QL

MOTS-CLES

Guide de référence
des mots-clés du Superbasic

Ce guide de référence présente tous les termes du SuperBASIC par ordre alphabétique. Une brève explication de leur fonction est donnée suivie par une définition de la syntaxe et des exemples d'utilisation.

La plupart des mots clés font référence à une entrée dans le Guide de Référence des CONCEPTS.

Ex: DRAW est une opération graphique et des renseignements supplémentaires peuvent être obtenus dans la section graphique du Guide de référence des Concepts.

Vous trouverez ci-joint un index qui vous permet, à partir d'un mot-clé de trouver la page où il est décrit dans le Guide de référence des mots-clés ainsi que la section concernée dans la partie des concepts.

by Stephen Berry (Sinclair Research Limited)

INDEX

ABS

Fonctions mathématiques

ABS renvoie la valeur absolue du paramètre. Il renvoie le paramètre si celui-ci est positif et renvoie à zéro moins le paramètre si celui-ci est négatif.

Syntaxe : **ABS**(*expression numérique*)

Exemples :

i. `PRINT ABS(0.5)`
ii. `PRINT ABS(valeur)`

ADATE

Horloge

ADATE permet de régler l'horloge interne du QL.

Syntaxe *Secondes:= expression numérique*
ADATE *secondes*

Exemples :

i. `ADATE 3600` (avance l'horloge d'une heure)
ii. `ADATE -60` (retarde l'horloge d'une minute)

ARC

ARC_R

Graphiques

ARC dessine un arc de cercle entre deux points dans la fenêtre reliée au canal par défaut ou à un canal particulier. Les extrémités de l'arc sont spécifiées en utilisant le système de coordonnées graphiques.

Des arcs multiples peuvent être dessinés avec une simple commande **ARC**.

Normalement vous devez donner les coordonnées des deux extrémités de l'arc. Ces points peuvent être donnés en coordonnées absolues (relativement à l'origine des graphiques) ou en coordonnées relatives (relativement au curseur graphique). Si le premier point n'est pas précisé alors l'arc est tracé à partir de la position du curseur graphique jusqu'au point donné suivant l'angle donné.

ARC trace toujours en coordonnées absolues (relativement à l'origine des graphiques) tandis que **ARC_R** trace toujours relativement au curseur graphique.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique
angle:= expression_numérique (en radians)
point:= x,y
paramètres:= | point TO point, angle1
| TO point, angle2

où 1 trace du point de départ au point d'arrivée en tournant de la valeur de l'angle.
2 trace à partir du dernier point utilisé au point donné en tournant de la valeur de l'angle.

ARC *[canal,]* paramètre * [,paramètre]*

Exemples :

i. `ARC 10,10 TO 50,50,PI/2`
[dessine un arc en partant du point 10,10 jusqu'au point 50,50 en tournant de 90° tous les points sont absolus]

ii. `ARC R TO 47,34, PI/4`
[dessine un arc en partant du dernier point jusqu'au point 47,34 en tournant de 450°, tous les points sont relatifs]

AT

Fenêtres

AT permet de modifier la position d'affichage sur une grille ligne-colonne basée sur la taille des caractères. **AT** utilise une forme modifiée du système de coordonnées par point (pixel) où rangée 0, colonne 0 est le coin en haut à gauche de la fenêtre.

Syntaxe : *ligne:= expression_numérique*
colonne:= expression_numérique
AT *[canal,] ligne,colonne*

Exemple : `AT 10,20 : PRINT "ceci est la ligne 10 et la colonne 20"`

ASIN:ATAN

ACOS:ACOT

Fonctions mathématiques

ATAN et **ACOT** calculent l'arctangente et l'arccotengente respectivement. Il n'y a pas de limite pour la valeur du paramètre.

Syntaxe : *angle:= expression_numérique* (en radians)
ASIN*(angle)*
ATAN*(expr. numérique)*
ACOS*(angle)*
ACOT*(angle)*

Exemples :

i. `PRINT ATAN(expr.numérique)`
ii. `PRINT ASIN(1)`
iii. `PRINT ATAN(a-b)`
iv. `PRINT ACOT(3.6574)`

AUTO

Basic

AUTO permet de générer automatiquement les numéros de lignes en entrant directement le programme dans l'ordinateur. **AUTO** génère le numéro suivant en séquence et entre ensuite dans l'éditeur de lignes du SuperBASIC pendant que vous tapez la ligne. Si la ligne existe déjà, alors elle apparaît précédée de son numéro de ligne. Lorsque vous appuyez sur- ENTREE quelle que soit la position du curseur, un test est effectué sur la syntaxe de la ligne entière qui est passée au programme.

AUTO se termine en appuyant simultanément sur :

CTRL	BARRE D'ESPACEMENT

Syntaxe : *première_ligne:= numéro_ de_ligne*
intervalle:= expression_numérique
AUTO*[première ligne] [,intervalle]*

Exemples :

i. `AUTO` (commence à la ligne 100 avec intervalle =10)
ii. `AUTO 10,5` (commence à la ligne 10 avec intervalle = 5)

BAUD

Communications

BAUD permet de déterminer la vitesse de transmission par les deux canaux RS-232-C. La vitesse ne peut pas être choisie séparément pour les deux canaux.

Syntaxe : *vitesse:= expression_numérique*
BAUD vitesse

La valeur du paramètre vitesse doit être une des valeurs ci-dessous, sinon une erreur est signalée.

75
300
600 1200
2400
4800
9600
19200 (uniquement en transmission)

Exemples :

i. `BAUD 9600`
ii. `BAUD vitesse8édition`

BEEP

Sons

BEEP active les fonctions sonores du QL. **BEEP** peut accepter un nombre variable de paramètres pour permettre plusieurs niveaux de contrôle sur le son produit. Le minimum à donner est la durée et le ton. **BEEP** utilisé sans aucun paramètre arrête le son généré.

Syntaxe :

durée:= expression_numérique	(de -32768 à 32767)
ton:= expression_numérique	(de 0 à 255)
répétition:= expression_numérique	(de 0 à 15)
grad x:= expression_numérique	(de -32768 à 32767)
grady:= expression_numérique	(de -8 à 7)
trouble:= expression_numérique	(de 0 à 15)
hasard:= expression_numérique	(de 0 à 15)

BEEP [durée,ton
[ton_2, grad_x, grad_y
[,répétition
[,trouble
[,hasard]]]]]

avec

durée indique la durée du son en unités de 72 microsecondes. Avec une durée de valeur 0, le son continuera jusqu'au prochain BEEP.

ton indique la tonalité du son généré. Une valeur 1 est haute et une valeur de 255 est basse.

ton_2 indique le ton le plus élevé sur lequel le son "rebondira".

grad_x définit l'intervalle de temps entre chaque pas de la progression d'un ton vers l'autre.

grad_y définit la taille de chaque pas. grad_x et grad_y contrôlent la manière dont le son « rebondira » entre les deux tons.

répétition force le son à se répéter le nombre de fois indiqué. Si ce paramètre est 15, le son se répète sans arrêt.

Commentaire : mieux vaut utiliser la commande BEEP que d'analyser sa syntaxe.

BEEPING

Son

BEEPING est une fonction qui retourne la valeur zéro (faux) si aucun son n'est émis par le QL et une valeur différente de zéro (vrai) si un son est émis.

Syntaxe : *BEEPING*

Exemples :

```
10 DEFine PROCedure silence
20 BEEP
30 END DEFine
40 IF BEEPING THEN silence
```

BLOCK

Fenêtres

BLOCK génère un bloc d'une taille et d'une forme données â la position relative donnée par rapport à l'origine de la fenêtre reliée au canal par défaut ou â un canal donné.

BLOCK utilise le système de coordonnées par points (pixel)

Syntaxe : *largeur := expression_numérique*
hauteur:= expression_numérique
x:= expression_numérique
v:= expression_numérique

BLOCK [canal,]*largeur,hauteur,x,y,couleur*

Exemples :

i.
```
BLOCK 10, 10, 5, 5, 7
```
(dessine un block blanc de 10x10 points â la position 5,5)

ii.
```
10 REMark petit programme de diagramme
20 PRINT "diagramme en barre"
30 LET bas=20:gauche=40:Largeur=10
40 FOR barre =1 TO 20
45   LET taille = RND (10 TO 150 )
50   LET couleur = RND(0 TO 255)
60   BLOCK largeur,taille, gauche+barre*largeur,bas,couleur
70   BLOCK largeur-2,taille-2,gauche+barre*largeur+1,bas+1,couleur
```
iii.
```
80 END FOR barre
```

(utilisez LET couleur=RND(0 TO 7) pour les téléviseurs)

BORDER

Fenêtres

BORDER ajoute un bord à la fenêtre reliée au canal par défaut ou au canal donné. Pour toutes les opérations sauf **BORDER** la taille de la fenêtre est réduite pour laisser de la place pour **BORDER**. Si une autre commande **BORDER** est utilisée, alors la fenêtre reprend sa taille d'origine avant que le bord ne soit ajouté ; donc des commandes **BORDER** successives changent la taille et la couleur d'un seul bord. Des bords multiples ne sont jamais créés sauf si une action spécifique est faite.

Si **BORDER** est utilisé sans préciser une couleur, alors un bord transparent de la hauteur donnée est créé.

Syntaxe *largeur:= expression_numérique*

BORDER [canal,]largeur[,couleur]

Exemples :

i. `BORDER 10,0,7` (bordure tramée noire et blanche)

ii.
```
10 REMark traçons des bords
20 FOR épaisseur = 50 TO 2 STEP –2
30   BORDER épaisseur,RND(0 TO 255)
40 END FOR épaisseur
50 BORDER 50
```

(Utilisez RND (0 TO 7) pour un téléviseur)

CALL

Code machine

Le code machine peut être accédé directement à partir du SuperBASIC en utilisant la commande **CALL**. **CALL** accepte jusqu'à treize paramètres qui sont placés en séquence dans les registres de données et d'adresses du 68008 (DO à D7 et AO à A5).

Aucune donnée n'est retournée par le CALL.

Syntaxe : *adresse:= expression_numérique*
donnée: =expression_numérique

CALL *adresse, *[données]*(13 paramètres maximum)*

Exemples :

i. `CALL 262144,0,0,0`
ii. `CALL 263500,12,3,4,1212,6`

Remarque : Le registre d'adresse A6 ne doit pas être utilisé à l'intérieur de routines assembleur, ainsi appelées.

Pour retourner au SuperBASIC, utilisez les instructions :

```
MOVEQ #0,D0
RTS
```

CHR$

Basic

CHR$ est une fonction qui renvoie le caractère dont la valeur décimale est donnée en paramètre.

CHR$ est l'inverse de **CODE**.

Syntaxe : *CHR$(expression_numérique)*

Exemples :

i. `PRINT CHR$ (27)` (affiche de caractère escape)
ii. `PRINT CHR$ (65)` (affiche A)

CIRCLE

CIRCLE_R

Graphiques

CIRCLE trace un cercle (ou une ellipse à un angle donné) sur l'écran â une position donnée et d'une taille déterminée. **CIRCLE** utilise le système de coordonnées graphiques. Le cercle sera tracé dans la fenêtre reliée au canal par défaut ou à un canal donné.

CIRCLE utilise le système de coordonnées graphiques en coordonnées absolues. (relativement à l'origine des graphiques)

CIRCLE_R utilise les coordonnées relatives par rapport au curseur des graphiques.

On peut tracer plusieurs cercles ou ellipses avec une seule instruction **CIRCLE**, chaque groupe de paramètres étant séparé du suivant par un point virgule(;). Le mot **ELLIPSE** peut être substitué à **CIRCLE** si nécessaire.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique
rayon:= expression_numérique
excentricité:= expression_numérique
angle:= expression_numérique (de 0 à 2PI)
paramètres:= | *x,y,rayon* *1*
|x,y,rayon,excentricité,angle *2*

1. trace un cercle
2. trace une ellipse d'excentricité et d'angle donnés

CIRCLE *[canal,]paramètres*[;paramètres]**

x *déplacement horizontal à partir de l'origine des graphiques ou du curseur.*
y *déplacement vertical à partir de l'origine des graphiques ou du curseur.*
rayon : *rayon de cercle*
excentricité : *rapport entre les deux axes de l'ellipse*
angle : *orientation de l'axe principal de l'ellipse relativement à la verticale de l'écran. Cet angle doit être donné en radians.*

Exemples :

i. `CIRCLE 50,50,20` (cercle à 50,50 de rayon 20)
ii. `CIRCLE 100,100,20,0.5,0` (ellipse à 100,100, grand axe=20 excentricité = 0.5 et alignée sur l'axe vertical)

CLEAR

Basic

CLEAR efface les zones de variables du SuperBASIC pour le programme ce qui libère cette place pour le système d'exploitation QDOS.

Syntaxe : CLEAR

Exemple : `CLEAR`

Remarque : **CLEAR** peut être utilisé pour remettre le système SuperBASIC dans un état connu. Par exemple, si un programme est arrêté par une erreur pendant qu'il est dans une procédure alors le SuperBASIC est encore bloqué dans la procédure même après l'arrêt du programme. CLEAR remet le SuperBASIC dans l'état normal.

CLOSE

Périphériques

CLOSE ferme le canal donné. Toute fenêtre associée avec le canal est désactivée.

Syntaxe : *canal:= #expression_numérique*

CLOSE *canal*

Exemples :

i. `CLOSE #4`
ii. `CLOSE #canal_entrée`

CLS

Fenêtres

CLS efface la fenêtre reliée au canal par défaut ou au canal donné dans la couleur actuelle déterminée par **PAPER** à l'exclusion de la bordure s'il y en a une. **CLS** accepte un paramètre optionnel précisant si une partie seulement de la fenêtre doit être effacée.

Syntaxe : *partie:= expression_numérique*

CLS [canal] [partie]

avec :
- *partie = 0* - *efface tout l'écran (par défaut si pas de paramètre)*
- *partie = 1* - *efface le haut sauf la ligne du curseur*
- *partie = 2* - *efface le bas sauf la ligne du curseur*
- *partie = 3* - *efface toute la ligne du curseur*
- *partie = 4* - *efface à droite sur la ligne y compris la position du curseur.*

Exemples :

i. `CLS` (efface toute la fenêtre)
ii. `CLS 3` (efface toute la ligne du curseur)
iii. `CLS #2,2` (efface le bas de la fenêtre sur le canal 2)

CODE

Basic

CODE est une fonction qui restitue le code interne utilisé pour représenter le caractère donné. Si une chaîne de caractères est donnée en paramètres, alors **CODE** renvoie la représentation interne du premier caractère de la chaîne.

CODE est l'inverse de **CHR$**

Syntaxe : CODE(chatne de caractères)

Exemples :

i. `PRINT CODE ("A")` [affiche 65]
ii. `PRINT CODE ("SuperBASIC")` [affiche 83]

CONTINUE

RETRY

Basic

CONTINUE permet à un programme qui a été arrêté de se poursuivre. **RETRY** permet d'exécuter à nouveau une instruction signalée en erreur, après rectification éventuelle de données.

CLEAR peut être utilisé pour remettre le SuperBASIC dans son état normal si **CONTINUE** ou **RETRY** ne sont pas appropriés.

Syntaxe : CONTINUE
RETRY

Exemples :
```
CONTINUE
RETRY
```

Attention : Un programme ne peut continuer que si :

1. aucune nouvelle ligne n'est ajoutée au programme.
2. aucune nouvelle variable n'est ajoutée au programme
3. aucune ligne n'a été changée

Les valeurs des variables existantes peuvent être changées avant **CONTINUE** ou **RETRY**.COPY

COPY_N

Périphériques

COPY copie un fichier d'un périphérique d'entrée vers un périphérique de sortie jusqu'à ce que la fin du fichier soit détectée.

COPY_N enlève les entêtes associés au fichier Microdrive, ce qui permet de copier les fichiers Microdrives vers d'autres types de périphériques.

Syntaxe : *COPY périphérique TO périphérique*
COPY_N périphérique TO périphérique

Il doit être possible de lire à partir du périphérique d'entrée et d'écrire sur le périphérique de sortie.

Exemples :

i. `COPY mdvl_fichier données TO con_`
(affiche le contenu du fichier à l'écran)

ii. `COPY neti_3 TO mdvl_données`
(copie des données venant de neti_3 dans le fichier mdv1_données)

iii. `COPY_N mdvl_test_données TO ser1`
(copie le fichier mdv1_test_données sur le port de sortie ser_1 en enlevant les informations d'entête).

COS

Fonctions mathématiques

COS calcule le cosinus de l'angle donné.

Syntaxe : *angle:= expression_numérique* (de -60000 à +60000 radians)

COS(angle)

Exemples :

i. `PRINT COS(alpha)`
ii. `PRINT COS(3.141592654/2)`

COT

Fonctions mathématiques

COT calcule la cotangente de l'angle donné.

Syntaxe : *angle:= expression_numérique* (de -30000 à +30000 radians)

COT(angle)

Exemples :

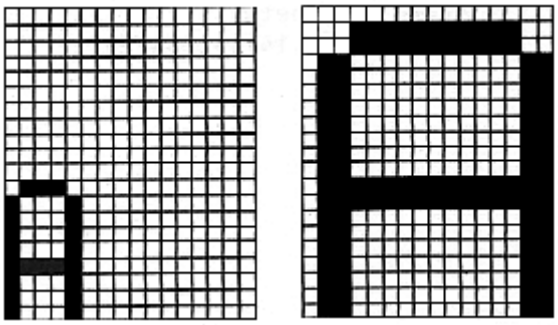
i. `COT(3)`
ii. `COT(3.141592654/2)`

CSIZE

Ecran

Définit la nouvelle taille des caractères pour la fenêtre attachée au canal par défaut ou à un canal donné. La taille standard est 0,0 en mode 512 et 2,0 en mode 2.56.

La largeur définit la taille horizontale du caractère espace. Huit tailles peuvent être définies pour le caractère espace. La taille du caractère est ajustée pour remplir la place disponible.



largeur	taille	hauteur	taille·
0	6 points	0	10 points
1	8 points	1	20 points
2	12 points		
3	16 points		

Syntaxe : *largeur:= expression numérique (de 0 à 3)*
hauteur:= expression numérique (de 0 à 1)

CSIZE*[canal,]largeur, hauteur*

Exemples :

i. `CSIZE 3,0`
ii. `CSIZE 3,1`

CURSOR

Fenêtre

CURSOR permet le positionner le curseur de l'écran n'importe où dans la fenêtre attachée au canal par défaut ou à un canal donné.

CURSOR utilise le système de coordonnées par points relativement à l'origine de la fenêtre et permet de définir la position du sommet gauche du curseur, la taille du curseur dépend de la taille des caractères.

Si **CURSOR** est utilisé avec quatre paramètres alors les deux premiers sont les coordonnées graphiques et les deux suivants, la position du curseur (en coordonnées par points) relativement au premier point.

Ceci permet de rajouter facilement des textes sur des diagrammes.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique

CURSOR [canal,]x,y,[,x,y]

Exemples :

i. `CURSOR 0,0`
ii. `CURSOR 20,30`
iii. `CURSOR 50,50,10,10`

DATA READ

RESTORE

Basic

READ, **DATA** et **RESTORE** permettent d'assigner des données contenues dans le SuperBASIC à des variables au moment de l'exécution.

DATA est utilisé pour définir la donnée,

READ accède à la donnée et l'assigne aux variables,

RESTORE permet de sélectionner des données.

DATA

Permet de définir une donnée à l'intérieur du programme. La donnée peut être lue par l'instruction **READ** et assignée à des variables. Une instruction **DATA** est ignorée par le SuperBASIC quand il la rencontre durant l'exécution du programme.

Syntaxe : DATA **[expression,]**

RESTORE

Repositionne le pointeur des données, par exemple, la position à partir de laquelle les instructions **READ** qui suivent vont lire leurs données. Si **RESTORE** est suivi par un paramètre, alors le pointeur des données est positionné sur cette valeur. Si aucun paramètre n'est précisé, alors le pointeur des données est positionné au début du programme. Il est conseillé de prévoir systématiquement un **RESTORE** au début des programmes comportant des **READ**.

Syntaxe : RESTORE *[numéro de ligne]*

READ

Transfère les données contenues dans les instructions **DATA** et les assigne aux variables. Les données sont initialement lues à partir de la première zone de la première instruction **DATA** du programme. Les autres **READ** transfèrent les unités suivantes de **DATA** et des **DATAs** suivants. Un message d'erreur apparaît si un **READ** ne trouve rien à lire. La commande **RESTORE** peut être utilisée pour donner le numéro de ligne à partir de laquelle les données seront lues.

Syntaxe : READ **[identifiant,]**

Exemples :

i.
```
10 REMark exemple d'utilisation de DATA
20 DIMension jour$(7,3) : REST0RE
30 FOR compteur = 1 TO 7: READ jour$(compteur)
40 PRINT jour$
50 DATA "LUN","MAR","MER","JEU","VEN"
60 DATA "SA M","DIM"
```

ii.
```
10 DIM mois$ (12,9)
20 REMark exemple d'utilisation de DATA
30 FOR compteur = 1 TO 12 : READ mois$(compteur)
40 PRINT mois$
50 RESTORE 10
60 DATA "Janvier","Février","Mars"
70 DATA "Avril","Mai","Juin"
80 DATA "Juillet","Août","Septembre"
90 DATA "Octobre","Novembre","Décembre"
```

Attention : Le **RESTORE** n'est pas fait automatiquement avant de lancer le programme. Cela permet à un même programme d'utiliser des données différentes.

DATE$

DATE

Horloge

DATE$ donne la date et l'heure contenues dans l'horloge du QL, qui doit être initialisée.

Le format de la chaîne de caractères envoyée par **DATE$** est : "aaaa mm jj hh:mm:ss"

avec aaaa est l'année : 1984, 1985, etc
mm est le mois janvier, etc...
jj est le jour 01 à 28, 29, 30, 31
hh est l'heure 0 à 23
mm les minutes 0 à 59
ss les secondes 0 â 59.

DATE renvoie la date comme un nombre en virgule flottante qui peut être utilisé pour stocker les dates et heures sous forme compacte.

Si **DATE$** est utilisé avec un paramètre numérique alors ce paramètre sera interprété comme une date en virgule flottante et sera converti en une chaîne de caractères.

DAY$

Horloge

DAY$** est une fonction qui renvoie la date courante de la semaine. Si un paramètre est spécifié alors **DAY$ interprète le paramètre comme une date et renvoie le jour correspondant de la semaine.

Syntaxe : *DAY$* (donne le jour à partir de l'horloge)

DAY$(*paramètre)* (donne le jour â partir du paramètre)

Exemples :

i. `PRINT DAY$`
ii. `PRINT DAY$(date)`

DEG

Fonctions mathématiques

DEG est une fonction qui convertit un angle donné en radians en un angle donné en degrés.

Syntaxe : **DEG***(expression_numérique)*

Exemple : `PRINT DEG(PI/2)` (donne 90)

DEFINE FUNCTION

END DEFINE

Fonctions et procédures

DEFine FUNction définit une fonction SuperBASIC. La séquence d'instructions comprise entre **DEFine** et **END DEFine** constitue la fonction. La définition de la fonction peut également inclure une liste de paramètres formels qui fournissent des données à la fonction. Les paramètres formels ou actuels doivent être mis entre parenthèses. Si la fonction ne réclame pas de paramètre il n'est pas nécessaire de mettre les parenthèses. Les paramètres formels prennent leurs types et caractéristiques à partir des paramètres actuels correspondants. Une réponse est envoyée par la fonction en ajoutant une expression à l'instruction **RETurn**. Le type de données est indiqué par le caractère rajouté au nom de la fonction. Un $ indique une chaîne de caractères, un % indique un entier, aucun caractère indique une donnée en virgule flottante. Une fonction est lancée en donnant son nom à l'intérieur d'une fonction SuperBASIC.

Les appels de fonctions en SuperBASIC peuvent être récursifs, ce qui veut dire qu'une fonction peut s'appeler elle-même directement ou indirectement par l'intermédiaire d'une séquence d'appels.

Syntaxe : *paramètres formels:= (expression*[,expression]*)*
paramètres actuels:= (expression[, expression]*)*
DEF FuNction identifiant l $ l % l [paramètres_formels]
[local identifiant [, identifIant]*]*
RETURN expression
END DEFine

RETURN peut se trouver n'importe où dans le corps de la procédure. **LOCAL** doit précéder la première instruction exécutable dans la fonction.

Exemple :

```
10 DEFine FuNction moyenne(a,b,c)
20    LOCAL réponse
30    LET réponse = (a+b+c)/3
40    RETURN réponse
50 END DEFine
60 PRINT moyenne(1,2,3)
```

DEFINE PROCEDURE

END DEFINE

Fonctions et Procédures

DEFine PROCedure définit une procédure SuperBASIC. La séquence d'instructions entre **DEFine PROCedure** et **END DEFine** constitue la procédure. La définition de la procédure peut également inclure une liste de paramètres formels qui fourniront les données à la procédure. Les paramètres formels doivent être mis entre parenthèses dans la définition de la procédure, mais les parenthèses ne sont pas nécessaires quand la procédure est appelée. Si la procédure ne réclame pas de paramètres, il n'est pas nécessaire de les mettre dans la définition.

Les paramètres formels prennent leurs types et leurs caractéristiques à partir de paramètres actuels correspondants.

Les variables peuvent être définies à l'intérieur de la procédure **LOCAL**. Ces variables locales n'ont pas d'effet sur les variables de même nom â l'extérieur de la procédure. Si cela est nécessaire, des tableaux peuvent également être définis dans la procédure à l'intérieur de l'instruction **LOCAL**.

La procédure est appelée en donnant son nom comme première commande d'une instruction SuperBASIC en y associant la liste des paramètres actuels. Les appels de procédure en SuperBASIC peuvent être récursifs, ce qui veut dire qu'une procédure peut s'appeler elle-même directement ou indirectement par l'intermédiaire d'une séquence d'appels.

On peut considérer une procédure comme une commande du SuperBASIC ; de nombreuses commandes système sont elles-mêmes définies comme des procédures.

Syntaxe : *paramètres_formels:= (expression *[,expression]*)*
paramètres_actuels:= expression[,expression]**

DEFine PROCedure identifiant [paramètres_formels]
*[LOCAL identifiant *[,identifiant]**
instructions SuperBASIC
[RETURN]
END DEFine

RETURN peut être placé n'importe où, dans le corps de la procédure. L'instruction **LOCAL** doit se trouver avant la première instruction exécutable dans la procédure. L'instruction **END DEFine** agit comme un retour automatique.

Exemples :

i.
```
10 DEFine PROCedure démarrage_écran
20   WINDOW 100,100,10,10
30   PAPER 7 : INK 0 :CLS
40   BORDER 4,255
50   PRINT "Bonjour tout le monde"
60   RETURN
70 END DEFine
```

ii.
```
10 DEFine PROCedure défilement_lent(défilement_limite)
20   LOCAL compteur
30   FOR comteur = 1 TO defilement_limite
40     SCROLL 2
50   END FOR compteur
60 END DEFine
```

Commentaire : Pour améliorer la lisibilité des programmes le nom de la fonction peut être rajouté derrière **END DEFine**, mais ceci n'est pas testé par le SuperBASIC.

DELETE

Microdrives

DELETE permet d'enlever un fichier contenu sur la cartouche dans le Microdrive spécifié.

Syntaxe : **DELETE** *périphérique*

Le périphérique est obligatoirement un Microdrive.

Exemples :

i. `DELETE MDV1_anciennes_données`
ii. `DELETE MDV2_fichier_lettre`

DIM

Tableaux

DIM permet de définir un tableau en SuperBASIC. On peut définir des tableaux de type chaîne de caractères, d'entiers et de numériques en virgule flottante. Les tableaux de type chaîne de caractères contiennent des chaînes de longueur fixe. Le paramètre final de **DIM** détermine la longueur de la chaîne.

Les indices d'un tableau vont de zéro à l'index maximum donné dans l'instruction. Donc **DIM** génère un tableau avec un élément de plus dans chaque dimension. Quand un tableau est spécifié, il est initialisé à zéro quand il est de type numérique et avec des chaînes de caractères de longueur nulle pour un tableau de type chaîne de caractères.

Syntaxe : *dimension:= expression_numérique*
tableau:= identifiant(dimension[,dimension]*)*

DIM *tableau*[,tableau]**

Exemples :

i. `DIMension chaine$(10,10,50)`
ii. `DIM matrice(100,100)`

DIMN

Tableaux

DIMN est une fonction qui renvoie la taille maximum de la dimension donnée pour ce tableau. Si le paramètre dimension n'est pas précisé, c'est qu'il s'agit de la première dimension du tableau ; si elle n'existe pas ou si l'identifiant n'est pas un tableau alors la réponse est zéro.

Syntaxe : *tableau:= identifiant*
dimension:= expression_numérique (1 pour la dimension 1, etc…)

DIMN*(tableau[,dimension])*

Exemples : Soit un tableau défini par DIM a(2,3,4)

i. `PRINT DIMN(a,1)` donne 2
ii. `PRINT DIMN (a,2)` donne 3
iii. `PRINT DIMN(a,3)` donne 4
iv. `PRINT DIMN (a)` donne 2
v. `PRINT DIMN(a,k)` donne 0

DIV

Opérateur

DIV est un opérateur qui effectue une division entière

Syntaxe : *expression_numérique DIV expression_numérique*

Exemples :

i. `PRINT 5 DIV 2` donne 2
ii. `PRINT-5 DIV 2` donne -3

DIR

Périphériques

DIR permet de visualiser le contenu d'une cartouche Microdrive.

Syntaxe : **DIR** *périphérique*

Le périphérique doit être un Microdrive.

Le format affiché par DIR est :

secteurs libres:=	nombre de secteurs libres
secteurs disponibles:=	nombre de secteurs disponibles en tout sur la cartouche.
nom_ fichier:=	nom d'un fichier
format affiché:	nom du volume secteurs _libres | secteurs _disponibles nom_fichier … nom_fichier

Exemples :

i. `DIR MDV 1__`
ii. `DIR "MDV2_"`
iii. `DIR "MDV_" & numéro_microdrive$ & "_"`

Exemple d'affichage à l'écran :

BASIC_2 FEB
183/221 sectors
demo_1
demo_1_ancien
demo_2

DLINE

Basic

DLINE supprime une ou une série de lignes d'un programme SuperBASIC.

Syntaxe : *intervalle:=* | *numéro_ligne TO numéro_ligne* 1
| *numéro_ligne TO* 2
| *TO numéro ligne* 3
| *numéro_ligne* 4

DLINE *intervalle[,intervalle]*

Avec :

1. suppression des lignes entre les deux numéros(compris)
2. suppression à partir de la ligne donnée jusqu'à la fin.
3. suppression à partir du début jusqu'à la ligne.
4. suppression de la ligne donnée.

Exemples :

i. `DLINE 10 TO 70, 80, 200 TO 400`
(supprime les lignes de 10 à 70 comprises, la ligne 80 et les lignes de 200 à 400 comprises).

ii. `DLINE`
(ne supprime rien)

EDIT

Editeur

La commande **EDIT** met en action l'éditeur de lignes de SuperBASIC.

La commande **EDIT** est très proche de la commande **AUTO**, la seule différence concerne les valeurs par défaut. Par défaut, **EDIT** a une incrémentation de lignes de zéro et donc édite une seule ligne, sauf si un second paramètre est donné pour définir la valeur de l'incrémentation de ligne. Si la ligne existe, elle est visualisée et son édition peut commencer. Si la ligne n'existe pas, alors le numéro de ligne est affiché et la ligne peut être saisie.

Le curseur peut être manipulé en utilisant les touches standards du QL. Quand la ligne est correcte, le fait d'actionner la touche **ENTREE** passe la ligne au programme.

Si un incrément est spécifié la prochaine ligne en séquence sera éditée, sinon la commande **EDIT** est terminée.

Syntaxe : *pas: =expression_numérique*

EDIT num*éro_ligne [,pas]*

Exemples :

i. `EDIT  10` (édite la ligne 10 seule)
ii. `EDIT 20,10` (édite les lignes 20, 30, 40, etc...)

[→] curseur à droite

[←] curseur à gauche

[CTRL] [→] efface le caractère de droite

[CTRL] [←] efface le caractère de gauche

EXEC

EXEC_W

Multitâches

EXEC et **EXEC_W** permettent de charger une séquence de programmes et de les exécuter en parallèle.

EXEC rend la main après que les programmes aient commencé leur exécution.

EXEC_W attend que tous les programmes soient terminés avant de rendre la main.

Syntaxe :

programme:= périphérique (utilisé pour déterminer le fichier microdrive contenant le programme.)

EXEC programme

Exemples :

i. `EXEC MDV1_communications`
ii. `EXEC_W MDV1_édition_imprimante`

EXIT

Répétition

EXIT permet de reprendre l'exécution après l'instruction **END** de la structure **FOR** ou **REPeat**.

Syntaxe : **EXIT** *identifiant*

Exemples :

i.
```
10 REM début de la boucle: Let compteur = 0
20 REPeat boucle
30   LET compteur = compteur + 1
40   PRINT compteur
50   IF compteur = 20 THEN EXIT boucle
60 END REPeat boucle
```
Dans cet exemple, on sort de la boucle quand compteur=20

ii.
```
10 REPeat boucle externe
20   FOR n = 1 TO 100
30     REM instructions
40     REM instructions
50     IF RND>.5 THEN EXIT boucle externe
55   END FOR n
60 END REPeat boucle externe
```
On sort des deux boucles quand un nombre aléatoire plus grand que 0,5 est généré.

EXP

Fonctions mathématiques

EXP renvoie la valeur de e élevé à la puissance donnée par le paramètre.

Syntaxe : **EXP***(expression_numérique)* de -500 à 500.

Exemples :

i. `EXP(3)`
ii. `EXP(PI)`

EOF

Périphériques

EOF est une fonction qui détermine la fin d'un fichier lorsqu'elle est atteinte sur le canal considéré. Si **EOF** est utilisé sans spécification de canal alors **EOF** sera effectif si les données « DATA » du programme sont entièrement lues.

Exemples :

i. `IF EOF (#6) THEN STOP`
ii. `IF EOF THEN print "Plus de données"`

FILL

Graphiques

FILL permet de mettre en place le mécanisme de remplissage ou de l'arrêter. **FILL** permet de remplir toute forme non réentrante (concave) dessinée par une procédure graphique au moment où elle se trace à l'écran. Les formes réentrantes doivent être découpées en formes non réentrantes pour garantir un remplissage correct.

Avant de tracer une nouvelle forme, **FILL** doit être remis à zéro grâce à l'instruction FILL0

Syntaxe : *paramètre:= expression_numérique (0 ou 1)*

FILL *paramètre*

Exemples :

i. `FILL 1 : LINE 10,10 TO 50,50 TO 30,90 TO 10,10`
dessine un triangle plein
ii. `FILL 1: CIRCLE 50,50,20`
dessine un cercle plein

FILL$

Chaînes

FILL$ est une fonction qui renvoie une chaîne de longueur donnée suivie d'une répétition de un ou deux caractères.

Syntaxe : **FILL$(***expression_chaine,expression_numérique)*

L'expression chaîne doit avoir 1 ou 2 caractères.

Exemples :

i. `PRINT FILL$("a",5)` (donne aaaaa)
ii. `PRINT FILL$("aB",7)` (donne aBaBaBa)
iii. `LET a$ = a$ + FILL$(" ",10)`

FLASH

Fenêtres

FLASH permet de faire clignoter des caractères ou d'enlever le clignotement. **FLASH** ne fonctionne qu'en basse résolution.

FLASH produit son effet dans la fenêtre associée au canal par défaut ou â un canal donné.

Syntaxe : *paramètre:= expression_numérique (0 à 1)*

FLASH *[canal,]paramètre*

paramètre = 0 , enlève le clignotement
paramètre = 1, met le clignotement

Exemples :

```
10 PRINT "A";
20 FLASH 1
30 PRINT "clignotant";
40 FLASH 0
50 PRINT "non clignotant"
```

Attention :
Réécrire sur des caractères clignotants peut donner des résultats curieux et doit être évité.

FOR

END FOR

Répétition

L'instruction **FOR** permet d'exécuter un groupe d'instructions SuperBASIC un nombre précis de fois. L'instruction **FOR** peut être utilisée sous deux formes : longue ou courte.

NEXT et **END FOR** peuvent être utilisés ensemble â l'intérieur d'une même boucle **FOR**. Ceci permet ce que l'on appelle un épilogue de boucle. Par exemple, un groupe d'instructions SuperBASIC qui n'est pas exécuté si la sortie de la boucle se fait par **EXIT**, s'exécutera si la boucle **FOR** se termine normalement. Donc **EXIT** permet de forcer la sortie d'une boucle **FOR NEXT** ou **END FOR**.

Définition *for_numéro:=* | *expression_numérique*
| *expression_numérique TO expression_numérique*
| *expression_numérique TO exp_num STEP exp_nurn*
for_liste:=for_numéro[,for_numéro]*

Forme courte

L'instruction **FOR** est suivie sur la même ligne logique par une séquence d'instruction SuperBASIC. L'exécution de la séquence d'instruction est alors rejetée sous contrôle de l'instruction **FOR**. Quand l'instruction **FOR** est terminée, le programme se continue sur la ligne suivante. L'instruction **FOR** ne nécessite pas de se terminer par **NEXT** ou **END FOR**.

Syntaxe : *FOR variable:=for_liste:instruction*[instruction]**

Exemples :

i. `FOR i = 1, 2, 3, 4 TO 7 STEP 2 : PRINT i`
ii. `FOR élément = premier TO dernier : tableau(élément) = 0`

Forme longue

L'instruction **FOR** est la dernière instruction sur la ligne. Les lignes suivantes contiennent une série d'instructions SuperBASIC terminées par une instruction **END FOR**. Les instructions comprises entre **FOR** et **END FOR** sont exécutées sous contrôle de l'instruction **FOR**.

Syntaxe : *FOR variable = for_liste*
instructions
END FOR variable

Exemples :

```
10 INPUT "envoyez les données S.V.P." ! x
30 LET factoriel = 1
40 FOR valeur = x TO 1 STEP -1
50   LET factoriel = factoriel*valeur
60   PRINT x !!! factoriel
70   IF factoriel> 1E20 THEN
80      PRINT "nombre trop grand"
90      EXIT factoriel
100  END IF
110 END FOR valeur
```

Commentaire :

Dans une boucle **FOR** simple (sans **EXIT**) on peut utiliser **END FOR** ou **NEXT**.

Attention :

Une variable en virgule flottante doit être utilisée pour contrôler une boucle **FOR**

FORMAT

Microdrives
(Microlecteurs)

FORMAT rend utilisable la cartouche contenue dans le Microdrive donné.

Syntaxe : *FORMAT[canal,]périphérique*

Périphérique désigne le microdrive qui est utilisé pour le formatage suivi du nom que l'on veut donner à cette cartouche. FORMAT affichera le nombre de secteurs utilisables et le nombre total de secteurs disponibles sur la cartouche.

Il est intéressant de formater une cartouche neuve plusieurs fois avant de l'utiliser, ceci permet de disposer d'une capacité plus importante.

Exemples :

i. `FORMAT MDV1_cartouche_données`
ii. `FORMAT MDV2_lettres_clients`

Attention !

FORMAT peut être utilisé pour réinitialiser une cartouche déjà utilisée. Dans ce cas, son contenu est perdu.

GOSUB

Basic

Le SuperBASIC pour permettre la compatibilité avec d'autres BASICs dispose de l'instruction **GOSUB**. **GOSUB** transfère l'exécution au numéro de ligne donné. L'instruction **RETurn** transfère l'exécution derrière l'instruction qui suit **GOSUB**. Le numéro de ligne peut être une expression.

Syntaxe : *GOSUB numéro_ligne*

Exemples :

i. `GOSUB 100`
ii. `GOSUB 4*variable_sélection`

Commentaire :

La construction de programmes structurés en SuperBASIC rend l'instruction GOSUB inutile.

GOTO

Basic

Pour permettre la compatibilité avec d'autres BASICs le SuperBASIC dispose de l'instruction **GOTO**.

GOTO transfère de manière inconditionnelle l'exécution au numéro de ligne donné. Le numéro de ligne peut être une expression.

Syntaxe : *GOTO numéro_ligne*

Exemples :

i. `GOTO début_ programme`
ii. `GOTO 9999`

Commentaire :

La construction de programmes structurés en SuperBASIC rend l'instruction GOTO inutile.

IF THEN

ELSE END IF

Extension

L'instruction **IF** permet de tester des conditions et au résultat de ce test de contrôler la suite du programme. Elle peut être utilisée sous trois formes (COURTE, LONGUE 1, LONGUE 2)

COURTE

Le mot-clé **THEN** est suivi sur la même ligne logique par une suite d'instructions de SuperBASIC. Ces instructions sont exécutées si l'expression comprise dans l'instruction **IF** est vraie ou si l'expression n'est pas égale à zéro.

Syntaxe : *instructions:= instruction* [:instruction]**

IF *expression* THEN *instructions [*:ELSE *instructions]*

Exemples :

i. `IF A = 32 THEN PRINT "limite atteinte": ELSE PRINT "o.k"`
ii. `IF données_testées = maximum THEN LET maximum = données_ testées`
iii. `IF a THEN PRINT "a n'est pas nul"`
iv. `IF "1"+1=2 THEN PRINT "facilité du SuperBASIC"`

LONGUE 1.

Le mot-clé **THEN** est le dernier élément sur la ligne logique. Une suite d'instructions de SuperBASIC suit les instructions **IF**. La séquence est terminée par l'instruction **END IF**. La séquence d'instructions SuperBASIC est exécutée si l'expression contenue dans l'instruction **IF** a pour valeur 1 ou si l'expression est vraie ou n'est pas égale â zéro.

Syntaxe : IF *expression* THEN
instructions
END IF

Exemple :

```
10 LET limite = RND(1 TO 50) : compteur_erreurs=0
20 FOR essai = 1 TO 20
30   INPUT "tapez un nombre": nombre
40   IF nombre>limite THEN
50     LET compteur_erreurs = compteur_erreurs+1
60     PRINT "en dehors, le nombre d'erreurs est"!
       compteur_erreurs
70     IF compteur erreurs = 5 THEN
80        PRINT "trop d'erreurs"
90    EXIT essai
100  END IF
110 END IF
120 END FOR essai
```

LONGUE 2.

Le mot-clé **THEN** est la dernière entrée sur la ligne logique. Une suite d'instructions de SuperBASIC suit sur les lignes suivantes, conclue par le mot-clé **ELSE**. Si l'expression contenue dans l'instruction **IF** est non nulle, alors cette première séquence d'instruction de SuperBASIC est traitée. Après le mot-clé **ELSE** une deuxième séquence d'instructions de SuperBASIC est entrée, conclue par le mot-clé **END IF**. Si l'expression évaluée par l'instruction **IF** est égale à 0, alors cette deuxième séquence d'instructions de SuperBASIC est exécutée.

Syntaxe : *IF expression THEN*
instructions
ELSE
instructions
END IF

Exemple :

```
10 LET limite = RND(1 TO 10)
30 INPUT "Tapez un nombre' ! nombre
40 IF nombre > limite THEN
50    PRINT "en dehors de limites"
60 ELSE
60   PRINT "dans les limites"
70 END IF
```

Commentaire **:**

Dans les trois formes de l'instruction **IF** le **THEN** est optionnel (facultatif). Dans la forme courte, il peut être remplacé par un double point pour distinguer la fin du **IF** et le début de l'instruction suivante. Dans les deux formes longues, il peut être supprimé entièrement.

Disposition :

Les instructions **IF** peuvent être imbriquées autant de fois que l'utilisateur le désire (dans les limites de la capacité de la mémoire). Cependant, une confusion peut apparaître pour déterminer quel **IF** correspond à quels **ELSE, END, IF**. SuperBASIC reliera les instructions **ELSE**, etc... à l'instruction **IF** la plus proche, par exemple :

```
10 IF a = b then
20   IF c = d then
30      PRINT "erreur"
40   ELSE
50      PRINT "pas d'erreur"
60   END IF
70 END IF
```

Le **ELSE** correspond au second **IF**.

INK

Fenêtre

Fixe la couleur de l'encre, c'est à dire, la couleur dans laquelle les caractères sont affichés. **INK** donne son effet sur la fenêtre reliée au canal par défaut ou au canal donné.

Syntaxe : INK*[canal,]couleur*

Exemples :

i. `INK 5`
ii. `INK 6,2`
iii. `INK #2,255`

INKEY$

Entrées/Sorties

INKEY$ est une fonction qui renvoie un seul caractère â partir du canal par défaut ou du canal donné.

Un délai optionnel d'attente du caractère peut être spécifié. Ce délai peut varier de zéro à l'infini. Si aucun paramètre n'est précisé il n'y a pas d'attente sur la fonction **INKEY$.**

Syntaxe : INKEY$[canal[,temps)]

avec temps = nombre d'unités de temps
temps = -1 attente infinie
temps = 0 retour immédiat

Exemples :

i. `PRINT INKEY$` entrée à partir du canal par défaut
ii. `PRINT INKEY$(#4)` entrée à partir du canal 4
iii. `PRINT INKEY$(50)` attente de 50 unités de temps et retour
iv. `PRINT INKEY$(0)` retour immédiat après balayage du clavier

INPUT

Entrées/Sorties

INPUT permet de passer des données à un programme SuperBASIC â partir du clavier du QL. Le SuperBASIC arrête le programme jusqu'à ce que la quantité d'informations demandée ait été tapée au clavier. Le programme continue alors sur l'instruction suivante. Chaque élément de données doit être terminé par l'appui de la touche **ENTREE**.

Un texte optionnel peut être précisé, si le texte contient une celle-ci doit faire partie d'une expression. Le plus simple pour est de mettre le texte entre doubles cotes (voir exemple).

INPUT permet de rentrer des données à partir du canal par du canal donné.

Si une donnée doit être rentrée à partir d'un canal particulier, de la fenêtre connecté par ce canal apparaît et clignote.

Syntaxe : *séparateur (voir PRINT)*
texte:= [canal] expression séparateur

INPUT *[texte] (canal] [variable]*[variable]*

Exemples :

i.
```
INPUT ("dernière proposition" & proposition &  "nouvelle
proposition ?")!proposition
```
ii.
```
INPUT "quelle est votre proposition ?";proposition
```
iii.
```
10 INPUT "taille du tableau ?":taille
20 DIM tableau(taille-1)
30 FOR élément = 0 TO taille-1
40   INPUT ("données pour l'élément" & élément)!
     tableau(élément)
50 END FOR élément
60 PRINT tableau
```

INSTR

Opérateur

INSTR est un opérateur qui permet de déterminer si une sous-chaîne est contenue dans la chaîne donnée. Si la chaîne est trouvée alors la position de la sous--chaîne est renvoyée. Si la sous-chaîne n'est pas trouvée alors **INSTR** renvoie zéro.

Zéro peut être interprété comme faux. Par exemple, la sous-chaîne n'est pas contenue dans la chaîne donnée. Une valeur différente de zéro, la position dans la chaîne peut être interprétée comme vraie. Par exemple, la sous-chaîne est contenue dans la chaîne donnée.

Syntaxe : *expression INSTR expression*

Exemples :

i. `PRINT "a" INSTR "cat"` (donne 2)
ii. `PRINT "CAT" INSTR "concatenation"` (donne 4)
iii. `PRINT "X" INSTR "oeufs"` (donne 0)

INT

Fonctions mathématiques

INT renvoie la partie entière d'une expression en virgule flottante.

Syntaxe : *INT(expression_numérique)*

Exemples :

i. `PRINT INT(X)`
ii. `PRINT INT (3.141592654/2)`

KEYROW

Le numéro de rangée et la position à l'intérieur de la rangée pour chaque touche du clavier sont donnés dans la table ci-dessous. L'exemple ci-dessous affiche la valeur de la touche et le numéro de rangée pour chaque touche du clavier.

A partir de l'exemple et de la table, on peut voir que **KEYROW** renvoie la position correcte que s'il n'y a pas plusieurs touches enfoncées simultanément. Il ne doit pas y avoir plus de deux touches enfoncées dans chaque rangée et colonne. Par exemple, il peut y avoir deux touches dans chaque rangée ou dans chaque colonne mais il ne peut pas y avoir deux touches dans une colonne et une rangée.

Syntaxe : *rangée:= expression_numérique (de 0 à 7)*
KEYR0W(rangée)

Exemple :

```
10 REMark lancez ce programme et appuyez sur des touches.
20 REPeat boucle
30   CURSOR 0,0
40   FOR rangée=0 TO 7
50     PRINT rangée!!! KEYROW (rangée) ; " "
60   END FOR rangée
70 END REPeat boucle
```

A titre d'exemple, voici le tableau du clavier QWERTY. Le clavier français, AZERTY est légèrement différent

COLONNE → / ROW ↓	1	2	4	8	16	32	64	128
7	SHIFT	CTRL	ALT	X	V	/	N	.
6	8	2	6	Q	E	0	T	U
5	9	W	I	TAB	R	-	Y	O
4	L	3	H	1	A	P	D	J
3	I	CAPS LOCK	K	S	F	=	G	;
2	|	Z	.	C	B	£	M	~
1	ENTER	←	↑	ESC	→	\	SPACE	↓
0	F4	F1	5	F2	F3	F5	4	7

LBYTES

Périphériques

LBYTES charge un fichier de données dans la mémoire à partir d'une adresse donnée.

Syntaxe : adresse_début := expression_numérique
LBYTES *périphérique, adresse_début*

Exemples :

i. `LBYTES mdv1_écran,131072` (charge une image écran)
ii. `LBYTES MDV1_programme, adresse_début`
(charge un programme â l'adresse donnée)

LEN

Chaînes

LEN est une fonction qui renvoie la longueur de la chaîne de caractères donnée.

Syntaxe : *LEN(chaine)*

Exemples :

i. `LEN("LEN donne la longueur de cette chaîne")` (donne 37)
ii. `LEN(adresse$)`

LET

LET est une instruction qui permet de donner une valeur à une variable. L'utilisation du mot-clé **LET** est optionnel. L'assignation peut être faite pour des chaînes de caractères et des variables numériques. SuperBASIC convertit automatiquement les types de données dans la forme correcte quand cela est possible.

Syntaxe : [LET]*variable = expression*

Exemples :

i. `LET a = 1 + 2`
ii. `LET a$ = "12345"`
iii. `LET a$ = 6784`
iv. `b$ = données_de_tests`

LINE

LINE_R

Graphiques

LINE permet de tracer une ligne droite entre deux points dans la fenêtre reliée au canal par défaut ou au canal donné. Les extrémités de la ligne sont données en utilisant le système de coordonnées graphiques. Plusieurs lignes peuvent être tracées avec une seule commande **LINE**.

Normalement on doit fournir les coordonnées des extrémités de la ligne, soit en coordonnées absolues (relativement à l'origine des graphiques) ou en coordonnées relatives (relativement au curseur des graphiques). Si le premier point n'est pas donné, alors la ligne est tracée à partir de la position du curseur des graphiques jusqu'au point donné. Si le second point n'est pas donné, alors le curseur des graphiques est déplacé mais aucune ligne n'est tracée.

LINE trace toujours en coordonnées absolues (relativement à l'origine des graphiques). Tandis que **LINE_R** trace toujours relativement au curseur des graphiques.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique
point:= x,y
paramètres:= | *point TO point *[TO point]** 1
| *TO point*[TO point]** 2
| *point* 3

avec : 1. qui trace la ligne d'un point à l'autre.
2. qui trace la ligne à partir du dernier point jusqu'au point donné.
3. positionne le curseur graphique sur le point sans tracer.

LINE[canal,],paramètres[paramètres]**
LINE_R[canal,]paramètres[paramètres]**

Si LINE_R est utilisé, alors les points sont toujours relatifs au curseur des graphiques.

Exemple :

i. `LINE 0,0 TO 0,50 TO 50,0 TO 50,50 TO 0,0` (carré)
ii. `LINE TO 0.75,0.5` (ligne)
iii. `LINE 25,25` (déplace le curseur des graphiques)

LIST

Basic

LIST permet d'obtenir la liste d'une ligne ou d'un groupe de lignes d'un programme SuperBASIC sur le canal par défaut ou donné.

Syntaxe : *ligne : =* | *numéro_ligne TO numéro_ligne* 1
| *numéro_ligne TO* 2
| *TO numéro_ligne* 3
| *numéro_ligne* 4

LIST *[canal,] ligne*[,ligne]** 5

Avec : 1. liste les lignes comprises entre ces limites
2. liste à partir de cette ligne jusqu'à la fin
3. liste jusqu'à cette ligne
4. liste une ligne
5. liste tout le programme

Exemples :

i. `LIST`
ii. `LIST 10 TO 300` (liste les lignes de 10 à 300)
iii. `LIST 12, 20, 50` (liste les lignes 12, 20 et 50)

Commentaire :

Si **LIST** est dirigée sur un canal ouvert en imprimante, **LIST** permet d'éditer le programme.

LOAD

Périphériques

LOAD charge un programme SuperBASIC à partir d'un périphérique du QL. **LOAD** fait automatiquement **NEW** avant de charger un autre programme. Si pendant le chargement la syntaxe d'une ligne SuperBASIC n'est pas correcte, le mot **MISTAKE** (faute) est inséré entre le numéro de la ligne et le corps de la ligne. A l'exécution une ligne de cette sorte génère une erreur.

Syntaxe : LOAD *périphérique*

Exemples :

i. `LOAD "MDV1_EASEL"`
ii. `LOAD mdvl_ARCHIVE`
iii. `LOAD neti_3`
iv. `LOAD ser1_e`

LOCAL

Fonctions et procédures

LOCAL permet de définir des variables à l'intérieur d'une fonction ou d'une procédure. Les variables locales n'existent qu'à l'intérieur de la fonction ou de la procédure dans laquelle elles sont définies. Elles sont perdues quand la fonction ou la procédure se terminent. Les variables locales sont indépendantes de variables de même nom, situées à l'extérieur de la fonction ou de la procédure. Des tableaux peuvent être définis pour être locaux en les dimensionnant à l'intérieur de l'instruction **LOCAL**.

Les instructions **LOCAL** doivent précéder la première instruction exécutable de la fonction ou de la procédure.

Syntaxe : LOCAL *identifiant *[,identifiant]**

Exemples :

i. `LOCAL a, b, C (10,10)`
ii. `LOCAL donnée_temporaire`

Commentaire :

Définir des variables locales avec **LOCAL**, permet d'utiliser les mêmes noms de variables à l'intérieur de la fonction ou de la procédure, sans risquer de modifier les variables à l'extérieur de la fonction ou de la procédure.

LN

LOG 10

Fonctions mathématiques

LN renvoie le logarithme néperien du paramètre donné.

LOG 10 renvoie le logarithme décimal.

Il n'y a pas de limite supérieure sur la valeur du paramètre autre que le nombre maximum que l'ordinateur peut stocker.

Syntaxe : LOG1O*(expression_numérique)* (plus grande que zéro)
LNG*(expression_numérique)* (plus grande que zéro)

Exemples :

i. `PRINT LOG10(20)`
ii. `PRINT LN(PI)`

LRUN

Super BASIC

LRUN charge et lance un programme SuperBASIC à partir du périphérique donné. Tout programme SuperBASIC stocké en mémoire sera effacé par **LRUN**.

Si une ligne du programme en cours de chargement présente une syntaxe incorrecte, le mot **MISTAKE** est inséré entre le numéro et le corps de la ligne .A l'exécution une telle ligne génère une erreur.

Syntaxe : LRUN *périphérique*

Exemples :

i. `LRUN MDV1_QUILL`
ii. `LRUN MDV1_jeux`

MRUN

Basic

MRUN permet de fusionner le programme SuperBASIC nommé avec celui en mémoire à cet instant. Si **MRUN** est utilisé en commande directe, le déroulement du programme commence au début.

Si **MRUN** est utilisé à l'intérieur d'un programme, l'exécution continue sur la ligne qui suit **MRUN**. Si une ligne du programme fusionné est incorrecte le mot **MISTAKE** est inséré entre le numéro et le corps de la ligne. A l'exécution cette ligne génère une erreur.

Syntaxe : MRUN *périphérique*

Exemples :

i. `MRUN MDV1_segments_programme`
ii. `MRUN MDV2_nouvelles_données`

MERGE

Périphériques

MERGE charge un fichier â partir d'un périphérique donné et l'interprète comme un programme SuperBASIC. Si le nouveau fichier contient un numéro de ligne qui n'apparait pas dans les programmes, cette ligne s'y ajoute. Si le nouveau fichier contient une ligne de remplacement alors la ligne est remplacée. Les autres lignes de l'ancien programme ne sont pas modifiées.

Si une ligne présente une syntaxe incorrecte pendant **MERGE**, le mot **MISTAKE** est inséré entre le numéro et le corps de la ligne. A l'exécution cette ligne génère une erreur.

Syntaxe : MERGE *périphérique*

Exemples :

i. `MERGE MDV1_chaine`
ii. `MERGE MDV1_données`

MOD

Opérateur

MOD est un opérateur qui donne le module ou le reste d'une division de deux entiers.

Syntaxe : *expression_numérique MOD expression_numérique*

Exemples :

i. `PRINT 5 MOD 2` (donne 1)
ii. `PRINT 5 MOD 3` (donne 2)

MODE

Ecran

MODE permet de choisir la résolution de l'écran et le nombre de couleurs qu'il peut visualiser. **MODE** efface toutes les fenêtres actuellement à l'écran mais conservent leurs positions et leurs tailles. Lorsque l'on peut passer en mode basse résolution (8 couleurs), la taille minimum des caractères est 2,0.

Syntaxe : MODE *expression_numérique*

où 8 ou 256 pour la basse résolution
4 ou 512 pour la haute résolution

Exemples :

i. `MODE 256`
ii. `MODE 4`

MOVE

Graphiques de tortue

MOVE permet de déplacer la tortue à une certaine distance de la position actuelle. Le facteur d'échelle des graphiques est utilisé pour déterminer le déplacement. Si on donne une distance négative, le déplacement se fait en arrière.

La tortue est déplacée dans la fenêtre reliée au canal par défaut ou au canal donné.

Syntaxe : *Distance := expression_numérique*

MOVE*[canal,]distance*

Exemples :

iii. `MOVE #2,20` déplace la tortue sur le canal 2 de 20 unités en avant
iv. `MOVE -50` déplace la tortue sur le canal par défaut de 50 unités en arrière

NET

Périphériques

NET permet de donner un numéro de QL sur le réseau local. Si ce numéro n'est pas donné, le numéro de station est égal à 1.

Syntaxe : *station:= expression_numérique (de 0 à 64)*

NET *station*

Exemples :

i. `NET 63`
ii. `NET 0`

Commentaire :

Il ne doit pas y avoir de numéro identique sur le réseau local.

NEW

Basic

NEW permet d'effacer les vieux programmes, les variables et les canaux autres que 0,1 et 2.

Syntaxe : NEW

Exemple : NEW

NEXT

Répétition

NEXT est utilisé pour contrôler les constructions **REPeat** et **FOR**.

Syntaxe : NEXT *identifiant*

L'identifiant doit correspondre à celui de la boucle que **NEXT** contrôle.

Exemples :

i.
```
10 REMark cette boucle doit se répéter indéfiniment.
20 REPeat boucle_infinie
30   PRINT "boucle en cours"
40 NEXT boucle_infinie
```

ii.
```
10 FOR indice = 1 TO limite.
20   INPUT "donnée ?", tableau (indice)
30 NEXT indice
```

iii.
```
10 REPeat impair
20   LET nombre = RND (1,100)
30   IF nombre/2= INT(nombre/2) THEN NEXT impair
40   PRINT nombre; "est impair"
50 END REPeat impair
```

Dans **REPeat** : si **NEXT** est utilisé dans une construction **REPeat**-**END**, **REPeat** cela obligera le traitement de se poursuivre à l'instruction suivant l'instruction **REPeat** correspondante.

Dans **FOR** : L'instruction **NEXT** peut être utilisée pour répéter la boucle **FOR** en faisant passer la variable de contrôle à sa valeur suivante. Si la fin de la suite des valeurs est atteinte, ou si la limite de la variable de contrôle a été dépassée, le traitement continuera à l'instruction suivant le **NEXT** ; sinon, le traitement continuera à l'instruction après le **FOR**.

ON…GOTO

ON…GOSUB

Compatibilité

Pour être compatible avec les autres BASICs, SuperBASIC prévoit les instructions ON GOTO et ON GOSUB. Ces instructions permettent à une variable de transférer le traitement à un numéro de ligne parmi plusieurs numéros de ligne.

Syntaxe *ON variable GOTO expression*[,expression]**

ON variable GOSUB expression[,expression]**

Exemples :

i.
```
ON x GOTO 10, 20, 30, 40
```

ii.
```
ON variable_choisie GOSUB 1000, 2000, 3000, 4000
```

Commentaire :

SELect peut être utilisé pour remplacer ces deux commandes du BASIC.

OPEN

OPEN_IN

OPEN_NEW

Périphériques

OPEN permet à l'utilisateur de faire un lien entre un canal logique et un périphérique physique du QL pour les entrées/sorties.

Si le canal est un microdrive, alors le fichier peut être nouveau ou déjà existant. Dans ce cas **OPEN_IN** permet d'ouvrir un fichier déjà existant pour la lecture et **OPEN_NEW** permet de créer un nouveau fichier Microdrive pour l'écriture.

Syntaxe : *canal:=#expression_numérique (de 0 à 16)*
OPEN *canal,périphérique*

Exemples :

i. `OPEN #5,f_nom$`

ii. `OPEN_IN #9, "MDV1_nom_fichier"`
(ouvre le fichier MDV1_nom_fichier)

iii. `OPEN_NEW #7, MDV1_fichier_données`
(ouvre le fichier MDV1_fichier_données)

iv. `OPEN #6, CON_10x20a20x20_32`
(ouvre le canal 6 sur le périphérique console créant une fenêtre de 10x20 à la position 20,20 avec un buffer clavier de 32 bytes)

OVER

Ecran

OVER détermine le type de surimpression requis. Le type choisi reste effectif, jusqu'à la prochaine utilisation de **OVER**.

Syntaxe : *paramètre:= expression_numérique (entre 0 et 1)*
OVER *[canal,]paramètre*

où :
paramètre = 0 entraîne INK sur STRIP
paramètre = 1 entraîne INK sur STRIP transparent
paramètre = -1 entraîne INK sur contenu précédent de l'écran.

Exemples :

i. `OVER 1`

ii.
```
10 REMark écriture ombragée
20 PAPER 7 :INK 0 : OVER 1
30 CSIZE 3,1
40 FOR i = 0 TO 10
50    CURSOR
60    IF i = 10 THEN INK 2
70      PRINT "ombre"
80 END FOR i
```

PAN

Fenêtre

La totalité de la fenêtre sera décalée d'un nombre déterminé de pixels à gauche ou à droite. Le "papier" (fond d'écran) défilera pour remplir les surfaces inutilisées. Un deuxième paramètre optionnel peut être précisé qui permettra à seulement une partie de l'écran d'être remplie.

Syntaxe : *distance:= expression_numérique*
partie:= expression_numérique

PAN *[canal,]distance[,partie]*

où partie = 0 - tout l'écran (par défaut)
partie = 3 - toute la ligne du curseur
partie = 4 - à droite du curseur, position du curseur comprise

Si l'expression est positive, alors le contenu de l'écran est décalé sur la droite.

Exemples :

i. `PAN 50` (décalage à droite de 50 pixels)
ii. `PAN-100` (décalage à gauche de 100 pixels)
iii. `PAN#2,50`
iv. `PAN 50,3` (décalage à droite de toute la ligne du curseur de 50 pixels)

Attention :

Si l'instruction **STRIP** a été utilisée précédemment et si l'on désire la même trame, la couleur de **PAN** doit être un multiple de 2 points (pixels).

PAPER

Fenêtre

PAPER face à une nouvelle couleur de fond (c'est la couleur qui sera utilisée par **CLS**, **PAN**, **SCROLL**, etc...). La couleur sélectionnée reste jusqu'à la prochaine utilisation de l'instruction **PAPER**.

PAPER fonctionne également la couleur **STRIP**. **PAPER** change la couleur du fond de la fenêtre reliée au canal par défaut ou au canal donné.

Syntaxe : PAPER*[canal,]couleur*

Exemples :

i. `PAPER #3,7` (papier blanc sur canal 3)
ii. `PAPER 7,2` (rayures blanches et rouges)
iii. `PAPER 255` (rayures blanches et noires)
iv.
```
10 REMark montrons les couleurs et les rayures
20 FOR couleur = 0 TO 7
30   FOR contraste = 0 TO 7
40     FOR rayure = 0 TO 7
50       PAPER couleur, contraste, rayure
60     END FOR rayure
70   END FOR contraste
80 END FOR couleur
```
(pas utilisable sur T.V.)

PAUSE

Tâches multiples

PAUSE met un programme en attente pendant une période de temps donnée. Une entrée au clavier arrête l'effet de l'instruction **PAUSE** et redémarre l'exécution du programme. Si aucun temps n'est donné en paramètres, alors le programme attend indéfiniment.

Syntaxe : *délai := expression_numérique*

PAUSE*[délai]*

Exemples :

i. `PAUSE 50` (attente 1 seconde)
ii. `PAUSE 500` (attente 10 secondes)

PEEK

PEEK_W

PEEK_L

Basic

PEEK est une fonction qui utilise le contenu d'un emplacement mémoire déterminé.

PEEK se présente sous trois formes qui permettent successivement d'accéder à un octet (8 bits), un mot (16 bits), un mot long (32 bits)

Syntaxe : *adresse:= expression_numérique*

PEEK*(adresse)*	accès à l'octet 8 bits
PEEK_W*(adresse)*	accès au mot 16 bits
PEEK_L*(adresse)*	accès au mot long 32 bits

Exemples :

i.	`PRINT PEEK(12245)`	affiche le contenu de la position 12245
ii.	`PRINT peek_W(12)`	affiche le contenu des positions 12 et 13
iii.	`PRINT PEEK_L(10)`	affiche le contenu de la position 10,11,12,13

Pour l'accès au mot et au mot long l'adresse doit être paire.

PENUP

PENDOWN

Graphiques de Tortue

Permet de mettre en action le "crayon" dans les graphiques. Si le "crayon" est en haut (up), alors rien ne s'inscrit. Si le "crayon" est en bas (down), alors les lignes sont tracées au fur et â mesure que la tortue se déplace le long de l'écran.

La ligne est tracée dans la fenêtre attachée au canal par défaut ou au canal donné.

La ligne est tracée dans la couleur courante pour le canal sur lequel la sortie est dirigée.

Syntaxe : PENUP*[canal]*
PENDOWN*[canal]*

Exemples :

i. `PENUP`
ii. `PENDOWN`

PI

Fonctions mathématiques

PI est une fonction qui renvoie la valeur de π

Syntaxe : *PI*

Exemple : PRINT PI

POINT

POINT_R

Graphiques

POINT permet de mettre un point à la position donnée dans la fenêtre reliée au canal par défaut ou au canal donné. Le point est tracé en utilisant le système de coordonnées graphiques relativement à l'origine des graphiques.

Si **POINT_R** est utilisé, alors tous les points se positionnent relativement à la position du curseur des graphiques, donc relativement au point précédent.

Des points multiples peuvent être positionnés avec un seul POINT.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique
paramètres:= x,y

POINT [canal,] paramètre* [,paramètres]*

Exemples :

i. `POINT 256,128` (met un point à 256,128)
ii. `POINT x,x*x` (met un point à x,x*x)
iii.
```
10 REPeat exemple
20   INK REND(255)
30   POINT RND(100),RND(200)
40 END REPeat exemple
```

POKE

POKE_W

POKE_L

Basic

POKE permet de changer le contenu d'une position mémoire. Pour l'accès aux mots et aux mots longs, les adresses données doivent être des adresses paires.

POKE se présente sous trois formes, chacune accède respectivement à un octet (8 bits), un mot (16 bits), un mot long (32 bits)

Syntaxe : *Adresse := expression_numérique*
donnée := expression_numérique

POKE *adresse, donnée*	(accès à l'octet)
POKE_W *adresse, donnée*	(accès au mot)
POKE_L *adresse, donnée*	(accès au mot long)

Exemples :

i. `POKE 12345,0` (met l'octet 12345 à 0)
ii. `POKE_W 12345,32768` (met le mot 12345 à 32768)
iii. `POKE_L 12345,131072` (met le mot long 12345 â 131072)

Attention :

Faire des POKE dans les zones mémoires utilisées par le QDOS peut provoquer des blocages et des pertes de données.

PRINT

Fichier

Permet d'envoyer des données sur le canal par défaut ou le canal donné. L'utilisation normale de **PRINT** est d'envoyer des données à l'écran du QL.

Syntaxe : *séparateur* | *!*
| *,*
| \
| *;*
| *TO exp_num*

Texte := | *expression*
| *canal*
| *séparateur*

PRINT *[texte]*

Exemples :

i. `PRINT "Bonjour tout le monde"`
affiche ce texte sur le canal par défaut : l'écran.

ii. `PRINT #5, "données",1,2,3,4`
envoie les données fournies sur le canal 5 qui doit avoir été ouvert avant.

Séparateurs

! Peut être considéré comme un espace intelligent. Son action normale consiste à insérer un espace entre les zones affichées à l'écran. Si la zone ne tient pas sur la ligne courante, un retour à la ligne est fait. Si la position d'impression actuelle est au début d'une ligne, l'espace n'est pas ajouté.
! produit son effet sur la zone suivante et doit donc être placé avant le nom de la zone à imprimer.
Un point-virgule ou un point d'exclamation doit être mis à la fin de la ligne si l'espacement doit se continuer sur une série d'instructions **PRINT**.

, Séparateur normal, le SuperBASIC positionne une tabulation toutes les 8 colonnes.

**** force le passage à la ligne suivante.

; laisse la position d'édition immédiatement après la dernière zone éditée. Si aucune action n'est prise le prochain **PRINT** se fait sur cette même ligne.

RAD

Fonctions mathématiques

RAD est une fonction qui convertir un angle donné en degrés ou un angle en radians.

Syntaxe : RAD*(expression_numérique)*

Exemple : PRINT RAD(180) (donne 3.141593)

RANDOMISE

Fonctions mathématiques

RANDOMISE permet.de donner une base au générateur de nombres aléatoires. Si un paramètre est donné, il est pris comme nouvelle base. Si aucun paramètre n'est donné, le générateur prend comme base une information interne.

Syntaxe : RANDOMISE *[expression_numérique]*

Exemples :

i. `RANDOMISE` (redonne la base par défaut)

ii. `RANDOMISE 3.2235` (donne la base 3.2235 au générateur de nombres aléatoires)

RECOL

Ecran

RECOL permet de changer la couleur de chaque point de l'écran en fonction de la couleur actuelle. Chaque paramètre donne dans l'ordre la nouvelle couleur de chaque point en fonction de sa couleur précédente. Par exemple, le premier paramètre donne la couleur pour chaque point bleu, le second paramètre donne la- nouvelle couleur de chaque point rouge, etc...

La couleur doit être pure, c'est-à-dire comprise entre 0 et 7.

Syntaxe : c1:= nouvelle couleur pour le bleu
c2:= nouvelle couleur pour le rouge
c3:= nouvelle couleur pour le magenta
c4:= nouvelle couleur pour le vert
c5:= nouvelle couleur pour le cyan
c6:= nouvelle couleur pour le jaune
c7:= nouvelle couleur pour le blanc
c8:= nouvelle couleur pour le noir

RECOL [canal] cl,c2,c3,c4,c5,c6,c7,c8

Exemple : RECOL 2,3,4,5,6,7,1,0

recolore le bleu en rouge, le rouge en magenta, le magenta en vert, etc...

REMARK

Basic

REMark permet d'insérer du texte d'explications à l'intérieur d'un programme. Le reste de la ligne est ignoré par le SuperBASIC.

Syntaxe : *REMark texte*

Exemple : REMark Ceci est un commentaire

Commentaire :

REMark doit être utilisé pour documenter les programmes, ce qui facilite leur lecture.

RENUM

Basic

RENUM permet de changer les numéros de ligne d'un groupe ou d'une série de groupes de lignes SuperBASIC. Si aucun paramètre n'est donné, la renumérotation du programme commence à la ligne 100 avec un pas de 10. Si le numéro de ligne de départ est donné, la renumérotation démarre à partir de cette ligne. Si un intervalle est donné la renumérotation se fait par cet intervalle.

Si un système **GOTO** ou **GOSUB** contient une expression qui est un numéro de ligne, alors celle-ci est renumérotée.

Syntaxe : *ligne début:= expression_numérique*
ligne_fin:= expression_numérique
numéro_début:= expression_numérique
pas:= expression_numérique

RENUM (début_ligne [TO ligne_fin] ;] [numéro_début] [,pas]

Exemples :

i. `RENUM` (renumérote le programme en commençant à la ligne 100 avec un pas de 10)
ii. `RENUM 100;200` (renumérote de 100 à 200 avec un pas de 10)

Attention :

On ne doit pas essayer d'utiliser **RENUM** pour renuméroter des lignes de programmes qui ne sont pas dans l'ordre. **RENUM** ne renumérote pas les instructions **RESTORE**.

REPEAT

END REPEAT

Répétition

REPeat permet la construction de boucles répétitives. **REPeat** doit être utilisé avec **EXIT** pour un effet maximum. **REPeat** existe sous forme courte et sous forme longue.

COURTE : Le mot-clé **REPeat** et l'identifiant de la boucle sont suivis sur la même ligne logique par un double point et une séquence d'instructions SuperBASIC. **EXIT** rétablira le traitement normal à la ligne logique suivante.

Syntaxe : *REPeat identifiant : instructions*

Exemple : `REPeat attente : IF INKEY$ < > "" then EXIT attente`
(boucle d'attente d'une touche de clavier)

LONGUE : Le mot-clé **REPeat** et l'identifiant de la boucle sont les seules instructions sur la ligne logique. Les lignes suivantes contiennent une suite d'instructions de SuperBASIC terminées par une instruction **END REPeat**. Les instructions entre le **REPeat** et le **END REPeat** sont traitées de manière répétitive par SuperBASIC

Syntaxe : *REPeat identifiant*
instructions
END REPeat identifiant

Exemple :

```
40 ET nombre = RND(1 TO 50)
50 REPeat réponse
60   INPUT "quelle est votre réponse",réponse
70   IF réponse = nombre THEN
80     PRINT "vous avez deviné"
90     EXIT réponse
100  ELSE
120    PRINT "mauvaise proposition"
130  END IF
140 END REPeat réponse
```

Commentaire : Normalement une instruction au moins de la boucle **REPeat** sera une instruction **EXIT**

RESPR

QDOS

Si vous désirez charger un programme en langage machine, vous devez demander au QDOS de vous réserver une zone à cet effet. C'est lui qui, en fonction de la taille de votre programme, décidera de l'adresse d'implantation de celui-ci (votre programme doit donc être relogeable).

RESPR est une fonction qui retourne l'adresse d'implantation décidée par le QDOS pour le nombre d'octets demandé.

Syntaxe : *ESPACE = exp_num*

RESPR *(ESPACE)*

Exemple : `PRINT RESPR(1024)` affiche l'adresse d'implantation d'un bloc de 1024 octets

RETURN

Fonctions et Procédures

RETURN est utilisé pour obliger une fonction ou une procédure à s'arrêter et rétablir le traitement à l'instruction après l'appel de la procédure ou de la fonction. Utilisée dans une fonction, l'instruction **RETURN** permet de restituer la valeur des fonctions.

Syntaxe : *RETURN [expression]*

Exemples :

i.
```
10 DEFine FuNction sinh (x )
20   IF ABS(x)<précision_autorisée THEN
30     RETURN EXP(x-LN 2)
40   ELSE
50     RETurn (exp(x) - exp(-x))/2
60   END IF
70 END DEFine
```

ii.
```
10 DEFine PROCedure mise_en_garde (n)
20 REM impression d'un message de mise en garde
30 IF drapeau_de_mise_en_garde THEN
40   PRINT 'ATTENTION:";
50   SELect ON n
60      ON n = 1
70        PRINT "microdrive plein"
80      ON n = 2
90        PRINT "zone de données pleine"
100     ON n = REMAINDER
110       PRINT "erreur dans le programme"
120     END SELect
130  ELSE
140  RETURN
150  END IF
160 END DEFine
```

Commentaire :

Avoir un **RETURN** dans une procédure n'est pas obligatoire. Si le traitement atteint le **END DEF**ine d'une procédure, alors la procédure se terminera automatiquement.

RND

SuperBASIC

RND génère un nombre aléatoire. Jusqu'à deux paramètres peuvent être désignés pour **RND**.

Si aucun paramètre n'est précisé, **RND** restitue un nombre en virgule flottante pseudo-aléatoire compris entre 0 et 1.

Si un seul paramètre est spécifié, alors **RND** restitue un entier compris entre 0 et le nombre spécifié.

Si deux paramètres sont repris, alors **RND** restitue un entier compris entre ces deux paramètres.

Syntaxe : *RND ([expression_numérique] [TO expression_numérique])*

Exemples :

i.
```
PRINT RND
```
ii.
```
PRINT RND (10 TO 20)
```
iii.
```
PRINT RND (1 TO 6)
```
iv.
```
PRINT RND (10)
```

RUN

SuperBASIC

RUN démarre l'exécution d'un programme SuperBASIC. Si un numéro de ligne est indiqué dans la commande **RUN**, le programme démarrera à cet endroit, sinon le programme démarrera au numéro de ligne le plus petit.

Syntaxe : *RUN [expression_numérique]*

Exemples :

i. `RUN` depuis le début
ii. `RUN 10` depuis la ligne 10
iii. `RUN2*20` depuis la ligne 40

Commentaire : Bien que RUN soit utilisable dans un programme, son usage habituel est de démarrer un programme en commande directe.

SAVE

Périphériques

SAVE permet de sauvegarder un programme SuperBASIC soit un périphérique QL.

Syntaxe : ligne := I *expression_numérique TO expression_numérique* 1
| *expression_numérique TO* 2
| *TO expression_numérique* 3
I expression_numérique 4
| 5

SAVE *périphérique ligne *[,ligne]**

Avec :

1 sauvegarde à partir de la ligne jusqu'à la ligne
2 sauvegarde à partir de la ligne jusqu'à la fin
3 sauvegarde à partir du début jusqu'à la ligne
4 sauvegarde la ligne
5 sauvegarde tout le programme

Exemples :

i. `SAVE mdvl_programme; 20 TO 70`
sauvegarde les lignes de 20 à 70 dans MDV1_programme
ii. `SAVE mdv2_programme_test;10,20,40`
sauvegarde les lignes 10,20 et 40 dans MDV2_programme-test
iii. `SAVE neto_3 _`
sauvegarde le programme entier sur le réseau
iv. `SAVE ser1`
sauvegarde le programme complet sur la sortie série RS232C

SBYTES

Périphériques

SBYTES permet de sauvegarder des zones de la mémoire QL sur un périphérique QL.

Syntaxe : *Adresse_ début:= expression_numérique*
longueur := expression_numérique

SBYTES *périphérique,adresse_début, longueur*

Exemples :

i. `SBYTES mdvl_écran_données,131072,32768`
(sauvegarde le contenu de l'écran de 32 K sur mdvl écran données)

ii. `SBYTES mdvl_programme_test,50000,10000`
(sauvegarde la mémoire à partir de 50000 sur 10000 octets dans mdv1_programme test)

iii. `SBYTES net3,32768,32768`
(sauvegarde la mémoire à 32768 sur 32768 octets sur le réseau local)

iv. `SBYTES ser1,0,32768`
(sauvegarde la mémoire de 0 à 32768 sur le canal série 1)

SDATE

Horloge

SDATE permet de mettre l'horloge du QL à l'heure.

Syntaxe : *année:= expression_numérique*
mois:= expression_numérique
jour:= expression_numérique
heures: =expression_numérique
minutes:= expression_numérique
secondes:= expression_numérique

SDATE *année, mois, jour, heures, minutes, secondes*

Exemples :

i. `SDATE 1984,4,0,0,0`
ii. `SDATE 1984,1,12,9,30,0`
iii. `SDATE 1984,21,3,0,0,0`

SIN

Fonctions mathématiques

SIN calcule le sinus du paramètre donné.

Syntaxe : *angle:= expression_numérique* (de -60000 (à 60000 radians)

SIN*(angle)*

Exemples :

i. `PRINT SIN(3)`
ii. `PRINT SIN(PI/2)`

SCALE

Fenêtres

SCALE permet de modifier le facteur d'échelle utilisé dans les procédures graphiques. Une échelle de "x" implique une ligne verticale de longueur "x" qui remplit l'arc vertical de la fenêtre dans laquelle la figure est dessinée. Par défaut, l'échelle est de 100.

SCALE permet également de préciser l'origine des coordonnées du système. Ceci permet de déplacer la fenêtre utilisée pour les graphiques de façon à disposer de plus de place.

Syntaxe : *x:= expression_numérique*
y:= expression_numérique

origine := x,y
échelle:= expression_numérique

SCALE*[canal,]échelle,origine*

Exemples :

i. `SCALE 0.5,0.1,0.1` (échelle=0.5, origine à 0.1,0.1)
ii. `SCALE 10,0,0` (échelle = 10 , origine =0,0)
iii. `SCALE 100,50,50` (échelle = 100, origine = 50,50)

SCROLL

Graphiques

SCROLL fait défiler la fenêtre vers le bas ou vers le haut. Le papier se déroule en bas et en haut de l'écran pour remplir l'espace disponible.

Un troisième paramètre optionnel peut être défini pour ne faire défiler qu'une partie de l'écran.

Syntaxe : *partie := expression_numérique*
Avec : *partie = 0* tout l'écran (par défaut)
partie = 1 le haut sauf la ligne du curseur
partie = 3 le bas sauf la ligne du curseur

SCROLL *[canal,] expression_numérique [,partie]*

Si l'expression est positive, alors l'écran défilera vers le haut.

Exemples :

i. `SCROLL 10` vers le haut de 10 points
ii. `SCROLL - 70` vers le bas de 70 points
iii. `SCROLL -10,2` défile la partie basse de la fenêtre vers le bas de 10 points

SELECT

END SELECT

Conditions

SELect permet d'engendrer différentes actions selon la valeur d'une variable.

Définition : *variable_select := variable numérique*
élément select : | expression
| expression TO expression
liste_d'éléments_sélect := | éléments_sélect*[,élément_sélect]*

Forme longue :

Permet à plusieurs actions d'être choisies selon la valeur d'une variable select. La variable est le dernier élément de la ligne logique. Suit un groupe d'instructions SuperBASIC, conclu par l'instruction **ON** suivante ou par l'instruction **END SELect**. L'instruction **ON REMAINDER** permet de déterminer ce qui fonctionnera si aucune autre condition select n'est satisfaite.

Syntaxe : SELect ON *variable_select*
*[[ON variable_select] = liste_éléments_select
instructions]*
[ON variable_select] = REMAINDER
instructions
END SELect

Exemple :

```
10 LET numéro_d_erreur = RND(1 TO 10)
15 SELect ON numéro_d_erreur
20   ON numéro_d_erreur = 1
30     PRINT "Division par 0"
25     LET numéro_d_erreur = 0
40   ON numéro_d_erreur = 2
50     PRINT "Pas de fichier"
60     LET numéro_d_erreur = 0
65   ON numéro_d_erreur = 3 TO 5
70     PRINT  "Pas de fichier Microdrive"
80     LET numéro_d_erreur = 0
85   ON numéro_d_erreur = REMAINDER
90     PRINT "erreur de type inconnu"
100 END SELect
```

Si la variable select est utilisée dans l'instruction **SELect**, alors elle doit correspondre à la variable select en tête de sélection.

Forme courte

La forme courte de l'instruction **SELect** permet de simples sélections de lignes. Sur la même ligne logique que l'instruction **SELec**, suit une séquence d'instruction SuperBASIC. Si la condition de l'instruction **SELect** est satisfaite alors la séquence est effectuée.

Syntaxe : SELect ON *variable_select = liste_éléments_select: instructions**
*[:instruction]**

Exemples :

i. `SELect ON donnée_testée = 1 TO 10 : PRINT "réponse dans les limites"`
ii. `SELect ON réponse = 0.00001 TO 0.00005 : PRINT "précision convient"`
iii. `SELect ON a=1 TO 10 : PRINT a ! "dedans"`

Commentaire :
La forme courte de l'instruction **SELect** permet de tester les limites d'une variable plus facilement qu'avec une instruction **IF**. Comparez l'exemple 2. avec l'instruction **IF** correspondante.

SEXEC

Tâches multiples

Permet de sauvegarder une zone de la mémoire dans une forme qui permet de le charger et de l'exécuter par la commande **EXEC**.

Les données sauvegardées doivent constituer un programme en code machine.

Syntaxe : *adresse_début := expression_numérique* (début zone)
longueur := expression_numérique (longueur zone)
taille_données := expression_numérique (longueur zone données)

SEXEC *périphérique,adresse_début,longueur,taille_données*

Exemple :

SEXEC mdv1_programme, 262144,3000,500

Commentaire : La documentation QDOS doit être lue avant d'utiliser cette commande.

SQRT

Fonctions mathématiques

SQRT calcule les racines carrées de l'argument. Celui-ci doit être plus grand que zéro.

Syntaxe : SQRT*(expression_numérique)*

Exemples :

i. `PRINT SQRT(3)`
ii. `LET C = SQRT (a^2 + b^2)`

STOP

Basic

STOP arrête l'exécution d'un programme et rend la main à l'interprète de commandes de SuperBASIC.

Syntaxe : STOP

Exemples :

i. `STOP`
ii. `IF n = 100 THEN STOP`

Commentaire : La dernière ligne d'un programme est équivalente à un STOP.

STRIP

Fenêtres

STRIP fixe la couleur de fond dans la fenêtre reliée au canal par défaut ou au canal donné. Cette couleur **STRIP** est utilisée quand **OVER** 1 est sélectionné.

Utiliser **PAPER** fera automatiquement passer la couleur **STRIP** à la nouvelle couleur **PAPER**.

Syntaxe : STRIP *[canal,] couleur*

Exemples :

i. `STRIP 7` met un fond blanc
ii. `STRIP 0,4,2` met un fond tramé noir et vert

Commentaire : L'effet de STRIP est identique à un crayon lumineux.

TAN

Fonctions mathématiques

TAN calcule la tangente de l'angle donné. Celui-ci doit être compris entre -30000 et 30000 radians.

Syntaxe : TAN *(expression_numérique)* (de -30000 à 30000)

Exemples :

i. `TAN (3)`
ii. `TAN (P1/2)`

TRA

Périphériques

Les caractères spéciaux (dont les caractères accentués) sont stockés en mémoire sous la forme de codes > 127 propres au QL. Il est donc nécessaire, si l'on veut communiquer ces caractères à un périphérique par la liaison RS 232, de passer par une table de conversion, afin que celui-ci puisse les interpréter convenablement.

A cet effet, le QL dispose d'une table de conversion permettant de piloter les imprimantes compatibles EPSON.

L'instruction TRA permet d'activer ou de mettre hors service cette conversion.

Forme simple :

Syntaxe : TRA *expr_numérique*

expr_numérique = 1 : mise en service
expr_numérique = 0 : mise hors service

Exemples :

i. `TRA 1`
ii. `TRA 0`

Forme supérieure :

L'expression TRA permet en outre de créer votre propre table de conversion RS 232 et de redéfinir les messages du QDOS.

Syntaxe : *TRA adr, 0* | *0, adr* adr = adresse d'implantation de la table

Exemples :

i. `TRA 262144,0` (indique que la table de conversion RS232 se trouve en 262144)
ii. `TRA 0,262147` (indique que la table des messages du QDOS se trouve en 262147)

Pour connaître la structure de ces tables, voir la documentation du QDOS.

TURN

TURNTO

Graphiques de tortue

TURN permet de faire tourner la direction d'une tortue d'un angle donné. **TURNTO** permet de faire tourner la tortue vers une direction donnée.

La rotation se fait dans la fenêtre reliée au canal par défaut ou au canal donné.

L'angle est donné en degrés. Un angle positif fait tourner dans le sens trigonométrique, un angle négatif dans l'autre sens (sens des aiguilles d'une montre).

A l'origine la tortue pointe à 0, sur la droite de la fenêtre.

Syntaxe : *angle:= expression_numérique* (angle en degrés)

TURN[canal,]angle
TURNTO [canal,]angle

Exemples :

i. `TURN 90`
ii. `TURNTO 0`

UNDER

Fenêtres

UNDER positionne ou enlève le soulignement des lignes qui vont être affichées. Le soulignement se fait dans la couleur **INK** de la fenêtre reliée au canal par défaut ou au canal donné.

Syntaxe : *paramètre:= expression_numérique* (0 ou 1)

UNDER *[canal,] paramètre*

Exemples :

i. `UNDER 1` positionne le soulignement
ii. `UNDER 0` enlève le soulignement

WINDOW

Fenêtres

WINDOW permet à l'utilisateur de changer la position et la taille d'une fenêtre. Les bords sont enlevés quand la fenêtre est redéfinie.

Les coordonnées sont les coordonnées par points relatives à l'origine de l'écran.

Syntaxe : *largeur := expression_numérique*
hauteur := expression_numérique
x := expression_numérique
y := expression_numérique

WINDOW [canal,] largeur, hauteur, x,y

Exemple : `WINDOW 30, 40, 10, 10` (fenêtre de 30x40 points à 10,10)

WIDTH

Périphériques

WIDTH permet de spécifier la largeur d'un périphérique autre qu'une console, par exemple, une imprimante.

Syntaxe : largeur_ligne := expr_num

WIDTH [canal,] largeur_ligne

Exemples :

i. `WIDTH 80` (sélectionne une largeur de 80 pour le périphérique lié au canal 1.)
ii. `WIDTH #6,72` (sélectionne une largeur de 72 pour le périphérique lié au canal 6.)