NewProg

Sommaire :

Purposes – Roles de NewProg

NewProg is a new programmation language based on the Tibasic. The Tibasic syntax is sometimes allowed in NewProg (for example the if then else endif block, 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 Tibasic ans NewProg instructions by simply adding a **start** and **end** block for programming in NewProg.
Each program containing Newprog instruction shall contains the init() instruction at the very beginning of the program (for compilation purposes).

Struct of a New program :

```
Myprgm()
Prgm
init()                                  //Note : this file must not exist in your varlink
```

Pause « Tibasic instruction »
start
prints(« NewProg instruction »
end
pause « Return to Tibasic »
EndPrgm

A NewProg programm can be edit in the Tibasic program éditor.

How to run a NewProg program :

You may have installed an os (like Preos for example) to use Newprog.

First, you have to compil the program by launching in the Homescreen the following command :
npcc(« prgm ») (or with **qbasicc(« prgm »))**
After the compilation, the program will ask you if you want to execute your program. Press enter if you are okay or press ESC if not. If have pressed ESC, you will be able to run your program by entering in the homescreen : « **qbasic(« out »)** ».

You can rename the output file with the name of your conveniance. If you have a compiled file named *runable* for example, you can execute it by entering in the Home screen the following command :
qbasic(« *runable* »)

Important : I am not responsible for any damages and data loses of your calculator.

1 Types de données –Data types

Les variables (se nomment comme les variables TiBasic, par exemple va). Jusqu'à 255 noms de variables sont utilisables.
Toutes les variables de newprog sont codés 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 toutes types de données

Les noms de variables à seul caractère sont interdits (par exemple *x,* d'ailleurs comme toutes les variables Tibasic).

Les listes :
Certaines commandes ne sont pas des variables mais s'en rapprochent :

- Les listes instanciées :

  Syntaxe : {*nom_de_la_variable,taille_des_éléments,élément1,élément2…*}

  Lors de l'exécution d'une liste instanciée par l'applet newprog, un espace mémoire est créé. Cet espace mémoire est alors initialisé comme une liste de *nombre_d'éléments* 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*.On peut lire ou écrire dans cette liste avec les fonctions lb(*taille des éléments*=1), lw(*taille des éléments*=2) etc pour la lecture et wb, ww etc pour l'écriture.On peut aussinon utiliser les crochets classiques du Tibasic []. Exemple : Pour lire le premier élément d'une liste instanciée *ll* avec une taille des éléments égal 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 commode d'utiliser les listes instanciées (et plus lisible aussi).

  Le pointeur de cet espace mémoire est alors affecté à la variable *nom_de_la_variable* et est retourné à l'exécution de l'instruction. Il faut veiller alors à libérer cet espace mémoire avant la fin du programme pour ne pas encombrer la mémoire de données inutiles (avec la fonction **free**(). Voir plus bas).

  Les fonctions seqb,seqw,seql,group permettent aussi de créer des listes instanciées (voir chapitres plus bas).

- Les structures 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 juste un pointeur utile pour la manipulation de cette structure. On pourra ainsi effectuer diverses opérations tel que l'exécution des éléments, la création d'une liste composée des renvois des éléments, le passage de plusieurs arguments à une fonction…
  Cette structure permet de passer des plusieurs argument à une fonction avec la fonction **group()** (possibilité de récursivité) , de passer des liste à une variable Tios mais pourquoi pas aussi d'exécuter plusieurs fois les éléments *element1, element2*... (avec #);cette méthode étant plus lisible que l'utilisation 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. Si vous êtes obligé de placer une variable en première position (sans que le compilateur la considère comme une liste instanciée), utiliser l'astuce suivante :
{nom_variable+0,élément2….}
Ainsi le compilateur sera trompé.

Variables (can be named in the same way than Tibasic, for instance va). You can use up to 100 variable names :

All the data var in NewProg programmation are 4 bytes long. Each data var can be of the following type :

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

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

Lists (Arrays) :

Few commands are not data var but are quite similar :

- Instanciated lists (Array) (created by seqb,seqw,seql,group functions).
  Syntax : {*variable_name,size_in_byte_of_an_element,element1,element2…*}
  During the execution of an instanciated list by the NewProg applet (qbasic.89z), an allocated memory block will be created. The first element of the list will be the number of elements contained in the list. The second element of this array will be *element1* and so on. 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() or peekb() and similar functions. You can also use the Tibasic syntax by using [] for reading or writting. NewProg 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 instanciated list, tape the following instruction : lb(*variable_name*,0) or ll[0]. For writting, you can just write for instance : ll[0]*2->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 lose of memory.
  The seqb,seqw,seql,group functions are also used to create instanciated lists.
- SyntaxNon instanciated lists
  Syntax : {*élément1,élément2…*}
  If you launch a non instanciated list without using the command **#** or **group**(), no instructions will be executed. NewProg will just returns a pointer that will be useful if you use the **#** or **group**() functions. This struct will be useful for example to pass argument to a program or to execute *element1, element2* when you want or to sto a list in a tios variable (so you can communicate with Tibasic).

Be cautious, do not put a variable name in the first element of the nilist because the compilator will assume that this is an instanciate list. You can bypass this problem by using the following trick or tip :
{*var_name+0,élément2….*}

2 Fonctions de sauts - jumps

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 : - 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.

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 the standarts **lbl** and **goto** functions.
Exemple :
```
:init()
:start
:lbl abc
:goto abc
:end
```
Est équivalent à – Is the same than :
```
:init()
:start
:pos()->position
:gotopos(position)
:end
```

Fonction : ptr pos
Numéro de la fonction : 0d81
Syntaxe : **pos**()
Description :
Renvoie la position dans le programme juste après la position de cette fonction.
Return the position in the program just after the position of this function (**pos**).

Fonction : - jsr
Numéro de la fonction : 0d82
Syntaxe : **jsr**(*nom_label*)
Description :
Saute vers la sous routine se trouvant à la position *nom_label*.
Make a jump to the sub routine which is placed at the position *nom_label.*
Exemple :
```
:init()
:start
:0->x
:While x<10
:isz(x)
:jsr(abc)
:Endwhile
:printf1(«abc launched ! »,0)
:keywait()
:finish()
:lbl abc
:printf1(« Dans abc ! x=%ld »,x)
:rts()
:end
```

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 : - nop
Numéro de la fonction : 0d250
Syntaxe : **nop**()
Description :
Ne fais rien.
Make nothing.

Fonction : value fc(*function_name,arg*) ; and also def , enddef, arg, rtn
Numéro de la fonction : 0d
Syntaxe : **fc**(), **def**() , **enddef**(), **arg**(), **rtn**()
Description :
Execute la fonction inline *function_name* (cad définie par la fonction **def**()). L'argument *arg* sera accessible dans la fonction grâce à la fonction **arg**(). 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. Il est

possible de faire des fonctions récursives ! (30 degrés d'imbrication max). Il n'est pas possible d'avoir des variables locales affecter à une fonction, mais voir l'astuce plus bas !

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()
:start
:clrscr()
:def(f1)
:rtn(" !")
:enddef(f1)
:def(f2)
:prints(arg())
:"World"->vv
:rtn(vv)
:enddef(f2)
:fc(f2,"Hello ")->str
:prints(str)
:prints(fc(f1,0))
:keywait()
:end
:EndPrgm
```

This program will print on the screen : Hello World !

How to make local variables (in this case, this is arg()[3]) : (the **group**() function is useful)

```
:aa()
:Prgm
:start
:def(prt)
:ll(arg(),0)->nb
:While ll(arg(),3)<nb
:wl(arg(),3,ll(arg(),3)+1)
:printf1("\nLocal var=%ld",ll(arg(),3))
:prints(ll(arg(),ll(arg(),3)))
:Endwhile
:free(arg())
:enddef(prt)
:fc(prt,group({" AZE "," RTY ",0}))
:prints(" Fini")
:keywait()
:end
:EndPrgm
```

This program will print "azerty" to the screen. In this example, the third argument of the array pointed by arg() is used as a local variable.

Fonction : ptr group
Numéro de la fonction : 0d
Syntaxe : **group**(*nilist*)
Description :
Cette fonction transforme une niliste en liste instanciée codée 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 peekl mais l'utilisation des [] est recommandé. Vous devez libérer l'espace mémoire en utilisant la fonction free avant l'arrêt du programme.
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**()).
Exemple :

```
:aa()
:Prgm
:start
:group({1," Hello ",code("c")})->temp
:clrscr()
:printf2("nb arg=%ld arg1=%ld",temp[0],temp[1])
:prints(temp[2])
:printc(temp[3])
:keywait()
```

:free(temp)
:end
:EndPrgm
The result of the program will be : nb arg=3 arg1=1 Hello c

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

3 Fonctions de déboguage - Debug

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 (a part peut être certaines fonctions de bouclages). Le nom de la fonction sera affiché sur l'écran et permettre ainsi peut être de comprendre l'erreur commise.
Set the debug mod. The execution of the program will be now in the step by step mod (whithout the while functions). The name of the function will be print to the screen.

Fonction : - debugoff
Numéro de la fonction : 0d241
Syntaxe : **debugoff**()
Description :
Désactive le mode de déboguage. L'exécution du programme n'est plus en mode pas à pas.
Retourne en mode d'exécution normal.
Set off the debug mode. Return in normal execution mod (go with **debugon**).

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

4 Fonctions de mémoire - Memory

Fonction : handle toos
Numéro de la fonction : 0d172
Syntaxe : **toos**(*entity,str_var*)
Description :
This function allow you to create a tios variable which will be affected to the entity value. If *entity* is a number so *str_var* will be a tios EXPR type, if the entity is a nilist or a pointer to a nilist or a pointer to a string (so use the #) so it will be a TIOS list type. You can also use functions for entity.
If the function return 0, there is an error. The file str_var has not been created (not enough memory?).
Exemple :
:init()
:start
:toos(123+0, « a »)
:end
Now, if you enter in the home screen :
a
The result will be 123.
The variable a has been created and affected to 123.

:init()
:start
:{1,"two",3}->var
:"Hello"->str
:toos({1,"two",3},"b")
:toos(#var,"a")
:toos(#str,"c")
:end

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"

Fonction : long seql
Numéro de la fonction : 0d
Syntaxe : **seql**(*var,start,end,step,var_dest,instruction*)
Description :
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*). You will have to free the allocated memory block before exiting the program (**free**()).
Example :
```
:init()
:start
:seql(va,1,4,1,list,va)
:0->vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
:end
```

Fonction : long seqw
Numéro de la fonction : 0d
Syntaxe : **seqw**(*var,start,end,step,var_dest,instruction*)
Description :
Allocate a block of memory of (end-start+1)*4 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*). You will have to free the allocated memory block before exiting the program (**free**()).
Example :
```
:init()
:start
:seqw(va,1,4,1,list,va)
:0->vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
:end
```

Fonction : long seqb
Numéro de la fonction : 0d
Syntaxe : **seqb**(*var,start,end,step,var_dest,instruction*)
Description :
Allocate a block of memory of (end-start+1)*4 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*). You will have to free the allocated memory block before exiting the program (**free**()).
Example :
```
:init()
:start
:seqb(va,1,4,1,list,va)
:0->vb
:while vb<4
:printld(list[vb])
:isz(vb)
:EndWhile
:free(list)
:keywait()
:end
```

Fonction :
Numéro de la fonction : 0d
Syntaxe :

Description :

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.
Si vous souhaitez utiliser les [] pour lire ou écrire dans cet espace mémoire (une fois que la valeur de retour de cette fonction est été affectée à une variable), il faut savoir que l'espace sera considéré comme étant une liste instanciée d'élément codés sur 4 bytes (octets).

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.
If you want to read or write in this block memory, you can use the [] notation. If you use [] notation (associated with a variable name of course), you must know that the memory block is equivalent to an instanciated list of 4 bytes width elements.

Fonction : - free
Numéro de la fonction : 0d252
Syntaxe : **free**(*ptr*)
Description :
Libère l'espace mémoire pointé par *ptr* et alloué avec **malloc**() ou **seqb()**, **seqw()**, **seql()**, **group()**.
Free the memory space allocated with **malloc** or **seqb(), seqw(), seql(), group()** functions ; and pointed by *ptr*.

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**(ptr_dest, ptr_src, len)
Description :
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()
:start
: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)
:end
```

Fonction : ptr memmove
Numéro de la fonction : 0d
Syntaxe : **memmove**(ptr_dest, ptr_src, len)
Description :
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*.

5 Fonctions d'arithmétique - arithmétics

Fonction : long isz
Numéro de la fonction : 0d7
Syntaxe : **isz**(*var*)
Description :
Incrémente la variable NewProg *var*.
Au final, on a var=var+1

Increment the NewProg variable var.
Finally, var=var+1.

Fonction : long dsz
Numéro de la fonction : 0d8
Syntaxe : **dsz**(*var*)
Description :
Décrémente la variable NewProg *var*.
Au final, on a var=var-1

Decrement *var*.

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

6 Fonctions clavier -Keyboard

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. La fonction ne fonctionne pas (calculatrice bloquée) lorsque les fonctions int1off() a été exécutée sans préalablement (sauf si int1on() a été après coup).

Waiting for a key to be pressed. Return the number of the key pressed. Similar to getkey() code in Tibasic function. Do not use when int1off() has been executed previously (so call int1on() to use keywait) because you can crash your calculator.

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

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 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

Row	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:

Column

Row	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	-

Exemple :

```
:init()
:start
1->xx
:while x<10000:isz(xx)
if keytest(1,3) then                    //if you want to test the left key only (on ti89), write if keytest(1,0b10) then
:prints(« Key pressed »)
:endif
:endwhile
:prints(« Finish »)
:keywait()
:end
```

Test if the keys Left and Up have been pressed (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 (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 : value gkey
Numéro de la fonction : 0d182
Syntaxe : **gkey**()
Description :
Similaire à la fonction getkey du Tibasic. INT1 ne doit pas avoir été désactivé (avec **int1off**()) pour utiliser cette fonction.

Is the same as the Tibasic getkey function. The INT1 must not have been desactivated (with **int1off**()) if you want to use this function.

Fonction : value keydelay
Numéro de la fonction : 0d183
Syntaxe : **keydekay**(*delay*)
Description :
Sets the initial autorepeat key delay

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 :
Numéro de la fonction : 0d
Syntaxe :
Description :

7 fonctions de sauts conditionnelles – conditionals jumps

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 tios function. The condition is true if !=0.

Fonction : all_types ifs
Numéro de la fonction : 0d23
Syntaxe : **ifs**(*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.* Ifs 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...*
Ifs 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).
Exemple :

```
:init()
:start
:prints(ifs(1,{"True1","True2"),{"False1"}))
:keywait()
:end
```

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

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.

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

Fonction : - repeat
Numéro de la fonction : 0d160
Syntaxe : **repeat**(*n,instruction*)
Description :
Répète *n* fois l'instruction *instruction*. Cette fonction à l'avantage d'être très rapide.

Repeat n times the instruction *instruction.* This function has the advantage to be fast.

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.
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 loop.
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 derniere valeur retournée.

Execute *num* instructions *instruction1,instruction2…* by considering of only one instruction. Be carreful, *num* must be equal to the sum of the instructions !
Return the last value returned.
Example :

```
:init()
:start
:0->va
:clrscr()
:map(var,1,10,1,multi(2,isz(va),printld(va)))
:keywait()
:end
```

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 : boolean up down left right second shift diamond alpha
Numéro de la fonction : 0d
Syntaxe : up() down() left() right() second() shift() diamond() alpha()
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 n'arrête pas l'exécution du programme (de même que la fonction **keytest**()) 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 tios

Fonction : - endh – exech - freeh
Numéro de la fonction : 0d11 0d12 0d10
Syntaxe : start :...:**endh**(*newprog_var*):*Tios commands* :start :...:**exech**(*newprog_var*):...:**freeh**(*newprog_var)*:...:end
Description :

When the **endh** 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*. We can now execute this instructions in the next NewProg section with the **exech** function.

Lorsque la fonction **endh** est exécutée, *Tios commands* n'est alors pas exécutée ! Un handle sera créé référant a *Tios_commands*. Cet handle est placé dans *newprog_var*. On peut ensuite exécutée ces instructions dans les sections NewProg suivantes à l'aide de la fonction **exech**. Comme un espace mémoire est créé (pointé par *newprog_var*), il faut le libérer avant la fin du programme à l'aide de la fonction **freeh**. Cette stratégie d'exécution n'a que pour seule avantage de lancer plus rapidement l'exécution de *Tios_commands*.

Fonction : long-ptr os
Numéro de la fonction : 0d171
Syntaxe : **os**(*string*)
Description :
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.
Retourne la valeur après exécution de *string*. Ne renvoie pour l'instant q'un entier ou un pointeur sur chaine de caractère. Attention, les donneés sont temporaires et doivent être sauvegardées pour ne pas les perdre. Cette fonction est lente.
Exemple :
:os(« 3»)
Return 3
Retourne 3
:os(« x+3+3 »)
Return the value of the tios variable x + 6.
Retourne la valeur de la variable tios x + 6.

Fonction : 0 execbas
Numéro de la fonction : 0d243
Syntaxe : **execbas**(*string*)
Description :
Execute the string *string* as a tios command. Return 0. Similar to the **os**() function but with no return.
Exécute *string* comme une commande tios. Similaire a **os**() mais ne renvoie pas le résultat de l'exécution.

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

9 Fonctions d'affichage de texte –Text printing

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")
:end
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()
:end
```

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 ce limite 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

Exemples :

```
:init()
:start
:clrscr()
:printf1(“One hundred :%ld”,100)
:printf2(“%s %ld”,”Two hundreds”,200)
:keywait()
:end
```

You will see print on the screen : One hundred :100Two hundreds 200

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 : - 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()
:end
```

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

10 Fonctions d'interruptions - Interrupts

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 : inter
Numéro de la fonction : 0d
Syntaxe : **inter**(*interupt_ID,tick,{instruction1,instruction2,...}*)
Description :

Allow the interupt *interupt_ID* to execute the instruction(s) in the {instruction 1, instruction 2,,,,} 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...).
To disable 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 disable all interrupts :
Inter(0,X,X);

Example :

```
:init()
:start
:0->va
:inter(1,20,{prints(“ Inside “),isz(va)})
:while va<5
:Endwhile
:inter(1,0,0)
:prints( « Finish ! »)
:keywait()
:end
```

This program will print to the screen :
Inside Inside Inside Inside Inside

Fonction : interoff
Numéro de la fonction : 0d
Syntaxe : **interoff**()
Description :
Disable the interupts from the auto int1 to the auto int5 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 usefull when you want have a maximum of speed . You can increase by 230% the speed of the execution of your program if you are using regulary the keyboard. However, if you want to use keys for a game in particular, you will have to use **keytest**() (or up(), down() etc...), because others keyboard functions will now not be sure. By launching this function, you will not be able to use functions like **keywait**() because they need auto int1 to be able.

Make caution : The grayscale printing is not possible when this function has been launched. You will have to use the functions **int1off**() and **int1on**() indeed of this function to abord this problem.

Fonction : interon
Numéro de la fonction : 0d
Syntaxe : **interon**()
Description :
Enable the interupts from the auto int 1 to the auto int 5. You will have to use this function after have using **interoff**().

Fonction : int1off
Numéro de la fonction : 0d
Syntaxe : **int1off**()
Description :
This function redirect the auto int 1 to nothing. The purpose of this handler is to redirect an interrupt vector to "nothing", in cases when disabling interrupts is not possible. For example, you can not disable auto-int 1 in grayscale programs (if you use **grayon**()), because grayscale support is based on it. It is also not possible when you have define your own interupts with **inter**() and that you want use them. Grayscale support installs its own auto-int 1 handler, which executes the previously installed handler at the end. Suppose that you don't want it to call the default auto-int 1 handler, which trashes the status line by displaying keyboard status indicators.

Fonction : int1on
Numéro de la fonction : 0d
Syntaxe : **int1on**()
Description :
Enable the interupts from the auto int1. You have to use this function if you already have launch the **int1off**() function.

Fonction : int5off
Numéro de la fonction : 0d
Syntaxe : **int5off**()
Description :
Disable the default interupt procedure of the tios. The auto int 5 will not be disable, but only the default procedure of the tios. This function can be usefull if you want to have a maximum of speed with keeping the interupts that you will have before (with **inter**()) .
Be carefull, launching this function disable the decrement of the counter of the auto int 5 (make by the default procedure of the tios), so functions **freet**(), **settimer**() and **timerexp**() will not work correctly.

Fonction : int5on
Numéro de la fonction : 0d
Syntaxe : **int5on**()
Description :
Enable the default interupt procedure of the tios of the auto int 5. This function will be use if you have disable it before with **int5off**().

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

Frees a notify (countdown) timer.

Freet() deactivates and frees the notify (countdown) timer *timer_no*. **Freet**() must be called before registering a timer using **settimer()** if the timer was already in use. Returns FALSE (=0) in a case of error, else returns TRUE (!=0).

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

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 programable rate generator), the current value of the timer is decremented by 1. You have to not have disable the auto int 5 with **interoff**() or with **int5off**() before. When the current value reaches zero (get 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. This happens if you give wrong parameters, or if the timer *timer_no* is already in use. So, you must first free the 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()
:start
:freet(2)
:settimer(2,100*20)
:end
```

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

Determines whether a notify (countdown) timer expired.

Timerexp() returns 1 if the notify (countdown) timer *timer_no* expired, else returns 0. See **settimer**() for more info. For example, a legal way to make a 5-second delay is:

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

Timerexp() also resets flag which tells that the timer was expired, so the calling this function again will return 0.

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

Determines a current value of a notify (countdown) timer.

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

Fonction : setrate
Numéro de la fonction : 0d
Syntaxe : **setrate**(*rate*)
Description : Not avalaible !!

Sets the speed at which the programmable rate generator is incremented.

The programmable rate generator's speed can be controlled by this function. Valid values for *rate* are as follows:

Value	Speed
0	OSC2/2^5 (highest rate)
1	OSC2/2^9 (default)
2	OSC2/2^12
3	OSC2/2^18 (lowest rate)

If you modify the rate generator's speed, it will modify the frequency of all auto int (auto int 1, auto int 5 in particular) so your interupts and timers will be launched with a different speed (**settimer**(), **timer**()). Faster if *rate*=0 for example, it's may be usefull for making a chronometer for example (with a better precision).
Note: Before exiting your program, it is good practice to restore the programmable rate generator to its previous value so that the AMS does not misbehave.

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

Switches the calculator to "idle" state for a while.

While idle is running, the calculator rests. idle turns the calculator in "low power" state until the next interrupt occurs (then "low power" state will be disabled, and idle returns).

While calculator is in "idle" state, the power consumption decreases significantly. TIOS very often calls idle, whenever it is in a kind of "idle loop". So it is very useful to be used in programs which waits in a loop for something (waiting for specific keypress, timer expiring, etc.). Many programs should use idle to save the batteries (editors, reflexive games, explorers, debuggers etc.). Thanks to Julien Muchembled for this info.

Note: Idle interferes with grayscale graphics, so the use of idle while grayscale mode is active is not recommended.

11 Fonctions de manipulation des chaînes de caractères – strings manipulations

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.

Fonction : ptr chars
Numéro de la fonction : 0d27
Syntaxe : **chars**(*char_value*)
Description :
Renvoie un pointeur sur une chaîne de caractères ayant comme seul composante un caractère correspondant à *char_value*. Il faut libérer l'espace mémoire après utilisation.

Return a pointeur on a string containing a chars corresponding to the ASCII code *char_value*. You must free the ptr after have use dit.

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

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

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.

Note: This routine is declared as "short" although the ANSI standard proposes "long". This is important, because TIOS the strncmp routine puts garbage in the higher half of the d0 register.

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

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 :

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*. The new line is replaced by a null character ('\0') in *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* (ANSI proposes returning of 0 (NULL) in a case of error, but this never occurs on the TI). For editing, the backspace key is supported

Fonction : value strlen
Numéro de la fonction : 0d
Syntaxe : **strlen**(*string*)
Description :
It calculates the length of *string*. Returns the number of characters in *string*, not counting the terminating null character.

Fonction : ptr newstr
Numéro de la fonction : 0d
Syntaxe : **newstr**(*str,var_dest*)
Description :
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 pointor of the new memory block and also return it. You must delete this block before exiting the program (using **free**()).
Example :

```
:init()
:start
:clrscr()
:"Hello"->str
:newstr(str,va)
:prints(va)
:keywait()
:free(va)
:end
```

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

12 Fonctions graphiques – Graphics fonctions

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*).

Display a line on the screen between the point (*xa,ya*) et the point (*xb,yb*).

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 : long gmode
Numéro de la fonction : 0d42
Syntaxe : **gmode**(*gmode*)
Description :
Sélectionne le mode d'affichage de toutes les fonctions graphiques. Les 3 premiers modes sont à utilisés pour toutes les fonctions. Les autres sont seulement pour les fonctions lignes (dline, dmline). Retourne l'ancienne valeur du mode graphique.
Select with this function the graphics mode for all graphics functions of newprog. The free first modes are support with all Newprog graphics functions. The others are for **dline**() and **dmline**(). Return le old value of the graphic mode.

0	Draw as normal
1	Draw as inverse
2	Draw as using XORing with the destination
3	Draw a double thick line
4	Draw the line using a vertical shading pattern
5	Draw the line using a horizontal shading pattern
6	Draw the line using a negative slope diagonal shading pattern
7	Draw the line using a positive slope diagonal shading pattern

Les valeurs de gmode sont différentes pour les fonctions sprt8,16,32.
The value of gmode must be different for the function sprt8,13,32.

Fonction : sprt8
Numéro de la fonction : 0d43
Syntaxe : **sprt8**(*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.

Display the 8-bits wide sprite *sprite* at the position (*x,y*) et with a height of *h* rows.

XOR mode : **gmode**(0) | OR mode : **gmode**(1) | AND mode : **gmode**(2)
Example : Similar to sprt16 example

Fonction : sprt16
Numéro de la fonction : 0d44
Syntaxe : **sprt16**(*x,y,h,sprite*)
Description :
Affiche le sprite (16 bits horizontales) *sprite* avec le coin supérieur au point (*x,y*) et une hauteur de *h* lignes.

See sprite8. This is quite similar but for a 16 bits wide sprite.

XOR mode : **gmode**(0) | OR mode : **gmode**(1) | AND mode : **gmode**(2)

Example : Draw a sprite

```
init()
start
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()
end
```

Fonction : sprt32
Numéro de la fonction : 0d53
Syntaxe : **sprt32**(*x,y,h,sprite*)
Description :
Affiche le sprite (32 bits horizontales) *sprite* avec le coin supérieur au point (*x,y*) et une hauteur de *h* lignes.

See sprite16. This is quite similar but for a 32 bits wide sprite.

XOR mode : **gmode**(0) | OR mode : **gmode**(1) | AND mode : **gmode**(2)

Example : Similar to sprt16 example

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.
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.
This function return the old value of graphics memory position.

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 function **gmode**().

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.

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.
Clear the screen without reseting the text cursor position. Faster for graphics purposes than **clrscr**(). Can be used to make grayscale.

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

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

Fonction : long grayon
Numéro de la fonction : 0d70
Syntaxe : **grayon**()
Description :
Active les niveaux de gris (4 niveaux).

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!

Fonction : long grayoff
Numéro de la fonction : 0d71
Syntaxe : **grayoff**()
Description :
Désactive les niveaux de gris.

Deactivates grayscale mode.

This function deactivates grayscale mode. If grayscale mode is not activated, this function does nothing.

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'.

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

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

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 :
DrawStr draws a string *str* at a specific (*x*, *y*) location. This does not interfer with the text cursor (with **prints** and other functions).
Cette fonction n'affecte pas le curseur de position du texte (fonctions **prints**...).

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

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 : - 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.

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()
start
clrscr()
gsprt(8,3,"bitmap",pp)
sprt8(50,50,3,pp)
free(pp)
keywait()
end
```

Fonction : - scrrect
Numéro de la fonction : 0d
Syntaxe : **scrrect**(*xa,ya,xb,yb*)
Description :
As default, the tios doesn't allow you to draw on the status line for example. It's due to an internal variable of the tios that is set as default to (0,0,94,159) so you can't have access for drawing to the horizontals lines 95 to 99 (I'm not sure of this value). By changing this internal variable, it is possible to force some TIOS commands which normally can not access to the status line area to get the access to this

"forbidden" zone, or to force some commands to use only smaller part of the screen. Use this possibility with great care, and only if you know exactly what you are doing!
Using this function affect the internat variable to the rectangle (xa,ya) ; (xb,yb). For drawing on all the screen, use **scrrect**(0,0,99,159) (for the ti89 obviously).

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.

Make a copy of the at the time video memory to a specified block of memory pointed by the index *index*. If there was a previous saved screen at the specified index *index*, newprog will overwrite and then made the copy. Newprog erase automatically the video memory block at the end of the execution of the program (**free**() using is not necessary). *Index* must be in [0,3] range.
To be used with the **loadscr**() function.

Fonction : ptr loadscr
Numéro de la fonction : 0d163
Syntaxe : loadscr(*index*)
Description :
Affiche à l'écran l'image préenregistrée référencée par l'index *index* (utilisation avec **savescr**()). Plus exactement, cela copie l'image dans la mémoire vidéo actuelle (**setlcd**(), **getlcd**()...). Elle peut être utilisée pour faire des niveaux de gris.

Copy the to the at the time video memory the previously stored video with the index *index* (to use with **savescr**() function). This function is available for the grayscale functions.

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

13 Fonctions de manipulation de bits, de nombres, de listes-bytes, lists

Fonction : long lrol
Numéro de la fonction : 0d94
Syntaxe : **lrol**(*expr1*,*expr2*)
Description :
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 :

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 : nott, neg, xorr, orr, and, inf, infeq, eq, supeq, sup, noteq, add, sub, mul, div
Numéro de la fonction : 0d120 0d122 0d130 0d131 0d132 0d133 0d134 0d135 0d136 0d137 0d138 0d139 0d143 0d145
Syntaxe : **nott, neg, xorr, orr, and, 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.

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

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.

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.

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*.

Return the value of the n-*elem* element of the 16 bits array (pointed by *byte_list_address*).

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*.

Return the value of the n-*elem* element of the 32 bits array (pointed by *byte_list_address*).
Example:
```
init()
start
seql(vv,1,4,1,dd,vv)
printld(ll(dd,3))
keywait()
end
```

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**(*adress*)
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*.

Fonction : - #
Numéro de la fonction : 0d14
Syntaxe : #*nilist_ptr*
Description :
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.
Exemple :

```
:init()
:start
:{keywait()}->var
:printf1("%ld",#var)
:keywait()
:end
```

This will execute the keywait() instruction (waiting for a key) and print the key number on the screen.

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

14 Fonctions de manipulation de fichiers-Files manipulations

Fonction : ptrfile fopen
Numéro de la fonction : 0d110
Syntaxe : **fopen**(*filename,mode*)
Description :

Opens a stream.

fopen opens the file named by *filename* (this is a normal C string, which should be in lowercase) and associates a stream with it. fopen returns a pointer to be used to identify the stream in subsequent operations. The *mode* string used in calls to fopen is one of the following values:

Mode	Description
r	Open for reading only.
w	Create for writing. If a file by that name already exists, it will be overwritten.
a	Append; open for writing at end of file, or create for writing if the file does not exist.
r+	Open an existing file for update (reading and writing).
w+	Create a new file for update (reading and writing). If a file by that name already exists, it will be overwritten.
a+	Open for append; open for update at the end of the file, or create if the file does not exist.

On successful completion, fopen returns a pointer to the newly opened stream. In the event of error, it returns NULL. Note that files on TI are in fact TIOS variables, so their maximal size is limited (as the size of the variable is limited). Maybe in the future I will try to implement files which are limited only by the amount of the free memory.
To specify that a given file is being opened or created in text mode, append a 't' to the mode string ("rt", "w+t", and so on). Similarly, to specify binary mode, append a 'b' to the mode string ("wb", "a+b", and so on). What "text" or "binary" exactly means will be explained a bit later. fopen also allows the t or b to be inserted between the letter and the '+' character in the mode string. For example, "rt+" is equivalent to "r+t". If a 't' or 'b' is not given in the mode string, 't' is assumed (this slightly differs from the ANSI convention: in ANSI C the mode in this case is governed by the global variable **_fmode**, which is not implemented here).

When a file is opened in "text" mode, it is assumed to be a TEXT variable. On creating, all necessary headers and tags for the TEXT variable will be created. On opening for reading, the file pointer will be set to the first character in the first text line (asuming that it IS a text variable). So, files created in "text" mode can be readed in the TI text editor, and you can read variables created in the text editor. All '\n' characters will be translated to '\r' 0x20 sequence during writting (to satisfy the format of the text in TEXT variables), and a character after '\r' will be swallowed during reading (to skip over the "command byte" at the begining of the each line). Here is an example:

```
f = fopen ("example", "w");
fputs ("First line\n", f);
fputs ("Second line\n", f);
fputs ("Third line\n", f);
fclose (f);
```

After this, you will have a TEXT variable called "example" which can be opened in text editor. You can read the content of a TEXT variable similarly.

When a file is opened in "binary" mode, nothing is assumed about the structure of the file. It can be a variable of any type. The user is responsible to create appropriate variable structure. There will no be any translation of characters, and after opening the file pointer will point to the first byte of the variable (after two "length" bytes), regardless of what the variable is supposed to be. For example, the string variable has the following structure: one zero byte, the content of the string, another zero byte, and finally, the string tag (STR_TAG or 0x2D byte). Here is an example of creating a file which represents a string variable:

```
f = fopen ("example", "wb");
fputc (0, f);
fputs ("This is a string", f);
fputc (0, f);
fputc (STR_TAG, f);
fclose (f);
```

When a file is opened for update (in both text or binary mode), both input and output can be done on the resulting stream. However, ANSI proposes that output cannot be followed directly by input without an intervening fseek or rewind, and that input cannot be directly followed by output without an intervening fseek, rewind, or an input that encounters end-of-file. I don't see any reason to implement such limitation here.

The filename pointed to by *filename* may also contain a path (i.e. a folder name may be given in front of the file name). If name of a folder which does not exist is given, and if fopen needs to create a new file, a dialog will appear which asks the user whether a new folder will be

created. If the answer is "NO", fopen fails, returning NULL. If no folder name is given, the current folder is assumed.

Note: All functions which accept a parameter which is a pointer to a FILE structure assumes that the pointer is valid, i.e. created using fopen command. As I have no any efficient method to check whether the pointer is valid or not, no checking is implemented. So, if you pass an invalid pointer to any file handling function, the results are unpredictable.

Fonction : long fclose
Numéro de la fonction : 0d111
Syntaxe : **fclose**(*ptrfile*)
Description :

Closes a stream.

fclose closes the stream associated to the structure pointed to by *ptrfile*. In this implementation, fclose unlocks the file (which is locked during it is opened), and frees the file descriptor structure pointed to by *stream*. fclose returns 0 on success. It returns -1 (EOF) if any errors were detected.

Fonction : long fputc
Numéro de la fonction : 0d112
Syntaxe : **fputc**(*c,ptrfile*)
Description :

Writes a character to a stream.

putc writes the character *c* to the stream given by ptrfile. It will update the stream's file pointer, and expands the size of associated variable if necessary. If the file is opened in "text" mode (see **fopen**()), all '\n' characters will be translated to '\r' 0x20 sequence during writting (to satisfy the format of the text in TEXT variables). On success, putc returns the character *c*. On error, it returns EOF (-1).

Fonction : ptr fgets
Numéro de la fonction : 0d113
Syntaxe : **fgets**(*s, n,ptrfile*)
Description :

Gets a string from a stream.

fgets reads characters from stream associated to the structure pointed to by *ptrfile* into the string *s*. It does this by calling **fgetc**() repeatedly. The function stops reading when it reads either *n* - 1 characters or a '\r' (0x0D) character, whichever comes first. **fgets** retains the newline character at the end of *s*, eventually translated to '\n' character if the stream is opened in "text" mode (see **fopen**()). A null byte is appended to *s* to mark the end of the string. On success, **fgets** returns the string pointed to by *s*. It returns 0 in a case of error.

Note: fgets is used mainly with files opened in "text" mode. As an example, this command may be useful for reading a text line from a TEXT variable.

Fonction : long fgetc
Numéro de la fonction : 0d114
Syntaxe : **fgetc**(*ptrfile*)
Description :

Gets a character from a stream.

fgetc gets the next character on the given input stream (associated with the structure pointed to by *ptrfile*), and increments the stream's file pointer to point to the next character. If the file is opened in "text" mode (see **fopen**()), a character after '\r' will be swallowed during reading (to skip over the "command byte" at the begining of the each line in a TEXT variable).

On success, **fgetc** returns the character read, after converting it to an integer without sign extension. On error (usually end-of-file), it returns –1 (EOF).

Fonction : long fputs
Numéro de la fonction : 0d115

Syntaxe : **fputs**(*s,ptrfile*)
Description :

Outputs a string to a stream.

fputs copies the null-terminated string *s* to the output stream associated to the structure pointed to by *ptrfile*. It does not append a newline character, and the terminating null character is not copied. On successful completion, **fputs** returns the last character written. Otherwise, it returns a value of –1 (EOF).

Fonction : value fseek
Numéro de la fonction : 0d
Syntaxe : fseek(*stream,offset,whence*)
Description :

Repositions the file pointer of a stream.

fseek sets the file pointer associated with *stream* (file pointer) to a new position that is *offset* bytes from the file location given by *whence*. For text mode streams (see fopen), *offset* should be 0 or a value returned by ftell. *whence* must be one of the following values (defined in enum SeekModes):

whence	File location
SEEK_SET = 0	File beginning
SEEK_CUR = 1	Current file pointer position
SEEK_END = 2	End-of-file

fseek discards any character pushed back using ungetc. fseek returns 0 if the pointer is successfully moved. It returns a nonzero value on failure.

Fonction : value ftell
Numéro de la fonction : 0d
Syntaxe : ftell(*stream*)
Description :

Returns the current file pointer.

ftell returns the current file pointer for the stream associated with the structure pointed to by *stream*. The offset is measured in bytes from the beginning of the file. The value returned by **ftell** can be used in a subsequent call to fseek. ftell returns the current file pointer position on success. It returns -1 on error.

Fonction : value feof
Numéro de la fonction : 0d
Syntaxe : feof(*stream*)
Description :

Tests a stream for an end-of-file indicator.

feof is a macro that tests the stream associated to the structure pointed to by *stream* for an end-of-file indicator. Once the indicator is set, read operations on the file return the indicator until fseek is called, or until the stream is closed. feof returns nonzero if an end-of-file indicator was detected on the last (usually input) operation on the given stream. It returns 0 if end-of-file has not been reached.

Fonction :
Numéro de la fonction : 0d
Syntaxe :

Description :

15 Fonctions appels de programmes assembleurs-Assembly programm call functions

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()
:start
:loadasm("asm")->asm
:repeat(100,execasm(asm))
:closeasm(asm)
:end
: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.

Fonction :
Numéro de la fonction : 0d
Syntaxe :
Description :

16 Autres fonctions utiles – Other usefull functions

Fonction : value rand
Numéro de la fonction : 0d101
Syntaxe : **rand**(*num*)
Description :
Retourne un nombre aléatoire compris entre 0 et num-1,

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

17 Instructions interdites, restrictions – Forbiden instructions, restrictions

Ils est interdits d'utiliser les instructions suivantes :
les variables de a à z, ainsi que les lettres grecques.
D'utiliser les boucles for:endfor, loop:endloop.

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

18 Rapport d'erreur oncalc – Oncalc error report

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

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

19 Divers – Miscalenous

Le compilateur ne vérifie pas (pour le moment) si le nombre d'argument passer à la plupart des fonctions est correct. A l'exécution du programme, si vous avez omis un argument à une fonction, la prochaine instruction sera prise en argument ! Alors penser à vérifier que vous n'avez pas omis d'argument car si vous utiliser des fonctions mémoires ou autre, cela peut faire planter la calculatrice. Cet inconvénient est parfois un avantage car il permet d'améliorer la visibilité du programme écrit dans l'éditeur de programme.
Exemple :
La fonction dline utilise 4 argument (dline(xa,ya,xb,yb))
Exécuter dline():0:0:50:100 est équivalent à exécuter dline(0,0,50,100)

The compilator do not check (for the moment), for almost of the functions, if the number of arguments passed to the function is correct. If you execute an instruction with a missing argument, the next instruction will be taken instead ! So be carreful, it can crash the calculator if you use a memory or system instruction.This problem is sometimes an advantage because it can help to enhance the lisibility of your program.
Example :
The function dline uses 4 args (dline(xa,ya,xb,yb))
Executing dline():0:0:50:100 is similar as execute dline(0,0,50,100)

20 Installation – Install

Send to your calculator the qbasic , qbasicc and npcc files. If you don't have an os installed, I recommand to use preos.

21 Remerciements - Thanks

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