# TP1 : Introduction au C++ (1)

## Préambule

### Qu'est-ce que le C++ ?

Le C++ est un langage très populaire qui allie puissance et rapidité. Voici quelques qualités du C++ que nous découvrirons au cours de ces TP :

- Il est très **répandu**. C'est un des langages de programmation les plus utilisés internationalement dans l'industrie et la recherche. Il y a donc beaucoup de documentation sur Internet et on peut facilement trouver de l'aide sur les forums.
- Il est très **rapide** (en temps de calcul) et est donc utilisé pour les applications qui ont besoin de performance (jeux vidéo, outils financiers, simulations numériques ...)
- Il est **portable**, c'est-à-dire qu'un même code source peut être transformé rapidement en exécutable sous Windows, Mac OS et Linux.
- Il existe de nombreuses bibliothèques pour le C++, nous en verrons quelques unes dans ces TP. Les bibliothèques sont des extensions pour le langage car de base, le C++ ne fournit que des outils bas-niveau mais en le combinant avec de bonnes bibliothèques, on peut créer des programmes très puissants.
- Il est **multi-paradigmes**, c'est-à-dire qu'on peut programmer de différentes façons en C++. La plus célèbre est la : **Programmation Orientée Objet (POO)**. Cette technique permet de simplifier l'organisation du code dans les programmes et de rendre facilement certains morceaux de codes réutilisables. Plusieurs TP y seront consacrés.

### Quelques liens utiles

N'hésitez pas à les consulter quand vous êtes bloqués ...

- Un polycopié (écrit par K. Santugini)
  htTP://www.math.u-bordeaux.fr/ ksantugi/downloads/CPlusPlus/PolyCxx.pdf
- Quelques sites web :
  http://stackoverflow.com/, http://www.cplusplus.com, http://cpp.developpez.com

### Pour la compilation et l'exécution

Pour compiler un fichier main.cc :

```
g++ -std=c++11 -o run main.cc
```

L'exécutable créé ainsi s'appelle run, on le lancera en tapant

```
./run
```

Pour compiler plusieurs fichiers :

```
g++ -std=c++11 -o run TP1.cc main.cc
```

Si l'on souhaite séparer la phase de compilation de la phase d'édition des liens, on exécutera les commandes suivantes :

```
g++ -std=c++11 -c fonctions.cc
g++ -std=c++11 -c main.cc
g++ -std=c++11 -o run fonctions.o main.o
```

## Quelques conseils

**Bien lire la console quand vous avez des erreurs à la compilation.** En effet le compilateur vous donne de nombreuses informations pour vous aider à corriger la syntaxe de votre code. Un exemple :

```
main.cc:20:35: error: expected ';' after expression
    cout << "Hello world!" << endl
                                  ^
                                  ;
1 error generated.
```

Le compilateur nous informe que l'erreur est dans le fichier « main.cc » à la ligne 20 et à la colonne 35. Ensuite il nous informe qu'il attend un « ; » à la fin de la ligne :

```
cout << "Hello world!" << endl
```

N'hésitez pas à **commenter votre code**, cela facilitera la recherche d'erreurs. Pour insérer un commentaire en C++ sur une ligne :

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

et sur plusieurs lignes :

```
/* Ceci est un commentaire
sur plusieurs lignes */
```

Pensez à bien **indenter** votre code pour faciliter sa lecture. Avec *Atom* (éditeur qui est conseillé), vous pouvez faire :
Edit > Lines > Auto Indent

Pour ce premier TP, de nombreux programmes vous sont donnés, n'hésitez pas à **modifier et/ou enlever** certaines lignes pour voir le rôle de chaque bout de code.

# Objectifs

- Faire un premier programme en C++
- Présenter les différents types de données en C++
- Utiliser la librairie *iostream* pour afficher des messages à l'écran (cout) et interagir avec l'utilisateur (cin)
- Apprendre les structures de contrôle classiques : boucle *for*, conditions *if* et *while*
- Apprendre à écrire des fonctions
- Passage des arguments (par valeur ou par référence)

# Exercices

## Exercice 1 - Premier programme

1. Créer un fichier main.cc et insérer le code suivant :

```
1  /* Chargement du fichier << iostream >> qui est une bibliothèque
2  d'affichage de messages à l'écran dans une console */
3  #include <iostream>
4
5  // Choisir parmi différentes bibliothèques celles que l'on veut
6  using namespace std;
7
8  /* C'est ici que commence vraiment le programme. Les programmes sont
9  essentiellement constitués de fonctions. Chaque fonction a un rôle et peut
10 appeler d'autres fonctions pour effectuer certaines actions.
11 Tous les programmes possèdent une fonction << main >>. C'est donc la fonction
12 principale. Elle renvoie toujours un entier d'où la présence du << int >>
13 devant le main. */
14 int main()
15 {
16     /* Première ligne composée de 3 éléments qui fait quelque chose de concret :
17     1. cout : affichage d'un message à l'écran
18     2. "Hello world!" : le message à afficher
19     3. endl : retour à la ligne dans la console. */
20     cout << "Hello world!" << endl;
21
22     /* Ce type d'instruction clôt généralement les fonctions. Ici, la fonction
23     main renvoie 0 pour indiquer que tout s'est bien passé (toute valeur
24     différente de 0 aurait indiqué un problème). */
25     return 0;
26 }
```

Le compiler et l'exécuter. Vous devez voir s'afficher dans le terminal :

```
1  Hello world!
```

2. Retirer le point virgule à la fin de la ligne 20. Compiler de nouveau et étudier la réaction du compilateur. Rajouter le point virgule.

3. Commenter la ligne 3. Remarquer que les commandes "cout" et "endl" ne sont plus reconnues. Enlever le commentaire.

4. Commenter la ligne 6. Remarquer que les commandes "cout" et "endl" ne sont plus reconnues. Cependant le compilateur vous propose à la place :

```
1  std::cout
2  std::endl
```

Essayer et conclure sur le rôle de cette ligne.

## Exercice 2 - Les types

Une variable est une information stockée en mémoire. Il en existe différents types en fonction de la nature de l'information à stocker : **string** (texte), **int** (entier), **unsigned int** (entier positif ou nul), **double** (réel), **bool** (true or false). Une variable doit être définie (une valeur doit lui être affectée) avant son utilisation. La valeur d'une variable peut être affichée avec cout. Les variables peuvent être déclarées (à peu près) n'importe où dans le code.

1. Tester le programme suivant.

```
#include <iostream>
//Pour utiliser les fonctions racine (sqrt) et puissance (pow)
#include <math.h>
using namespace std;
int main()
{
        double a(2.); // ou double a = 2.;

        string nomFonction("racine");

        int n(3); // ou int n = 3;

        cout << nomFonction << "(" << a << ") = " << sqrt(a) << endl;

        cout  <<  a  << "^" << n << " = " << pow(a,n) << endl;

        return 0;
}
```

**Remarque importante :**
Les 3 bouts de code ci-dessous sont équivalents :

```
double a(2);
```

```
double a = 2.;
```

```
double a;
a = 2;
```

Par contre, vous ne pouvez pas faire :

```
double a();
a = 2;
```

2. Il est bien sûr possible d'additionner, de soustraire (etc ...) avec le C++. Rajouter les lignes suivantes au code précédent. Tester le bout de code suivant :

```
double b(2.), c(3.), d(2.1); // (b = 2)
b++; // Ajoute la valeur 1 : cette commande a donné son nom au C++ ! (b = 3)
b += c; // Ajouter la valeur c à b (b = 6)
b -= d; // Retire la valeur d à b (b = 3.9)
cout << "Après de nombreux calculs, b vaut " << b << endl;
```

Essayer le code suivant :

```
int e(2), f(3);

int g = e/f;

cout << "La division de " << e << " par " << f << " vaut " << g << "." << endl;
```

Êtes-vous satisfaits du résultat ? Que se passe t-il si e, f et g sont des doubles ?

3. Il est aussi possible de demander à l'utilisateur de remplir une variable. Pour cela vous aurez besoin d'utiliser "cin" qui contrairement à "cout" permet de faire rentrer des informations. Par exemple pour demander l'âge de l'utilisateur, essayer les lignes suivantes :

```
int age(0);
cin >> age;
```

Coder pour obtenir sur la console ce genre de résultat :

```
Quel age avez-vous ?
23
Vous avez 23 ans !
```

Que se passe t-il si l'utilisateur ne rentre pas un entier ?

4. Les variables peuvent être considérées comme des cases mémoire avec une étiquette portant leurs noms, comme dans la figure ci-dessous.

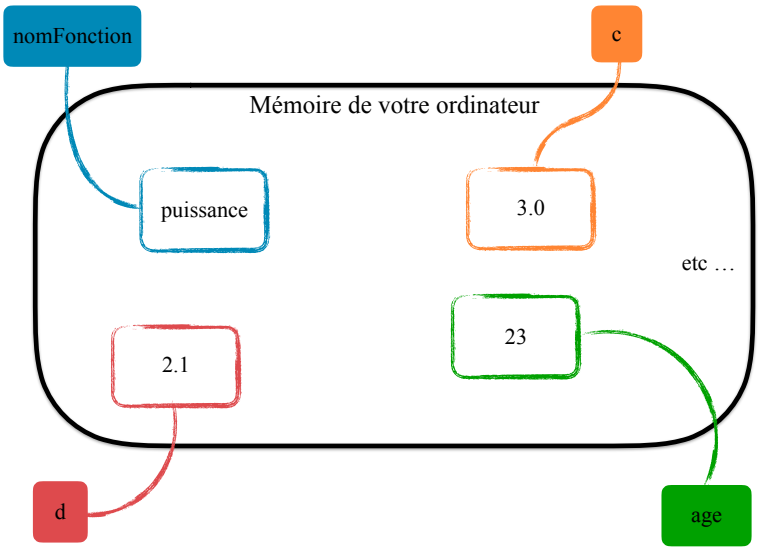


FIGURE 1 – Schéma de l'état de la mémoire après plusieurs déclarations

En C++, il est possible de coller une deuxième étiquette à une case mémoire à l'aide du symbole "&". On obtient alors un deuxième moyen d'accéder à la même case mémoire. Il s'agit d'une **référence**. En rajoutant ces quelques lignes :

```
1 int& laVariable(age);
2 cout << "Vous avez " << laVariable << " ans ! (par référence)" << endl;
```

Voilà à quoi ressemble la mémoire :

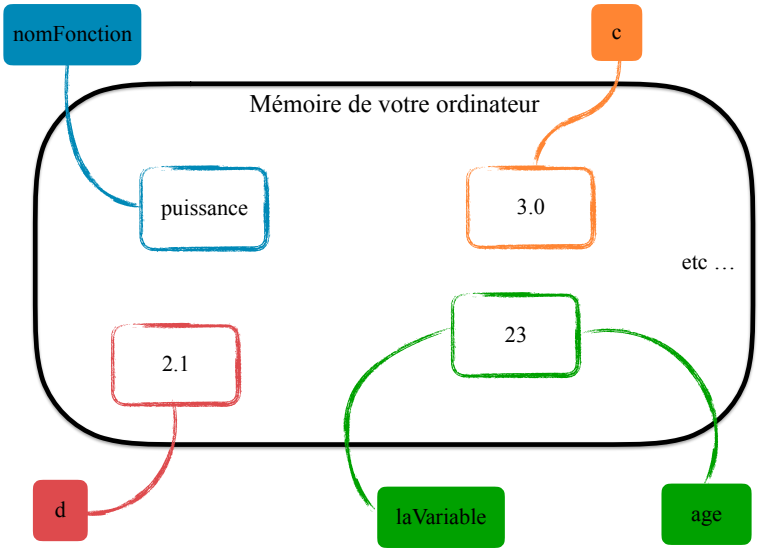


FIGURE 2 – Schéma de l'état de la mémoire après plusieurs déclarations

Bien sûr vous obtenez le même résultat que précédemment. On verra plus tard le rôle de ces références.

## Exercice 3 - Les structures de contrôle

Le programme suivant vous montre la structure de la condition "if" et de la boucle "for" en C++.

```cpp
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

int main()
{
  const int nb_essai(10);
  srand(time(0));
  const int nombre = (rand() % 100);
  int proposition(0), est_gagnant(nb_essai+1);
  cout << "Le nombre caché est un entier compris entre 0 et 100." << endl;
  cout << "Vous avez " << nb_essai << " essais pour le trouver." << endl;

  for (int essai = 1 ; essai <= nb_essai ; essai++)
  {
    cout << "Essai : " << essai << " : votre proposition est : ";
    cin >> proposition;
    if (proposition == nombre)
    {
      est_gagnant = essai;
      break;
    }
    else if (proposition > nombre)
    {
      cout << "Moins" << endl;
    }
    else
    {
      cout << "Plus" << endl;
    }
  }
  if(est_gagnant <= nb_essai)
  {
    if (est_gagnant == 1)
    {
      cout << "Vous êtes un devin : vous avez trouvé du premier coup !" << endl;
    }
    else
    {
      cout << "Bravo! Vous avez trouvé en " << est_gagnant << " essais." << endl;
    }
  }

  return 0;
}
```

1. Tester le code.

2. Que se passe t-il si vous commentez la ligne 9 ?

3. Quel est le rôle du const à la ligne 8 ? Que se passe t-il si vous mettez un const devant la ligne 11 ?

4. Modifier le code en mettant une boucle while pour permettre de rejouer tant qu'on n'a pas gagné.

## Exercice 4 - Les fonctions

La définition des fonctions est toujours la suivante :

```
type nomFonction(arguments)
{
        // Écrire ici ce que la fonction doit faire
}
```

Une fonction est composée de 3 éléments :

- Le premier indique le **type de donnée** qui est renvoyée par la fonction :

  int, double, string, void (= vide, quand elle ne renvoie rien) ...

Pour renvoyer une valeur, nous utilisons la commande "return" comme pour la fonction main().

- Le deuxième est le **nom de la fonction**.
- Entre les parenthèses, on trouve la **liste des paramètres** de la fonction. Il peut y avoir :

  aucun argument comme pour la fonction main() vue ci-dessus
  ou plusieurs comme pour la fonction puissance pow(double, int) (Ex 2, Q 1)

Les accolades délimitent le contenu de la fonction. Toutes les opérations qui seront effectuées se trouvent entre les deux accolades.

1. Faire une fonction *print_hello* qui affiche "Hello world !" et l'appeler dans la fonction main.

2. Faire une fonction *fibonacci* qui calcule la suite de Fibonacci :

$$u_{n+1} = u_n + u_{n-1}.$$

Si $u_0$ et $u_1$ sont des entiers, $u_n$ reste un entier pour tout $n$, le prototype de la fonction sera alors le suivant :

```
int nomFonction(int n, int u0, int u1)
{
        int un(0);
                .
                .
                .
        return un;
}
```

Pour $u_0 = u_1 = 1$, calculer $u_{10}, u_{30}$ et $u_{50}$. Que se passe t-il ? Essayer avec int64_t au lieu de int en rajoutant :

```
#include <stdint.h>
```

Conclure.

## Exercice 5 - Passage des arguments (par valeur ou par référence)

1. Par défaut, les arguments d'une fonction sont passés par valeur :

```
#include <iostream>
using namespace std;

void valeur(int b)
{
  b=5;
}

int main()
{
  int a=3;
  cout << "a avant la fonction valeur " << a << endl;
  valeur(a);
  cout << "a après la fonction valeur " << a << endl;

  return 0;
}
```

Compiler et exécuter ce code. Que se passe t-il ?

2. Comme vous avez pu le voir, le nombre imprimé est 3, et non 5 ! La fonction valeur a reçu en argument une copie de a et ne change que la valeur de cette copie. Dans la fonction main, la valeur de a ne change pas. Il est cependant possible de passer les arguments par référence. Elles ont été introduites dans l'exercice 2 à la question 4. Pour les utiliser, il suffit de rajouter le symbole "&" dans le type de l'argument. Tester cette nouvelle fonction :

```
void reference(int& b)
{
  b=5;
}
```

Cette fois-ci, la console imprime 5 à l'écran car le passage a eu lieu par référence.
**Remarque :** Une référence n'a de sens que s'il y a une variable en mémoire derrière :

```
reference(&(3)); //Cela ne compile pas car cela n'a aucun sens.
```

3. Pour que le code soit plus efficace, on peut aussi passer par référence de gros objets tout en déclarant que leur valeur ne doit pas être modifiée. Pour le moment la notion d'objet peut vous sembler lointaine (TP 3) mais par exemple si vous souhaitez envoyer un email avec une photo de 4Mo, le téléchargement va prendre du temps alors que si vous envoyez l'adresse web de cette image, c'est beaucoup plus rapide ! Quand il s'agit juste de faire passer cet objet et qu'il ne doit pas être modifié, on rajoute le mot-clef const devant le type de l'argument. Cela permet au compilateur de mieux vérifier le code et aide à trouver les bugs. C'est une très bonne habitude à prendre pour "mieux programmer". Tester pour voir l'erreur que renvoie le compilateur :

```
void const_reference(const int& b)
{
  b=5;
}
```

## Exercice 6 - Résolution d'une équation différentielle

On considère l'équation différentielle suivante :

$$\left\{ \begin{array}{rcl} y'(t) & = & f(t, y(t)) \\ y(0) & = & y^0 \end{array} \right.$$

1. Écrire une fonction qui permet de calculer $y^{n+1}$ à partir de $y^n$ avec le schéma d'Euler explicite :

$$y^{n+1} = y^n + \Delta t f(t^n, y^n).$$

Le prototype de cette fonction sera le suivant :

```
// y : y^n entrée, y^(n+1) en sortie --> pour le modifier passer y par référence
// tn : t^n
// dt : pas de temps
// f : fonction f --> on passe la fonction par référence
void Euler(double& y, double tn, double dt, double (&f)(double, double));
```

2. Dans la fonction main, écrire la boucle principale permettant de calculer $y^1, y^2, \cdots, y^N$ à partir de $y^0$ en appelant la fonction "Euler" à chaque pas de temps.

3. Tester le programme pour les deux fonctions suivantes :

$$\left\{ \begin{array}{rcl} y'(t) & = & y \\ y(0) & = & -2 \end{array} \right.$$

et

$$\left\{ \begin{array}{rcl} y'(t) & = & y^2\, t \\ y(0) & = & -2 \end{array} \right.$$

Comparer avec les solutions exactes (qui sont bien sur respectivement : $y(t) = y^0 \exp(t)$ et $y(t) = 2y^0/(2 - t^2 y^0)$).

4. Vérifier l'ordre de convergence de la méthode d'Euler :
a. Calculer l'erreur entre la solution approchée et la solution exacte au temps final pour $\Delta t = 0.001$ et pour $T_{final} = 1$. On la note $e_{\Delta t}$.
b. Faire de même avec $\Delta t/2$. On la note $e_{\Delta t/2}$.
c. Calculer l'ordre de la méthode par :

$$\text{ordreConvEuler} = \log 2 \left( e_{\Delta t} \,/\, e_{\Delta t/2} \right).$$