NEWPROG2 & NPPTOC

Table des matières

A-NEWPROG, c'est quoi ? Et NPPTOC ?

Newprog est un langage de programmation on-calc qui étend grandement les possibilités du Ti-basic et offre de grandes vitesses d'exécutions (30 à 3000 fois pus rapide que le Tibasic). Les fonctionnalités des instructions de Newprog permettent :

- Une édition des programmes directement dans l'éditeur de programme Tibasic classique

NEWPROG2 & NPPTOC

- Une syntaxe proche du Tibasic
- Un accès et une manipulation de bas niveau à la mémoire et au système de fichiers du TIOS
- L'utilisation de fonctions graphiques dérivées de l'assembleur
- Un accès de haut niveau et de bas niveau aux claviers
- L'utilisation de timers et interruptions
- D'entrelacer du code Tibasic avec du code Newprog
- La création et l'utilisation de librairies (étendant ainsi les capacités vers celles du C)

Les fichiers sources sont compilables sous deux formats :

1. En **bytecode** (comme en python ou java) : nécessite l'utilisation d'une machine virtuelle pour être exécutés
2. Depuis Newprog2 et l'apparition associée de **NPPTOC** : En **langage machine** (fichiers ASM – 89z) : directement exécutables

>> Une compilation préalable en bytecode est nécessaire

Note : Voir les sections correspondantes pour plus de détails

Nouveautés avec la sortie de Newprog 2 :

1. Réécriture de la documentation. Elle devient plus explicite et rationalisée.
2. Correction de bugs.
3. Modifications légères de la syntaxe de certaines fonctions.

   Mais surtout :

4. Possibilité de créer et d'utiliser ses propres librairies ASM (codes sources en Newprog ou en C). Ainsi, étend les possibilités à toutes les instructions du C.
5. Possibilité de convertir un fichier Newprog bytecode en code machine, accroissant encore significativement la rapidité d'exécution du programme. Il faudra pour cela passer par une étape intermédiaire qui consiste à traduire un fichier en bytecode Newprog en langage C directement (et seulement) sur la calculatrice avec NPPTOC puis de le compiler avec un compilateur C.

   Les compilateurs envisageables sont :

   - GTC version installé directement sur la calculatrice (oncalc)
   - GCC4TI pour PC

   ◦ Chaque compilateur a ses avantages et inconvénients :

   ◦ **GTC version oncalc :**

   Avantage :

   ◦ Compilation à 100% sur la calculatrice : Pas besoin de transferts de TI vers PC et PC

vers TI.

Inconvénients :

- Fichiers exécutables moins optimisés donc plus lents et plus volumineux
- Compilation très gourmande en mémoire RAM. De ce fait, seuls des programmes de tailles modérées pourront aller aux termes de leurs compilations.
- Utilisation conjointe d'une librairie externe ('npplib2.89z') pour certaines fonctions (graphiques principalement) → son chargement prend du temps et requiert un peu de mémoire RAM
- **GCC4TI pour une compilation sur ordinateur :**

  Avantages :

- Fichiers exécutables (code machine) optimisés en taille et rapidité.
- Peut compiler des fichiers sources volumineux
- Pas besoin de la librairie externe 'npplib2.89z' donc exécution au lancement plus rapide et demande moins de mémoire RAM

  Inconvénients :

- Nécessité de transferts des fichiers sources C de TI vers PC
- Puis demande quelques manipulations dans l'explorateur de fichiers de Windows :
  - Convertir les fichiers sources .89t en fichiers .C et éventuellement en fichiers .h
  - Transformer la premier ligne de 'cout.C' en commentaire (ajout de '//')
  - Déplacer ces fichiers dans les répertoires adéquats de GCC4TI
- Puis de créer un projet C dans l'IDE de GCC4TI et d'y insérer les fichiers convertis dans l'étape juste au dessus (cela fait, la compilation peut être lancée).
- Puis de transférer l'exécutable du PC vers la TI

B-Utilisation de NEWPROG

1) Structure d'un programme NEWPROG

Le code source d'un programme Newprog a la particularité d'être écrit directement dans l'éditeur de programme des programmes Tibasic classiques : APPS + Program Editor

La syntaxe Newprog est donc proche de la structure d'un programme Tibasic.

L'instruction 'init()' doît être placée en première ligne. Elle caractérise un programme Newprog.

Une particularité du langage Newprog est qu'elle permet d'entrelacer des séquences de codes

NEWPROG2 & NPPTOC

Newprog avec des séquences Tibasic classiques. A l'exécution (voir section correspondante), la machine virtuelle, ou interpréteur, en fera la distinction et interprétera les instructions dans le langage adéquat. C'est très utile car elle permet d'offrir la très grande majorité des possibilités du Tibasic, en plus des possibilités des instructions Newprog. Pour spécifier une séquence Tibasic, il suffit de placer ses instructions entre les balises 'basic' et 'endbasic'.

Exemple de structure de programme Newprog :

```
:init()              //obligatoire. Correspond à l'initialisation de l'interpréteur (ou machine virtuelle).
:newprog_instruction1    //première instruction Newprog
:newprog_instruction2    //deuxième instruction Newprog
:basic                   //balise spécifiant le début d'une séquence Tibasic
:tibasic_instruction1    //première instruction Tibasic
:tibasic_instruction2    //deuxième instruction Tibasic
:endbasic                //balise spécifiant la fin d'une séquence Tibasic
:newprog_instruction3    //troisième instruction Newprog
```

2) Compilation NEWPROG en bytecode et exécution

Pour compiler et exécuter un programme source Newprog, la méthode conseillée est d'exécuter dans le HOME screen la commande suivante :
npcc(" myprgm ") où 'myprgm' est le nom du votre programme source.

Pour information, le programme 'npcc.89p' procède en interne aux étapes suivantes :

1. **Vérification syntaxique** (comme dans tout lancement d'un programme Tibasic). Revient à exécuter 'myprgm()' et à taper 'esc' quand un message d'erreur apparaît.
2. **Archivage du programme source** en vue d'un éventuel crash de la calculatrice
3. **Copie du programme source** dans le répertoire 'zbackup'
4. **Désarchivage de fichier 'out.NPP'** en vue d'être écrasé à l'étape suivante
5. **Conversion en bytecode Newprog** : création du fichier 'out.NPP'. Cette étape procède à une pré vérification de la structure du programme. Si une erreur apparaît, la conversion sera abordée. Cela revient à exécuter 'newprogc(" myprgm ")'.
6. **Affichage d'avertissements éventuels** : Aiguille l'utilisateur sur d'éventuelles erreurs (principalement des oublis secondaires)
7. Affichage d'un message invitant à **tester le programme**.
8. **Exécution du programme** dans la machine virtuelle (revient à exécuter 'newprog("out")')
9. Une fois l'exécution du programme terminé, **demande à désarchiver le programme source** (sélectionner non si la calculatrice semble instable)

L'utilisateur peut renommer le fichier 'out.NPP' générer par le nom qu'il souhaite. Pour exécuter ce programme (.NPP) dans la machine virtuelle (appelée aussi interpréteur), comme vue plus haut, il faut exécuter 'newprog("mynpp.NPP")' où 'mynpp' est le nouveau nom donné.

NEWPROG2 & NPPTOC

3) Translation en C avec NPPTOC, compilation en code machine et exécution

Translation en langage C du fichier Newprog 'out.NPP' :

A l'exécution dans le HOME screen de la commande 'npptoc()', le fichier 'out.NPP' sera converti en langage C. Le fichier de sortie 'cout.TEXT' (fichier C principal) et éventuellement d'autres fichiers (.TEXT) seront générés (fichiers headers).

Afin d'éviter l'affichage d'une erreur précisant un manque de mémoire pendant la translation, veuillez libérer un maximum de mémoire RAM, soit en archivant soit en supprimant des fichiers. Si cela ne suffit pas, il sera nécessaire de reseter votre calculatrice et de relancer le processus.

La translation est terminée.

Compilation vers fichier .ASM avec GTC oncalc et exécution :

Vous devez avoir installer préalablement GTC sur votre calculatrice.
Pour compiler les fichiers sources C générés préalablement, suivez les étapes suivantes :

1. Désarchiver le fichier 'outbin.ASM' si il y a
2. Lancer 'gtc_ide()'
3. Sélectionner le fichier 'main\cout'
4. Appuyer sur F5 pour lancer la compilation
5. Le fichier exécutable de sortie est 'outbin.ASM'. Il peut être testé.

Si la compilation a affiché une erreur de mémoire, archivé tous vos fichiers et relancer la compilation. Si l'erreur mémoire est toujours présente, reseter votre calculatrice. Si l'erreur est toujours là, vous serez obligé de compiler sur PC.

Remarque : l'exécution de 'make("ouput_name")' automatisera ces étapes et renommera le fichier de sortie en "output_name".

Compilation vers fichier .ASM avec GCC4TI sur PC et exécution :

1. Transférer les fichiers C (*.TEXT') générés précédemment vers votre PC.
2. Renommer le fichier 'cout.89t' en 'cout.c' et les autres fichiers headers en '*.h' (exemple 'execbash.h')
3. Ouvrer ces fichiers un à un avec un éditeur de texte (Notepad par exemple) et supprimer les caractères inutiles en début de fichier et en fin de fichier
4. Dans le fichier 'cout.c', mettre en commentaire la première ligne en ajoutant '//' en début de ligne : // #define GTC_ONCALC //undo this line if with GCC4TI
5. Déplacer/insérer ces fichiers vers vos répertoires de GCC4TI
6. Lancer la compilation (avoir créer un projet GCC4TI au préalable)
7. Le fichier 89z est généré et peut être testé

NEWPROG2 & NPPTOC

C-Variables, espaces de données, typage en Newprog et cas des listes non instanciées

1) Variables

Les variables en Newprog s'écrivent de façon similaire aux variables classiques en Tibasic et en C. Elles sont écrites au minimum avec deux caractères et au maximum avec huit caractères. Exemple :

- valeur ou var1 ou vv ou xx ou index

Elles ont toutes une taille codé sur 4 octets, leurs valeurs peuvent être comprises entre :

- -2 147 483 648 à + 2 147 483 647

Les variables, suivant leur types (nombre ou pointeurs, voir le paragraphe A-3), peuvent aussi servir à manipuler des espaces de données (voir le paragraphe A-2)).

2) Espaces de données

Comme en Tibasic ou en C, un *espace de données* est un espace mémoire composé de données de même type, stockées de manière contiguë en mémoire (les unes à la suite des autres). Le pointeur de cet espace mémoire est l'adresse de la première donnée. Cette valeur est contenue dans une variable. On pourra manipuler l'espace de données grâce à cette variable.

Un espace de données est donc une suite de cases (espace mémoire) de même taille. La taille de chacune des cases est conditionnée par le type de donnée que l'espace de données contient (voir plus bas).

Les éléments du tableau peuvent être :

- Si le type de la variable est 1 : chaque donnée est codée sur 1 octet
- Si le type de la variable est 2 : chaque donnée est codée sur 2 octets
- Si le type de la variable est 4 : chaque donnée est codée sur 4 octets
- Si le type de la variable est autre : chaque donnée est une suite de X octets, où X est le nombre d'octets

Exemple :

Adresses	A + type*0	A + type*1	A + type*X	A+ type*(N-1)
Élément n°	1	2	X+1	N

Cet espace mémoire contient N données de type 'type' et est pointé par A.

Ils peuvent être fixes ou alloués dynamiquement, mais leur structures suivent les mêmes principes, seules leurs moyens de constructions sont différents. Dans le cas d'une allocation dynamique, il faudra veiller à les supprimer de la mémoire RAM avant la fin de l'exécution du programme sous peine de perdre de la quantité de mémoire disponible (si oubli, il faudra reseter la mémoire RAM pour la récupérer).

On peut par exemple y stocker une chaîne de caractères pour y stocker du texte, des informations comme des nombres de différentes longueurs (l'âge d'une personne, etc..). Ils sont communs à la très grande majorité des langages de programmation. Ces espaces de données conservent les propriétés de ceux du langage C.

Pour lire manuellement les éléments d'un espace de données, voir la fonction '[]' (comme en Tibasic ou C) (ou accessoirement 'l-w-lb()' 'peekb-w-l()' etc...).

Pour modifier manuellement les éléments d'un espace de données, voir la fonction '→' (comme en Tibasic ou C) ou 'sto()' (ou 'wb-w-l()' 'pokeb-w-l()' etc...).

Type liste :

Une liste est un espace de données servant à sauvegarder des nombres de tous types.

Définition statique de listes :

On définit chaque éléments de la liste avec la notation 'b-w-l:...:y'. On ne peut définir statiquement une liste avec les types 1, 2 ou 4 :

- b-y pour une liste de type 1
- w-y pour une liste de type 2
- l-y pour une liste de type 4

Exemple :

b(nom_liste)

valeur élément1

valeur élément2

….

valeur élémentN

y

L'équivalent C est :

char nom_liste[]={valeur élément1,valeur élément2,...,valeur élémentN} ;

Afin de soulager le programmeur, quand une série de 'nombre d'occurence' éléments de même valeur 'valeur' est souhaité, on peut utiliser le mot clé 'repeat(nombre_d'occurence,valeur)'.
Exemple :

w(nom_liste)

valeur élément1

repeat(5,0)

valeur élément7

y

L'équivalent C est :

int nom_liste[]={valeur élément1,0,0,0,0,0,valeur élément7} ;

Cette notation est très utile pour définir des sprites.

Par exemple, pour définir un sprite qui représente un carré de coté 8x8 de la façon suivante (valeur définies en binaire) :

b(nom_sprite)

0b11111111

0b10000001

0b10000001

0b10000001

0b10000001

0b10000001

0b10000001

0b11111111

y

Ici, on pourrait remplacer les six lignes '0b10000001' par 'repeat(6,0b10000001)'.

Définition dynamique de listes :

a) Il est possible de créer dynamiquement des listes en utilisant la notation :

'{list_name,elems_size,elem1,elem2,...,elemN}'

A l'exécution, un espace mémoire de elems_size*N octets sera créé et chacun de ses éléments (tous de taille 'elems_size' (seulement 1,2,4 autorisées) sera affecté à la valeur 'elemX'.

Veiller à libérer l'espace mémoire avant la fin de l'exécution du programme avec la fonction 'free()'.

Pour des informations complémentaires, voir la section correspondante.

Exemple :

Le code Newprog '{list,4,12,23}' revient au code C suivant :

char *list ;

if(!list=malloc(4*2)) return;

list[0]=12 ;

list[1]=23 :

Ce code C peut également être écrit avec son analogue Newprog.

b) On peut également utiliser la fonction 'group()' pour définir une liste de façon dynamique. Voir section correspondante.

Type chaîne de caractères ou string :

Une chaîne de caractères (ou string) est un espace de données codées sur 1 octet (type 1) et qui sert à manipuler des phrases (en vue par exemple de les afficher dans le programme).
Ils sont définis de la manière suivante :
" Ceci est une chaîne de caractères " : Il faut placer sont contenu entre chaque ' " ' (Idem qu'en Tibasic). Pour pouvoir la manipuler, on enregistre son pointeur, c'est à dire l'adresse du premier caractère, dans une variable par exemple grâce à la fonction '→' ou 'sto()'.

De manière plus complexe, on peut aussi la définir comme n'importe quelle liste, où il faudra affecté chaque élément avec le code ASCII de la lettre correspondante. Le dernier élément devra être égal à 0 pour pouvoir manipuler cette chaîne de caractères avec les fonctions qui vont bien ('printf-1-2()','strlen()','strcat()', etc...).

Exemple de code Newprog :

```
:init()

: "Hello"→str //enregistre le pointeur sur l'espace de données de la chaîne de caractères dans la variable nommée str. Son type est de 1 (voir A-3) ).

:text str                    //On affiche alors Hello à l'écran par exemple
```

Si on représente sa structure en mémoire, elle sera comme suit :

Adresses	A+0	A+1	A+2	A+3	A+4	A+5	A+6	A+7
Codes ASCII	66	111	110	106	111	117	114	0 (obligatoire)
Caractères	B	o	n	j	o	u	r	\0

La valeur de str sera égale à A+0 soit A. A chaque caractère correspond un code ASCII (idem qu'en Tibasic et C).

Pour lire n'importe quelle valeur à sa position correspondante dans cet espace, il est commode d'utiliser la notation '[ ]' (d'autres fonctions comme 'lb()' ou 'peekb()' existent). Je ne vais pas m'étendre car c'est comme en Tibasic et en C.

Remarque : Les variables Tibasic sont aussi un genre d'espace de données. Le contenu de cet espace défini la variable elle même. Un espace de données peut être nécessaire pour créer des variables Tibasic de tous types. Voir fonction 'fopen()' et 'fcreate()'.

3) Typage des variables

Chaque variable est associée à un type. Le typage par une valeur est propre à Newprog. La valeur du type peut aller de 1 à 65535. Ces valeurs de types, hormis le type 5, correspondent à des pointeurs sur données. Le type 5 est le type par défaut, correspondant aux entiers signés type 'long' en C.

Le type peut être défini manuellement avec la fonction 'settype()' ou de manière automatique lors de

NEWPROG2 & NPPTOC

l'utilisation de certaines fonctions (voir ci-dessous).

Les différents types possibles :

- Le type 1 correspond à un pointeur sur des données codées sur 1 octets. L'équivalent C est 'char *',
- Le type 2 correspond à un pointeur sur des données codées sur 2 octets. L'équivalent C est 'int *'. Comme en C, ce pointeur doit être paire,
- Le type 4 correspond à un pointeur sur des données codées sur 4 octets. L'équivalent C est 'long *'. Comme en C, ce pointeur doit être paire,
- Le type 5 correspond à une variable entière signée de valeur standard, comme n'importe quel nombre. Il est dédié aux calculs arithmétiques classiques + - * /. L'équivalent C est 'long',
- Les autres types (hormis 5), sont des pointeurs sur des données de longueurs personnalisées, propre à Newprog et n'ayant pas d'équivalent en C. Leur équivalent C restent tout de même des 'char *'. Ils ont été imaginés pour simplifiés certaines tâches. Ils peuvent être utiles avec les fonctions 'files()' et 'reps()' mais pour d'autres besoins du programmeur (voir fonction 'sto()').

Une variable ne peut avoir qu'un et un seul type dans tout le corps du programme (depuis Newprog 2 pour assurer une bonne conversion en langage C avec NPPTOC).

Les variables et leurs types respectifs sont déclarés à leur première apparition dans le corps du programme. On ne peut affecter plus d'un type à une variable (un message d'erreur apparaîtra).

Affectation automatique des types :

1. Première apparition comme variable de destination (nommée dest) dans les fonctions et instructions suivantes affecteront directement dest au type 1 :,
    - newstr(X,dest)
    - seqb(dest,X,X,X,X,X)
    - "X"→dest
    - string(X)→dest
    - char(X)→dest
    - fopen(X)→dest
2. Première apparition comme variable de destination (nommée dest) dans les fonctions et instructions suivantes affecteront directement dest au type 2 :
    - seqw(dest,X,X,X,X,X)
3. Première apparition comme variable de destination (nommée dest) dans les fonctions et instructions suivantes affecteront directement dest au type 4 :
    - seql(dest,X,X,X,X,X)
    - group(nilist)→dest

4. Première apparition comme variable de destination (nommée dest) dans les fonctions et instructions suivantes affecteront directement dest au type 10 :
    - reps(dest)
    - files(X,dest)

Exemples et cas d'usages ; codes Newprog relatif au typage des variables :

Exemple 1 :

```
:init()
:3→num              //num est du type par défaut, c'est à dire 5
:settype(num,5)     //Inutile, mais pas d'erreur car le type est déjà de 5
:num*2→num2         //num2 est du type 5 par défaut
```

Exemple 2 :

```
:init()
:seqb(listb,va,1,4,1,va)        //Le type de listb est fixé à 1
:listb[3]+1→vb                  //vb est de type 5
:listb+1→vc                     //vc est de type 5. En C, le type de vc serait de type 'char *'. La logique ferait qu'il en soit de même en Newprog. Newprog ne le fait pas, il faut donc fixer le type de vc avant cette ligne. On écrira alors :
:init()
:seqb(listb,va,1,4,1,va)        //Le type de listb est fixé à 1
:listb[3]+1→vb                  //vb est de type 5
:settype(vc,1)                  //on fixe le type de vc avant sa première utilisation
:listb+1→vc                     //vc est toujours de type 1
```

Exemple 3 :

```
:init()
:3→num              //num est du type par défaut, c'est à dire 5
:settype(num,4)     //interdit car le type est déjà  de 5
```

Exemple 4 :

```
:init()
:3→num                          //num est du type par défaut, c'est à dire 5
```

:seqb(num,X,X,X,X) //interdit car le type de num est déjà de 5

Exemple 5 :

:init()

:char(100)→str //Le type de str est 1

Hors sujet, mais pour information, :str[0] donnerait 100.

4) Listes non instanciées

Le langage Newprog offre une possibilité unique qui est un hybride entre le Tibasic et le langage C, ce sont les listes non instanciées (niliste).

Elles s'écrivent de la manière suivante : {instruction1[,instruction2…]} où les instructions entre '[]' sont optionnels. Cette écriture peut être confondante avec les listes instanciées dynamique (voir section correspondante) dans le cas où instruction1 est une variable seule. Afin que le compilateur ne la confonde pas avec une liste instanciée dynamique, une astuce est de le tromper en réécrivant l'instruction1 originale nommée pour l'exemple 'instuc1' de la façon suivante : 'instuc1'+0

Elles ont une utilité au cas par cas suivant les fonctions Newprog qui les utlisent (pour cela, voir leurs sections correspondantes). Ces fonctions sont les suivantes :

- 'toos()'
- 'when()'
- 'inter()'
- 'group()'
- 'libfuncs()'
- 'testfc()'

NEWPROG2 & NPPTOC

D-Liste des fonctions

1-Fonctions mémoire

Fonction : *valeur→destination* ou *sto(destination,valeur)*

Affecte à destination la valeur valeur. Différentes écritures sont possibles :

1) Destination peut être une variable.Value et var peuvent être de tout type. Affecte la valeur value à la variable nommée 'var' :

:value→var

Code C équivalent :

var=value ;

Pour les cas ci-dessous, il est conseillé de vous référer à la fonction settype() pour apréhender la notion de type de variable.

2) Destination peut être une l'élément d'une liste de type 1 ou 2 ou 4 (voire 5, revient à un type 4). Affecte la valeur value à l'élément d'index index de la liste 'list' :

:value→list[i]

Le code C équivalent serait :

Si type est 1 :

```
long sto_in_list(char *list , unsinged int index  ,char value)
{
        list[index]=value;
}
void _main(void)
{
        char list[size];
        sto_in_list(list,index,value) ;
}
```

Si type est 2, remplacer 'char list[size]; 'par 'int list[size];'

Si type est 4 (voire 5), remplacer 'char list[size];' par 'long list[size];'

3) Destination peut être une liste d'un autre type (différent de 1 2 4/5). Dans ce cas, value sera considérée comme un pointeur de type 1 et son contenu à hauteur de type octets sera copié vers le pointeur correspondant à list[elem] (en C, correspond à 'list+elem*type').

NEWPROG2 & NPPTOC

```
:init()
:b(buff1)                    //buff1 est un espace mémoire composé de 20 fois 0. Il est de type 1
:repeat(20,0)
:y
:settype(buff2,15)           //comme en Newprog2, on ne peut pas modifier le type d'un pointeur, alors on créé buff2 avec un type égal à une taille arbitraire de 15 (inférieure à 20)
:buff1→buff2                 //buff2 pointe sur le même espace mémoire que buff1
:" Hello World ! " →buff2[0]  //copie "  Hello World ! " à l'adresse buff2[0] soit buff+0 x 15
:prints(buff2[0])                  //affiche : " Hello World ! "
:keywait()
```

Le code en C équivalent serait :

```
void _main(void)
{
        char buff1[20],*buff2;
        memset(buff1,0,20) ;
        buff2=buff1 ;
        memmove(buff2+0*15, " Hello World ! ",15) ;
        printf(" %s ",buff2 + 0*15) ;
        ngetchx() ;
}
```

4) Destination est la fonction 'osvar(os_var)' comme dans : value →osvar(os_var). Ici, la valeur value sera enregistrée dans la variable tibasic nommée os_var. Veillez à ce que os_var ne soit pas archivée ou quelle ne soit pas une variable système. Des informations complémentaires sont données dans la section Fonctions Tibasic.

- Valeur classique : Enregistre la valeur 666 dans la variable tibasic xx

  ```
  :666→osvar(xx)
  ```

- Valeur étant une string directe (sans #) : Affecte la variable tibasic xx avec la string tibasic " Hello World ! "

  ```
  :" Hello World ! "→osvar(xx)
  ```

- Valeur étant un pointeur de string en utilisant # : Affecte la variable tibasic xx avec la string tibasic " Hello World ! "

  ```
  :" Hello World ! "→str          //str est le pointeur de string
  :#str→osvar(xx)
  ```

  Ne fonctionnant pas avec NPPTOC :

- #value avec value étant une référence sur liste non instanciée:.Affecte une variable tibasic avec le contenue de la référence d'une liste non instanciée (étant value). Exemple, enregistre la liste tibasic {1, "hello ",3} dans la variable tibasic xx

  ```
  :{1,  "hello ",3}→nil
  :#nil→osvar(xx)
  ```

NEWPROG2 & NPPTOC

Fonction : *settype(data_size,var)*

Cette fonction est attachée à la notion de type (voir la section correspondante). D'une certaine manière, elle peut faciliter la programmation. A utiliser avec la notation [].

Sélectionne le type de donnée pointée par la variable newprog var (en définissant leur taille data_size exprimée en octets). La donnée d'entrée data_size peut prendre les valeurs suivantes :

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

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

Exemple pour data size ayant une valeur de 'lcdsize' (différent de 1,2,4):

```
:init()
:settype(ptr,lcdsize)
:settype(lcd,lcdsize)

:malloc(lcdsize*2)→ptr        //creates buffer
:if ptr=0 then
:text "mem error"
:finish()
:endif

:getlcd()→lcd
:lcd→ptr[0]                   //copy the screen area in ptr[0] area
:text "Screen saved"

:memset(ptr[1],-1,lcdsize)    //fill ptr[1] area (ie ptr[0]+lcdsize) with 0b11111111
:ptr[1]→lcd[0]                //copy ptr[1] area to screen area
:text "Screen filled"

:ptr[0]→lcd[0]                //copy ptr[0] area to screen area
```

NEWPROG2 & NPPTOC

:text "Screen restored"

:free(ptr)

Fonction : *malloc(*size_in_bytes*)*

Alloue dynamiquement à la demande un espace mémoire de size_in_bytes bytes (octets). La fonction renvoie le pointeur de l'espace mémoire alloué. On doit libérer manuellement l'espace mémoire avec la fonction 'free()' avant la fin du programme pour éviter les pertes de mémoires, pour cela il est indispensable de stocker le pointeur de retour de cette fonction (communément dans une variable). Si vous souhaitez utiliser la notation '[]' pour lire ou écrire dans cet espace mémoire (par le biais d'une variable), le type de l'espace mémoire sera converti à celui du type de la variable de destination dans le cas d'une notation 'malloc(x)→var' (fonction 'settype()').

Equivalent en C : fonction malloc()

Code de la fonction Newprog 'newstr2()' proche de la fonction 'newstr()' :

```
:def(newstr2(str ))
:malloc(strlen(str)+1)→buff
:if not buff:rtn(0)
:strcpy(buff,str)
:rtn(buff)
:enddef(newstr2())
```

Fonction : *realloc(ptr,new_size)*

'realloc()' permet de modifier la taille du bloc mémoire préalablement alloué pointé par ptr (avec malloc(), seqb(),etc..) à la taille new_size. Realloc() retourne un pointeur sur le nouveau bloc mémoire (le bloc ayant pu changer d'emplacement dans la mémoire). Le contenu du bloc mémoire n'est pas perdu pendant le redimensionnement. Retourne 0 et libère le bloc mémoire pointé par ptr si le bloc mémoire n'a pu être ré-alloué. Comme 'malloc()', il faudra libérer manuellement l'espace mémoire avec la fonction 'free()'.

Equivalent en C : fonction realloc()

Fonction : free*(ptr)*

Libère l'espace mémoire pointé par ptr lequel a été précédemment alloué avec des fonctions telles que 'malloc()' ou 'seqb()', 'seqw()', 'seql()', 'group()' par exemple. Si ptr ne pointe pas sur un espace mémoire existant alloué dynamiquement, la fonction plantera la calculatrice.

Equivalent en C : fonction free()

NEWPROG2 & NPPTOC

Fonction : *memset(ptr,value,nb_bytes)*

Affecte nb_bytes octets par la valeur value à partir de l'adresse ptr. Cette fonction est rapide.
Equivalent en C : fonction memset()
Code de la fonction Newprog 'memset2()' équivalent de la fonction 'memset()' :

```
:def(memset2(ptr,value,nb))
:settype(ptr,1)
:for i,0,nb-1,1
:value→ptr[i]
:endfor
:trn(ptr)
:enddef(memset2())
```

Equivalent en C :

```
long memset2(char *ptr,char value,unsigned int nb)
{
        unisgned int i ;
        for(i=0;i<nb;i++) ptr[i]=value ;
        return  ptr ;
}
```

Fonction : *memcpy(dest,src,len)*

Copie un bloc mémoire de taille len octets de l'espace mémoire pointé par src vers l'espace mémoire pointé par dest. Si src chevauche dest, memcpy() peut planter la calculatrice (dans ce cas utiliser la fonction memmove()).
Equivalent en C : fonction memcpy()
Exemple :

```
:init()
:malloc(lcdsize)→ptr //Allow space for lcd saving
:if not ptr:finish()
:memcpy(ptr,getlcd(),lcdsize) //Save the screen in ptr
:text " screen saved "
:memset(getlcd(),255,lcdsize) //Fill the screen with black
:text “The screen is filled”
:memset(getlcd(),ptr,lcdsize)
:text " screen restored "
:free(ptr)
```

NEWPROG2 & NPPTOC

Fonction : *memmove(ptr_dest,ptr_src,len)*

Assez similaire à la fonction 'memcpy()'. Copie un espace mémoire sur len octets de ptr_src vers ptr_dest. En plus de la fonction 'memcpy()', 'memmove()' permet d'avoir les espaces mémoires pointés par ptr_src et ptr_dest qui se chevauchent. Cette fonction retourne ptr_dest.

Equivalent en C : fonction memmove()

Fonction : *memchr(ptr,value,len)*

Syntaxe : memchr(str,c,len) Description :

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

Equivalent en C : fonction memchr()

Exemple : Affiche " 345 "

:init()

: " 12345 "→ptr

:text memchr(ptr,ord(" 3 "),strlen(ptr))

Fonction : *seql(var_dest,var,start,end,step,instruction)*

Attention : Cette fonction a été modifiée depuis la version 2.0.

Alloue dynamiquement un espace mémoire de (end-start+1)*4 octets et affecte dans l'ordre chacun des éléments codés sur quatre octets de cet espace mémoire par la valeur de l'exécution de l'instruction instruction et cela pour une valeur de la variable var allant de start vers end avec un pas de step. Cette fonction retourne le pointeur sur l'espace mémoire alloué et stocke aussi sa valeur dans la variable var_dest Retourne 0 si pas assez de mémoire. Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (avec 'free()').

Si le type de la variable de destination n'a pas encore été défini, alors il sera automatiquement affecté au type 4. Si le type était déjà défini et différent de 4, un message d'erreur apparaîtra à l'exécution ou à la traduction en C avec NPPTOC.

Le code Newprog suivant :

:seql(listl,xx,0,5,1,2*xx)

donnera le contenu de l'espace mémoire pointé par long *listl[]={0,2,4,6,8,10} ;

Code C équivalent :

```
long *listl,xx,i;
listl=malloc(4*(5-0+1)) ;
if(listl)
{
```

```
        i=0 ;
        for(xx=0;xx !=(5+1);xx+=1)
        {
                listl[i]=2*xx ;
                i++ ;
        }
}
```

Exemple d'exécution de code Newprog :

```
:init()
:seql(list,va,1,4,1,va)
:if not list:finish()       //équivalent à:if list=0:finish()
:for vb,0,3,1
:printld(list[vb])
:EndFor
:free(list)
:keywait()
```

Affichera : "1234 "

Fonction : *seqw(var_dest,var,start,end,step,instruction)*

Attention : Cette fonction a été modifiée depuis la version 2.0.

Alloue dynamiquement un espace mémoire de (end-start+1)*2 octets et affecte dans l'ordre chacun des éléments codés sur deux octets de cet espace mémoire par la valeur de l'exécution de l'instruction instruction et cela pour une valeur de la variable var allant de start vers end avec un pas de step. Cette fonction retourne le pointeur sur l'espace mémoire alloué et stocke aussi sa valeur dans la variable var_dest. Cette fonction retourne 0 si l'espace mémoire vient à manquer. Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (avec 'free()').

Si le type de la variable de destination n'a pas encore été défini, alors il sera automatiquement affecté au type 2. Si le type était déjà défini et différent de 2, un message d'erreur apparaîtra à l'exécution ou à la traduction en C avec NPPTOC.

Le code Newprog suivant :

```
:seqw(listw,xx,0,5,1,2*xx)
```

donnera le contenu de l'espace mémoire pointé par int *listw[]={0,2,4,6,8,10} ;

Code C équivalent :

```
int *listw ;
long xx,i;
listw=malloc(2*(5-0+1)) ;
```

```
if(listw)
{
	i=0 ;
	for(xx=0;xx !=(5+1);xx+=1)
	{
		listw[i]=2*xx ;
		i++ ;
	}
}
```

Exemple d'exécution de code Newprog :

```
:init()
:seqw(list,va,1,4,1,va)
:if not list:finish()	//équivalent à:if list=0:finish()
:for vb,0,3,1
:printld(list[vb])
:EndFor
:free(list)
:keywait()
```

Affichera : "1234 "

Fonction : *seqb(var_dest,var,start,end,step,instruction)*

Attention : Cette fonction a été modifiée depuis la version 2.0.

Alloue dynamiquement un espace mémoire de (end-start+1)*1 octets et affecte dans l'ordre chacun des éléments codés sur un octet de cet espace mémoire par la valeur de l'exécution de l'instruction instruction et cela pour une valeur de la variable var allant de start vers end avec un pas de step. Cette fonction retourne le pointeur sur l'espace mémoire alloué et stocke aussi sa valeur dans la variable var_dest. Cette fonction retourne 0 si l'espace mémoire vient à manquer. Il faudra veiller à supprimer cet espace mémoire avant la fin de l'exécution du programme (avec 'free()').

Si le type de la variable de destination n'a pas encore été défini, alors il sera automatiquement affecté au type 1. Si le type était déjà défini et différent de 1, un message d'erreur apparaîtra à l'exécution ou à la traduction en C avec NPPTOC.

Le code Newprog suivant :

```
:seqb(listb,xx,0,5,1,2*xx)
```

donnera le contenu de l'espace mémoire pointé par listb: char *listb[]={0,2,4,6,8,10} ;

Code C équivalent :

```
char *listb ;
```

```
long xx,i;
listb=malloc(2*(5-0+1)) ;
if(listb)
{
        i=0 ;
        for(xx=0;xx !=(5+1);xx+=1)
        {
                listb[i]=2*xx ;
                i++ ;
        }
}
```

Exemple d'exécution de code Newprog :

```
:init()
:seqb(list,va,1,4,1,va)
:if not list:finish()       //équivalent à:if list=0:finish()
:for vb,0,3,1
:printld(list[vb])
:EndFor
:free(list)
:keywait()
```

Affichera : "1234 "

Fonction : *group(nilist)*

Cette fonction transforme la liste non instanciée nilist en liste instanciée avec ses éléments codés sur 4 octets (bytes). Cette fonction retourne un pointeur sur le bloc mémoire créé dynamiquement. Retourne 0 si le bloc mémoire n'a pas pu être alloué. Ce bloc mémoire contiendra en première position le nombre d'éléments convertis puis contient ensuite à la suite les éléments instanciés de nilist (peuvent être des instructions élaborées). Il convient d'enregistrer la valeur du pointeur, communément dans une variable (affecter alors manuellement le type en type 4 ou 5, fonction 'settype()'). L'intérêt de cette fonction est qu'elle est concise en soulageant le programmeur d'affecter élément par élément par lignes d'affectations successives un espace mémoire.

Vous devez libérer l'espace mémoire en utilisant la fonction 'free()' avant l'arrêt du programme. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Exemple de code Newprog : Création d'une liste de pointeurs de strings élaborées et affichent son contenu éléments par éléments

```
:init()
:settype(strlist,4)
```

NEWPROG2 & NPPTOC

```
:group({" string "&string(1),  " string2 ",  " string"&" 3 "})→strlist
:clrscr()
:For vv,1,strlist[0],1
:prints(strlist[vv]):nl()
:EndFor
:keywait()
:free(strlist)
```

Résultat à l'écran :
string1
string2
string3

Equivalent en code C de l'instruction : group({1,1+1,3*1})

```
long *list ;
list=malloc(4*4) ;
if(!list) return 0;
list[0]=3 ;
list[1]=1 ;
list[2]=1+1 ;
list[3]=3*1 ;
return list ;
```

L'équivalent en code Newprog serait d'une structure proche. On voit le gain en lignes d'écritures.

Fonction :*{dest_var,size_elem,elem1,elem2,...,elemn}*

Les listes instanciées définies dynamiquement. Lorsque cette instruction est exécutée, un espace mémoire est créé. Le pointeur de cet espace mémoire est alors affecté automatiquement à la variable dest_var (l'instruction renverra aussi la valeur du pointeur). Cet espace mémoire est alors initialisé comme une liste de n éléments, chacun d'une taille size_elem octets. Les éléments de cette liste seront elem1,elem2 etc... Les éléments doivent être du type d'un nombre (voire constante prédéfinie) ou d'une variable. Veillez à libérer l'espace mémoire créé avant la fin du programme pour éviter les pertes mémoires (fonction 'free()').

Cette instruction se différencie sur deux points de la fonction 'group()' :

-La taille des éléments peut être personnalisé (1,2,4,5)

-Mais ne peut pas contenir des instructions élaborées (hors nombre directs, variables et constantes prédéfinies)

NEWPROG2 & NPPTOC

Fonction : *newstr(str,var_dest)*

Alloue un espace mémoire de la taille de la chaîne de caractères str plus 1 (pour le caractère nul), puis copie tous les caractères de str dans cet espace mémoire. La variable newprog var_dest est alors affectée à ce pointeur. Ce bloc mémoire doit être libérer avec la fonction 'free()' avant la fin de l'exécution du programme sous peine de perdre définitivement de la mémoire RAM disponible (un reset est alors nécessaire pour la récupérer). La fonction retourne le pointeur de cet espace mémoire alloué. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Exemple de code Newprog : Copie une string et la compare avec l'originale

```
:init()
:" Hello World ! " →str1
:newstr(str1,str2)
:if not str2:finish()
:if strcmp(str1,str2)=0:text " str1 & str2 are the same "
:free(str2)
```

Fonction : *lb(address,elem)*

Retourne la valeur signée du elem élément de l'espace mémoire codé sur 1 octet pointé par address.

Le premier élément a l'indice 0.

L'équivalent en C est : *((char*)(address)+elem) ;

Exemple :

Le groupe d'instruction suivante :

```
:settype(lcd,1)
:getlcd()→lcd
:text " First screen byte= "&string(lcd[0])
```

Revient à l'écriture plus courte :

```
:text " First screen byte= "&string(lb(getlcd(),0))
```

Fonction : *peekb(address)*

Retourne la valeur signée codée sur 1 octet de l'adresse address.

Fonction : *lw(address,elem)*

Retourne la valeur signée du elem élément de l'espace mémoire codé sur 2 octets pointé par address.

Le premier élément a l'indice 0.

L'équivalent en C est : *((int*)(address)+elem) ;

NEWPROG2 & NPPTOC

Exemple :
S'inspirer de celui de 'lb()'.

Fonction : *peekw(address)*

Retourne la valeur signée codée sur 2 octets de l'adresse address.

Fonction : *ll(address,elem)*

Retourne la valeur signée du elem élément de l'espace mémoire codé sur 4 octets pointé par address.
Le premier élément a l'indice 0.
L'équivalent en C est : *((long*)(address)+elem) ;

Exemple :
S'inspirer de celui de 'lb()'.

Fonction : *peekl(address)*

Retourne la valeur signée codée sur 4 octets de l'adresse address.

Fonction : *wb(address,elem,value)*

Affecte la valeur value à l'élément elem codé sur 1 octet de l'espace mémoire pointé par address.
Le premier élément a l'indice 0.
L'équivalent en C est : *((char*)(address)+elem)=value ;

Fonction : *pokeb(address,value)*

Affecte à la valeur value l'élément codé sur 1 octet pointé par address.

Fonction : *ww(address,elem,value)*

Affecte la valeur value à l'élément elem codé sur 2 octets de l'espace mémoire pointé par address.
Le premier élément a l'indice 0.
L'équivalent en C est : *((int*)(address)+elem)=value ;

Fonction : *pokew(address,value)*

Affecte à la valeur value l'élément codé sur 2 octets pointé par address.

NEWPROG2 & NPPTOC

Fonction : *wl(address,elem,value)*

Affecte la valeur value à l'élément elem codé sur 4 octets de l'espace mémoire pointé par address.
Le premier élément a l'indice 0.
L'équivalent en C est : *((long*)(address)+elem)=value ;

Fonction : *pokel(address,value)*

Affecte à la valeur value l'élément codé sur 4 octets pointé par address.

2-Fonctions sauts

Fonction : *if condition then:...:else:...:endif*

Même syntaxe et fonctionnement que la fonction Tibasic. Si condition est vraie (cad différente de 0), alors la séquence d'instructions 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. Dans ce cas, si condition est fausse, saute directement après 'endif'.

Equivalent en C :

```
if(condition)
{
        … ;
}
else
{
        … ;
}
```

Fonction : *if condition:unic_instruction*

Même syntaxe et fonctionnement que la fonction Tibasic. Si condition est vraie (cad différente de 0), alors l'instruction unique unic_instruction sera exécutée. Si fausse, passe à la ligne d'instruction suivante.
Équivalent en C :

```
if(condition) unic_instruction ;
```

NEWPROG2 & NPPTOC

Fonction : *finish()*

Quitte le programme pour retourner au TIOS. L'utilisation de cette instruction dans une librairie est interdite.

Exemple : Alloue un espace mémoire. Si pas assez de mémoire, affiche l'erreur " Not enough memory " et quitte le programme

```
:init()
:malloc(30000)→buff
:if not buff then
:text " Not enough memory "
:finish()
:Endif
:text "Memory allocated "
:free(buff)
```

Fonction : *while cond::endwhile*

Tant que la condition cond est vraie (cad differente de 0), répète les instructions entre ' ::'.

Idem au Tibasic et au C.

Fonction : *for varname,begin,end,step::endfor*

Exécute les instructions entre les ' ::' en faisant varier la variable varname de la valeur begin à la valeur end avec un pas de step. Il faut veiller à ce que la valeur end soit bien atteinte pour éviter une boucle infinie.

Idem au Tibasic.

Les fonctions associées 'break()' et 'next()' apportent des fonctionnalités supplémentaires aux boucles 'For::EndFor'. Voir sections correspondantes.

Fonction : *break()*

Termine une boucle 'For:EndFor' ou 'While:EndWhile' en sautant après celle-ci. En cas d'imbrication de telles boucles, la dernière boucle active sera terminée. Cela permet de quitter une boucle manuellement.

Idem au C.

NEWPROG2 & NPPTOC

Exemple : Affichera par lignes successives : vv=1 vv=2 vv=3 vv=4 vv=5

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

Fonction : *next()*

Saute au début de la dernière boucle 'For:EndFor' ou 'While:EndWhile' active. Si il s'agit d'une boucle 'For:EndFor', la variable sera affectée a sa prochaine valeur.
Similaire à la fonction C 'continue'.

Fonction :*def(fc_name([arg1,arg2,...])):Enddef(fc_name())*
Syntaxe associée : *fc_name([arg1,arg2,...])*
Fonction associée : *rtn(value)*

Le bloc 'def():enddef' défini une fonction utilisateur de nom fc_name. Si passage d'argument(s) à la fonction il y a (arg1,arg2...), ils seront considérés comme des variables locales. Une variable locale peut avoir le même nom qu'une variable déjà définie dans le programme, mais elle aura sa valeur propre jusqu'à la fin de l'exécution de la fonction. L'utilisateur peut modifier sa valeur à l'intérieur de la fonction comme une variable locale ordinaire (comme en Tibasic et en C). Sa valeur initiale étant égale à l'argument passé à la fonction.
Il faut définir la fonction utilisateur en amont dans le code avant l'appel de cette fonction. En bonne pratique, on liste toutes ces fonctions dans la partie supérieure du programme après la définition éventuelle des types des variables Newprog (fonction 'settype()', même philosophie que le C).
L'utilisation de la fonction 'rtn(value)' dans le corps du bloc 'def():enddef()' permet de quitter la fonction en retournant la valeur value (similaire à l'instruction 'return' du Tibasic et du C).

Pour appeler la fonction, on utilisera la syntaxe conventionnelle suivante :

- fc_name([arg1,arg2,...]) : il doit y avoir le même nombre d'arguments que celui défini en amont

Exemple : Affiche "Hello World !" et ensuite "And have a good day !"

```
:init()
:"And have a good day !"→str          //Ici la variable globale 'str' est affectée
:def(hello(str))                      //Début de définition de la fonction utilisateur 'hello()'
```

NEWPROG2 & NPPTOC

```
:str&" "→str                    //Ajoute à la variable locale 'str' un espace et réaffecte 'str'
:rtn(str)                       //Valeur de retour de la fonction
:enddef(hello())                //Fin de définition de la fonction utilisateur 'hello()'
:def(world())                   //Début de définition de la fonction utilisateur 'world()'
:rtn("World !")                 //Valeur de retour de la fonction
:enddef(world())                //Fin de définition de la fonction utilisateur 'world()'
:Text hello("Hello")&world()    //Affiche la concaténation de"Hello " + "World !"
:Text str                       //Affiche "And have a good day !"
```

Fonction : *when(cond,{instruc1[,instruc2…]},{instruca[,instrucb…]})*

Cette fonction est utile car elle simplifie l'écriture des conditions 'if then else'. Si cond est vraie (i.e. différent de 0) alors instruc1,etc... sera/seront exécutée(s), si cond est fausse, ce sera instruca,etc.... 'When()' retourne la valeur de la dernière instruction exécutée. Elle est assez similaire à la fonction du tibasic When() mais en plus puissante car elle permet d'avoir plusieurs instructions. Néanmoins, et seulement en bytecode Newprog (pas en NPPTOC), 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).

Exemple : Demande les âges respectifs de Wilfried et de John puis affiche qui est le plus âgé.

```
:init()
:basic
:request "Age de Wilfried",age1          //Saisie de age1 (variable tibasic)
:request " Age de John",age2             //Saisie de age2 (variable tibasic)
:endbasic
:text when(atol(osvar(age1))>atol(osvar(age2)),{" Wilfried "},{" John "})& "  is older "
```

Fonction : *i(condition):instructions:y*

Cette fonction est similaire à un 'If :Endif' classique avec certaines caractéristiques additionnelles. Si condition est vraie (c'est à dire différente de 0), alors les instructions instructions seront exécutées. Si condition est fausse, passera les instructions sans les exécuter. Cette fonction à un intérêt car elle est plus concise et plus simple à écrire qu'un 'If-Endif' classique. Autre intérêt mineur, on peut insérer un 'i():y' dans des listes non instanciées sans que le compilateur affiche un message d'erreur à la compilation contrairement au 'If-Endif' classique.

ATTENTION : Cette syntaxe est à utiliser avec prudence. Préférer au maximum la syntaxe classique 'if then endif'. De plus, avec NPPTOC, l'utilisation de 'goto' et 'label' à l'intérieur d'un 'i():y' est interdite sous peine de crash. Il y a aussi un risque que les variables locales (passées en arguments à une fonction par exemple) ne soient pas reconnues.

Exemple : Utilisation dans la définition d'une interruption. Affiche un message si on appuie sur la touche 2nd. Ici, on ne pourrait pas utiliser un 'If:Endif' classique car non toléré entre les crochets '{}'.

NEWPROG2 & NPPTOC

```
:init()
:clrscr()
:inter(1,1,{i(second()),prints(" Second pressed ! "),y})
:While not esc()
:Endwhile
```

Fonction : *x:instructions:y*

Cette fonction est difficile à manipuler, n'est pas indispensable et est souvent compliquée à lire. Elle a de l'intérêt surtout utilisée conjointement avec la fonction 'map()' car permet un gain de vitesse comparée à une boucle 'For:Endfor' (voir 'map()') (seulement en bytecode Newprog, pas en NPPTOC).

A l'exécution de 'x', toutes les instructions instructions jusqu'à 'y' seront exécutées en un passage. Ressemble à l'instruction 'i():y' mais sans condition. Cette fonction a le gros avantage de pouvoirs exécuter plusieurs instructions juste au passage de x". Comme 'x' est vu comme un seul argument, on peut faire dépasser les instructions à exécuter (voir exemples) en dehors des parenthèses. Retourne la valeur de la dernière instruction.

Ressemble à l'appel d'une fonction mais sans avoir à en définir une (intéressant pour les gros programmes car le nombre de fonctions utilisateur est limité à 30).

Limitations :

- 'x' ne peut être placé qu'en dernier argument d'une instruction le cas échéant (voir exemple).
- On ne peut manipuler 'x', on doit l'utilisé seul (brut). Exemple d'erreur :
  - :text " Est mineur ? : "&x
  - :if age<18 then
  - " oui "→ss
  - :else
  - " non "→ss
  - endif
  - ss //valeur de retour
  - y
- Il est interdit d'utiliser cette instruction dans dans une liste non instanciée ('{}')
- L'utilisation de 'goto' et 'label' à l'intérieur d'un 'x:y' est interdite.
- L'utilsation de 'rtn()' à l'intérieur d'un 'x:y' est interdite.

Exemple 1: Remplissage écran avec un sprite noir et affiche le nombre de sprites affichés. Plus rapide qu'avec deux boucles 'For:EndFor'.

```
:init()
:0→nbdisp
:seqb(sprt,xx,1,8,1,-1)          //créé le sprite (carré noir)
:if not sprt:finish()            //si pas assez de mémoire alors quitte le programme
:map(xx,0,19*8,8,x)              //boucle en X, 'x' est bien le dernier argument
```

NEWPROG2 & NPPTOC

```
:map(yy,0,12*8,8,x)          //boucle en Y, 'x' est bien le dernier argument
:sprt8(xx,yy,8,sprt)         //Affiche un sprite
:isz(nbdisp)                 //incrémente le compteur de sprites
:y
:y
:text " Nb displayed : "&string(nbdisp)      //affiche le nombre de sprites affichés
:free(sprt)                  //libère la mémoire liée au sprite précédemment créé
```

Exemple 2:Demande deux âges et affiche quel âge est le plus élevé

```
:init()
:basic
:request " Age1 ",age1                  //Saisie de age1 (variable tibasic)
:request " Age2 ",age2                  //Saisie de age2 (variable tibasic)
:endbasic
:text x                                 //x est bien le dernier argument et est utilisé 'brut'
:if atol(osvar(age1))>atol(osvar(age2)) then //si age1>age2
:"Older="&osvar(age1)→ss                //string de retour age1 dans ss
:else
:"Older="&osvar(age2)→ss                //string de retour age2 dans ss
:endif
:ss                                     //valeur de retour de x est ss (car dernière instruction)
y
```

Est équivalent à la structure plus classique (qui est plus lisible et à privilégiée) :

```
:init()
:def(maxage(a1,a2))
:if a1>a2 then                          //si age1>age2
:rtn("Older="&string(a1))               //string de retour age1
:else
:rtn("Older="&string(a2))               //string de retour age2
:endif
:enddef(maxage())
:basic
:request " Age1 ",age1                  //Saisie de age1 (variable tibasic)
:request " Age2 ",age2                  //Saisie de age2 (variable tibasic)
:endbasic
:text maxage(atol(osvar(age1)),atol(osvar(age2)))
```

NEWPROG2 & NPPTOC

Fonction : *two(instruc1,instruc2)*

Cette fonction est dépréciée. Elle n'a qu'un rare intérêt qu'avec la fonction 'map()'. Elle permet d'exécuter les instructions instuc1 et instruc2 en une seule instruction (celle de 'two()'). La philosophie est proche du bloc 'x:y' mais pour seulement 2 instructions et ne requiert pas de dépassement de parenthèses comme cette dernière.

Fonction : *lbl label_name*

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 variables NewProg, le nom des labels label_name doivent être d'au moins 2 caractères.

Fonction : *goto label_name*

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'.

Si placer dans une fonction utilisateur 'def():enddef()', il faudra veiller à ce que l'instruction 'lbl' correspondante soit bien dans le corps de cette même fonction.

Il est à noter qu'on ne peut utiliser cette fonction dans les blocs 'i():y'et 'x:y'.

Fonction : *map(var,start,end,step,instruction)*

Exécute instruction en faisant varier var de start à end avec un pas de step. Cette instruction est plus rapide qu'une boucle 'While:EndWhile' classique ainsi que les boucles 'For:Endfor' (hors NPPTOC). Retourne la dernière valeur retournée par instruction.

En bytecode Newprog (mais pas en NPPTOC), l'utilisation de 'map' est jusqu'à 3 fois plus rapide que une boucle 'For:EndFor'. Si le besoin de vitesse n'est pas primordiale, utiliser une boucle 'For:EndFor' classique car le code paraîtra plus 'propre'.

Remarque : Les valeurs de fin end et de pas step ne seront pas réévaluées à chaque itération.

Exemple : Remplissage écran avec un sprite noir et affiche le nombre de sprites affichés. Ce code utilisa la notation 'x:y'. Voir section correspondante.

```
:init()
:0→nbdisp
:seqb(sprt,xx,1,8,1,-1)        //créé le sprite (carré noir)
:if not sprt:finish()          //si pas assez de mémoire alors quitte le programme
:map(xx,0,19*8,8,x)            //boucle en X
:map(yy,0,12*8,8,x)            //boucle en Y
```

```
:sprt8(xx,yy,8,sprt)            //Affiche un sprite
:isz(nbdisp)                    //incrémente le compteur de sprites
:y
:y
:text " Nb displayed : "&string(nbdisp)      //affiche le nombre de sprites affichés
:free(sprt)                     //libère la mémoire liée au sprite précédemment créé
```

3-Fonctions affichages de texte

Fonction : *clrscr()*

Efface l'écran actuel et réinitialise au coordonnées (0,0,) le curseur de texte.
Equivalent C : 'clrscr()'

Fonction : *prints(string_ptr)*

Affiche à l'écran (buffer vidéo) la chaîne de caractères (ou string) à la position du curseur de texte.
Equivalent C : 'printf("%s",string_ptr)'

Exemple : Affiche à l'écran "Hello World !"

```
:init()
:clrscr()
:prints("Hello World !")
:keywait()
```

Fonction : *printc(ascii_code)*

Affiche à l'écran (buffer vidéo) le caractère correspondant au code ascii_code as à la position du curseur de texte.
Equivalent C : 'printf("%c",ascii_code)'

Exemple : Petit programme de traitement de texte

```
:init()
:clrscr()
:while not esc()
:printc(keywait())
:EndWhile
```

NEWPROG2 & NPPTOC

Fonction : *printld(value)*

Affiche à l'écran (buffer vidéo) la valeur value à la position du curseur de texte.
Equivalent C : 'printf("%ld",(long)value)'

Fonction : *setfont(text_size)*

Modifie la taille des caractères affichés à l'écran (ou buffer) à la valeur text_size. text_size peut prendre les valeurs 0 (petit),1 (moyen) ou 2 (grand). Retourne l'ancienne valeur de la taille.
Equivalent C : 'FontSetSys(text_size)'

Fonction : *newline()* ou *nl()*

Fait un retour chariot du curseur de texte.
Equivalent C : 'printf("\n")'

Fonction : *printf1("format",arg)*

La fonction 'printf1()' permet d'afficher sur la fenêtre de l'écran (ou buffer) un texte qui suit un format défini par le programmeur où format est une chaîne de caractères qui contient :

– du texte à imprimer tel quel ;

– une spécification de format qui indique comment afficher la variable arg fournie en paramètre. Ce sont des codes formats.

Chaque code format commence par le symbole '%' suivi par une (ou deux) lettre(s) indiquant le format d'affichage du paramètre arg correspondant. La valeur de arg remplacera son code format à l'affichage.

La lettre placée après le symbole % dans le code format indique le type du paramètre associé au code format (voir plus bas).

En Newprog, les codes formats les plus utilisés sont :

%ld >> affiche un nombre ou pointeur

%s >> affiche une chaîne de caractères (ou string)

%lx >> affiche un nombre au format hexadécimal en minuscule

NEWPROG2 & NPPTOC

%lX >> affiche un nombre au format hexadécimal en majuscule

Remarque : %c %d %x %X ne fonctionne pas car Newprog manipule que des chiffres long.

Equivalent C : 'printf("format",arg)'

Exemple : Affiche à l'écran “One hundred :100”
:init()
:clrscr()
:printf1(“One hundred :%ld”,100)
:keywait()

Fonction : *printf2("format",arg1,arg2)*

Idem que 'printf1()' mais avec deux arguments.

Equivalent C : 'printf("format",arg1,arg2)'

Fonction : *printxy(x,y,"format",arg)*

Idem que 'printf1()' avec la particularité d'afficher la chaîne de caractères au coordonnées x et y de l'écran (ou du buffer).

Equivalent C : 'printf_xy(x,y,"format",arg)'

Fonction : *cwidth(ascii_code)*

Renvoi la largeur en pixels d'un caractère ayant pour code ascii ascii_code suivant la taille de caractère active (fonction 'setfont()').

Equivalent C : 'FontCharWidth(ascii_code)'

NEWPROG2 & NPPTOC

Fonction : *moveto(x,y)*

Affecte la position actuel du curseur de position (des fonctions printf1(), prints(),etc) à la positon (x, y)

Equivalent C : 'MoveTo(x,y)'

Fonction : *pause(string_ptr)*

Affiche la chaîne de caractères pointée par string_ptr sur une nouvelle ligne à l'écran à partir du curseur de texte. Attend l'appui d'une touche. Retourne la position du texte sur une nouvelle ligne.

Equivalent C : 'printf("\n%s",string_ptr) ; ngetchx() ;'

Fonction : *text str*

Fonction identique à la fonction Tibasic 'text'.
Affiche dans une boite de dialogue le texte str.
Réalise un appel en interne de la fonction 'keydisp()'.

Equivalent C :

```
long text_(unsigned char *str)
 {

        unsigned char len,CESI_list[50];
        CESI_list[0]=0;
        strcpy(CESI_list+1,str);
        len=strlen(str);
        CESI_list[len+2]=STR_TAG;
        if(save_int_1)
        {
                SetIntVec (AUTO_INT_1, save_int_1);
                save_int_1=NULL;
        }
        TRY
        cmd_text (CESI_list+len+2);
        ONERR
        return 0;
        ENDTRY
        return 1;
 }
```

NEWPROG2 & NPPTOC

4-Fonctions manipulations de texte

Fonction : *char(ascii_code)*

Renvoie un pointeur sur une chaîne de caractères ayant comme seul composante un caractère correspondant à ascii_code. La valeur de retour sera valable que si cette fonction n'est pas appelée plus de trois fois après celle-ci. Donc pensez à sauvegarder le résultat si nécessaire.

Fonction similaire à la fonction Tibasic de même nom.

Equivalent C : sprintf(buffer,"%c",ascii_code) //buffer=espace mémoire de la chaine de caractères renvoyée

Exemple : Petit programme de traitement de texte

```
:init()
:clrscr()
:while not esc()
:prints(char(keywait()))
:EndWhile
```

Fonction : *ord(char_str)*

Renvoie le code ASCII du caractère présent dans la chaîne de caractères char_str. Fonction similaire à la fonction Tibasic de même nom. Cette fonction est l'inverse de la fonction 'char()'.

Fonction similaire à la fonction Tibasic de même nom.

Fonction : *string(value)*

Convertit un nombre value en chaîne de caractères. Retourne le pointeur de cette chaîne de caractères. La valeur de retour de cette fonction sera valable jusqu'au quatrième prochain appel de cette fonction donc penser à faire une copie si nécessaire (avec 'newstr()' par exemple). Cette fonction est l'inverse de la fonction 'expr()' ou 'atol()'.

Cette fonction ressemble à la fonction Tibasic de même nom.

Equivalent C : sprintf(buffer,"%ld",value) ;

Exemple : Affiche : "I'm 20 years old"

```
:init()
:text "I'm "&string(20)&" years old"
```

Fonction : *strcat(dest_str,str)*

Ralonge (concatène) la chaîne de caractères pointée par dest_str avec la chaîne de caractères pointée par str. Veiller à ce que la taille de l'espace mémoire pointé par dest soit suffisamment grand pour

pouvoir accueillir la chaîne de caractères str. La fonction renvoie dest_str.

Equivalent C : strcat(dest_str,str)
Exemple : Affiche : "I'm 20 years old"

```
:init()
:b(buff):repeat(30,0):y
:strcat(buff,"I'm ")
:strcat(buff,string(20))
:strcat(buff," years old")
:text buff
```

Fonction : *strcat2(dest_str,str)* ou *dest_str&str*

Assez similaire à la fonction 'strcat()' tout en permettant d'utiliser la notation dest_str&str (comme en Tibasic). Cependant, dest_str n'est pas modifiée après l'exécution de l'instruction. Le résultat de l'addition (concaténation) des deux chaînes de caractères est sauvegardé dans un espace mémoire spécial qui restera valable temporairement en mémoire. Il faut veiller à ce que la chaîne de caractères générée ne dépasse pas 50 caractères sous peine de crasher la calculatrice. Cette fonction est intéressante pour afficher du texte à l'écran. Si on utilise cette fonction plus de deux fois dans une même ligne, le résultat sera faussé. Donc pensez à sauvegarder régulièrement le résultat de retour avec par exemple la fonction 'newstr()'. Cette fonction retourne le pointeur sur la chaîne de caractères générée.

Exemple : Affiche : "I'm 20 years old"

```
:init()
:text "I'm "&string(20)&" years old"
```

Fonction : *strcmp(str1,str2)*

Compare le contenu des deux chaînes de caractères str1 et str2 entre elles. La fonction 'strcmp()' commence par le premier caractère dans chaque chaîne de caractères et passe au caractère suivant et ainsi de suite jusqu'à ce qu'un caractère diffère ou que la fin de la chaîne de caractères est atteinte. 'strcmp()' retourne les valeurs suivantes :

- < 0 si str1 est plus petit que str2
- == 0 si str1 est identique à str2
- > 0 si str1 est plus grande que str2

Plus précisément, si les chaînes de caractères diffèrent, la valeur du premier caractère qui diffère de str2 moins celui du caractère correspondant de la chaîne str1 est renvoyé.

Equivalent C : strcmp(str1,str2)

NEWPROG2 & NPPTOC

Fonction : *direct_str=str2*

Compare la chaîne de caractère explicite (écrite directement) direct_str avec la chaîne de caractères str2 (explicite ou non). Si les deux chaînes de caractères ont le même contenu, alors renvoie 1. Aussi-non renvoie 0.

Equivalent C :
if(strcmp(direct_str,str2)==0) return 1 ;
else return 0 ;

Exemple : Si la chaîne de caractères saisie par l'utilisateur est bien "john", alors affiche "Hello John". Aussi-non, affiche "Hello stranger"

```
:init()
:basic
:request "Enter your name",namestr
:endbasic
:if "john"=osvar(namestr) then
:text "Hello john"
:else
:text "Hello stranger"
:endif
```

Fonction : *sprintf(buffer,formated_str,data)*

Cette fonction effectue l'écriture de texte selon un certain format (de la donnée data dans formated_str) à partir d'un espace mémoire buffer. Il faut s'assurer que buffer recevant les données soit assez grand pour recevoir le contenu demandé. Le comportement de cette fonction est similaire à la fonction 'printf1()' sauf qu'il retourne son résultat dans une chaîne de caractères (buffer) plutôt que de l'afficher.
Retourne la taille de la chaîne de caractères ainsi créée (sans le '\0' final).

Equivalent C : sprintf(buffer,formated_str,data)

Exemple : Affiche "9+10=19"

```
:init()
:b(buff):repeat(10,0):y
:sprintf(buff,"9+10=%ld",9+10)
:text buff
```

NEWPROG2 & NPPTOC

Fonction : *strcpy(buff,src)*

Copie une chaîne de caractères pointée par src vers l'espace mémoire pointé par buff. Le caractère 0 de fin de chaîne de caractères est aussi copié. La fonction retourne buff. Note : Si les chaines de caractères pointée par src et buff se chevauchent en mémoire, le comportement est indéfini. 'strcpy()' considère que dest est un espace mémoire assez grand pour accueillir src.

Equivalent C : strcpy(buff,src)

Exemple : Equivalent en code Newprog de la fonction Newprog 'newstr()'

```
:init()
:def(newstr2(str,vardest))
:malloc(strlen(str)+1)→vardest
:strcpy(vardest,str)
:enddef(newstr2())
```

Fonction : *gets(buffer)*

Récupère une chaîne de caractères saisie au clavier jusqu'à temps que la touche ENTER soit pressée et la copie dans buffer. La touche espace est supportée.
Veillez à ce que l'espace mémoire pointé par buffer soit assez grand pour éviter un crash.

Equivalent C : gets(buffer)

Exemple : Demande votre nom et affiche "Hello XXXX"

```
:init()
:b(buff):repeat(20,0):y
:clrscr()
:prints("Enter your name :"):nl()
:gets(buff)
:text "Hello "&buff
```

Fonction : *strlen(string)*

Renvoie la longueur en caractères de la chaîne de caractères string. Le caractère nul de fin de string n'est pas compté. Cette fonction est très utile.

NEWPROG2 & NPPTOC

Fonction : *expr(string)* ou *atol(string)*

Converti la chaîne de caractères string en un nombre et renvoie ce nombre. Similaire à expr() du Tibasic mais ne prends que des strings ou des pointeurs de string en arguments (avec un éventuel +/- ou espaces). Ces fonctions sont l'inverse de la fonction 'string()'.

Equivalent C : atol(string)

5-Fonctions graphiques

Fonction : *clrlcd()*

Efface la mémoire vidéo actuelle. Cette fonction n'affecte pas la position du curseur de texte comme le fait le fonction 'clrscr()'. Elle peut être utilisée pour faire des niveaux de gris. Elle est légèrement plus rapide que la fonction 'clrscr()'.

Fonction : *gmode(gmode)*

Sélectionne le mode d'affichage de toutes les fonctions graphiques. Retourne l'ancienne valeur du mode graphique. Les différentes valeurs possibles de gmode sont :

0 = gor : mode classique OR – Par défaut
1 = gerase = greplace : Mode effacement – Mode replacement pour les fonctions de sprites
2 = gxor : Mode XOR >> inverse l'état des pixels
3 = gand : Mode ET – Seulement pour les fonctions de sprites – Permets de masquer

Fonction : *setlcd(*video_memory_address*)*

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

NEWPROG2 & NPPTOC

Equivalent C :

```
unsigned char* setlcd(video_memory_address)
{
        unsigned char *old_lcdmem;
        old_lcdmem=*lcdad_ptr;              // lcdad_ptr étant le pointeur vidéo courant
        PortSet ((*lcdad_ptr=video_memory_address), 239, 127);
        return old_lcdmem;
}
```

Exemple : Affiche la suite des 185 premiers entiers positifs sans visualiser leurs « traçages »

```
:init()
:setfont(0)                     //Petite police de texte
:malloc(lcdsize)→scrbuff        //Nouvelle zone graphique
:if not scrbuff:finish()
:clrscr()
:prints("Drawing in progress...")
:getlcd()→video                 //Récupère le pointeur de la zone graphique visible (=0h4c00)
:setlcd(scrbuff)                //Affecte les tracés pour la nouvelle zone graphique
:clrscr()
:for vv,1,185,1
:printf1("%ld ",vv)             //Effectue les traçages
:endfor
:memcpy(video,scrbuff,lcdsize)  //Copie la zone graphique « remplie » dans la zone graphique visible
:keywait()
:setlcd(video)                  //Réaffecte les tracés dans la zone visible (par défaut)
:free(scrbuff)
```

Fonction : *getlcd()*

Retourne l'adresse de la mémoire vidéo actuelle (Par défaut, =0h4c00 si les niveaux de gris ne sont pas activés).

Fonction : *drawpic(x,y,pic_str)*

Affiche aux coordonnées (x,y) l'image bitmap Tibasic dont le nom est pic_str (rep\file). Le mode d'affichage est défini avec la fonction 'gmode()'. Cette fonction est proche des fonctions Tibasic telle que rplcpic et rclpic. Retourne 0 si une erreur est survenue, 1 aussi-non.

NEWPROG2 & NPPTOC

Fonction : *getpic(x1,y1,x2,y2,pic_str)*

Sauvegarde le contenu de l'écran entre les points (x1,y1) et (x2,y2) dans la variable Tibasic pic_str. Renvoi la taille en octets du fichier créé. Assez similaire à la fonction Tibasic stopic, avec toutefois la possibilité d'atteindre tout l'écran. Cette fonction est lente. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Fonction : *sprt8(x,y,h,sprite)*

Affiche le sprite (8 bits horizontales) sprite avec le coin supérieur au point (x,y) et une hauteur de sprite de h lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. 'sprt8()' affiche environ 6000 sprites par secondes. Voir 'gmode()' pour les modes d'affichage possibles.

Exemple : Affiche au centre de l'écran un sprite qui représente un carré

```
:init()
:b(sprt)
:0b11111111
:0b10000001
:0b10000001
:0b10000001
:0b10000001
:0b10000001
:0b10000001
:0b11111111
:y
:clrlcd()
:sprt8(76,46,8,sprt)
:keywait()
```

Fonction : *sprt82(x,y,h,sprite)*

'sprt82()' est une variante de 'sprt8()'. Elle permet de gérer les collisions. Contrairement à 'sprt8()', elle retournera une valeur non nulle si au moins un pixel actif du sprite sprite est entré en collision avec un pixel à l'écran lors de son affichage (aussi-non retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur deux octets chacun (valable jusqu'au prochain appel d'une fonction 'sprtX2()'). Le premier élément de cette liste sera égale à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite. Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction 'settype()' pour pouvoir utiliser la notation [ ]. Aussinon,

l'utilisation de 'lb(pointeur_sur_la_liste,élément)' peut être utiliser. Voir 'gmode()' pour les modes d'affichage possibles.

Exemple : Voir la fonction analogue 'sprt162()'

Fonction : *sprt16(x,y,h,sprite)*

Affiche le sprite (16 bits horizontales) sprite avec le coin supérieur au point (x,y) et une hauteur de sprite de h lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. 'sprt16()' affiche environ 5000 sprites par secondes. Voir 'gmode()' pour les modes d'affichage possibles.

Exemple : Affiche un rectangle de 16x4 pixels rempli à l'écran

```
:init()
:seqw(sprite,vv,1,4,1,-1)          //On pourrait aussi utiliser la syntaxe 'w(sprite):repeat(4,-1):y'
:clrlcd()
:sprt16(72,48,4,sprite)
:keywait()
:free(sprite)
```

Fonction : *sprt162(x,y,h,sprite)*

'sprt162()' est une variante de 'sprt16()'. Contrairement à 'sprt16()', elle retournera une valeur non nulle si au moins un pixel actif du sprite sprite est entré en collision avec un pixel à l'écran lors de son affichage (aussi-non retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur deux octets chacun (valable jusqu'au prochain appel d'une fonction 'sprtX2()'). Le premier élément de cette liste sera égale à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite. Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction 'settype()' pour pouvoir utiliser la notation [ ]. Aussinon, l'utilisation de 'lw(pointeur_sur_la_liste,élément)' peut être utiliser. Voir 'gmode()' pour les modes d'affichages possibles.

Exemple : Affiche un parcours et fait bouger avec les touches un sprite si il ne rentre pas en collision. Si rentre en collision, affiche les informations de retour des 'sprt162()'.

```
:init()
:w(sprt)
:repeat(6,0b0001110001110000)
```

NEWPROG2 & NPPTOC

```
:0b1100000000000011
:0b1111111111111111
:repeat(8,0b1100110011110011)
:y
:clrscr()
:40→yy
:80→xx
:0→nc
:settype(rr,2)
:gmode(2)
:fillrect(0,0,40,127)
:fillrect(160-40,0,160,127)
:fillrect(0,0,160,10)
:fillrect(0,95,160,100)
:dline(50,0,40,100)
:dline(110,0,120,100)
:dline(60,0,60,30)
:dline(60,99,65,80)
:sprt16(xx,yy,16,sprt)
:While gkey()!=esc
:wait(5)                    //Pour ralentir le mouvement
:isz(inc)
:if leftt() Then
:sprt16(xx,yy,16,sprt)
:if sto(rr,sprt162(dsz(xx),yy,16,sprt)) Then
:sprt16(isz(xx),yy,16,sprt)
:EndIf
:EndIf
:if rightt() Then
:sprt16(xx,yy,8,sprt)
:if sto(rr,sprt162(isz(xx),yy,16,sprt)) Then
:sprt16(dsz(xx),yy,16,sprt)
:EndIf
:EndIf
:if up() Then
:sprt16(xx,yy,16,sprt)
:if sto(rr,sprt162(xx,dsz(yy),16,sprt)) Then
:sprt16(xx,isz(yy),16,sprt)
:EndIf
:EndIf
:if down() Then
```

```
:sprt16(xx,yy,16,sprt)
:if sto(rr,sprt162(xx,isz(yy),16,sprt)) Then
:sprt16(xx,dsz(yy),16,sprt)
:EndIf
:EndIf
:if rr Then
:printxy(0,0,"height=%ld         ",rr[0])
:printxy(70,0,"value=%ld         ",rr[1])
:Else
:EndIf
:EndWhile
```

Fonction : *sprt32(x,y,h,sprite)*

Affiche le sprite (32 bits horizontales) sprite avec le coin supérieur au point (x,y) et une hauteur de sprite de h lignes. Dans le cas ou x ou et y sont négatifs, les pixels en dehors de la fenêtre (0,0,239,127) ne seront pas traités. 'sprt16()' affiche environ 5000 sprites par secondes. Voir 'gmode()' pour les modes d'affichage possibles.

Fonction : *sprt322(x,y,h,sprite)*

'sprt322()' est une variante de 'sprt32()'. Contrairement à 'sprt32()', elle retournera une valeur non nulle si au moins un pixel actif du sprite sprite est entré en collision avec un pixel à l'écran lors de son affichage (aussinon retournera 0). Si collision, le sprite, ne sera pas affiché et la fonction renverra un pointeur sur une liste de nombres codés sur quatre octets (ce pointeur est valable jusqu'au prochain appel d'une fonction 'sprtX2()'). Le premier élément de cette liste sera égal à la hauteur ou est intervenue la collision détectée. Le deuxième élément sera égal à la valeur du sprite (à la hauteur renvoyée en premier élément). Cette valeur permet de donner une indication sur la position de la ou des collision(s). Par exemple, si cette valeur est négative, c'est qu'il y a eu collision avec le bit le plus à gauche (à la hauteur spécifiée). Si la valeur est égale à 1, il y a eu collision seulement sur le bit le plus à droite. Pour récupérer ces valeurs, il est intéressant d'utiliser préalablement la fonction 'settype()' pour pouvoir utiliser la notation [ ]. Aussinon, l'utilisation de 'll(pointeur_sur_la_liste,element)' peut être utiliser. Voir 'gmode()' pour les modes d'affichage possibles.

Exemple : Voir analogue 'sprt162()'

Fonction : g*sprt8(x,y,h,sprite_dest)*

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

NEWPROG2 & NPPTOC

Fonction : g*sprt16(x,y,h,sprite_dest)*

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

Fonction : g*sprt32(x,y,h,sprite_dest)*

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

Fonction : g*sprt8x(x,y,h,byte_width,sprite_dest)*

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

Exemple : Remplie l'écran de la copie du quart supérieur gauche de l'écran initial

```
:init()
:malloc(10*50)→sprt
:gsprt8x(0,0,50,10,sprt)          //Capture le coin supérieur gauche
:clrscr()
:sprt8x(0,0,50,10,sprt)           //Affiche dans le coin supérieur gauche
:sprt8x(10*8,0,50,10,sprt)        //Affiche dans le coin supérieur droit
:sprt8x(0,50,50,10,sprt)          //Affiche dans le coin inférieur gauche
:sprt8x(10*8,50,50,10,sprt)       //Affiche dans le coin inférieur droit
:keywait()
:free(sprt)
```

NEWPROG2 & NPPTOC

Fonction : *sprt8x(x,y,h,byte_width,sprite)*

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

Fonction : *gsprt(w,h,bitmap_str,sprtdest)*

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_str. 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. Le grand intérêt de cette fonction est de pouvoir utiliser les fonctions de sprites ultérieurement lesquelles sont beaucoup plus rapides que l'affichage d'une image bitmap.

Exemple : Créé le sprite pp à partir d'une image nommé 'bitmap' que l'utilisateur à préalablement créé (de largeur 8x8 pixels obligatoirement) et affiche ce sprite au centre de l'écran.

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

Fonction : *dline(xa,ya,xb,yb)*

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

Fonction : *dmline(list_of_x,list_of_y,num_pts)*

Trace une première ligne puis une deuxième 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.

Exemple : Trace un rectangle centré

```
:init()
```

NEWPROG2 & NPPTOC

```
:b(lx)
:20
:140
:140
:20
:20
:y
:b(ly)
:20
:20
:80
:80
:20
:y
:clrscr()
:dmline(lx,ly,5)
:keywait()
```

Fonction : *dcircle(xc,yc,radius)*

Affiche un cercle de centre (xc,yc) avec un rayon égal à radius. Voir 'gmode()' pour le mode d'affichage. Les coordonnées du centre du cercle doivent être compris dans le cadre de la fenêtre graphique.

Fonction : *fillcirc(xc,yc,radius)*

fDessine un cercle plein de centre (xc,yc) avec un rayon égal à radius. Voir 'gmode()' pour le mode d'affichage. Les coordonnées du centre du cercle peuvent être en dehors de la fenêtre graphique (la fonction est clippée).

Fonction : *dpix(x,y)*

Dessine un pixel à l'écran aux coordonnés (x,y). Le mode d'affichage est sélectionné comme avec toutes les fonctions graphiques à l'aide de la fonction 'gmode()'.

Fonction : *gpix(x,y)*

Retourne l'état du pixel pointé par x et y : 1 si allumé 0 si éteint.

NEWPROG2 & NPPTOC

Fonction : *fillrect(xa,ya,xb,yb)*

Dessine un rectangle plein entre le points (xa,ya) et (xb,yb). Le mode d'affichage est sélectionné avec la fonction 'gmode()'.

Fonction : *grayon()*

Active les niveaux de gris (4 niveaux). Cette fonction désactive les boites de dialogues d'affichage d'erreurs et réactivera la valeur au prochain appel de 'grayoff()'.

Equivalent C:GrayOn()

Exemple : Voir fonction 'light()'

Fonction : *grayoff()*

Désactive les niveaux de gris (4 niveaux).

Equivalent C:GrayOff()

Exemple : Voir fonction 'light()'

Fonction : *light()*

Active le plan des gris clairs. La fonction grayon() doit avoir été exécutée précédemment. Cette fonction affecte l'adresse de la mémoire vidéo au plan 'light'.

Exemple : Affiche un rectangle en gris clair et un rectangle en gris foncé. Leur intersection sera noire.

```
:init()
:grayon()                 //Active les niveaux de gris
:light()                  //Sélectionne le plan gris clair
:fillrect(10,10,90,90)    //Premier rectangle plein
:dark()                   //Sélectionne le plan gris foncé
:fillrect(70,20,150,80)   //Deuxième rectangle plein
:keywait()
:grayoff()                //Désactive le mode niveaux de gris
```

NEWPROG2 & NPPTOC

Fonction : *dark()*

Active le plan des gris foncés. La fonction grayon() doit avoir été exécutée précédemment. Cette fonction affecte l'adresse de la mémoire vidéo au plan 'dark'.

Exemple : Voir fonction 'light()'

Fonction : *drawstr(x,y,str)*

Affiche au coordonnées (x,y), la chaîne de caractères str. Cette fonction n'affecte pas le curseur de position du texte (fonctions 'prints()', ...).

Fonction : *savescr(index)*

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()' libérera 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 libérer par vous même avec la fonction free. A utiliser avec la fonction 'loadscr()'. Index ne peut prendre que les valeurs suivantes : 0,1,2,3. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Fonction : *loadscr(index)*

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

Exemple : Sauvegarde, efface et restore l'écran

```
:init()
:savescr(0)
:text ”Screen saved”
:clrscr()
:text ”Screen cleared”
:loadscr(0)
:text ”Screen restored”
```

Fonction : *lcdup()*

Augmente le contraste de 1.Retourne la nouvelle valeur du niveau de contraste.

Fonction : *lcddown()*

Abaisse le contraste de 1.Retourne la nouvelle valeur du niveau de contraste.

Fonction : *prettyxy(x,y,expr_str)*

Affiche sous la forme « pretty print » (comme dans l'écran home) au coordonées (x,y) l'expression expr_str (chaine de caractères). Retourne 0 si l'expression expr_str contient une erreur, 1 aussi-non.

Exemple : Voir fonction 'getwbt()'

Fonction : *getwbt(expr_str,width_dest_var,bottom_dest_var,top_dest_var)*

Retourne des informations relatives à la dimension d'un bloc affiché à l'écran à l'aide de la fonction 'prettyxy()'. Les informations seront sauvegardées dans les trois variables width_dest_var (largeur en pixel),bottom_dest_var (ordonnée la plus basse),top_dest_var (ordonnée la plus haute).

Exemple :Affiche une expression dans le coin supérieur gauche, au centre de l'écran et dans le coin inférieur bas.

```
:init()
:clrscr()
:"sin(7*x^2+y)"→ex
:getwbt(ex,ww,bb,tt)
:prettyxy(0,0+tt,ex)
:prettyxy(80-ww/2,49,ex)
:prettyxy(159-ww,99-tt+bb,ex)
:keywait()
```

Fonction : *lscroll(num_line)*

Effectue une translation de 1 pixel vers la gauche de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 160 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo pour obtenir l'effet voulu (fonction 'setlcd()') .

NEWPROG2 & NPPTOC

Fonction : *rscroll(num_line)*

Effectue une translation de 1 pixel vers la droite de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 160 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo pour obtenir l'effet voulu (fonction 'setlcd()') .

Fonction : *uscroll(num_line)*

Effectue une translation de 1 pixel vers le haut de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 160 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction 'setlcd()').

Fonction : *bscroll(num_line)*

Effectue une translation de 1 pixel vers le bas de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 160 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction 'setlcd()').

Fonction : *lscroll2(num_line)*

Effectue une translation de 1 pixel vers la gauche de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 240 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo pour obtenir l'effet voulu (fonction 'setlcd()') .

Fonction : *rscroll2(num_line)*

Effectue une translation de 1 pixel vers la droite de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 240 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo pour obtenir l'effet voulu (fonction 'setlcd()') .

Fonction : *uscroll2(num_line)*

Effectue une translation de 1 pixel vers le haut de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 240 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction 'setlcd()').

NEWPROG2 & NPPTOC

Fonction : *bscroll2(num_line)*

Effectue une translation de 1 pixel vers le bas de num_line lignes à partir de l'adresse de la mémoire vidéo sur une largeur de 240 pixels. Il peut être utile de modifier l'adresse de la mémoire vidéo actuelle pour obtenir l'effet voulu (fonction 'setlcd()').

NEWPROG2 & NPPTOC

6-Fonctions claviers

Fonction : *keywait()*

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

Equivalent C:ngetchx()

Fonction : *gkey()*

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

Codes des touches :

TI-89: Key	Normal	+Shift	+2nd	+Diamond	+alpha
Up	337	8529	4433	16721	33105
Right	344	8536	4440	16728	33112
Down	340	8532	4436	16724	33108
Left	338	8530	4434	16722	33106

TI-92+: Key	Normal	+Shift	+2nd	+Diamond	+alpha
Up	338	16722	4434	8530	33106
Right	340	16724	4436	8532	33108
Down	344	16728	4440	8536	33112
Left	337	16721	4433	8529	33105

Equivalent C :

```
long gkey(void)
 {
```

```
        unsigned short key;
        kbq = kbd_queue ();
        if (!OSdequeue (&key, kbq))
        {
        return key;
        }
 return 0;
}
```

Exemple :Appuyer sur la touche du haut et la touche '2nd' en même temps

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

Fonction : *gets(buffer)*

Récupère une chaîne de caractères saisie au clavier jusqu'à temps que la touche ENTER soit appuyée et la copie dans buffer. La touche espace est supportée. Renvoie buffer.

Equivalent C:gets(buffer)

Fonction : *keydelay(delay)*

Affecte le délai initial d'auto répétition des touches à la valeur delay pour les fonctions 'keywait()' et 'gkey()'. Le temps de mesure est de 1/395 s. La valeur par défaut est 336 (légèrement plus court que une seconde). La valeur minimale de delay est 3. Retourne le précédent délai.

Equivalent C : OSInitKeyInitDelay(delay)

Fonction : *keyspeed(rate)*

Affecte le taux d'auto répétition des touches à la valeur rate pour les fonctions 'keywait()' et 'gkey()'. Le temps de mesure est de 1/395 s. La valeur par défaut est 48. Retourne le précédent taux.

NEWPROG2 & NPPTOC

Equivalent C : OSInitBetweenKeyDelay(rate)

Fonction : *keytest(row,column_mask)*

'keytest()' est une fonction bas niveau pour savoir si une ou plusieurs touches sont appuyées simultanément. Pour les touches les plus courantes, il est plus simple d'utiliser les fonctions 'up()', 'down()', 'second()', etc... Cette fonction peut être utilisée si 'keyclear()' a été précédemment exécutée. Voir exemples pour fonctionnement.

TI-89:

Column

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	-

Equivalent C : _rowread_inverted(row)&_rowread_inverted(row)&column_mask

Exemple 1 : Teste si la touche CLEAR est pressée

Sur TI89 :

:keytest(0b10,0b1000000)

Sur TI92 :

:Keytest(0b10000000,0b100000)

Exemple 2 : Teste si la touche Left et Up ont été appuyées simultanément (pour une Ti89). Retourne un nombre non nul si au moins une des deux touches a été appuyée. Tester des touches sur la même colonn

:init()

:pause « Press Up and left simultaneously »

:while not esc()

:if keytest(1,0b11)=3 then //if you want to test the left key only (on ti89), write if keytest(1,0b10)
:prints(« Up and Left pressed»)

:endif

:endwhile

:pause « Finish »

Fonction : *keyclear()*

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

Equivalent C :

```
void keyclear(void)
 {
        if(save_int_1==NULL)
        {
                save_int_1 = GetIntVec (AUTO_INT_1);SetIntVec (AUTO_INT_1,
DUMMY_HANDLER);
        }
 }
```

Fonction : *keydisp()*

Réactive l'affichage de l'état des touches. A utiliser si vous avez exécutée 'keyclear()' précédemment.

Equivalent C :

```
void keydisp(void)
 {
        if(save_int_1)
        {
                SetIntVec (AUTO_INT_1, save_int_1);
                save_int_1=NULL;
        }
 }
```

Fonctions : *up() down() leftt() rightt() second() shiftt() diamond() alpha() esc()*

Teste si la touche correspondante haut, bas, gauche, droite, 2nd, majuscule, diamant ou alpha a été appuyée pendant le laps de temps que la fonction a été exécutée. La fonction ne stoppe pas l'exécution du programme.

NEWPROG2 & NPPTOC

7-Fonctions Tibasic

Fonction : *os(string)*

'os()' exécute string comme une commande Tibasic. Ne peut renvoyer qu'un entier ou un pointeur sur chaîne de caractères. Le retour de liste et de nombre flottant n'est pour l'instant pas implémenté (faire manuellement avec la fonction 'fopen()'). Attention, les données sont temporaires et doivent être sauvegardées pour ne pas les perdre. Cette fonction est lente (environ 33 appels par secondes). Cette fonction est intéressante pour exécuter des instructions Tibasics de façon dynamique ou pour simplifier l'écriture d'une séquence 'basic:endbasic'.

Exemple : Retourne xx^3, en affectant préalablement xx à la valeur 2

```
:init()
:clrscr()
:2→osvar(xx)
:printf1("xx^3=%ld »,os("xx^3"))
:keywait()
```

Fonction : *osvar(tibasic_var)* ou *osvar(direct_tibasic_var_str)*

Cette fonction peut être utilisée pour lire une variable Tibasic ou pour écrire dans une variable Tibasic. La variable Tibasic est de nom explicite tibasic_var ou de nom explicite écrit entre "" (chaîne de caractères directe direct_tibasic_var_str) et doit être composé d'au moins deux caractères (utiliser la fonction 'os()' dans le cas contraire) :

- Lecture : Renvoie la valeur d'une variable Tibasic. Cette fonction est plus rapide que la fonction 'os()' (1300 appels par secondes). 'osvar()' renverra un nombre si var_string est une expression, ou un pointeur sur chaîne de caractères si var_string est de type string. Dans le cas ou 'osvar()' retourne un pointeur sur chaîne de caractères, il peut être nécessaire de sauvegarder la chaîne de caractères dans un espace mémoire (fonctions 'strcpy()', 'newstr()', etc...) car le pointeur retourné pas 'osvar()' est temporaire. Les variables systèmes ne peuvent pas être récupérer avec cette fonction. Utiliser alors la fonction 'os()'.
- Ecriture : Combinée avec la notation '→' pour écrire dans une variable Tibasic (voir la fonction 'toos()' également). Un descriptif avec des informations complimentaires est présent dans la section Fonctions Mémoire ('→' ou 'sto()')

Exemple : Affiche la chaîne de caractères "Hello World !" et le nombre 666

```
:init()
:"Hello World !"→osvar(hellostr)
:666→osvar("num")
```

NEWPROG2 & NPPTOC

:text osvar("hellostr")
:text string(osvar(num))

Fonction : *toos(tibasic_var_str,entity)*

Cette fonction est similaire à l'écriture 'entity→osvar(tibasic_var_str)' (section Fonctions Mémoire). Cette fonction retourne 0 si l'espace mémoire vient à manquer. Pour rappel, le contenu de la variable créée est fonction de l'argument entity. Voici les cas possibles :

- Entity est un nombre (ou retour de n'importe quelle fonction), alors la variable crée sera du type EXPR.
- Entity est une chaînes de caractères écrite explicitement, alors la variable crée sera une chaîne de caractères de type STR.
- Entity est écrite de la forme #pointeur. 'pointeur' sera considéré comme un pointeur sur chaine de caractères et la variable créée sera du type STR et sera affectée au contenu de la chaîne de caractères pointée.

Fonction : *seqe(tibasic_var_dest_string,var,start,end,step,instruction)*

Créé une liste Tibasic composée de valeur numérique dont le nom est tibasic_var_dest (voir son homologue 'seqs()' pour la création d'une liste de chaînes de caractères). Le contenu de cette liste sera composé de (end-start)/step éléments. Le premier élément de cette liste sera la valeur retournée par l'exécution d'instruction pour une valeur de la variable var égale à start. Le deuxième élément sera égal au résultat pour var = var +step. Et ainsi de suite pour les éléments suivants. Cette fonction permet de créer des listes Tibasic à éléments variables. Retourne le nombre d'éléments de la liste. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Exemple : Affiche « {1,2,3,4,5} »
:init()
:seqe("x",vv,1,5,1,vv)
:basic
:clrio
:pause x

Fonction : *seqs*(tibasic_var_dest_string ,var,start,end,step,instruction)

Créé une liste Tibasic composée de chaines de caractères dont le nom est tibasic_var_dest (voir son homologue 'seqe()' pour la création d'une liste de valeurs numériques). Le contenu de cette liste sera composée de (end-start)/step éléments. Le premier élément de cette liste sera la chaîne de caractères renvoyée par l'exécution d'instruction pour une valeur de la variable var égale à start. Le deuxième élément sera le résultat pour var = var +step. Et ainsi de suite pour les éléments suivants. Retourne le nombre d'éléments de la liste. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

NEWPROG2 & NPPTOC

Exemple :Demande de choisir un répertoire dans un menu et affiche les noms des fichiers contenus et retourne le chemin de la sélection

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

Fonction : *fopen(filename_str)*

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

Fonction : *fcreate(data_ptr,file_dest_str)*

Copie dans un fichier le contenu d'un bloc mémoire. Vous pouvez ainsi créer des fichiers Tibasic de tous types. Voir Annexe pour comprendre les structures des fichiers Tibasic les plus communs. La taille (soustraite de 2) du bloc mémoire devra être enregistrée sur les deux premiers octets de ce bloc mémoire (grâce à la fonction 'wb()' par exemple). Retourne 1 si effectuer avec succès, aussi-non retourne 0.

Exemple : Créé la variable Tibasic nommée "hello" de type STR contenant "Hello World"

```
:init()
```

NEWPROG2 & NPPTOC

```
:malloc(20)→ptr                  //Créé un espace mémoire temporaire suffisamment grand
:"Hello world"→ss
:strcpy(ptr+3,ss)                //Copie le contenu de ss dans la zone mémoire adéquate
:wb(ptr,2+1+strlen(ss)+1,0h2d)   //Affecte le type (0h2d correspond au type STR)
:wb(ptr,2,0)                     //Inscrit '0' au troisième octet (obligatoire pour le type STR)
:ww(ptr,0,strlen(ss)+1+1+1)      //Inscrit la taille en tête
:fcreate(ptr, "hello")           //Créé le fichier
:free(ptr)
```

Fonction : *open(*{filetag1,filetag2,...,0}*)*

Ouvre une boite de dialogue Open. Ouvre une boîte de dialogue demandant de choisir un type de fichier parmi {filetag1, filetag2, etc}. Si l'utilisateur souhaite sélectionner des fichiers de types personnalisés, soit dérivés du type OTH (comme les fichiers NPP par exemple), il devra placer le type othtag comme élément de la liste suivi d'une chaîne de caractères contenant le nom du type personnalisé. Par exemple, si l'on souhaite afficher les fichiers de type NPP (extension des exécutables de Newprog) ainsi que les images (pictag), la commande sera la suivante : open({othtag,"NPP",pictag}) Une fois le type de fichier sélectionné, l'utilisateur peut sélectionner un répertoire parmi ceux disponibles sur la calculatrice. Au final, l'utilisateur sélectionne un fichier parmi ceux proposés, lesquels correspondant au répertoire et au type de fichier sélectionné précédemment. L'utilisateur valide son choix en appuyant sur ok. Cette fonction est limitée aux types suivants de fichiers : STR,LIST,TEXT,ASM,OTH (NPP, etc...) et PIC. Avant de lancer cette fonction, veillez à ce que les niveaux de gris ne soient pas actifs sous peine de crash. La fonction retourne une chaîne de caractères contenant le fichier sélectionné (valable jusqu'au prochain appel de la fonction). Si l'utilisateur à appuyé sur la touche ESC ou si le fichier n'existe pas, la fonction retourne une chaîne de caractères vide.

Exemple:Demande de sélectionner un fichier parmi les types TEXT et PIC et affiche son chemin

```
:init()
:open({texttag,pictag})→filestr
:text filestr
```

Fonction : *isarchi(var_str)*

Retourne 1 si le fichier identifié par la chaîne de caractères var_str est archivé, 0 si il n'est pas archivé. Renvoi -1 si le fichier n'existe pas.

Fonction : *archi(var_str)*

Archive un fichier. Similaire à la fonction Tibasic archive mais en prenant comme argument un nom de fichier sous forme de chaîne de caractères (var_str). Elle permet néanmoins une écriture plus concise. Retourne 1 en cas de succès. Retourne 0 si la mémoire archive vient à manquer.

NEWPROG2 & NPPTOC

Fonction : *unarchi(var_str)*

Désarchive un fichier. Similaire à la fonction Tibasic unarchiv mais en prenant comme argument le nom de fichier sous forme de chaîne de caractères (var_str). Elle permet néanmoins une écriture plus concise. Retourne 1 en cas de succès, 0 aussi-non.

Fonction : *files(dest_direct_var,rep_str)*

Cette fonction alloue un espace mémoire dont le pointeur sera enregistré dans la variable dest_direct_var et sauvegarde dans cet espace tous les noms de fichiers présents dans le répertoire rep_str. Affecte à 10 le type de dest_direct_var (voir les typages), ainsi les noms de fichiers sont espacés en mémoire de 10 octets entre eux. La notation [] est alors utilisable. Cette fonction retourne le nombre de noms de fichiers récupérés. Le nombre de noms de fichiers renvoyés est limité à 80. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Exemple 1 : Affiche tous les noms de fichiers présents dans MAIN

```
:init()
:files(filesl,"main")→nbfiles
:clrscr()
:map(vv,0,nbfiles-1,1,printf1("%s ",filesl[vv]))
:keywait()
:free(filesl)
```

Exemple 2 : Voir fonction 'seqs()'

Fonction : *reps(dest_direct_var)*

Cette fonction alloue un espace mémoire dont le pointeur sera enregistré dans la variable dest_direct_var et sauvegarde dans cet espace tous les noms de répertoires présents dans la calculatrice. Affecte à 10 le type de dest_direct_var (voir les typages), ainsi les noms de répertoires sont espacés en mémoire de 10 octets entre eux. La notation [] est alors utilisable. Cette fonction retourne le nombre de noms de répertoires récupérés. Le nombre de noms de répertoires renvoyés est limité à 50. Cette fonction retourne 0 si l'espace mémoire vient à manquer.

Exemple : Voir fonction 'seqs()'

NEWPROG2 & NPPTOC

8-Fonctions interruptions/timers

Fonction : *inter(*interupt_ID,tick,{instruction1,instruction2,…}*)* ou *inter(*interupt_ID,tick,x):instruction1,instruction2,...:y

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

Depuis Newprog 2.0, il est interdit d'utiliser des variables locales aux seins des instructions instruction1, instruction2... De plus, il est également à noter qu'un identifiant interupt_ID ne peut être associé qu'à une seule séquence d'instructions ; ainsi la redéfinition de sa séquence d'instructions est dorénavant interdite.

Exemple : Affiche une valeur incrémentée à vitesse variable grâce à une interruption. Le programme est quitté automatiquement au bout de 30 secondes grâce à une autre interruption.

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

NEWPROG2 & NPPTOC

```
:0 → time                                   //Initialise la valeur variable
:18→ rate                                   //Au départ, la valeur variable est incrémentée toutes les secondes
:0→ reached                                 //Variable de fin d'exécution au bout de 30 secondes
:printxy(0,92,"Tick = %ld ",rate)           //Affiche le Tick modifiable par les touches gauche et droite
:inter(1,rate,{printxy(10,70,"Moving value : %ld  ",isz(time))})     //Affiche la valeur variable
:inter(2,onesec*30,x)                       //Au bout de 30 secondes, quitte le programme automatiquement (onesec=18)
:1→ reached
:y
:While sto(kk,gkey())!=264 and not reached
:if second() Then
:inter(1,0,0)                               //Mets en pause l'affichage de la valeur variable
:EndIf
:if diamond() Then
:inter(1,rate,0)                            //Relance l'affichage de la valeur variable
:EndIf
:if kk=leftt and rate>1 Then                //Décrémente le Tick augmentant le tau de rafraîchissement de la valeur variable
:printxy(0,92,"Tick = %ld   ",dsz(rate))
:inter(1,rate,0)
:EndIf
:if kk=rightt Then                          //Incrémente le Tick diminuant le tau de rafraîchissement de la valeur variable
:printxy(0,92,"Tick = %ld   ",isz(rate))
:inter(1,rate,0)
:EndIf
:EndWhile
:clrscr()
:inter(0,0,0)                               //Mets en pause toutes les interruptions avant de quitter
:Pause "finished"
```

Fonction : *settimer(timer_no,T)*

Affecte un timer ayant l'identifiant timer_no, avec une valeur intiale égale à T. Une fois cette fonction exécutée, à chaque déclenchement de l'auto-int 5, c'est à dire environ 18 fois par seconde, la valeur du timer sera décrémentée de 1. La fonction 'timerval()' permet de récupérer la valeur du timer, tandis que la fonction 'timerexp()' permet de savoir si la valeur à atteint 0. Les identifiants des timers vont de 1 à 6 sur tous les AMS disponibles. Cette fonction partage les identifiants du système d'exploitation de la TI89 et c'est pourquoi il faudra sélectionner judicieusement quel identifiant utiliser. De manière certaine, les identifiant 1 et 6 sont libres d'utilisation, ils sont donc à privilégier. Le timer 4 est utilisé pour le clignottement du curseur, le timer 2 pour l'extinction automatique de la calculette, le timer 5 pour le délai de la fonction. Cyclepic. Il reste donc libres les identifiants 1,3 et 6. 'settimer()' retournera 0 si le timer est déjà utilisé ou s'il dépasse le nombre maximum de timers utilisables. L'exemple ci-dessous permet de modifier le délai avant l'extinction automatique de la calculatrice.

NEWPROG2 & NPPTOC

Exemple : La calculatrice s'éteindra au bout de 5 secondes. Ensuite passe ce délai à 100 secondes

```
:init()
:settimer(5*18)
:text "Will turn off in 5 sec"
:settimer(100*18)
```

Fonction : *freet(timer_no)*

Libère un timer actif ayant l'identifiant timer_no. Retourne 0 en cas d'erreur, 1 aussi-non.

Fonction : *timerexp(timer_no)*

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

Exemple : Attends 5 secondes avant d'afficher "Timer expired"

```
:init()
:settimer(1,5*18)
:nl():prints("Please wait")
:while not timerexp(1)
:printld(timerval(1)):nl()
:endwhile
:freet(1)
:text "Timer expired"
```

Fonction : *timerval(timer_no)*

Retourne la valeur du timer identifié par timer_no.

Exemple : Voir 'timerexp()'

NEWPROG2 & NPPTOC

9-Fonctions opérations binaires

Fonction : *lrol(expr1,expr2)*

Effectue un décalage sur la gauche de expr2 bits sur expr1. Expr2 doit être positif. Les nouveaux bits (sur la droite) seront mis à zéro

Equivalent C : expr1<<expr2

Fonction : *rrol(expr1,expr2)*

Effectue un décalage sur la droite de expr2 bits sur expr1. Expr2 doit être positif.

Equivalent C : expr1>>expr2

Fonction : *notb(expr)*

Effectue un non binaire sur la valeur expr et retourne le résultat.

Equivalent C : ~expr

Fonction : *andb(expr1,expr2)*

Effectue un et binaire entre expr1 et expr2 et retourne le résultat.

Equivalent C : expr1 & expr2

Fonction : *orb(expr1,expr2)*

Effectue un ou binaire entre expr1 et expr2 et retourne le résultat.

Equivalent C : expr1| expr2

Fonction : *xorb(expr1,expr2)*

Effectue un ou exclusif binaire entre expr1 et expr2 et retourne le résultat.

Equivalent C : expr1 ^ expr2

10-Fonctions opérations arithmétiques et logiques

Fonctions : < > ≤ ≥ *and or not* + - * /

Ce sont les mêmes opérateurs qu'en Tibasic.

Equivalent C :

<	<
>	>
≤	<=
≥	>=
and	&&
or	||
not	*!*

Fonction : *isz(direct_var)*

Incrémente de 1 la variable direct_var. Retourne la valeur incrémentée.

Equivalent C : ++direct_var

Fonction : *dsz(direct_var)*

Décrémente de 1 la variable direct_var. Retourne la valeur décrémentée.

Equivalent C : --direct_var

Fonction : *unsigb(data)*

Convertie un nombre signé codé sur 4 octets (data) en un nombre non signé codé sur un octet. Cette fonction trouve son intérêt avec les fonctions de lecture ('lb()' ou []) d'éléments de listes (listes de type 1 ou 'b()' soit l'équivalent C char[]) car ces fonctions retournent des nombres signés. Cette fonction permettra par exemple d'utiliser des listes d'éléments codés sur un octet pour stocker des abscisses de points à tracer à l'écran, sans être limité à une abscisse maximale de 127 (étant la

valeur maximale d'un nombre signé positif codé sur un octet). En effet, l'écran allant en abscisse jusqu'à 159 pixels pour une TI89 ou 239 pixels pour une TI92 ou V200.

Exemple : Affiche les valeurs d'une liste de type 1 (signed char) avec ou sans 'unsigb()'

```
init()
:b(list)
:1
:100
:200
:-1
:-100
:-200
:y
clrscr()
:printld(list[0]):nl()                  //Affiche 1
:printld(list[1]):nl()                  //Affiche 100
:printld(list[2]):nl()                  //Affiche -56
:printld(list[3]):nl()                  //Affiche -1
:printld(list[4]):nl()                  //Affiche -100
:printld(list[5]):nl()                  //Affiche 56
:keywait()
:clrscr()
:printld(unsigb(list[0])):nl()          //Affiche 1
:printld(unsigb(list[1])):nl()          //Affiche 100
:printld(unsigb(list[2])):nl()          //Affiche 200
:printld(unsigb(list[3])):nl()          //Affiche 255
:printld(unsigb(list[4])):nl()          //Affiche 156
:printld(unsigb(list[5])):nl()          //Affiche 56
:keywait()
```

Fonction : *unsigw(data)*

Convertie un nombre signé codé sur 4 octets (data) en un nombre non signé codé sur deux octets. Cette fonction trouve son intérêt avec les fonctions de lecture ('lw()' ou []) d'éléments de listes (listes de type 2 ou 'w()' soit l'équivalent C int[]) car ces fonctions retournent des nombres signés.

NEWPROG2 & NPPTOC

11-Fonctions diverses

Fonction : *rand(num)*

Retourne un nombre aléatoire compris entre 0 et num-1

Fonction : *off()*

Éteint la calculatrice.

Fonction : wait*(delay)*

Tourne en boucle pendant delay centièmes de seconde.

Fonction : *catalog()*

Affiche le menu catalog. Retourne une string de la sélection. Ce pointeur sera valable jusqu'au prochain appel de la fonction.

Fonction : *debugon()*

Ne pas utiliser avec NPPTOC.

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

Fonction : *debugoff()*

Ne pas utiliser avec NPPTOC.

Désactive le mode d'exécution pas à pas (voir 'debugon()'). Retourne en mode d'exécution normal.

Fonction : *nop()*

Ne fait rien.

NEWPROG2 & NPPTOC

12-Fonctions librairies

Pour plus de détails sur la notion de librairies, voir la section correspondante (« Les librairies »).

Fonction : *loadlib(lib_name_str)*

Charge en mémoire la librairie dont le nom lib_name_str est sous la forme d'une chaîne de caractères. Retourne une référence qui sera utile par la suite pour exécuter ses fonctions internes et pour fermer cette librairie. Si il manque de la mémoire, affichera une erreur.

Fonction : *exelib**X**(lib_ref,func_id[,arg1,...,arg**X**])* - **X** *allant de 0 à 7*

Exécute la fonction numéro func_id de la librairie référencée par lib_ref (voir 'loadlib()') et retourne le cas échéant une valeur. La valeur **X** correspond au nombre d'arguments à passer à la fonction de la librairie :

X=0 : Pas d'argument à passer à la fonction interne de la librairie. On a alors juste exelib0(lib_ref,func_id).

X=1 : Un seul argument à passer à la fonction interne de la librairie. On a alors exelib1(lib_ref,func_id,arg1)

…

X=7 : Sept arguments à passer. On a alors exelib7(lib_ref,func_id,arg1,arg2,arg3,arg4,arg5,arg6,arg7)

Fonction : *closelib(lib_ref)*

Ferme la librairie dont la référence est lib_ref.

NEWPROG2 & NPPTOC

E-Les librairies

1) Introduction

Une librairie est un regroupement de fonctions stockées uniquement dans fichier .ASM-.89z (code machine). Elles peuvent être appelées depuis n'importe quel programme exécutable Newprog (bytecode ou code machine). Il est possible de leurs passer jusqu'à 7 arguments.

Elles peuvent être créées depuis un :

- code source Newprog directement sur la calculatrice : des instructions particulières les différencieront d'un code Newprog classique.
- code source C pur dans l'IDE de GCC4TI : cette solution permet d'utiliser toutes les fonctions/possibilités du C.

Les librairies doivent obligatoirement être compilées en code machine pour pour pouvoir être utilisées depuis un exécutable Newprog.

L'utilisateur peut également créer ses propres librairies, à partir d'un programme Newprog adapté ou à partir d'un programme C également adapté. Voir plus bas.

Remarque : La librairie 'npplib2.ASM-89z' est elle même une librairie Newprog. Son utilisation est transparente à l'utilisateur.

2) Généralités : Chargement – Appels de fonctions – Fermeture dans un programme Newprog

L'utilisation d'une librairie Newprog dans un programme Newprog passe par trois étapes successives : son chargement, l'utilisation de ses fonctions et sa fermeture.

- Chargement :

Pour pouvoir appeler les fonctions d'une librairie Newprog au sein d'un programme Newprog, il faut au préalable l'avoir chargée en mémoire à l'aide de la fonction 'loadlib()'.

Remarque : Le chargement se chargera également en interne de préparer l'utilisation de cette librairie en réservant un espace mémoire contenant entre autres les valeurs des arguments et les pointeurs des fonctions référencées. Le pointeur de cet espace mémoire est renvoyé par la fonction 'loadlib()'.

- Appel d'une fonction de librairie Newprog :

Pour exécuter une fonction, utiliser la fonction 'exelibX()' où 'X' va de 0 à 7. 'X' correspond au nombre d'argument passée à la fonction de la librairie (et non pas à la fonction 'exelibX()').

Remarque : L'appel d'une fonction de librairie est l'équivalent en C de l'appel d'une fonction externe par l'intermédiaire de son pointeur en mémoire.

- Fermeture :

On décharge la librairie de la mémoire (si archivée) Pour fermer une librairie, utiliser la fonction 'closelib()'.

Remarque : Désalloue également l'espace mémoire créé pendant le chargement.

>>Voir les sections correspondantes dans la liste des fonctions pour les détails de leurs utilisations. Un exemple est donné dans la section suivante.

3) Création de librairies personnalisées à partir d'un code Newprog

La création/définition de librairies à partir d'un code Newprog passe par l'utilisation de mots clés/instructions. Ils permettent :

- d'informer le compilateur que le code Newprog définit une librairie >> instruction 'aslib()' placée juste après la commande initiale 'init()'. Cette ligne est obligatoire.
- d'informer le compilateur que l'on souhaite récupérer un argument pour une fonction librarie donnée >> instruction 'arg(X)' où 'X' est le numéro de l'argument (1 à 7). Au besoin.
- d'informer le compilateur que l'on retourne une valeur >> 'result(value)'. Au besoin.
- d'informer le compilateur quelles sont les fonctions qui seront accessibles depuis l'extérieur >> instruction 'libfuncs({fa,fb,...})'. Cette ligne déclarative est obligatoire.

  L'indice de la première fonction déclarée sera affectée à l'indice 0, la deuxième à l'indice 1 et ainsi de suite. Pour appeler une fonction extérieure depuis un programme Newprog, on passera l'indice correspondant dans 'exelibX()'.
- de tester préalablement à la compilation en code machine les fonctions accessibles depuis l'exterieur >> instruction 'testfc()'. Cette étape est facultative mais conseillée.

La compilation en code machine d'une librairie Newprog suit les même étapes que pour tout autre programme Newprog.

Exemple de définition d'une librairie utilisateur Newprog :

```
:init()
:aslib()                        //obligatoire >> Indique que l'on définit une librairie

:"Next Year, i will be :"→str  //Définit une string globale utile pour cet exemple

:def(fc0(str1,str2))            //Définit la fonction interne 'fc0()' qui concatène deux strings
:settype(str1,1)
:settype(str2,1)
:rtn(str1&str2)
:enddef(fc0())

:def(hello())                                   //Définit la fonction extérieure 'hello()'
:text fc0("Hello ","World!")
:enddef(hello())
```

```
:def(myage())                                  //Définit la fonction extérieure 'myage()'
:text "I'm "&string(arg(1))&" years old."      //récupère et utilise l'argument d'indice 1
:result(str&string(arg(1)+1))                  //idem et retourne une concaténation de strings
:enddef(myage())

:libfuncs({hello(),myage()})          //Déclaration des fonctions extérieures : hello() et myage()
:testfc(hello(),{})                   //Lors du test, affiche "Hello World!"
:testfc(myage(),{20})          //Lors du test, affiche "I'm 20 years old.". Retourne "Next Year, i will be :21"
```

Une fois la librairie compilée en code machine et nommée pour l'exemple 'mylib', on pourra par exemple utiliser la librairie dans un programme Newprog de la manière suivante :

```
:init()
:loadlib("mylib")→lib      //Charge la librairie 'mylib'. La référence est affectée dans 'lib'
:exelib0(lib,0)            //Exécute la fonction extérieure d'indice 0 (pas de passage d'argument)
:text exelib1(lib,1,20)    //Exécute la fonction extérieure d'indice 1 (passage de la valeur 20 au premier argument)
:closelib(lib)             //Ferme la librairie 'mylib'
```

Sera affiché à l'écran à l'exécution :

"Hello World !" puis "I'm 20 years old." et "Next Year, i will be :21"

4) Création de librairies personnalisées à partir d'un code C pur

La définition d'une librairie Newprog en C est basée sur le modèle du fichier 'clibsrc.c' présent dans le bundle de Newprog 2.0. Son contenu est répété plus bas. Une librairie peut aussi bien être créer sur GTC oncalc que sur GCC4TI sur PC.

Dans tous les cas, les apports du programmeur sont cantonnés dans la zone en gras comme dans l'exemple plus bas, soit entre :

//WRITE YOUR OWN DATAS HERE BELOW//

et

//END OF YOUR DATAS//

Les différentes rubriques à complétées sont :

- **//WRITE YOUR OWN FUNCTIONS BODYS BELOW :**
  - Placer ici le corps de vos fonctions extérieures (et éventuellement internes si besoin). Le type de retour des fonctions extérieures est obligatoirement de type void func_name(void).
  - Si l'utilisateur souhaite récupérer la valeur d'un argument d'indice 'X' (allant de 1 à 7), utiliser la syntaxe 'ARGX'. Si l'utilisateur souhaite qu'une fonction extérieure retourne une valeur 'value', utiliser la syntaxe 'RETURN value '. Dans ces deux cas, il faudra obligatoirement placer la sentence 'FC_HEADER' au tout début du corps de la fonction.
- **//ADD TO THE LIST HERE BELOW THE NAMES OF YOUR FUNCTIONS :**

- Déclarer ici successivement les fonctions extérieures dans la liste nommée 'func'. La position de déclaration d'une fonction extérieure dans cette liste corresponds au numéro d'indice utilisé dans l'appel de 'exelib*()'. Le premier élément aura l'indice 0, le deuxième l'indice 1 et ainsi de suite.

- **//WRITE YOUR OWN INTERNAL DATAS BELOW :**
  - Déclarer ici les éventuels données/buffers internes qui pourront être utilisées par exemple dans les fonctions internes. Voir exemple ci-dessous.

Exemple de définition en C d'une librairie extérieure :

```
#define MIN_AMS 200              // Compile for AMS 2.00 or higher
#define ARG1 data_ptr[0]         //retrieve first argument passed to the lib
#define ARG2 data_ptr[1]         //and so on
#define ARG3 data_ptr[2]
#define ARG4 data_ptr[3]
#define ARG5 data_ptr[4]
#define ARG6 data_ptr[5]
#define ARG7 data_ptr[6]
#define ARG8 data_ptr[7]
#define RETURN data_ptr[8]=      //for returning a value

#define ANSWER data_ptr[8]       //do not use
#define FC_LIST data_ptr[9]      //do not use
#define EXECHDL2 data_ptr[10]    //do not use
#define COPIEHDL2 data_ptr[11]   //do not use

#define FC_HEADER long *data_ptr; data_ptr=*(long*)0x5B04;      //put FC_HEADER statement at the beginning of your LIB function if your if you use arguments ARG* or RETURN keywords.

#include <tigcclib.h>

//WRITE YOUR OWN DATAS HERE BELOW

//WRITE YOUR OWN INTERNAL DATAS BELOW
char *hello_str="Hello World !";          //Chaine de caractères (string) qui sera utilisée en interne

//WRITE YOUR OWN FUNCTIONS BODYS BELOW
long add(long a1, long a2)                //fonction interne utilisée dans add_fc()
{
        return a1+a2;
}

void add_fc(void)                         //fonction exterieure
{
```

NEWPROG2 & NPPTOC

```
	FC_HEADER    //obligatoire si on lit les arguments passés à la fonction ou si on retourne une valeur
	clrscr();
	printf("\nARG1+ARG2=%ld",add(ARG1,ARG2));		//affiche la somme
	ngetchx();
	RETURN ARG1+ARG2;		//retourne la somme de l'argument 1 et de l'argument 2
}

void hello_fc(void)			//fonction extérieure
{
	clrscr();
	printf("%s",hello_str);
	ngetchx();
}
//ADD TO THE LIST HERE BELOW THE NAMES OF YOUR FUNCTIONS
void (*func[])(void)={add_fc,hello_fc};	//Définition des fonctions extérieures, indices 0 et 1


//END OF YOUR DATAS//

// Fonction main(), utiliser comme telle
void _main(void)
 {
	unsigned int ii;
	long *data_ptr;

	if(*(long*)0x5B04!=123456)
	{
		clrscr();
		printf("\nNot intended to be launched !");
		ngetchx();
		return;
	}

	data_ptr=malloc(12*4);

	if(data_ptr==0)
	{
		printf("\nNot enough memory when launching LIB.");
		ngetchx();
		*(long*)0x5B04=0;
		return;
	}

	FC_LIST=func;
	*(long*)0x5B04=data_ptr;

}
```

NEWPROG2 & NPPTOC

Une fois la librairie compilée en code machine et nommée pour l'exemple 'mylib', on pourra par exemple utiliser la librairie dans un programme Newprog de la manière suivante :

```
:init()
:loadlib("mylib")→lib                    //Charge la librairie 'mylib'. La référence est affectée dans 'lib'
:text "Somme="&string(exelib2(lib,0,10,2)) //Exécute la fonction extérieure d'indice 0 (argument1=10, argument2=2)
:exelib0(lib,1)                          //Exécute la fonction extérieure d'indice 1 (pas de passage d'argument)
:closelib(lib)                           //Ferme la librairie 'mylib'
```

Sera affiché à l'écran à l'exécution :
"ARG1+ARG2=12" puis "Somme=12" puis "Hello World !".

F-Rapports d'avertissements/d'erreurs à la compilation

- Compilation Newprog en bytecode :

Lorsqu'une erreur de compilation est détectée, le compilateur renvoie un fichier de sortie (npout) décrivant l'erreur et donne des indices quant à sa localisation dans le programme (dernière fonction et variable traitée). Lorsqu'une erreur est détectée lors de l'exécution d'un programme, il est conseillé de réinitialiser la calculette (reset) afin de résoudre les problèmes de fuites de mémoire.

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

- Traduction en langage C avec NPPTOC :

Une erreur peut apparaître pendant la traduction en langage C. Les erreurs les plus communes sont celles relatives au fonctions interdites avec NPPTOC (mais pas en bytecode) et celles relatives au typage des variables.

- Compilation en code machine :

Avec GTC oncalc, si un avertissement (warning) apparaît, vous pouvez tout de même continuer la compilation en appuyant sur ENTER. Cela ne porte pas à conséquence.

Avec tout type de compilateur, si une erreur apparaît à la compilation (dans de rares cas), il s’agit d'un problème de conversion en langage C. Si vous disposez de bonnes connaissances en C, vous pourrez éventuellement repérer et corriger l'erreur.

NEWPROG2 & NPPTOC

G-Constantes prédéfinies

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

Listing des constantes prédéfinies :

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

TI89 seulement :

"up" 337
"down" 340
"rightt" 344
"leftt" 338
"shiftt" 8192
"second" 4096
"diamond" 16384
"alpha" 32768
"esc" 264

TI92/V200 seulement :

"up2" 338
"down2" 344
"rightt2" 340
"leftt2" 337
"shiftt2" 16384
"second2" 4096
"diamond2" 8192
"alpha2" 32768

Exemple : Quitte le programme si les touches haut et 2nd sont pressées simultanément

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

NEWPROG2 & NPPTOC

H-Remerciements

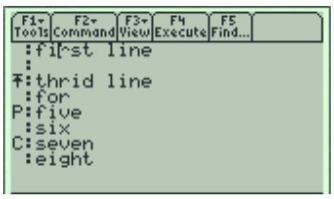
Je tiens à remercier tous ceux qui ont contribuer à la création de GCC4TI et GTC. Je remercie également la TICT pour leur formidable librairie graphique EXTGRAPH.

NEWPROG2 & NPPTOC

Annexe 1 – Structures internes des fichiers courants sur TI68k

Cette annexe décrit (en anglais) la structure interne des différents types de fichiers les plus utilisés sur TI68k. Vous pourrez par vous même vérifier ces informations en utilisant l'éditeur hexadécimal fournit avec le bundle (mtihex.89z). Pour créer des fichiers, les fonctions mémoires sont très utiles (et particulièrement la fonction 'fcreate()').

TEXT files example



Ti89 text editor

Hexadecimal editor

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

OTH files example

From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h0025 (16) = 37 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 37 + 2 = 39 bytes)
2)Data bytes (put what you want)
3) Optional sequence : Specify a personalize type for the file :
- 0h00
- Name of the type. For example {N,P,P}
- 0h00

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

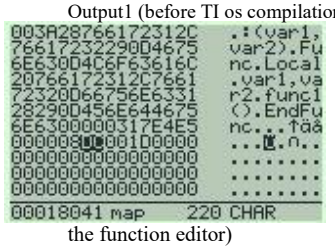
ASM files example

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

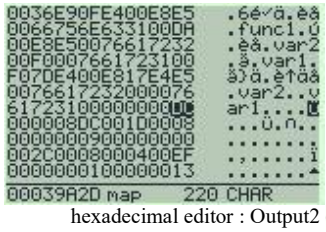
FUNC and PRGM files example

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

TI89 Func editor (example for a function called ff(), replace Func and Endfunc by Prgm and Endpgrm for a program) hexadecimal editor : Output1 (before TI os compilation (or before launching the func if you have just before edited the function in



the function editor)



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

Output 1 struct (before compilation)

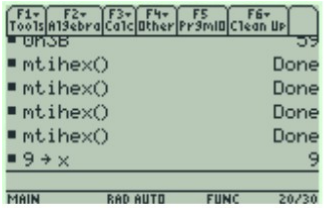
From the byte of the lower adress to the file to the upper address :
1)Size of the file : for this example 0h003A (16) = 58 (10) (in reality, the real size of the file, like you can see in the TI Var-link, is 58 + 2 = 60 bytes)
2)Line struct :
2-1) Line content (can have no byte for an empty line)
2-2)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

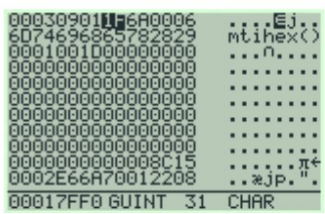
Output 2 struct (after compilation (ie after running))

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

EXPR files example

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

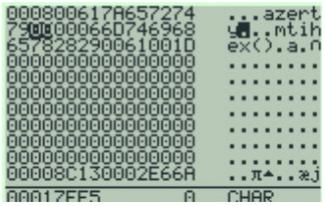




For the variable x = 9. Hexadecimal editor

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

For the variable x = azerty.



Hexadecimal editor

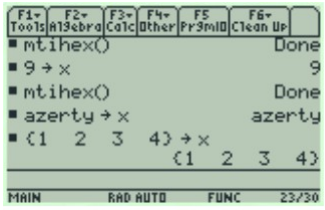
For this data type, if you want to read the value contained in this file, you have to read from the last byte of the file (the one who have the higher address to the one with the lower address) to the lower address of the file.

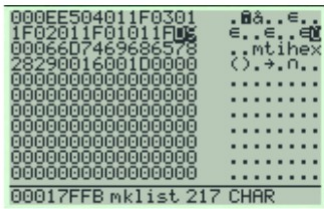
To determine the position of the last byte, you will have to addition the first byte address of the file with the size of the file -1.
The size of the file is equal to the content of the first two bytes coded value of the file, in this example +2. For this example, the size of the file is size = 0h00003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

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

LIST file example

A list contained a multiple of EXPR.



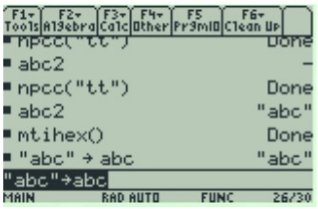


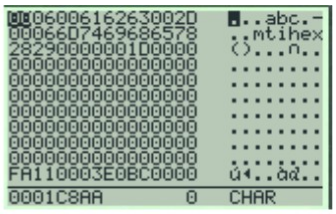
1) The first two bytes content is equal to the size of the file -2.
2) 0hE5 = endtag (specify the end of the list)
3) Put EXPR ; STRING or LIST sequences (In the same way than in this document, but wihout the two byte of the size.
4) The last byte of the file is equal to 0hD9 = 217.

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

STR file example

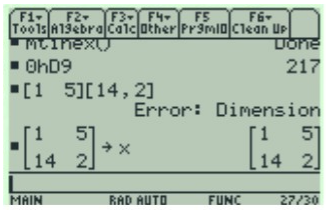
1) First two bytes = size of the file – 2 (size of the file must be equal to the size you will see in the varlink)
2) 0h00
3) Put your string with the end null character
4) **STR_TAG = 0h2D**

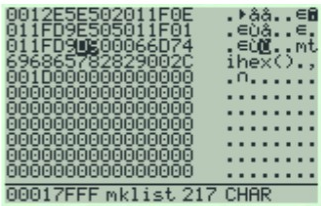




MATRIX file example

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





PIC file example

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

Annexe2 – Historique des versions

2021 : Newprog Release 2.0 – Improvements since release V1.0

Réécriture de la documentation (plus détaillée, plus claire, plus précise, corrections de fautes d'orthograhes)
Ajout de la notion de librairies
Ajout du mode de compilation en langage machine (NPPTOC)
Nombreuses corrections de bugs.
Ajout des fonctions :

- loadlib()
- exelibX()
- closelib()
- wait()

Modifications des fonctions :

- seqb_w_l_e_s() : New syntax : seqb_w_l(dest_var_ptr,moving_var,start,end,step,expr).
- seqe_s() : New syntax : seqe_s(basic_dest_var_str,moving_var,start,end,step,expr).
- Files : New syntax files(dest_var,rep_str)
- inter()

Suppression des fonctions inutiles:

- loadasm()
- execasm()
- closeasm()
- multi()
- two()
- Basich()
- execbas()
- freeh()
- pos()
- gotopos()
- jsr()
- rts()

Nombre de fonctions utilisateurs limitées à 30 dans le corps d'un programme.

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

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

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

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

ifs() renamed by when().
chars() renamed by char(). The return value is now available in memory up to the next call of this function (you don't have to free the return value before the end of the program).
code() renamed by ord().
int1off() renamed by keyclear().
int1on() renamed by keydisp().

fopen(), great modifications : The function is totaly modified.
or , and and not functions work now has logic test and not test on binary operations (see new functions orb(), andb() , notb() for binary operations).
While:Endwhile : a bug has been corrected (when two endwhile were closed).
gmode : graphic mode value has been change for real compatibility between all graphics functions.
keywait() : a bug is corrected (when using with keyclear() (previous name of keyclear : int1off()).

New Functions :
= for strings comparaison (simplier to use than strcmp()). Return 1 if equal, else 0.
andl : binary and operation (former and operator in newprog release v0.1).
orl : binary or operation (former or operator in newprog release v0.1).
sprt82 : display a sprite even if in the border of the screen.
sprt162 : display a sprite even if in the border of the screen.
sprt322 : display a sprite even if in the border of the screen.
newline or nl : Return to a new line (with the printf1... functions).
when : replaced the ifs functions in release 0.1
i : is a simple writing for if endif function (with no else for the moment). As fast as an if endif statement. Use the following syntax : i(condition):instructions:y
catalog : Launch the catalog dialog and return a string of the
selection. For:Endfor
open : display an open dialog box. The user select the folder and the file (and can choose the type). Returns a string :"rep\varname"
mod : modulo function. Example : mod(9,7) returns 2.
atol : convert a string to number (like expr in tibasic)
& : similar to tibasic. The output shall be smaller than 50 bytes for avoiding crash. If the size is bigger than 50 bytes, use strcat() instead.
lcdup : contrast up.
lcddown : constrast down.
b,w,l : 3 functions to define predifined instantiated list.
prettyxy : display an expression into pretty print format (very impressive).
getwbt : retrieve the dimension of the expression to be displayed with prettyxy.
settype : set the type of data (ie their size) contained in a list pointed by a Newprog variable. Returns the old value. The function could just returns the old value without modifying it.moveto : Sets the current pen position.
pause : Display a string to the screen and wait for a key to be pressed
seqs : Great function for making Tibasic list filled with strings
seqe : Great function for making Tibasic list filled with expressions (ie number only)
orb : binary or
andb : binary and
notb : binary not
fopen(file_str) : open an existing file for reading and modificating. If you want to save your modifications, you have to use the fcreate function.
fcreate(file_str, string_type) : create a file or overwrite a file
next : returns to the beginning of the last opened and active For or While loop.
break : Ends the last opened For or While loop.
esc : acts like up() down() etc but for escape key.
off : turns off the calculator
isarchi : Return 1 if file is archived, 0 if not, -1 if doesn't exist.
archi : archive a file.
unarchi : unarchi a file.
gsprt8x : retrieve from the screen a sprite with custom bytewidth.
sprt8x : Display to the screen the sprite (in particular from gsprt8x).
dcircle : Draw an outlined circle.
fillcirc : Draw a filled clipped circle.
memchr : Search for a character in a memory area.
text : Same function than the Tibasic one.
unsigb : converts a number coded onto 4 bytes (data) on a signed number coded onto one byte
unsigw : converts a number coded onto 4 bytes (data) on a signed number coded onto two bytes
osvar : Return the value of a Tibasic variable
if : if cond:if_true
ord : converts a string of one character to a code ascii

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

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

<u>In the compilator point of view :</u>

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

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

Mains Newprog features defined.