On l'a vu, notre ordinateur est en premier lieu une machine à calculer. Quelque soit l'activité en cours sur la machine, tout ce qu'elle fait est réaliser des calculs. Dans ce chapitre, on va apprendre à ordonner certains calculs dont les ordinateurs sont capables.

On utilisera le contenus des chapitres I. et II. pour :

- ajouter et multiplier les valeurs entre elles,
- enregistrer le résultat dans une autre variable...

Les calculs de base

Voici les calculs que notre ordinateur peut réaliser :

- Addition ;
- Soustraction ;
- Multiplication ;
- Division ;
- Modulo (le "reste" des divisions euclidiennes)
- Carrés,
- Puissances ;
- Logarithmes...

On va programmer ces calculs, c'est-à-dire expliquer à l'ordinateur **comment réaliser ces calculs.**

Dieu merci, il existe une bibliothèque mathématique pour le langage C ; on n'aura donc pas besoin de les réecrire, car nous ne sommes pas masochistes.

Les signes mathématiques

Voici les signes à utiliser pour chaque opération en C :

Operation	Sign
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

Pour une addition, on utilise donc `>| +`. Le calcul doit être placé dans une variable, qu'on va par exemple déclarer en `>| int` et nommer `>| somme` :

```
`int somme = 0;
somme = 6 + 4`
```

Evidemment, le résultat ne s'affichera pas à l'écran ; il faut donc indiquer au programme qu'on veut afficher le résultat :

```
{   int somme = 0;
somme = 40 + 58;
printf ("Le résultat de l'opération est %d", somme);
return 0;}

// Résultat :

cd "c:\Users\coral\Desktop\School\Code\" && gcc math.c -o math && "c:\Users\coral\Desktop\School\Code\"math
Le résultat de l'opération est 98
```

La méthode est identique pour les autres opérations de ce type, tant que les nombres sont entiers. Quelques exemples ci-après :

```
    { int somme = 0;
    somme = 18 - 7;
    printf ("Le résultat de l'opération est %d", somme);
      return 0; }

      // Résultat

      cd "c:\Users\coral\Desktop\School\Code\" && gcc math.c -o math && "c:\Users\coral\Desktop\School\Code\"math
      Le résultat de l'opération est 11
```

En revanche, si on essaye de faire de même avec une division, par exemple, on s'aperçoit rapidement qu'il y a un soucis :

```
{   int somme = 0;
    somme = 5 / 2;
    printf ("Le résultat de l'opération est %d", somme);
    return 0;}

        // Résultat

    cd "c:\Users\coral\Desktop\School\Code\" && gcc math.c -o math && "c:\Users\coral\Desktop\School\Code\"math
    Le résultat de l'opération est 2
```

C'est bien évidément faux ! Mais alors, pourquoi se passe-t-il ceci ?

> C'est tout simple ! En fait, la machine voit ici des nombres entiers ; elle répond donc avec un autre nombre entier, quitte à tronquer les décimales. C'est une **division euclidienne** . Contrairement à ce qu'on pourrait penser, il ne suffit pas de modifier la variable pour lui faire accepter les nombres décimaux. Il faudra également déclarer les nombres du calcul comme des nombres décimaux !

Pour réaliser une division non-euclidienne, c'est-à-dire avec des nombres décimaux, on utilise donc `> double` , et on spécifie les décimales des chiffres entiers, c'est-à-dire qu'on dit `> 5.00` au lieu de `> 5` :

```
{   double somme = 0;
    somme = 5.00 / 2.00;
    printf ("Le résultat de l'opération est %lf", somme); // on oublie                pas d'utiliser %lf au lieu
de %d pour récupérer le résultat de la                  variable.
    return 0;}

        // Résultat :
        cd "c:\Users\coral\Desktop\School\Code\" && gcc math.c -o math &&
"c:\Users\coral\Desktop\School\Code\"math
        Le résultat de l'opération est 2.500000
```

Faire des calculs entre variables

Entraînons-nous à faire des calculs entre plusieurs variables !

La ligne `> resultat = nombre1 + nombre2;` fait la somme des deux variables `> nombre` et stocke le résultat dans la variable `> resultat`.

On va maintenant programmer une calculette toute simple.

Exercice On va coder une calculette qui permet de faire des additions. Pour ce faire :

- On demande deux nombres à l'utilisateur,
- On fait la somme des variables, qu'on stocke dans une troisième variable,
- Puis on affiche le résultat à l'utilisateur.

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

int main(int argc, char *argv[])
{
        int somme = 0, nombre1 = 0, nombre2 = 0; // on déclare toutes les variables du programme
        printf ("Indiquez le chiffre entier de votre choix\n");
        scanf ("%d", &nombre1); // l'utilisateur saisi sa première valeur
        printf ("Super ! Maintenant, un autre !\n");
        scanf ("%d", &nombre2); // l'utilisateur saisi sa deuxième valeur
        somme = nombre1 + nombre2; // le programme détermine la nouvelle valeur de la variable somme
        printf ("La somme de %d et %d est %d ! \n", nombre1, nombre2, somme); // le programme récupère toutes les
valeurs et les affiche pour l'utilisateur
        return 0;

        // Résultats :

        PS C:\Users\coral\Desktop\School\Code> .\calculator
        Indiquez le chiffre entier de votre choix
        65 // valeur de l'utilisateur pour nombre1
        Super ! Maintenant, un autre !
        57 // idem pour nombre2
        La somme de 65 et 57 est 122 ! // affiche les valeurs de nombre1 et nombre2, puis la valeur de somme.
}
```

On peut évidemment faire de même avec les autres types de calculs qu'on a vu, ou ajouter des valeurs pour permettre à l'utilisateur de calculer davantage de variables.

Quelques raccourcis

L'objectif principal du code est de s'économiser autant que possible, surtout pour éviter la répétition des opérations. Il existe donc pléthores de raccourcis : en voilà quelques uns.

Incrémenter et décrémenter une valeur

Il est courant d'avoir des variables qui augmentent au cours du programme, de 1 en 1. On appelle cela l'incrémentation.

Admettons une variable `nombre` Pour ajouter 1 à valeur précédente de `nombre`, on peut bien sûr écrire `nombre = nombre + 1;`, mais pour s'économiser, on utilisera plutôt `nombre++;` . Et voilà, on a incrémenté `nombre` : sa valeur est désormais sa valeur précédente + 1. Si elle valait 8, elle vaut maintenant 9.

On peut évidemment faire de même dans l'autre sens, c'est à dire retirer 1 à la valeur précédente ; pour ce faire, au lieu de taper `nombre = nombre - 1`;, on tapera `nombre--;`.