien curieuse addition. Frappez les lignes suivantes :
A$ = "BONJOUR " vous définissez ainsi la variable A$ comme étant la chaîne de caractères BONJOUR
B$ = "MONSIEUR" même chose mais pour B$ et MONSIEUR
PRINT A$ & B$ vous voyez alors s'afficher BONJOUR MONSIEUR. Votre ordinateur a ajouté les chaînes A$ et B$, c'est-à-dire qu'il a placé leurs contenus bout à



bout. Remarquez que l'espace entre BONJOUR et MONSIEUR se trouvait en fin de la chaîne BONJOUR. Cette addition de chaînes de caractères s'appelle aussi, si l'on veut "faire bien " la CONCATENATION de chaîne de caractères.

Bien sûr, c'est la seule opération "arithmétique" que l'on puisse effectuer sur des chaînes de caractères et il est impossible de les soustraire, de les diviser ou de les multiplier.

### La fonction LEN :

La fonction LEN permet de connaître la longueur d'une chaîne de caractères. Pour l'instant cela peut vous sembler inutile puisque c'est vous qui frappez les chaînes de caractères utilisées et que, de ce fait, vous connaissez ces longueurs mais vous verrez que ce n'est pas toujours le cas dans un programme.
Pour essayer LEN, faites l'expérience suivante :

A$ = "BONJOUR" qui définit la variable chaîne de caractères A$ et lui affecte la valeur BONJOUR
PRINT LEN (A$) vous fait alors afficher 7 ce qui correspond bien au nombre de caractères contenus dans la variable chaîne A$. Essayez avec d'autres chaînes de caractères si vous le voulez et n'oubliez pas qu'un espace compte pour un caractère ainsi PRINT LEN ("EXL 100") donne 7 en raison de l'espace entre EXL et 100.

La fonction LEN s'utilise donc de la façon suivante : LEN (EXPRESSION CHAINE DE CARACTERES) et donne comme résultat le nombre de caractères de la chaîne.

Un dernier exemple vous convaincra du fait que l'on peut mettre une expression derrière LEN.

A$ = "EXL "
B$ = "100"
PRINT LEN (A$&B$) donne 7 qui est le nombre de caractères de EXL 100.

## Ne mélangeons pas les variables

Notre ordinateur sait manipuler des nombres et des chaînes de caractères ; malheureusement, il ne peut et ne doit pas les mélanger et il est impossible d'écrire par exemple A$ + AB car A$ est une chaîne de caractères et AB une variable numérique. Pourtant, dans le paragraphe précédent, nous avons utilisé le 100 de EXL comme une chaîne de caractères. Cela n'avait rien d'anormal car nous l'avions défini comme B$ = "100" c'est-à-dire comme une vraie chaîne de caractères.
Si vous faites cela, l'ordinateur considèrera le nombre ainsi défini comme une chaîne de caractères et vous ne pourrez pas l'utiliser en tant que nombre. Pour vous en convaincre, faites :

A$ = "100" vous définissez ainsi la chaîne de caractères 100
B$ = "200" vous définissez ainsi la chaîne de caractères 200
PRINT A$ & B$ ne va pas faire afficher 300 puisque A$ et B$ ne sont pas des nombres mais fera afficher 100200 qui est la concaténation des deux chaînes de caractères.

Parfois, il serait pourtant bien pratique de transformer un nombre en chaîne de caractères ou vice-versa. Nous allons voir que c'est possible avec STR$ et VAL.

## La fonction VAL :

Cette fonction transforme une chaîne de caractères en un nombre ; bien sûr, il faut que la chaîne en question représente déjà un nombre sinon la conversion est impossible. Faites l'essai suivant :

A$ = "100"
B = 200
PRINT B + VAL (A$)
Vous voyez alors s'afficher 300 car la fonction VAL a converti la chaîne de caractères 100 en le nombre 100 qui peut alors être ajouté à la variable B.



La fonction VAL ignore les espaces qui peuvent se trouver au début ou à la fin de la chaîne à convertir ainsi VAL ("  100") est identique à VAL ("100  ") et donne toujours 100.

Si la chaîne ne peut pas représenter un nombre, l'ordinateur vous le signale comme vous pouvez le constater en frappant :
PRINT VAL ("ABC") ; vous verrez alors apparaître le message d'erreur 29.
Comme il est parfois utile dans un programme de savoir si VAL peut être utilisé pour convertir une chaîne en nombre, une fonction est à votre disposition pour cela.

## La fonction NUMERIC

Cette fonction permet de savoir si une chaîne de caractères est une représentation valide d'un nombre et si la fonction VAL pourra être utilisée sans problème.
L'utilisation de NUMERIC est très simple ; essayez :
PRINT NUMERIC ("ABCD")
vous voyez s'afficher 0 ce qui signifie que la chaîne n'est pas la représentation d'un nombre.
Essayez alors :
PRINT NUMERIC ("1234")
vous voyez s'afficher -1 ce qui signifie que la chaîne est la représentation valide d'un nombre.
Attention ! NUMERIC sait aussi reconnaître les nombres en notation scientifique ; essayez :
PRINT NUMERIC ("1E+6")
et vous verrez s'afficher -1 puisque 1E+6 est la représentation valide du nombre 1 million.

Les valeurs 0 et -1 que donne la fonction NUMERIC servent dans des instructions de test qui vous sont présentées dans un prochain chapitre.

## La fonction STR$

Cette fonction est l'inverse de la fonction VAL vue ci-avant, c'est-à-dire qu'elle convertit un nombre en une chaîne de caractères. Ceci explique pourquoi STR$ est suivie du symbole dollar puisque le résultat de STR$ est une chaîne de caractères.

Faites l'expérience suivante :

A$ = "EXL "
B = 100
PRINT A$ & STR$ (100) et vous obtiendrez l'affichage de la chaîne de caractères EXL 100.

La fonction STR$ s'utilise donc de la façon suivante : STR$ (EXPRESSION NUMERIQUE) et donne comme résultat la chaîne de caractères identique au résultat de l'expression numérique.

## Un peu de gymnastique

Prises isolément, ces fonctions sont très simples aussi nous vous proposons de les combiner et de réaliser les petits exercices suivants. Pour ce faire, essayez de deviner le résultat des diverses lignes que nous vous proposons et vérifiez-le ensuite sur votre EXL 100.

VAL (STR$(100))
STR$(VAL("100"))
"EXL " & STR$(VAL("99") + 1)
VAL (STR$(LEN("ABCDEF")+2))

Vous avez tout trouvé ! Vous l'avez vérifié sur votre EXL 100 ? Alors nous pouvons continuer.

## Codes et caractères

Les circuits électroniques d'un ordinateur ne peuvent manipuler et garder en mémoire que des chiffres; il faut donc employer une astuce pour qu'un ordinateur puisse manipuler des lettres et des symboles graphiques. Cette astuce consiste à représenter tous les caractères que peut utiliser l'ordinateur par un nombre compris entre 0 et 255. Ce nombre s'appelle le CODE CARACTERE. Un certain nombre de codes sont normalisés dont le célèbre code ASCII. Cest ce code qui est utilisé sur votre EXL 100 ce qui lui permet de dialoguer très facilement avec d'autres ordinateurs au moyen d'extensions adéquates.
Le code ASCII vous est indiqué en fin de manuel en annexe 4.



Pour manipuler ces codes dans un programme, deux fonctions existent ; ce sont ASC et CHR$

Essayez la ligne suivante :
PRINT CHR$(69) ; vous voyez s'afficher E ce qui est normal car si vous consultez le code ASCII de l'annexe 4, vous constaterez que 69 est justement le code de la lettre E.

La fonction CHR$ (expression numérique) donne le caractère dont le code est égal à l'expression. Comme les codes sont compris entre 0 et 255, il faut que l'expression soit dans ces limites sinon vous verrez apparaître un message d'erreur.

La fonction ASC fait l'inverse de CHR$ et vous donne le code du premier caractère de la chaîne qui la suit. Faites l'essai suivant :

PRINT ASC ("BONJOUR") et vous allez voir s'afficher 66 qui le code de B. Si la chaîne est vide (ASC " ") le résultat 0 est fourni.

## Des caractères cachés

Si vous consultez une table donnant le code ASCII officiel, vous constaterez que tous les codes inférieurs à 32 ne correspondent pas à des caractères. Ce sont des codes dits de CONTROLE qui, sur certains ordinateurs, permettent de déclencher certaines opérations. Comme ces codes étaient inutiles sur l'EXL 100, ils ont été récupérés pour vous offrir des caractères qui normalement n'existents pas dans le code ASCII.
Le seul moyen d'accéder à ces caractères qui ne figurent sur aucune touche du clavier est d'utiliser CHR$ ; ainsi si vous faites un :
PRINT CHR$(20) verrez-vous s'afficher un c cédille. Vous pouvez explorer vous-même ces codes de 0 à 31 inclus mais sachez que les caractères qu'ils représentent figurent en annexe 4. Par ailleurs, l'utilisation d'une boucle (voir les chapitres suivants) vous permet de voir tous ces caractères d'un seul coup ...

### Coupons les chaînes en morceaux :

Nous avons vu que l'on pouvait ajouter des chaînes de caractères pour en faire de plus longues mais il est aussi possible d'extraire des chaînes de l'intérieur d'autres chaînes. Pour cela, certains Basic utilisent les instructions LEFT$, RIGHT$, et MID$ ; dans l'EXL 100, une seule fonction arrive à faire la même chose c'est SEG$. Essayer :

PRINT SEG$("VIVE L'EXL 100",6,5)
Vous voyez apparaître L'EXL ; la fonction SEG$ a pris 5 caractères de la chaîne VIVE L'EXL 100 à partir du 6ème caractère de cette même chaîne.

La fonction SEG$ (chaîne,N,M) extrait de la chaîne qui la suit M caractères à partir du N ième. Le premier caractère de la chaîne porte le numéro 1.

Cette instruction peut être utilisée avec autre chose que PRINT et peut servir à définir des variables par exemple. Ainsi, pouvez-vous écrire :

```
100 A$ = "BONJOUR MONSIEUR"
110 B$ = SEG$ (A$,9,8)
```

B$ est alors égal à la chaîne de caractères MONSIEUR comme vous pouvez le vérifier en ajoutant la ligne 120 PRINT B$ et en faisant exécuter ce programme.

### Répétez une chaîne à l'infini

Lorsque vous devez créer une chaîne constituée par la répétition en N exemplaires de quelques caractères, il existe une instruction capable de faire cela toute seule ; c'est RPT$. Essayez :

```
100 A$ = RPT$ ("ABCD",10)
110 PRINT A$
```

et vous allez voir s'afficher :
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCD qui n'est autre que la chaîne ABCD répétée 10 fois.



Cette opération est cependant limitée par le fait qu'il est impossible de créer des chaînes de plus de 255 caractères ; si tel est le cas, la chaîne obtenue est tronquée à cette valeur.

### Si vous cherchez votre chaîne :

Il peut arriver que l'on ait besoin de chercher une chaîne de caractères dans une autre. Pour ce faire, existe POS qui peut vous dire où se trouve une chaîne dans une autre chaîne. De plus, POS peut ne commencer la recherche qu'à partir du caractère de votre choix. Essayez :

```
100 A$ = "BONJOUR MONSIEUR"
110 PRINT POS (A$,"ON",1)
```

Vous allez voir s'afficher 2 ; en effet POS indique la position du premier caractère de la chaîne ON dans la chaîne A$ et la recherche doit commencer à partir du premier caractère de la chaîne. POS détecte donc le ON de BONJOUR et le O est en deuxième position.
Faites maintenant :

```
110 PRINT POS (A$,"ON",5)
```

Vous voyez apparaître 10 car la recherche n'a commencé qu'au cinquième caractère ; le ON de BONJOUR a donc été ignoré mais le ON de MONSIEUR a été trouvé et son O est en dixième position.
Si la chaîne ne peut être trouvée, POS fournit une valeur nulle.

## CHAPITRE 7

# Discutez avec votre EXL 100

### Les secrets de l'instruction PRINT :

Nous utilisons l'instruction PRINT depuis le chapitre

3 mais nous n'avons exploité qu'une infime partie de ses possibilités. Nous allons voir maintenant qu'elle peut faire de nombreuses choses.

Faites l'essai suivant, après avoir effacé la mémoire de votre ordinateur :
PRINT "BONJOUR ";"MONSIEUR"
Vous allez voir s'afficher en bas de l'écran BONJOUR MONSIEUR, l'espace entre BONJOUR et MONSIEUR faisant partie de la chaîne BONJOUR.

Essayez maintenant :
PRINT "BONJOUR ","MONSIEUR"
vous allez voir s'afficher BONJOUR
MONSIEUR.

Entre ces deux lignes existe une seule différence ; dans la première les deux chaînes de caractères sont séparées par un point virgule et dans la deuxième par une virgule.

Lorsque, à la suite d'une instruction PRINT, des variables de n'importe quel type sont séparées par un point virgule, elles sont affichées les unes à la suite des autres sans espace intermédiaire autre que ceux faisant éventuellement partie des variables.
Si, à la suite d'une instruction PRINT, des variables sont séparées par des virgules, les premiers caractères de chaque variable sont affichés à 10 divisées artificiellement en zones de 10 caractères de large. Chaque virgule fait sauter au début de la zone suivante.
Ce dernier point pouvant prêter à confusion, faites l'essai suivant :

PRINT 1,200,3000,4000
Vous allez voir sur votre écran
1 200 3000
400000

En effet le 1 apparaît à la première position de la première zone puis la virgule ordonne le saut au début de la zone suivante soit 10 caractères plus loin; le 2 est placé au début de cette zone; la virgule fait ensuite sauter à la zone suivante au début de laquelle est affiché le 3000



et enfin la dernière virgule fait sauter à la zone suivante c'est-à-dire au début de la ligne d'après; en effet, comme les lignes ne peuvent contenir que 40 caractères, la quatrième zone commence ligne suivante.

La commande PRINT, comme vous l'avez constaté, fait afficher ce qui la suit mais génère aussi automatiquement un saut à la ligne après cet affichage. Essayez le programme suivant :

```
100 PRINT "BONJOUR"
110 PRINT "MONSIEUR"
120 PRINT "L'INFORMATICIEN"
```

vous obtenez lors de l'exécution :
BONJOUR
MONSIEUR
L'INFORMATICIEN

preuve qu'un saut de ligne a bien eu lieu après chaque PRINT.

Cela a deux conséquences :
- Un PRINT utilisé seul fait sauter une ligne (en fait on affiche une ligne vide de caractère)
- Il existe un moyen de ne pas générer de saut de ligne après un PRINT.

Essayez maintenant le programme suivant après avoir effacé le précédent ou en utilisant les possiblités de correction déjà vues :

```
100 PRINT "BONJOUR ";
110 PRINT "MONSIEUR ";
120 PRINT "L'INFORMATICIEN "
```

Lors de l'exécution vous obtenez :
BONJOUR MONSIEUR L'INFORMATICIEN

Le point virgule en fin de ligne PRINT a interdit le saut de ligne et l'affichage a donc continué à la suite de ce qui existait déjà.

De la même façon, nous aurions pu terminer les lignes de PRINT par des virgules; le saut à la ligne n'aurait pas eu lieu non plus et l'affichage aurait continué 10 espaces plus loin.

L'instruction PRINT peut être suivie d'un nombre quelconque de variables de tout type ce qui permet d'afficher des informations sous forme très claire. Faites l'essai suivant :

```
100 A$ = "votre prénom"
110 PRINT "BONJOUR ";A$
```

L'exécution de ce programme fera afficher BONJOUR suivi de votre prénom.
Essayez :

```
100 P = 3.40
110 PRINT "UN LITRE DE LAIT VAUT";P;"FRANCS"
```

L'éxécution de ce programme fera afficher UN LITRE DE LAIT VAUT 3.4 FRANCS.

Nous allons voir au fur et à mesure des exemples de programmes qui vont suivre toutes ces possibilités de PRINT mises en valeur.

**Affichez des textes où vous voulez ;**

Les possibilités de PRINT sont bien intéressantes mais pour l'instant elles ne vous permettent que d'afficher des choses sur la dernière ligne de l'écran; qu'à cela ne tienne, nous allons faire mieux avec l'instruction LOCATE.

Essayez le programme suivant :

```
100 LOCATE (10,12)
110 PRINT "BONJOUR"
```

et vous allez voir s'afficher BONJOUR à peu près au centre de votre écran. L'instruction LOCATE a défini



la position d'affichage compte tenu des règles suivantes : LOCATE (A,B) positionne le curseur d'affichage utilisé par PRINT sur la ligne A et la colonne B. Compte tenu de la taille de l'écran de l'EXL 100 et de certaines contraintes internes, le numéro de ligne peut aller de 1 à 22 inclus; 1 étant la ligne la plus haute de l'écran. Le numéro de colonne peut aller lui de 1 à 40 inclus, 1 étant la colonne la plus à gauche de l'écran.

Attention! Même en utilisant LOCATE, vous ne pourrez jamais faire afficher quelque chose sur la ligne de service en haut de l'écran; elle porte le numéro 0 et est réservée à l'EXL 100.

**Lorsque l'EXL 100 vous interroge :**

Jusqu'à maintenant, votre EXL 100 vous a fourni de nombreux messages mais vous ne lui avez donné en retour aucune information autre que les programmes que vous avez écrits. Il existe une instruction pour cela : c'est INPUT.

Essayez le programme suivant :

```
100 INPUT "QUEL EST VOTRE PRENOM?";A$
```

Si vous lancez son exécution, la ligne 100 va faire afficher la question QUEL EST VOTRE PRENOM? et le curseur va se positionner après cette question en attendant que vous répondiez. Frappez alors votre prénom suivi d'un retour de chariot : BONJOUR suivi de votre prénom vont apparaître ligne suivante :

La ligne INPUT a donc permis d'afficher un message, d'attendre votre réponse et d'affecter votre réponse à votre variable qui était ici A$.

L'instruction INPUT affecte à la variable qui la suit la donnée que vous frappez au clavier. Tant que la donnée n'est pas fournie, le programme reste en attente. La donnée doit être du même type que la variable

(chaîne de caractères pour une variable chaîne de caractères, nombre pour une variable numérique).

Attention, comme un nombre peut être considéré comme une chaîne de caractères, vous pouvez très bien répondre par un nombre quelconque à une instruction INPUT qui attend une chaîne mais, par la suite, ce nombre ne pourra être utilisé dans des calculs sauf si vous le convertissez au moyen de VAL (chapitre 6).

Essayez maintenant le programme suivant :

```
100 INPUT "PREMIER NOMBRE : ";A
110 INPUT "DEUXIEME NOMBRE : ";B
120 PRINT "LA SOMME VAUT ";A+B; " ET LE PRODUIT ";A*B
```

Son comportement correspond bien à ce que nous avons expliqué. Remarquez la ligne 120 utilisant les possibilités de PRINT pour présenter les résultats de façon agréable. Essayez de répondre à une des deux questions par autre chose qu'un nombre (une lettre par exemple) ; le programme s'arrête et un message d'erreur est affiché signalant l'anomalie.

L'instruction INPUT peut être utilisée aussi pour accepter plusieurs réponses d'un seul coup ; modifiez le programme précédent en remplaçant 100 et 110 par :

```
100 INPUT "FRAPPEZ DEUX NOMBRES : ";A,B
```

dans ces conditions, le message FRAPPEZ DEUX NOMBRES est affiché et le curseur se positionne juste après ; vous pouvez alors frapper un premier nombre qui sera affecté à A. Ce nombre devra être suivi d'un retour chariot qui fera alors apparaître un point d'interrogation suivi du curseur sur la ligne suivante indiquant que l'EXL 100 attend une autre donnée ; vous pourrez alors frapper le deuxième nombre qui sera affecté à B. Cette forme peut être étendue à un nombre quelconque de variables.

Il est aussi possible d'utiliser le INPUT seul, sans message, sous la forme :

```
100 INPUT A
```

dans ce cas, un point d'interrogation est affiché suivi du curseur en attendant la frappe de la donnée qui sera affectée à la variable A.



L'instruction LOCATE vue ci-avant fonctionne aussi avec un INPUT suivi par du texte et permet de positionner celui-ci où vous voulez sur l'écran ; il suffit d'exécuter le LOCATE avant le INPUT concerné ; ainsi :

```
100 LOCATE (10,12)
110 INPUT "VOTRE PRENOM ?";A$
```

fait afficher VOTRE PRENOM ? ligne 10 colonne 12.

**Des INPUT spéciaux :**

L'instruction INPUT vue ci-avant arrête l'exécution du programme jusqu'à ce que vous ayez frappé la réponse attendue. C'est utile dans bien des cas, mais pour certaines applications, telles que des jeux où vous devez tirer sur une cible par exemple, il vaut mieux ne pas arrêter le programme en attendant que vous frappiez une touche sinon cela n'a plus d'intérêt.
Pour cela, existe l'instruction KEY1 qui permet de savoir si une touche est frappée sur le clavier ou la manette de jeux n° 1 (option). Il existe de même KEY2 pour la manette de jeux n° 2 (option également).
Ces instructions s'utilisent de la façon suivante :

```
CALL KEY1(A,B)
```

ce qui a pour effet d'affecter à la variable A le code de la touche qui est actionnée et à la variable B une valeur qui indique comment la touche a été actionnée. Pour bien comprendre ce mécanisme, essayez le programme suivant ; deux instructions vous sont inconnues mais n'y faites pas attention pour le moment :

```
100 CALL KEY1(A,B)
110 PAUSE .2
120 PRINT A,B
130 GOTO 100
```

Ce programme tourne en permanence et exécute sans arrêt l'instruction CALL KEY avec une pause de 0,2 seconde entre deux exécutions ; il fait afficher ensuite les valeurs données à A et B par CALL KEY.
Vous constatez que :
- Tant qu'auncune touche n'est appuyée A vaut 255 et B vaut 0

- Si une touche est appuyée un court instant A vaut le code ASCII de cette touche et B vaut 1.
- Si la même touche est maintenue appuyée, A vaut toujours son code mais B vaut -1.

Remarquez aussi que CALL KEY1 ne nécessite pas de retour chariot et n'arrête pas le programme contrairement à INPUT.

Arrêtez ce programme sans fin en faisant un BREAK (CTL puis C).

Une autre instruction, intermédiaire entre INPUT et KEY1 existe ; c'est KEY$. Comme INPUT, KEY$ arrête l'exécution du programme mais KEY$ n'attend qu'un caractère qui est impérativement traité comme une chaîne de caractères et ne nécessite pas de retour chariot. Essayez :

```
100 A$ = KEY$
110 PRINT A$
```

Si vous lancez ce programme, rien ne se passe et le curseur disparaît ; dès que vous frappez une touche, le programme continue et la ligne 110 vous fait afficher la touche frappée.

Enfin, une dernière instruction d'entrée existe sur l'EXL 100, c'est LINPUT qui signifie LINE INPUT ou entrée de ligne. Cette instruction ne peut accepter qu'une variable qui doit être une chaîne de caractères ; par contre, vous pouvez frapper ce que vous voulez en réponse à un LINPUT, toute la ligne que vous frappez, jusqu'au retour chariot de fin, est affectée à la variable chaîne de caractères. Essayez :

```
100 LINPUT "FRAPPEZ N'IMPORTE QUOI : " ;A$
110 PRINT A$
```

Lancez ce programme et frappez tous les caractères de votre choix ; lors de la frappe du retour chariot la ligne 110 fait afficher A$ qui vous permet de constater que tous vos caractères ont été pris en compte par LINPUT.



CHAPITRE 8

# Les boucles

**Des lignes qui se répètent toutes seules :**

Pour l'instant notre ordinateur n'a pas fait de gros efforts, se contentant d'exécuter les quelques lignes que nous avions frappées. Nous allons voir maintenant tout l'intérêt que peut prendre un programme avec des boucles.

Supposons qu'un boulanger veuille réaliser une table donnant le prix des baguettes de pain en fonction du nombre, lui évitant ainsi de fastidieux calculs mentaux. Il pourrait faire :

```
100 A = 2.45 (A est le prix d'une baguette de pain !)
110 N = 1
120 PRINT "1 BAGUETTE DE PAIN COUTE "; A ;
" FRANCS"
130 N = N + 1
140 PRINT "2 BAGUETTES COUTENT "; N * A ;
" FRANCS"
150 N = N + 1
```

etc ... jusqu'à ce que N ait atteint le nombre maximum de baguettes pour lequel il veut faire sa table. Cela fonctionnerait mais serait un très mauvais programme qui, en plus, serait fort long.
Nous pouvons faire mieux, plus court et plus rapide avec une boucle.
Ecrivons le programme suivant :

```
100 A = 2.45 (A est le prix d'une baguette)
110 FOR I = 1 TO 10 (si 10 est le nombre maximum
de baguettes !)
120 PRINT I ; " BAGUETTES COUTENT "; A * I ;
" FRANCS"
130 NEXT I
```

Si vous le faites exécuter vous allez voir apparaître successivement sur votre écran :

1 BAGUETTES COUTENT 2.45 FRANCS
2 BAGUETTES COUTENT 4.9 FRANCS
...jusqu'à
10 BAGUETTES COUTENT 24.5 FRANCS

Votre programme vient d'exécuter une boucle FOR NEXT. Cela se passe de la façon suivante. Lors de l'exécution de l'instruction FOR TO contenue ligne 110, la variable I se voit affecter la valeur 1 à cause du FOR I = 1 et l'ordinateur "se souvient" du fait qu'elle ne devra pas dépasser 10 à cause du TO 10. La ligne 120 se comporte alors comme un PRINT normal dans lequel I vaut 1 et fait afficher la première ligne que vous avez vue sur l'écran. Le NEXT (qui veut dire suivant en anglais) fait alors augmenter I de 1 et ordonne au programme de recommencer l'exécution à la ligne qui suit immédiatement l'instruction FOR TO. La ligne 120 s'exécute donc une autre fois mais avec I = 2 cette fois-ci.

Cela continue jusqu'à ce que I soit égal à 10, auquel cas la ligne 120 est exécutée mais, lorsque l'on arrive sur le NEXT, celui-ci ne fait plus augmenter I ni recommencer l'exécution car la ligne 110 avait dit que I devrait aller de 1 à 10 seulement.
Essayez de remplacer 10 par un autre nombre et vous constaterez que l'exécution continue jusqu'à ce nombre.

Admettons maintenant que notre programme s'adresse à un crémier qui souhaite calculer le prix de vente de ses œufs. Il peut utiliser le même programme que le boulanger mais cela ne lui convient pas car il vend les œufs par demi douzaine (6). Il lui suffit alors de faire :

```
100 A = .85 (prix d'un œuf)
110 FOR I = 0 TO 36 STEP 6
120 PRINT I/6 "DEMI DOUZAINES COUTENT";
A * I "FRANCS"
130 NEXT I
```

pour voir apparaître sur son écran :

0 DEMI DOUZAINES COUTENT 0 FRANCS
1 DEMI DOUZAINES COUTENT 5.1. FRANCS
... jusqu'à
6 DEMI DOUZAINES COUTENT 30.6. FRANCS



Le comportement de la boucle a été le même que dans le programme précédent mais, en raison du STEP 6 suivant le FOR TO, I a augmenté de 6 à chaque fois. En fait, une boucle FOR TO comporte toujours le mot complémentaire STEP qui indique de combien I augmente à chaque fois mais si l'on ne l'écrit pas, STEP est pris égal à 1.

La variable qui apparaît dans le FOR TO est appelée LA VARIABLE DE BOUCLE. Le nombre spécifié après STEP est le PAS de la boucle.

L'expression générale d'une instruction de boucle est donc :
FOR I = A TO B STEP C avec, juste après la dernière instruction faisant partie de la boucle : NEXT I. Le nom de la variable de boucle peut être quelconque mais doit être un nom de variable numérique.
Si A est supérieur à B, la boucle n'est pas exécutée et le programme saute directement de la ligne contenant l'instruction FOR TO à la ligne qui suit l'instruction NEXT.
Si le pas de la boucle est négatif, la boucle s'exécute jusqu'à ce que la variable de boucle devienne égale ou inférieure à B. Pour le constater faites l'essai suivant :

```
100 FOR I = 0 TO -10 STEP -1
110 PRINT I
120 NEXT I
```

Vous verrez ainsi s'afficher la variable de boucle I et pourrez suivre son évolution.

Les boucles du type FOR NEXT sont peu contraignantes ; il suffit de bien prendre la précaution de mettre après le NEXT qui termine une boucle le même nom de variable que celui utilisé dans le FOR TO. Cela est d'autant plus important lorsque l'on cherche à imbriquer des boucles les unes dans les autres.

Par contre, lorsqu'aucun doute n'est possible parce que vous n'utilisez qu'une boucle ou des boucles distantes et non imbriquées, vous pouvez omettre le nom de variable de boucle après le NEXT ; l'EXL 100 comprendra tout de même et associera le NEXT avec le FOR TO le plus proche.

## Imbriquons des boucles :

Il est possible d'imbriquer des boucles les unes dans les autres, encore faut-il faire cela comme il faut ; la règle suivante étant fondamentale :

Pour imbriquer des boucles les unes dans les autres, il faut que chaque boucle soit toute entière contenue dans une autre boucle.

Réalisez le petit programme suivant qui va reproduire sur votre écran un damier :

```
100 LOCATE (1,1)
110 FOR C = 1 TO 20 début de la boucle "extérieure"
de variable C
120 FOR M = 1 TO 20 début de la boucle "intérieure"
de variable M
130 PRINT CHR$(12);" ";
140 NEXT M fin de la boucle "intérieure" de variable M
150 FOR N = 1 TO 20 début d'une autre boucle "intérieure" de variable N
160 PRINT "  ";CHR$(12);
170 NEXT N fin de la boucle "intérieure" de variable N
180 NEXT C fin de la boucle "extérieure" de variable C
```

Cet exemple d'imbrication est correct puisque les deux boucles intérieures sont entièrement contenues dans la boucle extérieure. L'exécution du programme a du vous montrer la logique de son fonctionnement. La variable C de la boucle extérieure est mise à 1 et les deux boucles intérieures sont exécutées entièrement (M évolue de 1 à 20 et N de 1 à 20) puis le NEXT C fait augmenter C de 1 et les boucles intérieures sont à nouveau exécutées entièrement et ainsi de suite. Les deux boucles intérieures sont exécutées en tout 11 fois du fait de leur imbrication dans la boucle extérieure ou C évolue de 1 à 11.

Essayez maintenant le programme suivant :

```
100 FOR N = 10 TO 15 début de la 1ère boucle
110 PRINT N
120 FOR J = 30 TO 35 début de la 2ème boucle
130 PRINT J
```



```
140 NEXT N fin de la 1ère boucle
150 NEXT J fin de la 2ème boucle
```

L'éxécution de ce programme est impossible et conduit à un message d'erreur car votre imbrication est incorrecte : la deuxième boucle de variable J se termine après la première ce qui est interdit.

## Des boucles de taille variable :

Il ne faudrait pas croire, à voir les exemples précédents, que les boucles sont de taille fixe spécifiée une fois pour toutes par les valeurs contenues dans le FOR TO ; en effet, la valeur de départ et la valeur maximum de la variable de boucle peuvent elle-même être des variables comme le montre l'exemple ci-après :

Reprenez le programme de calcul du prix des baguettes de pain mais modifiez le de la façon suivante :

```
100 A = 2.45 (prix de la baguette)
110 INPUT "NOMBRE MAXIMUM DE BAGUETTES ? ";N
120 FOR I = 1 TO N
130 PRINT I ; " BAGUETTES COUTENT ";A * I ;
" FRANCS
140 NEXT
```

Vous pouvez ainsi entrer tout à loisir le nombre maximum de baguettes et, donc, le nombre de "tours" que fera la boucle.

CHAPITRE 9

# Branchements et sous programmes

## Perturbons le déroulement d'un progamme

Nous vous avons dit au chapitre 5 que même si l'on frappait les lignes d'un programme dans n'importe quel ordre, elles seraient toujours classées et exécutées par ordre de numéro croissant et c'est vrai. Par contre, il peut être utile de modifier, non pas l'ordre de rangement des lignes car ce serait la pagaille, mais leur ordre d'exécution.

```
100 PRINT "BONJOUR ";
110 GOTO 130
120 PRINT "MONSIEUR";
130 PRINT "MADAME"
```

Lors de l'exécution vous voyez s'afficher BONJOUR MADAME, exactement comme si la ligne 120 n'existait pas.

L'instruction GOTO contenue ligne 110 est une instruction de SAUT ou de BRANCHEMENT. Elle donne l'ordre au programme de continuer son exécution au numéro de ligne qui la suit.

Le saut qu'effectue un programme avec un GOTO peut avoir lieu dans n'importe quelle direction. essayez le programme suivant :

```
100 PRINT "BONJOUR"
110 GOTO 100
```

L'exécution de ce programme provoque l'impression d'une suite de BONJOUR sur l'écran puisque la ligne 110 avec son GOTO 100 ordonne au programme de continuer son exécution avec la ligne 100. Il tourne donc



en rond indéfiniment. Nous avons ainsi réalisé une boucle mais vous ne devez pas la comparer à une boucle FOR NEXT, en effet ici il n'existe aucun moyen de contrôle sur la boucle qui va tourner sans arrêt.

## Comment interrompre un programme:

Il arrive souvent, et vous vous en apercevrez vite lorsque vous mettrez au point vos propres programmes, que l'on ait besoin d'interrompre un programme en cours d'exécution parce qu'il ne fait pas ce que l'on veut ou tout simplement parce qu'il ne s'arrête plus de lui-même. Le programme que nous venons de vous faire lancer en est un bon exemple.

Au beau milieu de ces BONJOUR activez la fonction BREAK en faisant ..... CTL C bien sûr. Un message Err 37 line 110 apparaît immédiatement. Vous avez interrompu l'exécution du programme, ce que vous signale le message d'erreur, à la ligne indiquée. Vous pouvez alors faire ce que vous voulez.
Si vous faites CON ou CONTINUE vous allez faire reprendre l'exécution du programme à l'endroit où il avait été interrompu. Mais vous pouvez aussi décider que vous en avez assez et faire un LIST, un NEW ou toute autre fonction de votre choix.

Cette fonction BREAK peut être utilisée dans tous les programmes pour en interrompre l'exécution. Vous pouvez même répondre BREAK lors d'un INPUT si vous voulez.
Il peut arriver, par contre, que lors de l'essai d'un programme de votre conception, BREAK ne fonctionne pas si vous avez vraiment fait de très grosses erreurs. Dans ce cas, il ne vous restera plus qu'à faire un RESET mais attention, dans ce cas votre programme sera perdu !

Le fonctionnement de BREAK peut aussi être annulé ou modifié par l'instruction ON BREAK décrite dans un prochain chapitre.

**Les sous programmes :**

Nous allons à nouveau parler de baguettes de pain mais cette fois-ci, notre boulanger veut faire un tableau indiquant le prix des baguettes, des "restaurants" et des croissants. Pour cela, il peut écrire un programme selon le modèle déjà vu ; cela ressemblera à :

```
100 A = 2.45 prix d'une baguette
110 B = 3.7 prix d'un "restaurant"
120 C = 2.5 prix d'un croissant (au beurre !)
130 FOR I = 1 TO 10 on limite à 10 pour simplifier
l'exemple
140 PRINT I ; " BAGUETTES COUTENT "; A * I ;
" FRANCS"
150 NEXT I
160 FOR I = 1 TO 10
170 PRINT I ; " RESTAURANTS COUTENT ";
B * I ; " FRANCS"
180 NEXT I
190 FOR I = 1 TO 10
200 PRINT I ; " CROISSANTS COUTENT ". C *
I ; " FRANCS"
210 NEXT I
```

Cela fonctionne et produit bien les trois tableaux demandés mais ce programme n'est pas beau ; de plus il comporte à peu de choses près, trois fois les mêmes lignes en 130 - 140 - 150, 160 - 170 - 180 et 190 - 200 - 210. Trois fois, c'est peu mais dans certains programmes c'est plusieurs dizaines de fois que peuvent revenir les mêmes lignes ; pour éviter cela, existent les sous programmes.

Un sous programme est un ensemble d'instructions bien délimitées qui peut être utilisé autant de fois que nécessaire dans un programme. Par opposition au terme sous programme, le programme qui utilise des sous programmes s'appelle le PROGRAMME PRINCIPAL.

Nous allons écrire notre programme du boulanger avec un sous programme et vous allez tout de suite comprendre l'intérêt de ce procédé.



```
100 A = 2.45 prix de la baguette
110 A$ = " BAGUETTES" affectation de BAGUET-
TES à la variable chaîne A$
120 GOSUB 200 appel du sous programme
130 A = 3.7 prix du restaurant
140 A$ = " RESTAURANTS"
150 GOSUB 200 appel du sous programme
160 A = 2.5 prix du croissant (toujours au beurre !)
170 A$ = " CROISSANTS"
180 GOSUB 200
190 STOP
200 FOR I = 1 TO 10
210 PRINT I ; A$ ;" COUTENT " ; A * I ; " FRANCS"
220 NEXT I
230 RETURN
```

si vous faites exécuter ce programme, vous constaterez qu'il fonctionne comme le précédent; par contre il est écrit différemment et utilise un sous programme.

Le sous programme commence ligne 200 et se termine par un RETURN ligne 230.

Un sous programme doit toujours se terminer par une instruction RETURN. Par contre, aucune instruction particulière n'indique le début d'un sous programme.

Le sous programme est APPELE une première fois ligne 120, ; cela signifie que l'ordinateur saute de la ligne 120 au début du sous programme qui est ligne 200 (à cause du 200 placé derrière le GOSUB qui le lui indique) ; il exécute alors les instructions qu'il trouve à partir de la ligne 200 jusqu'à rencontrer un RETURN ; ce dernier fait revenir le programme à la ligne qui suit celle qui contient le GOSUB ; donc à la ligne 130 dans ce cas. L'exécution se poursuit alors jusqu'au GOSUB suivant où le même phénomène se reproduit.
On peut donc appeler un sous programme autant de fois que l'on veut dans un programme ; il suffit d'utiliser GOSUB aux endroits où c'est nécessaire.
Si vous êtes observateur, vous avez du remarquer un STOP ligne 190. La raison de sa présence est simple ; il arrête l'exécution en 190. S'il n'était pas là, lorsque le programme arriverait ligne 190 il continuerait son

exécution et rentrerait donc dans le sous programme ligne 200 ; les lignes 210 et 220 s'exécuteraient bien mais arrivé au RETURN de la ligne 230, ce serait la panique car l'ordinateur ne saurait plus où aller puisqu'aucun GOSUB n'a précédé ce RETURN. Un message d'erreur serait alors affiché.

Un sous programme peut être placé n'importe où dans le programme principal ; il faut seulement faire en sorte que le programme principal ne puisse pas rentrer dans le sous programme autrement que par un GOSUB. Une bonne méthode consiste à mettre tous les sous programmes après la fin du programme principal ; ce dernier étant terminé par un STOP comme dans l'exemple précédent.

Dernière remarque importante : un sous programme peut appeler autant de sous programmes qu'il lui est nécessaire. Ce sous programme se comporte alors vis-à-vis des sous programmes qu'il appelle comme un programme principal et tout ce que nous avons expliqué ci-avant reste valable.

**Les sauts calculés :**

Les instructions GOTO et GOSUB peuvent revêtir une autre forme mise en évidence dans l'exemple suivant :

```
100 INPUT "FRAPPEZ UN NOMBRE EGAL A 1 ,
2 OU 3 : ";A
110 ON A GOTO 130 , 150, 170
120 STOP
130 PRINT "VOUS AVEZ FRAPPE 1"
140 STOP
150 PRINT "VOUS AVEZ FRAPPE 2"
160 STOP
170 PRINT "VOUS AVEZ FRAPPE 3"
180 STOP
```

Si vous lancez ce programme, il va attendre un nombre égal à 1, 2 ou 3 et, selon celui-ci, affichera le message VOUS AVEZ FRAPPE N ou N est votre nombre. Cela n'a pas un intérêt extraordinaire mais permet de comprendre le fonctionnement du ON GOTO.



ON GOTO s'utilise sous la forme ON variable GOTO numéro, numéro, numéro, etc... si la variable est égale à 1, le premier numéro est sélectionné (130 dans notre exemple) ; si la variable est égale à 2, le deuxième numéro est sélectionné (150 dans notre exemple) et plus généralement, si la variable est égale à N, le Nième numéro est sélectionné. Si la variable est négative, nulle ou supérieure au nombre maximum de numéros de lignes disponibles, une erreur est générée.

Cette instruction est très utilisée dans les programmes qui vous proposent des MENUS c'est-à-dire des choix de fonctions que vous pouvez sélectionner par la frappe d'un chiffre.

La même possibilité existe pour les appels de sous programmes avec ON GOSUB qui s'utilise de la même façon que ON GOTO.

**L'instruction CALL et les sous programmes particuliers :**

Vous avez pu remarquer, depuis le début de ce manuel, que certaines instructions étaient précédées par le mot CALL ; ainsi avons-nous écrit CALL CLEANUP et CALL KEY1 par exemple. La raison est simple : les instructions écrites sous la forme CALL XXX sont en fait des sous programmes internes au Basic de l'EXL 100. Pour l'instant, vous n'avez pas à vous en préoccuper et pouvez considérer que ce sont des instructions comme les autres. Vous constaterez dans l'avant dernier chapitre de cette initiation que ces possibilités vont plus loin car vous pouvez vous-même définir des sous programmes appelés par un CALL grâce à SUB, SUBEND et SUBEXIT.
Ces sous programmes sont cependant des particularités de l'EXL 100 et lorsque les livres ou les journaux spécialisés parlent de sous programmes Basic, ils font référence aux GOSUB et RETURN vus ci-avant.

CHAPITRE 10

# Des sons et des couleurs

## Commentez vos programmes

Lorsque l'on écrit un programme, on sait généralement à quoi il sert et comment il fonctionne ; par contre, lorsqu'on veut l'utiliser plusieurs mois après ce n'est plus pareil. Pour éviter ce genre de problème, les bons programmeurs commentent leurs programmes au moyen d'une instruction appelée REM.
REM s'utilise dans un programme comme toutes les autres instructions mais tous les caractères qui suivent REM sont considérés comme du commentaire et sont ignorés par l'ordinateur. Essayez le programme suivant :

```
100 REM * DEMONSTRATION DE L'INSTRUC-
TION REM *
110 FOR I = 1 TO 20
120 REM INTERIEUR DE LA BOUCLE DE VARIA-
BLE I
130 REM 1234567890 - ABCDEFGHIJKLMNOPQRS-
TUVWXYZ < > 2 ; *
140 NEXT I
150 REM LE PROGRAMME EST FINI.
```

Si vous faites exécuter ce programme, il ne se passera rien puisqu'il va ignorer les lignes contenant REM et exécuter seulement la boucle FOR NEXT pour I allant de 1 à 20. Vous obtiendrez donc au bout de quelques dixièmes de secondes le retour du curseur indiquant que c'est terminé. Remarquez que tous les caractères situés après le REM sont ignorés par le programme, même le I en fin de ligne 120 bien que I soit le nom de la variable de boucle.

Tous les caractères qui suivent l'instruction REM sont ignorés par le programme et sont considérés comme des commentaires.



Attention toutefois si vous mettez de nombreux REM dans vos programmes; en effet il faut de la place dans la mémoire de l'ordinateur pour conserver les caractères de vos REM, même s'ils ne servent à rien.
Si, lorsque vous recopiez un programme dans un livre, la mémoire de votre EXL 100 est pleine avant que vous n'ayez pu frapper toutes les lignes, essayez de supprimer tous les REM et les commentaires qui les suivent, cela vous fera gagner un peu de place.

Enfin pour économiser votre frappe et de la place, il est possible d'abréger REM par un point d'exclamation.

## Vous allez en voir de toutes les couleurs :

Si ce n'est le fond d'écran bleu ciel, la ligne de service bleu foncé et les inscriptions jaunes qui y apparaissent, vous n'avez pas profité pour l'instant des couleurs que peut générer votre EXL 100. Cela va être chose faite avec l'instruction CLS.
Frappez :
CLS "RGC»
Votre écran devient vert, les caractères rouges et le pourtour de l'écran bleu ciel. De plus, tous les textes qui pouvaient se trouver sur l'écran ont disparu.

L'instruction CLS a deux rôles : utilisée seule, elle efface l'écran sans en modifier les couleurs ; suivie d'une chaîne de caractères, elle efface l'écran et définit la couleur des caractères, la couleur du fond et la couleur du pourtour de l'écran.

La chaîne doit comporter deux ou trois caractères qui sont dans l'ordre, la couleur des caractères, du fond et du pourtour. Si deux caractères seulement existent, le pourtour garde sa couleur précédente.
Les couleurs sont notées en langue anglaise abrégée : R pour rouge, G pour vert, M pour magenta ou mauve, C pour cyan ou bleu ciel, Y pour jaune, b (attention c'est un b minuscule !) pour bleu foncé, W pour blanc et B (majuscule) pour noir.

Pour ramener l'écran aux couleurs initiales, vous pouvez faire un RESET ou un CLS "BCC".

**Perdons du temps :**

L'instruction PAUSE permet d'arrêter l'exécution d'un programme pendant un certain temps ; pour le constater faites :

```
100 PAUSE 10
110 PRINT "LA PAUSE EST FINIE"
```

et lancez le programme, le message n'apparaîtra que 10 secondes après environ. Lancez à nouveau le programme et, avant que les 10 secondes ne soient écoulées, frappez retour chariot ; le message apparaît aussitôt. Changez la ligne 10 en PAUSE -10 et relancez le programme, si vous ne faites rien, le message apparaît à nouveau au bout de 10 secondes ; par contre même si vous frappez le retour chariot le message n'apparaît pas plus vite.

PAUSE suivi par une expression numérique arrête le programme pendant N secondes si N est la valeur de l'expression. Si N est positif, la pause peut être interrompue par la frappe d'un retour chariot ; si N est négatif, la pause ne peut être interrompue. Si N est inférieur à 0,1 la pause n'est pas exécutée.

**Arrêtons-nous :**

Nous l'avons déjà utilisée sans l'avoir présentée au préalable tellement son sens est évident ; il s'agit de l'instruction STOP. Lorsqu'un programme arrive sur une ligne contenant STOP il s'arrête.
Cette instruction peut être intervertie avec END dont elle a les mêmes fonctions.

**Un ordinateur qui a de la voix :**

L'EXL 100 est équipé d'un synthétiseur sonore capable de produire de très nombreux sons et même la voix humaine comme vous pourrez le constater avec certains jeux commercialisés (Tennis, Wizzord, Imagix, etc...).
Pour vos propres programmes, il est possible d'utiliser ce synthétiseur au moyen d'un CALL SPEECH.
Ce CALL SPEECH s'utilise de la façon suivante :
- CALL SPEECH ("L,chaîne de caractères") pour faire fonctionner le synthétiseur.



- CALL SPEECH ("A", chaîne de caractères") pour arrêter le synthétiseur.
- CALL SPEECH ("R, chaîne de caractères") pour faire répéter sans arrêt le même son.
Le son lui-même est choisi par la chaîne de caractères dont quelques exemples sont donnés en annexe en fin de ce manuel. Plus le son est complexe, plus la chaîne doit être longue car cette chaîne est en fait une succession de codes hexadécimaux correspondants à la "description» du son pour le synthétiseur.
Pour faire un bruit de "pistolet galactique" essayez :

```
100 A$ = "0D9816C301EDE72CB8B4CBECBD7D8
8AA6F33A2DD41AAF6BDB46407AFDAE3"
110 B$="CEAA1EAA6AB7BE4A7CB86ADC6D37
C9E04BE87411B508C044302FE"
120 CALL SPEECH ("L,"&A$&B$)
```

Le bruit est émis à chaque lancement du programme. Si vous changez le L de la ligne 120 en R, le son se répète sans arrêt jusqu'à ce que vous exécutiez un nouveau CALL SPEECH avec l'ordre d'arrêt.

## CHAPITRE 11

# Lorsque l'ordinateur prend des décisions

**Les opérateurs de comparaison :**

Jusqu'à présent, nous avons vu que l'ordinateur pouvait travailler sur des chiffres ou des lettres mais il peut

aussi comparer des données et savoir si le résultat de sa comparaison est vrai ou faux. Pour identifier le vrai du faux dans sa mémoire, l'ordinateur utilise une représentation numérique : un -1 veut dire vrai et un 0 faux. Faites l'essai suivant :

PRINT 4 >3 qui est, bien sûr, vrai ;

Votre EXL 100 va répondre -1 puisque PRINT lui demande d'afficher le résultat de la comparaison. Si vous faites maintenant :

PRINT 5 <2

vous verrez 0 puisque c'est faux.

Le Basic dispose de plusieurs OPERATEURS DE COMPARAISON ou de RELATION qui sont :
= pour l'égalité
< > pour la différence
> pour supérieur à (A > B signifie A supérieur à B)
< pour inférieur à (A < B signifie A inférieur à B)
> = pour supérieur ou égal à (A > = B signifie A supérieur ou égal à B)
< = pour inférieur ou égal à (A < B signifie A inférieur ou égal ou à B).

L'utilisation d'un quelconque de ces opérateurs positionne une variable interne à -1 ou 0 selon que le résultat de la comparaison est vrai ou faux.
Ce résultat peut être utilisé tel quel mais il est plus souvent exploité par une instruction de prise de décision décrite ci-après.

Attention à une représentation singulière qui peut poser des problèmes comme dans le cas suivant :

TEST = 10 = 20

Cette expression signifie (revoyez si nécessaire le sens du signe égal expliqué au chapitre 4) : je donne à la variable TEST la valeur du résultat de la comparaison de 10 et 20 par le signe = ; dans ce cas TEST vaudra 0 car 10 n'est pas égal à 20.



Les opérateurs de relation peuvent aussi agir sur des variables chaînes de caractères et l'on peut ainsi écrire A$ = "OUI" pour tester si la variable chaîne A$ est égale à la chaîne de caractères OUI.

**L'instruction IF THEN ELSE:**

Cette instruction permet à l'ordinateur de prendre une décision en fonction d'une condition. Essayez le programme suivant :

```
100 INPUT N
110 IF N > 10 THEN GOTO 140
120 PRINT "N INFERIEUR OU EGAL A 10"
130 STOP
140 PRINT "N PLUS GRAND QUE 10"
```

et exécutez le programme ; sa première ligne lui fait attendre un nombre que vous allez frapper inférieur à 10. Vous allez alors voir s'afficher N INFERIEUR OU EGAL A 10 ; la ligne 120 a donc été exécutée. Faites un RUN à nouveau et frappez un nombre supérieur à 10 ; vous allez voir le message N PLUS GRAND QUE 10 apparaître, preuve que la ligne 120 ne s'est pas exécutée mais que 140 a eu lieu.

L'ordinateur a donc pris une décision, au niveau de la ligne 110, sur ce qu'il devait faire en fonction de la comparaison de N à 10 grâce à une instruction IF THEN qui fonctionne de la façon suivante.

L'instruction IF THEN s'utilise sous la forme IF condition THEN action. Si la condition est vraie, l'action qui suit le THEN est exécutée. Si la condition est fausse, l'action est ignorée et le programme continue à la ligne suivante.

Vous pouvez vérifier sur notre exemple que c'est bien ce qui s'est passé.

Comme cela ne suffit pas toujours, il est aussi possible d'écrire IF condition THEN action n°1 ELSE action n° 2. Dans ce cas, si la condition est vraie l'action n° 1 est exécutée puis le programme passe à la ligne suivante, si la condition est fausse, l'action n° 2 est exécutée et le programme passe ensuite à la ligne suivante.

Il faut faire attention, lors de l'utilisation du IF THEN, à ce que l'on utilise comme action après le THEN ; en effet, écrivez le programme suivant :

```
100 INPUT N
110 IF N > 10 THEN PRINT "N SUPERIEUR A 10"
120 PRINT "N INFERIEUR OU EGAL A 10"
```

Si vous frappez un nombre inférieur à 10, tout va bien se passer car comme la condition n'est pas vérifiée, l'action placée après le THEN ne va pas avoir lieu ; par contre si votre nombre est supérieur à 10, l'action située après le THEN va avoir lieu mais une fois celle-ci terminée, le programme va passer à la ligne suivante et vous afficher le deuxième message.
Pour éviter ce genre de problème, l'action placée après le THEN est assez souvent un GOTO

Comme sur la majorité des Basic, celui de l'EXL 100 autorise l'omission du GOTO après le THEN ou le ELSE et :
If condition THEN GOTO N peut être écrit IF condition THEN N ; de même IF condition THEN XXX ELSE GOTO N peut être écrit IF condition THEN XXX ELSE N.

Nous vous avons dit que les comparaisons fonctionnaient sur les chaînes de caractères, en voici un exemple que l'on retrouve souvent en fin de programme (les numéros de lignes sont quelconques) :

```
1000 INPUT "UNE AUTRE PARTIE ? ";A$
1010 IF A$ = "OUI" THEN GOTO 10 (si le programme commence en 10)
1020 STOP
```

Vous devez avoir compris sans peine le fonctionnement.

**Encore des opérateurs :**

Il arrive souvent que l'on ait besoin de tester plusieurs conditions simultanément dans un IF THEN. Pour ce faire, il est possible d'écrire plusieurs IF THEN à la suite les uns des autres mais c'est long et peu pratique.



Reprenons l'exemple de la fin de programme vu ci-avant et supposons que l'on veuille autoriser comme réponse OUI ou O. On peut écrire :

```
1000 INPUT "UNE AUTRE PARTIE ? ";A$
1010 IF A$ = "OUI" THEN GOTO 10
1020 IF A$ = "O" THEN GOTO 10
1030 STOP
```

De cette façon, si l'on répond OUI la ligne 1010 nous fera revenir en début de programme et si l'on répond O, la ligne 1010 sera sans effet mais la ligne 1020 détectera le O et fera revenir en début de programme.

Il existe une solution plus élégante et plus efficace qui consiste à utiliser des OPERATEURS LOGIQUES. Ces opérateurs sont quatre : OR, AND, XOR et NOT qui signifient respectivement OU, ET, OU EXCLUSIF et NON. Ils remplissent exactement les fonctions logiques qui leur correspondent mais nous allons voir qu'il suffit d'un peu de bon sens pour comprendre leur principe.

Dans l'exemple que nous avons pris, nous voulons pouvoir répondre à la question par OUI OU par O nous allons donc écrire :

```
1000 INPUT "UNE AUTRE PARTIE ? ";A$
1010 IF A$ = "OUI" OR A$ = "O" THEN GOTO 10
1020 STOP
```

La ligne 1010 veut bien dire si A$ est égal à OUI OU si A$ est égal à O, aller en 10.

L'opérateur ET s'utilise lorsque l'on veut que deux conditions soient vraies simultanément. Supposons que dans un programme nous devions effectuer un branchement si et seulement si deux variables dépassent des limites pré-définies ; on peut écrire (les valeurs et numéros de ligne sont quelconques) :

```
100 IF A > 100 AND B < 10 THEN GOTO 1000.
```

Dans ce cas, il faudra que simultanément A soit supérieur à 100 et B soit inférieur à 10 pour que le programme aille en 100. Si l'une seulement des conditions est réalisée, le programme continuera son exécution à la ligne suivante.

Lorsque deux conditions sont reliées par l'opérateur logique OR (ou), il suffit que l'une OU l'autre des conditions soit réalisée. Lorsque deux conditions sont reliées par l'opérateur logique AND (et) il faut que les deux conditions soient réalisées simultanément.

Le OU ainsi spécifié est le OU dit inclusif c'est-à-dire qu'il signifie l'un OU l'autre OU Les deux à la fois. Il existe aussi le OU EXCLUSIF noté XOR qui signifie l'un OU l'autre mais pas les deux à la fois. Il s'emploie comme OR et AND lorsque c'est nécessaire.

Le dernier opérateur logique est NOT, d'un emploi assez rare, et qui inverse la signification d'une condition. Pour le comprendre faites :

PRINT 3 > 4 ; vous obtenez 0 puisque votre comparaison est fausse ; faites alors :
PRINT NOT ( 3 > 4 ) ; vous obtenez -1 puisque vous demandez d'afficher le contraire du résultat de la comparaison. Remarquez les parenthèses autour de 3 > 4 pour que NOT s'applique bien au résultat de la comparaison.

CHAPITRE 12

# Les fonctions scientifiques

## La constante PI :

Vous connaissez tous la lettre grecque PI qui permet, entre autres choses, le calcul de la surface ou de la circonférence d'un cercle. Certains Basic ne disposent pas de sa valeur en mémoire et il faut la frapper lors de chaque utilisation. Votre EXL 100 par contre propose PI qui peut être utilisé directement sous cette forme dans



les calculs ; ainsi pour calculer la circonférence d'un cercle, vous pouvez faire :

```
100 INPUT "RAYON DU CERCLE ? ";R
110 PRINT "LA CIRCONFERENCE EST :";2 * PI * R
```

On peut difficilement faire plus simple. le PI conservé en mémoire de l'EXL 100 comporte 11 décimales et est égal à 3.14159265359.

## Fonctions trigonométriques

Comme toute calculatrice scientifique qui se respecte, un ordinateur travaillant en Basic dispose d'un grand nombre de fonctions scientifiques. Parmi celles-ci, se trouvent les fonctions trigonométriques classiques : SIN pour sinus, COS pour cosinus et TAN pour la tangente. La cotangente n'existe pas directement puisqu'il suffit de faire 1 / TAN pour l'obtenir.

Ces fonctions travaillent avec des angles exprimés en radians, degrés ou grades selon l'unité choisie. Cette unité est rappelée en permanence sur la ligne de service et peut être sélectionnée à tout instant, en mode direct ou dans un programme, par DEG pour les degrés, RAD pour les radians et GRAD pour les grades. A l'initialisation et après un RESET, les radians sont automatiquement sélectionnés.

Si vous voulez réaliser une table de fonctions trigonométriques bien connue de tout lycéen qui se respecte vous pouvez faire, par exemple :

```
100 DEG
110 PRINT "TABLE TRIGONOMETRIQUE"
120 PRINT
130 PRINT "SINUS","COSINUS"
140 PRINT
150 FOR I = 0 TO 90 STEP 5 (si vous la voulez de
5° en 5°)
160 PRINT SIN (I) , COS (I)
170 NEXT
```

C'est facile n'est-ce-pas ?

**Fonctions trigonométriques inverses :**

De même que l'on dispose des fonctions trigonométriques, il existe aussi leurs inverses avec :

ASN pour l'arc sinus, ACS pour l'arc cosinus et ATN pour l'arc tangente.

Le résultat des fonctions trigonométriques sinus et cosinus étant toujours compris entre -1 et +1, l'argument utilisé dans ASN et ACS doit être compris dans ces mêmes limites sinon une erreur est générée.

Si vous voulez vérifier la précision de votre Basic, vous pouvez essayer le petit exemple suivant :

```
100 DEG
110 FOR I = 0 TO 90
120 PRINT I,ASN(SIN(I))
130 NEXT
```

Attention ! De même que les fonctions trigonométriques travaillent avec des angles en radians, degrés ou grades, le résultat des fontions trigonométriques inverses est exprimé en radians, degrés ou grades, selon le choix d'unité que vous avez fait au préalable.

**Les fonctions logarithmes et exponentielle :**

Les fonctions logarithme naturel ou Népérien, logarithme décimal et exponentielle sont disponibles sur votre EXL 100 sous les noms LN, LOG et EXP ; ainsi :

PRINT LOG (2) donne le logarithme décimal de 2
PRINT LN (2) donne le logarithme naturel ou Népérien de 2
PRINT EXP (3) donne l'exponentielle de 3.

Les contraintes "mathématiques" des logarithmes doivent être respectées et dans LOG X, X doit être strictement positif sinon une erreur est générée.

**Les fonctions scientifiques diverses :**

Cinq fonctions sont disponibles ; ce sont :



- L'élévation à une puissance notée ↑ . Ainsi 4 élevé à la puissance 5 se note 4 ↑ 5. Attention ! cette fonction utilise, en interne, les logarithmes ; le nombre élevée à la puissance choisie doit donc toujours être strictement positif. Pour les puissances entières de petites tailles (carrés, cubes) la multiplication est plus précise, plus rapide et accepte les nombres négatifs ; il vaut mieux faire 5 * 5 * 5 pour élever 5 au cube que 5 ↑ 3.

- La valeur absolue d'un nombre notée ABS. Ainsi ABS (expression numérique) donne la valeur absolue de cette expression.

- La fonction signe d'un nombre notée SGN. Ainsi SGN (expression numérique) donne comme résultat -1 si l'expression est négative, 0 si elle est nulle et +1 si elle est positive.

- La fonction partie entière d'un nombre notée INT. Ainsi INT (expression numérique) donne la partie entière de l'expression, c'est-à-dire l'entier immédiatement inférieur. Attention à un piège classique : si INT (3.4) vaut 3 ce qui est logique,
INT (-3.4) vaut -4 car l'entier immédiatement inférieur à -3.4 est -4 !

La fonction racine carrée est notée SQR. Ainsi SQR (expression numérique) donne la racine carrée de l'expression. Comme on l'apprend en mathématiques, l'expression doit être positive ou nulle.

**Hasard et informatique :**

Un ordinateur est une machine logique qui ne sait faire que des calculs exacts. Pourtant, dans de nombreuses applications, il est nécessaire de faire intervenir le hasard si vous voulez faire un programme de tirage des résultats du Loto par exemple ou si vous voulez faire un jeu de dés. Pour cela existent trois instructions qui permettent de fournir des nombres dits PSEUDO ALEATOIRES que nous pouvons considérer comme étant tirés au hasard. En fait, ils sont obtenus avec un calcul mathématique tellement compliqué qu'il est absolument impossible de les prévoir et que, de plus, leur réparti-

tion est conforme aux règles habituelles des probabilités.

La première instruction est RND et fournit, à chaque utilisation, un nombre aléatoire supérieur ou égal à 0 et strictement inférieur à 1. Pour l'essayer faites :

```
100 FOR I = 1 TO 10
110 PRINT RND
120 NEXT
```

Vous allez voir s'afficher 10 nombres compris entre 0 et 1 absolument quelconques.
Si vous refaites tourner ce programme, vous allez encore voir 10 nombres quelconques différents des précédents.

Nous allons maintenant faire appel à la deuxième possibilité liée à la fonction RND qui est l'instruction RANDOMIZE. Elle s'utilise en mode direct ou dans un programme et permet de fixer un paramètre utilisé par la fonction RND pour fournir les nombres aléatoires. Si vous faites RANDOMIZE, les nombres générés par RND seront vraiment quelconques. Si, par contre, vous faites RANDOMIZE N ou N est compris entre 1 et 65535, vous allez fixer une variable interne. L'instruction RND générera toujours des nombres aléatoires mais, chaque fois que vous ferez RANDOMIZE N et que vous utiliserez ensuite RND, vous obtiendrez les mêmes nombres aléatoires. Réalisez l'expérience suivante et vous comprendrez :

```
100 INPUT "FRAPPEZ UN NOMBRE ENTIER : ";N
110 RANDOMIZE N
120 FOR I = 1 TO 10
130 PRINT I , RND
140 NEXT
150 GOTO 100
```

Faites tourner le programme plusieurs fois et notez les nombres sur un papier ; vous constatez que les dix nombres affichés sont quelconques mais chaque fois que vous frappez le même nombre en réponse à la ligne 100, vous obtenez à nouveau la même séquence.
Cette possibilité permet de faire des programmes qui trichent ...



Comme des nombres décimaux entre 0 et 1 ne sont pas toujours très pratiques et plutôt que d'utiliser la formule de conversion que l'on emploie sur certains Basic, votre EXL 100 dispose d'une troisième instruction de hasard qui est INTRND. Elle fonctionne comme RND mais génère un nombre entier compris entre 1 et l'entier qui suit INTRND inclus. Ainsi pour simuler un jet de dé, pouvez-vous écrire :

```
100 PRINT "FRAPPEZ UNE TOUCHE POUR LANCER LE DE"
110 CALL KEY1(A,B)
120 IF A = 255 THEN 110
130 PRINT INTRND (6)
140 GOTO 100
```

Attention, du fait du fonctionnement de KEY1, si vous maintenez votre doigt sur une touche trop longtemps, vous allez jeter plusieurs fois le dé ; si vous voulez éviter cela, il faut tester aussi la variable B ce que nous vous laissons le soin d'écrire.

## CHAPITRE 13

# L'EXL 100 dessinateur

### Le mode haute résolution :

Jusqu'à présent nous n'avons utilisé l'EXL 100 que pour afficher du texte ; cela s'appelle le mode ou affichage ALPHANUMERIQUE mais ses possibilités ne s'arrêtent pas là car votre appareil peut aussi faire des graphiques de très haute qualité grâce à un mode GRAPHIQUE HAUTE RESOLUTION.
Lorsqu'un ordinateur est en mode alphanumérique, on peut afficher des caractères prédéfinis dans l'appareil au

moyen de diverses instructions. En mode graphique, on a accès à tous les points de l'écran et il faut alors les allumer ou les éteindre pour faire soi-même des dessins. Nous allons voir qu'avec quelques instructions particulières c'est très simple.

La première chose à faire est de mettre un morceau de l'écran en mode haute résolution ; en effet, à la mise sous tension, tout l'écran de votre EXL 100 est en mode alphanumérique. Vous avez le droit de mettre au maximum 20 lignes d'écran en mode haute résolution mais vous pouvez choisir où placer cette portion haute résolution sur l'écran. Pour cela, essayez :

CALL HRON ("R",1,11)

Vous voyez apparaître un rectangle rouge sous la ligne de service. Vous avez mis en haute résolution 11 lignes, à partir de la ligne 1 et vous les avez initialisées en rouge. Essayez maintenant :

CALL HRON ("Y",5,6)

Vous voyez apparaître un rectangle jaune à partir de la 5ème ligne d'écran et qui s'étale sur l'équivalent de 6 lignes. cela doit vous avoir permis de comprendre que HRON ("C",N,M) place en mode haute résolution M lignes à partir de la ligne N et les initialise de la couleur C. Le codage des couleurs est celui décrit pour l'instruction CLS (chapitre 10).

Attention ! il est interdit de mettre plus de 20 lignes en haute résolution et de placer en haute résolution des lignes situées au delà de la ligne 20. cela vous permet de conserver une fenêtre alphanumérique en bas d'écran où vous pouvez continuer à travailler comme vous en avez l'habitude jusqu'à maintenant.

Si vous voulez récupérer tout votre écran en alphanumérique après avoir utilisé le mode haute résolution, il vous suffit de faire un CALL HROFF mais attention, tous les dessins que vous aviez pu faire dans la partie haute résolution de l'écran sont définitivement perdus.



**Allumez des points :**

Une fois que vous avez passé une partie de votre écran en haute résolution, il faut pouvoir y allumer ou éteindre des points. Pour cela, vous disposez de l'instruction PLOT.
Mettez 11 lignes de votre écran en haute résolution et en jaune avec :

CALL HRON ("Y",1,11)

Nous allons maintenant allumer des points dans cette partie ; essayez :

CALL PLOT ("R",160,50)

Vous allez voir un point rouge (attention il est petit !) au centre de votre zone haute résolution.

Vous avez, grâce à PLOT, défini un point par : sa couleur grâce au caractère qui suit le CALL PLOT, et ses coordonnées par les deux chiffres qui suivent. Les coordonnées sont sous la forme colonne, ligne sachant qu'en mode haute résolution vous disposez de 320 colonnes repérées de 0 à 319 et de (10 x N)-1 lignes ou N est le nombre de lignes mises en haute résolution.
Le point de coordonnées 0,0 est situé en haut et à gauche de l'écran.

Nous venons, avec notre exemple, d'allumer un point rouge. Pour l'éteindre, il n'existe pas d'instruction spéciale ; c'est encore PLOT qu'il faut employer mais en "allumant" un point de même couleur que le fond. Essayez :

CALL PLOT ("Y",160,50)

et votre point rouge disparaît puisque vous avez défini à sa place un point jaune comme le fond.
Bien sûr, vous avez compris que le codage des couleurs dans PLOT est le même que dans HRON et dans CLS ; un ordinateur aime la logique !

Cette instruction PLOT Permet de faire des graphiques très facilement ; supposons que nous souhaitions tra-

cer une sinusoïde dans notre écran haute résolution jaune ; il suffit de faire :

```
100 DEG
110 FOR I = 0 TO 319
120 J = I * 4
130 CALL PLOT ("R",I,50 - 50 *SIN(J) )
140 NEXT
```

Pour voir apparaître un peu plus de trois périodes de sinusoïde rouge sur fond jaune. Remarquez le 50 - 50 * SIN qui permet de centrer la sinusoïde dans la portion d'écran disponible (sachant qu'un sinus est toujours compris entre -1 et +1).

Si votre tracé ne vous plaît pas sans axes, vous pouvez très simplement ajouter ceux-ci de la façon suivante :

```
100 DEG
110 FOR I = O TO 319
120 CALL PLOT ("M",I,50)
130 NEXT
140 FOR I = 0 TO 100
150 CALL PLOT ("M",0,I)
160 NEXT
170 FOR I = 0 TO 319
180 J = I * 4
190 CALL PLOT ("R",I,50 - 50 * SIN(J))
200 NEXT
```

l'axe des x est tracé par les lignes 110 à 130 et celui des Y par les lignes 140 à 160

### Une seule instruction pour une droite :

Bien que notre programme précédent fonctionne, vous avez pu remarquer que le tracé des axes était long et coûteux en nombre d'instructions ; pour éviter cela une instruction spéciale est disponible. Elle permet de tracer une droite dont on précise origine et extrémité. C'est plus rapide, plus simple et plus puissant que l'utilisation de PLOT, en effet on peut tracer de cette façon des obliques !
Cette instruction s'appelle LINE et nous allons l'essayer ; faites :



```
100 CALL HRON ("W",1,11)
110 CALL LINE ("M",0,0,319,109)
```

Vous voyez apparaître instantanément une ligne mauve en diagonale de votre zone haute résolution. LINE est donc beaucoup plus rapide qu'un PLOT dans une boucle et doit être utilisé toutes les fois que c'est possible.

La syntaxe de LINE est simple ; la lettre qui suit codifie la couleur du tracé puis viennent les coordonnées du point origine (même principe et valeurs limites que pour PLOT) et ensuite les coordonnées du point extrémité (même principe et valeurs limites que pour PLOT).

Attention à un piège du mode haute résolution : si votre écran haute résolution n'est pas assez grand pour accueillir le tracé demandé, les coordonnées qui apparaissent dans PLOT et LINE seront réduites modulo la taille de l'écran haute résolution ; cela peut parfois conduire à des résultats curieux.

### A cheval entre graphique et alphanumérique :

Outre ces possibilités graphiques et les possibilités alphanumériques que nous avons utilisées jusqu'à maintenant, votre EXL 100 dispose d'un troisième mode de fontionnement que l'on peut qualifier d'intermédiaire entre les deux en étant toutefois plus proche de l'alphanumérique.

Ce mode vous permet d'écrire de la couleur de votre choix (et ce indépendamment de la couleur de caractères définie par un éventuel CLS préalable), de choisir la taille des caractères parmi quatre, d'inverser la couleur du fond et celle des caractères et enfin de faire clignoter des caractères. Tout cela utilise une seule instruction : COLOR, de la façon suivante : COLOR (chaîne de caractères).
Le contenu de la chaîne précise de nombreuses informations détaillées dans la fiche de référence de cette instruction en troisième partie de ce manuel que nous vous demandons de lire avant de continuer ce paragraphe (afin de ne pas répéter ici ce qui est écrit dans la fiche).
Vous avez lu, alors essayez :

100 CALL HROFF au cas où votre écran serait encore en haute résolution
100 CALL COLOR ("0RGL")
100 PRINT "BBOONNJJOOUURR"

Vous voyez s'afficher BONJOUR en rouge sur fond vert. C'est normal, 0 a sélectionné le générateur de caractères standard (nous allons voir dans le paragraphe suivant ce que c'est), R des caractères rouges et V un fond vert. Essayez maintenant :

100 CALL COLOR ("0RGL")
100 PRINT "BBOONNJJOOUURR"

vous voyez s'afficher BONJOUR en rouge et en double largeur. Par contre il n'y a pas de fond vert; c'est normal, la fiche de COLOR vous indique que dans les modes doubles, le fond ne peut être choisi. Remarquez que les caractères ont été doublés dans le PRINT; en effet l'écriture en double largeur occupe deux places de caractères "normaux" pour un caractère; il faut donc occuper ces deux places avec le même caractère qui sera ainsi représenté en deux fois plus gros.

Essayez maintenant :
100 CALL COLOR ("0RLF")
110 PRINT "BBOONNJJOOUURR"

BONJOUR apparaît en double largeur mais, en plus, clignote à cause du F.

Pour la double hauteur c'est un petit peu plus compliqué ; en effet, comme pour la double largeur, les caractères occupent aussi deux positions mais elles ne sont pas l'une à côté de l'autre mais l'une au-dessous de l'autre ; il faut utiliser LOCATE de la façon suivante :
100 CLS
110 LOCATE (2,1)
120 CALL COLOR ("0RH")
130 PRINT "BONJOUR"
140 LOCATE (3,1)
150 PRINT "BONJOUR"

Pour voir BONJOUR en double hauteur. Les LOCATE ont permis de positionner les deux



PRINT "BONJOUR" l'un au-dessous de l'autre.

Si maintenant vous voulez faire de la double largeur, double hauteur, il suffit de combiner ces deux programmes de la façon suivante :

100 CLS
110 LOCATE (2,1)
120 CALL COLOR ("0RHL")
130 PRINT "BBOONNJJOOUURR"
140 LOCATE (3,1)
150 PRINT "BBOONNJJOOUURR"

et le tour est joué.

Attention ! Lorsqu'une définition de générateur de caractères, de couleur et de taille a été faite par COLOR dans un programme, elle reste valable pour tous les affichages qui ont lieu dans la suite de ce programme jusqu'à ce qu'une nouvelle instruction COLOR vienne les modifier.

**Définissez vos propres caractères :**

En présentant l'instruction COLOR nous avons parlé de générateurs de caractères et de fait, la chaîne qui suit COLOR peut commencer par 0 pour sélectionner le générateur standard, celui qui contient les caractères qui correspondent au code ASCII et à ses extensions dont nous avons parlé au chapitre 6. Mais cette chaîne peut aussi commencer par 1 et l'on a alors accès au générateur de caractères utilisateur dans lequel vous pouvez définir jusqu'à 127 caractères qui vous seront propres.
Lors d'un RESET, le générateur utilisateur est identique au générateur standard ce qui fait que vous pouvez utiliser 0 ou 1 dans COLOR sans constater de différence. Par contre, vous pouvez ensuite définir vos propres caractères dans le générateur utilisateur au moyen de l'instruction CHAR.

Ici aussi, et puisque vous devez commencer à être de bons programmeurs, nous n'allons pas faire de longs développements car la fiche de l'instruction CHAR est très détaillée et comporte un bon exemple d'utilisation.

Lisez-la avant de continuer ce paragraphe.
C'est fait ? Vous avez dû y voir qu'avec CHAR on pouvait définir ce que l'on voulait puisque l'on dessine littéralement les caractères de son choix dans une matrice 8 sur 10. L'exemple choisi représentait d'ailleurs un petit bonhomme (un peu stylisé il est vrai !)
Le seul problème que vous pouvez rencontrer avec CHAR est la conversion des 1 et des 0 qui constituent chaque ligne de points en code HEXADECIMAL.
Si vous ne connaissez pas cette façon de compter, propre aux informaticiens, un tableau de conversion est là, en annexe de ce manuel, pour vous éviter toute recherche. Utilisez-le sans hésiter.

Attention ! les caractères que vous avez pu définir et placer dans le générateur utilisateur sont impitoyablement détruits par un RESET puisque ce dernier rend le générateur utilisateur identique au standard.

CHAPITRE 14

# Utilisation d'un magnétophone

### Un magnétophone pour quoi faire ?

Nous vous avons dit et fait vérifier dans les premiers chapitres que le fait de couper l'alimentation ou de faire RESET effaçait tout ce qui était contenu en mémoire. Cela veut dire que lorsque vous avez frappé un programme et que vous éteignez votre EXL 100 ou que vous faites un RESET, celui-ci est perdu.

Pour éviter cela, vous avez la possibilité d'enregistrer votre programme sur cassette et, lors d'une utilisation ultérieure de votre ordinateur, de relire cette cassette pour mettre à nouveau le programme en mémoire.



Cette façon de faire vous permet aussi d'acheter des programmes pour votre EXL 100 chez votre distributeur habituel et de CHARGER ceux-ci en mémoire lorsque vous voulez les utiliser.
C'est aussi un excellent moyen d'échanger des programmes entre amis, vous évitant ainsi d'avoir à recopier un listing de programme écrit sur papier.
Pour exploiter ce procédé, commencez par connecter un magnétophone à cassettes à votre EXL 100 comme expliqué dans la première partie de ce manuel.

### La commande SAVE :

Pour essayer le fonctionnement avec un magnétophone, frappez un programme quelconque mais comportant tout de même plusieurs lignes. Si vous manquez d'inspiration, reprenez un quelconque des exemples précédents.
Lorsque c'est fait, frappez :

SAVE "1.XXXX» où XXXX est le nom que vous voulez donner à votre programme. Ce nom doit comporter au maximum 4 caractères et au minimum 1 caractère.
Le message ESC → Enr. apparaît alors sur la ligne de service ; il signifie que l'enregistrement ne comencera que lorsque vous frapperez la touche ESC. Cela vous permet de mettre en marche votre magnétophone, d'attendre que la bande amorce de la cassette soit passée car il est impossible d'enregistrer sur cette partie puis de n'actionner ESC que lorsque tout est prêt ; le message se transforme alors en Enr. XXXX où XXXX est le nom que vous avez donné au programme. Simultanément, le son enregistré sur cassette est reproduit par le haut parleur du récepteur TV.
Lorsque c'est terminé, le curseur réapparaît et vous pouvez alors faire un LIST pour constater que votre programme est toujours en mémoire ; en effet, la commande SAVE SAUVEGARDE le programme sur cassette mais ne modifie absolument pas le contenu de la mémoire.

### La commande LOAD :

C'est l'inverse de SAVE c'est-à-dire qu'elle permet de charger en mémoire un programme qui se trouve sur cassette.

Pour essayer LOAD, commencez par effacer de la mémoire de votre EXL 100 le programme qu'elle contient en faisant NEW. Frappez ensuite LOAD "1.XXX» où XXXX est exactement identique au nom que vous avez utilisé lors du SAVE. Le message ESC → Lect. apparaît alors sur la ligne de service indiquant que la lecture ne commencera qu'après avoir appuyé sur ESC. Rebobinez votre cassette, mettez votre magnétophone en lecture et frappez ESC ; le message devient Lect. XXXX → et dès que le programme est trouvé, son nom apparaît devant la flèche. Il se charge alors en mémoire et le curseur réapparaît lorsque c'est terminé. Vous pouvez alors faire un LIST pour constater que tout s'est bien passé.

Si le signal délivré par votre magnétophone est insuffisant, le chargement s'arrête de lui-même au bout d'une minute et vous rend le curseur. Si des erreurs sont détectées pendant le chargement (bande de mauvaise qualité, magnétophone au fonctionnement incorrect), un message est affiché et le chargement est interrompu. Autrement, il est impossible d'interrompre une opération de chargement sauf en faisant un RESET.

Si vous avez oublié le nom donné au programme que vous voulez charger, vous pouvez faire LOAD "1." et le premier programme rencontré sera chargé en mémoire. Son nom vous sera rappelé par la ligne de service sur laquelle il apparaît comme expliqué ci-avant.

Si vous avez plusieurs programmes à la suite sur une même cassette, l'utilisation de LOAD suivi par un nom de programme fait ignorer tous les programmes ne portant pas ce nom pour ne charger que le bon.

### Si ça ne marche pas :

Il peut arriver qu'il vous soit impossible de charger en mémoire un programme sauvegardé sur cassette. Si un tel phénomène se produit, il faut faire les quelques vérifications suivantes parmi lesquelles se trouve certainement la solution du problème.



La première vérification à faire consiste à écouter la cassette sur laquelle vous avez sauvegardé un programme ; vous devez entendre sur le haut parleur du magnétophone le même son que celui que vous entendiez par le haut parleur du récepteur TV lors de l'exécution de la commande SAVE. Le niveau sonore doit être normal c'est-à-dire identique à celui que vous obtenez avec des cassettes de musique par exemple.
Si vous n'entendez rien sur la cassette, vous avez certainement fait une erreur de branchement de votre cordon de raccordement (inversion des jacks par exemple ou utilisation d'un cordon DIN "croisé").
Si le son reproduit est très faible, vous avez peut-être utilisé la mauvaise entrée du magnétophone (il faut utiliser l'entrée "micro", voir chapitre 1) ou les têtes de votre magnétophone sont très encrassées. Si tel est le cas, nettoyez-les avec un coton imbibé d'alcool à 90° à l'exclusion de tout autre produit qui pourrait attaquer les pièces en plastique du magnétophone ; pour faire ce travail facilement, utilisez un "coton tige" que l'on trouve en pharmacie pour se nettoyer les oreilles...

Si vous entendez un son normal sur la cassette, le problème se situe ailleurs. Deux cas peuvent se produire, l'EXL 100 n'arrive rien à lire sur la cassette ce qui se vérifie par l'absence d'affichage du titre du programme sur la ligne de service et l'apparition d'un message d'erreur Err 0,255 ⇒1 ; ou l'EXL 100 arrive bien à lire quelque chose mais avec des erreurs, le message Err 0,16 ⇒1 apparaît alors ou, si ce message n'apparaît pas, le listing du programme chargé en mémoire est incomplet ou comporte des fautes.

Dans les deux cas il faut commencer par modifier la position du potentiomètre de volume du magnétophone vers le haut ou vers le bas et refaire des essais de chargement jusqu'à obtenir un fonctionnement satisfaisant. Normalement, et sur la majorité des magnétophones du commerce, le bouton de réglage du volume doit être placé vers les 3/4 de sa course environ.
Si vous ne trouvez aucune position qui convient, essayez un autre magnétophone (faites-vous le prêter par un ami) car le vôtre peut avoir des problèmes de vitesse

d'entraînement de la cassette, surtout si il est ancien (car des courroies internes s'usent) ou si vous l'avez utilisé un peu partout et que de la poussière s'est introduite à l'intérieur.
Essayez aussi d'éloigner le magnétophone du récepteur TV; en effet certains postes génèrent un champ magnétique important qui peut perturber le fonctionnement du magnétophone.

Si aucun des essais décrits ci-avant ne donnent de résultat, consultez votre revendeur qui sera à même de vous conseiller et de vous présenter des magnétophones qui fonctionnent correctement.

**Quelques conseils :**

Sur une face de cassette peuvent rentrer une centaine de programmes courts et on peut donc être tenté d'utiliser au maximum la place disponible.
Ce n'est pas une bonne méthode car cela conduit à des erreurs et à des temps de recherche très longs pour les programmes qui sont en fin de cassette.

Mieux vaut prendre l'habitude de mettre un ou deux programmes par face, pas plus.

Prenez aussi l'habitude de repérer soigneusement vos programmes en marquant sur les cassettes et sur leurs boîtes les noms de ceux-ci.

Pour les cassettes, il est inutile d'acheter des modèles Hi Fi ou au chrome ou encore des cassettes métal. Contentez-vous de bonnes cassettes au fer appelées aussi Ferro ou Ferro dynamiques. Evitez les cassettes vendues en promotion sous des marques inconnues qui posent souvent de gros problèmes. Choisissez toujours des cassettes C 60, les cassettes de durée plus élevée C 90 ou C 120 ont une bande plus fine qui peut patiner dans le magnétophone et provoquer des erreurs de chargement.

Enfin, gardez toujours propres les têtes de votre magnétophone en les nettoyant régulièrement avec de l'alcool à 90° et un petit coton. N'utilisez surtour pas de solvant tel que le trichloréthylène.



Comme pour des enregistrements musicaux, prenez soin de vos cassettes ; ne les exposez pas à la chaleur, à l'humidité ni surtout aux champs magnétiques. Ces derniers sont produits par tout appareil comportant un transformateur ou un haut parleur. Ne posez donc pas vos cassettes sur votre EXL 100 ni trop près du récepteur TV.

**On peut aussi mettre des données sur cassettes :**

Nous vous avons montré, au début de ce paragraphe, que l'on pouvait sauvegarder sur cassette des programmes. Il est également possible d'y mettre des données afin de réaliser des programmes type "carnet d'adresses" par exemple. Ces données peuvent ensuite être lues par le programme qui les a sauvegardées sur cassette mais aussi par n'importe quel autre programme.
Pour utiliser cette fonction il faut faire appel à l'instruction OPEN. OPEN permet d'affecter à un équipement physique (ici le magnétophone à cassettes) un numéro logique. Ensuite, les instructions PRINT et INPUT vont pouvoir utiliser ce numéro logique pour envoyer ou aller chercher leurs informations sur l'équipement physique qui lui a été associé par OPEN. Un exemple simple va clarifier tout cela.

Supposons que l'on veuille enregistrer des données sur cassette, on va procéder de la façon suivante :

```
100 OPEN #3,"1.DATA",OUTPUT
110 PRINT #3,liste de variables et de constantes
```

La ligne 100 ouvre le fichier baptisé DATA en écriture (à cause du OUTPUT qui suit) et lui affecte le numéro logique 3. La ligne 110 fait alors "afficher" sur le numéro logique 3 (c'est-à-dire notre fichier DATA) la liste de variables et de constantes qui suit. Nous allons faire un essai très simple au moyen de ce petit programme :

```
100 A=123
110 B=456
120 C=789
130 A$="EXL 100"
140 OPEN #3,"1.DATA",OUTPUT
150 PRINT #3,A;",";B;",";C;",";A$
```

Si vous lancez son exécution, vous allez voir apparaître le message que vous avez déjà vu pour un SAVE à savoir ESC -> Enr. qui signifie que vous devez mettre le magnétophone en enregistrement puis appuyer sur ESC. Les variables A, B, C et A$ sont alors sauvegardées sur cassette dans le fichier qui s'appelle 1.DATA. Remarquez la syntaxe très spéciale de la ligne de PRINT. En effet, les variables doivent être séparées dans le fichier par le code ASCII de la virgule ; il faut donc utiliser ce mode d'écriture pour y parvenir.
Pour relire les données ainsi sauvegardées il suffit de faire :

```
100        OPEN        =4,INPUT
110       INPUT        =4,A,B,C,A$
120       PRINT        A,B,C,A$
```

et le message ESC-> Lect. apparaîtra sur l'écran vous demandant de mettre le magnétophone en lecture (après l'avoir rebobiné bien sûr) et d'appuyer sur ESC. Les données sont alors lues par INPUT et affectées aux variables A, B, C, A$; le PRINT de la ligne 120 fait afficher celles-ci pour que vous puissiez constater que la lecture s'est bien passée.

Remarquez ici qu'OPEN est suivi par l'option INPUT ce qui signifie que le fichier est ouvert en lecture. Remarquez aussi que les variables lues par INPUT sont séparées par des virgules. Leurs noms peuvent être quelconques et ne sont pas forcément les mêmes que ceux utilisés lors de la sauvegarde; par contre, il faut absolument que les variables lues soient du même type que les données sauvegardées (un nom de variable numérique ne pourra pas lire une donnée chaîne de caractères).

Il reste encore deux choses à voir à propos de ces possibilités. Tout d'abord, si vous ne précisez rien de plus que INPUT ou OUTPUT après un OPEN, un enregistrement ne pourra comporter que 80 caractères au maximum; il existe donc une autre option que vous pouvez placer après INPUT ou le OUTPUT et qui s'écrit VARIABLE N où N est la nouvelle taille d'un enregistrement. Ainsi :

```
100 OPEN =4,"1.TOTO",OUTPUT,VARIABLE 1000
```



ouvre le fichier TOTO en écriture, lui affecte le numéro logique 4 et prévoit un enregistrement de 1 000 caractères.
Attention si l'option VARIABLE est utilisée lors de la sauvegarde d'un fichier de données, la même option avec le même nombre doit être utilisée lors de la re-lecture du fichier.

Le dernier point est celui de l'utilisation des boucles avec OPEN pour sauvegarder sur cassette une table par exemple. Voici un exemple d'utilisation qui parle mieux qu'un long discours :

```
100 OPEN =4,"1.TABL",OUTPUT
110 FOR I = 1 TO 9
120 PRINT =4,I;",";I↑2;",";
130 NEXT
```

Ce programme va sauvegarder dans le fichier TABL la table des nombres de 1 à 9 et de leurs carrés grâce à la boucle FOR - NEXT. Remarquez l'écriture du PRINT qui est identique à l'exemple sans boucle sauf qu'elle se termine par un ","; qui permet d'attendre le caractère suivant.

La lecture d'un tel fichier se fera au moyen d'une boucle de la façon suivante :

```
100 OPEN =4,"1.TABL",INPUT
110 FOR I =1 TO 9
120 INPUT =4,A(I),B(I),
130 PRINT A(I),B(I)
140 NEXT
```

Les nombres seront lus dans A(I) alors que leurs carrés seront lus dans B(I). Remarquez la virgule en fin de ligne INPUT afin de permettre la lecture séquentielle de tous les éléments du tableau.

Bien sûr, pour ces opérations comme pour toutes les autres, votre EXL 100 aime la logique et il y aura génération d'un message d'erreur si vous essayez de lire plus de données que n'en contient un fichier.

CHAPITRE 15

# Pour programmer comme un "pro"

## READ, DATA et RESTORE :

Ce trio d'instructions permet d'affecter des valeurs à des variables de façon quasi automatique et, en tout cas, totalement programmable. Faites l'essai suivant :

```
100 DATA 10,BONJOUR,"EXL 100",12345
110 READ A,B$,C$,D
120 PRINT A;D;B$;C$
```

vous allez voir s'afficher sur votre écran :
10 12345 BONJOUR EXL 100

Comme cet affichage vient de la ligne 120, cela signifie que les variables A, D, B$ et C$ ont pris les valeurs 10, 12345, BONJOUR et EXL 100, dans cet ordre. C'est effectivement ce qui s'est passé grâce aux instructions READ et DATA.
DATA permet de définir une suite de données quelconques et qui doivent seulement être séparées les unes des autres par des virgules. DATA est tellement tolérante que les chaînes de caractères peuvent être écrites sans guillemets (cas du BONJOUR ci-dessus) si elles ne comportent pas d'espace interne.
READ permet d'affecter à une ou plusieurs variables les données qui suivent les instructions DATA. La règle d'affectation est assez spéciale :
- La première variable du premier READ d'un programme se verra affecter la première donnée du premier DATA du même programme, QUELLE QUE SOIT LA PLACE DU DATA PAR RAPPORT AU READ.
- La deuxième donnée du premier READ se verra affecter la deuxième donnée du premier DATA et ainsi de suite.
- Lorsque le premier DATA est épuisé, le second DATA est lu et le processus continue.



- Si plusieurs READ ont lieu en des endroits divers, le deuxième READ commencera à lire les données des DATA à l'endroit où le READ précédent s'était arrêté.

Pour bien comprendre ce processus très logique, vous pouvez raisonner aussi de la façon suivante : toutes les données de tous les DATA sont rangées les unes derrière les autres en respectant l'ordre d'apparition des DATA. Toutes les variables de toutes les instructions READ sont aussi rangées les unes derriere les autres en respectant l'ordre d'exécution des READ.
Il ne reste plus qu'à placer l'un au-dessus de l'autre ces arrangements pour voir quelle donnée est affectée à quelle variable.

Ce bel ordonnancement est assez sujet à erreur (de votre part, pas de la part de l'EXL 100) ; il faut en effet toujours veiller à ce que les types de données que l'on trouve dans les DATA correspondent aux types de variables auxquelles elles vont se voir affectées. Il faut aussi veiller à ce qu'il y ait toujours au moins autant de données dans tous les DATA que de variables dans tous les READ sinon certaines variables ne pourront se voir affecter de valeurs. Par contre, il peut y avoir plus de données que de variables, ce n'est pas grave ; les données en trop sont ignorées.

Ce bel ordonnancement est parfois gênant dans certains programmes où l'on voudrait affecter plusieurs fois de suite à des variables différentes les mêmes données. Pour cela a été créée RESTORE. Essayez l'exemple suivant :

```
100 DATA 10 , 20 , 30 , 40
110 READ A,B
120 RESTORE
130 READ C,D
140 PRINT A,B,C,D
```

fait afficher 10 20 10 20 lors de l'exécution. L'instruction RESTORE a ordonné au READ suivant de reprendre la lecture des données au début des DATA.

Comme cette forme de RESTORE n'est pas très souple, il existe une autre possibilité avec RESTORE suivi d'un numéro de ligne ; essayez :

```
100 DATA 10 , 20 , 30 , 40
110 READ A,B
120 RESTORE 1000
130 READ C,D
140 PRINT A;B;C;D
1000 DATA 50 , 60 , 70
```

Vous voyez s'afficher 10 20 50 60 ; en effet, le RESTORE 1000 a donné l'ordre au READ qui suit de lire les données à partir de la première donnée de l'instruction DATA de la ligne 1000.

Il ne faudrait pas croire que READ, DATA et RESTORE ne peuvent s'utiliser que de cette façon. Une boucle peut aussi servir à lire toute une ligne de DATA pour les affecter successivement à la même variable. Essayez :

```
100 DATA 30 , 50 , 10 , 20 , 1000
110 FOR I = 1 TO 5
120 READ I
130 PRINT A;
140 NEXT
```

vous allez voir s'afficher 30 50 10 20 1000 car à chaque tour de boucle un READ A est exécuté et A se voit affecter tour à tour les diverses données contenues dans le DATA de la ligne 100.

Remplacer le TO 5 par TO 6, vous allez encore voir s'afficher la même chose pour les 5 premières lectures de A, mais la sixième fera générer un message d'erreur car toutes les données contenues après DATA auront déjà été lues.

### Exploitez vos erreurs :

Jusqu'à maintenant, nous avons parlé de messages d'erreur mais sans plus. Ces messages sont générés chaque fois que l'EXL 100 décèle une anomalie.



Il indique alors le type d'erreur grâce à un code numérique dont la signification vous est donnée en annexe en fin de ce manuel et il signale la ligne où l'erreur s'est manifestée.
Certaines erreurs sont fatales et entraînent un arrêt obligatoire du programme. D'autres ne le sont pas et permettent tout de même de continuer après réparation de l'erreur. Ainsi, si un INPUT attend un nombre et que vous frappiez une lettre, ce n'est pas grave, il suffit de «se débrouiller» pour refaire exécuter le INPUT pour que vous puissiez répondre comme il faut et que tout rentre dans lordre. Pour cela existent deux instructions : ON ERROR et ERR.
Ici encore, comme vous devenez de bons programmeurs, nous vous renvoyons aux fiches de ces instructions. Disons seulement que ON ERROR permet de modifier le comportement de l'EXL 100 en présence d'une erreur. Selon le type de ON ERROR que vous choisissez, le message d'erreur peut ne plus être affiché et certaines erreurs peuvent déclencher un sous programme qui vous proposera alors de rattraper votre erreur.
L'instruction ERR qui s'appelle par un CALL permet de connaître le numéro de code de l'erreur et la ligne où elle s'est produite. Ces informations sont généralement exploitées dans le sous programme déclenché par le ON ERROR pour corriger ou non l'erreur. Un exemple d'utilisation est donné dans la fiche de ON ERROR à laquelle vous pouvez vous reporter.

### Les points d'arrêt :

Lorsque vous allez écrire vos premiers programmes, il se peut qu'ils ne fassent pas tout à fait ce que vous auriez souhaité ; rassurez-vous, celà arrive tous les jours à des programmeurs chevronnés ! Pour cela nous avons vu dans un des premiers chapitres que l'on pouvait interrompre un programme au moyen de la commande BREAK (CTL C). Cela permet parfois de savoir où l'on est coincé mais ne suffit pas toujours.
Votre EXL 100 vous offre donc la possibilité de mettre dans un programme des POINTS D'ARRET ; en arri-

vant sur ces points d'arrêt, votre programme s'arrêtera (eh oui !) et vous pourrez alors faire afficher des variables par exemple pour essayer de comprendre ce qui ne va pas.
Pour mettre des points d'arrêt, vous pouvez :
- Soit les placer dans le programme lors de son écriture en écrivant BREAK à toutes les lignes où vous souhaitez le voir s'arrêter.
- Soit avant d'exécuter votre programme frapper BREAK (en toutes lettres) suivi d'un ou plusieurs numéros de lignes séparés par des virgules.
Dans les deux cas, le programme s'arrête toujours sur un point d'arrêt en affichant Err 37 Line XX où XX est la ligne de l'arrêt. Si tout est normal vous pouvez faire poursuivre l'exécution avec CONTINUE ou CON.
Vous pouvez aussi enlever les points d'arrêt avec UNBREAK suivi par un ou plusieurs numéros de lignes.

### Pour utiliser les points d'arrêt à autre chose :

De même qu'il existe un ON ERROR pour traiter les erreurs, il existe un ON BREAK pour modifier le comportement de l'EXL 100 en présence des points d'arrêt.
On peut ainsi les faire fonctionner comme indiqué ci-avant si l'on n'utilise pas ON BREAK ou que l'on utilise ON BREAK STOP.
On peut aussi les faire ignorer avec ON BREAK NEXT mais attention ceci fait aussi ignorer le touche BREAK (CTL C) ; votre programme ne peut donc plus être interrompu ce qui peut être dangereux s'il n'est pas tout à fait au point !
On peut enfin faire traiter les points d'arrêt comme des erreurs par l'utilisation du ON BREAK ERROR.
La fiche de ON BREAK donne tous les détails relatifs à ces possibilités qui ne présentent aucune difficulté d'emploi.



CHAPITRE 16

# Fabriquez vos propres instructions

### Les instructions qui manquent :

Si vous êtes un habitué du Basic Microsoft (marque déposée) vous avez dû remarquer l'absence de RIGHT$, LEFT$ et MID$ ; nous allons voir ci-après qu'il est possible de remplir les fonctions de ces instruction «standards» avec ce dont dispose l'EXL 100.
RIGHT$ ( A$ , I ) peut être remplacé par SEG$ ( A$, I , LEN(A$))
LEFT$ ( A$ , I ) peut être remplacé par SEG$ ( A$, 1 , I )
MID$ (A$ , I , J ) peut être remplacé par SEG$

( A$ , I , J ).

C'est simple n'est-ce-pas ?

**Définissez vos propres instructions :**

Nous avons parlé rapidement au chapitre 9 des sous programmes appelés par un CALL et nous en avons utilisés de nombreux comme CALL COLOR, CALL HRON, CALL PLOT, etc ... Ces sous programmes font partie intégrante du Basic de l'EXL 100. Il vous est, par contre, possible d'en définir d'autres vous-mêmes au moyen des instructions SUB, SUBEND et SUBEXIT.

Plutôt que de nous lancer dans de longues explications, nous allons vous donner quelques exemples de ce type et vous pourrez ensuite lire les fiches des instructions précédentes pour compléter votre culture.

Le premier exemple permet de tracer des cercles en haute résolution, remplaçant ainsi l'instruction CIRCLE que l'on trouve sur certains Basic. Le voici :

```
100 CALL HRON ("Y",1,11)
110 INPUT "RAYON : ";R
120 INPUT "LIGNE DU CENTRE : ";C1
130 INPUT "COLONNE DU CENTRE : ";C2
140 CALL CERCLE (R , C1 ,C2)
150 STOP
160 SUB CERCLE
(RAYON,LICENTRE,COLCENTRE)
170 DEG
180 FOR I = 0 TO 360
190 CALL PLOT ("b",COLCENTRE + RAYON *
COS(I),LICENTRE + RAYON *SIN(I))
200 NEXT
210 SUBEND
```

Nous avons défini un sous programme des lignes 160 à 210. Ce sous programme s'appelle CERCLE et demande trois paramètres RAYON, LICENTRE et COLCENTRE.



Le programme principal qui va de 100 à 150 appelle alors ce sous programme avec un CALL en fournissant les paramètres demandés qui sont dans ce cas les variables R , C1 et C2.

Essayez et vous verrez que cela trace bien un cercle ; mais attention, nous n'avons inclus aucun test sur les valeurs que vous fournissez au programme ce qui peut conduite à des tracés curieux si vous donnez n'importe quoi. Vous pouvez, à titre d'exercice, améliorer cet exemple en ajoutant des tests ou en ajoutant des paramètres au sous programme tel que la couleur du tracé par exemple.

Voici maintenant un autre exemple un peu plus long qui définit et appelle un sous programme appelé TRACE. Ce sous programme peut tracer n'importe quelle courbe sur l'écran haute résolution et trace automatiquement ses axes en indiquant les valeurs maximum atteintes.

```
100 ATTACH FN1
110 ATTACH FN2
120 PRINT TAB(13);"COULEURS"
130 PRINT "R:ROUGE","Y:JAUNE","C:CYAN",
"G:VERT","b:BLEU","B:NOIR",
140 PRINT "M:MAGENTA","W:BLANC"
150 INPUT "COULEUR TRACE : ";COUL$
160 INPUT "COULEUR AXES : ";AXE$
170 INPUT "COULEUR FOND : ";FON$
180 ON BREAK ERROR
190 ON ERROR 210
200 CALL TRACE
(0,41,165,-1,1,-1,1,COUL$,FON$,AXE$,17):PAUSE
210 CALL HROFF:CLS "GBb":CALL COLOR
("0GB")
220 STOP
230 SUB TRACE
(T1,T2,NB,XMIN,XMAX,YMIN,YMAX,COUL$,
FON$,AXE$,H)
240 CLS AXE$&FON$&"b"
250 CALL HRON (FON$,2,H):H=H*10 -1
260CALL COLOR ("0"&AXE$&FON$)
```

```
270 LOCATE (1,17):PRINT "YMAX = ";YMAX
280LOCATE (18,17):PRINT "YMIN = ";YMIN
290 PRINT
300 PAS =
(T2-T1)/NB:EX=319/(XMAX-
XMIN):EY=-H/(YMAX-YMIN):X0=-EX*XMIN:
Y0=-EY*YMAX
310 PRINT TAB (14):"Y = COS(T)";PRINT TAB
(14);"X = SIN(T) * COS(T/13)"
320 IF X0<0 THEN X0=0 ELSE IF X0>319 THEN
X0=318
330 CALL LINE (AXE$,X0,0,X0,H)
340 FOR I=0 TO H STEP 10 :CALL PLOT
(AXE$,X0+1,I):NEXT
350 IF Y0<0 THEN Y0=1 ELSE IF Y0>H
THEN Y0=H
360 CALL LINE (AXE$,0,Y0, 319,Y0)
370 FOR I=0 TO 319 STEP 10:CALL PLOT
(AXE$,I,Y0-1):NEXT
380 CALL FN1(X1,T1):CALL FN2(Y1,T1)
390FOR I=T1 TO T2+PAS STEP PAS:CALL
FN1(X,I):CALL FN2(Y,I)
400 CALL
LINE(COUL$,X1*EX+X0,Y1*EY+Y0,X*EX+
X0,Y*EY+Y0):X1=X:Y1=Y:NEXT:SUBEND
410 SUB
FN1(X,T):X=SIN(T)*COS(T/13)/1.5:SUBEND
420 SUB FN2(Y,T):Y=COS(T):SUBEND
```

Nous vous laissons le soin de l'analyser car cela constitue un très bon exercice et, au stade où nous en sommes, cela ne devrait pas vous poser de problème (un petit coup d'oeil à la fiche de l'instruction ATTACH est nécessaire mais c'est fort simple).
Délassez-vous ensuite en essayant ce programme. Si le tracé obtenu ne vous plaît pas, rien ne vous empêche de le changer en modifiant les deux lignes 410 et 420 qui définissent la fonction représentée.

**Définissez vos propres touches de fonction :**

Vous avez dû remarquer que la grille transparente que l'on pose sur le clavier comportait des repères vierges



d'inscription au-dessus des touches 1 à 0. Ces touches peuvent en effet être définies par vos soins comme des touches de fonction ; c'est-à-dire que chaque touche peut mémoriser une chaîne de 80 caractères quelconques et faire afficher ceux-ci par la suite en appuyant sur FONCTION puis sur la touche désirée. Voici comment il faut procéder :

Supposons que vous utilisiez souvent dans un programme :
CALL HRON ("R",1,11)
Frappez cela exactement comme indiqué ci-dessus et ne faites pas de retour chariot mais appuyez sur CTL puis sur la touche de votre choix, de 1 à 0, dans laquelle vous voulez mémoriser cette chaîne. Vous pouvez ensuite faire retour chariot.
Maintenant chaque fois que vous frappez FONCTION puis votre touche, vous voyez apparaître la chaîne ci-avant.

Attention, ces définitions sont effacées par un RESET !

**Le langage machine :**

A l'intérieur de votre EXL 100 se trouvent un certain nombre de composants électroniques. Le cerveau de l'appareil est un microprocesseur, ici un TMS 7020 ; c'est lui qui effectue tous les calculs et qui prend toutes les décisions en fonction des ordres qu'il reçoit.
Un microprocesseur ne peut manipuler que des signaux électriques, aussi toutes les informations qui lui sont données le sont au moyen de ces signaux avec un code particulier qui permet de représenter tous les chiffres avec des absences de tension ou des présences de tension. Une absence de tension est représentée par un 0 et une présence par un 1 ce qui permet de dire que le microprocesseur peut travailler en binaire c'est-à-dire en numération à base 2.
Pour comprendre les instructions du Basic, il faut que celles-ci soient traduites dans ce langage ; cela se fait par un programme contenu dans la cassette de ROM Basic. Ce programme s'appelle l'INTERPRETEUR Basic parce qu'il interprète les différentes instructions du Basic.

Pour rester toujours dans la machine il a été placé dans une mémoire appelée MEMOIRE MORTE ou ROM. Une telle mémoire garde l'information qu'elle contient même lorsqu'elle n'est pas alimentée. La ROM Basic de votre EXL 100 est un 32 K octect c'est-à-dire qu'elle contient 32768 mots d'informations ; chaque mot pouvant représenter un nombre entre 0 et 255.
Vos programmes et vos variables sont, par contre, rangés dans une MEMOIRE VIVE ou RAM c'est-à-dire une mémoire dans laquelle on peut lire et écrire à tout instant. Une telle mémoire présente cependant l'inconvénient de perdre son contenu lorsqu'elle n'est plus alimentée ; c'est pour cela que vous pouvez sauvegarder vos programmes sur cassettes. La RAM de votre EXL 100 contient 32768 mots d'informations.
D'autres circuits complètent cela et se chargent de fabriquer les images pour le récepteur TV, les signaux pour le magnétophone à cassette et les sons.

Pour accéder facilement aux divers emplacements mémoire de l'EXL 100, ceux-ci sont repérés par des adresses. Chaque emplacement correspondant à une adresse unique.

Si vous voulez faire des programmes très rapides ou très spéciaux, il est possible d'écrire des programmes directement dans le langage que comprend le microprocesseur ; on appelle cela programmer en LANGAGE MACHINE. Nous ne pouvons pas développer cela ici car il faudrait un livre au moins aussi épais que ce manuel pour vous l'expliquer. Si une telle programmation vous intéresse, il existe de nombreux livres dans les magasins spécialisés consacrés à ce sujet. Il faut seulement faire attention à vous en procurer un pour le microprocesseur TMS 7020.

Pour de petites opérations sur les mémoires de l'EXL 100, il existe des instructions Basic que nous allons présenter maintenant.

**Lisez dans la mémoire :**

Il est possible, à partir d'un programme Basic, de lire directement le contenu des mémoire de l'EXL 100. Pour



cela, il suffit de préciser l'adresse où l'on veut aller lire puisque c'est cela qui sert à repérer les mémoires ; l'instruction PEEK se charge alors du reste.
Essayez le programme suivant :

```
100 PRINT "ADRESSE";TAB (8);"CONTENU"
110 FOR A = 0 TO 19
120 CALL PEEK ( A , B )
130 PRINT A ; TAB(8) ; B
140 NEXT A
```

Ce programme lit les mémoires d'adresses 0 à 19 et affiche leur contenu sur l'écran.

**Ecrivez dans la mémoire :**

Lorsque vous frappez n'importe quelle touche du clavier, vous écrivez en mémoire mais ce n'est pas vous qui contrôlez cette écriture. Pour certaine applications particulières il est utile d'aller écrire directement à une adresse particulière en mémoire ;
cela se fait avec l'instruction POKE par un CALL POKE ( M , N) ou M est l'adresse à laquelle vous allez écrire et où N est la donnée à y écrire. M. doit être compris entre 0 et 65535 et N entre 0 et 255 sinon une erreur a lieu.

## CHAPITRE 17

# Pour finir

**A vous de jouer :**

Cette troisième partie se termine avec ce chapitre ; en effet nous vous avons présenté toutes les instrucitons de l'EXL 100 à de très rares exceptions

près, pour lesquelles une explication particulière ne nous a pas semblé nécessaire. Vous disposez maintenant de tous les atouts pour programmer. En cas de doute, les fiches de référence des instructions qui composent la troisième partie de ce manuel sont là pour vous guider. Elles décrivent en détail toutes les possibilités et restrictions de toutes les instructions.
Ces fiches sont à utiliser pour écrire vos propres programmes bien sûr mais aussi lorsque vous recopierez des programmes dans des livres ou des journeaux spécialisés pour voir comment adapter ceux-ci à l'EXL 100. Une telle adaptation ne devrait pas poser de problème vu les possibilités de son Basic.

S'il est possible de vous donner un conseil : lorsqu'un de vos programmes ne marche pas, soyez logique, essayez de vous mettre à la place de la machine et de voir ce qu'elle peut faire pour chaque instruction de celui-ci. Par ailleurs, les points d'arrêt et les messages d'erreur sont là pour vous aider, n'hésitez pas à les exploiter.

## Les extensions :

De nombreuses extensions sont commercialisées ou en cours de l'être pour votre EXL 100 ; votre distributeur peut vous fournir toutes les informations utiles à ce sujet.
Les plus courantes sont les manettes de jeux ou joystick qui, comme le clavier, sont à infrarouge et ne nécessitent donc pas de câble. Les instructions pour les utiliser sont déjà prévues dans le Basic avec KEY1 et KEY2 ; par ailleurs, une notice accompagne ces manettes pour vous donner toutes les indications nécessaires à leur sujet.

Une interface imprimante est également prévue pour se connecter à l'EXL 100 ; elle permet de brancher une imprimante série et une imprimante parallèle. Ces imprimantes peuvent être quelconques mais il existe aussi une imprimante EXELVISION spécialement adaptée à l'appareil. Là aussi les instructions sont prévues dans le Basic avec OPEN et CLOSE dont nous n'avons pas parlé puisqu'une



notice accompagne l'interface imprimante pour en expliquer l'utilisation.

Des cassettes de RAM CMOS peuvent aussi se connecter à l'EXL 100 et autorisent quasiment les mêmes fonctions que celles offertes par LOAD et SAVE pour un magnétophone à cassettes. La vitesse de sauvegarde ou de chargement est, par contre, infiniment plus rapide. Là aussi, toutes informations à leur sujet sont fournies avec les cassettes de RAM CMOS.

D'autres extensions plus complexes permettent de raccorder votre EXL 100 au réseau téléphonique via un modem et permettent son utilisation comme minitel «intelligent» ou pour dialoguer avec d'autres ordinateurs ou des banques de données. Enfin, une extension lecteurs de disquettes permet d'aborder des applications plus professionnelles.

3ème PARTIE

# FICHES

# DE

# REFERENCE

## ABS

**FORMAT :** ABS (expression numérique)

**DESCRIPTION ;**

L'expression contenue entre parenthèses est évaluée, puis la fonction ABS en donne sa valeur absolue. Si l'expression est positive ou nulle, la valeur absolue est égale à l'expression. Si l'expression est négative, la valeur absolue est égale à l'expression changée de signe. Le résultat de la fonction ABS est donc toujours positif ou nul.

**EXEMPLE :**

10 PRINT ABS (3.45) ; ABS (-12.5)

donne 3.45 12.5 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ACCEPT

**FORMAT :** ACCEPT SIZE (expression numérique) BEEP VALIDATE (type de données) NULL (expression), variable.

**DESCRIPTION :**

L'instruction ACCEPT suspend l'exécution du programme en cours jusqu'à ce qu'une donnée soit frappée au clavier. Les options qui suivent l'instruction ACCEPT permettent de préciser un certain nombre de fonctions ou contraintes relatives à la donnée à fournir.

BEEP fait émettre un son à l'EXL 100 lors de l'exécution de l'instruction ACCEPT signalant ainsi que le programme attend une donnée. Si une donnée non conforme à celle attendue est fournie, le son est émis à nouveau et ce pour chaque donnée frappée jusqu'à ce que les contraintes éventuelles exprimées soient vérifiées.

SIZE permet de préciser le nombre maximum de caractères que devra comporter la donnée fournie en réponse à l'instruction ACCEPT ; pour cela, l'expression numérique contenue entre parenthèses est évaluée et arrondie à l'entier le plus proche puis le résultat obtenu sert à préciser ce nombre.

VALIDATE permet de spécifier quels types de caractères seront acceptés en réponse à l'instruction ACCEPT. VALIDATE peut être suivi par une chaîne de caractères auquel cas, tous les caractères de la chaîne seront acceptés mais les expressions suivantes peuvent aussi être utilisées derrière VALIDATE :

- ALPHA qui autorise tous les caractères alphabétiques.
- UALPHA qui n'autorise que les caractères alphabétiques majuscules.
- DIGIT qui autorise les chiffres de 0 à 9
- NUMERIC qui autorise les chiffres de 0 à 9 ainsi que . , + , - et E
- ALPHANUM qui autorise tous les caractères alphabétiques et les chiffres de 0 à 9
- UALPHANUM qui autorise les caractères alphabétiques majuscules et les chiffres de 0 à 9.

NULL permet de préciser la valeur qui sera donnée à la donnée attendue par ACCEPT si aucune donnée n'est fournie en réponse à l'instruction et que la touche «retour chariot» seule est actionnée.

Vous pouvez consulter également les fiches de INPUT, LINPUT et KEY$.

**EXEMPLES :**

10 ACCEPT VALIDATE(«ON») SIZE (1),A$

attend un seul caractère qui ne peut être que O ou N.

10 ACCEPT BEEP NUL (PI),A



fait émettre un son et attend une donnée quelconque qui sera affectée à A ; si un retour chariot est frappé seul, PI ( 3.141592) est affecté à la variable A.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ACS

**FORMAT :** ACS (expression numérique)
**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis la fonction ACS en donne l'arc cosinus c'est-à-dire l'angle dont l'expression est le cosinus. Le résultat de cette fonction est donc un angle et est exprimé en radians, degrés ou grades selon l'unité d'angles sélectionnée au préalable. L'angle obtenu est toujours ramené à l 'intervalle 0 -PI ( 0 - 180° ).

La valeur de l'expression numérique dont on cherche l'arc cosinus doit être comprise entre -1 et + 1 sinon il a génération d'une erreur.

**EXEMPLE :**

10 A = COS(PI/3)
20 PRINT A ; ACS(A)

donne .5 1.047197551 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ASC

**FORMAT ;** ASC (expression chaîne de caractères)

**DESCRIPTION :**

La fonction ASC donne le code ASCII du premier caractère de la chaîne comprise entre parenthèses. Un message d'erreur est généré si la chaîne est vide. Le code ASCII vous est donné en annexe dans la dernière partie de ce manuel.

Cette fonction est l'inverse de la fonction CHR$ Vous pouvez consulter également la fiche de l'instruction CHR$

**EXEMPLE :**

10 PRINT ASC(«EXL»)

donne comme résultat lors de l'exécution 69 qui est le code ASCII (exprimé en décimal) de E.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ASN

**FORMAT :** ASN (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis la fonction ASN en donne l'arc sinus c'est-à-dire l'angle dont l'expression est le sinus. Le résultat de cette fonction est donc un angle et est exprimé en radians, degrés ou grades selon l'unité d'angle préalablement sélectionnée. L'angle obtenu est toujours ramené à l'intervalle 0 -PI ( 0 - 180°).

La valeur de l'expression numérique dont on cherche l'arc sinus doit être comprise entre -1 et +1 sinon il y a génération d'une erreur.

**EXEMPLE :**

10 A = SIN(PI/6)
20 PRINT A ; ASN(A)

donne .5 .5235987756 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ATN

**FORMAT :** ATN (expression numérique)



**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis la fonction ATN en donne l'arc tangente c'est-à-dire l'angle dont l'expression est la tangente. Le résultat de cette fonction est donc un angle et est exprimé en radians, degrés ou grades selon l'unité d'angle préalablement sélectionnée. L'angle obtenu est toujours ramené à l'intervalle -PI/2 +PI/2 (-90° +90°).

**EXEMPLE :**

10 A = TAN(PI/4)
20 PRINT A ; ATN(A)

donne 1 .7853981634 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ATTACH

**FORMAT :** ATTACH sous programme n° 1, sous programme n° 2, ...

**DESCRIPTION :**

L'instruction ATTACH permet de préserver les valeurs de variables utilisées dans des sous programmes. Lorsque ATTACH est exécutée, de la place mémoire est réservée pour les variables et leurs valeurs sont initialisées. Les variables ne sont alors plus initialisées lorsque les sous programmes sont appelés et les valeurs qu'elles prennent dans ceux-ci ne sont pas détruites à la fin des sous programmes.

L'utilisation de ATTACH pour un sous programme fréquemment appelé est un bon moyen d'accroître la vitesse d'exécution d'un programme. Il ne faut cependant pas oublier que ATTACH utilise de la mémoire pour stocker les variables ainsi préservées.

L'instruction RELEASE permet de libérer les variables préservées par ATTACH.

Vous pouvez consulter également la fiche de l'instruction RELEASE.

**EXEMPLE :**

```
10 FOR I = 1 TO 5
20 CALL X
30 NEXT I
```

fait afficher 0 0 0 0 0 car les variables utilisées dans le sous programme X sont initialisées à chaque appel de celui-ci.

```
40 ATTACH X
50 CALL X
60 NEXT I
```

fait afficher 0 1 2 3 4 car les variables ne sont plus initialisées lors de l'appel du sous programme X.

```
70 SUB X
80 PRINT I
90 I = I + 1
100 SUBEND
```

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## BREAK

**FORMAT :**

BREAK liste de numéros de lignes
BREAK
CTL C

**DESCRIPTION :**

L'instruction BREAK permet de placer des «points d'arrêt» dans un programme. Ces points d'arrêt peuvent être spécifiés de deux façons différentes : l'instruction BREAK peut être placée dans un programme comme toute autre instruction auquel cas



le programme s'arrêtera en arrivant à la ligne qui la contient. BREAK peut aussi être utilisée comme une commande et être suivie par un ou plusieurs numéros de lignes séparés par des virgules. Dans ce cas, l'exécution du programme s'arrêtera immédiatement avant les numéros précisés.

Lorsqu'un point d'arrêt a été placé dans un programme par un BREAK suivi par un ou plusieurs numéros de ligne, celui-ci reste actif tant qu'une commande UNBREAK n'a pas été exécutée ou tant que la ligne où se trouve le point d'arrêt n'a pas été éditée ou effacée.

Lorsqu'un programme a été arrêté par un BREAK, les variables peuvent être affichées afin de déterminer pourquoi le programme ne s'exécute pas comme prévu. L'utilisation de CONTINUE permet de poursuivre ensuite l'exécution.

L'arrêt sur un BREAK est signalé par le message Err 37 Line XX ou XX est le numéro de ligne concerné. L'erreur 37 correspond à l'exécution d'un BREAK comme vous pouvez le constater en examinant la liste des messages d'erreurs annexe 2.

La touche BREAK (CTL C) réalise un BREAK au moment de l'action sur celle-ci. Le point d'arrêt ainsi généré est immédiatement pris en compte et provoque l'affichage du message d'erreur vu ci-avant indiquant quelle était la ligne en cours d'exécution. Les seuls cas où la touche BREAK est inactive sont : une grossière erreur de programmation dans le programme en cours ou l'invalidation des points d'arrêt par un ON BREAK NEXT.

Vous pouvez consulter également les fiches des instructions CONTINUE, ON BREAK et UNBREAK.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CALL

**FORMAT :** CALL nom de sous programme (liste de paramètres)

**DESCRIPTION :**

L'instruction CALL permet d'appeler un sous programme. Le premier sous programme rencontré, par ordre croissant de numéros de lignes, ayant le nom spécifié est exécuté. Après exécution de celui-ci, le programme principal continue avec la première instruction suivant le CALL.
Lors de l'appel des sous programmes, les sous programmes interne à l'EXL 100 sont prioritaires.

La liste de paramètres qui suit le nom du sous programme est utilisée pour passer des paramètres à ce dernier. Elle doit être conforme à ce que le sous programme attend sinon un message d'erreur est généré.

Les sous programmes du Basic de l'EXL 100 sont décrits dans leurs fiches respectives auxquelles nous vous demandons de bien vouloir vous reporter.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CHAR

**FORMAT** : CALL CHAR (code, chaîne de caractères)

**DESCRIPTION :**

CHAR est un sous programme qui doit être appelé par un CALL. Il permet de définir jusqu'à 127 caractères de votre choix inscrits dans une matrice de 10 lignes de 8 colonnes. Les deux paramètres qui suivent le CALL CHAR sont le code du caractère qui est un nombre compris entre 0 et 127, la valeur 32 étant exclue et une chaîne de caractères qui définit le caractère point par point dans la matrice 8 sur 10.
Le ou les caractères ainsi définis sont mémorisés dans le générateur de caractères utilisateur de l'EXL 100 d'où ils peuvent être rappelés par l'instruction COLOR. Un RESET vide le générateur de



caractères utilisateur des caractères ainsi définis.

La chaîne de caractères est constituée de la façon suivante :
- Le caractère à définir est représenté point par point dans une matrice de 10 lignes sur 8 colonnes.
- Chaque point «allumé» est représenté par un 1 et chaque point «éteint» par un 0.
- Le code ainsi obtenu est converti en hexadécimal (un tableau de conversion vous est fourni en annexe 7).
- La chaîne est alors formée par la succession des codes hexadécimaux de chaque ligne en commençant par la ligne du haut de la matrice et en utilisant deux chiffres par code (1 est noté 01, F est noté 0F, etc...).

Vous pouvez consulter également les fiches des instructions CHR$ et COLOR.

**EXEMPLE :**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 00000000 soit 00 |
| | | | * | * | | | | 00011000 soit 18 |
| | | | * | * | | | | 00011000 soit 18 |
| | | * | * | * | * | | | 00111100 soit 3C |
| | * | | * | * | | * | | 01011010 soit 5A |
| * | | | * | * | | | * | 10011001 soit 99 |
| | | | * | * | | | | 00011000 soit 18 |
| | | * | | | * | | | 00100100 soit 24 |
| | * | | | | | * | | 01000010 soit 42 |
| * | | | | | | | * | 10000001 soit 81 |

Un tel caractère est défini par : CALL CHAR (25,«0018183C5A9918244281») si l'on désire lui donner le code 25. Il peut être utilisé dans un progamme de la façon suivante :

```
100 CALL CHAR (25,«0018183C5A9918244281»)
110 CALL COLOR («1RG»)
120 PRINT CHR$(25)
```

ceci fera afficher le caractère de couleur rouge sur fond vert en raison du CALL COLOR («1RG»). Le 1 est indispensable dans ce CALL pour indiquer que l'on utilise le générateur de caractères utilisateur.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CHR$

**FORMAT :** CHR$ (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée et le résultat est considéré comme un code ASCII. La fonction CHR$ donne alors le caractère correspondant à ce code. Le code ASCII vous est donné en annexe dans la dernière partie de ce manuel.
Cette fonction est l'inverse de la fonction ASC.

Cette fonction est souvent utilisée lors d'instructions PRINT pour faire afficher des caractères originaux définis par vos soins ou pour manipuler les caractères spéciaux du code ASCII (caractères de code inférieur à 32).

Vous pouvez consulter également les fiches des instructions ASC et CHAR.

**EXEMPLE :**

10 PRINT CHR$(69) ; CHR$(88) ; CHR$(76)
donne EXL comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CLEANUP

**FORMAT :** CALL CLEANUP



**DESCRIPTION :**

CLEANUP est un sous programme qui doit être appelé par un CALL. Il a pour fonction d'effacer de la mémoire de l'EXL 100 toutes les variables qui ne sont pas utilisées par le programme présentement en mémoire. Tous les fichiers qui pourraient éventuellement être restés ouverts sont également fermés par CLEANUP. Si un CALL CLEANUP est exécuté lorsqu'un programme a été arrêté par un point d'arrêt, la commande CONTINUE ne peut plus être utilisée pour poursuivre l'exécution.

CALL CLEANUP ne peut pas être utilisé dans un programme.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CLOSE

**FORMAT :** CLOSE # numéro de fichier

**DESCRIPTION :**

CLOSE permet de terminer l'association réalisée entre un fichier et un numéro de fichier telle que celle-ci a été définie par l'instruction OPEN.
Après exécution d'un CLOSE, le numéro de fichier est rendu disponible et peut être assigné à toute autre chose par l'instruction OPEN.
Toute tentative de CLOSE sur un numéro de fichier qui n'a pas été défini par un OPEN se traduit par une erreur.

Un certain nombre d'actions réalisent automatiquement un CLOSE sur tous les fichiers existants :
- Le fait d'éditer le programme
- Le fait d'utiliser NEW, RENUMBER, RUN ou SAVE
- Le fait de lister le programme sur un équipement périphérique
- Le fait d'utiliser le sous programme CLEANUP
- Le fait d'actionner le poussoir de RESET.

Vous pouvez consulter également la fiche de l'instruction OPEN.

**EXEMPLE :**

10 CLOSE #6

«ferme» le fichier affecté au numéro 6 par une commande OPEN préalable.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## CLS

**FORMAT :** CLS
CLS (expression chaîne de caractères)
CLS expression chaîne de caractères

**DESCRIPTION :**

L'instruction CLS utilisée seule efface l'écran sans modifier les paramètres relatifs aux couleurs (caractères, fond, pourtour de l'écran). L'instruction CLS suivie par une chaîne de caractères efface l'écran et redéfinit un ou plusieurs paramètres. Le premier caractère de la chaîne fixe la couleur des caractères, le deuxième caractère de la chaîne fixe la couleur du fond de l'écran et le troisième caractère fixe la couleur du pourtour de l'écran.
Le troisième caractère est facultatif ; s'il est omis, la couleur du pourtour préalablement définie est conservée.
Le codage des couleurs s'effectue par une lettre de la façon suivante :
R - rouge, G - vert, M - magenta ou mauve, C - cyan ou bleu ciel, Y - jaune, b - bleu-foncé, W - blanc et B - noir.

**EXEMPLE :**

10 CLS

efface l'écran sans modifier les couleurs pré-définies.



10 CLS «RGC»

efface l'écran et définit des caractères rouges, un fond vert et un pourtour bleu ciel.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## COLOR

**FORMAT :** CALL COLOR (chaîne de caractères)

**DESCRIPTION :**

COLOR est un sous programme qui doit être appelé par un CALL. Il permet de définir le générateur de caractères à utiliser, la couleur du caractère à afficher et la couleur du fond ainsi que la taille de celui-ci, une éventuelle inversion de couleur entre fond et caractère ainsi qu'une possibilité de clignotement.
Ces définitions ont lieu au moyen de la chaîne de caractères qui suit COLOR.

Le premier caractère de la chaîne doit être 0 ou 1 et spécifie le générateur de caractères choisi. Un 0 correspond au générateur standard et un 1 au générateur utilisateur. Si ce dernier n'a pas été défini, il est identique au générateur standard.

Le deuxième caractère précise la couleur des caractères qui seront affichés ensuite à l'exécution du CALL COLOR. Le codage est identique à celui décrit ci-avant pour CLS. Si ce deuxième caractère n'est pas présent, la définition de couleur en cours est utilisée.

Le troisième caractère de la chaîne précise la couleur du fond à l'emplacement du caractère. Le codage des couleurs est le même que pour CLS vu ci-avant. Si ce troisième caractère est omis, la définition de couleur en cours est utilisée. La définition de couleur de fond est impossible si une option double largeur ou double hauteur de caractère est sélectionnée. Dans ce cas, ce troisième caractère est facultatif et est ignoré s'il est présent ;

la couleur du fond étant obligatoirement celle de l'écran.

Le quatrième caractère est optionnel et peut être un L ou un H. Un L spécifie un affichage en double largeur; un H en double hauteur. En cas d'affichage simultané en double largeur et double hauteur, le quatrième caractère est rendu égal à HL ou LH.

Le cinquième (ou sixième dans le cas de la double largeur + double hauteur) caractère est optionnel et peut être un I. Dans ce cas, il spécifie une inversion entre la couleur du caractère précédemment définie et la couleur du fond.

Le sixième (ou septième dans le cas de double largeur + double hauteur) caractère est optionnel et peut être un F. Dans ce cas, il spécifie le mode d'affichage clignotant.

Attention ! Toutes les définitions d'affichage faites par un CALL COLOR (couleurs des caractères et du fond, double largeur, double hauteur, inversion, clignotement) restent valables tant qu'un nouveau CALL COLOR les modifiant n'a pas été exécuté.

Vous pouvez consulter également la fiche de l'instruction CLS.

**EXEMPLE :**

```
10 CALL COLOR («0RGL»)
20 PRINT «BBOONNJJOOUURR»
```

sélectionne le générateur de caractère système (0), des caractères rouges, un fond vert (G) et la double largeur (L). Le PRINT suivant fait donc afficher BONJOUR en rouge sur fond vert et en double largeur.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■



## CONTINUE

**FORMAT :** CONTINUE
CON
CONTINUE numéro de ligne
CON numéro de ligne

**DESCRIPTION :**

La commande CON, utilisable seulement en mode direct, permet de reprendre l'exécution d'un programme qui a été interrompu. L'exécution reprend à partir du numéro de ligne spécifié s'il existe ou à partir de l'endroit où le programme en cours a été interrompu.

La commande CON ne peut pas être utilisée si, après l'interruption du programme, une ou plusieurs des actions suivantes ont eu lieu :
- Modification du programme
- Utilisation d'une des commandes RUN, RENUMBER, SAVE ou NEW
- Sortie du listing du programme sur un équipement périphérique.
- Utilisation du poussoir de RESET.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## COS

**FORMAT :** COS (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée et le résultat est considéré comme une mesure d'angle exprimée en radians, degrés ou grades selon l'unité d'angle préalablement sélectionnée. La fonction COS donne alors le cosinus de cet angle.

**EXEMPLE :**

```
10 PRINT COS(PI/3)
```

donne .5 comme résultat lors de l'exécution.

## DATA

**FORMAT :** DATA donnée, donnée, donnée, ....

**DESCRIPTION :**

L'instruction DATA est utilisée conjointement à l'instruction READ pour affecter des valeurs à des variables. Lorsque l'instruction READ est exécutée, les données qui suivent l'instruction DATA sont affectées aux variables qui suivent l'instruction READ.

Les données qui suivent l'instruction DATA peuvent être numériques ou être des chaînes de caractères, disposées dans n'importe quel ordre et séparées les unes des autres par des virgules. Les espaces sont ignorés. Si une chaîne de caractères doit contenir des espaces qui doivent être pris en compte ou si elle contient une ou des virgules, la chaîne doit être écrite entre des guillemets. Si une chaîne de caractères doit contenir un guillemet, il doit être représenté par deux guillemets adjacents.

L'instruction DATA doit être la seule et unique instruction d'une ligne. Elle peut être placée n'importe où dans un programme, même après le READ qui l'utilise. Il peut y avoir plusieurs instructions DATA dans un même programme. Lors de l'exécution du premier READ du programme, l'instruction DATA ayant le numéro de ligne le plus bas sera utilisée la première et ainsi de suite par ordre croissant de numéro de ligne.

Vous pouvez consulter également les fiches des instructions READ et RESTORE.

**EXEMPLE :**

```
10 DATA 10,BONJOUR,«EXL 100»,345
20 READ A,Z$,C$,B
30 PRINT A;B;Z$;C$
```

donne 10 345 BONJOUR EXL 100 comme résultat lors de l'exécution.



## DEG

**FORMAT :** DEG

**DESCRIPTION :**

Cette instruction définit le degré comme unité d'angle pour tous les calculs ultérieurs. Cette instruction n'est annulée que par l'exécution de GRAD ou RAD ou par une ré-initialisation de l'EXL 100.

Le passage dans ce mode angulaire est matérialisé par l'affichage du mot DEGRE sur la ligne de service en haut de l'écran.

Vous pouvez consulter également les fiches des instructions GRAD et RAD.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## DELETE

**FORMAT :** DELETE liste de numéros de lignes
DEL liste de numéros de lignes

**DESCRIPTION :**

Cette commande, utilisable en mode direct et programmé, permet d'effacer une ou plusieurs lignes d'un programme.
DELETE suivi par un numéro de ligne ou par des numéros de lignes séparés les uns des autres par des virgules efface tous les numéros ainsi spécifiés.
DELETE peut aussi être suivi par un tiret suivi par un numéro de ligne; dans ce cas, toutes les lignes à partir de la première et jusqu'à la ligne spécifiée incluse sont effacées.
DELETE peut aussi être suivi par un numéro de ligne suivi d'un tiret ; dans ce cas, toutes les lignes à partir de la ligne spécifiée jusqu'à la dernière ligne du programme sont effacées.
DELETE peut aussi être suivi par deux numéros de lignes séparés l'un de l'autre par un tiret ; dans

ce cas, les lignes comprises entre ces deux numéros, bornes comprises, sont effacées.

**EXEMPLES :**

DELETE 120-160 efface les lignes 120 à 160 incluses.

DELETE - 100 efface du début à la ligne 100 incluse.

DELETE 120 - efface de la ligne 120 incluse à la fin.

DELETE 10,120,345 efface les lignes 10, 120 et 345.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## DIM

**FORMAT :** DIM nom de tableau (entier, entier, entier)

**DESCRIPTION :**

L'instruction DIM spécifie les caractéristiques d'un tableau et réserve la place nécessaire en mémoire. DIM est suivi par un nom de tableau puis par un, deux ou trois entiers compris entre parenthèses et séparés par des virgules selon que le tableau est à une, deux ou trois dimensions.

Les indices servant à repérer les éléments d'un tableau commencent à partir de 0 ; en conséquence DIM A ( N ) réservera N + 1 éléments pour le tableau A.

Lors du lancement d'un programme, tous les éléments des tableaux numériques sont mis à 0 et les éléments des tableaux chaînes de caractères sont rendus égaux à la chaîne vide.

Si un tableau n'est pas défini par une instruction DIM avant son utilisation dans un programme, la valeur maximum de chaque indice est fixée à 10.



Un tableau doit être défini une fois et une seule dans un même programme. La définition au moyen de DIM ne doit pas être utilisée au sein d'un IF THEN ELSE.

**EXEMPLES :**

10 DIM A$(20)
réserve de la place en mémoire pour 21 éléments du tableau chaîne de caractères appelé A$.

20 DIM A(100) , B(10,12)

réserve de la place en mémoire pour les 101 éléments du tableau A et pour les 143 éléments ( 11 x 13 ) du tableau B. Tous ces éléments sont initialisés à 0.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## END

**FORMAT :** END

**DESCRIPTION :**

L'instruction END termine un programme. Elle peut être interchangée avec l'instruction STOP. Bien que END puisse apparaître n'importe où dans un programme, elle est généralement placée sur la dernière ligne.

L'utilisation de END est facultative car l'exécution d'un programme cesse automatiquement dès que le numéro de ligne le plus élevé qu'il comporte a été atteint.

Vous pouvez consulter également la fiche de l'instruction STOP.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ERR

**FORMAT :** CALL ERR (code d'erreur, type d'erreur, numéro de fichier, numéro de ligne).

**DESCRIPTION :**

ERR est un sous programme qui doit être appelé par un CALL. ERR permet d'affecter à des variables un numéro de code d'erreur, un numéro de type d'erreur, un numéro de fichier et un numéro de ligne de la dernière erreur rencontrée dans le programme en cours d'exécution. Lorsqu'une erreur a lieu, l'utilisation du ON ERROR conjointement à CALL ERR permet de prendre les décisions qui s'imposent. Une erreur est considérée comme traitée après l'exécution du RETURN qui termine le sous programme appelé par ON ERROR.

Le code d'erreur est un nombre compris entre 0 et 127 qui correspond au numéro type de l'erreur tel que ceux-ci sont décrits dans l'annexe 2.

Le type d'erreur est toujours 0 sauf en cas d'erreur d'entrée/sortie auquel cas le type est un nombre compris entre 1 et 255 tel que ceux définis dans l'annexe 2.

Le numéro de fichier est toujours 0 sauf en cas d'erreur d'entrée/sortie auquel cas le numéro de fichier est celui relatif à l'instruction qui a causé l'erreur.

Le numéro de ligne est celui de la ligne en cours d'exécution lorsque l'erreur a été détectée. Ce n'est pas obligatoirement celui de la ligne contenant l'erreur car celle-ci peut provenir de calculs antérieurs par exemple.

Si aucune erreur n'a eu lieu, tous les numéros fournis par CALL ERR sont nuls.

Vous pouvez consulter également les fiches des instructions ON ERROR et RETURN.

**EXEMPLE :**

100 CALL ERR (A,B,C,D)

donne à A le numéro de code de l'erreur la plus récente non traitée, à B le numéro de type d'erreur,



à C le numéro de fichier responsable de l'erreur et à D le numéro de ligne en cours d'exécution lorsque l'erreur s'est produite.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## EXEC

**FORMAT :** CALL EXEC (adresse, données)

**DESCRIPTION :**

EXEC est un sous programme qui doit être appelé par un CALL. Ce sous programme permet de faire exécuter un programme en langage machine contenu en mémoire de l'EXL 100. L'information adresse qui suit est obligatoire et spécifie l'adresse mémoire de la première instruction du programme à exécuter. Elle doit être comprise entre 0 et 65535 sinon une erreur est générée.
Les données qui suivent sont facultatives et servent à passer des paramètres dans les deux sens entre le programme Basic et le programme en langage machine.

L'écriture de ce programme en langage machine est détaillée dans le manuel du programme assembleur de l'EXL 100 (non fourni dans la version de base de l'appareil).

Vous pouvez consulter également les fiches des instructions PEEK et POKE.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## EXP

**FORMAT :** EXP (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis la fonction EXP en calcule l'exponentielle. Rappelons que l'exponentielle de X n'est

autre que e puissance X ou e est la base des logarithmes Népériens ou naturels et est égal à 2.718281828.

**EXEMPLE :**

10 PRINT EXP(2)

donne 7.389056099 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## FOR TO STEP

**FORMAT :** FOR variable de boucle = valeur initiale TO valeur finale STEP pas de la variable de boucle.

**DESCRIPTION :**

L'instruction FOR TO est utilisée conjointement à l'instruction NEXT pour réaliser une boucle c'est-à-dire un ensemble de lignes de programme qui sont exécutées un certain nombre de fois. La variable de boucle, la valeur initiale, la valeur finale et le pas de la variable de boucle sont des expressions numériques.

Lorsque l'instruction FOR est exécutée pour la première fois, la variable de boucle se voit affecter la valeur initiale. Si la valeur finale est supérieure à la valeur initiale, la boucle n'est pas exécutée et les instructions comprises entre FOR et NEXT sont ignorées ; le programme continue son exécution avec la première instruction suivant le NEXT. Dans tous les autres cas, les instructions qui suivent FOR sont exécutées jusqu'à arriver à l'instruction NEXT. Cette instruction fait augmenter la variable de boucle du pas de la variable de boucle spécifiée après STEP. Si la variable de boucle est inférieure ou égale à la valeur finale, les instructions comprises entre FOR et NEXT sont à nouveau exécutées.



Lorsqu'à la suite de l'exécution de l'instruction NEXT, la variable de boucle devient supérieure à la valeur finale, le programme continue son déroulement avec les instructions qui suivent NEXT ; la boucle est terminée.

Une boucle peut être contenue dans une autre boucle sous réserve de respecter les deux conditions suivantes :
- La boucle «intérieure» doit être comprise intégralement dans l'autre boucle c'est-à-dire que toutes ses instructions, FOR et NEXT compris, doivent se trouver incluses dans la boucle «extérieure»
- Les variables de boucle utilisées pour chaque boucle doivent impérativement avoir des noms différents.

La partie STEP suivie du pas de la variable de boucle est optionnelle ; en son absence, ce pas est fixé à 1. STEP spécifie la quantité qui sera ajoutée à la variable de boucle lors de l'exécution de l'instruction NEXT. STEP peut être positif ou négatif ; dans ce dernier cas la variable de boucle diminue à chaque tour et la boucle est exécutée tant que la variable de boucle n'est pas inférieure à la valeur finale. Toujours dans ce cas, la boucle est ignorée si la valeur initiale est inférieure à la valeur finale.

Il est permis de sortir d'une boucle par un GOTO, GOSUB ou IF THEN ELSE et d'y rentrer à nouveau ensuite. Il est par contre interdit d'entrer pour la première fois dans une boucle sans exécuter le FOR TO initial car la variable de boucle est alors indéterminée.

Vous pouvez consulter également la fiche de l'instruction NEXT.
EXEMPLES :

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

fait imprimer la variable de boucle à chaque tour c'est-à-dire les nombres dé 1 à 10 inclus.

```
10 FOR I = 0 TO 100 STEP 10
20 PRINT I
30 NEXT I
```

fait imprimer les nombres de 1 à 100 en allant de 10 en 10 en raison du STEP 10.

```
10 FOR I = 10 TO 1 STEP —1
20 PRINT I
30 NEXT I
```

fait imprimer les nombres de 10 à 1 inclus puisque le pas est de —1.

```
10 FOR I = 1 TO 10
. . . . . . . . . . . . . . .
80 FOR J = 2 TO 8
. . . . . . . . . . . . . . .
120 NEXT J
. . . . . . . . . . . . . . .
200 NEXT I
```

est un exemple correct de boucles car la boucle intérieure, de variable J, est entièrement contenue dans la boucle extérieure de variable I.

```
10 FOR 1 = 1 TO 10
. . . . . . . . . . . . . . .
80 FOR J = 2 TO 8
. . . . . . . . . . . . . . .
120 NEXT I
. . . . . . . . . . . . . . .
200 NEXT J
. . . . . . . . . . . . . . .
```

est un exemple incorrect de boucles imbriquées. La boucle intérieure se termine après la fin de la boucle extérieure.



## GOSUB

**FORMAT :** GOSUB numéro de ligne

**DESCRIPTION :**

L'instruction GOSUB est un appel de sous-programme dont la première instruction se trouve au numéro de ligne spécifié. Lorsqu'un programme arrive sur un GOSUB, l'exécution se poursuit au numéro de ligne spécifié et ce jusqu'à ce que l'on rencontre une instruction RETURN. L'exécution du RETURN fait alors poursuivre l'exécution du programme par l'instruction qui suit le GOSUB.

En d'autres termes, le sous programme ainsi appelé commence au numéro de ligne spécifié et se termine par un RETURN.

Un sous-programme appelé par un GOSUB doit impérativement se terminer par un RETURN.

Un même sous-programme peut être appelé autant de fois que nécessaire dans un même programme. Un sous-programme peut lui-même appeler d'autres sous-programmes au moyen de GOSUB sans restriction.

Vous pouvez consulter également les fiches des instructions ON GOSUB et RETURN.

**EXEMPLE :**

```
10 GOSUB 120
20 PRINT A; B; C
. . . . . . . . . . . .
120 A = 3 * 2
130 B = 4 * 2
140 C = 5 * 2
150 RETURN
```

donne 6 8 10 comme résultat lors de l'exécution.

## GOTO

**FORMAT :** GOTO numéro de ligne

**DESCRIPTION :**

L'instruction GOTO est une instruction de saut ou branchement inconditionnel. Lorsqu'un programme arrive au niveau d'une instruction GOTO, l'exécution se poursuit au numéro de ligne spécifié. Contrairement au GOSUB, aucune fonction de retour "automatique" après le GOTO n'existe.

Vous pouvez consulter également la fiche de l'instruction ON GOTO.

**EXEMPLE :**

```
10 PRINT "BONJOUR"
20 GOTO 40
30 PRINT "CETTE LIGNE EST SAUTEE"
40 PRINT "JE SUIS L'EXL 100"
```

donne comme résultat lors de l'exécution

```
BONJOUR
JE SUIS L'EXL 100
```

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## GRAD

**FORMAT :** GRAD

**DESCRIPTION :**
Cette instruction définit le grade comme unité d'angles pour tous les calculs ultérieurs. Cette instruction n'est annulée que par l'exécution d'une instruction DEG ou RAD ou lors d'une réinitialisation de l'EXL 100.

Le passage dans ce mode angulaire est matérialisé par l'affichage des quatre lettres GRAD sur la ligne de service en haut de l'écran.

Vous pouvez consulter les fiches des instructions DEG et RAD.



■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## HRON

**FORMAT :** CALL HRON ("caractère", n° de ligne début, nombre de lignes)

**DESCRIPTION :**

HRON est un sous-programme qui doit être appelé par un CALL. HRON permet de mettre une partie de l'écran en mode haute résolution et d'en définir la couleur du fond. Le caractère qui suit le HRON définit la couleur du fond ; il peut être exprimé directement ou être une variable chaîne de caractères. Il est codifié comme indiqué dans la fiche de l'instruction CLS.
Le numéro de ligne qui suit indique à partir de quelle ligne l'écran doit passer en haute résolution. Les lignes sont numérotées de 1 à 24 ; la ligne 1 étant la ligne située en haut de l'écran.
Le nombre qui suit indique combien de lignes doivent passer en haute résolution. Ce nombre doit être compatible de la taille de l'écran compte tenu du numéro de ligne qui le précède et du fait que seules les lignes 1 à 20 incluses peuvent passer en haute résolution.

Toute tentative de faire passer plus de lignes ou d'autres lignes que les lignes 1 à 20 incluses en haute résolution conduit à la génération d'une erreur.

En mode haute résolution, l'écran dispose de 320 points horizontaux repérés de 0 à 319 ; 0 étant sur le bord gauche de l'écran et de N x 10 points verticaux, N étant le nombre de lignes mises en haute résolution. Les points sont repérés verticalement à partir de 0, 0 étant le point situé en haut de l'écran.

Vous pouvez consulter également les fiches des instructions LINE et PLOT.

**EXEMPLE :**

100 CALL HRON («R»,1,10)

place les 10 premières lignes de l'écran en mode haute résolution (vous disposez ainsi de 320 points horizontaux sur 100 points verticaux) et les initialise en rouge.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## HROFF

**FORMAT :** CALL HROFF

**DESCRIPTION :**

HROFF est un sous programme qui doit être appelé par un CALL. HROFF permet de remettre l'écran en mode alphanumérique normal et annule tout passage en mode haute résolution préalable. Les graphiques ou affichages ayant pu être faits en mode haute résolution sont définitivement perdus après un CALL HROFF.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## IF THEN ELSE

**FORMAT :** IF condition THEN action n° 1 ELSE action n° 2

**DESCRIPTION :**

L'instruction IF THEN ELSE exécute une action, parmi deux possibilités spécifiées, en fonction d'une condition. Dans l'expression de l'instruction ci-avant, si la condition est vraie, l'action n°1 sera exécutée ; si la condition est fausse, l'action n°2 sera exécutée. La présence du ELSE suivi par l'action n°2 est facultative. Si ELSE est omis, l'action n°2 est tout simplement la ligne de programme qui suit celle contenant le IF THEN.
Après l'exécution de l'action n°1 ou de l'action n°2, et si celle-ci ne comporte pas de branchement



inconditionnel, l'exécution du programme se poursuit à la ligne suivante.

La condition peut être une expression utilisant des opérateurs de comparaison ou être une expression numérique. Si le fait d'être vrai ou faux ne pose pas de problème pour les comparaisons, pour les expression numériques la convention est la suivante : si l'expression est nulle, elle est considérée comme fausse alors que toute valeur non nulle est considérée comme vraie.

Action n°1 et action n°2 peuvent être des numéros de lignes, des instructions ou plusieurs instructions séparées par le caractère :. Si un numéro de ligne est utilisé le fait d'exécuter l'action consiste à continuer l'exécution du programme à ce numéro de ligne. Si des instructions sont utilisées, le fait d'exécuter l'action fait exécuter les instructions spécifiées.

Le IF THEN ELSE doit être contenu tout entier dans une ligne et se termine avec la fin de la ligne. Les IF THEN ELSE peuvent être imbriqués les uns dans les autres en employant, par exemple, un IF THEN ELSE en guise d'action n°1 ou d'action n°2 dans un autre IF THEN ELSE. Si une telle imbrication ne comporte pas en nombre égal des THEN et des ELSE, les THEN et les ELSE les plus proches les uns des autres sont appariés.

Une instruction IF THEN ELSE ne doit pas contenir d'instruction DIM.

**EXEMPLES :**

```
10 IF X < 4 THEN 100
20 . . . . . . . . . . . . . . .
```

fait sauter le programme à la ligne 100 si X est inférieur à 4 sinon l'exécution continue ligne 20.

```
10 IF A THEN B = 2 ELSE B = 3 :GOTO 200
20 . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

si A est différent de 0, B est rendu égal à 2 et le programme continue ligne 20 ; si A est égal à 0 (donc faux), B est rendu égal à 3 et le programme continue ligne 200.

10 IF A = 1 THEN IF B = 2 THEN C = 3 ELSE D = 4

Si A est égal à 1 et si B est égal à 2, alors C est rendu égal à = et le programme continue à la ligne suivante. Si A est égal à 1 et si B n'est pas égal à 2, alors D est rendu égal à 4 et le programme continue à la ligne suivante. Enfin, si A n'est pas égal à 1, le programme passe à la ligne suivante.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## IMAGE

**FORMAT :** IMAGE chaîne de caractères

**DESCRIPTION :**

IMAGE permet de définir un format d'affichage pour l'instruction PRINT USING. Au lieu d'indiquer ce format après le USING de la ligne contenant le PRINT USING, il est possible de spécifier celui-ci au moyen d'une ligne contenant l'instruction IMAGE. Le PRINT USING qui doit l'utiliser y fait ensuite référence sous la forme PRINT USING numéro de ligne où le numéro de ligne est celui de la ligne contenant l'instruction IMAGE désirée. IMAGE doit être la seule instruction d'une ligne et la chaîne de caractères qui la suit doit être comprise entre des guillemets si les espaces qu'elle contient doivent être significatifs.
La syntaxe que doit respecter la chaîne de caractères qui suit l'instruction IMAGE est absolument identique à ce qui est exposé dans la fiche de l'instruction PRINT USING à laquelle vous êtes priés de vous reporter.



**EXEMPLE :**

100 IMAGE "LES DEUX NOMBRES SONT ✱ ✱ ✱ ✱ ✱ ✱ . "
. . . . . .
450 PRINT USING 100 , A , B

a le même effet que

450 PRINT USING "LES DEUX NOMBRES SONT ### ET # #.# #" , A , B .

Le résultat de l'exécution de cet exemple est détaillé dans la fiche de l'instruction PRINT USING.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## INPUT

**FORMAT :** INPUT chaîne de caractères ; variable, variable, ...

**DESCRIPTION :**

Cette instruction permet de fournir des données à un programme à partir du clavier. Lors de son exécution le programme s'arrête tant que toutes les données attendues n'ont pas été fournies.

La chaîne de caractères qui suit le INPUT peut être une variable chaîne de caractères auquel cas, son nom est simplement écrit après INPUT et est suivi d'un point virgule ; ce peut être aussi, une chaîne de caractères qui doit alors être comprise entre deux guillements et qui doit aussi être suivie par un point virgule. Enfin, cette chaîne de caractères peut ne pas exister auquel cas la liste de variables suit le INPUT sans interposition du point virgule.

La chaîne de caractères est affichée sur l'écran à partir de la position courante du curseur due à la précédente opération d'entrée/sortie ou à l'exécution d'une instruction de positionnement de celui-ci. Si la chaîne de caractères est inexistante, un

point d'interrogation suivi d'un espace est automatiquement affiché par INPUT.

Les variables qui suivent l'instruction INPUT peuvent être de n'importe quel type ; elles doivent être séparées les unes des autres par des virgules.

Les données fournies en réponse à un INPUT sont affectées aux variables dans l'ordre de la liste de celles-ci. Les espaces précédant ou suivant les chaînes de caractères sont ignorés ; s'ils ont une importance il faut placer les chaînes de caractères entre des guillemets. Si une chaîne de caractères contient une ou plusieurs virgules, il faut aussi placer la chaîne entre des guillemets.

Lorsque plusieurs variables suivent le INPUT, il faut fournir les données qui leurs seront affectées une par une ; chaque donnée étant séparée de la précédente par un retour chariot.

Un contrôle des données fournies est assuré par INPUT qui vérifie leur conformité avec les possibilités du Basic et avec le type de variable spécifiée après le INPUT. Une erreur est générée dans les cas suivants :

- Tentative de fournir plus d'une donnée à la fois
- Donnée chaîne de caractères alors qu'une variable numérique est prévue pour la recevoir.
- Fourniture d'un nombre de valeur absolue supérieure à 9.9 E + 127.

Vous pouvez consulter également les fiches des instructions ACCEPT, KEY1, KEY2, KEY$ et LINPUT.

**EXEMPLES :**

10 INPUT «FRAPPEZ TROIS NOMBRES »; A , B, C

fait afficher le message FRAPPEZ TROIS NOMBRES puis le calculateur attend trois nombres qui seront respectivement affectés à A pour le premier, B pour le second et C pour le troisième.



10 INPUT X$

fait afficher un point d'interrogation puis le calculateur attend une chaîne de caractères qui sera affectée à X$.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## INT

**FORMAT :** INT (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis la fonction INT donne le plus grand entier qui lui est immédiatement inférieur ou égal.

**EXEMPLES :**

10 PRINT INT ( 3.999999 )

donne 3 comme résultat lors de l'exécution.

10 PRINT INT ( -3.000001 )

donne -4 lors de l'exécution puisque -4 est le plus grand entier immédiatement inférieur à -3.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## INTRND

**FORMAT :** INTRND (expression numérique)

**DESCRIPTION :**

L'expression numérique est évaluée et est arrondie à l'entier le plus proche. INTRND donne alors une nombre pseudo aléatoire supérieur ou égal à 1 et inférieur ou égal à l'entier obtenu.
Un message d'erreur est généré si l'entier obtenu après évaluation de l'expression est inférieur à 1.

Cette fonction est équivalente à : INT ( RND * INT ( X + .5)) + 1 .

Vous pouvez consulter également les fiches des instructions RANDOMIZE ET RND.

**EXEMPLE :**

10 PRINT INTRND (6)

affiche un nombre pseudo aléatoire compris entre 1 et 6 simulant ainsi un jet de dé.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## KEY1 et KEY2

**FORMAT** : CALL KEY1 (variable n°1, variable n°2)
CALL KEY2 (variable n°1, variable n°2)

**DESCRIPTION :**

KEY1 et KEY2 sont deux sous programmes qui doivent être appelés par un CALL. Ils affectent à la variable n°1 le code ASCII de la touche du clavier ou d'une des manettes de jeux qui est pressée au moment de l'exécution de l'instruction. Si aucune touche n'est pressée, la variable n°1 est rendue égale à 255.
KEY1 doit être utilisé pour le clavier et pour la manette de jeux n°1 c'est-à-dire celle dont les poussoirs de tir sont oranges. KEY2 doit être utilisé pour la manette de jeux n°2 c'est-à-dire celle dont les poussoirs de tir sont blancs.

La variable n°2 est utilisée pour indiquer l'état de la touche actionnée de la façon suivante :
- Si la variable n°2 est rendue égale à 1 cela signifie qu'une nouvelle touche a été pressée depuis la dernière utilisation de CALL KEY.
- Si la variable n°2 est rendue égale à -1 cela signifie que la même touche que celle déjà pressée lors de la dernière utilisation de CALL KEY est encore actionnée.
- Si la variable n°2 est rendue égale à 0 cela signifie qu'aucune touche n'a été pressée.



**EXEMPLE :**

```
100 FOR I = 1 TO 1000
110 CALL KEY1(A,B)
120 PRINT I , A , B
130 NEXT I
```

permet de bien visualiser le fonctionnement de CALL KEY1.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## KEY$

**FORMAT** : KEY$

**DESCRIPTION :**

Cette instruction suspend l'exécution du programme et attend la frappe d'une touche du clavier. Dès que cette frappe a eu lieu, le programme continue son exécution et KEY$ est égale à la touche frappée. Contrairement à INPUT et LINPUT, KEY$ ne nécessite pas de retour chariot après la frappe de la touche.

Vous pouvez consulter également les fiches des instructions INPUT, KEY1 et KEY2 et LINPUT.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## LEN

**FORMAT :** LEN (expression chaîne de caractères)

**DESCRIPTION :**

La fonction LEN donne le nombre de caractères contenus dans l'expression chaîne de caractères spécifiée. Un espace compte pour un caractère.

**EXEMPLE :**

10 PRINT LEN («EXL 100»)

donne 7 comme résultat lors de l'exécution.

## LET

**FORMAT :** LET variable numérique = expression numérique
LET variable chaîne de caractères = expression chaîne de caractères

**DESCRIPTION :**

L'instruction LET affecte à la variable numérique ou chaîne de caractères (selon la forme employée) le résultat de l'évaluation de l'expression placée à droite du signe égal.
Plusieurs variables peuvent être exprimées dans une même commande LET ; elles sont alors placées les unes à la suite des autres séparées par des virgules. Dans ce cas, toutes les variables se voient affecter la valeur de l'expression placée à droite du signe égal. De ce fait, il est évident que toutes les variables doivent être du même type.

L'instruction LET est optionnelle et peut être omise.

**EXEMPLES :**

```
10 LET A = 2
20 PRINT A
```

fait afficher 2 lors de l'exécution.

```
10 LET A , B , C = 10
20 PRINT A ; B ; C
```

fait afficher 10 10 10 lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## LINE

**FORMAT :** CALL LINE («caractère», coordonnées début, coordonnées fin)

**DESCRIPTION :**

LINE est un sous programme qui doit être appelé par un CALL. Il permet de tracer une droite sur



l'écran en spécifiant la couleur, l'origine et l'extrémité de celle-ci.
Le caractère qui suit le CALL LINE permet de spécifier la couleur du tracé ; il est codifié comme indiqué dans la fiche de l'instruction CLS. Il peut être placé directement dans le CALL LINE ou être constitué par une variable chaîne de caractères.
Viennent ensuite les coordonnées des extrémités de la droite qui sont indiquées sous la forme colonne, ligne. Colonne peut évoluer de 0 à 319 inclus et ligne 0 à (10 x N) - 1 où N est le nombre de lignes d'écran mises en haute résolution.

La colonne 0 est située à gauche de l'écran et la ligne 0 en haut de l'écran.

Si la portion d'écran mise en haute résolution n'est pas assez grande pour recevoir le tracé demandé, les coordonnées sont réduites modulo la taille de l'écran et le tracé est effectué comme cela. Aucun message d'erreur n'est généré.

Vous pouvez consulter également les fiches des instructions HRON, HROFF et PLOT.

**EXEMPLE :**

```
100 CALL HRON («R»,1,15)
110 FOR I = 0 TO 300
120 CALL LINE («Y»,I,0,300-I,100)
130 NEXT I
```

dessine deux triangles jaunes opposés par le sommet sur un écran haute résolution rouge.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## LINPUT

**FORMAT :** LINPUT chaîne de caractères ; variable chaîne de caractères

**DESCRIPTION :**

Cette instruction permet de fournir une donnée à

un programme à partir du clavier. Lors de son exécution le programme s'arrête tant que la donnée n'a pas été fournie.

La chaîne de caractères qui suit le LINPUT peut être une variable chaîne de caractères auquel cas son nom est simplement écrit après LINPUT et est suivi d'un point virgule ; ce peut être aussi une chaîne de caractères qui doit alors être comprise entre deux guillemets et qui doit aussi être suivie par un point virgule.

La chaîne de caractères est affichée sur l'écran à partir de la position courante du curseur due à la précédente opération d'entrée/sortie ou à l'exécution d'une instruction de positionnement de celui-ci. Si la chaîne de caractères est inexistante, un point d'interrogation suivi d'un espace est automatiquement affiché par LINPUT.

La donnée qui doit être fournie en réponse à une instruction LINPUT est impérativement du type chaîne de caractères. Elle est constituée par la ligne entière frappée en réponse à l'instruction LINPUT, virgules, espaces et guillemets (s'il y en a) inclus. C'est le caractère retour chariot de fin de ligne qui termine cette chaîne. Ce caractère ne fait jamais partie de la chaîne.

Vous pouvez consulter également les fiches des instructions KEY$, KEY1, KEY2 et INPUT.

**EXEMPLE :**

10 LINPUT «NOM ET PRENOM : »; A$

fait afficher NOM ET PRENOM ; le calculateur attend alors votre réponse et affecte toute la ligne frappée à la variable chaîne de caractères A$.

## LIST

**FORMAT :** LIST
LIST numéro de ligne
LIST numéro - numéro
LIST «numéro d'équipement»

**DESCRIPTION :**

Cette commande, utilisable seulement en mode direct, permet de lister tout ou partie d'un programme selon la syntaxe employée.

LIST utilisé seul fait afficher toutes les lignes d'un programme du début à la fin.

LIST suivi par un numéro de ligne fait afficher la ligne spécifiée seulement.

LIST suivi par deux numéros de lignes séparés par un tiret fait lister le programme contenu entre ces deux numéros de lignes, extrémités comprises.

LIST suivi par un numéro de ligne suivi par un tiret fait lister le programme de la ligne spécifiée à la fin.

Pour suspendre un listing pendant son déroulement, frappez n'importe quelle touche du clavier; une nouvelle frappe faisant continuer le listing.

La forme LIST «numéro d'équipement» offre les mêmes possibilités que celles exposées ci-avant, mais le listing a lieu sur un équipement périphérique tel qu'une imprimante par exemple. L'imprimante série est le périphérique n° 10 et l'imprimante parallèle le n° 20. Plus de précisions sont fournies dans la notice de l'interface imprimante.

**EXEMPLES :**

LIST 100 - 200 fait lister les lignes 100 à 200 incluses.

LIST - 200 fait lister le programme du début à 200 incluses.

LIST 100 - fait lister le programme de 100 incluses à la fin.

LIST fait lister tout le programme.

## LOAD

**FORMAT :** LOAD "numéro,nom de programme"
LOAD "numéro,"

**DESCRIPTION :**

Cette commande permet de charger un programme en mémoire à partir d'un magnétophone à cassettes ou à partir d'une cassette de RAM CMOS. Elle efface tout programme éventuellement contenu dans la mémoire avant son exécution.
Le numéro à indiquer après la commande LOAD est, soit 1 pour un chargement à partir d'un magnétophone à cassettes, soit 2 pour un chargement à partir d'une cassette de RAM CMOS. Le nom de programme doit comporter quatre caractères et doit être identique à celui utilisé lors de la commande de sauvegarde SAVE. Si plusieurs programmes se trouvent sur la cassette avant le programme désiré, ils seront ignorés par la commande LOAD.
Si vous ne connaissez pas le nom de programme qui a été utilisé lors de la sauvegare, vous devez utiliser la deuxième forme de la commande LOAD présentée ci-dessus ; dans ce cas, le premier programme rencontré sera chargé en mémoire.

Afin de faciliter l'utilisation de cette commande un message apparaît sur la ligne de service en haut de l'écran et permet de suivre le déroulement de la commande LOAD.

L'EXL 100 ne disposant pas de télécommande de magnétophone, le message ESC → Lect. apparaît sur la ligne de service après la frappe de la commande LOAD. Il signifie que la lecture de la cassette par l'EXL 100 ne commencera qu'après la frappe de la touche ESC. Cela permet de préparer le magnétophone et de ne déclencher la lecture que lorsque tout est prêt.



En cas d'absence de signal ou de signal très insuffisant en provenance du magnétophone pendant une durée supérieure à 1 minute, la commande LOAD est interrompue. En cas d'erreur de chargement lors d'une commande LOAD, la commande est également interrompue et un message d'erreur est affiché. Ce sont les deux seules causes d'interruption possibles de la commande LOAD ; en particulier la touche BREAK (CTL C) est inactive pendant un LOAD.

Lors de la lecture, le message Lect. → apparaît sur la ligne de service en haut de l'écran. Dès qu'un programme est rencontré sur la cassette, et qu'il soit chargé ou non en mémoire, son nom s'affiche sur la ligne de service. Cela permet de retrouver des noms de programmes oubliés et de savoir où en est le déroulement de la cassette.

Vous pouvez consulter également la fiche de la commande SAVE.

**EXEMPLE :**

LOAD "1.ABCD"

charge le programme ayant pour nom ABCD à partir d'un magnétophone à cassettes.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## LOCATE

**FORMAT :** LOCATE (expression numérique n° 1, expression numérique n° 2)

**DESCRIPTION :**

Les expressions numériques contenues entre parenthèses sont évaluées et arrondies à l'entier le plus proche. Le curseur d'affichage alphanumérique est alors positionné sur la ligne spécifiée par l'expression n° 1 et sur la colonne spécifiée par l'instruction n° 2. Le numéro de ligne doit être compris entre 1 et 22 inclus, la ligne 1 étant située en

haut de l'écran. Le numéro de colonne doit être compris entre 1 et 40, la colonne 1 étant située à gauche de l'écran.

**EXEMPLE :**

```
100 LOCATE (12,15)
110 PRINT "BONJOUR"
```

affiche BONJOUR ligne 12 colonne 15 soit à peu près au centre de l'écran.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## LOG

**FORMAT :** LOG (expression numérique)

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée. Si sa valeur est strictement positive, la fonction LOG en donne le logarithme décimal. Si la valeur de l'expression est négative ou nulle, un message d'erreur est généré.

Vous pouvez consulter également la fiche de l'instruction LN.

**EXEMPLE :**

```
10 PRINT LOG( 2 )
```

donne .3010299957 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## NEW

**FORMAT :** NEW

**DESCRIPTION :**

La commande NEW, qui ne s'utilise qu'en mode direct, prépare le calculateur à recevoir un nouveau



programme. Elle efface la mémoire de programme ainsi que toutes les variables qui pouvaient être contenues en mémoire.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## NEXT

**FORMAT :** NEXT variable de boucle
NEXT

**DESCRIPTION :**

Cette instruction s'utilise conjointement avec FOR TO STEP pour former une boucle.

Si une variable de boucle est précisée derrière NEXT, elle doit avoir le même nom que celui utilisé dans l'instruction FOR TO STEP associée. En l'absence de variable de boucle précisée, NEXT agit avec le FOR TO STEP le plus proche ne disposant pas d'un NEXT associé.

Reportez-vous à la fiche de FOR TO STEP pour la description du fonctionnement des boucles ainsi que pour les règles d'imbrication.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## NUMBER

**FORMAT :** NUMBER
NUMBER numéro initial , pas
NUM
NUM numéro initial , pas

**DESCRIPTION :**

Cette commande, utilisable seulement en mode direct, permet d'activer la numérotation de lignes automatique. NUM ou NUMBER utilisée seule fait afficher 100 comme premier numéro de ligne et fait augmenter ce numéro de 10 en 10 pour chaque nouvelle ligne frappée. Le numéro de ligne est

automatiquement suivi d'un espace permettant ainsi de frapper aussitôt l'instruction désirée.

Si un numéro de ligne initial est spécifié, la numérotation commencera avec cette valeur et augmentera de 10 en 10 pour chaque nouvelle ligne. Enfin, il est possible de spécifier le numéro de ligne initial et le pas, c'est-à-dire, la différence entre deux numéros de lignes consécutifs.

Pour terminer la frappe d'un programme en mode de numérotation de lignes automatique, frappez un retour chariot immédiatement après l'affichage du numéro de ligne ou frappez BREAK ( CTL C ) lors de l'affichage de n'importe quelle ligne.

Vous pouvez également consulter la fiche de la commande RENUMBER.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## NUMERIC

**FORMAT :** NUMERIC (expression chaîne de caractères)

**DESCRIPTION :**

Cette fonction teste si l'expression chaîne de caractères qui la suit est une représentation valide (relativement aux règles de syntaxe de l'EXL 100) d'un nombre. NUMERIC est égal à -1 si la chaîne est une représentation valide et à 0 dans le cas contraire.

Les espaces situés en début et en fin de chaîne de caractères sont ignorés par NUMERIC et n'influent pas sur le résultat ; de ce fait, NUMERIC permet de savoir si VAL pourra fonctionner correctement sur la chaîne de caractères ainsi testée.

Vous pouvez consulter également la fiche de l'instruction VAL.



**EXEMPLE :**

```
100 LINPUT «FRAPPEZ UN NOMBRE :»;A$
110 IF NOT NUMERIC (A$) THEN LINPUT
«ERREUR , RECOMMENCEZ :»;A$ : GOTO 110
120 A = VAL (A$)
```

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ON BREAK

**FORMAT :** ON BREAK STOP
ON BREAK NEXT
ON BREAK ERROR

**DESCRIPTION :**

L'instruction ON BREAK permet de définir le comportement qu'aura l'EXL 100 en présence d'un point d'arrêt. En l'absence d'instruction ON BREAK dans un programme, l'option ON BREAK STOP est automatiquement sélectionnée.

ON BREAK STOP qui est l'option «normale de ON BREAK a pour effet de faire arrêter l'exécution d'un programme à chaque point d'arrêt comme expliqué dans la fiche de l'instruction BREAK.

ON BREAK NEXT permet d'ignorer les points d'arrêt définis sous la forme BREAK numéro de ligne ; par contre, s'il existe dans le programme des points d'arrêt sous forme d'instructions BREAK, ceux-ci sont pris en compte et traités comme dans le cas d'un ON BREAK STOP. La touche BREAK ( CTL C ) est également ignorée suite à l'exéction d'un ON BREAK NEXT.

ON BREAK ERROR permet de traiter les points d'arrêt comme des erreurs ce qui autorise l'emploi de ON ERROR. Reportez-vous à la fiche de l'instruction ON ERROR pour plus de détails sur son emploi.

Une instruction ON BREAK XXX reste active tant qu'une autre instruction ON BREAK n'a pas été exécutée.

Vous pouvez consulter également les fiches des instructions BREAK, ON ERROR et UNBREAK.

**EXEMPLE :**

```
100 BREAK 140 place un point d'arrêt ligne 140
110 ON BREAK NEXT
120 BREAK
130 FOR A = 1 TO 500
140 PRINT «BREAK EST INACTIF»
150 NEXT A
160 ON BREAK STOP
170 FOR A = 1 TO 500
180 PRINT «BREAK FONCTIONNE»
190 NEXT A
```

ce programme rend inactif les BREAK suivis d'un numéro de ligne et la touche BREAK de la ligne 110 à la ligne 160 ce qui est mis en évidence par l'impossibilité d'interrompre la boucle des lignes 130 à 150. Par contre, le BREAK contenu ligne 120 est exécuté (frappez CON Ou CONTINUE pour continuer le programme).
Ligne 160 Le fonctionnement normal des points d'arrêt est rétabli ce que vous pouvez vérifier en interrompant la boucle des lignes 170 à 190 au moyen de la touche BREAK ( CTL C ).

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ON ERROR

**FORMAT :** ON ERROR STOP
ON ERROR numéro de ligne

**DESCRIPTION :**

Cette instruction permet de définir le comportement qu'aura le calculateur lorsqu'une erreur aura lieu pendant l'exécution d'un programme. Toute erreur survenant dans un programme au delà de la ligne contenant l'instruction ON ERROR sera traitée compte tenu de l'option choisie parmi les deux proposées.



ON ERROR STOP est le mode normal de traitement des erreurs ; c'est-à-dire que, lorsqu'une erreur a lieu, le programme s'arrête et un message d'erreur est affiché précisant le type d'erreur et le numéro de ligne où celle-ci est intervenue. Ce mode est automatiquement sélectionné lorsque l'on lance l'exécution d'un programme et il est inutile d'écrire ON ERROR STOP à son début. L'instruction ON ERROR STOP s'utilise, par contre, lorsque l'on souhaite rétablir ce mode de traitement d'erreurs au sein d'un programme ou un ON ERROR numéro de ligne a été utilisé préalablement. Il suffit alors de placer ON ERROR STOP sur la ligne à partir de laquelle on souhaite rétablir ce mode normal.

ON ERROR numéro de ligne permet de définir, dans un programme, son propre mode de traitement d'erreurs. Lorsqu'une erreur est détectée après qu'une ligne contenant un ON ERROR numéro de ligne ait été exécutée, le programme saute au numéro de ligne spécifié après le ON ERROR. Les instructions s'y trouvant sont exécutées en séquence jusqu'à rencontrer une instruction RETURN qui fait reprendre l'exécution du programme à l'endroit où il avait été interrompu par l'erreur (diverses options sont possibles).
Pendant l'exécution du sous programme de traitement d'erreur, l'option ON ERROR STOP est automatiquement activée à moins que la première instruction du sous programme soit elle-même un autre ON ERROR numéro de ligne.

Différentes options existent au niveau du RETURN qui termine le sous programme de traitement d'erreurs, consultez la fiche RETURN pour plus de précisions à ce sujet.

Dans le sous programme de traitement d'erreur, l'utilisation de CALL ERR permet de savoir quel type d'erreur a eu lieu et de prendre les décisions qui en découlent.

Une instruction ON ERROR numéro de ligne reste valide, au sein d'un programme, tant qu'une nouvelle instruction ON ERROR ne vient pas définir un nouveau numéro de ligne ou le passage ne mode ON ERROR STOP.

Vous pouvez consulter également les fiches des instructions ERR, ON BREAK, ON WARNING et RETURN.

**EXEMPLE :**

```
10 ON ERROR 100
20 INPUT "FRAPPEZ UN NOMBRE ";A
30 . . . . . . . . . . . . . . . . . . . . . . . . .
100 CALL ERR (CODE,TYPE,FICH,LINE)
110 IF LINE < > 20 THEN RETURN 220
120 IF CODE < > 07 THEN RETURN 220
130 PRINT "REPONSE INCORRECTE - RECOMMENCEZ !"
140 RETURN
220 PRINT "ERREUR ";CODE; " LIGNE ";LINE
```

Dans ce programme, un sous programme de traitement d'erreurs est défini ligne 10 comme commençant ligne 100. Lors d'une erreur (par exemple lors de l'entrée de donnée ligne 20) le programme exécutera le sous programme de gestion d'erreur. Au moyen du sous programme ERR, le type d'erreur sera déterminé ; s'il n'est pas causé par la frappe d'une mauvaise réponse ligne 20, la ligne 220 fera afficher le numéro d'erreur et la ligne en erreur et les choses en resteront là. Si l'erreur vient bien de la ligne 20, la ligne 130 fera afficher un message approprié et relancera l'exécution du programme.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ON GOSUB

**FORMAT :** ON expression numérique GOSUB ligne n°1, ligne n°2, ...



**DESCRIPTION :**

Cette instruction permet de déterminer quel sous programme sera appelé par le GOSUB qu'elle contient de la façon suivante. L'expression numérique est évaluée et seule la partie entière du résultat obtenu est conservée. Si cette partie entière est 1, le sous programme commençant à la ligne spécifiée par ligne n°1 est exécuté ; si cette partie entière est 2, le sous programme commençant à la ligne spécifiée par ligne n°2 est exécuté ; et ainsi de suite. Si la partie entière de l'expression est nulle, négative ou supérieure au nombre total de numéros de ligne se trouvant après le GOSUB, une erreur est générée.
Les différents numéros de lignes placés après le GOSUB doivent être séparés les uns des autres par des virgules.

Le RETURN de fin de sous programme fait continuer l'exécution du programme principal avec l'instruction suivant le ON GOSUB.

Vous pouvez consulter également les fiches des instructions GOSUB et RETURN.

**EXEMPLE :**

```
10 ON A GOSUB 30,150,20,2000
```

fait exécuter le sous programme commençant ligne 30 si A vaut 1, ligne 150 si A vaut 2, ...., ligne 2000 si A vaut 4. Si A est négatif, nul ou supérieur à 4, il y a génération d'une erreur.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ON GOTO

**FORMAT :** ON expression numérique GOTO ligne n°1, ligne n°2, ....

**DESCRIPTION :**

Cette instruction permet de déterminer à quelle

ligne d'un programme va avoir lieu le saut inconditionnel réalisé par le GOTO qu'elle contient. L'expression numérique est évaluée et seule la partie entière du résultat obtenue est conservée. Si cette partie entière vaut 1, le saut a lieu au numéro de ligne spécifié par ligne n°1 ; si cette partie entière vaut 2, le saut a lieu au numéro de ligne spécifié par ligne n°2 ; et ainsi de suite. Si la partie entière de l'expression est nulle, négative ou supérieure au nombre de numéros de lignes disponibles, une erreur est générée.

Vous pouvez consulter également la fiche de l'instruction GOTO.

**EXEMPLE :**

10 ON A GOTO 100 , 40 , 200, 1500

fait poursuivre l'exécution du programme ligne 100 si A vaut 1, ligne 40 si A vaut 2, ..., ligne 1500 si A vaut 4. Si a est nul, négatif ou supérieur à 4, une erreur est générée.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## ON WARNING

**FORMAT :** ON WARNING PRINT
ON WARNING NEXT
ON WARNING ERROR

**DESCRIPTION :**

L'instruction ON WARNING permet de définir le comportement qu'aura l'EXL 100 si. lors de l'exécution d'un programme, un warning a lieu. Un warning est un message, analogue à un message d'erreur, mais qui indique une erreur non fatale pour la suite du programme. Un warning peut avoir lieu par exemple si l'utilisation d'un RPT$ conduit à une chaîne de plus de 255 caractères ; la chaîne obtenue est tronquée, un message est affiché mais on peut tout de même décider de poursuivre l'exécution du programme.



ON WARNING PRINT qui est l'option "normale" du ON WARNING a pour effet de faire afficher les messages de warning et autorise la poursuite du programme par la frappe d'un retour chariot. Cette option est sélectionnée automatiquement lors du lancement d'un programme s'il ne contient aucune option contraire.

ON WARNING NEXT impose à l'EXL 100 de ne pas afficher les messages de warning éventuels qui pourraient exister. L'exécution du programme n'est donc pas perturbée même si des warnings ont lieu.

ON WARNING ERROR permet de traiter les warnings comme des erreurs ce qui autorise ensuite l'emploir de ON ERROR. Reportez-vous à la fiche de l'instruction ON ERROR pour plus de détails sur son emploi.
Une instruction ON WARNING XXX reste active tant qu'une autre instruction ON WARNING n'a pas été exécutée.

Vous pouvez consulter également la fiche de l'instruction ON ERROR.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## OPEN

**FORMAT :** OPEN # numéro de fichier, "numéro d'équipement"

**DESCRIPTION :**

Cette instruction permet d'associer un équipement externe à l'EXL 100 tel que imprimante, cassette de RAM CMOS, etc ... à un numéro de fichier qui peut ensuite être utilisé dans une instruction PRINT par exemple.
Le numéro de fichier doit être compris entre 1 et 255 et le numéro d'équipement dépend du type de

celui-ci. L'imprimante série porte le numéro d'équipement 10 et l'imprimante parallèle le numéro 20.

L'association reste valide pendant toute la durée du programme ou jusqu'à l'exécution d'une instruction CLOSE.

Après exécution d'un CLOSE, le numéro de fichier est de nouveau disponible et peut être associé à n'importe quel équipement.

Un certain nombre d'options peuvent être placées après l'instruction OPEN. Certaines sont propres à des équipements et sont décrites dans la notice de ceux-ci ; d'autres sont propres à l'utilisation de fichiers de données sur cassettes et sont décrites ci-après :
- OUTPUT spécifie que le fichier concerné par OPEN va être ouvert en écriture.
- INPUT spécifie que le fichier concerné par OPEN va être ouvert en lecture.
- VARIABLE N permet de modifier la taille des enregistrements. Par défaut celle-ci est de 80 caractères mais peut prendre n'importe quelle valeur spécifiée par N.
Plus d'informations à ce sujet sont données en fin de chapitre 14 dans la deuxième partie de ce manuel.

**EXEMPLES :**

```
100 OPEN #2,"20"
110 PRINT #2,"BONJOUR"
```

fait afficher BONJOUR sur l'imprimante parallèle.

```
100 OPEN #4,"1.DATA",OUTPUT,VARIABLE
100
```

ouvre le fichier DATA en écriture et fixe la taille des enregistrements à 100 caractères.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## PAUSE

**FORMAT :** PAUSE expression numérique



**DESCRIPTION :**

L'instruction PAUSE suspend l'exécution du programme pendant un certain temps ou jusqu'à la frappe d'un retour chariot. L'expression numérique qui suit PAUSE détermine les divers modes de fonctionnement.

Si PAUSE est utilisée seule, la durée de l'interruption est indéterminée et ne prend fin qu'avec la frappe d'un retour chariot.

Si l'expression numérique est présente, PAUSE suspend l'exécution du programme pendant un nombre de secondes égal à la valeur absolue de l'expression. Si l'expression est négative, ce temps ne peut être modifié ; par contre, si l'expression est positive, la frappe d'un retour chariot pendant l'interruption fait reprendre immédiatement l'exécution du programme.
Si la valeur absolue de l'expression est inférieure à 0.1, elle est considérée comme nulle et PAUSE est sans effet.

**EXEMPLE :**

```
100 FOR I = O TO 100
110 PRINT I;
120 PAUSE .9
130 NEXT
```

réalise un compteur de secondes rudimentaire.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## PEEK

**FORMAT :** CALL PEEK (adresse, variable 1, variable 2, ...)

**DESCRIPTION :**

Ce sous programme permet de lire directement en mémoire, à partir de l'adresse spécifiée, la ou les

variables exprimées. L'adresse doit être exprimée en décimal et être comprise entre 0 et 65535. Les variables doivent être numériques et les valeurs qui leurs seront assignées seront comprises entre 0 et 255. Si plusieurs variables sont utilisées, elles sont séparées les unes des autres par des virgules. L'adresse est séparée de la première variable qui lui fait suite par une virgule.
Lorsque plusieurs variables sont utilisées, elles sont lues en mémoire les unes après les autres, dans le même ordre que leur ordre d'écriture dans l'instruction PEEK. La variable 1 est lue à l'adresse indiquée, la variable 2 à l'adresse indiquée augmentée de 1 et ainsi de suite.

Vous pouvez consulter également la fiche de l'instruction POKE.

**EXEMPLE :**

10 CALL PEEK ( 4096 , A , B , C )

lit le contenu des mémoires d'adresses 4096 , 4097 et 4098 et affecte celui-ci respectivement à A , B et C.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## PI

**FORMAT :** PI

**DESCRIPTION :**

La fonction PI donne la valeur de PI soit 3.14159265359.

**EXEMPLE :**

```
10 INPUT "RAYON DU CERCLE ? ";R
20 PRINT "SURFACE DU CERCLE : "; PI * R * R
```

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■



## PLOT

**FORMAT :** CALL PLOT ("caractère", coordonnées d'un point)

**DESCRIPTION :**

PLOT est un sous programme qui doit être appelé par un CALL. PLOT permet de définir la couleur d'un point et de "l'allumer" (si sa couleur est différente de celle du fond) ou de "l'éteindre" (si sa couleur est identique à celle du fond).
La couleur du point est spécifiée par le caractère qui suit le PLOT selon la codification définie dans la fiche de l'instruction CLS. Ce caractère peut être exprimé directement ou sous forme d'une variable chaîne de caractères.

Les coordonnées du point sont spécifiées sous la forme colonne, ligne ou colonne est compris entre 0 et 319 et ligne entre 0 et (10 x N) -1, N étant le nombre de lignes d'écran mises en mode haute résolution. Le point de coordonnées 0,0 est situé en haut à gauche de l'écran.

Si les coordonnées du point sont incompatibles de la taille d'écran mise en haute résolution, elles sont réduites modulo cette taille et l'instruction PLOT est exécutée en fonction des valeurs ainsi obtenues.

Vous pouvez consulter également les fiches des instructions HRON, HROFF et LINE.

**EXEMPLE :**

```
100 DEG
110 CALL HRON ("R",1,11)
120 FOR I = 0 TO 319
130 J = I * 4
140 CALL PLOT ("Y",I,50 - 50 * SIN (J))
150 NEXT I
```

dessine une portion de sinusoïde jaune sur un fond d'écran haute résolution rouge.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## POKE

**FORMAT :** CALL POKE (adresse, variable1, variable 2, ...)

**DESCRIPTION :**

Ce sous programme permet d'écrire directement en mémoire, à partir de l'adresse spécifiée, la ou les variables exprimées. L'adresse doit être exprimée en décimal et être comprise entre 0 et 65535. Les variables doivent être numériques et les valeurs qui leurs sont assignées doivent être comprises entre 0 et 255. Si plusieurs variables sont utilisées, elles sont séparées les unes des autres par des virgules. L'adresse est séparée de la première variable qui lui fait suite par une virgule.

Lorsque plusieurs variables sont utilisées, elles sont placées en mémoire les unes après les autres, dans le même ordre que leur ordre d'écriture dans l'instruction PEEK. La variable 1 est placée à l'adresse indiquée, la variable 2 à l'adresse indiquée augmentée de 1 et ainsi de suite.

Attention ! Une utilisation irréfléchie de cette instruction peut rendre inutilisable un programme contenu en mémoire et nécessiter un RESET de l'appareil. Evitez d'employer cette instruction dans la mémoire vive réservée aux variables système

Vous pouvez consulter également la fiche de l'instruction PEEK.

**EXEMPLE :**

10 POKE ( 4096 , A , B , C)



écrit dans les mémoires d'adresse 4096, 4097 et 4098 les valeurs affectées respectivement aux variables A, B et C.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## POS

**FORMAT :** POS (chaîne de caractères n°1, chaîne de caractères n°2, expression numérique)

**DESCRIPTION :**

L'expression numérique est évaluée, à la suite de quoi la fonction POS fournit la position de la première apparition de la chaîne de cractères n°2 dans la chaîne n°1. La recherche commence à partir de la position spécifiée par la valeur de l'expression numérique.
Si la chaîne n°2 ne peut être trouvée dans la chaîne n°1, la fonction POS fournit un résultat nul.
Les positions de caractères sont repérées à partir de la gauche et commencent à 1.

**EXEMPLES :**

100 PRINT POS («PAN»,«A»,2)

fait afficher 2 car A est la deuxième lettre de PAN et que la recherche commence à partir de la première position.

100 PRINT POS («APAN»,«A»,2)

fait afficher 3 car A est la troisième lettre et que la recherche commence à partir de la deuxième lettre, le premier A est donc ignoré.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## PRINT

**FORMAT :** PRINT liste de constantes et variables
PRINT # numéro de fichier, liste de constantes et variables.

**DESCRIPTION :**
L'instruction PRINT permet d'afficher des informations sur l'écran, informations qui peuvent être des constantes numériques, des chaînes de caractères, des variables numériques et des variables chaînes de caractères. Ces divers éléments à afficher sont séparés les uns des autres par divers types de séparateurs ayant une action sur le positionnement de ces éléments sur l'écran.
Les divers éléments contenus dans une ligne de PRINT sont traités dans leur ordre d'apparition, de gauche à droite.
PRINT peut aussi être utilisé avec un numéro de fichier préalablement défini par une instruction OPEN ; dans ce cas, l'affichage n'a pas lieu sur l'écran mais sur l'équipement ainsi défini (généralement une imprimante).

L'instruction PRINT utilisée seule, sans élément à imprimer ni séparateur, fait afficher une ligne vide.

Les expressions chaînes de caractères sont évaluées afin de produire une chaîne globale qui est alors affichée. Les chaînes de caractères utilisées directement doivent être placées entre des guillemets. L'instruction PRINT n'ajoute aucun espace avant ou après l'affichage d'une chaîne de caractères, il faut donc inclure ceux-ci dans la chaîne s'ils sont désirés.

Les expressions numériques sont évaluées afin de produire un nombre qui est alors affiché en respectant les règles décrites ci-après. Le dernier chiffre du nombre est suivi d'un espace.

Il doit y avoir au moins un séparateur entre deux informations consécutives à afficher. Les séparateurs peuvent être des virgules ou des points virgules ; leur signification est différente et est décrite ci-dessous.

Un point virgule placé entre deux éléments fait afficher ceux-ci l'un à la suite de l'autre sans espace



entre eux autre que celui ou ceux éventuellement contenus dans les éléments eux-mêmes.

Une virgule fait afficher l'élément qui la suit au début du champ d'affichage suivant. Les champs d'affichage sont des zones de 10 caractères de longueur qui commencent en colonnes 1, 11, 21 et 31. Si le nombre de virgules utilisées est tel que le dernier champ d'affichage est dépassé, l'affichage continue sur la ligne suivante.

Si le dernier élément à imprimer d'une instruction PRINT n'est suivi par aucun séparateur, la place disponible sur la ligne d'affichage utilisée est débarassée de tout caractère. L'exécution d'une instruction PRINT suivant ou d'un INPUT avec affichage de chaîne de caractères aura lieu sur la ligne qui suit.

Si le dernier élément à afficher est suivi d'un séparateur, le curseur reste sur la dernière position utilisée sur la ligne et, lors du PRINT suivant ou d'un INPUT avec affichage de chaîne de caractères, l'affichage continue sur la même ligne en respectant la fonction du séparateur (pas d'espace si c'est un point virgule, saut au champ suivant si c'est une virgule).

Ces divers positionnements peuvent également être obtenus ou être modifiés en incluant une instruction TAB au sein des éléments à imprimer. La fiche de l'instruction TAB donne toutes indications à ce sujet.

Compte tenu de leur taille, les nombres peuvent être affichés sous forme décimale ou en notation scientifique. Dans le cas de la forme décimale, les règles ci-après sont respectées :
- Les entiers sont affichés sans point décimal
- Les nombres fractionnaires sont affichés avec un point décimal. Les zéros non significatifs sont éliminés. Si le nombre comporte plus de dix chiffres significatifs, il est arrondi à dix chiffres.

- Un nombre dont la valeur absolue est inférieure à 1 est affiché sans 0 significatif à gauche du point décimal.

Dans le cas de la notation scientifique, les règles ci-après sont respectées :
- Les nombres sont affichés sous la forme mantisse E exposant.
- La mantisse comporte au maximum sept chiffres avec un chiffre et un seul à gauche du point décimal.
- Les zéros non significatifs de la mantisse ne sont pas affichés.
- Les exposants sont affichés avec un signe plus ou moins et deux ou trois chiffres significatifs.
- Lorsque l'exposant est à deux chiffres, la mantisse est limitée à sept chiffres, lorsque l'exposant est à trois chiffres, elle est limitée à six chiffres.

Vous pouvez consulter également les fiches des instructions INPUT, LINPUT, LOCATE, PRINT USING et TAB.

**EXEMPLES :**

```
10 PRINT "EXL"; 100
```

fait afficher EXL 100 alors que :

```
10 PRINT "EXL",100
```

fait afficher EXL  100.

```
10 PRINT "BONJOUR, COMMENT";
20 PRINT "ALLEZ VOUS ?"
```

fait afficher BONJOUR, COMMENT ALLEZ-VOUS ?

```
10 PRINT "DONNEZ";
20 INPUT "VOTRE NOM :";A$
```

fait afficher DONNEZ VOTRE NOM :.



## PRINT USING

**FORMAT :**
PRINT USING chaîne de caractères, liste de constantes et variables
PRINT USING n° de ligne, liste de constantes et variables
PRINT USINGn° de fichier, n° de ligne, liste de constantes et variables
PRINT USING n° de fichier, chaîne de caractères, liste de constantes et variables.

**DESCRIPTION :**

Cette instruction fonctionne comme l'instruction PRINT présentée dans la fiche précédente avec des possibilités de formatage des informations affichées plus importantes grâce à l'ajout de USING et de la chaîne de caractères qui suit. Par ailleurs, la présence de USING annule la signification particulière des virgules utilisées en tant que séparateurs et fait traiter ces dernières comme des points virgules.
La forme PRINT USING numéro de fichier permet comme pour PRINT de diriger l'affichage sous un équipement autre que l'écran et préalablement associé à un numéro de fichier par une instruction OPEN.

La chaîne de caractères qui suit le USING peut être définie directement dans l'instruction PRINT USING utilisée sous la première forme présentée ou être spécifiée dans une ligne d'instruction IMAGE dont le numéro sera alors placé après le USING dans la deuxième forme présentée. Dans les deux cas, la chaîne de caractères utilisée après USING ou après IMAGE suit les règles exposées ci-après.

La chaîne de caractères se compose de caractères dits littéraux et de caractères de formatage revêtant une signification particulière. Le format

d'affichage peut s'appliquer à cinq types de données différentes présentées ci-après.
Les caractères de formatage sont le dièse (#), le point décimal (.) et le symbole d'élévation à une puissance (↑).
Le dièse définit l'emplacement d'un chiffre dans le format d'affichage. Le point décimal permet de spécifier la position du point décimal alors que le symbole d'élévation à une puissance spécifie la position des chiffres de l'exposant lors d'une représentation en notation scientifique.

Pour l'affichage des entiers :
- Jusqu'à 14 chiffres peuvent être demandés.
- La position de chaque chiffre est matérialisée par le symbole
- Lorsque le nombre ne remplit pas toute la place définie par les symboles #, il est décalé sur la droite de la zone.
- Lorsque le nombre est trop long pour la place définie par les symboles #, il n'est pas affiché mais est remplacé par des astérisques (*).
- Un nombre fractionnaire est arrondi à l'entier le plus proche.
- Lorsque le nombre est négatif, la position d'un chiffre est occupée par le signe moins (—).

Pour l'affichage des nombres fractionnaires :
- Jusqu'à 14 chiffres peuvent être demandés.
- La position de chaque chiffre est matérialisée par le symbole et celle du point décimal par le point décimal lui-même . Ce dernier peut être placé en n'importe quelle position.
- Lorsque la partie entière du nombre est plus longue que la place qui lui est attribuée compte tenu de la position du point décimal , le nombre n'est pas affiché et est remplacé par des astérisques (*).
- Lorsque le nombre est négatif, il faut prévoir au minimum un symbole # avant le point décimal pour recevoir le signe moins.

Pour l'affichage des nombres en notation scientifique :



- Jusqu'à 14 chiffres peuvent être demandés.
- Un nombre en notation scientifique est constitué par une zone du type entier ou nombre fractionnaire qui représente la mantisse et d'une zone qui représente l'exposant. Cette dernière zone est matérialisée par un certain nombre de symboles d'élévation à une puissance.
- Le nombre est arrondi conformément au format imposé pour la mantisse.
- Quel que soit le signe de la mantisse il faut prévoir au moins un symbole # avant le point décimal pour recevoir le signe moins si la mantisse est négative et un espace si la mantisse est positive.

Pour l'affichage des chaînes de caractères :
- La taille de la zone n'est limitée que par la taille de la chaîne de caractères de définition de format qui suit le USING.
- Une zone chaîne de caractères peut être définie au moyen d'un quelconque des trois symboles ., = ou ↑.
- Si une chaîne est plus courte que la taille de la zone qui lui a été réservée, elle est décalée vers la gauche.
- Si une chaîne est plus longue que la taille de la zone qui lui a été réservée, elle n'est pas affichée et est remplacée par des astérisques (*).

Pour l'affichage des zones littérales :
- La taille de la zone n'est limitée que par la taille de la chaîne de caractères qui suit le USING.
- Une zone littérale est consituée par n'importe quels caractères autres que =,. et ↑.
- Les zones littérales apparaissent sur l'écran exactement comme elles sont écrites après le USING.

Vous pouvez consulter également les fiches des instructions PRINT et IMAGE.

**EXEMPLES :**

```
10 PRINT USING "LES DEUX NOMBRES SONT
# # #   ET # #   . # # ", A, B
```

donne les résultats suivants en fonction des valeurs de A et B

| A | B | AFFICHAGE OBTENU | |
|---|---|---|---|
| −99 | −9.99 | LES DEUX NOMBRES SONT | −99 et −9.99 |
| −7 | −3.459 | LES DEUX NOMBRES SONT | − 7 ET −3.46 |
| 0 | 0 | LES DEUX NOMBRES SONT | 0 ET .00 |
| 14.8 | 12.75 | LES DEUX NOMBRES SONT | 15 ET 12.75 |
| 795 | 852 | LES DEUX NOMBRES SONT | 795 ET ***** |
| −948 | 64.7 | LES DEUX NOMBRES SONT | *** ET 64.70 |

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RAD

**FORMAT :** RAD

**DESCRIPTION :**

L'instruction RAD définit les radians comme unité angulaire pour tous les calculs ultérieurs. Cette définition ne peut être annulée que par l'exécution d'une instruction DEG ou GRAD. De plus, cette unité est automatiquement choisie après toute réinitialisation de l'EXL 100.

Le mot RADIAN est affiché sur la ligne de service située en haut d'écran lorsque cette unité est sélectionnée.

Vous pouvez consulter également les fiches des instructions DEG et GRAD.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RANDOMIZE

**FORMAT :** RANDOMIZE (expression numérique) ou RANDOMIZE

**DESCRIPTION :**

L'instruction RANDOMIZE fait générer une séquence totalement imprévisible au générateur de nombres pseudo aléatoires interne.



Si RANDOMIZE est suivi par une expression numérique, celle-ci est évaluée et est utilisée par le générateur de nombres pseudo aléatoires interne. Tout appel de RANDOMIZE ultérieur avec la même expression numérique fait générer la même séquence de nombres pseudo aléatoires.
Des expressions numériques différentes donnent, par contre, des séquences différentes.

Vous pouvez consulter également les fiches des instructions INTRND et RND.

**EXEMPLE :**

```
100 INPUT "FRAPPEZ UN NOMBRE : ";N
110 RANDOMIZE N
120 FOR A = 1 TO 10
130 PRINT A; RND
140 NEXT A
150 GOTO 100
```

met en évidence le rôle de RANDOMIZE dans la génération de nombres pseudo aléatoires.

## READ

**FORMAT :** READ liste de variables

**DESCRIPTION :**

L'instruction READ est utilisée conjointement à l'instruction DATA pour affecter des valeurs à des variables. Lorsque l'instruction READ est exécutée, les variables qui la suivent se voient affecter les valeurs qui suivent le ou les diverses instructions DATA du programme selon les règles ci-après.
La liste de variables qui suit l'instruction READ est constituée par des variables numériques et (ou) chaînes de caractères sans restriction. Les variables doivent être séparées les unes des autres par des virgules.

L'instruction READ commence à lire les données de la première instruction DATA du programme et

affecte les valeurs qu'elle y trouve aux variables de sa liste en en respectant l'ordre. Si le nombre de données contenu après la première instruction DATA n'est pas suffisant, READ utilise le DATA suivant par ordre de numéro de ligne. Si le nombre de données est plus important que le nombre de variables, toute instruction READ ultérieure à celle qui vient d'être exécutée continuera la lecture des données à partir de la première donnée non encore lue. Ce comportement peut être modifié par l'exécution d'une instruction RESTORE.

Si le nombre de variables est supérieur au nombre total de données contenu dans toutes les instructions DATA d'un programme, une erreur est générée. Si les types des variables attendues après un READ ne correspondent pas aux types de données trouvées après le DATA qui doit normalement être utilisé, une erreur est également générée. Attention, si une variable chaîne de caractères est attendue, la donnée lui correspondant peut être un nombre qui sera alors traité comme étant une chaîne de caractères.

Vous pouvez consulter également les fiches des instructions DATA et RESTORE.

**EXEMPLE :**

Voir la fiche de l'instruction DATA.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RELEASE

**FORMAT :** RELEASE nom de sous-programme n° 1, nom de sous-programme n° 2...

**DESCRIPTION**

Cette instruction est le contraire de l'instruction ATTACH c'est-à-dire qu'elle annule les préservations de valeurs de variables utilisées par les sous-programmes concernés.



L'utilisation de RELEASE avec un sous-programme fréquemment utilisé réduit la vitesse d'exécution mais réduit aussi l'occupation mémoire puisqu'il n'est plus nécessaire de réserver de place pour les variables utilisées par le sous-programme.

Vous pouvez consulter également la fiche de l'instruction ATTACH.
EXEMPLE :

```
100 ATTACH X
110 FOR J= 1 TO 5
120 CALL X
130 NEXT J
```

fait afficher 0 1 2 3 4 car les variables sont préservées lors de l'appel du sous-programme X.

```
140 RELEASE X
150 FOR J = 1 TO 5
160 CALL J
170 NEXT J
```

fait afficher 0 0 0 0 car, à cause du RELEASE, les variables ne sont plus préservées lors de l'appel du sous-programme X et sont donc ré initialisées à chaque appel.

```
180 SUB X
190 PRINT J ;
200 J = J + 1
210 SUBEND
```

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## REM

**FORMAT :** REM texte quelconque

**DESCRIPTION :**

Cette instruction permet de placer des commentaires dans un programme; commentaires qui seront visibles, après le REM, sur le listing. L'instruction REM est ignorée lors de l'exécution du pro-

gramme mais elle consomme tout de même de la place en mémoire pour ranger le texte qui la suit. Ce texte peut être quelconque et contenir tous les caractères disponibles, y compris le séparateur d'instructions (:); en conséquence REM doit toujours être la dernière instruction d'une ligne.

L'instruction REM peut être abrégée par un point d'exclamation (!). Dans ce cas, si l'instruction REM représentée par un point d'exclamation est utilisée sur la même ligne qu'une autre instruction, le séparateur d'instruction (:) est inutile.

Un banchement (GOTO, GOSUB, IF THEN ELSE, et...) peut être effectué sur une ligne commençant par un REM. Le REM sera ignoré et la ligne suivante sera exécutée.

**EXEMPLES :**

```
10 GOSUB 100
. . . . . . . . . .
100 REM DEBUT DU SOUS-PROGRAMME
.....................................................
ou
10 A = COS (B) ! CALCUL DU COSINUS DE B
```

## RENUMBER

**FORMAT :** RENUMBER
RENUMBER ligne initiale, pas
REN
REN ligne initiale, pas

**DESCRIPTION :**

Cette commande utilisable seulement en mode direct permet de refaire la numérotation des lignes d'un programme. Utilisée seule, RENUMBER ou REN refait la numérotation des lignes en partant de 100 et en augmentant les numéros de lignes de 10 en 10.
Il est possible de spécifier un numéro de ligne initial, auquel cas la numérotation part de cette valeur avec une augmentation de 10 en 10.



Il est également possible de spécifier un numéro de ligne initial et un pas auquel cas la numérotation part de cette valeur et augmente du pas spécifié à chaque ligne.
Toutes les références à des numéros de lignes sont automatiquement prises en compte et modifiées en conséquence par RENUMBER.

Si, du fait des valeurs choisies implicitement ou explicitement, l'utilisation de RENUMBER conduit à un numéro de ligne supérieur à 32766, un message d'erreur est généré et le programme n'est pas modifié.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RESTORE

**FORMAT :** RESTORE
RESTORE numéro de ligne

**DESCRIPTION :**

L'instruction RESTORE est utilisée pour modifier l'ordre de lecture des données qui suivent un DATA au moyen de l'instruction READ.

Lors de l'exécution d'une instruction RESTORE, l'instruction READ suivante lira les données à partir du DATA se trouvant au numéro de ligne spécifié après l'instruction RESTORE. Si aucun numéro de ligne n'est indiqué, la première ligne DATA du programme sera utilisée. Si le numéro de ligne indiqué ne correspond pas à une ligne contenant une instruction DATA, la ligne suivante la plus proche contenant un DATA sera utilisée.

Vous pouvez consulter également les fiches des instructions DATA et READ.

**EXEMPLES :**

```
10 DATA 10,20,30,40
20 READ A,B
```

```
30 RESTORE
40 READ C,D
50 PRINT A;B;C;D
```

donne 10 20 10 20 comme résultat lors de l'exécution.

```
10 DATA 10,20,30,40
20 READ A,B
30 RESTORE 100
40 READ C,D
50 PRINT A;B;C;D
. . . . . .
100 DATA 50,60,70
```

donne 10 20 50 60 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RETURN

**FORMAT :** RETURN
RETURN NEXT
RETURN numéro de ligne

**DESCRIPTION :**

L'instruction RETURN peut être utilisée pour matérialiser la fin d'un sous-programme ou la fin d'un programme de traitement d'erreurs associé à un ON ERROR. Dans le premier cas l'instruction RETURN s'emploie seule, dans le deuxième cas elle peut s'utiliser sous une des trois formes indiquées.

Utilisé en fin de sous-programme, RETURN s'emploie seul et permet au programme de continuer son exécution avant l'instruction qui suit le GOSUB ou le ON GOSUB ayant appelé le sous-programme. Plusieurs RETURN peuvent être utilisés dans un même sous-programme qui aurait, par exemple, des terminaisons multiples.



Utilisé en fin de programme de traitement d'erreur activé par un ON ERROR, RETURN peut s'employer seul auquel cas il fait continuer l'exécution au niveau de l'instruction ayant provoqué l'erreur. Si RETURN est suivi de NEXT, l'exécution continue avec l'instruction qui suit l'instruction ayant provoqué l'erreur. Enfin RETURN suivi d'un numéro de ligne fait continuer l'exécution à la ligne spécifiée.

L'instruction RETURN ne doit pas être employée sans être associée à au moins un GOSUB ou ON ERROR sinon il y a génération d'une erreur.

Vous pouvez consulter également les fiches des instructions GOSUB, ON GOSUB et ON ERROR.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RND

**FORMAT :** RND

**DESCRIPTION :**

La fonction RND fournit un nombre pseudo aléatoire supérieur ou égal à 0 et strictement inférieur à 1 lors de chaque appel. Ce nombre est extrait de la séquence pseudo aléatoire courante; cette séquence est définie par l'instruction RANDOMIZE à la fiche de laquelle vous êtes prié de vous reporter pour plus de précisions.

**EXEMPLE :**

```
10 A = B * RND + C
```

A est un nombre pseudo aléatoire supérieur ou égal à C et strictement inférieur à B + C.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## RPT$

**FORMAT :** RPT$ (expression chaîne de caractères, expression numérique)

**DESCRIPTION :**

La fonction RPT$ fournit une chaîne de caractères constituée par la répétition de l'expression chaîne de caractères contenue entre parenthèses un nombre de fois égal à celui donné par l'expression numérique.
Si la chaîne ainsi produite comporte plus de 255 caractères, elle est tronquée au 255e caractère et un message d'avertissement est affiché.

**EXEMPLE :**

100 A$ = RPT$ ("ABCD", 4)

donne à A$ la valeur "ABCDABCDABCDABCD".

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

RUN

**FORMAT :** RUN
RUN numéro de ligne

**DESCRIPTION :**

Cette commande, utilisable en mode direct et programmé, permet de lancer l'exécution d'un programme. Si RUN est utilisé seul, l'exécution commence à la ligne de plus faible numéro du programme. Si RUN est suivi par un numéro de ligne, l'exécution commence à ce numéro.

Avant que l'exécution n'ait lieu, les événements suivants se produisent :

- Les variables sont initialisées c'est-à-dire que les variables numériques sont mises à zéro et que les variables chaînes de caractères sont rendues égales à la chaîne vide.
- Certaines erreurs telles le FOR sans NEXT ou un branchement à un numéro de ligne inexistant sont détectées.
- Le mode ON ERROR STOP est sélectionné.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■



SAVE

**FORMAT :** SAVE"numéro.nom de programme".

**DESCRIPTION :**

Cette commande utilisable seulement en mode direct permet de sauvegarder un programme sur un équipement périphérique. Le numéro spécifie le type d'équipement et est égal à 1 pour un magnétophone à cassettes et à 2 pour une cassette de RAM CMOS.
Le nom de programme doit impérativement être spécifié et doit comporter au maximum quatre caractères. Il doit être séparé du numéro d'équipement par un point ou une virgule.

Lors de l'exécution de cette commande, le message ESC -> Enr. est affiché sur la ligne de service en haut de l'écran. La sauvegarde proprement dite ne commence qu'avec la frappe de la touche ESCAPE (ESC) ce qui permet de préparer le magnétophone et de mettre celui-ci en enregistrement.

Lorsque la sauvegarde a lieu, le nom donné au programme sauvegardé est affiché sur la ligne de service en haut d'écran et le signal envoyé sur le magnétophone à cassettes est reproduit par le haut-parleur du récepteur TV.

Dans tous les cas, la commande SAVE ne touche pas au programme contenu en mémoire lors de son exécution.

**EXEMPLES :**

SAVE "1.ABCD" sauvegarde le programme contenu en mémoire sur cassette et lui donne le nom ABCD.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SEG$

**FORMAT :** SEG$ (expression chaîne de caractères, position, longueur).

**DESCRIPTION :**

La fonction SEG$ extrait de la chaîne de caractères spécifiée une sous-chaîne de caractères. Cette sous-chaîne commence à la position précisée et fait la longueur également précisée.
La position est repérée à partir de la gauche et le premier caractère est repéré 1.
Si la position précisée se trouve au-delà de la fin de la chaîne, SEG$ fournit une chaîne vide (''').
Si la longueur précisée est telle que l'on s'étende au-delà de la fin de la chaîne, seuls les caractères allant de la position spécifiée à la fin de la chaîne constitueront la sous-chaîne.
Cette fonction permet de réaliser les fonctions LEFT$, RIGHT$ ET MID$ de certains Basic.

**EXEMPLES :**

100 PRINT SEG$ (''EXL 100'', 5,2)

fait afficher 10 lors de l'exécution.
100 PRINT SEG$ (''EXL 100'', 2, 2)

fait afficher XL lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SGN

**FORMAT :** SGN (expression numérique).

**DESCRIPTION :**

L'expression numérique contenue entre parenthèses est évaluée, puis la fonction SGN fournit son signe sous la forme suivante : si l'expression est strictement positive, SGN est égale à 1; si l'expression est strictement négative, SGN est égale à —1; si l'expression est nulle, SGN est égale à 0.



**EXEMPLE :**

10 PRINT SGN (—2); SGN (3); SGN (0).
donne comme —1 1 0 lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SIN

**FORMAT :** SIN (expression numérique).

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée et le résultat est considéré comme une mesure d'angle exprimée en radians, degrés ou grades selon l'unité d'angle préalablement sélectionnée. La fonction SIN donne alors le sinus de cet angle.

**EXEMPLE :**

10 PRINT SIN (PI/6)

donne .5 comme résultat lors de l'exécution.

## SPEECH

**FORMAT :** CALL SPEECH (expression chaîne de caractères).

**DESCRIPTION :**

SPEECH est un sous-programme qui doit être appelé par un CALL. Il a pour fonction de commander le synthétiseur de paroles de l'EXL 100. L'expression chaîne de caractères contenue entre parenthèses doit respecter les règles de syntaxe suivantes :
- Elle doit commencer par L, A ou R selon la fonction désirée. L commande la mise en marche du synthétiseur et la production du son correspondant à la chaîne qui suit. A commande l'arrêt du synthétiseur et R commande la répétition continue du son correspondant à la chaîne qui suit.

- L'une de ces trois lettres doit être suivie par une virgule.
- Les caractères qui suivent correspondent au son ou au mot à produire conformément au codage propre au synthétiseur de l'EXL 100. Quelques chaînes typiques sont indiquées en annexe mais ce n'est nullement limitatif.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SQR

**FORMAT :** SQR (expression numérique).

**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis, si elle est positive ou nulle, la fonction SQR en donne sa racine carrée. Dans le cas contraire, une erreur est générée.

**EXEMPLE :**

10 PRINT SQR (4)

donne 2 comme résultat lors de l'exécution.

## STOP

**FORMAT :** STOP

**DESCRIPTION :**

L'instruction STOP arrête l'exécution d'un programme. Elle peut être interchangée avec l'instruction END.

Vous pouvez consulter également la fiche de l'instruction END.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## STR$

**FORMAT :** STR$ (expression numérique).



**DESCRIPTION :**

L'expression contenue entre parenthèses est évaluée, puis l'instruction STR$ la convertit en une chaîne de caractères. Aucun espace n'est inclus au début ou à la fin de la chaîne ainsi obtenue. Cette fonction est l'inverse de la fonction VAL.

Vous pouvez consulter également les fiches des instructions NUMERIC et VAL.

**EXEMPLE :**

```
10 A$ = "EXL "
20 B$ = STR$ (100)
30 PRINT A$ & B$
```

donne EXL 100 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SUB

**FORMAT :** SUB nom de sous-programme (liste de paramètres).

**DESCRIPTION :**

L'instruction SUB est la première instruction d'un sous-programme qui sera ensuite appelé par un CALL. Le nom de sous-programme est obligatoire car c'est par lui que sera appelé le sous-programme grâce à un CALL. Il peut être constitué par 1 à 15 caractères et le premier caractère doit être une lettre ou un tiret de soulignement; les autres caractères peuvent être alphanumériques. Compte tenu du procédé de recherche des sous-programmes par CALL, vos sous-programmes doivent avoir un nom différent de ceux déjà existants dans le Basic (LINE, PEEK, etc...) car ces derniers sont prioritaires et vos sous-programmes ne pourraient jamais être appelés.

La liste de paramètres définit les informations qui sont passées au sous-programme par le pro-

gramme principal. Ces paramètres peuvent être des variables numériques, chaînes de caractères ou tableaux. Dans ce dernier cas, les tableaux à une dimension sont notés A() (si A est le nom du tableau concerné), ceux à deux dimensions A(,), et ceux à trois dimensions A(,,).

Les informations sont passées au sous-programme grâce à la liste de paramètres qui suit le CALL appelant le sous-programme. Les noms des paramètres n'ont pas besoin d'être identiques dans la ligne de CALL et dans la ligne de définition du sous-programme contenant le SUB. Seuls les types et le nombre de paramètres doivent être les mêmes.

Toutes les variables utilisées dans un sous-programme autre que celles de la liste de paramètres sont locales et sont propres au sous-programme elles peuvent donc avoir des noms identiques à certaines variables du programme principal ou plusieurs sous-programmes différents peuvent utiliser les mêmes noms de variables locales.

Toutes les variables locales d'un sous-programme sont initialisées à chaque appel de celui-ci sauf si l'instruction ATTACH a été utilisée au préalable pour le sous-programme concerné.

Un sous-programme se termine par SUBEND ou SUBEXIT et l'exécution se poursuit alors avec l'instruction qui suit le CALL ayant appelé le sous-programme.

Un sous-programme ne doit pas contenir d'autre sous-programme. Les sous-programmes doivent être rangés en fin de programme principal. Le fait pour le programme principal de rencontrer l'instruction SUB équivaut à un STOP, ce qui implique le rangement des sous-programmes après la dernière instruction exécutable du programme principal.



Les fonctions ON BREAK, ON WARNING et ON ERROR qui étaient valides avant l'appel du sous-programme le restent pendant le sous-programme. Si ce dernier les modifie, elles retrouvent leurs définitions initiales lors du retour au programme principal.
Vous pouvez consulter également les fiches des instructions ATTACH, CALL, ON BREAK, ON ERROR, ON WARNING, RELEASE, SUBEND et SUBEXIT.

**EXEMPLE :**

Un très bel exemple de définition de sous-programme et d'utilisation de celui-ci avec passage de paramètres est donné chapitre 16 dans le programme de tracé de courbes auquel vous êtes prié de vous reporter.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SUBEND

**FORMAT :** SUBEND

**DESCRIPTION :**

Cette instruction matérialise la fin d'un sous-programme dont elle doit impérativement être la dernière instruction. L'exécution de cette instruction renvoie au programme principal à l'instruction qui suit le CALL ayant appelé le sous-programme.

SUBEND ne doit pas être placé au sein d'une instruction IF THEN ELSE.

Vous pouvez consulter également les fiches des instructions SUB et SUBEXIT.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## SUBEXIT

**FORMAT :** SUBEXIT

**DESCRIPTION :**

L'instruction SUBEXIT termine l'exécution d'un sous-programme mais n'en matérialise pas la fin. L'exécution de l'instruction SUBEXIT renvoie au programme principal à l'instruction qui suit le CALL ayant appelé le sous-programme.
De même que RETURN, l'instruction SUBEXIT peut apparaître autant de fois que c'est nécessaire dans un sous-programme qui aurait plusieurs terminaisons possibles.

Vous pouvez consulter également les fiches des instructions SUB et SUBEND.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## TAB

**FORMAT :** TAB (expression numérique).

**DESCRIPTION :**

L'instruction TAB ne peut être employée qu'au sein d'une instruction PRINT pour définir la position d'affichage sur la ligne. L'expression numérique comprise entre parenthèses est évaluée puis, si elle est négative ou nulle la position d'affichage est fixée colonne 1. Si elle est strictement positive, la valeur obtenue est réduite modulo 40 et sert à définir la position d'impression sur la ligne. Si la position ainsi obtenue est au-delà de la position d'affichage déjà atteinte, le curseur alphanumérique est placé à l'endroit ainsi spécifié. Si la position d'affichage courante est déjà au-delà de la position spécifiée par TAB, le positionnement a lieu à la ligne suivante.

TAB est considéré par PRINT comme un élément à imprimer; TAB peut donc être placé dans une ligne de PRINT aux endroits nécessaires autant de fois que c'est nécessaire. TAB doit être séparé des autres éléments de la ligne par des séparateurs



qui peuvent être des virgules ou des points virgules.
Ces séparateurs sont traités normalement; celui qui précède le TAB est interprété avant l'exécution de TAB; celui qui suit le TAB est interprété après l'exécution de TAB.

Vous pouvez consulter également les fiches des instructions PRINT et PRINT USING.

**EXEMPLE :**

PRINT TAB (10); "EXL 100"

fait afficher EXL 100 à partir de la colonne 10.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## TAN

**FORMAT :** TAN (expression numérique)

**DESCRIPTION :**
L'expression contenue entre parenthèses est évaluée et le résultat est considéré comme la mesure d'un angle exprimée en radians, degrés ou grades selon l'unité d'angle sélectionnée au préalable. La fonction TAN donne alors la tangente de cet angle. Une erreur de dépassement de capacité est générée si l'angle est exactement égal à PI/2 ou —PI/2. La tangente étant infinie dans ces deux cas.

**EXEMPLE :**

```
10 A = PI/4
20 PRINT TAN (A)
```

donne 1 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## UNBREAK

**FORMAT :** UNBREAK
UNBREAK liste de numéros de lignes.

**DESCRIPTION :**

L'instruction UNBREAK utilisée seule enlève tous les points d'arrêt d'un programme. Si une suite de numéros de lignes est spécifiée après le UNBREAK, seuls les points d'arrêt placés sur ces lignes sont enlevés. Les numéros de lignes peuvent être écrits dans n'importe quel ordre et doivent être séparés les uns des autres par des virgules.

Vous pouvez consulter également la fiche de l'instruction BREAK.

**EXEMPLES :**

UNBREAK enlève tous les points d'arrêt d'un programme.

1000 UNBREAK 120,230

lors de l'exécution de la ligne 1000, enlève les points d'arrêt des lignes 120 et 230.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## VAL

**FORMAT :** VAL (expression chaîne de caractères)

**DESCRIPTION :**

La fonction VAL fournit la valeur numérique de l'expression chaîne de caractères contenue entre parenthèses. Les espaces éventuellement présents en début ou fin de chaîne sont ignorés par la fonction VAL.

Cette fonction est l'inverse de la fonction STR$.

Si la chaîne de caractères n'est pas la représentation d'un nombre valide, une erreur est générée.

Vous pouvez consulter également les fiches des instructions NUMERIC et STR$.



**EXEMPLE :**

```
10 A$ = "1234"
20 B = VAL (A$) + 1
30 PRINT B
```

donne 1235 comme résultat lors de l'exécution.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

## VERSION

**FORMAT :** CALL VERSION (variable numérique)

**DESCRIPTION :**

VERSION est un sous-programme qui doit être appelé par un CALL. Après un CALL VERSION, la variable numérique contient le numéro de version de l'interpréteur Basic en votre possession.

# ANNEXE I

Liste des instructions de l'EXL 100

Les commandes, instructions et fonctions de l'EXL 100 sont classées ci-après par ordre alphabétique. Les commandes ne peuvent être utilisées qu'en mode direct alors que toutes les instructions, sauf celles suivies d'une astétisque, et toutes les fonctions peuvent être exécutées en mode direct et en mode programmé.

## Commandes :

| | | |
|---|---|---|
| CALL CLEANUP | CONTINUE ou CON | LIST |
| NEW | NUMBER ou NUM | RENUMBER ou REN |
| SAVE | | |

## Instructions :

| | | |
|---|---|---|
| ACCEPT* | ATTACH | BREAK |
| CALL | CALL CHAR | CLOSE |
| CLS | CALL COLOR | DATE |
| DEG | DELETE ou DEL | DIM |
| END | CALL ERR | CALL EXEC |
| FOR TO STEP | FORMAT | GOSUB* |
| GOTO* | GRAD | CALL HRON |
| CALL HROFF | IF THEN ELSE | IMAGE |
| INPUT* | CALL KEY1 | CALL KEY2 |
| LET | CALL LINE | LINPUT* |
| LOAD | LOCATE | NEXT |
| ON BREAK | ON ERROR | ON GOSUB* |
| ON GOTO* | ON WARNING | OPEN |
| PAUSE | CALL PEEK | CALL PLOT |
| CALL POKE | PRINT | PRINT USING |
| RAD | RANDOMIZE | READ* |
| RELEASE | REM | RESTORE |
| RETURN* | RUN | CALL SPEECH |
| STOP | SUB* | SUBEND* |
| SUBEXIT* | UNBREAK | CALL VERSION |

## Fonctions :

| | | |
|---|---|---|
| ABS | ACS | ASC |
| ASN | ATN | CHR$ |
| COS | ESP | INT |
| INTRND | KEY$ | LEN |
| LN | LOG | NUMERIC |
| PI | POS | RND |
| RPT$ | SEG$ | SGN |
| SIN | SQR | STR$ |
| TAB | TAN | VAL |

# ANNEXE 2

Signification des messages d'erreur

Code Type

00 - Erreur d'entrée / sortie; est suivi par un code propre aux erreurs de ce type.

01 - Erreur de syntaxe telle que : absence de parenthèse ou de guillemet, absence de virgule, absence de séparateur entre deux instructions, absence d'opérateur entre deux instructions, absence d'opérateur entre deux variables, variable incorrecte, absence de mot clé, mot clé mal placé, caractère incorrect dans un nom de variable.

02 - Expression trop complexe. Trop de variables ou d'opérateurs dans la même expression.

03 - Confusion nombre - chaîne de caractères. Une expression numérique a été fournie à la place d'une chaîne de caractères ou vice versa.

04 - Donnée incorrecte. Une donnée ou une variable est incorrecte dans une instruction : par exemple élévation d'un nombre négatif à une puissance non entière ou variable hors limites dans un ON GOTO ou ON GOSUB.

05 - Débordement de la pile.

06 - NEXT sans FOR préalable. Un NEXT a été rencontré sans FOR préalable ou le nom de variable de boucle d'un NEXT ne correspond à aucun FOR ou le nombre de NEXT d'un programme est supérieur au nombre de FOR.

07 - Erreur lors d'un INPUT. Tentative de fournir plus d'une donnée à la fois lors d'un INPUT ou d'un ACCEPT.

08 - Erreur de dimensionnement d'un tableau.



09 - Variable déjà définie par ailleurs. Tentative de définition d'un tableau avec un nom de variable déja utilisé.

10 - Exécution de la commande ou de l'instruction impossible. Par exemple : utilisation de CONTINUE alors que le programme n'a pas été arrêté sur un point d'arrêt ou rencontre d'un SUBEND alors qu'aucun SUB ne précède.

11 - Une instruction autre que SUB, REM ou END suit un SUBEND

12 - Référence à un numéro de ligne inexistant.

13 - Erreur d'imbrication de boucles FOR - NEXT

14 - Programme de traitement d'erreur non terminé par RETURN.

15 - Programme ou sous-programme non trouvé

16 - Numéro de ligne non trouvé.

17 - Numéro de ligne hors de la plage 1 - 37266.

18 - Type de programme incorrect. Tentative d'appeler un sous-programme comme un programme ou vice-versa, tentative d'appeler un programme en langage machine comme un programme Basic.

19 - Commande utilisable seulement en mode direct.

20 - Numéro non utilisé.

21 - Sous-programme en cours d'utilisation. Un sous-programme s'est appelé lui-même directement ou indirectement.

22 - Variable non définie.

23 - Erreur de définition lors d'une instruction IMAGE.

24 - Erreur dans l'utilisation d'un fichier.

25 - Table des noms de variables pleine. Le nombre maximum de variables d'un programme ne doit pas dépasser 95.

26 - Parenthèses en nombre impair. Il manque une parenthèse dans une expression.

27 - Ligne trop longue. Une ligne comporte plus de 80 caractères.

28 - Nom de variable trop long. Un nom de variable comporte plus de 15 caractères.

29 - Donnée ou variable incorrecte dans un LOG, CHR$, SIZE, VALIDATE, TAB ou paramètre lors d'un CALL en désaccord avec ceux attendus par le sous-programme appelé.

30 - FOR sans NEXT. trop d'instructions FOR pour le nombre de NEXT disponibles.

31 - Numéro non utilisé.

32 - Utilisation incorrecte d'un indice.

33 - Débordement arithmétique. Un nombre ou le résultat d'un calcul est supérieur à 9.9999999999999E+127.

34 - Division par 0.

35 - Numéro non utilisé.

36 - Chaîne de caractères tronquée. Une concaténation ou RPT$ ont conduit à une chaîne de plus de 255 caractères.
37 - BREAK. Un point d'arrêt a eu lieu ou le programme a été interrompu par un CTL C.
38 - Numéro non utilisé.
39 - SUBEXIT ou SUBEND rencontré dans un programme principal.
40 - Numéro non utilisé.
41 - SUB doit être la première instruction d'une ligne.
42 - SUBEND manquant dans un sous-programme ou SUB rencontré dans un sous-programme.
43 - Erreur dans un DATA. Nombre de données derrière un DATA insuffisant, types de données non conformes à celles attendues par un READ.
44 - Utilisation en mode direct d'une instruction qui ne peut être employée qu'en mode programmé.
45 - RETURN sans GOSUB préalable.
126 - Erreur système. La mémoire réservée au système a été modifiée de façon incorrecte (par POKE ou EXEC par exemple).
127 - Mémoire pleine.

**Erreurs d'entrée / sortie :**

Les codes ci-après suivent un message d'erreur de numéro 00 pour préciser le type d'erreur d'entrée / sortie. Seuls sont précisés ici les codes correspondant à l'EXL 100 de base. Les codes relatifs à des extensions (cassette de RAM CMOS, imprimante, lecteurs de disquettes) sont précisés dans les notices de celles-ci;

| Code | Type d'erreur |
|---|---|
| 06 | Erreur dans la communication avec le périphérique. |
| 16 | Erreur de CHEKSUM en provenance du périphérique. Une erreur sur un caractère de contrôle est détectée indiquant une erreur probable de lecture du périphérique. |
| 255 | Erreur de TIME OUT. La communication avec le périphérique n'a pu être établie dans les temps impartis ou le périphérique demandé n'est pas connecté. |



# ANNEXE 3

Priorités des opérateurs

Lorsque plusieurs opérateurs apparaissent dans une expression, ils sont exécutés les uns après les autres par ordre de priorité décroissante. Ces priorités sont repérées par un chiffre de 3 à 10 comme indiqué ci-après; 10 est la plus forte priorité.

| Opérateur | Opération réalisée | Priorité |
|---|---|---|
| ↑ | Elévation à une puissance | 10 |
| — | Signe moins devant une variable | 9 |
| * | Multiplication | 8 |
| / | Division | 8 |
| + | Addition | 8 |
| & | Concaténation de chaînes de caractères | 6 |
| — | Soustraction | 6 |
| = | Egal à | 5 |
| < > | Différent de | 5 |
| > | Supérieur à | 5 |
| < | Inférieur à | 5 |
| > = | Supérieur ou égal à | 5 |
| < = | Inférieur ou égal à | 5 |
| NOT | Complément logique | 4 |
| AND | ET logique | 3 |
| OR | OU logique | 3 |
| XOR | OU EXCLUSIF logique | 3 |

Les parenthèses permettent de modifier l'ordre d'exécution des opérations dans une expression.

# ANNEXE 4

## Code ASCII et caractères spéciaux

Ce code ASCII est indiqué ci-après en décimal et hexadécimal avec le caractère qui correspond à chaque code. Les codes de 0 à 31 inclus (décimal) sont particuliers à l'EXL 100 et ne correspondent pas aux définitions qui leur sont normalement attribuées par le code ASCII.

| DEC | HEX | CARACTERE | DEC | HEX | CARACTERE |
|---|---|---|---|---|---|
| 00 | 00 | | 27 | 1B | ô |
| 01 | 01 | ● | 28 | 1C | ù |
| 02 | 02 | à | 29 | 1D | |
| 03 | 03 | ǀ | 30 | 1E | — |
| 04 | 04 | — | 31 | 1F | ǀ |
| 05 | 05 | + | 32 | 20 | ESPACE |
| 06 | 06 | ┐ | 33 | 21 | ! |
| 07 | 07 | ┘ | 34 | 22 | " |
| 08 | 08 | └ | 35 | 23 | # |
| 09 | 09 | ┌ | 36 | 24 | $ |
| 10 | 0A | ± | 37 | 25 | % |
| 11 | 0B | → | 38 | 26 | & |
| 12 | 0C | ■ | 39 | 27 | ' |
| 13 | 0D | ◘ | 40 | 28 | ( |
| 14 | 0E | ↓ | 41 | 29 | ) |
| 15 | 0F | â | 42 | 2A | * |
| 16 | 10 | è | 43 | 2B | + |
| 17 | 11 | é | 44 | 2C | , |
| 18 | 12 | ê | 45 | 2D | – |
| 19 | 13 | ǀ | 46 | 2E | . |
| 20 | 14 | ç | 47 | 2F | / |
| 21 | 15 | ë | 48 | 30 | 0 |
| 22 | 16 | ┬ | 49 | 31 | 1 |
| 23 | 17 | ┤ | 50 | 32 | 2 |
| 24 | 18 | ┴ | 51 | 33 | 3 |
| 25 | 19 | ├ | 52 | 34 | 4 |
| 26 | 1A | ï | 53 | 35 | 5 |

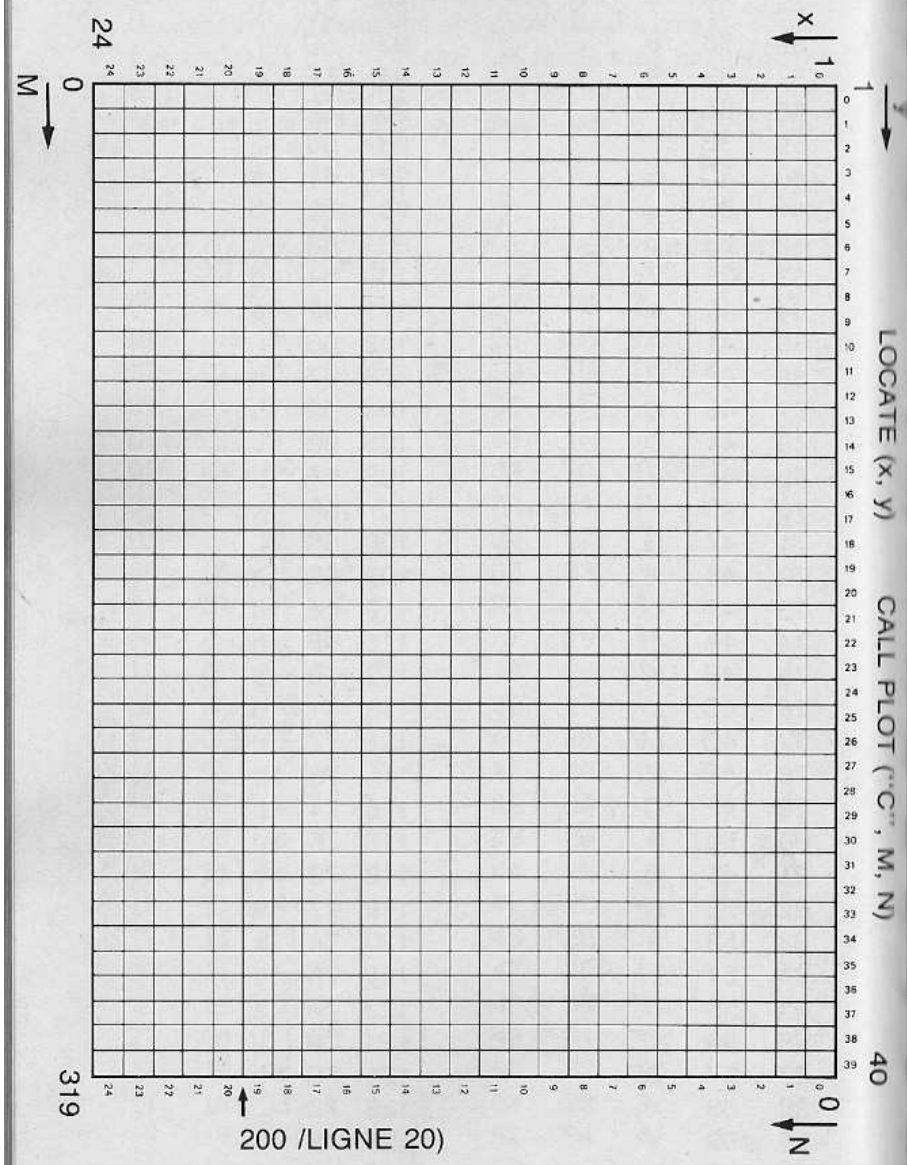




| DEC | HEX | CARACTERE | DEC | HEX | CARACTERE |
|---|---|---|---|---|---|
| 54 | 36 | 6 | 91 | 5B | [ |
| 55 | 37 | 7 | 92 | 5C | \ |
| 56 | 38 | 8 | 93 | 5D | ] |
| 57 | 39 | 9 | 94 | 5E | ↑ |
| 58 | 3A | : | 95 | 5F | _ |
| 59 | 3B | ; | 96 | 60 | ` |
| 60 | 3C | < | 97 | 61 | a |
| 61 | 3D | = | 98 | 62 | b |
| 62 | 3E | > | 99 | 63 | c |
| 63 | 3F | ? | 100 | 64 | d |
| 64 | 40 | a | 101 | 65 | e |
| 65 | 41 | A | 102 | 66 | f |
| 66 | 42 | B | 103 | 67 | g |
| 67 | 43 | C | 104 | 68 | h |
| 68 | 44 | D | 105 | 69 | i |
| 69 | 45 | E | 106 | 6A | j |
| 70 | 46 | F | 107 | 6B | k |
| 71 | 47 | G | 108 | 6C | l |
| 72 | 48 | H | 109 | 6D | m |
| 73 | 49 | I | 110 | 6E | n |
| 74 | 4A | J | 111 | 6F | o |
| 75 | 4B | K | 112 | 70 | p |
| 76 | 4C | L | 113 | 71 | q |
| 77 | 4D | M | 114 | 72 | r |
| 78 | 4E | N | 115 | 73 | s |
| 79 | 4F | O | 116 | 74 | t |
| 80 | 50 | P | 117 | 75 | u |
| 81 | 51 | Q | 118 | 76 | v |
| 82 | 52 | R | 119 | 77 | w |
| 83 | 53 | S | 120 | 78 | x |
| 84 | 54 | T | 121 | 79 | y |
| 85 | 55 | U | 122 | 7A | z |
| 86 | 56 | V | 123 | 7B | { |
| 87 | 57 | W | 124 | 7C | ǀ |
| 88 | 58 | X | 125 | 7D | } |
| 89 | 59 | Y | 126 | 7E | → |
| 90 | 5A | Z | 127 | 7F | DEL |

# ANNEXE 5

Grille d'adressage de l'écran alphanumérique et graphique






# ANNEXE 6

Chaînes de génération de sons pour CALL SPEECH

## Laser :

0D9816C301EDE72CB8B4CBECBD7D88AA6F3A2DD41AAF6BDB46407AFDAE3CEAA1EAA6AB7BE4A7CB86ADC6D37C9E04BE87411B508C04302FE

## Phaser :

0E683FC7031EF040043CE0010F78C00216B040042C60010F5840031AD08006FE

## Explosion :

0BA02163031B18008D1E139840073A108100F0901E81080480478F084420020570CEBD02110840704E118840043CE0010F78C00316B080052C20010948800214A0000528

## Tic :

0D7C8BAABAB602FC

## Toc :

0D7C91B2BAB602FC

## N0N 80

AA2A4864AD129BAA62E6B349E4AA810556AB49A886106ADAC4B1085997C20E9BB8A543871844E213343D6A9291AF7B8CA2274D3DF2D263DA5077DB438A6DC3D34D352E8E0B13B513D8F8DAD9D5F2FF03

## BRAVO 100

24D2ACA2CC1B9384A98DB468A5B2485AC32
C6E8BB2BA32A9302DE8A2D3A9232FAB984D
B3B572C4123B2D57F1E50E9B737AE4C82A42
7643654939285995187694A417B66214B69393
CCA94155A95E09A33931DC322E8EF65164CA
98328A17E1BEACEBFF07

## PARFAIT (NEUTRE) 93

0CF0212271C584A7B6E4C395E25E957236D39
8A65B0D5B4E49BBFE320B310AE8EA4401D12
608A0E966011C91AA010948400053B5252B6B
978C923DEC1254A3D3EA70AAE7C80D39DD6
D6E55EAD52EC5B95968453055E124CA91A94
AFE1F

## COUCOU 87

02882BCD51B69A6855B340F1E16A338903E51
7699E2A6154B80A929504815395AA500D0006
C4996684D44B84358E125455D1AED646B0832
5369C58410972B0C4B86321B061A21B49848A
77156B82122EDCE31D3F0FF01

## BIEN 101

A850A222221A918CF3F14C77440AC5C623222
6AB1C0B4FCF54A9F886213213BBFA0BE4884
A12EB526E22E27971463995B3E59A5E557725
9A6569552D955AB4E951D756AA958A673DEA
9AD2B843D62C67D6DC628A2D934E925354EE
50458AB5C1DD93BCFD3F

## CENT 60

04F826830214A0000538C081144D1466A14D4
EDC4585192D3A695335A6BAE8A6232EB24ED
923F7A0B945E422BBE564463B05E1B28DE26
B0B6C28FE7F



## MILLE 103

AC4CC4359B5BFB3216A188CC3492BB96A25
2310D4B338A25892D73F6CC577E1DB17A2E9
ED94A8F1121ABD92B5ECA88C3178DE83056
D15CDCFC83D9C57A51F31654E7983AD51E6
695C1E1356AE8C5394FB6E1A6669D3AEE469
CE2BD799A5313AAA7A7S951E5FF01

## ET 44

235EC298376B9D78FA44CEA873A31B9D6038
07C99A7DFCA90D6D6AF2B48AD92453771671
46E4E25721F6FF01

## PARFAIT 90

0CD0A1B4D1459685259D41A5DAE16E680655
422779B21AF4941D4E29C500191605A0A4060
F1820C924012C55CA01066CE32601034CD196
E4228B5393EF26A70095F2927EA3639F4AC8C
A429E7CE913A8B5C9BAE6B9B060F2FF

## ZERO 120

2A8DAC28B54BBBB26297B06890DA25CB982
571EE52BBCC86AB382EE934A72C6EAB4555
07B874CD6116A55A7E96A73A946694C3D6F4
522A4F2ECFD043CBAA4C234B25BBB075478
FECE22C6F183DB1CBA29A11E656A10876469
97AE72AD4B64C30FB4AD7B9BCCE6C2FF5A0
465B474962169B833F4391CCFF0F

## UN 56

C9AEB5D334AA1CAFA476F3AA73BDE3969AA
A99B2AFBBBDE59554CA5E56865F5C1A454B
4951F22E5C49C5A5C13B7B191551246F82999
4FF1F

## DEUX 86

24673225AB629292CAB68C32892AC9D3A322
02A904ED8C4849ABDC41435515AF6AAB0C16
F6728221CC29A4CF75B73DAC2A4238BD55B3
AA0B26D797CD2DDC852B5A263F72B7685F61
E2A8593CBD2429925454EF46FF0F

## TROIS 69

0CD8D6420103AF0B20DA7301047ECD00ADB
64219456F08750969E679A6D4BA06C38F6156
CD674E6F7AC34B694A2D2757DF04A1B690B9
6915169161A8915856ED76B295FC7F

## QUATRE 94

0E58816268DD3BA457ECE39650B251B5AF7D
AC5CD3D236C976839BEC487A9844E054A52A
54030002504DAA024A1A56C0A0570A68F2DD
B583FD8479A2D454BE6E62955B53CC84894F
E96D6BB36937CF30ADADBA4D350B87A6B83
0F516F4FF

## CINQ 122

043036890095A63060AA710E7080031CE080B
3A6ED48F3D4CBA9B6D9B50A1FA7CB7575AF
74ACEAD6C43CCCB24B1E958830DB5D5EB52
D9211BAF94D05B97B28153961F69856484E55
E6282746E054A52A540300C000434B86689F9
1747152B2A65C2CB24C6B9B29518F52B50F4
3B6A9516ED8F4C5B84A84E1FF07

## SIX 102

0C58768D031C9080043410B29B9630244DC9A
612528ED93D2BD9E4C2A271A765478B30DB9
3166FC92C1AB78B3F299367DC4E5E83C25D5
EDA79DB048B6A4201749522010948400212 90
80048C918D535847367A16CD1D195469C9154
FBA11A1C664561164FE1F



## SEPT 91

0C686684031C6040D7A30CA86E8C015D974B
4001D996B6B49A90D08A74B26693C36BF6CD
6EBA82E24CDAD78AF392315C2B32089CAA5
4856A00008002AA4A73FAE42DA1A9609845B6
B475E96D343DE9F69636E2541BC4D203E9FF03

## HUIT 87

212F2A90C3FA84A20441495D5D8A1A83D0A5
4B299A0FC6E03EB5A879CF7BD69311F5A048
AD9B5730AC1394AF5EEED1C218B9643ACD3
98299BC9F34673366C99D53DDE04F4C227D6
14C988166CF101171023A1B41EBFF01

## NEUF 81

22F328F55212E9C296AE488275D49BFA2216B
1086566CA69BA7A14D5B58ACAEC9D9FAC99
360B9D75A3E517D39CE983BB1DAC2083CB2
34CB0970E35D6580A98AA550312900003960E
61C0D26912E0C0FF03

## DIX 117

24663C2DA72DB142F190EA8C482AA555665C
14A98474ED49C9ACD6CCBDD223B7E698204
BAD588A4595386AF2C8B72652CB39237D865
93C1FCF68795732B3C79C662D2CE8555383D
511D34C5412C0D669129080042420010970406
947B61ABB43A3E4C4D2DE608D12237B87415
3B2B7E9AE49FDFF01

# UTILISATION DU SYNTHETISEUR VOCAL

L'instruction "Call Speech" du basic permet d'envoyer 128 octets (data) au synthétiseur.
En effet, une variable chaîne sous basic contient 255 caractères maximum et permet de coder 128 octets en hexadécimal.

Il est possible d'utiliser des tables de données plus conséquentes grâce à une routine assembleur (11 octets). Ces données devront être implantées en mémoire CPU grâce à CALL POKE.
Le programme de génération sonore sera appelé par CALL EXEC.

Données routine assembleur
152, 79,10, 120, 11, 10, 121, 0, 9, 251, 10

Exemple :

100 DATA 152, 79, 10, 120, 11, 10, 121, 0, 9, 251, 10
110 DATA Donnée speech ... (en décimal) par exemple 490 Données.
120 DATA Donnée speech ... (en décimal)
200 FOR I = 0 TO 500
210 READ A : CALL POKE (50688 + I,A) : NEXT Chargement de la table
220 CALL EXEC (50688) Lancement du son

Exemple d'une table de 133 octets
soit 266 caractères en représentation chaîne

Codage décimal de "BONJOUR"

40, 213, 170, 76, 53, 179, 144, 148, 185, 50, 149, 110, 92, 194, 228, 215, 4, 171, 114, 210, 66, 217, 0, 139, 72, 129, 247, 163, 152, 213, 147, 38, 218, 133, 86, 177, 76, 146, 168, 152, 6, 213, 202, 89, 161, 51, 79, 117, 203, 52, 10, 40, 80, 93, 1, 219, 118, 37, 125, 219, 27, 83, 149, 156, 196, 192, 83, 35, 42, 108, 18, 173, 13, 13, 47, 37, 78, 180, 209, 76, 106, 88, 25, 201, 52, 150, 176, 80, 108, 101, 37, 171, 46, 90, 25, 174, 181, 172, 100, 217, 149, 179, 210, 177, 148, 148, 36, 171, 83, 133, 219, 49, 146, 45, 206, 158, 169, 74, 201, 137, 142, 81, 164, 91, 2, 228, 248, 74, 128, 46, 175, 254, 31.



# ANNEXE 7

Tableau de conversion décimal - hexadécimal binaire

Tableau de conversion décimal - hexadémical - binaire utilisable pour la définition de caractères grâce à l'instruction CALL CHAR.

| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 0 | 00 | 00000000 |
| 1 | 01 | 00000001 |
| 2 | 02 | 00000010 |
| 3 | 03 | 00000011 |
| 4 | 04 | 00000100 |
| 5 | 05 | 00000101 |
| 6 | 06 | 00000110 |
| 7 | 07 | 00000111 |
| 8 | 08 | 00001000 |
| 9 | 09 | 00001001 |
| 10 | 0A | 00001010 |
| 11 | 0B | 00001011 |
| 12 | 0C | 00001100 |
| 13 | 0D | 00001101 |
| 14 | 0E | 00001110 |
| 15 | 0F | 00001111 |
| 16 | 10 | 00010000 |
| 17 | 11 | 00010001 |
| 18 | 12 | 00010010 |
| 19 | 13 | 00010011 |
| 20 | 14 | 00010100 |
| 21 | 15 | 00010101 |
| 22 | 16 | 00010110 |
| 23 | 17 | 00010111 |
| 24 | 18 | 00011000 |
| 25 | 19 | 00011001 |
| 26 | 1A | 00011010 |
| 27 | 1B | 00011011 |
| 28 | 1C | 00011100 |
| 29 | 1D | 00011101 |
| 30 | 1F | 00011110 |

| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 31 | 1F | 00011111 |
| 32 | 20 | 00100000 |
| 33 | 21 | 00100001 |
| 34 | 22 | 00100010 |
| 35 | 23 | 00100011 |
| 36 | 24 | 00100100 |
| 37 | 25 | 00100101 |
| 38 | 26 | 00100110 |
| 39 | 27 | 00100111 |
| 40 | 28 | 00101000 |
| 41 | 29 | 00101001 |
| 42 | 2A | 00101010 |
| 43 | 2B | 00101011 |
| 44 | 2C | 00101100 |
| 45 | 2D | 00101101 |
| 46 | 2E | 00101110 |
| 47 | 2F | 00101111 |
| 48 | 30 | 00110000 |
| 49 | 31 | 00110001 |
| 50 | 32 | 00110010 |
| 51 | 33 | 00110011 |
| 52 | 34 | 00110100 |
| 53 | 35 | 00110101 |
| 54 | 36 | 00110110 |
| 55 | 37 | 00110111 |
| 56 | 38 | 00111000 |
| 57 | 39 | 00111001 |
| 58 | 3A | 00111010 |
| 59 | 3B | 00111011 |
| 60 | 3C | 00111101 |
| 61 | 3D | 00111101 |
| 62 | 3E | 00111110 |
| 63 | 3F | 00111111 |
| 64 | 40 | 01000000 |
| 65 | 41 | 01000001 |
| 66 | 42 | 01000010 |
| 67 | 43 | 01000011 |
| 68 | 44 | 01000100 |
| 69 | 45 | 01000101 |
| 70 | 46 | 01000110 |



| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 71 | 47 | 01000111 |
| 72 | 48 | 01001000 |
| 73 | 49 | 01001001 |
| 74 | 4A | 01001010 |
| 75 | 4B | 01001011 |
| 76 | 4C | 01001100 |
| 77 | 4D | 01001101 |
| 78 | 4E | 01001110 |
| 79 | 4F | 01001111 |
| 80 | 50 | 01010000 |
| 81 | 51 | 01010001 |
| 82 | 52 | 01010010 |
| 83 | 53 | 01010011 |
| 84 | 54 | 01010100 |
| 85 | 55 | 01010101 |
| 86 | 56 | 01010110 |
| 87 | 57 | 01010111 |
| 88 | 58 | 01011000 |
| 89 | 59 | 01011001 |
| 90 | 5A | 01011010 |
| 91 | 5B | 01011011 |
| 92 | 5C | 01011100 |
| 93 | 5D | 01011101 |
| 94 | 5E | 01011110 |
| 95 | 5F | 01011111 |
| 96 | 60 | 01100000 |
| 97 | 61 | 01100001 |
| 98 | 62 | 01100010 |
| 99 | 63 | 01100011 |
| 100 | 64 | 01100100 |
| 101 | 65 | 01100101 |
| 102 | 66 | 01100110 |
| 103 | 67 | 01100111 |
| 104 | 68 | 01101000 |
| 105 | 69 | 01101001 |
| 106 | 6A | 01101010 |
| 107 | 6B | 01101011 |
| 108 | 6C | 01101100 |
| 109 | 6D | 01101101 |
| 110 | 6E | 01101110 |

| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 111 | 6F | 01101111 |
| 112 | 70 | 01110000 |
| 113 | 71 | 01110001 |
| 114 | 72 | 01110010 |
| 115 | 73 | 01110011 |
| 116 | 74 | 01110100 |
| 117 | 75 | 01110101 |
| 118 | 76 | 01110110 |
| 119 | 77 | 01110111 |
| 120 | 78 | 01111000 |
| 121 | 79 | 01111001 |
| 122 | 7A | 01111010 |
| 123 | 7B | 01111011 |
| 124 | 7C | 01111100 |
| 125 | 7D | 01111101 |
| 126 | 7E | 01111110 |
| 127 | 7F | 01111111 |
| 128 | 80 | 10000000 |
| 129 | 81 | 10000001 |
| 130 | 82 | 10000010 |
| 131 | 83 | 10000011 |
| 132 | 84 | 10000100 |
| 133 | 85 | 10000101 |
| 134 | 86 | 10000110 |
| 135 | 87 | 10000111 |
| 136 | 88 | 10001000 |
| 137 | 89 | 10001001 |
| 138 | 8A | 10001010 |
| 139 | 8B | 10001011 |
| 140 | 8C | 10001100 |
| 141 | 8D | 10001101 |
| 142 | 8E | 10001110 |
| 143 | 8F | 10001111 |
| 144 | 90 | 10010000 |
| 145 | 91 | 10010001 |
| 146 | 92 | 10010010 |
| 147 | 93 | 10010011 |
| 148 | 94 | 10010100 |
| 149 | 95 | 10010101 |
| 150 | 96 | 10010110 |



| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 151 | 97 | 10010111 |
| 152 | 98 | 10011000 |
| 153 | 99 | 10011001 |
| 154 | 9A | 10011010 |
| 155 | 9B | 10011011 |
| 156 | 9C | 10011100 |
| 157 | 9D | 10011101 |
| 158 | 9E | 10011110 |
| 159 | 9F | 10011111 |
| 160 | A0 | 10100000 |
| 161 | A1 | 10100001 |
| 162 | A2 | 10100010 |
| 163 | A3 | 10100011 |
| 164 | A4 | 10100100 |
| 165 | A5 | 10100101 |
| 166 | A6 | 10100110 |
| 167 | A7 | 10100111 |
| 168 | A8 | 10101000 |
| 169 | A9 | 10101001 |
| 170 | AA | 10101010 |
| 171 | AB | 10101011 |
| 172 | AC | 10101100 |
| 173 | AD | 10101101 |
| 174 | AE | 10101110 |
| 175 | AF | 10101111 |
| 176 | B0 | 10110000 |
| 177 | B1 | 10110001 |
| 178 | B2 | 10110010 |
| 179 | B3 | 10110011 |
| 180 | B4 | 10110100 |
| 181 | B5 | 10110101 |
| 182 | B6 | 10110110 |
| 183 | B7 | 10110111 |
| 184 | B8 | 10111000 |
| 185 | B9 | 10111001 |
| 186 | BA | 10111010 |
| 187 | BB | 10111011 |
| 188 | BC | 10111100 |
| 189 | BD | 10111101 |
| 190 | BE | 10111110 |

| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 191 | BF | 10111111 |
| 192 | C0 | 11000000 |
| 193 | C1 | 11000001 |
| 194 | C2 | 11000010 |
| 195 | C3 | 11000011 |
| 196 | C4 | 11000100 |
| 197 | C5 | 11000101 |
| 198 | C6 | 11000110 |
| 199 | C7 | 11000111 |
| 200 | C8 | 11001000 |
| 201 | C9 | 11001001 |
| 202 | CA | 11001010 |
| 203 | CB | 11001011 |
| 204 | CC | 11001100 |
| 205 | CD | 11001101 |
| 206 | CE | 11001110 |
| 207 | CF | 11001111 |
| 208 | D0 | 11010000 |
| 209 | D1 | 11010001 |
| 210 | D2 | 11010010 |
| 211 | D3 | 11010011 |
| 212 | D4 | 11010100 |
| 213 | D5 | 11010101 |
| 214 | D6 | 11010110 |
| 215 | D7 | 11010111 |
| 216 | D8 | 11011000 |
| 217 | D9 | 11011001 |
| 218 | DA | 11011010 |
| 219 | DB | 11011011 |
| 220 | DC | 11011100 |
| 221 | DD | 11011101 |
| 222 | DE | 11011110 |
| 223 | DF | 11011111 |
| 224 | E0 | 11100000 |
| 225 | E1 | 11100001 |
| 226 | E2 | 11100010 |
| 227 | E3 | 11100011 |
| 228 | E4 | 11100100 |
| 229 | E5 | 11100101 |
| 230 | E6 | 11100110 |



| Décimal | Hexadécimal | Binaire |
|---|---|---|
| 231 | E7 | 11100111 |
| 232 | E8 | 11101000 |
| 233 | E9 | 11101001 |
| 234 | EA | 11101010 |
| 235 | EB | 11101011 |
| 236 | EC | 11101100 |
| 237 | ED | 111011011 |
| 238 | EE | 11101110 |
| 239 | EF | 11101111 |
| 240 | F0 | 11110000 |
| 241 | F1 | 11110001 |
| 242 | F2 | 11110010 |
| 243 | F3 | 11110011 |
| 244 | F4 | 11110100 |
| 245 | F5 | 11110101 |
| 246 | F6 | 11110110 |
| 247 | F7 | 11110111 |
| 248 | F8 | 11111000 |
| 249 | F9 | 11111001 |
| 250 | FA | 11111010 |
| 251 | FB | 11111011 |
| 252 | FC | 11111100 |
| 253 | FD | 11111101 |
| 254 | FE | 11111110 |
| 255 | FF | 11111111 |

# ANNEXE 8

Cartographie mémoire de l'EXL 100

| | |
|---|---|
| 0000 | RESERVE |
| 0200 | MEMOIRE MORTE BASIC OU MODULE DE JEU 32 K OCTETS |
| 8000 | MODULE RAM CMOS 16 K OCTETS |
| C000 | RAM CPU 2 K OCTETS |
| C800 | PERIPHERIQUES 10 K OCTETS |
| F000 | RESERVE |
| F806 | ROM TMS 7020 4 K OCTETS |
| FFFF | |

| | |
|---|---|
| 0000 | MEMOIRE VIVE ECRAN ET PROGRAMME UTILISATEUR 32 K OCTETS |
| 8000 | |



# ANNEXE 9

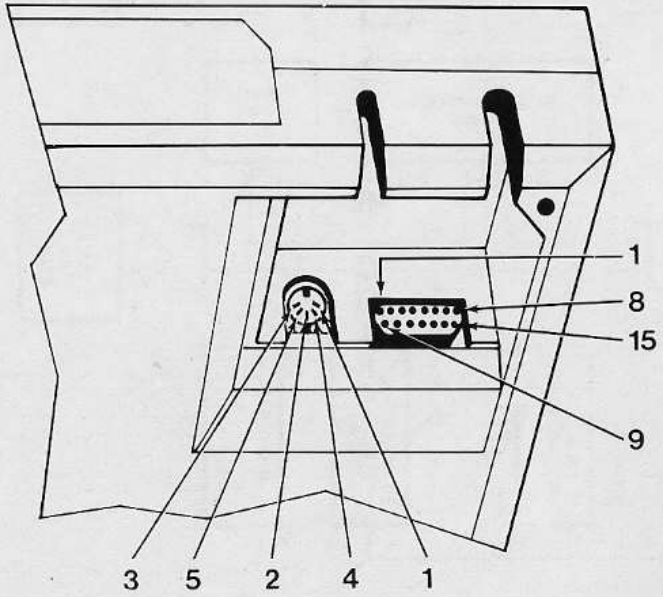
Brochage des prises de l'EXL 100

**Prise DIN**

| | |
|---|---|
| 1 | Sortie cassette |
| 2 | Masse |
| 3 | Entrée cassette |
| 4 | Sortie cassette |
| 5 | Entrée cassette |

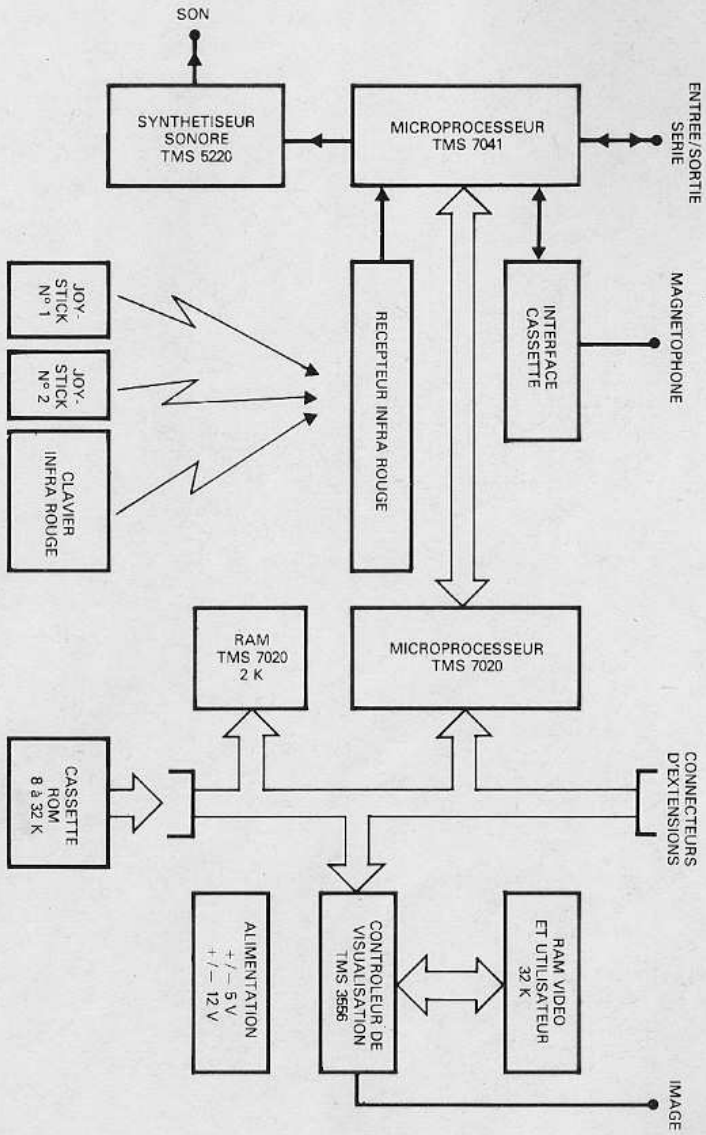
**Prise péritélévision**

| | |
|---|---|
| 1 | Commutation lente |
| 2 | Sortie audio |
| 3 | Synchronisation |
| 4 | Commutation rapide |
| 5 | Entrée vidéo TV |
| 6 | Vidéo bleue |
| 7 | Vidéo verte |
| 8 | Vidéo rouge |
| 9 | Masse |
| 10 | Masse |
| 11 | Masse |
| 12 | Masse |
| 13 | Masse |
| 14 | Masse |
| 15 | Masse |



# ANNEXE 10

## Schéma synoptique de l'EXL 100



Imprimerie BERGER-LEVRAULT, NANCY
779316-12-84
Dépôt légal : 1-1985.