Chapitre 3

Programmons en C

I:\ Quelques remarques concernant ce tutorial

Nous allons commencer ce chapitre par quelques remarques sur la forme que nous avons choisi de donner à ce tutorial, et peut-être aussi quelques remarques sur son fond, afin d'expliquer les choix que nous avons été amené à effectuer.

A: Un tutorial

Lorsque l'on souhaite écrire un document traitant d'un langage de programmation, tel le C, plusieurs approches s'offrent à nous. En particulier, il faut choisir entre deux possibilités :

- Une approche chapitre par chapitre, chacun d'entre eux traitant d'un sujet différent,
- ou une approche plus linéaire.

Dans le second cas, le tutorial commencerait par énormément de théorie, et finirait par présenter quelques fonctions de base... Autrement dit, nous commencerions par un bla-bla sans fin, extrêmement lassant et complexe à assimiler sans pratiquer, et finirions par ce qui est, pour ainsi dire, le plus simple... De plus, il nous faudrait attendre la fin, ou presque, du tutorial, pour parvenir à réaliser des programmes faisant quelque chose d'utiles, ce qui n'est pas notre but, et probablement pas le votre non plus !
Dans le premier cas, nous présenterions la théorie au fur et à mesure que nous en aurions besoin, ce qui nous permettrait d'immédiatement la mettre en application, de façon à mieux vous la faire assimiler. Certes, une telle approche entrerait certainement moins dans les détails de chaque sujet, mais rien ne nous empêchera d'y revenir plus tard, lorsque nous aurons besoin de ces détails...
Nous estimons que la première approche est la plus adaptée pour l'apprentissage d'un langage tel le C, en particulier pour des débutants en programmation, de par le fait qu'elle nous permet de couvrir différents sujets, qu'elle vous permettra rapidement d'écrire des programmes simples, et qu'elle est, sans nulle doute, la moins lassante. Nous estimons que la seconde approche serait plus adaptée pour l'écriture d'une référence, non d'un tutorial.

Ce tutorial existe en version à consulter en ligne, mais aussi à consulter hors-ligne, téléchargeable en version PDF (c'est la version hors-ligne qui est imprimable via les icones d'imprimante en haut, et en bas, de chaque page du tutorial). Il n'y a pas de différence majeure de contenu entre ces deux versions, mais il peut y avoir quelques différences de forme ou de mise en page.
Par exemple, la version on-line présente en en-tête de chaque chapître une citation, sélectionnée aléatoirement. La version hors-ligne ne les présente pas. Notez que ces citations touchent à la programmation en général, et pas uniquement au langage C, ce qui explique le fait que, par exemple, un nombre non négligeable de citations soient de Bjarne Stroustrup, créateur du langage C++, mais aussi le fait que ces même citations soient aussi utilisées pour le tutorial de la TCI traitant du langage d'Assembleur.

En plus du contenu chapitre par chapitre de ce tutorial, vous avez à votre disposition une page de conseils, qu'il serait bon, à mon avis, que vous lisiez de temps en temps, sachant que plus vous avancerez dans le tutorial, mieux vous comprendrez le contenu de cette page, et une FAQ, dans

laquelle j'ai regroupé quelques questions fréquemment posées, avec leurs réponses.
Avant de me poser des questions par mail, j'aimerai que vous utilisiez ces deux pages, ainsi que les divers forums que vous pourrez trouver sur Internet, notamment du fait que je n'ai que très peu de temps pour m'occuper de mon courrier...

B: Des exemples, indispensables dans un tutorial

Tout au long de ce tutorial, vous trouverez des exemples de codes sources. Au maximum, j'essayerai de les présenter avec la coloration syntaxique par défaut de l'IDE de TIGCC, afin de ne pas changer vos habitudes, à une différence prêt : je ne mettrai pas, dans les exemples de ce tutorial, pour raison de lisibilité, en gras les types de donnés, écris en bleu. Je tenterai aussi de respecter certaines habitudes d'écriture et de présentation, par exemple en indentant, ou en plaçant des caractères d'espacement à certains endroits, afin de rendre le source plus lisible, même si cela n'est nullement une obligation pour le langage C.
Je ne peux que vous encourager à bien présenter votre code source, afin de le rendre plus lisible, ce qui ne peut que faciliter sa compréhension, que ce soit pour vous, si vous êtes amené à le reprendre quelques temps après, ou pour n'importe quel lecteur, si vous diffusez vos programmes en open-source.

Tout au long de ce tutorial, en plus de la coloration syntaxique, j'ai choisi de respecter une norme de présentation pour tous les exemples que je donnerai.
Dans un cadre de couleur violette, vous trouverez les exemples d'algorithmes, ainsi que les notations théoriques conformes à la grammaire du C. Dans un cadre de couleur noire, les exemples de codes sources ; lorsque ceux-ci seront compilables par un simple copier-coller, ils seront suivi de la mention "Exemple Complet" ; si cette mention est absente, c'est que le code source que je vous proposerai ne sera pas entier : il s'agira simplement d'une portion de code source, mettant l'accent sur LE point précis que j'ai choisi de mettre en valeur par cet exemple.

Pour bien vous montrer comment les exemples sont présentés, en voici :

```
Ceci est un algorithme, ou une notion théorique de grammaire.
```

```
Ceci est un exemple, une portion de programme ciblant un point précis.
```

```
Et ceci est un exemple complet, de code source compilable sous TIGCC v0.95
```

Exemple Complet

Vous remarquerez probablement qu'il est assez rare que je mette des exemples complets... En effet, je préfère ne mettre que des portions de sources, plus courtes que des exemples complets, afin de mieux cibler LE point important que l'exemple doit montrer. De plus, je considère qu'il vaut mieux pour vous que vous ayez un peu à réfléchir pour pouvoir utiliser le code que je vous propose en exemple, afin que vous cherchiez à comprendre commet il fonctionne et comment il peut interagir avec le reste d'un programme, plutôt que de vous présenter un programme complet, que vous vous contenterez d'exécuter sans même réfléchir à son fonctionnement.

C: Programmation en C, et normes internationales

Si vous souhaitez en savoir plus sur la programmation en C, je vous conseille vivement d'aller dans une librairie (une grande librairie, de préférence, disposant d'un rayon informatique, afin que vous ayez un choix important ; une FNAC fait généralement parfaitement l'affaire), de feuilleter quelques livres traitant du C, et d'en acheter un. J'insiste sur le fait qu'il est bon de feuilleter plusieurs livres, afin que celui que vous retiendrez vous plaise, ait une présentation agréable, ... , afin que vous ayez envie de l'étudier. Si je peux me permettre un conseil de plus, ne choisissez pas un livre par sa taille : un livre trop petit risque de survoler des notions importantes à force de vouloir les résumer, et un livre trop gros risque d'être indigeste et de vous lasser.
Certes, ces livres traiteront de la programmation en C pour PC ; mais le C est un langage universel : sa logique, que ce soit pour PC (préférez un livre traitant de la programmation en ligne de commande à un livre traitant de programmation en interface graphique style Windows... La programmation en interface graphique est extrêmement complexe pour un débuttant en programmation, à mon avis, et s'éloigne grandement de la programmation pour TI), ou pour TI, est la même.
Pour écrire ce tutorial, il m'est arrivé, m'arrive, et m'arrivera, d'utiliser le livre Le Langage C, seconde édition (The C Programming Language, 2nd edition), écrit par Brian W. Kernighan et Denis M. Ritchie, les deux fondateurs du langage C. Ce livre est généralement appelé "K&R", du nom de ses auteurs, et constitue en quelque sorte la référence. Cela dit, je ne le conseille pas à des débutants en programmation... Bien que conforme à l'esprit du langage dont il est la Bible, il ne me paraît pas vraiment adapté à des néophytes... (Ceci est un avis personnel ; libre à quiconque me lisant de penser le contraire, bien entendu).

Je finirai cette partie en disant qu'il existe deux standards concernant la programmation en C :

- Le standard ANSI, qui est admis par tous les compilateurs C, et qui est LE standard,
- et le standard GNU, qui est admis par le compilateur GCC et ses dérivés, dont TIGCC.

Le standard GNU englobe les fonctionnalités du standard ANSI, et rajoute des "extensions GNU", qui le rendent plus riche de par certains aspects.
Cela dit, bon nombre d'extensions GNU ne sont valables que sous le compilateur GCC (et donc TIGCC), et pas sous d'autres compilateurs non-GNU, tels Microsoft Visual C++ (qui est aussi capable de compiler du code C), par exemple, qui suit la norme ANSI.
Notons tout de même que la logique du GNU-C est la même que celle de l'ANSI-C, et que seules des petits extensions ont été rajoutées ; ce ne sont pas des évolutions majeures, mais juste des facilités pour le programmeur, en général. Si vous avez l'habitude d'utiliser du GNU-C, il ne vous sera pas difficile de vous adapter à un compilateur ANSI (Ayant moi-même commencé par apprendre le GNU-C, je n'ai eu aucune difficulté à utiliser des compilateurs ANSI, et parle donc d'expérience vécue).
Une partie des extensions GNU est même généralement acceptée par les compilateurs qui se disent ANSI, même si elles ne sont pas (pas encore, devrai-je dire) officialisées !

II:\ Mais qu'est-ce qu'une TI ?

Avant de commencer à programmer, nous allons, rapidement, voir ce qu'est une TI. Nous n'avons pas besoin, pour programmer en C, de connaître autant de détails que pour programmer, par exemple, en Assembleur, ce qui nous permettra d'être assez bref, et ne ne voir que la couche superficielle.

Ce tutorial est prévu pour les calculatrices Texas Instrument modèles TI-89/92/V-200. A peu de choses près, ces trois machines sont identiques : leurs différences majeures sont au niveau de la taille de l'écran, et du clavier.

Il existe deux versions matérielles différentes : les HW1, et HW2.
Les HW1 sont les plus anciennes, les HW2 les plus récentes. Les HW2 comportent quelques fonctionnalités supplémentaires par rapport aux HW1 (par exemple, les HW1 n'ont pas de support d'horloge). La V-200 est considérée comme une HW2 (Il n'existe pas de V-200 HW1).
Au cours de notre apprentissage de la programmation en Assembleur, il sera rare que nous ayons à nous soucier des différences entre les différentes versions matérielles, mais, quand il le faudra, nous préciserons que c'est le cas.

Il existe aussi plusieurs versions de "cerveaux". Ce "cerveau" est le logiciel qui vous permet de faire des mathématiques, de programmer en TI-BASIC, de dessiner, de lancer des programmes en Assembleur, ... ; bref, ce "cerveau" est ce grâce à quoi votre machine est plus qu'un simple tas de composants électroniques.
Différentes versions allant de 1.00 sur TI-92+, 1.01 sur TI-89, et 2.07 sur V-200, à, pour le moment, 2.09 (sortie durant le second trimestre 2003) ont été diffusées par Texas Instrument. En règle générale, plus la version est récente, plus elle comporte de fonctions intégrées, directement utilisables en Assembleur.
Ce "cerveau" est généralement appelé "AMS" (pour Advanced Mathematic Software), ou, par abus de langage, "ROM" (Pour Read Only Memory), puisque l'AMS est stocké dans un mémoire flashable de ce type.
Le plus souvent possible, nous essayerons de rédiger des programmes compatibles entre les différentes versions de ROM, mais, dans les rares cas où ce ne sera pas possible (par exemple, parce que nous aurons absolument besoin de fonctions qui ne sont pas présentes sur certaines anciennes ROM), nous le signalerons.

Nos TIs sont dotées d'un processeur Motorola 68000, cadencé à 10MHz sur HW1, et à 12MHz sur HW2.

III:\ Un premier programme

Maintenant que nous avons beaucoup parlé, nous allons pouvoir passer à quelque chose d'un peu plus intéressant, notre premier programme.
En fait, nous allons reprendre le code généré par défaut par TIGCC, que nous avons pu voir au premier chapitre, et expliquer ce qu'il contient qui soit, pour le moment, interessant. Nous n'étudierons probablement pas chacun des détails de ce code source, ce serait vouloir aller trop vite, je pense...

Pour nous remettre ce code en mémoire, le voici :

```
// C Source File
// Created 06/10/2003; 00:02:40

#include <tigcclib.h>

// Main Function
void _main(void)
{
    // Place your code here.
}
```

Exemple Complet

Ou alors, si vous utilisez une version de TIGCC antérieure à la 0.95, ce que je vous déconseille, puisque la 0.95 offre plus de fonctionnalités, et que c'est celle dont nous traiterons au cours de ce tutorial, vous aurez quelque chose approchant le code source présenté ci-dessous :

```
// 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)
{
  // Place your code here.
}
```

Exemple Complet

A: Un peu d'anglais, avant tout

La première chose que l'on remarque si on parcoure ce code source est que beaucoup de texte est en anglais. Il faut que vous compreniez une chose : en informatique, l'anglais est la langue de référence ! Si vous cherchez de la documentation sur Internet, vous finirez par tomber sur de l'anglais, si vous lisez la documentation de TIGCC, ce que je vous conseille vivement, vous constaterez qu'elle est en anglais, si vous avez l'intention de parler programmation sur Internet, vous aboutirez à des chans en anglais, un jour ou l'autre, ...
Même si vous n'aimez pas beaucoup l'anglais, si vous voulez vous lancer dans l'informatique, il vous faudra apprendre, comprendre, savoir lire, écrire, et parler l'anglais. Si vous souhaitez effectuer vos études dans un des domaines de l'informatique, l'anglais vous sera quasiment indispensable, croyez-moi.
Étant donné l'importance de l'anglais, je ne prendrai pas la peine de traduire chaque phrase ou mot que nous rencontrerons dans cette langue ; je traduirai parfois certaines notions capitales, mais il vous faudra vous débrouiller pour le reste : je fais parti de ceux qui pensent qu'ils vaut mieux vous pousser à vous améliorer, plutôt que de tout vous mettre entre les mains.

B: Commentons, commentons

La seconde chose que l'on remarque, si l'on prête un tant soit peu attention à la coloration syntaxique, est la forte proportion de vert. Sous l'IDE de TIGCC, cette couleur est utilisée pour représenter ce qu'on appelle "commentaires". Un commentaire est du texte, qui ne sera pas pris en compte par le compilateur lors de la compilation ; les commentaires vous permettent, comme leur nom l'indique, de commenter votre code source, afin de faciliter sa relecture, et sa compréhension. Par exemple, si une ligne vous a posé de gros problèmes à l'écriture, vous pouvez expliquer en commentaire ce qu'elle fait. Si vous utilisez un algorithme particulier, vous pouvez expliquer comment il fonctionne.

Je vous encourage à fortement commenter vos codes sources ; cela vous permettra de mieux les comprendre si vous les relisez plus tard. Certes, au moment où vous écrivez votre programme, vous parvenez sans mal à comprendre ce qu'il fait... Mais six mois plus tard, quand vous aurez oublié la façon dont vous l'avez écrit et les algorithmes que vous avez utilisé, je vous assure que vous aurez beaucoup plus de mal à le comprendre ! Certes, pour un petit programme de quelques centaines de lignes, il n'est pas très difficile de retrouver son fonctionnement en lisant le code source... Mais quand vous avez un gros projet de plus de vingt mille lignes pour vos études, d'une durée de quelques mois, plus deux projets de cinq mille lignes chacun pour TI sur lesquels vous travaillez quand vous avez un peu de temps libre, et qui s'étalent donc eux-aussi sur plusieurs mois, plus quelques TP d'un ou deux milliers de lignes en trois ou quatre langages différents, je suis en mesure de vous assurer que vous comprenez vite l'intérêt des commentaires !
Et ceci est encore plus vrai lorsque vous travaillez à plusieurs sur un projet : commenter permet d'éviter que, toutes les cinq minutes, les gens avec qui vous travaillez ne viennent vous demander ce que fait telle ou telle fonction, que vous n'avez pas commenté en vous disant que c'était évident !
Cela dit, ne tombez pas non plus dans l'extrême : si vous avez une ligne de programme contenant "2+2", il n'est probablement pas très judicieux de la commenter en notant que "celle ligne permet de faire l'addition de deux et deux" ! Ce serait une perte de temps totale que de commenter ce genre de choses !
Pour résumer, le plus difficile avec les commentaires est sûrement de juger où est-ce qu'ils sont utiles, et où est-ce qu'ils ne le sont pas... Mais cela vient avec l'expérience :-)

Il existe deux manières d'écrire des commentaires :

La première, qui est utilisée dans le code source présenté plus haut, est de commencer le commentaire par une suite de deux caractères slash : //
Tout ce qui suit, jusqu'à la fin de la ligne, fera parti du commentaire.
Cette solution pour marquer les commentaires fait parti de la norme C99 (commentaires de style C++), mais je n'ai jamais rencontré de compilateur la refusant, même pour les compilateurs plus anciens.

```
// Ceci est un commentaire
```

La seconde permet d'écrire des commentaires couvrant plusieurs lignes, ou seulement une portion de ligne.
Pour commencer un commentaire de ce type, il faut écrire un slash suivit d'une étoile (symbole de la multiplication) : /*
Et un commentaire de ce type se termine par une étoile suivie d'un slash : */
Autrement dit, ce qui est compris entre /* et */ est le commentaire.
Notez que ces commentaire ne peuvent s'imbriquer : ouvrir un commentaire de ce genre à l'intérieur d'un autre est une erreur, et sera refusé par le compilateur.

```
/* Ceci est
un autre
commentaire */
```

C: La fonction _main

Nous n'étudierons pas au cours de ce chapitre ce qui écrit en **vert foncé gras** dans le code source présenté plus haut ; disons que cette ligne est très bien comme elle est (ces lignes, dans le cas de versions antérieures à la 0.95, ou dans le cas où vous n'auriez pas supprimé ce que nous vous avons conseillé de supprimer, au chapitre 1), pour des programmes simples, et que nous n'avons pas vraiment besoin de nous en préoccuper pour l'instant : mieux vaut se soucier de ce qui est réellement important. Naturellement, nous expliquerons ce que tout ceci signifie, mais dans quelques chapitres, quand nous aurons vu quelques bases avant.
Cela dit, pour la version 0.94 de TIGCC, rien ne vous empêche d'essayer de comprendre les lignes inclues par défaut de par vous-même ; certaines d'entre elles sont aisément compréhensibles grêce à leurs commentaires. D'autres le sont moins... si la curiosité vous brule, et que vous ne pouvez attendre, libre à vous de consulter la documentation de TIGCC (Ces commandes sont toujours documentées, même si elles ne sont normalement plus utilisées directement, depuis la version 0.95 de TIGCC).

Ce que nous allons ici étudier est la fonction principale du programme, la fonction _main, qui correspond à la portion de code reproduite ci-dessous :

```
void _main(void)
{
  // Place your code here.
}
```

Tout d'abord, essayons de définir ce qu'est une fonctions :
Une fonction est un "morceau" de code source, qui permet de regrouper certains traitement dans une sorte de boite, dont on peut ensuite se servir sans même savoir comment elle a été programmée. Avec une fonction correctement conçue, on peut ne pas se soucier de la façon dont un traitement est effectué ; il nous suffit de savoir quel est ce traitement. D'ailleurs, au cours de ce tutorial, et même dans quasiment tous les programmes que vous écrirez, vous utiliserez des fonctions dont vous ne connaîtrez pas le code source : vous saurez comment les utiliser et ce qu'elles font, mais pas comment elles le font. Cette possibilité est une des grandes forces du C.
Une fonction est définie de la façon suivante :

```
type_de_retour nom_de_la_fonction(liste_des_arguments_éventuels)
{
  // code de la fonction.
}
```

La fonction _main est celle qui est appelée automatiquement au lancement du programme ; c'est l'équivalent de la fonction main, sans underscore, sur la majorité des plate-formes.
Pour nos calculatrices, le type_de_retour de cette fonction doit être void, ce qui signifie qu'elle ne retourne rien, et la liste_des_arguments_éventuels doit être void aussi, ce qui signifie qu'elle ne prend aucun argument. Nous verrons plus tard ce que sont les arguments et le type de retour, ainsi que comment les utiliser ; pour l'instant, tout ce que vous devez savoir est que la fonction _main attend void pour les deux.

Dans notre exemple, la fonction _main ne contient qu'un commentaire, c'est-à-dire, aux yeux du compilateur, rien. Donc, lors de l'exécution du programme, rien ne se passera (Ou plutôt devrai-je dire que rien de visible ne se passera... Nous verrons plus tard pourquoi, mais cela est du aux lignes en **vert foncé gras** disparues depuis la version 0.95 de TIGCC, remplacées par des cases à cocher dans les options du projet).

Au cours des programmes que nous serons amené à écrire, nous placerons le code que nous

souhaitons exécuter dans la fonction _main, à la place du commentaire nous disant de "placer notre code ici". Puis, lorsque nos programmes commencerons à être suffisamment important, ou lorsque cela sera judicieux, nous les découperons en plusieurs fonctions, appelées par la fonction _main, et pouvant s'appeler les unes les autres.

Au cours du prochain chapitre, nous étudierons comment appeler une fonction intégrée à la ROM, et n'attendant pas de paramètre, puis nous verrons comment utiliser des fonctions attendant des paramètres.