Réalisez votre ordinateur individuel

MODE D'EMPLOI DU BASIC ETENDU sur disquette

CE numéro va, comme les trois précédents, être consacré au mode d'emploi d'un logiciel, en l'occurrence le Basic sur disquette. Que les passionnés du fer à souder se rassurent tout de suite, le mois prochain nous allons reprendre les réalisations de cartes avec la carte IPT09 qui est une carte d'interface parallèle supportant de un à trois PIA ou VIA et un timer programmable, carte qui servira à piloter notre futur programmateur d'UVPROM. Il nous a semblé opportun de présenter ce mode d'emploi immédiatement après ceux du DOS, de l'éditeur, de l'assembleur et des extensions du DOS car il forme avec ces derniers un tout en conférant à votre système une puissance très satisfaisante pour nombre d'applications.

Ce mode d'emploi va comporter des parties communes avec celui du Basic sur cassette décrit en son temps dans la revue ; cependant, comme de nombreux points seraient à corriger au sein du texte déjà publié et que cela rendrait le mode d'emploi global ainsi constitué difficilement lisible, nous avons préféré le reprendre ici dans son intégralité (d'autant que certains d'entre vous nous ont reproché de ne pas avoir adopté cette solution pour le mode d'emploi de l'éditeur disque).

Présentation générale

Ce Basic est, bien sûr, fourni sur une disquette simple face puisque tous les lecteurs, qu'ils soient simple ou double face peuvent lire et copier les disquettes simple face. Si vous ne le précisez pas lors de votre demande, la disquette fournie est 40 pistes (mais peut être lue dans un lecteur 35 pistes car son contenu n'occupe que les quelques premières pistes) ; il est cependant souhaitable que vous précisiez toujours lors de vos demandes de logiciels sur disquette si vous voulez du 40 pistes ou du 80 pistes, cela évite les erreurs et les incertitudes.

La disquette qui vous est fournie supporte quatre fichiers :

– TBASIC.CMD qui est l'interpréteur Basic proprement dit et que vous allez pouvoir copier tel quel sur votre disquette système de façon à pouvoir ensuite appeler le Basic en frappant TBASIC comme si vous frappiez n'importe quelle commande du DOS.

– RENUMBER.CMD qui est un programme de renumérotation de vos programmes Basic et dont nous présenterons le rôle ci-après ; ce programme doit aussi être copié sur votre disquette système comme TBasic à moins que vous ne souhaitiez pas l'utiliser, ce qui serait curieux.

– ERBAS.SYS est un fichier de messages d'erreurs qui englobe celui du DOS (le ERREURS.SYS de votre disquette DOS) et les messages propres au Basic. ERBAS est la version non accentuée des messages et c'est donc celle que vous devez utiliser si vous n'avez pas les générateurs de caractères accentués sur votre carte IVG09. Il faut copier ERBAS.SYS en lieu et place de ERREURS.SYS de votre disquette DOS ; en d'autres termes, vous devez effacer ERREURS.SYS de votre disquette DOS et faire un COPY 1.ERBAS.SYS, O.ERREURS.SYS. Si vous n'avez qu'un lecteur, procédez de même mais, comme COPYSD ne permet pas de changer les noms de fichiers pendant la copie, vous ferez (après avoir effacé ERREURS.SYS de votre disque système) COPYSD ERBAS.SYS puis, lorsque la commande sera terminée, il suffira de faire un RENAME ERBAS.SYS, ERREURS.SYS.

– ERBASA.SYS est la même chose que ERBAS.SYS mais avec les minuscules accentuées. Vous utiliserez donc ce fichier si votre carte IVG09 est équipée du générateur accentué. La procédure de mise en place est identique à celle décrite ci-avant pour ERBAS.

Si vous ne mettez pas en place un de ces fichiers d'erreurs à la place de votre ERREURS.SYS, le Basic vous indiquera les erreurs par un numéro de code au lieu de vous indiquer cela en clair et en français.

La mise en service

Une fois que les manipulations précédentes ont été réalisées, la mise en service est très simple puisqu'il suffit de frapper TBASIC pour que, 12 secondes après environ, le Basic soit opérationnel ce qui est indiqué par l'apparition du message PRET sur l'écran.

Le Basic réside en mémoire de 0 à 492C (soit près de 19 K qui se chargent en 12 secondes ; c'est tout de même mieux qu'avec une cassette !) et il utilise pour ses programmes tout l'espace mémoire compris entre 492D et BFFF puisque le DOS se trouve à partir de C000. Cela permet de manipuler des programmes de taille considérable et, si vous avez des tableaux très importants à gérer dans ceux-ci, sachez déjà que cela peut être fait directement sur la disquette sans mettre le ou les tableaux en mémoire ; il est donc très rare d'arriver à un blocage à cause de la taille mémoire.

Lorsque vous frappez TBASIC, celui-ci se charge et se lance tout seul comme toutes les autres commandes du DOS. Si vous désirez, pour une raison quelconque, le lancer à partir de TAVBUG09, il vous suffit de savoir que son adresse de lancement initiale est 0 ; un G 0 lance donc le Basic comme lors d'un TBASIC c'est-à-dire que sa mémoire de programme est mise à zéro et que toutes les variables sont initialisées. Il existe un deuxième point de lancement à l'adresse 3 qui n'efface pas la mémoire de programme et qui n'initialise pas les variables du Basic, c'est-à-dire que si vous êtes sorti de l'interpréteur pour une raison quelconque et que vous le relanciez par un G 3, vous retrouverez votre programme Basic tel que vous l'avez laissé.

Définitions et conventions

Certaines touches ont un rôle particulier, rôle que l'on retrouve d'ailleurs sur tous nos logiciels. Ce sont :

– CNTRL H qui efface le dernier caractère frappé et fait revenir le curseur en arrière d'une position. CNTRL H peut être frappé autant de fois que nécessaire au sein d'une même ligne.

– CNTRL X qui efface la ligne sur laquelle se trouve le curseur, le « message » ^ X est alors affiché.

– CNTRL C qui permet d'interrompre un programme Basic lorsque celui-ci attend une entrée de données et fait revenir le Basic en mode d'attente de commande ; le message BREAK LIGNE XX est alors imprimé, XX étant le numéro de la ligne où est intervenu le CNTRL C.

Pour nos amis lecteurs novices, précisons que CNTRL Z signifie qu'il faut appuyer sur la touche CONTROL et, pendant qu'elle est enfoncée, appuyez sur la touche Z.

Précisons aussi que sur la plupart des claviers, CNTRL H peut être ramplacé par la touche « flèche vers la gauche » (qui correspond au retour arrière du curseur) tandis que CNTRL X peut être remplacé par DELETE.

Par ailleurs, pour clarifier certaines parties de l'exposé qui va suivre, nous allons adopter deux conventions de notation. Dans les expressions que nous allons présenter, les paramètres indispensables seront représentés entre crochets (<>) tandis que les paramètres facultatifs seront représentés entre parenthèses (()).

Enfin, et avant de poursuivre, précisons que ce qui va suivre n'est pas un cours de Basic mais seulement le mode d'emploi de notre Basic.

Lignes, constantes et variables

Le Basic possède deux modes de fonctionnement, le mode programmé et le mode immédiat. En mode immédiat chaque ligne frappée est exécutée immédiatement tandis qu'en mode programmé, un ensemble de lignes constituant un programme est exécuté sur commande. La différence entre les deux modes est faite de la façon suivante : le fait de commencer une ligne par un numéro place le Basic en mode programmé, tandis que le fait de frapper une commande sans numéro de ligne fait exécuter celle-ci aussitôt, le Basic étant alors en mode immédiat.

Une ligne est un ensemble de caractères terminé par un retour chariot. Elle peut comporter jusqu'à 127 caractères (et ce même si les lignes de votre terminal ont une capacité inférieure, cela n'a aucune influence). Il est possible de placer plusieurs instructions Basic sur la même ligne à condition de séparer celles-ci entre elles par deux points (:). Par ailleurs, les commandes et instructions Basic peuvent être frappées en majuscules ou minuscules indifféremment. Le Basic les convertit automatiquement en majuscules ; par contre, dans les chaînes de caractères, le Basic respecte vos désirs et ne modifie aucunement ce que vous avez frappé et l'on peut donc travailler en majuscules ou en minuscules sans problème.

Dans un programme, les lignes sont numérotées ; le numéro doit être placé immédiatement en début de ligne sans signe ou espace le précédant. Les numéros doivent être compris entre 1 et 32767 et doivent être uniques. Le fait de frapper deux lignes avec le même numéro ne fait conserver en mémoire que la dernière des deux. Par ailleurs, nous vous rappelons qu'il est d'usage, lors de l'écriture initiale d'un programme d'écrire les numéros de 10 en 10 ; cela permet de rajouter par la suite des lignes intermédiaires que vous auriez pu oublier. Cela est possible car, quel que soit l'ordre de frappe, les lignes sont toujours exécutées dans l'ordre numérique croissant.

Le fait de frapper un numéro de ligne seul suivi d'un retour chariot efface la ligne qui portait ce numéro, si aucune ligne n'existait sous ce numéro, cette action est sans effet.

Au sein d'une ligne, et après le numéro de début, les espaces sont ignorés et peuvent donc être utilisés comme vous le désirez, ainsi :
— 10 PRINT SIN (X) aura le même effet que
— 10 PRINTSIN(X), mais cette dernière ligne ne rendra pas vos listings particulièrement lisibles !

Le Basic travaille comme il se doit sur des réels et il peut manipuler tout nombre positif ou négatif compris entre 10 puissance −38 et 10 puissance 38 ; de plus, le mode de représentation adopté confère au Basic une précision de 16 à 17 chiffres significatifs ; les nombres réels sont en effet codés sur 56 bits en mémoire ! Les nombres réels sont frappés de façon classique ; il faut seulement prendre soin de remplacer la virgule par un point ; ainsi 2,5 sera frappé 2.5 pour que le Basic comprenne.

Il est également possible de fournir au Basic des nombres en notation scientifique ; le format est normalisé comme pour tous les Basic, à savoir que, par exemple, 3,456 que multiplie 10 puissance −5 sera frappé : 3.456E-5. Le 10 puissance est matérialisé par E suivi de la puissance à laquelle était élevé 10.

Il est aussi possible d'utiliser comme nombre des expressions ainsi, si un tiers doit figurer dans un calcul, plutôt que de frapper sa valeur approchée 0.333333, vous pouvez très bien frapper 1/3 et écrire, par exemple 6 + 1/3 − 2/5 ; le Basic comprendra.

D'autre part, comme la représentation des réels est faite sur 56 bits et que cela consomme de la mémoire et du temps machine, il est possible, pour les nombres compris entre + 32767 et − 32768 de les définir comme étant des entiers ; ils sont alors codés sur 16 bits et, non seulement cela n'occupe que peu de place en mémoire, mais de plus, vu la structure interne du 6809, cela accroît de façon considérable la vitesse d'exécution des calculs concernés. Pour définir un nombre comme un entier plutôt que comme un réel, il suffit qu'il soit dans la plage − 32768 + 32767 et qu'il ne contienne pas de point décimal ; ainsi 2 sera un entier alors que 2.0 qui a la même valeur numérique sera un réel.

Enfin un autre type de constantes auxquelles vous n'êtes pas, en général, habitués est constitué par les chaînes de caractères. Ainsi pourrons-nous définir la constante « BONJOUR » ou la constante « HAUT-PARLEUR ». Remarquez qu'une chaîne de caractères est constituée par n'importe quel ensemble de caractères compris entre deux guillemets ou entre deux apostrophes.

Nous avons parlé constantes jusqu'à maintenant ; il est évidemment possible de définir des variables. Celles-ci peuvent être du même type que les constantes, auquel cas elles peuvent recevoir un nom constitué par une lettre ou deux lettres ou une lettre suivie par un chiffre de 0 à 9 ; ainsi A, XX, AO, BC, D3, E sont des noms de variables corrects, par contre, 5C ou 8 ne seraient pas admis.

Cette représentation est valable pour les variables numériques ; pour ce qui est des variables chaînes de caractères, les mêmes règles s'appliquent mais leur nom doit être suivi du symbole dollar ($) ainsi : A9$, BD$ ou C$ seront des variables chaînes de caractères.

Une autre notation est également utilisable pour les variables ; c'est celle définissant une variable numérique comme un entier ; en effet, en l'absence d'indication particulière, une variable numérique est considérée comme un réel. Si le nom d'une variable numérique est suivi par le symbole « pour cent » (%), la variable sera considérée comme étant un entier, ainsi A%, A9% seront des entiers. Attention, si vous définissez une variable comme étant un entier et que vous essayez ensuite de lui donner une valeur réelle (valeur hors de la plage des entiers ou comportant un point décimal) vous ferez générer un message d'erreur.

Il existe enfin un dernier type de variables qui sont les variables indicées qui servent à constituer des tableaux de valeurs. Ces variables répondent aux règles exposées ci-avant mais doivent être dimensionnées avant utilisation et s'utilisent ensuite avec un indice. Ainsi, par exemple, soit l'instruction DIM A (3), cela va avoir pour effet de créer 4 variables appelées A(0), A(1), A(2) et A(3).

Le dimensionnement peut être double et l'on peut ainsi créer des tableaux ou des matrices comme montré en exemple figure 1.

Dans un même programme, le même nom peut être employé pour une variable classique et pour une variable indicée ; ainsi A5 sera différent de A5(0) ; par contre il est interdit de donner le même nom à une variable indicée simple et à une variable indicée double.

Les opérateurs

Il existe en Basic quatre types d'opérateurs que nous allons étudier successivement.

Commençons par les plus classiques qui sont les opérateurs mathématiques.

Ils sont au nombre de 5 : l'addition (+), la soustraction (–), la multiplication (* et non pas × !), la division (/) et l'élévation à une puissance (^) et non ** comme sur certains Basic). Quand on les rencontre dans une expression et sauf s'il y a des parenthèses pour modifier ce qui suit, les opérateurs sont exécutés dans l'ordre de priorité suivant :

– Elévation à une puissance
– Changement de signe (– devant un nombre)
– Multiplication et division
– Addition et soustraction.

Viennent ensuite les opérateurs logiques au nombre de trois : NOT, AND et OR.

– NOT réalise le complément bit à bit de la variable spécifiée
– AND réalise le ET logique entre deux variables
– OR réalise le OU logique entre deux variables.

Ces opérateurs s'emploient presque exclusivement dans des tests conditionnels ; ainsi pourrons-nous écrire :

– IF A > 0 AND B > 0 THEN GOTO 100 qui signifiera si A est positif ET si B est positif, aller en 100.

Cet exemple a introduit le troisième type d'opérateurs qui est celui des opérateurs de relation. Ils sont au nombre de 6 :

– = qui s'utilise sous la forme A = B et qui signifie A égal B
– < > qui s'écrit A < > B et qui signifie A différent de B
– < qui s'écrit A < B et qui signifie A inférieur à B
– > qui s'écrit A > B et qui signifie A supérieur à B
– <= et >= qui s'écrivent A >= B ou A <= B et qui signifient respectivement A supérieur ou égal à B ou A inférieur ou égal à B.

La dernière famille d'opérateurs surprend en général les personnes non habituées à l'informatique car ils permettent de travailler sur les chaînes de caractères. Ce sont la concaténation notée + qui « ajoute » les chaînes de caractères ; ainsi si A$ = « HAUT » et B$ = « PARLEUR », A$ + B$ vaudra « HAUT PARLEUR », mais aussi les opérateurs de relation vus ci-avant. Dans ce cas la comparaison ne peut pas être numérique, elle est donc alphabétique et permet ainsi de réaliser un classement très simplement. Par exemple, la chaîne « CLAUDE » sera inférieure à la chaîne « MICHEL ».

Tous les opérateurs vus ci-avant peuvent parfois être combinés dans des expressions complexes ; leurs priorités relatives sont alors celles indiquées dans le tableau de la figure 2.

Les commandes

Nous avons vu que le Basic pouvait fonctionner en mode calculateur ou immédiat et en mode programmé, un certain nombre de « fonctions » ne peuvent être exécutées qu'en mode immédiat : ce sont les commandes du Basic. Ces commandes ne sont pas des instructions mais des ordres relatifs au fonctionnement général du Basic. Elles sont au nombre de 13 et nous allons les étudier ci-après. Il faut évidemment les frapper sans numéro de ligne et il est donc interdit de les utiliser dans un programme (nous ne voyons pas d'ailleurs ce qu'elles pourraient y faire !).

– CLEAR : met à zéro toutes les variables d'un programme, cette commande est automatiquement exécutée lors d'un RUN.

– COMPILE : bien que ce Basic soit un Basic interprété, il est possible au moyen de cette commande de « compiler » le listing source sous une forme qui, bien que n'ayant aucun rapport avec du langage machine, occupe moins de place sur le disque et s'exécute plus rapidement. Cette pseudo-compilation traduit en fait les mots clés du Basic en des codes particuliers qui conduisent à un programme beaucoup plus condensé. La syntaxe est COMPILE « FICHIER » où FICHIER est le nom du fichier qui va recevoir le programme compilé, le lecteur par défaut est celui de travail et l'extension par défaut est BAC (BAsic Compilé). Il faut bien noter qu'une fois compilé, un programme ne peut plus être listé ni chargé en mémoire du Basic avac la commande LOAD ; il ne peut plus qu'être exécuté au moyen d'une forme particulière de la commande RUN. C'est un moyen pratique pour diffuser des programmes que vous avez réalisés et que vous ne voulez pas voir plagiés (à moins que le plagiaire ne possède un « décompilateur » car, malheureusement cela existe).

– CONT : permet de continuer l'exécution d'un programme qui a été interrompu par une instruction STOP, auquel cas l'on repart sur l'instruction qui suit immédiatement le STOP ou suite à l'arrêt d'un programme par un CNTRL C lors d'un INPUT, on repart alors au niveau de cet INPUT. La commande CONT ne peut faire repartir un programme interrompu par une erreur de même qu'elle ne fonctionnera pas si vous avez modifié le programme entre la cause de l'arrêt et la frappe de CONT.

– DISK : permet de passer à nouveau sous le contrôle du DOS et fait apparaître les trois signes + indiquant que celui-ci est en attente de commande. C'est la commande normale de sortie du Basic lorsque l'on a fini de travailler avec celui-ci.

– EXIT : permet de sortir du Basic et de passer sous le contrôle de TAVBUG09.

Le Basic n'est pas modifié par cette commande et si vous le relancez par un G à l'adresse 3, le programme qu'il contenait en mémoire ne sera pas modifié tandis qu'un G en 0 initialiserait à nouveau la mémoire du Basic et détruirait son contenu.

– LIST : permet de visualiser les lignes d'un programme. LIST employé seul fait visualiser tout le programme. LIST NN où NN est un numéro de ligne fait visualiser la ligne NN et LIST NN-MM fait visualiser depuis la ligne numéro NN jusqu'à la ligne numéro MM.

– LOAD : permet de charger en mémoire du Basic un programme contenu sur disquette. La syntaxe est LOAD « FICHIER » où FICHIER est le nom du fichier contenant le programme à charger. Le lecteur pris par défaut est celui de travail et l'extension prise par défaut est BAS (pour Basic, mais vous l'aviez deviné !). Cette commande n'a que pour effet de charger le fichier en mémoire du Basic et ne lance pas son exécution.

– NEW : prépare la mémoire du Basic pour recevoir un nouveau programme en effaçant tout ce qui s'y trouve contenu.

– RUN : lance l'exécution du programme contenu en mémoire. Toutes les variables sont mises à zéro et les instructions DATA sont initialisées. Il existe une forme particulière de cette commande qui est RUN « FICHIER » ; dans ces conditions, FICHIER représente le nom d'un fichier contenant un programme Basic « compilé » (voir la commande COMPILE ci-avant) et cette forme particulière de RUN est le seul moyen de lancer l'exécution d'un tel programme. FI-

DIM × (3,2) **réserve en mémoire le tableau suivant :**		
× (0,0) × (1,0) × (2,0) × (3,0)	× (0,1) × (1,1) × (2,1) × (3,1)	× (0,2) × (1,2) × (2,2) × (3,2)

Fig. 1. – Définitions d'un tableau à deux dimensions avec un DIM.

1° : () Expression entre parenthèses
2° : ↑ Elévation à une puissance
3° : – Changement de signe
4° : * et / Multiplication et division
5° : + et – Addition et soustraction
6° : = <>, >, <, <=, >= Opérateurs de relation
7° : NOT
8° : AND
9° : OR

Fig. 2. – Priorité relative des opérateurs du Basic.

CHIER est pris par défaut avec l'extension BAC (Basic Compilé) et sur le lecteur de travail.
– SAVE : permet de sauvegarder un programme sur disquette ; la syntaxe est SAVE « FICHIER » où FICHIER est le nom du fichier qui contiendra le programme ainsi sauvegardé. L'extension par défaut est BAS et le lecteur par défaut celui de travail. Attention, cette commande efface automatiquement, et sans avertir, tout fichier de nom identique existant déjà éventuellement sur la disquette.
– SCALE : permet de spécifier le nombre de chiffres à droite de la virgule qui seront conservés par le Basic. Le fait de spécifier 0 désactive cette commande et la valeur maximum est 6. La syntaxe est simplement SCALE N. L'utilisation de cette commande est à faire avec précaution. En effet, le Basic applique cette commande pour tous les nombres qui sont entrés en mémoire lors de la frappe d'un programme ou lors d'un LOAD de celui-ci, il n'est donc plus possible de changer ce paramètre une fois un programme Basic en mémoire puisque les variables ont déjà été traitées.
– TRON : met le Basic en mode pas à pas ; il imprime alors le numéro de chaque ligne au fur et à mesure de son exécution ce qui permet de mettre au point un programme au comportement imprévu (!).
– TROFF : remet le Basic en mode normal suite à un TRON.
– + : cette commande est particulière en ce sens qu'elle permet, tout en restant sous le contrôle du Basic, de faire exécuter des commandes au DOS ; ainsi, tout en étant sous BASIC, le fait de frapper +DIR fera afficher le répertoire des fichiers du lecteur de travail ; une fois cela réalisé, le contrôle sera rendu au Basic. Cette commande fonctionne avec toutes les commandes du DOS ; il faut cependant faire attention à ne pas utiliser de commandes du DOS qui utilisent le même espace mémoire que le Basic ; ainsi sur le DOS de base et les extensions, vous ne pouvez pas utiliser + avec EDIT, ASMB, SAVE.LOW, COPYSD. La principale utilisation de + est l'appel de RENUMBER dont nous verrons le rôle ci-après dans ce mode d'emploi.

Les instructions

Par opposition aux commandes, les instructions peuvent (et doivent) être utilisées dans un programme. Certaines fonctionnent en mode immédiat, d'autres ne fonctionnent qu'au sein d'un programme, vous comprendrez aisément pourquoi en lisant leur description ci-après..
– GOSUB <numéro de ligne> : le programme continue son exécution au numéro de ligne spécifié et ce jusqu'à ce qu'il rencontre une instruction RETURN qui le fait alors revenir à la ligne suivant immédiatement le GOSUB. Ce GOSUB n'est donc rien d'autre qu'un appel à un sous-programme.
– GOTO <numéro de ligne> : le programme continue son exécution à la ligne spécifiée mais c'est, contrairement au GOSUB, définitif, c'est-à-dire qu'il n'y aura pas de retour automatique à la ligne qui suit le GOTO. GOTO est un saut inconditionnel.
– ON <expression> GOSUB <suite de numéros de lignes> : l'expression est calculée et son résultat est tronqué à sa partie entière ; le programme exécute alors un GOSUB à la ligne déterminée comme suit : les numéros de lignes dans la liste ont une position allant de 1 à N s'il y a N numéros ; le numéro sélectionné est celui dont la position dans la liste est égale au résultat de l'expression. Ainsi si A = 4, ON A GOSUB 80, 90, 100, 110, 120 fera exécuter un GOSUB 110 puisque 110 occupe la quatrième position dans la liste.
– ON <expression> GOTO <liste de numéros de ligne> : fonctionne comme le ON GOSUB mais exécute un GOTO à la ligne déterminée au lieu d'un GOSUB.
– ON ERROR GOTO <numéro de ligne> : si, lors de l'exécution du programme une erreur de numéro de code inférieur à 50 se produit, le programme saute à la ligne spécifiée.
– RESUME <numéro de ligne> : permet de rendre le contrôle au programme principal après l'exécution de la partie de programme déclenchée par un ON ERROR GOTO. voyez le paragraphe spécialement consacré à ce sujet pour plus de détails.
– RETURN : termine impérativement tout sous-programme appelé par un GOSUB et permet au Basic de continuer l'exécution par la ligne qui suit le GOSUB ayant appelé le sous programme.
– IF <expression> GOTO <numéro de ligne> : l'expression est évaluée et, si elle est vraie, le Basic saute à la ligne spécifiée après le GOTO. Dans le cas contraire le Basic continue à exécuter normalement le programme à la ligne qui suit le IF GOTO.
– IF <expression> THEN <numéro de ligne> ou <instruction> : fonctionne de la même façon que le IF GOTO si THEN est suivi d'un numéro de ligne ; par contre THEN peut être suivi d'une instruction Basic qui sera alors exécutée si l'expression est vraie avant que le programme ne continue normalement. Ainsi IF A= 0 THEN PRINT « A est nul » fera imprimer A est nul si A = 0 avant de passer à la ligne suivante et ne fera rien imprimer du tout si A est différent de O.
– IF < expression> THEN <numéro de ligne ou instruction> ELSE <numéro de ligne ou instruction> : fonctionne comme IF THEN mais, dans ce cas, lorsque l'expression est fausse, l'on ne passe pas immédiatement à la ligne suivante, on exécute d'abord ce qui suit le ELSE.

Les instructions d'entrées/ sorties

Elles permettent au Basic de dialoguer avec l'utilisateur du programme et l'expérience montre qu'elles sont souvent les plus nombreuses dans un programme, elles sont au nombre de 3 non compris les entrées/sorties séquentielles décrites ci-après dans ce mode d'emploi.
– INPUT (« chaîne de caractères » ;) <liste de variables> : cette instruction fait imprimer la chaîne de caractères (si celle-ci existe puisqu'elle est optionnelle) suivie par un point d'interrogation puis attend autant de variables que spécifié par la liste de celles-ci. Les variables de la liste doivent être séparées entre elles par des virgules et être du type de ce que va répondre l'opérateur ; ainsi si vous voulez que l'opérateur réponde par une chaîne de caractères il faudra faire un INPUT A$ par exemple. L'opérateur doit fournir à une commande INPUT autant de variables que ce que vous avez spécifié ; ces variables seront affectées dans l'ordre de votre liste, elles doivent être frappées séparées par des virgules et terminées par un retour chariot. Le fait de fournir moins de variables que ce qui est demandé fait imprimer par le Basic un nouveau point d'interrogation en attente des variables manquantes. Le fait de fournir plus de variables que ce qui était demandé fait tout simplement ignorer les variables surnuméraires. Le fait de frapper un CNTRL C en réponse à un INPUT interrompt le programme et rend la main au Basic.

Voici quelques exemples d'input : INPUT « Quel est votre âge » ; A auquel il faudra répondre par une variable numérique ; INPUT « Une autre partie » B$ auquel il faudra répondre par une chaîne de caractères (généralement OUI ou NON dans un tel exemple) et enfin INPUT A,B, C$,D auquel il faudra fournir dans l'ordre deux variables numériques qui seront affectées respectivement à A et B, une chaîne de caractères qui sera affectée à C$ et enfin encore une variable numérique affectée à D.
– INPUT LINE <nom de variable type chaîne de caractères > : cette commande permet d'entrer une ligne entière comme chaîne de caractères sous le nom spécifié. Une seule variable est admise après INPUT LINE et aucun texte ne peut être imprimé contrairement à INPUT.
– PRINT (variable, ou ; variable, ou ; chaîne de caractères, ou ;... etc...) fait imprimer ce qui suit le PRINT en respectant les règles énoncées ci-après. Si PRINT n'est suivi d'aucune variable, un simple saut ligne sera effectué. La commande PRINT sépare l'écran du termi-

nal (ou le papier de l'imprimante) en 5 zones de 16 caractères. Quand plusieurs variables spécifiées après PRINT sont séparées par des virgules, chaque virgule fait passer à la zone suivante ; ainsi PRINT A, B fera imprimer la valeur de la variable A en position 1 sur l'écran (1re zone) et la valeur de la variable B en position 16 (2e zone) et ainsi de suite. Le fait de frapper plusieurs virgules est autorisé ; ainsi PRINT A, B fera imprimer B en position 32 (début de troisième zone). Si le nombre de variables spécifié conduit au-delà de la cinquième zone (position supérieure à 64) un retour chariot – saut ligne est automatiquement fait par le Basic qui continue l'impression sur la première zone de la ligne suivante. Lorsque les variables qui suivent le PRINT sont séparées par des points-virgules, elles sont imprimées les unes à la suite des autres sans utilisation des zones définies ci-avant. Ainsi si A = 2, PRINT « La valeur de A est » ; A fera imprimer : la valeur de A est 2 (remarquez que l'espace entre est et 2 avait été fourni par nos soins dans la définition de la chaîne de caractères). Les variables à imprimer peuvent être placées dans n'importe quel ordre après un PRINT et des virgules et des points-virgules peuvent apparaître sur une même ligne sans que cela ne cause d'erreur (hormis peut-être dans la présentation de vos résultats si vous n'avez pas fait assez attention !).

– PRINT USING <CHAINE>, <LISTE DE VARIABLES> : a un rôle analogue à la commande PRINT vue ci-avant mais permet un contrôle beaucoup plus précis du format de présentation des données au moyen de CHAINE qui est une image de la ligne à imprimer. LISTE DE VARIABLES est une liste de variables à imprimer, exactement comme dans une commande PRINT classique sauf que les séparateurs de variables, qu'ils soient des virgules ou des points-virgules n'ont plus aucune signification sauf en fin de la commande PRINT USING où ils reprennent la même signification que dans un PRINT classique. CHAINE peut contenir un certain nombre de caractères ayant une signification particulière décrite ci-après. Un point d'exclamation ordonne l'impression d'un seul caractère, ainsi : PRINT USING '!!!','01','AB','()' fera imprimer : 0 A (. L'utilisation du « back slash » (\) permet de demander l'impression d'autant de caractères que l'on souhaite ; le nombre de caractères sera égal au nombre de caractères compris entre les back slash + 2 (en d'autres termes, les back slash sont comptés dans le nombre de caractères). Ainsi : PRINT USING '/2345\','LE HAUT PARLEUR' fera imprimer : LE HAU (4 caractères compris entre les back slash + 2). Les caractères compris entre les back slash peuvent être quelconques et ne servent à rien d'autre qu'à spécifier le nombre de caractères à imprimer ; une bonne pratique consiste, comme dans l'exemple ci-dessus, à y mettre des nombres pour simplifier ensuite la lecture du listing.

Le dièse (#) permet de définir l'impression d'une valeur numérique ; ainsi PRINT USING '##.##',12.34567 fera imprimer 12.35. Le nombre est donc formaté en fonction de ce que lui imposent les signes #. S'il comporte des décimales, un arrondi automatique est réalisé comme dans l'exemple ci-avant. Si le nombre ne peut tenir dans le format demandé, comme par exemple : PRINT USING '#.##',25.34, le nombre est imprimé sans formatage et est précédé du signe « pour cent » (%). Attention, compte tenu de l'arrondi automatique cité ci-avant, une erreur peut être introduite involontairement ; par exemple, PRINT USING '#.#',9.99 ne fonctionnera pas normalement car 9.99 arrondi pour n'avoir qu'une décimale devient 10.0 qui ne tient plus dans le formatage demandé, une telle ligne ferait alors afficher % 10.0.

L'astérisque est utilisé pour combler les « blancs » qui peuvent être amenés à précéder un nombre lorsque l'on utilise le formatage précédent. C'est une façon de faire très employée par les banques sur leurs chèques. Ainsi PRINT USING '**#.##',12.3 fera imprimer **12.3. Il faût remarquer que les deux astérisques correspondent à un emplacement imprimable supplémentaire par rapport à celui défini par les dièses, ainsi PRINT USING '**#.##', 12.34 fera imprimer *12.34.

La virgule n'est pas très intéressante pour nous Français. En effet elle permet d'insérer des virgules au sein des nombres mais avec la signification américaine. En effet, chez eux notre virgule est remplacée par un point et ils utilisent la virgule pour séparer les blocs de trois chiffres des nombres, ainsi peuvent-ils écrire : 1000000 sous la forme 1,000,000. L'utilisation est fort simple : PRINT USING'###,###,###',1E6 fera imprimer 1,000,000

Le signe moins(–) utilisé dans cette chaîne peut avoir une signification particulière lors de l'utilisation de l'astérisque vue ci-avant. En effet pour représenter un nombre négatif lorsqu'il est précédé d'astérisque, une pratique courante veut que le signe moins soit placé à la fin du nombre. Dans ces conditions, nous écrirons par exemple : PRINT USING '**#.##–',–12.34 qui fera imprimer *12.34–.

Enfin, il est possible d'employer le caractère « flèche vers le haut » ou « accent circonflexe » (^) pour indiquer l'emploi de la notation scientifique. Il faut impérativement employer quatre flèches, ni plus ni moins car celles-ci doivent occuper exactement l'emplacement de la notation scientifique (E+XX ou E-XX). L'utilisation de cette notation est simple, ainsi écrirons-nous : PRINT USING '#.####^^^^', LOG(X) qui fera imprimer par exemple 1.2345E-01.

L'exposé de l'utilisation de cette commande peut sembler un peu hermétique à ceux d'entre vous qui ne connaissent pas le PRINT USING ; la meilleure solution pour se familiariser avec ses possibilités est de l'utiliser ; d'autant plus que vous n'avez pas besoin d'écrire un programme pour cela, il suffit de l'utiliser en mode immédiat pour voir aussitôt le résultat de vos essais.

Les boucles

Il n'existe en Basic qu'un moyen de faire des boucles automatiques, c'est en utilisant le classique FOR TO que nous allons étudier.

– FOR <variable>= <expression 1> TO <expression 2> (STEP <expression 3>) : fait exécuter toutes les instructions comprises entre cette ligne et celle contenant un NEXT (voir ci-après) autant de fois qu'il est spécifié par expression 1 et expression 2 selon le principe suivant. La variable spécifiée est la variable de boucle. Elle sert à compter le nombre de tours de boucle réalisés. Sa valeur initiale est fixée par expression 1 et à chaque tour de boucle, la valeur de la variable est augmentée par expression 3 spécifié après le STEP. Si STEP n'est pas précisé la valeur prise par défaut pour expression 3 est +1. La boucle est exécutée tant que la variable de boucle ne devient pas égale ou supérieure à expression 2 dans le cas où expression 3 est positive. Si expression 3 est négative, la boucle continue tant que la variable de boucle ne devient pas inférieure ou égale à expression 2. Quelles que soient les valeurs de expression 1, expression 2 et expression 3, la boucle sera toujours exécutée au moins une fois. Les boucles peuvent être imbriquées les unes dans les autres en nombre illimité (si ce n'est par la taille de la mémoire !) mais il faut alors utiliser des noms de variables de boucles différents. Une boucle peut être quittée prématurément par un GOTO mais il ne faut pas entrer dans une boucle autrement que par le FOR TO initial sinon les résultats sont imprévisibles car la valeur initiale de la variable de boucle n'est alors pas connue.

– NEXT <variable> : est utilisée pour spécifier où se termine une boucle et fait incrémenter la variable de boucle de la valeur spécifiée après le STEP.

Un exemple de boucle très simple :

```
10 FOR 1 % = 1 TO 10
20 PRINT 1 %, 1 % ^ 2
30 NEXT 1 %
```

fera imprimer en position 1 sur l'écran les nombres de 1 à 10 et en position 16 sur l'écran (à cause de la virgule entre 1 et 1^2) leurs carrés (STEP n'ayant pas été précisé, I augmente de 1 à chaque tour de boucle).

Remarquez que, lorsque la variable de boucle est un entier, ce qui est très souvent le cas, il y a intérêt à employer la notation utilisant le symbole « pour cent » pour imposer au Basic le codage de celle-ci en entier ; cela réduit la taille du programme et surtout l'accélère d'autant plus que le nombre de tours de boucle est grand. Ainsi, si l'on fait tourner les deux boucles suivantes :

10 FOR I = 1 TO 10000
20 NEXT I
et
10 FOR I % = 1 TO 10000
20 NEXT I %

la première met à peu près 19 secondes contre environ 6 secondes pour la seconde qui ne fait appel qu'à des entiers. L'écart est encore plus significatif si la variable de boucle est utilisée dans celle-ci pour des calculs.

Les instructions de fin de programmes

Il n'en existe que deux vu le rôle assez limité de ce genre d'instruction.

– END : termine l'exécution d'un programme lorsque l'on passe sur la ligne qui le contient. Sa présence est optionnelle, le Basic s'arrêtant alors sur la dernière ligne rencontrée. Un programme terminé par un END ne peut être relancé par une CONT.

– STOP : suspend l'exécution d'un programme et fait imprimer le message : STOP LIGNE XX ou XX est le numéro de la ligne contenant le STOP. Le programme peut être relancé par un CONT ; il part alors de l'instruction qui suit la ligne contenant le STOP.

Les assignations de valeurs

Il existe plusieurs instructions qui permettent de donner à des variables les valeurs de votre choix et ce, par programme.

– LET <variable> = <expression> : donne à la variable spécifiée la valeur de l'expression. Ce Basic admet de plus le LET implicite c'est-à-dire qu'il revient au même d'écrire : LET A = 2 que A = 2.

– DATA > nombre ou chaîne de caractères <(, <nombre ou chaîne de caractères>, etc.) : définit une liste de valeurs qui seront affectées aux variables rencontrées dans les instructions READ décrites ci-après. La quantité de nombres ou de chaînes de caractères qui suivent DATA n'est pas limité sinon par la longueur maximum de la ligne autorisée par le Basic. Un programme peut contenir autant d'instructions DATA que nécessaire ; leurs contenus seront considérés comme un ensemble global dont les éléments seront placés conformément à l'ordre d'apparition des diverses lignes DATA. Il est interdit de placer des DATA dans des lignes comportant plusieurs instructions. Les divers nombres ou chaînes de caractères doivent être séparés par des virgules. Si une chaîne de caractères comporte une virgule la chaîne complète doit être placée entre guillemets ; dans le cas contraire et si le DATA n'est suivi que par des chaînes de caractères, celles-ci peuvent être écrites sans guillemets ; ainsi : DATA JANVIER, FEVRIER, MARS sera valable.

– READ <variable>(, <variable>, etc.) : est le complément de DATA. Cette instruction affecte à la première variable spécifiée la première donnée rencontrée dans le premier DATA du programme et ainsi de suite. Le nombre de variables figurant dans un READ peut être inférieur au nombre de données spécifiées dans un DATA, les données surnuméraires seront ignorées. Par contre, le nombre de variables spécifiées dans un READ ne doit pas dépasser le nombre de données spécifiées dans l'ensemble des DATA du programme sinon il y a génération de l'erreur 31. Il faut aussi faire attention à ce que les variables définies après le READ soient du même type que les données qui vont leur correspondre dans les DATA (nombre pour une variable numérique, chaîne de caractères pour une variable chaîne de caractères).

De plus, lors de l'exécution du premier READ d'un programme, le pointeur dont dispose le Basic pour ces fonctions est remis à zéro et pointe donc sur la première donnée du premier DATA disponible.

– RESTORE : cette instruction remet à zéro le pointeur des données utilisé par les instructions READ ; c'est-à-dire que le premier READ qui va suivre un RESTORE, au lieu de prendre la donnée disponible à la suite dans la liste des DATA, va aller prendre à nouveau la première donnée du premier DATA disponible comme lors de l'exécution du premier READ.

Les instructions diverses

Elles n'entrent dans aucune des catégories précédentes et ce ne sont pas non plus des « fonctions » étudiées ci-après ; nous les avons donc groupées ici.

– DIM <variable 1> (N ou N, M) (, <variable 2> (P ou P, Q), etc.) : cette instruction a déjà été évoquée en début de ce mode d'emploi lors de la description des variables indicées. Elle doit être impérativement utilisée pour toutes les variables indicées apparaissant dans un programme et doit être placée avant la première utilisation de la ou des variables concernées.

– POKE <adresses >, <donnée> : place la donnée spécifiée à l'adresse indiquée. Ces deux valeurs devant être exprimées en décimal. L'adresse doit être comprise entre 0 et 65535 et la donnée doit être comprise entre 0 et 255. Cette instruction est à employer avec précaution car elle agit directement sur la mémoire ce qui peut avoir des conséquences fatales si vous touchez par erreur à la zone contenant le Basic ou ses variables.

– DPOKE fonctionne comme POKE mais la donnée est ici considérée comme étant sur 16 bits et elle peut donc varier entre 0 et 65535. Attention : si vous utilisez un DPOKE avec une donnée qui pourrait tenir sur 8 bits, le Basic codera quand même la donnée sur 16 bits ; ainsi DPOKE 100,1 fera placer 0001 à l'adresse 100 (et 101 donc) alors que POKE 100,1 ferait placer 01 à l'adresse 100.

– DEF FN <variable> (variable « bidon ») = <expression> : permet de définir autant de fonctions que vous le désirez avec les restrictions suivantes. La « variable » accolée à FN doit être une variable numérique (voir le début de ce mode d'emploi pour les noms autorisés) ainsi : FNAB ou FNX2 seront des fonctions valides mais pas FNA$ (A$ n'est pas numérique). La variable « bidon » doit être numérique et ne sert qu'à passer un paramètre à la fonction. Un nom de fonction peut être utilisé plusieurs fois dans un programme avec des définitions différentes car seule sa dernière définition est prise en compte. Une fonction définie de cette façon doit l'être en une seule ligne Basic, doit n'utiliser qu'une seule variable « bidon » de passage de paramètre et les fonctions utilisant des chaines de caractères ne sont pas admises. Un exemple :

10 DEF FNZZ (X) = X * 2
100 LET X = 10
200 Y = 250 + FNZZ (X)

donnera à Y la valeur 270 (250 plus 2 fois 10).

– REM (commentaires) : cette instruction n'en est pas une à proprement parler puisqu'elle ne sert qu'à placer du commentaire dans un listing. Le Basic se limite à reproduire celui-ci intégralement lors d'un LIST mais ne tient jamais compte de REM lors de l'exécution d'un programme. Attention ! REM consomme de la place mémoire surtout si vous faites comme certains auteurs qui utilisent des REM pour imprimer le mode d'emploi du programme sur le listing !

– DIGITS <TOTAL>(,VIRGULE) : permet de spécifier au Basic, et ce, indépendamment des commandes PRINT USING, le nombre de chiffres à imprimer.

TOTAL représente le nombre total de chiffres qui peut être compris entre 1 et 17 alors que le paramètre optionnel VIRGULE représente le nombre de chiffres à conserver à droite de la virgule. Il est évident que VIRGULE doit être inférieur ou au plus égal à TOTAL sinon une erreur sera générée. Il n'est, par ailleurs, pas recommandé d'utiliser 17 chiffres significatifs car l'exactitude du dernier ne peut être garantie vu le codage des données en mémoire ; il est plus sage de se limiter à 16. L'utilisation est simple ; ainsi DIGITS 4,3 suivi de PRINT PI fera imprimer 3.142 (le dernier chiffre est automatiquement arrondi). DIGITS peut être employé n'importe où dans un programme et c'est le dernier rencontré qui est valable à un instant donné.

– SWAP <VARIABLE1>, <VARIABLE2> : permet d'échanger les appellations de deux variables ; ainsi, si A=1000 et B=3 à un instant donné dans un programme ; après un SWAP A,B, A vaudra 3 et B vaudra 1000. Il est évident que SWAP ne peut agir que sur des variables de même type ; de plus SWAP ne doit pas être utilisée pour les variables de tableaux virtuels. Cette instruction est très utile dans les programmes de tri où elle peut faire gagner beaucoup de temps.

Les fonctions mathématiques

Elles sont classiques sur tout Basic digne de ce nom sauf peut-être l'arc « tangente » qui n'est pas toujours proposé.

– EXP (X) : fournit l'exponentielle de X c'est-à-dire « e » (la base des logarithmes népériens soit 2,718281828459045) à la puissance X. La valeur maximum autorisée pour X sans provoquer de débordement est 88,02969193111306 !

– LOG (X) : fournit le logarithme népérien ou naturel de X c'est-à-dire le logarithme à base « e ». Rappelons que pour passer en logarithme d'une autre base, il suffit de faire LOG (X) en base B = LOG (X)/LOG(B). X doit, bien sûr, être strictement positif.

– SQR (X) : donne la racine carrée de X qui doit être positif ou nul.

– SIN (X) : donne le sinus de X ; X étant exprimé en radians.

– COS (X) : donne le cosinus de X ; X étant exprimé en radians.

– TAN (X) : donne la tangente de X ; X étant exprimé en radians.

– ATN (X) : donne l'arc tangente de X ; la valeur fournie étant toujours comprise entre – PI/2 et + PI/2.

– PI : est la constante PI soit 3,1415926535897933 et peut être utilisée sous ce nom dans les calculs ; ainsi pour calculer la surface d'un cercle écrirons-nous :

LET S = PI * R ^ 2

– RND (X) : génère un nombre aléatoire compris entre 0 et 1 qui peut être exploité pour générer un nombre aléatoire sur n'importe quel intervalle en utilisant la formule (M – N) * RND (0) + N ; le nombre ainsi généré sera compris entre N et M. Lorsque X = 0 un nouveau nombre aléatoire est généré à chaque appel de RND (0) ; c'est l'utilisation normale de cette fonction. Si X est positif, RND (X) fournit le dernier nombre aléatoire qui a été généré. Si X est négatif, un nouveau nombre est généré à chaque appel de RND (X) mais chaque fois que X prend une valeur déjà utilisée au préalable, le même nombre aléatoire est généré.

– SGN (X) : donne le signe de X sous la forme suivante, SGN (X) est égal à + 1 si X est positif, à – 1 si X est négatif et à 0 si X est nul.

– ABS (X) : est la valeur absolue de X ; ABS (X) = X si X est positif et ABS (X) = – X si X est négatif.

– INT (X) : est le plus grand entier inférieur à X. Pour les nombres positifs pas de problème, INT (4,3) = 4 ; par contre attention aux nombres négatifs ; INT (– 5.3) = – 6.

– FRE(0) : donne le nombre d'octets libres disponibles. Le paramètre 0 n'a aucune signification mais sa présence est indispensable pour une syntaxe correcte.

– DATE$: donne la date qui a été fournie au DOS soit lors de son lancement, soit lors de commandes DATE ultérieures. DATE$ est une chaîne de caractères qui se présente sous la forme JJ-MMM-AA où JJ et AA sont le jour et les deux derniers chiffres de l'année alors que MMM sont les trois premières lettres du mois comme ce que vous voyez lors d'une commande DIR par exemple. Comme juin et juillet commencent pareil, juin est JUN et juillet JUI.

– PTR <NOM DE VARIABLE> : fournit l'adresse de la variable spécifiée dans NOM DE VARIABLE. Si celle-ci est un réel, l'adresse est celle du premier octet sachant que les réels occupent huit octets consécutifs ; les sept premiers étant la mantisse avec le signe en bit de poids fort du premier octet et le dernier octet étant l'exposant. Si la variable est un entier, l'adresse fournie est celle du premier octet des deux utilisés sachant qu'un entier est codé sur 16 bits en complément à deux. Si la variable est une chaîne de caractères, l'adresse fournie est celle d'un groupe de quatre octets qui est le descripteur de chaîne ; les deux premiers octets de ce groupe contiennent l'adresse de début de la chaîne tandis que les deux octets suivants contiennent sa longueur.

Les fonctions chaînes de caractères

Il est possible d'exécuter certaines fonctions décrites ci-après sur les chaînes de caractères.

– ASC (X$) : donne le code ASCII du premier caractère de la chaîne X$. Un 0 est fourni si la chaîne est la chaîne nulle, c'est-à-dire ne contient aucun caractère.

– CHR$ (I %) : est l'inverse de ASC puisqu'elle fournit le caractère dont le code ASCII est égal à la valeur de I. I doit donc être compris entre 0 et 255 ; la notation I % est à utiliser puisque I est toujours un entier.

– HEX (X$) : convertit la chaîne de caractères X$ supposée être de l'hexadécimal en son équivalent décimal, ainsi : HEX (« 80 ») donnera 128 puisque 80 hexadécimal est égal à 128 en décimal.

– LEFT$ (X$, I %) : prélève dans la chaîne X$ les I caractères de gauche ; ainsi si X$ = « HAUT PARLEUR », LEFT$ (X$, 4) = « HAUT », comme pour CHR$, I est à représenter avec le symbole % de façon à ce que le Basic le code en entier puisque c'est toujours le cas. Le fait de ne pas écrire I % ne génère pas d'erreur mais augmente la longueur du programme et le ralentit inutilement.

– LEN (X$) : donne le nombre total de caractères de la chaîne X$. Tous les caractères sont comptés y compris les espaces.

– MID$ (X$, I %, J %) : prélève dans la chaîne X$ les caractères compris entre les positions I et J pour former une nouvelle chaîne de caractères, ainsi si X$ = « HAUT PARLEUR », MID$ (X$, 5, 8) = « PAR », même remarque que ci-avant I % et J %.

– RIGHT$ (X$, I %) : prélève dans la chaîne X$ les I caractères de droite pour former une nouvelle chaîne, ainsi avec notre X$ précédent, RIGHT$ (X$, 7) = « PARLEUR ».

Pour les trois fonctions LEFT$, RIGHT$ et MID$, I doit être positif ou nul et inférieur à 32767 sinon il y a génération d'une erreur.

– STR$ (X) : fournit une chaîne de caractères qui représente la valeur numérique de X ; ainsi si X = 123, STR$ (X) = « 123 », ce 123 étant maintenant une chaîne de caractères.

– VAL (X$) : réalise l'inverse de STR$ et convertit la chaîne de caractères X$ en sa valeur numérique. VAL (X$) = 0 si le premier caractère de la chaîne autre qu'un espace est autre chose qu'un signe + ou – ou qu'un nombre.

– INCH$(I %) : a pour fonction de lire un caractère dans le fichier ou l'équipement référencé par I. Cette fonction s'utilise avec les définitions de canaux d'entrées/sorties décrites plus avant dans cette notice. Si I = 0, l'équipement utilisé sera la console du système.

– INSTR(I %, S$, P$) : recherche la sous-chaîne de caractères P$ dans la chaîne S$. Le

paramètre I indique à la commande à partir de quel caractère de la chaîne S$ la recherche doit commencer. La commande fournit une valeur numérique qui est le rang du caractère où commence la sous-chaîne P$ dans la chaîne S$. Si la recherche ne donne pas de résultat, la valeur 0 est fournie.

Les fonctions diverses

Nous n'avons pu les classer ailleurs, elles sont au nombre de 7.

– PEEK (I %) : donne le contenu décimal de la mémoire d'adresse spécifiée en décimal par I. I doit être compris entre 0 et 32767 sauf si la fonction HEX est utilisée auquel cas I peut atteindre 65535. La valeur fournie par PEEK est comprise entre 0 et 255.

– DPEEK fonctionne comme PEEK mais fournit une valeur sur 16 bits (même principe que POKE et DPOKE) et donne donc le mot de 16 bits qui commence à l'adresse spécifiée, ainsi si nous avons 12 en 100 et 34 en 101, DPEEK (100) fournira 1234.

– POS(I %) : indique la position de la « tête » d'impression de l'équipement spécifié par le numéro I ; cette définition d'équipement fait appel aux notions de canaux d'entrées/sorties vues ci-après. Si la « tête » d'impression est au début de la ligne, la valeur 0 est fournie.

– SPC (I %) : cette fonction ne doit être utilisée que lors d'un PRINT et fait imprimer I espaces sur le terminal ou l'imprimante.

– TAB (I %) : cette fonction ne doit être utilisée que lors d'un PRINT et a pour effet de déplacer le curseur ou la tête de l'imprimante sur la colonne I ; si la colonne est déjà dépassée, cette commande est ignorée. I doit être positif et inférieur à 256.

Utilisation du ON ERROR GOTO

ON ERROR GOTO est en fait une fonction qui peut être validée ou non à tout instant dans un programme. L'activation de cette possibilité a lieu en plaçant dans un programme ON ERROR GOTO <numéro de ligne> ; dans cette condition, toute erreur de code inférieur à 50 (strictement) fera sauter le programme à la ligne spécifiée pour y exécuter l'ensemble d'instructions s'y trouvant. Cet ensemble d'instructions sera impérativement terminé par un RESUME qui fera alors reprendre l'exécution soit au niveau de ce qui a causé l'erreur, soit à la ligne spécifiée après le RESUME car l'on peut aussi écrire RESUME (numéro de ligne).

Le programme de traitement de l'erreur peut faire appel à deux variables positionnées par le Basic pour savoir ce qui s'est passé ; ces deux variables sont ERR et ERL. ERR est égale au numéro de code de l'erreur tandis que ERL est égale au numéro de la ligne de programme ayant causé l'erreur.

La fonction ON ERROR GOTO peut être désactivée à tout instant dans un programme en écrivant simplement ON ERROR GOTO ou ON ERROR GOTO 0. Dans ces conditions, toutes les erreurs qui suivent provoqueront l'arrêt du programme et l'impression d'un message d'erreur.

Les numéros de codes d'erreur sont ceux qui sont imprimés si vous n'avez pas mis en place le fichier ERBAS ou ERBASA.SYS en lieu et place de ERREURS.SYS, puisque ces numéros vous sont généralement inconnus, la figure 3 vous donne la correspondance entre les messages et les numéros que l'on peut trouver dans ERR.

La fonction USER

Cette possibilité du Basic est à réserver aux programmeurs expérimentés ; elle permet d'appeler un programme en langage machine à partir du Basic avec passage de paramètre dans les deux sens. Son fonctionnement est strictement conforme à ce qui suit.

La syntaxe est A (ou LET A) = USR (B). Le Basic évalue alors la variable B et la convertit en un entier sur 16 bits en complément à deux puis place la valeur obtenue en MEMAX-4. Il va ensuite chercher en MEMAX-2 une adresse et effectue un JSR à cette adresse où il espère trouver un pro-

CODE	SIGNIFICATION
1	CODE DE FONCTION FMS ILLEGAL
2	LE FICHIER DEMANDE EST DEJA UTILISE
3	LE FICHIER SPECIFIE EXISTE DEJA
4	LE FICHIER DEMANDE EST INTROUVABLE
5	ERREUR DANS LE REPERTOIRE DES FICHIERS - RECHARGEZ LE DOS
6	LE REPERTOIRE DES FICHIERS EST PLEIN
7	TOUTE LA PLACE DISPONIBLE SUR LE DISQUE A ETE UTILISEE
8	FIN DE FICHIER RENCONTREE EN LECTURE
9	ERREUR DE LECTURE SUR LE DISQUE
10	ERREUR D'ECRITURE SUR LE DISQUE
11	LE DISQUE OU LE FICHIER EST PROTEGE EN ECRITURE
12	LE FICHIER EST PROTEGE - EFFACEMENT IMPOSSIBLE
13	BLOC DE CONTROLE DE FICHIER ILLEGAL
14	APPARITION D'UNE ADRESSE DISQUE ILLEGALE
15	LE LECTEUR DEMANDE N'EXISTE PAS
16	LE LECTEUR DEMANDE N'EST PAS PRET
17	LE FICHIER EST PROTEGE ACCES REFUSE
18	LE FICHIER EST PROTEGE ACCES REFUSE
19	POINTEUR D'ACCES DIRECT ERRONE
20	FMS INACTIF - RECHARGEZ LE DOS
21	LE NOM DE FICHIER SPECIFIE EST INCORRECT
22	ERREUR DE FERMETURE D'UN FICHIER
23	DEBORDEMENT DE LA TABLE DES SECTEURS - DISQUE TROP SEGMENTE
24	LE NUMERO D'ENREGISTREMENT DEMANDE N'EXISTE PAS
25	FICHIER DETERIORE
26	ERREUR DE SYNTAXE DANS LA LIGNE DE COMMANDE
27	COMMANDE INTERDITE PENDANT L'IMPRESSION
28	CONFIGURATION MATERIELLE INSUFFISANTE
30	TYPE DE DONNEE INCORRECT
31	NOMBRE DE "DATA" INSUFFISANT LORS D'UN "READ"
32	MAUVAIS ARGUMENT DANS UNE COMMANDE "ON"
34	ARRET SUR UN CNTRL C
37	ARRET SUR LA SEQUENCE "ESCAPE RETOUR CHARIOT"
40	MAUVAIS NUMERO DE FICHIER
41	FICHIER DEJA OUVERT
42	LE FICHIER DOIT ETRE OUVERT PAR UN "NEW" OU UN "OLD"
43	LE FICHIER N'A PAS ETE OUVERT
44	ERREUR DANS LE MOT D'ETAT DU FICHIER
45	ERREUR DANS LA DIMENSION DU FICHIER
46	EXTENSION DE FICHIER SEQUENTIEL IMPOSSIBLE
47	NUMERO 0 NON AUTORISE
48	IL FAUT UTILISER UN FICHIER "ALEATOIRE"
50	COMMANDE INCONNUE

Fig. 3. – Correspondance entre messages d'erreurs et numéros de code pour exploitation par ON ERROR GOTO.

gramme en langage machine sauf si le contenu de MEMAX-2 est nul. Le programme ainsi appelé doit se terminer impérativement par un RTS et doit restituer la pile S dans l'état où elle se trouvait lors de l'appel. De plus, ce programme ne doit pas utiliser plus de 256 octets de pile. Lors du retour au Basic, celui-ci va lire la valeur contenue en MEMAX-4 (et MEMAX-3 car il prend une valeur sur 16 bits) et donne cette valeur à la variable A de notre exemple précédent.

L'adresse appelée MEMAX dans cette description de la commande USER n'est autre que l'adresse la plus élevée utilisable ; comme le DOS commence en C000, MEMAX est égal à BFFF.

Les entrées/ sorties séquentielles

Elles constituent les formes les plus simples de manipulation de fichiers et n'en sont pas moins performantes pour autant. Elles permettent de créer ou de lire des fichiers existants. Sauf spécification contraire de votre part, ces fichiers sont lus ou créés sur le lecteur de travail par défaut et avec l'extension DAT. Les fichiers ainsi créés peuvent être lus et manipulés par d'autres commandes du DOS sans aucune restriction.

– La commande OPEN : elle doit être employée avant toute utilisation d'un fichier dans un programme Basic que ce soit pour le lire ou le créer. La syntaxe en est OPEN OLD <CHAINE> AS <NUMERO> ou OPEN NEW <CHAINE> AS <NUMERO> où CHAINE est le nom du fichier à lire ou à créer, étant entendu que le lecteur par défaut est celui de travail et l'extension par défaut est DAT. NUMERO est le numéro de canal qui sera affecté au fichier et au moyen duquel il lui sera ensuite fait référence. Ce numéro est compris entre 1 et 12 (1 et 12 étant inclus) ce qui signifie qu'à un instant donné vous ne pouvez avoir plus de 12 fichiers ouverts en même temps (ce qui n'est déjà pas mal !). OPEN OLD est à utiliser pour ouvrir un fichier qui existe déjà en lecture. Si le fichier n'est pas trouvé, une erreur sera générée (erreur de code 4 que vous pouvez traiter avec le ON ERROR GOTO). Ainsi OPEN OLD « DEMO » AS 1 ouvrira en lecture le fichier DEMO.DAT et lui affectera le numéro de canal 1. OPEN NEW est à utiliser pour ouvrir un fichier en écriture, fichier qui est réputé ne pas déjà exister sur le disque. Si tel n'était pas le cas, le fichier existant déjà sous ce nom serait effacé sans avertissement. Ainsi, OPEN NEW DEMO AS A ouvrira en écriture le fichier DEMO.DAT (donc créera le fichier DEMO.DAT) et lui affectera le numéro de canal défini par la variable A.

Il faut remarquer que la commande OPEN ne fait que préparer les ouvertures de fichiers et que ceux-ci sont utilisés uniquement lorsque vous y accédez réellement : cela signifie que les erreurs pouvant éventuellement être générées n'apparaîtront pas au niveau du OPEN mais seulement au niveau de la première utilisation du canal ainsi défini.

– Les sorties séquentielles : elles utilisent une forme particulière de la commande PRINT (ou PRINT USING) sous la forme :
PRINT #<NUMERO>, <USING <CHAINE>) <LISTE DE VARIABLES> ou NUMERO est le numéro du canal sur lequel doit se faire la sortie et où LISTE DE VARIABLES est la liste des variables à sortir (exactement comme dans une commande PRINT classique). Le USING facultatif entre parenthèses est là pour montrer que PRINT USING peut aussi s'utiliser avec un numéro de canal. Précisons que le numéro de canal n'est pas obligatoirement spécifié par un chiffre (encore que ce soit le plus souvent le cas) mais que ce peut être une variable ou une expression égale à ce nombre. Voici un exemple d'utilisation de cette commande :

```
10 OPEN NEW « DEMO » AS 1
20 PRINT #1, «PREMIERE LIGNE DU FICHIER DEMO »
30 PRINT #1, « DEUXIEME LIGNE DU FICHIER DEMO »
```

Si vous exécutez ce programme vous allez créer sur votre lecteur de travail un fichier DEMO.DAT qui contiendra deux lignes : PREMIERE LIGNE DU FICHIER DEMO et DEUXIEME LIGNE DU FICHIER DEMO. Ce fichier sera strictement compatible du DOS et vous pourrez le lister, l'éditer ou en faire ce que vous voulez.

– Les entrées séquentielles : elles font appel à une forme particulière de la commande INPUT ou INPUT LINE exactement comme les sorties faisaient appel à un PRINT modifié. La syntaxe est tout simplement :
INPUT #<NUMERO> <LISTE DE VARIABLES> ou INPUT LINE #<NUMERO>, <VARIABLE CHAINE DE CARACTERES> où NUMERO est le numéro du canal à utiliser ; celui-ci pouvant être indiqué en clair ou via une variable ou une expression comme ci-avant. Aucun point d'interrogation n'apparaît lors de ces commandes puisque les données demandées sont prises directement sur le canal spécifié qui est un fichier disque. Ainsi :

```
10 OPEN OLD « NOMBRES » AS 2
20 INPUT #2, A, B
```

fera lire les valeurs des variables A et B dans le fichier NOMBRE.DAT pris sur le lecteur de travail. Le contenu du fichier doit être présenté exactement comme si vous frappiez au clavier les réponses demandées par le INPUT ainsi les diverses valeurs doivent être séparées par des virgules ou doivent se trouver à raison d'une par ligne. En particulier, les données contenues dans le fichier concerné doivent être du même type que celles demandées par le INPUT sinon il y aura génération d'un message d'erreur.

– La commande CLOSE : permet de fermer un canal qui a été préalablement ouvert. Cette commande doit impérativement être utilisée avant la fin d'un programme sur tous les canaux que celui-ci a utilisés. Par ailleurs, lorsque vous avez fini de travailler avec un canal, il est conseillé de le fermer aussitôt car cela libère son numéro pour, si nécessaire, ouvrir un autre canal. Le fait de fermer un canal qui n'a pas été ouvert fait générer un message d'erreur. La syntaxe est très simple :
CLOSE <NUMERO> (,NUMERO, NUMERO, ...) où NUMERO est (sont) le (les) numéro(s) du (des) canal(aux) à fermer. L'on peut en effet fermer autant de canaux qu'on le souhaite avec un seul CLOSE. Les numéros peuvent être spécifiés en clair ou via un nom de variable ou une expression comme indiqué ci-avant pour OPEN, PRINT et INPUT.

– INPUT sur le canal 0 : nous avons dit que les numéros de canaux allaient de 1 à 12, il existe cependant la possibilité de faire un INPUT sur le canal 0. Cette utilisation est particulière en ce sens qu'il ne faut pas faire de OPEN sur ce canal et qu'un INPUT #0 équivaut à un INPUT normal à partir du terminal mais aucun point d'interrogation n'est généré et le retour chariot – saut ligne automatique lors de l'INPUT classique n'a pas lieu lors de l'INPUT #0. Cela permet des contrôles précis du mouvement du curseur lors des entrées de caractères et c'est très agréable dans certains programmes.

– PRINT sur le canal 0 : de même que l'INPUT sur le canal 0, cette commande a une signification particulière. Si vous faites un PRINT #0 à la place d'un PRINT normal sans avoir fait de OPEN 0 au préalable, votre PRINT #0 aura le même effet que le PRINT classique et fera imprimer ce que vous désirez sur le terminal du système. Si, par contre, vous avez fait avant un OPEN « 0.PRINT » AS 0, la commande PRINT #0 fera imprimer sur l'imprimante pilotée par le fichier PRINT.SYS du DOS. En d'autres termes, le PRINT #0 permet de diriger les impressions sur autre chose que le terminal du système. L'utilisation de ce PRINT #0 est généralement réservée à l'imprimante : la syntaxe étant alors : OPEN « 0.PRINT » AS 0 avant de faire des PRINT #0. Remarquez qu'il faut spécifier le numéro du lecteur dans le OPEN puisque PRINT.SYS est sur le disque système (en général) et non sur le disque travail ; par contre l'extension n'a pas besoin d'être fournie, le

BASIC sait que dans ce cas c'est SYS. Cette façon de faire est très souple puisque le Basic n'a plus à savoir quel type d'imprimante vous utilisez. Il vous suffit, comme nous l'avons expliqué le mois dernier, d'avoir écrit votre fichier PRINT.SYS pour votre imprimante pour que toutes les commandes du DOS dont le Basic puisse l'utiliser.

– La commande KILL : permet d'effacer un fichier existant sur le disque. Cette commande ne doit pas être confondue avec + suivi de DELETE ; en effet alors que + suivi de DELETE ne peut fonctionner qu'en mode immédiat, la commande KILL peut être incluse dans un programme. La syntaxe est : KILL « NOM DE FICHIER » où NOM DE FICHIER est le nom du fichier à effacer dont l'extension par défaut est BAS et le lecteur par défaut celui de travail. Cette commande peut être utilisée dans un programme, nous venons de le dire, mais aussi en mode immédiat. Attention, cette commande ne pose aucune question avant de s'exécuter !

– La commande RENAME : elle a une fonction analogue à celle de la commande RENAME du DOS. Elle peut s'utiliser en mode immédiat ou dans un programme et a pour effet de changer le nom d'un fichier. La syntaxe est : RENAME « FICHIER 1 », « FICHIER 2 » où FICHIER 1 est le nom de fichier à changer et où FICHIER 2 est le nouveau nom à donner à FICHIER 1. Si FICHIER 1 n'existe pas, un message d'erreur est généré. Les extension et lecteur par défaut sont analogues à ceux de la commande KILL ci-avant.

– La commande RENUMBER : par opposition aux deux précédentes, cette commande ne peut être utilisée qu'en mode immédiat et ne doit pas être employée dans un programme. Elle s'utilise conjointement à la commande + décrite en début de notice sous la forme : + RENUMBER (DEBUT), (INCREMENT) et a pour effet de refaire automatiquement la numérotation du programme en mémoire du Basic. Si DEBUT et INCREMENT ne sont pas spécifiés, la commande va refaire la numérotation en affectant 10 comme numéro de la première ligne de votre programme et en faisant croître les numéros de 10 en 10 comme le veut la pratique classique. Vous pouvez préciser en DEBUT la valeur de ce premier numéro de ligne et en INCREMENT la différence entre deux numéros de ligne consécutifs. Cette commande effectue une vraie renumérotation de vos programmes en ce sens que tous les numéros de lignes sont changés, ce qui fait que le programme ainsi traité est tout de suite prêt à fonctionner sans qu'il soit nécessaire de modifier quoi que ce soit. Attention, pour les programmes longs, cette commande peut demander plusieurs dizaines de secondes voire jusqu'à quelques minutes pour s'exécuter.

L'utilisation de l'imprimante avec le Basic

Mais non ! Rassurez-vous, l'auteur ne radote pas encore et nous savons très bien que l'utilisation de l'imprimante a été décrite ci-avant lors de la commande PRINT #0 ; mais nous savons aussi, par expérience, que certains d'entre vous ne lisent ces articles que très rapidement ; alors, pour ceux-ci, nous avons créé ce paragraphe qui n'a d'autre but que de les renvoyer quelques lignes plus tôt dans ce texte, au niveau du PRINT sur le canal 0 pour y lire tout ce qui concerne l'imprimante.

Les possibilités de programmation évoluées

Les possibilités de travail avec les disques ne sont pas limitées aux entrées/sorties séquentielles vues ci-avant et il est en particulier possible de créer des tableaux virtuels sur disque, des fichiers à accès aléatoire, etc. Cet article étant déjà bien volumineux, nous sommes obligés de reporter cette partie du mode d'emploi dans notre prochain numéro où nous verrons donc toutes ces possibilités de programmation évoluée.

Réponses à vos questions

Vous êtes nombreux à nous poser des questions relatives aux circuits imprimés proposés par Saint-Ignan Informatique. Ces circuits ne sont pas les mêmes que ceux de FACIM et correspondent à une réalisation qui, sans différer beaucoup de celle que nous avons décrite dans ces pages, est organisée autrement, les circuits réalisés l'étant au format Exorciser et non à notre format « spécial ». Pour répondre à une question qui revient souvent, les circuits décrits jusqu'à maintenant dans cette revue sont donc fournis par FACIM et ce exclusivement puisque cette société est propriétaire du dessin de ceux-ci. Les circuits proposés par Saint-Ignan seront, soit décrits ultérieurement dans cette revue, soit livrés accompagnés d'un descriptif. Les logiciels que nous avons présentés tournent sur le système équipé des cartes telles que nous les avons décrites ; ces mêmes logiciels existeront aussi, avec les mêmes possibilités sur un système équipé des cartes Saint-Ignan.

Certains d'entre vous se sont plaints, avec juste raison, des délais que leur a imposés l'auteur de ces lignes en janvier, février, mars pour la fourniture de logiciels. Ce délai dont il vous prie de bien vouloir l'excuser a eu deux causes : son changement de domicile début 83 et la saturation causée par le nombre de vos demandes. C'est ce dernier point qui nous a conduit à confier la distribution des logiciels à une société spécialisée.

A propos des fournisseurs de matériel pour cette réalisation, sachez que l'auteur attache une grande importance au courrier que vous lui adressez concernant tel ou tel fournisseur, que ce soit en bien ou en mal. En particulier, lorsque vous nous faites parvenir des critiques au sujet de l'un d'entre eux, soyez assurés qu'elles lui sont transmises pour suite à donner. Par contre, nous déplorons la pratique peu élégante consistant, lorsque vous avez une critique à formuler, à adresser une lettre à l'auteur avec copie au rédacteur en chef du journal quand ce n'est pas à la direction de celui-ci.

A propos du boîtier proposé par Saint-Ignan Informatique, précisons qu'il est différent de celui d'Incodec (moitié plus petit) mais qu'il offre les mêmes possibilités tant au niveau du montage des cartes et des lecteurs de disquettes qu'au niveau des connexions en face arrière. Ce boîtier peut, de plus, recevoir les cartes au format Saint-Ignan, les cartes au format Exorciser (Motorola) et les cartes au format Facim et ce sans avoir à faire de mécanique. Vous pouvez donc très bien monter le système tel que nous l'avons décrit jusqu'à maintenant dans ce boîtier si vous le désirez. Le transformateur proposé avec est un peu plus puissant que le modèle initial prévu pour permettre le passage aux disques durs Winchester ultérieurement. De plus, le dessin du circuit imprimé de l'alimentation a été revu mais reste conforme au schéma que nous vous avions proposé, cela afin de permettre le raccordement de celle-ci au circuit imprimé du bus, au transformateur et aux interrupteurs au moyen de connecteurs enfichables.

Conclusion

Nous développerons un peu plus longuement les possibilités et les particularités de ce boîtier dans un prochain numéro. Pour l'instant, nous vous présenterons dans le prochain article les possibilités évoluées du Basic et le schéma de la carte IPT09 évoquée en introduction. Nous essaierons de faire suivre cela par l'étude de la carte de visualisation couleur graphique haute résolution si les délais de réalisation du prototype en circuit imprimé de celle-ci nous le permettent.

... à suivre...

C. TAVERNIER

Dernière minute

Nous sommes heureux d'informer nos amis lecteurs de Lyon et de sa région que la société CREE, 3, rue Bossuet, 69006 Lyon dispose de tous les composants nécessaires à la réalisation de notre ordinateur individuel.