Chapitre 5

Effacer l'écran ; Afficher un message en quittant

Ce chapitre va nous permettre de voir comment effacer l'écran, afin que ce que nous y affichons y apparaisse de façon plus satisfaisante, sans avoir le fond du TIOS ; ensuite, nous verrons comment il se fait que l'écran soit restauré à la fin du programme, comment empêcher ceci, et pourquoi.
Finalement, nous verrons ce que signifient les commandes commençant par des # qui sont mises par défaut au début du fichier source, à sa création.

I:\ Quelques remarques au sujet des bases de la syntaxe du C

Même si ce n'est pas le sujet de ce chapitre, j'estime qu'il est temps de faire quelques petites remarques concernant la syntaxe du langage C. En fait, j'en ferai deux, pour être précis.

Au cours des brefs exemples que nous avons jusqu'ici rencontré, vous avez sans doute pu remarque la présence de caractères point-virgule en certains endroits du code. Peut-être même avez-vous aussi pu remarquer que leur absence posait problème au compilateur, qui générait alors une erreur...
En C, le point-virgule constitue le marquer de fin d'instruction : le compilateur a besoin qu'on lui indique quand est-ce qu'une instruction prend fin, grêce à ce symbole. Cela permet, par exemple, d'écrire une instruction sur plusieurs lignes, avec ou sans espaces, afin d'améliorer la lisibilité (par exemple, pour que l'instruction tienne sur la largeur de l'écran).

La seconde remarque que j'aimerai faire ici, et que nous illustrerons par un exemple dans quelques chapitres, est qu'il est indispensable de bien faire la différence entre les majuscules et les minuscules ! Par exemple, _main n'est pas la même chose que _MAIN !

II:\ Effacer l'écran, et constater qu'il est restauré

A: Effacer l'écran

Effacer l'écran est une opération particulièrement simple, puisqu'il suffit d'appeler un ROM_CALL, nommé ClrScr, qui signifie "Clear Screen".
C'est une instruction particulièrement utile, qui permet d'afficher des graphismes à l'écran sans conserver le fond du TIOS, qui, rappelez-vous un des exemples vu précédemment, gâche terriblement l'effet...

Nous pourrions utiliser cet exemple-ci, qui trace une ligne après avoir effacé l'écran, et qui attend ensuite une pression sur une touche, afin que l'utilisateur ait le temps de voir la ligne :

```
// C Source File
// Created 03/07/2003; 18:46:33

#include <tigcclib.h>          // Include All Header Files

// Main Function
void _main(void)
{
    ClrScr();
    DrawLine(10, 30, 70, 50, A_NORMAL);
    ngetchx();
}
```

Exemple Complet

Il n'y a rien de particulier à dire sur cet exemple ; si vous tentez de l'exécuter, vous pourrez constater qu'il fonctionne comme nous le souhaitions...

B: Sauvegarde et Restauration "automatique"

Comme vous pouvez le remarquer, une fois le programme terminé, le contenu de l'écran est restauré, automatiquement pourrions-nous dire. En fait, cette restauration n'est pas automatique, dans le sens où ce n'est pas le TIOS qui le re-dessine, et dans le sens où c'est une commande de notre code source qui donne l'ordre de le sauvegarder au lancement, et de le restaurer au moment de quitter.

1: Avec TIGCC 0.95

En fait, une option permet d'indiquer au compilateur si l'on souhaite ou non que l'écran soit redessiné à la fin du programme.
Pour la trouver, cliquez sur Project, Options, Onglet Compilation, Bouton Program Options, et enfin, Onglet Home Screen. Là, vous trouverez une case à cocher intitulée Save/Restore LCD Contents.
Si cette case est cochée, ce qui est le cas par défaut, l'écran sera sauvegardé au lancement du programme, et restauré au moment où celui-ci se termine. Si vous dé-cochez cette case, vous pourrez constater que l'écran n'est plus restauré à la fin du programme : la barre de menus en haut ne devrait plus être visible (mais le redeviendra si vous appuyez sur une des touches de fonction, ou que vous ouvrez, par exemple, le var-link), et la ligne séparant la status line (en bas de l'écran) et le reste de l'écran aura disparu, et ne ré-apparaîtra pas (c'est la seule partie de l'écran qu'il n'est pas possible de récupérer sans utiliser un programme en C ou Assembleur pour la redessiner, ou sans faire de reset).

2: Avec TIGCC 0.94 ou antérieurs

En fait, c'est la ligne suivante :

```
#define SAVE_SCREEN // Save/Restore LCD Contents
```

qui indique, comme nous le prouve le commentaire, au compilateur qu'il va devoir ajouter au programme le code nécessaire pour sauvegarder et restaurer l'écran.
Si vous supprimez cette ligne, ou la passez en commentaire, vous pourrez constater que l'écran n'est plus restauré à la fin du programme : la barre de menus en haut ne devrait plus être visible (mais le redeviendra si vous appuyez sur une des touches de fonction, ou que vous ouvrez, par exemple, le var-link), et la ligne séparant la status line (en bas de l'écran) et le reste de l'écran aura disparu, et ne ré-apparaitra pas (c'est la seule partie de l'écran qu'il n'est pas possible de récupérer sans utiliser un programme en C ou Assembleur pour la redessiner, ou sans faire de reset).

Même si vous ne pensez pas un jour utiliser une version de TIGCC antérieure à la 0.95, je vous conseille de lire ce que je dis à son sujet : cela peut vous aider à comprendre des sources que vous aurez l'occasion de lire, et qui ont été écrites par des programmeurs ayant utilisé une ancienne version de TIGCC, ou n'ayant pas pris la peine de changer leurs habitudes.
Par exemple, définir SAVE_SCREEN a été la méthode conseillée pendant plus de deux ans, il me semble... ce qui correspond à un nombre assez impressionnant de programmes !!!
Cette remarque est vraie pour la sauvegarde de l'écran... elle l'est aussi pour pas mal d'autres choses, même si je ne prendrai pas la peine de la reformuler à chaque fois.

C: Supprimer la sauvegarde et restauration automatique de l'écran

Il n'est pas difficile de supprimer la sauvegarde et restauration automatique de l'écran : nous l'avons fait juste au-dessus, en dé-cochant l'option correspondante (version 0.95 de TIGCC), ou en en supprimant la ligne correspondante, ou en la passant en commentaire (versions 0.94 et antérieures).

Ce sur quoi je voudrai surtout insister, c'est sur l'utilité de ne pas restaurer automatiquement l'écran.
Pour le moment, pour les petits programmes que nous avons réalisé, la restauration automatique est bien pratique, et il est vrai qu'elle le restera dans l'avenir, même pour des programmes plus importants. Cela dit, parfois, par exemple, on peut avoir envie de laisser un message à l'écran, une fois le programme terminé, et cela est impossible si la restauration automatique est activée, puisque l'écran sera redessiné une fois que toutes nos instructions auront été exécutées.
Nous verrons dans la prochaine partie de ce chapitre comment faire pour pouvoir afficher un message à l'écran, qui reste une fois le programme terminé, mais sans que l'écran ne soit pas redessiné.

III:\ Laisser un message une fois le programme terminé

A la fin de votre programme, vous pouvez souhaiter laisser, par exemple, un message dans la barre de status en bas de l'écran, renvoyant vers votre site web, ou annonçant votre nom... tout en voulant que l'écran soit redessiné, puisque le fait ne pas restaurer l'écran est, comme nous l'avons vu, assez peu esthétique !
Pour cela, il va vous falloir sauvegarder l'écran "à la main" au début du programme, et le restaurer, toujours "à la main", à la fin de celui-ci. Une fois l'écran restauré, vous pourrez afficher votre message, qui ne sera donc pas écrasé par la restauration de l'écran.
Du temps où j'ai commencé à programmer en C, il fallait d'ailleurs toujours faire comme ça, car l'instruction SAVE_SCREEN n'existait pas encore.

A: Sauvegarder l'écran

Pour sauvegarder l'écran, il vous faut déclarer une variable de type LCD_BUFFER, de la façon suivante :

```
LCD_BUFFER sauvegarde_ecran;
```

(Nous verrons plus en détails au chapitre suivant la déclaration de variables ; pour l'instant, sachez juste que c'est ceci qu'il faut faire, mais que vous pouvez remplacer 'sauvegarde_ecran' par autre chose de plus évocateur à vos yeux, si vous le souhaitez)
Ensuite, il faut sauvegarder l'écran ; pour cela, nous utiliserons la macro LCD_save, en lui passant en argument le nom de la variable que nous venons de déclarer, comme indiqué ci-dessous :

```
LCD_save(sauvegarde_ecran);
```

A présent, nous pouvons effacer l'écran, et dessiner ce que nous souhaitons dessus, comme nous l'avons déjà fait auparavant.

B: Restaurer l'écran

Pour restaurer l'écran, c'est aussi extrêmement simple : il nous suffit, comme montré dans la portion de code reproduite ci-dessous, d'appeler la macro LCD_restore, en lui passant en argument, un fois encore, la variable que nous avons déclaré plus haut.

```
LCD_restore(sauvegarde_ecran);
```

Ce n'est pas plus difficile que cela.

C: Exemple de programme

Maintenant que nous savons sauvegarder et restaurer l'écran, passons à un exemple d'utilisation :

```
// C Source File
// Created 08/10/2003; 14:17:20

#include <tigcclib.h>

// Main Function
void _main(void)
{
    LCD_BUFFER sauvegarde_ecran;
    LCD_save(sauvegarde_ecran);
    ClrScr();
    DrawLine(10, 30, 70, 50, A_NORMAL);
    ngetchx();
    LCD_restore(sauvegarde_ecran);
    ST_helpMsg("Chapitre 5 du tutorial C");
}
```

Exemple Complet

Avec une version de TIGCC antérieure à la 0.95, nous aurions ceci :

```
// C Source File
// Created 03/07/2003; 18:46:33

#define USE_TI89              // Compile for TI-89
#define USE_TI92PLUS          // Compile for TI-92 Plus
#define USE_V200              // Compile for V200

// #define OPTIMIZE_ROM_CALLS // Use ROM Call Optimization

#define MIN_AMS 100           // Compile for AMS 1.00 or higher

//#define SAVE_SCREEN         // Save/Restore LCD Contents

#include <tigcclib.h>         // Include All Header Files

// Main Function
void _main(void)
{
    LCD_BUFFER sauvegarde_ecran;
    LCD_save(sauvegarde_ecran);
    ClrScr();
    DrawLine(10, 30, 70, 50, A_NORMAL);
    ngetchx();
    LCD_restore(sauvegarde_ecran);
    ST_helpMsg("Chapitre 5 du tutorial C");
}
```

Exemple Complet

Comme nous pouvons le voir, le programme s'exécute exactement de la même façon que plus haut (nous avons, une fois de plus, repris le même exemple, en le complétant), et, une fois le programme fini, un message reste affiché dans la barre de status.
Ce message s'effacera dès que l'utilisateur appuiera sur une touche.

IV:\ Les commandes inclues par TIGCC avec les options par défaut

Pour terminer ce chapitre, nous allons parler des commandes inclues par TIGCC au début de notre code : les lignes commençant par un caractère #. Nous n'examinerons que celles qui sont incluses avec les options par défaut ; c'est-à-dire celles que nous avons utilisées dans nos codes sources, jusqu'à présent sans réellement se soucier de ce qu'elles faisaient.
Dans cette partie, la plupart des commandes ont été remplacées par des cases à cocher dans les options du projet lors de la sortie de TIGCC 0.95... en fait, seule la dernière "commande", le #include, n'a pas été remplacé de la sorte. Cela dit, vous serez probablement amené à rencontrer ces options si vous lisez des codes sources écrits par d'autres que vous ; je vous encourage donc à ne pas ignorer cette partie de ce chapitre...

A: Modèles de calculatrices pour lesquels le programme doit être compilé

```
#define USE_TI89                // Compile for TI-89
#define USE_TI92PLUS            // Compile for TI-92 Plus
#define USE_V200                // Compile for V200
```

Ces trois lignes indiquent au compilateur pour quelles machines il doit créer le programme. Vous pouvez créer votre programme pour l'une, les deux, ou les trois machines.
Pour votre usage personnel, autant ne pas se fatiguer à créer des programmes compatibles entre les différentes calculatrices ; cela dit, si vous destinez votre programme à la diffusion, il peut être bon qu'il fonctionne sur chaque modèle de machine, afin de toucher le plus d'utilisateurs possible.

B: Optimisation des ROM_CALLs

```
#define OPTIMIZE_ROM_CALLS // Use ROM Call Optimization
```

Cette directive permet ordonne au compilateur d'optimiser les appels de ROM_CALLs. Cela dit, il faut quelques instructions pour que cette optimisation se fasse, ce qui explique que cette directive fasse grossir la taille du programme si peu d'appels aux fonctions inclues dans la ROM sont effectués.
Cette optimisation réserve aussi un registre, pour toute la durée du programme ; ce registre ne peut donc pas être utilisé pour les calculs. Les registres n'étant que très peu nombreux, et étant les mémoires les plus rapides de la machine, dans le cas d'un programme n'utilisant que peu de ROM_CALL (un jeu évolué, par exemple), on préférera souvent ne pas utiliser cette "optimisation", de façon à accélérer les calculs.

C: Version minimale d'AMS requise

```
#define MIN_AMS 100            // Compile for AMS 1.00 or higher
```

Chaque nouvelle ROM, c'est-à-dire chaque nouvelle version d'AMS (Advanced Mathematical Software, le "cerveau" logiciel de la calculatrice) sortie par Texas Instrument apporte de nouveaux ROM_CALLs ; il est donc des ROM_CALLs qui ne sont pas disponibles sur les anciennes ROMs... Si vous utilisez certains de ceux-là, il vous faut préciser à partir de quelle version de ROM votre programme peut fonctionner, afin qu'il ne puisse pas être lancé sous des versions plus anciennes, où il aurait un comportement indéterminé.

Par exemple, si vous souhaitez utiliser le ROM_CALL AB_getGateArrayVersion, défini uniquement à partir de la ROM 2.00, d'après la documentation, il vous faudra définir ceci :

```
#define MIN_AMS 200
```

Si vous ne le définissez pas, en laissant, par exemple, la valeur par défaut de 100, le compilateur vous renverra un message d'erreur, disant qu'il ne connaît pas ce ROM_CALL.

Cela dit, je vous encourage fortement à n'utiliser que des ROM_CALL existant depuis le plus longtemps possible, afin que votre programme puisse fonctionner sur le maximum de versions de ROM !

D: Sauvegarde/Restauration automatique de l'écran

```
#define SAVE_SCREEN            // Save/Restore LCD Contents
```

Nous avons déjà vu ceci plus haut, puisque c'était le sujet primaire de ce chapitre ; nous ne reviendrons pas dessus.

E: Include standard

```
#include <tigcclib.h>         // Include All Header Files
```

Pour finir ce chapitre, parlons rapidement de la directive #include.
Elle permet, comme son nom l'indique, d'ordonner au compilateur d'inclure du texte, tel, par exemple, du code, depuis un fichier, pris dans le répertoire par défaut (c:\Program Files\TIGCC\Include\C si vous avez suivi l'installation par défaut) si le nom de ce fichier est écrit entre chevrons (entre le caractère '<' et le caractère '>'), ou dans le répertoire courant s'il est écrit entre guillemets doubles ("").

Le fichier tigcclib.h contient les prototypes des ROM_CALLs, ainsi que de nombre autres fonctions. Le fait de l'inclure permet au compilateur de savoir quels paramètres les ROM_CALLs doivent recevoir, quelles sont leurs valeurs de retour, ...
Il est généralement nécessaire d'inclure ce fichier pour parvenir à compiler un programme sous TIGCC.

Nous voila parvenu à la fin de ce chapitre. Le prochain nous permettra de parler de la notion de variables, et d'apprendre comment en déclarer, ainsi que les bases de leur utilisation.