****
****

****

FLIB 3.2

****

par François LEIBER

http://leiber.free.fr

****
****

Ce texte contient une description détaillée de toutes les fonctions de Flib et de Flib2, regardez le Lisez-moi pour une présentation plus générale.

Sommaire

1. Utilisation
2. Infos
3. Fonctions générales
4. Fonctions graphiques
5. Fonctions texte
6. Fonctions high-scores
7. Fonctions de la Status Line
8. Fonctions touches
9. Fonctions de variables

Utilisation

Il suffit, dans un programme BASIC en général, de taper flib("command1" [,"command2",...])

exemple : flib("clrscr","msg:Hello, World !","slrcl")

Si vous utilisez des fonctions de flib2, alors :

exemple : flib2("version","mkhs:10,500,fl,sc","hide:sc")

Les arguments numériques peuvent être de n'importe quel type, exactement comme en BASIC : nombres, mais aussi noms de variables, expressions mathématiques, bref tout ce qui renvoie un nombre. Mais attention, un argument numérique doit commencer soit par un chiffre, soit par un espace :

exemple, remarquez l'espace devant 'liste', 'int' et 'sin' : flib2("drawline:0,1/2-1/5, liste[5], int(a+7*b), sin(atan(x^2))")

Un nombre ou un nom de variable seul sont traités rapidement et sans problèmes (je l'ai programmé à la main), alors que les expressions sont traitées de manière plus lente, et ne reconnaissent pas les variables locales. Si un argument comporte une erreur de syntaxe, Flib s'arrêtera et lancera une erreur comme le ferait une instruction BASIC standard.

Flib renvoie toujours une liste, vide à la rigueur, contenant les résultats de toutes les fonctions dans l'ordre où elles sont appelées. Cette liste est stockée dans la variable 'fl', qui peut être déclarée comme variable locale.

Si jamais vous ne comprenez pas une fonction, REPORTEZ-VOUS AUX PROGRAMMES D'EXEMPLE fournis avec Flib, qui sont là pour ça : 99% des questions peuvent être ainsi évitées, je vous en remercie. Je ne répondrai plus aux nombreuses questions qui trouvent leur réponse dans ces exemples, je n'ai vraiment plus le temps pour ça... De plus, Flib n'est qu'une librairie complètant les lacunes du BASIC TI, il vous faut déjà savoir programmer en BASIC avant de l'utiliser !

Infos

Les fonctions de la deuxième librairie (Flib2) sont précédées d'une étoile.

Tous les noms de fonctions sont en minuscule et le plus court possible afin de les rendre plus faciles et rapides à retenir et à écrire.

J'ai maintenant inclus toutes les protections possibles, je pense que Flib ne peut provoquer d'erreur, même si vous entrez des arguments erronés !

Fonctions générales

off

Eteint la calculatrice.

breakoff

Désactive le break (touche ON, voir exemple 'flpass'). Attention, l'utilisation de la fonction getkey() réactive le break, mais aussi tout appel à un programme, une fonction ou à string(), c'est pourquoi je n'ai pas mis de fonction pour le réactiver. Enfin, vu que cette fonction désactive toutes les interruptions, elle accélère vos programmes.

beep:freq[,duree]

Joue la note à la fréquence *freq* pendant le temps *duree*, donné en centièmes de secondes. *duree* vaut 4 par défaut (note brève, du genre petit bruit quand on appuie sur une touche).
Remarquez au passage que je me suis cassé pour que les notes soient aussi justes sur HW1 que sur HW2.

beep2:[tempo,]musique

Joue la mélodie *musique* au tempo *tempo* (temps par minute), 120 par défaut. S'arrête si l'utilisateur appuie sur une touche.
musique doit respecter les conventions suivantes (regardez l'exemple 'sound') :

Do	Do#	Ré	Ré#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
A	B	C	D	E	F	G	H	I	J	K	L

Pour changer d'octave, tapez 'O' suivi du numéro de l'octave, qui doit être compris entre 0 et 4 (2 par défaut).
Pour changer le type des notes (noire par défaut), tapez :

1	2	3	4	5	6	7	8	9
Ronde	Blanche	Noire pointée	Noire	Croche pointée	Croche	Double croche	Triple croche	Triolet
4 temps	2 temps	1+1/2 temps	1 temps	1/2+1/4 temps	1/2 temps	1/4 temps	1/8 temps	1/3 temps

Enfin, pour une pause, tapez 'P', elle sera de la même longueur qu'une note qui aurait été jouée à la place.

wait:valeur

Attend un certain temps sans rien faire, *valeur* est donné en dixièmes de secondes. Cette fonction est étalonnée avec précision pour une HW2.

*apd:valeur

Fixe la valeur de l'apd (nombre de secondes avant que la 89 ne s'éteigne toute seule). Renvoie la valeur enregistrée précédemment.

*error:nbre

Crée une erreur n° *nbre*, voir le manuel de la 89 pour les numéros des différentes erreurs. L'avantage d'une erreur provoquée manuellement est que vous pouvez ainsi avertir l'utilisateur d'une erreur, tout en ne quittant pas le programme en cours.

*reset

Réinitialise la 89. Cette fonction n'est pas vraiment la plus utile, mais elle est sûrement la plus marrante !

(*)version

Renvoie la version de votre librairie Flib.

+

Augmente le contraste.

-

Diminue le contraste.

getcont

Renvoie le contraste courant de la calculatrice, soit un entier entre 0 (très clair) et 31 (très foncé).

setcont:valeur

Fixe la valeur du contraste courant. *valeur* doit être un nombre entre 0 et 15 compris.

Fonctions graphiques

Dans toutes ces fonctions, l'argument *mode* peut être omis, il sera dans ce cas remplacé par 1. Les coordonnées sont comptées à partir du coin supérieur gauche, de coordonnées (0,0) jusqu'au inférieur droit, de coordonnées (159,99) (ou (239,127) sur une 92+).

gray

Active les niveaux de gris, efface l'écran dans les deux plans, et active le plan 1. A la fin de l'appel de Flib, la calculatrice attend que vous appuyez sur une touche, puis désactive les niveaux de gris, restaure l'écran initial et renvoie la touche appuyée (voir exemples 'grayex' et 'anni').

gray2

Même chose que 'gray', sauf que la fonction n'attend pas l'appui d'une touche à la fin de l'appel. Si vous appelez cette fonction une deuxième fois, elle désactivera les niveaux de gris.

plane:x

Change le plan actif pour les niveaux de gris : toutes les fonctions graphiques que vous appelez après dans le même appel de Flib seront effectuées sur ce plan. *x* peut valoir 0 ou 1.
Un pixel de couleur * correspond aux plans * allumés :

- blanc : aucun
- gris clair : plan 0
- gris foncé : plan 1
- noir : plan 0 + plan 1

clrscr

Efface tout l'écran.

sprite:[x,][y,][mode,]pic

Affiche une image *pic* aux coordonnées (*x*, *y*).
mode peut prendre les valeurs suivantes :

- 0 affiche l'image en inversé
- 1 affiche l'image en superposition
- 2 affiche l'image en 'xor'
- 3 efface la zone de l'image
- 4 affiche l'image en écrasant les pixels du dessous
- 6 affiche l'image en 'and'.

x et *y* valent 0 par défaut.

savpic:x1,y1,x2,y2,pic

Enregistre la partie de l'écran comprise entre les points (*x1, y1*) et (*x2, y2*) dans la variable *pic*.

rclscr:pic

Restaure l'image *pic* d'un écran enregistrée précédemment avec 'savpic'.

savscr:pic

Enregistre l'écran dans la variable *pic*.

picsize:pic

Renvoie dans une liste la taille horizontale et verticale de l'image *pic*.

rect:x1,y1,x2,y2[,mode]

Dessine un rectangle vide entre les points (*x1, y1*) et (*x2, y2*).
mode peut prendre les valeurs suivantes :

- 0 efface le rectangle
- 1 dessine le rectangle en noir
- 2 inverse le rectangle
- 32 pour un rectangle aux bords arrondis
- 64 pour un rectangle double
- 128 pour un rectangle avec les coins supérieurs tronqués

En fait, si vous recherchez un effet en particulier, essayez d'autres arguments, vous pouvez avoir des rectangles ronds inversés, des arrondis doubles, des rectangles avec les côtés en pointillés, etc.
De plus, les valeurs indiquées ci-dessus correspondent à des rectangles blancs par défaut, vous devez rajouter l'une des trois premières valeurs pour avoir des rectangles noirs ou inversés.

Ex : 97 (= 64 + 32 + 1) pour un rectangle arrondi double noir.

fillrect:x1,y1,x2,y2[,mode]

Dessine un rectangle plein entre les points (*x1, y1*) et (*x2, y2*).
mode peut prendre les valeurs suivantes :

- 0 efface le rectangle
- 1 dessine le rectangle en noir
- 2 inverse le rectangle.

filltri:x1,y1,x2,y2,x3,y3[,mode]

Dessine un triangle plein entre les points (*x1, y1*), (*x2, y2*) et (*x3, y3*).
mode peut prendre les mêmes valeurs que la fonction 'line'.

fillpoly:x1,y1,x2,y2,x3,y3,x4,y4[,mode]

Dessine un quadrilatère plein entre les lignes (*x1, y1*)-(*x2, y2*) et (*x3, y3*)-(*x4, y4*).
La première ligne doit se situer plus près du bas de l'écran que la deuxième, sinon rien n'est dessiné.
mode peut prendre les mêmes valeurs que la fonction' line'.

ellipse:x,y,r1,r2[,mode]

Dessine une ellipse de centre (*x, y*), de rayon horizontal *r1* et de rayon vertical *r2*.
mode peut prendre les même valeurs que 'fillrect'.

line:x1,y1,x2,y2[,mode]

Dessine une ligne entre les points (*x1, y1*) et (*x2, y2*).
mode peut prendre les valeurs suivantes :

- 0 efface la ligne
- 1 dessine la ligne en noir
- 2 inverse la ligne
- 7 pour une ligne double
- 8 pour une ligne avec un ombrage vertical
- 9 pour une ligne avec un ombrage horizontal
- 10 pour une ligne avec un ombrage en diagonal négatif
- 11 pour une ligne avec un ombrage en diagonal positif

En fait, les ombrages fonctionnent seulement avec les lignes ayant une pente supérieure à 45°.

pix:x,y[,mode]

Dessine un point de coordonnées (*x, y*).
mode peut prendre les même valeurs que 'fillrect'.

pixtest:x,y

Renvoie 1 si le pixel (*x, y*) est allumé, 0 s'il est éteint.

mkpic:largeur,hauteur,var,données

Enregistre dans la variable *var* une image de *largeur* et *hauteur* fixées, où l'image elle-même est directement fournie sous forme d'octets (une chaîne de caractères).
La structure des *données* est la suivante : chaque ligne est donnée par un nombre entier de caractères, chaque caractère contenant 8 pixels (si la largeur de l'image n'est pas un multiple de 8, les derniers bits du dernier octet de chaque ligne ne seront pas pris en compte). Les lignes sont écrites les unes à la suite des autres.

Pour contourner les problèmes des caractères non utilisables en BASIC, remplacez le caractère n°0 par '00', le n°2 par '02' et le n°13 par '13'.

Pour comprendre, regardez les utilitaires BASIC 'mkpic' et 'mkpic2' qui créent des chaînes de caractères exploitables par 'mkpic' ; vous pouvez aussi regarder mon Othello BASIC (lire les docs respectives).
Si j'ai fait une telle fonction, très près de la machine, c'est à force de voir des programmeurs BASIC distribuer leurs programmes avec un million de petites images dans divers répertoires. Cette fonction permet donc de créer ces images de manière temporaire avec très peu de mémoire et de manière très pratique !

map:[largeur,][x,][y,][horiz,][vert,][mode,][temps,]préfixe,données

Affiche une série d'images en une seule passe les unes à la suite des autres, et par lignes successives (voir exemple 'map').
largeur est le nombre d'images qui seront dessinées horizontalement ; quand ce nombre est atteint, la fonction continuera sur la ligne suivante. Quand cet argument est omis, toutes les images sont tracées sur la même ligne.
Les images seront dessinées à partir du point (*x*, *y*) ; ce point est (0, 0) par défaut.
horiz et *vert* indiquent quelle place doit être laissée à chaque image : si des images sont plus grandes que la place réservée, elles se chevaucheront. Quand *horiz* et *vert* sont omis, 'map' prend la taille des images comme arguments, mais dans ce cas il vaut mieux que les images aient toutes la même taille, sous peine d'avoir des décalages lors de l'affichage.
Enfin, *mode* peut prendre les mêmes valeurs que pour 'sprite'.
Toutes les images doivent avoir des noms composés de *préfixe* + 'un caractère' : *données* est la succession de ces caractères correspondant à toutes les images que vous voulez afficher. Vous pouvez donc afficher avec un seul appel à 'map' autant d'images différentes qu'il y a de caractères permis pour les noms de variables, soit plus de 90.
temps est un argument astucieux : son emploi transforme cette fonction en CyclePic évolué. En effet, la fonctions va alors tourner en boucle tant qu'aucune touche n'est appuyée, puis renvoyer le numéro de la touche. *temps* est le temps en centième de secondes que la fonction attend entre deux dessins d'images. Si vous donnez 0 et 0 pour*horiz* et *vert*, toutes les images vont être affichées les unes sur les autres, comme un CyclePic.

Ex : flib("map:,x,y,0,,4,temps,pic,123456")
où *pic1*, *pic2*,... forment une petite vidéo, et *x*, *y* et *temps* doivent être remplacés par des nombres.
En mettant le temps à 0, vous pouvez même faire des niveaux de gris (voir exemple 'grayex'), pratique pour afficher une image en niveaux de gris dans une interface en noir et blanc, ce que ne permet pas 'gray' (mais la qualité n'est pas aussi bonne quand même).

Fonctions texte

font:taille

Change la taille de la police actuelle.
taille peut prendre les valeurs suivantes :

- 0 pour une petite police (nx5 pixels)
- 1 pour la police standard (6x8 pixels)
- 2 pour la grande police (8x10 pixels).

drawstr:x,y,[mode,]string

Affiche la chaîne de caractère *string* aux coordonnées (*x*, *y*).
Si *x* vaut 999, alors la chaîne sera centrée horizontalement sur l'écran (n'abusez pas de ce centrage automatique, vos programmes de 89 risqueraient de moins bien passer sur une 92+ à cause de la taille de l'écran).

mode peut prendre les valeurs suivantes :

- 0 écrit en blanc sur noir
- 1 écrit en noir
- 2 écrit en inversé
- 3 écrit en grisé sur blanc
- 4 écrit en noir sur blanc.

Attention, si vous omettez l'argument *mode*, *string* ne doit pas commencer par un chiffre ou un espace, ou Flib croira que c'est *mode*. Une autre solution que je recommende consiste à mettre une virgule sans mettre d'argument.

drawstrv:x,y,[mode,]string

Affiche verticalement la chaîne de caractères *string* aux coordonnées (*x*, *y*).
mode peut prendre les mêmes valeurs que 'drawstr'.

pretty:x,y,[mode,]expr

Affiche en mode Pretty Print l'expression *expr* aux coordonnées (*x*, *y*).
mode peut prendre les mêmes valeurs que 'drawstr'.

pinfo:expr

Renvoie la largeur qu'aura l'expression affichée avec pretty, suivie de l'ordonnée de son haut et de son bas.
En clair, si vous l'affichez à (x,y), elle sera comprise entre (x, y+haut) et (x+largeur, y+bas).

width:taille,str

Renvoie la largeur en pixels que prend la chaîne de caractères *str* dans la police *taille*.

msg:string

Affiche la chaîne de caractères *string* en inversé sur un fond noir encadré et attend que l'utilisateur appuie sur une touche., puis restaure l'écran et renvoie la valeur de la touche appuyée.
Pour afficher le message sur plusieurs lignes, séparez-les par '|'. Chaque ligne sera tronquée à 25 caractères (38 sur une 92+) puis centrée sur l'écran, le nombre de lignes est limité à 8 (12 sur une 92+).

Attention, n'appelez pas cette fonction quand les niveaux de gris sont activés : les deux fonctions se servent de la même zone mémoire pour stocker l'écran de manière temporaire, donc Flib ne restaurera plus correctement l'écran à la fin de l'appel, comme elle devrait le faire avec 'gray'. Mais bon, vous pouvez le faire manuellement...

*menu:[hauteur,][x,][y,]titre,éléments...

Fait apparaître un menu rectangulaire avec autant d'éléments et de sous-éléments que vous voulez, avec le coin supérieur gauche au point (*x*, *y*). Les trois premiers arguments sont facultatifs, le menu aura la taille idéale et sera centré si vous les omettez. Si *hauteur* est insuffisant pour afficher tous les éléments, vous pourrez faire défiler le menu.
Chaque élément du menu est précédé d'une virgule, et chaque élément de sous-menu est précédé d'une '|'. Il ne peut y avoir qu'une profondeur de sous-menu.
La valeur renvoyée dans 'fl' correspond au numéro de l'élément sur lequel l'utilisateur a appuyé sur ENTER, ou 0 si il a appuyé sur ESC.

Essayez par exemple de taper ce qui suit sur votre calculatrice pour comprendre, ou reportez-vous à l'exemple 'menu'.

Ex : flib2("menu:20,titre,option1,option2|sous-option1|sous-option2")

*numline:var

Renvoie le nombre de lignes de la variable texte *var*.
En cas de problème - *var* n'existe pas ou n'est pas une variable texte - la fonction ne renvoie rien.

*getline:n,var

Renvoie la *n*-ième ligne de la variable texte *var*.
Dans un texte, la première ligne est numérotée 0 (valable aussi pour les deux fonctions suivantes).
En cas de problème, la fonction ne renvoie rien.

*delline:n,var

Supprime du texte *var* sa *n*-ième ligne.
Renvoie 1 en cas de succès, 0 sinon.

*insertline:n,var,str

Insère la ligne *str* entre la (*n*-1)-ième et la *n*-ième ligne du texte *var*.
Renvoie 1 en cas de succès, 0 sinon.

Fonctions high-scores

Une table de high-scores est une variable contenant plusieurs scores, formés chacun d'un nombre, de 0 à 65535, et d'un nom, de 10 caractères maximum.
La variable se présente sous la forme d'une chaîne de caractères, mais vous ne verrez que *'Highscores by Flib'*, vous n'avez pas accès au reste des données. Vous pouvez trouver un exemple d'utilisation dans 'memory'.

*mkhs:n,rec,nom,var

Crée une variable *var* contenant une table de *n* records, initialisés à la valeur *rec* et au nom *nom*.
Si *nom* fait plus de 10 caractères, il sera tronqué.

*puths:rec,nom,var

Insère automatiquement dans la table de record *var* le record donné par le nombre *rec* et le nom *nom*.
Retourne la classement du nouveau record, ou 0 si *rec* est un record trop mauvais pour être classé.
Si *nom* fait plus de 10 caractères, il sera tronqué.

*geths:n,var

Renvoie dans 'fl' le *n*-ième record de la variable *var*, sous la forme d'un nombre suivi d'une chaîne de caractères pour le nom.

Fonctions de la Status Line

slclr

Efface le contenu de la Status Line.

slrcl

Redessine la ligne au dessus de la Status Line.

slmsg:string

Affiche la chaîne de caractères *string* dans la Status Line.

busy:mode

Change l'indicateur de la Status Line.
mode peut prendre les valeurs suivantes :

- 1 affiche 'BUSY'
- 2 affiche 'PAUSE'
- 3 efface l'indicateur.

Fonctions touches

keywait

Attend que l'utilisateur appuie sur une touche et renvoie la valeur de la touche appuyée.
Attention, les touches suivantes ont des codes différents que ceux renvoyés par getkey() en basic, les autres sont les mêmes.

	Normale	2nd	Shift	Diamant	Alpha
Haut	337	4433	8529	16721	33105
Gauche	338	4434	8530	16722	33106
Bas	340	4436	8532	16724	33108
Droit	344	4440	8536	16728	33112

Attention : pour une 92+, les touches Haut et Gauche sont inversées.

keylow

Permet de récupérer à bas niveau le code de touches appuyées : ce code ne détecte que les appuis sur les flèches et les quatre touches 2nd, Shift, Diamant et Alpha, idéal pour les jeux d'action (voir exemple 'keylow').
De plus, cette fonction ne vidange pas le tampon : en clair, elle est indépendante des getkey() et autres fonctions sur les touches que vous pouvez utiliser, elle renvoie juste l'état actuel du clavier. Enfin, il détecte quand plusieurs touches sont appuyées à la fois.
Il renvoie une chaîne de caractères dont les bits représentent successivement les états des touches 2nd, Shift,

Diamant, Alpha (Lock pour les 92+), Gauche, Droite, Haut et Bas.

Par exemple, "10001010" signifie que les touches 2nd, Gauche et Haut sont appuyées, et que toutes les autres sont relâchées.

initdelay:valeur

Fixe la valeur (395èmes de seconde) au bout de laquelle la répétition des touches est activée, 0 désactive la répétition des touches.
Si vous mettez une valeur du genre 1, vous n'arriverez plus à diriger le curseur correctement car il avancera de plusieurs crans à chaque fois (marrant comme blague)...
La valeur par défaut est 336, mais de toute façon elle est restaurée à chaque fois que l'on éteint-rallume la 89.
Renvoie la valeur enregistrée précédemment.

delay:valeur

Fixe le temps (395èmes de seconde) entre deux répétitions de touches.
La valeur par défaut est 48, mais de toute façon elle est restaurée à chaque fois que l'on éteint-rallume la 89.
Renvoie la valeur enregistrée précédemment.

Fonctions de variables

*hide:var

Cache la variable ou le dossier *var* (la rend invisible du TIOS).
Pour un dossier, la fonction commence commence par le replier, afin de cacher aussi les variables qu'il contient.

*unhide:var

Décache la variable ou le dossier *var* (voir 'hide').

*size:var

Renvoie la taille de la variable *var*.

*state:var

Renvoie l'état d'une variable ou d'un dossier sous forme d'un nombre de trois bits : le premier bit indique si *var* est archivé, le deuxième s'il est verrouillé et le dernier s'il est caché.
En clair, vous devez faire ceci si vous voulez récupérer les caractéristiques une par une :

```
flib("state:var")
fl[1] and 1->archived
fl[1] and 2->locked
fl[1] and 4->hidden
```

*folder:var

Renvoie la liste de tous les répertoires où se trouve la variable *var*.

*getfolder

Renvoie la liste de tous les répertoires présents dans la 89.

*getfile:[type,]folder

Renvoie dans une liste le nom de toutes les variables présentes dans le répertoire *folder*.
De plus, vous pouvez ajouter un argument *type*, alors la fonction ne renverra que les variables de type *type*.
type peut prendre les valeurs suivantes :

- 45 -> pour ne renvoyer que les chaînes de caractère
- 217 -> listes ou matrices
- 220 -> programmes ou fonctions
- 221 -> data
- 222 -> GDB
- 223 -> images
- 224 -> textes
- 225 -> figures
- 226 -> macros
- 243 -> programmes ou librairies en assembleur

Les expressions n'ont pas de valeur assignée, c'est pourquoi elles n'apparaissent pas ici.

*getbyte:n[,nbre],var

Renvoie la valeur du *n*-ième octet de la variable *var*.
Si l'argument *nbre* est présent, Flib renvoie sous la forme d'une chaîne de caractères *nbre* octets de la variable *var* en partant du numéro *n*.
n doit être compris entre 0 et 'size(*var*)-3'.
L'argument *nbre* est beaucoup plus puissant et pratique pour programmer un éditeur que de récupérer les octets un par un, cependant vous pouvez rencontrer une erreur lorsque la chaîne contient des caractères non autorisés par le TIOS. De plus, *nbre* est limité à 4997.

*setbyte:n,val,var

Remplace la valeur du *n*-ième octet de *var* par *val*.
n doit être compris entre 0 et 'size(*var*)-3', val entre 0 et 255.
Cette fonction vous donne un énorme contrôle sur les variables, mais aussi sur la stabilité du TIOS...

*memcpy:ndest,nsrc,num,dest,src

Copie *num* octets à partir du *nsrc*-ième de la variable *src* dans la variable *dest* à partir de son *ndest*-ième octet.

Ex : "Flib by François Leiber"->a : "Flib2 & "->b : flib2("memcpy:9,1,23,b,a")

*komp:var

Si la variable *var* n'est pas compressée, alors cette fonction la compresse, c'est à dire la transforme en une autre variable de type KOMP, prenant moins de place en mémoire. Si var est déjà de type KOMP, décompresse automatiquement la variable.
La fonction renvoie un nombre, qui peut prendre les valeurs suivantes :

- 0 : (dé-)compression réussie
- 1 : la variable *var* n'existe pas
- 2 : pas de compression effectuée (variable non compressée à l'origine), la variable compressée prendrait

plus de place que l'originale
- 3 : pas assez de mémoire pour lancer la (dé-)compression.

*kinfo:var

Renvoie, à la suite, le type (voir fonction 'getfile') et la taille d'origine de la variable compressée *var*. Renvoie juste 0 si *var* n'est pas de type KOMP.

*group:[n,][var1,var2,...]var

Lorsqu'il y a plus d'un nom de variable en argument, la fonction regroupe les variables *var1*, *var2*, ... dans la variable *var*, qui sera de type GRP. Si *var* est déjà de type GRP et qu'il n'y a qu'un nom de variable *var1* en plus, la fonction rajoute *var1* dans la variable *var*. La somme de la taille des variables ne doit pas dépasser 65 Ko, par limitation du TIOS. Les noms de variables peuvent inclure un nom de répertoire.
Renvoie le nombre de variables enregistrées avec succès dans *var*.

Lorsqu'il n'y a qu'un nom de variable, la fonction comprend qu'il faut extraire les fichiers contenus dans cette variable de type GRP (ne fait rien si la variable est d'un autre type). Si en plus, il y a un argument numérique, extrait seulement le *n*-ième fichier de la variable.
Renvoie le nombre de variables extraites avec succès de *var*.

Cette fonction est pratique pour éviter d'avoir des programmes avec des milliers de fichiers (voir les exemples). En plus, la compression avec 'komp' est plus efficace quand vous regroupez plein de fichiers du même genre avant la compression, que de les compresser un par un puis de les regrouper.

*ginfo:var

Renvoie la liste des fichiers contenus dans la variable GRP *var*.

*mkvar:taille,[type,][extension,]nom

Crée une variable appelée *nom* de *taille* octets, remplie de 0.
type est un nombre permettant de donner le type de la variable (voir 'getfile' pour connaître les différentes possibilités). Par défaut, 'mkvar' crée un texte (*type* vaut alors 224).
extension est une chaîne de 4 lettres maximum pour créer une variable de type personnel, *extension* est le type qui s'affichera dans Var-Link.
Si vous mettez à la fois les arguments *type* et *extension*, c'est cette dernière qui l'emportera.

Ex : flib2("mkvar:7,aaa","setbyte:2,32,aaa")
pour créer un texte vide appelé 'aaa'. Le deuxième octet fixé à 32 indique le début de la première ligne, indispensable pour que le TIOS ne s'emmêle pas les pédales.

Ex : flib2("mkvar:50,pers,aaa")
pour créer une variable de 50 octets qui apparaîtra de type 'pers' dans Var-Link.

*type:var

Renvoie l'extension de la variable de type personnel *var* (créée par exemple avec 'mkvar').
Ne renvoie rien si la variable n'a pas un type personnel.

Retour au sommet de la page