NEW PROG DOC *(french and english version)*

Sommaire / Summary:

0 Purposes/Launching prgm – Qu'est-ce que NewProg/utiliser Newprog

See below for english translation.

Newprog est un langage de programmation on-calc qui étend grandement les possibilités du Ti-basic et donne accès à des fonctions propres au C. Newprog est un langage qui souhaite apporter de nouvelles fonctionnalités à des programmes existants (flib).
La package newprog comprend un compilateur (newprogc), un interpréteur (newprog) et une interface rendant plus accessible l'utilisation de ces programmes pour débuttants et expérimentés (npcc.89p).
Le code source peut être édité directement dans l'éditeur de programme de votre TI89/Ti92/V200. La syntaxe Newprog ressemble à du Tibasic, elle est même identique pour plusieurs fonctions. Cependant l'exécution sera beaucoup plus rapide en Newprog. Newprog permet de passer d'instructions Newprog à des instructions Tibasic de manière aisée par l'utilisation des mots clés **basic** et **endbasic**.
Newprog est souvent jusqu'à 300 fois plus rapide que le Tibasic avec une vitesse d'exécution proche du C (estimé de 10 à 30 fois plus lent que le C) et de l'assembleur tout en gardant une simplicité de programmation en comparaison à l'assembleur et au C. Newprog permet des opérations arithmétiques basiques, l'accès à la mémoire, la création et la lecture de fichiers, l'utilisation de timers/compteurs, l'affichage de fonctions

graphiques (en noirs et blancs et en niveaux de gris), d'importer des programmes en assembleurs, et bien d'autres. La rapidité de newprog vous permettra de réaliser directement sur votre calculatrice tous vos projets sans être limité (dans une certaine mesure) par la lenteur d'exécution du programme.
Des informations complémentaires, notamment sur les fichiers fournis dans le fichier compressé, sont données ci-dessous en langue anglaise (notamment sur comment éditer, compiler et exécuter un programme).

NewProg is a new programmation language based on the Tibasic. The source program will look like a Tibasic program, but The Tibasic syntax is sometimes allowed in NewProg (for example the if then else endif block, For Endfor , While endwhile, → (sto), {} list, « string », ( ), [] for list, label, goto) to help the programmer to make the transition from a simple programmation langage to a more closer to machine langage. You will need to understand Tibasic before using NewProg.
NewProg performs some significants ameliorations (in comparaison with Tibasic, vertel, flib) in terms of speed, possibility (next to C and assembly) and program size (2 times smaller). Powerfuls and usefuls functions are implemented for this purposes. You can run assembly programm directly in Newprog Programs. You are allowed to file acessing, gray printing, screen scrolling, Interruption and timers, keys... The execution speed is very fast.
NewProg allowed to program also in Tibasic directly in a NewProg program (only the *Local* function is not supported). You make the transition between Newprog and Tibasic instructions by simply adding the key words **basic** and **endbasic**.
Each program containing Newprog instructions shall contains the init() instruction at the very beginning of the program (for compilation purposes).

How to edit and write a Newprog program (program source) :

```
Myprgm()                                  //the program source is : « myprgm.89p »
Prgm
init()                                    //You have to put this indication at the beginning of the
Pause «Newprog instruction1 »             //program. If you omit to write init(), the compilation will
Pause « Newprog instruction2 »            //fail
basic                                     //indicate that it will now execute Tibasic instructions
prints(« Tibasic instruction1»
endbasic
pause « Newprog instruction3 »
basic
prints(« Tibasic instruction2 »
endbasic                                  //indicate that it return in Newprog execution mod
pause « Return to Newprog »
EndPrgm
```

A NewProg program must be edited in the Tibasic program editor.

How to compile and run a NewProg program :

If you have not a TI89 HW4, install preos.
The compilation of your program will be done by launching in the Homescreen the following command :
npcc(« prgm »)
This is the best way to compile a program because it display a browser and help to prevent unwanted files deleting after a crash (automaticaly archiving). If compilation is successful, Npcc will ask you if you want to execute the generated output file.
In an other way, you can compile your program by launching :**newprogc(« prgm »).** But if « prgm » has not been previously precompiled (by the Tios in Tibasic), it will crash your calculator. To avoid problem, use **npcc(« prgm »)** .

The output file at the compilation is out.NPP by default. You can rename this file as needed. Fore example, if you have renamed the file out.NPP by *runable.NPP*, you can execute with the following command :
newprog(« *runable* »)

If you have not an HW4 TI89 :
Because Newprog uses huge executables, you may have installed an os to use Newprog. An os is furnished in the bundle of Newprog. Install it with the preos() command in the home screen after have transferring to your calculator all the preos files.

Very Important : I am not responsible for any damages and data loses on your calculator. Because Newprog allow the user to use low level functions, Newprog can crash the calculator if you have made an error in your program. To avoid data lost, you may archive your data before using this programming.

1 Types de données –Data types

See below for english translation.

Les variables (se nomment comme les variables TiBasic, par exemple va). Jusqu'à 255 noms de variables sont autorisés.
Toutes les variables de newprog ont des valeurs codées sur 4 octets. Chaque variable peut être des types suivants :

- des entiers signés : de -2 147 483 648 à + 2 147 483 647
- des pointeurs sur tous types de données

Les noms de variable à 1 seul caractère sont interdits (par exemple *x* et autres lettres grecques...) car elles sont réservées à d'autres usages.

Les listes :
Elles ne sont pas des variables mais plutôt des espaces mémoires de dimensions variables réservés aux stockage de certains types de données. L'adresse de cet espace mémoire sera (souvent) stocker dans une variable afin de pouvoir le manipuler ultérieurement. Elles sont soient prédéfinies à la compilation, on parlera alors de listes instanciées prédéfinies, soient définies lors de l'exécution du programme, on parlera alors de listes instanciées dynamiques. Les listes prédéfinies ont l'avantage de ne pas devoir être libérer de la mémoire avant la fin de l' exécution du programme contrairement aux listes instanciées dynamiques. Les listes instanciées dynamiquement ont l'avantage de pouvoir être créées autant de fois que nécessaire lors l'exécution du programme mais ont comme inconvénient de devoir être libérer manuellement (avec la fonction **free**()) avant la fin du programme sous peine de perdre définitivement de la mémoire ram (un reset sera alors nécessaire pour restaurer la mémoire).
Un troisième type de liste, dit liste non instanciée existe. L'usage de ce type de liste est moins fréquent que ceux des listes instanciées. Ce type de liste n'a pas d'espace mémoire associé. Il ressemble aux listes Tibasic car elles peuvent être réévaluer (exécuté) à la demande (fonction **expr**() ou #). On pourra la transformer en liste instanciée (fonction **group**()).

- **1-a Les listes instanciées définies dynamiquement** (c'est à dire créées lors de l'exécution d'instructions dédiées) :
  Syntaxe 1 : {*nom_de_la_variable,taille_des_éléments,élément1,élément2…,élémentn}*

Lors de l'exécution d'une liste instanciée, un espace mémoire est créé. Le pointeur de cet espace mémoire est alors affecté automatiquement à la variable *nom_de_la_variable* (pour information, l'instruction renverra aussi la valeur du pointeur). Cet espace mémoire est alors initialisé comme une liste de *n* éléments, chacun d'une taille *taille_des_éléments* en octets. Les éléments de cette liste sont alors *élément1,élément2* etc… Le premier élément de la liste sera *élément1*. Les *éléments* doivent être du type d'un nombre (voire constante prédéfinie) ou d'une variable (les fonctions sont interdites). On peut lire ou écrire dans cette liste avec les fonctions **lb**(*taille des éléments*=1), **lw**(*taille des éléments*=2), **ll**(*taille des éléments*=4) pour la lecture et **wb**(), **ww**() **wl**() pour l'écriture. On peut aussi utiliser les crochets classiques du Tibasic [].
Exemple : Pour lire le premier élément d'une liste instanciée nommée *ll (=nom_de_la_variable),* définie avec une taille de ses éléments égale à 1, utiliser l'instruction suivante : lb(*ll*,0) ou ll[0]. La lecture (et l'écriture) par crochet détecte automatiquement la *taille_des_éléments*, c'est donc un moyen plus simple d'utiliser les listes instanciées.
Syntaxe 2 : A l'aide des fonctions seqb(),seqw(),seql() et autres fonctions (voir leurs définitions plus bas)

- **1-b Les listes instanciées prédéfinies à la compilation** (c'est à dire créées lors de la compilation, elles sont stockées directement dans le fichier exécutable) :

Ce type de variable est très utile car permet de définir facilement (plus aisément que la syntaxe {} des listes instanciées dynamiques) et directement dans le corps du programme chacune des valeurs des éléments d'une liste instanciée. Cela peut être pour définir des sprites par exemples. La position de la définition dans le code source n'a pas d'importance, une instruction en début de programme connaitra une liste instanciée prédéfinie définie en fin de programme.
Lors de l'exécution du programme, l'accès aux éléments de la liste peut être réalisé de la même façon que pour une liste instanciée définie dynamiquement ([] ou **lb**(),**lw**(),**ll**()). Il est également possible de modifier les valeurs des éléments.

Syntaxe 1 : ***b****(nom_de_la_variable)*:*data1*:*data2*:...:*datan*:*y*

Lors de la compilation, un espacé mémoire sera alloué directement dans le corps du programme exécutable. Cet espace mémoire contiendra autant d'éléments codés sur un octet (byte) que de nombre de *data*, soit n éléments. Le premier éléments aura la valeur de *data1*, la dernière

celle de *datan*. Chaque *data* pourra être de la forme décimale,binaire ou hexadécimale. Afin de pouvoir définir plusieurs *data* en une seule instruction dans le code source, ce qui permet de rendre le code source plus concis, l'instruction *repeat*() peut être utilisée (ne pas la confondre avec la fonction repeat() plus bas car elle n'a pas la même fonctionnalité). Repeat prend deux arguments : **repeat**(*count, value*). Lors du passage sur cette fonction, le compilateur ajoutera autant d'éléments que *count* et les définira à la valeur *value*.
A l'exécution, le pointeur sur l'espace de données sera à nouveau affecté à la variable nom_de_la_variable.

Exemple : Affiche un sprite (un H à l'horizontal). (Remarque : remplacer ce bloc b:y par b(sprite):repeat(3,-1):y affichera 1 sprite plein).

```
init()
clrscr()
b(sprite)
0b11111111
0b00011000
0b11111111
y
sprt8(0,0,3,sprite)
keywait()
```

Syntaxe 2 : ***w****(nom_de_la_variable)*:*data1*:*data2*:...:*datan*:*y*
Similaire à **b**()::y mais avec des éléments codés sur 2 octets.
Syntaxe 3 : ***l****(nom_de_la_variable)*:*data1*:*data2*:...:*datan*:*y*
Similaire à **b**()::y mais avec des éléments codés sur 4 octets.

- **2- Les listes non instanciées :**

Syntaxe : {*élément1*,*élément2*...}
Lors du lancement d'une ligne de commande contenant une structure non instanciée, rien ne se passera. La structure renverra seulement un pointeur utile pour la manipulation de cette structure. On pourra ainsi effectuer diverses opérations tel que l'exécution des éléments à la demande (fonction **expr**() ou #), le passage de donnée à des variables Tibasic (fonction **toos**()), la création d'une liste instanciée dynamique composée des renvois des éléments (fonction **group**()).
Un des usages est qu'elle permet de passer des listes à une variable Tios. Un autre usage est qu'elle permet d'exécuter plusieurs fois les éléments *element1, element2*... (avec **expr**() ou **#**) ; cette méthode étant parfois plus simple que l'appel d'une fonction.
Il faut bien faire attention à ne pas placer une variable comme premier argument aussinon le compilateur la confondra avec une liste instanciée dynamique. Si vous êtes obligé de placer une variable en première position (sans désirer que le compilateur la considère comme une liste instanciée dynamique), utiliser par exemple l'astuce suivante (le premier élément est donc la fonction somme et non plus un nom de variable) :
{nom_variable+0,élément2….}
Ainsi le compilateur sera trompé en ne la considérant pas comme une liste instanciée dynamique.

ENGLISH TRANSLATION :

Variables (can be named in the same way than Tibasic, for instance va). You can use up to 250 variable names :

All the data var in NewProg programmation are 4 bytes long. Each data var can have the following types :

- signed integer : from -2 147 483 648 up to + 2 147 483 647
- pointers (points to the desired area of the calculator memory)

Varirable written with only one character are forbiden (example var *x*).

Lists (Arrays) :

They represents allocated memory blocks where the user can store data into it. Theirs total amount of bytes dimensions are defined by the user. They are useful for storing data than can't be stored in only 4 bytes data long. The pointer (ie adress) of this memory block is commonly store in a standard newprog variable (see just above) in the purpose to be used after in the program. A list can be of two kinds : an instantiated list defined dynamicaly or an instantiated list defined previously during compilation. A list defined dynamically is created during the execution of the program and its content set dynamically too. A list defined previously during compilation is created and set during the compilation and stored directly in the source program. A dynamicaly created list advantage is that it can be created how often the user wants during the execution (instead of previously defined list). Their inconvenient is that the user have to free the allocated memory block before the end of the execution of the program to avoid memory loose (so use **free**() function). The previously defined list has the advantage to be easier to defined manually by the user and to have not to be free before the end of the execution of the program.
I have to mention a minor type of list that is the non instantiated lists. This king of list is less commonly used than the instantiated lists. This kind of list have no allocated memory block. They are next to the tibasic list because its elements can be « relaunched » when needed (with the **expr**() or **#** functions). You will be able to convert a non instantiated list to an instantiated list (**group**() function).
See theirs writing just below :

1-a The instantiated list defined dynamicaly

Syntax 1: {*variable_name,size_in_byte_of_an_element,element1,element2…,elementn*}
During the execution of a dynamicaly instantiated list by NewProg, an allocated memory block will be created and its pointer will be stored into the variable *variable_name*. The first element of this array (ie list) will be *element1*, the second will be *element2* and so on. The *element* must be a number (can be predefined constant) or a variable (function are forbiden). You can read the value with lb()(for *size_in_byte_of_an_element*=1),lw() (for *size_in_byte_of_an_element*=2),ll() (for *size_in_byte_of_an_element*=4) or peekb() and similar

functions. You can also use the Tibasic syntax by using [] for reading or writting (as the way of tibasic). By using the [] notation, NewProg will detects automatically the *size_in_byte_of_an_element*, so this method is easier to use.
Example : for read the first element in a byte instantiated list named *variable_name*, tape the following instruction : lb(*variable_name*,0) or *variable_name*[0]. For writting, you can simply write for instance : 1->ll[0]
The location of the memory block wil be placed in *variable_name* and also returned at the end of the execution of the instruction. You will have to delete manually the allocated memory block manually (by using the function **free()**. See below) to avoid losing of memory.

Syntax 2 : By using seqb(),seqw(),seql() and other functions (documented in this document)

1-b The instantiated list defined previously during compilation

This kind of data is useful because it allow to define (in an easier way than using {} of the dynamicaly instantiated list) directly in the source program each one of the value of the elements of an instantiated list. It can be used to defined sprites for example (for graphic purposes). You will have to take no care of the position of the definition of the instantiated list in the source program, in such way you can defined your data at the end of your program source if you prefer. As an dynamicaly instantiated list, you can access to the data by using the [] notation or by using **lb**(), **lw**() or **ll**() functions. You are allow to modify the content of your data too.

Syntax 1: **b**(*name_of_the_destination_var*):*data1*:*data2*:...:*datan*:**y**

During compilation, a memory block is allocated and its pointer is stored in the variable *name_of_the_destination_var*. This memory block will contains an amount of *n* elements written on one byte. The first element will be equal to *data1*, the second element to *data2* and so on. Each *data* could be on the decimal, binary or hexadecimal format. In aim to define several data in only one instruction directly in the source program, a dedicated function (**repeat**()) is available. So the source program will be simplier. **Repeat**() takes two arguments : **repeat**(*count*,*value*). The compilator interprets this function by adding an amount of *count* elements affected to the value *value*.
At the execution, the pointer to the memory block will be stored again in the variable *name_of_the_destination_var*.

Example : Dispay a sprite (horizontal H). (Note : If you replace the b:y block by b(sprite):repeat(3,-1):y ; it will displays a filled rect).

```
init()
clrscr()
b(sprite)
0b11111111
0b00011000
0b11111111
y
sprt8(0,0,3,sprite)
keywait()
```

Syntax 2: **w**(*name_of_the_destination_var*):*data1*:*data2*:...:*datan*:**y**

Similar to **b**()::**y** but with elements coded on two bytes.

Syntax 3: **l**(*name_of_the_destination_var*):*data1*:*data2*:...:*datan*:**y**

Similar to **b**()::**y** but with elements coded on for bytes.

2 The non instantiated lists

Syntax : {*element1,element2…*}

If you launch a non instantiated list without using the command **expr**(),**#** or **group**(), no instructions will be executed. NewProg will just returns a pointer. This pointer will be useful if you use the **#, expr(), toos()** or **group**() functions. It will allow you for example to execute *element1, element2* as needed or to store a list in a tios variable (so you can communicate with Tibasic, using **toos**() function).
Be cautious, do not put a variable name in the first element of the nilist because the compilator will assume that it is an instantiated list. You can bypass this problem by using the following trick :
{*var_name+0,élément2…*}

2 Fonctions mémoire – Memory functions

Certaines fonctions allouent explicitement ou implicitement de l'espace mémoire, c'est le cas notamment pour **seqb**(), **seqw**(), **seql**(), **malloc**(), **realloc**(), **group**(), **gsprt**(), **newstr**(), **getpic**(), **seqe**(), **seqs**(), **archi**(), **unarchi**(),**toos**(),**files**(),**reps**(), **savescr(), loadscr()** et les listes instanciées dynamiquement de type **{varname,elem_size,...}**. Elle peuvent être confrontées à des problèmes de manque de mémoire disponible. Dans ce cas, elles ont été conçues pour renvoyer le plus souvent la valeur 0 comme indicateur de manque de mémoire. Pour faciliter la programmation, elles permettent aussi l'affichage automatique (voir fonction **disperr**()) d'une boite de dialogue d'erreur qui peut permettre à l'utilisateur de quitter l'exécution du programme ou bien de la continuer.

A memory function can allocate dynamically memory. This is the case of **seqb**(), **seqw**(), **seql**(), **malloc**(), **realloc**(), **group**(), **gsprt**(), **newstr**(), **getpic**(), **seqe**(), **seqs**(), **archi**(), **unarchi**(),**toos**(),**files**(),**reps**(), **savescr(), loadscr()** and the dynamically instanciated list **{varname,elem_size,...}**. These functions can encounter memory problem (not enough memory). In this case, they will often return the value 0 to indicate to the user that there is a memory error. In the aim to facilitate the programmation, these functions now can display automatically an error dialog box which one will let choose the user to continue or to quit the program execution (see **disperr**() function).

Fonction : val sto or →
Numéro de la fonction : 0d172
Syntaxe : s**to**(var_dest,*data*) or data→var_dest
Description :
Affecte la variable Newprog *var_dest* à la valeur *data*. Même fonctionnalité que l'écriture Tibasic classique.
Certaines fonctionnalités intéressantes propres à Newprog ont été ajoutées, notamment avec l'utilisation des crochets []. Pour plus d'information, voir la fonction **setttype**().

Store is the Newprog variable *var_dest* the value *data*. Similar functionality than Tibasic writing.
Others interesting functionality have been added, in particular when using [] notation. See **settype**() function for more informations.

Example : Print "My age : 20" to the screen and wait for a key press.
```
init()
20 → myage
clrscr()
pause "My age : "&string(myage)
```

Fonction : ptr malloc
Numéro de la fonction : 0d251
Syntaxe : **malloc**(*size_in_bytes*)
Description :
Alloue un espace mémoire de *size_in_bytes* bytes. On doit libérer l'espace mémoire avant la fin du programme pour éviter les pertes de mémoire, pour cela il est indispensable de stocker le pointeur de retour de cette fonction (communément dans une variable).
Si vous souhaitez utiliser les [] pour lire ou écrire dans cet espace mémoire (par le biais d'une variable), le type de l'espace mémoire sera converti à celui du type de la variable (par défaut étant de 4 bytes, mais modifiable avec la fonction **settype**()).
La fonction renvoie le pointeur de l'espace mémoire alloué.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).
Note : la fonction **realloc**() permet de modifier la taille du block mémoire à la demande.

malloc allocates a block of s*ize_in_bytes* bytes from the memory heap. It allows a program to allocate memory explicitly as it's needed, and in the exact amounts needed. On success, malloc returns a pointer to the newly allocated block of memory. If not enough space exists for the new block, it returns 0. If the argument *Size* is zero malloc also returns 0.
You must free the memory space before exiting the program. To do so, you must save the return value of this function (for instance in a variable).
If you want to read or write in this block memory, you can use the [] notation (or **lb**(),**lw**(),**ll**(),**wb**()...). If you use [] notation (associated with a variable name of course), the memory block will be of the type of the variable (by default set to 4, but you can change its type with **settype**()).
If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.
Note : la fonction **realloc**() permet de modifier la taille du block mémoire à la demande.

Fonction : ptr realloc
Numéro de la fonction :
Syntaxe : **realloc**(*ptr*, *new_size*)
Description :
Realloc() permet de modifier la taille du bloc mémoire préalablement alloué pointé par *ptr* (avec **malloc**(), **seqb**(),etc..) à la taille *new_size*.
Realloc() retourne un pointeur sur le nouveau block mémoire (le block ayant pu changer d'emplacement dans la mémoire). Le contenu du block mémoire n'est pas perdu pendant le redimensionnement.
Retourne 0 et libère le block mémoire pointé par *ptr* si le block mémoire n'a pu être réalloué. Cette fonction affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Reallocates allocated memory. **Realloc**() attempts to shrink or expand the previously allocated block to *NewSize* bytes. The *Ptr* argument points to a memory block previously obtained by calling **malloc, seqb, realloc** etc.... It may also be NULL; in this case the function simply calls malloc. **Realloc()** adjusts the size of the allocated block to *new_size*, copying the contents to a new location if necessary (note that the block will not stay in high memory after the reallocation, but it will still remain locked; see other functions from this header file for more info). realloc returns the address of the reallocated block, which can be different than the address of the original block. If the block cannot be reallocated, realloc returns NULL.
Usually **realloc**() keeps the current contents of the block intact; only the portion of the block which is newly allocated is not initialized and contains random content.

The function frees the block pointed to by *ptr* if there was an error, so you can not assume that the data pointed to by *ptr* is still valid. If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function **disperr**()), an error dialog box will appear.

Fonction : - free
Numéro de la fonction : 0d252
Syntaxe : **free**(*ptr*)
Description :
Libère l'espace mémoire pointé par *ptr* et alloué avec des fonctions telles que **malloc**() ou **seqb()**, **seqw()**, **seql()**, **group()** par exemple. Si *ptr* ne pointe pas sur un espace mémoire existant alloué dynamiquement, la fonction plantera la calculatrice.

Free the memory space allocated dynamically with a Newprog function (like **malloc**(), **seqb**(),etc...). If *ptr* does not point to an active dynamically allocated memory block, the function will crash the calculator, so be carreful.

Fonction : ptr memset
Numéro de la fonction : 0d253
Syntaxe : **memset**(*ptr,val,num*)
Description :
Affecte *num* bytes par la valeur *val* à partir de l'adresse *ptr*.

Set *num* bytes with the value *val* by starting to the *ptr* address.

Fonction : ptr memcpy
Numéro de la fonction : 0d
Syntaxe : **memcpy**(dest, src, len)
Description :
Copie un block mémoire de taille *len* octets de l'espace mémoire pointé par *src* vers l'espace mémoire pointé par *dest*. Si *src* chevauche *dest*, **memcpy**() peut planter la calculatrice (dans ce cas utiliser la fonction **memmove**()). La fonction retourne *dest*.

Copies a block of *len* bytes from *src* to *dest.* **Memcpy**() copies a block of *len* bytes from *src* to *dest*. If *src* and *dest* overlap, the memcpy is ill-behaved (in such case, use memmove instead). memcpy returns *dest*.

Example :

```
:init()
:memset(getlcd(),255,3840)          //Fill the screen with black
:malloc(3000)->ptr                  //Allow space for memory save
:memcpy(ptr,getlcd(),3840)          //Save the screen into ptr
:keywait()
:prints("The screen is cleared")
:keywait()
:clrscr()
:memcpy(getlcd(),ptr,3840)          //Restore the screen
:prints("Screen reloaded")
:keywait()
:free(ptr)
```

Fonction : ptr memmove
Numéro de la fonction : 0d
Syntaxe : **memmove**(ptr_dest, ptr_src, len)
Description :
Copie un espace mémoire sur *len* octets de *src* vers *dest*. En plus de la fonction **strcpy**(), **memmove**() permet d'avoir les espaces mémoires pointés par *src* et *dest* qui se chevauchent. Cette fonction retourne *dest*.

Copies a block of *len* bytes from *src* to *dest*, with possibility of overlaping of source and destination block. **memmove**() copies a block of *len* bytes from *src* to *dest*. Even when the source and destination blocks overlap, bytes in the overlapping locations are copied correctly (in opposite to memcpy). memmove returns *dest*.

Fonction : ptr memchr()
Numéro de la fonction : 0d
Syntaxe : **memchr**(*str,c,len*)
Description :

Cherche dans les *len* premiers octets du bloc mémoire pointé par *str*, le pointeur de la première occurrence du caractère *c*. Renvoi ce pointeur. Si **memchr**() n'a pas trouvé le caractère *c*, alors renvoi 0.

Searches the first *len* bytes of array *str* for character *c*. **memchr**() searches the first *len* bytes of the block pointed to by *str* for character *c*. On success, **memchr**() returns a pointer to the first occurrence of *c* in *str*. Otherwise, it returns 0.

Example :Will display 4 to the screen, corresponding to the position-1 of the character "5" in the string.

```
init()
```

clrscr()
"123456789" → str
Pause “Position (-1) of 5 in the string:“&string(memchr(str,ord("5"),strlen(str))-str)

Fonction : ptr seql
Numéro de la fonction : 0d
Syntaxe : **seql**(*var,start,end,step,var_dest,instruction*)
Description :
Alloue un espace mémoire de (end-start+1)*4 octets et affecte chacun des éléments codés sur quatre octets de cet espace mémoire par la valeur de l'exécution de l'instruction *instruction* et cela pour une valeur de la variable *var* allant de *start* vers *end* avec un pas de *step*. Cette fonction retourne le pointeur sur l'espace mémoire alloué et stocke aussi sa valeur dans la variable *var_dest*.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()). Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (**free**()).

Allocate a block of memory of (end-start+1)*4 bytes and fill this space with longs integers which is in fact the results of the executions of the instruction *instruction* for each value of *var* between *start* and *end* (with the step *step*). The pointer is store in the Newprog variable var_dest. The function returns this pointer. You will have to free the allocated memory block before exiting the program (**free**()).
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.
Example :
```
:init()
:seql(va,1,4,1,list,va)
:0 → vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
```

Fonction : ptr seqw
Numéro de la fonction : 0d
Syntaxe : **seqw**(*var,start,end,step,var_dest,instruction*)
Description :
Alloue un espace mémoire de (end-start+1)*2 octets et affecte chacun des éléments codés sur deux octets de cet espace mémoire par la valeur de l'exécution de l'instruction *instruction* et cela pour une valeur de la variable *var* allant de *start* vers *end* avec un pas de *step*.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()). Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (**free**()).

Allocate a block of memory of (end-start+1)*2 bytes and fill this space with words (ie 2 bytes length) which is in fact the results of the executions of the instruction *instruction* for each value of *var* between *start* and *end* (with the step *step*).The pointer is store in the Newprog variable *var_dest*. The function returns this pointer. You will have to free the allocated memory block before exiting the program (**free**()).
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.
Example :
```
:init()
:seqw(va,1,4,1,list,va)
:0 → vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
```

Fonction : ptr seqb
Numéro de la fonction : 0d
Syntaxe : **seqb**(*var,start,end,step,var_dest,instruction*)
Description :
Alloue un espace mémoire de (end-start+1)*1 octets et affecte chacun des éléments codés sur un octet de cet espace mémoire par la valeur de l'exécution de l'instruction *instruction* et cela pour une valeur de la variable *var* allant de *start* vers *end* avec un pas de *step*. Cette fonction retourne le pointeur sur l'espace mémoire alloué et stocke aussi sa valeur dans la variable *var_dest*.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()). Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (**free**()).

Allocate a block of memory of (end-start+1)*1 bytes and fill this space with byte which is in fact the results of the executions of the instruction *instruction* for each value of *var* between *start* and *end* (with the step *step*). The pointer is store in the Newprog variable *var_dest.* The function returns this pointer. You will have to free the allocated memory block before exiting the program (**free**()).
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.
Example :
```
:init()
:seqb(va,1,4,1,list,va)
:0 → vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
```

Fonction : ptr group
Numéro de la fonction : 0d
Syntaxe : **group**(*nilist*)
Description :
Cette fonction transforme une niliste en liste instanciée avec éléments codés sur 4 octets (bytes). Cette fonction retourne un pointeur sur un bloc mémoire. Ce bloc mémoire contient en première position le nombre d'éléments convertis puis contient à la suite les éléments instanciés de la niliste. Pour lire ou écrire dans cette liste, il faudra utiliser les fonctions **ll**() et **wl**() voire **polel**() et **peek**()l mais l'utilisation des **[]** est plus commode. Vous devez libérer l'espace mémoire en utilisant la fonction free avant l'arrêt du programme.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

This function returns a pointer to an allocated memory block initialised with long number (4 bytes) (load and write with **ll**() and **wl**() or with the **[]** syntax). The first number of the allocated memory is the number of elements of the *nilist*, next are placed the evaluations of the elements of the nilist. You must free the returned pointer before the end of the program to avoid memory space problems (**free**()).
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function **disperr**()), an error dialog box will appear.
Exemple :
```
:aa()
:Prgm
:group({1," Hello ",ord("c")})->temp
:clrscr()
:printf2("nb arg=%ld arg1=%ld",temp[0],temp[1])
:prints(temp[2])
:printc(temp[3])
:keywait()
:free(temp)
:EndPrgm
```
The result of the program will be : nb arg=3 arg1=1 Hello c

Fonction : ptr newstr
Numéro de la fonction : 0d
Syntaxe : **newstr**(*str,var_dest*)
Description :
Alloue un espace mémoire de la taille de la chaîne de caractères *str* plus 1 (pour le caractère nul), puis copie tous les caractères de *str* dans cet espace mémoire. La variable newprog *var_dest* est alors affectée à ce pointeur. Ce bloc mémoire doit être libérer avec la fonction **free**() avant la fin de l'exécution du programme sous peine de perdre définitivement de la mémoire RAM disponible (un reset est alors nécessaire pour la récupérer). La fonction retourne le pointeur de cet espace mémoire alloué.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Allocate a block of memory of the length of the string str + 1 (for '\0') and copie all the bytes of str in the new memory block. Copy in the newprog variable the pointer of the new memory block and also return it. You must delete this block before exiting the program (using **free**()).
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function **disperr**()), an error dialog box will appear.
Example :
```
:init()
:clrscr()
:"Hello"->str
:newstr(str,va)
:prints(va)
:keywait()
:free(va)
```

Fonction : value settype
Numéro de la fonction :
Syntaxe : **settype**(*var*,*data_size*)
Description :
Cette fonction est intéressante. D'une certaine manière, elle peut faciliter la programmation.
A utiliser avec la notation []. Sélectionne le type de donnée pointée par la variable newprog *var* (en définissant leur taille *data_size* exprimée en octets). La donnée d'entrée *data_size* peut prendre les valeurs suivantes :

- **1** : *var* pointe sur un tableau de données codée sur 1 octet (= 1byte). (Lecture : var[ii] ; écriture : data_to_store → var[ii])
- **2** : *var* pointe sur un tableau de donnée codée sur 2 octets. (Lecture : var[ii] ; écriture : data_to_store → var[ii])
- **4** : *var* pointe sur un tableau de donnée codée sur 4 octets. (Lecture : var[ii] ; écriture : data_to_store → var[ii])
- **autres valeurs** : *var* pointe sur un tableau de chaine de caractères. Chaque chaine de caractères doit avoir une taille maximale de *data_size*-1 (sans compter le zéro de fin de chaine de caractères). Ou dit autrement, l'espace mémoire de chaque élément est de *data_size* octets. Pour utiliser les listes de retours des fonctions **reps**() et **files**(), *data_size* doit être égale à 10. (Lecture : var[ii] ; écriture : pointeur_data_à_copier → var[ii] (pour information, effectue en interne un **memcpy**())).
- **0** : Cas particulier : renvoi seulement le type de donnée actuel de *var* sans le modifier.

Les fonctions **seqb**(),**seqw**(),**seql**() affecte automatiquement la taille des éléments à la variable de sortie lors de leurs exécutions.

This is an interresting function. It allows the user to facilitate the programmation.
Settype() is intended to be used with [] notation. The user select the size in bytes (*data_size*) of the data on which the newprog variable *var* points (if *var* contains a pointer ; for example the value 0h4c00 which is the pointer to the lcd address). *data_size* can have the following value :

- **1** : var points on a list of datas, each one written onto 1 byte. (Reading : var[ii] ; writing : data_to_store → var[ii])
- **2** : var points on a list of datas, each one written onto 2 bytes. (Reading : var[ii] ; writing : data_to_store → var[ii])
- **4** : var points on a list of datas, each one written onto 4 bytes. (Reading : var[ii] ; writing : data_to_store → var[ii])
- **other values** : *var* points on a list of strings. Each strings must have a maximal size of *data_size*-1 (value returns by **strlen**(), because of not taking acount on the null last byte of the string). Or, say in an another way, the memory area of each element is of *data_size* bytes. (Note :the function **reps**() et **files**() returns lists which each element have 10 bytes long, so in this case *data_size*=10). Reading : var[ii] ; Wrinting : pointer_to_the_datas_to_copy → var[ii] (for information, perform internally a **memcpy**()).
- **0** : Particular case : returns only the current data size of *var* without modifying it

Functions **seqb()**,**seqw()**,**seql()** sets automatically the size of the elements of the output list according to their type.

Example:

```
init()
malloc(lcdsize*2)→ptr
i(ptr=0):prints("mem error"):finish():y
settype(ptr,lcdsize)
getlcd()->lcd
lcd → ptr[0]
moveto(0,0)
prints("Screen saved in ptr[0]"):nl()
keywait()
clrscr()
memset(ptr[1],1,lcdsize)
settype(lcd,lcdsize)
ptr[1]→lcd[0]
prints("Screen filled"):nl()
keywait()
ptr[0]→lcd[0]
moveto(0,0)
prints("Screen restored")
keywait()
free(ptr)
```

Fonction : long lb
Numéro de la fonction : 0d165
Syntaxe : **lb**(*byte_list_address,elem*)
Description :

Retourne la valeur du *elem* élément de la liste d'octets (8 bits) pointée par *byte_list_address*. Le premier élément d'une liste est l'élément 0. La valeur retournée est de type signée. Par exemple, si l'élément de la liste =255, il sera interprété comme égal à -1 et non pas égal à 255. Pour l'interpréter comme une valeur non signée (255 pour l'exemple précédent), utiliser la fonction **unsigb**().

Return the value of the n-*elem* element of the 8 bits array (pointed by *byte_list_address*). The first element of a list is the number 0. The return value is assumed to be a signed value. For instance, if the element *elem* of the list *yte_list_address* is equal to 255, so the return value will be -1, and not 255. If you want to retrieve a non signed value (255 for the previous example), you can use the **unsigb**() function.

Example :Will display 1,-2,-1,254,255

```
init()
b(list)
1
2
255
y
clrscr()
printld(lb(list,0)):nl()
printld(lb(list,1)):nl()
printld(lb(list,2)):nl()
printld(unsigb(lb(list,1))):nl()
printld(unsigb(lb(list,2))):nl()
keywait()
```

Fonction : long lw
Numéro de la fonction : 0d166
Syntaxe : **lw**(*word_list_address,elem*)
Description :
Retourne la valeur du *elem* élément de la liste de mots (16 bits) pointée par word_*list_address*. Le premier élément d'une liste est l'élément 0. La valeur retournée est de type signée. Par exemple, si l'élément de la liste =65535, il sera interprété comme égal à -1 et non pas égal à 65535. Pour l'interpréter comme une valeur non signée (65535 pour l'exemple précédent), utiliser la fonction **unsigw**(). Voir l'exemple de **lb**().

Return the value of the n-*elem* element of the 16 bits array (pointed by *byte_list_address*). The first element of a list is the number 0. The return value is assumed to be a signed value. For instance, if the element *elem* of the list word_*list_address* is equal to 65535, so the return value will be -1, and not 65535. If you want to retrieve a non signed value (65535 for the previous example), you can use the **unsigw**() function. See the example of **lb**().

Fonction : long ll
Numéro de la fonction : 0d167
Syntaxe : **ll**(*long_list_address,elem*)
Description :
Retourne la valeur du *elem* élément de la liste de longs (32 bits) pointée par *long_list_address*. Le premier élément d'une liste est l'élément 0. La valeur retournée est de type signée.

Return the value of the n-*elem* element of the 32 bits array (pointed by *byte_list_address*). The first element of a list is the number 0. The return value is assumed to be a signed value.

Example:Will display "4" to the screen.

```
init()
seql(vv,1,4,1,dd,vv)
clrscr()
printld(ll(dd,3))
keywait()
free(dd)
```

Fonction : long wb
Numéro de la fonction : 0d165
Syntaxe : **wb**(*byte_list_address,elem,val*)
Description :
Write the value *val* to the element *elem* of the byte list pointed by *byte_list*.

Fonction : long ww
Numéro de la fonction : 0d166
Syntaxe : **ww**(word_*list_address,elem,val*)
Description :
Write the value *val* to the element *elem* of the word list pointed by *word_list*.

Fonction : long wl
Numéro de la fonction : 0d167
Syntaxe : **wl**(*long_list_address,elem,val*)
Description :
Write the value *val* to the element *elem* of the long list pointed by *long_list*.

Fonction : long pokeb
Numéro de la fonction : 0d190
Syntaxe : **pokeb**(*adress,value*)
Description :
Ecrit la valeur *value* sur 8 bits à l'adresse *adress*.

Write the value *value* along 8 bits to the adress *adress*.

Fonction : long pokew
Numéro de la fonction : 0d191
Syntaxe : **pokew**(*adress,value*)
Description :
Ecrit la valeur *value* sur 16 bits à l'adresse *adress*.

Write the value *value* along 16 bits to the adress *adress*.

Fonction : long pokel
Numéro de la fonction : 0d192
Syntaxe : **pokel**(*adress,value*)
Description :
Ecrit la valeur *value* sur 32 bits à l'adresse *adress*.

Write the value *value* along 32 bits to the adress *adress*.

Fonction : long peekb
Numéro de la fonction : 0d200
Syntaxe : **peekb**(*address*)
Description :
Renvoie la valeur de l'octet (8 bits) situé a l'adresse *adress*.

Return the 8 bits value placed to the adress *adress*.

Fonction : long peekw
Numéro de la fonction : 0d201
Syntaxe : **peekw**(*adress*)
Description :
Renvoie la valeur du mot (16 bits) situé a l'adresse *adress*.

Return the 16 bits value placed to the adress *adress*.

Fonction : long peekl
Numéro de la fonction : 0d202
Syntaxe : **peekl**(*adress*)
Description :
Renvoie la valeur du nombre long (32 bits) situé a l'adresse *adress*.

Return the 32 bits value placed to the adress *adress*.

3 Fonctions sauts – Jumps functions

Fonction : finish
Numéro de la fonction : 0d22
Syntaxe : **finish**()
Description :
Quitte le programme en cours pour retourner au tios.

Exit form the program to the home screen. Stop running the program.

Fonction : - while -endwhile
Numéro de la fonction : 0d61 0d21
Syntaxe : **while** *condition* :… :**endwhile**
Description :
Si condition est vraie, alors la boucle while n'est pas sautée aussi non elle le sera. Même fonctionnalité qu'en Tibasic.

If condition is true, so the while loop will not be jump. If not, she will.

Fonction : - for-endfor
Numéro de la fonction :
Syntaxe : **for** *varname ,begin ,end step* :*instructions*:**endfor**
Description :
Même fonctionnalité qu'en Tibasic. Les fonctions associées **break**() et **next**() apportent des fonctionnalités supplémentaire aux boucles For:Endfor.

Is the same as the Tibasic function. The associated functions **break**() and **next**() adds functionality to this kind of loop.

Fonction : - break
Numéro de la fonction :
Syntaxe : **break**()
Description :

Saute à la fin de la boucle For:Endfor ou While:Endwhile activée en dernier. Cela permet de quitter une boucle manuellement.

Jump to the end of the last activated loop For:Endfor or While:Endwhile. It allows you to end manually the execution of a loop.

Example :

```
init()
clrscr()
for vv,1,10,1
prints("vv="&string(vv))
nl()
if vv=5 then
break()
endif
endfor
keywait()
```

Fonction : - next
Numéro de la fonction :
Syntaxe : **next**()
Description :
Saute au début de la dernière boucle For ou While active. Si il s'agit d'une boucle For, la variable sera affectée a sa prochaine valeur.

Jumps to the beginning of the last active For or While loop. If the loop is a For:Endfor, the variable will be affected to its new value.

Example :

```
()
Prgm
init()
clrscr()
For ii,1,5,1
For vv,1,2,1
next()
prints("Will not be printed")
nl()
EndFor
prints("Will be printed, ii="&string(ii))
nl()
EndFor
keywait()
EndPrgm
```

Fonction : - if then -else - endif
Numéro de la fonction : 0d9 0d79

Syntaxe : **if** *condition* **then:...:else:...:endif**
Description :
Même syntaxe que la fonction du tios. Si *condition* est vraie, c'est à dire différent de 0, alors la séquence après then sera exécutée, sinon elle sera passée et ce sera alors la 2eme séquence après else qui sera exécutée. Dans cette fonction, le bloc else n'est pas obligatoire.

Same syntax and functionality than the tibasic function.

Fonction : - if:
Numéro de la fonction :
Syntaxe : **if** *condition***:***instruction_if_true*
Description :
Même syntaxe et fonctionnalité que la fonction du tios. Si *condition* est vraie, c'est à dire différent de 0, alors l'instruction *instruction_if_true* sera exécutée, sinon elle sera passée.

Same syntax and functionality than the Tibasic function.

Fonction : - I - y
Numéro de la fonction :
Syntaxe : **i(***condition):instructions:**y***
Description :
Cette fonction est similaire à un **if endif** classique. Si *condition* est vraie, alors les instructions *instructions* seront exécutées. Si condition est fausse, passera les instructions sans les exécuter.
Cette fonction à un intérêt car elle est plus concise et plus simple à écrire qu'un **If-Endif** classique. Autre intérêt mineur, on peut insérer un **i():y** dans des nilistes sans que le compilateur affiche un message d'erreur à la compilation contrairement au If-Endif classique.

This function is similar to the **if endif** of the Tibasic. Si condition est vraie, alors les instructions instructions will be executed. If condition is false, this function will pass the instructions without execute them.
This function is interesting because is simplier to write than a **if endif**. An another advantage is it can be insert in a nilist without throwing an error during the compilation.

Example :

Will display "executed"
init()
clrscr()
"Not executed"->variable
{1,2,3,i(1=1),sto(variable,"executed"),y,5} → ni
#ni
prints(variable)
keywait()

Fonction : long map
Numéro de la fonction : 0d220
Syntaxe : **map**(*var,start,end,step,instruction*)
Description :
Exécute instruction en faisant varier *var* de *start* à *end* avec un pas de *step*. Cette instruction est plus rapide qu'une boucle While classique ainsi que les boucles For.
Retourne la dernière valeur retournée par *instruction*.

Execute instruction by making *var* going from *start* to *end* by a step *step*. This function is faster than a while:endwhile and For:Endfor loops.
Return the last result of instruction (for *var=end*)

Fonction : long multi
Numéro de la fonction : 0d221
Syntaxe : **multi**(*num,instruction1,instruction2,...*)
Description :
Exécute *num* instructions *instruction1,instruction2*...en une seule instruction. Attention, *num* doit être égal à la somme des instructions !
Retourne la dernière valeur retournée. Il est préférable d'utiliser les blocs x:y car plus simple à utiliser et moins source d'erreur.

Execute *num* instructions *instruction1,instruction2*… by considering that it is only one instruction. Be careful, *num* must be equal to the sum of the instructions ! You should prefer use the x:y block because simpler to use and more safe.
Return the last value returned.
Example :
:init()
:0->va
:clrscr()
:map(var,1,10,1,multi(2,isz(va),printld(va)))
:keywait()

Fonction : long x - y
Numéro de la fonction : 0d222
Syntaxe : **x***:instructions:***y**
Description :
Execute the instructions *instructions* placed between x and y tag. This x:-:y instruction is considered to be part of only one instruction. This method is simplier to use than **multi**() and is as fast. Returns the value of the last instruction executed.
Execute les instructions *instructions* placées entre **x** et **y**. Cette fonction permet d'exécuter plusieurs instructions en une seule. Cette écriture est plus simple que **multi**(). Retourne la valeur retournée par la dernière instruction exécutée.

Example :
Displays black sprite to the screen and display the amout of displayed sprites. You can write this example with For:Endfor but the prgm below is about 3 time faster when using map and x:y.

```
init()
clrscr()
0 → nbdisp
seqb(vv,1,8,1,sprt,-1)
map(xx,0,10*8,8,x)
map(yy,0,8*8,8,x)
sprt8(xx,yy,8,sprt)
isz(nbdisp)
y
y
printf1(«Amount of displayed sprites : %ld»,nbdisp)
keywait()
free(sprt)
```

Fonction : long two
Numéro de la fonction : 0d222
Syntaxe : **two**(*i1*,*i2*)
Description :
Exécute les instruction i1 et i2 (en une seule instruction). Assez similaire à **multi**() mais pour seulement 2 instructions (optimisé pour deux instruction contrairement à **multi()**).
Renvoie la valeur retournée par i2.

Execute the instructions *i1* and *i2* in only one instruction. Quite similar to **multi**() but for only two instructions (optimized).
Return the value returned by *i2*.

Fonction : - lbl
Numéro de la fonction : 0d4
Syntaxe : **lbl** *label_name*
Description :
Cette fonction a le même rôle que la fonction du tios. Elle permet de placer une étiquette du nom de *label_name*. Elle est a utiliser avec la fonction **goto**. Comme les noms de varialbes NewProg, le nom des labels *label_name* doivent être d'au moins 2 caractères.

This function has the same properties than the tios function. It allows you to put a label *label_name* is the program. Use it with the **goto** function. As the NewProg data vars, the name of the label *label_name* must be written with 2 letters minimum.

Fonction : - goto
Numéro de la fonction : 0d5
Syntaxe : **goto** *label_name*
Description :
Cette fonction a le même rôle que la fonction du tios. Elle permet d'effectuer un saut vers l'étiquette portant le même nom que *label_name*. A utiliser avec la fonction **lbl**.

This function has the same properties than the tios function. It allow you to make a jump to the *label_name* position in the program.

Fonction : - jsr
Numéro de la fonction : 0d82
Syntaxe : **jsr**(*nom_label*)
Description :
Saute vers la sous routine se trouvant à la position *nom_label*.

Makes a jump to the sub routine which is placed at the position *nom_label.*
Example :

```
:init()
:0 → x
:While x<10
```

```
:isz(x)
:jsr(abc)
:Endwhile
:pause «abc launched »
:finish()
:lbl abc
:printf1(« In abc ! x=%ld »,x)
:rts()
```

Fonction : - rts
Numéro de la fonction : 0d83
Syntaxe : **rts**()
Description :
Effectue le retour de la routine appelée par **jsr** vers la routine principale.
Regarder l'exemple précédent.

Perform the return to the main routine called by **jsr** to the main routine.
Look to the exemple of the **jsr** function.

Fonction : value fc(*function_name,arg1,arg2,...*) or more usually *function_name(arg1,arg2,...)*; and also def , enddef, arg, rtn
Numéro de la fonction : 0d
Syntaxe : **fc**() (or ( )), **rtn**()
Description :

Execute la fonction *function_name* (définie grâce aux fonctions **def**() et **enddef**(), obligatoirement en amont du programme). Jusqu'à 250 arguments peuvent être transmis à la fonction (arg1, arg2 etc).
La fonction **rtn**() permet de fixer la valeur de retour à la fin de la fonction. Toute fonction doit être définie avant sa première exécution grâce à **def**() et **enddef**(). Pour cela, il est recommandé de placer les définitions de fonctions au début du programme.

Execute the inline function *function_name* (ie define with the **def**() function). The argument *arg* will be accessible in the function with the **arg**() function. The function **rtn**() allow you to fix the return value to the end of the execution of the function. All function must be define before before its first execution with the **def**() **enddef**() function. So it is recommended to place the definition at the beginning of the program. It is possible to make recursives functions ! (30 subcalls max). See below to see how to make locals variables.

```
Prgm
init()
clrscr()
"And have a good day !" → str
def(hello(str))
rtn(str&" ")
enddef(hello)
def(world())
rtn("World")
enddef(world)
Pause hello("Hello")&world()
Pause str
EndPrgm
```

This program will print on the screen : Hello World !

Fonction : - def & enddef
Numéro de la fonction : 0d
Syntaxe : **def**(*function_name*(*arg_name1,arg_name2,...*)):instruction1:instruction2:etc...:**enddef**(*function_name()*)
Description :
Définie une fonction personnalisée. Les arguments passés à la fonction seront sauvegardés dans les variables locales *arg_name1, arg_name2*, etc (le nombre d'argument passé à la fonction peut être nul).
Une variable locale peut avoir le même nom qu'une variable déjà définie dans le programme. Si c'est le cas, l'ancienne valeur est sauvegarder puis restaurer à la fin de l'exécution de la fonction (en atteignant la fin de la fonction (**enddef**()) ou en utilisant la fonction **rtn**()). L'utilisateur peut modifier sa valeur à l'intérieur de la fonction comme une variable ordinaire. Sa valeur initiale étant égale à l'argument passé à la fonction (fonction **fc**()).

Define a custom function. The arguments passed to the function will be saved in the locals variables *arg_name1, arg_name2*, etc (the number of arguments passed to the function can be null).
A local variable can have the same name than a variable previously defined in the program. In this case, the former value is saved then restored at the end of the execution of the function (by reaching **enddef**() function or by executing **rtn**()). The programmer can modify its value inside the function like a classic variable. Its initial value is equal to the argument value passed to the function (function **fc**()).

Fonction : - rtn
Numéro de la fonction : 0d
Syntaxe : **rtn**(*value*)
Description :
Placée à l'intérieur d'une fonction (entre **def**() et **enddef**()), termine l' exécution de cette fonction. La valeur retournée par le fonction sera égale à *value*.

Placed inside a function (between **def**() and **enddef**()), ends the execution of this function. The return value of this function will be equal to *value*.

Fonction : value when
Numéro de la fonction : 0d23
Syntaxe : **when**(*cond,{instruction1,instruction2…},{instructiona,instructionb…}*)
Description :
Fonction utile car elle simplifie l'écriture des conditions if then else. Si *cond* est vraie (i.e. différent de 0) alors *instruction1,instruction2* seront exécutée, si *cond* est fausse, ce sera *instructiona,instructionb. When* retourne la valeur de la dernière instruction exécutée. Elle est assez similaire à la fonction du tibasic When() mais en plus puissante car avec plusieurs instructions possibles. Néanmoins, quand il y a beaucoup d'instructions ou que le maximum de vitesse est demandée, il vaut mieux utiliser **if then else endif** (50% plus rapide)**.**

Useful function, if *cond* is not null (i.e true) so the *instruction1,instruction2...* will be executed. Else, it will be the *instructiona,instructionb...* **When** return the result of the last instruction executed. This fuction is quite similar with the when() function of the tios but accepts more instructions.
You will prefer to use the classic **if then else endif** functions if you have a lot of instructions to launch and want to have max of speed (50% faster).

Example : This program will print *True2* to the screen because first arg *1* is true and *True2* is the last instruction of true instuctions section.

```
:init()
:clrscr()
:pause when(1,{"True1","True2"),{"False1"})
```

Fonction : - gotopos
Numéro de la fonction : 0d80
Syntaxe : **gotopos**(*position*)
Description :
Effectue un saut vers la position *position*. La variable *position* doit avoir une valeur valide pour ne pas crasher le programme. Elle doit *être* obtenue par la fonction **pos.** Cette fonction à l'avantage de pouvoir effectuer des sauts dynamiques dans le programme.
Dans la plupart des cas, l'utilisation des fonctions natives **lbl** et **goto** sont amplement suffisante.

Make a jump to the position *position* in the program. The variable *position* must be valid to prevent crashs at execution. *Position* must be obtain by the **pos** function. This function has the advantage to make dynamics jumps in the program. You will prefer to use the standarts **lbl** and **goto** functions.
Exemple :

```
:init()
:lbl abc
:goto abc
:basic
```

Est équivalent à – Is the same than :

```
:init()
:pos() → position
:gotopos(position)
```

Fonction : ptr pos
Numéro de la fonction : 0d81
Syntaxe : **pos**()
Description :
Renvoie la position dans le programme juste après la position de cette fonction.
Il faut préférer l'utilisation des fonction **lbl** et **goto** à l'utilisation de cette fonction.
Cette fonction peut être utile pour un objectif de déboguage pour personnes très expérimentées en Newprog.

Returns the position in the program just after the position of this function (**pos**). To use only if you are very experimentated with Newprog.

4 Fonctions affichage de texte – Text printing functions

Fonction : clrscr
Numéro de la fonction : 0d100
Syntaxe : **clrscr**()
Description :
Efface l'écran et reset le curseur de texte (fonctions **prints**…).
Clear screen and reset the text cursor (use with **printf1**, **prints**...).

Fonction : - printc
Numéro de la fonction : 0d25
Syntaxe : **printc**(*value*)
Description :
Affiche à l'écran le caractère représenté par la valeur ASCII *value*.

Print to the screen the chars corresponding to ASCII code *value*. Increment cursor position.

Fonction : - prints
Numéro de la fonction : 0d18
Syntaxe : **prints**(*string_ptr*)
Description :
Affiche à l'écran la string pointée par *string_ptr*.

Print to the screen the string pointed by *string_ptr*. Increment cursor position.
Exemple :
```
:init()
: « Hello ! »->str
:clrscr()
:prints(str)
:keywait()
:prints("hello2")
```
You will see : Hello !hello2

Fonction : - printld
Numéro de la fonction : 0d
Syntaxe : **printld**(*value*)
Description :
Print to the screen the value *value*. Increment cursor position.
Exemple :
```
:init()
: 99->val
:clrscr()
:printld(val)
:keywait()
```

Fonction : long setfont
Numéro de la fonction : 0d76
Syntaxe : **setfont**(*font*)
Description :

Change la taille des caractères affichés à la valeur *font*. *Font* peut prendre les valeurs 0 (petit),1 (moyen) ou 2 (grand). Retoune l'ancienne valeur de la taille.

Sets the current font**. setfont()** changes the current text font. All subsequent characters written to the screen will use this font. The supported values for *font* are 0,1,2 (F_4x6, F_6x8, and F_8x10). The 4x6 font is a proportional font while the 6x8 and 8x10 fonts are fixed-width. FontSetSys returns the previously active font number.

Fonction : - newline or nl
Numéro de la fonction : 0d17
Syntaxe : **newline**() or **nl**()
Description :
Go to a new line. The screen will been scrolled if necessary. To be used with printld(), prints(), printf1(), printf2() functions.

Fonctions : ptr printf1**and** ptr printf2
Numéro de la fonction : 0d15
Syntaxe : **printf1**(*string_ptr,arg*) - **printf2**(*string_ptr,arg1,arg2*)
Description :
Lire les informations suivantes. Ces fonctions se limitent respectivement à 1 et 2 arguments seulement. Elles sont très puissantes. A noter que dans NewProg, toutes les variables internes sont codées sur 4 octets, il faudra donc utiliser %ld pour afficher le contenu d'une variable. Ces fonctions sont dérivées de la fonction printf du C. Des exemples sont données plus bas.
L'explication est en anglais, ce serait trop long à traduire en francais, donc bonne lecture :

printf is nearly full implementation of standard ANSI C printf function, which sends the formatted output to the screen in terminal (TTY) mode. In fact, it does the following:

- accepts a series of arguments;
- applies to each a format specifier contained in the format string pointed to by *format*;
- outputs the formatted data to the screen.

The printed text will wrap on the right end of the screen. Characters '\n' will be translated to "next line" (and this is the only control code which has a special implementation). The screen will scroll upwards when necessary (i.e. after printing a text in the last screen line). Note that all TI fonts are supported. Of course, printf will update current "print position" to a new one after the text is printed.

printf applies the first format specifier to the first argument after *format*, the second to the second, and so on. The format string, controls how printf will convert and format its arguments. There must be enough arguments for the format; if there are not, the results will be unpredictable and likely disastrous. Excess arguments (more than required by the format) are merely ignored. The format string is a character string that contains two types of objects: plain characters and conversion specifications. Plain characters are simply copied verbatim to the output string. Conversion specifications fetch arguments from the argument list and apply formatting to them. printf format specifiers have the following form:

% [flags] [width] [**.**prec] [{h|l}] type

Here is a complete table of supported formatting options (see any book about C language for more info):

Flags	**Meaning**
none	Right align (pad spaces or zeros to left)
-	Left align (pad spaces to right)
+	Always force sign (include prefix '+' before positive values)
z	Don't postfix padding (this option is non-ANSI, i.e. TI specific)
space	Insert space before positive values
#	Prefix octal values with 0 and hex values (>0) with '0x') Force '**.**' in float output (and prevent trunctation of trailing zeros)
^	TI-Float format: special character for the exponent and for the minus sign, no '+' prefix in the exponent, 0. instead of 0, no leading zeros if the magnitude is smaller than 1 (this option is non-ANSI, i.e. TI specific)
|	Center the output in the field (this option is non-ANSI, i.e. TI specific)

Width	**Meaning**
num	Print at least *num* characters - padded the rest with blanks
0*num*	(Zero prefixed) Same as above but padded with '0'
*	The width is specified in the arguments list (before value being formatted)

Precision	**Meaning**
none	Default precision

num	*num* is number of chars, decimal places, or number of significant digits (*num*<=16) to display depending on type (see below)
-1	Default = 6 digits (this option is non-ANSI, i.e. TI specific)
*	The precision is specified in the argument list (before value being formatted)

Size {h|l}	**Meaning**
h	Force short integer
l	Force long integer

Type	**Meaning**
d, i	Signed decimal integer
u	Unsigned decimal integer
x	Lowercase hexadecimal integer
X	Uppercase hexadecimal integer
e	Floating point, format [-]*d.dddd*e[sign]*ddd* (exponential format)
E	Like 'e' but with uppercase letter for the exponent
f	Floating point, format [-]*dddd.dddd*
g	Floating point: most compact float format available ('e' or 'f'); this is the most common option, used for most dialog floats
G	Like 'g' but with uppercase letter for the exponent
r	Floating point, engineering form (this option is non-ANSI, i.e. TI specific)
R	Like 'r' but with uppercase letter for the exponent
y	Floating point, mode specified float format (this option is non-ANSI, i.e. TI specific)
Y	Like 'y' but with uppercase letter for the exponent
c	Character
s	String
p	Pointer; principally the same as 'x' - do not use without 'l' modifier
%	None; the character '%' is printed instead

Examples :
```
:init()
:clrscr()
:printf1("One hundred :%ld",100)
:printf2("%s %ld","Two hundreds",200)
:keywait()
```
You will see print on the screen : One hundred :100Two hundreds 200

Fonction : - printxy
Numéro de la fonction : 0d
Syntaxe : **printxy**(*x,y,format,arg*)
Description :

Sends formatted output to the fixed place on the screen.

Printxy() is similar to the standard ANSI C printf function, except:

- this function displays formatted output to the screen at the strictly specified position, more precise, starting from the point (*x*, *y*);
- text printed with printf_xy will not wrap at the right end of the screen (if the text is longer, the result is unpredictable);
- characters '\n' will not be translated to "new line";
- this function will never cause screen scrolling;
- current print/plot position remains intact after executing this function.

Exemple :
```
:init()
:clrscr()
:printxy(20,10,"%s","Hello World !")
:keywait()
```

Fonction : value cwidth
Numéro de la fonction : 0d
Syntaxe : **cwidth**(*char_ascii_code*)
Description :
Renvoi la largeur en pixels d'un caractère ayant pour code ascii *char_ascii_code* suivant la taille de caractère active (fonction **setfont**()).

Returns the width in pixel of the character *char_ascii_code according* to the current font settings (**setfont**() function).

Fonction : - moveto
Numéro de la fonction : 0d
Syntaxe : **moveto**(xx,yy)
Description :
moveto() affecte la position actuel du curseur de position (des fonctions **printf1**(), **prints**(),etc) à la positon (*xx*, *yy*).

moveto() sets the current pen position to (*xx*, *yy*) (for **printf1**(), **prints**(),etc).

Fonction : value pause
Numéro de la fonction : 0d
Syntaxe : **pause**(*msg_ptr*) or **pause** *msg_ptr*
Description :
Affiche la chaîne de caractère *msg_ptr* sur une nouvelle ligne à l'écran. Attend l'appui d'une touche. Retourne la position du texte sur une nouvelle ligne (**newline**()) Renvoie le numéro de la touche appuyée.

Output the string *msg_ptr* in a new line to the screen. Wait for a key pressed. Returns the pen position to a new line (**newline**()). Returns the code of the key pressed.

Example :
```
init()
clrscr()
20->myage
pause «I am « &string(myage)
```

Fonction : value text
Numéro de la fonction : 0d
Syntaxe : **text**(*str*) or **text** *str*
Description :
Fonction identique à la fonction Tibasic **text().** Réalise un appel en interne de la fonction **keydisp**().

Same function than the Tibasic one. Will call automatically the **keydisp**() function.

5 Fonctions pour chaîne de caractères – Strings functions

Fonction : ptr char
Numéro de la fonction : 0d27
Syntaxe : **char**(*char_value*)
Description :
Renvoie un pointeur sur une chaîne de caractères ayant comme seul composante un caractère correspondant à *char_value*. La valeur de retour sera valable que si cette fonction n'est pas appelée plus de trois fois après celle-ci. Donc penser à sauvegarder le résultat si nécessaire.

Returns a pointer on a string containing a chars corresponding to the ASCII code *char_value*. The return value will be available up to the next 3 calls of **char**(). So it can be necessary to store the result.

Fonction : value ord
Numéro de la fonction : 0d27
Syntaxe : **ord**(*char_str*)
Description :
Renvoi le code ASCII du caractère présent dans la chaîne de caractères *char_str*. Fonction similaire à la fonction Tibasic de même nom.

Returns the ASCII code of the character in the string *char_str*. Same functionality than the ord() function in Tibasic.

Fonction : ptr string
Numéro de la fonction : 0d
Syntaxe : **string**(*value*)
Description :
Convertit un nombre *value* en chaine de caractères. Retourne le pointeur de cette chaine de caractères. Cette fonction utilise en interne la fonction **sprintf**(). La valeur de retour de cette fonction sera valable jusqu'au quatrième prochain appel de cette fonction donc penser à faire une copie si nécessaire (avec **newstr**() par exemple).

Converts a number with the value *value* into a string. Returns the pointer of the string result. This function is made internally with the **sprintf**() function. The return value will be available (before self erasing) for only 4 calls of this function so make a copy if necessary (with **newstr**() for example).

Example :
```
init()
clrscr()
prints("I am "&string(20)&" years old.")
keywait()
```

Fonction : strcat
Numéro de la fonction : 0d28
Syntaxe : **strcat**(*dest,src*)
Description :

Ralonge la chaîne de caractères pointée par dest avec la chaîne de caractères pointée par *src*. Veiller à ce que la taille de l'espace mémoire pointé par *dest* soit suffisament grand pour pouvoir accueillir la chaîne de caractères *src* . La fonction renvoie *dest*.

Appends *src* to *dest.* strcat appends a copy of *src* to the end of *dest*, overwriting the null character terminating the string pointed to by *dest*. The length of the resulting string is `strlen(dest) + strlen(src)`. strcat returns a pointer to the concatenated strings (this is *dest*, in fact).

Note: This routine assumes that *dest* points to a buffer large enough to hold the concatenated string.

Fonction : strcat2 or &
Numéro de la fonction : 0d28
Syntaxe : **strcat2**(*src,dest*) or *dest***&***src*
Description :

Assez similaire à la fonction **strcat**() tout en permettant d'utiliser la notation *dest* **&** *src* (comme en Tibasic). Cependant, *dest* n'est pas modifier après l'exécution de l'instruction. Le resultat de l'addition des deux chaînes de caractères est sauvegardé dans un espace mémoire spécial qui restera valable temporairement en mémoire. Il faut veiller à ce que la chaîne de caractères générée ne dépasse pas 50 caractères sous peine de crasher la calculatrice. Cette fonction est intéressante pour afficher du texte à l'écran. Pour sauvegarder définitvement le résultat de retour de cette fonction, la fonction **newstr**() peut par exemple être utilisée. Cette fonction retourne le pointeur sur la chaine de caractères générée.

Appends *src* to *dest in a temporary memory (available only during the execution of the line). The dest is not modified during the execution of this instruction*. Strcat2 (or &) appends a copy of *src* to the end of a copy of *dest*, overwriting the null character terminating the string pointed by the copy of *dest*. The length of the resulting string is `strlen(dest) + strlen(src)`. **strcat2** returns a pointer to the concatenated strings that is the address of a temporary memory area. You may be sure that `strlen(dest) + strlen(src)` is not over 50 characters for avoiding crash. The result of this function is temporary so you will have to copy it in memory if needed (with **newstr**() for example. By using **newstr**(), you will have to free the created memory block before exiting the program). This function as the advantage to be similar to the Tibasic function and to be simple to write in a program with the common **&**.

Fonction : strcmp or =
Numéro de la fonction : 0d
Syntaxe : **strcmp**(s1,*s2*) or s1=s2

Description :

Compare le contenu de deux chaînes de caractères entre elles. La fonction **strcmp**() commence par le premier caractère dans chaque chaîne de caractères et passe au caractère suivant et ainsi de suite jusqu'à ce qu'un caractère diffère ou que la fin de la chaîne de caractères est atteinte. **Strcmp**() retourne les valeurs suivantes :

- < 0 si *s1* est plus petit que *s2*
- == 0 si *s1* est identique à *s2*
- > 0 si *s1* est plus grande que *s2*

Plus précisément, si les chaînes de caractères diffèrent, la valeur du premier caractère qui diffère de s2 moins celui du caractère correspondant de la chaîne s1 est renvoyé.

Cette fonction peut aussi s'écrire en utilisant l'opérateur comparaison "=". Cette écriture peut être plus simple à utiliser et plus lisible que **strcmp**(). La syntaxe est la suivante : s1=s2. Seulement, il faut veiller à ce que s1 soit une chaîne de caractères saisie directement et non un pointeur sur une chaîne de caractères (aussinon le compilateur ne fera pas la différence avec l'opérateur classique de comparaison de nombre). Autrement dit, les commandes "hello"=str ou "hello"="XXX" effectueront bien une comparaison de chaînes de caractères, mais les commandes str="hello" et str=str2 n'effectueront pas une comparasion sur chaîne de caractères mais sur valeur (de pointeur ou de nombre). La fonction retourne 1 si les chaines de caractères sont identiques, aussinon si différente.

Compares one string to another. **strcmp**() performs an unsigned comparison of *s1* to *s2*. It starts with the first character in each string and continues with subsequent characters until the corresponding characters differ or until the end of the strings is reached. **strcmp**() returns a value that is

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

More precisely, if the strings differ, the value of the first nonmatching character in *s2* subtracted from the corresponding character in *s1* is returned.

You can also use the "=" command which is more readable for string comparison. The only restriction is that you must have *s1* hard coded in your program (not through a variable). *S2* can be a string pointer or an hard coded string. The command "hello"=str or "hello"="XXX" will work as a string comparator, but the command str="Hello" or str=str2 will work as classical pointer (or number) comparison.
The "=" string comparison will return 1 if string are identical, else 0.

Fonction : long sprintf
Numéro de la fonction : 0d26
Syntaxe : **sprintf**(*buffer*,*format*,*arg*)
Description :

Sends formatted output to a string. sprintf sends formatted output to a string. In fact, it does the following:

- accepts only one argument *arg*;
- applies the format specifier contained in the format string pointed to by *format*;
- outputs the formatted data to the string pointed to by *buffer*;

sprintf applies the format specifier to thet argument. The format string, controls how sprintf will convert and format its arguments. See printf1 for more info about format specifiers.

sprintf returns the number of bytes output, not including the terminating null byte in the count.

Example : print "9+10=19"
Prgm
init()
clrscr()
malloc(20) → buffer
sprintf(buffer,"9+10=%ld",9+10)
Pause buffer
free(buffer)

EndPrgm

Fonction : strcpy
Numéro de la fonction : 0d
Syntaxe : **strcpy**(dest,*src*)
Description :

Copie une chaîne de caractères pointée par *src* vers l'espace mémoire pointé par *dest*. Le caractère 0 de fin de chaîne de caractères est aussi copié. La function retourne *dest*.

Note : Si les chaines de caractères pointée par *src* et *dest* se chevauchent en mémoire, le comportement est indéfini. **Strcpy**() considère que *src* est un espace mémoire assez grand pour accueillir *dest*.

Copies string *src* to *dest***. strcpy**() copies string *src* to *dest*, stopping after the terminating null character has been moved. Returns *dest*.

Note: If the objects pointed to by *src* and *dest* overlap in memory, the behavior is undefined. strcpy assumes that *src* points to a buffer large enough to hold *dest*.

Fonction : ptr gets
Numéro de la fonction : 0d117
Syntaxe : **gets**(*buffer*)
Description :
Récupère une chaîne de caractères saisie au clavier jusqu'à temps que la touche ENTER soit appuyée et la copie dans **buffer**. La touche espace est supportée.

Gets() gets a string from the keyboard. **gets()** collects a string of characters terminated by a new line from the keyboard and puts it into *buffer*. **gets()** returns when it encounters a new line (i.e. when the ENTER key is pressed); everything up to the new line is copied into *buffer*. **gets()** returns the string argument *buffer* . For editing, the backspace key is supported.

Fonction : value strlen
Numéro de la fonction : 0d
Syntaxe : **strlen**(*string*)
Description :
Calcul et renvoi la longueur en caractères de la chaîne de caractères *string*. Le caractère nul de fin de *string* n'est pas compté.

It calculates the length of *string*. Returns the number of characters in *string*, not counting the terminating null character.

Fonction : val atol
Numéro de la fonction : 0d
Syntaxe : **atol**(*string*)
Description :
Converti la chaine de caractères *string* en un nombre et renvoie ce nombre. Similaire à **expr**() mais ne prends que des strings ou des pointeurs de string en arguments (avec un éventuel +/- ou espaces). Cette fonction est légèrement plus rapide que **expr**() dans ce cas.

Converts the string *string* to a number (possibility to use +/- and spaces). Returns this number. Similar with **expr**() but takes only string and pointer to string in argument. This function is a bit faster than **expr**() in this case.

Example : Display 666 to the screen
```
init()
clrscr()
printld(atol("666"))
keywait()
```

Fonction : val expr
Numéro de la fonction : 0d
Syntaxe : **expr**(*string* or *nilist* or *other*)
Description :
Converti la chaine de caractères *string* en un nombre et renvoie ce nombre. Permet aussi d'exécuter directement une nilist *nilist* (sans utilisation du #) et de renvoyer l'évaluation de la dernière instruction exécutée. Si l'argument est une valeur ou d'un autre type (*other*), renvoie le résultat de son exécution sans traitement particulier. Cette fonction a l'intérêt d'avoir la même écriture que la fonction homologue Tibasic. Elle est plus évoluée que la fonction **atol**() mais un peu plus lente.

Converts the string *string* to a number and returns this number. Allow to execute directly a non instantiated list *nilist* (without using #) and to returns the evaluation of the last instruction executed. If the argument is a value or an another type (*other*), returns the result of its execution

as it. This function has the interest to have the same writing than the similar Tibasic function expr(). This function is more evolved than the **atol**() function so it is a bit slower.

6 Fonctions graphiques – Graphics functions

Fonction : clrlcd
Numéro de la fonction : 0d55
Syntaxe : **clrlcd**()
Description :
Efface la mémoire video actuelle. Cette fonction n'affecte pas la position du curseur de texte (fonctions printf…) comme le fait le fonction **clrscr**(). Elle peut être utilisée pour faire des niveaux de gris. Elle est légèrement plus rapide que la fonction **clrscr**().
Clear the screen without reseting the text cursor position. Faster for graphics purposes than **clrscr**(). Can be used to make grayscale. Slightly faster the **clrscr**().

Fonction : long gmode
Numéro de la fonction : 0d42
Syntaxe : **gmode**(*gmode*)
Description :
Sélectionne le mode d'affichage de toutes les fonctions graphiques. Retourne l'ancienne valeur du mode graphique.
Select with this function the graphics mode for all graphics functions of newprog. Returns the former value of the graphic mode.

0 = gor	:Draw as normal (Or)
1 = gerase = greplace	:Draw as reverse (blanking) - Draw as replace for sprites
2 = gxor	:Draw as using XORing with the destination
3 = gand	:Draw by performing And (For sprites functions only, MASK).

Fonction : ptr setlcd
Numéro de la fonction : 0d50
Syntaxe : **setlcd**(*video_memory_address*)
Description :
Spécifie la position de la mémoire vidéo. Toutes les fonctions graphiques s'appliqueront a l'écran graphique pointé par *video_memory_address*. La valeur par défaut est 0h4C00 si les niveaux de gris n'ont pas été activés. Si *video_memory_address* est différente de 0h4c00, les graphiques réalisés ne seront pas visibles. Il faudra copié le contenu de la mémoire vidéo pointée par *video_memory_address* vers l'adresse 0h4c00 pour les visualiser (avec **memcpy**(0h4c00,*video_memory_address,*lcdsize)). Cette fonction est utilisée (entre autres) lorsque l'on ne souhaite pas visualiser la construction d'une image, mais seulement le résultat (c'est la méthode appelée « double buffering », elle est souvent utilisée dans les jeux 2D évolués). Il faut veiller à restaurer l'adresse initiale avant de quitter le programme.
Cette fonction retourne l'ancienne adresse de la mémoire graphique.

Specify the position of the graphic memory adress. All graphics functions will use the adress *video_memory_address* for theirs drawings.The default value is 0h4c00. If *video_memory_address* is not equal to 0h4c00, all drawings will not be visibles. To visualize them, you will have to copy le content of the memory memory pointed by *video_memory_address* to the memory area pointed by 0h4c00 (with **memcpy**(0h4c00,*video_memory_address,*lcdsize). This function is often used when we don't want to see the construction of graphics, but only the result. This method is called "double buffering" ; it is often used when making evolved 2D games). You will have to restore the video memory address before exiting your program for avoiding crash.
This function returns the previous value of graphics memory address.

Fonction : ptr getlcd
Numéro de la fonction : 0d60
Syntaxe : **getlcd**()
Description :
Retourne l'adresse de la mémoire vidéo actuelle (Pour info, =0h4c00 si les niveaux de gris ne sont pas activés).

Return the adress of the current video memory (For information, =0h4c00 if the grayscale are disabled).

Fonction : val drawpic
Numéro de la fonction : 0d
Syntaxe : drawpic(*x,y, pic_str*)
Description :
Affiche aux coordonnées (*x,y*) l'image bitmap Tibasic dont le nom est *pic_str* (rep\file). Le mode d'affichage est défini avec la fonction **gmode**(). Cette fonction est proche des fonctions Tibasic telle que rplcpic et rclpic. Retourne 0 si une erreur est survenu, 1 aussinon.

Displays at the coordinates (*x,y*) to the screen the bitmap picture *pic_str* (it is a Tibasic picture). The display mod is set with the **gmode**() function. Quite similar to the Tibasic functions rplcpic and rclpic. Return 0 in case of error, else 1.

Fonction : val getpic
Numéro de la fonction : 0d
Syntaxe : **getpic**(*x1,y1,x2,y2, pic_str*)
Description :
Sauvegarde le contenu de l'écran entre les points (*x1,y1*) et (*x2,y2*) dans la variable Tibasic *pic_str*. Renvoi la taille en octets du fichier créé. Assez similaire à la fonction Tibasic stopic, avec toutefois la possibilité d'atteindre tout l'écran. Cette fonction est lente.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Save the content of the screen localize between the point *(x1,y1)* et *(x2,y2)* in the Tibasic variable *pic_str*. Returns the size in bytes of the file created. This function is quite similar to the stopic Tibasic function, with in addition the possibility to reach all the screen. This function is slow.
If the calculator has not enough memory for creating the output file, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Fonction : sprt8 - value sprt82
Numéro de la fonction : 0d43
Syntaxe : **sprt8**(*x,y,h,sprite*) - **sprt82**(*x,y,h,sprite*)
Description :
Affiche le sprite (8 bits horizontales) *sprite* avec le coin supérieur au point (*x,y*) et une hauteur de *h* lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. **Sprt8**() affiche environ 6000 sprites par secondes (4000 pour sprt82).
Sprt82() est une variante de **sprt8**(). Contrairement à **sprt8**(), elle retournera une valeur non nulle si au moins un pixel actif du sprite *sprite* est entré en collision avec un pixel à l'écran lors de son affichage (aussinon retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur deux octets chacun (valable jusqu'au prochain appel d'une fonction **sprtX2**()). Le premier élément de cette liste sera égale à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite.
Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction **settype**() pour pouvoir utiliser la notation []. Aussinon, l'utilisation de **lb**(pointeur_sur_la_liste,élément) peut être utiliser.
Voir **gmode**() pour les modes d'affichage possibles.

Sprt8() draws a 8 bits sprite *sprite* to the current screen at the position *x* and *y*, with a height of the sprite in pixels equals to *h*. The sprites are clipped outside the rectangle whose extreme points are (0,0) and (239,127). Draw about 6000 sprites (8 bits x 8 lines) per second.

Sprt82() is closed to **sprt8**() but will return a non null value if at least one of its pixel is entered in collision with one pixel of the screen during the display (else 0). The function will not display the sprite if a collision is detected. The function returned non null value that is a pointer on a list of data written on 4 bytes (which one will be available until a next coll of **spriteX2**() function). The first element of the list is equal to the height on which a collision has been detected. The second element is the value, to the corresponding collision height, of the sprite. This value will helps you to localize the collision. For example, if the value is negative, it means that at least the byte on extreme left is entered in collision. If the number is equal to 1, it means that the bit on the right is the only one entered in collision. The example (n°2) in **sprt16**() description will helps you to understand this function.

See **gmode**() for allowed displaying mods.

Example : Similar to sprt16 example

Fonction : sprt16 – value sprt162
Numéro de la fonction : 0d44
Syntaxe : **sprt16**(*x,y,h,sprite*) - **sprt162**(*x,y,h,sprite*)
Description :
Sprt16() affiche le sprite (16 bits horizontales) *sprite* avec le coin supérieur au point (*x,y*) et une hauteur de *h* lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. Affiche environ 5000 sprites (carrés) par seconde.

Sprt162() est une variante de **sprt16**(). Contrairement à **sprt16**(), elle retournera une valeur non nulle si au moins un pixel actif du sprite *sprite* est entré en collision avec un pixel à l'écran lors de son affichage (aussinon retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur deux octets chacun (valable jusqu'au prochain appel d'une fonction **sprtX2**()). Le premier élément de cette liste sera égale à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite.
Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction **settype**() pour pouvoir utiliser la notation []. Aussinon, l'utilisation de **lw**(pointeur_sur_la_liste,élément) peut être utiliser. Voir exemple n°2.
Voir **gmode**() pour les modes d'affichages possibles.

Sprt16() is quite similar to **sprt8**() but for a 16 bits wide sprite. The sprites are clipped outside the rectangle whose extreme points are (0,0) and (239,127). Draw about 5000 sprites (16 bits x 16 lines) per second.

Sprt162() is close to **sprt16**() but will return a non null value if at least one of its pixel is entered in collision with one pixel of the screen during the display (else 0). The function will not display the sprite if a collision is detected. The function returned a non null value that is a pointer on a list of data written on 2 bytes (which one will be available until a next call of **spriteX2**() function). The first element of the list is equal to the height on which a collision has been detected. The second element is the value, to the corresponding collision height, of the sprite. This value will helps you to localize the collision. For example, if the value is negative, it means that at least the byte on extreme left is entered in collision. If the number is equal to 1, it means that the bit on the right is the only one entered in collision. The example (n°2) will helps you to understand this function.

See **gmode**() for allowed displaying mods.

Example 1 – **sprt16** : Draw a sprite (filled rect)

```
init()
seqw(ii,1,4,1,sprite,-1)                    //-1 = 0b11111111 make a line of pixels | can be replaced by :{sled,2,-1,-1,-1,-1}
clrlcd()
gmode(0)
sprt16(30,30,4,sprite)
keywait()
free(sprite)
```

Example 2 : Draw a sprite (box). A best way to use sprites.

```
init()
w(sprite)
0b1111111111111111
repeat(14,0b1000000000000001)
0b1111111111111111
y
clrlcd()
gmode(0)
sprt16(30,30,16,sprite)
keywait()
```

Example 2 – **sprt162** : A little landscape with obstacles

```
()
Prgm
init()
w(sprt)
repeat(6,0b0001110001110000)
0b1100000000000011
0b1111111111111111
repeat(8,0b1100110011110011)
y
clrscr()
40 → yy
80→ xx
0→ nc
settype(rr,2)

gmode(2)
fillrect(0,0,40,127)
fillrect(160-40,0,160,127)

fillrect(0,0,160,10)
fillrect(0,95,160,100)

dline(50,0,40,100)
dline(110,0,120,100)

dline(60,0,60,30)

dline(60,99,65,80)

sprt16(xx,yy,16,sprt)
While gkey()!=esc
```

```
 repeat(500,0)
isz(inc)
 if lefttt()  Then
 sprt16(xx,yy,16,sprt)
  if sto(rr,sprt162(dsz(xx),yy,16,sprt)) Then
 sprt16(isz(xx),yy,16,sprt)
 EndIf
 EndIf
if rightt() Then
 sprt16(xx,yy,16,sprt)
 if sto(rr,sprt162(isz(xx),yy,16,sprt)) Then
 sprt16(dsz(xx),yy,16,sprt)
 EndIf
 EndIf
 if up() Then
 sprt16(xx,yy,16,sprt)
 if sto(rr,sprt162(xx,dsz(yy),16,sprt)) Then
 sprt16(xx,isz(yy),16,sprt)
 EndIf
 EndIf
 if down() Then
 sprt16(xx,yy,16,sprt)
 if sto(rr,sprt162(xx,isz(yy),16,sprt)) Then
 sprt16(xx,dsz(yy),16,sprt)
 EndIf
 EndIf
 if rr Then
 printxy(0,0,"height=%ld    ",rr[0])
 printxy(70,0,"value=%ld    ",rr[1])
 Else
 EndIf
EndWhile
EndPrgm
```

Fonction : sprt32 - value sprt322
Numéro de la fonction : 0d53
Syntaxe : **sprt32**(*x,y,h,sprite*) - **sprt322**(*x,y,h,sprite*)
Description :
Sprt322() affiche le sprite (32 bits horizontales) *sprite* avec le coin supérieur au point (*x,y*) et une hauteur de *h* lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. Affiche environ 5000 sprites par seconde.

Sprt322() est une variante de **sprt32**(). Contrairement à **sprt32**(), elle retournera une valeur non nulle si au moins un pixel actif du sprite *sprite* est entré en collision avec un pixel à l'écran lors de son affichage (aussinon retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur quatre octets (ce pointeur est valable jusqu'au prochain appel d'une fonction **sprtX2**()). Le premier élément de cette liste sera égale à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite.
Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction **settype**() pour pouvoir utiliser la notation []. Aussinon, l'utilisation de **ll**(pointeur_sur_la_liste,element) peut être utiliser.

Voir **gmode**() pour les modes d'affichage possibles.

Sprt32() is quite similar to **sprt32**() but for a 32 bits wide sprite. The sprites are clipped outside the rectangle whose extreme points are (0,0) and (239,127). Draw about 5000 sprites (32 bits x 32 lines) per second.

Sprt322() is closed to **sprt32**() but will return a non null value if at least one of its pixel is entered in collision with one pixel of the screen during the display (else 0). The function will not display the sprite if a collision is detected. The function returned a non null value that is a pointer on a list of data written on 2 bytes (which one will be available until the next call of **spriteX2**() function). The first element of the list is equal to the height on which a collision has been detected. The second element is the value, to the corresponding collision height, of the sprite. This value will helps you to localize the collision. For example, if the value is negative, it means that at least the byte on extreme left is entered in collision. If the number is 1, it means that the bit on the right is the only one entered in collision. The example (n°2) in **sprt16**() description will helps you to understand this function.

See **gmode**() for allowed displaying mods.

Example : Similar to sprt16 example

Fonction : gsprt8
Numéro de la fonction :
Syntaxe : **gsprt8**(*x,y,h,sprite_dest*)
Description :
Copie un sprite de largeur 8 pixels (soit un octet ou byte) et de hauteur *h* lignes à partir de l'écran à la postion x et y et le sauvegarde dans le buffer *sprite_dest*. Veiller à ce que le buffer *sprite_dest* soit assez large pour éviter de crasher la calculatrice. Le buffer peut être créé avec **malloc**() ou **b**():**y** par exemple.

gsprt8() copies a sprite from the screen at *x* and *y* to *sprite_dest*. This routine is fast. Additionally sprites retrieved can be used as arguments for all the Sprite routines (**sprt8**(),**sprt16**(),**sprt32**(),etc...). *x* and *y* are the coordinates of the upper left corner of the sprite. *h* is the height of the sprite. sprite_*dest* is the pointer to a buffer (array, ilist) which is large enough to store the fetched sprite data. The buffer can be created by using **malloc**() or **b**():**y** for example.

Fonction : gsprt16
Numéro de la fonction :
Syntaxe : **gsprt16**(*x,y,h,sprite_dest*)
Description :
Copie un sprite de largeur 16 pixels (soit 2 octets ou 2 bytes) et de hauteur *h* lignes à partir de l'écran à la postion x et y et le sauvegarde dans le buffer *sprite_dest*. Veiller à ce que le buffer *sprite_dest* soit assez large pour éviter de crasher la calculatrice. Le buffer peut être créé avec **malloc**() ou **w**():**y** par exemple.

gsprt16() copies a sprite from the screen at *x* and *y* to *sprite_dest*. This routine is fast. Additionally sprites retrieved can be used as arguments for all the Sprite routines (**sprt8**(),**sprt16**(),**sprt32**(),etc...). *x* and *y* are the coordinates of the upper left corner of the sprite. *h* is the height of the sprite. sprite_*dest* is the pointer to a buffer (array, ilist) which is large enough to store the fetched sprite data. The buffer can be created by using **malloc**() or **w**():**y** for example.

Fonction : gsprt32
Numéro de la fonction :
Syntaxe : **gsprt32**(*x,y,h,sprite_dest*)
Description :
Copie un sprite de largeur 32 pixels (soit 4 octets ou 4 bytes) et de hauteur *h* lignes à partir de l'écran à la postion x et y et le sauvegarde dans le buffer *sprite_dest*. Veiller à ce que le buffer *sprite_dest* soit assez large pour éviter de crasher la calculatrice. Le buffer peut être créé avec **malloc**() ou **l**():**y** par exemple.

gsprt32() copies a sprite from the screen at *x* and *y* to *sprite_dest*. This routine is fast. Additionally sprites retrieved can be used as arguments for all the Sprite routines (**sprt8**(),**sprt16**(),**sprt32**(),etc...). *x* and *y* are the coordinates of the upper left corner of the sprite. *h* is the height of the sprite. sprite_*dest* is the pointer to a buffer (array, ilist) which is large enough to store the fetched sprite data. The buffer can be created by using **malloc**() or **l**():**y** for example.

Fonction : gsprt8x
Numéro de la fonction :
Syntaxe : **gsprt8x**(*x,y,h,byte_width,sprite_dest*)
Description :
La fonction **gsprt8x()** diffère des fonctions de récupérations de sprites vues plus haut par le fait qu'elle permet de récupérer des sprites d'une largeur en octets paramétrable (variable *byte_width*).
Gsprt8x() récupère un sprite à partir de l'écran actif aux coordonnées *x* et *y* et l'enregistre dans le buffer passer en argument *sprite_dest*.
Cette fonction est beaucoup plus rapide pour sauvegarder une portion de l'écran que la fonction **getpic**(). *X* et *y* étant les coordonnées du coin supérieur gauche du sprite à récupéré. *h* est la hauteur en pixels du sprite.
Veiller à ce que le buffer de destination soit assez grand pour éviter les crashs.
Pour afficher le sprite récupérer, utiliser la fonction **sprt8x**(). Dans les cas particuliers où *byte_width* est égal à 1 ou 2 ou 4, les fonctions **sprt8,sprt82,sprt16**,etc... peuvent être utilisées.
Les données du sprite récupéré sont organisées de la façon suivante (pour *byte_width*=3) : line1/byte1, line1/byte2, line1/byte3, line2/byte1, line2/byte2, line2/byte3 etc.
La fonction n'étant pas clippée, veiller à ne pas sortir du cadre de l'écran sous peine de crash.
Voir l'example plus bas.

The **gsprt8x()** function differs from the rest of the grabbing sprites functions (**gsprt8**(), etc...). It is not fixed to a specific width, but you can specify the width in bytes as parameter (*byte_width)*.

gsprt8x() grabs a sprite from the screen at x and y to *sprite_dest*. This routine is many times faster than the **getpic**() function. *x* and *y* are the coordinates of the upper left corner of the sprite. *h* is the height (in pixels) of the sprite. *sprite_dest* is the pointer to a buffer (array) which is large enough to store the fetched sprite data.
To display a sprt8x sprites, you will have to use the **sprt8x**() function. If *byte_width* is equal to 1 or 2 or 4, sprites retrieved can be used as arguments for all the Sprite routines (**sprt8,sprt82,sprt16,etc..**).
The sprite data fetched by SpriteX8Get is organized like this (example for *byte_width* == 3): line1/byte1, line1/byte2, line1/byte3, line2/byte1, line2/byte2, line2/byte3 etc.
This function is not clipped, so be aware to stay in the screen frame.
See the example below.

Example : fill the screen with the upper left corner of the initial screen

```
init()
malloc(lcdsize)→sprt
gsprt8x(0,0,50,10,sprt)
clrscr()
sprt8x(0,0,50,10,sprt)          //display in left upper corner
sprt8x(10*8,0,50,10,sprt)       //display in right upper corner
sprt8x(0,50,50,10,sprt)         //display in left down corner
sprt8x(10*8,50,50,10,sprt)      //display in right down corner
keywait()
free(sprt)
```

Fonction : dsprt8x
Numéro de la fonction :
Syntaxe : **sprt8x**(*x,y,h,byte_width,sprite*)
Description :
La fonction **sprt8x()** diffère des fonctions d'affichage de sprites classiques par le fait qu'elle permet d'afficher des sprites d'une largeur en octets paramétrable (variable *byte_width*). **sprt8x**() affiche le sprite *sprite* à l'écran actif aux coordonnées *x* et *y,* avec une hauteur en pixel de *h* pixels. Cette fonction est faite pour être utiliser avec la fonction **gsprt8x**().
La fonction n'étant pas clippée, veiller à afficher l'intégralité de l'image dans le cadre de l'écran sous peine de de crash.
Les modes d'affichage sont définis avec la fonction **gmode**().

The function **sprt8x**() allow the user to display sprite with parametric bytes width (*byte_width* parameter). The sprite will be drawn to the coordinates (*x,y*) with a height in pixels of *h*. This function is intended to be used with the **gsprt8x**() function.
This function is not clipped, so be aware to stay in the screen frame.
The displaying mode is set with the **gmode**() function.

Fonction : ptr gsprt
Numéro de la fonction : 0d102
Syntaxe : **gsprt**(*w,h,bitmap,sprtdest*)
Description :
Créé un sprite de *w* octets de larges (*w* = 1 ou 8 ; 2 ou 16 ; 4 ou 32) et de *h* lignes de hauteur à partir de l'image tibasic (type PIC) *bitmap*. En sortie, *Sprtdest* pointe sur le sprite ainsi créé. Vous devez exécuter **free**(*sprtdest*) avant le fin du programme si vous ne souhaitez pas perdre inutilement de la mémoire.
Il est important que bitmap soit d'une largeur réelle de *w* pour retranscrire correctement l'image.

Create a sprite from a Tibasic PIC file named *bitmap* with a width of w (*w* = 1 or 8 ; 2 or 16 ; 4 or 32) and with a high of *h* lines. In output, *Sprtdest* points to the created sprite. You should erase the sprite *sprtdest* before the end of the execution of the program if you don't want lose memory.
Bitmap must have a real *w* width for having *sprtdest* looking as nice as possible.

Example :you must have edited previously a Tibasic PIC var named “bitmap” with a real width of 8 pixels.

```
Init()
clrscr()
gsprt(8,3,”bitmap”,pp)
sprt8(50,50,3,pp)
free(pp)
keywait()
```

Fonction : dline
Numéro de la fonction : 0d40
Syntaxe : **dline**(*xa,ya,xb,yb*)
Description :
Affiche une ligne de caractère à l’écran entre le point (*xa,ya*) et (*xb,yb*). La ligne doit être entièrement contenue dans la fenêtre (0,0,239,127). Dans le cas contraire, la calculatrice plantera.

Display a line on the screen between the point (*xa,ya*) et the point (*xb,yb*). All pixels of the line shall be in the screen rect (0,0,239,127) to avoid crash.

Fonction : dmline
Numéro de la fonction : 0d41
Syntaxe : **dmline**(*list_of_x,list_of_y,num_pts*)
Description :
Trace une première ligne puis une deuxieme avec le dernier point de la première ligne et ainsi de suite. Les lignes seront tracées en allant des premiers éléments des listes jusqu'au *num_pts* éléments des listes *list_of_x* et *list_of_y*. Au total, il y a *num_pts*-1 lignes.

Draw a first line and a second one and so on. The lines will be drawn by going from the firsts elements of the lists to the *num_pts* elements of the lists *list_of_x* and *list_of_y*. There will have *num_pts*- lines.

Fonction : dcircle
Numéro de la fonction : 0d40
Syntaxe : **dxircle**(*xc,yc,radius*)
Description :
Affiche un cercle de centre (*xc,yc*) avec un rayon égal à *radius*. Voir **gmode()** pour le mode d'affichage. Les coordonnées du centre du cercle doit être compris dans le cadre de la fenêtre graphique.

Draw an outlined circle which center is at (*xc,yc)* and with a radius of *radius*. See **gmode()** for displaying mode. The coordinates of the center must be placed in the screen frame.

Fonction : fillcirc
Numéro de la fonction : 0d40
Syntaxe : **fillcirc**(*xc,yc,radius*)
Description :
Dessine un cercle plein de centre (*xc,yc*) avec un rayon égal à *radius*. Voir **gmode()** pour le mode d'affichage. Les coordonnées du centre du cercle peut être en dehors de la fenêtre graphique (la fonction est clippée).

Draw a filled circle which center is at (*xc,yc)* and with a radius of *radius*. See **gmode()** for displaying mode. The coordinates of the center can be placed out of the screen frame.

Fonction : - dpix
Numéro de la fonction : 0d51
Syntaxe : **dpix**(*x,y*)
Description :
Dessine un pixel à l'écran. Le mode d'affichage est sélectionné comme avec toutes les fonctions graphiques à l'aide de la fonction **gmode**.

Draw a pixel on the screen. The display mode is select with the **gmode**() function.

Fonction : long gpix
Numéro de la fonction : 0d52
Syntaxe : **gpix**(*x,y*)
Description :
Retourne l'état du pixel pointé par *x* et *y* :
1 si allumé
0 si éteint

Return the state of the pixel which is placed at the position (*x,y*) :
1 if dark
0 if white

Fonction : - fillrect
Numéro de la fonction : 0d54
Syntaxe : **fillrect**(*xa,ya,xb,yb*)
Description :
Dessine un rectangle plein entre le point *(xa,ya)* et *(xb,yb)*. Le mode d'affichage est réglé avec la fonction **gmode().**

Draw a filled rect. The graphic mod is set with **gmode**().

Fonction : long grayon
Numéro de la fonction : 0d70
Syntaxe : **grayon**()
Description :
Active les niveaux de gris (4 niveaux). Cette fonction désactive les boites de dialogues d'affichage d'erreurs et réactivera la valeur au prochain appel de **Grayoff**() (voir fonction **disperr**()). Des exemples sont fournis pour vous aider à comprendre comment faire des niveaux de gris.

Activates grayscale mode with four shades of gray. **Grayon**() activates grayscale mode. This works on both hardware version 1 and 2 calculators because the calculator type is detected automatically.**Grayon**() returns 0 if there was an error in switching to grayscale mode,

otherwise it returns1. Don't forget to switch off grayscale mode before your program terminates, or your TI will crash very soon! This function deactivate the error displaying up to the next call of **grayoff**() (see **disperr**()). See example furnished in the bundle.

Fonction : long grayoff
Numéro de la fonction : 0d71
Syntaxe : **grayoff**()
Description :
Désactive les niveaux de gris. Va de paire avec **grayon**().
Réaffecte le mode d'affichage des erreurs à l'état précédant l'appel de la fonction préalable **grayon**() (voir fonction **disperr**()).

Deactivates grayscale mode. This function deactivates grayscale mode. If grayscale mode is not activated, this function does nothing. Restore the mode of error displaying before **grayon**() (see **disperr**() function).

Fonction : - light
Numéro de la fonction : 0d72
Syntaxe : **light**()
Description :
La fonction **grayon()** doit avoir été exécutée précédemment. Cette fonction affecte l'adresse de la mémoire vidéo au plan 'light'. Cela correspond à un affichage gris clair.

Grayon() must have been use before. This function set the adress of the graphic memory to the light plane.

Exemple :
```
init
seqw(vv,1,16,1,ss,-1)
grayon()
light()
clrlcd()
sprt16(50,50,16,ss)
dark()
clrlcd()
sprt16(58,58,16,ss)
keywait()
grayoff()
```

Fonction : - dark
Numéro de la fonction : 0d73
Syntaxe : **dark**()
Description :
La fonction **grayon()** doit avoir été exécutée précédemment. Cette fonction affecte l'adresse de la mémoire vidéo au plan 'dark'.

The same thing than **light**() but for the dark plane.

Fonction : drawstr
Numéro de la fonction : 0d75
Syntaxe : **drawstr**(*x,y,str*)
Description :
Affiche au coordonnées (*x,y*), la chaîne de caractères *str*.Cette fonction n'affecte pas le curseur de position du texte (fonctions **prints**...).

Drawstr() draws a string *str* at a specific (*x*, *y*) location. This does not interfer with the text cursor (with **prints()** and other functions).
Fonction : - svscroff
Numéro de la fonction : 0d
Syntaxe : **svscroff**()
Description :
By default (if you don't launch this function), at the end of the execution of the program, the previous LCD state is restore.
By using this function, you will disable the auto screen restore at the end of the execution of the program (for graphic purposes, when switching between Tibasic and NewProg program for example).

Fonction : ptr savescr
Numéro de la fonction : 0d162
Syntaxe : savescr(*index*)
Description :
Enregistre la mémoire vidéo dans l'espace mémoire spécifié par l'index *index* et retourne un pointeur sur cet espace mémoire. Si l'espace mémoire spécifié par index était déjà occupé, **savescr**() liberera l'espace mémoire de l'ancienne sauvegarde d'écran avant de copier l'écran actuel (toujours référencé par *index*). NewProg effacera automatiquement les sauvegardes d'écrans à la fin de l'éxécution du programme.

Vous pouvez cependant le libérez par vous même avec la fonction free. A utiliser avec la fonction **loadscr**(). *Index* ne peut prendre que les valeurs suivantes : 0,1,2,3.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Make a copy of the screen to a specified block of memory identified by the index *index*. **Savescr**() returns the pointer of the memory block where the screen has been saved. If there was a previous saved screen at the specified index *index*, newprog will overwrite it and then made the copy. Newprog erase automatically the video memory block at the end of the execution of the program (**free**() use is not necessary). *Index* must be in [0,3] range. To be used with the **loadscr**() function.
If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Fonction : ptr loadscr
Numéro de la fonction : 0d163
Syntaxe : loadscr(*index*)
Description :
Affiche à l'écran l'image pré enregistrée référencée par l'index *index* (utilisation avec **savescr**()). Plus exactement, cela copie l'image vers la mémoire vidéo actuelle (**setlcd**(), **getlcd**()...). Elle peut être utilisée pour faire des niveaux de gris. Retourne un pointeur sur l'espace mémoire qui a été copié à l'écran.

Copy to the screen the video memory saved previously with the **savescr**() function. The video memory to copy is identified by *index* (must be in [0,3]). This function is available for the grayscale functions. Return the pointer to the source video memory .

Example :
```
init()
savescr(0)
clrscr()
prints("Press a key")
keywait()
clrscr()
loadscr(0)
prints("Screen reloaded")
keywait()
```

Fonction : val lcdup
Numéro de la fonction : 0d
Syntaxe : **lcdup**()
Description :
Augmente le contraste de 1.Retourne la nouvelle valeur du niveau de contraste.

Increases the contrast. **lcdup**() increases by one the display contrast. Returns the new value of the contrast level.

Fonction : val lcddown
Numéro de la fonction : 0d
Syntaxe : **lcddown**()
Description :
Diminue le contraste de 1. .Retourne la nouvelle valeur du niveau de contraste.

Decreases the contrast. **lcddown**() decreases by one the display contrast. Returns the new value of the contrast level.

Fonction : val prettyxy
Numéro de la fonction : 0d
Syntaxe : **prettyxy**(*x,y,expr_str*)
Description :
Affiche sous la forme « pretty print » (comme dans l'écran home) au coordonées (x,y) l'expression *expr_str* (chaine de caractères). Retourne 0 si l'expression *expr_str* contient une erreur, 1 sinon.

Performs "pretty printing" (or "2D printing") to the screen at the coordinates (*x,y*) of the expression written in the string *expr_str*. Returns 0 if an error occured, else 1.

Example: See function **getwbt**() example.

Fonction : val getwbt
Numéro de la fonction : 0d
Syntaxe : **getwbt**(*expr_str,width_dest_var,bottom_dest_var,top_dest_var*)
Description :
Retourne des informations relatives à la dimension d'un block afficher à l'écran à l'aide de la fonction prettyxy(). Les informations seront sauvegardées dans les trois variables *width_dest_var* (largeur en pixel),*bottom_dest_var* (ordonnée la plus basse),*top_dest_var* (ordonnée la plus haute).

Gets information about dimensions of block which will be "pretty printed". **getwbt**() gets information about the screen space which will be occupied by displayed expression (*expr_str*). **Getwbt**() stores the information in three variables : *width_dest_var,bottom_dest_var,top_dest_var* . *width_dest_var* will be affected to the width of the displayed block, and *top_dest_var* and *bottom_dest_var* represent distances of top and bottom edge of the block measured from the wanted y position. More precise, if the expression is printed at (x, y) using prettyxy, the left corner of the occupied space will be at (x, y-*top_dest_var*), and the right corner of the occupied space will be at (x+*width_dest_var*, y+*bottom_dest_var*). So, the 2D expression has an imaginary center line above which is the "Top", and below (counting this line too) is the "Bottom".

Example : Displays an expression in the up left corner, in the middle and in the bottom centered areas of the screen.

```
init()
clrscr()
"sin(7*x^2+y)"→ex
getwbt(ex,ww,bb,tt)
prettyxy(0,0+tt,ex)
prettyxy(80-ww/2,49,ex)
prettyxy(80-ww/2,99-tt+bb,ex)
keywait()
```

Fonction : - lscroll (for 160x100 screen) lscroll2(for ti92+ and V200 and ti89 for 240x120 screen)
Numéro de la fonction : 0d90
Syntaxe : **lscroll**(*num_line*)
Description :
Effectue une translation de 1 pixel vers la gauche de *num_line* lignes à partir de l'adresse de la mémoire vidéo. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction **setlcd**()).

Make a left scroll of the screen of *num_lines* since the adress of the memory video. It can be usefull to modify the adress of the graphic memory for making a scrolling since the third lines (raws) for example. (with **setlcd**()).

Fonction : - rscroll (for 160x100 screen) rscroll2(for ti92+ and V200 and ti89 for 240x120 screen)
Numéro de la fonction : 0d91
Syntaxe : **rscroll**(*num_line*)
Description :
Effectue une translation de 1 pixel vers la droite de *num_line* lignes à partir de l'adresse de la mémoire vidéo. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction **setlcd**()).

Make a right scroll of the creen. Look to **lscroll**() for details.

Fonction : - uscroll (for 160x100 screen) uscroll2(for ti92+ and V200 and ti89 for 240x120 screen)
Numéro de la fonction : 0d92
Syntaxe : **uscroll**(*num_line*)
Description :
Effectue une translation de 1 pixel vers le haut de *num_line* lignes à partir de l'adresse de la mémoire vidéo. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction **setlcd**()).

Make a up scroll of the creen. Look to **lscroll**() for details.

Fonction : - bscroll (for 160x100 screen) bscroll2(for ti92+ and V200 and ti89 for 240x120 screen)
Numéro de la fonction : 0d93
Syntaxe : **bscroll**(*num_line*)
Description :
Effectue une translation de 1 pixel vers le bas de *num_line* lignes à partir de l'adresse de la mémoire vidéo. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction **setlcd**()).

Make a bottom (down) scroll of the creen. Look to **lscroll**() for details.

7 Fonctions clavier – Keypad functions

Fonction : long keywait
Numéro de la fonction : 0d6
Syntaxe : **keywait**()
Description :
Stop l'exécution du programme et attend que l'on appuie sur une touche. Renvoie alors le numéro de la touche appuyée. La valeur est comparable à la fonction getkey() du Tibasic (voir fonction **gkey**() pour le code des touches les plus courantes). Des constantes ont été définies pour éviter d'avoir à mémoriser le code des touches les plus courantes. (Réalise un appel en interne de la fonction **keydisp**().)

Waiting for a key to be pressed. Return the number of the key pressed. Similar to getkey() code in Tibasic function (or newprog **gkey**() function). You can use predefined constants to avoid memorise currents keys. (Perfoms internally a call of the **keydisp**() function.)

Fonction : value gkey
Numéro de la fonction : 0d182
Syntaxe : **gkey**()
Description :
Gkey() est la fonction la plus souple à utiliser pour tester si une touche est appuyée sans imposer d'arrêt au programme lors de son exécution. Renvoie le même code que la fonction keywait().
Similaire à la fonction getkey du Tibasic. **Keyclear**() ne doit pas avoir été exécutée précédemment pour ne pas interférer avec cette fonction. Pour cette fonction, des constantes prédéfinies peuvent être utiliser (voir exemple ci dessous).

Gkey() is the best way to test if one key is pressed without stoping the execution of your program. Returns the same key code than keywait() function.
Is the same as the Tibasic getkey() function. **Keyclear**() must not be enable for avoiding to freeze the calculator.
The user can use predifined constants (see example below).

TI-89:

Key	Normal	+Shift	+2nd	+Diamond	+alpha
Up	337	8529	4433	16721	33105
Right	344	8536	4440	16728	33112
Down	340	8532	4436	16724	33108
Left	338	8530	4434	16722	33106

TI-92+:

Key	Normal	+Shift	+2nd	+Diamond	+alpha
Up	338	16722	4434	8530	33106
Right	340	16724	4436	8532	33108
Down	344	16728	4440	8536	33112
Left	337	16721	4433	8529	33105

Example :
Exit the programme if the keys up and shift are pressed simultaneously:

```
init()
clrscr()
prints("press up  + second to exit")
while gkey()!=up+second
endwhile
```

Fonction : ptr gets
Numéro de la fonction : 0d117
Syntaxe : **gets**(*buffer*)
Description :
Récupère une chaîne de caractères saisie au clavier jusqu'à temps que la touche ENTER soit appuyée et la copie dans **buffer**. La touche espace est supportée.

Gets() gets a string from the keyboard. **gets()** collects a string of characters terminated by a new line from the keyboard and puts it into *buffer*. **gets()** returns when it encounters a new line (i.e. when the ENTER key is pressed); everything up to the new line is copied into *buffer*. **gets()** returns the string argument *buffer* . For editing, the backspace key is supported.

Fonction : value keydelay
Numéro de la fonction : 0d183
Syntaxe : **keydelay**(*delay*)
Description :
Sets the initial autorepeat key delay for the gkey() and keywait() functions.

keydelay sets the time that a key has to be held down before it starts to repeat to *delay* (note that only few keys have autorepeat feature, like arrow keys and backspace). Measuring unit for this function is 1/395 s (because Auto-Int 1 is triggered 395 times per second), and the default value for *delay* is 336 (slightly shorter than 1 second). Returns previous autorepeat key delay (the default key delay is restored automatically at the end of the execution of the program). The min value for delay seems to be 3.

Fonction : value keyspeed
Numéro de la fonction : 0d184
Syntaxe : **keyspeed**(*rate*)
Description :
Sets the rate at which a key autorepeats for the gkey() and keywait() functions.

keyspeed() sets the rate at which a key autorepeats to *rate* (note that only few keys have an autorepeat feature, namely arrow keys and backspace). The measuring unit for this function is 1/395 s (because Auto-Int 1 is triggered 395 times per second), and the default value for *rate* is 48. Returns the previous autorepeat rate. (the default key speed (rate) is restored automatically at the end of the execution of the program). The min value for delay seems to be 3.

Fonction : long keytest
Numéro de la fonction : 0d56
Syntaxe : **keytest**(*row,column_mask*)
Description :

keytest() est une fonction pour savoir si une ou plusieurs touches sont appuyées simultanément. Pour les touches les plus courantes, il est plus simple d'utiliser les fonctions **up**(), **down**(), **second**(), etc... Lire la description en anglais ci-dessous ainsi que les exemples.

keytest is a function for low-level keyboard reading. It is implemented for simultaneous reading of more than one key (useful in games), or for reading keys when interrupts (int1) are disabled (useful if you want to avoid displaying status line indicators, which are displayed from Auto-Int 1). See below the **up**(), **down**(), **second**()**...** functions for an other way (easier) to test the current keys. They are simpler to use for a simple test of the keyboard but are still based on keytest.
keytest returns the number of column hit by the user if the key is being held down (or if multiple key pressed, return a combinaison of column), and 0 otherwise.
See below for some examples.

Here is a table which describes how the keyboard matrix is organized on both the TI-89 and TI-92 Plus:

TI-89:

C o l u m n

R o w	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit 0	alpha	Diamnd	Shift	2nd	Right	Down	Left	Up
Bit 1	F5	CLEAR	^	/	*	-	+	ENTER
Bit 2	F4	BckSpc	T	,	9	6	3	(-)
Bit 3	F3	CATLG	Z	)	8	5	2	.
Bit 4	F2	MODE	Y	(	7	4	1	0
Bit 5	F1	HOME	X	=	|	EE	STO	APPS
Bit 6								ESC

TI-92 Plus:

C o l u m n

R o w	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Bit 0	Down	Right	Up	Left	Hand	Shift	Diamnd	2nd
Bit 1	3	2	1	F8	W	S	Z	
Bit 2	6	5	4	F3	E	D	X	
Bit 3	9	8	7	F7	R	F	C	STO
Bit 4	,	)	(	F2	T	G	V	Space
Bit 5	TAN	COS	SIN	F6	Y	H	B	/
Bit 6	P	ENTER2	LN	F1	U	J	N	^
Bit 7	*	APPS	CLEAR	F5	I	K	M	=
Bit 8		ESC	MODE	+	O	L	θ	BckSpc
Bit 9	(-)	.	0	F4	Q	A	ENTER1	-

Example 1 :
:Test if CLEAR touch is held down

On Ti89:
Keytest(0b10,0b1000000)

On Ti92 and V200:
Keytest(0b10000000,0b100000)

Example 2:

```
:init()
pause « Press Up and left simultaneously »
:while gkey()!=264
if keytest(1,3)=3 then                    //if you want to test the left key only (on ti89), write if keytest(1,0b10) then
:prints(« Up and Left pressed»)
:endif
:endwhile
:pause « Finish »
```

Test if the keys Left and Up have been pressed simultaneously (for a ti89). Return a number not null if a key has been pressed (one or two keys). This is recommended to not test key on the same column (for a ti89, test for example UP and ESC individually is not sure).

Test si la touche Left et Up a été appuyée simultanément (pour une Ti89). Retourne un nombre non nul si au moins une des deux touches a été appuyée. Tester des touches sur la même colonne risque de ne pas fonctionner (UP et ESC par exemple). Il est préférable de tester des touches qui ne sont pas dans les mêmes colonnes.

Fonction : keyclear
Numéro de la fonction : 0d
Syntaxe : **keyclear**()
Description :
Permet de ne pas afficher à l'écran l'état des touches alpha, second, majuscule ou diamant. En utilisant cette fonciton, les fonctions keywait() et **pause**() sont à proscrire car elle rétabliront d'office l'affichage de l'état des touches. Pour cela, utiliser les autres fonctions de test de touches que sont keytest et autres fonctions **up**(), **down**(),...,**second**(). La fonction **gkey**() ne fonctionnera pas si **keyclear**() a été lancée. Voir la fonction **keydisp**() pour rétablir l'affichage de l'état des touches ainsi que pour pouvoir réutiliser les fonctions telles que **gkey**(). Une autre caractéristique intéressante de cette fonction est qu'elle permet d'éviter le ralentissement constaté de la calculatrice (de l'ordre de 25%) dès qu'une touche est appuyée.

Disable the interupts named auto int1 of the tios. So you will not have the displaying going on of the status line each time a key is pressed (in particular Shift, diamond, alpha, maj). This function is also useful when you want have a maximum of speed. However, if you want to use keys detection for a game in particular, you will have to use only **keytest**() (or **up**(), **down**() etc... functions), because others usual keyboard functions will sometimes restore the displaying mod automatically (**keywait**() and **pause**()) or will not work (**gkey**()).
This function is also interesting because it avoids the common calculator slowing (about 25%) since a key is pressed.

Fonction : keydisp
Numéro de la fonction : 0d
Syntaxe : **keydisp**()
Description :
Réactive l'affichage de l'état des touches. A utiliser si vous avez exécuter **keyclear**() précédemment.

Enable the interupts from the auto int1, so the shift, diamond, alpha and maj sprites will be displayed to the screen when key pressed. You have to use this function if you already have launch the **keyclear**() function.

Fonction : boolean up down lefft rightt second shiftt diamond alpha esc
Numéro de la fonction : 0d
Syntaxe : up() down() leftt() rightt() second() shiftt() diamond() alpha() esc()
Description :
Teste si la touche haut, bas, gauche, droite, seconde, majuscule, diamant ou alpha a été appuyée pendant le lapse de temps que la fonction a été exécutée. La fonction ne stoppe pas l'exécution du programme (de même que la fonction **keytest**()) et contrairement à **keywait**(). Ces fonctions sont plus simples à utiliser que la fonction **keytest()**.

Test if the up, down, left, right, second, shift, diamond or alpha key has been pressed during the execution of the function. These functions don't stop the execution of the program (like the **keytest** function do, but easier to read) unlike the **keywait()** function.

8 Fonctions Tibasic/Asm – Tibasic/Asm functions

Fonction : val disperr
Numéro de la fonction : 0d
Syntaxe : **disperr**(*mode*)
Description :
Disperr() permet d'activer ou de désactiver l'apparition d'une boîte de dialogue d'erreur éventuelle lors de l'exécution d'instructions Tibasic ou de fonctions mémoire. Si *mode*=0, la boîte de dialogue n'interrompra pas l'exécution du programme par l'apparition d'une boite de dialogue. Si *mode*=1, le programme sera interrompu par une boite de dialogue. Dans tous les cas, si une erreur intervient dans une séquence Tibasic, la séquence Tibasic terminera son exécution sur le champ pour laisser la main aux instructions Newprog. Par défaut, l'affichage des erreurs est activé. Retourne l'ancien mode d'affichage d'erreur.

Disperr() allow to activate or desactivate the standard error dialog box pop up when an error occured in a Tibasic program sequence or in a memory function. If *mode* is equal to 0, the error dialog box will appears and will not pause the program execution. If *mode* is equal to 1, the dialog box will pop up and pause the program execution. In all case, if an error occured in a Tibasic sequence, the Tibasic sequence will be terminated and give the hand to the next Newprog sequence. By default, the error displaying is activated. **Disperr**() returns the last displaying error mode.

Example : Will display the error number for each Tibasic sequence executed.

```
init()
disperr(false)          //desactivate the error dialog box pop up
jsr(sequence)           //execute the Tibasic sequences
disperr(true)           //activate the error dialog box pop up
jsr(sequence)           //execute the Tibasic sequences
finish()                //terminate the execution of the program

Lbl sequence
basic → errnum
expr("Noerror")                         //if disperr(false) & disperr(true), will not display an error dialog box
endbasic
clrscr()
Pause "Error num 0:"&string(errnum)     //will print Error num 0:0

basic→errnum
expr("Will--display an error")          //if disperr(true), will display an error dialog box
endbasic
clrscr()
Pause "Error num 1:"&string(errnum)     //will print Error num 1: value!=0

clrscr()
Pause "os(""3"")="&string(os("3"))      //if disperr(false) & disperr(true), will not display an error dialog box

clrscr()
Pause "os(""+-*/"")="&string(os("+-*/"))  //if disperr(true), will display an error dialog box

basich(bas)
Pause "Valid Tibasic instruction"       //if disperr(false) & disperr(true), will not display an error dialog box
endbasic
exech(bas)→errnum
clrscr()
Pause "Error num 2:"&string(errnum)     //will print Error num 2:0
freeh(bas)

basich(bas)
expr("+-*/")                            //if disperr(true), will display an error dialog box
endbasic
exech(bas)→errnum
clrscr()
Pause "Error num 3:"&string(errnum)     //will print Error num 3:value!=0
freeh(bas)

rts()
EndPrgm
```

Fonction : value os
Numéro de la fonction : 0d171
Syntaxe : **os**(*string*)
Description :
Retourne la valeur après exécution de *string*. Ne peut renvoyer qu'un entier ou un pointeur sur chaine de caractère. Le retour de liste n'est pour l'instant pas implémenté (faire manuellement avec la fonction **fopen**()). Attention, les donneés sont temporaires et doivent être sauvegardées pour ne pas les perdre. Cette fonction est lente (environ 33 appels par secondes). Cette fonction est intéressante pour exécuter des instructions Tibasics de façon dynamique voire aussi pour récupérer le contenu d'une variable Tibasic. Pour avoir un accès plus rapide au contenu d'une variable Tibasic, la fonction **osvar**() est préférable.
Si une erreur à l'exécution intervient et que l'affichage d'erreur Tibasic est active (voire fonction disperr()), une fenêtre d'affichage d'erreur apparaîtra.

Return the value return by the execution of the tios instruction *string*. This function return a long integer or a pointer to a string but the return of a list or a nilist is not implemented yet. Be aware that data are not stored definitively in the memory and that you have to store the returned value immediatly. This function is slow (33 calls per seconde). This function is useful when you want to execute simple Tibasic instructions in a dynamic way (ie through a string) or to retrieve the content of Tibasic variable (but you will prefer **osver**() function for this purpose because is many times faster)).
If an error occurred during the execution of the instructions contained in *string*, and if the Tibasic error displaying is active (see **disperr**() function), an error dialog box will pop up.

Example :
:os(« 3→x:x »)
Returns 3
Retourne 3

:os(« x+3+3 »)
Returns the value of the tios variable x + 6.
Retourne la valeur de la variable tios x + 6.

Fonction : value osvar
Numéro de la fonction :
Syntaxe : **osvar**(var_*string*) or **osvar**(tibasic_var)
Description :
Renvoi la valeur d'une variable Tibasic de nom explicitement écrit dans *var_string* (avec la notation "") ou explicitement nommé par *tibasic_var* (sans ""). Cette fonction est plus rapide que la fonction **os**() (1300 appels par secondes) et peut être combinée avec la notation → pour écrire dans une variable Tibasic (voir fonction **toos**()).
Concernant la deuxième syntaxe c'est le seul cas en Newprog ou *tibasic_var* décrit le nom d'une variable Tibasic et non le nom d'une variable Newprog. De ce fait, le passage d'un pointeur sur chaîne de caractères en argument n'est pas possible
Osvar() renverra un nombre si var_string est une expression, ou un pointeur sur chaîne de caractères si *var_string* est de type string. Dans le cas ou **osvar**() retourne un pointeur sur chaîne de caractères, il peut être nécessaire de sauvegarder la chaîne de caractères dans un espace mémoire (fonctions **strcpy**(), **newstr**(), etc...) car le pointeur retourné pas **osvar**() est temporaire.
Remarque : Les variables systèmes ne peuvent pas être récupérer avec cette fonction. Utiliser alors la fonction **os**().

Returns the value of a Tibasic variable whose name is *var_string* (if written with "") or explicitly express by *tibasic_var* (this is the only case in Newprog that a tibasic variable name is expressed without "" notation). In consequence, passing pointer on string as argument is not possible.
Osvar() is faster than the **os**() function (about 1300 calls per second) and can be used with the → **osvar**() notation for writing in a Tibasic variable (see **toos**() function).
Osvar() will returns a number if the Tibasic variable is an expression (a number), or a pointer to a string if the Tibasic variable is a string. In the case that the returned value is a pointer, it can be necessary to store the string in a fix place in memory (with **strcpy**(), **newstr**(), etc...), due to the fact that the pointer is temporary.
Rem : You can't retrieve hidden system variable (like "ok") with this function. You will have to use the **os**() function.

Example : Will display three consecutive dialog box, showing each one the content of a Tibasic variable.

```
Prgm
init()
basic
"Hello" → str
5→num1
 666→num2
endbasic
Text osvar("str")                 //Display Hello
Text string(osvar("num1"))        //display 5
Text string(osvar("num2"))        //display -666
basic
DelVar str,num1,num2
EndPrgm
```

Fonction : value toos or → osvar()
Numéro de la fonction : 0d172
Syntaxe : **toos**(*str_var,entity*) or **toos**(osvar(tibasic_var),*entity*) or **toos**(osvar(str_var),*entity*) or entity → osvar(*tibasic_var*)
Description :
Cette fontion est très utile pour créer simplement des variables Tibasic de type nombre, chaîne de caractères (strings) ou même des listes.
La variable Tibasic aura le nom contenu dans la chaine de caractères *str_var*. Si le fichier existe déjà et qu'il n'est pas verrouillé (par exemple quand il est archivé), la fonction **toos**() retournera 0, aussinon renverra le nombre d'éléments copiés dans cette variable.
Cette fonction peut aussi s'écrire sous la forme usuelle →**osvar**(), mais est légèrement plus limité dans ces fonctionnalités que l'écriture sous la forme **toos**(). En effet, cette notation ne permet pas de passer le nom de la destination via un pointeur sur chaîne de caractères (limitation du compilateur Tibasic). Pour sauvegarder dans la variable Tibasic "x" (*tibasic_var)* par exemple, veiller exceptionnelement que l'argument passé à **osvar**() ne soit pas écrit sous la forme "", autrement dit écrivez directement x et non "x". C'est le seul cas dans Newprog ou x sera considéré comme le nom d'une variable Tibasic et non comme le nom d'une variable Newprog.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).
Le contenu de la variable crée est fonction de l'argument entity. Voici les cas possibles :
1)Entity est un nombre (ou retour de n'importe quelle fonction), alors la variable crée sera du type EXPR
2)Entity est une chaînes de caractères écrite explicitement, alors la variable crée sera une chaîne de caractères STR
3)Entity est une liste non instanciée (nilist) écrite explicitement, alors la variable crée sera du type LIST
4) Si *Entity* est écrite de la forme #pointeur, si pointeur pointe sur une nilist (avec la commande {} → pointeur), alors la variable crée sera de type LIST et contiendra les éléments de la nilist pointée. Autrement, pointeur sera considéré comme un pointeur sur chaine de caractères et la variable crée sera du type STR et sera affectée au contenu de la chaîne de caractères pointée.

This function allow you to create a tios variable with name *str_var* which will be affected to the *entity* value.
The common → **osvar**() notation can also be used, but is a bit limited comparing to **toos**() function but will fits in most cases . In deed, → **osvar**() expects no expicit "" notation for the argument (*ti_basic)* passed to **osvar()**. Say in an another way, the argument *ti_basic* must be the explicit Tibasic name of the output file with no "" notation (this is an execption in Newprog, only true for this function). So, you will not able to created output file whose name is passed through a pointer. This is due to a limitation of the Tibasic compiler.
The element *entity* can have multiple types :
1)If *entity* is a number (or function return) so *str_var* will be a tios EXPR type.
2)If entity is a string written explicitly, so the output file will be a tios STRING.
3) If the entity is a nilist ({,}) written explicitly, so the created variable will be a tios LIST.
4) If entity have the following form : #variable, with variable as a pointer, the created variable will be of the type of variable. Say in an another way, if variable is pointer to a nilit ({,}), so the output will be a tios LIST. Else, the output will be a tios STRING.
The function returns the number of elements that were stored in the output *str_var*. If the function return 0, there was an error during file creation (memory error, file name already existing and archived, or file name restricted).
If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Example1 : With constant output name

The code below

```
:init()
:{1,"two",3} → var
:"Hello" → str
:toos("a",#var)
:toos("b",{1,"two",3})
:toos("c",#str)
:toos("d",expr({0,expr("666")}))
:toos("e",666)
:toos("f","Hello world !")
```

will products the same results than the following more redeable code :

```
:init()
:{1,"two",3} → var
:"Hello" → str
:#var→osvar(a)
:{1,"two",3}→osvar(b)
:#str→osvar(c)
:expr({0,expr("666")})→osvar(d)
:666→osvar(e)
:"Hello world !"→osvar(f)
```

If you enter a or b to the homescreen, you will have :
{1,"two",3}
If you enter c, you will see : (str is a pointer so you have to put the symbol # to store in c tios var)

“Hello”
If you enter d or e, you wil see :
666
If you enter f, you will see :
“Hello world !”

Example2 : Without constant output name – difference between notations

So the **toos**() notation :
init()
“output”&string(rand(10))→ str
toos(str,1)

And the → **osvar**() notation :
“output”&string(rand(10))→ str
1→ **osvar**(str)

The **toos**() notation will created the “outputX” output equals to 1. The → **osvar**() will created the “str” output equals to 1.

Fonction : value seqe
Numéro de la fonction : 0d
Syntaxe : **seqe**(*var,start,end,step,tibasic_var_dest_string,instruction*)
Description :
Créé une liste Tibasic composée de valeur numérique dont le nom est *tibasic_var_dest* (voir son homologue **seqs**() pour la création d'une liste de chaines de caractères). Le contenu de cette liste sera composée de (*end-start*)/*step* éléments. Le premier élément de cette liste sera la valeur retournée par l'exécution d'*instruction* pour une valeur de la variable *var* égale à start. Le deuxième élément sera égal au résultat pour var = var +step. Et ainsi de suite pour les éléments suivants. Cette fonction permet de créer des listes Tibasic sans se préoccuper de sa structure interne (laquelle est détaillée en annexe A). Retourne le nombre d'éléments de la liste.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Creates a Tibasic list *tibasic_var_dest* filled with numbers (see **seqs**() for filling Tibasic list with strings). Each element of the list will be be the result of the execution of *instruction* for *var* going from *start* to *end* with a step of *step*. **Seqe**() is a simplier way to create list than setting each byte one by one on a file (with **fcreate**() function, according to annex 1).Returns the number of elements of the created list
If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Example : Create a Tibasic list x which each element is equal from 1 to 5. Display x (in IO screen) and delete it.
Prgm
init()
seqe(vv,1,5,1,"x",vv)
basic
clrio
pause x
delvar x
EndPrgm

Fonction : value seqs
Numéro de la fonction : 0d
Syntaxe : **seqs**(*var,start,end,step,tibasic_var_dest_string,instruction*)
Description :
Créé une liste Tibasic composée de chaines de caractères dont le nom est *tibasic_var_dest* (voir son homologue **seqe**() pour la création d'une liste de valeurs numériques). Le contenu de cette liste sera composée de (*end-start*)/*step* éléments. Le premier élément de cette liste sera la chaine de caractères renvoyée par l'exécution d'*instruction* pour une valeur de la variable *var* égale à start. Le deuxième élément sera le résultat pour var = var +step. Et ainsi de suite pour les éléments suivants. Cette fonction permet de créer des listes Tibasic sans se préoccuper de sa structure interne (laquelle est détaillée en annexe A). Retourne le nombre d'éléments de la liste.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Creates a Tibasic list *tibasic_var_dest* filled with strings (see **seqe**() for filling Tibasic list with numbers). Each element of the list will be be the result of the execution of *instruction* for *var* going from *start* to *end* with a step of *step*. **Seqs**() is a simplier way to create list than setting each byte one by one on a file (with **fcreate**() function, according to annex 1).Returns the number of elements of the created list
If the calculator has not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Example : Ask for choosing a repertory (in popup menu) then display all files names contained in it (in popup) and returns the path of the file.
init()
reps(rep)→nbreps
seqs(vv,1,nbreps,1,"replist",rep[vv-1])

```
basic
popup replist,x
endbasic
os("x")-1→repsel
files(rep[repsel],file)»nbfiles
seqs(vv,1,nbfiles,1,"filelist",file[vv-1])
basic
popup filelist,x
endbasic
os("x")-1→filesel
clrscr()
pause "File selected : "&rep[repsel]&"\"&file[filesel]
free(file)
free(rep)
basic
delvar replist,filelist,x
EndPrgm
```

Fonction : - basich – exech - freeh
Numéro de la fonction : 0d11 0d12 0d10
Syntaxe : endbasic :...:**basich**(*newprog_var*):*Tios commands* :endbasic :...:**exech**(*newprog_var*):...:**freeh**(*newprog_var)*:...:basic
Description :
Lorsque la fonction **basich()** est exécutée, *Tios commands* n'est pas exécutée. Un handle sera créé référant a *Tios_commands*. Cet handle est stocké dans *newprog_var*. Une fois **basich**() exécutée, on peut alors exécuter ces instructions à l'aide de la fonction **exech()** avec pour argument *newprog_var*.
Comme un espace mémoire est créé par l'exécution de **basich**() (pointé par *newprog_var*), il faut le libérer avant la fin du programme à l'aide de la fonction **freeh()**.
Comparer à l'écriture classique **basic:endbasic**, cette stratégie d'exécution n'a que pour seule avantage de lancer plus rapidement l'exécution de *Tios_commands*. Hormis cet avantage, est a l'inconvénient d'être plus lourde à mettre en place.

When the **basich** function is executed, *Tios comands* is so not executed. An handle will be created and will refer to *Tios commmands*. This handle is stored in *newprog_var*. Since an handle has been created with **basich**() function, you will be able to execute these instructions with the **exech()** function, with *newprog_var* as parameter.
Because a memory space is created by the **basich**() function, you will have to free it with the **freeh**() function at least before the end of the execution to the program.
The **exech**() function is faster than a classical **basic**:**endbasic** sequence, but have the disadvantage to be heavy to set up.

Fonction : ptr fopen
Numéro de la fonction : 0d110
Syntaxe : **fopen**(*filename_str*)
Description :

Copie en mémoire le contenu du fichier *filename_str* (avec les deux premiers bytes de taille en mémoire). Voir annexe pour plus de précision sur la structure d'un fichier. L'espace mémoire créé doit être libéré avant la fin de l'exécution du programme pour éviter les pertes mémoire. Si le fichier existe déjà, il sera remplacé. Si le fichier n'existe pas, retourne 0.

Copy in an allocated memory block the content of the Tios file called *filename_str (with t*he first two bytes of memory length). See in the annex below for informations about the structure of a tios programm. You must free the memory created block before the end of the execution of your program. If the file already exists, it will be replaced. If the file does not exist, return 0.

Fonction : - fcreate
Numéro de la fonction : 0d
Syntaxe : fcreate(*data_ptr*,*file_dest_str*)
Description :
Copie dans un fichier le contenu d'un bloc mémoire. La taille (soustraite de 2) du bloc mémoire devra être enregistrée sur les deux premiers octets de ce bloc mémoire (grâce à la fonction **wb**()). Retourne 1 si effectuer avec succès, aussinon retourne 0.

Copy in a file the content of a memory block. The size (substracted by 2) of the memory block to copy in the file must be stored in the first to byte of the file (using **wb**() function). In case of success, return 1, else return 0.

Example :
I want to store Hello World in STR var named hello :

```
init()
malloc(20)→ptr
```

« Hello world »→ss
strcpy(ptr+3,ss)
wb(ptr,2+1+strlen(ss)+1,0h2d)
wb(ptr,2,0)
ww(ptr,0,strlen(ss)+1+1+1)
fcreate(ptr, »Hello »)
free(ptr)

Fonction : str open
Numéro de la fonction : 0d
Syntaxe : **open**({*filetag1,filetag2*,...,0}) or **open**(*filetag_list_ptr*)
Description :
Ouvre une boite de dialogue Open. Ouvre une boîte de dialogue demandant de choisir un type de fichier parmi {filetag1, filetag2, etc} ou parmi les éléments de la liste codée sur 1 octet *filetag_list_ptr* (la liste doit terminer par un 0) . Si l'utilisateur souhaite sélectionner des fichiers de type personnalisé, soit dérivés du type OTH (comme les fichiers NPP par exemple), il devra placé le type othtag comme élément de la liste suivi d'une chaîne de caractères contenant le nom du type personnalisé. Par exemple, si l'on souhaite afficher les fichiers de type NPP (extension des exécutables de Newprog) ainsi que les images (pictag), la commande sera la suivante : **open**({othtag,"NPP",pictag})
Une fois le type de fichier sélectionné, l'utilisateur peut sélectionné un répertoire parmi ceux disponibles sur la calculatrice.
Au final, l'utilisateur sélectionne un fichier parmi ceux proposés, lesquels correspondant au repertoire et au type de fichier sélectionné précédemment. L'utilisateur valide son choix en appuyant sur ok.
Cette fonction est limitée au types suivants de fichiers : STR,LIST,TEXT,ASM,OTH (NPP, etc...) et PIC. Pour sélectionner d'autres types de fichiers, il est possible d'utiliser la fonction **catalog**().

Avant de lancer cette fonction, veiller à ce que les niveaux de gris ne soient pas actifs sous peine de crash.
La fonction retourne une chaîne de caractères contenant le fichier sélectionné (valable jusqu'au prochain appel de la fonction). Si l'utilisateur à appuyer sur la touche esc ou si le fichier n'existe pas, la fonction retourne 0.

Displays the standard "Open" dialog. **Open**() implements the standard "Open" dialog. The user may select a type, the folder to look in, and finally a file name in the selected folder which matches the selected type. The type will be among the non instantiated list {filetag1, filetag2,...} or among the one byte coded instantiated list pointed by *filetag_list_ptr* (the list must be null terminated). You will be able to select personalize file type, for example NPP type (Newprog executable), by adding othtag as an element of the list, which one just followed by the string "NPP". So, for this example, the command will be : **open**({othtag,"NPP"}).

You can use this function for STR,LIST,TEXT,ASM,OTH (NPP, etc...), PIC file type. If you want select another file type, you can use the **catalog**() function.

Be aware that the grayscale must not be desactivated before launching this function.

The function return a string containing the file selected (available until the next call of the function). If key esc was pressed, the function return 0.

Example : Open dialog box for pictures and NPP files and display the selection.
init()
clrscr()
open({pictag,othtag,"NPP"}) → file
pause file

Fonction : val isarchi
Numéro de la fonction : 0d
Syntaxe : **isarchi**(var_str)
Description :
Retourne 1 si le fichier identifié par la chaîne de caractères *var_str* est archivé, 0 si il n'est pas archivé. Renvoi -1 si le fichier n'existe pas.

Return 1 if the file *var_str* is archived , 0 if not archived. Return -1 if the file doesn't exist.

Fonction : val archi
Numéro de la fonction : 0d
Syntaxe : **archi**(*file1_str, file2_str*)
Description :
Similaire à la fonction Tibasic archive mais en prenant comme argument le(s) nom(s) de fichier(s) sous forme de chaîne de caractères. Cette fonction n'est pas indispensable car on peut faire sans avec la fonction Tibasic citée avant, mais permet néanmoins dans certain cas soulager la programmation.
Retourne 1 en cas de succès. Cette fonction retourne 0 et affiche une erreur si la mémoire archive vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Similar to the Tibasic function archive but with string argument(s). The use of this function can be avoid by using the standard Tibasic function. Anyway, it can helps the user to make more readable program.
Returns 1 in success. If the calculator has not enough archive memory, the function will returns 0. If the error displaying mod is enabled (see function disperr()), an error dialog box will appear.

Fonction : val unarchi
Numéro de la fonction : 0d
Syntaxe : **unarchi**(*file1_str, file2_str*)
Description :
Similaire à la fonction Tibasic unarchiv mais en prenant comme argument le(s) nom(s) de fichier(s) sous forme de chaîne de caractères. Cette fonction n'est pas indispensable car on peut faire sans avec la fonction Tibasic, mais permet néanmoins dans certain cas de soulager la programmation. Retourne 1 en cas de succès, 0 aussinon.

Similar to the Tibasic function unarchiv but with string argument(s). The use of this function can be avoid by using the standard Tibasic function. Anyway, it can helps the user to make more readable program. Return 1 in success, else 0.

Fonction : ptr files
Numéro de la fonction : 0d
Syntaxe : **files**(*rep_str,dest_array_ptr*)
Description :
Cette fonction alloue un espace mémoire qui sera pointé par *var_dest* et sauvegarde dans cet espace tous les noms de fichiers présents dans le répertoire *rep_str*. Les noms de fichiers sont espacés en mémoire de 10 octets entre eux. **Files**() fait en sorte que l'on puisse récupérer le pointeur sur le nom (string) de chacun des fichiers avec la notation *var_dest*[i], ou I est le numéro du fichier désiré (en fait, **files**() effectue en interne un **settype**(*var_dest*,10)). L'équivalent sans la notation [] est *var_dest*+10*i. Cette fonction retourne le nombre de fichiers copiés. Le nombre de fichiers renvoyés est limité à 50.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Stores in *var_dest* a pointer on an allocated memory block containing all the files names contained in the folder *rep_str*. Each files names is written in this memory block all 10 bytes. **Files**() made internally a **settype**(var_dest,10), so you can use the [] notation to retrieve a file name. For retrieving a file name string pointer indentified by i, you can use the [] notation by entering *var_dest*[i].. See example below for an another pointers accessing method (using var_dest+10*rep_index). Returns the number of files retrieved.
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function **disperr**()), an error dialog box will appear.
Example :
Displays all the file names contained in main repertory. The file[ii] writing can be replaced by file + 10*ii.

```
Init()
files("main",file)→nbfile
clrscr()
map(ii,1,nbfiles,1,printf1("%s",file[ii-1]))
keywait()
free(file)
```

Fonction : ptr reps
Numéro de la fonction : 0d
Syntaxe : **reps**(*var_dest*)
Description :
Cette fonction alloue un espace mémoire qui sera pointé par *var_dest* et sauvegarde dans cet espace tous les noms de répertoires présents sur la calculatrice. Les noms de répertoires sont espacés en mémoire de 10 octets entre eux. **Reps**() fait en sorte que l'on puisse récupérer le pointeur sur le nom (string) de chacun des répertoires avec la notation *var_dest*[i], ou I est le numéro du répertoire désiré (en fait, **reps**() effectue en interne un **settype**(*var_dest*,10)). L'équivalent sans la notation [] est *var_dest*+10*i. Cette fonction retourne le nombre de répertoires copiés. Le nombre de répertoires renvoyés est limité à 50.
Cette fonction retourne 0 et affiche une erreur si l'espace mémoire vient à manquer et que l'affichage des erreurs est actif (voir fonction **disperr**()).

Store in *var_dest* a pointer on an allocated memory block containing all the repertory names presents in the calculator. Each repertory names is written in this memory block all 10 bytes. **Reps**() made internally a **settype(***var_dest*,10), so you can use the [] notation to retrieve a repertory name. For retrieving a repertory name string pointer indentified by i, you can use the [] notation by entering *var_dest*[i].. See example below for an another pointers accessing method (using var_dest+10*rep_index). Returns the number of folders retrieved.
If the calculator have not enough memory for allocating memory block, the function will returns 0. If the error displaying mod is enabled (see function **disperr**()), an error dialog box will appear.

Example : Ask for choosing a repertory (in popup menu) then display all files names contained in it (in popup) and display the selections

```
init()
reps(rep)→nbreps
seqs(vv,1,nbreps,1,"replist",rep[vv-1])
```

```
basic
popup replist,x
endbasic
os("x")-1 → repsel
files(rep[repsel],file) → nbfiles
seqs(vv,1,nbfiles,1,"filelist",file[vv-1])
basic
popup filelist,x
endbasic
os("x")-1→filesel
clrscr()
pause "File selected : "&rep[repsel]&"\"&file[filesel]
free(file)
free(rep)
basic
delvar replist,filelist,x
EndPrgm
```

Fonction : asmcall
Numéro de la fonction : 0d
Syntaxe : **asmcall**(ptr)
Description :
This function has not been tested. It's not an interesting one.
Theorically, it's execute the asm code which is placed to *ptr*. The registers are save and restore. This may be usefull if you want to execute assembly programs (<8kbyte) to extend the capacity of NewProg. You will prefer to use **loadasm**() **execasm**() and **closeasm**() functions.

Fonction : ptr loadasm
Numéro de la fonction : 0d
Syntaxe : **loadasm**(*string_name*)
Description :
Load and copy the content of the assembly instructions include in the assembly program which is name is *string_name* toward memory. Return the pointeur of the memory block created. Once you have called this function, you may execute the content of the assembly file by using **execasm**(). You will have to delete the allowed memory space when you will not need to call further by using the newprog function **closeasm**().
You can create your own assembly programm oncalc by using cc or as (take a look on internet). Very powerfull to implement powerfull functions to newprog ! I have not test those functions for an assembly file > 8kb size.
Exemple : Execute a hundred time the assembly file called on the calculator "asm". Very fast execution !

```
:Prgm
:init()
:loadasm("asm")->asm
:repeat(100,execasm(asm))
:closeasm(asm)
:EndPrgm
```

Fonction : - execasm
Numéro de la fonction : 0d
Syntaxe : *execasm*(*asm_ptr*)
Description :
Execute the assembly instructions pointed by the pointor *asm_ptr*. See **loadasm**() for more informations.

Fonction : - closeasm
Numéro de la fonction : 0d
Syntaxe : **closeasm**(*asm_ptr*)
Description :
You would have to use this function if you have before use the **loadasm**() newprog function. See **loadasm**() for more informations.

9 Fonctions interruptions/timers – Interrupts and timers functions

Fonction : inter
Numéro de la fonction : 0d
Syntaxe1 : **inter**(*interupt_ID,tick,{instruction1,instruction2,...}*)
Syntaxe2 : **inter**(*interupt_ID,tick,x):instruction1:instruction2:...:y*
Description :

Cette fonction permet d'affecter et de désaffecter des interruptions. Une interruption est une séquence d'instructions qui, selon les paramètres définis par l'utilisateur, s'exécutera automatiquement **à écart de temps régulier**. Les interruptions permettent en quelque sorte de faire des tâches de fond. Newprog permet de définir jusqu'à 20 interruptions, chacune étant repérée par un identifiant *interrupt_ID* compris entre 1 et 21. L'écart de temps entre chaque exécution de la même séquence d'instruction est égal à : *tick*/18 sec. Si deux interruptions différentes se déclenchent en même temps, celle qui aura l'identifiant le plus faible sera exécutée en première. Les deux syntaxes ci-dessus sont semblables sauf que la deuxième syntaxe permet d'utiliser des commandes autres que des fonctions () tel que les tests **if then else endif** par exemple.
Elle permet aussi d'avoir un programme plus clair car permet de retourner à la ligne. Ne pas utiliser en instruction la fonction **finish**() dans une séquence d'interruption car le résultat est imprévisible. Veiller à ce que le temps d'exécution de toutes les séquences d'interruptions à la suite ne dépasse pas un certain sous peine que la vague d'interruption suivante ne rentre pas en conflit avec la précédente.
Pour stopper temporairement l'exécution d'une interruption, appeler la fonction avec *tick* ayant une valeur nulle (le troisième argument ne sert dans ce cas à rien, il peut ainsi simplement être mis à 0). Pour la réactiver, appeler la fonction avec tick différent de 0 et avec le troisième argument étant littéralement « 0 ».
Pour stopper toutes les interruptions en une seule instruction, appeler la fonction avec un identifiant nul (les autres arguments ne servent à rien dans cas, on peut alors les mettre à 0 par exemple).

Allow the interupt *interupt_ID* to execute the instruction(s) in the {instruction 1, instruction 2,,,,}sequence statement. In case *tick*=1, all the instructions (*instruction 1, instruction 2,...*)will be executed 18/1 times per second (by default of the AUTO-INT5). In case *tick*=2, the instruction will be executed 18/2 times per second, etc... In case *tick*=0, the instructions will not be executed.
You can allow an amount of 20 interupts (from 1 up to 20). The interupts defined with the lower *interupt_ID* will be executed first (and so on...).
I warned you on the fact that the execution time of all the interruptions sequences should not been too long. It is due to the fact that if the next interruption sequence is triggered before the previous one was finished, it can crash your program. So the **keywait**() and others functions should not be used.
You must not quit a newprog program by launching the command **finish**() in an interruption sequence because it will crash. I don't know why, I have not find out the reason for the moment.

To pause an interupt, you have to enter :
Inter(interupt_ID,0,X) where X could be of any type, for example X=0 will be simple. (Inter(interupt_ID,0,0))
To continue an interrupt, you have to enter :
inter(Interrupt_ID,tick!=0,0)
To pause all interrupts :
Inter(0,X,X);

Example : Display a clock, with variable speed

```
Prgm
init()
clrscr()
prints("Second : stop counting"):nl()
prints("Diamond : Continue"):nl()
prints("Left : decrease tick"):nl()
prints("Right : increase tick"):nl()
prints("ESC to exit"):nl()
nl()
prints("Will close in real 30s")

0 → time
18→ rate
0→ reached

printxy(0,92,"Tick = %ld  ",rate)

inter(1,rate,{printxy(10,70,"Time expired : %ld sec",isz(time))})

inter(2,18*30,x)
1→ reached
y

While sto(kk,gkey())!=264 and not reached
 if second() Then
 inter(1,0,0)
 EndIf
if diamond() Then
 inter(1,rate,0)
 EndIf
if kk=leftt and rate>1 Then
 printxy(0,92,"Tick = %ld  ",dsz(rate))
```

```
  inter(1,rate,0)
EndIf
if kk=rightt Then
 printxy(0,92,"Tick = %ld  ",isz(rate))
 inter(1,rate,0)
EndIf
EndWhile
clrscr()
inter(0,0,0)
Pause "finished"
EndPrgm
```

Fonction : long settimer
Numéro de la fonction : 0d
Syntaxe : **settimer**(*timer_no, T*)
Description :

Affecte un timer ayant l'identifiant *timer_no*, avec une valeur intiale égale à T. Une fois cette fonction exécutée, à chaque déclenchement de l'auto-int 5, c'est à dire environ 20 fois par seconde, la valeur du timer sera décrémentée de 1. La fonction **timerval**() permet de récupérer la valeur du timer, tandis que la fonction **timerexp**() permet de savoir si la valeur à atteint 0. Les identifiants des timers vont de 1 à 6 sur tous les AMS disponibles. Cette fonction partage les identifiants du système d'exploitation de la TI89 et c'est pourquoi il faudra sélectionner judicieusement quel identifiant utiliser. De manière certaine, les identifiant 1 et 6 sont libres d'utilisation, ils sont donc à privilégier. le timer 4 est utilisé pour le clignottement du curseur, le timer 2 pour l'extinction automatique de la calculette, le timer 5 pour le délai de la fonction Cyclepic. Settimer() retournera 0 si le timer est déjà utilisé ou s'il dépasse le nombre maximum de timers utilisables (souvent 6). L'exemple ci-dessous permet de modifier le délai avant l'extinction automatique de la calculatrice.

Registers a notify (countdown) timer. TIOS has a 6 notify (countdown) timers, numbered from 1 to 6. **Settimer**() initializes the timer which ID number is *timer_no*, and sets its initial value to *T*. Every time the Auto-Int 5 is triggered (20 times per second if you didn't change the programmable rate generator), the current value of the timer is decremented by 1. When the current value reaches zero (can retrieve value with **timerval**()), nothing special happens, but a flag is set which indicates that the timer is expired. This flag may be check using function **timerexp**().

Settimer returns *timer_no* if the registration was successful, else returns zero. It returns 0 if you give wrong parameters, or if the timer *timer_no* is already in use. You can free a timer using **freet**(). Notify timers 2, 3, 4 and sometimes 5 are used in TIOS for internal purposes, and it seems that timers 1 and 6 are free for use (especially I expected that 6 are surely unused, and I am not so sure for timer 1). Timer 5 is sometimes used for measuring time in some TI-Basic functions like CyclePic. Timer 4 is used for cursor blinking. Timer 3 is used for link communication. Timer 2 is used for automatic power-down (APD) counting, so this is an official method to change APD rate to, for example, 100 seconds:

```
:init()
:settimer(2,100*20)
```

Fonction : freet
Numéro de la fonction : 0d
Syntaxe : **freet**(*timer_no*)
Description :

Libère un timer actif ayant l'identifiant *timer_no.* Retourne 0 en cas d'erreur, 1 aussinon.

Frees a notify (countdown) timer. **Freet**() deactivates and frees the notify (countdown) timer *timer_no*. Returns FALSE (=0) in a case of error, else returns TRUE (!=0).

Fonction : long timerexp
Numéro de la fonction : 0d
Syntaxe : **timerexp**(*timer_no*)
Description :

Détermine si oui ou non le timer ayant l'identifiant *timer_no* a expiré. Retourne 1 si il a expiré, 0 aussinon. L'appel de cette fonction réinitialisera le drapeau d'expiration du timer, la valeur retournée au prochain appel immédiat sera donc 0.

Voir settimer() pour plus d'informations. L'exemple ci-dessous attend 5s avant de quitter le programme.

Determines whether a notify (countdown) timer expired. **Timerexp**() returns 1 if the notify (countdown) timer *timer_no* expired, else returns 0. **Timerexp**() also resets flag which tells that the timer was expired, so the calling this function again will return 0.

See **settimer**() for more info. For example, a legal way to make a 5-second delay is:

Example : Waits 5sec before exiting

```
:init()
:clrscr()
:prints("Ready ?")
:keywait()
:freet(6)
:settimer(6,20*5)
:while timerexp(6)=0
:endwhile
```

Fonction : long timerval
Numéro de la fonction : 0d
Syntaxe : **timerval**(*timer_no*)
Description :

Retourne la valeur du timer identifié par *timer_no*.

Determines a current value of a notify (countdown) timer. **Timerval**() returns a current value of the timer *timer_no*.

10 Fonctions opérations binaires – binary operations functions

Fonction : long lrol
Numéro de la fonction : 0d94
Syntaxe : **lrol**(*expr1,expr2*)
Description :
Effectue un décalage sur la gauche de *expr2* bits sur *expr1*. *Expr2* doit être positif. Les nouveaux bits (sur la droite) seront mis à zéro.

The normal integral promotions are performed on *expr1* and *expr2*, and the type of the result is the type of the promoted *expr1*. If *expr2* is negative or is greater than or equal to the width in bits of *expr1*, the operation is undefined.
The result of the operation is the value of *expr1* left-shifted by *expr2* bit positions, zero-filled from the right if necessary.

Fonction : long rrol
Numéro de la fonction : 0d95
Syntaxe : **rrol**(*expr1 ,expr2*)
Description :
Effectue un décalage sur la droite de *expr2* bits sur *expr1*. *Expr2* doit être positif.

The normal integral promotions are performed on *expr1* and *expr2*, and the type of the result is the type of the promoted *expr1*. If *expr2* is negative or is greater than or equal to the width in bits of *expr1*, the operation is undefined.
The result of the operation is the value of *expr1* right-shifted by *expr2* bit positions.

Fonction : long notb
Numéro de la fonction : 0d95
Syntaxe : **notb**(*expr*)
Description :
Effectue un non binaire sur la valeur *expr* et retourne le résultat.

Perform a binary not on *expr* and returns the result.

Fonction : long andb
Numéro de la fonction : 0d95
Syntaxe : **andb**(*expr1,expr2*)
Description :

Effectue un et binaire entre les valeurs *expr1* et *expr2* et retourne le résultat.

Perform a binary and on *expr* and returns the result.

Fonction : long orb
Numéro de la fonction : 0d95
Syntaxe : **orb**(*expr1,expr2*)
Description :
Effectue un ou binaire entre les valeurs *expr1* et *expr2* et retourne le résultat.

Perform a binary or on *expr* and returns the result.

Fonction : long xorb
Numéro de la fonction : 0d95
Syntaxe : **xorb**(*expr1,expr2*)
Description :
Effectue un ou exclusif binaire entre les valeurs *expr1* et *expr2* et retourne le résultat.

Perform a binary xor on *expr* and returns the result.

11 Fonctions opérateurs et logiques – Operations and logics functions

Fonction : nott, neg, xorr, orr, and, inf, infeq, eq, supeq, sup, noteq, add, sub, mul, div
Numéro de la fonction :
Syntaxe : **nott, neg, xorr, or, and,orb,andb, inf, infeq, eq, supeq, sup, noteq, add, sub, mul, div**
Description :
Ces fonctions peuvent être appelées. Il vaut toutefois mieux utiliser les termes <,= par exemple pour une meilleure lisibilité du code.
La fonction orb() effectue un ou logique entre deux valeur et retourne le résultat de cette opération.
La fonction andb() effectue un et logique entre deux valeur et retourne le résultat de cette opération.

This functions can be call. However, this is more usefull to use the tibasic syntax (=, <, >....=).

Fonction : + - / * ()
Numéro de la fonction : 0d7
Syntaxe : **isz**(*var*)
Description :
Idem Tibasic.

Fonction : long isz
Numéro de la fonction : 0d7
Syntaxe : **isz**(*var*)
Description :
Incrémente la variable NewProg *var*. Après exécution, on aura *var*=*var*+1. Retourne la valeur de *var*+1.

Increment the NewProg variable *var*. After execution, we will have *var*=*var*+1. Returns *var* +1.

Fonction : long dsz
Numéro de la fonction : 0d8
Syntaxe : **dsz**(*var*)
Description :
Décrémente la variable NewProg *var*. Après exécution, on aura *var*=*var*-1. Retourne la valeur de *var*-1.

Decrement the NewProg variable *var*. After execution, we will have *var*=*var*-1. Returns *var* -1.

Fonction : strcmp or =
Numéro de la fonction : 0d
Syntaxe : **strcmp**(s1,*s2*) or s1=s2
Description :

Compare le contenu de deux chaînes de caractères entre elles. La fonction **strcmp**() commence par le premier caractère dans chaque chaîne de caractères et passe au caractère suivant et ainsi de suite jusqu'à ce qu'un caractère diffère ou que la fin de la chaîne de caractères est atteinte. **Strcmp**() retourne les valeurs suivantes :

- < 0 si *s1* est plus petit que *s2*
- == 0 si *s1* est identique à *s2*
- > 0 si *s1* est plus grande que *s2*

Plus précisément, si les chaînes de caractères diffèrent, la valeur du premier caractère qui diffère de s2 moins celui du caractère correspondant de la chaîne s1 est renvoyé.

Cette fonction peut aussi s'écrire en utilisant l'opérateur comparaison "=". Cette écriture peut être plus simple à utiliser et plus lisible que **strcmp**(). La syntaxe est la suivante : s1=s2. Seulement, il faut veiller à ce que s1 soit une chaîne de caractères saisie directement et non un pointeur sur une chaîne de caractères (aussinon le compilateur ne fera pas la différence avec l'opérateur classique de comparaison de nombre). Autrement dit, les commandes "hello"=str ou "hello"="XXX" effectueront bien une comparaison de chaînes de caractères, mais les commandes str="hello" et str=str2 n'effectueront pas une comparasion sur chaîne de caractères mais sur valeur (de pointeur ou de nombre). La fonction retourne 1 si les chaines de caractères sont identiques, aussinon si différente.

Compares one string to another. **strcmp**() performs an unsigned comparison of *s1* to *s2*. It starts with the first character in each string and continues with subsequent characters until the corresponding characters differ or until the end of the strings is reached. **strcmp**() returns a value that is

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

More precisely, if the strings differ, the value of the first nonmatching character in *s2* subtracted from the corresponding character in *s1* is returned.

You can also use the "=" command which is more readable for string comparison. The only restriction is that you must have *s1* hard coded in your program (not through a variable). *S2* can be a string pointer or an hard coded string. The command "hello"=str or "hello"="XXX" will work as a string comparator, but the command str="Hello" or str=str2 will work as classical pointer (or number) comparison.
The "=" string comparison will return 1 if string are identical, else 0.

Fonction : unsigned char value unsigb()
Numéro de la fonction : 0d
Syntaxe : **unsigb**(*data*)
Description :
Convertie une nombre signé codé sur 4 octets (*data*) en un nombre non signé codé sur un octet. Cette fonction trouve son intérêt avec les fonctions de lecture d'éléments de listes (fonctions **lb**() et notation **[]**) car ces fonctions retournent des nombres signés. Si *data* est supérieure à 255, la fonction **unsigb**() retournera le nombre non signé avec un modulo appliqué à ce nombre.
Cette fonction permettra par exemple d'utiliser des listes d'éléments codés sur un octet pour stocker des abscisses de points à tracer à l'écran, sans être limité à une abscisse maximale de 127 (étant la valeur maximale d'un nombre signé codé sur un octet). En effet, l'écran allant en abscisse jusqu'à 159 pixels pour une TI89 ou 239 pixels pour une TI92 ou V200.

Unsigb() converts a number coded onto 4 bytes (*data*) on a signed number coded onto one byte. This function is useful when used with the **lb**() and **[]** notation because it allow the user to return non signed number. If *data* is superior to 255 , unsigb() will return the non signed number with a modulo 256 performed.
For instance, this function can be interresting in the case that you want to retrieve the x coordinates (of a sprite for example) in a one coded element list. By this way, the x coordinates will not be now limited to 127 (as it is the case with all signed values coded on one byte) but by 255 (that is much than 239, corresponding to the max x coordinate of a V200&TI92).

Example : Will display to the screen : 1,-2,-1,254,255,3

```
init()
b(list)
1
-2
255
y
clrscr()
printld(list[0]):nl()
printld(list[1]):nl()
printld(list[2]):nl()
```

```
printld(unsigb(list[1])):nl()
printld(unsigb(list[2])):nl()
printld(unsigb(259)):nl()
keywait()
```

Fonction : unsigned word value unsigw()
Numéro de la fonction : 0d
Syntaxe : **unsigw**(*data*)
Description :
Convertie une nombre signé codé sur 4 octets (*data*) en un nombre non signé codé sur deux octets. Cette fonction trouve son intérêt avec les fonctions de lecture d'éléments de listes (fonctions **lw**() et notation **[]**) car ces fonctions retournent des nombres signés. Si *data* est supérieure à 65535, la fonction **unsigw**() retournera le nombre non signé avec un modulo appliqué à ce nombre. Voir l'exemple de la fonction **unsigb**().

Unsigw() converts a number coded onto 4 bytes (*data*) on a signed number coded onto two bytes. This function is useful when used with the **lw**() and **[]** notation because it allow the user to return non signed number. If *data* is superior to 65535 , **unsigw**() will return the non signed number with a modulo 65536 performed. See the example to the **unsigb**() function.

12 Fonctions Diverses – Other functions

Fonction : value rand
Numéro de la fonction :
Syntaxe : **rand**(*num*)
Description :
Retourne un nombre aléatoire compris entre 0 et num-1,

Generates a random number between 0 and (*num*-1).

Fonction : value catalog
Numéro de la fonction :
Syntaxe : **catalog**()
Description :
Affiche le menu catalog. Retourne une string de la sélection. Ce pointeur sera valable jusqu'au prochain appel de la fonction.

Display the catalog menu. Returns a string containing the selection. This string will be available until the next call of this function.

Fonction : - off
Numéro de la fonction :
Syntaxe : **off**()
Description :
Eteint la calculatrice.

Turns off the calculator.

Fonction : 0 debugon
Numéro de la fonction : 0d240
Syntaxe : **debugon**()
Description :
Active le mode de déboguage. L'exécution du programme sera alors en mode pas a pas. Le nom de la fonction sera affiché sur l'écran et permettre ainsi peut être de comprendre l'erreur commise. Certain bouclage ne seront pas exécuté en mode pas à pas (bloc x:y par exemple). Il est interdit d'exécuter la fonction debugon() à l'intérieur d'une fonction. Pour exécuter pas à pas une fonction, il faudra exécuter la fonction **debugon**() avant de l'éxécuter (dans le flux principal).

Set the debug mod. The execution of the program will be now in the step by step mod. Some block will not be executed step by step (like x:y block). You will not be able to execute **debugon**() in a function (will throw a compilation warning). If you want to run step by step a function, execute **debugon**() before launching the function (in the main thread).

Fonction : - debugoff
Numéro de la fonction : 0d241
Syntaxe : **debugoff**()
Description :
Désactive le mode d'exécution pas à pas (voir **debugon**()). Retourne en mode d'exécution normal.

Set off the debug mode (**debugon()**). Return in normal execution mod.

Fonction : - #

Numéro de la fonction : 0d14
Syntaxe : *#nilist_ptr*
Description :
Cette fonction permet d'exécuter tous les arguments d'une liste non instanciée laquelle est pointée par *nilist_ptr*. Cette fonction retourne la valeur de la dernière instruction exécutée.

This function execute all arguments of the nilist pointed by *nilist_ptr* and returns the result of the last executed instruction.
You can't type #{...,...} directly. You must use a pointer to trick the Tibasic compilator.
Example :
:init()
:{keywait()}→var
:printf1("%ld",#var)
:keywait()
This will execute the keywait() instruction (waiting for a key) and print the key number on the screen.

Fonction : - nop
Numéro de la fonction : 0d250
Syntaxe : **nop**()
Description :
Ne fais rien.

Do nothing.

13 Instructions interdites, restrictions – Forbiden instructions, restrictions

Ils est interdit d'utiliser les instructions suivantes :
les noms de variables de a à z, ainsi que les lettres grecques. La fonction local.

It is forbiden to use the following instructions :
The vars name made with only one character (from a to z, including Grec character). The local function.

14 Rapport d'erreur oncalc – Oncalc error report

Lorsqu'une erreur de compilation est détectée, le compilateur renvoie un fichier de sortie (npout) décrivant l'erreur et donne des indices quant à sa localisation dans le programme (dernière fonction et variable traitée).

Lorsqu'une erreur est détectée lors de l'exécution d'un programme, il est conseillé de réinitialiser la calculette (reset) afin de résoudre les problèmes de fuites de mémoire qui peuvent survenir si les fonctions seqb, seqw, seql, group, openfile, listes instanciées {var_name,...} et createfile ont été exécutées.

Attention : Afin de ne pas perdre de données personnelles, veillez à archiver toutes vos données importantes avant d'effectuer la réinitialisation de la calculatrice (reset par appui simultanée sur 2nd, gauche, droite et on).

When the compilator detects an error, it throws an output file describing the error (npout). The lasts function and variable name treated by the compilator are quoted so it will help you for searching the error in the program.

When an error is detects during the execution of your programm, this is advise to reset the calculator because you have probably lost ram memory available, in particular if you have used the seqb, seqw, seql, group, openfile, instantiated lists {var_name,...} and createfile functions (Don't forget to archive your data before reseting the calculator).

15 Constantes prédéfinies – Predifined constants

Lors de l'édition d'un programme dans l'éditeur de programme, afin de facilité la programmation, il est possible d'utiliser des constantes prédéfinies. Taper le nom d'une constante prédéfinie dans votre code source et alors elle sera remplacée directement par sa valeur lors de la compilation.
Vous ne pouvez pas utiliser de variable correspondant au nom d'une constante prédéfinie car il y aurait ambiguité.

Some Newprog variable names will be replace automatically by a special value during the compilation. It will helps you to avoid to remember difficults values. They are named predefined constants. These variables names are so restricted.

Predefined constants/Liste des constantes prédéfinies :

“false”	0
“true”	1
"lcdsize"	3840
"strtag"	0h2D
"listtag"	0hD9
"texttag"	0hE0
"asmtag"	0hF3
"functag"	0hDC
"posexpr"	0h1F
"negexpr"	0h20
"exprtag"	0
"othtag"	0hF8
“pictag”	0hDF
“onesec”	18
“gor”	0
“greplace”	1
“gxor”	2
“greverse”	1
“gand”	3
“gerase”	1

TI89 only :

"up"	337
"down"	340
"rightt"	344
"lefft"	338
"shifft"	8192
"second"	4096
"diamond"	16384
"alpha"	32768
“esc”	**264**

TI92/V200 only :

"up2"	**338**
"down2"	**344**
"rightt2"	**340**
"lefft2"	**337**
"shifft2"	**16384**
"second2"	**4096**
"diamond2"	**8192**
"alpha2"	**32768**

Example :
Exit the programme if the keys up and shift are pressed simultaneously:

```
init()
clrscr()
prints(“press up  + second to exit”)
while gkey()!=up+second
endwhile
```

16 Divers – Miscalenous

1)Passage d'arguments à une instruction :

Le compilateur vérifie si le nombre d'arguments passés à une fonction est correct en affichant un warning. Conscient de ce message d'alerte, l'utilisateur choisit lui même d'exécuter ou non le programme. Le choix est laissé à l'utilisateur car il peut être dans l'intention du programmeur d'avoir sorti un argument en dehors des parenthèses pour une question de lisibilité (afin d'éviter par exemple d'avoir une ligne de code trop longue en la scindant en deux lignes ce qui sera plus lisible). Il est préférable de veiller à écrire les fonctions sous leurs formes standard (et ainsi d'éviter les warnings).

Exemple :
La fonction dline utilise 4 arguments (dline(xa,ya,xb,yb))
Exécuter dline():0:0:50:100 est équivalent à exécuter dline(0,0,50,100)

1)Passing arguments to an instruction :

The compilator verify if the number of arguments passed to an instruction is correct or not and display a warnig if not. The user has the choice to execute anyway his program. It can be in the intention of the programmer to have written an argument out of the () for lisibility reasons (for avoiding to have a too long line for example).
Example :
The function dline uses 4 args (dline(xa,ya,xb,yb))
Executing dline():0:0:50:100 is similar to execute dline(0,0,50,100)

2)Récupérer le numéro d'une erreur lors de l'éxécution d'instructions Tibasic (avec basic-endbasic)

Les blocs Try:else:endtry ne fonctionnent lors de l'exécution de séquences Tibasic, l'on ne peut donc pas récupérer le numéro d'une erreur par cette méthode. Dans ce cas, le numéro de l'erreur Tibasic sera retounée par le mot clé **basic**. Voir l'exemple ci-dessous. Voir la fonction **disperr**() pour un autre exemple plus complet.

2)Retrieve the error number after the exectution of Tibasic instructions (with basic-endbasic block)

The use of the classic Try:Else:Endtry syntax will note fit for retrieving the error number at the execution of Tibasic instructions. So, the only way to retrieve the error number after Tibasic instructions execution is to look at the value returned by the key word **basic**. If the returned value is null, no error occurred. If not null, an error occurred. You can use for instance the → (**sto**()) function to store the error code in a Newprog variable. See the example n°2 below. See function **disperr**() for a more interesting example.

Example n°2 :
```
init()
clrscr()
disperr(0)
basic → errnum
expr("++")
endbasic
if errnum!=0 then
pause "There was an error during execution. Error number="&string(errnum)
endif
```

17 Remerciements - Thanks

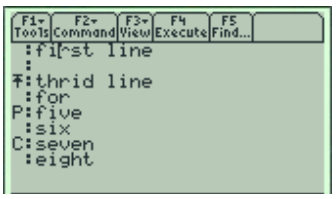
Merci à tous ceux qui ont contribués à mettre en place la programmation ASM et C sur TI68K.
Merci à TIGCC et à la TICT

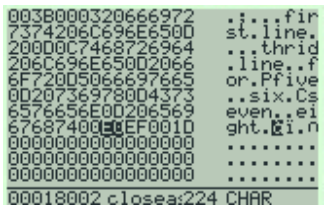
Thanks for TIGCC and extgraph (by the Tichess Team).

Annexe 1 - Structures des fichiers - Commons files structures

You can see below how is defined common files on a Ti89-Ti92-V200 calculators. I have used an hexadecimal editor : **mtihex**() (joined in the bundle). It will help you to make your own files or to read existing TIbasic data.

TEXT files example





Ti89 text editor Hexadecimal editor

From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h003B (16) = 59 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 0h00003B + 0h2 = 59 + 2 = 61 bytes)
2)Bookmark position : for this example 0h0003 = 3
3)Line struct :
3-1) first byte of the line struct corresponds to the command (F2) :
20 =no command
0h0c=page break
0h50=Print Obj
0h43=command
3-2) Line content (can have no byte for an empty line)
3-3) Endline tag = 0h0D = 13 or =0 for the last line
4) File type : TEXT Tag=0hE0 = 224.(its adress is equal to first byte adress of the file+size of the file (here 59)+1)

OTH files example

From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h0025 (16) = 37 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 37 + 2 = 39 bytes)
2)Data bytes (put what you want)
3) Optional sequence : Specify a personalize type for the file :

- 0h00
- Name of the type. For example {N,P,P}
- 0h00

4)File type : OTH_TAG = 0hF8 = 248 (its adress is equal to first byte adress of the file+size of the file (here 37)+1)

ASM files example

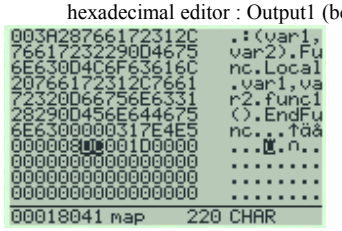
Similar to OTH files but with file type = ASM_TAG = 0hF3 = 243.

FUNC and PRGM files example

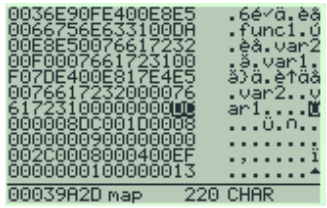
For this file type, when editing the file (ff() for example) with an hexadecimal editor, we can have two type of output :

TI89 Func editor (example for a function called ff(), replace Func and Endfunc by Prgm and Endpgrm for a program)

hexadecimal editor : Output1 (before TI os compilation (or before launching the func if you have just before edited the function in



the function editor)



hexadecimal editor : Output2 (after TI os compilation (ie file execution)

Output 1 struct (before compilation)

From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h003A (16) = 58 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 58 + 2 = 60 bytes)
2)Line struct :
2-1) Line content (can have no byte for an empty line)
2-2) Endline tag = 0h0D = 13 or =0 for the last line
3) Ending sequence :
3-1) Bookmark position on two bytes, on the example = 0h0003
3-2) Byte = 0h17, you must enter this value if you want to have a FUNC type (it also a mean to differanciate FUNC type files ans PRGM types files, see further below). If you want to have a PRGM file type, you will have to enter 0h19 instead of 0h17.
3,3) Four bytes = {0hE4 ,0hE5, oh00, 0h00, 0h08} (don't know why this values)
4) File type : FUNC Tag=0hDC = 220 (for both FUNC and PRGM file type).(its adress is equal to first byte adress of the file+size of the file (here 58)+1)

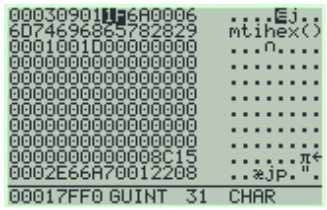
Output 2 struct (after compilation (ie after running))

From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h0036 (16) = 54 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 54 + 2 = 56 bytes)
2) Func codebyte (will not be explained by me because very complicated)
3) File type : FUNC Tag=0hDC = 220.(its adress is equal to first byte adress of the file+size of the file (here 54)+1)

EXPR files example

case 1 : The var x contained a number (here, x=9)

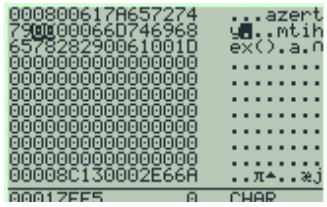
For the variable x = 9.



Hexadecimal editor

case 2 : The var x contained a var name (here azerty)

For the variable x = azerty.



Hexadecimal editor

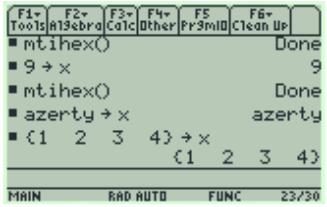
For this data type, if you want to read the value contained in this file, you have to read from the last byte of the file (the one who have the higher address to the one with the lower address) to the lower address of the file.

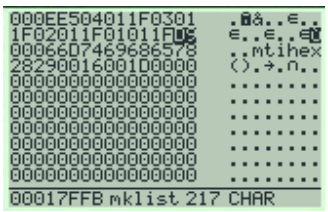
To determine the position of the last byte, you will have to addition the first byte address of the file with the size of the file -1.
The size of the file is equal to the content of the first two bytes coded value of the file, in this example +2. For this example, the size of the file is size = 0h00003 + 2 = 5 bytes length.
Knowing the size of the file, you can calculate the address of the last byte :
last_byte_adress = first byte address of the file + size – 1.

So, going from the last byte to the first byte :
1) Last_byte : corresponding to the value_type of the data stored in the file (it will always be shown with “EXPR” tag in the ti-varlink and in Newprog too (with the gettype() function).
The value_type can have this values :
0h1F : positive data value (case 1)
0h20 : negative data value (case 1)
0h00 : variable (in symbolic form) (for example, when : azerty->x) (case 2)
2)DATA sequence (always going from the last byte to the first byte) :
2-1) In case of positive data value or negative data value (case 1)) :
- first following byte : number of byte data length (here 1 because 9 is <255 so can be coded on only one byte)
- following bytes (their number is stored in the previous byte) for the data value storing. (in big endian if you read from the lower address to the upper address or in little endian when reading from the highest address to the lower address).
2-2) If the variable doesn't contained a value but a var name (case 2), the data sequence contains the name of the variable.
3) The last two bytes contained the size of the file -2 (in the same way than other files).

LIST file example

A list contained a multiple of EXPR.



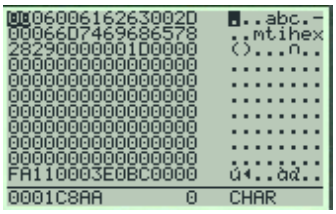


1) The first two bytes content is equal to the size of the file -2.
2) 0hE5 = endtag (specify the end of the list)
3) Put EXPR ; STRING or LIST sequences (In the same way than in this document, but wihout the two byte of the size.
4) The last byte of the file is equal to 0hD9 = 217.

For reading one by one each values contained in the list, you have to go from the end of the file to the beginning of the file. After the last byte (which is localized at the first byte address + size – 1) you will have the tag of an EXPR data type. To read this value, see the EXPR data sequence written above. Just after this data sequence, you will have the tag of the second EXPR data type. So, by this way, you can fetch each value contained in the value one by one by going going through the file. The end of the list is indicated by the tag 0hE5.

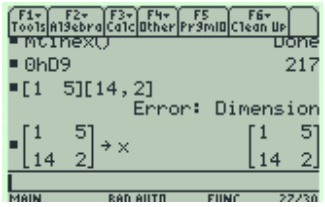
STR file example

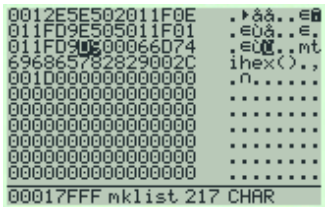
1) First two bytes = size of the file – 2 (size of the file must be equal to the size you will see in the varlink)
2) 0h00
3) Put your string with the end null character
4) **STR_TAG = 0h2D**



MATRIX file example

A matrix is a list of LIST. Here below an example :





PIC file example

1) The first two bytes content is equal to the size of the file -2.
2) The next two bytes contained the number of rows (for instance = 0h0009 for a pic containing 9 lines)
3) The next two bytes contained the number of columns in pixels
4) Bitmap sequence. For a all black picture of 8 rows x 8 columns, we will have : 0hFFFFFFFFFFFFFFFF
5) Pic tag = 0hDF

Annexe 2 - Historiques – Releases improvements

03/2010 : Newprog Release 1.0 – Improvements since release V0,1

Now Newprog assumes that the first instructions in the program are Newprog instructions (instead of release 0,1).
Keys words end and start have been renamed by **basic** et **endbasic**.

A memory leak in qbasic has been corrected.
Now, qbasic verify whether the file passed is really a NPP file => Crash corrected.
Now, if an error occured in a Tibasic instruction, the program will not been stopped by an error displaying (nothing will happen).

In clrlcd() : Great speed impovement of about 17 time faster (now 6500 calls / sec)
In clrscr() : Great speed impovement of about 24 time faster (now 6000 calls / sec)
In savescr : Great speed improvement of about 5 time faster.
In loadscr : Great speed improvment of about 5 time faster.
In dline : Great speed improvement (up to 10 times faster)
In fillrect : Great speed improvement (16 time faster than previous routine).
In sprt8,sprt16,sprt32 : Speed increased of about 71%. Now worked as clipped (ie will not crash if coordinates are out of [0,0,239,127]).
In dpix : Great speed improvement.
In gpix : Speed increased of about 30%.
In memcopy : Execution speed is about 6.5 time faster for a an amount of bytes to copy at least equals to 384 bytes.
In while:endwhile : speed increased of about 30%
In Tibasic sequence : A bug has been corrected when size of the sequence was >255 bytes.
In def, enddef, fc : A bug has been corrected when at least 30 variables was defined.
In jsr : A bug has been corrected when at least 30 variables was defined.
In rts : A bug has been corrected when launch without previous jsr().

ifs() renamed by when().
chars() renamed by char(). The return value is now available in memory up to the next call of this function (you don't have to free the return value before the end of the program).
code() renamed by ord().
int1off() renamed by keyclear().
int1on() renamed by keydisp().
fopen(), great modifications : The function is totaly modified.
or , and and not functions work now has logic test and not test on binary operations (see new functions orb(), andb() , notb() for binary operations).
While:Endwhile : a bug has been corrected (when two endwhile were closed).
gmode : graphic mode value has been change for real compatibility between all graphics functions.
keywait() : a bug is corrected (when using with keyclear() (previous name of keyclear : int1off()).

New Functions :
= for strings comparaison (simplier to use than strcmp()). Return 1 if equal, else 0.
andl : binary and operation (former and operator in newprog release v0.1).
orl : binary or operation (former or operator in newprog release v0.1).
sprt82 : display a sprite even if in the border of the screen.
sprt162 : display a sprite even if in the border of the screen.
sprt322 : display a sprite even if in the border of the screen.
newline or nl : Return to a new line (with the printf1... functions).
when : replaced the ifs functions in release 0.1
i : is a simple writing for if endif function (with no else for the moment). As fast as an if endif statement. Use the following syntax : i(condition):instructions:y
catalog : Launch the catalog dialog and return a string of the selection.
For:Endfor
open : display an open dialog box. The user select the folder and the file (and can choose the type). Returns a string :"rep\varname"
mod : modulo function. Example : mod(9,7) returns 2.
atol : convert a string to number (like expr in tibasic)
& : similar to tibasic. The output shall be smaller than 50 bytes for avoiding crash. If the size is bigger than 50 bytes, use strcat() instead.
lcdup : contrast up.
lcddown : constrast down.
b,w,l : 3 functions to define predifined instantiated list.
prettyxy : display an expression into pretty print format (very impressive).
getwbt : retrieve the dimension of the expression to be displayed with prettyxy.
settype : set the type of data (ie their size) contained in a list pointed by a Newprog variable. Returns the old value. The function could just returns the old value without modifying it.

moveto : Sets the current pen position.
pause : Display a string to the screen and wait for a key to be pressed
seqs : Great function for making Tibasic list filled with strings
seqe : Great function for making Tibasic list filled with expressions (ie number only)
orb : binary or
andb : binary and
notb : binary not
fopen(file_str) : open an existing file for reading and modificating. If you want to save your modifications, you have to use the fcreate function.
fcreate(file_str, string_type) : create a file or overwrite a file
next : returns to the beginning of the last opened and active For or While loop.
break : Ends the last opened For or While loop.
esc : acts like up() down() etc but for escape key.
off : turns off the calculator
isarchi : Return 1 if file is archived, 0 if not, -1 if doesn't exist.
archi : archive a file.
unarchi : unarchi a file.
gsprt8x : retrieve from the screen a sprite with custom bytewidth.
sprt8x : Display to the screen the sprite (in particular from gsprt8x).
dcircle : Draw an outlined circle.
fillcirc : Draw a filled clipped circle.
memchr : Search for a character in a memory area.
text : Same function than the Tibasic one.
unsigb : converts a number coded onto 4 bytes (data) on a signed number coded onto one byte
unsigw : converts a number coded onto 4 bytes (data) on a signed number coded onto two bytes
osvar : Return the value of a Tibasic variable
if : if cond:if_true
ord : converts a string of one character to a code ascii

New : Predefined constants (operating during compilation)
Functions removed because do not work correctly : fputc();ftell();fseek();fputs();fclose();interoff();interon();int5off();int5on() etc few others ones.

Program stability enhanced (by verifying whether the function executed is still in the programm memory area(C exec_non_stop_secured() function)).
Program stability enhanced when executing Tibasic.
Memory leak corrected.
In jsr, bug corrected when to many jsr called (memory problem).

In the compilator point of view :

The first intruction of the program is assumed to be a Newprog instruction and not a Tibasic one like it was the case in the previous release.
An error about the compilation of the writing [] has been deleted. Now the list[1,prints("Crazy")] will be executed as list[x]:prints("crazy").
The compilator compares if the number of functions allocating memory is equal to the amount of free() function detected. If amounts not equal, the compilator displays a warning.
The compilator now displays a warning if the number of arguments passed to the program is wrong.
The compilator not crash if an unknown Tibasic command is used.
We can now put variables and predefined constants in the definition of a dynamically instantiated list. Example : {varn,4,1,7,title,4,strtag}
New instantiated list : b,w and l.
Can now launch a function with the () notation (with possibility to pass locals variables)
Bug corrected in the error output. The last foncname and var name and line number (where the error occured) are now correctly displayed.
Some errors message written in french have been rewritten in english.
If a variable is alone, a warning will be displayed.
if : if cond:if_true notation (Like in Tibasic)
Add : false (value=0) and true (value=1).
Predifined constants routine defined.
@ notation for comments
Speed optimizations :
Now, if you write ord("a"), it will be written internally 97. And so on...

06/2009 : Newprog Release 0.1 (beta version) :

Mains Newprog features defined.

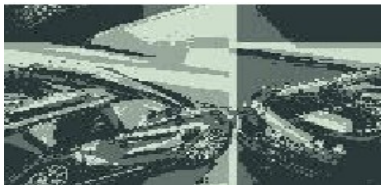
Annexe 3 – Captures d'écrans – Screens captures

Newprog examples :

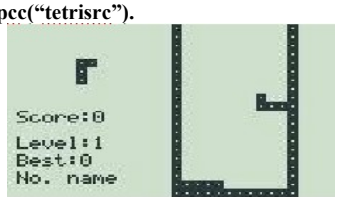
How to lauch this program :
Enter the command in the Home screen : npcc(“bugatti”)

NB : After the compilation, an output file named “out.NPP” is created. You can execute it without recompiling your program by entering newprog(“out”). You can rename out.NPP as your conveniance. So if you want just to play without programming in Newprog, just keep in memory the output file.

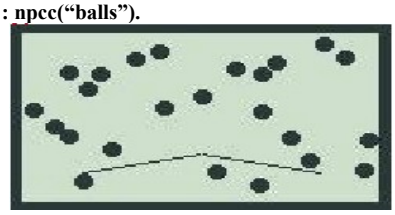
bugatti.89p : Démonstration graphique : scrolling + niveaux de gris
Exécution après compilation du code source : npcc(“bugatti”). Les images bugatti0 et bugatti1 doivent aussi être copiées dans le répertoire courant (généralement main).



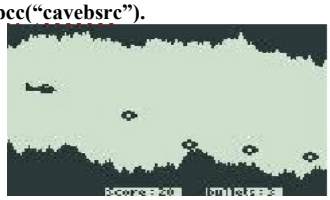
Tetriscrc.89p : Le jeu célèbre
Exécution après compilation du code source : npcc(“tetrisrc”).



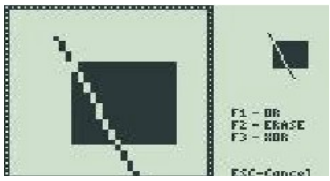
balls.89p : Démonstation graphique gerant des collisions.
Exécution après compilation du code source : npcc(“balls”).



cavebsrc.89p : Survivez le plus longtemps dans une grotte, aidez vous de vos missiles pour dégager le passage.
Exécution après compilation du code source : npcc(“cavebsrc”).



spritemk.89p : Un éditeur de sprite puissant. A utiliser pour vos création Newprog.
Exécution après compilation du code source : npcc(“spritemk”).


F1 - OR
F2 - ERASE
F3 - NOR