CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

# Programmation

## Introduction au langage C++

Prof : Kamal Boudjelaba

14 août 2023

B T S S N

**LTP Charles Carnus**
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

# Table des matières



# Liste des tableaux

# 1. Installation du logiciel

Il faut installer certains logiciels spécifiques pour programmer en C++.

**Editeur de texte :**
Permet d'écrire le code source du programme en C++. L'idéal est d'avoir un éditeur de texte intelligent qui colore tout seul le code, ce qui permet de repérer bien plus facilement les différentes instructions.

**Compilateur :**
Permet de compiler (transformer) le code source en binaire.

**Debugger :**
Permet de détecter certaines erreurs de programmation.

## 1.1 Procédure d'Installation :

— On récupère chacun de ces 3 programmes séparément (méthode compliquée) ;
— Ou on utilise un programme qui combine éditeur de texte, compilateur et debugger. Ces programmes sont appelés IDE : Integrated Development Environment (EDI en français : Environnement de Développement Intégré).

Quand on code un programme, l'ordinateur génère plusieurs fichiers de code source : des fichiers.cpp, .h, les images du programme ...
Le rôle d'un IDE est de rassembler tous ces fichiers au sein d'une même interface. ⟶ on a accès à tous les éléments du programme.
Un IDE est un logiciel informatique qui rassemble un certain nombre d'outils nécessaires ou commodes pour l'écriture de programmes informatiques, leur compilation, leur exécution et, si besoin est, le dépannage (débogage) quand le programme ne donne pas le résultat escompté.

## 1.2 Choix de l'IDE :

— **Code::Blocks :** Gratuit et disponible pour la plupart des systèmes d'exploitation (Windows, Mac OS 32 bits et Linux).
— **Visual Studio Code :** Gratuit et disponible pour la plupart des systèmes d'exploitation, mais il nécessite l'installation d'extensions.
— **XCode :** Gratuit et disponible sur Mac OS uniquement.
— **...**

## 2. Structure de base d'un programme en C++

Il est très courant d'introduire un langage de programmation en utilisant un programme qui affiche le texte "Hello World" à l'écran. Un exemple simple d'un programme C++ qui fait cela est illustré ci-dessous :

**Exemple 2.1**

```
1  #include <iostream>
2  using namespace std;
3
4  int main()       // Ou int main(int argc, char* argv[])
5  {
6      cout << "Hello world!" << endl;
7      // Ou std::cout << "Hello World!\n"; si on supprime la ligne 2
8
9      return 0;
10 }
```

### Lancement de la phase de compilation

Pour compiler le fichier source, utilisez l'icône Build.
Si vous n'avez pas d'erreur, vous pouvez passer à la phase suivante. Sinon, corrigez les erreurs ...

### Exécution du programme

**Note 2.1.**
L'exécution n'est possible que si la compilation a été faite sans erreurs.

Les messages d'erreurs de compilation permettent de corriger les fautes de syntaxe. Corrigez les erreurs et relancez jusqu'à obtenir une compilation sans erreurs.
Les warnings sont de simples avertissements et n'empêchent pas l'exécution. Faites attention néanmoins à ces warnings.
Le lancement de l'exécution se fait par l'icône Run

**Note 2.2.**
L'icône Build and Run (compiler et exécuter) lance la compilation et l'exécution par la suite sauf en cas d'erreur de compilation bien sûr.

### Analyse du programme

**1ère ligne :**

`#include <iostream>`
Charger des fonctionnalités du C++ pour pouvoir effectuer certaines actions.
On doit donc charger des extensions (bibliothèques) pour utiliser certaines possibilités. C'est l'équivalent en Python de : import numpy as np
Charger le fichier `iostream` :

- permet l'affichage de messages à l'écran dans une console,
- permet aussi de récupérer ce que saisit l'utilisateur au clavier.

`iostream` signifie "Input Output Stream" (Flux d'entrée-sortie).
Dans un ordinateur, l'entrée correspond en général au clavier ou à la souris, et la sortie à l'écran.

### 2ème ligne :

`using namespace std;`
C'est l'équivalent en Python de :
from scipy.io.wavfile import read
Permet d'indiquer dans quel lot de fonctionnalités notre fichier source va aller piocher.
**std** : correspond à la bibliothèque standard

### 3ème ligne :

`int main()`
Signifie fonction principale : cette partie doit contenir les différentes instructions de calcul de notre programme.
Cette fonction a la forme suivante :

```
1 int main()
2 {
3
4 }
```

Les accolades déterminent le début et la fin de la fonction (les instructions sont écrites entre les deux accolades).

### 5ème ligne :

`cout << "Hello world!" << endl;`
**cout** (console **out**put) : une instruction (commande) qui permet d'afficher un message à l'écran.
**"Hello world !"** : le message à afficher.
**endl** : crée un retour à la ligne dans la console.

### Dernière ligne de la fonction :

`return 0;`
On demande à la fonction `main` de renvoyer 0 pour indiquer que tout s'est bien passé.

**Remarque 2.1 :**

Chaque instruction du code se termine par un point-virgule.

## 2.1 Conseils : débogage du code

Il existe de nombreux outils conçus pour faciliter le débogage du code. Le plus basique d'entre eux est le compilateur et les flags associés au compilateur. Des outils plus sophistiqués existent, mais ils sont destinés à des projets à plus grande échelle. Plutôt que d'apprendre à utiliser un outil de débogage sophistiqué lors des premières étapes de l'apprentissage du C++, il est conseillé d'utiliser des techniques plus simples pour déboguer le code.

**Compiler fréquemment le code :** Sauvegarder et compiler le code chaque fois que quelques déclarations sont ajoutées est un diagnostic utile pour voir si des problèmes potentiels sont introduits. S'il y a des problèmes, commenter les nouvelles instructions et recompiler. Ajouter ensuite les instructions une par une jusqu'à ce que la ligne problématique soit identifiée.

**Enregistrer fréquemment le projet :** Si vous avez du code qui fonctionne et que vous devez ajouter de nouvelles fonctionnalités, ne jetez pas l'ancienne version. Si les choses tournent mal, vous pourrez voir exactement ce que vous avez changé et si tout le reste échoue, vous aurez une version de travail à laquelle revenir. S'il est essentiel que vous puissiez revenir à une version fonctionnelle du code, ou si vous êtes dans un projet collaboratif, il est recommander d'utiliser un système de contrôle de version.

**Tester le code avec un exemple simple :** Testez toujours le code avec un exemple simple. Par exemple, si vous écrivez du code pour ajouter les éléments de deux tableaux, vérifiez la sortie par comparaison avec un calcul que vous avez effectué vous-même.

**Comprendre les erreurs qui surviennent lors de l'exécution du code :** Si le programme signal une "erreur de segmentation" lors de son exécution, il est probable que vous ayez tenté d'accéder à un membre d'un tableau qui est hors plage : c'est-à-dire que vous avez peut-être tenté d'accéder à la 6e élément d'un tableau qui n'a été déclaré avoir que 4 éléments.

**Utiliser la sortie :** Si vous avez besoin de savoir où votre programme plante et pourquoi, imprimez quelques valeurs de variables à des points clés de l'exécution.

**Utiliser un débogueur :** .Si tout le reste échoue, déboguez votre programme à l'aide d'un débogueur.

## 3. Les commentaires

Les commentaires sont nécessaires pour expliquer le fonctionnement du programme.
Ces commentaires ne sont pas lus (ne sont pas exécutés) par le compilateur.

### Commentaires sur une ligne :

```
// Votre commentaire
```

### Commentaires sur plusieurs lignes :

```
/* Votre commentaire
Suite de votre commentaire
Fin de votre commentaire */
```

## 4. Les variables

Dans la plupart des programmes, en particulier les applications de calcul scientifique, nous souhaitons stocker des entités et effectuer des opérations sur celles-ci. Ces entités sont appelées variables. Dans les programmes C++, comme dans la plupart des langages compilés, les variables ainsi que leur types doivent être déclarés avant d'être utilisées.

Une variable peut être de type nombre, caractère, tableau, vecteur ...

- le nom de la variable doit être constitué uniquement de lettres, de chiffres et du tiret-bas "_"
- le premier caractère doit être une lettre
- on n'utilise pas d'accents dans le nom de la variable
- on n'utilise pas d'espaces dans le nom de la variable

| Nom du type | Type d'élément de la variable |
|---|---|
| bool | Une valeur parmi 2 possiblités : vrai (true) ou faux (false) |
| char | Un caractère |
| int | Un entier |
| unsigned int | Un nombre entier positif ou nul |
| double | Un nombre réel |
| string | Une chaîne de caractères (mot, phrase) |

**Table 1.** Types de variables

## 4.1 Déclaration de variables

La déclaration se fait de la manière suivante :
`Type Nom_de_la_variable(Valeur);` ou `Type Nom_de_la_variable = Valeur;`

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

int main()
{
    int var1(16);           // var1 est un entier qui vaut 16
    double var2 = 4.53;     // var2 est un réel qui vaut 4.53
    bool var3(true);        // var3 est un booléen vrai (true)
    char var4('a');         // var4 contient le caractère "a"

    return 0;
}
```

On peut déclarer une variable sans l'initialiser (sans lui attribuer une valeur) :
`Type Nom_de_la_variable;`

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

int main()
{
    string nomFormation;
    int nombreMatieres;
    bool test;
    return 0;
}
```

Déclarer plusieurs variables de même type :
`Type variable1 = Valeur1, variable2 = Valeur2;` ou `Type variable1(Valeur1), variable2(Valeur2);`

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

int main()
{
    int a(10),b(-5),c(32);
    string nom("Carnus"), ville("Rodez");
    return 0;
}
```

La déclaration d'une variable de type chaîne de caractères se fait de la manière suivante :
ajouter la ligne `include <string>` pour gérer ces chaînes.

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

int main()
{
    string var5("Lycée Charles Carnus");
    return 0;
}
```

Parfois, on souhaite qu'une variable soit constante dans tout le code, par exemple la valeur numérique utilisée pour $\pi$. On peut s'assurer qu'une variable reste inchangée dans tout le code en attribuant une valeur à la variable lorsqu'elle est déclarée, ainsi qu'en utilisant le mot-clé `const` comme indiqué dans le fragment de code suivant :
`const double pi = 3.14;`
On peut également définir une constante en utilisant `#define`

**Remarque 4.1 :**

On peut vouloir fixer l'erreur sur une mesure à un très petit nombre, par exemple $10^{-12}$. Clairement, on peut définir cette erreur en utilisant le fragment de code ci-dessous :

```
double erreur = 0.000000000001;
```

L'instruction ci-dessus n'est clairement pas idéale, un simple coup d'oeil au code ne permet pas de faire facilement la distinction entre, disons, $10^{-10}$ et $10^{-12}$. Ce serait beaucoup plus clair si on peut écrire la valeur numérique en notation scientifique. Ceci est donné dans le code ci-dessous :

```
double erreur = 1.0e-12;
```

La lettre "e" peut être lue comme "fois 10 à la puissance de"

## 4.2 Les caractères ASCII

Les caractères ASCII sont des chiffres, des lettres majuscules, des lettres minuscules et quelques autres symboles couramment utilisés : la plupart des caractères du clavier sont des caractères ASCII. Les variables qui sont des caractères ASCII sont déclarées à l'aide du mot-clé `char`. Un exemple de code utilisant un caractère ASCII est présenté ci-dessous :

```
#include <iostream>
int main(int argc, char* argv[])
{
    char lettre;
    lettre = 'a';
    std::cout << "Le caractère est : " << lettre << "\n";
    return 0;
}
```

## 4.3 Les variables booléennes

Les variables booléennes prennent soit la valeur True (vrai), soit la valeur False (faux). Ces variables sont couramment utilisées pour spécifier si une portion de code doit être exécutée dans les instructions *if* et *while*. Un exemple de variables booléennes est donné ci-après : `bool x1 = true, x2 = false;`

## 4.4 Les chaînes de caractères

Le type de données `char` représente un caractère ASCII. Une chaîne de caractères peut être considérée comme une suite ordonnée de caractères. Par exemple, "C++" est une chaîne composée de la liste ordonnée de caractères "C", "+" et "+".

Pour utiliser des chaînes en C++, il faut inclure le fichier d'en-tête `string` (`#include <string>`). La bibliothèque accessible à l'aide de ce fichier d'en-tête contient des fonctionnalités importantes pour l'utilisation et la manipulation de chaînes. Dans l'exemple de code ci-dessous, on montre comment déclarer une chaîne, comment déterminer la longueur d'une chaîne, comment accéder aux caractères individuels de la chaîne et comment imprimer une chaîne sur la console.

Une chaîne en C++ est un peu comme un tableau de caractères avec une couche de fonctionnalités supplémentaires.

```
#include <iostream>
#include <string>

int main(int argc, char* argv[])
{
    std::string ville;  // noter la présence de std::
    ville = "Rodez";    // double quote
    std::cout << "Longueur de la chaîne = " << ville.length() << "\n";
    std::cout << "Le troisième caractère = " << ville.at(2) << "\n";
    std::cout << "Le troisième caractère = " << ville[2] << "\n";
    std::cout << ville << "\n";             // Affiche la chaîne ville
    std::cout << ville.c_str() << "\n";     // Imprime également la ville
    return 0;
}
```

## 5. Affichage des variables et du texte

Pour afficher la valeur d'une variable, on utilise l'instruction `cout <<` (console output)

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

int main()
{
    int nombreMatieres(6);
    cout << "Le nombre de matières est : ";
    cout << nombreMatieres;

    return 0;
}
```

Après exécution, la console affiche :

```
Le nombre de matières est : 6
```

Le code ci-dessous est équivalent au code précédent

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

int main()
{
    int nombreMatieres(6);
    cout << "Le nombre de matières est : "<< nombreMatieres << endl;
    return 0;
}
```

Afficher plusieurs variables

```

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

int main()
{
    int ageU(20);
    string nomU("Charles");

    cout << "Nom : " << nomU << ", age : " << ageU << "ans" << endl;
    return 0;
}
```

Après exécution, la console affiche :

```
Nom : Charles, age : 20 ans
```

| Commande | Symbole |
|---|---|
| Nouvelle ligne | $\backslash n$ |
| Tabulation | $\backslash t$ |
| ’ | $\backslash'$ |
| ” | $\backslash$” |
| ? | $\backslash?$ |

**Table 2.** Certaines commandes de formatage pour la sortie de la console

## 6. Lecture depuis la console

La lecture des valeurs tapées sur le clavier s'effectue en utilisant l'instruction `cin >>` (console in).

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

int main()
{
    cout << "Entrer le nombre d'itérations : " << endl;
    int nombreIt(0); //On affecte 0 à nombreIt
    cin >> nombreIt; //nombreIt prend la valeur tapée au clavier
    cout << "Le nombre d'itérations est  " << nombreIt <<  endl;
    return 0;
}
```

Après exécution, la console affiche :

```
Entrer le nombre d'itérations :
10
Le nombre d'itérations est 10
```

```cpp
// Entrées multiples
#include <iostream>
using namespace std;

int main()
{
    string prenom;
    int age;

    cout << "Saisir votre prénom et taper sur la touche ENTREE puis votre age et taper sur la touche ENTREE" << "\n";
    cin >> prenom >> age;

    cout << "Nom : " << prenom << endl;
    cout << "Age : " << age << endl;

    return 0;
}
```

Après exécution, la console affiche :

```
Saisir votre prénom et taper sur la touche ENTREE puis votre age et taper sur la touche ENTREE
Charles
19
Nom : Charles
Age : 19
```

```cpp
// Programme pour illustrer l'utilisation de cin.getline
#include <iostream>
using namespace std;

int main()
{
    char mot[30];
    cout << "Saisir un mot ou une phrase : ";
    // Lire le flux de 11 caractères (une lettre accentuée est comptée comme 2 caractères !)
    cin.getline(mot, 12);
    cout << mot << endl;

    return 0;
}
```

Après exécution, la console affiche :

```
Saisir un mot ou une phrase : Lycee Charles Carnus
Lycee Charl
```

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

int main() {
  char nom[20], adresse[30];

  cout << "Nom : ";
  // cin avec getline()
  cin.getline(nom, 20);

  cout << "Adresse : ";
  cin.getline(adresse, 30);

  cout << endl << "Votre saisie " << endl;
  cout << "Nom = " << nom << endl;
  cout << "Adresse = " << adresse << endl;

  return 0;
}
```

Après exécution, on obtient :

```
Nom : Charles Carnus
Adresse : Avenue de Saint-Pierre

Votre saisie
Nom = Charles Carnus
Adresse = Avenue de Saint-Pierre
```

**Remarque 6.1 :**

Tant qu'un programme n'est pas au point, il ne faut pas utiliser de `cin`, mais mettre les valeurs des données dans le programme sous la forme `x = 3.12;` etc. Sinon, à chaque exécution destinée à tester le programme, il faut taper les données au clavier, ce qui est une perte de temps. Ensuite, si le programme est destiné à être utilisé, on peut mettre des `cin`, mais à condition qu'ils soient en nombre très limité. Sinon, l'entrée des données au clavier est trop fastidieuse et comporte trop de risques d'erreur. Il vaut mieux placer d'abord les données dans un fichier qui leur est dédié et les faire lire par le programme dans ce fichier (voir le cours sur la gestion des fichiers).
Inversement, il n'est pas normal que l'utilisateur d'un programme, considéré comme achevé, soit contraint d'entrer ses données dans le fichier du programme lui-même.

## 6.1 Le problème des espaces

Soit le programme ci-dessous :

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

int main()
{
    cout << "Quel est votre nom ?" << endl;
    string nom("Sans nom");
    cin >> nom;

    cout << "Donner un multiple de 5" << endl;
    double chiffre(-1.);
    cin >> chiffre;

    cout << "Vous vous appelez " << nom << " et vous avez saisi le nombre " << chiffre << "." <<
                                                endl;

    return 0;
}
```

Exécuter le programme et saisir vos nom et prénom.

**Note 6.1.**
Le programme n'a rien demandé pour le multiple de 5 et votre prénom a disparu.

Tester cette version améliorée du programme :

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

int main()
{
    cout << "Quel est votre nom ?" << endl;
    string nom("Sans nom");
    getline(cin, nom);

    cout << "Donner un multiple de 5 " << endl;
    double chiffre(-1.);
    cin >> chiffre;

    cout << "Vous vous appelez " << nom << " et vous avez saisi le nombre " << chiffre << "." <<
                                                endl;

    return 0;
}
```

**Note 6.2.**
Si on utilise `cin >>` puis getline() (on demande le multiple de 5 puis le nom et le prénom), le code ne fonctionnera pas correctement. Pour y remédier, il faut ajouter la commande cin.ignore() ; après la ligne contenant `cin >> ...`

Tester cette version améliorée du programme :

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

int main()
{
    cout << "Donner un multiple de 5 " << endl;
    double chiffre(-1.);
    cin >> chiffre;

    cin.ignore();

    cout << "Quel est votre nom ?" << endl;
    string nom("Sans nom");
    getline(cin, nom);

    cout << "Vous vous appelez " << nom << " et vous avez saisi le nombre " << chiffre << "." <<
                                                    endl;

    return 0;
}
```

Cas où l'entrée est une chaîne de caractères séparée par des espaces

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

int main()
{
    cout << "Quel est le nom du lycée ?" << endl;
    string nL("Sans nom");
    getline(cin, nL); //On affecte à nL toute la ligne tapée

    cout << "Combien d'étudiants inscrits ?" << endl;
    double nE(-1.);
    cin >> nE;

    cout << "Au lycée " << nL << "il y a " << nE << "étudiants." << endl;

    return 0;
}
```

Cas où l'entrée est une chaîne de caractères séparée par des espaces

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

int main()
{
    cout << "Combien d'étudiants inscrits ?" << endl;
    double nE(-1.);
    cin >> nE;

    cin.ignore();

    cout << "Quel est la nom du lycée ?" << endl;
    string nL("Sans nom");
    getline(cin, nL);
    cout << "Au lycée " << nL << "il y a " << nE << " étudiants" << endl;
    return 0;
}
```

## 7. Le flux d'erreurs standard

Le `cerr` est un flux de sortie standard pour les erreurs. Cet objet est déclaré dans le fichier d'en-tête `iostream`. Il est essentiellement utilisé pour afficher des informations ou des erreurs sur la console.

**Remarque 7.1 :**

L'objet `cerr` n'est pas mis en mémoire tampon et chaque insertion de flux dans `cerr` entraîne l'affichage immédiat de sa sortie.

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

int main()
{
    cerr << "Erreur !!!" << endl;
    return 0;
}
```

Après exécution, on obtient :

```
Erreur !!!
```

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

int main() {
  string Nom_fichier = "data.txt";
  ifstream fichier(Nom_fichier);

  if(fichier) {
    cout << fichier.rdbuf();
  }
  else {
    // Affichage du message d'erreur
    cerr << "Erreur d'ouverture du fichier " << Nom_fichier << endl;
  }
  return 0;
}
```

```
Erreur d'ouverture du fichier data.txt
```

# 8. Les opérateurs en C++

## 8.1 Les opérateurs arithmétiques

| Opération | Symbole (opérateur) |
|---|---|
| Addition | + |
| Soustraction | − |
| Multiplication | * |
| Division | / |
| Modulo (reste de la division euclidienne) | % |

**Table 3.** Opérateurs arithmétiques

**Note 8.1.**
L'opérateur % (exp : $a\%b$) renvoie, dans cet exemple, le reste de la division euclidienne de $a$ par $b$.
Cette opérateur est utilisé uniquement sur des entiers.
Dans la division euclidienne de 7 par 3, le quotient est 2 et le reste est 1 ($7 = 2\times3+1$). Donc $7\%3$ renvoie 1.

Un exemple de code C++ permettant d'effectuer une variété d'opérations mathématiques sur des variables est donné ci-dessous :

**Remarque 8.1 :**
Pour réaliser des calculs mathématiques (racine carrée, sin ...), il faut inclure la librairie `cmath`.
`#include <cmath>`

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

int main(int argc, char* argv[])
{
    double x = 1.0, y = 2.0, z;
    z = x/y;         // division
    z = x*y;         // Multiplication
    z = sqrt(x);     // Racine carrée
    z = exp(y);      // Exponentiel
    z = pow(x, y);   // x à la puissance de y
    z = M_PI;        // Valeur de pi
    return 0;
}
```

## 8.2 Les raccourcis pour certaines opérations arithmétiques

| Opération | Raccourci | |
|---|---|---|
| `a = a + b;` | `a += b;` | |
| `a = a - b;` | `a -= b;` | |
| `a = a * b` | `a *= b;` | |
| `a = a / b;` | `a /= b;` | |
| `a = a % b;` | `a %= b;` | si `a` et `b` sont des entiers (`a` mod `b`) |
| `a = a + 1;` | `a++;` | si `a` est un entier |
| `a = a - 1;` | `a--;` | si `a` est un entier |

**Table 4.** Les raccourcis

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

int main()
{
    int x = 4;
    x++; // Incrémentation : Augmente d'une unité la variable x  (4+1 = 5)
    x--; // Décrémentation : Diminue d'une unité la variable x (4-1 = 3)
    return 0;
}
```

Tester ce programme, puis remplacer la ligne contenant le commentaire 'Opération' par l'une des instructions ci-dessous :

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

int main()
{
   double nombre(1.5);
   nombre++;         //'Opération'
   cout << "La valeur actuelle est : " << nombre << endl;
   return 0;
}
```

**Instructions :**

```cpp
++nombre;
nombre--;
--nombre;
nombre += 1.7;
nombre *= 2.;
nombre -= 1.1;
nombre /= 3.;
```

## 8.3 La division des entiers

Une erreur courante fréquemment commise par certains programmeurs C++ (Arduino) consiste à diviser un entier par un autre entier. Soit le fragment de code ci-dessous :

```cpp
int i =5, j = 2, k;
k = i / j;
std::cout << k << "\n";
```

Ce code affichera la valeur 2, alors que la valeur de division de 5 par 2, c'est-à-dire 2.5, était réellement prévue. Il y a deux problèmes potentiels avec ce code tel qu'il est écrit ci-dessus. La première opération qui sera effectuée lors de l'exécution de la ligne 2 est de diviser l'entier `i` par l'entier `j`. La valeur résultant de cette opération sera alors stockée dans la mémoire allouée à `k`. En C++, la division d'un entier par un autre entier ne renverra que la partie entière (quotient) de cette division : donc diviser `i` par `j` stockera la partie entière de 2.5, qui est 2 (car tout ce qui suit la virgule sera ignoré). La deuxième partie de cette instruction - l'opérateur d'affectation - attribuera alors la valeur 2 à la variable entière `k`.
On peut penser que la modification du fragment de code ci-dessus afin que `k` soit défini comme une variable réelle peut résoudre le problème, comme indiqué dans le fragment de code ci-dessous :

```cpp
int i =5, j = 2;
double k;
k = i / j;
std::cout << k << "\n";
```

Cela ne donne toujours pas la valeur correcte de 2.5. En effet, la division est effectuée à la ligne 3 avant que le résultat ne soit stocké en tant que variable réelles `k`. Comme la division d'un entier par un autre entier en C++ renvoie

la partie entière de la division, la division de `i` par `j` renvoie la valeur 2 comme expliqué ci-dessus. Cette valeur est ensuite stockée sous la forme du nombre réel 2.0 dans la mémoire allouée à `k`. Pour diviser deux nombres entiers comme s'il s'agissait de variables réelles, on peut convertir les nombres entiers en réels, comme indiqué dans le code ci-dessous :

```
int i =5, j = 2;
double k;
k = ((double)(i)) / ((double)(j));
std::cout << k << "\n";
```

Le code `((double)(i))` est connu sous le nom de "conversion de type explicite" et permet de traiter la variable entière `i` comme une variable réelle, et donc ce code produit la valeur correcte de 2.5.

## 8.4 L'opérateur d'affectation

L'opérateur d'affectation sert à affecter une valeur à une variable.

| Opération | Syntaxe |
|---|---|
| Affectation | `identificateur = expression` |

**Table 5.** Opérateurs d'affectation

On commence par évaluer `expression` et on met le résultat dans la variable `identificateur`.

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

int main()
{
    int a = 5;  // affectation à la déclaration (déclaration avec initialisation) :
                // assigne la valeur 5 à la variable "a"
    double x = 3.5, y = 1.43; // affectation à la déclaration
    double z;
    z = x + y;      // affectation : résultat d'une expression
    int b, c;
    b = c = 10 + a; // le résultat d'une même expression peut être assigné à plusieurs variables
    return 0;
}
```

## 8.5 Les opérateurs binaires (bit-à-bit)

Ces opérateurs traitent des données selon leur représentation binaire mais retournent des valeurs numériques standard dans leur format d'origine. Les opérateurs suivants effectuent des opérations bit-à-bit, c'est-à-dire avec des bits de même poids.

### Le ET binaire

`exp1 & exp2` : Le résultat est obtenu en faisant un ET logique sur chaque bit de la représentation binaire de `exp1` avec le bit correspondant de la représentation binaire de `exp2`. Le ET logique a pour résultat 1 si les deux bits correspondants en entrée sont à 1 (pour 1,1), et sinon 0 (pour 0,1, 1,0 ou 0,0).

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

int main()
{
    int a = 53;
    int b = 167;
    int c = a & b;
    cout << "c vaut " << c << endl;

    return 0;
}
```

Exécution :

```
c vaut 37
```

Explication :

```
 53 s'écrit en binaire  0000 0000 0000 0000 0000 0000 0011 0101
167 s'écrit en binaire  0000 0000 0000 0000 0000 0000 1010 0111
ET binaire              0000 0000 0000 0000 0000 0000 0010 0101
==> le résultat vaut 37 en décimal
```

### Le OU binaire

exp1 | exp2 : Le résultat est obtenu en faisant un OU logique sur chaque bit de la représentation binaire de exp1 avec le bit correspondant de la représentation binaire de exp2. Le OU logique a pour résultat 1 si l'un des deux bits correspondant en entrée est à 1 (pour 1,1, 1,0 ou 0,1), et sinon 0 (pour 0,0).

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

int main()
{
    int a = 53;
    int b = 167;
    int c = a | b;
    cout << "c vaut " << c << endl;

    return 0;
}
```

Exécution :

```
c vaut 183
```

Explication :

```
 53 s'écrit en binaire 0000 0000 0000 00000 0000 0000 0011 0101
167 s'écrit en binaire 0000 0000 0000 00000 0000 0000 1010 0111
OU binaire             0000 0000 0000 00000 0000 0000 1011 0111
==> le résultat vaut 183 en décimal
```

### Le OU EXCLUSIF binaire

exp1 ^ exp2 : Le résultat est obtenu est faisant un OU exclusif sur chaque bit de la représentation binaire de exp1 avec le bit correspondant de la représentation binaire de exp2. Le OU Exclusif logique a pour résultat 1 si les deux bits correspondant en entrée sont différents (pour 1,0 ou 0,1), et sinon 0 (pour 1,1 ou 0,0).

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

int main()
{
    int a = 53;
    int b = 167;
    int c = a ^ b;
    cout << "c vaut " << c << endl;

    return 0;
}
```

Exécution :

```
c vaut 146
```

Explication :

```
 53 s'écrit en binaire  0000 0000 0000 00000 0000 0000 0011 0101
167 s'écrit en binaire  0000 0000 0000 00000 0000 0000 1010 0111
OU exclusif binaire     0000 0000 0000 00000 0000 0000 1001 0010
==> le résultat vaut 146 en décimal
```

### Le NON binaire

~exp : Le résultat est obtenu en inversant tous les bits de la représentation de exp. Le 1 devient 0 et vice versa.

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

int main()
{
    int a = 53;
    int b = ~a;
    cout << "b vaut " << b << endl;
    return 0;
}
```

Exécution :

```
b vaut -54
```

Explication :

```
53 s'écrit en binaire   0000 0000 0000 0000 0000 0000 0011 0101
NON binaire             1111 1111 1111 1111 1111 1111 1100 1010
==> le résultat vaut -54 en décimal (signed int)
           ou 4294967242 en décimal (unsigned int)
```

## 8.6 Les opérateurs de décalage (rotation)

Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standard dans leur format d'origine.
Les opérateurs suivants effectuent des rotations sur les bits, c'est-à-dire qu'ils décalent chacun des bits d'un nombre de bits vers la gauche ou vers la droite. La première opérande désigne la donnée sur laquelle on va faire le décalage, la seconde désigne le nombre de bits duquel elle va être décalée.

| Syntaxe | Effet |
|---|---|
| `valeur << decalage` | Décale la valeur de **decalage** bits vers la gauche. |
| `valeur >> decalage` | Décale la valeur de **decalage** bits vers la droite. |

**Table 6.** Opérateurs de décalage

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

int main()
{
    int a = 5;
    int b;

    b = a << 3;
    cout<<"b vaut : "<<b<<endl;

    b = (a+1) << 3;
    cout<<"b vaut : "<<b<<endl;

    return 0;
}
```

Après exécution :

```
b vaut 40
b vaut 48
```

# 9. Exercices

## Exercice 1 :

Écrire un programme qui calcule la moyenne entre deux nombres réels, en suivant les étapes ci-dessous :

- Déclaration de 3 variables a, b et c
- Affichage du message "Exercice 1" sur l'écran
- Affectation de valeurs réelles aux variables a et b
- Calcul de la moyenne c
- Affichage de a
- Affichage de b
- Affichage de c (la moyenne entre a et b)

Refaire le même exercice, en remplaçant les réels par des entiers.

## Exercice 2 :

Écrire puis exécuter le programme ci-dessous. Conclure.

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

int main()
{
    int a = 12345000, b = 60000000, produit;
    produit = a*b;
    cout<<"a*b = "<<produit<<"\n";

    cout <<"Pour continuer frapper une touche... (Pour Code::Blocks)"<<"\n";
    return 0;
}
```

## Exercice 3 :

Calculer et afficher $a+b$, $a-b$, $a*b$, $a/b$ et $a\%b$.
Avec $a$ et $b$ sont des entiers ($a=-52340$ et $b=6872$)

## Exercice 4 :

Calculer et afficher $a+b$, $a-b$, $a*b$ et $a/b$.
Avec $a$ et $b$ sont des réels ($a=-52340$ et $b=6872$)

## Exercice 5 :

Écrire un programme qui demande à l'utilisateur de saisir deux nombres réels $a$ et $b$, puis calcule et affiche leur somme "$s$".