Prise en main rapide: le CELTIC III

Ou comment pousser plus loin la programmation en TI Basic pour TI 83 Plus et TI 84 Plus (Silver Edition)

Créateur : Iambian Zenith (Rodger Weisman)

Créateur de ce tutorial: Persalteas pour Espace TI:Forum

Lien de téléchargement de l'archive : http://www.omnimaga.org/index.php?action=downloads;sa=downfile&id=479

Introduction :

Bonjour à tous, je fais un deuxième tutoriel sur une librairie pour TI 8x , le Celtic (version 3).
Vous allez voir que le Celtic permet de nombreuses choses que le TI-Basic pur ne permet pas.

-Qu'est-ce qu'une librairie ?

C'est une application, ou un programme, qui permet de rajouter des fonctions au TI Basic normal.

-Pourquoi Celtic III et pas une autre librairie ?

Parce que c'est une de celles qui propose le plus d'options.
Elle permet l'accès aux AppVars, au Groupes, et la manipulation des programmes eux-mêmes.

-Qu'ai-je besoin de savoir-faire pour apprendre le Celtic III ?

Je considère que vous savez déjà programmer en TI Basic pour TI 83+ ou 84+, et que vous avez également appris le Xlib ou au moins lu mon mini-tuto sur le site.
je considère que vous avez déjà jeté un œil sur des programmes en assembleur (ASM), même si vous ne savez pas les programmer (ce qui est mon cas). Je vais évoquer plusieurs fois ION ou MirageOS.
Je considère également que vous savez envoyer une application à votre TI.
Si ce n'est pas le cas, demandez de l'aide sur le forum.

-Comment ma TI va-t-elle comprendre quand j'exécute une fonction Celtic, si le programme n'est pas compilé ?

Chaque commande du Celtic va commencer par det(, identity(, ou real(.

real(se trouve dans les menus de la TI en faisant **[MATH][>][>][2]**
identity(se trouve dans les menus de la TI en faisant **[2nd][x-1][>][5]**
det(se trouve dans les menus de la TI en faisant **[2nd][x-1][>][ENTER]**

Je précise dès le début : **l'utilisation de la commande real(est exactement la même qu'en Xlib** (voir mon mini-tuto).

Alors…

Définitions générales :

- La fonction real(va précéder toutes les instructions du Xlib.
- La fonction identity(va précéder toutes les instructions <compléter>
- La fonction det(va précéder toutes les instructions Celtic qui restent (celles de base, en fait).

- quand on aura besoin d'accéder à un programme, on aura besoin de préciser son nom dans une chaine : "NOMDUPROG"
- quand on aura besoin d'accéder à une AppVar, on aura besoin de préciser son nom dans une chaine précédé de la **fonction rowSwap(SANS REFERMER LA PARENTHESE**: "rowSwap(NOMDELAPP"
- quand on aura besoin d'accéder à un groupe, on aura besoin de préciser son nom dans une chaine précédé de la **fonction *row(SANS REFERMER LA PARENTHESE** : "*row(NOMDUGROUPE"
- De manière générale, ne lancez pas de programme utilisant Celtic III ou même Xlib **à partir d'un shell comme MirageOS ou ION**. Ces programmes utilisent les mêmes aires de la mémoire qu'utilise Xlib et Celtic.
- Dans ce tutoriel (comme dans le Readme en Anglais fourni avec Celtic), je vais vous montrer comment fonctionnent les instructions selon ce schéma:

  fonction(numéro , argument 1 , argument 2 , …

  Il est possible que j'encadre parfois un argument avec des crochets [argument] , cela signifie que l'argument est facultatif. On n'est donc pas obligé de le mettre, la TI comprendra automatiquement qu'il est de 0 (pour les commandes det et identity) ou de 32 (pour les commandes real).
- Quand je dis qu'une fonction "retourne" un nombre, cela veut dire que Ans a pris la valeur de ce nombre juste après qu'on ait exécuté la fonction.
- Si vous voyez des balises <à préciser> dans ce tuto, c'est que je ne maitrise pas totalement ce dont on parle, et que je dois tester un ou deux trucs. Normalement, si vous lisez ce tuto, j'ai dû les enlever.

I. Manipuler un Programme et ses propriétés.

-qu'est-ce que c'est, les propriétés d'un programme ?

Il s'agit de savoir si il est archivé ou non, protégé contre l'édition, ou même caché

(C'est-à-dire qu'il n'apparait pas dans la liste de programmes)

Nous allons utiliser la fonction

det(0 , "NOMDUPROG" , fonction

fonction: il vous faut ici entrer un chiffre, qui va dire à la fonction ce qu'elle doit faire:

0 = Archiver/désarchiver le programme. Retourne 0 si c'est réussi. (je vous rappelle qu'au lieu de mettre 0, vous pouvez ne rien mettre, pour économiser des octets ;) …)

1 = Protéger/déprotéger le programme contre l'édition (apparait ou non dans le menu EDIT). Retourne 1 si le fichier est protégé, 0 s'il n'est plus protégé, 2 si vous avez essayé avec une AppVar.

2 = Envoyer le programme dans une AppVar, ou inversement, une AppVar dans un programme <à préciser>

3 = Cache/fait réapparaitre le programme. Retourne 0 si le programme est visible, 1 si il est caché.

4 = Supprime le programme. Action irréversible, attention.

5 = Crée un nouveau programme avec le nom que vous avez entré. Si ce programme existait déjà, ça ne marche pas. Le programme sera vide, mais vous pourrez le remplir.

6 = Copie le code du programme dans Ans, sous forme de chaine <à préciser>

7 = Affiche l'état du programme: Plus précisément, Ans prend la valeur d'une chaine contenant trois lettres suivies de 5 chiffres. Les 5 chiffres sont la taille du programme en octets, les trois lettres:

1ère letter: A= archivé, R= dans la RAM

2ème lettre: H= hidden, V= visible

3ème lettre: W= writable, L= locked

NB: si vous voulez juste savoir la taille du programme, faites donc:

```
:expr(sub(det(Ø,"NOMDUPROG",7),4,5
```

cela vous renverra dans Ans la valeur de la sous-chaine du 4e caractère pendant 5 caractères de la chaine obtenue. ;)

II. Manipuler une variable réelle

je vais vous donner en vrac les astuces, car il n'y a pas vraiment de continuité la dedans, et rien de difficile.

En fait, je vous les donne par ordre croissant de det(.

Comment copier une variable quelle qu'elle soit dans une chaine:

Ça marche pour les nombres réels (autrement dit les "lettres"), les listes et les matrices.

det(1 , variable

Ans sera égal à une chaine qui contiendra la valeur de la variable. Combien de fois avez-vous rêvé en TI Basic de pouvoir faire A→Str1 ? ça va mieux, hein, comme ça !

De plus, cette fonction vous permet de stocker des nombres plus légèrement (9 au lieu de 18 octets) et avec une précision bien plus grande (les 14 décimales peuvent être dépassées !)

Comment lire une liste archivée :

Il y a une fonction qui permet de lire une liste archivée.

det(2 , "liste" , [numéro]

C'est l'équivalent du L1 (numéro) en TI Basic, mais pour une liste archivée. Ceci dit, ça marche parfaitement pour une liste dans la RAM, mais c'est plus court d'écrire ʟSTATS(2) que det(2," ʟSTATS",2).

Vous pouvez utiliser les listes de L1 à L6, ainsi que les listes à nom personnel, n'oubliez pas de mettre le nom entre guillemets.

Vous voyez que l'argument numéro est facultatif. Si vous ne le mettez pas (ou que vous mettez 0 à la place), l'instruction retourne la dimension de la liste.

Contrôler la nature d'une variable:

Vous devez être conscient qu'en triturant les octets comme ça, on risque les erreurs. Pour préserver la variable Ans, et pour être sûr de savoir ce qu'il y a dans chaque variable, on peut utiliser la commande:

det(3, variable

Selon la valeur de Ans, on peut dire ce que contient la variable.

0 = nombre réel	3 = Equation	13 = liste de nombres complexes
1 = liste	4 = Chaine	
2 = Matrice	12 = nombre complexe	

III. utilitaires système

Ici, je vais vous donner les fonctions permettant de connaitre des informations sur l'état actuel de votre TI.

det(4, fonction

fonction: 0 = retourne la valeur de la RAM libre. (Je rappelle à tout hasard que rien mettre, c'est comme mettre 0.)

1 = retourne la valeur de la mémoire d'archive libre (ou ROM libre)

2 = retourne dans une chaine les 10 premiers chiffres du numéro d'ID de la TI.

3 = retourne dans une chaine la version de l'OS (1.19 pour les 83+ par exemple)

(utile par exemple pour les 84+ : si vous avez l'OS 2.55MP (avec les fonctions de mathématiques joliment présentées), Celtic III et Xlib ne sont pas compatibles avec votre TI.)

IV. Trafic de code source

Ces fonctions vous permettent de lire et d'**écrire** dans les programmes via un programme. C'est fort, quand même.

Lire une partie de programme seulement <à préciser> :

Ça, c'est fort, quand même. Fini l'usage du Lbl-Goto à toutes les sauces, on peut demander au programme de lire le programme TRUC à partir de la 8e ligne pendant 7 lignes par exemple.

det(5, "NOMDUPROG", numéro ligne de départ , [nombre de lignes en plus]

NOMDUPROG est donc bien sur le programme qui contient la portion de code que vous voulez lire.

Numéro de ligne de départ: là où le programme commence à lire. Si vous mettez 0, la fonction retourne le nombre de lignes dans le programme.

Nombres de lignes en plus: si vous voulez que le programme continue à lire les lignes après celle demandée. Si vous ne mettez rien ou zéro, il lira juste la ligne indiquée "de début".

Écrire du code dans un programme:

Cette fois ci, vous aller pouvoir inclure dans le programme source du code (ou des données) que vous aviez enregistré dans une chaine (ou dans Ans).

det (6 , "NOMDUPROG" , "chaine" , numéro de ligne

Comme vous l'avez tous deviné, NOMDUPROG est le nom du programme de destination. Attention, ce programme **DOIT DEJA EXISTER.** La fonction ne peut pas créer de nouveau programme.

"chaine" le nom de la chaine qui contient le code à inclure.

les cas les plus fréquents sont "Ans" ou "Str1" par exemple.

numéro de ligne: le numéro de la ligne du programme ou la TI doit "coller" la portion de code.

Supprimer des lignes de code:

Cette fois, si on veut bousiller une partie de notre code, au lieu de l'exécuter:

det(7 , "NOMDUPROG" , numéro de ligne , [nombre de lignes supplémentaires]

Cette fonction supprimera le code source du programme NOMDUPROG à partir de la ligne indiquée. On peut demander à la TI de supprimer en plus quelques lignes suivant celle indiquée avec le dernier argument, facultatif.

Cette opération est irréversible, faites attention…

Rechercher quels programmes contiennent une certaine portion de code:

Personnellement je ne trouve pas ça utile, mais puisque ça existe…

det(9 , ["texte à rechercher"]

Le texte à rechercher peut être du code.

La fonction retourne une chaine qui donne la liste des noms des programmes contenant ce texte ou cette portion de code. Utile pour faire des éditeurs de texte, quoi, mais c'est tout.

Cela peut aussi servir, en empilant la fonction det(17 que nous verrons plus loin, à détecter les programmes pour ION par exemple: `det(9,det(17,"BB6DC93Ø"))`. cela recherche le header ION du programme ASM.

Si vous n'indiquez pas de texte à rechercher, cela vous donnera une liste complète de tous les programmes et AppVars, même ceux du système qui ne sont pas utilisables par le programmateur.

Bon, on complique ? Et si au lieu de commencer à lire un programme en choisissant la ligne de départ, on choisissait l'octet de départ ? Personnellement je n'ai encore pas trouvé d'utilité à ça, mais ça va venir.

Et puis ça fait genre le tuto est difficile.

Lire un programme à partir d'un octet précis.

det(14, "NOMDUPROG", octet de départ , nombre d'octets à lire

La fonction retourne une chaine contenant une suite hexadécimale, qui correspond à la suite d'octets demandés.

ne demandez pas 150 octets, vous serez surpris par la longueur du résultat.

Le début du programme correspond à l'octet 0.

-Bon. Soit. mais si on peut lire, on peut donc écrire et supprimer ?

Et oui. ça tombe bien, c'est ce dont j'allais parler. Quelle coïncidence étonnante.

Écrire une suite d'octets dans un programme. (rappel: un octet= 2 caractères hexadécimaux)

det(15, "NOMDUPROG" , "chaine en hexa" , octet de départ

Cette fonction va insérer la chaine en hexa dans le code source du prog, mais en comptant non pas en lignes, mais en octets.

- Mais à quoi ça peut bien servir ?

Et bien à bidouiller des programmes en Assembleur précompilés (en hexadécimal), comme par exemple ceux servant à tracer automatiquement des images à l'écran.

Supprimer une suite d'octets

det(16 , "NOMDUPROG" , octet de départ , nombre d'octets à supprimer

ça va, je vous explique pas ? C'est pareil qu'avant.

Bon. Puisqu'on commence à rentrer dans le sujet, on va continuer à compliquer.

J'espère que vous savez en gros comment fonctionne l'hexadécimal. Sinon, va falloir vous passer de cette section, ou lire le tutoriel de mon confrère Matrefeytontias.

V. Les bidouillages Hexadécimaux.

Alors, je vais commencer par répondre à quelques questions. Sachez avant tout que je ne suis moi-même pas super calé sur le sujet, alors j'essaie de ne pas dire de bêtises, je vais essayer en tout cas, et à priori ce que je dis est vrai.

(ben oui, sinon on s'en sort pas).

Un token est le nom qu'on donne a un caractère affiché normalement, ou a une fonction de la TI qui n'est pas divisible, dirait un programmateur pas doué qui connait mal son sujet et le vocabulaire qui correspond.

ça tombe bien, parce que comme j'en suis là, on va rester sur ces appellations.

Donc A, B, C D, +, 8, ? , % , If, Pt-On(, prgm, int(etc… sont des tokens, pour les prendre comme exemple.

Chaque token a une valeur hexadécimale qui lui correspond, et cette valeur pèse un octet pour la plupart des caractères. (Là, je crois ne pas trop me tromper.)

Comme un octet ça fait deux chiffres hexadécimaux, chaque caractère peut en fait être exprimé par deux chiffres hexa. (je rappelle que les chiffres hexa sont 0/1/2/3/4/5/6/7/8/9/A/B/C/D/E et F).

Ainsi, A a pour valeur 42, B a 43 etc… Pourquoi ces valeurs là et pas d'autres ? ~~Parce que 42 c'est la réponse à la grande question sur l'univ…~~ Je n'en ai pas la moindre idée.

Toujours est-il que ça nous permet de repérer des choses dans les programmes ASM.

Il existe donc des fonctions de Celtic III qui en parlent. Même pas mal.

Convertir un token en sa valeur, et la valeur en son token…

Hexa ➔ Token: **det(17, "chaine en hexadécimal"**

Attention, la chaine en hexa doit bien contenir un nombre **PAIR** de chiffres hexa, sinon, on coupe les octets en deux, et on n'en finit pas… Le résultat sera une chaine de caractères correspondants aux valeurs données.

Token ➔ Hexa **det(18, "chaine de caractères"**

Cette fois, la fonction va retourner une chaine de caractères hexadécimaux.

Comme on l'a vu tout à l'heure, det(17 couplée à det(9 permet d'identifier les headers des programmes en assembleur pour les identifier.

Si vous voulez une liste des tokens et de leurs valeurs hexadécimales, regardez dans le dossier MOREINFO fourni avec Celtic III, dans le fichier Tokens.txt, c'est très moche, mais la liste y est.

A tout hasard, vous savez que l'hexadécimal, ça peut se convertir en décimal.

Vous savez donc aussi que "42" en hexadécimal ne veut pas dire quarante-deux, mais 4x16^1+2*16^0=66.

Il y a donc une fonction qui sert à convertir les valeurs hexadécimales en décimales:

det(28, "4 caractères hexadécimaux"

oui, la fonction ne gère que les chaines de 4 caractères, si vous en mettez plus, ceux en trop ne seront pas lus, si vous en mettez moins, ceux manquants seront comptés comme des zéros.

Attention: pour calculer 42, il faudra bien indiquer "0042" , car "42" serait compris comme "4200" ce qui n'est pas pareil du tout.

<ici expliquer les fonctions 19 et plus quand je saurai à quoi elles servent >

VI. gestion des Groupes.

- vous parlez de groupes, depuis le début, mais qu'est-ce que c'est au juste ? Une variable ?

Non, les groupes ne sont pas des variables.

Sur ordinateur, leur extension de fichier finit par g (*.8xg , *.83g …)

Quand on les envoie à la TI dans la mémoire archive, il faut ensuite les dégrouper via le menu mémoire pour obtenir ses fichiers disponibles.

Ils peuvent contenir toutes sortes de variables, des programmes, et même certains paramètres système comme la configuration du tableur, la positon des Axes.

- quelle est leur utilité, si on a déjà les AppVars ?

Comme ils peuvent contenir plusieurs fichiers, ils sont pratiques pour ne pas saturer le menu mémoire de tas de variables. De plus, lors de l'installation d'un programme, il est plus facile d'envoyer un groupe à la TI que 7 programmes, 9 variables et 2 paramètres système.

Sonder un groupe:

Si vous êtes curieux de savoir ce que contient un groupe.

cette fonction permet de vous donner une liste de tous les programmes et AppVars trouvées dans le groupe, et surtout, l'ordre dans lequel ils se trouvent.

det(11 , "NOMDUGROUPE"

La fonction retourne une chaine faisant la liste du contenu en programmes et Appvars dans un ordre.

Dégrouper un groupe: *Attention, l'auteur précise qu'il peut arriver que cette fonction bogue.*

Il conseille de l'utiliser surtout pour des groupes ne contenant que des programmes.

Cette fonction sert donc à extraire les fichiers du groupe dans le RAM de la TI.

L'option sert dans le cas où un fichier du même nom que celui en train d'être extrait existerait déjà dans la TI.

det(10, "NOMDUGROUPE" , [option]

option: 0 ou non précisé: sauter le fichier et passer au suivant.

1 : remplacer le fichier.

Le groupe n'est pas supprimé après extraction, c'est une copie de fichiers.

<à préciser: pas de *row(???>

-et si je veux extraire juste un seul fichier de mon groupe ? Par exemple, le prgmJEUX dans le groupe DIVERS ?

C'est là qu'il faut connaitre la position du prgmJEUX dans le groupe DIVERS.

Pour ça, il faut le repérer sur la liste qu'on obtient avec det(11 .

det(12 , "NOMDUGROUPE" , numéro du fichier dans le groupe

Par exemple, si dans la liste obtenue avec det(11 le prgmJEUX est en 4e place, il faudra faire:

```
det(12,"DIVERS",4
```

- comment savoir si j'ai assez de RAM libre pour extraire ce fichier ?

Eh ben, c'est dingue, comme par hasard, y'a une fonction prévue pour.

Elle à la même syntaxe que det(12, sauf qu'elle n'extrait pas, elle donne la taille en octets de l'item en question.

det(13, "NOMDUGROUPE" , numéro du fichier dans le groupe

Ainsi, si vous voulez extraire votre programme seulement si il y a assez de RAM libre, faites:

```
:If det(13,"DIVERS",4)<det(4)      // teste si la taille du programme JEUX est inférieure à la RAM libre
:det(12,"DIVERS",4                 //extrait le fichier.
```

C'est dingue ce que le monde est bien fait quand même.

IL MANQUE PLEIN DE FONCTIONS !!! Oui, j'ai abandonné le Celtic III au profit du langage Grammer, désolé, je n'ai pas eu le temps de le finir...