NEWPROGCX DOC v1.01

Introduction :

Newprogcx is a programming langage close to C and basic for Ti nspire CX * and CXII*. It allows high execution speed.
Your 'Newprogcx source code' can be compiled on PC or oncalc and then executed on an oncalc Virtual Machine.
Newprogcx capabilities :

- Allocate and manipulate memory like in C (with pointers : char, short and int). Files manipulations. Low levels manipulations. Close to C.
- Write your own NewprogCX functions (recursivity possibilities)
- Use advanced graphicals instructions (sprites, doublebuffering)
- Use keyboard inputs functions, strings or numbers request windows.
- Manipulate time

Files summary :

'newprogcx-compilator-PC.exe' : windows side compilator
'newprogcx.prg.tns' : oncalc side compilator
'npvmcx.prg.tns' : oncalc virtual machine
'npcxfiles.tns' : Files selection (open it with 'notepad' (windows) or 'nTxt master' (oncalc))

Newprog cx examples programs files :

- **'falldownsrc.tns' : source code of falldown game**
- **'falldown.tns' : Program bytes code of the falldown game**

- **minesweepersrc.tns : source code of minesweeper game**
- **minesweeper.tns : Program bytes code of the minesweeper game**
- **minesweeperpic.bmp.tns : Sprites bundle of the game**

Newprog cx compilator and virtual machine files :

- **tokens68k.c : C source code of the compilator (oncalc and windows or cygwin)**
- **npvmcx.c : C source code of the virtual machine (oncalc only)**

Other files :

- **codebbmp.tns : sprite for examples of the documentation (this file)**

Newprogcx editor, compilation and execution :

The source code can be edited in the 'notepad' application (windows) or directly on calc with the 'nTxt' program. See the matching instructions for specials characters.

The source code length must be under 30kbytes.

To set the source code file and the compilation byte code output file, you have to open (in nTxt or notepad) the file named 'npcxfiles.tns'. The first line must start by 'INPUTSRC=' and the second line by 'EXEC='.

Entries example : If by instance your source code is contained in 'falldownsrc.tns' and the desired bytecode output is 'falldown.tns', so enter :
INPUTSRC=falldownsrc
EXEC=falldown

Newprogcx Coding advice :

For readability purpose, it is advised to write your newprogcx code like this :

```
//put your lists
//or matrix definitions at beginning

//yours functions definitions with 'func'...'endfunc'

//all others code (Main code)
```

Example :

```
//At first, all lists or matrix defintions...
datauc str[20]= "Hello World"   //only one line defintion in this case

//All functions definitions...Must be writen before the launching instruction of the functions
func dispstr(arg1)
printf("\n%s",arg1)
endfunc

func waitakey()
keywait()
endfunc

//The firsts globals instructions...

fillscreen(0)   //fill screen in black color
dispstr(str)    //launching instruction of the function 'dispstr()'
waitakey()      //launching instruction of the function 'waitakey()'
//Program exit
```

Newprogcx code comments

1- First way. You can use the '//' notation to significate than since the '//' up to the end of the current line is a comment.
Example : printf("\nHello World") *//print 'Hello World' to the screen*
2- Second way. You can use the '/* */' notation. Between the '/*' and the next '*/' will be comment. Note :By this way, you can comment on multiple lines.
Example 1 :

```
fillscreen(0)              //fill screen in black
var=1+2/*comment*/+3       //comment between '2' and '+'.Will be understand as 'var=1+2+3'
printf("\nvar=%d",var)     //will print 'var=6'
keywait()                  //wait for a key press
/*                                          //comment beginning
printf("\nThis text will not be printed")       //not executed
```

```
keywait()                                   //not executed
*/                                          //end of comment
```

Newprogcx variables, lists and matrix :

In Newprogcx, you can use simple variables, lists variables or matrix variables. Like in C, lists and matrix variables pointers are stored in their names (see the section about lists and matrix defintions).
Simple variables are coded on 4 bytes length only. They are signed. They can have a value from -2147483648 to 2147483647. The C equivalence is : signed int 'varname'. The Newprogcx simple variables are automaticaly defined at the first apparition in the program. Their initial value is 0. The compilator doesn't takes care of upper case for functions names and vars names.

Example of code using simple var :

```
//Below : disp 0. The var 'value' is automaticaly defined (and set to 0)
printf("\nInitial value=%d",value)

value=10                                //affect 'value' to 10
printf("\nValue=%d",value)      //disp : Value=10
keywait()                               //wait a key pressed before exit
```

//Statics Lists and matrix creation and content

Be careful : The statics Lists and matrix creation can be compiled only on PC. To avoid this problem, use the 'newdata()' function instead.

Not empty lists or matrix definition :

Affectation with '{}' notation :

```
data** list_name[nbelem]={v0,v1...}        //first elem=v0, etc...
data** matrix_name[nbrows][nbcols]={v0,v1...}
```

You can have 6 types of data for lists and matrix definition. Replace the '**' by :
**=uc = unsigned char elements (1 byte length). From 0 to 255.
**=sc = signed char elements (1 byte length). From -128 to 127.
**=us = unsigned short elements (2 bytes length). From 0 to 65535.
**=ss = signed short elements (2 bytes length). From -32768 to 32767.
**=ui = unsigned int elements (4 bytes length). From 0 to 4294967295.
**=si = signed int elements (4 bytes length). From -2147483648 to 2147483647.

'nbelem', 'nbrows', 'nbcols' and each elements 'v0,v1...' must be direct values (or predifined constants). The '{,}' must be on the same line.

Particular case of matrix definition :'{}' imbrication forbiden.
Arrray Elements ordering:
For a matrix definition example: data** m[4][2]={1,2,3,4,5,6,7,8}
so m[0][0] = 1, m[0][1] = 2, m[1][0] = 3, m[1][1] = 4,...,m[3][1] = 8

Particular case of 'datauc' and 'datasc' :

With these types, if you want to create a list or matrix array without defining each elements values, you should write a '{}'. Example :
data** list[20]={}   //create an array (list) without defining each elements
data** matrix[4][4]={}   //the same but for a matrix
Other than that, you can define only the first elements (lists and matrix). Example :
 data** list[20]={1,2}   //Only the first and second elements are defined. The others elements values are undeterminated.

Affectation with '" "' notation (only for 'datauc' and 'datasc') :

The initialisation of the pointers are made at the program start, only one time.
An other way to defined the elements of a list or a matrix is to use the "" notation.
datauc list[nbrows]="abcdefg..." //list have nbrows elements (maxi).
list[0]==charord("a"), list[1]==charord("b"), etc.. ('a', 'b'... in C) (This is not the hexadecimal numbers). The string input are only available with 'datauc' and 'datasc'.

Note : you can affect the pointer of a string directly to a variable with the example code :
str="Hello"   //str is a pointer to the array. (The null end char is include)
This is a faster way to define an array, but with this notation, your are not allowed to use the '[]' notation.

Once a list or a matrix is defined, they cannot be redefined later (on other(s) line(s)). They are setup at compilation and at initialisation of the program (just before the execution of the program).

Example :
Correct : But this is still 'bad code'
line a+0 :for(i=0,i<10,i++)
line a+1 :datauc list[5]={1,2,3,4,5} //'list' is defined just on one line. Will be skiped at execution.
Line a+2 :endfor

Not correct : Two times defined (on differents lines)
line1 :datauc list[5]={1,2,3,4,5} //'list' is defined at first time
line2 :datauc list[5]={6,7,8,9,10} //This is not correct, because defined on an other line.

Empty statics lists or matrix definition. So just pointers :

With 'void' : Define just a pointer, no array created.

- void datauc list[nbrows]   //'list' is an empty list pointer (no elements). 'nbrows' must be written, even if has no use.

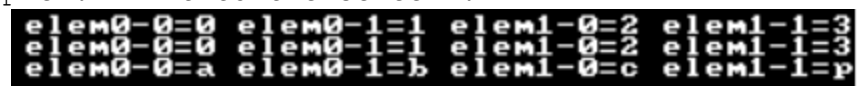
C equivalent :unsigned char *list;

- void datauc mat[nbrows][nbcols] //'mat' is an empty matrix pointer (no elements).

Example : fill the half left screen in white

```
void datass lcdmat[lcdheight][lcdwidth]    //void matrix pointer
lcdmat=getlcdad()               //affect lcdmat to screen adress
for(x=0,x<lcdwidth/2,x++)
for(y=0,y<lcdheight,y++)
lcdmat[y][x]=white                    //-1=white color
endfor
endfor
keywait()
```

Another example : Print to the screen :




```
fillscreen(black)     //clear screen
datauc matrix[2][2]={0,1,2,3}   //set a memory area filled with '0'...
at compilation
void datauc matrix2[2][2]  //just a pointer on a non existing unsigned
char matrix of size [2][2]
printf("\n elem0-0=%d elem0-1=%d elem1-0=%d elem1-1=%d",matrix[0]
[0],matrix[0][1],matrix[1][0],matrix[1][1])
keywait()
matrix2=matrix
printf("\n elem0-0=%d elem0-1=%d elem1-0=%d elem1-1=%d",matrix2[0]
[0],matrix2[0][1],matrix2[1][0],matrix2[1][1])
keywait()
matrix2="abcd"
matrix2[1][1]=charord("p")
printf("\n elem0-0=%c elem0-1=%c elem1-0=%c elem1-1=%c",matrix2[0]
[0],matrix2[0][1],matrix2[1][0],matrix2[1][1])
keywait()
```

//Dynamics Lists and matrix creation with the 'newdata()' function

Dynamics lists and matrix creation is made with the 'newdata()' function. This is the best way to create lists and matrix. The compilation can be done oncalc (instead of statics ones).
The 'newdata()' function allocate dynamicaly a memory area. So be sure to use the 'free()' function to free memory before the end of the execution of the program. If you ommit to free the memory block, the only way to retrieves this memory block is to reset your calculator.
The reading with '[]' or '[][]' is the same as the static ones (see above).

//newdata(direct_var,data_type,{nbrow[,nbcol]},{data1,...,datan})

'newdata()' is closed to static lists or matrix with the advantage to be compiled oncalc.
Allocates a memory block (on heap) pointed by 'direct_var' with type equals to 'data_type' :
'uctype' or '1' value : unsigned char (1 byte len)
'ustype' or '2' value : unsigned short   (2 bytes len)
'uitype' or '4' value : unsigned int (4 bytes len)
'sctype' or '11' value : signed char (1 byte len)
'sstype' or '22' value : signed short (2 bytes len)
'sitype' or '44' value : signed int   (4 bytes len)

This function allows to create lists or matrix dynamicaly. For create a list, you have to ommit the 'nbcol' parameter.
The dimension of the allocated memory block in bytes is equal to 'nbrow' for a list and 'nbrow' * 'nbcol' for a matrix, the all multiplicated by the length of the 'data_type' (1,2 or 4 len).
Each element of the list or matrix is set with 'data1', etc...
'data1'...can be the result of an operation (not directs values) instead of statics lists or matrix.
Imbricated '{}' are forbiden. Only in a raw allowed (like statics ones).

Be carreful, if 'n' (of 'data') is greater than 'nbrow'*'nbcol', the behaviour is undefined.

Example 1 : Will print : « 0-0=255 0-1=2 3-4=20 »

```
fillscreen(black)
newdata(m,1,{4,5},{-1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20})
printf("\n0-0=%d 0-1=%d 3-4=%d",m[0][0],m[0][1],m[3][4])
keywait()
if(m):free(m)
```

Example 2 : Will print « 0-0=-1 0-1=0 1-0=1 »

```
fillscreen(black)
newdata(m,11,{2,2},{-1,-1+1,1,2})
printf("\n0-0=%d 0-1=%d 1-0=%d",m[0][0],m[0][1],m[1][0])
keywait()
if(m):free(m)
```

Example 3 : prints « Hello World ! »

```
fillscreen(black)
newdata(m,4,{2},{"Hello","World !"})
printf("\n%s %s",m[0],m[1])
keywait()
if(m):free(m)
```

Example 4 : Storing example - Prints « 1=-666 3=2»

```
fillscreen(black)
newdata(m,22,{4},{-1,-1+1,1,2})
m[1]=-666
printf("\n 1=%d 3=%d",m[1],m[3])
keywait()
if(m):free(m)
```

//Classicals operators :

// =

Storing operator. Like in C.
Storing in a var : var_name = value_to_store
Storing in a list or matrix element :

```
     list_name[row] = value
     matrix_name[row][column] = value
```

// + - * /

Nothing to say.

// not value or !value

Logical not of 'value'

Note : 'if(not value)' is equivalent to 'if(value==0)'
Note 2 : A not null value is interpreted as true.

//Logical equal : v1==v2

//Logical not equal : v1!=v2
//Logical AND : 'v1 and v2' or 'v1 && v2'
//Logical OR : 'v1 or v2' or 'v1 || v2'
Logical AND and OR.

//Logical superior, equal or superior : v1>v2 , v1>=v2
//Logical inferior, equal or inferior: v1<v2 , v1<=v2

//Binary AND : v1 andb v2 or v1 & v2
//Binary OR : v1 orb v2 or v1 | v2
Binary AND and OR
Example : Operations examples


```
fillscreen(black)
v1=0b1001  //==9
v2=0b1000  //==8
printf("\nv1 andb v2=%d\nv1 orb v2=%d",v1 andb v2,v1 orb v2)
keywait()
```

//value<<nbshifts or value>>nbshifts
C equivalent. Left shift or right shift.

Example : Multiply by 4 and divide by 2 with 'shift_operators'
9 = 0b1001 - 36 = 0b100100 - 4 = 0b100


```
fillscreen(black)
printf("\nOriginal number=%d, left=%d right=
%d",0b1001,0b1001<<2,0b1001>>1)
keywait()
```

//v1 modulo v2

Produce the remainder of an integer division (v1 / v2).

C equivalent : v1%v2

Example :

```
fillscreen(black)
printf("\n 12 modulo 10=%d",12 modulo 10)
keywait()
```

//++ and -- instructions

Like in C. Post increments or decrements a variable.
It's works too with the *(**uc***)(ptr),...,*(si*)(ptr) and lists and matrix.

Example 1 : Prints « Hello World ! »

```
fillscreen(black)
str="Hello World !"
len=strlen(str)
for(i=0,i<len,i++)
printf("%c",*(uc*)(str+i))
endfor
keywait()
```

Example 2 : Print « var=65 »

```
fillscreen(black)
var=66
(*(ui*)((ad*)var))--
printf("var=%d",var)
keywait()
```

Example 3 : Prints m0=67 m1=66

```
newdata(m,1,{2},{66,67})
m[0]++
m[1]--
printf("m0=%d m1=%d",m[0],m[1])
keywait()
```

Newprogcx jumps

See the matching section for functions definition.

//lbl lbl_name

This function has the same role as the function of the tibasic. It allows you to place a label of the name by 'label_name'. It is to be used with the 'goto' function.

//goto lbl_name

This function has the same role as the function of the tibasic. It allows you to jump to the label with the same name as 'lbl_name'. To be used with the 'lbl' function.

//if(cond):instruc_if_true

If the condition 'cond' is true (ie!=0) then execute the instruction 'instruc_if_true'. If false, skip the instruction 'instruc_if_true'.

Example : Will print « Hello World »

```
fillscreen(black)                //Fill the screen in black
if(1):printf("\nHello ")   //not null
if(0):printf("-Error-")    //not executed because null
if(123):printf("World")    //123 for example, not null
keywait()
```

//if(cond) then;instruc1;instruc2....;endif

If the condition 'cond' is true (ie!=0) then execute the instruction(s) 'instruc1', 'instruc2'... and then goto after 'endif'. If 'cond' is false, skip after endif.

Example : Print "Cond true"

```
fillscreen(black)          //Fill the screen in black
if(1) then
printf("\nCond true")
endif
keywait()
```

//if(cond) then;instruc1;instruc2....;else;instruca;instrucb;endif

If the condition 'cond' is true (ie!=0) then execute the instruction(s) 'instruc1', 'instruc2'... and then goto after 'endif' . If 'cond' is false, skip up to 'else' and execute the instruction(s) 'instruca', 'instrucb'....

Example :
Print "Cond1 is true"
Print "Cond2 is false"

```
fillscreen(black)          //Fill the screen in black
cond1=1
cond2=0

if(cond1) then
printf("\nCond1 is true")
else
printf("\nCond1 is false")
endif

if(cond2) then
printf("\nCond2 is true")
else
printf("\nCond2 is false")
endif

keywait()
```

//while(cond):instruction(s):endwhile

As long as the cond condition 'cond' is true (ie different from 0), repeat the instructions between '::'.
Same as Tibasic and C.

Example : Wait for 'esc' key to be pressed

```
fillscreen(black)                    //Fill the screen in black
while(getkey()!=kesc)
printf("\nPress ESC key...")
endwhile
printf("\nExited")
keywait()
```

//for(statement1,statement2,statement3);code block;endfor

There are two types of 'for' loop. This is the first one. Like in C.
When you know exactly how many times you want to loop through a block of code, use the 'for' loop instead of a 'while' loop.

- Statement1 is executed (one time) before the execution of the code block.
- Statement2 defines the condition for executing the code block.
- Statement3 is executed (every time) after the code block has been executed.

Example : Print « 0 1 2 3 4 »

```
fillscreen(black)                    //Fill the screen in black
for(i=0,i<5,i++)
printf("\n%d",i)
endfor
keywait()
```

//for(var,begin_value,end_value,step_value);code block;endfor

This is the second type of 'for' loops.
Execute the instructions between the ';;' ('code block') by varying the variable 'var' from the value 'begin_value' to the value 'end_value' with a step of 'step_value'. Care must be taken that the end value is reached to avoid an infinite loop.
Same as Tibasic

Example : Print « 0 1 2 3 4 »

```
fillscreen(black)                    //Fill the screen in black
for(i,0,4,1)
printf("\n%d",i)
endfor
keywait()
```

//break

The 'break' statement ends the loop ('for' or 'while') immediatly when it is encontered.
See 'continue' for example.

//continue

The 'continue' statement skips the current iteration of the loop ('for' or 'while') and continues with the next iteration.

Example : Print '001234'

```
fillscreen(black)                    //Fill the screen in black
for(i,0,4,1)
printf("%d",i)
break                              //break the 'for' loop instantly
endfor
                     //Fill the screen in black
for(i,0,4,1)
printf("%d",i)
continue             //launch the next iteration of the 'for' loop
printf("Not executed")          //not executed
endfor
keywait()
```

//finish()

Ending the program (return to os). Be sure to have freed all memory blocks that you have eventualy created (with 'free()' function).

//when(condition,instruc_true,instruc_false)

If 'condition' is true, so execute and return the instruction 'instruc_true'. If false, return the instruction 'instruc_false'.
C equivalent : « condition ? instruc_true : instruc_false »

Example : Display :
« true-1 »
« false-2 »

```
fillrect(0,0,320,240,0)

when(1,printf("\ntrue-1"),printf("\nfalse-1"))
when(0,printf("\ntrue-2"),printf("\nfalse-2"))
keywait()
```

Newprogcx functions definitions

//func fcname([v1],...):[local([va],...]:function body:endfunc
Create a function named 'fcname', with eventual(s) argument(s) 'v1'... (maxi 6). These argument(s) is/are local(s).
You can defined local var(s) (maxi 10) with local instruction(just next line of 'func...'.
You can do recursive functions.
Use the 'return' or 'return value' to escape the function (without or with a return value 'value'). If no 'return *', the function escape when reaching 'endfunc'.

Example 1:
print to the screen :
In 'square()', a=30, v=5
square(5)=25, a=10, v=20

```
a=10
v=20

fillscreen(black)          //erase screen in black
```

```
func square(v)        //your own function named 'square'. 'v' is local
local(a)              //The var 'a' is local
a=30                  //storing 30 in the local var 'a'
printf("\nIn 'square()', a=%d, v=%d",a,v)
return v*v            //return v²
endfunc

printf("\nsquare(5)=%d, a=%d, v=%d",square(5),a,v)

keywait()
```

Example 2: C equivalent program

```
int a=10,v=20;

int square_fc(int v)
{
     int a=30;
     printf("\nIn 'square_fc()', a=%d, v=%d",a,v);
     return v*v;     //return v*v
}

void main(void)
{
     printf("\nsquare_fc(5)=%d, a=%d, v=%d",square_fc(5),a,v) ;

     keywait() ;     //wait for a key pressed
}
```

Sprites format in Newprog CX :

NewprogCX sprites functions use the n2DLib library :
n2DLib is a C graphics library for the TI-Nspire calculators intended to be used with the Ndless 3rd-party jailbreak.

The lib (can) uses buffering to display smoothly ; that means that all drawing is done to a separate buffer in memory which is then copied to the screen.

All colors, unless specified, must be in R5G6B5 format, meaning a 16-bits number organized this way :

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Color	R	R	R	R	R	G	G	G	G	G	G	B	B	B	B	B

n2DLib uses a variation on the NTI image format to store images. All images must be arrays of unsigned short organized this way :

Entry	0	1	2	3-...
Use	Width in pixels	Height in pixels	Transparent color	Image data

//colorrgb(red,green,blue)

Return the combined color value (in unsigned short 'us', two bytes length ; unsigned value) from the red, green and blue components.
This combined color value is used by all graphicals functions of Newprogcx.
See 'Sprites format in Newprog CX' section for more informations.

Example :

```
colorrgb(0,0,0)=0     //black
colorrgb(255,255,255)=-1   //or 0xFFFF - white
```

Example : Fill the screen in black,red, lowered red,green,blue, purple and white.

```
fillscreen(colorrgb(0,0,0))           //fill screen in black
keywait()                             //wait for a key press
fillscreen(colorrgb(255,0,0))         //fill screen in red
keywait()
fillscreen(colorrgb(120,0,0))         //fill screen in lowered red
keywait()
fillscreen(colorrgb(0,255,0))         //fill screen in green
keywait()
fillscreen(colorrgb(0,0,255))         //fill screen in blue
keywait()
fillscreen(colorrgb(255,0,255))       //fill screen in purple
keywait()
fillscreen(colorrgb(255,255,255))     //fill screen in white
keywait()
```

//'+' or '-' with pointers in Newprogcx

Adding or substract from a pointer works in a different way than in C. In Newprogcx, it is really adding or substracting by the value specified (and not dependant of the kind of data).

Example with 'ui' (the length of a 'ui' data is 4 bytes) :

```
dataui list[5]={1,2,3,4,5}

printf("\nlist[0]=%d ",*(ui*)(list+0))     //print 1
printf("\nlist[1]=%d ",*(ui*)(list+4))     //print 2
printf("\nlist[2]=%d ",*(ui*)(list+8))     //print 3
keywait()
```

In C, you will have to do like this :

```
unsigned int list[5]={1,2,3,4,5} ;

printf("\nlist[0]=%d ",*(unsigned int*)(list+0)) ;    //print 1
printf("\nlist[1]=%d ",*(unsigned int*)(list+1)) ;    //print 2
printf("\nlist[2]=%d ",*(unsigned int*)(list+2)) ;    //print 3
getchar() ;
```

Messages and Requests popups

//msgbox(title_str,msg_str)

Popup a message box. Its title is 'title_str' and below the msg is 'msg_str'. The box dimensions are automatics.
To avoid bug, don't use the '\n' character in both strings.

Example : Show a message and print if OK key or ESC key has been pressed


```
fillscreen(black)
rtn=msgbox("Message","Hello World !")

if(rtn==1) then
printf("Enter key pressed\n")
else
printf("Esc key pressed\n")
endif

keywait()
```

//request1str(title_str,msg_str,default_str,(ad*)str)

Open a dialog box that allows to input a string by the user.
Its title will be 'title_str', the main message will be 'msg_str'.
The input area is set intialy with 'default_str'. The maximum input string length is about 15 characters (so a buffer of 16 bytes).
You can lock the upper case input by using 'SHIFT' key.
Be sure to not use the '\n' character.
The output is true until the next launch of this function, so store it in a buffer can be useful.

Example : Input your name and age and display then in a message box.


```
//buffers definitions
datauc namebuf[20]={} //16 length is enough
datauc agebuf[20]={}  //16 length is enough
datauc msgbuf[60]={}  //60 length is enough

request1str("Enter your Name","Your Name","Peter",(ad*)str)
```

```
strcpy(namebuf,str)

lbl lblage
request1str("Enter your Age","Your Age","20",(ad*)str)
if(not isnumstr(str)):goto lblage
strcpy(agebuf,str)

sprintf(msgbuf,"%s, %s is yours age",namebuf,agebuf)

msgbox("Hello !",msgbuf)
```

Memory functions

//(ad*)
Return the pointer of a variable or a list element or a matrix element
(ad*)var or (ad*)list[elem] or (ad*)mat[row][col].

Example : Modify the value of the variable 'a' with its pointer.

```
func varmodif(varptr,newvalue)
*(si*)varptr=newvalue
endfunc


fillscreen(black)
a=10            //initial value
printf("\nInitial value of var a=%d",a)//print Initial value of var a=10

varmodif((ad*)a,20)
printf("\nNew value of var a=%d",a)//print new value of var a=20
keywait()
```

Example :disp :
0-0=1 1-1=4
0-0=1 1-1=66

```
dataui m[2][2]={1,2,3,4}   //define a matrix 'm' with 4 elements
printf("\n0-0=%d 1-1=%d",m[0][0],m[1][1]) //so print '1' and '4'
*(ui*)(ad*)m[1][1]=66      //sto the value '66' to the elem [1][1] thanks
to its pointer.
printf("\n0-0=%d 1-1=%d",m[0][0],m[1][1])
keywait()
```

//Reading or storing values from pointers

<u>For reading :</u>
(uc)ptr; *(sc*)ptr; //returns the unsigned or signed char value at the adress 'ptr' (1 byte length).
C equivalent : *(unsigned char *)ptr or *(signed char *)ptr

(us)ptr; *(ss*)ptr; //returns the unsigned or signed short value at the adress 'ptr' (2 bytes length)
C equivalent : *(unsigned short *)ptr or *(signed short *)ptr

(ui)ptr; *(si*)ptr; //returns the unsigned or signed int value at the adress 'ptr' (4 bytes length)
C equivalent : *(unsigned int *)ptr or *(signed int *)ptr

For storing : In a same way than for reading

Store to a memory area pointed by 'ptr', the value 'value'.
1-byte lenght :
(uc)ptr=value for storing unsigned char value, C equivalent : *(unsigned char*)ptr=value
(sc)ptr=value for storing signed char value, C equivalent : *(signed char*)ptr=value
2-bytes lenght :
(us)ptr=value for storing unsigned short value, C equivalent : *(unsigned short*)ptr=value
(ss)ptr=value for storing signed short value, C equivalent : *(signed short*)ptr=value
4-bytes lenght :
(ui)ptr=value for storing unsigned int value, C equivalent : *(unsigned int*)ptr=value
(si)ptr=value for storing signed int value, C equivalent : *(signed int*)ptr=value

Example : Fill the screen in white (color=0xFFFF or -1), then in green

```
ptr=getlcdad()

//fill in white (slower)
for(i=0,i<lcdnbbytes,i++)
*(uc*)(ptr+i)=-1                        //-1 for white
endfor
keywait()
//fill in green (faster because two bytes at once)
for(i,0,lcdnbpix-1,1)
*(us*)(ptr+2*i)=colorrgb(0,255,0)     //in green
endfor
keywait()
```

//bufon()

Set to double buffering mod (full screen, 16bpp). Then, the graphics functions will draw on the buffer. Return the buffer adress (0 if failed). If you wish later, you can retrieve the buffer adress by using the 'getlcdad()' function. You can flip to the real screen by using the 'bufflip()' function. This function is useful to avoid flickering pixels, but needs time. Before the end of the program, you must call 'bufoff()' function to retrieve memory.

Example :

```
//first without buffering. Will flicker
fillscreen(black)
printf("\nUse arrows to move the rect (no buffering). Esc to quit")
keywait()
x=0
```

```
k=0
while((k=getkey())!=kesc)
if(x>0 and k==kleft) then
fillscreen(black)     //fill the screen in black
x--
fillrect(x,50,40,40,-1)    //draw new position
endif
if(x+40<320 and k==kright) then
fillrect(x,50,40,40,0)     //erase
x++
fillrect(x,50,40,40,-1)    //draw new position
endif
delay(2)   //wait 2 msec
endwhile

//Now, with buffering. No flickering
if(not bufon()):finish()
fillscreen(black)
printf("\nUse arrows to move the rect (with buffering). Esc to quit")
keywait()
x=0
k=0
while((k=getkey())!=kesc)
if(x>0 and k==kleft) then
fillrect(x,50,40,40,0)     //erase
x--
fillrect(x,50,40,40,-1)    //draw new position
bufflip()
endif
if(x+40<320 and k==kright) then
fillrect(x,50,40,40,0)     //erase
x++
fillrect(x,50,40,40,-1)    //draw new position
bufflip()
endif
delay(2)   //wait 2 msec
endwhile
bufoff()
```

//bufoff()

Stop and free the buffering screen (if you have used the 'bufon()' function before). Restore the drawing area to the real screen lcd adress.

//bufflip()

Draw the buffered screen. Must be used with 'bufon()' and 'bufoff()'. Internaly, it copy the content of the drawing buffer area to the real screen lcd adress.

//updaterect(x,y,w,h)

To be used after the 'bufon()' function. Instead the 'bufflip()' function, 'updaterect()' refresh only the {x,y,w,h} screen area and not the entire screen. So it can be faster to use this function instead of 'bufflip()' function.

//getlcdad()

Return the current drawing area pointer. All graphical functions draw on it.
If 'bufon()' has been called previously, 'getlcdad()' will return the graphical buffer adress.

//setlcdad(new_graphing_area_ptr)

Set the current drawing area to 'new_graphing_area_ptr'. All graphical functions will draw on it.

//fillscreen(color)

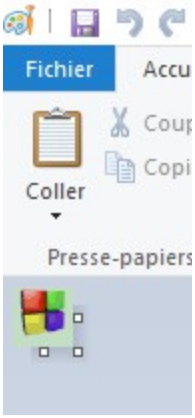
Fill the current graphic memory area with the color 'color'.

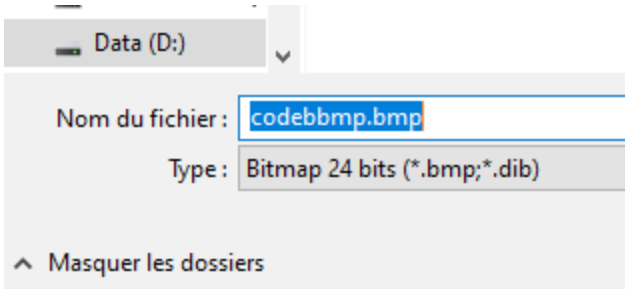
Example : Fill in white first, then in green

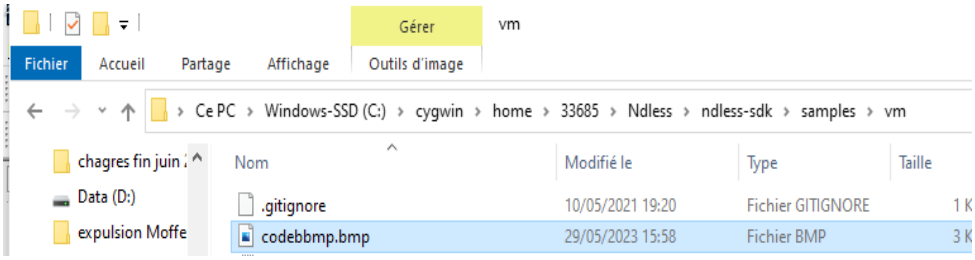
```
fillscreen(-1)
keywait()
fillscreen(1000)
keywait()
```

//loadbmp(pic_name_str,mask_color)

Load a bitmap file (24 bbp) named 'pic_name_str'.tns and convert it to the sprite format of Newprogcx. Returns the created sprite ptr. You have to free this memory area before the end of your program by using the 'free()' function.
The created sprite will have the same width and height than the bitmap file.
The mask_color will be equal to 'mask_color'. The mask color is a color which will not be drawn.
To create a bitmap file, use for example MS-paint (on windows). Save it with the 'Bitmap 24 bits' format. Change now the 'bmp' extension to 'tns' extension. When done, transfer it to your calc.
Here below, an example :

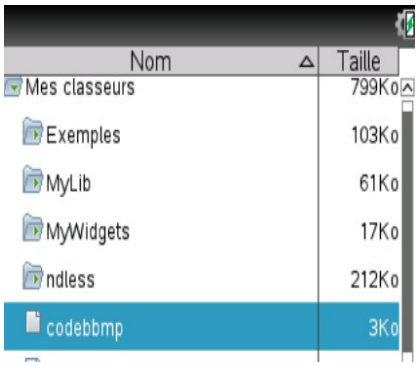






Change the extension to '.tns'.

Transfert it to your nspire :



See the 'sprite()' function for example.

//sprite(sprt,x,y)

Draw the sprite 'sprt' at the coordinate x and y (when x=0 and y=0 for the upper left of screen).

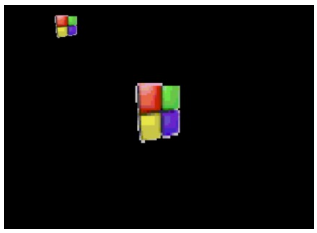
Example : Display a sprite

```
pic=loadbmp("codebbmp.tns",-1) //-1=white is the transparency  color
if(pic==0) then       //if error when loading sprite, so quit the program
printf("\nError mem")
keywait()
finish()
endif

fillscreen(black)          //fill the screen in black (==0)
sprite(pic,50,50)     //draw the sprite at x=50 and y 50
keywait()             //wait for a key pressed

free(pic)             //free the memory area of the sprite
```

//spritescaled(pic,x_center,y_center,newwidth,newheight)

Print scaled sprite. Scale an existing sprite 'pic' with the new width 'newwidth' and the new height 'newheight' at the coordinates x_center and y_center (the center of the new scaled sprite).

Example : Draw original (uper left) and scaled sprite (center).
Previously, the 'codebbmp.tns' must be on the calc.



```
pic=loadbmp("codebbmp.tns",-1) //-1=white is the transparency  color
if(pic==0) then
printf("\nError mem")
finish()
endif

fillscreen(black)
spritescaled(pic,lcdwidth/2,lcdheight/2,60,80)
sprite(pic,50,10)

keywait()
while(not keytest(kesc)):endwhile
lbl fin
if(pic):free(pic)
```

//spriterotated(sprite,x_center,y_center,theta)

Draw a rotated sprite from 'sprite'. The center of the rotated sprite will be drawn to coordinates 'x_center' and 'y_center'.

if theta=128 rotated of 180°
if theta=-128 rotated of -180°
if theta=0 rotated of 0°

Example : Rotate a sprite with arrows keys. Previously, the 'codebbmp.tns' must be on the calc.


```
pic=loadbmp("codebbmp.tns",-1)
if(pic==0) then
printf("\nError mem")
finish()
endif

spriterotated(pic,160,120,0)
a=0
while(not keytest(kesc))
if(keytest(kup)) then
a++
spriterotated(pic,160,120,a)
```

```
endif
if(keytest(kdown)) then
a--
spriterotated(pic,160,120,a)
endif

endwhile
lbl fin
if(pic):free(pic)
```

//getspritew(pic) and getspriteh(pic)
Return width or height of the sprite 'pic'.

Example :
```
w=getspritew(pic);h=getspriteh(pic)
printf("\nw=%d h=%d",w,h)
```

//setpix(x,y,color)

Draw the pixel at 'x' and 'y' coordinates with the color 'color' to the actual drawing memory area (see 'setlcdad()' and 'getlcdad()' functions). The color is writen on 2 bytes length data.

Example : Fill the screen in white :

```
for(x,0,lcdwidth-1,1)
for(y,0,lcdheight-1,1)
setpix(x,y,-1)
endfor
endfor

keywait()
```

//getpix(x,y)

Return the color (2 bytes len value) of the pixel at the (x,y) coordinates on the actual drawing memory area (see 'setlcdad()' and 'getlcdad()' functions).

//fillrect(x,y,width,height,color)

Fill in 'color' color the drawing memory area pointed by ('x','y') and with 'width' width and 'height' height.

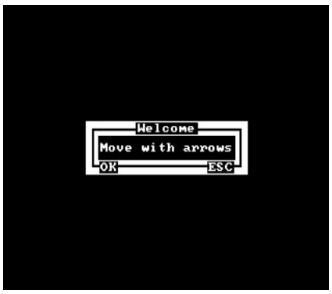
Example : Fill the screen in white

```
fillrect(0,0,lcdwidth,lcdheight,-1)
keywait()
```

//fillcircle(xc,yc,radius,color)

Draw a filled circle in color 'color' which center is {'xc','yc'} and radius is 'radius'.
If only one pixel is out of the screen, no pixels will be drawn at all.

Example : Move a filled circle with arrows keys (left and right)



```
fillscreen(black)                       //fill screen in black
x=160;y=120

fillcircle(x,y,5,white)                 //init filled circle

msgbox("Welcome","Move with arrows")  //draw a message box
msgbox("Ready ?","Press ESC to exit") //draw a message box

while((k=getkey())!=kesc)               //Wait until ESC key is pressed
if(x-5>0 and k==kleft) then             //if left arrow pressed
x--
fillrect(x-6,y-6,15,15,black)           //erase previous circle
fillcircle(x,y,5,white)                 //draw new position
endif
if(x+5<lcdwidth-1 and k==kright) then //if right arrow pressed
x++
fillrect(x-6,y-6,15,15,black)
fillcircle(x,y,5,white)
endif
endwhile
```

//drawcircle(xc,yc,radius,color)

Draw a circle in color 'color' which center is {'xc','yc'} and radius is 'radius'. If some pixels are out of screen, the inside (of screen) pixels will be drawn (instead of 'fillcircle()').

//scrollleft(x,y,w,h,nb_pix)

Perform scrolling to the left of nb_pix pixels of the rect(x,y,w,h) of the current lcd adress (retrieved whith the function 'getlcdad()').

If nb_pix is not even (1,3,5,7,...) :
The execution speed will be two times slowers than with an even nb_pix (about 25 executions by second for full screen)

If nb_pix is even :
In this case, the x,y,and h parameters must be even.
The execution speed will be about two times faster (about 50 executions by second for full screen) than with a not even nb_pix.

Example : Use the arrows keys to scroll to left or right of the full screen.

```
pic=loadbmp("codebbmp.tns",-1)
```

```
if(pic==0) then
printf("\nError mem")
keywait()
finish()
endif

fillscreen(black)
sprite(pic,50,10)

while(not keytest(kesc))
if(keytest(kleft)) then
scrollleft(0,10,320,50,1)
endif
if(keytest(kright)) then
scrollright(0,10,320,50,1)
endif
endwhile

lbl fin
if(pic):free(pic)
```

Example : Create a scrolling to left (2 pixels here) in pure newprogcx language. (just 3,3 times slower)

```
void dataus a[240][320]
a=getlcdad()

func scroll(nbpixels)      //scrolling to left
for(y,0,239,1)
for(x,0,319-nbpixels,1)
a[y][x]=a[y][x+nbpixels]
endfor
endfor
endfunc
```

//scrollright(x,y,w,h,nb_pix)

Similar to scrollleft() but scroll to the right

//scrolldown(x,y,w,h,nb_pix)

Similar to scrollleft() but scroll to the bottom.
x,y,w, can be even or not (and same speed for each nb_pix), 40 executions by second for full screen.

//scrollup(x,y,w,h,nb_pix)

Similar to scrolldown() but scroll to up.
x,y,w, can be even or not (and same speed for each nb_pix), 40 executions by second for full screen.

//printf(format,[a1],...,a[5])

Similar to the C printf function. Allow to print chars, numbers and strings to screen.
Set the font color with the 'settxtcolor()' function.
Retrieve the X position with 'gettxtposx()' function.

Retrieve the Y position with 'gettxtposy()' function.
Set the {x,y} position pen with the 'settxtpos()' function.

'Parameters' (optionnals) : a[1] to a[5]

'format' - This is the string that contains the text to be written to screen. It can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested. Format tags prototype is %specifier, which is explained below.

Specifier :
c == Character
d == Signed decimal integer
o ==Signed octal
s == String of characters
u == Unsigned decimal integer
x == Unsigned hexadecimal integer
X == Unsigned hexadecimal integer (capital letters)

Example : Print 'My age is 20'

```
fillscreen(black)                    //clear screen in balck
printf("\n %s%c is %d","My ag",charord("e"),20)
keywait()
```

//drawstrxy(x,y,str_ptr)

Draw to the current screen, at the position {'x','y'}, the string 'str_ptr'. Do not affect the current pen position (functions 'settxtpos()', 'gettxtposx',etc..).
The color is the current text color

Example : Print :


```
fillscreen(black)
drawstrxy(100,50,"Hello World")      //Do not affect the current pen pos
drawstrxy(100,60,":-)")              //         "

printf("At the upper left corner")

keywait()
```

//sprintf(dest_buffer,format,[a1],...,a[5])

Similar to the 'printf()' function but send the formated text output to the buffer 'dest_buffer'.

Example : Print 'Hello World'.

```
func helloworld(a1,a2)
local(buf)
buf=malloc(strlen(a1)+strlen(a2)+1+1) //+1 for 0 terminating char and +1
for space char
if(buf==0):finish()
sprintf(buf,"%s %s",a1,a2)
printf(buf)
free(buf)
endfunc

fillscreen(black)
helloworld("Hello","World")
```

//setfont(f)

Set the font type :
'f'= 'font6x8' or '0' little font width=6, height=8
'f'= 'font8x8' or '1' little font width=8, height=8
'f'= 'font8x12' or '2' little font width=8, height=12
'f'= 'font16x16' or '3' little font width=16, height=16

Example : Print « Hello World ! » with the 4 fonts


```
fillscreen(black)

setfont(font6x8)      //font6x8=0
printf("\nHello World !")

setfont(font8x8)      //font8x8=1
printf("\nHello World !")

setfont(font8x12)          //font8x12=2
printf("\nHello World !")

setfont(3)      //font16x16=3
printf("\nHello World !")


keywait()
```

//settxtpos(x,y)

Position of text functions ('printf()',etc...) to the x,y coordinates.
The x=0 and y=0 are to the upper left of the screen.

Example 1 : Print text to the upper left corner and on the middle-right


```
fillscreen(black)

settxtpos(0,0)  //to the upper left corner
printf("Hello World 1")

settxtpos(lcdwidth/2,lcdheight/2)
printf("Hello World 2")

keywait()
```

//gettxtposx()

Returns the x position of text functions.

//gettxtposy()

Returns the y position of text functions.

//settxtcolor(color)

Set the text color for all characters printing functions ('printf()',...). (the 'colorrgb()' function can be useful).

Example :


```
fillscreen(black)

settxtcolor(colorrgb(255,0,0))  //text color=red
printf("Hello World 1")

settxtcolor(colorrgb(0,255,0))  //text color=green
settxtpos(lcdwidth/2,lcdheight/2)
printf("Hello World 2")

keywait()
```

//charord(char_str)

```
Return the value of the first char of char_str
example : charord(«a») = 97          //in C = 'a'
```

//for(initialisation,cond,post instruction):instructions...:endfor [1]
// or
//for(var,begin value,final value,step):instructions...:endfor [2]

Classicals loops. The second case ([2]) is 12 times faster than the first case [1]
The instructions 'break', 'continue' are availables (behave like in C) :
- 'break' for exiting loop (so goto after endfor)
- 'continue' triger to the beginning of the loop (with post instruction trigered or isz or dsz)

```
Example 1 : print to the screen 0 1 2 3 4
for(i=0,i<5,i++)
printf(«%d »,i)
endfor

Example 2 : print to the screen 0 1 2 3 4
for(i=0,i<10,i++)
printf(«%d »,i)
if(i>=4):break
endfor

Example 3 : print to the screen 0 1 2 3 4 5
for(i,0,5,1)
printf(«%d »,i)
endfor

Example 4 : print to the screen 0 1 2 3 4
for(i,0,10,1)
printf(«%d »,i)
if(i<=3) then
continue
else
break
endif
endfor
```

Input key detection functions :

Introduction – constants :

'keynum' can take the values below :

```
    {"kctrl",7},
    {"kreturn",64},
    {"kenter",37},
    {"kspace",70},
```

```
{"kminus",36},
{"kz",69},
{"kdot",35},
{"ky",68},
{"K0",34},
{"kx",67},
{"ktrig",12},
{"kcomma",56},
{"kplus",30},
{"kw",66},
{"k3",29},
{"kv",65},
{"k2",28},
{"ku",63},
{"k1",27},
{"kt",62},
{"kexp",25},
{"kpi",47},
{"kquestion",46},
{"kquesex",46},
{"ksub",31},
{"ks",61},
{"k6",22},
{"kr",60},
{"k5",21},
{"kq",59},
{"k4",20},
{"kp",58},
{"ktenx",26},
{"kee",38},
{"kmul",23},
{"ko",57},
{"k9",15},
{"kn",54},
{"k8",14},
{"km",53},
{"k7",13},
{"kl",52},
{"ksquare",19},
{"kii",100},
{"kplus",30},
{"kdiv",24},
{"kk",51},
{"ktan",101},
{"kj",50},
{"kcos",102},
{"ki",49},
{"ksin",103},
{"kh",48},
{"kexp",18},
{"kgthan",104}, //???
{"kapostrophe",105},
{"kcataolog",17},
{"kfrac",24},
{"kg",45},
{"krp",33},
{"kf",44},
{"klp",32},
```

```
    {"ke",43},
    {"kvar",9},
    {"kd",42},
    {"kdel",10},
    {"klthan",110},
    {"kflag",55},
    {"kclic",1000},
    {"kc",41},
    {"khome",2},
    {"kb",40},
    {"kmenu",6},
    {"ka",39},
    {"kesc",1},
    {"kbar",3},
    {"ktab",5},
    {"kequ",11},

    {"kup",1008},
    {"kupright",10086},
    {"kright",1006},
    {"krightdown",10062},
    {"kdown",1002},
    {"kdownleft",10024},
    {"kleft",1004},
    {"kleftup",10048},
    {"kshift",8},
    {"kdoc",4},
    {"ktrig",12},
    {"kscratch",2000}
```

//keytest(keynum)

Return non 0 if the key matching with keynum is pressed while the execution of this instruction.
'keynum' is limited to (instead 'getkey()' and 'keywait()') :
kup,kright,kleft,kdown,kupright,krightdown ,kdownleft,kleftup,
kesc,khome,kdoc,ktab,kmenu,kctrl,kshift,kvar,kdel,kenter.

This function is the fastest way to detect keys pressed. It is too the only way to detect multi keys press.

Example : Wait the 'esc' key to be hit

```
fillscreen(black)     //erase screen
printf("\nWaiting for 'ESC' key to be pressed")
while(not keytest(kesc))                          //cte 'kesc'==1
endwhile
printf("\nFinish")
while(not keytest(kenter))
endwhile
```

//getkey()

If no key is pressed, return 0. If a key is pressed during the execution of this instruction, return its key_num (see before for values...).

//keywait()
Wait for a key to be hit. Return the key code (same values than 'getkey()').

Example : Print chars

```
c=0
fillscreen(black)
printf("\nHit keyboard :")
while(c!=kesc)
c=keywait()
if(keyascii(c)):printf("%c ",keyascii(c))
endwhile
```

//keyascii(key_num)

Return the ascii code associated to key_num. Retrun 0 if not a valid ascii char.

Example 1 : Print 'hello world'

```
fillscreen(black)
printf("\n%cello world",keyascii(kh))
while(getkey()!=kesc)
endwhile
```

Example 2 : Print chars matching with hitted keys

```
output cba
lbl main

fillscreen(black)
printf("\nHit keyboard :")
while((k=getkey())!=kesc)
if(c=keyascii(k)) then          //print only printable chars
printf("%c",c)
endif
endwhile
```

//isnumstr(num_str)

Returns if the string pointer 'num_str' contains a pure numeric value. Characters '+' and '-' are forbidens (so 'isnumstr()' returns 0 value).

Example : Ask your age and display it if it is a correct numeric input.

```
lbl lblage
request1str("Enter your Age","Your age","20",(ad*)str)
if(not isnumstr(str)):goto lblage
strcpy(namebuf,str)

msgbox("Your age is",str)
```

//spritepart(sprite,x,y,{sprt_x,sprt_y,sprt_w,sprt_h})

Display the sub part '{,,,}' of sprite 'sprt' to x and y coords to the current graphic memory area.

Note : As some similarity with the 'blit()' function.

Example : display the part {0,0,20,20} of the sprite 'pic' (from 'codebbmp.tns')

```
pic=loadbmp("codebbmp.tns",-1)  //sprite w>=20 h>=20 expected
if(pic==0) then
printf("\nError loading 'codebbmp.tns'");
keywait()
goto fin
endif

fillscreen(black)     //erase screen
sprite(pic,100,100)        //source sprite. Full display
spritepart(pic,10,30,{0,0,20,20})     //display the part 0,0,20,20 of pic

keywait()

lbl fin
if(pic):free(pic)
```

//newsprite(width,height,mask_color)

Allocate memory for a sprite (width, height and mask color specified). Don't forget to use 'free()' before the end of program.

Example : Copy a sprite (from 'pic' (from bitmap file) to destination 'nwsprt' and dispay it

```
nwsprt=0
pic=0

func myspritecopy(spriteori)
local(spritecopy,sw,sh,x,y)
spritecopy=newsprite(getspritew(spriteori),getspriteh(spriteori),-1)
      //Alloc memory
if(spritecopy==0) then
printf("\nError mem newsprite");
keywait()
goto end
endif
sw=getspritew(pic)
sh=getspriteh(pic)
for(x,0,sw-1,1)
for(y,0,sh-1,1)
(*(us*)(spritecopy+3*2+x*2+y*2*sw))=(*(us*)
(spriteori+3*2+x*2+y*2*sw)) //the use of 'memmove()' should be faster
endfor
endfor
return spritecopy
endfunc

pic=loadbmp("codebbmp.tns",-1)  //sprite w=30 h=30, "codebbmp.tns"
should be exist
```

```
if(pic==0) then
printf("\nError loading 'codebbmp.tns'");
keywait()
goto end
endif

nwsprt=myspritecopy(pic)

fillscreen(black)
sprite(pic,50,100)    //original
sprite(nwsprt,100,100)     //the copy (the two sprites displayed are the
same)


keywait()

lbl end
if(pic):free(pic)
if(nwsprt):free(nwsprt)
finish()
```

An another way of the ' myspritecopy()' function with 'memmovess ()' (faster)

```
func myspritecopy(spriteori)
local(spritecopy,sw,sh,x,y)
spritecopy=newsprite(getspritew(spriteori),getspriteh(spriteori),-1)
      //Alloc memory
if(spritecopy==0) then
printf("\nError mem newsprite");
keywait()
goto end
endif
sw=getspritew(pic)
sh=getspriteh(pic)    //no need

for(y,0,sh-1,1)
memmovess(getfirstpix(spritecopy)+y*2*sw,getfirstpix(spriteori)
+y*2*sw,sw)
endfor

return spritecopy
endfunc
```

Another way :

```
func myspritecopy(spriteori)
local(spritecopy,sw,sh,x,y)
spritecopy=newsprite(getspritew(spriteori),getspriteh(spriteori),-1)
      //Alloc memory
if(spritecopy==0) then
printf("\nError mem newsprite");
keywait()
goto end
endif
sw=getspritew(pic)
sh=getspriteh(pic)    //no need
```

```
for(y,0,sh-1,1)
memmovess(getfirstpix(spritecopy)+y*2*sw,getfirstpix(spriteori)
+y*2*sw,sw)     //copy each line
endfor

return spritecopy
endfunc
```

Another another way : with 'malloc()'

```
func myspritecopy(spriteori)
local(spritecopy)
spritecopy=malloc(getspritesize(spriteori))     //Alloc memory
if(spritecopy==0) then
keywait()
goto end
endif
memmove(spritecopy,spriteori,getspritesize(spriteori))
return spritecopy
endfunc
```

//getfirstpix(sprite_ptr) or getspritefirstpixadress(sprite_ptr)

Return the adress of the first pixel of the sprite 'sprite_ptr'.
C equivalent :

```
unsigned short * getfirstpix(unsigned short *sprite_ptr)
{
     return sprite_ptr+3 ; //or : « return (unsigned
char*)sprite_ptr+3*2 ; »
}
```

//getspritew(sprite_ptr)

Returns the width of the sprite 'sprite_ptr'.
Another way to retrieve the width of the sprite :'*(us*)(sprite_ptr+0)'

//getspriteh(sprite_ptr)

Returns the height of the sprite 'sprite_ptr'.
Another way to retrieve the height of the sprite :'*(us*)(sprite_ptr+2)'

//getspritemask(sprite_ptr)

Returns the mask color of the sprite 'sprite_ptr'. So, each pixel of the sprite who have this value will not be overlapsed.
Another way to retrieve the mask color of the sprite :
'*(us*)(sprite_ptr+4)'

//setspritemask(sprite_ptr,mask_color)

Set the mask color of the sprite 'sprite_ptr' to 'mask_color'.
Another way to set the mask color of the sprite :
'*(us*)(sprite_ptr+4=mask_color)'

//getspritesize(sprite_ptr)

Returns the total size in bytes of the sprite 'sprite_ptr' (including the first 6 bytes containing the width, height and mask color)

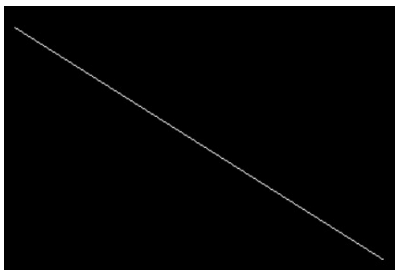
Example : Pure Newprogcx getspritesize equivalent

```
func mygetspritesize(spriteptr)
return getspritew(spriteptr)*2 + getspriteh(spriteptr)*2 + 2*3
     //2*3=header data of the sprite
endfunc
```

//drawline(x1,y1,x2,y2,color)

Draw a line to the current drawing area with the color 'color' between the (x1,y1) and (x2,y2) coordinates.

Example : Diagonal line in white



```
fillscreen(black)

drawline (10,20,310,230,-1)

keywait()
```

//blit({src_ptr,src_width,src_height},{src_x,src_y,src_width,src_height}, {dest_ptr,dest_width,dest_height},{dest_x,dest_y})

This performs a fast blit from the source surface (defined by the first pixel pointer (src_ptr) and its width (src_width) and its height (src_height)) to the destination surface (defined by the first pixel pointer (dest_ptr) and its width (dest_width) and its height (dest_height).
This function is very powerful.
Be sure to blit in a correct memory area or the calculator will crash !

Example 1 : My own 'copysprite()' function :

```
func copysprite(sprt)
local(s,w,h,m)
w=getspritew(sprt)
h=getspriteh(sprt)
m=getspritemask(sprt)
s=newsprite(w,h,m)    //Alloc memory - New void sprite
if(s==0) then
printf("\nMem Error")
keywait()
if(pic):free(pic)
finish()   //exit the program
endif
//copy all pixels  :
blit({getfirstpix(sprt),w,h},{0,0,w,h},{getfirstpix(s),w,h},{0,0})
return s
endfunc

fillscreen(black)     //erase the screen in black
pic=0
pic=loadbmp("codebbmp.tns",-1) //-1=white is the transparency  color
if(pic==0) then
printf("\nError mem")
keywait()
finish()
endif

printf("\nOriginal sprite (left), copied sprite (right)  :")
sprite(pic,20,100)
nsprite=copysprite(pic) //copy the sprite
sprite(nsprite,200,100)
keywait()

free(pic)
free(nsprite)
```

Example 2 : Scroll the screen horizontally with keys

```
fillscreen(black)     //erase the screen in black
pic=0
pic=loadbmp("codebbmp.tns",-1) //-1=white is the transparency  color
if(pic==0) then
printf("\nError mem")
keywait()
finish()
endif

//Fill the screen with the sprite
w=getspritew(pic)
h=getspriteh(pic)
nx=lcdwidth/w
ny=lcdheight/h
for(x=0,x<nx,x++)
for(y=0,y<ny,y++)
sprite(pic,x*w,y*h)
endfor
endfor
```

```
scr=malloc(lcdnbbytes)     //lcdnbbytes = lcdwidth(320)*lcdheight(240)*2
if(scr==0) then
free(pic)
finish()
endif
memmove(scr,getlcdad(),lcdnbbytes)    //copy the screen in th buffer
pointed by scr

keywait()
x=0
while((k=getkey())!=kesc)
if(k==kright and x<lcdwidth):x++
if(k==kleft and x>0):x--
blit({scr,lcdwidth,lcdheight},{x,0,lcdwidth-x,lcdheight},
{getlcdad(),lcdwidth,lcdheight},{0,0})
endwhile

free(scr)
free(pic)
```

Example 3 : Do the same than the 'sprite()' function (without mask management).

```
output cba
lbl main

func mysprite(sprt,x,y)
blit({getfirstpix(sprt),getspritew(sprt),getspriteh(sprt)},
{0,0,getspritew(sprt),getspriteh(sprt)},{getlcdad(),lcdwidth,lcdheight},
{100,10})
endfunc

fillscreen(black)
pic=loadbmp("codebbmp.tns",-1) //-1=white is the transparency  color
if(pic==0) then
printf("\nError mem")
keywait()
finish()
endif

mysprite(pic,10,10)
keywait()

if(pic):free(pic)
```

//memmove(dest_ptr,src_ptr,nb_bytes) or memcpy(dest_ptr,src_ptr,nb_bytes)

These two functions do the same.
Copies the values of *nb_bytes* bytes from the location pointed by *src_ptr* to the memory block pointed by dest_ptr. Copying takes place as if an intermediate buffer were used, allowing
the *destination* and *source* to overlap.

The underlying type of the objects pointed by both
the *source* and *destination* pointers are irrelevant for this function; The result is a binary copy of the data.

The function does not check for any terminating null character

in *source* - it always copies exactly *nb_bytes* bytes.

To avoid overflows, the size of the arrays pointed by both the *destination* and *source* parameters, shall be at least *nb_bytes* bytes.

Example : copy a string

```
datauc deststr[20]={}      //size of 20 >strlen(str)+1
str="Hello World  !"

memmove(deststr,str,strlen(str)+1)    //copy  in dest_str. '+1' for '\0'
fillscreen(black)                          //clear screen
printf("\n%s",deststr)                //print "Hello World  !"
keywait()
```

//memmovess(dest_ptr,src_ptr,nb_ss)
Similar to 'memmove()' but for signed short values (ss) instead bytes values. This function is useful for copying sprites for example (because one pixel is writen in a signed short value (2 bytes length)) because faster.
So 'dest_ptr' and 'src_ptr' must be even ptr (if not, will crash).

C equivalent :

```
int memmovess(signed short *dest,signed short *ori,unsigned short
nbshortt)
{
    unsigned int i;
    for(i=0;i<nbshortt;i++)
    {
        *(dest+i)=*(ori+i);
    }
    return dest;
}
```

Example : Newprogcx function for copying a sprite (spriteori)

```
func myspritecopy(spriteori)
local(spritecopy,sw,sh,x,y)
spritecopy=newsprite(getspritew(spriteori),getspriteh(spriteori),-1)
     //Alloc memory
if(spritecopy==0) then
printf("\nError mem newsprite");
keywait()
goto end
endif
sw=getspritew(pic)
sh=getspriteh(pic)    //no need

for(y,0,sh-1,1)
memmovess(getfirstpix(spritecopy)+y*2*sw,getfirstpix(spriteori)
+y*2*sw,sw)     //copy each line
endfor

return spritecopy
endfunc
```

//memset(dest_ptr,value,nb_bytes)

The function 'memset()' copies the character 'value' (an unsigned char) to the first 'nb_bytes' characters of the data area pointed by the 'dest_ptr'.

Example : Fill the screen in white

```
memset(getlcdad(),-1,lcdnbbytes)      //-1 = 0xFFFF  for white
keywait()
```

//memsetss(dest_ptr,value,nb_sshort)

The function 'memset()' copy the value 'value' (an unsigned short – 2 bytes length) to the first 'nb_sshort' of the data area pointed by the 'dest_ptr'. 'dest_ptr' must be an even adress (if not, will crash). This function is usefull to set a color to many pixels at once.

Example 1 : Fill the screen in white

```
memsetss(getlcdad(),-1,lcdnbpix)      //-1 = 0xFFFF  - white
keywait()
```

Example 2 : Draw a horizontal line in white

```
memsetss(getlcdad()+lcdwidth*2*50,-1,lcdwidth)   //y=50
keywait()
```

//strlen(str)

Return the length in bytes of the string 'str'. The Null terminating char is ignored.

Example :
strlen("Hello") is 5
strlen("Hello World") is 11.

//strcat(dest,src)

This function appends the string pointed to by 'src' at the end of the string pointed to by 'dest'.
'dest' This is a pointer to the destination array, which should contain a string, and should be large enough to contain the concatenated resulting string.
'src' This is the string to be appended. This should not overlap the destination.
Return 'dest'.

Example :


```
fillscreen(black)
datauc str1[20]="Hello "   //size of 20 is enough
str2="World"
strcat(str1,str2)
printf("\nstr1=%s",str1)
```

```
keywait()
```

//strcpy(dest,src)

This function copies the string pointed by 'src' (including the null character)to the destination. Returns the copied string.
Be careful, be sure that the destination memory area is large enough.

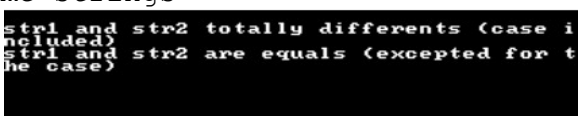
Example : Print :


```
fillscreen(black)
datauc str1[20]={}
str2="Hello World"
strcpy(str1,str2)
printf("\nstr1=%s",str1)
keywait()
```

//strcmp(str1,str2)

Compare the contents of the two 'str1' and 'str2' strings with each other. The 'strcmp ()' function start with the first character in each character string and move on to the next character and so on until a character differs or the end of the character string is reached.
'strcmp ()' returns the following values:
• <0 if 'str1'' is less than 'str2'
• == 0 if 'str1' is the same as 'str2'
•> 0 if 'str1' is greater than 'str2'
Specifically, if the strings differ, the value of the first character that differs from 'str2' minus the corresponding character in the string 'str1' is returned.
C equivalent: strcmp (str1, str2)

Example : Compare some strings




```
fillscreen(black)
str1="Hello"
str2="hello"

if(strcmp(str1,str2)!=0) then              //take care of case
printf("\nstr1 and str2 totally differents (case included)")
endif

if(strcasecmp(str1,str2)==0) then          //indifferent to case
printf("\nstr1 and str2 are equals (excepted for the case)")
endif

keywait()
```

//strcasecmp(str1,str2)

The same than the 'strcmp' but don't takes care of the lower or upper

case.
Compare the contents of the two 'str1' and 'str2' strings with each other. The 'strcmp ()' function start with the first character in each character string and move on to the next character and so on until a character differs or the end of the character string is reached.
'strcmp ()' returns the following values:
• <0 if 'str1'' is less than 'str2'
• == 0 if 'str1' is the same as 'str2'
•> 0 if 'str1' is greater than 'str2'
Specifically, if the strings differ, the value of the first character that differs from 'str2' minus the corresponding character in the string 'str1' is returned.
C equivalent: strcasecmp (str1, str2)

Example : See 'strcmp()' function.

//gets(input_str_length_max)

Request for a message (string) to be entered on the keyboard. Its length can't be superior to 'input_str_length_max' (must be inferior to 99 too). Affect the current pen position ('settextpos(), etc...)'. Use with 'settxtcolor()' and 'setfont()' to set text features.
You case lock the upper case with the 'SHIFT' key.

Example : Request for a string and display it in a box.


```
fillscreen(black)
settxtcolor(0b1111100000000000) //red
setfont(2)
printf("Enter a string :")
rtn=gets(20)
msgbox("Your answer",rtn)
settxtpos(0,200)
printf("%s",rtn)
keywait()
```

//atoi(num_str) or atol(num_str) (both are equals)

Converts and return the value written in the string 'num_str'.
Before using this function, you can check if it is a correct numeric string with the 'isnumstr()' function.

Example : Request your age, check if correct numeric str, and display it

```
datauc buf[60]={0}

lbl lblage
request1str("Enter your Age","Your age","20",(ad*)str)
```

```
if(not isnumstr(str)):goto lblage

age=atoi(str)
sprintf(buf,"Your age is %d",age)

msgbox("Message",buf)
```

//fopen(filename_str,mod_str)

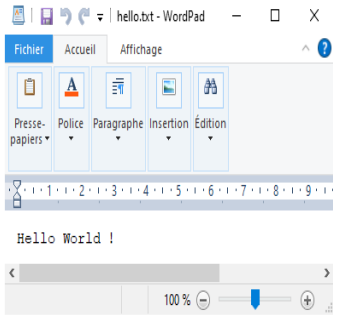
Like the C function.
Open/create a file named 'filename_str'.
The opening mod is 'mod_str' (see below).
Returns 0 if an error occured.

The string of the filename 'filename_str' must ending with ".tns" characters.
The string of the file name must begin with the directory. For current directory, write the "./" at the beginning.
For example, a valid file name str : "./file.tns".

The opening mod string can be :
"r"
Searches file. Opens the file for reading only. If the file is opened successfully fopen() loads it into memory and sets up a pointer that points to the first character in it. If the file cannot be opened fopen() returns NULL.
"w"
Searches file. If the file exists already, its contents are overwritten. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open the file. It creates a new file for writing only(no reading).
"a"
Searches file. If the file is opened successfully fopen() loads it into memory and sets up a pointer that points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open the file. The file is opened only for appending(writing at the end of the file).
"r+"
Searches file. Opens the file for both reading and writing. If opened successfully, fopen() loads it into memory and sets up a pointer that points to the first character in it. Returns NULL, if unable to open the file.
"w+"
Searches file. If the file exists, its contents are overwritten. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open the file. The difference between w and w+ is that we can also read the file created using w+.
"a+"
Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. If the file doesn’t exist, a new file is created. Returns NULL, if unable to open the file. The file is opened for reading and appending(writing at the end of the file).
"rb"
Open the binary file in read mode. If the file does not exist, the open() function returns NULL.
"wb"
Open the binary file in write mode. As the pointer is set to the start of the file, the contents are overwritten. If the file does not exist, a new

file is created.
"ab"
Open the binary file in append mode. The file pointer is set after the last character in the file. A new file is created if no file exists with the name.
"rb+"
Open the binary file in read and write mode. If the file does not exist, the open() function returns NULL.
"wb+"
Open the binary file in read and write mode. Contents are overwritten if the file exists. It will be created if the file does not exist.
"ab+"
Open the binary file in read and append mode. A file will be created if the file does not exist.

Example : Create a notepad file named 'hello.txt.tns'. For information, you can open it with notepad (be sure to have deleted the '.tns' chars) :




```
fillscreen(black)

void datauc str[1]          //create a pointer of type unsigned char*

str="Hello World !"         //The string to put in the output file

f=0
f=fopen("./hello.txt.tns","w")   //Creates the file
if(f) then
printf("\nopened")
for(i=0,i<strlen(str),i++)
fputc(str[i],f)                  //puts each chars
endfor
fclose(f)                        //close the file
printf("\nFile 'hello' created")
else
printf("\nError")
keywait()
endif
```

//fclose(fiden)

Closed an opened file (with 'fopen()'). If successful, returns 0.
Otherwise, returns EOF value (constant 'EOF').
Like the C function.

Example : see 'fopen()' function for one example

//fputc(char_value,fiden)

Puts the char value 'char_value' inside the file pointed by 'fiden' (see 'fopen()' function).
If an error occured, returns EOF value (constant). If no error, return 'char_value'.

Example : See the 'fopen()' section

//ftell(f_iden)

Returns the current position in file pointed by 'f_iden'. To be used with 'fseek()' function.
Like the C function.

//fseek(pointer,offset,position)

'fseek()' is used to move the file pointer associated with a given file to a specific position.
Returns the current position index of the file identified by 'f_iden' (see 'fopen()'). It is usefull for use with 'fseek()' function.
It returns zero if successful, or else it returns a non-zero value.

See 'ftell()' to retrieves current position.

Like the C function.

Arguments :
pointer: It is the pointer to a FILE object that identifies the stream.
offset: It is the number of bytes to offset from the position
position: It is the position from where the offset is added. Position defines the point with respect to which the file pointer needs to be moved. It has three values:
'seekend' constant: It denotes the end of the file.
'seekset' constant: It denotes starting of the file.
'seekcur' constant: It denotes the file pointer’s current position.

//timer functions : timeron() timerset(value) timerread() timeroff()

Newprogcx allow the use of one timer.
The TI-Nspire calculators have 1 hardware timer that allows for accurate timing of any task. The timer runs at 32 kHz. Newprogcx provides a precise use of it, which is as follows :
- Every 0.000030517578125 second (roughly 30 µs, that is 32 768 times per second), timer's value is decreased by 1.
- When a timer reaches 0, it stops decreasing until you reload a value in it.
To sum up, when a timer reads 0, it means the amount of time you passed has run out.

Note : the constant 'onesec' is equal to 32768

//timeron()

Start the use of the timer. Don't forget to close it with 'timeroff()' before the end of your program.

//timerset(value)

Set the value 'value' of the timer.

//timerread()

Load the timer value.

//timeroff()

Close the timer. You have to call this function before ending your program.

Example : Wait for 5 seconds

```
fillscreen(black)
timeron()

printf("\nWait 5sec. Ready (press CTRL) ?")
while(not keytest(kctrl))
endwhile

timerset(onesec*5)
printf("...Launched")

while(not keytest(kesc))
t=timerread()
if(t==0):break
endwhile
timeroff()

printf("\nFini")
keywait()
```

//abs(number)

Returns the absolute value of 'number' (ie positive).

//delay(nb_msec)

Do nothing for 'nb_msec' mili second.

//repeat(nb,instruction)

Repeat nb times the instruction 'instruction'. 'nb' must not be Null.

//malloc(nb_bytes)

Similar to the C function. Allocate dynamicaly a memory area of 'nb_bytes' bytes.
Don't forget to free the memory area with 'free()' for avoiding memory lose.

Example : Copy a string and display it

```
func copystr(str)
local(newstr)
newstr=malloc(strlen(str)+1)
if(newstr==0):finish()                //memory missing
memmove(newstr,str,strlen(str)+1)     //copy the content
```

```
return newstr
endfunc

fillscreen(black)
str2=copystr("Hello World")
printf("%s",str2)
keywait()

free(str2)      //free the memory area
```

//realloc(ptr,newsize)

Attempts to resize the memory block pointed by 'ptr' that was previously allocated with 'malloc()' to the new size 'newsize' (in bytes). Returns the eventually new pointer. The content is saved.
Use the 'free()' function before ending the program.

Example : Print : "str1=Hello World"

```
fillscreen(black)
void datauc str1[2]
str1=malloc(20)            //20 is enough
strcpy(str1,"Hello ")
str1=realloc(str1,30)      //new size of 30 bytes
str2="World"
strcat(str1,str2)
printf("\nstr1=%s",str1)
keywait()
```

//free(mem_area_ptr)

Free the memory area pointed by 'mem_area_ptr' previously allocated with 'malloc()' or 'newsprite()' etc... functions.
Don't forget to use it before the end of your program for avoiding memory loses.

//finish()

Ending the program (return to os). Be sure to have freed all memory blocks that you have eventualy created (with 'free()' function).

//rand(top_value)

Returns a random value from 0 to 'top_value'-1.
The generated numbers are all the sames. To avoid this, you have to use a trick as below :
Example : How to generate reals random numbers by using timer and a msgbox (asynchronus)

```
timeron()       //start timer
timerset(onesec*1000) //set the timer of a number big enough
//Below : the random timer value is due to the time to hit a key
if(msgbox("Welcome to minesweeper","Start a new game")==0):end()
v=timerread()   //so the readed timer value is random
timeroff()
```

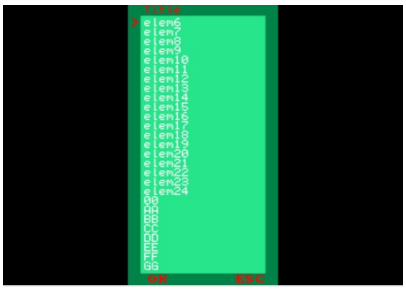
```
for(i=0,i<v modulo 1000),i++)
rand(20)        //execute 'v' timer value the 'rand()' function (random)
endfor
```

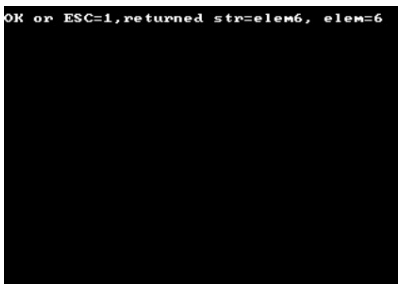
//menu(title_str,elems_list_str,(ad*)rtn_str,(ad*)index)

Display a scrolling menu, which title is 'title_str', which elements are defined in 'elems_list_str' (see below for the parsing).
Use arrows keys Down or Up to select an element. The 'CTRL' key can be toggled to accelerated jump (10 by 10). You can hit the keyboard with 'a' to 'z' ('0' to '9' also) for fast travel in the menu (in this case, it can be useful to have elements list in alphabetical order).
The function returns 1 if Enter key had been pressed, 0 if Esc key. Also, it return the element string pointer in NewprogCX var 'rtn_str' (valid until next time) and the index in 'index' NewprogCX var (the first element has index = 1).
The dimensions of the windows can change (according to its content).
If elements strings are too long to be displayed, they will be truncated.
To the popup, the cursor points on the initial value of index.

Elements string list format (parsing) :
Each element must be separated with the « // ».
For example : "elem1//elem2//elem3//elem4//elem5"

Example : Display a menu and display the answer






```
fillscreen(black)          //fill screen in black

str="elem1//elem2//elem3//elem4//elem5//elem6//elem7//elem8//elem9//elem1
0//elem11//elem12//elem13//elem14//elem15//elem16//elem17//elem18//elem19
//elem20//elem21//elem22//elem23//elem24//00//AA//BB//CC//DD//EE//FF//GG/
/HH//END"

ans=menu("Title",str,(ad*)rtnstr,(ad*)index)
printf("\nOK or ESC=%d,returned str=%s, elem=%d",ans,rtnstr,index)
keywait()
```

Limitations :

Compilator :
Instruction maximum lenght is about 500 bytes.
A direct string can't be longer than 3999 chars.

A variable or a function ('func') should not start by a NewprogCX function name or keyword. In that case, the error message will appears at compilation : 'Error line X:'Func funcname(...)' expected'

Example :

```
func func1
...
endfunc
```

All NewprogCX constants (except keys codes) :

```
==>Text fonts
    {"font6x8",0},
    {"font8x8",1},
    {"font8x12",2},
    {"font16x16",3},

==>Timers functions
    {"onesec",32768},

==>Screen attributes
    {"lcdnbpix",76800},
    {"lcdnbbytes",153600},
    {"lcdsize",153600},
    {"lcdwidth",320},
    {"lcdheight",240},

==>Colors
    {"white",-1},
    {"black",0},
    {"red",0b1111100000000000},
    {"blue",0b0000000000011111},
    {"green",0b0000011111100000},
    {"yellow",0b1111111111100000},
    {"purple",0b1111100000011111}

==>Files functions (fopen(),...)
    {"eof",EOF},
    {"seekset",SEEK_SET},
    {"seekcur",SEEK_CUR},
    {"seekend",SEEK_END},

==data types
    {"uctype",1},     //unsigned char
    {"sctype",11},    //signed char
    {"ustype",2},     //unsigned short
    {"sstype",22},    //signed short
    {"uitype",4},     //unsigned int
    {"sitype",44},    //signed int

==>Formated text (printf(),fputc()...)
    {"newline",'\n'}
```