



Accueil > Cours > Apprenez à programmer en C ! > Une bête de calcul

Apprenez à programmer en C !

40 heures  Moyenne

Mis à jour le 29/07/2019



Une bête de calcul

 [Connectez-vous](#) ou [inscrivez-vous](#) gratuitement pour bénéficier de toutes les fonctionnalités de ce cours !

Je vous l'ai dit dans le chapitre précédent : votre ordinateur n'est en fait qu'une grosse machine à calculer. Que vous soyez en train d'écouter de la musique, regarder un film ou jouer à un jeu vidéo, votre ordinateur ne fait que des calculs.

Ce chapitre va vous apprendre à réaliser la plupart des calculs qu'un ordinateur sait faire. Nous réutiliserons ce que nous venons tout juste d'apprendre, à savoir les variables. L'idée, c'est justement de faire des calculs avec vos variables : ajouter des variables entre elles, les multiplier, enregistrer le résultat dans une autre variable, etc.

Même si vous n'êtes pas fan des mathématiques, ce chapitre vous sera absolument indispensable.

Les calculs de base



Il faut savoir qu'en plus de n'être qu'une vulgaire calculatrice, votre ordinateur est une calculatrice très basique puisqu'on ne peut faire que des opérations très simples :

- addition ;
- soustraction ;
- multiplication ;
- division ;
- modulo (je vous expliquerai ce que c'est si vous ne savez pas, pas de panique).

Si vous voulez faire des opérations plus compliquées (des carrés, des puissances, des logarithmes et autres joyeusetés) il vous faudra les programmer, c'est-à-dire **expliquer à l'ordinateur comment les**

faire.

Fort heureusement, nous verrons plus loin dans ce chapitre qu'il existe une bibliothèque mathématique livrée avec le langage C qui contient des fonctions mathématiques toutes prêtes. Vous n'aurez donc pas à les réécrire, à moins que vous souhaitiez volontairement passer un sale quart d'heure (ou que vous soyez prof de maths).

Voyons donc l'addition pour commencer.

Pour faire une addition, on utilise le signe + (sans blague !).

Vous devez mettre le résultat de votre calcul dans une variable. On va donc par exemple créer une variable `resultat` de type `int` et faire un calcul :

```
1 int resultat = 0;  
2  
3 resultat = 5 + 3;
```

Pas besoin d'être un pro du calcul mental pour deviner que la variable `resultat` contiendra la valeur 8 après exécution.

Bien sûr, rien ne s'affiche à l'écran avec ce code. Si vous voulez voir la valeur de la variable, rajoutez un `printf` comme vous savez maintenant si bien le faire :

```
1 printf("5 + 3 = %d", resultat);
```

À l'écran, cela donnera :

5 + 3 = 8

Voilà pour l'addition.

Pour les autres opérations, c'est la même chose, seul le signe utilisé change (voir tab. suivante).

Opération	Signe
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Si vous avez déjà utilisé la calculatrice sur votre ordinateur, vous devriez connaître ces signes. Il n'y a pas de difficulté particulière pour ces opérations, à part pour les deux dernières (la division et le

modulo). Nous allons donc parler un peu plus en détail de chacune d'elles.

La division

Les divisions fonctionnent normalement sur un ordinateur quand il n'y a pas de reste. Par exemple, `6 / 3` font 2, votre ordinateur vous donnera la réponse juste. Jusque-là pas de souci.

Mais prenons maintenant une division avec reste comme `5 / 2` ... Le résultat devrait être 2.5. Et pourtant ! Regardez ce que fait ce code :

```
1 int resultat = 0;
2
3 resultat = 5 / 2;
4 printf ("5 / 2 = %d", resultat);
```

5 / 2 = 2

Il y a un gros problème. On a demandé `5 / 2`, on s'attend à avoir 2.5, et l'ordinateur nous dit que ça fait 2 !

Il y a anguille sous roche. Nos ordinateurs seraient-ils stupides à ce point ?

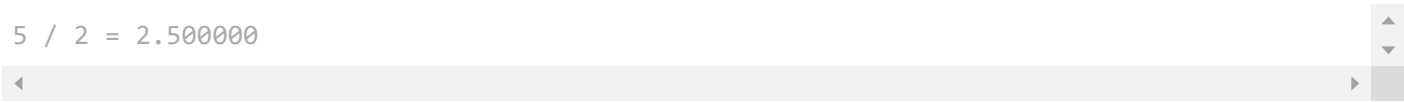
En fait, quand il voit les chiffres 5 et 2, votre ordinateur fait une division de nombres entiers (aussi appelée « division euclidienne »). Cela veut dire qu'il tronque le résultat, il ne garde que la partie entière (le 2).

Hé mais je sais pourquoi ! C'est parce que `resultat` est un `int` ! Si ça avait été un `double`, il aurait pu stocker un nombre décimal à l'intérieur !

Eh non, ce n'est pas la raison ! Essayez le même code en transformant juste `resultat` en `double`, et vous verrez qu'on vous affiche quand même 2. Parce que les nombres de l'opération sont des nombres entiers, l'ordinateur répond par un nombre entier.

Si on veut que l'ordinateur affiche le bon résultat, il va falloir transformer les nombres 5 et 2 de l'opération en nombres décimaux, c'est-à-dire écrire 5.0 et 2.0 (ce sont les mêmes nombres, mais l'ordinateur considère que ce sont des nombres décimaux, donc il fait une division de nombres décimaux) :

```
1 double resultat = 0;
2
3 resultat = 5.0 / 2.0;
4 printf ("5 / 2 = %lf", resultat);
```

5 / 2 = 2.500000

Là, le nombre est correct. Bon : il affiche des tonnes de zéros derrière si ça lui chante, mais le résultat reste quand même correct.

Cette propriété de la division de nombres entiers est très importante. Il faut que vous reteniez que pour un ordinateur :

- $5 / 2 = 2$;
- $10 / 3 = 3$;
- $4 / 5 = 0$.

C'est un peu surprenant, mais c'est sa façon de calculer avec des entiers.

Si vous voulez avoir un résultat décimal, il faut que les nombres de l'opération soient décimaux :

- $5.0 / 2.0 = 2.5$;
- $10.0 / 3.0 = 3.33333$;
- $4.0 / 5.0 = 0.8$.

En fait, en faisant une division d'entiers comme $5 / 2$, votre ordinateur répond à la question « Combien y a-t-il de fois 2 dans le nombre 5 ? ». La réponse est deux fois. De même, « combien de fois y a-t-il le nombre 3 dans 10 ? » Trois fois.

Mais alors me direz-vous, comment on fait pour récupérer le reste de la division ? C'est là que super-modulo intervient.

Le modulo

Le modulo est une opération mathématique qui permet d'obtenir **le reste d'une division**. C'est peut-être une opération moins connue que les quatre autres, mais pour votre ordinateur ça reste une opération de base... probablement pour justement combler le problème de la « division d'entiers » qu'on vient de voir.

Le modulo, je vous l'ai dit tout à l'heure, se représente par le signe `%`.

Voici quelques exemples de modulus :

- $5 \% 2 = 1$;
- $14 \% 3 = 2$;
- $4 \% 2 = 0$.

Le modulo $5 \% 2$ est le reste de la division $5 / 2$, c'est-à-dire 1. L'ordinateur calcule que $5 = 2 * 2 + 1$ (c'est ce 1, le reste, que le modulo renvoie).

De même, $14 \% 3$, le calcul est $14 = 3 * 4 + 2$ (modulo renvoie le 2). Enfin, pour $4 \% 2$, la division tombe juste, il n'y a pas de reste, donc modulo renvoie 0.

Voilà, il n'y a rien à ajouter au sujet des modulus. Je tenais juste à l'expliquer à ceux qui ne connaîtraient pas.

En plus j'ai une bonne nouvelle : on a vu toutes les opérations de base. Finis les cours de maths !

Des calculs entre variables

Ce qui serait intéressant, maintenant que vous savez faire les cinq opérations de base, ce serait de s'entraîner à faire des calculs entre plusieurs variables.

En effet, rien ne vous empêche de faire :

```
1 resultat = nombre1 + nombre2;
```

Cette ligne fait la somme des variables `nombre1` et `nombre2`, et stocke le résultat dans la variable `resultat`.

Et c'est là que les choses commencent à devenir très intéressantes. Tenez, il me vient une idée. Vous avez maintenant déjà le niveau pour réaliser une mini-calculatrice. Si, si, je vous assure !

Imaginez un programme qui demande deux nombres à l'utilisateur. Ces deux nombres, vous les stockez dans des variables.

Ensuite, vous faites la somme de ces variables et vous stockez le résultat dans une variable appelée `resultat`. Vous n'avez plus qu'à afficher le résultat du calcul à l'écran, sous les yeux ébahis de l'utilisateur qui n'aurait jamais été capable de calculer cela de tête aussi vite.

Essayez de coder vous-mêmes ce petit programme, c'est facile et ça vous entraînera !

La réponse est ci-dessous :

```
1 int main(int argc, char *argv[])
2 {
3     int resultat = 0, nombre1 = 0, nombre2 = 0;
4
5     // On demande les nombres 1 et 2 à l'utilisateur :
6
7     printf("Entrez le nombre 1 : ");
8     scanf("%d", &nombre1);
9     printf("Entrez le nombre 2 : ");
10    scanf("%d", &nombre2);
11
12    // On fait le calcul :
13
14    resultat = nombre1 + nombre2;
15
16    // Et on affiche l'addition à l'écran :
17
18    printf ("%d + %d = %d\n", nombre1, nombre2, resultat);
19
20    return 0;
21 }
```

Entrez le nombre 1 : 30

Entrez le nombre 2 : 25

30 + 25 = 55

Sans en avoir l'air, on vient de faire là notre premier programme ayant un intérêt. Notre programme est capable d'additionner deux nombres et d'afficher le résultat de l'opération !

Vous pouvez essayer avec n'importe quel nombre (du moment que vous ne dépassez pas les limites d'un type `int`), votre ordinateur effectuera le calcul en un éclair. Encore heureux, parce que des opérations comme ça, il doit en faire des milliards en une seule seconde !

Je vous conseille de faire la même chose avec les autres opérations pour vous entraîner (soustraction, multiplication...). Vous ne devriez pas avoir trop de mal vu qu'il y a juste un ou deux signes à changer. Vous pouvez aussi ajouter une troisième variable et faire l'addition de trois variables à la fois, ça fonctionne sans problème :

```
1 resultat = nombre1 + nombre2 + nombre3;
```

Les raccourcis

Comme promis, nous n'avons pas de nouvelles opérations à voir. Et pour cause ! Nous les connaissons déjà toutes. C'est avec ces simples opérations de base que vous pouvez tout créer. Il n'y a pas besoin d'autres opérations. Je reconnais que c'est difficile à avaler, se dire qu'un jeu 3D ne fait rien d'autre au final que des additions et des soustractions, pourtant... c'est la stricte vérité.

Ceci étant, il existe en C des techniques permettant de raccourcir l'écriture des opérations. Pourquoi utiliser des raccourcis ? Parce que, souvent, on fait des opérations répétitives. Vous allez voir ce que je veux dire par là tout de suite, avec ce qu'on appelle l'incrémement.

L'incrémement

Vous verrez que vous serez souvent amenés à ajouter 1 à une variable. Au fur et à mesure du programme, vous aurez des variables qui augmentent de 1 en 1.

Imaginons que votre variable s'appelle `nombre` (nom très original, n'est-ce pas ?). Sauriez-vous comment faire pour ajouter 1 à cette variable, sans savoir quel est le nombre qu'elle contient ?

Voici comment on doit faire :

```
1 nombre = nombre + 1;
```

Que se passe-t-il ici ? On fait le calcul `nombre + 1`, et on range ce résultat dans la variable... `nombre` ! Du coup, si notre variable `nombre` valait 4, elle vaut maintenant 5. Si elle valait 8, elle vaut maintenant 9, etc.

Cette opération est justement répétitive. Les informaticiens étant des gens particulièrement fainéants, ils n'avaient guère envie de taper deux fois le même nom de variable (ben oui quoi, c'est fatigant !).

Ils ont donc inventé un raccourci pour cette opération qu'on appelle **l'incrémentation**. Cette instruction produit exactement le même résultat que le code qu'on vient de voir :

```
1 nombre++;
```

Cette ligne, bien plus courte que celle de tout à l'heure, signifie « Ajoute 1 à la variable `nombre` ». Il suffit d'écrire le nom de la variable à incrémenter, de mettre deux signes +, et bien entendu, de ne pas oublier le point-virgule.

Mine de rien, cela nous sera bien pratique par la suite car, comme je vous l'ai dit, on sera souvent amenés à faire des incrémentations (c'est-à-dire ajouter 1 à une variable).

Si vous êtes perspicaces, vous avez d'ailleurs remarqué que ce signe ++ se trouve dans le nom du langage C++. C'est en fait un clin d'œil des programmeurs, et vous êtes maintenant capables de le comprendre ! C++ signifie qu'il s'agit du langage C « incrémenté », c'est-à-dire si on veut « du langage C à un niveau supérieur ». En pratique, le C++ permet surtout de programmer différemment mais il n'est pas « meilleur » que le C : juste différent.

La décrémentation

C'est tout bêtement l'inverse de l'incrémentation : on enlève 1 à une variable.

Même si on fait plus souvent des incrémentations que des décrétements, cela reste une opération pratique que vous utiliserez de temps en temps.

La décrémentation, si on l'écrit en forme « longue » :

```
1 nombre = nombre - 1;
```

Et maintenant en forme « raccourcie » :

```
1 nombre--;
```

On l'aurait presque deviné tout seul ! Au lieu de mettre un ++, vous mettez un - - : si votre variable vaut 6, elle vaudra 5 après l'instruction de décrémentation.

Les autres raccourcis

Il existe d'autres raccourcis qui fonctionnent sur le même principe. Cette fois, ces raccourcis fonctionnent pour toutes les opérations de base : + - * / % .

Cela permet là encore d'éviter une répétition du nom d'une variable sur une même ligne. Ainsi, si vous voulez multiplier par deux une variable :

```
1 nombre = nombre * 2;
```

Vous pouvez l'écrire d'une façon raccourcie comme ceci :

```
1 nombre *= 2;
```

Si le nombre vaut 5 au départ, il vaudra 10 après cette instruction.

Pour les autres opérations de base, cela fonctionne de la même manière. Voici un petit programme d'exemple :

```
1 int nombre = 2;
2
3 nombre += 4; // nombre vaut 6...
4 nombre -= 3; // ... nombre vaut maintenant 3
5 nombre *= 5; // ... nombre vaut 15
6 nombre /= 3; // ... nombre vaut 5
7 nombre %= 3; // ... nombre vaut 2 (car 5 = 1 * 3 + 2)
```

(Ne boudez pas, un peu de calcul mental n'a jamais tué personne !)

L'avantage ici est qu'on peut utiliser toutes les opérations de base, et qu'on peut ajouter, soustraire, multiplier par n'importe quel nombre.

Ce sont des raccourcis à connaître si vous avez un jour des lignes répétitives à taper dans un programme.

Retenez quand même que l'incréméntation reste de loin le raccourci le plus utilisé.

La bibliothèque mathématique



En langage C, il existe ce qu'on appelle des bibliothèques « standard », c'est-à-dire des bibliothèques toujours utilisables. Ce sont en quelque sorte des bibliothèques « de base » qu'on utilise très souvent.

Les bibliothèques sont, je vous le rappelle, des ensembles de fonctions toutes prêtes. Ces fonctions ont été écrites par des programmeurs avant vous, elles vous évitent en quelque sorte d'avoir à réinventer la roue à chaque nouveau programme.

Vous avez déjà utilisé les fonctions `printf` et `scanf` de la bibliothèque `stdio.h`.

Il faut savoir qu'il existe une autre bibliothèque, appelée `math.h`, qui contient de nombreuses fonctions mathématiques toutes prêtes.

En effet, les cinq opérations de base que l'on a vues sont loin d'être suffisantes ! Bon, il se peut que vous n'ayez jamais besoin de certaines opérations complexes comme les exponentielles. Si vous ne savez pas ce que c'est, c'est que vous êtes peut-être un peu trop jeunes ou que vous n'avez pas

assez fait de maths dans votre vie. Toutefois, la bibliothèque mathématique contient de nombreuses autres fonctions dont vous aurez très probablement besoin.

Tenez par exemple, on ne peut pas faire de puissances en C ! Comment calculer un simple carré ? Vous pouvez toujours essayer de taper `52` dans votre programme, mais votre ordinateur ne le comprendra jamais car il ne sait pas ce que c'est... À moins que vous le lui expliquiez en lui indiquant la bibliothèque mathématique !

Pour pouvoir utiliser les fonctions de la bibliothèque mathématique, il est indispensable de mettre la directive de préprocesseur suivante en haut de votre programme :

```
1 #include <math.h>
```

Une fois que c'est fait, vous pouvez utiliser toutes les fonctions de cette bibliothèque.

J'ai justement l'intention de vous les présenter.

Bon : comme il y a beaucoup de fonctions, je ne peux pas en faire la liste complète ici. D'une part ça vous ferait trop à assimiler, et d'autre part mes pauvres petits doigts auraient fondu avant la fin de l'écriture du chapitre. Je vais donc me contenter des fonctions principales, c'est-à-dire celles qui me semblent les plus importantes.

Vous n'avez peut-être pas tous le niveau en maths pour comprendre ce que font ces fonctions. Si c'est votre cas, pas d'inquiétude. Lisez juste, cela ne vous pénalisera pas pour la suite.

Ceci étant, je vous offre un petit conseil gratuit : soyez attentifs en cours de maths, on ne dirait pas comme ça, mais en fait ça finit par servir !

fabs

Cette fonction retourne la valeur absolue d'un nombre, c'est-à-dire $|x|$ (c'est la notation mathématique).

La valeur absolue d'un nombre est sa valeur positive :

- si vous donnez -53 à la fonction, elle vous renvoie 53 ;
- si vous donnez 53 à la fonction, elle vous renvoie 53.

En bref, elle renvoie toujours l'équivalent positif du nombre que vous lui donnez.

```
1 double absolu = 0, nombre = -27;  
2  
3 absolu = fabs(nombre); // absolu vaudra 27
```

Cette fonction renvoie un `double`, donc votre variable `absolu` doit être de type `double`.

Il existe aussi une fonction similaire appelée `abs`, située cette fois dans `stdlib.h`.

La fonction `abs` marche de la même manière, sauf qu'elle utilise des entiers (`int`). Elle renvoie donc un nombre entier de type `int` et non un `double` comme `fabs`.

ceil

Cette fonction renvoie le premier nombre entier après le nombre décimal qu'on lui donne. C'est une sorte d'arrondi. On arrondit en fait toujours au nombre entier supérieur.

Par exemple, si on lui donne 26.512, la fonction renvoie 27.

Cette fonction s'utilise de la même manière et renvoie un `double` :

```
1 double dessus = 0, nombre = 52.71;
2
3 dessus = ceil(nombre); // dessus vaudra 53
```

floor

C'est l'inverse de la fonction précédente : cette fois, elle renvoie le nombre directement en dessous.

Si vous lui donnez 37.91, la fonction `floor` vous renverra donc 37.

pow

Cette fonction permet de calculer la puissance d'un nombre. Vous devez lui indiquer deux valeurs : le nombre et la puissance à laquelle vous voulez l'élever. Voici le schéma de la fonction :

```
1 pow(nombre, puissance);
```

Par exemple, « 2 puissance 3 » (que l'on écrit habituellement 2^3 sur un ordinateur), c'est le calcul $2 * 2 * 2$, ce qui fait 8 :

```
1 double resultat = 0, nombre = 2;
2
3 resultat = pow(nombre, 3); // resultat vaudra 2^3 = 8
```

Vous pouvez donc utiliser cette fonction pour calculer des carrés. Il suffit d'indiquer une puissance de 2.

sqrt

Cette fonction calcule la racine carrée d'un nombre. Elle renvoie un `double`.

```
1 double resultat = 0, nombre = 100;
2
3 resultat = sqrt(nombre); // resultat vaudra 10
```

sin, cos, tan

Ce sont les trois fameuses fonctions utilisées en trigonométrie.

Le fonctionnement est le même, ces fonctions renvoient un `double`.

Ces fonctions attendent une valeur en **radians**.

asin, acos, atan

Ce sont les fonctions arc sinus, arc cosinus et arc tangente, d'autres fonctions de trigonométrie.

Elles s'utilisent de la même manière et renvoient un `double`.

exp

Cette fonction calcule l'exponentielle d'un nombre. Elle renvoie un `double` (oui, oui, elle aussi).

log

Cette fonction calcule le logarithme népérien d'un nombre (que l'on note aussi « ln »).

log10

Cette fonction calcule le logarithme base 10 d'un nombre.

En résumé

- Un ordinateur n'est en fait qu'une **calculatrice géante** : tout ce qu'il sait faire, ce sont des opérations.
- Les opérations connues par votre ordinateur sont très **basiques** : l'addition, la soustraction, la multiplication, la division et le modulo (il s'agit du reste de la division).
- Il est possible d'**effectuer des calculs entre des variables**. C'est d'ailleurs ce qu'un ordinateur sait faire de mieux : il le fait bien et vite.
- L'**incrément** est l'opération qui consiste à ajouter 1 à une variable. On écrit `variable++`.
- La **décrément** est l'opération inverse : on retire 1 à une variable. On écrit donc `variable--`.
- Pour augmenter le nombre d'opérations connues par votre ordinateur, il faut charger la **bibliothèque mathématique** (c'est-à-dire `#include <math.h>`).
- Cette bibliothèque contient des **fonctions mathématiques plus avancées**, telles que la puissance, la racine carrée, l'arrondi, l'exponentielle, le logarithme, etc.

[UN MONDE DE VARIABLES](#)[LES CONDITIONS](#)

Le professeur

Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...



Livre



PDF

OpenClassrooms

L'entreprise

Alternance

Forum

Blog

Nous rejoindre

Entreprises

Business

En plus

Devenez mentor

Aide et FAQ

Conditions Générales d'Utilisation

Politique de Protection des Données Personnelles

Nous contacter



Français



