Manuel du

BASIC NIVEAU II

Cat. Nr. 26-9403

Radio Shack®

TRS-80

MICRO COMPUTER SYSTEM

CUSTOM MANUFACTURED IN U S A BY RADIO SHACK A DIVISION OF TANDY CORPORATION

Avertissement

Nous avons rédigé ce manuel de référence en partant de l'idée que vous aviez déjà une longue habitude de la programmation en BASIC. Notre manuel du NIVEAU I – qui a du reste remporté un succès considérable – était destiné au débutant absolu, mais celui-ci ne se situe pas du tout dans la même optique.

En effet, si vous en êtes à ressentir le désir, ou le besoin, des ressources du BASIC II, c'est bien, selon nous, que vous avez des bases sérieuses en programmation et que le NIVEAU I n'a plus de secret pour vous.

Si vous faites vos premières armes dans le domaine de la programmation des micro-ordinateurs, nous vous conseillons vivement de commencer par travailler sur un TRS-80 NIVEAU I à l'aide du manuel que nous avons spécialement conçu à cet effet.

Si, d'autre part, vous avez déjà eu l'occasion de travailler avec d'autres versions du BASIC (autres micro-ordinateurs ou systèmes en temps partagé), vous devriez pouvoir aborder sans difficultés le manuel de référence du NIVEAU II.

Notre BASIC NIVEAU II offre infiniment plus de possibilités que le NIVEAU I; si vous utilisiez ce dernier depuis un certain temps, d'agréables surprises vous attendent – même si vous êtes quelque peu déconcerté au premier abord (par exemple, vos programmes en NIVEAU I devront être modifiés pour pouvoir tourner sur un TRS-80 NIVEAU II). Le manuel que voici se veut un outil de référence complet, et non une méthode progressive d'apprentissage ni un catalogue d'applications (ceci viendra en son temps).

Nous accueillerons avec reconnaissance toutes les critiques et suggestions que vous souhaiteriez formuler à propos de ce manuel.

FIRST EDITION
FIRST PRINTING - 1978

Printed in Belgium

TABLE DES MATIERES

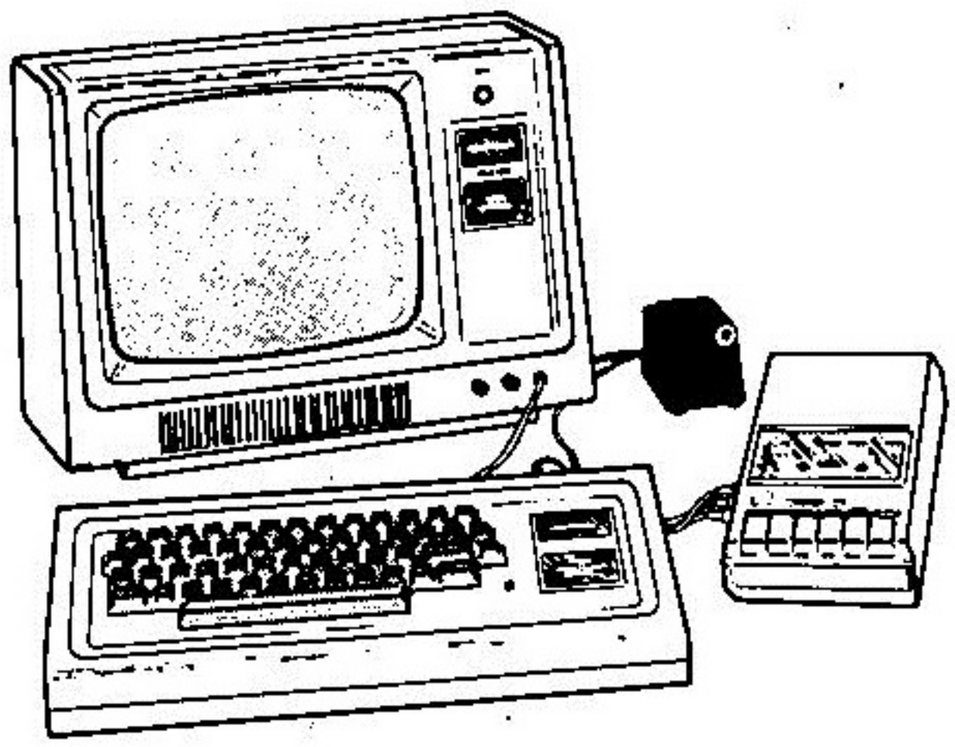
Annexes

Installation du système

Déballez soigneusement et complètement l'ensemble. Assurez-vous que vous avez bien tous les câbles, les notices, les cassettes, etc. Conservez l'emballage pour le cas où vous devriez transporter votre ordinateur.

Raccordement de l'écran vidéo et du clavier :

1. Raccordez le cordon d'alimentation de l'écran vidéo à une prise de courant (220 V CA).
2. Raccordez le cordon du bloc d'alimentation à une prise de courant (220 V CA).
3. Connectez le câble gris qui sort à l'avant de l'écran vidéo à la prise VIDEO située à l'arrière du bloc clavier. Attention : orientez convenablement les broches, la fiche ne s'adaptant que d'une seule façon.
   NB : Avant de continuer, assurez-vous que le bouton d'allumage (POWER) situé à l'arrière du clavier est sur la position arrêt (c'est-à-dire n'est pas enfoncé).
4. Connectez le câble gris du bloc d'alimentation à la prise POWER située à l'arrière du bloc clavier, Ici encore, attention à l'orientation des broches.



Raccordement de l'enregistreur à cassettes :

Cet enregistreur est un CTR-41 ou un CTR-80; sauf mention contraire, les instructions sont valables pour les deux appareils.

NB : Cette opération n'est utile qu'au moment d'enregistrer des programmes ou de charger dans le TRS-80 des programmes enregistrés sur cassettes.

1. Branchez l'enregistreur sur le secteur (220 V CA). Il n'est pas conseillé de le faire fonctionner sur piles.
2. Connectez le câble court (terminé par une fiche DIN à une extrémité et trois fiches à l'autre) à la prise TAPE située à l'arrière du bloc clavier. Attention à l'orientation des broches.
3. Connectez à l'enregistreur les 3 fiches de l'autre extrémité.

A. la noire à la prise EAR (écouteur) sur le côté de l'enregistreur; cette connexion permet à l'enregistreur d'envoyer des signaux au TRS-80, pour charger des programmes enregistrés dans la mémoire de l'ordinateur.
B. la plus grande des deux fiches grises à la prise AUX (source auxiliaire) de l'enregistreur; cette connexion permet au TRS-80 d'envoyer des signaux à l'enregistreur, pour enregistrer sur cassettes des programmes contenus en mémoire. Mettez en outre la fiche factice (fournie avec le CTR-41) dans la prise MIC (micro) ce qui déconnecte le micro incorporé et l'empêche donc de capter les bruits ambiants pendant que vous enregistrez des programmes sur cassettes.

NB : N'oubliez jamais cette fiche factice quand vous enregistrez. Attention, le CTR-80 ne nécessite pas de fiche factice.

fiche factice

C. la plus petite des deux fiches grises à la prise REM (commande à distance) de l'enregistreur; cette connexion permet au TRS-80 de commander automatiquement le moteur de l'enregistreur, c'est-à-dire de faire défiler la bande et de l'arrêter lors de l'enregistrement et de la reproduction.

Notes sur l'utilisation de l'enregistreur à cassettes :

Il y a un certain nombre de choses qu'il faut garder à l'esprit lors de l'utilisation de l'enregistreur :

1. Pour lire une cassette (c'est-à-dire pour charger un programme enregistré dans le TRS-80) réglez le volume de l'enregistreur entre 4 et 6. Enfoncez ensuite la touche PLAY de l'enregistreur puis tapez CLOAD et enfin poussez sur la touche **ENTER**. Tout ceci déclenchera le défilement de la bande. Un * apparait à la ligne supérieure de l'écran; un second * s'allume et s'éteint, ce qui indique que le programme est en cours de chargement. Quand ce chargement est terminé, le TRS-80 arrête automatiquement l'enregistreur et affiche READY sur l'écran. A ce moment, tout est prêt pour que vous fassiez exécuter le programme : tapez RUN puis poussez sur **ENTER** .
2. Pour enregistrer un programme à partir du TRS-80, enfoncez simultanément les touches RECORD et PLAY de l'enregistreur. Tapez ensuite CSAVE suivi d'un nom de fichier d'une seule lettre entre guillemets puis poussez sur **ENTER** pour introduire cette commande. A la fin de l'enregistrement, le TRS-80 arrête automatiquement l'enregistreur et affiche READY sur l'écran. Votre programme est maintenant enregistré sur cassette tout en existant toujours dans la mémoire du TRS-80. De nombreux utilisateurs réalisent un second, voire un troisième enregistrement, afin d'être certains d'en avoir au moins un de bonne qualité.
3. Servez-vous du compteur de l'enregistreur pour repérer le début et la fin de vos programmes enregistrés sur cassettes.
4. C'est avec les cassettes "Computer Tape" Radio Shack d'une durée de 10 minutes que vous obtiendrez les meilleurs résultats : elles sont spécialement conçues pour l'enregistrement de vos programmes. Si vous utilisez des cassettes ordinaires, veillez à n'employer que la meilleure qualité – comme les cassettes "SUPERTAPE" Realistic. N'oubliez pas que les cassettes audio (destinées à enregistrer du son) ont aux deux extrémités des amorces –

ce sont les sections bleues en mylar non magnétique – **sur lesquelles il est impossible d'enregistrer:**
avant d'enregistrer un programme, faites avancer la bande jusqu'au-delà de l'amorce.

5. Si vous ne vous servez pas de l'enregistreur pour charger ou enregistrer un programme, ne laissez pas les touches RECORD et PLAY enfoncées (poussez sur STOP).
6. Si vous souhaitez rebobiner (REWIND) une cassette ou la faire avancer rapidement (FAST-F), tapez CLOAD et poussez sur **ENTER**. Lorsque la cassette est positionnée à l'endroit voulu, enfoncez le bouton RESET (retour au point de départ) à l'arrière gauche du bloc clavier (sous le clapet de la sortie d'expansion). Vous pourriez aussi retirer la petite fiche grise de la prise REM; cependant, la répétition de cette opération pourrait déteriorer la fiche et n'est donc pas conseillée.
   Attention, ces opérations ne sont pas nécessaires pour le CTR-80.
7. Si vous désirez conserver de façon permanente un programme enregistré, brisez les ergots anti-effacement situés à l'arrière de la cassette (voir notice d'emploi de l'enregistreur).
8. N'exposez pas vos cassettes enregistrées à l'action des champs magnétiques. Evitez de les laisser à proximité du bloc d'alimentation.
9. Pour vérifier si un programme a été enregistré sur une cassette donnée, débranchez la fiche de la prise EAR (écouteur). (Pour le CTR-41 débranchez également la fiche REM afin de pouvoir le commander au moyen de ses touches). Faites défiler la bande; vous entendrez un sifflement correspondant aux signaux du programme enregistré.
10. Pour obtenir des enregistrements de bonne qualité, veillez à l'extrême propreté des têtes et du chemin de la bande. Un nouvel enregistreur devrait être nettoyé avant utilisation ainsi que toutes les 4 heures d'utilisation. D'autre part les têtes seront démagnétisées regulièrement.
    Vous trouverez dans votre magasin TANDY Radio Shack habituel une gamme complète d'accessoires pour enregistreurs (liquide de nettoyage, coton-tiges, cassette démagnétisante, ...).

Remarques :

1. Avant de charger un programme dans l'ordinateur, assurez-vous que la bande est positionnée à un endroit sans signal précédant un enregistrement. Si vous essayez de charger à partir du milieu d'un programme, vous risquez de bloquer l'ordinateur (auquel cas vous devrez pousser sur le bouton RESET pour le libérer).
   La même remarque s'applique à la commande CLOAD? qui ordonne la comparaison du programme en mémoire avec un programme sur une cassette.
2. Le CTR-41 doit avoir l'interrupteur TONE positionné sur HIGH; le CTR-80, quant à lui, n'a pas de commande de tonalité.

Indications sur le chargement de programmes à partir d'une cassette

Plusieurs facteurs peuvent réduire l'efficacité du système à cassettes. Le plus sensible est le **volume.** Un volume trop bas peut entraîner une perte d'information. Un volume trop élevé peut provoquer des distorsions et la transmission du bruit de fond comme information valide. Ces deux situations entraînent des erreurs de lecture.

Les valeurs recommandées * pour le chargement à partir d'une cassette sont les suivantes :

	Enregistrée par l'utilisateur	Pré-enregistrée par Radio Shack
NIVEAU I	4-6	5 1/2 - 6 1/2
NIVEAU II	7-8	7 1/2 - 8 1/2

Si les astérisques n'apparaissent pas durant le chargement, essayez de diminuer le volume. Il est utile également de repérer à l'ouïe le début du programme (débrancher la fiche EAR). Si les astérisques apparaissent mais que la seconde ne clignote pas, essayez d'augmenter le volume. Si le problème n'est pas encore résolu, nettoyez les têtes.

Que faire en cas d'erreur au chargement ?

Il peut arriver, mais le cas est très rare, qu'une erreur mineure se produise lors du chargement d'un programme mais qu'aucun message d'erreur n'apparaisse. La meilleure manière de vérifier le programme est de le LISTer. S'il semble correct, utiliser la commande CLOAD? qui comparera le programme qui est en mémoire à celui qui est sur cassette. S'ils ne sont pas exactement identiques, le message "BAD" (c'est-à-dire "MAUVAIS") sera affiché. On remédiera généralement à cet incident en réglant le volume (par une légère augmentation par exemple).

*Ces valeurs sont valables pour les enregistreurs RADIO SHACK dont le volume s'étend de 0 à 10. Pour d'autres modèles, on recherchera par têtonnements le volume optimal.

I / Informations générales

Vous trouverez dans ce chapitre un aperçu du BASIC NIVEAU II qui reprend ses principales caractéristiques, les différences qu'il présente avec le NIVEAU I, et, d'une manière générale, tout ce dont vous aurez besoin pour commencer. Vous trouverez aussi un bref glossaire en fin de ce chapitre.

Allumage

Connectez le bloc clavier, le vidéo et le bloc d'alimentation selon les instructions de la section précédente. Branchez le vidéo et l'alimentation sur le secteur (220 V CA). Enfoncez les boutons POWER du vidéo et du bloc clavier (à l'arrière). Laissez chauffer le tube cathodique du vidéo quelques instants.

Les mots MEMORY SIZE? – apparaissent à l'écran. C'est le moment de spécifier la taille de la mémoire à protéger dans laquelle seront chargés des programmes en langage machine grâce à l'instruction SYSTEM. Vous répondrez soit en indiquant l'adresse du segment de mémoire à protéger, soit en poussant sur **ENTER** si vous ne voulez pas protéger de segment, ce qui sera le cas le plus fréquent; dans ce dernier cas toute la mémoire (c'est-à-dire 3284 bytes pour une machine de 4K ou 15572 bytes pour une machine de 16K) sera disponible pour les programmes BASIC.

Vous verrez alors apparaître à l'écran :

```
RADIO SHACK LEVEL II BASIC
READY
>_
```

Vous pouvez alors travailler en BASIC NIVEAU II.

Remarque :

D'une manière générale, Je même qu'en NIVEAU I, vous devrez pousser sur **ENTER** après avoir tapé une réponse de manière à faire accepter cette entrée par l'ordinateur. Nous verrons plus tard qu'il est possible de faire accepter des données tapées sans qu'il soit nécessaire à l'utilisateur de pousser sur **ENTER**.

Modes de travail

Ceux-ci sont au nombre de quatre : Commande, Exécution, Editeur et Moniteur. Les modes Commande et Exécution sont ceux du NIVEAU I. Dans le mode Commande, l'ordinateur exécute votre commande dès que vous l'avez introduite. C'est dans ce mode que vous introduirez des programmes et que vous exécuterez des calculs en mode "calculatrice". Ce mode est caractérisé par l'apparition de >_ à l'écran.

Le mode Exécution est celui qui suit l'introduction de RUN provoquant ainsi l'exécution du programme BASIC résident. Contrairement au NIVEAU I, le NIVEAU II initialise les variables numériques à zéro et les variables alpha-numériques à vide lorsque vous commandez RUN.

Le mode Editeur du NIVEAU II vous fera gagner beaucoup de temps. Il vous permet d'*éditer* (modifier, ajouter ou supprimer des fragments) les lignes d'un programme. Au lieu de retaper entièrement une ligne, il vous suffit de modifier la partie incorrecte.

Remarque :
Lorsque l'ordinateur rencontre, lors de l'exécution d'un programme, une erreur de syntaxe, il vous rend le contrôle en mode Editeur pour la ligne correspondant à l'erreur. Pour quitter le mode Editeur, vous taperez simplement : Q.

Le mode Moniteur vous permet de charger en mémoire des données ou des programmes en langage machine à partir d'une cassette. Ces programmes (ou données) peuvent être utilisés par vos programmes BASIC ou peuvent être indépendants.

Les touches de fonction

Le BASIC NIVEAU II offre un certain nombre de touches de fonction dont celles du NIVEAU I. La fonction d'une touche dépend du mode de travail de l'ordinateur.

En mode Commande :

ENTER Indique la fin de la ligne qui vient d'être introduite; l'ordinateur examine cette ligne et réagit selon son contenu. Si la ligne ainsi introduite ne comporte pas de numéro initial, l'ordinateur interprétera la (ou les) commande(s) qu'elle contient. Si la ligne comporte un numéro, l'ordinateur la stockera comme ligne du programme résident.

← Déplacement du curseur d'une position vers la gauche et effacement du caractère courant.

SHIFT ← Efface la ligne logique courante et positionne le curseur à son début.

↓ Passage à la ligne physique suivante.

: Sépare deux instructions BASIC dans une même ligne.
Exemple : PRINT A : INPUT B, C

→ Avance le curseur au taquet de tabulation suivant. Les taquets occupent les positions 0, 8, 16, 24, 32, 40, 48, 56.

SHIFT → Commande l'affichage à 32 caractères par ligne.

CLEAR Efface l'écran et commande l'affichage à 64 caractères par ligne.

En mode Exécution :

SHIFT @	Pause; interrompt l'exécution du programme. La frappe de n'importe quelle autre touche entraîne la reprise de l'exécution. Permet également d'interrompre l'affichage d'un LIST pour examiner les lignes du programme.
BREAK	Arrête l'exécution du programme. L'exécution peut être reprise grâce à la commande CONT.
ENTER	Lorsque l'ordinateur attend l'entrée de données (instruction INPUT) **ENTER** indique que cette introduction est terminée et que l'ordinateur peut examiner ce que vous venez de taper.

Les touches de fonction du mode Editeur seront étudiées au chapitre 9.

Noms de variables

Le premier caractère d'un nom de variable doit être une lettre; elle peut être suivie d'une lettre ou d'un chiffre. Les noms suivants sont donc valides et désignent des variables distinctes :

A A2 AA AZ G9 GP M MU ZZ Z1

Un nom peut comporter plus de deux caractères, mais seuls les deux premiers serviront à l'identification de la variable; par exemple, "SOMME", "SOUS" et "SO" désignent une seule et même variable NIVEAU II.

Comme vous pouvez l'imaginer, cela met à votre disposition un nombre considérable de noms distincts (plus de 900). **Cependant, un nom de variable ne peut contenir de mot ayant une signification en BASIC NIVEAU II.** Par exemple, "BONUS" ne peut être utilisé puisqu'il contient le mot clé BASIC "ON". La liste complète des "mots réservés" qui ne peuvent apparaître dans un nom de variable pourra être consultée dans l'annexe A de ce manuel.

Types de variables

Il y a quatre types de variables en NIVEAU II : les variables entières, simple précision, double précision et alphanumériques. Les variables des trois premiers types peuvent contenir des valeurs numériques de différents degrés de précision; celles du dernier type peuvent contenir des chaînes (ou séquences) de 0 à 255 caractères – lettres, chiffres et signes spéciaux. Le NIVEAU I n'admettait que deux variables alphanumériques, A$ et B$; le NIVEAU II permet d'en utiliser un nombre quelconque. La déclaration du type d'une variable peut être réalisée en suffixant le nom de la variable (en ajoutant à sa droite) d'un caractère spécial comme indiqué dans le tableau ci-dessous.

Type de variable	Caractère de déclaration	Exemples	Exemples de valeurs
Entières (nombres entiers de –32768 à 32767)	%	A%, B9%	–30, 123, 3, 5001
Simple précision (6 chiffres significatifs)	!	A!, AA!, Z1!	1, –50, .123456, 345678, 0.0123456
Double précision (16 chiffres significatifs)	#	B0#, X#, DPR#	–300.12345678, 3.141592653589, 1.000000000000001
Alphanumériques (Chaînes inférieures à 256 caractères)	$	A$, TX$, NOM$	"CLAIRE LEMERCIER", "AUJOURD'HUI 2 AVRIL", "1 + 2 = ?"

Un même nom peut être utilisé pour désigner des variables de types différents; **c'est grâce au caractère de déclaration que l'ordinateur les distinguera.**

Par exemple, A$, A%, A! et A# sont les noms de variables distinctes. Des noms non suffixés d'un caractère de déclaration désignent, par défaut, des variables en simple précision; cette déclaration implicite peut être modifiée par des instructions de DEFinition (voir chapitre 4).

Tableaux

Tout nom de variable valide peut être utilisé pour désigner un tableau non seulement à une dimension (comme en NIVEAU I) mais à un nombre quelconque de dimensions. L'instruction DIM permet de fixer les dimensions des tableaux. En fonction du type qui sera déclaré, un tableau pourra contenir des valeurs entières, simple précision, double précision ou alphanumériques. Un chapitre complet sera consacré aux tableaux.

Exemples : A$ (X, Y, Z) correspond à un tableau alphanumérique à trois dimensions.
G3 (I, J) correspond à un tableau à deux dimensions de valeurs en simple précision.
G# (K) correspond à un tableau en double précision à une seule dimension.

Opérateurs arithmétiques

Le NIVEAU II utilise les mêmes opérateurs arithmétiques que le NIVEAU I : + (addition), — (soustraction), * (multiplication) et / (division); il en possède un supplémentaire, particulièrement utile : l'opérateur exposant ↑ (2 ↑ 3 = 8)
Par exemple, pour calculer $6*2^{1/3}$, on écrira : 6*2 ↑ (1/3)

Remarque :

Certains TRS-80 impriment le caractère [au lieu de la flèche ↑ .

Les opérateurs de comparaison

Ils sont les mêmes qu'en BASIC NIVEAU I :

< (plus petit que)	> (plus grand que)	= (égal à)
<> (différent de)	<= (inférieur ou égal à)	>= (supérieur ou égal à)

On utilisera ces opérateurs dans les instructions IF ... THEN ainsi que dans les expressions arithmétiques logiques.

Exemple :

```
100  IF C<=0  THEN C=127
```

Les opérateurs logiques

En BASIC NIVEAU I, nous utilisions les symboles * et + pour désigner les opérateurs logiques ET et OU. En NIVEAU II, nous utiliserons directement les termes AND et OR, qui sont les traductions anglaises de ET et OU respectivement. Nous disposons d'un opérateur supplémentaire : NOT (traduit par NON) dont l'effet est d'inverser une condition.

Exemples :

```
 50  IF Q = 13 AND R2 = 0 THEN PRINT "PRET"
```

```
100  Q = (G1 <0) AND (G2 <L)
```

Q = –1 si les deux expressions sont vraies; sinon, Q = 0.

Q = –1 si l'une au moins des expressions est vraie; sinon Q = 0.

```
300  Q = NOT (C >3)
```

Q = –1 si l'expression est fausse; si elle est vraie, Q = 0.

```
400  IF NOT (P AND Q) THEN PRINT "P ET Q NE SONT PAS TOUS DEUX
     EGAUX A –1"

500  IF NOT (P OR Q) THEN PRINT "NI P NI Q NE VALENT –1"
```

Remarque :
La valeur VRAI est représentée par –1 et la valeur FAUX par 0.

Les opérateurs de chaînes de caractères

Des chaînes de caractères peuvent être, en BASIC NIVEAU II, comparées et concaténées (c'est-à-dire mises bout à bout pour constituer une nouvelle chaîne). Le chapitre 5 leur sera consacré.

Symbole	Signification	Exemple
<	précède dans l'ordre alphabétique	"A" < "B"
>	suit dans l'ordre alphabétique	"JULES" > "JEAN"
=	est égal à	"OUI" = B$
< >	est différent de	IF A$ < > B$ THEN C = 1
< =	précède ou est égal à	IF A$ < = ZZ$ PRINT "FIN"
> =	suit ou est égal à	IF L1$ > = "DUPONT" THEN 400
+	concaténation des deux chaines	A$ = B1$ + B2$ TR$ = "TRS-" + "80" + A$

Ordre d'exécution des opérations

Les opérations situées à l'intérieur des parenthèses les plus intérieures sont réalisées en premier lieu; ensuite, celles du niveau suivant sont effectuées, et ainsi de suite jusqu'aux opérations qui ne sont pas entre parenthèses. A un même niveau, les opérations sont réalisées dans l'ordre suivant :

Exposant : A ↑ B

Négation : –X

* et / (de gauche à droite)

+,– (de gauche à droite)

<, >, =, < =, > =, < > (de gauche à droite)

NOT

AND

OR

Fonctions intrinsèques

La plupart des sous-routines du manuel NIVEAU I sont intégrées au NIVEAU II. Elles sont plus rapides, plus précises (6-7 chiffres significatifs) et plus faciles à utiliser.

Fonctions graphiques

Le NIVEAU II possède les mêmes fonctions graphiques SET, RESET ET POINT que le NIVEAU I pour allumer, éteindre et tester des segments de l'écran. (il existe cependant quelques différences – voir chapitre 8).
Une caractéristique importante du NIVEAU II est la possibilité de choisir un affichage en 64 ou en 32 caractères par ligne. A l'allumage, la machine est en mode 64 c/l; si l'on tape simultanément SHIFT et →, elle est en mode 32 c/l. L'exécution de CLS ou NEW, ou la frappe de CLEAR la fait retourner en mode 64 c/l. Il est également possible de passer en mode 32 c/l en exécutant PRINT CHR$ (23). Voir Chapitre 5.

Messages d'erreur

En NIVEAU I, la détection d'une erreur était indiquée par l'affichage de HOW?, WHAT?, ou SORRY et de la ligne incriminée contenant un point d'interrogation à l'endroit de l'erreur. Le NIVEAU II vous donne des informations beaucoup plus précises sur le type d'erreur grâce à un jeu étendu de codes d'erreurs (voir Annexe). Le numéro de la ligne est également indiqué. Cependant, c'est à vous de localiser l'erreur dans la ligne.

Abréviations

Peu d'abréviations sont permises en NIVEAU II. Il faudra donc oublier les R., L., P., etc. du NIVEAU I. Cependant, le stockage en mémoire des programmes est plus efficace qu'en NIVEAU I; vous pourrez ainsi écrire des programmes plus longs dans un même espace de mémoire.

Les abréviations autorisées sont :

? pour PRINT

' pour REM

. pour désigner la dernière ligne tapée, imprimée ou éditée, ou encore celle où une erreur s'est produite.

Frappe multiple (Rollover)

Sur le TRS-80 NIVEAU I (et sur beaucoup d'autres ordinateurs) vous deviez relâcher une touche avant de frapper sur la suivante, sinon l'ordinateur ignorait cette dernière. Cette contrainte disparaît en NIVEAU II, ce qui permet une frappe beaucoup plus rapide.

Glossaire pour le BASIC NIVEAU II

adresse valeur spécifiant l'emplacement d'un byte en mémoire; le NIVEAU II en utilise l'expression décimale.

alphanumérique caractérise l'ensemble des lettres A-Z, des chiffres 0-9 et de différents signes de ponctuation et caractères spéciaux.

argument valeur fournie à une fonction et qui servira à déterminer le résultat.

tableau arrangement ordonné d'éléments selon une ou plusieurs dimensions.

ASCII American Standard Code for Information Interchange; code numérique de représentation des caractères alphanumériques ainsi que de différentes commandes ou indicateurs; en NIVEAU II, les codes sont utilisés sous leur forme décimale.

assembleur programme qui convertit un programme en langage symbolique (qui s'appelle aussi, et malheureusement, assembleur) en un programme équivalent en langage machine.

BASIC Beginner's All-purpose Symbolic Instruction Code, c'est-à-dire Code d'Instructions Symboliques "Tous-usages" pour Débutants.

baud unité de vitesse de transmission mesurée en bits par seconde; l'interface cassette du NIVEAU II opère à 500 bauds, soit 500 bits par seconde.

nombre binaire représentation d'un nombre en base 2, exprimée à l'aide de "0" et de "1".

bit chiffre binaire; la plus petite quantité d'information et de mémoire.

byte groupe de 8 bits (appelé aussi octet) qui est la plus petite cellule de mémoire accessible en BASIC

nombre décimal représentation d'un nombre en base 10, exprimée à l'aide des chiffres "0" à "9".

expression combinaison d'une ou plusieurs constantes, variables, fonctions et opérations fournissant une valeur après évaluation.

fichier collection structurée de données (p. ex. fichier d'adresses, de factures)

nombre hexadécimal représentation d'un nombre en base 16, exprimée à l'aide des chiffres "0" à "9" et des lettres "A" à "F".

fonction intrinsèque fonction (généralement complexe) directement codée dans la ROM BASIC et qui peut être simplement appelée par une instruction BASIC.

expression logique expression qui peut prendre pour valeur Vrai (représenté par −1) ou FAUX (représenté par 0).

langage machine langage directement compréhensible par l'ordinateur et qui s'écrit en binaire.

port l'un des 256 canaux par lesquels l'ordinateur peut recevoir et envoyer des données

RAM ou Mémoire Vive mémoire à accès direct dans laquelle il est possible de lire et d'écrire; la RAM est disponible à l'utilisateur pour stocker ses programmes et ses données.

ROM ou Mémoire Morte mémoire à accès direct dans laquelle il n'est possible que de lire des programmes et des données préalablement écrits par le constructeur; le BASIC NIVEAU II est écrit dans une ROM.

routine séquence d'instructions réalisant une fonction déterminée.

instruction phrase complète en BASIC.

chaîne séquence de caractères alphanumériques d'une longueur comprise entre zéro (chaîne vide) et 255.

sous-routine routine qui peut être exécutée plusieurs fois à partir d'endroits différents bien qu'elle ne soit écrite qu'à un seul endroit.

variable objet qui peut prendre toute valeur d'un ensemble déterminé.

nom de variable étiquette (sous forme de chaîne) permettant de désigner une variable.

2 / Les commandes

Dès que le signal d'invite **>** apparaît sur l'écran, l'ordinateur se trouve en mode Commande. Vous pouvez alors taper une commande, l'introduire en poussant sur **ENTER** ; l'ordinateur vous répondra immédiatement. Ce chapitre décrit les commandes que vous utiliserez pour donner des ordres à l'ordinateur, – changer de mode, commencer une opération d'entrée et de sortie, modifier un programme qui est en mémoire, etc. Toutes les commandes, à l'exception de CONT, peuvent également être utilisées comme instructions dans un programme. Cela peut être très utile dans certains cas.

Ce chapitre décrira les commandes suivantes :

AUTO	CONT	EDIT	SYSTEM
CLEAR	CSAVE	LIST	TROFF
CLOAD	DELETE	NEW	TRON
CLOAD?		RUN	

AUTO *numéro de ligne, incrément*

Déclenche la fonction de numérotation automatique des lignes, vous offrant ainsi une introduction plus aisée des programmes – il ne vous reste qu'à taper la ligne sans vous préoccuper de son numéro. Vous pouvez spécifier le numéro initial ainsi que l'incrément (ou pas) des numéros successifs. Vous pouvez aussi taper simplement AUTO puis pousser sur **ENTER** , dans ce cas le numéro initial sera 10 et l'incrément 10 également. Chaque fois que vous pousserez sur **ENTER** , l'ordinateur passera au numéro de ligne suivant.

Exemples :	**produit la séquence suivante :**
AUTO	10, 20, 30, ...
AUTO 5,5	5, 10, 15, ...
AUTO 100	100, 110, 120, ...
AUTO 100,25	100, 125, 150, ...

Pour arrêter cette fonction, enfoncez la touche BREAK.
Remarque : lorsque cette fonction produit le numéro d'une ligne qui existe déjà, un astérisque apparaît à la suite de ce numéro affiché. Si vous ne voulez pas détruire cette ligne, arrêtez la fonction par BREAK.

CLEAR *n*

[en français : REINITIALISER]

Utilisée saus argument (tapez CLEAR puis poussez sur **ENTER**) cette commande réinitialise toutes les variables numériques à zéro et toutes les variables alphanumériques à vide. Utilisée avec un argument (p. ex. CLEAR 100), cette commande réalise, en plus de la fonction ci-dessus, la réservation du nombre de bytes spécifiés pour y ranger ultérieurement des chaînes de caractères. A l'allumage de l'ordinateur, un CLEAR 50 est exécuté automatiquement.
Exemple : CLEAR 120 réserve 120 bytes pour des chaînes de caractères.

CLOAD *"nom de fichier"*

[en français : CHARGER A PARTIR DE LA CASSETTE]

Cette commande vous permet de charger dans la mémoire un programme BASIC stocké sur une cassette. Assurez-vous préalablement que les connexions et les réglages sont corrects, rebobinez la cassette et poussez sur la touche PLAY de l'enregistreur.

Remarque :

En NIVEAU II, CLOAD et CSAVE travaillent à une vitesse de 500 bauds, soit deux fois celle du NIVEAU I. Le volume doit alors être réglé à un niveau plus faible. Pour un CTR-41 ou CTR-80, essayez un volume de 4 à 6 lors du chargement de cassettes que vous avez enregistrées. Des cassettes pré-enregistrées pourront exiger un volume plus élevé. Faites quelques essais pour obtenir des résultats corrects.

La commande CLOAD sans argument va faire démarrer l'enregistreur et va charger le premier programme rencontré. Cependant, le NIVEAU II vous permet également de spécifier un "fichier" dans votre commande. Par exemple, CLOAD "A" va entraîner une recherche sur la cassette du premier programme portant le nom "A", son chargement en mémoire puis le retour en mode commande. Il n'est pas nécessaire de connaître exactement l'emplacement de ce programme : vous faites commencer la recherche au début de la bande, et l'ordinateur fera le reste. Pendant cette recherche, le nom des programmes rencontrés apparaît sur l'écran, suivi de l'habituel astérisque clignotant.

Seul le premier caractère du nom spécifié est pris en compte par l'ordinateur dans une commande CLOAD, CLOAD? et CSAVE.
Le chargement d'un programme à partir d'une cassette efface automatiquement le programme précédent dans la mémoire. Voir aussi **CSAVE.**.

CLOAD? *"nom de fichier"*

Vous permet de comparer le programme qui est sur une cassette avec celui qui est actuellement dans la mémoire centrale. Cette opération est particulièrement utile lorsque vous avez sauvé un programme sur une cassette (par CSAVE) et que vous désirez vérifier que cette copie s'est effectuée correctement. Si le programme sauvé porte un nom, vous pouvez spécifier ce nom : CLOAD? *"nom de fichier"*. Si vous omettez cet argument, la comparaison se fera avec le premier programme trouvé. Pendant l'exécution de CLOAD?, le programme sur la cassette et celui qui est en mémoire sont comparés byte par byte. S'il y a la moindre différence, le message "BAD" apparaîtra, indiquant que le sauvetage a été "MAL" effectué. Il ne vous restera alors qu'à recommencer ce sauvetage. (CLOAD?, contrairement à CLOAD, n'efface pas le programme résident).

CONT

Lorsque l'exécution d'un programme a été suspendue par une instruction STOP dans le programme, ou la frappe de la touche BREAK, tapez CONT pour faire reprendre l'exécution à partir du point d'arrêt.
Pendant cet arrêt, vous pouvez examiner le contenu des variables (par une commande PRINT) et même le modifier. Tapez ensuite CONT puis poussez sur **ENTER** et l'exécution se poursuivra avec ces nouvelles valeurs. L'usage de BREAK, STOP et CONT constitue surtout un outil d'aide à la mise au point de programmes.

Remarque : CONT ne peut plus être utilisé si vous avez, entretemps, modifié votre programme; il en est de même lorsque le programme s'est terminé normalement par l'exécution d'un END. Voir aussi **STOP**.

CSAVE *"nom de fichier"* [en français : SAUVER SUR CASSETTE]

Permet le sauvetage du programme résidant en numéro sur une cassette. (veillez au réglage correct, enfoncez les touches RECORD et PLAY simultanément). Vous devez ici spécifier un nom de fichier qui peut être tout caractère alphanumérique autre que les guillemets ("). Le programme sauvé portera le nom, par lequel il pourra être repéré lors de l'exécution d'une commande CLOAD "nom de fichier". Il est important d'écrire également ce nom sur l'étiquette de la cassette en vue d'une utilisation ultérieure.

Exemples :

CSAVE "1" sauve le programme résident sous le nom "1"

CSAVE "A" sauve le programme résident sous le nom "A"

Voir aussi **CLOAD** et **CLOAD?**

DELETE *numéro de ligne-numéro de ligne* [en français : SUPPRIMER]

Efface les lignes de programme dont les numéros sont spécifiés. Il est possible de supprimer une seule ligne ou plusieurs :

DELETE *numéro de ligne*	efface la ligne spécifiée.
DELETE *numéro de ligne-numéro de ligne*	efface les lignes dont les numéros vont du premier numéro spécifié jusqu'au deuxième.
DELETE-*numéro de ligne*	efface les lignes depuis la première du programme jusqu'à celle dont le numéro est spécifié.

Le numéro supérieur spécifié doit correspondre à une ligne existant réellement.

Exemples :

DELETE 5	efface la ligne 5 de la mémoire; si cette ligne n'existe pas, une erreur est annoncée.
DELETE 11-18	effacer les lignes dont les numéros vont de 11 à 18 (celles-ci incluses).

Si vous venez d'introduire ou d'éditer une ligne, celle-ci est dite "courante" et est désignée par le signe . ; tapez simplement DELETE. pour la supprimer.

EDIT *numéro de ligne*

Fait entrer l'ordinateur en mode Editeur dans lequel vous pouvez modifier le programme résident. Ce mode est d'autant plus utile que votre programme est long et complexe. Le mode Editeur disposant de ses propres sous-commandes, nous lui avons consacré le chapitre 9.

LIST *numéro de ligne-numéro de ligne*

Provoque l'affichage des lignes spécifiées du programme résident. Utilisée sans argument, la commande LIST entraîne l'affichage en rouleau de toutes les lignes du programme (scrolling). Pour arrêter ce défilement automatique, enfoncez simultanément les touches SHIFT et @. Pour faire redémarrer l'affichage en rouleau, poussez sur n'importe quelle touche.
Si vous voulez examiner une ligne déterminée, spécifiez-en le numéro dans la commande LIST; si vous êtes intéressé par une séquence déterminée, donnez-en les numéros extrêmes.

Exemples :

LIST 50	affiche la ligne 50
LIST 50-150	affiche les lignes depuis 50 jusqu'à 150 (celles-ci incluses)
LIST 50-	affiche les lignes depuis 50 jusqu'à la fin du programme
LIST.	affiche la ligne courante (qui vient d'être introduite ou éditée)
LIST -50	affiche les lignes depuis la première du programme jusqu'à la ligne 50 (incluse).

NEW

[en français : NOUVEAU]

Efface toutes les lignes du programme résident, réinitialise les variables numériques à zéro et les variables alphanumériques à vide. L'effet d'un CLEAR *n* précédent n'est cependant pas modifié.

RUN *numéro de ligne*

Commande à l'ordinateur d'exécuter le programme résident. Si aucun numéro de ligne n'est spécifié, l'exécution commence à partir de la ligne de numéro le plus bas. Si un numéro est spécifié, c'est à la ligne portant ce numéro que l'exécution commencera (un numéro de ligne inexistant est un cas d'erreur). La prise en compte d'une commande RUN entraîne l'exécution automatique d'un CLEAR.

Exemples :

RUN	exécution du programme à partir de la première ligne.
RUN 100	exécution du programme à partir de la ligne 100.

RUN peut être inséré dans un programme; il s'agit d'un moyen pratique de recommencer l'exécution dans un état initial lors de l'exécution de programmes à boucles sans fin tels que des programmes de jeux.

SYSTEM

Fait entrer l'ordinateur en mode Moniteur, qui vous permet de charger (et de faire exécuter) des programmes en langage machine ou des données. Radio Shack propose plusieurs programmes complets (des "progiciels" !) en langage machine tels que le système IN-MEMORY INFORMATION, ou encore l'EDITOR/ ASSEMBLER, qui est un programme en langage machine vous permettant de créer vous-même vos propres programmes en langage machine (ce que l'on appelle un "programme objet")
Pour charger un tel programme, la procédure à suivre est la suivante : Tapez SYSTEM et poussez sur **ENTER** .

*? va alors apparaître. Vous tapez ensuite le nom du fichier correspondant au programme ou aux données concernés. Lorsque le chargement est terminé,

*? apparaîtra à nouveau. Tapez / suivi de l'adresse (sous forme décimale) où l'exécution doit débuter. Vous pouvez aussi taper simplement / sans adresse puis pousser sur **ENTER** . L'adresse de début est, dans ce cas, celle qui est spécifiée par le programme objet lui-même.

TROFF

[en français : DECLENCHER TRACE]

Arrête la fonction Trace. Voir **TRON.**

TRON

[en français : ENCLENCHER TRACE]

Enclenche la fonction Trace qui vous permet de suivre "à la trace" l'exécution d'un programme en vue de sa mise au point. Chaque fois que l'ordinateur passe à l'exécution d'une autre ligne du programme, son numéro est affiché sous la forme : <numéro de ligne >.

Introduisons par exemple le programme suivant :

```
10 PRINT "DEBUT"
20 PRINT "EN COURS"
30 GOTO 20
40 PRINT "FIN"
```

Tapez TRON, **ENTER** , puis RUN **ENTER** ; vous obtiendrez :

```
<10>    DEBUT
<20>    EN COURS
<30>    <20> EN COURS
<30>    <20> EN COURS
```

etc.

Pour arrêter l'exécution et "geler" ainsi l'affichage, poussez simultanément sur SHIFT et @. Frappez n'importe quelle touche pour faire reprendre l'exécution.

Comme vous pourrez le voir sur l'écran, ce programme réalise une boucle sans fin.

Cette liste de numéros vous montre exactement ce qui se passe. Pour consulter ou modifier des variables, stoppez le programme par BREAK puis faites-lui reprendre son exécution.

L'arrêt de cette fonction est réalisé par TROFF. Les commandes TRON et TROFF peuvent être insérées à l'intérieur d'un programme de façon à indiquer qu'une certaine ligne est exécutée.

Par exemple, la séquence

```
50    TRON
60    X=X/3.14
70    TROFF
```

vous permet d'observer l'exécution de la ligne 60, à condition bien sur qu'elle ne résulte pas d'un branchement direct vers 60, auquel cas la ligne 50 ne serait pas exécutée.
Chaque fois que ces trois lignes seront exécutés, <60 > <70 > apparaîtra sur l'écran, indiquant que la division a bien été effectuée.
Lorsqu'un programme est ainsi vérifié et corrigé, les instructions TRON et TROFF peuvent être supprimés.

3 / Les Entrées / Sorties

Les instructions décrites dans ce chapitre vous permettent d'envoyer des données du clavier vers l'ordinateur, de l'ordinateur vers l'écran, ainsi que de l'ordinateur vers l'interface cassette et vice versa. Elles serviront surtout, à l'intérieur des programmes, à introduire des données et à produire des résultats et des messages.

Les instructions que l'on examinera sont les suivantes :

PRINT
@ (paramètre de PRINT)
TAB (paramètre de PRINT)
USING (PRINT avec format)
INPUT
DATA
READ
RESTORE
PRINT # (sortie vers cassette)
INPUT # (entrée par cassette)

PRINT *liste de valeurs*

[en français : IMPRIMER]

Cet ordre permet d'imprimer une valeur ou une liste de valeurs sur l'écran. Les valeurs peuvent être sous forme de constantes alphanumériques (entre guillemets), de variables alphanumériques, de constantes numériques (nombres), de variables numériques, ou d'expressions faisant intervenir les éléments ci-dessus. Les valeurs à imprimer seront séparées dans la liste par des virgules ou des points-virgules. La virgule fait avancer le curseur au début de la zone d'impression suivante (l'écran est divisé en 4 zones de 16 caractères). Le point-virgule entraîne l'impression de la valeur qui suit sans espace de séparation.

Exemples

```
50    X=5
100   PRINT 25; "EST EGAL A"; X ↑ 2
RUN

25  EST EGAL A 25
```

```
10    A$="CHAINE"
20    PRINT A$;A$,A$; "  ";A$

RUN

CHAINECHAINE        CHAINE CHAINE
```

Les nombres positifs sont imprimés précédés d'un espace (au lieu du signe +); Tous les nombres sont imprimés suivis d'un espace; des chaînes de caractères sont imprimées sans espace ni devant ni derrière (vous pouvez en insérer comme constante entre guillemets comme ci-dessus à la ligne 20).

```
10 PRINT "ZONE 1", "ZONE 2","ZONE 3", "ZONE 4", "ZONE 1 ETC"

RUN
ZONE 1          ZONE 2          ZONE 3          ZONE 4
ZONE 1 ETC
```

```
10 PRINT "ZONE 1", ,"ZONE 3"

RUN

ZONE 1                          ZONE 3
```

Le curseur passe à la zone suivante chaque fois qu une virgule est rencontrée.

```
10 PRINT "INSTRUCTION 10 ";
20 PRINT "INSTRUCTION 20"

RUN

INSTRUCTION 10 INSTRUCTION 20
```

A la fin d'une instruction PRINT, un point-virgule empêche le curseur de se positionner au début de la ligne suivante; par conséquent, l'ordre PRINT suivant viendra écrire sur la même ligne, à la suite de ce qui est déjà écrit.

L'absence de signe à la fin d'un PRINT entraîne un retour du curseur au début de la ligne suivante.

PRINT @ *position, liste de valeurs* [en français : IMPRIMER EN]

Spécifie l'adresse de l'endroit, sur l'écran, où l'impression doit commencer (il s'agit de l'équivalent de AT en NIVEAU I). Le paramètre @ doit suivre directement PRINT; la position est repérée par une expression correspondant à un nombre de 0 à 1023. Référez-vous au schéma de l'écran de l'annexe E qui vous indiquera la position de chaque caractère sur l'écran.

```
100 PRINT @ 550, "POSITION 550"
```

Exécutez cette instruction, elle vous indiquera l'emplacement de la position 550.

Chaque fois que vous exécutez un PRINT, il y a un retour du curseur au début de la ligne suivante; lorsque vous imprimez sur la dernière ligne vous provoquez une remontée d'une ligne du texte imprimé. Pour l'éviter, utilisez un point-virgule final.

Exemple :

```
100 PRINT @ 1000, 1000;
```

PRINT TAB (*expression*)

Avance le curseur à la position indiquée par *expression* dans la ligne courante (ou dans les lignes suivantes si vous spécifiez des positions dépassant 63). TAB peut être utilisé plusieurs fois dans un ordre PRINT. La valeur désignée par *expression* doit aller de 0 à 255 inclus.

Exemples :

```
10 PRINT TAB(5) "POS 5"; TAB(25) "POS 25"
```

Il n'y a pas de signe de ponctuation après un paramètre TAB.

```
5  X=3
10 PRINT TAB(X) X; TAB(X ↑ 2) X ↑ 2; TAB(X ↑ 3) X ↑ 3
```

Des expressions numériques quelconques peuvent être utilisées pour désigner les positions. Ce paramètre peut ainsi être très utile pour dessiner les courbes de fonctions mathématiques, des tableaux, etc. Il n'est cependant pas possible de faire se mouvoir le curseur vers la gauche de la position courante. Toute tentative en ce sens sera simplement ignorée.

PRINT USING *chaîne ; liste de valeurs*

[en français : IMPRIMER SELON LE FORMAT]

Cette instruction vous permet d'imprimer des valeurs numériques et alphanumériques selon un format décrit par la chaîne de caractères *chaîne*. Elle sera utilisée dans de nombreuses applications telles que l'impression d'entêtes d'états imprimés ou de rapports comptables où un format d'impression spécifique est exigé.

La description du format *chaîne* peut être spécifiée sous forme d'une constante ou d'une variable. Le résultat obtenu est l'impression de *chaîne* après insertion des valeurs de la liste dans les spécificateurs de zones de cette chaîne.

Les spécificateurs de zones suivants peuvent être utilisés dans la *chaîne :*

\# Ce signe indique la position d'un chiffre d'une valeur numérique; une suite de # constitue une zone numérique. Si cette zone est plus grande qu'il n'est nécessaire pour une valeur, des espaces remplaceront les chiffres absent à gauche. Dans une zone numérique peut être inséré un point qui définit la position du point décimal. Les positions inutilisées à droite du point décimal sont garnies par des zéros. Le nombre de # à droite du point décimal définit également un arrondi s'il ne permet pas l'impression de toutes les décimales du nombre.

Si une virgule est placée entre le premier # et le point décimal, elle définit la position d'un chiffre, tout comme un #, mais en outre elle entraîne l'impression d'une virgule entre les groups de trois chiffres à gauche du point décimal.

** Deux astérisques au début d'un spécificateur de zone numérique indiquent la position de deux chiffres; en outre, ils entraîneront l'impression d'astérisques dans les positions inutilisées à gauche du premier chiffre significatif.

$$ Deux signes $ au début d'un spécificateur de zone numérique entraîneront l'impression d'un signe $ à la position qui précède le premier chiffre significatif. Ils indiquent la position d'un chiffre.

**$ Au début d'un spécificateur de zone numérique, ils combinent les effets de ** et $$; un $ est imprimé devant le premier chiffre, et si des positions sont laissées libres, elles sont remplies par des astériques. Ces trois signes indiquent la position de deux chiffres.

+ Placé au début ou à la fin d'un spécificateur de zone numérique, il commande l'impression devant ou derrière les chiffres d'un signe + si la valeur est positive, et – si la valeur est négative.

– Placé à la fin d'un spécificateur de zone numérique, il commande d'impression derrière les chiffres d'un signe – si la valeur est négative, et d'un espace si la valeur est positive.

% *espaces* % Spécificateur de zone alphanumérique. La longueur de cette zone est de 2 plus le nombre d'espaces entre les deux signes %.

! Spécificateur de zone alphanumérique. La longueur de cette zone est de 1 caractère. Ces deux derniers spécificateurs définissent des zones qui contiendront les caractères les plus à gauche de la valeur à imprimer.

Le programme suivant illustre l'emploi de ces spécificateurs :

```
10 INPUT A$, A
20 PRINT USING A$; A
30 GOTO 10
```

Exécutez ce programme et donnez-lui différentes valeurs de A$ et A.

Par exemple :

```
RUN
? ##.##, 12.12
12.12
? ###.##, 12.12
  12.12
? ##.##, 121.12
% 121.12
```

Le signe % est imprimé automatiquement si la zone n'est pas de taille suffisante pour contenir tous les chiffres de la valeur numérique. Tous les chiffres précédant le point décimal sont imprimés, ainsi que le signe – éventuel.

```
? ##.##, 12.127
12.13
```

Remarquez que le nombre a été arrondi à deux décimales.

```
? +##.##, 12.12
+12.12
? +##.##, –12.12
–12.12
? ##.##+, 12.12
12.12+
? ##.##+, –12.12
12.12–
? ##.##–, 12.12
12.12
? ##.##–, –12.12
12.12–
? **##, 12.12
**12
? **##.##, 1212.12
1212.12
? $$##.##, 12.12
  $12.12
? "##,###.#", 12121.2
  12,121.2
? "####,#", 12121.2
  12,121
? ###, 1212
% 1212
```

Essayons à présent d'utiliser le PRINT USING pour imprimer des chaînes de caractères, en utilisant les spécificateurs ! et % espaces %.

Exemples :

```
PRINT USING "!"; "ABCD"
PRINT USING "%   %"; A$
```

La première forme commande l'impression du premier caractère de la chaîne "ABCD" tandis que la seconde fera imprimer les 4 premiers caractères de la valeur de A$. Le spécificateur peut ici encore être le contenu d'une variable comme dans le programme qui suit :

```
10 INPUT A$, B$
20 PRINT USING A$; B$
30 GOTO 10
```

Exécutez-le :

```
? !, ABCDE
A
?%%, ABCDE
AB
?%   %, ABCDE
ABC
```

S'il y a plus de valeurs à imprimer que de zones de format, les spécificateurs sont réutilisés jusqu'à ce que toutes les valeurs aient été imprimées.
Par exemple :

```
10 INPUT A$, B$, C$
20 PRINT USING "!"; A$; B$; C$
```

Donnerait lors de son exécution :

```
? ABC, DEF, GHI
ADG
```

Ou encore, en changeant le spécificateur "!" en "! " :

```
? ABC, DEF, GHI
A D G
```

Le même résultat aurait été obtenu en utilisant le spécificateur "! ! !".
Essayez ce programme avec le spécificateur "%%".

Le programme suivant illustre un usage possible du PRINT USING.

```
 10 CLS
 20 A$="**$##,######.## DOLLARS"
 30 INPUT "PRENOM"; P$
 40 INPUT "DEUXIEME PRENOM"; D$
 50 INPUT "NOM"; N$
 60 INPUT "MONTANT A PAYER"; P
 70 CLS: PRINT "PAYER A L'ORDRE DE";
 80 PRINT USING "!! !!"; P$; "."; D$; ".";
 90 PRINT N$
100 PRINT: PRINT USING A$; P
110 GOTO 10
```

Si vous voulez être économe, utilisez ? au lieu de PRINT. Un exemple d'exécution de ce programme serait le suivant :

```
PRENOM ? JEAN
DEUXIEME PRENOM ? PAUL
NOM ? LEMERCIER
MONTANT A PAYER ? 12345.6

PAYER A L'ORDRE DE J.P. LEMERCIER

*****$12,345.60 DOLLARS
```

Si vous désirez utiliser des nombres plus grands que 999999 sans arrondi et sans affichage en notation scientifique, faites suivre le nom P du signe # aux lignes 60 et 100. La variable P sera alors en double précision, ce qui vous permettra d'utiliser des nombres réclamant jusqu'à 16 décimales.

INPUT *liste de variables*

[en français : INTRODUIRE]

L'ordinateur s'arrête jusqu'à ce que vous ayez introduit autant de valeurs au clavier qu'il y a de variables dans la liste. Ces variables peuvent être de différents types numériques ou alphanumérique. Les valeurs introduites doivent être séparées par des virgules.

```
100 INPUT X$, X1, Z$, Z1
```

Cette instruction vous demande d'introduire dans l'ordre, une chaîne de caractères suivie d'un nombre, suivi d'une autre chaîne, puis enfin un nombre. Ces valeurs seront séparées par des virgules. Si une chaîne de caractères commence par des espaces, ou contient des virgules ou des "deux-points", vous devez l'inclure entre des guillemets.
Lorsqu'il rencontre cette instruction, l'ordinateur affiche :

```
?_
```

Vous pouvez alors introduire les données en une fois ou une à une. Dans le premier cas, vous pourriez répondre :

JEAN, 50, JACQUES, 40 puis **ENTER**

L'ordinateur affectera ces valeurs comme suit :

X$="JEAN" X1=50 Z$="JACQUES" Z=40

Si par contre vous poussez sur **ENTER** avant d'avoir introduit toutes les données, l'ordinateur affichera :

```
??_
```

pour indiquer qu'il attend des données supplémentaires. Continuez d'introduire des données jusqu'à ce que toutes les variables de la liste soient garnies. A ce moment l'ordinateur aura terminé l'exécution de l'INPUT et passera à l'instruction suivante.

Assurez-vous que la donnée que vous introduisez corresponde bien au type de la variable à laquelle vous l'affectez. N'essayez pas par exemple de fournir une chaîne de caractères à une variable numérique; le cas échéant, l'ordinateur affichera :

```
?REDO                    (c'est-à-dire "REFAITES")
?_
```

et vous donnera une nouvelle occasion d'introduire une donnée correcte. Attention cependant, vous devez alors réintroduire toutes les valeurs correspondant à la liste de variables.

Remarque : Vous ne pouvez pas introduire une expression (autre qu'une constante) comme valeur d'une variable numérique (LE NIVEAU I vous permettait de spécifier une expression et même un nom de variable).

Exemple :

```
100 INPUT X1, Y1 $
200 PRINT X1, Y1 $

RUN

?_ [Vous tapez :] 7+3 ENTER
? REDO

?_ [Vous tapez :] 10 ENTER
??_ [Vous tapez :] "VOICI DEUX-POINTS :,"

10                VOICI DEUX-POINTS :,
```

La présence de la virgule nous oblige ici à introduire la chaîne de caractères entre guillements.

Si vous introduisez plus de données qu'il n'y a de variables, l'ordinateur affichera :

? EXTRA IGNORED (c'est-à-dire : les données superflues ont été ignorées)

puis continuera l'exécution normale du programme.

Vous pouvez également faire précéder l'attente des valeurs par l'impression d'un message précisant les renseignements à fournir. Ce message doit être inséré entre INPUT et la liste de variables, et séparé de cette dernière par un point-virgule.

Exemple :

```
100 INPUT "TAPEZ NOM ET AGE"; N $, A

RUN

TAPEZ NOM ET AGE? _
```

DATA *liste de valeurs*

[en français : DONNEES]

Cette instruction vous permet de stocker dans votre programme des données qui seront lues par des ordres READ. Ces données seront lues séquentiellement, depuis la première donnée de la première instruction DATA jusqu'à la dernière donnée de la dernière instruction DATA. Les valeurs seront spécifiées sous forme de constantes numériques et alphanumériques mais pas sous forme d'expressions plus complexes. Les chaînes de caractères commençant par des espaces ou contenant des virgules ou "deux points" devront être incluses entre guillemets.
Il est essentiel que les données qui sont lues correspondent, quant à leur type, aux variables des ordres READ.

Les instructions DATA peuvent apparaître n'importe où dans le programme.

On peut par exemple les grouper consécutivement à la fin ou au début du programme; on peut aussi les placer à proximité des ordres READ qui les concernent.

Exemples :

```
500    READ N1$,N2$,N1,N2
1000   DATA "DUPONT, E.F.", "CODD, J.P."
2000   DATA 150,175
```

Voir aussi **READ, RESTORE.**

READ *liste de variables*

[en français : LIRE]

Commande à l'ordinateur de lire autant de valeurs qu'il y a d'éléments dans la liste de variables et de les affecter, dans cet ordre, à ces éléments. La première instruction READ exécutée lit à partir du début de la première ligne DATA; l'instruction READ exécutée ensuite lira les données à partir de celle qui suit la dernière qui a été lue, qu'elle se trouve sur la même ligne DATA ou sur la suivante. Cette lecture ne dépend pas du groupement des données selon les lignes DATA mais uniquement de l'ordre dans lequel on les rencontre en lisant les lignes DATA de gauche à droite et de la première à la dernière. Si vous tentez de lire plus de données qu'il n'y en a dans les lignes DATA, vous provoquerez une erreur du type OD. Le programme qui suit correspond à une application classique des instructions READ et DATA.

```
50     PRINT "NOM", "AGE"
100    READ N$
110    IF N$="FIN" PRINT "FIN DE LISTE":END
120    READ AGE
130    IF AGE < 18 PRINT N$, AGE
140    GOTO 100
200    DATA "THOMAS, B", 32, "MERCIER", 15
210    DATA "DELOBEL, C", 19, "BRETECHER", 21
220    DATA "COTY, A.M.", 17, FIN
RUN
NOM                AGE
MERCIER            15
COTY, A.M.         17
FIN DE LISTE

READY
>_
```

Ce programme imprime le nom et l'âge des personnes mineures dont la description est reprise dans des lignes DATA.

Vous remarquerez l'emploi de la valeur "FIN" qui permet de travailler sur des listes de données de longueur quelconque.

Voir aussi **DATA, RESTORE.**

RESTORE

[en français : REPOSITIONNER]

Permet au prochaine ordre READ exécuté de recommencer la lecture au début de la première ligne DATA. Cette instruction vous permet de réutiliser les mêmes données.

Exemple :

```
100     READ X
110     RESTORE
120     READ Y
130     PRINT X, Y
140     DATA 50,60
RUN
 50              50
READY
>_
```

A cause de l'instruction RESTORE, le second READ a recommencé la lecture au début de la ligne DATA.

Voir aussi **READ, DATA.**

PRINT #-1, *liste de valeurs*

Ecrit les valeurs spécifiées dans la liste sur la cassette de l'enregistreur n° 1 (celui qui est directement connecté au bloc clavier). Si vous disposez de l'interface d'expansion (voir chapitre 10), vous pouvez y connecter deux enregistreurs; le premier sera commandé par PRINT #−1 et le second par PRINT #−2; le numéro de l'enregistreur peut aussi se trouver dans une variable : PRINT #−X est valide à condition que la valeur de X soit 1 ou 2. En utilisation normale, vous n'aurez probablement besoin que de l'enregistreur n° 1.

Exemple :

```
5       A1=−30.334:B$="MESURE TOTALE"
10      PRINT#−1, A1, B$, "FIN"
```

Ce programme écrit les valeurs de A1 et de B$, suivies de la chaîne "FIN" sur la cassette. Il sera possible, plus tard, de relire ces données sur la bande à l'aide d'une instruction INPUT # spécifiant des variables correspondant en nombre et en type aux valeurs lues.

Voir aussi **INPUT #.**

Remarques :

La longueur totale des valeurs de la liste ne peut excéder 248 caractères. Les caractères situés au-delà de cette limite ne sont pas enregistrés.

Par exemple,

```
PRINT#−1, A#, B#, C#, D#, E#, F#, G#, H#, I#, J#, A$
```

risque d'écrire des données trop longues si A$ est supérieur à quelque 65 caractères. Dans ce cas, A$ ne serait pas enregistrée et un INPUT #−1 ultérieur risque de provoquer une erreur OD (Trop peu de données). Lors de l'écriture, les nombres sont convertis en chaînes de caractères dont la longueur est fonction du nombre de chiffres significatifs de ces nombres, les différentes valeurs écrites sont séparées par des virgules. Les valeurs écrites par un ordre PRINT # constituent un enregistrement ou un bloc sur la cassette.

INPUT #-1, *liste de variables*

Lit sur la cassette de l'enregistreur n° 1 des données et les range dans les les variables (ou éléments de tableau) de la liste. De même que pour l'instruction PRINT #, il est possible de lire sur le deuxième enregistreur grâce à INPUT #–2. La forme INPUT #–X est aussi acceptée.

Exemple :

```
50      INPUT#–1,X,P$,T$
```

Lorsque cette instruction est exécutée, l'ordinateur fait démarrer l'enregistreur, lit les valeurs nécessaires et les range dans les variables puis arrête l'enregistreur avant de passer à l'instruction suivante. Si une chaîne de caractères non numériques est lue alors que la liste indique une variable numérique, une erreur de fichier invalide se produira (erreur FD). S'il n'y a pas assez de données dans la liste sur cassette pour garnir touts les variables, une erreur OD se produira.

Un INPUT # ne lira qu'un bloc sur la cassette. Assurez-vous donc que la liste des variables de l'INPUT # est de même longueur et de mêmes types (dans le même ordre) que la liste des valeurs du PRINT # qui a créé le bloc.

Exemple de programme

Utilisez le programme donné dans la description du PRINT # pour créer un petit fichier de données. Rebobinez la cassette, règlez l'enregistreur pour la reproduction puis exécutez le programme suivant :

```
10      INPUT#–1,A1,B$,L$
20      PRINT A1,B$,L$
30      IF L$="FIN" END
40      GOTO 10
```

Ce programme est capable de lire un fichier de longueur quelconque pourvu que les blocs contiennent trois valeurs et que la dernière valeur du dernier bloc soit "FIN".

4 / Les instructions de contrôle

LE BASIC NIVEAU II admet a priori un certain nombre d'options sur les modalités d'exécution d'un programme. Par exemple :

* Les variables utilisées sont en simple précision (sauf lors de l'utilisation de caractères de déclaration – voir chapitre 1, "Types de variables").

* Une quantité fixe de mémoire est allouée pour les chaînes de caractères et chaque tableau, que vous les utilisiez ou non.

* L'exécution est séquentielle, à partir de la première instruction jusqu'à la dernière.

Les instructions décrites dans ce chapitre vous permettent de modifier ces options par défaut afin de donner à vos programmes plus de souplesse et de puissance.

Remarque : à l'exception de INPUT et INPUT#, toutes les instructions du NIVEAU II peuvent être utilisées aussi bien dans un programme qu'en mode Commande.

Dans ce chapitre seront examinées les instructions suivantes :

Définition de type	**Allocation et affectation**	**Contrôle de séquence**	**Instructions conditionnelles**
DEFINT	CLEAR *n*	END	IF
DEFSNG	DIM	STOP	THEN
DEFDBL	LET	GOTO	ELSE
DEFSTR		GOSUB	
		ON ... GOTO	
		ON ... GOSUB	
		FOR-NEXT-STEP	
		ERROR	
		ON ERROR GOTO	
		RESUME	
		REM	

On y trouvera également une discussion des conversions numériques en NIVEAU II; vous pourrez ainsi déterminer et commander la manière dont les constantes et le résultat d'expressions seront stockés – comme entier, simple ou double précision.

DEFINT *intervalle de lettres* [en français : DEFINIR COMME ENTIERES]

Les variables dont le nom commence par une lettre spécifiée dans l'intervalle seront considérées comme entières sauf si un caractère de déclaration en modifie le type. Les valeurs entières occupent en mémoire moins de place que les autres valeurs; les calculs sur des valeurs entières sont aussi plus rapides que ceux portant sur des valeurs en simple et double précision. Une variable entière peut contenir des valeurs comprises entre – 32768 et + 32767 (inclus).

Exemples :

```
10      DEFINT A,I,N
```

Après l'exécution de cette ligne, toutes les variables dont le nom commence par A, I ou N seront des variables entières. Par exemple, A1, AA, I3 et NN seront des variables entières, alors que A1#, I3# et N9# seront en double précision en raison du caractère de déclaration, qui est prioritaire sur l'instruction de déclaration.

```
10      DEFINT I-N,X
```

Toutes les variables dont le nom commence par I, J, K, L, M, N et X seront des variables entières.

DEFINT peut être placée n'importe où dans le programme; cependant, comme elle peut changer le type des variables sans caractère de déclaration, il est prudent de l'écrire au début du programme. **Attention** : l'effet de DEFINT est annulé par l'exécution de CLEAR.

Vois aussi **DEFSNG, DEFDBL,** et chapitre 1, "Types de variables".

DEFSNG *intervalle de lettres* [en français : DEFINIR EN SIMPLE PRECISION]

Les variables dont le nom commence par une lettre spécifiée dans l'intervalle seront considérées comme étant en simple précision sauf si un caractère de déclaration en modifie le type. Les variables et les constantes en simple précision sont stockées avec 7 chiffres significatifs et imprimées avec 6 chiffres significatifs. En l'absence de toute définition (DEF ou caractère de définition), une variable est par défaut en simple précision; par conséquent, cette instruction servira surtout à redéfinir des variables qui avaient été définies comme étant entières ou en double précision.

Exemple :

```
100     DEFSNG I, W-Z
```

Après l'exécution de cette ligne, toutes les variables dont le nom commence par I, W, X, Y et Z seront des variables en simples précision. Par contre, I% sera une variable entière et Z# une variable en double précision grâce à l'usage du caractère de déclaration.

Voir aussi **DEFINT, DEFDBL** et chapitre 1, "Types de variables".

DEFDBL *intervalle de lettres* [en français : DEFINIR EN DOUBLE PRECISION]

Les variables dont le nom commence par une lettre spécifiée dans l'intervalle seront considérées comme étant en double précision, sauf si un caractère de déclaration en modifie le type. Il y correspond une précision de 17 chiffres dans les calculs et le stockage, et 16 chiffres lors de l'impression.

Exemple :

```
10      DEFDBL S-Z, A-E
```

Les variables ayant un nom dont la première lettre est comprise entre A et E ou S et Z seront considérées comme étant en double précision.

Cette instruction sera généralement placée en tête du programme de manière à modifier le type des variables avant leur utilisation. **Attention** : l'effet de DEFDBL est annulé par l'exécution de CLEAR.

Voir aussi **DEFINT, DEFSNG** et chapitre 1, "Types de variables".

DEFSTR *intervalle de lettres*

[en français : DEFINIR COMME CHAINES]

Les variables ayant un nom dont la première lettre est spécifiée dans l'intervalle seront considérées comme étant des variables alphanumériques sauf si un caractère de définition vient en modifier le type. Si une quantité suffisante de bytes à été allouée par un CLEAR*n*, chaque variable pourra contenir des chaînes de 0 à 255 caractères.

Exemple :

```
10      DEFSTR L-Z
```

Les variables dont le nom commence par une lettre de L à Z seront des variables alphanumériques (sauf présence d'un caractère de définition). Il sera alors possible d'écrire : L1 = "ATHENES". **Attention** : l'effet de DEFSTR est annulé par l'exécution de CLEAR.

Voir aussi **CLEAR** *n*, chapitre 1 "Types de variables" et chapitre 5.

CLEAR *n*

[en français : REINITIALISER]

Utilisée sans argument, cette instruction a pour effet de réinitialiser toutes les variables numériques à zéro et les variables alphanumériques à "vide". Lorsqu'elle est utilisée avec un argument numérique (constante ou expression), elle réalise, outre la fonction ci-dessus, la réservation d'autant de bytes qu'il est spécifié dans l'argument destinés à contenir des chaînes de caractères. A l'allumage du TRS-80, il y a réservation automatique de 50 bytes pour les chaînes.

L'espace de mémoire réservé pour les chaînes par un CLEAR doit être au moins égal au nombre maximum de caractères pouvant exister à tout instant dans les variables alphanumériques lors de l'exécution. Si cet espace vient à manquer, une erreur OS se produira (OUT OF STRING SPACE = plus d'espace pour les chaînes)

Exemple :

```
10      CLEAR 1000
```

Réinitialise toutes les variables et réserve 1000 bytes pour les chaînes.

Essayez de calculer exactement la quantité strictement nécessaire de manière à user de votre mémoire de façon économe. Par exemple, si votre programme n'utilise pas de chaîne de caractères, commencez votre programme par un CLEAR 0. L'argument de CLEAR doit être positif ou nul.

Voir **ON ERROR** et instructions **DEF.**

DIM *nom* (*dim1*, *dim2*, ..., *dimK*:

[en français : DIMENSIONS]

Cette instruction vous permet de déclarer un tableau et d'en préciser la taille. Le tableau a autant de dimensions qu'il y a d'expressions *dim ;* les valeurs de *dim1*, *dim2*, ..., *dimK* indiquent le nombre d'éléments dans la première, la deuxième, ..., la Kème dimension. En fait, il y a dans chaque dimension un élément de rang zéro; le nombre d'éléments dans la dimension I est donc *dimI* + 1. En l'absence de déclaration, tout tableau est supposé avoir un nombre quelconque de dimensions, et 11 éléments dans chacune d'elles (éléments de 0 à 10).

Exemple :

```
10    DIM A(5),B(2,3),C$(20)
```

Cette instruction déclare un tableau A à une dimension et contenant 6 éléments de rang 0 à 5, puis un tableau B à deux dimensions et contenant 12 éléments de rang (0,0), (0,1), ..., (2,3) et enfin un tableau de chaînes à une dimension et contenant 21 éléments. S'il n'y a pas eu de déclaration de type préalable, A et B contiennent des éléments numériques en simple précision.

L'instruction DIM peut être située n'importe où dans le programme; le paramètre *dim* doit être une constante ou une expression positive ou nulle.

Exemple :

```
40    INPUT "NOMBRE DE NOMS"; N
50    DIM NA(N,2)
```

Si vous désirez donner de nouvelles dimensions à un tableau vous devez d'abord exécuter une instruction CLEAR avec ou sans argument; si vous ne le faites pas, vous obtiendrez une erreur DD.

Exemple de programme :

```
10    AA(4)=11.5
20    DIM AA(7)

RUN

?DD   ERROR IN 20     (c'est-à-dire ERREUR EN 20)
```

L'utilisation de AA à la ligne 10 déclarait implicitement AA(10), comme le sont tous les tableaux non déclarés explicitement; la ligne 20 redéclare AA à nouveau, ce qui est refusé lorsqu'un CLEAR n'a pas été exécuté.

Voir chapitre 6, "Les tableaux".

LET *variable = expression*

[en français : SOIT *variable = expression*]

Cette instruction réalise l'affectation d'une valeur à une variable ou à un élément de tableau. En BASIC NIVEAU II Radio Shack, le mot LET est facultatif dans cette instruction; vous pouvez cependant l'utiliser de manière à rendre vos programmes compatibles avec des versions BASIC qui l'exigent.

Exemples :

```
100   LET A$="UNE ROSE EST UNE ROSE"
110   LET B1=1.23
120   LET X=X-Z1
```

Dans chaque instruction, la variable à gauche du signe d'affectation (=) reçoit la valeur de l'expression (et en particulier de la constante) se trouvant à droite de ce signe.

END

[en français : FIN]

Cette instruction termine normalement l'exécution du programme (sans le message BREAK). Certaines versions du BASIC exigent que END soit la dernière instruction du programme, ce qui est par contre facultatif en NIVEAU II. END sera surtout utilisé pour arrêter l'exécution d'un programme à un autre endroit que la fin physique du programme.

Exemple :

```
10      INPUT S1,S2
20      GOSUB 100
.
.
.
99      END
100     H=SQR(S1*S1+S2*S2)
110     RETURN
```

Sous la ligne 99 qui arrête définitivement l'exécution, celle-ci se poursuivrait en exécutant les lignes 100 et 110, provoquant ainsi une erreur. Les lignes 100 et 110 ne sont accessibles que par un GOSUB 100.

STOP

Cette instruction provoque une suspension de l'exécution et l'impression du message "BREAK IN *numéro de ligne*" (c'est-à-dire : INTERRUPTION EN *numéro de ligne*). Pendant cette interruption, vous pouvez consulter et modifier le contenu des variables. La commande CONT peut alors être utilisée pour faire reprendre l'exécution à partir du point d'arrêt (Attention, ceci n'est valable que si vous n'avez pas modifié le programme). STOP est avant tout un outil de mise au point des programmes.

Exemple :

```
10      X=RND(10)
15      STOP
20      GOSUB 1000

RUN

BREAK IN 15
READY
>_
```

Vous pouvez alors demander la valeur de X par un PRINT X (ou ?X) avant de permettre à l'exécution de se poursuivre. Lorsque le programme vous semblera correct, vous supprimerez la ligne 15.

GOTO *numéro de ligne*

[en français : ALLER EN]

Provoque un transfert de l'exécution non pas à l'instruction suivante, mais bien à celle(s) de la ligne spécifiée. Utilisée telle quelle, elle constitue un "branchement inconditionnel"; comme branche d'une alternative IF, elle constitue un "branchement conditionnel".

Exemple :

```
200     GOTO 10
210     PRINT X,Y
```

L'exécution de la ligne 200 oblige l'exécution à se poursuivre à partir de la ligne 10 et non à la ligne 210.

Vous pouvez aussi utiliser un GOTO en mode commande au lieu de RUN. Vous pouvez ainsi commencer l'exécution à la ligne spécifiée **sans que le CLEAR automatique soit exécuté au préalable,** ce qui serait le cas de RUN; cela vous permet de conserver le contenu précédent des variables (introduit en mode Commande par exemple).

Voir **IF, THEN, ELSE, ON... GOTO.**

GOSUB *numéro de ligne*

[en français : ALLER A LA SOUS-ROUTINE]

Provoque un transfert de l'exécution à la sous-routine qui commence à la ligne spécifiée. Par la suite, lorsque l'ordinateur rencontrera une instruction RETURN, un retour de l'exécution s'opérera à l'instruction qui suit le GOSUB. Un GOSUB peut être, tout comme le GOTO, inclus dans une branche de l'alternative IF.

Exemple :

```
100     GOSUB 200
110     PRINT "RETOUR DE LA SOUS-ROUTINE" : END
200     PRINT "EXECUTION DE LA SOUS-ROUTINE"
210     RETURN

RUN

EXECUTION DE LA SOUS-ROUTINE
RETOUR DE LA SOUS-ROUTINE
```

La ligne 100 provoque le branchement vers la sous-routine constituée des lignes 200 à 210. La dernière instruction de la sous-routine provoque un retour à l'instruction qui suit le GOSUB ayant provoqué le branchement, soit ici un END.

Il est plus naturel de voir dans le GOSUB non pas un branchement mais un ordre d'exécution d'une sous-fonction (FAIRE *numéro de sous-routine*), qui est définie par une suite d'instructions dont l'exécution se termine par un retour à l'instruction appelante (RETURN).

Voir **IF, THEN, ELSE, ON... GOSUB.**

RETURN

[en français : RETOUR]

Termine l'exécution d'une sous-routine et transfère l'exécution à l'instruction qui suit le dernier GOSUB exécuté. Si un RETURN est exécuté sans qu'il y corresponde un GOSUB précédent, une erreur RG se produira.

Voir **GOSUB.**

ON *expression* GOTO *n° ligne, ..., n° ligne*

[en français : SELON *expression* ALLER EN *n° ligne, ..., n° ligne*]

Il s'agit d'une instruction de branchement à plusieurs voies commandée par la valeur d'une expression (et en particulier une simple variable).

Lors de l'exécution de cette instruction, l'expression est évaluée puis tronquée à sa partie entière (INT (*expression*)). Soit x la valeur obtenue.

L'ordinateur extrait alors le $x^{ème}$ numéro de ligne spécifié dans l'instruction, puis exécute un branchement à la ligne portant ce numéro.

Si x est égal à 0 ou supérieur au nombre de numéros de lignes, l'instruction est sans effet et l'exécution se poursuit à l'instruction suivante.

Si x est négatif ou supérieur à 255, une erreur FC (paramètre invalide) se produit.

Il n'y a pas de limite (autre que la longueur de la ligne) au nombre de numéros de ligne.

Exemple :

```
100     ON M GOTO 150, 160, 170, 150, 180
110     ...
```

est exécutée comme suit :

"Evaluer la partie entière de M;
si elle est égale à 1, aller en 150
si elle est égale à 2, aller en 160
si elle est égale à 3, aller en 170
si elle est égale à 4, aller en 150
si elle est égale à 5, aller en 180
si elle est égale à 0 ou supérieure à 5, aller en 110
si elle est négative ou supérieure à 255; erreur FC."

Exemple de programme :

```
100     INPUT "TAPEZ UN NOMBRE"; X
110     ON SGN(X)+2 GOTO 120, 130, 140
120     PRINT "NEGATIF" : END
130     PRINT "NUL" : END
140     PRINT "POSITIF" : END
```

La fonction SGN fournit la valeur –1, 0, +1 selon que son argument est négatif, nul ou positif. En ajoutant 2 à cette valeur, on obtient le résultat 1, 2 ou 3 selon que l'argument est négatif, nul ou positif. Le ON... GOTO provoque alors le branchement approprié.

ON *expression* **GOSUB** *n° ligne, ..., n° ligne*

[en français : SELON *expression* ALLER A LA SOUS-ROUTINE *n° ligne, ..., n° ligne*]

Cette instruction agit comme ON... GOTO, hormis le fait qu'il n'y a pas branchement, mais exécution d'une sous-routine spécifiée par son numéro de ligne.

Exemple :

```
100     INPUT "CHOISISSEZ-VOUS 1, 2 ou 3"; I
105     ON I GOSUB 200, 300, 400
110     END
200     PRINT "CHOIX #1" : RETURN
300     PRINT "CHOIX #2" : RETURN
400     PRINT "CHOIX #3" : RETURN
```

Les contraintes portant sur la valeur et la forme de l'*expression* sont les mêmes que pour ON... GOTO.

Voir **ON... GOTO, GOSUB.**

FOR *nom=expr* **TO** *expr* **STEP** *expr*

[en français : POUR *nom* ALLANT DE *expr* A *expr* PAR PAS DE *expr*]

NEXT *nom*

[en français : *nom* SUIVANT]

Ces instructions définissent une boucle itérative (ou répétitive) telle que la séquence d'instructions qu'elles encadrent soit exécutée un certain nombre de fois. D'une manière plus précise, la boucle se présente sous la forme suivante (les éléments facultatifs sont entre crochets) :

n° ligne FOR *variable-compteur* = *valeur initiale* TO *valeur finale* [STEP *incrément*]
[*séquence d'instructions*]
n° ligne NEXT [*variable-compteur*]

Dans l'instruction FOR, *valeur initiale, valeur finale* et *incrément* peuvent être des constantes, des variables ou des expressions en général.
Lors de l'exécution de l'instruction FOR, ces trois grandeurs sont évaluées et leurs valeurs sont sauvées; il ne sera donc plus possible de les modifier à l'intérieur de la boucle. Lors des itérations successives de la boucle, la valeur de la variable-compteur sera modifiée automatiquement; la plus grande prudence s'impose donc si l'on désire la modifier soi-même à l'intérieur de la boucle.

La boucle FOR-STEP-NEXT fonctionne de la manière suivante : avant la première itération de la boucle, la variable-compteur reçoit la *valeur initiale ;* la *séquence d'instructions* est alors exécutée jusqu'à ce qu'une instruction NEXT soit rencontrée. L'*incrément* est alors ajouté à la valeur de la variable-compteur (Si cet incrément est négatif, cette valeur diminue donc). Si STEP *incrément* est omis, il est supposé être égal à 1.

La valeur de la variable-compteur est alors comparée à la *valeur-finale ;* si la valeur de la variable est **supérieure,** l'exécution de la boucle est terminée; sinon elle se poursuit, pour l'itération suivante, à la première instruction de la *séquence d'instructions.* Si l'incrément est négatif, l'exécution de la boucle se termine lorsque la valeur de la variable-compteur est **inférieure** à la *valeur-finale.*
Lorsqu'une boucle a été exécutée, l'exécution du programme se poursuit à la première instruction qui suit le NEXT,

Exemples de programmes :

```
10      FOR I=10 TO 1 STEP-1
20      PRINT I;
30      NEXT
40      PRINT "FIN"

RUN

10  9  8  7  6  5  4  3  2  1  FIN

READY
>_
```

```
10      FOR K=0 TO 1 STEP .3
20      PRINT K;
30      NEXT

RUN

  0   .3   .6   .9
READY
>_
```

A la fin de la quatrième itération, K a la valeur .9, puis après addition de l'*incrément* .3, K prend la valeur 1.2. Cette valeur étant plus grande que la *valeur finale* 1 (et l'incrément étant positif), la boucle termine son exécution sans imprimer cette valeur finale.

```
10   FOR K=4 TO 0
20   PRINT K;
30   NEXT

RUN
  4
READY
>_
```

STEP n'étant pas précisé, l'incrément a pour valeur 1. Après la première itération, qui a imprimé la valeur 4 de K, cette variable prend la valeur 5. La valeur finale 0 étant dépassée; l'exécution s'arrête. On remarquera donc **qu'une boucle est toujours exécutée au moins une fois.**

```
10   J=3 : K=8 : L=2
20   FOR I=J TO K+1 STEP L
25   J=0 : K=0 : L=0
30   PRINT I;
40   NEXT

RUN

  3  5  7  9
READY
>_
```

Les variables et expressions de la ligne 20 sont évaluées et les valeurs sont sauvées, devenant ainsi des constantes pour la boucle. Une modification des valeurs de J, K et L à la ligne 25 ne modifie pas le comportement de la boucle.

Des boucles FOR-NEXT peuvent être "emboitées" :

```
10   FOR I=1 TO 3
20   PRINT "BOUCLE EXTERIEURE"
30   FOR J=1 TO 2
40   PRINT "BOUCLE INTERIEURE"
50   NEXT J
60   NEXT I
```

```
RUN

BOUCLE EXTERIEURE
    BOUCLE INTERIEURE
    BOUCLE INTERIEURE
BOUCLE EXTERIEURE
    BOUCLE INTERIEURE
    BOUCLE INTERIEURE
BOUCLE EXTERIEURE
    BOUCLE INTERIEURE
    BOUCLE INTERIEURE
```

Vous remarquerez que les instructions NEXT spécifient le nom de la variable-compteur de la boucle; ceci n'a d'autre utilité que d'aider le programmeur à construire ses boucles correctement. Vous pouvez omettre ces noms dans les instructions NEXT, mais si vous les utilisez, vous **devez** les spécifier correctement : le NEXT de la boucle la plus intérieure non encore terminée doit apparaître avant les autres.

Cette spécification est aussi utile lorsque vous sortez de la boucle sans passer par son NEXT (par exemple par un GOTO).

Vous pouvez remplacer plusieurs NEXT successifs par un seul NEXT dans lequel vous spécifiez la liste des noms de variables des NEXT que vous remplacez.

Par exemple, supprimez la ligne 50 ci-dessus et remplacez la ligne 60 :

```
60      NEXT J,I
```

Vous pouvez emboiter des boucles à 3, 4 niveaux, ou plus encore. La seule limite est celle de l'espace disponible en mémoire centrale.

ERROR *code*

Cette instruction vous permet de simuler l'apparition d'une erreur déterminée pendant l'exécution d'un programme.

Elle servira surtout à vérifier le fonctionnement des routines d'erreur (exécutées à la suite d'un ON ERROR GOTO). Lorsque cette instruction est rencontrée, l'ordinateur se comporte exactement comme si l'erreur dont le code numérique est *code* venait de se produire. Le lecteur trouvera à l'annexe B la liste des codes d'erreurs et leurs significations.

Exemple de programme :

```
100     ERROR 1

RUN

?NF ERROR
READY
>_
```

1 est le code de l'erreur qui consiste à "tenter d'exécuter une instruction NEXT alors qu'aucune instruction FOR correspondante n'a été rencontrée".

Voir aussi **ON ERROR GOTO, RESUME.**

ON ERROR GOTO *nº ligne* [en français: EN CAS D'ERREUR ALLER EN *nº ligne*]

Normalement, lorsque l'ordinateur détecte une erreur lors de l'exécution d'un programme, il arrête cette dernière et imprime un message d'erreur.
Grâce à cette instruction ON ERROR GOTO, vous pouvez désormais "saisir au vol" cette erreur, la faire analyser et éventuellement corriger par une routine de traitement d'erreur, puis faire poursuivre, sans marquer d'arrêt, l'exécution du programme.

En principe, vous prévoyez le type d'erreur qui pourra se produire lorsque vous écrivez cette instruction.

Si, par exemple, votre programme effectue des opérations de division, et que vous n'admettiez pas de division par zéro, vous pourriez écrire une routine qui traite l'erreur de division par zéro et prévoir un branchement vers cette routine chaque fois que cette erreur se produira.

Exemple :

```
5       ON ERROR GOTO 100
6       INPUT X
10      C=1/X
```

Si vous introduisez zéro à la demande de la ligne 6, la ligne 10 provoquera une division par zéro, erreur qui provoquera le branchement vers la ligne 100, où vous aurez prévu de traiter cette erreur.

Remarque : un ON ERROR GOTO doit être exécuté avant qu'une erreur se produise; sinon l'erreur sera traitée normalement.

Une instruction ON ERROR GOTO peut être désactivée par l'exécution d'un ON ERROR GOTO 0. Dans une routine de traitement d'erreur, ceci permet au BASIC de traiter l'erreur courante de façon normale. **Attention** : l'exécution d'un CLEAR désactive également le dernier ON ERROR GOTO exécuté.

Toute routine de traitement d'erreur doit se terminer par une instruction RESUME.

Voir **RESUME.**

RESUME *nº ligne* [en français : REPRENDRE EN *nº ligne*]

Comme dernière instruction d'une routine de traitement d'erreur, RESUME indique l'endroit où l'exécution doit se poursuivre.

RESUME 0 et RESUME sans numéro de ligne font reprendre l'exécution à l'instruction qui a provoqué l'erreur.

RESUME suivi d'un numéro de ligne fait reprendre l'exécution à partir de la ligne spécifiée.

RESUME NEXT fait reprendre l'exécution à l'instruction qui suit celle qui a provoqué l'erreur.

Exemple de programme contenant un traitement d'erreur :

```
5       ON ERROR GOTO 100
10      INPUT "RACINE CARREE DE"; X
20      PRINT SQR(X)
30      GOTO 10
100     PRINT "RACINE IMAGINAIRE:";SQR(—X);"*I"
110     RESUME 10
```

Exécutez ce programme et introduisez des nombres négatifs.

REM

Cette instruction indique à l'ordinateur qu'il doit ignorer le reste de la ligne. Ceci vous permet de documenter un programme : vous pouvez en effet insérer des commentaires (REMarques) qui vous permettront (à vous ou à tout autre personne), lors d'une lecture ultérieure, de relire plus aisément ce programme. Dans une ligne à plusieurs instructions, REM ne peut être que la dernière instruction.

Exemple :

```
10  REM     **   PROGRAMME DE CALCUL GEOMETRIQUE          **
15  REM     **   AUTEUR : R. DUPONT                       **
20  REM     **   DATE : 18/2/1911                         **
25  REM     **                                            **
30  REM     **   VARIABLES : R=RAYON                      **
35  REM     **   C = LONGUEUR CIRCONF.                    **
40  REM     **   D = DIAMETRE                             **
45  REM
50  INPUT "RAYON"; R : REM CECI EST LA PREMIERE INSTRUCTION
    EXECUTABLE
```

Ce programme vous montre les possibilités et l'intérêt des instructions REM. Vous pouvez introduire n'importe quel caractère imprimable derrière le mot REM; la longueur totale maximum est de 255 caractères, comme pour les autres instructions.

En BASIC NIVEAU II, l'apostrophe ' (SHIFT 7) remplace le mot REM :

```
100   ' CECI EST EGALEMENT UNE REMARQUE
```

IF *expression logique liste d'actions* [en français : SI]

Cette instruction fait évaluer (tester ou vérifier) l'*expression logique*. Si celle-ci est vraie, la *liste d'actions* est exécutée jusqu'à ce que l'ordinateur rencontre soit un ELSE, soit la fin de la ligne. Si elle est fausse, l'ordinateur exécute les instructions qui suivent le ELSE correspondant s'il y en a, ou, s'il n'y en a pas, passe à la ligne suivante. Si l'expression logique est une expression, une variable ou une constante numérique, zéro est interprété comme faux et tout nombre non nul est interprété comme vrai.

Exemples :

```
100   IF X > 127 PRINT "SORTIE DE L'ECRAN" : END
```

Si X est supérieur à 127, les deux instructions PRINT puis END sont exécutées. Si X n'est pas supérieur à 127, l'exécution se poursuit directement à la ligne suivante, en sautant PRINT et END.

```
100   IF 0<=X AND X<=90 THEN Y=X+180
```

Si les deux expressions logiques sont vraies (c'est-à-dire si X est compris entre 0 et 90), Y reçoit la valeur de X + 180. Sinon, l'exécution passe à la ligne suivante, en sautant la clause THEN.

Remarque : dans l'instruction précédente, ainsi que dans d'autres du même type, THEN est facultatif. Il est cependant parfois nécessaire pour lever une ambiguïté. Par exemple, 400 IF Y=M THEN M=O ne fonctionnerait pas sans THEN.

```
500    INPUT A$ : IF A$="YES" THEN 100
600    INPUT A$ : iF A$="YES" GOTO 100
700    INPUT A$ : IF A$="YES" THEN GOTO 100
```

Ces trois instructions sont équivalentes. On notera qu'à la ligne 500, THEN est obligatoire, comme dans toute instruction du type IF *expression* THEN *n° ligne*.

```
100    IF A>0 AND B>0 PRINT "NOMBRES POSITIFS"
```

L'expression logique peut être constituée de plusieurs conditions unies par les opérateurs logiques AND (ET) et OR (OU)

Vois aussi **THEN, ELSE.**

THEN *liste d'actions ou n° ligne*

[en français : ALORS]

Définit la branche "vrai" d'une instruction IF. THEN est facultatif sauf lorsqu'il est suivi d'un numéro de ligne; il correspond alors à un branchement à la ligne spécifiée, comme dans IF A<0 THEN 100. THEN peut aussi apparaître dans la forme IF-THEN-ELSE.

ELSE *liste d'actions ou n° ligne*

[en français : SINON]

Définit la branche "faux" d'une instruction IF. Cette partie de l'instruction est exécutée lorsque l'expression logique du IF est reconnue fausse (en l'absence de branche ELSE, l'exécution se poursuit simplement à la ligne suivante).

Exemples :

```
100    INPUT A$ : IF A$="OUI" THEN 300 ELSE END
```

Dans cette ligne, si A$ est égal à "OUI", alors l'exécution se poursuit à la ligne 300. Si A$ n'est pas égal à "OUI", alors la branche ELSE est exécutée; elle commande l'arrêt de l'exécution.

```
200    IF A<B PRINT "A<B" ELSE PRINT "B<=A"
```

Si A est inférieur à B, le texte "A <B" est imprimé, puis l'exécution passe à la ligne suivante. Si A n'est pas inférieur à B, le texte "B <=A" est imprimé, puis l'exécution passe également à la ligne suivante. Dans les deux cas, une seule branche de l'alternative est exécutée.

```
200    IF A>.001 THEN B=1/A : A=A/S : ELSE 260
```

Si A >.001 est vrai, alors les deux instructions entre THEN et ELSE sont exécutées, affectant de nouvelles valeurs à B et A. Le programme s'exécute ensuite à partir de la ligne suivante, le ELSE étant ignoré.

Cependant, si A >.001 est faux, le programme ne change pas les valeurs de B et A mais effectue un branchement à la ligne 260. On notera que l'on aurait pu écrire aussi ELSE GOTO 260; d'autre part, le séparateur ":" est facultatif devant ELSE.

Des formes IF-THEN-ELSE peuvent être emboîtées à condition que l'on fasse correspondre soigneusement les IF et les ELSE.

```
10      INPUT "TAPEZ 2 NOMBRES"; A,B
20      IF A<=B THEN IF A<B PRINT A; : ELSE PRINT "AUCUN N' "; : ELSE
        PRINT B;
30      PRINT "EST LE PLUS PETIT"
```

Faites exécuter ce programme et fournissez-lui différents couples de nombres. Le programme recherche et imprime le plus petit nombre de chaque couple. Dans la ligne 20, THEN et les ":" peuvent être omis.

Conversion des données numériques

Chaque nombre utilisé pendant l'exécution d'un programme est stocké sous une forme (un type) entière, simple précision ou double précision. Il en résulte la nécessité de pouvoir convertir un nombre d'un type dans un autre. Ceci peut conduire à des résultats inattendus – à moins que vous ne compreniez les règles qui régissent la forme du stockage ainsi que de la conversion de types.

Type des constantes

Une constante est la représentation explicite d'un nombre (donc pas un nom de variable) qu'utilise le BASIC pendant l'exécution. Les constantes apparaissent dans votre programme (par exemple à droite d'un signe d'affectation : X=32) mais doivent également être stockées sous une forme interne afin de pouvoir intervenir dans les calculs relatifs à une expression. Les règles qui définissent le type de cette forme interne sont les suivantes :

I. Si la constante est représentée par au moins 8 chiffres significatifs ou si D apparaît devant l'exposant, le nombre correspondant est en double précision. On peut également ajouter le caractère de déclaration # pour forcer toute constante à être en double précision.

II. Si le nombre n'est pas en double précision selon la règle I, mais est extérieur à l'intervalle –32768, +32767, ou s'il contient un point décimal, il sera stocké en simple précision. Il en sera de même si l'exposant est indiqué par la lettre E.

III. Si les règles I et II ne sont pas d'application, la constante sera stockée sous forme entière.

Exemple de programme :

```
10      PRINT 1.234567, 1.2345678
RUN

  1.23457                1.2345678
READY
>_
```

La première constante contient 7 chiffres; elle est donc stockée en simple précision, en accord avec les règles I et II. D'autre part un nombre en simple précision est imprimé avec 6 chiffres significatifs, le dernier étant arrondi préalablement.

La seconde constante contient 8 chiffres et devient donc un nombre en double précision selon la règle I; celui-ci est stocké sous la forme 1.2345678000000000. Il est alors imprimé avec 8 chiffres significatifs, les zéros de droite étant supprimés.

Conversion de types

Le résultat d'une opération effectuée sur un ou deux nombres sera du type entier, simple précision ou double précision selon le type de ces nombres et l'opération.

Le résultat d'une opération +, − et * à la précision (type) de l'opérande le plus précis. Par exemple, l'addition d'un nombre en simple précision et d'un nombre en double précision donne un résultat en double précision. Si les deux opérandes sont du type entier, le résultat d'une opération +, − ou * sera du type entier s'il appartient à l'intervalle −32768, +32767, et du type simple précision s'il tombe en dehors de l'intervalle.

La division suit la même règle que les trois opérations précédentes, hormis le fait que le résultat sera au moins en simple précision, même si les deux opérandes sont du type entier.

Lors d'une comparaison, les opérandes sont d'abord convertis dans le même type, qui est le plus précis de celui des opérandes. Cette conversion peut entraîner des problèmes de précision (voir conversion de simple en double précision ci-dessous).

Les opérateurs logiques de manipulation de bits (AND, OR, NOT) convertissent d'abord leurs opérandes sous forme entière (voir chapitre 8). Si l'un de ces opérandes tombe en dehors de l'intervalle des entiers, il apparaîtra une erreur de "débordement" (overflow). Le résultat sera toujours un nombre entier.

Certaines fonctions, telles que SQR, EXP ou SIN, fournissent un résultat en simple précision.

Influence de la conversion de types sur la précision

Lorsqu'un nombre est converti en entier, il est "arrondi inférieurement"; c'est-à-dire qu'il lui correspond le plus grand entier qui ne lui soit pas supérieur. Ce principe est d'ailleurs également celui de la fonction INT.

Lorsqu'un nombre est converti de double en simple précision, il subit un arrondi 4/5 : le chiffre le moins significatif du résultat est augmenté de 1 si les chiffres qui le suivent sont supérieurs ou égaux à 5000..., et reste inchangé dans le cas contraire; ces chiffres suivants sont alors supprimés.

Examinez les exemples suivants en vous souvenant qu'un nombre en simple précision possède 7 chiffres significatifs mais n'en présente que 6 lors de l'impression (pour permettre un arrondi correct); de même, un nombre en double précision possède 17 chiffres significatifs mais 16 seulement sont imprimés.

Exemples de programmes :

```
10      A#=1.6666666666666667
20      B!=A#
30      C%=A#
40      PRINT B!,C%

RUN
 1.66667         1
READY
>_
```

Lorsqu'un nombre en simple précision est converti en double précision, seuls les 7 premiers chiffres sont exacts. Soyez prudents si le nombre initial contient moins de 7 chiffres significatifs.

Exemples :

```
10      A!=1.3
20      A#=A!
30      PRINT A#

RUN
1.29999952316284
READY
>_
```

```
10      A#=2/3
20      PRINT A#

RUN

 .6666666865348816
READY
>_
```

2/3 est convertit en simple précision, donc avec 7 chiffres exacts seulement.

```
10      A#=2/3#
20      PRINT A#

RUN

 .6666666666666667
READY
>_
```

L'évaluation de 2/3 # fournit un résultat en double précision, donc avec 16 chiffres exacts, le dernier étant "arrondi 4/5". (**Attention** il n'y a que 16 chiffres imprimés).

Lorsque vous affectez une constante à une variable, veillez à lui donner autant de chiffres significatifs que possible (jusqu'à 17).

Si votre constante n'a pas plus de 7 chiffres significatifs, utilisez alors une variable en simple précision.

Exemples :

```
10      PI#=3.1415926535897932  (valeur de π)
20      E#=2.7182818284590452   (valeur de e)
```

5 / Les chaînes de caractères

"Un ordinateur qui ne pourrait manipuler des chaînes de caractères ne serait qu'une puissante calculatrice". Il y a du vrai dans cette exagération, et c'est en utilisant les instructions de manipulation de chaînes du NIVEAU II que vous le découvrirez.

Le BASIC NIVEAU I possédait deux variables alphanumériques (c'est-à-dire qui peuvent contenir des chaînes de caractères alphabétiques et numériques) qui vous permettaient de rendre vos programmes plus "amicaux", à l'aide de messages personnalisés dans le genre : LE SCORE DE GEORGES EST 114. Vous pouvez aller bien au delà en NIVEAU II. D'abord, vous n'êtes pas limité à deux variables puisque tout nom de variable valide peut désigner une variable alphanumérique pourvu qu'il soit déclaré par DEFSTR ou suivi du caractère de déclaration $. En outre, chaque variable alphanumérique peut contenir jusqu'à 255 caractères.

Le NIVEAU II vous permet de comparer des chaînes, par exemple pour les arranger dans l'ordre alphabétique. Vous pouvez aussi les décomposer ou les assembler (c'est-à-dire les *concaténer*). Vous trouverez des renseignements utiles au Chapitre 1, "Types de variable" et "Glossaire", ainsi qu'au chapitre 4, **DEFSTR.**

Ce chapitre traitera des sujets suivants :

"Entrée/Sortie de chaînes"
"Comparaison de chaînes"
"Opérations sur les chaînes"
ASC
CHR$
FRE (*chaîne*)
INKEY$
LEN
LEFT$
MID$
RIGHT$
STR$
STRING$
VAL
INSTRING-
(sous-routine)

Entrée/sortie de chaînes

Il est possible de fournir à un programme des valeurs alphanumériques (c'est-à-dire des séquences de caractères alphanumériques) de la même manière que pour des valeurs numériques, en utilisant INPUT, INPUT # (cassette) et READ/DATA. Vous pourrez généralement les introduire sans guillemets :

```
10      INPUT "OUI OU NON"; R$
20      IF R$="OUI" PRINT "VOILA QUI EST POSITIF!" : END
30      PRINT "POURQUOI NON?"

RUN

OUI OU NON?–[vous tapez] OUI ENTER
VOILA QUI EST POSITIF!
READY
>_
```

Vous devrez cependant entourer une valeur de guillemets lorsqu'elle contient des virgules ou des deux-points, ou lorsqu'elle commence ou se termine par des espaces.

```
10      INPUT "NOM ET PRENOM";N$
20      PRINT N$

RUN

NOM ET PRENOM ?- [vous tapez] "LEMERCIER, CLAIRE" ENTER
LEMERCIER, CLAIRE
READY
>_
```

Cette contrainte s'applique également aux valeurs d'une liste DATA.

```
10      READ M$, C$, F$
20      PRINT M$, C$, F$
30      DATA "TOTAL:", "MILLE, DEUX CENTS"
40      DATA FRANCS
```

La première valeur exige des guillemets à cause de la présence de deux-points, la deuxième valeur à cause de la virgule.

Comparaison de chaînes

Il est possible de comparer des chaînes pour vérifier leur égalité ou leur position respective selon l'ordre alphabétique. L'égalité est vérifiée si chaque caractère est identique dans les deux chaînes, y compris les espaces de début et de fin.

```
600   IF Z$="END" THEN 999
```

La comparaison s'effectue caractère par caractère, de gauche à droite. En réalité, la comparaison porte sur le code ASCII des caractères; le caractère dont le code est le plus petit est considéré comme précédant le second (voir Codes ASCII, Annexe C).

Par exemple, le valeur "A!" précède la valeur "A #" puisque "!" (code ASCII 33 en décimal) précède " #" (code ASCII 35). Lorsque deux chaînes ont des longueurs différentes, **la plus courte précède l'autre si ses caractères sont les mêmes que les caractères correspondants de la seconde.** Par exemple, "A" précède "A ".

Vous pourrez utiliser les symboles de comparaison suivants :

=	< >	<	< =	>	> =

Remarque : une constante alphanumérique, utilisée dans une comparaison ou une instruction d'affectation, doit être entourée de guillemets :

```
10      A$="CONSTANTE"
20      IF A$="CONSTANTE" PRINT A$
```

(les guillemets sont ici obligatoires).

Opérations sur les chaînes

Si l'on excepte les fonctions décrites ci-dessous, il n'y a qu'un seul opérateur de chaîne : la concaténation, représentée par le symbole +.

Exemple de programmes :

```
10      CLEAR 75
20      A$="UNE ROSE"
30      B$="EST UNE ROSE"
40      C$=A$+B$+B$+B$+"."
50      PRINT C$

RUN

UNE ROSE EST UNE ROSE EST UNE ROSE EST UNE ROSE.
READY
>_
```

La ligne 40 réalise la concaténation des chaînes, c'est-à-dire leur assemblage.

```
10      T$="100"
20      SUB$="5"
30      CODE$="32L"
40      LC$=T$+"."+SUB$+CODE$
50      PRINT LC$

RUN

100.532L
READY
>_
```

ASC (*chaîne*)

Cette fonction fournit, sous forme décimale, le code ASCII du premier caractère de la *chaîne* spécifiée.

```
100     PRINT ASC("A")
110     T$="AB": PRINT ASC(T$)
```

Ces deux lignes vont imprimer la même valeur : 65

L'argument de ASC peut être une expression faisant appel à des opérateurs et des fonctions alphanumériques :

```
200     PRINT ASC(RIGHT$(T$,1))
```

On se référera à la table des codes ASCII de l'annexe C. On observera que le code des lettres minuscules est égal au code de la majuscule correspondante plus 31.

ASC vous permet donc de convertir les majuscules en minuscules, ce qui est particulièrement utile si vous possédez une imprimante adaptée aux minuscules ainsi que les interfaces nécessaires.

ASC peut également servir à élaborer des fonctions de codage et décodage (voir l'exemple en fin de ce chapitre).

CHR$ (*expression*)

Réalise la fonction inverse de ASC : elle fournit une chaîne d'un caractère correspondant au code ASCII, graphique ou de contrôle dont la valeur est celle de l'*expression*. Cette valeur doit être positive ou nulle et inférieure à 256. L'argument doit se trouver entre parenthèses.

```
100     PRINT CHR$(37)          imprimé le symbole %
```

Cette fonction vous permet également d'introduire des guillemets dans une chaîne, alors qu'il s'agit d'un délimiteur de constante. Le code ASCII des guillemets est 34. Par conséquent, CHR$(34) correspond au symbole".

```
100     A$=CHR$(34)
110     PRINT "IL PRECISA,"; A$; "JE REVIENDRAI"; A$
RUN

IL PRECISA, "JE REVIENDRAI"
READY
>_
```

CHR$ vous permet aussi d'afficher chacun des 64 caractères graphiques. (voir Annexe C, les Codes Graphiques)

```
10      CLS
20      FOR I=128 TO 191
30      PRINT I;CHR$(I),
40      NEXT
50      GOTO 50
```

(Ce programme affiche les caractères graphiques disponibles).

Les codes 0-31 sont des codes de commande de l'écran. L'impression du "caractère" correspondant provoque simplement l'exécution de la commande. Par exemple, 23 est le code du format d'affichage à 32 caractères par ligne; la commande PRINT CHR$(23) convertit l'affichage en format 32c/l. (La touche CLEAR, ou l'exécution CLS permet le retour en 64 c/l).

FRE (*chaîne*)

[en français : LIBRE]

Utilisée avec un argument alphanumérique (constante ou variable), cette fonction fournit la taille de la mémoire réservée aux chaînes de caractères qui reste disponible. L'argument, qui n'a pas d'utilité en soi et constitue simplement un argument fictif, doit être entre parenthèses.

```
500     PRINT FRE(A$), FRE(L$), FRE("Z")
```

La fonction fournit ici trois fois la même valeur.

Voir Chapitre 4, **CLEAR** *n*

INKEY$

[en français : LIRE CLAVIER]

Cette fonction entraîne une consultation du clavier et fournit comme résultat une chaîne constituée du caractère correspondant à la touche frappée. Si aucune touche n'a été enfoncée pendant la consultation, une chaîne vide (de longueur nulle est donc égale à "") sera fournie. Cette fonction de "lecture" du clavier est particulièrement utile pour permettre l'introduction de données pendant l'exécution d'un programme sans faire appel à l'instruction INPUT qui perturbe l'écran. Les jeux TV dans lesquels vous tirez sur une cible, vous guidez un engin dans un labyrinthe ou vous jouez au tennis peuvent tous être réalisés grâce à la fonction INKEY$ (ainsi qu'à quelques autres instructions !).

Les caractères fournis **ne sont pas affichés** à l'écran.
En raison de la courte durée du cycle de consultation (quelques micro-secondes) la fonction INKEY$ sera généralement placée dans une boucle de lecture du clavier.

Exemple :

```
10      CLS
100     PRINT @ 540,INKEY$ : GOTO 100
RUN
```

Vous constaterez que l'écran reste vide jusqu'au moment où vous tapez sur une touche pour la première fois. Le caractère affiché subsiste jusqu'à ce que vous en tapiez un autre (lorsque vous ne tapez pas de caractère pendant la consultation par INKEY$, le programme affiche le résultat, c'est-à-dire une chaîne vide, ou encore "rien". Ce "rien" n'a pas d'effet sur l'affichage précédent.

L'exemple suivant est celui d'un programme qui construit par INKEY$ une chaîne de trois caractères.

```
90      PRINT "TAPEZ TROIS CARACTERES"
100     A$=INKEY$ : IF A$=""THEN 100 ELSE PRINT A$;
110     B$=INKEY$ : IF B$=""THEN 110 ELSE PRINT B$;
120     C$=INKEY$ : IF C$=""THEN 120 ELSE PRINT C$;
130     D$=A$+B$+C$
```

Ce programme vous permet d'introduire une chaîne de trois caractères sans utiliser la touche **ENTER** ni l'instruction INPUT.

Remarque : IF A$="" compare la valeur de A$ à une chaîne vide.

LEFT$ (*chaîne, n*)

[en français : GAUCHE]

Fournit les *n* premiers caractères de la *chaîne*.
Les arguments *chaîne* et *n* peuvent être des constantes ou des expressions en général; ils doivent se trouver entre parenthèses. Si *n* est supérieur à LEN (*chaîne*), seule *chaîne* est fournie.

Exemple de programme :

```
10      A$="TIMOTHEE"
20      B$=LEFT$(A$,3)
30      PRINT B$; "EST UNE ABREVIATION DE"; A$
```

```
RUN

TIM EST UNE ABREVIATION DE TIMOTHEE
READY
>_
```

LEN (*chaîne*)

[en français : LONGUEUR]

Fournit le nombre de caractères de la *chaîne*. L'argument, entre parenthèses, peut être une constante ou une expression.

```
10      A$=""
20      B$="JEAN"
30      PRINT A$,B$,B$+B$
40      PRINT LEN(A$), LEN(B$), LEN(B$+B$)
```

```
RUN

                JEAN                    JEANJEAN
0                4                       8
READY
>_
```

MID$ (*chaîne, p, n*)

Fournit le fragment de *chaîne* de longueur *n* et commençant au caractère de position *p*. Les trois arguments, qui doivent se trouver entre parenthèses, peuvent être des constantes ou des expressions. Par exemple, MID$ (L$,3,1) est le fragment de L$ constitué du troisième caractère.

Exemple de programme :

Un numéro de téléphone international est constitué d'un préfixe de deux chiffres indiquant un numéro étranger, d'un code de 2 chiffres représentant un pays puis du numéro national proprement dit. Le programme suivant extrait le code du pays d'un numéro international :

```
10      INPUT "NUMERO INTERNATIONAL (SANS ESPACE NI TIRET)"; T$
20      P$=MID$(T$,3,2)
30      PRINT "CODE DU PAYS:"; P$
```

Exemple :

STRING$(64,191) crée une chaîne de 64 blocs graphiques.

VAL (*chaîne*)

Réalise la fonction inverse de STR$: elle fournit la valeur numérique représentée par la *chaîne* de caractères. Si A$="12" et B$="31", alors VAL(A$+"."+B$) fournit la valeur numérique 12.31. Quant à VAL(A$+"E"+B$), elle fournit la valeur 12E31 soit encore $12x10^{31}$.

Si *chaîne* contient des chiffres et des caractères, seuls les chiffres initiaux sont traités, les caractères suivants étant abandonnés. Cependant, le point décimal et les caractères d'exposant E et D sont pris en compte.

Exemple : VAL("100 DOLLARS") fournit la valeur 100.

Le programme suivant extrait le numéro du domicile dans une adresse :

```
10      INPUT "ADRESSE : NUMERO ET RUE"; AD$
20      IF VAL(AD$)=0 PRINT "PAS DE NUMERO" ELSE PRINT VAL(AD$)
30      GOTO 10
```

Faites exécuter ce programme et introduisez des données telles que "115 RUE DE RENNES" ou "AVENUE LOUISE".

Programme d'encodage/décodage : exemple de traitement de caractères

```
5       CLS: PRINT CHR$(23)
10      CLEAR 1000
20      INPUT "QUEL EST VOTRE MESSAGE"; M$
30      FOR K=1 TO LEN(M$)
40        T$=MID$(M$,K,1)
60        CD=ASC(T$)+5: IF CD>255 THEN CD=CD-255
70        NU$=NU$+CHR$(CD)
80      NEXT
90      PRINT "LE MESSAGE CODE EST"
100     PRINT NU$
110     FOR K=1 TO LEN(NU$)
120       T$=MID$(NU$,K,1)
130       CD=ASC(T$)-5: IF CD<0 THEN CD=CD+255
140       ANC$=ANC$+CHR$(CD)
150     NEXT
160     PRINT "L'ANCIEN MESSAGE ETAIT"
170     PRINT ANC$
```

Les lignes 30-80 et 110-150 montrent comment il est possible d'extraire un à un les caractères d'une chaîne; les lignes 60 et 130 illustrent la manipulation de codes ASCII.

Recherche d'un fragment donné dans une chaîne : sous-routine "INSTRING"

Les fonctions LEN ET MID$ vont nous permettre de construire une sous-routine de manipulation de chaîne extrêmement utile. Elle réalise la fonction suivante: étant donné deux chaînes X$ et Y$, elle détermine la présence ou l'absence de Y$ dans X$. Elle vous permettra de rechercher un mot, un groupe de mots ou une phrase dans un texte. Voici cette sous-routine :

```
999     END: "END DE PROTECTION"
1000    'SOUS-ROUTINE INSTRING(X$,Y$,I)
1010    FOR I=1 TO LEN(X$)-LEN(Y$)+1
1020    IF Y$=MID$(X$,I,LEN(Y$)) RETURN
1030    NEXT : I=0 : RETURN
```

Pour utiliser cette sous-routine, affectez à Y$ le fragment cherché et à X$ la chaîne dans laquelle la recherche doit être réalisée (le domaine de recherche). Après l'appel par GOSUB, la sous-routine a affecté à I soit 0 si Y$ n'a pas été trouvé dans X$, soit, dans le cas contraire, la position dans X$ du début de la première apparition de Y$.

Le programme suivant est un exemple d'utilisation de cette sous-routine. (n'oubliez pas les lignes 999-1030).

```
5       CLEAR 1000 : CLS
10      INPUT "TAPEZ LE DOMAINE DE RECHERCHE": X$
20      INPUT "TAPEZ LE FRAGMENT A RECHERCHER; Y$
30      GOSUB 1000
40      IF I=0 THEN 70
50      PRINT Y$; "EST UN FRAGMENT DE"; X$
55      PRINT "POSITION: DE"; I; "A"; I+LEN(Y$)-1
60      PRINT:PRINT:GOTO 10
70      PRINT Y$; "N'EST PAS UN FRAGMENT DE"; X$
80      PRINT:PRINT:GOTO 10
```

Essayez ce programme avec différentes valeurs.

6 / Les tableaux

Un tableau est un arrangement ordonné d'éléments pouvant chacun contenir une valeur. En NIVEAU II, ces valeurs seront des nombres ou des chaînes de caractères selon la DEFinition du tableau ou la façon dont son nom est constitué. Le tableau permet la manipulation rapide de gros volumes de données. Pour illustrer ces possibilités, ce chapitre présentera l'utilisation d'un tableau pour la tenue à jour d'un carnet de chèques. Nous y enregistrerons pour chaque chèque, son numéro, la date d'émission et son montant.

Vous trouverez également dans ce chapitre plusieurs sous-routines de manipulation de matrices représentées sous forme de tableaux. Elles vous permettront dè réaliser des opérations telles que l'addition, la multiplication et la transposition de matrices.

Remarque : tout au long de ce chapitre, les éléments d'indice zéro seront généralement ignorés pour des raisons de simplicité. On se souviendra cependant qu'ils sont disponibles, et qu'ils devraient être utilisés afin d'économiser la mémoire centrale. Par exemple, après DIM A(4), cinq éléments sont disponibles : A(0), A(1), A(2), A(3), A(4).

Vous trouverez aux Chapitres 4(**DIM**) et 1("Tableaux") les renseignements nécessaires sur les tableaux et leur déclaration.

Tenue d'un carnet de chèques

Considérons la table de renseignements suivante :

Numéro chèque	Date émission	Montant
025	79-1-13	170.00
026	79-1-18	520.50
027	79-2-4	1300.00
028	79-3-2	920.00
029	79-3-18	250.00
030	79-3-23	2750.00

Chaque valeur de cette table peut être désignée en spécifiant simplement le numéro de la ligne et le numéro de la colonne où elle se trouve. Par exemple, la ligne 4 et la colonne 3 définissent la valeur 920.
La paire de nombre (4,3) s'appelle les *indices* de la valeur 920 dans la table.

Constituons un tableau de nom CH contenant les données de cette table. Puisqu'il nous faut 6 lignes et 3 colonnes, nous donnerons à CH deux dimensions. Ses éléments, définis chacun par deux indices (indice de ligne et indice de colonne) pourraient être visualisés comme suit :

A(1,1)=025	A(1,2)=79.0113	A(1,3)= 170.00
.	.	.
.	.	.
.	.	.
.	.	.
A(6,1)=030	A(6,2)=79.0323	A(6,3)=2750.00

On notera la façon dont les dates ont été représentées : *aa.mmjj* où *aa* sont les deux derniers chiffres de l'année, *mm* le numéro du mois et *jj* le numéro du jour dans le mois. Tous les éléments d'un tableau étant homogènes (numériques ou alphanumériques), il a fallu transformer en valeurs numériques les dates de la forme *aa-mm-jj*.

En l'absence d'une déclaration DIM, tout tableau est supposé contenir 10 éléments dans chacune des directions (plus l'élément zéro!), soit donc pour CH, 10x10=100 éléments (en fait 11x11). Puisqu'il ne nous en faut que 6x3=18, nous éviterons ce gaspillage en déclarant :

```
10      DIM CH(6,3) : 'CONSTITUTION D'UN TABLEAU DE 6x3 (0 EXCLUS)
```

Ajoutons-y les instructions qui vont affecter aux éléments les valeurs voulues :

```
20      FOR LIGNE=1 TO 6
30      FOR COL=1 TO 3
40      READ CH(LIGNE, COL)
50      NEXT COL, LIGNE
90      DATA 025, 79.0113, 170
91      DATA 026, 79.0118, 520.50
92      DATA 027, 79.0204, 1300
93      DATA 028, 79.0302, 920
94      DATA 029, 79.0318, 250
95      DATA 030, 79.0323, 2750
```

Le tableau CH étant garni, nous allons l'exploiter en profitant de sa structure. Calculons par exemple, la somme des montants émis. Complétons le programme par les lignes suivantes :

```
100     FOR LIGNE=1 TO 6
110         TOT=TOT+CH(LIGNE,3)
120     NEXT
130     PRINT "TOTAL DES CHEQUES EMIS";
140     PRINT USING "**###.#"; TOT
```

Les lignes suivantes vous permettent d'imprimer les renseignements sur les chèques émis à une date déterminée :

```
200     PRINT "DATE D'EMISSION SOUS LA FORME (AA.MMJJ)";
210     INPUT DATE
220     PRINT:PRINT "NUMERO", "MONTANT": PRINT
230     FOR LIGNE=1 TO 6
240          IF CH(LIGNE,3)=DATE PRINT CH(LIGNE,1), CH(LIGNE,3)
250     NEXT
```

Ce programme peut aisément être modifié pour représenter un carnet de chèques plus important.

Il suffit pour cela d'augmenter la taille de CH et de modifier en conséquence les limites des boucles.

Imaginons un carnet de 300 chèques. Modifiez le programme comme suit :

```
10      DIM CH(300,3) : 'CONSTITUTION D'UN TABLEAU DE 300x3
20      FOR LIGNE=1 TO 300
```

et d'ajouter les lignes DATA décrivant tous les chèques émis. Il sera probablement nécessaire d'écrire plus de données par ligne que dans la version précédente. N'oubliez pas non plus de modifier les lignes :

```
100     FOR LIGNE=1 TO 300
.
.
230     FOR LIGNE=1 TO 300
```

Autres types de tableaux

En NIVEAU II, le nombre de dimensions et la taille des tableaux ne sont limités que par l'espace de mémoire disponible. Vous pouvez donc définir des tableaux à plus de deux dimensions.

D'autre part, il est possible de définir des tableaux alphanumériques, dont les éléments peuvent contenir des chaînes de caractères. Par exemple, DEFSTR A va rendre alphanumériques tous les tableaux dont le nom commence par A; de même, C$(X) est un élément du tableau alphanumérique C$.
Un exemple d'utilisation d'un tel tableau est le stockage d'un texte à traiter.

```
10      CLEAR 1200
20      DIM TXT$(10)
```

Ces définitions vous permettront de stocker 10 lignes de texte. Si chaque ligne comporte soixante caractères, il vous restera 600 caractères destinés à d'autres variables alphanumériques.

Sous-routines de manipulation de tableaux

Ces sous-routines réalisent des opérations classiques sur les tableaux. Le lecteur un tant soit peu mathématicien aura déjà constaté qu'une matrice ou un vecteur peuvent aisément être représentés par un tableau; d'où l'intérêt de ce paragraphe ! Le programme appelant doit avoir initialisé les variables N1, N2, N3 qui correspondent aux bornes des tableaux.

```
30100   REM GARNISSAGE DU TABLEAU A (2 DIMENSIONS) PAR "INPUT"
30110   FOR I=1 TO N1
30120   PRINT "NUMERO DE LIGNE"; I
30130      FOR J=1 TO N2
30140      INPUT A(I,J)
30160   NEXT J,I
30170   RETURN
```

```
30200   REM GARNISSAGE DU TABLEAU A (3 DIMENSIONS) PAR "READ/DATA"
30210   FOR K=1 TO N3
30220      FOR I=1 TO N1
30230         FOR J=1 TO N2
30240         READ A (I,J,K)
30270   NEXT J,I,K
30280   RETURN
```

```
30300   REM MISE A ZERO DU TABLEAU A (3 DIMENSIONS)
30310   FOR K=1 TO N3
30320      FOR J=1 TO N2
30330         FOR I=1 TO N1
30340         A (I,J,K)=0
30370   NEXT I,J,K
30380   RETURN
```

```
30400   REM IMPRESSION DU TABLEAU A (3 DIMENSIONS)
30410   FOR K=1 TO N3
30420      FOR I=1 TO N1
30430         FOR J=1 TO N2
30440         PRINT A (I,J,K),
30450         NEXT J:PRINT
30460      NEXT I:PRINT
30470   NEXT K:PRINT
30480   RETURN
```

```
30500   REM GARNISSAGE DU TABLEAU A (3 DIMENSIONS) PAR "INPUT"
30510   FOR K=1 TO N3
30520   PRINT "NUMERO DE PAGE"; K
30530      FOR I=1 TO N1
30540      PRINT "NUMERO DE LIGNE"; 1
30550         FOR J=1 TO N2
30560         INPUT A (I,J,K)
30570         NEXT J
30580      NEXT I
30590   PRINT:NEXT K
30595   RETURN
```

Multiplication scalaire (Tableau x valeur) à 3 dimensions; les éléments de A sont multipliés par X pour fournir le tableau B.

```
30600   FOR K=1 TO N3
30610      FOR J=1 TO N2
30620         FOR I=1 TO N1
30630         B (I,J,K)=A (I,J,K)*X
30640         NEXT I
30650      NEXT J
30660   NEXT K
30670   RETURN
```

Transposition du tableau A dans le tableau B (2 dimensions).

```
30700   FOR I=1 TO N1
30710      FOR J=1 TO N2
30720      B(J,I)=A(I,J)
30730      NEXT J
30740   NEXT I
30750   RETURN
```

Addition des deux tableaux A et B; le résultat se trouve dans C (3 dimensions).

```
30800   FOR K=1 TO N3
30810      FOR J=1 TO N2
30820         FOR I=1 TO N1
30830         C(I,J,K)=A(I,J,K)+B(I,J,K)
30840         NEXT I
30850      NEXT J
30860   NEXT K
30870   RETURN
```

Multiplication des deux tableaux A et B élément par élément; le résultat se trouve dans C (3 dimensions).

```
30900   FOR K=1 TO N3
30910      FOR J=1 TO N2
30920         FOR I=1 TO N1
30930         C(I,J,K)=A(I,J,K)*B(I,J,K)
30940      -  NEXT I
30950      NEXT J
30960   NEXT K
```

Multiplication matricielle C=A*B; A est un tableau de N1xN3, B un tableau de N3xN2 et C un tableau de N1xN2.

```
40000   FOR I=1 TO N1
40010      FOR J=1 TO N2
40020      C(I,J)=0
40030         FOR K=1 TO N3
40040         C(I,J)=C(I,J)+A(I,K)*B(K,I)
40050         NEXT K
40060      NEXT J
40070   NEXT I
```

7 / Les fonctions Arithmétiques

Le BASIC NIVEAU II offre une large gamme de fonctions "intrinsèques" (c'est-à-dire directement intégrée au BASIC) qui réalisent des opérations mathématiques et des opérations spéciales. Ces dernières font l'objet du chapitre suivant.

Les fonctions mathématiques classiques fournissent un résultat en simple précision (soit 6 chiffres significatifs). ABS, FIX et INT fournissent un résultat dont la précision dépend de celle de l'argument. Les fonctions de conversion (CINT, CDBL, etc) définissent elles-mêmes la précision de leur résultat. L'argument d'une fonction, toujours entre parenthèses, peut être une constante, une variable ou, d'une manière générale, une expression.

Ce chapitre décrit les fonctions suivantes :

ABS	COS	INT	SGN
ATN	CSNG	LOG	SIN
CDBL	EXP	RANDOM	SQR
CINT	FIX	RND	TAN

ABS (*x*)

Fournit la valeur absolue de l'argument. ABS(X)=X si X est supérieur ou égal à 0, et ABS(X)=–X si X est négatif.

```
100  IF ABS(X) <1E–6 PRINT "TROP PETIT"
```

ATN (*x*)

Fournit, en radians, l'arc-tangente de l'argument, c'est-à-dire l'angle (ou l'arc) dont la tangente est x. Le résultat exprimé en degrés est obtenu en multipliant le résultat de ATN par 57.29578.

```
100  Y=ATN(B/C)
```

CDBL (*x*)

Fournit une représentation de l'argument en double précision. Le résultat contient 17 chiffres, mais seuls les chiffres présents dans l'argument sont exacts.

CDBL permet de forcer l'exécution d'une opération en double précision même si les opérandes sont en simple précision ou entiers. Par exemple, le résultat de CDBL (I%)/J% contiendra 17 chiffres significatifs.

```
100  FOR I%=1 TO 25 : PRINT 1/CDBL(I%), : NEXT
```

CINT (*x*)

Fournit le plus grand entier qui n'est pas plus grand que l'argument. Par exemple, CINT (1.5) a pour résultat 1, tandis que CINT (–1.5) a pour résultat –2. L'argument doit appartenir à l'intervalle –32768 à +32767.

CINT peut être utilisée pour accélérer les opérations sur des arguments en simple ou double précision lorsque l'on ne s'intéresse qu'à leur partie entière.

COS (*x*)

Fournit le cosinus de l'argument (qui doit être exprimé en radians).
Si X est exprimé en degrés, son cosinus sera obtenu par COS (X*.0174533)

```
100     Y=COS(X+3.3)
```

CSNG (*x*)

Fournit une représentation de l'argument en simple précision. Un argument en double précision donne un résultat à 6 chiffres significatifs avec "arrondi 4/5" du dernier chiffre. Par exemple, CSNG(.6666666666666667) vaut .666667, tandis que CSNG(.3333333333333) vaut .333333.

```
100     PRINT CSNG(A#+B#)
```

EXP (*x*)

Fournit l'exponentielle de l'argument, soit, si celui-ci est X, e^X. Il s'agit de l'inverse de la fonction LOG, d'où la relation : X=EXP(LOG(X))

```
100     PRINT EXP(–X)
```

FIX (*x*)

Fournit une représentation tronquée de l'argument. Tous les chiffres à la droite du point décimal sont simplement abandonnés, fournissant ainsi un résultat entier. Si X est non-négatif, FIX(X)=INT(X); si X est négatif, FIX(X)=INT(X)+1. Par exemple, FIX(2.2) vaut 2, tandis que FIX(–2.2) vaut –2.

```
100     Y=ABS(A–FIX(A))
```

Cette instruction fournit la partie fractionnaire de A.

INT (*x*)

Fournit le plus grand entier qui n'est pas plus grand que l'argument. A la différence de CINT, la valeur de l'argument **n'est pas** limitée à l'intervalle −32768 à +32767. INT(2.5) fournit 2; INT(−2.5) fournit −3; INT(1234567.8) fournit 1234567.

```
100     Z=INT(A*100+.5)/100
```

Cette instruction range dans Z la valeur de A arrondie à deux décimales (pour A non négatif).

LOG (*x*)

Fournit le logarithme naturel de l'argument, c'est-à-dire $\log_e(x)$. Il s'agit de l'inverse de la fonction EXP, d'où la relation X=LOG(EXP(X)). Le logarithme d'un nombre dans une base *b* s'obtient par la formule : $\log_b(X) = \log_e(X)/\log_e(b)$. Par exemple, LOG(32767)/LOG(2) fournit le logarithme de 32767 en base 2.

```
100     PRINT LOG(3.3*X)
```

RANDOM

[en français : ALEATOIRE]

RANDOM est en réalité une instruction complète et non une fonction. Elle réinitialise le générateur de nombres aléatoires. Utilisée en début de programme elle permet d'obtenir, par l'exécution de RND, une séquence pseudo-aléatoire réellement imprévisible chaque fois que vous allumez l'ordinateur et que vous exécutez le programme après l'avoir chargé.

```
10      RANDOM
20      C=RND(6)+RND(6)
.
.
.
80      GOTO 20 : 'RANDOM N'EST EXECUTE QU'UNE FOIS.
```

RND (*x*)

Génère un nombre pseudo-aléatoire à partir du nombre origine courant (inaccessible à l'utilisateur, réinitialisable par RANDOM).
RND peut être utilisée pour produire des nombres entre 0 et 1, ou des nombres entiers plus grands que 0 selon la valeur de l'argument *x*.

RND(0) fournit un nombre en simple précision entre 0 et 1.
RND (*entier*) fournit un nombre entre 1 et *entier* inclus; *entier* doit être compris entre 1 et 32768 inclus. Par exemple, RND(55) fournit un nombre entier supérieur à 0 et inférieur à 56.
RND réalise préalablement un INT (*entier*) de telle sorte que RND(55.2) soit équivalent à RND(55).

```
100     X=RND(2) : ON X GOTO 200, 300
```

SGN (*x*)

Cette fonction "signe de *x*" fournit –1 si *x* est négatif, 0 si *x* est nul et +1 si *x* est positif.

```
100     ON SGN(X)+2 GOTO 200,300,400
```

SIN (*x*)

Fournit le sinus de l'argument (qui doit être exprimé en radians).
Si X est exprimé en degrés, son sinus sera obtenu par SIN(X*.0174533).

```
100     PRINT SIN(A*B–B)
```

SQR (*x*)

Fournit la racine carrée de l'argument. SQR(X) est équivalente à X↑(1/2) tout en étant plus rapide à l'évaluation.

```
100     Y=SQR(X↑2–H↑2)
```

TAN (*x*)

Fournit la tangente de l'argument (qui doit être exprimé en radians).
Si X est exprimé en degrés, sa tangente sera obtenue par
TAN (X*.0174533).

```
100     Z=TAN(2*A)
```

Remarque : bien d'autres fonctions peuvent être construites à partir de ces fonctions de base. Vous en trouverez quelques-unes dans l'annexe F, "Fonctions Dérivées".

8 / Les fonctions Spéciales

Le BASIC NIVEAU II propose quelques fonctions et instructions spéciales qui méritent que l'on y consacre un chapitre entier. Certaines vous sembleront destinées à des spécialistes, mais au fur et à mesure que vous avancerez en programmation, elles vous paraîtront de plus en plus utiles. D'autres, par contre, vous sembleront dès le départ d'un intérêt plus évident, telles par exemple, les fonctions graphiques. Enfin, les deux fonctions INP et OUT, seront surtout utilisées avec l'Interface d'Extension.

Ce chapitre sera consacré à l'étude des fonctions, instructions et opérateurs suivants :

Graphique :

SET
RESET
CLS

POINT

Contrôle d'erreur :

ERL
ERR

Opérateurs logiques :

AND
OR
NOT

Autres fonctions et instructions :

INP
MEM
PEEK

POKE
POS
OUT
USR
VARPTR

SET (*x*, *y*)

Allume le segment graphique dont la position est spécifiée par les coordonnées *x* et *y*. En utilisation graphique, l'écran est divisé selon une grille de 128 colonnes de 48 lignes. L'axe des *x* est défini de gauche à droite sur les valeurs 0 à 127. L'axe des *y* est défini de haut en bas sur les valeurs 0 à 47. Par conséquent, le point (0,0) est dans le coin supérieur gauche de l'écran tandis que le point (127,47) est dans le coin inférieur droit. Voir à ce sujet la feuille de travail Vidéo à l'annexe E.

Les arguments *x* et *y* peuvent être des constantes, des variables ou des expressions en général; leurs valeurs ne doivent pas nécessairement être entières car SET les arrondit (par INT) avant affichage. Set (X,Y) n'est valide que pour:

$0 <= x < 128$
$0 <= y < 48$

Exemples :

```
100    SET(RND(128)—1, RND(48)—1)
```

Allume un segment au hasard sur l'écran.

```
100    INPUT X,Y: SET(X,Y)
```

Allume le segment spécifié au clavier.

RESET (x, y)

Eteint le segment graphique dont la position est spécifiée par les coordonnées x et y. Les contraintes de validité et les caractéristiques sont les mêmes que pour SET.

```
200     RESET(X,3)
```

CLS

Eteint tous les segments graphiques de l'écran, efface les caractères et positionne le curseur dans le coin supérieur gauche. CLS vous permet de "nettoyer" l'écran avant d'y afficher des informations.

```
5       CLS
10      SET(RND(128)—1, RND(48)—1)
20      GOTO 10
```

POINT (x, y)

Teste le segment de coordonnée (x, y) pour déterminer s'il est allumé. Si ce segment est allumé (par un SET), POINT fournit la valeur VRAI (c'est-à-dire –1 en BASIC NIVEAU II). Sinon, POINT fournit la valeur FAUX (0). POINT sera généralement utilisé dans une instruction IF-THEN.

```
100     SET (50, 28) : IF POINT (50, 28) THEN PRINT "OUI" ELSE PRINT "NON"
```

Cette ligne imprimera toujours le message "OUI", car POINT fournira toujours la valeur VRAI.

ERL

Fournit le numéro de la ligne à laquelle s'est produite la dernière erreur. Cette fonction sera surtout utilisée dans une routine de traitement d'erreur activée par ON ERROR GOTO. Si aucune erreur ne s'est produite ERL fournit la valeur 0. Si l'erreur s'est produite en mode direct ERL fournit le plus grand entier de deux bytes, soit 65535.

Exemple de programme utilisant ERL

```
5       ON ERROR GOTO 1000
10      CLEAR 10
20      INPUT "TAPEZ UN MESSAGE"; M$
30      INPUT "TAPEZ UN NOMBRE"; N: N=1/N
40      REM SUITE DU PROGRAMME
.
.
.
999     END
1000    IF ERL=20 THEN 1010 ELSE IF ERL=30 THEN 1020
1005    ON ERROR GOTO 0
1010    PRINT "10 CARACTERES AU MAXIMUM–RECOMMENCEZ"
1015    RESUME 20
1020    PRINT "NOMBRE NON NUL–RECOMMENCEZ"
1025    RESUME 30
```

Faites exécuter ce programme. Répondez par un message trop long (plus de 10 caractères) puis par le nombre zéro. A la ligne 1000, ERL est testé pour déterminer l'endroit où l'erreur s'est produite afin de décider des mesures à prendre.

ERR/2 + 1

Utilisée parallèlement à ERL, ERR fournit une valeur qui est fonction du code de l'erreur qui vient de se produire. ERL et ERR permettent d'écrire des routines de traitement d'erreurs particulièrement puissantes.
ERR ne fournit pas directement le code d'erreur tel qu'il est donné à l'annexe B, "Codes d'erreur"; on a en fait les relations suivantes :

ERR/2+1=code d'erreur
2*(code d'erreur –1)=ERR

Exemple de programme :

```
10      ON ERROR GOTO 1000
20      DIM A(15) : I=1
30      READ A(I)
40      I=I+1 : GOTO 30
50      REM SUITE DU PROGRAMME
.
.
.
100     DATA 2,3,5,7,1,13
999     END
1000    IF ERR/2+1=4 RESUME 50
1010    ON ERROR GOTO 0
```

Remarque : 4 est le code d'erreur "Fin de données".

INP (*port*)

Fournit la valeur du byte lu au port d'entrée/sortie dont le numéro est *port*. L'utilisation effective de INP (ainsi que de OUT) se fera généralement via l'Interface d'Extension TRS-80 à partir d'appareils périphériques adéquats. Il y a 256 ports, numérotés de 0 à 255. Par exemple,

```
100     PRINT INP(50)
```

imprime la valeur décimale du byte lu au port 50.

MEM

Fournit la quantité de mémoire non protégée qui reste disponible. Cette fonction peut être utilisée en mode Commande pour évaluer l'espace occupé par un programme; elle peut aussi contrôler, de l'intérieur d'un programme, la mémoire restante en cours d'exécution, afin d'éviter des erreurs OM (Out of Memory ou "plus de mémoire disponible"); le programme peut alors allouer moins d'espace pour les chaînes et déclarer des tableaux plus petits.
MEM ne demande pas d'argument.

Exemple :

```
100     IF MEM < 80 THEN 900
110     DIM A(15)
.
.
.
```

Vous utiliserez PRINT MEM (ou ? MEM) en mode Commande pour déterminer la quantité de mémoire non utilisée par le programme résident, les variables, les chaînes de caractères, la pile (stack), ni réservée pour les programmes binaires.

OUT *port, valeur*

Il s'agit d'une instruction commandant l'envoi de *valeur* au port d'entrée/sortie dont le numéro est *port*. Les deux arguments, séparés par une virgule, ont une valeur comprise entre 0 et 255 inclus.

Exemple :

```
OUT   255,10
```

envoie la valeur 10 au port 250.

Tout comme INP, OUT sera surtout utilisée avec l'Interface d'Extension.
Voir **INP.**

PEEK (*adresse*)

Fournit la valeur contenue dans le byte dont l'adresse est *adresse* (exprimée en décimal) en mémoire centrale. Pour déterminer les régions où réaliser cette lecture, vous consulterez l'annexe D, "Carte de la Mémoire"; quant à l'interprétation de la valeur lue, elle vous sera donnée à l'annexe C, "Table des Codes".

Si, par PEEK successifs, vous examinez un programme en binaire ("programme objet"), vous aurez besoin également des codes d'instructions du micro-processeur Z80 (Vous le trouverez dans le manuel Editor/Assembler TRS-80 ou la brochure du TBUG).

PEEK, de même que POKE, permet à un programme BASIC de communiquer avec des routines en language machine; c'est ainsi qu'il pourra obtenir des résultats rangés par les routines à des adresses déterminées dans la mémoire.
Par exemple,

```
A=PEEK(17999)
```

consulte le byte 17999 et affecte son contenu à la variable A.

PEEK peut également être utilisé pour retrouver des données rangées par POKE; vous pouvez ainsi réaliser des systèmes de stockage de caractères très compacts (cela correspond à un tableau de bytes). Consultez la Carte de la Mémoire pour déterminer l'endroit le plus approprié pour ce stockage.

Voir **POKE, USR.**

POKE *adresse, valeur*

Range la *valeur* dans le byte dont l'adresse est *adresse* (exprimée en décimal) en mémoire centrale. Il s'agit d'une instruction complète et non d'une fonction. La *valeur* doit être comprise entre 0 et 255 inclus. Consultez la Carte de la Mémoire (annexe D) pour repérer les emplacements utilisables.

Un POKE ou un PEEK d'une adresse **supérieur à 32767** réclame une petite astuce : utilisez l'*adresse* suivante : – (adresse désirée –32767) comme argument.

POKE sera également utile dans les applications graphiques. Si vous examinez la feuille de travail Vidéo (annexe E), vous constaterez que chaque position pouvant contenir un caractère est décomposé en 6 subdivisions ou segments. En réalité, à chaque position de caractère sur l'écran correspond un byte en mémoire centrale dans un espace réservé à cet usage (la Mémoire Vidéo, qui va des adresses 15360 à 16383).

La valeur de chaque byte est interprétée comme suit :
— si le premier bit est 0, les 6 derniers bits sont considérés comme le code ASCII d'un caractère; ce caractère apparaît alors sur l'écran.
— si le premier bit est 1, les 6 derniers bits sont considérés comme le code d'un caractère graphique. A chaque bit correspond l'un des 6 segments associés au byte. Si le bit est 0 le segment est éteint, s'il est à 1 le segment est allumé. Le deuxième bit n'est pas utilisé.

Puisqu'un POKE garnit un byte, donc 8 bits, cette unique instruction peut positionner 6 segments à la fois, alors que SET et RESET ne positionne qu'un seul segment. POKE peut donc être 6 fois plus rapide que SET et RESET. Vérifiez-le avec le programme suivant :

```
10      CLS
20      FOR X=15360 TO 16383
30      POKE X, 191
40      NEXT
50      GOTO 50
```

191 a pour valeur binaire 10111111 et correspond donc à l'allumage de 6 segments.

L'utilisation de POKE demande beaucoup de prudence car il est possible ainsi d'atteindre n'importe quelle partie de la mémoire RAM, et en particulier des endroits critiques où se trouve votre programme, ou pire encore les données nécessaires au fonctionnement du NIVEAU II. Vous pourriez par inadvertance provoquer une catastrophe qui vous obligerait à recommencer votre travail à zéro (réinitialiser par le bouton RESET ou même éteindre puis rallumer).

Voir **PEEK, USR, SET** et, au Chapitre 4 **CHR$**

POS (*x*)

Fournit un nombre de 0 à 63 indiquant la position courante du curseur d'affichage sur l'écran. *x* est un argument numérique fictif sans utilité (mais obligatoire pour des raisons de standardisation).

```
100     PRINT TAB(40) POS(0)
```

imprime 40 à la position 40 de la ligne courante; en fait 40 est précédé d'un espace réservé au signe et le chiffre 4 se trouve donc à la position 41. La valeur 0 de POS (0) sert d'argument fictif.

```
100     PRINT "CES" TAB(POS(0)+5) "MOTS" TAB(POS(0)+5) "SONT"
        TAB(POS(0)+5) "EQUIDISTANTS"

RUN

CES     MOTS     SONT     EQUIDISTANTS
READY
>_
```

USR (*x*)

Cette fonction appelle une sous-routine écrite en langage machine en lui "passant" la valeur de l'argument *x* (qui peut être fictif) et fournit un résultat délivré par la sous-routine.
Cette sous-routine peut avoir ètè chargée à partir d'une cassette ou créée par un programme BASIC par des POKE successifs.
L'utilisation de USR demande d'être familiarisé avec la programmation en langage machine (voir TBUG et l'Editor/Assembler TRS-80 ou tout autre manuel de programmation du Z80); en effet la moindre erreur d'utilisation de la fonction USR (tout comme POKE) peut entraîner la destruction d'informations vitales pour le fonctionnement du TRS-80 (programmes en mémoire, pile, etc).
Faites quelques exercices avant de vous lancer dans son utilisation !

Il n'y a qu'une seule fonction USR en BASIC NIVEAU II alors que le BASIC DISK NIVEAU II en offre jusqu'à 10 : de USR0 à USR9.

Exemple :

```
100    X=USR(N)
```

L'ordinateur va commencer à exécuter la sous-routine dont l'adresse aura été rangée (par deux POKE) dans les bytes 16526 et 16527. La transmission de la valeur de l'argument N (entier signé de deux bytes) peut être demandée par la sous-routine par l'appel d'une sous-routine du système localisée en 2687 (décimal). La sous-routine peut, en terminant son exécution, transmettre un résultat en retour (entier signé de deux bytes). C'est cette valeur qui est affectée à X. Si aucune valeur n'est transmise, c'est la valeur de N qui est affectée à X.
N doit être un entier compris entre – 32768 et 32767 inclus.

Pour utiliser une sous-routine en langage machine à partir du BASIC, il est nécessaire de lui réserver une zone protégée en partie haute de la mémoire (voir annexe D, "Carte de la Mémoire"). Déterminez la taille de la sous-routine. Retirez cette valeur de la plus haute adresse de la mémoire (qui dépend donc de la taille de cette mémoire : 4K, 16K, 32K ou 48K). Le résultat est l'adresse du début de l'espace à protéger. C'est cette adresse que vous renseignerez lorsqu'à l'allumage l'ordinateur vous demandera : MEMORY SIZE?
A aucun moment, l'ordinateur n'utilisera la zone protégée sans que vous ne l'y autorisiez (par POKE, PEEK et USR).

Vous chargerez alors la sous-routine en utilisant des POKE ou à partir d'une cassette par la commande SYSTEM (voir chapitre 2, **SYSTEM**). Un programme BASIC peut être chargé sans danger pour le contenu de la zone protégée.
Lorsque vous voudrez exécuter la sous-routine, utilisez une instruction contenant USR, comme, par exemple :

```
50     PRINT USR(N)
```

ou

```
50     A=USR(I%)+B
```

Il y a quelques renseignements techniques qu'il faut connaître pour mettre en œuvre les liaisons entre un programme BASIC et une telle sous-routine.

t oui acquerra la valeur de l'argument, la sous-routine doit avant tout exécuter la sous-routine d'acquisition par un CALL OA7FH (hexa) ou CALL 2687 (décimal); celle-ci convertit l'argument en entier et place la valeur dans le registre HL. Ce passage de valeurs peut également se faire par des POKE préalables à un endroit connu de la sous-routine.

Le retour au programme BASIC peut se réaliser de deux manières :
— Si vous ne désirez pas renvoyer de résultat, utilisez l'instruction RET.
— Si vous désirez renvoyer un résultat, placez celui-ci dans le registre HL (entier signé de deux bytes) et exécutez un JP OA9AH (hexa) ou JP 2714 (décimal) qui fournira comme résultat le contenu de HL comme entier signé de deux bytes.

Il reste alors à spécifier au programme BASIC l'adresse de début de la sous-routine. Vous placerez par des POKE le byte le moins significatif de l'adresse en 16526 et le byte le plus significatif en 16527.

Considérons par exemple une sous-routine dont la première instruction est en 32000, ou encore 7D00 en hexadécimal. Il faudra ranger 00 (hexa) ou 0 (décimal) en 16526 et 7D (hexa) ou 125 (décimal) en 16527. Avant l'exécution de cette sous-routine, il faudra exécuter :

```
POKE 16526,0 : POKE 16527,125
```

L'apparition d'un USR provoquera un branchement vers 32000.

Remarque : les bytes 16526-16527 contiennent initialement l'adresse de la routine de traitement de l'erreur FC (Appel de fonction invalide).

Les sous-routines USR disposent d'office d'une pile de 8 niveaux (16 bytes). Si cela vous semble insuffisant, sauvez le pointeur de pile (SP) et utilisez votre propre pile. Cependant soyez prudents lors de ces manipulations. Voir chapitre 2, **SYSTEM** ainsi que **PEEK** et **POKE** dans ce chapitre.

VARPTR (*nom de variable*)

fournit l'adresse de la *variable* ou d'un descripteur de cette *variable*. Si la variable n'a pas encore reçu de valeur, une erreur FC se produira.

VARPTR (*variable entière*) fournit l'adresse K telle que :
Le byte K est le byte le moins significatif (BMS) de la valeur de la *variable* en complément à 2).
Le byte K+1 est le byte le plus significatif (BPS) de la valeur de la *variable*.

Pour acquérir la valeur de l'argument, la sous-routine doit avant tout exécuter la sous-routine d'acquisition par un CALL OA7FH (hexa) ou CALL 2687 (décimal); celle-ci convertit l'argument en entier et place la valeur dans le registre HL. Ce passage de valeurs peut également se faire par des POKE préalables à un endroit connu de la sous-routine.

Le retour au programme BASIC peut se réaliser de deux manières :
— Si vous ne désirez pas renvoyer de résultat, utilisez l'instruction RET.
— Si vous désirez renvoyer un résultat, placez celui-ci dans le registre HL (entier signé de deux bytes) et exécutez un JP OA9AH (hexa) ou JP 2714 (décimal) qui fournira comme résultat le contenu de HL comme entier signé de deux bytes.

Il reste alors à spécifier au programme BASIC l'adresse de début de la sous-routine. Vous placerez par des POKE le byte le moins significatif de l'adresse en 16526 et le byte le plus significatif en 16527.

Considérons par exemple une sous-routine dont la première instruction est en 32000, ou encore 7D00 en hexadécimal. Il faudra ranger 00 (hexa) ou 0 (décimal) en 16526 et 7D (hexa) ou 125 (décimal) en 16527. Avant l'exécution de cette sous-routine, il faudra exécuter :

```
POKE 16526,0 : POKE 16527,125
```

L'apparition d'un USR provoquera un branchement vers 32000.

Remarque : les bytes 16526-16527 contiennent initialement l'adresse de la routine de traitement de l'erreur FC (Appel de fonction invalide).

Les sous-routines USR disposent d'office d'une pile de 8 niveaux (16 bytes). Si cela vous semble insuffisant, sauvez le pointeur de pile (SP) et utilisez votre propre pile. Cependant soyez prudents lors de ces manipulations. Voir chapitre 2, **SYSTEM** ainsi que **PEEK** et **POKE** dans ce chapitre.

VARPTR (*nom de variable*)

Fournit l'adresse de la *variable* ou d'un descripteur de cette *variable*. Si la variable n'a pas encore reçu de valeur, une erreur FC se produira.

VARPTR (*variable entière*) fournit l'adresse K telle que :
Le byte K est le byte le moins significatif (BMS) de la valeur de la *variable* (en complément à 2).
Le byte K+1 est le byte le plus significatif (BPS) de la valeur de la *variable*.

Si A!=−5, sa valeur est représentée par :

BMS	BPS suivant	BPS	Exposant
0	0	160	131

Si A!=−.5, on obtient :

0	0	128	128

Si A!=.9, il vient :

102	102	102	128

La valeur 0 est représentée conventionnellement par un exposant nul, les autres bytes étant quelconques.

Les opérateurs logiques

Nous avons vu dans le Chapitre 1 comment les opérateurs AND, OR et NOT pouvaient être utilisés pour combiner des conditions simples dans une instruction IF tel que dans l'exemple :

```
100    IF A=C AND NOT (B>40) THEN 60 ELSE 50
```

Ces opérateurs peuvent aussi servir à réaliser des manipulations de bits, des comparaisons bit à bit (par masques) et des opérations Booléennes. Nous verrons dans ce paragraphe comment réaliser de telles opérations en BASIC NIVEAU II. Nous ne tenterons pas d'expliquer l'algèbre Booléenne, l'arithmétique binaire, les conversions décimal-binaire, etc. Nous laisserons cette tâche à des ouvrages tels que **Understanding Digital Computers** de Radio·Shack (nº catal. 62-2027) qui constitue (pour qui comprend l'anglais!) une excellente initiation en cette matière.

AND, OR et NOT convertissent d'abord leurs arguments en entiers de 16 bits, signés et sous la forme de complément à 2 (−32768 à 32767). Ils réalisent alors l'opération puis fournissent le résultat sous ce même format. Si les arguments sortent de cet intervalle, une erreur "FC" se produira.

L'opération est réalisée bit à bit, c'est-à-dire qu'elle est réalisée successivement sur les 2 bits de chaque position dans les deux opérandes, ou sur chaque bit de l'unique opérande (NOT).

Les opérateurs obéissent aux tables de vérité suivantes :

Opérateur	Argument 1	Argument 2	Résultat
AND	1	1	1
	1	0	0
	0	1	0
	0	0	0

Opérateur	Argument 1	Argument 2	Résultat
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0

Opérateur	Argument	Résultat
NOT	1	0
	0	1

EXEMPLES :

(Dans ces exemples, les bits initiaux (de poids fort ou les plus significatifs) à zéro ont été ignorés.)

63 AND 16=16	63 correspond à 111111 en binaire et 16 correspond à 100000; le résultat d'un AND est donc 100000, soit 16 en décimal.
15 AND 14=14	15 correspond à 1111 en binaire et 14 correspond à 1110; le résultat est 1110, soit 14 en décimal.
–1 AND 8=8	–1 correspond à 1111111111111111 et 8 correspond à 1000; le résultat reste 1000 soit 8 en décimal.
4 AND 2=0	4 correspond à 100 et 2 à 10, de sorte que le résultat est 0, aucune position n'étant simultanément à 1 dans les deux opérandes.
4 OR 2=6	100 binaire et 10 binaire donnent après un OR 110 binaire, soit 6 en décimal.
10 OR 10=10	L'opération OR entre une valeur et elle-même fournit à nouveau cette valeur.
–1 OR –2=–1	–1 correspond à 1111111111111111 et –2 correspond à 1111111111111110; un OR donne un résultat égal à 1111111111111111, soit –1 en décimal.
NOT 0=–1	le complément de 0000000000000000 est 1111111111111111, soit –1 en décimal. De même, NOT –1=0.
NOT X	NOT X est égal à –(X+1). En effet le complément à 2 d'un nombre de 16 bits est obtenu en ajoutant 1 à son complément (à 1).
NOT 1=–2	NOT 1 vaut, en binaire 1111111111111110=–2 en binaire (=–(1+1)).

Une application classique de ces opérateurs est le test d'un ou plusieurs bits d'un byte lu à un port d'entrée du TRS-80; ces bits représentent par exemple l'état d'un appareil extérieur (L'usage de l'Interface d'Extension est ici nécessaire).

Considérons que les bits d'un byte sont numérotés comme suit : 7 pour le bit le plus à gauche (de poids fort) et 0 pour le bit plus à droite (de poids faible). Supposons que le bit 1 (l'avant dernier) du port nº 5 soit à 0 lorsque la porte d'entrée est fermée, et à 1 lorsqu'elle est ouverte. Le programme suivant détectera l'ouverture de la porte d'entrée :

```
10      IF INP(5) AND 2 THEN PRINT "ALERTE": GOTO 100
20      GOTO 10
```

Voir Chapitre 1, "Opérateurs logiques"

9 / l'Edition de Programmes

Il n'est pas rare qu'un utilisateur du NIVEAU I perde du temps à retaper de longues lignes de programme à cause d'une erreur de typographie ou d'un changement mineur. Lorsqu'une ligne a été introduite, il n'y a d'autre méthode pour la corriger que de la retaper entièrement.

Le NIVEAU II vous offre des outils qui réduiront votre travail de correction au minimum. Cette facilité de correction vous rendra plus audacieux dans l'utilisation des lignes à plusieurs instructions, des expressions complexes, etc.

Les commandes, sous-commandes et touches à fonctions spéciales décrites dans ce chapitre sont les suivantes :

EDIT	L	*n*D
ENTER	X	*n*C
n Barre d'espace	I	*n*S*c*
n ←	A	*n*K*c*
SHIFT ↑	E	
	Q	
	H	

EDIT *n° de ligne*

Cette commande met l'ordinateur en mode Editeur, dans lequel vous pouvez corriger une ligne. Vous spécifierez la ligne à corriger de l'une des façons suivantes :

EDIT *n° de ligne* **ENTER** — Vous permet de corriger la ligne *n° de ligne*. S'il n'y a pas dans votre programme de ligne correspondante, une erreur FC se produira.

EDIT. **ENTER** — Vous permet de modifier la ligne courante - il s'agit selon le cas de la dernière ligne écrite ou modifiée, ou de celle dans laquelle une erreur vient de se produire.

Introduisez (**ENTER**) la ligne suivante :

```
100    FOR I=1 TO 10 STEP .5 : PRINT I, I↑2, I↑3 : NEXT
```

Nous utiliserons cette ligne dans tout ce chapitre.

Tapez ensuite EDIT. et **ENTER** . L'ordinateur affichera :

```
100_
```

Vous êtes à présent en mode Editeur dans lequel vous pouvez commencer à corriger cette ligne 100.

La touche ENTER

Lorsque vous enfoncez la touche **ENTER** en mode Editeur, l'ordinateur enregistre toutes les modifications que vous venez de faire et retourne en mode Commande.

n Barre d'espace

Lorsque vous enfoncez la barre d'espace en mode Editeur, le curseur avance d'une position vers la droite et le caractère de la ligne à la position précédente apparaît à l'écran. Par exemple, à la suite d'un EDIT 100, l'ordinateur affiche :

```
100_
```

Tapez un espace. Le curseur avance d'une position, laissant apparaître le caractère F à la position précédente. Ce caractère est le premier de la ligne même si celui-ci est un espace; l'écran contient donc :

```
100 F_
```

A chaque frappe de la barre d'espace vous ferez apparaître le caractère suivant de la ligne à corriger. Pour accélérer cette progression faites précéder la frappe de la barre d'espace du nombre de caractères que vous voulez faire apparaître. Après l'affichage de 100 F–, tapez 5 puis frappez la barre d'espace. Vous obtiendrez :

```
100     FOR I=_
```

Tapez ensuite 8 puis un espace. Le curseur avancera de 8 positions et les 8 caractères suivants s'afficheront.

n ← (touche de retour en arrière)

Fait reculer le curseur de *n* positions vers la gauche. Si *n* n'est pas spécifié, le curseur recule d'une position. Pendant cette régression, les caractères à droite du curseur sont effacés sur l'écran mais **ils ne sont pas détruits dans la ligne de programme.** Supposons que sur l'écran soit affiché :

```
100     FOR I=1 TO 10_
```

Tapez 7 ←. Vous verrez alors apparaître à l'écran, à la place de la ligne précédente :

```
100     FOR I=_
```

(le résultat dépend évidemment du nombre d'espaces introduits dans cette ligne).

SHIFT ↑

Frappez simultanément les touches SHIFT et ↑ pour terminer l'une des sous-commandes d'insertion X, I ou H. L'ordinateur se retrouve alors en mode Editeur et le curseur conserve sa position courante. Enfoncer **ENTER** est une autre manière de sortir à la fois de la sous-commande et du mode Editeur.

L (List Line)

[en français : LISTING DE LA LIGNE]

Lorsqu'en mode Editeur, et en dehors de l'exécution d'une sous-commande X, I ou H, vous tapez L, le reste de la ligne s'affiche à l'écran; en outre, à la ligne d'écran suivante apparaît à nouveau le numéro de la ligne de programme, suivi du curseur. Supposons que vous en soyez au stade suivant :

```
100_
```

Tapez L (sans **ENTER**). Le reste de la ligne est affiché :

```
100     FOR I=1 TO 10 STEP.5 : PRINT I, I↑2, I↑3 : NEXT
100_
```

Vous pouvez ainsi examiner la ligne dans son état actuel tout en poursuivant sa correction.

X (End of Line and Insert)

[en français : INSERTION EN FIN DE LIGNE]

Le reste de la ligne est affiché et le curseur est déplacé jusqu'à la fin de la ligne. L'ordinateur vous permet alors d'insérer (mode Insertion) des caractères supplémentaire à la fin de cette ligne. Si, par exemple, vous en êtes au stade :

```
100_
```

et que vous tapiez X (sans **ENTER**), la ligne complète apparaît, suivie du curseur :

```
100     FOR I=1 TO 10 STEP.5 : PRINT I, I↑2, I↑3 : NEXT_
```

Il est alors possible d'ajouter une instruction à cette ligne ou de supprimer des caractères à l'aide de la touche ←. Tapez par exemple : PRINT "FIN", puis **ENTER** pour sortir du mode Editeur. Un LIST 100 vous montrera la nouvelle version de la ligne :

```
100     FOR I=1 TO 10 STEP.5 : PRINT I, I↑2, I↑3 : NEXT : PRINT "FIN"
```

I (Insert)

[en français : INSERTION]

Vous permet d'insérer des caractères à la position courante du curseur.
L'usage de la touche ← dans ce mode vous permet de supprimer des caractères.
Supposons que dans l'édition de notre ligne 100, vous en soyez au stade suivant :

```
100     FOR I=1 TO 10 STEP._
```

Supposons encore que vous désiriez changer l'incrément .5 en .25, c'est-à-dire insérer un 2 entre le . et le 5. Tapez I (sans **ENTER**).
Vous êtes alors (ou du moins l'ordinateur) en mode Insertion. Tapez 2, le caractère à insérer, pour obtenir :

```
100     FOR I=1 TO 10 STEP.2_
```

Si vous voulez terminer là la modification, vous devez sortir du mode Insertion en poussant simultanément sur les touches SHIFT↑. Tapez ensuite L pour examiner l'état actuel de la ligne :

```
100     FOR I=1 TO 10 STEP.25 : PRINT I, I↑2, I↑3 : NEXT : PRINT "FIN"
100_
```

Vous pouvez aussi sortir non seulement du mode Insertion mais aussi du mode Editeur par un **ENTER** , sauvant ainsi les modifications dans le programme.

A (Cancel and Restart)

[en français : REPRENDRE A ZERO]

Les modifications réalisées dans la ligne sont supprimées et le curseur est repositionné au début de la ligne. Par exemple, si après avoir réalisé des insertions, modifications ou suppressions, vous désirez supprimer l'effet de ces changements, tapez d'abord SHIFT↑ pour sortir du mode Insertion éventuel, puis tapez A. Le numéro de la ligne en cours de modification apparaît alors à la ligne d'écran suivante, suivi du curseur.

E (Save Changes and Exit)

[en français : SAUVER ET SORTIR]

L'ordinateur sauve les modifications réalisées sur la ligne courante puis sort du mode Editeur. Il faut au préalable vous assurer que vous n'êtes plus en mode Insertion (X,I,H); tapez donc SHIFT↑ avant E.

Q (Cancel and Exit)

[en français : SORTIR SANS SAUVER]

L'ordinateur sort du mode Editeur mais sans sauver les modifications. La ligne de programme n'est donc pas altérée par ces modifications.
Vous devez auparavant sortir du mode Insertion (X,I,H) éventuel par SHIFT↑.

H (Hack and INSERT)

[en français : TRONQUER ET INSERER]

Permet de supprimer les caractères à partir de la position du curseur jusqu'à la fin de la ligne puis de les remplacer par des caractères à insérer (l'ordinateur est alors en mode Insertion). En tapant ←, vous pouvez également supprimer des caractères. Reprenons la ligne 100 telle qu'elle a été modifiée dans le paragraphe X, et positionnons le curseur au début de l'instruction PRINT "FIN" :

```
100     FOR I=1 TO 10 STEP.5 : PRINT I, I↑2, I↑3 : NEXT :_
```

A présent, tapez H, puis END et poussez sur **ENTER**. Un LIST 100 affichera :

```
100     FOR I=1 TO 10 STEP.5 : PRINT I, I↑2, I↑3 : NEXT : END.
```

Pour sortir de ce mode Insertion mais rester en mode Editeur, tapez simultanément SHIFT ↑.

n D (Delete)

[en français : SUPPRIMER]

Permet de supprimer *n* caractères à partir de la position courante du curseur. L'ordinateur vous montre, entourés de points d'exclamation, les caractères supprimés. A titre d'exemple, supprimons l'instruction PRINT dans notre ligne 100. Positionnons le curseur sur le premier caractère de l'instruction :

```
100     FOR I=1 TO 10 : STEP.5 :_
```

Tapez ensuite 16D pour indiquer à l'ordinateur que vous désirez supprimer 16 caractères à partir de la position du curseur :

```
100     FOR I=1 TO 10 : STEP:5 : ! PRINT I, I↑2, I↑3 :!_
```

Vérifiez par un LIST que l'instruction PRINT a bien disparu.

n C (Change)

Indique à l'ordinateur que vous désirez changer les *n* caractères à partir de la position courante du curseur.
L'absence de *n* signifie que vous ne désirez modifier qu'un seul caractère.
Les *n* caractères introduits à la suite de cette commande vont se substituer aux anciens caractères, puis l'ordinateur se retrouve automatiquement en mode Editeur (il quitte donc la sous-commande C). Changeons par exemple la valeur limite 10 en 25 dans la boucle FOR de la ligne 100. Positionnons le curseur sur le premier des caractères à changer :

```
100     FOR I=1 TO _
```

Tapez 2C puis les deux nouveaux caractères 25. Faites afficher la ligne pour vérifier que les caractères ont été changés :

```
100     FOR I=1 TO 25 STEP.5 : NEXT : END
```

n S *c* (Search)

[en français : RECHERCHER]

Demande à l'ordinateur de rechercher la $n^{ème}$ occurrence du caractère *c* et d'arrêter le curseur à sa position. Si *n* n'est pas spécifié, l'ordinateur recherche la première occurrence de *c*. Si ce caractère n'a pas été trouvé, le curseur est placé à la fin de la ligne. La recherche s'effectue à partir de la position courante du curseur (les caractères à sa gauche ne sont pas examinés). Reprenons la ligne 100 et positionnons le curseur en son début. Tapez 2S: pour demander à l'ordinateur de positionner le curseur sur la deuxième occurrence du caractère ":". Le résultat sera le suivant :

```
100     FOR I=1 TO 25 STEP.5 : NEXT_
```

Vous pouvez à présent faire exécuter toute autre sous-commande. Supposons que vous désiriez ajouter le nom de la variable compteur après NEXT. Tapez I pour indiquer que vous avez l'intention d'insérer des caractères. Vous tapez ensuite ces derniers : *espace* puis I, puis vous sortez du mode Insertion par SHIFT↑. Faites afficher la ligne à titre de vérification :

```
100     FOR I=1 TO 25 STEP.5 : NEXT I : END
```

n K *c* (Search and Kill)

[en français : RECHERCHER ET SUPPRIMER]

Demande à l'ordinateur de supprimer tous les caractères depuis celui de la position courante du curseur jusqu'à la $n^{ème}$ occurrence du caractère *c* (celui-ci non compris). Supprimons par exemple la boucle de la ligne 100 dans l'état où elle se trouve. Positionnons le curseur au début de cette ligne et tapons 2K : pour demander la suppression des caractères jusque, et non compris, la seconde occurence de ":" :

```
100     !FOR I=1 TO 25 STEP.5 : NEXT I !_
```

Il faut encore supprimer le caractère ":" sous le curseur en tapant D :

```
100     !FOR I=1 TO 25 STEP.5 : NEXT I I I : !_
```

Vérifiez le résultat en tapant **ENTER** , puis LIST 100 **ENTER** :

```
100     END
```

10 / L'Interface d'Extension

Il est possible d'adjoindre un Interface d'Extension à tout ordinateur TRS-80 NIVEAU II. Cet interface permet l'emploi d'appareils d'entrée/sortie supplémentaires; il permet également d'ajouter de la mémoire RAM. L'interface va permettre quatre extensions principales du TRS-80:

1. Un lecteur de cassette supplémentaire.
2. L'imprimante TRS-80.
3. De un à quatre Mini-Disques.
4. Jusqu'à 32K de mémoire RAM, portant la mémoire totale à 48K.

Tous ces appareils sont disponibles dans les magasins Radio-Shack. Consultez les instructions de montage pour raccorder l'Interface d'Extension et les différents appareils à l'ordinateur TRS-80.

Lorsque l'Interface d'Extension est branché au TRS-80, celui-ci suppose qu'un Mini-Disque est disponible. Ce Mini-Disque met à votre disposition des commandes et des instructions supplémentaires dont la liste sera donnée dans ce chapitre. Même si vous n'avez pas connecté de Mini-Disque, l'ordinateur supposera qu'il y en a un (à cause du Contrôleur de Disque de l'Interface) et tentera d'y lire des informations spéciales. Dès lors, si vous n'avez pas de Mini-Disque, il faut éviter de laisser l'ordinateur se mettre en mode Mini-Disque et lui permettre un fonctionnement normal en NIVEAU II. Pour ce faire, maintenez enfoncée la touche BREAK pendant l'allumage du bloc clavier du TRS-80. Vous agirez de même lorsque vous devrez enfoncer le bouton RESET en cas de blocage de l'ordinateur. Attention cependant, un tel RESET vous renvoie à l'étape MEMORY SIZE et non à l'étape READY.

Utilisation d'un second enregistreur à cassette

L'usage de deux enregisteurs vous permet de modifier des données sur bande avec une efficacité beaucoup plus grande qu'avec un seul enregistreur. Par exemple, un fichier d'adresses stocké sur bande peut être lu, une ligne à la fois, modifié, puis recopié, une ligne à la fois, sur le second enregistreur.
La routine correspondante pourrait ressembler à ceci :

```
5       DEFSTR A–Z
10      INPUT #–1,A,B,C,D : IF A="FIN" PRINT #–2,A,B,C,D : END
20      PRINT A,B,C,D
30      INPUT "VOULEZ-VOUS CORRIGER (0/N)."; R
40      IF R="0" INPUT A,B,C,D
50      PRINT #–2,A,B,C,D
60      GOTO 10
```

Il s'agit d'une application très élémentaire. Il est cependant possible d'envisager des programmes très puissants qui mettent en œuvre des entrées/sorties à partir de deux enregistreurs.

Vous aurez remarqué la façon dont on spécifie le numéro de l'enregistreur; une méthode plus souple encore est d'utiliser une variable comme dans

I=2 : INPUT #–I,A,B,C,D qui est équivalent à INPUT #–2,A,B,C,D

L'utilisation de deux enregistreurs peut également être étendue aux commandes CLOAD, CSAVE et CLOAD? On écrira par exemple:

CLOAD#–1, "A" (ou CLOAD "A")

pour charger un programme à partir du premier enregistreur, puis

CSAVE #–2, "A"

pour le recopier sur le second enregistreur. La vérification se commandera par :

CLOAD#–2,? "A"

Voir Chapitre 2, **CLOAD, CLOAD?, CSAVE.**

L'imprimante (Line printer)

Une imprimante vous permet de conserver une trace écrite des informations créée par votre TRS-80. Différentes instructions et commandes vous permettent d'écrire sur l'imprimante.

LLIST

Le fonctionnement est le même que celui de LIST avec une sortie vers l'imprimante et non vers l'écran.

LLIST	Ecrit sur l'imprimante le programme résident.
LLIST 100-	Ecrit la ligne 100 et les suivantes sur l'imprimante.
LLIST 100-200	Ecrit les lignes 100 à 200 sur l'imprimante.
LLIST .	Ecrit la ligne courante sur l'imprimante.
LLIST -100	Ecrit les premières lignes du programme jusque la ligne 100 sur l'imprimante.

Voir Chapitre 2, **LIST.**

LPRINT

Cette instruction (qui est aussi une commande) vous permet d'écrire des informations sur l'imprimante. Par exemple, LPRINT A va y imprimer la valeur de A. LPRINT utilise toutes les options de PRINT, **sauf PRINT @**

Exemples :

LPRINT *variable ou expression* imprime les valeurs de *variable* ou *expression* sur l'imprimante.

LPRINT USING réalise une impression selon le format spécifié.

LPRINT TAB déplace la tête d'impression de l'imprimante vers la droite comme spécifié dans le paramètre TAB; attention cependant, cette position ne peut dépasser 63 comme sur l'écran,

Voir Chapitre 3, **PRINT.**

Codes de l'imprimante

Certains codes de commande peuvent servir à actionner l'imprimante. Un code sera envoyé à l'imprimante grâce à LPRINT et CHR$. Par exemple,

```
PRINT CHR$ (10)
```

commande le passage à la ligne suivante.

CODE	FONCTION
10	Passage à la ligne suivante (carriage return/line feed)
11	Espacement de ligne (carriage return/line feed)
12	Saut à la page suivante (Form feed)
13	Retour de la tête en début de ligne (carriage return)

Remarque : à moins qu'elles ne se terminent par; un code de passage à la ligne suivante est toujours transmis à la fin des données d'un LPRINT.

L'imprimante écrit à raison de 6 lignes par pouce et 66 lignes par page.
Vous pouvez cependant modifier la taille de la page en introduisant le nouveau nombre de lignes par page dans le byte 16424 à l'aide d'un POKE.

Exemple :

```
POKE 16424, 40
```

définit des pages de 40 lignes.

Les Mini-Disques – (BASIC DISK NIVEAU II)

Le système Mini-Disk TRS-80 utilise une version réduite du disque souple appelée mini-disquette (ou encore, en anglais "mini-floppy"). Le disque offre un espace de stockage de gros volumes de données ainsi qu'un temps d'accès beaucoup plus court que celui d'une cassette. Le premier disque peut contenir 55K bytes de programmes et de fichiers, tandis que chaque disque supplémentaire dispose d'un espace de 89.600 bytes. Le système Mini-Disk dispose de son propre jeu de commandes qui permet de manipuler des fichiers et de les utiliser dans des programmes. Ce système offre l'accès séquentiel et l'accès direct aux fichiers. Il complète le BASIC NIVEAU II de nouvelles instructions et fonctions :

Instructions et commandes :

CLOSE	LSET	PUT
FIELD	NAME	RSET
GET	OPEN	MERGE
KILL	PRINT # *nom fichier*	LOAD
DISKDUMP	LINE INPUT	SAVE
EOF	INPUT # *nom fichier*	LOF

Fonctions d'entrée/sortie

CVD	MKD$
CVI	MKI$
CVS	MKS$

Ajouts au NIVEAU II

Dix appels USR (USR0 à USR9)

Constantes octales &0

Constantes hexadécimales &H

DEFUSR (adresses des USR)

LINE INPUT

MID$ (à gauche du signe =)

INSTR (réalise la fonction de la sous-routine INSTRING du Chapitre 4)

TIME$ (date et horloge temps réel de 24 h)

DEF FN (définition de fonctions par le programmeur)

Vous trouverez les explications relatives à ces commandes dans le manuel d'utilisation du système Disk TRS-80.

Extension de la mémoire RAM

L'Interface d'Extension TRS-80 peut accueillir de la mémoire RAM supplémentaire. En ajoutant des circuits intégrés de RAM il est possible d'augmenter la mémoire de 32768 bytes. Pour tout renseignement et pour l'installation de ces circuits, consultez votre vendeur Radio Shack.

11 / Economisez le temps et l'espace

La plupart des programmes en NIVEAU II sont plus rapides et occupent moins de place en mémoire que leur équivalent en NIVEAU I. Cependant, outre cette plus grande efficacité propre au NIVEAU II, quelques règles vous permettront d'améliorer encore les performances de vos programmes.

Comment diminuer l'espace occupé

1) Lorsque votre programme fonctionne correctement, supprimez les commentaires (REM) qui vous semblent inutiles dans la version opérationnelle du programme.

2) Evitez les espaces inutiles entre les instructions, les opérateurs, etc.

3) Chaque nouvelle ligne vous coûte 5 bytes au moins. Utilisez donc des lignes de plusieurs instructions lorsque c'est possible.

4) Une variable entière occupe 5 bytes, une variable en simple précision 7 et une variable en double précision 11. Utilisez donc des variables entières partout où la chose est possible comme dans certaines boucles :

```
FOR I%=1 TO 10
```

5) Lorsque des opérations sont exécutées à plusieurs endroits différents, faites-en une sous-routine. Si une sous-routine n'est appelée que d'un seul endroit, remplacez le GOSUB et le RETURN par des GOTO. En effet, au moment de l'exécution, chaque GOSUB actif consomme 6 bytes tandis qu'un GOTO n'en consomme pas.

6) Utilisez aussi peu de parenthèses que possible dans vos expression (voir Chapitre 1, "Opérateurs arithmétiques"). Le traitement des parenthèses prend 4 bytes; en outre, les opérations à l'intérieur des parenthèses étant effectuées en premier, le résultat intermédiaire de chaque expression entre parenthèses doit être stocké (chacun occupe 12 bytes).

7) Dimensionnez vos tableaux avec parcimonie. Chaque tableau utilisé mais non déclaré se voit réserver 11 éléments pour chaque dimension même si cet espace n'est pas utilisé. Utilisez les éléments d'indice 0 puisqu'ils sont toujours disponibles.

8) Utilisez des instructions DEF lorsque vous travaillez avec des valeurs autres qu'en simple précision (chaînes alphanumériques, entières, double précision). Une instruction DEF prend 6 bytes; il seront rapidement récupérés par l'absence de caractère de déclaration associé aux noms des variables.

Comment augmenter la vitesse d'exécution

La vitesse d'exécution d'un programme dépend de la complexité et du nombre des instructions qui sont exécutées. Dans la plupart des programmes, même parmi les plus complexes, la vitesse d'exécution n'est pas un problème. Que le programme réponde en 0.5 ou en 0.1 seconde ne constitue guère une différence fondamentale pour la majorité des utilisateurs. Cependant, lorsque le nombre d'opérations à effectuer augmente, la vitesse d'exécution peut devenir un facteur important.

Voici quelques suggestions qui vous aideront à concevoir des programmes plus rapides.

1) Supprimez les lignes inutiles lors de l'exécution (REM par exemple).

2) Groupez plusieurs instructions en une seule ligne lorsque la chose est possible.

3) Utilisez des variables plutôt que des constantes dans les opérations (**ceci est très important**). Votre TRS-80 travaille normalement avec des valeurs représentées en virgule flottante (mantisse et exposant). Accéder à une variable est beaucoup plus rapide que convertir une constante en représentation en virgule flottante. Si vous utilisez la valeur de π plusieurs fois dans un programme, définissez π comme une variable (PI=3.14159) que vous utiliserez dans les opérations.

4) En mode graphique, utilisez POKE. Vous pourriez dans certains cas accélérer l'affichage graphique d'un facteur 6.

5) Définissez d'abord les variables qui seront utilisées le plus fréquemment. Quand une variable est définie (la première fois qu'elle est rencontrée dans une instruction à exécuter), elle est placée à la fin de la table des variables déjà connues. Les premières variables définies sont donc en tête de la table. Lors d'un accès à une variable, la table est parcourue de haut en bas jusqu'à ce que cette variable soit trouvée. Les premières variables sont donc celles qui sont retrouvées le plus rapidement. Définissez une variable en lui affectant une valeur quelconque de manière à ce qu'elle soit connue de l'ordinateur.

6) Utilisez des variables entières, en particulier dans les boucles FOR-NEXT, chaque fois que c'est possible. **Ceci est notre conseil le plus important.**

Annexes

A / Résumé du NIVEAU II

Caractères spéciaux et abréviations

Mode Commande

ENTER	Passage à la ligne logique suivante et interprétation de la commande.
←	Recul du curseur et suppression du dernier caractère introduit.
SHIFT ←	Retour du curseur au début de la ligne logique; effacement de celle-ci.
↓	Passage à la ligne d'écran suivante.
:	Séparateur d'instructions dans une même ligne
→	Avance du curseur à la position de tabulation suivante. Les positions d'arrêt sont les suivantes : 0, 8, 16, 24, 32, 40, 48, 56.
SHIFT →	Conversion de l'affichage en 32 caractères par ligne.
CLEAR	Effacement de l'écran et conversion en 64 caractères par ligne.

Mode Exécution

SHIFT @	Suspension de l'exécution; suspension de l'affichage pendant un LIST
BREAK	Arrêt de l'exécution
ENTER	Envoi des données introduites au clavier et demandées par INPUT

Abréviations

?	Mis pour PRINT
'	Remplace : REM
.	Désigne la ligne courante dans LIST, EDIT,...

Caractères de déclaration de type

Caractère	Type	Exemples
$	alphanumérique (chaîne)	A$, ZZ$
%	entier	A1%, SOMME%
!	simple précision	B!, NI!
#	double précision	A#, 1/3#
D	double précision (notation exponentielle)	1.23456789D-12
E	simple précision (notation exponentielle)	1.23456E+30

Opérateurs arithmétiques

+ addition	− soustraction	* multiplication
/ division	↑ puissance (p.ex., 2 ↑ 3 = 8)	

Opérateur alphanumérique

+ concaténation (assemblage de chaînes)	"AB"+"CD"="ABCD"

Opérateurs de comparaison

Symbole	signification numérique	alphanumérique
<	inférieur à	précède
>	supérieur à	suit
=	égal à	est égal à
<= ou =<	inférieur ou égal à	précède ou est égal à
>= ou =>	supérieur ou égal à	suit ou est égal à
<> ou ><	différent de	est différent de

Ordre d'exécution des opérations
(les opérateurs sur une même ligne ont même priorité)

↑

–

*, /

+, –

Opérateurs de comparaison

NOT

AND

OR

Commandes

Commande	Fonction	Exemples
AUTO *mm, nn*	Enclenche la numérotation automatique des lignes à partir de *mm*, par incrément de *nn*	AUTO AUTO 10 AUTO 5,5 AUTO .,10
CLEAR	Réinitialise les variables numériques à zéro et les variables alphanumériques à vide.	CLEAR
CLEAR *n*	Même fonction que CLEAR avec réservation de *n* bytes pour les chaînes	CLEAR 500 CLEAR MEM/4
CONT	Fait reprendre une exécution arrêtée par un BREAK	CONT
DELETE *mm-nn*	Supprime les lignes *mm* à *nn* dans le programme	DELETE 100 DELETE 10-50 DELETE.
EDIT *mm*	Entrée dans le mode Editeur pour corriger la ligne *mm* (voir ci-dessous)	EDIT 100 EDIT.
LIST *mm-nn*	Affiche les lignes *mm* à *nn* du programme	LIST LIST 30-60 LIST 30- LIST -60 LIST .

NEW	Effacement de programme résident, réinitialisation des variables, pointeurs, etc.	NEW
RUN *mm*	Exécuter le programme à partir de la ligne *mm* ou de la ligne dont le numéro est le plus petit.	RUN 55 RUN
SYSTEM	Entrée dans le mode moniteur pour charger des programmes en langage machine écrits sur cassette.	voir Chapitre 2
TROFF	Arrêt de la fonction Trace	TROFF
TRON	Enclenchement de la fonction Trace	TRON

Sous-commandes et touche de fonction du mode Editeur

Sous-commande/ touche de fonction	Fonction
ENTER	Fin des corrections et retour au mode Commande.
SHIFT ↑	Fin du mode Insertion; reste en mode Editeur.
n barre d'espace	Avance du curseur de *n* positions vers la droite.
n ←	Recul du curseur de *n* positions vers la gauche.
L	Affichage du reste de la ligne et retour en début de ligne.
X	Affichage du reste de la ligne, positionnement du curseur en fin de ligne et entrée en mode Insertion.
I	Entrée en mode Insertion; tout caractère tapé est inséré à la position du curseur; sortie par SHIFT ↑
A	Suppression de l'effet des corrections et retour du curseur en début de ligne.
E	Fin des corrections et retour au mode Commande.
Q	Suppression de l'effet des corrections et retour au mode Commande.

H	Suppression du reste de la ligne et entrée en mode Insertion; tapez SHIFT ↑ pour en sortir.
*n*D	Suppression de *n* caractères à partir de la position courante du curseur.
*n*C	Remplacement des *n* caractères à partir de la position du curseur par les *n* caractères qui vont être tapés.
*n*S*c*	Positionnement du curseur sur la $n^{ième}$ occurrence du caractère *c* à partir de la position du curseur.
*n*K*c*	Suppression des caractères entre la position du curseur et la $n^{ième}$ occurrence du caractère *c*.

Instructions d'Entrée/Sortie

Instruction*	Fonction	Exemples
PRINT *exp*	Afficher à l'écran la valeur de *exp* *Exp* peut être une constante ou une expression numérique ou alphabétique, ou une liste de tels éléments.	PRINT A$ PRINT X+3 PRINT "D=";D
,	Les virgules modifient un PRINT en avançant le curseur jusqu'à la zone d'impression suivante. A la fin de *exp* supprime le passage à la ligne suivante.	PRINT 1,2,3,4 PRINT "1", "2", PRINT 1,,2
;	Les point-virgules modifient un PRINT en insérant un espace après une valeur numérique dans *exp* (mais pas après une valeur numérique). A la fin de *exp* supprime le passage à la ligne suivante.	PRINT X;"=REPONSE" PRINT X;Y;Z PRINT "REPONSE=";

* *exp* peut être une constante, une variable, un élément de tableau, une expression en général, ou une liste de ces éléments.

PRINT @ *n*	Modificateur de PRINT; commence l'impression à l'adresse *n* de l'écran.	PRINT @ 540, "CENTRE" PRINT@ N+3, X*3
TAB *n*	Modificateur de PRINT : avance le curseur jusqu'à la position *n* de la ligne courante (*n* est une expression)	PRINT TAB(N) N
PRINT USING *chaîne*; *exp*	Spécificateur de format d'impression = *exp* est imprimée selon le format *chaîne* (voir ci-dessous)	PRINT USING A$;X PRINT USING "#.#"; Y+Z
INPUT *"message"*; *variable*	Imprimer le *message* (s'il y en a un) et attend des données du clavier.	INPUT "NOM="; A$ INPUT "VALEUR"; X INPUT "DEUX NOMBRES"; X,Y INPUT A,B,C,D$
PRINT #–1	Ecriture sur la cassette nº 1	PRINT #–1,A,B,C,D$
INPUT #–1	Lire sur la cassette nº 1	INPUT #–1,A,B,C,D$
DATA *liste valeurs*	Ligne contenant des données sous forme d'une *liste* de *valeurs*	DATA 22,33,11,1.2345 DATA "HENRI", 74, "JEAN", 16
READ *liste variables*	Affecte aux *variables* de la *liste* les valeurs suivantes à partir de la valeur courante des lignes DATA	READ A,A1,A2,A3 READ A$,B$,C$,D
RESTORE	Repositionne le pointeur courant au début des lignes DATA.	RESTORE

Spécificateurs de zone des instructions PRINT USING

Caractère numérique	Fonction	Exemple
#	Zone numérique (un chiffre par #).	# # #
.	Position du point décimal	# #.# # #
+	Impression du signe en début ou en fin de zone (signe + et signe –).	+ #.# # # #.# # #+
–	Impression du signe – en fin de zone si le nombre est négatif.	# #.#–
**	Remplacement des zéros initiaux par des astérisques	**# #.#
$$	Imprime un $ à la gauche du premier chiffre significatif.	$$# # # #.#
**$	Cfr. $$ avec remplissage des positions de gauche inoccupées par des astérisques.	**$# #.# #
↑↑↑↑	Format exponentiel, le premier # correspondant à un zéro.	#.# # # ↑↑↑↑ # # #.# ↑↑↑↑ .# # # ↑↑↑↑

Caractère alphanumérique	Fonction	Exemple
!	Un seul caractère.	!
% *espaces* %	Chaîne de *n* caractères où *n*=2 + nombre d'*espaces*	%% % %

Instructions

Instruction	Fonction	Exemples
Définition de type		
DEFDBL *liste de lettres ou d'intervalles*	Définit de type double précision toutes les variables dont le nom commence par une lettre appartenant à la *liste de lettres ou d'intervalles.*	DEFDBL J DEFDBL X,Y,A DEFDBL A-E, J-L, X
DEFINT *liste de lettres ou d'intervalles*	Définit de type entier toutes les variables dont le nom commence par une lettre appartenant à la *liste de lettres ou d'intervalles.*	DEFINT A DEFINT C, G, T DEFINT F, G-T, B
DEFSNG *liste de lettres ou d'intervalles*	Définit de type simple précision toutes les variables dont le nom commence par une lettre appartenant à la *liste de lettres ou d'intervalles.*	DEFSNG F, O DEFSNG Z, V-X DEFSNG A-C, I-K
DEFSTR *liste de lettres ou d'intervalles*	Définit de type alpanumérique toutes les variables dont le nom commence par une lettre appartenant à la *liste de lettres ou d'intervalles.*	DEFSTR A, B, C DEFSTR S, V-Y DEFSTR C-E, G-K
Assignation et allocation		
CLEAR *n*	Réinitialise les variables et réserve *n* bytes pour les valeurs alphanumériques.	CLEAR 750 CLEAR MEM/10 CLEAR 0
DIM *tab* (*dim1*, ..., *dim k*)	Alloue de l'espace pour un tableau *tab* de *k* dimensions ayant successivement une taille de *dim,1* ..., *dim k*. DIM peut déclarer plusieurs tableaux.	DIM A(2,3) DIM A1(15), A2(15) DIM B(X+2), C(J,K) DIM T(3,3,5)

Instruction	Fonction	Exemples
LET *variable*=*expression*	Assigne la valeur de l'*expression* à la *variable*. LET est facultatif en BASIC NIVEAU II.	LET A$="HENRI" LET B1=C1 LET A%=I#

Enchaînement des instructions

END	Termine l'exécution et provoque le retour en mode commande.	99 END
STOP	Arrête l'exécution et imprime le message BREAK suivi du numéro de la ligne courante. L'utilisateur peut faire reprendre l'exécution par CONT.	100 STOP
GOTO *nº ligne*	Provoque un branchement à la ligne *nº ligne*.	GOTO 100
GOSUB *nº ligne*	Provoque l'exécution de la sous routine commençant à la ligne *nº ligne*.	GOSUB 300
RETURN	Provoque une branchement à l'instruction qui suit le dernier GOSUB exécuté.	RETURN
ON *exp* GOTO *nl 1, ..., nl k*	L'expression *exp* est évaluée; si INT (*exp*) est égal à l'un des nombres de *1* à *k*, il y aura branchement vers la ligne approprieé, sinon il y aura passage à l'instruction suivante.	ON K+1 GOTO 100,200,300
ON *exp* GOSUB *nl 1, ..., nl k*	Même fonction que ON GOTO avec exécution de la sous-routine *nl 1, ..., nl k* selon la valeur de *exp*.	ON J GOSUB 400, 500

Instruction	Fonction	Exemples
FOR *var*=*exp 1* TO *exp 2* STEP *exp 3*	Ouvre une boucle FOR-NEXT. STEP est facultatif; son absence correspond à un pas de 1 (voir Chapitre 4).	FOR I=1 TO 50 STEP 1.5 FOR M%=J% TO K–1 %
NEXT *var*	Clôture une boucle FOR-NEXT. *var* peut être omis ou être remplacé par une liste de noms de variable (voir Chapitre 4).	NEXT NEXT I NEXT I,J,K
ERROR (*code*)	Simule l'erreur spécifiée par *code* (voir table des codes d'erreur).	ERROR (14)
ON ERROR GOTO *nº ligne*	Si une erreur survient dans la suite du programme, elle provoquera un branchement à la routine d'erreur commençant à la ligne *nº ligne*.	ON ERROR GOTO 999
RESUME *n*	Retour à la ligne *n* après exécution d'une routine de traitement d'erreur; si *n* est 0 ou est absent, le retour se fait à l'instruction ayant entrainé l'erreur; si *n* est "NEXT", le retour se fait à l'instruction qui suit celle qui a entraîné l'erreur.	RESUME RESUME 0 RESUME 100 RESUME NEXT
RANDOM	Réinitialise le générateur de nombres aléatoires.	RANDOM
REM	Spécifie un commentaire et fait ignorer le reste de la ligne.	REM A=ALTITUDE DU POINT

Instruction	Fonction	Exemples
Instruction conditionnelle		
IF *exp* THEN *instr 1* ELSE *instr 2*	*exp* est évaluée : si *exp* est VRAI, les instructions *instr 1* sont exécutées puis l'exécution se poursuit à la ligne suivante (sauf si *instr 1* se termine par un GOTO). Si *exp* est FAUX, les instructions *instr 2* sont exécutées.	IF A=0 THEN PRINT "OUI" ELSE PRINT "NON"
Instructions graphiques		
CLS	Efface l'écran.	CLS
RESET (*x*, *y*)	Eteint le segment graphique de coordonnée horizontale *x* et de coordonnée verticale *y*. 0 <=*x* <128 et 0 <=*y* <48	RESET (8+B,11)
SET (*x*, *y*)	Allume le segment graphique de coordonnées *x* et *y*. Mêmes limites que pour RESET.	SET (A*2,B+C)
Instructions spéciales		
POKE *adresse*, *valeur*	Range la *valeur* dans le byte *d'adresse* spécifiée. Les arguments sont exprimés sous forme décimale et 0 <=*valeur* <255	POKE 15635,34 POKE I+1, 3*K
OUT *port*, *valeur*	Envoie la *valeur* au *port* de numéro spécifié. Les arguments sont tous deux compris entre 0 et 255.	OUT 255,10 OUT I, J

Fonctions alphanumériques

Fonction*	Action	Exemples
ASC (*chaîne*)	Fournit le code ASCII du premier caractère de la *chaîne*.	ASC(B$) ASC("H")
CHR$ (*exp*)	Fournit le caractère correspondant au code *exp*. (voir table des Codes).	CHR$(34) CHR$ (I+1)
FRE (*chaîne*)	Fournit la quantité de mémoire disponible pour les chaînes. *Chaîne* est un argument fictif.	FRE(A$)
INKEY$	Lit le clavier et fournit une chaîne constituée du caractère correspondant à la touche frappée. Cette chaîne est vide si aucune touche n'a été frappée.	INKEY$
LEN (*chaîne*)	Fournit la longueur de la *chaîne*. (Zéro si celle-ci est vide).	LEN (A$+B$) LEN ("LONGUEUR")
LEFT$ (*chaîne, n*)	Fournit les *n* premiers caractères de la *chaîne*.	LEFT$(A$,1) LEFT$(A$+B$,2*I)
MID$ (*chaîne, p, n*)	Fournit les *n* caractères de la *chaîne* commençant à la position *p*.	MID$(M$,5,2) MID$(A$+B$,P,L-1)
RIGHT$ (*chaîne, n*)	Fournit les *n* derniers caractères de la *chaîne*.	RIGHT$(NA$,7) RIGHT$(B$+C$,N-1)
STR$ (*exp*)	Fournit la représentation en caractères de la valeur numérique de *exp*.	STR$(1.2345) STR$(A+2*B)
STRING$ (*n, car*)	Fournit une chaîne de *n* fois le premier caractère de *car*	STRING$(32,"#") STRING$(K,A$)
VAL (*chaîne*)	Fournit la valeur numérique que représente la *chaîne*.	VAL ("1"+A$+"."+B$) VAL ("3.1416")

*Chaîne peut être une constante, une variable ou une expression.

Fonctions mathématiques*

Fonction	Action	Exemples
ABS (*exp*)	Fournit la valeur absolue de *exp*.	ABS(L*.7) ABS(SIN(X))
ATN (*exp*)	Fournit l'arc-tangente (en radians) de *exp*.	ATN(2.7) ATN(A*3)
CDBL (*exp*)	Fournit la valeur de *exp* en double précision.	CDBL(A) CDBL(A+1/3#)
CINT (*exp*)	Fournit le plus grand entier inférieur ou égal à *exp*. $-32768 <= exp <= 32768$.	CINT(A#+B)
COS (*exp*)	Fournit le cosinus de *exp* (en radians).	COS(2*A) COS(A/57.29578)
CSNG (*exp*)	Fournit la valeur de *exp* en simple précision; si *exp* est en double précision le chiffre le moins significatif subit un arrondi 5/4.	CSNG(A#) CSNG(.33*B#)
EXP (*exp*)	Fournit l'exponentielle de *exp*, soit : e^{exp}	EXP(34.5) EXP(A*B*C-1)
FIX (*exp*)	Fournit la partie entière de *exp* (suppression des décimales).	FIX(A-B)
INT (*exp*)	Fournit le plus grand entier inférieur ou égal à *exp*.	INT(A+B*C)
LOG (*exp*)	Fournit le logarithme naturel (en base *e*) de *exp*. *exp* doit être positif.	LOG (34.5) LOG(A↑B+C)
RND (0)	Fournit un nombre pseudo-aléatoire entre 0.000001 et 0.999999 inclus.	RND(0)
RND (*exp*)	Fournit un nombre entier pseudo-aléatoire compris entre 1 et INT (*exp*) inclus. $1 <= exp < 32768$.	RND(40) RND(A+B)
SNG (*exp*)	Fournit −1 si *exp* est négatif, 0 si *exp* est nul et +1 si *exp* est positif.	SGN(A*B+3) SGN(COS(X))

**exp* est une constante, une variable ou une expression de type numérique.
Sauf spécification contraire, $-1.7E+38 <= exp <= 1.7E+38$.

SIN (*exp*)	Fournit le sinus de *exp* (en radians).	SIN(A/B) SIN(90/57.29578)
SQR (*exp*)	Fournit la racine carrée de *exp* *exp* doit être positif ou nul.	SQR(A*A-B*B)
TAN (*exp*)	Fournit la tangente de *exp* (en radians).	TAN(X) TAN(X*.0174533)

Fonctions spéciales

Fonction	Action et limites	Exemples
ERL	Fournit le numéro de la ligne où l'erreur courante s'est produite.	ERL
ERR	Si une erreur s'est produite, fournit une valeur qui vaut : 2*(code d'erreur –1). D'où également : code d'erreur=ERR/2+1.	ERR/2+1
INP (*port*)	Fournit la valeur lue au *port* d'entrée/sortie spécifié. Cette valeur ainsi que *port* sont comprises entre 0 et 255 inclus.	INP(55)
MEM	Fournit la quantité de mémoire non protégée et non utilisée.	MEM
PEEK (*adr*)	Fournit la valeur du byte d'adresse *adr* en mémoire. *adr* doit être une valeur d'adresse valide sous forme décimale (voir Carte de la Mémoire, Annexe D).	PEEK(15370)
POINT (*x*, *y*)	Teste l'allumage du segment graphique de coordonnée horizontale *x* et de coordonnée verticale *y*. Si le segment est allumé, POINT fournit la valeur VRAI (–1), sinon elle fournit la valeur FAUX (0). Les limites sont : 0 <=*x* <128 et 0 <=*y* <48.	POINT (2*A,B)
POS (0)	Fournit la position du curseur dans la ligne d'écran (de 0 à 63). 0 est un argument fictif.	POS(0)
USR (*n*)	Branche vers une sous-routine en langage machine et fournit la valeur qu'elle calcule. Voir chapitre 8.	USR (45+I)
VARPTR (*var*)	Fournit l'adresse de la variable *var* ou de son descripteur. *var* doit être un nom de variable valide ou d'un élément de tableau. Fournit 0 si la variable n'a pas encore reçu de valeur.	VARPTR(A$) VARPTR (A(I))

Mots réservés du NIVEAU II*

@
ABS
AND
ASC
ATN
CDBL
CHR$
CINT
CLEAR
CLOSE
CLS
CMD
CONT
COS
CSNG
CVD
CVI
CVS
DATA
DEFDBL
DEFFN
DEFINT
DEFSNG
DEFUSR
DEFSTR
DELETE
DIM
EDIT
ELSE
END
ERL
ERR
ERROR
EXP
FIELD
FIX
FOR
FRE
GET
GOSUB
GOTO
IF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LET
LSET
LEN
LINE
LIST
LOAD
LOC
LOF
LOG
MEM
MERGE
MID$
MKD$
MKI$
MKS$
NAME
NEW
NEXT
NOT
ON
OPEN
OUT
PEEK
POINT
POKE
POS
PRINT
PUT
RANDOM
READ
REM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
SAVE
SET
SGN
SIN
SQR
STEP
STOP
STRING$
STR$
TAB
TAN
THEN
TIME$
TROFF
TRON
USING
USR
VAL
VARPTR

* Certains de ces mots ne correspondent pas à des fonctions du BASIC NIVEAU II, mais sont utilisés dans le BASIC DISK NIVEAU II. Aucun mot réservé ne peut être présent dans un nom de variable.

Limites et consommation de mémoire

Limites

Entiers	−32768 à +32767 inclus.
Simple précision	−1.701411E+38 à +1.701411E+38 inclus.
Double précision	−1.70141183454455 6E+38 à 1.70141183454455 6E+38 inclus.

Chaînes alphanumériques : 0 à 255 caractères.

Numéros de ligne : entiers de 0 à 65529 inclus.

Longueur des lignes de programme : jusqu'à 255 caractères.

Consommation statique de mémoire

Chaque ligne de programme exige au moins 5 bytes :

Numéro de ligne	−2 bytes
Pointeur ligne suivante	−2 bytes
Fin de ligne	−1 byte

En outre chaque caractère de la ligne occupe un byte sauf pour les mots réservés qui, chacun, n'occupent qu'un byte.

Consommation de mémoire à l'exécution

Variable entière : 5 bytes (longueur : 1, nom : 2, valeur : 2)

Variable simple précision : 7 bytes (longueur : 1, nom : 2, valeur : 4)

Variable double précision : 11 bytes (longueur : 1, nom : 2, valeur : 8)

Variable alphanumérique : 5 bytes minimum (longueur : 1, nom : 2, pointeur vers la valeur : 2, valeur : 1 byte par caractère)

Tableau : longueur élément : 1, nom : 2, longueur totale : 2, nombre de dimensions : 1, taille de chaque dimension : 2, pour chaque valeur : 2, 3, 4 ou 8 selon le type.

Chaque boucle FOR-NEXT active réclame 16 bytes.

Chaque GOSUB actif (avant RETURN) réclame 6 bytes.

Chaque niveau de parenthèses réclame 4 bytes plus 10 pour chaque résultat intermédiaire.

B / Codes d'erreur du NIVEAU II

Code	Abréviation	Erreur
1	NF	NEXT sans FOR préalable
2	SN	Erreur de syntaxe
3	RG	RETURN sans GOSUB préalable
4	OD	Pas assez de données dans les lignes DATA
5	FC	Appel de fonction invalide
6	OV	Débordement (nombre trop grand)
7	OM	Mémoire insuffisante
8	UL	Numéro de ligne non défini
9	BS	Indice de tableau trop grand
10	DD	Redéclaration d'un tableau existant
11	/0	Division par zéro
12	ID	Instruction invalide en mode Commande
13	TM	Combinaison de types invalide
14	OS	Plus d'espace disponible pour les chaînes
15	LS	Chaîne trop longue
16	ST	Expression alphanumérique trop complexe
17	CN	Commande CONT invalide
18	NR	Pas de RESUME
19	RW	RESUME sans erreur
20	UE	Code invalide dans ERROR
21	MO	Opérande manquant
22	FD	Données incorrectes sur bande
23	L3	Fonction du BASIC DISK

Interprétation des messages d'erreur

NF NEXT sans FOR : un NEXT a été rencontré sans qu'un FOR correspondant ait été exécuté. Cette erreur peut également correspondre à des NEXT *variable* intervertis accidentellement.

SN Erreur de syntaxe : résulte généralement d'une ponctuation incorrecte, de parenthèses non fermées, d'un caractère illégal ou d'une commande mal orthographiée.

RG RETURN sans GOSUB : un RETURN a été rencontré alors qu'aucun GOSUB n'est actif.

OD Plus de données (Out of Data). Une instruction READ ou INPUT # a été exécutée alors qu'il n'y avait plus assez de données pour la satisfaire (lignes DATA supprimées ou insuffisantes).

FC Appel illégal de fonction. Le programme a tenté d'exécuter une opération sur un paramètre invalide (racine carrée d'un nombre négatif, dimension de tableau négative, argument de LOG négatif ou nul, appel de USR sans POKE de l'adresse préalable, etc.).

OV Débordement : une valeur introduite ou calculée est trop grande ou trop petite pour être manipulée par l'ordinateur.

OM Mémoire insuffisante : toute la mémoire disponible a été utilisée ou protégée. Ceci peut survenir dans le cas de tableaux très grands, de boucles FOR-NEXT ou d'appels GOSUB emboîtés.

UL Numéro de ligne non défini : un branchement ou référence est tenté vers une ligne inexistante.

BS Indice de tableau invalide : un indice d'un tableau n'est pas compris dans l'intervalle valide pour ce tableau.

DD Redéclaration de tableau : déclaration d'un tableau ayant déjà fait l'objet d'une déclaration explicite (DIM) ou implicite. Il est prudent de grouper ces déclarations au début du programme.

/0 Division par zéro : tentative d'exécution d'une division dont le dénominateur est nul.

ID Utilisation de INPUT en mode commande (et non dans un programme).

TM Discordance de type : tentative d'assignation d'une valeur alphanumérique à une variable numérique ou vice versa.

OS Plus d'espace disponible pour les chaînes : demande de plus d'espace réservé aux chaînes qu'il n'en reste effectivement.

LS Chaîne trop longue : une chaîne de plus de 255 caractères est assignée à une variable.

ST Expression alphanumérique trop complexe pour être manipulée par l'ordinateur. Décomposez-la en expressions plus simples.

CN Commande CONT invalide, c'est-à-dire intervenant à un instant où le programme ne peut être poursuivi, par exemple après un END ou un EDIT.

NR Pas de RESUME : la fin du programme survient lors de l'exécution d'une routine de traitement d'erreur.

RW RESUME sans erreur : RESUME doit être exécuté après exécution d'un ON ERROR GOTO et à la suite d'une erreur.

UE Code invalide dans ERROR. Le paramètre de ERROR ne correspond pas à un code d'erreur valide.

MO Opérande manquant. Tentative d'exécution d'une opération sans la totalité des opérandes exigés.

FD Données incorrectes (ou dans le désordre) obtenues à partir d'une source externe (p. ex, cassette).

L3 Fonction du BASIC DISK. Exécution d'instruction, fonction ou commande qui n'est disponible qu'en BASIC DISK lorsque le Mini Disque TRS-80 est connecté via l'Interface d'Extension.

C / Table des codes de commande et des codes ASCII et graphiques

Codes de commande : 0-31

Code	Fonction
0-7	Néant
8	Recul du curseur, effacement du dernier caractère
9	Néant
10-13	Passage à la ligne suivante
14	Allumage du curseur
15	Extinction du curseur
16-22	Néant
23	Conversion en 32 caractères par ligne
24	← Recul du curseur d'une position
25	→ Avance du curseur d'une position
26	↓ Passage à la ligne suivante
27	↑ Retour au début de la ligne précédente
28	Retour du curseur en haut à gauche (0,0)
29	Retour en début de ligne
30	Effacement de la fin de la ligne
31	Effacement de la fin de l'écran

Codes des caractères ASCII : 32-128

Code	Caractère	Code	Caractère
32	espace	76	L
33	!	77	M
34	"	78	N
35	#	79	O
36	$	80	P
37	%	81	Q
38	&	82	R
39	'	83	S
40	(	84	T
41	)	85	U
42	*	86	V
43	+	87	W
44	,	88	X
45	-	89	Y
46	.	90	Z
47	/	91	↑ ou [
48	0	92	↓
49	1	93	←
50	2	94	→
51	3	95	—
52	4	96-127	Minuscules des codes 64-95
53	5		
54	6	128	Espace
55	7		
56	8		
57	9		
58	:		
59	;		
60	<		
61	=		
62	>		
63	?		
64	@		
65	A		
66	B		
67	C		
68	D		
69	E		
70	F		
71	G		
72	H		
73	I		
74	J		
75	K		

Codes graphiques : 129 - 191

Vous examinerez ces codes en exécutant :

```
10 FOR X=129 TO 191
20 PRINT X; CHR$(X),
30 NEXT
```

Code de compression : 192 - 255

Code	Fonction
192 - 255	Taquet de tabulation de 0 à 63 respectivement

D/CARTE DE LA MEMOIRE TRS-80 NIVEAU II

ADRESSE DECIMAL	ADRESSE HEXADECIMAL	
0	0000	
		ROM BASIC NIVEAU II
12288	3000	
		Réservé
14302	37DE	Adresse état communication
14303	37DF	Adresse donnée communication
14304	37E0	Adresse verrou interruption
14305	37E1	Adresse verrou sélection unité disque
14308	37E4	Adresse verrou sélection cassette
14312	37E8	Adresse imprimante
14316	37EC	Adresse contrôleur disque
14336	3800	
		Mémoire clavier TRS-80
15360	3C00	
		Mémoire vidéo TRS-80
16383	3FFF	
16384	4000	
		RAM fixe BASIC NIVEAU II
		Vecteur branchements (RST 1 à 7)
16402	4012	
16405	4015	Bloc de contrôle du clavier
		BC + 0 = Type de bloc + 1 = Adresse "Driver" (BMS) + 2 = Adresse "Driver" (BPS) + 3 = 0 + 4 = 0 + 5 = 0 + 6 = "K" + 7 = "I"
16413	401D	Bloc de contrôle du vidéo
		BC + 0 = Type de bloc + 1 = Adresse "Driver" (BMS) + 2 = Adresse "Driver" (BPS) + 3 = Position curseur (BMS) + 4 = Position curseur (BPS) + 5 = Caractère curseur + 6 = "D" + 7 = "0"
16421	4025	Bloc de contrôle de l'imprimante
		BC + 0 = Type de bloc + 1 = Adresse "Driver" (BMS) + 2 = Adresse "Driver" (BPS) + 3 = Lignes/Page + 4 = Compteur ligne + 5 = 0 + 6 = "P" + 7 = "R"

16429	402D	Réservé
16463	404F	
16474	4050	Vecteur Interruption FDC
16466	4052	Vecteur interruption communications
16468	4054	Réservé
16476	405C	
16478	405E	Interruption horloge à 25 msec
		Réservé
16512	4080	RAM BASIC NIVEAU II
		Réservé
16870	41E6	Tampon Entrée/Sortie
17127	42E7	
17128	42E8	Toujours zéro
17129	42E9	
		↓ Texte du programme
		↓ Variables élémentaires
		↓ Tableaux
		Mémoire disponible
		↑ Pile
		↑ Espace chaînes alphanumériques
		Espace protégé par le paramètre "MEMORY SIZE" et destiné par exemple aux programmes en langage machine.
20479 (4K) 32767 (16K)	4FFF (4K) 7FFF (16K)	Fin de la mémoire du bloc clavier

E/Feuille de travail Vidéo

TAB→ PRINT AT Y↓ X→

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
64	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
128	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
192	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
256	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
320	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
384	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
448	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
512	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
576	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
640	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
704	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
768	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
832	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
896	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
960	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

↓Y

F / Fonctions dérivées

Fonction	Expression en termes des fonctions du NIVEAU II
Sécante	SEC(X)=1/COS (X)
Cosécante	CSC(X)=1/SIN(X)
Cotangente	COT(X)=1/TAN(X)
Arc-sinus	ARCSIN(X)=ATN(X/SQR(–X*X+1))
Arc-cosinus	ARCCOS(X)=–ATN(X/SQR(–X*X+1))+1.5708
Arc-sécante	ARCSEC(X)=ATN(SQR(X*X–1))+(SGN(X)–1)*1.5708
Arc-cosécante	ARCCSC(X)=ATN(1–SQR(X*X–1))+(SGN(X)–1)*1.5708
Arc-cotangente	ARCCOT(X)=–ATN(X)+1.5708
Sinus hyperbolique	SINH(X)=(EXP(X)–EXP(–X))/2
Cosinus hyperbolique	COSH(X)=(EXP(X)+EXP(–X))/2
Tangente hyperbolique	TANH(X)=–EXP(–X)/(EXP(X)+EXP(–X))*2+1
Sécante hyperbolique	SECH(X)=2/(EXP(X)+EXP(–X))
Cosécante hyperbolique	CSCH(X)=2/(EXP(X)–EXP(–X))
Cotangente hyperbolique	COTH(X)=EXP(–X)/(EXP(X)–EXP(–X))*2+1
Arc-sinus hyperbolique	ARCSINH(X)=LOG(X+SQR(X*X+1))
Arc-cosinus hyperbolique	ARCCOSH(X)=LOG(X+SQR(X*X–1))
Arc-tangente hyperbolique	ARCTANH(X)=LOG((1+X)/(1–X))/2
Arc-sécante hyperbolique	ARCSECH(X)=LOG((SQR(–X*X+1)+1)/X)
Arc-cosécante hyperbolique	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
Arc-cotangente hyperbolique	ARCCOTH(X)=LOG((X+1)/(X–1))/2

G / Changement de base

Nombre en binaire	Nombre en octal	Nombre en hexadécimal	Nombre en décimal
00000000	000	00	0
00000001	001	01	1
00000010	002	02	2
00000011	003	03	3
00000100	004	04	4
00000101	005	05	5
00000110	006	06	6
00000111	007	07	7
00001000	010	08	8
00001001	011	09	9
00001010	012	0A	10
00001011	012	0B	11
00001100	014	0C	12
00001101	015	0D	13
00001110	016	0E	14
00001111	017	0F	15
00010000	020	10	16
00011000	030	18	24
00100000	040	20	32
00101000	050	28	40
00110000	060	30	48
00111000	070	38	56
01000000	100	40	64
01001000	110	48	72
01010000	120	50	80
01011000	130	58	88
01100000	140	60	96
01101000	150	68	104
01110000	160	70	112
01111000	170	78	120
10000000	200	80	128
10001000	210	88	136
10010000	220	90	144
10011000	230	98	152
10100000	240	A0	160
10101000	250	A8	168
10110000	260	B0	176
10111000	270	B8	184
11000000	300	C0	192
11001000	310	C8	200
11010000	320	D0	208
11011000	330	D8	216
11100000	340	E0	224
11101000	350	E8	232
11110000	360	F0	240
11111000	370	F8	248
11111001	371	F9	249
11111010	372	FA	250
11111011	373	FB	251
11111100	374	FC	252
11111101	375	FD	253
11111110	376	FE	254
11111111	377	FF	255

H / Exemples de programmes

L'Atterrissage

Ce programme vous permet de simuler la phase d'atterrissage d'un vaisseau spatial sur les planètes. Terre, Lune ou Mars ou sur l'astéroïde Vesta.

Vous commandez la vitesse de descente par des impulsions des rétro-fusées d'une durée de 10 secondes. Vous choisissez la consommation en carburant pendant chaque impulsion (de 0 à 100 kg par seconde; au delà, l'accélération serait intolérable). Par exemple, une consommation de 10 kg/sec correspond à une combustion de 100 kg de carburant.

L'objectif est de posez l'appareil dans de bonnes conditions compte tenu d'une quantité initiale limitée de carburant.

Avant chaque impulsion, les renseignements suivants seront affichés :
Temps écoulé (secondes)
Altitude (km)
Vitesse (km/h – **une vitesse négative correspond à une ascension** !)
Quantité de carburant restant disponible (kg).

Remarques :

— Une vitesse négative résulte d'un freinage trop intense; vous vous éloignez de l'objectif.

— Vous pouvez imposer des consommations non entières (p. ex : 15.6 kg/sec).

— La consommation du carburant diminue le poids du véhicule, rendant son freinage plus aisé.

— Les conditions d'atterrissage varient selon la gravité de chaque planète : 981 cm/sec² pour la Terre, 162 cm/sec² pour la Lune, 372 cm/sec² pour Mars et 17,5 cm/sec² pour Vesta.

— Dans le texte ci-dessous, remplacez le symbole [par ↑ (exposant)

Bonne chance !

```
100 CLS
110 PRINT@ 18,"* * * ATTERRISSAGE * * *"
120 PRINT : PRINT "TAPEZ 1 (TERRE), 2 (LUNE), 3 (MARS), 4 (VESTA)"
140 INPUT X : ON X GOTO 500,600, 700, 800
142 GOTO 100
145 CLS
147 PRINT@ 980, A$;
150 G2 = G1/36
160 G3 = SQR(G2) * 100 : G3 = FIX(G3) : IF G3<175 THEN G3=175
170 G4 = G3 * 55 : G4 = FIX(G4) : IF G4<10000 THEN G4=10000
180 G5 = G4 * (LOG(G1)/20) + 10000
190 A1 = -6400 : A2 = 5000 : A3 = 15000 : A4 = 10
200 B4 = A4 : B2 = A2 : N3 = G3 : N4 = G4
205 PRINT@0,"TEMPS     ALTITUDE     VITESSE       CARBURANT   CONSOMMATION";
206 PRINT@64,"ECOULE       (KM)        (KM/H)          RESTANT     (KG/SEC)";
210 PRINT@ 128+0, T1; TAB(10) N3; TAB(24) B2; TAB(39) N4; TAB(53); : INPUT F
250 IF F=0 GOTO 280
260 IF F<0 OR F>100 GOTO 320
```

```
270 T=N4/F : IF T<10 THEN B4=T
280 N4=N4-F*B4
285 V1=B3
286 T1=T1+B4
290 B5=(G2+ (G2*N3)/(-2*A5)) - (F*G5)/(A3+N4)
295 B3=B2+B5*B4
298 N5=N3
300 N3=N3+((B3+B2)/A1)*B4
305 B2=B3
307 IF N3<0 GOTO 450
310 IF N4<=0 GOTO 400 : GOTO 210
312 Q=Q+64 : IF Q+128 > 960 THEN Q=832
315 GOTO 205
320 PRINT " -->> CONSOMMATION INVALIDE  -- ESSAYEZ ENCORE (0 A 100)" : GOTO 210
400 V2 = SQR( B2[2 + N3 * G2 * 5650) : PRINT "RESERVOIRS VIDES A"; T1; "SECONDES"
410 V3 = ABS(V2)*10000/3600
420 T1 = T1 + LOG(V3 * N3 * 10000 / G1)
430 GOTO 1000
450 V2 = SQR(ABS(N5 / (26 * B5))) * (26 * B5) + V1 : GOTO 1000
460 T1 =T1-(10-B4)
500 G1 = 980.7 : A5 = 6371 : A$ ="TERRE" : GOTO 145
600 G1 = 162 : A5 = 1738 : A$="LUNE" : GOTO 145
700 G1 = 374 : A5 = 3380 : A$="MARS" : GOTO 145
800 G1 = 17.5 :A5 = 195  : A$="VESTA": GOTO 145
1000 PRINT
1010 IF V2<20 PRINT"VOUS AVEZ ATTERRI" : GOTO 1100
1020 IF V2<100 PRINT"VOUS VOUS ETES ECRASE" : GOTO1140
1030 IF V2<250 PRINT"VOUS ETES REDUIT EN BOUILLIE" : GOTO 5000
1040 IF V2<5000 PRINT"VOUS AVEZ FORME UN NOUVEAU CRATERE" : GOTO 5000
1050 IF V2>4999 PRINT"VOUS APPROCHEZ DU CENTRE DE LA PLANETE !!!":GOTO5000
1100 IF V2<1 PRINT " EN DOUCEUR -- BRAVO" :GOTO 5000
1105 GOTO 5000
1110 IF V2<5 PRINT " HONNETEMENT" : GOTO 5000
1120 PRINT " UN PEU RUDEMENT" : GOTO 5000
1140 IF V2<30 PRINT "VOTRE APPAREIL EST ENDOMMAGE" : GOTO 5000
1150 IF V2<45 PRINT"VOUS ETES BLESSE ET VOTRE APPAREIL EST DETRUIT" : GOTO 5000
1160 PRINT"IL N'Y A PAS DE SURVIVANT"
5000 PRINT "VITESSE AU MOMENT DE L'IMPACT :"; ABS(V2); "KM/H"
5010 PRINT "TEMPS ECOULE                  :"; T1; "SEC"
5020 END
```

Manipulation d'un fichier de renseignements

Ce programme vous permet d'enregistrer des informations dans un fichier. Les renseignements choisis sont les nom, adresse et numéro de téléphone d'un ensemble de personnes. Vous pouvez construire, consulter, modifier, sauver et relire ce fichier grâce à une table de commandes (''Menu''). Vous n'aurez pas de difficulté à adapter ce programme à vos propres besoins.

```
10 CLEAR 1000 : CLS : DIM N$(50),A$(50),T$(50)
20 CLS :PRINT@ 10,"* * MENU * *":PRINT:PRINT
30 PRINT"1 : CONSTRUIRE UN FICHIER"
40 PRINT"2 : AFFICHER TOUT LE FICHIER"
50 PRINT"3 : AFFICHER UNE LIGNE"
60 PRINT"4 : CORRIGER UNE LIGNE"
70 PRINT"5 : SAUVER LE FICHIER SUR CASSETTE"
80 PRINT"6 : LIRE UN FICHIER SUR CASSETTE"
85 PRINT: PRINT"TAPEZ LE NUMERO DE L'OPERATION DESIREE ";
90 INPUT Q: ON Q GOTO 100,200,300,400,500,600 :GOTO20
100 PRINT:PRINT"POUR CLOTURER LE FICHIER INTRODUISEZ LE NOM : 999"
105 INPUT" LORSQUE VOUS ETES PRET POUSSEZ SUR 'ENTER'";X
110 FOR I=1 TO 50: CLS
112 INPUT"NOM ET PRENOM (SANS VIRGULE),PUIS 'ENTER' ";N$(I)
115 IF N$(I)="999" THEN P1=I: GOTO 150
120 INPUT"ADRESSE (SANS VIRGULE),PUIS 'ENTER' ";A$(I)
130 INPUT"NUMERO DE TELEPHONE, PUIS 'ENTER'   ";T$(I)
135 IF FRE(X$) < 100 THEN P1=I:GOTO 150
140 NEXT
145 P1=I-1
150 PRINT:PRINT "FICHIER CLOTURE -- POUR AFFICHER LE MENU POUSSEZ SUR 'ENTER'";:INPUT X
160 GOTO 20
200 CLS: FOR I=1 TO P1-1: PRINT N$(I), A$(I), T$(I): NEXT :PRINT
210 INPUT"POUR AFFICHER LE MENU POUSSEZ SUR 'ENTER' ";X : GOTO 20
300 CLS: INPUT"NOM ET PRENOM (SANS VIRGULE) ";N$
310 FOR I=1 TO P1: IF N$(I)=N$ THEN 330
315 NEXT
320 PRINT"CE NOM N'EST PAS DANS LE FICHIER": GOTO340
330 PRINT N$(I), A$(I), T$(I)
340 PRINT : PRINT"POUR UN AUTRE NOM TAPEZ 1, SINON 0 ";:INPUT X
350 IF X=1 GOTO 300 ELSE 20
400 CLS: PRINT"NOM ET PRENOM (SANS VIRGULE) DE LA LIGNE A MODIFIER"
405 INPUT N$
410 FOR I=1 TO P1 : IF N$=N$(I) GOTO 430
415 NEXT
420 PRINT"CE NOM N'EST PAS DANS LE FICHIER": GOTO 460
430 PRINT"TAPEZ LA LIGNE MODIFIEE : NOM, ADRESSE, NO TEL. "
440 INPUT N$(I), A$(I), T$(I)
460 PRINT: PRINT"POUR UNE AUTRE MODIFICATION TAPEZ 1, SINON 0";
465 INPUT X
470 IF X=1 GOTO 400
480 GOTO 20
500 CLS: INPUT"PREPAREZ LA CASSETTE PUIS POUSSEZ SUR 'ENTER'";X
510 PRINT"COPIE ..."
520 PRINT #-1, P1
530 FOR I=1 TO P1: PRINT #-1, N$(I), A$(I), T$(I) :NEXT
540 PRINT"TERMINE -- NOTEZ L'INDICATION DU COMPTEUR"
550 INPUT"POUR AFFICHER LE MENU POUSSEZ SUR 'ENTER' ";X: GOTO 20
600 CLS : INPUT"PREPAREZ LA CASSETTE PUIS POUSSEZ SUR 'ENTER'";X
610 PRINT"LECTURE ..."
620 INPUT #-1,P1
630 FOR I=1 TO P1: INPUT #-1,N$(I), A$(I), T$(I) :NEXT
640 PRINT"TERMINE": INPUT"POUR AFFICHER LE MENU POUSSEZ SUR 'ENTER'";X
650 GOTO 20
```

Démonstration graphique de calcul de triangles

Ce programme illustre l'utilisation de fonctions mathématiques et graphiques dans un problème connu des élèves de géometrie : le calcul d'un triangle. (Remplacez [par ↑).

```
10 CLS
100 PRINT"CE PROGRAMME CALCULE L'AIRE D'UN TRIANGLE"
110 PRINT"CONNAISSANT 3 PARAMETRES ET DESSINE LE TRIANGLE A L'ECHELLE"
120 PRINT:PRINT"TAPEZ   CCC POUR 3 COTES; CAC POUR 2 COTES ET UN ANGLE; ACA POUR 1 COTE ET 2 ANGLES"
135 PI=3.14159
140 INPUT A$ :CLS:IF A$="CAC" GOSUB 300
150 IF A$="ACA" GOSUB 400
200 ' *** 3 COTES ***
210 PRINT"DONNEZ LES 3 COTES EN COMMENCANT PAR LE PLUS GRAND";
220 INPUT L1,L2,L3
225 IF L2>L1 OR L3>L1 PRINT "* * * LE PLUS GRAND D'ABORD * * *":PRINT:GOTO 210
230 S=(L1+L2+L3)/2
235 IF S <= L1 PRINT " * * * CECI N'EST PAS UN TRIANGLE * * *":PRINT:GOTO 210
240 Y3 = 2 * SQR( S * (S-L2) * (S-L1) * (S-L3)) / L1
250 A = Y3/L2 :A=ATN( A / SQR(-A * A + 1))
260 X3 = COS(A) * L2
270 AR = (L1*Y3) / 2
280 GOTO 500
300 ' *** 2 COTES ET 1 ANGLE ***
310 PRINT "DONNEZ LES 2 COTES ET L'ANGLE : AB, AC, THETA ( AB>=AC )";
320 INPUT L1,L2,T
325 T = (T * PI)/180
330 Y3 = L2 * SIN(T)
340 X3 = L2 * COS(T)
350 AR = (L1 * Y3)/2
360 GOTO 500
400 ' *** 2 ANGLES ET 1 COTE ***
410 PRINT "DONNEZ LES 2 ANGLES ET LE COTE : THETA1, THETA2, AB";
420 INPUT T1, T2, L2
425 T1 = (T1 * PI)/180 : T2 = (T2 * PI)/180
430 Y3 = L2 * SIN(T1)
440 B1 = COS(T1) * L2
450 B2 = Y3/TAN(T2)
460 L1 = B1 + B2 : X3 = B1
470 AR = (L2 * Y3)/2
500 CLS : F=1 : IF L1>50 OR Y3>30 OR L2>50 THEN GOSUB 700
510 VC = (PI * (L1 * F - X3 * F) * (Y3 * F) [ 2) / 3
520 VS = (PI * (X3 * F) * (Y3 * F) [ 2) / 3 :VT = VC + VS
525 IF F=6 GOTO 610
530 S1=Y3/X3 : S2=Y3/(X3-L1)
532 IF INT(X3) = 0 THEN 1100
533 IF INT(X3) = INT(L1) THEN 1000
534 IF X3<0 THEN 1299
535 IF X3>L1 THEN 1199
537 IF X3=L2 THEN 1000
540 FOR Y=20 TO L1*2+20 STEP 2: SET(Y,Y3+5): NEXT
550 FOR X=0 TO X3 : SET(X*2+20,S1*(X3-X)+5) : NEXT
560 FOR X=X3 TO L1 : SET(X*2+20,Y3+(S2*(L1-X)+5)) : NEXT
590 PRINT@ 64*INT((Y3+5)/3)+69, "A (0,0)"; TAB(L1); "B("; L1*F; ",0)"
600 PRINT@ (X3+20)/2, "C ("; X3*F; ","; Y3*F; ")";
610 PRINT@832, "AIRE ="; AR; " CM2";
620 PRINT@ 896, "VOLUME ENGENDRE PAR LA REVOLUTION DU TRIANGLE ";
625 PRINT "AUTOUR DE L'AXE X (LIGNE AB) ="; VT; " CM2";
```

```
630 PRINT@ 768, "*"; : INPUT "POUR RECOMMENCER TAPEZ 1 ";B6 : IF B6=1 THEN 10
640 END
700 IF L1<60 THEN F=2 : GOTO 750
710 IF L1<90 THEN F=3 : GOTO 750
720 IF L1<120 THEN F=4 : GOTO 750
730 IF L1<150 THEN F=5 : GOTO 750
740 PRINT "ECHELLE TROP GRANDE POUR ETRE DESSINEE"; : F=6 . GOTO 510
750 L1=L1/F : Y1=Y1/F : Y2=Y2/F : Y3=Y3/F : X1=X1/F : X2=X2/F : X3=X3/F
760 RETURN
1000 FOR Y=5 TO Y3+5 : SET(X3*2+20,Y) : NEXT :GOTO540
1100 FOR Y=5 TO Y3+5 : SET(20,Y) : NEXT :GOTO540
1199 IF X3>127 GOSUB 700
1200 FOR X=L1 TO X3 : SET(X*2+20,Y3+(S2*(L1-X)+5))   NEXT : GOTO540
1299 IF X3 < -10 GOSUB 700
1300 FOR X=X3 TO 0 : SET(X*2+20,Y3+(S1*(0-X)+5)) : NEXT : GOTO 540
```

Tir sur cible mobile

Ce programme utilise la fonction INKEY$ pour réaliser l'un des jeux TV les plus populaires. Vous remarquerez le nombre réduit d'instructions qu'il réclame. Vous pourriez l'améliorer en y incorporant la mémorisation et l'affichages des scores.

```
1 CLS:PRINT:PRINT CHR$(23);"TAPEZ 'Z' POUR VISER A GAUCHE"
2 PRINT "TAPEZ '/' POUR VISER A DROITE"
3 PRINT "TAPEZ UN ESPACE POUR TIRER"
4 FOR I=1 TO 2000:NEXT
6 CLS: INPUT"VITESSE DE LA CIBLE  (0.1 TO 1.5)";S1
10 CLS:CA=928:I=1:PRINT @CA,"*";:PRINT @991,"***";
20 F=0
30 IF I>=15 PRINT @124,"          ";:I=1
40 PRINT @64+I*4,"      ";:I=I+RND(0)*S1:PRINT @64+I*4," --> ";
50 IF F=0 THEN 200
60 RESET(MX,MY): MX=MX-MD: MY=MY-8: IF MX<=0 OR MX>=127 THEN 20
70 IF MY>2 SET(MX,MY): GOTO30
80 IF ABS(I*8-MX)>4 THEN 20
90 FOR J=1 TO 6: PRINT @64+4*I,"*****";: FOR K=1 TO 50: NEXT
95 PRINT @64+4*I,"     ": FOR K=1 TO 50: NEXT K,J
100 GOTO 10
200 Y$=INKEY$
205 IF F=1 STOP
210 IF Y$<>"Z" THEN 250
220 IF CA<922 THEN 30
```

```
230 PRINT @CA," ";: CA=CA-1: GOTO 280
250 IF Y$<>"/" THEN 300
260 IF CA>934 THEN 30
270 PRINT @CA," ";: CA=CA+1
280 PRINT @CA,"*";: GOTO 30
300 IF Y$<>" " THEN 30
310 F=1: MD=928-CA: MY=40: MX=64-3*MD: SET(MX,MY): GOTO 30
311 END
```

Le "point qui rebondit"

Vous souvenez-vous du programme du "point qui rebondit" du NIVEAU I ?
Le programme proposé en reprend l'idée pour réaliser un jeu d'adresse pour 1 à 4 joueurs.
Il s'agit pour le joueur de fixer le point de départ (de 1 à 126) du point qui va tomber puis rebondir. Pendant ce mouvement, un missile se déplace de gauche à droite et inversement. Lorsque le missile rencontre le point, il le détruit. La position initiale du point est donnée par 3 chiffres (ou 1 ou 2 chiffres suivis de ENTER).
L'ordinateur fixe lui-même le premier coup; c'est ensuite au premier joueur de tirer.

```
3 REM PROGRAMME DE TIR
5 DIM N$(4)
6 CLS: INPUT "NOMBRE DE JOUEURS :";X1: PRINT"TAPEZ";X1;"NOMS:"
7 FOR XI=1 TO X1: INPUT N$(XI): NEXT: XI=1
10 CLS
20 FOR M=0 TO 127: SET(M,0): SET(M,47): NEXT
30 FOR M=0 TO 47: SET(0,M): SET(127,M): NEXT
35 FOR X=1 TO 121 STEP 10: RESET(X,0): NEXT
40 RANDOM:Y=RND(40)+1: X=RND(110)+4
50 D=1: Q=1: Z=64
60 RESET(Z,Y-D): RESET(X-Q*4,24)
70 SET(Z,Y): SET(X,24): GOSUB 500
80 Y=Y+D: X=X+Q
90 IF X=123 OR X=4 THEN GOSUB 700
100 IF Y=47 THEN 120
105 IF Y=0 GOSUB 900
110 IF Y<>-1 OR X<>-1 THEN 60
120 Y=Y-2*D: D=-D: GOTO 60
500 IF X=Z OR X=Q+Z OR X=2*Q+Z OR X=3*Q+Z OR X=Q*4+Z THEN IF Y=24 GOSUB 600
510 IF Y=23 OR Y=24 OR Y=25 THEN IF X=Z GOSUB 600
520 RETURN
600 X=1
610 FOR Z=1 TO 50: PRINT@ 550,"TOUCHE !!!";: NEXT
620 FOR Z=1 TO 25: PRINT@ 550,"          ";: NEXT
630 X=X+1: IF X<5 GOTO 610
640 GOTO 2000
700 X=X-2*Q: Q=-Q: RETURN
900 T$=INKEY$: A$="":B$="":C$=""
1000 A$=INKEY$: IF LEN(A$)=0 THEN 1000
1005 PRINT@0,A$;
1010 B$=INKEY$: IF LEN(B$)=0 THEN 1010
1015 PRINT@ 1,B$;
1020 C$=INKEY$: IF LEN(C$)=0 THEN 1020
1030 RESET(Z,1): X$=A$+B$+C$: Z=VAL(X$): IF Z>126 THEN 1100
```

```
1033 PX=PX+1
1035 GOTO120
1040 RETURN
1100 FOR X=1 TO 50: PRINT@ 50,"TROP GRAND, ESSAYEZ ENCORE":NEXT
1110 PRINT@ 50,"                ": Z=1: GOTO 1000
2000 IF PX=0 GOSUB 3000
2010 CLS: PRINT "                    * * * ";N$(XI);" * * *":PRINT:PRINT
2017 PX(XI)=PX+PX(XI): PH(XI)=PH(XI)+1
2020 PRINT,"TIRS        SUCCES          POURCENTAGE"
2030 PRINT: PRINTTAB(17) PX;TAB(28)"1";TAB(42) (1/PX)*100
2035 IF PX(1)=0 THEN X(1)=1
2040 PRINT: PRINT"TOTAL         ";TAB(17) PX(XI)*100
2045 FOR X=1 TO 2500: NEXT
2050 XI=XI+1
2060 IF XI>X1 THEN XI=1
2065 PX=0
2070 GOTO10
2115 IF PX=0 GOSUB 3000
3000 PRINT@ 0,"BRAVO !!!": PX=1: RETURN
```

Notes :

Notes :

Notes :

Notes :

Notes :

Notes :

Notes :

Notes :

Notes :

TANDY CORPORATION

AUSTRALIA	BELGIUM	U.K.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

PRINTED IN BELGIUM