COMPARAISON LSE/BASIC

INTRODUCTION

LSE et BASIC sont les deux langages retenus par le ministère de l'Education pour être utilisés sur les micro-ordinateurs équipant les lycées.

Nous proposons au lecteur une description succincte des deux langages permettant ensuite d'étayer une comparaison de leur spécificité propre. Nous ne comparerons pas des réalisations particulières, mais les projets de norme dont devraient se rapprocher les réalisations...

I. — LSE

1. — Généralités

LSE (Langage Symbolique d'Enseignement) a été défini en 1971 à la demande du ministère de l'Education.

La définition de ce langage a été faite avec la volonté de satisfaire trois objectifs principaux :

— que le langage soit simple à apprendre, le but cherché étant que des non-spécialistes puissent l'aborder et l'utiliser rapidement ;

— que le langage, bien que simple, ne soit pas « simpliste ». C'est-à-dire qu'il devait permettre de programmer un grand éventail d'applications de complexité variée, les applications allant du calcul scientifique classique aux programmes d'aide à l'enseignement dans des domaines aussi divers que le latin, les sciences naturelles, l'économie...

Les applications de ce langage ne se limitent d'ailleurs pas à l'enseignement et il est utilisable dans de nombreux autres domaines ;

— que le langage puisse être utilisé efficacement sur de petits ordinateurs.

Dans la réalisation d'une application, le langage est une chose importante certes, mais le programmeur a aussi à utiliser un système.

Les définisseurs de LSE ont donc défini un Langage de Commande et un environnement prévus pour optimiser la mise au point et l'utilisation de programmes rédigés en LSE.

Cet ensemble de commandes est suffisamment complet et l'utilisateur n'a pas besoin, dans la plupart des cas, de connaître les spécificités de la machine hôte.

On peut dire qu'on a ainsi défini une « Machine LSE ». Le mode de travail de cette Machine LSE est le mode conversationnel.

Ce mode conversationnel a été rendu possible par l'apparition des systèmes en temps partagé, et l'arrivée des systèmes individuels est une nouvelle incitation à la réalisation de systèmes conversationnels.

Dans la suite de cet exposé, nous présenterons d'abord rapidement un exemple d'application, ceci pour permettre d'emblée de situer le langage.

Nous décrirons ensuite les différents aspects du langage de programmation, puis du langage de commande.

2. — Exemple de programme

Le programme que nous présentons lit un nombre en numération romaine et imprime sa valeur en numération décimale :

```
 1 * LECTURE D'UN NOMBRE EN CHIFFRES ROMAINS
 2 *
 3 CHAINE DICO,ROM,CAR
 5 DICO ← ' M1ØØØD5ØØC1ØØL5ØX1ØV5I1 '
 7 AFFICHER ' NOMBRE ROMAIN ? : ' ; LIRE ROM ; S←Ø
 8 FAIRE 19 POUR I←1 JUSQUA LGR (ROM)
 9 CAR←SCH (ROM,I,1) ; IND←POS (DICO,1,CAR)
13 Si IND=Ø ALORS DEBUT AFFICHER [' ERREUR ...'] ;
   ALLER EN 7 FIN
15 POIDS←CNB (DICO,IND+1)
18 S←S+POIDS
19
21 AFFICHER ' EQUIVALENT DECIMAL : ',S
23 ALLER EN 7
```

La lecture rapide de ce programme permet de voir qu'un programme écrit en LSE utilise un vocabulaire français et est découpé en lignes, chaque ligne ayant un numéro. Ces numéros servent à indiquer dans quel ordre les lignes sont exécutées. Ils servent aussi d'étiquette pour les instructions de contrôle.

Une ligne peut comporter plusieurs instructions, celles-ci étant alors séparées par des points-virgules.

Une instruction commençant par un astérisque est un commentaire. Un commentaire s'étend jusqu'à la fin de la ligne.

Les lignes de numéro 1 et 2 sont donc ici des commentaires.

La méthode de conversion consiste à extraire de la chaîne chacun des chiffres romains et à ajouter à la variable S leur équivalent numérique ; elle ne traite donc pas les cas IV, IX, etc.

3. — Description rapide de LSE

Le LSE a été conçu pour être compilé ligne par ligne. Il est à noter que les déclarations de type sont des instructions exécutables et peuvent être mises n'importe où dans le programme, pourvu qu'elles soient exécutées avant utilisation des variables qu'elles déclarent. Les variables numériques ne sont pas déclarées (déclaration par défaut).

Les noms des variables utilisées dans un programme LSE peuvent avoir jusqu'à 5 caractères.

Le premier doit être une lettre, les autres peuvent être une lettre ou un chiffre.

LSE gère quatre types de variables :

— les variables chaînes,
— les variables numériques,
— les variables booléennes,
— les variables numériques étendues.

3.1. — Traitement des chaînes

Une importance particulière a été donnée au traitement des chaînes. Leur utilisation permet de simplifier la programmation de beaucoup de problèmes.

Une variable chaîne X se déclare à l'aide de l'instruction CHAINE X.

Une chaîne LSE est une suite de caractères. Chaque caractère est repéré par sa position. Le caractère le plus à gauche a la position de numéro un.

Un caractère peut avoir 256 valeurs possibles. La longueur d'une chaîne est le nombre de caractères que contient la chaîne ; elle peut être quelconque (nulle en particulier).

A la différence d'autres langages, LSE définit le codage interne utilisé pour les caractères imprimables et de service ; il est dérivé du code ISO 7 bis (ASCII).

On peut définir des constantes-chaînes : les constantes composées de caractères imprimables sont représentées par ces caractères encadrés par des apostrophes.

Si une apostrophe est présente dans la chaîne, on la double.

Les caractères non imprimables sont représentés par leur équivalent numérique encadré par des points.

Exemples : 'VOICI NANTES'
.13 10.

Le dernier exemple est la représentation de la chaîne composée des caractères retour-chariot et nouvelle ligne de code 13 et 10.

On peut affecter une chaîne à une variable-chaîne par une instruction d'affectation :

MOIS ← 'DECEMBRE'

Il existe un seul opérateur, l'opérateur de concaténation, noté !, qui permet d'accoler deux chaînes pour en faire une nouvelle.

Exemple : '12 JANVIER '!' 1980'

Résultat : la chaîne 12 JANVIER 1980.

On dispose de fonctions pour effectuer les traitements sur chaînes ; ces fonctions se répartissent en deux catégories :

— les fonctions rendant comme résultat un nombre,
— les fonctions rendant comme résultat une chaîne.

Pour alléger les descriptions, nous prendrons les notations suivantes :

Org Une expression chaîne donnant la valeur de la chaîne d'origine. Sa valeur sera notée Corg.

Dep Une expression numérique donnant la position de départ dans la chaîne origine. Sa valeur sera notée Vdep.

Crit (Pour critère), un paramètre qui peut être soit une chaîne, soit une expression numérique. Sa valeur sera notée Crit ou Vcrit selon les cas.

Cr Variable de compte rendu numérique. Une valeur lui est affectée par la fonction pour préciser le déroulement de l'évaluation de la fonction. La présence de cette variable est optionnelle.

Ch Expression chaîne de valeur Cch.

Ea Expression numérique de valeur Vea.

LSE dispose de 18 fonctions chaînes :

ASP (Ch1,Ch2)	Appel de procédure spéciale
CCA (Ea,Ch)	Conversion d'un nombre en caractères
CNB (Org,Dep,Cr)	Conversion d'une chaîne numérique en nombre
DAT ( )	Date et heure courantes
EQC (Ea)	Equivalent caractère
EQN (Org,Dep)	Equivalent numérique

GRL (Org,Dep,Ca)	Groupe de lettres
ICH (Org)	Inversion de chaîne
LGR (Ch)	Longueur de chaîne
MCH (Org,Dep,Crit,Ch)	*Modification de chaîne*
POS (Org,Dep,Ch)	Position
PTR (Org,Dep,Ch)	Pointeur
RCC (Org,Ch1,Ch2)	Relations entre chaînes
REP (Org,Ea)	Répétition
SCH (Org,Dep,Crit,Cr)	Sous-chaînes
SKP (Org,Dep,Crit)	Saut
TMA (Org)	Transformation en majuscules
TMI (Org)	Transformation en minuscules

Parmi ces fonctions, nous n'en détaillerons que quelques-unes :

3.1.1. — SCH (Org,Dep,Crit,Cr)

Pour **S**ous-**CH**aîne - résultat chaîne.

Donne une chaîne (sous-chaîne) extraite de la chaîne Corg à partir du caractère défini par Vdep.

La *fin de la sous-chaîne est définie par Crit.*

— Si Crit est une expression arithmétique, Vcrit donne la longueur de la sous-chaîne.

Exemple : SCH(' LE CHATEAU ', 3, 7) = ' CHATEA '

— Si Crit est une chaîne, les caractères qui la composent ne doivent pas faire partie de la sous-chaîne, c'est-à-dire que celle-ci s'arrêtera à la rencontre du premier caractère appartenant à Crit. On trouvera dans Cr (s'il existe dans l'appel) la position du premier caractère *ne faisant pas partie de la sous-chaîne extraite.*

Exemple : SCH (' LE CHATEAU ', 3, THR ', I)
= ' C , J=5.

3.1.2. — PTR (Org,Dep,Crit)

Pour **PoinTeuR** - résultat numérique.

Fonction permettant de savoir où s'arrête une sous-chaîne de Corg. La sous-chaîne est définie par Crit (voir SCH).

Exemple : PTR (' ÇA SUFFIT ', 3, ' F ') = 6.

Si Crit est omis, la valeur de PTR est la position du *premier caractère non alphabétique situé après la position* de départ.

3.1.3. — MCH (Org,Dep,Crit,Ch)

Pour **M**odification de **CH**aîne - résultat chaîne.

Donne une copie de la chaîne Corg dans laquelle la sous-chaîne définie par Dep et Crit (voir SCH) est *remplacée par Cch.*

Exemple : MCH (' JE SUIS LA ', 4, ' ', ' SERAI ')
= JE SERAI LA.

3.1.4. — POS (Org,Dep,Ch)

Pour **POS**ition - résultat numérique.

Donne la position de la première occurrence de Cch dans la chaîne Corg, à partir du caractère de rang Vdep.

Exemple : POS (' CABCABD ', 3, ' AB ') = 5.

3.1.5. — GRL (Org,Dep,Cr)

Pour **GR**oupe de **L**ettres - résultat chaîne.

Donne une chaîne extraite de la chaîne ORG et ne comportant que des lettres. Dans ce cas, Cr est la position du premier caractère non alphabétique suivant le groupe de lettres.

Exemple : GRL (' JE SUIS, DONC ', 3, K)
= ' SUIS ', K = 8.

3.1.6. — SKP (Org,Dep,Crit)

Pour **SKi**P - résultat numérique.

Cette fonction donne la position dans Corg du premier caractère n'appartenant pas à la chaîne Ccrit.

Cet exposé succinct montre la richesse de LSE pour le traitement de chaînes. L'analyse de réponse est particulièrement facilitée par la présence de ces fonctions.

3.2. — Traitement numérique

LSE permet aussi la manipulation de variables numériques.

Les opérateurs disponibles sont :

l'addition	notée	+
la soustraction	*notée*	—
la multiplication	notée	*
la division	notée	/
l'élévation à une puissance	notée	↑

— les fonctions mathématiques classiques :
SIN, COS, TAN, ATG, LGN, EXP, RAC, ABS, ENT

— les fonctions sur chaîne à résultat numérique que nous *avons déjà décrites ;*

— des fonctions d'utilité générale. Parmi celles-ci citons :

ALE	*donne une valeur pseudo-aléatoire*
TEM	donne l'heure
TYP	donne l'état d'une variable
IND	donne les dimensions d'un tableau.

Ces fonctions et les opérateurs permettent de construire des expressions numériques.

3 + SIN(X) * EXP(Z)

A la différence de nombreux langages, LSE définit de manière stricte l'ordre d'évaluation des expressions arithmétiques, ce qui améliore la portabilité des programmes.

3.3. — Traitement des variables booléennes

On peut déclarer en LSE qu'une variable est booléenne par la déclaration :

BOOLEEN X

Ceci signifie que les variables X ne peuvent prendre que les valeurs .VRAI. ou .FAUX.

On peut ranger dans une variable booléenne la valeur d'une condition :

TROUVA ← X=Y

Les opérations possibles sur les variables booléennes sont les opérations classiques :

ET,OU,NON

3.4. — *Traitement des variables numériques étendues*

Il est possible d'utiliser des variables représentant les nombres sous une forme interne qui permet de garantir une précision (relative ou absolue) arbitraire. Sur ces nombres, les quatre opérations + — * / sont possibles ainsi que les fonctions ABS, ENT et la conversion en d'autres types.

Cette arithmétique a été définie pour les applications de gestion, ou des applications scientifiques où une grande précision est demandée.

3.5. — **Structures**

LSE permet de grouper en tableau toutes les variables des types définis précédemment. Le nombre de dimensions possibles d'un tableau est limité à 1Ø.

La gestion des tableaux est dynamique, c'est-à-dire que la dimension d'un tableau peut n'être définie qu'à l'exécution.

3.6. — **Instructions de contrôle**

Les instructions de contrôle sont au nombre de 5.

3.6.1. — **ALLER EN**

Sa forme générale est :

ALLER EN exp

Exp étant une expression numérique, la valeur de l'expression exp (arrondie à l'entier le plus voisin) donne le numéro de la prochaine ligne à exécuter.

Un cas particulier important est :

ALLER EN A [I]

qui représente un aiguillage.

3.6.2. — ***SI***

Sa forme est :

Si condition ALORS inst 1

ou

Si condition ALORS inst 1 sinon inst 2

Cette instruction commande à l'ordinateur d'exécuter inst 1 si la condition est VRAI. Si la condition est fausse inst 2 (si elle existe) sera seule exécutée.

Exemples :

SI A = 3 ALORS X ← 2 SINON 8 ← 3

SI CLE OU B ET C≠Ø ALORS ALLER EN 7

3.6.3. — **FAIRE**

Sa forme générale est :

FAIRE exp1 POUR var←exp2 PAS exp3

ou

FAIRE exp1 TANT QUE exp. bool.

JUSQUA exp4

TANT QUE exp. bool.

Elle permet de commander la répétition de l'exécution d'un groupe d'instructions (boucle).

expl indique le numéro de la dernière ligne faisant partie de la boucle.

Le pas et l'expression conditionnant la fin de boucle sont réévalués à la fin de chaque boucle.

La variable de contrôle var peut être modifiée à l'intérieur de la boucle.

Le pas peut être omis ; il est alors pris égal à 1.

3.6.4. — **EXÉCUTER**

Sa forme générale est :

EXECUTER Ch,Ea,Cr

Elle indique que la prochaine instruction à exécuter se trouve dans le programme dont le nom est la chaîne Ch, à la ligne Vea (programme se trouvant actuellement en mémoire auxiliaire).

Exemple : EXECUTER ' SUITE ',3,X

L'exécution de cette instruction remplacera le programme présent en mémoire par le programme de nom SUITE, les variables du programme quitté étant perdues, et lancera l'exécution à la ligne 3 du programme SUITE.

Cette instruction permet donc l'enchaînement de l'exécution de programmes. Ceci est utile pour des applications dont les programmes sont trop gros pour être contenus en même temps dans la mémoire disponible.

La variable Cr est une variable (optionnelle) de compte rendu. En cas d'incident dans le traitement de l'instruction EXECUTER, cette variable contient une valeur permettant de connaître la cause de l'incident.

3.6.5. — **PAUSE, TERMINER**

Ces instructions permettent d'interrompre le déroulement d'un programme, ou d'indiquer sa fin.

3.7. — **Procédures**

On peut définir dans LSE des procédures : procédures sous-programmes et procédures-fonctions.

Une procédure-fonction a une valeur et peut être utilisée dans des expressions numériques ou expressions chaînes.

Une procédure a un nom, qui se compose du caractère & suivi d'un identificateur. Une procédure est généralement utilisée avec des paramètres. Le passage des paramètres peut se faire soit par valeur, soit par référence.

Une procédure peut être définie en dehors du programme qui l'utilise ; cette particularité permet de ne définir les sous-programmes nécessaires à une application qu'une seule fois et incite à une programmation modulaire.

La définition d'une procédure est simple. Supposons que l'on ait besoin d'une procédure initialisant une colonne de tableau à une valeur définie par une autre procédure. La définition se fera par :

```
1ØØ PROCEDURE &INIT (A,C,L,&F) LOCAL C,L,I
1Ø2 FAIRE 1Ø4 POUR I ← 1 JUSQUA L
1Ø4 A[C,I] ← &F (C,I)
1Ø6 RETOUR
```

Ces quatre lignes décrivent la procédure de nom &INIT qui a comme paramètres formels :

A — qui sera un tableau (dont on passe l'adresse : paramètre ' référence ').

C,L — qui sont deux variables (dont on passe la valeur), C représente le numéro de colonne, L le numéro de la dernière ligne à initialiser.

&F — qui est le nom d'une procédure-fonction.

La déclaration LOCAL C,L,I permet :

— d'une part pour les paramètres C et L de spécifier un passage par valeur ;

— d'autre part de définir une variable I qui n'est connue qu'à l'intérieur de la procédure, et qui n'a donc rien à voir avec d'éventuelles variables de même nom du programme appelant. Ces variables ' locales ' disparaissent dès que l'on quitte la procédure.

Cette procédure &INIT pourra par exemple être utilisée dans le programme suivant :

```
 8   LIRE N
1Ø   TABLEAU TRUC[N,N]
12   FAIRE 14 POUR J ← 1 JUSQUA N
14   &INIT (TRUC, J, N, &UN)
45   TERMINER
50   PROCEDURE &UN (I,J) LOCAL I,J
52   RESULTAT SI I = J ALORS 1 SINON Ø
```

Les lignes 5Ø et 52 décrivent une procédure-fonction. C'est l'instruction RESULTAT qui définit la valeur rendue par cette fonction.

L'utilisation de cette fonction dans l'appel de &INIT (ligne 14) aura pour effet de la faire utiliser dans la procédure &INIT sous le nom &F.

La ligne 1Ø4 permet donc d'affecter à A[C,I] la valeur de &UN (C,I).

SI l'on examine les autres paramètres de la ligne 14, on constate que A représente le tableau TRUC, C a même valeur que J, L a même valeur que N. Donc I variera de 1 à N et la ligne 1Ø4 est donc équivalente à :

1Ø4 TRUC[J,I] ← &UN[J,I]

A la fin de l'exécution de la boucle définie ligne 12, le tableau TRUC aura donc tous ses éléments TRUC[J,I] initialisés avec la valeur de la fonction &UN (J,I). Les éléments du tableau TRUC seront donc initialisés à Ø si J ≠ I, à 1 sinon.

Remarque

Une procédure peut être récursive. C'est-à-dire que l'on peut, au cours de l'exécution d'une procédure, réappeler la même procédure.

Cette particularité permet de simplifier la programmation de certains problèmes.

Exemple :

La procédure permettant de calculer les combinaisons de N objet pris P à P en utilisant la forme

$$C_n^P = C_{n-1}^{P-1} + C_{n-1}^{P}$$ peut s'écrire

```
1ØØ   PROCEDURE &C (N,P) LOCAL N,P,
1Ø2   SI N = 1 OU P = Ø ALORS RESULTAT 1 ;
1Ø4   RESULTAT &C (N-1, P) + &C (N-1, P-1)
```

3.8. — Instructions d'entrées-sorties

LSE a des instructions d'entrées-sorties simples à utiliser.

On distingue 3 types d'instructions d'entrées-sorties :

— les instructions console LIRE, AFFICHER ;

— les instructions sur fichier CHARGER, GARER, SUPPRIMER ;

— les instructions d'entrée-sortie généralisées.

3.8.1. — Instructions console

3.8.1.1. — LIRE

L'instruction LIRE permet d'affecter depuis la console une valeur à une variable.

La forme la plus simple est

```
LIRE X
```

où X peut être :

— *une variable arithmétique,*

— une variable-chaîne,

— *un nom de tableau.*

L'exécution de cette instruction met en lecture la console, *après émission d'un signal pour prévenir l'utilisateur.*

La console sera activée tant que la valeur demandée n'aura pas été fournie correctement.

LSE exécute une lecture par valeur à lire, ce qui permet de supprimer les problèmes de synchronisation entre l'utilisateur et son programme.

En cas de lecture d'un tableau, il faudra fournir la valeur de tous les éléments du tableau.

Si on a plusieurs éléments à lire, on peut écrire :

LIRE X,Y
qui est interprété comme
LIRE X ; LIRE Y

3.8.1.2. — **AFFICHER**

AFFICHER permet l'édition de résultats vers l'extérieur.

LSE considère l'organe de sortie comme un terminal travaillant caractère par caractère.

L'utilisateur a le contrôle complet de ce qui est envoyé au terminal. Il existe cependant une forme implicite utilisable dans les cas simples.

Exemple d'instructions AFFICHER :

```
1Ø    AFFICHER B
2Ø    AFFICHER ' RACINE (' ,B, ') = ', RAC (B)
3Ø    AFFICHER [' VALEUR DE X : ' ,F5.2] B
4Ø    AFFICHER [/ , ' CØMPTE ' , * ' DEBITEUR '] B
```

La ligne 1Ø montre la forme la plus simple.

Par convention, l'absence de format provoque un retour à la ligne. B sera édité sous la forme d'un format U que nous décrirons plus bas.

La ligne 2Ø est du même type, on a à afficher 4 valeurs dont 2 sont des chaînes.

Il y aura retour à la ligne puis impression des 4 valeurs consécutivement.

On voit que cette forme permet la réalisation d'états imprimés simples.

La ligne 3Ø montre l'utilisation d'un format dans lequel on spécifie comment la valeur doit être éditée.

Il y a trois formats principaux pour éditer des valeurs numériques :

— le format F de la forme F e.d

e représente le nombre minimum de caractères désiré avant le point décimal.

d représente le nombre de caractères de la partie fractionnaire.

La forme F donne une présentation sous forme décimale.

— le format E de la forme E e.d

e,d ont même signification que pour le format F.

La forme E donne une représentation sous forme mantisse-exposant.

— le format U permet d'afficher un nombre X sous forme naturelle.

Dans ce cas, la forme F sera utilisée si :

$$1Ø^{-3} \leqslant |X| < 1Ø^{6}$$

La forme E sera utilisée dans les autres cas.

Le système choisit le nombre de chiffres de façon à représenter le nombre sans perte de précision, et un espace est placé après le nombre.

La ligne 4Ø montre des exemples de spécifications de mise en page.

/ indique qu'il faut passer à la ligne suivante.

' COMPTE ' indique qu'il faut éditer le texte : COMPTE.

* ' DEBITEUR ' indique qu'il faut éditer ' DEBITEUR ' le nombre de fois défini par la valeur correspondante dans la liste (soit B dans ce cas). Si X vaut Ø ce texte ne sera pas édité.

L'astérisque peut se mettre aussi devant les spécifications de mise en page, qui sont :

/ passage à la ligne,

X impression d'un espace,

C retour au début de la ligne,

L passage à la ligne suivante (sans retour au début de ligne).

3.8.1.3. — **Lecture avec FORMAT**

La séquence d'instruction

AFFICHER [/ , ' VALEUR DE X '] ; LIRE X
peut s'écrire
LIRE [/ , ' VALEUR DE X ']X

Les instructions d'entrées-sorties sont simples à utiliser et permettent la programmation facile de la plupart des éditions.

3.8.2. — **Les instructions sur fichier**

LSE permet de créer et d'exploiter des fichiers grâce aux instructions GARER et CHARGER.

Un fichier est défini comme une suite d'enregistrements, chaque enregistrement portant un numéro différent, mais quelconque (clé numérique). Un enregistrement ne contient qu'un objet (variable numérique ou chaîne, tableau).

La forme de GARER est :

GARER Nom, EA, Fich, Cr
où
Nom de l'identificateur d'un objet :

EA est une expression numérique donnant un numéro d'enregistrement.

Ch est une expression chaîne donnant un nom de fichier.

Cr est une variable optionnelle de compte rendu permettant de savoir comment s'est passée l'opération.

Par exemple :

GARER TAB, 4, ' TEMPO ', IND

Cette instruction provoque le rangement de l'objet de nom TAB dans l'enregistrement de clé 4 du fichier TEMPO.

L'opération inverse de GARER se fait par l'instruction CHARGER. Elle est de la forme :
CHARGER Nom, Ea, Ch, Cr

Les paramètres ont la même signification que pour GARER.

Par exemple :
CHARGER X, 3, 'TEMPO', IND

On peut supprimer un fichier entier par
SUPPRIMER Ch
ou un enregistrement par
SUPPRIMER Ch, Ea

3.8.3. — **Protection**

Un système d'identification permet de protéger les fichiers contre tout accès (sauf personne autorisée).

3.8.4. — **Entrées Sorties généralisées**

Elles sont la fusion de LIRE, CHARGER pour ENTRER et de AFFICHER, GARER pour SORTIR.

La syntaxe générale en est :
LIRE < Fich,Cle,Cr > [Format] liste
SORTIR < Fich,Cle,Cr > [Format] liste.

Pour permettre l'accès séquentiel, une fonction, NES (Fich,Clé) fournit le numéro de l'enregistrement de numéro immédiatement supérieur à Vclé (Ø s'il n'y en a pas).

Le partage de fichier est possible grâce aux instructions RESERVER, RELACHER.

4. — **Langage de commande**

Le langage de commande défini avec LSE permet de développer une application sans connaître les particularités du système hôte.

Le langage de commande est organisé autour de la notion de session. On peut définir une session comme le temps passé à une console LSE par un utilisateur donné.

Pendant cette session, l'utilisateur ne pourra manipuler qu'un programme à la fois, le 'programme courant'.

Le programme courant est soit chargé à partir de la mémoire auxiliaire (commande APPELER, voir plus bas), soit entré, ligne par ligne, par l'utilisateur.

Le simple fait de frapper une ligne de LSE avec un numéro de ligne fait adjoindre au programme courant cette ligne (si elle est syntaxiquement correcte).

Un mini-éditeur de texte permet de corriger facilement toute ligne du programme.

On peut supprimer les lignes du programme courant, à l'aide de la commande EFFACER.

Toutes les commandes du LSE sont mises en œuvre en frappant leurs 2 premières lettres, suivies du caractère (X-OFF) ; le système complète alors le libellé de la commande, et se met en lecture de ses éventuels paramètres.

Ainsi, si l'on veut effacer de la ligne 15 à la ligne 25, on frappera EF (X-ØFF).

L'ordinateur complétera par :
FACER LIGNES
et se mettra en lecture.

On frappera alors : 15 A 25 (X-OFF)
et la commande sera exécutée.

Il existe 24 commandes LSE ; citons les principales :

4.1. — **EXÉCUTER A PARTIR DE N1**

Lance l'exécution du programme courant à partir de la ligne de numéro N1.

4.2. — **LISTER N1 A N2**

Liste le programme courant de la ligne de numéro N1 à la ligne de numéro N2.

4.3. — **EFFACER LIGNE**

Déjà vue.

4.4. — **RANGER TRUC**

Range le programme courant en mémoire auxiliaire sous le nom TRUC.

4.5. — **REMPLACER TRUK**

Remplace le programme mis en mémoire auxiliaire sous le nom TRUK par le programme courant.

4.6. — **APPELER MACHI**

Charge en mémoire le programme mis en mémoire auxiliaire sous le nom MACHI, l'ancien programme courant est effacé.

4.7. — **ENTRÉE AQ4**

Indique au système qu'il doit lire les commandes dans le fichier AQ4 (au lieu de les attendre à la console).

4.8. — **SORTIE SØ7**

Indique que les impressions provoquées par la prochaine commande doivent être dirigées dans le fichier SØ7.

5. — **Mise au point**

Il est possible de faire exécuter un programme en pas à pas, c'est-à-dire avec arrêt de l'exécution au début de chaque ligne. On peut aussi spécifier une ligne d'arrêt au lancement de l'exécution.

L'exécution peut être suspendue à tout moment par la frappe d'une touche et reprise au gré de l'utilisateur (commandes **CO**NTINUER, **POU**RSUIVRE JUSQU'A).

Il est également possible de faire exécuter immédiatement des instructions LSE entrées depuis le terminal.

Cette exécution immédiate s'appelle le mode Machine de Bureau.

Ce mode permet notamment de connaître le contenu des variables du programme en cours de mise au point. On peut aussi modifier des variables du programme, ce qui, dans certains cas, permet une mise au point rapide du programme.

6. — Conclusion

LSE est un langage riche et cependant simple à employer ; il est utilisé dans l'Education nationale pour réaliser des programmes d'aide à l'enseignement et pour l'apprentissage de la programmation.

Son domaine d'application n'est pas limité à l'enseignement ; il a été, en particulier, réalisé un système permettant le contrôle de processus, le LST. Ce système qui utilise LSE comme langage de programmation est utilisé dans l'industrie, notamment pour la commande des bancs de tests automatiques de matériel ; la facilité de mise au point en LSE (outils de mise au point, richesse des diagnostics d'erreurs) sont un avantage important dans ce genre d'application, où les programmes doivent être très fréquemment modifiés.

II. — BASIC

1. — Généralités

Basic est constitué des initiales de : Beginner's All-purpose Symbolic Instruction Code, ce qui signifie : Code d'instructions symbolique pour tout usage du débutant.

Il fut défini au Darthmouth College en 1965.

Comme son nom l'indique, le but de ses pères était de définir un langage simple à apprendre, servant d'introduction avant l'apprentissage de langages classiques comme FORTRAN ou COBOL.

De plus, ce langage permettait de tirer parti des possibilités des systèmes conversationnels qui commençaient à apparaître.

Son usage s'est énormément répandu sous l'effet de deux phénomènes :

— l'arrivée sur le marché du travail américain de scientifiques formés à BASIC ;

— l'apparition de mini-ordinateurs, puis de micro-ordinateurs destinés au petit calcul scientifique.

Il existe une normalisation du BASIC minimum que nous désignerons par BASIC minimum. Le succès de BASIC a entraîné la réalisation d'un grand nombre d'installations sur une multitude d'ordinateurs. La concurrence commerciale, le désir de répondre aux besoins d'une clientèle donnée a entraîné la création d'une multitude de variantes.

Pour mettre un peu d'ordre, on a cherché à normaliser BASIC ; il existe actuellement une norme pour un sous-ensemble minimum de BASIC, et un projet de norme pour un sur-ensemble du BASIC initial.

Nous décrirons, dans ce qui suit, les traits caractéristiques de ce sur-ensemble que nous désignerons sous le nom de BASIC étendu.

2. — Exemple de programme

Nous donnons, ci-dessous, le programme BASIC équivalent au programme LSE.

La version utilisée est celle d'un micro-ordinateur de grande diffusion.

```
 1  REM LECTURE D'UN NOMBRE EN CHIFFRES
    ROMAINS
 3  D$ = " M1ØØØ DØ5ØØ CØ1ØØ LØØ5Ø XØØ1Ø VØØØ5
IØØØ1 "
 5  PRINT " NOMBRE ROMAIN " ;
 7  INPUT R$
 9  S = Ø
11  FOR I = 1 TO LEN (R$)
13  C$ = MID$ (R$,I,1)
17  FOR J = 1 TO LEN (D$)
19  E$ = MID D$,J,1)
21  IF E$ = C$ THEN 27
23  NEXT J
25  PRINT " ERREUR "
26  GOTO 7
27  P = VAL (MID(D$,J+ 1,4)
31  S = S+P
33  NEXT
35  PRINT " EQUIVALENT DECIMAL ", S
37  GOTO 7
```

Il est à noter que le BASIC étendu n'est disponible généralement que sur des systèmes de coût relativement important.

Nous signalerons plus loin les caractéristiques les plus souvent absentes des BASIC disponibles sur micro-ordinateurs.

3. — Structure générale d'un programme BASIC

Un programme BASIC est composé de lignes, chaque ligne se composant d'un numéro suivi d'une instruction. Les instructions sont exécutées dans l'ordre des numéros croissants.

Objets manipulés

Les objets manipulés sont les quantités numériques et les chaînes de caractères.

Identificateurs

Les identificateurs de variables numériques sont composés d'une lettre, éventuellement suivie d'un chiffre. Un élément de tableau est noté par le nom du tableau suivi de la valeur des indices mise entre parenthèses.

Exemple :

A1 (3)
T2 (3,4)

Les identificateurs de variables chaînes de caractère ont leur nom suivi du caractère $.

Exemple :

C$, C1$

3.1. — **Variables numériques**

Les variables numériques sont d'un type unique. Le BASIC étendu envisage la possibilité de types numériques diversifiés (Réel, entier, décimal).

Les expressions numériques sont évaluées selon les règle usuelles. BASIC prévoit des fonctions, qui sont représentées par leur nom suivi de (ou des) arguments entre parenthèses ; par exemple :

Sin (X)
COS (Y/2)

BASIC étendu prévoit une quarantaine de fonctions différentes. Les BASIC disponibles sur micro-ordinateur n'en donnent qu'un sous-ensemble comparable à celui de LSE : il possède des fonctions sans paramètres.

Citons :

DATE : donne une valeur définie par 1ØØØ fois le quantième de l'année dans le siècle + le quantième du jour dans l'année
EPS : le plus petit nombre positif représentable
RND : un nombre pseudo-aléatoire ($Ø \leqslant RND < 1$)
TIME : le temps écoulé en secondes depuis Ø heure.

Affectation numérique :

BASIC accepte les affectations multiples ; on peut écrire :

X ; Z=3

qui est équivalent à :

X=3
Z=3

3.2. — **Chaînes de caractères**

Les chaînes de caractères ont une longueur variant de Ø à une longueur maximum définie par la réalisation ou par une déclaration explicite.

Si C$ est une chaîne, C$ (4 : 8) représentera une sous-chaîne, au 4e caractère et finissant au 8e. Ainsi, si C est égale à " LA VIE AU SØLEIL ", C (4 : 8) est égale à " VIE A ".

Le seul opérateur sur les chaînes est la concaténation noté &.

Il existe 4 fonctions ayant un résultat chaîne :

3.2.1. — **CHR$ (X)**

Donne une chaîne de 1 caractère dont le code est égal à X.

3.2.2. — **DATE$**

Donne la date dans une chaîne ; ainsi le 14 janvier 1980

DATES vaudra " 80/01/14 ".

3.2.3. — **STR$ (X)**

Donne une chaîne de caractères représentant la valeur de la variable X ; par exemple :

Si X vaut 7.42, STRS (X) = " 7.42 ".

3.2.4. — **TIME$**

Donne l'heure dans le jour ;

par exemple à 9 h 43 mn 34 s

TIMES vaut : " 09 : 43 : 34 ".

Il existe quatre fonctions dont au moins un des arguments est une chaîne et qui rendent une valeur numérique :

3.2.5. — **LEN$ (AS)**

Donne le nombre de caractères contenus dans A$.

3.2.6. — **ORD (AS)**

Donne le rang dans l'alphabet du caractère désigné par AS.

Ainsi :

Si A$ vaut " A "
ORD (AS) = 65.
Si A$ vaut " BS " (Back Space)
ORD (AS) = 8.

3.2.7. — **POS (AS,BS)**

Donne la position de la chaîne B dans la chaîne A, c'est-à-dire le numéro du 1er caractère de A de la première apparition de la chaîne B.

Exemple :

POS (" LE GRAND CHEF ", " ND ") = 7.

Si la chaîne BS n'apparaît pas dans AS la fonction prend la valeur Ø :

POS (" CECI EST, " TRUC ") = Ø.

POS peut avoir un troisième paramètre numérique :

POS (AS,BS,dep) ce troisième paramètre permet de préciser à partir de quel caractère il faut chercher l'apparition de la chaîne BS.

Exemple :

POS (" GRAND MAMAN ", " MA ",1) = 7
POS (" GRAND MAMAN ", " MA ",8) = 9.

3.3. — **Structures**

BASIC admet les tableaux de nombres et de chaînes à une ou deux dimensions. Les tableaux sont statiques, c'est-à-dire qu'ils sont déclarés à la compilation. La forme de la déclaration est :

DIM A (5), B (7,10), AS (3,4).

Les éléments sont numérotés à partir de Ø.

La déclaration DIM est facultative. Si on emploie un tableau non déclaré par ailleurs, celui-ci est défini avec des indices pouvant aller jusqu'à la valeur 1Ø.

3.3.1. — **Opération sur les structures**

3.3.1.1. — **Tableaux des valeurs numériques** (généralement constantes sur micro-ordinateur)

Le BASIC étendu permet des opérations sur les tableaux. Ces opérations sont annoncées par le préfixe MAT.

Exemple :

MAT A = A*C

indique que l'on veut affecter à A le produit matriciel B*C. A sera dynamiquement redimensionné sans toutefois que l'encombrement du tableau puisse excéder l'encombrement initial.

Les opérateurs autorisés sont : + — *.

Notons T1,T2,T3, des noms de tableau
E,F une expression arithmétique.

Les expressions autorisées peuvent être de la forme :
T1+T2.

Il existe deux fonctions de tableau donnant pour résultat un tableau ; ce sont :

TRN (T1) : TRN donne la transposée de T
INV (T1) : INV donne l'inverse de T.

Il est possible de générer des matrices remarquables grâce aux trois fonctions suivantes :

ZER (E,F) : génère une matrice nulle de A lignes de B colonnes.

CON (E,F) : génère une matrice contenant des 1 sur ses A lignes de B colonnes.

IDN (E,F) : génère une matrice carrée de A lignes et A colonnes.

Enfin, il existe deux fonctions à arguments tableaux et à résultat numérique :

DET (T1) : donne la valeur du déterminant de la matrice carrée définie par T1.

DOT (T1,T2) : donne le produit interne des tableaux T1 et T2 à une dimension.

3.3.1.2. — **Tableaux de chaînes**

BASIC étendu définit aussi les opérations sur les tableaux de chaînes. Elle sont aussi énoncées par le préfixe MAT.

Par exemple :

MAT A$ = B$&C$

indique que l'on veut créer un tableau de chaîne AS obtenu en faisant la concaténation terme à terme des éléments des tableaux B$ et C$.

La seule opération permise est la concaténation (&). Les deux opérandes doivent être de même dimension.

A$ sera redimensionné. Le nouvel encombrement ne peut dépasser l'encombrement défini lors de la déclaration initiale.

Les termes de la concaténation peuvent être tous deux des tableaux. L'un des termes peut aussi être une expression chaîne mise entre parenthèses.

Exemple :

MAT A$ = (" CE CI ") &B$.

Quand un nom de tableau est utilisé comme terme de la concaténation, il peut être suivi d'une spécification de sous-chaîne.

Exemple :

MAT A$ = A$ (3,1)&BS.

3.4. — *Instructions de contrôle*

Les instructions de contrôle sont au nombre de six :

A) GO TO
par exemple GO TO 1ØØ

B) IF ... THEN ... ELSE ...
par exemple IF I = 3 THEN 2Ø ELSE 4Ø

C) ON ... GO TO ...
par exemple ON I GO TO 7,14,2Ø ELSE 1ØØ

D) GOSUB ...
par exemple GOSUB 2ØØ

E) ON ... GOSUB ...
par exemple ON K1 GOSUB 7Ø,95

F) FOR ... I = 1 TO 1Ø STEP 2
....
NEXT I.

Ces instructions de contrôle ont la signification suivante :

3.4.1. — GO TO

L'instruction GO TO indique que la prochaine instruction à exécuter est celle dont le numéro de ligne suit. Seul un numéro de ligne peut suivre GO TO.

3.4.2. — IF ... THEN ... ELSE

L'instruction IF ... THEN ... ELSE demande l'évaluation de la condition décrite entre IF et THEN ; si cette condition est réalisée, la prochaine instruction à exécuter est celle dont le numéro suit le THEN. Si la condition n'est pas réalisée, l'instruction à exécuter est celle dont le numéro suit le ELSE.

La partie ELSE ... peut être absente ; dans ce cas, si la condition n'est pas réalisée, l'exécution se poursuit en séquence.

3.4.3. — ON ... GO TO

La forme ON I GO TO 7,14,2Ø ELSE 1ØØ est une sténographie pour :

IF I = 1 THEN 7
IF I = 2 THEN 14
IF I = 3 THEN 2Ø ELSE 1ØØ.

Si I n'est pas entier, BASIC utilisera comme valeur de choix l'entier le plus proche de I. La partie ELSE peut être omise.

A la suite de ELSE, on peut trouver une instruction de type impératif ; par exemple :

ELSE A = 94.

3.4.4. — GOSUB ...

La forme GOSUB 2ØØ définit l'instruction 2ØØ comme devant être exécutée. De plus, il est gardé en mémoire le numéro de la ligne de l'instruction GOSUB exécutée. La séquence d'instructions commençant en 2ØØ sera

exécutée jusqu'à la rencontre d'une instruction RETURN. Cette instruction RETURN provoquera l'exécution des instructions suivant le GOSUB.

Le " GOSUB " est une forme rudimentaire d'appel de sous-programme. On notera qu'on ne donne pas de nom au sous-programme et qu'on ne lui transmet pas de paramètres. Cette caractéristique du GOSUB ne facilite pas la lisibilité des programmes.

3.4.5. — ON ... GOSUB

La forme ON 71 GOSUB 7Ø,95 ELSE 2ØØ est semblable à l'instruction ON ... GO TO. Elle permet de sélectionner un sous-programme donné.

3.4.6. — FOR ... TO ... STEP ...

La forme FOR I = 1 TO 1Ø STEP 2 permet de faire répéter l'exécution d'un groupe d'instructions en faisant varier la variable de boucle (ici I). Le groupe d'instructions à répéter suit l'instruction FOR et s'arrête à l'instruction NEXT I.

Illustrons l'emploi de FOR dans le calcul de factorielle n :

```
1Ø  F = 1
15  FOR I = 2 TO N STEP 1
2Ø  F = F*N
25  NEXT I.
```

La ligne 15 indique qu'il faut exécuter ce qui suit pour I = 2,3, ... jusqu'à I = N.

L'ensemble des instructions à répéter commence après la ligne 15 et se termine avec l'instruction NEXT I.

L'ensemble des instructions à répéter peut ne pas être exécuté du tout. Tel sera le cas dans notre exemple si N = 1. En effet, dans ce cas, BASIC considèrera que la limite est atteinte et sautera les instructions contrôlées jusqu'au NEXT correspondant.

La spécification STEP 1 indique qu'il faut faire progresser de 1 à chaque ré-exécution de la boucle. Une expression quelconque peut suivre le mot STEP.

On remarquera que si on omet la spécification STEP, BASIC prend par défaut un incrément de 1.

3.5. — Autres instructions de contrôle BASIC étendu

La norme ANS BASIC étendu prévoit trois autres structures de contrôle qui sont moins répandues actuellement (surtout sur les micro-ordinateurs).

3.5.1. — LOOP

Exemple :

```
1Ø  INPUT A
15  X = A/2
2Ø  LOOP
3Ø  IF ABS ((X*X-A)/(2*A))<1E-4 THEN EXIT
4Ø  X = (X+A/X)/2
5Ø  END LOOP.
```

Cette construction fait exécuter les lignes situées entre LOOP et END LOOP jusqu'à ce que l'instruction EXIT soit exécutée.

3.5.2. — BLOC IF

Cette instruction inspirée de FORTRAN 78 a la forme :

```
1Ø IF  condition THEN
   ... instructions
5Ø ELSE
   ... instructions
1ØØ END IF.
```

Cette construction permet de réaliser un aiguillage plus facile à lire que l'instruction ON ... GO TO ...

Illustrons son emploi par des exemples :

```
10 SELECT A$ (1 : 1)
2Ø CASE " A " TO " Z "
3Ø PRINT AS, " LETTRE "
4Ø CASE " Ø " TO " 9 "
5Ø PRINT AS, " CHIFFRE "
6Ø CASE ELSE
7Ø PRINT " ERREUR "
8Ø END SELECT.
```

Dans le domaine numérique, la même construction peut être utilisée :

```
1Ø INPUT A,B,C
2Ø D=B*B-4*A*C
3Ø SELECT D
4Ø CASE < Ø
5Ø PRINT " PAS DE RACINES "
6Ø CASE > Ø
65 PRINT " DEUX RACINES REELLES "
7Ø CASE ELSE
75 PRINT " RACINES DØUBLES "
8Ø END SELECT.
```

Cette instruction est inspirée notamment de PASCAL. On notera que ces trois dernières instructions tranchent sur le style des instructions de contrôle précédentes.

On peut notamment constater que ces constructions nécessitent plusieurs lignes pour être exprimées, ce qui n'était pas le cas généralement en BASIC (à l'exception de la boucle FOR).

3.6. — Segmentation du programme

Il est possible en BASIC de définir des fonctions. On peut ainsi définir :

```
FNA (X,Y)=SQRT (X*X+Y*Y)
```

qui définit la longueur de l'hypothénuse d'un triangle-rectangle. Un nom de fonction doit commencé par FN et ne peut être que de la forme FNac où a est une lettre et c un chiffre. La liste des paramètres formels suivant le nom de la définition de la fonction se fera en écrivant derrière le signe = l'expression définissante.

On remarquera que cette construction ne permet de définir que les fonctions ayant une expression analytique simple. Ainsi, la fonction qui vaut :

Ø si X < Ø
1 si Ø < X < 1
Ø si X ≥ 1

ne peut être définie comme une fonction en BASIC.

3.7. — Sous-programmes

La nouvelle norme BASIC étendu introduit la notion de sous-programme de façon similaire à FORTRAN. Les sous-programmes doivent être définis dans le programme qui l'utilise. Ainsi, si l'on veut utiliser le sous-programme INITA qui met tous les éléments d'un tableau à la valeur dépendant de l'indice, on appellera le sous-programme par :

```
1Ø CALL INITA (T1( ),N)
```

La définition de INITA se fera par :

```
1ØØ SUB INITA (T2( ),N)
11Ø FOR I = 1 TO N
12Ø T2 (I)=I*I
13Ø NEXT I
14Ø SUB EXIT.
```

Les paramètres sont passés par adresse.

3.8. — Chaînage de programmes

La nouvelle norme BASIC permet le chaînage de programmes, c'est-à-dire la possibilité pour un programme de faire mettre en mémoire un nouveau programme le remplaçant et d'en faire lancer l'exécution.

Ceci se réalise par une instruction du type

```
CHAIN " SUITE "
```

l'exécution de cette instruction provoque le chargement en mémoire du programme désigné par " SUITE " puis son exécution.

Aucune valeur du programme n'est transmise au second.

S'il faut communiquer des valeurs d'un programme à un programme chaîné, le programme demandeur rangera ces valeurs sur une mémoire auxiliaire. Le programme chaîné devra lire ces valeurs depuis la mémoire auxiliaire.

3.9. — Initialisation des variables

Il arrive que l'on ait besoin de donner des valeurs initiales à un certain nombre de variables. Ceci peut être fait par une série d'affectations, par exemple :

```
1Ø X = 3
2Ø X2 = 7.5
3Ø Y2 = 9
```

une instruction permettant d'affecter ces variables sous une forme plus condensée :

```
1Ø READ X1,X2,Y2
2Ø DATA 3,7.5,9
```

L'instruction DATA définit une liste de valeurs qui seront utilisées par une ou plusieurs instructions READ. Il y aura affectation d'une valeur à la variable citée dans l'instruction READ. On peut forcer la machine à reprendre les valeurs depuis le début de la liste en exécutant une *instruction* RESTORE.

Ainsi, le programme :

```
1Ø READ X1
2Ø READ X2,Y2
3Ø RESTORE
4Ø DATA 3,7.5,9
5Ø READ I,J
55 RESTORE
6Ø READ K
```

affectera à X1 la valeur 3.
à X2 la valeur 7.5

Y2	9
I	3
J	7.5
K	3

On regrettera que BASIC ait utilisé le verbe READ pour cette construction. En effet, dans nombre d'autres langages, ce verbe a une connotation bien précise : *celle de l'entrée-sortie, c'est-à-dire de la relation* avec le monde extérieur.

3.10. — ENTRÉES-SORTIES

Une des causes principales du succès de BASIC a *été la simplicité de ses entrées-sorties. A l'époque,* nombre de débutants étaient « traumatisés » par les entrées-sorties FORTRAN. BASIC propose une alternative suffisante pour les applications simples.

3.10.1. — LECTURE

La lecture se fait grâce à l'instruction INPUT.

Ainsi, l'instruction :

```
1Ø INPUT X,Y
```

provoque la mise en lecture de la machine sur le périphérique principal, et l'affectation aux variables X et Y dans le cas général.

Il existe des variantes de cette instruction permettant d'envoyer un message avant la mise en lecture :

```
INPUT PROMPT " DEGRE ? " : D
```

de lire une ligne complète pour une variable chaîne de la liste :

```
LINPUT A$,B$
```

Dans le cas normal, les différents éléments de la liste d'entrée sont séparés par des virgules. Il en résulte l'impossibilité de lire une chaîne de caractères contenant des virgules. L'ordre LINPUT pallie cet inconvénient.

3.10.2. — IMPRESSION

L'impression des résultats se fait par l'ordre PRINT.

Cet ordre est de la forme :

```
1Ø PRINT " SOLUTION " ; Y ; " ECART " ; Z
```

Sous cette forme, si Y = 3714 et Z = Ø.ØØØØØ1 on aura une impression de la forme :

SOLUTION : 3714 ECART 1E-6

Les différents éléments sont séparés par des points-virgules. S'ils étaient séparés par des virgules, il y aurait tabulation, c'est-à-dire mise en colonnes des différents éléments de la liste.

Dans l'ordre PRINT élémentaire, le format d'impression est défini par l'ordre de grandeur des quantités à imprimer. Si l'on veut un autre format d'impression, on peut utiliser la forme :

```
5Ø PRINT USING 1ØØ : X ; Z
1ØØ IMAGE : NOMBRE DE ROUES ###
                          PTIX ###.###
```

ce qui donnera si X = 347 et Y = 258.92 :

NOMBRE DE ROUES 347 PRIX 258.92

On peut spécifier trois formes de représentation des nombres que nous illustrerons en représentant 123.76 par les formats possibles :

###	124
###.##	123.76
###.#	123.8
#.##	1 24 E+Ø2

On remarquera les arrondis faits quand le nombre de chiffres est insuffisant.

3.10.3. — Entrée-sortie de tableau

BASIC permet de lire et d'imprimer un tableau à l'aide d'une seule ligne ; c'est le rôle des instructions MATINPUT, MATLINPUT, MATPRINT d'une syntaxe semblable à celle des instructions dont elles sont dérivées. On peut aussi initialiser un tableau grâce à l'instruction MATREAD.

3.10.4. — Traitement des fichiers

BASIC prévoit la possibilité de traitement des fichiers, c'est-à-dire que l'on peut faire des entrées-sorties sur mémoire auxiliaire. Le seul type de fichier utilisable est le fichier séquentiel, ce qui est gênant dans certains types d'applications.

3.11. — Autres possibilités de BASIC

La norme BASIC prévoit d'autres extensions de BASIC que nous ne ferons que citer :

3.11.1. — Extension graphique

Cette extension permet de définir des images sur des périphériques graphiques et de les transformer facilement. Il est permis de penser que la baisse prévisible du coût du matériel conduira à une extension de la diffusion du BASIC graphique.

4. — Langage de commande

La norme BASIC étendu prévoit un ensemble de commandes permettant de mettre en mémoire un programme, de le modifier et de le mettre au point.

Les commandes, disponibles d'après la norme sont :

4.1. — DELETE

De la forme DELETE 1Ø TO 2Ø cette commande effacera les lignes du numéro 1Ø au numéro 2Ø.

4.2. — LIST

De la forme LIST ou LIST 25 TO 3Ø. Cette commande visualise tout le programme dans le premier cas et les lignes de numéro 25 à 3Ø dans le deuxième.

4.3. — RENUMBER

Dans la forme RENUMBER 1ØØ TO 2ØØ STEP 5 AT 5ØØ provoque la renumérotation des lignes du programme comprise entre 1ØØ à 2ØØ en leur donnant comme numéro 5ØØ,5Ø5,....

4.4. — Instruction de mise au point

Le BASIC donne à l'utilisateur des instructions de mise au point, ce sont :

4.4.1. — BREAK

Cette instruction, lorsqu'elle est rencontrée, suspend l'exécution du programme. Si on a exécuté précédemment l'instruction DEBUG ON. Si on a exécuté DEBUG OFF, l'instruction BREAK est ignorée.

4.4.2. — TRACE ON

L'exécution de cette instruction provoque l'édition de renseignements (valeur des variables affectées, numéro de la ligne atteinte), propres à faciliter la mise au point.

L'instruction TRACE OFF annule cette trace.

Ces facilités, jointes au mode interactif de BASIC, rendent l'utilisation de BASIC agréable.

5. — Conclusion

BASIC existe depuis 1968 et s'est répandu largement. Il doit son succès à son côté conversationnel et à ses outils de mise au point. On peut regretter ses limitations :

— nom de variables de 1 ou 2 caractères seulement ;

— non-régularité de la syntaxe : ainsi la notion de sous-programme est traitée de trois façons différentes (DEF, GOSUB, CALL). Un autre handicap de BASIC est la multiplication présente des versions de BASIC qui sont incompatibles entre elles ;

— les versions généralement disponibles sur micro-ordinateurs sont moins puissantes que ce que nous avons décrit (moins de fonctions, pas d'instruction de manipulation de tableau, pas de structures de contrôle évoluées...).

III. — COMPARAISON LSE-BASIC

BASIC et LSE ont été inspirés par le même état d'esprit : fournir au débutant un outil simple de program-

mation. Les deux langages ont été conçus pour permettre un découpage en lignes indépendantes. Résumons rapidement les critiques que l'on peut faire aux deux langages :

1. — Points faibles de BASIC

— BASIC ne permet que des noms d'identificateurs composés d'un ou deux caractères. Cette caractéristique rend périlleuse l'écriture et la modification des programmes importants.

— BASIC gère la mémoire statiquement, ce qui impose des déclarations de taille fixe des objets manipulés. Cette contrainte ne permet pas toujours une utilisation optimale de la mémoire.

Les instructions de contrôle de base tendent à rendre un programme illisible puisqu'elles impliquent l'utilisation de " GO TO " dans tous les cas.

La concurrence commerciale a provoqué une prolifération de versions BASIC qui rendent la portabilité des programmes aléatoire.

Les versions disponibles sur micro-ordinateur sont généralement d'un niveau rudimentaire et certaines n'ont de BASIC que le nom.

— BASIC n'a pas défini une structure unique de décomposition en sous-programmes. La forme GOSUB est indigente, et n'encourage pas à la programmation modulaire.

— BASIC ne permet pas la récursivité, ce qui le rend mal adapté au traitement de certains problèmes.

— Les entrées-sorties BASIC sont soit rudimentaires, soit d'un emploi peu souple.

— Les entrées-sorties fichiers sont très rudimentaires et ne sont pas d'un emploi aisé pour les débutants.

2. — Points faibles de LSE

— LSE a été conçu pour être compilé ligne par ligne, ce qui impose qu'une ligne forme un tout. On peut regretter, de ce fait, l'absence de structure de contrôle nécessitant plusieurs lignes, du type BLOC IF, SELECT.

— LSE n'a pas d'opération sur les tableaux. Remarquons, cependant, que le système de procédures dynamiquement chargées permet aux utilisateurs gênés par cette caractéristique de pallier facilement cette absence.

— LSE, dans sa forme actuelle, ne traite pas les structures graphiques. Notons, cependant, qu'il existe une extension graphique de LSE, nommée GRAPHOL, qui manipule des structures graphiques d'une façon beaucoup plus élaborée que ce qui est prévu dans la norme BASIC étendu.

— LSE n'a pas d'équivalent de l'instruction DATA. La souplesse du traitement de chaînes donne cependant des possibilités d'initialisation simple.

3. — Points forts de BASIC

— BASIC a une diffusion mondiale. Il existe une bibliothèque de programmes dans de nombreux domaines.

Il est disponible sur de nombreux micro-ordinateurs (dans la version minimum généralement).

Ces programmes doivent cependant être généralement modifiés avant d'être utilisés sur un matériel donné, du fait du grand nombre de « dialectes » BASIC.

4. — Points forts de LSE

— LSE utilise un vocabulaire français. Il est bien défini.

— La définition de procédures permet une programmation modulaire, et permet d'enrichir le langage dans les domaines d'intérêt de l'utilisateur.

— Ses entrées-sorties sont souples et simples.

— La gestion de la mémoire est dynamique.

— La gestion de fichier est agréable à utiliser tout en étant plus puissante que celle de BASIC.

— La syntaxe de LSE est régulière et systématique.

— Il existe un important ensemble de programmes adaptés à l'enseignement français.

— Il est bien adapté au traitement de chaînes, et très efficace dans l'analyse de réponses, ce qui en fait un langage de choix pour la rédaction de programmes destinés à l'enseignement.

5. — Conclusion

Nous nous garderons bien d'établir un classement. Nous faisons simplement remarquer que l'on reconnaît dans BASIC la même démarche pragmatique que dans FORTRAN, LSE se réclame plutôt de la démarche cartésienne qui a déjà donné jour à ALGOL.

Stéphane BERCHE,
Ingénieur E.S.E.,
chef de travaux au service Informatique
de l'Ecole Supérieure d'Electricité,
Gif-sur-Yvette.