

Tutorial n°2 - La mémoire

Catégorie : C

Publié par [Elfen](#) le 24/5/2006

Dans ce tuto nous expliquerons le code du tuto 1 et nous introduirons une nouvelle notion : la mémoire.

La mémoire

Vous voilà dans votre premier vrai tuto de C, je vous conseille de le lire attentivement car il est totalement indispensable et car il introduit pas mal de nouvelles notions.

1/ Mise au point

Je suppose que les notions que nous avons abordés dans le premier tuto sont encore obscures pour vous, ceci est normal car le premier tuto n'avait pas pour but de vous apprendre le C mais de vous présenter les outils qui vous serviront pour programmer en C et la console.

Maintenant nous allons analyser le "Hello, world" en détail, profitez-en car ce sera le seul programme que vous verrez qui n'utilisera pas les mathématiques (non n'ayez pas peur, vous n'aurez besoin que de savoir ce que sont l'addition, la soustraction, la division et la multiplication).

Regardons donc notre "Hello, world" de plus près :

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
    printf("Hello, world\n");
    getch();
    return 0;
}
```

Premièrement : les `#include`, et bien je ne peux pas vous dire grand chose dessus, dites vous seulement qu'ils incluent un autre fichier à votre code et que ce fichier est nécessaire pour utiliser certaines fonctions.

Ensuite : le `int main()`, c'est la fonction `main()` de notre programme, dites vous que c'est entre `{` et `}` qu'on mettra (pour l'instant) le code proprement dit, `{` et `}` délimitent la fonction `main()`, cela sera plus clair quand on verra les fonctions ...

Après :

```
printf("Hello, world\n");
```

, c'est la célèbre fonction printf(), comme nous n'avons pas fait les fonction je ne vous donnerais, comme pour la fonction main(), la fonction printf(), sert à afficher du texte, elle est mieux que la plupart (et il y en a pas mal) des fonctions qui affichent du texte pour des raisons que nous verrons dans la suite de ce tuto, en gros on lui envoi du texte entre les parenthèses et entre guillemets et elle nous l'affiche gentilleement à l'écran, le '\n'

est considéré comme un caractère seul, et en C les caractères seuls se mettent entre ' et ont une coloration différente des chaînes de caractères, qui elles sont écrites entre guillemets) sert à dire à printf() "pourrait-tu, s'il te plais, aller à la ligne", ce à quoi printf() ne répond pas "mais avec joie ... ", mais au moins il fait ce qu'on lui demande, c'est déjà ça ...

A la ligne suivante : getch() est une fonction qui attend l'appui sur une touche, rassurez-vous elle sert à faire plus de choses mais pour cela attendez la suite.

Enfin : **return 0;**

, là encore vous ne comprendrez qu'après le tuto sur les fonctions, en fait return indique que l'on va renvoyer une valeur, mais renvoyer à qui ? Et bien la fonction main() est un cas particulier où on renvoie une valeur au système d'exploitation (windows par exemple) et 0 est la valeur qui indique à au moins 99.99999% des systèmes d'exploitations que le programme s'est bien terminé.

2/ La mémoire

Nous allons ici en apprendre un peu plus sur notre ordinateur, vous savez sûrement tous qu'il utilise de la mémoire : je suppose que vous connaissez au moins l'existence du disque dur ... et bien sachez que ce n'est pas le seul type de mémoire que votre ordinateur utilise.

L'ordinateur utilise comme type de mémoire :

1. Les registres
2. Le cache
3. La RAM
4. Le disque dur

Nous ne travaillerons que sur la RAM et le disque dur, même si ce sont les plus lents.

Attention : la RAM et le disque durs ne sont pas lents, ce sont les plus lent, tout est relatif ...

Reattention : ce n'est pas parce que nous ne travaillerons pas sur les registres et le cache que notre programme ne les utilisera pas.

Chaque mémoire a ses avantages, les registres sont extrêmement rapides, mes de très petite taille, le cache est plus grand mais plus lent, la RAM est encore plus lente mais offre aujourd'hui un très grand espace de stockage quand au disque dur, il offre un espace de stockage gigantesque et ne se vide pas au redémarrage de l'ordinateur.

Nous allons maintenant nous intéresser a la RAM, il est indispensable de connaître à peut près son fonctionnement.

En gros la RAM permet de stocker des valeurs et de les retrouver grâce a leurs adresses, imaginons que je veuille stocker le nombre 5, je dis a mon OS (windows) : "Voilà, je voudrais stocker une valeur", il me répond : "Bien sur, tu peut la mettre à l'adresse 4112412014", maintenant 5 est stocké dans 4112412014 et je peut y réaccéder quand je veux, l'interet est que je peut mettre presque n'importe quelle valeur dans 4112412014, si vous ne comprenez pas vous verrez dans la suite du cours quand on passera à la pratique qu'en fait c'est très simple.

Dernier détail : la valeur ne peut être qu'un nombre, pour stocker une lettre il suffit de stocker un nombre et de dire à l'ordinateur : "Je veux que tu m'affiche la valeur contenue dans 4112412014 sous forme de lettre", celui-ci regarde la valeur et vois un 5, il regarde alors dans son petit manuel et lit que 5 correspond à '5', il affiche donc '5' (bon OK c'est pas un bon exemple car vous pourrez me dire que '5' n'est pas une lettre, cependant si on essaye avec 97 l'ordinateur affiche 'a' ...).

3/ Les variables

Bonne nouvelle ! Vous n'avez pas à écrire les adresses des variables en C, vous donnerez un nom à ces variables, par conséquent vous appellerez une variable par son nom et non son adresse.

Mais quel nom donner à ses variables ?

Le nom d'une variable est composé uniquement de lettres, de chiffres et peut utiliser le caractère underscore : _

Le nom d'une variable commence nécessairement par une lettre, les accents sont interdits.

En C un 'A' et un 'a' ne sont pas les mêmes lettres, donc la variable NoMdUjOuEuR n'est pas la même que nOmDuJoUeUr.

Je vous conseille de faire des noms de variable complets car quand qu'il y a 4 variables dans un programme, si elles s'appellent a,b,c et d, on ne s'y retrouve plus.

Il y a différents styles pour écrire ses variables, je vous recommande soit d'écrire vos noms de variables tout en minuscules avec de _ pour les espaces (style gnu) soit d'écrire tout en minuscule sans rien pour représenter les espaces (style C traditionnel). Personnellement j'utilise le style gnu.

Il existe plusieurs types de variables, les principaux sont les suivants :

char : les nombres de -128 à 128, on y met aussi en général les caractères (taille : 1 byte)

int : les nombres de -2 147 483 648 à 2 147 483 647, on y met parfois des caractères (taille : 4 byte)

long : les nombres de -2 147 483 648 à 2 147 483 647 (taille : 4 byte)

float : les nombres décimaux de -3.4×10^{38} à 3.4×10^{38} (taille : 4 byte)

double : les nombres décimaux de -1.7×10^{308} à 1.7×10^{308} (taille : 8 byte)

Vous avez remarqué que le type int et le type long sont exactement les mêmes, il n'en a pas toujours été ainsi, mais il est vrai que de nos jours nous avons tellement de mémoire ... (Les int étaient autrefois plus petits que les long, pour des raisons de compatibilité avec des anciens programmes on a gardé ces deux types) Cependant j'utilise les long pour vraiment stocker des nombres et les int pour les nombres booléens (vous saurez ce que c'est plus tard).

Je précise que vous ne pouvez pas utiliser la virgule pour les nombres décimaux mais le point, pour écrire 5,8 vous taperez 5.8 (perso c'est ce que je fait tout le temps).

Dernière précision avant de vraiment passer au C : tous les types permettant de stocker des nombres entiers sont par défaut signés (signed), cela veut dire qu'ils peuvent être négatif ou positif, on peut écrire unsigned pour qu'ils ne puissent contenir que des nombres positifs, et ainsi pouvoir stocker des nombres deux fois plus grands (par exemple un long est compris entre -2 147 483 648 et 2 147 483 647 et un unsigned long entre 0 et 4 294 967 295).

4/ Passons à la pratique

D'abord comment dire à l'ordinateur qu'on voudrais créer une variable ?

Et bien on écrit au début du programme (de préférence) ceci :

```
type_de_variable nom_de_variable;
```

et l'ordinateur crée une variable.

Quelle est la valeur de cette variable ?

Elle est inconnue, c'est la valeur que le dernier programme qui a utilisé l'adresse de notre variable a laissé, cela peut être zéro si le programme a laissé un zéro ou si cette adresse n'a pas été utilisé depuis le démarrage de l'ordinateur.

Comment donner une valeur à notre variable ?

En écrivant :

```
nom_de_variable = valeur;
```

les espaces ne sont pas indispensables, ils servent à aérer le code.

Faisons donc un petit code :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long variable;
    variable = 0;
    return 0;
}
```

nous avons créé une variable variable et lui avons donné pour valeur 0, on aurait aussi pu faire :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long variable = 0;
    return 0;
}
```

on peut changer la valeur d'une variable à volonté.

Le fait de dire à l'ordinateur de créer une variable s'appelle la déclaration de variable.

On peut déclarer plusieurs variables à la fois, il suffit de les séparer par des virgules :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long var = 0, var2, var3 = 274, var4 = var2 + var3;
    return 0;
}
```

Comme nous l'avons vu nous pouvons donner une valeur à une variable n'importe quand, mais maintenant imaginons que nous puissions proposer à l'utilisateur d'entrer une variable, ce serait génial car nous pourrions interagir avec ce dernier. Et bien aussi étonnant que cela puisse paraître, nous pouvons le faire !

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long variable = 0;
    printf("entrez la variable : ");
    scanf("%ld", &variable);
    return 0;
}
```

Ce programme écrit "entrez la variable : " et vous propose d'écrire quelque chose, vous devez absolument entrer un nombre.

Analysons la fonction scanf(), on lui donne deux choses : "%ld" et le nom de la variable précédé de & (le nom de la variable précédé de & correspond en fait à l'adresse de la variable, mais nous ne verrons bien cette notion que dans longtemps) ceci est très important car sinon votre programme a 99.999999999...% de chances de planter (il ne marchera que si on écrit d'abord par exemple IMG "variable = &variable;" mais ce serait stupide et de toute façon ça rallongerait le code).

Expliquons ces deux choses : %ld indique à scanf() [font] et [font=courier]printf() la présence d'un nombre de type long à demander pour le premier et à afficher pour le second (on verra plus précisément comment afficher une variable dans quelques lignes) on doit donc envoyer "%ld" à scanf() pour lui dire qu'elle va écrire dans un long (les guillemets indiquent une chaîne de caractères, en l'occurrence %ld) quand à &variable, nous l'avons brièvement expliqué plus haut, cela indique à scanf() où écrire son long.

Maintenant écrivons notre variable à l'écran, si vous avez bien compris vous devriez écrire ça :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long variable = 0;
    printf("entrez la variable : ");
    scanf("%ld", &variable);
    printf("la variable vaut : %ld", &variable);
    return 0;
}
```

Seulement vous pouvez constater que le programme ne donne pas la valeur de la variable mais une autre valeur que vous ne comprenez sûrement pas, cette valeur est l'adresse de variable, vous savez que le & indique que l'on veut l'adresse, et bien si scanf() travaille avec l'adresse de la variable

qu'elle doit modifier, printf() fonctionne directement avec la variable qu'elle doit afficher. On doit donc écrire :

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long variable = 0;
    printf("entrez la variable : ");
    scanf("%ld",&variable);
    printf("la variable vaut : %ld",variable);
    return 0;
}
```

Et enfin ça marche !!!

Voilà, terminons par une liste des différentes "instructions" printf() et scanf() (cette liste n'a pas pour vocation d'être exhaustive) :

%d pour les nombres entiers décimaux contenus dans des variables int ou char

%ld pour les nombres entiers décimaux contenus dans des variables long

%p pour les nombres entiers hexadécimaux contenus dans des variables int ou char (permet d'afficher ou de demander un chiffre en hexadécimal)

%lp pour les nombres entiers hexadécimaux contenus dans des variables long (permet d'afficher ou de demander un chiffre en hexadécimal)

%f pour des nombres "à virgule" contenus dans des variables float

%lf pour des nombres "à virgule" contenus dans des variables double

%c pour des caractères

%s pour des chaînes de caractères (ne pas utiliser tant que vous n'avez pas vu le cours sur les chaînes de caractères)

J'ajoute que vous pouvez utiliser la fonction getchar() pour demander un caractère de la façon suivante :

```
variable = getch()
```

bien sur vous pouvez aussi utiliser scanf() mais je préfère que vous choisissiez getchar() pour les caractères.

Voilà, vous savez maintenant gérer la mémoire, pour l'instant ça ne sert pas à grand chose (un peu quand même) mais vous verrez son incroyable utilité lorsque nous verrons les conditions et les fonctions.

Exercices

A partir de cette leçon je vous propose des exercices, vous pourrez demander de l'aide sur le forum mais j'interdis quiconque de donner la solution, seulement de l'aide, un message "aide à la correction" dans lequel je vous dirai ce que doit donner le programme sera par ailleurs posté (ainsi vous comparerez les bons résultats aux vôtres pour savoir si votre programme est juste)

Exercice 1 : Vous écrirez un programme qui demande d'entrer un caractère et qui affiche le chiffre

correspondant (par exemple si j'entre un 'a' je devrais obtenir '97').

Indice : il suffit de demander un caractère, de le stocker dans une variable puis d'afficher cette variable sous forme de nombre.

Exercice 2 : faites le contraire (on entre un nombre et le programme nous donne un le caractère correspondant)

PS : je vous invite à aller sur le forum de top logiciel car cela vous permetrat de résoudre (ou plutôt de faire résoudre ■■■

) vos problèmes quand à la programmation en C (entre autre) et donc de progresser.

J'ajoute que j'ai d'ailleurs créé une [FAQ C](#) dans le forum C.