

Les bases du C/C++



SOMMAIRE

<u>La structure d'un programme.....</u>	<u>1-2</u>
<u>La compilation.....</u>	<u>3</u>
<u>Un monde de variables.....</u>	<u>4-8</u>
<u>Les calculs.....</u>	<u>9</u>
<u>Raccourcis utiles.....</u>	<u>10</u>
<u>Les conditions.....</u>	<u>11-14</u>
<u>Les boucles.....</u>	<u>15-19</u>
<u>Les fonctions.....</u>	<u>20-21</u>

La structure d'un programme

Le code minimal

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

} Directives du préprocesseur

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

} Fonction principale du programme

Instructions

La structure d'un programme

- Deux fichiers :

PROGRAMME.CPP

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

int main (int argc, char *argv[]) {
    // Affiche à l'écran « Hello world! »
    printf("Hello world!\n");
    return 0;
}
```

PROGRAMME.H

```
#ifndef Main
#define Main

// La ligne suivante est l'énoncé de
// la fonction principale
int main (int argc, char *argv[]);

#endif
```

La compilation

Pour que le programme devienne exécutable (et donc qu'on puisse 'voir' ce qu'il fait), il faut pouvoir le **compiler**.

La compilation est différente en fonction du **langage de programmation** dans lequel on code notre programme.

Compiler en C (dans le terminal)

```
$ gcc -o nom_de_l'exécutable nom_du_fichier_source
```

Ex:

```
$ gcc -o le_pendu le_pendu.c
```

Compiler en C++ (dans le terminal)

```
$ g++ -o nom_de_l'exécutable nom_du_fichier_source
```

Un monde de variables

Dans tous les programmes : besoin de variables pour stocker des informations, les modifier ...

Plusieurs **types** de variables qui vont prendre plus ou moins de place en mémoire :

Nom du type	Nombres stockables
char	-128 à 127
int	-2 147 483 648 à 2 147 483 647
long	-2 147 483 648 à 2 147 483 647
float	-3.4 x 10 puissance 38 à 3.4 x 10 puissance 38
double	-1.7 x 10 puissance 308 à 1.7 x 10 puissance 308

Un monde de variables

Au club, on va plutôt utiliser : **char** pour les caractères, **int** pour les entiers et **double** pour les décimaux.

Utile : les nombres non signés (seulement +)

unsigned char	0 à 255
unsigned int	0 à 4 294 967 295
unsigned long	0 à 4 294 967 295

Nouveauté du C++ par rapport au C : les **booléens**

bool	true false
------	--------------

Un monde de variables

Déclarer une variable :

```
int noteDeMaths;  
double sommeArgentRecu;  
unsigned int personnesEnTrainDeLireUnNomDeVariableUnPeuLong;
```

Initialiser une variable :

```
int noteDeMaths = 18;
```

Modifier une variable :

```
noteDeMaths = 17;
```


Un monde de variables

Les constantes :

```
const int NOTE_DE_MATHS_MOYENNE = 10;
```

Attention à l'écriture: conventions

- ECRITURE_DE_MA_CONSTANTE
- EcritureDeMaFonction
- ecritureDeMaVariable

Un monde de variables

Afficher le contenu d'une variable :

```
printf("Il vous reste %d vies", nombreDeViesEntier);  
printf("Il reste %f kilomètres", nombreDeKmDecimal);  
printf("Il vous reste %d vies au kilomètre %f", nombreEntier, nombreDécimal);
```

Au club robot, quelque chose de plus pratique

```
logDebug("Il vous reste", nombreDeViesEntier, "vies");
```

Info : il existe logDebug0, logDebug1, logDebug2,... pour afficher différentes couleurs sur le terminal.

Les calculs

En C/C++ on peut faire toutes sortes de calculs :

```
int resultat = 0;
```

Addition :

```
resultat = 5 + 3;
```

Multiplication:

```
resultat = 5 * 3;
```

Modulo:

```
resultat = 5 % 3;
```

Soustraction :

```
resultat = 5 - 3;
```

Division :

```
resultat = 5 / 3;
```

On peut aussi faire des calculs entre variables :

```
int nombre1 = 5, nombre2 = 3, resultat = 0;
```

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

Raccourcis utiles

Incrémentation:

resultat = resultat + 1; ⇔ resultat++;

Décrémentation:

resultat = resultat - 1; ⇔ resultat--;

Autres raccourcis :

resultat = resultat * 2; ⇔ resultat *= 2;

resultat = resultat / 2; ⇔ resultat /= 2;

resultat = resultat + 2; ⇔ resultat += 2;

resultat = resultat - 2; ⇔ resultat -= 2;

resultat = resultat % 2; ⇔ resultat %= 2;

Les conditions

If...else :

```
if (/* Votre condition */) {  
    // Instructions à exécuter si la condition est vraie  
}  
else if (/* Deuxième condition */) {  
    // Instructions si la deuxième condition est vraie  
}  
else {  
    // Instructions à exécuter si le reste est faux  
}
```

Exemple :

```
if (age >= 18) // Si l'âge est supérieur ou égal à 18  
{  
    printf ("Vous etes majeur !");  
}  
else // Sinon...  
{  
    printf ("Vous etes mineur !");  
}
```

Les conditions

Quelques symboles à connaître :



Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Plusieurs conditions à la fois :

&&	ET
	OU
!	NON

Les conditions

Le « switch »:

```
switch (age)
{
case 2:
    printf("Salut bebe !");
    break;
case 6:
    printf("Salut gamin !");
    break;
case 12:
    printf("Salut jeune !");
    break;
default:
    printf("Tu as plus de 12 ans");
    break;
}
```

Très utile pour tester des types énumérés !

Les conditions

La condition condensée (ternaire) :

```
if (majeur)
```

```
    age = 18;
```

```
else
```

```
    age = 17;
```

⇔

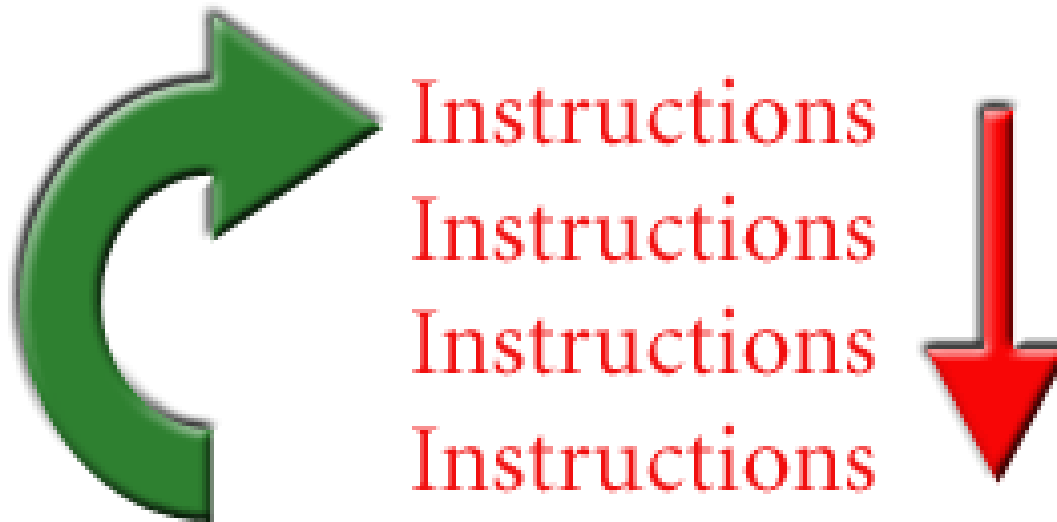
```
age = (majeur) ? 18 : 17;
```



A n'utiliser qu'en cas de besoin d'argument variable dans l'énoncé de fonctions.
Ce code est **illisible** !

Les boucles

Quelque soit le type de boucle, le schéma est le même



Il faut mettre une **condition** pour entrer ou sortir de la boucle !

Les boucles

La boucle **While** (tant que... faire) :

```
while (/* Condition */)  
{  
    // Instructions à répéter  
}
```

Exemple : afficher 10 fois le même message

```
int compteur = 0;  
while (compteur < 10)  
{  
    printf("La variable compteur vaut %d\n", compteur);  
    compteur++;  
}
```



AUX BOUCLES INFINIES...

Les boucles

La boucle **do...while** (faire...tant que) :

- Plus rare
- La condition est vérifiée **à la fin de la boucle.**

Exemple : afficher 10 fois le même message

```
int compteur = 0;
do
{
    printf("Salut les Zeros !\n");
    compteur++;
} while (compteur < 10);
```



POINT VIRGULE à la fin !

Les boucles

La boucle **For** (pour x allant de [.] à [.] faire...) :

```
int compteur;  
for (compteur = 0 ; compteur < 10 ; compteur++)  
{  
    printf("Salut les Zeros !\n");  
}
```

Avantages :

- On peut initialiser le compteur à ce qu'on veut
- Même condition qu'un while
- On peut incrémenter d'autant qu'on veut [ex : compteur += 2;]
- Plus de « compteur++ » dans la boucle

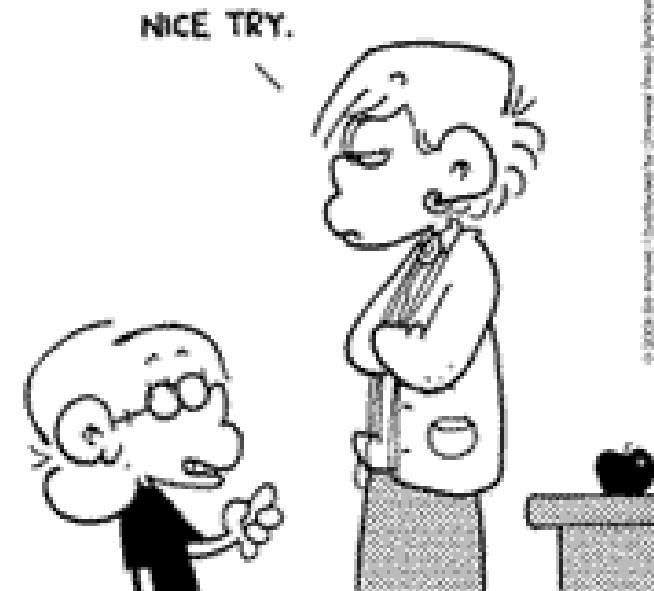


POINT VIRGULE dans la parenthèse et pas virgule !

Les boucles

```
#include <stdio.h>
int main(void)
{
    int count;

    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```



Les fonctions

Deux types de fonctions :

- Les fonctions de base
- Les fonctions que l'on décide de créer

Les fonctions de base :

- Elles sont disponibles dans des **librairies** qu'il faut inclure.
- Les librairies basiques : <stdio.h> et <stdlib.h>
- Librairie de fonctions mathématiques : <maths.h>
- Librairie pour manipuler des chaînes de caractères : <string.h>

Les fonctions que l'on crée :

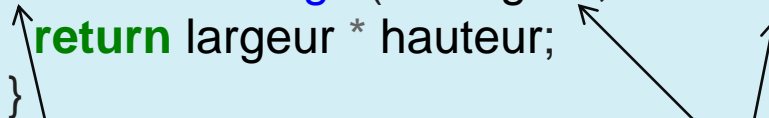
- Elles sont disponibles dans des **headers** qu'il faut inclure aussi.
- Quelques exemples :

```
#include "IAHomologation.h"  
#include "log/Log.h »
```

Les fonctions

Déclaration d'une fonction (fichier .cpp):

```
int aireRectangle(int largeur, int hauteur) {  
    return largeur * hauteur;  
}
```



Type retourné par la fonction

Arguments

Enoncé d'une fonction (fichier .h):

```
int aireRectangle(int largeur, int hauteur);
```

Appel d'une fonction (fichier .cpp):

```
int variable = 0;  
variable = aireRectangle(1500, 200);
```