

Programmation

Le langage C++

K. Boudjelaba

BTS SN-EC, Carnus



Introduction

Langage de haut niveau et de bas niveau

Installation

Téléchargement

Programmer en C++

Création d'un projet

Les commentaires

Les variables

Déclarer une variable

Affichage des variables

Lecture depuis la console

Les structures de contrôle

Les fonctions

Les classes

Les tableaux

Exercices

Un ordinateur est une machine qui ne comprend qu'un langage très simple constitué de 0 et de 1, appelé le langage machine.

→ Le programme doit être programmé en binaire : une suite de 0 et 1 (compliqué).

Afin de simplifier la programmation, les informaticiens ont développé des langages intermédiaires, plus simples que le binaire.

Étapes d'interprétation d'un programme par l'ordinateur

- ▶ Écriture du programme (instructions) dans un langage de programmation (C++, Python ...)
- ▶ Traduction des instructions en binaire grâce à un programme de traduction (compilateur : transformer le code, écrit dans un langage de programmation, en un programme exécutable par la machine)
- ▶ Lecture du code binaire et execution des instructions

Langage de haut niveau et de bas niveau

Le langage de haut niveau est un langage éloigné du binaire (langage machine) et permet généralement de développer de façon plus souple et rapide.

Le langage de bas niveau est plus proche du langage machine, il demande en général un peu plus d'efforts et permet un contrôle plus simple des instructions.

- ▶ Programmes de haut niveau : Python, Matlab, Java ...
- ▶ Programmes de bas niveau : C++ ...
- ▶ Programme machine : assembleur.

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

Éditeur 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.

Installation

- ▶ On récupère chacun de ces 3 programmes séparément (méthode compliquée)

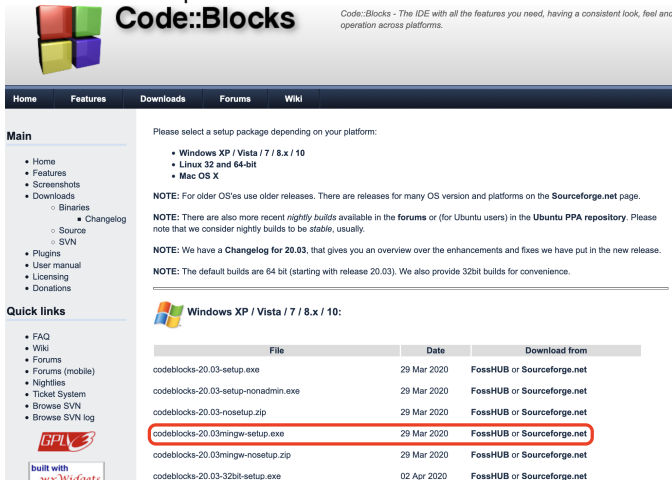
- ▶ Ou on utilise un programme qui combine éditeur de texte, compilateur et debugger. Ces programmes sont appelés IDE (EDI en français : Environnement de Développement Intégré).
Quand on code un programme, l'ordinateur génère plusieurs fichiers de code source : des fichiers.cpp, .h, les images du programme...
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.

Choix de l'IDE

- ▶ **Code::Blocks** : Gratuit et disponible pour la plupart des systèmes d'exploitation (Windows, Mac OS 32 bits et Linux).
- ▶ **Visual C++ Express** : gratuit et fonctionne sous Windows uniquement.
- ▶ **XCode** : Gratuit et disponible sur Mac OS uniquement.

<http://www.codeblocks.org/downloads/binaries>

Télécharger le logiciel en choisissant le programme dont le nom contient **mingw** (Par exemple : codeblocks-20.03mingw-setup.exe). Les autres versions sont sans compilateur.



The screenshot shows the Code::Blocks website. The main navigation bar includes Home, Features, Downloads, Forums, and Wiki. The 'Downloads' section is active, showing a list of download packages. The 'codeblocks-20.03mingw-setup.exe' package is highlighted with a red box. The website also features a sidebar with 'Main' and 'Quick links' sections, and a footer with the GPL logo and 'built with wxWidgets'.

Code::Blocks
Code::Blocks - The IDE with all the features you need, having a consistent look, feel and operation across platforms.

Home Features Downloads Forums Wiki

Main

- Home
- Features
- Screenshots
- Downloads
 - Binaries
 - Changelog
 - Source
 - SVN
- Plugins
- User manual
- Licensing
- Donations

Quick links

- FAQ
- Wiki
- Forums
- Forums (mobile)
- Nightlies
- Ticket System
- Browse SVN
- Browse SVN log

Windows XP / Vista / 7 / 8.x / 10:

File	Date	Download from
codeblocks-20.03-setup.exe	29 Mar 2020	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	29 Mar 2020	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	29 Mar 2020	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	29 Mar 2020	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	29 Mar 2020	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	02 Apr 2020	FossHUB or Sourceforge.net

built with wxWidgets

Pourquoi créer un projet ?

Un programme d'une certaine importance est constitué de plusieurs parties, souvent écrites par des personnes différentes dans une équipe. On constitue un projet en rassemblant dans un fichier des informations exploitables par le compilateur qui va créer le programme complet.

Création d'un nouveau projet

- ▶ Aller dans le menu **File** → **New** → **Project**
- ▶ Dans la fenêtre qui s'ouvre, choisir Console application
- ▶ Cliquer sur **Go** pour créer le projet
- ▶ Cliquer sur **Next**
- ▶ Choisir C++ puis **Next**
- ▶ Donner un nom à votre projet, par exemple TD01_Prog01. Indiquer l'endroit où il doit être sauvegardé (dossier d'enregistrement) puis **Next**

Création d'un nouveau projet (suite)

- ▶ Choisir de quelle façon le programme doit être compilé. Laisser les options par défaut (Debug ou Release doit être coché) :
 - ▶ L'option Create "Debug" configuration, permet à l'environnement Code::Blocks de faire de la mise au point.
 - ▶ L'option Create "Release" configuration conduit à la création d'un fichier exécutable autonome (pour le mettre sur une clé USB par exemple ...).
- ▶ Cliquer sur Finish

La configuration de notre projet est terminé. Nous pouvons passer à l'édition du programme.

Dans le panneau de gauche intitulé Projects, développer l'arborescence en cliquant sur le petit + pour afficher la liste des fichiers du projet. Il doit y avoir au moins un fichier `main.cpp` créé par défaut que vous pouvez renommer par un clic droit sur `main.cpp` en choisissant l'option *Rename file ...*

Maintenant, vous pouvez l'ouvrir en faisant un double-clic dessus.

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Code::Blocks a créé un programme très simple qui affiche à l'écran le message « Hello world! ».

Notez que l'éditeur possède la coloration syntaxique : un commentaire, un `#include`, une instruction, un entier, un réel, un caractère auront des couleurs différentes ce qui limite les erreurs ... en principe.

Lancement de la phase de compilation

Pour compiler le fichier source, utilisez l'icône **Build** (Veillez à ce que le nom de votre fichier soit sélectionné : le fichier apparaît alors en vidéo inverse). Si vous n'avez pas d'erreur, vous pouvez passer à la phase suivante. Sinon, corrigez les erreurs ...

Exécution du programme

Note : 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**

Remarque : 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.

La compilation se lance alors. Dans la section Build log, l'IDE affiche quelques messages en bas de l'IDE. Et si tout va bien, une console apparaît avec le résultat de l'exécution du programme.

Pour fermer la console, taper sur la touche **Return** ou **Enter** du clavier pour quitter l'exécution du programme.

Analyse du programme `main.cpp` généré par défaut :

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 :

```
int main()  
{  
  
}
```

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 : 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 : chaque instruction du code se termine par un point-virgule

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 */
```


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 possibilités : 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)

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

Exemple 1 : Déclarer des variables

```
#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;
}
```

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.

Exemple 2 : Déclarer une variable de type chaîne de caractères

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

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

Exemple 3 : Déclarer plusieurs variables de même type

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

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

On peut déclarer une variable sans l'initialiser (sans lui attribuer une valeur).

Type Nom_de_la_variable ;

Exemple 4 : Déclarer une variable sans l'initialiser

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

int main()
{
    string nomFormation;
    int nombreMatières;
    bool test;

    return 0;
}
```

Pour afficher la valeur de la variable nombreMatières, on utilise l'instruction :

```
cout << nombreMatières;
```

Exemple 5 :

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

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

Après exécution, la console affiche :

Le nombre de matières est : 6

Exemple 6 : Équivalent à l'exemple précédent

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

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

Exemple 7 :

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

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

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

Après exécution, la console affiche :

Nom est Charles, age est 20 ans

La lecture des valeurs tapées sur le clavier s'effectue en utilisant l'instruction `cin >>`

Exemple 8 :

```
#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, on obtient :

Entrer le nombre d'itérations :

10

Le nombre d'itérations est 10

Exemple 9 : 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 nomL 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;
}
```


Exemple 10 : 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;
}
```

Remarque : Pour réaliser des calculs mathématiques (racine carrée, sin ...), il faut inclure la librairie cmath.

```
#include <cmath>
```

Condition *if*

```
#include <iostream>

using namespace std;

int main()
{
    int a(10);

    if (a > 0)
    {
        cout << "Vous avez gagné" << endl;
    }

    cout << "Fin du programme" << endl;

    return 0;
}
```

Condition *if*, *else*

```
#include <iostream>

using namespace std;

int main()
{
    int a(0);

    if (a > 0)
    {
        cout << "Vous avez gagné" << endl;
    }
    else
    {
        cout << "Vous avez perdu" << endl;
    }

    cout << "Fin du programme" << endl;
    return 0;
}
```

Condition *else if*

```
#include <iostream>
using namespace std;
int main()
{
    int a(2);

    if (a == 0)
    {
        cout << "Moyen" << endl;
    }
    else if (a == 1)
    {
        cout << "Assez bien" << endl;
    }
    else if (a == 2)
    {
        cout << "Bien" << endl;
    }
    else
    {
        cout << "Très bien" << endl;
    }
    cout << "Fin du programme" << endl;
    return 0;
}
```

Condition *switch*

```
#include <iostream>
using namespace std;
int main()
{
    int a(2);

    switch (a)
    {
        case 0:
            cout << "Moyen" << endl;
            break;

        case 1:
            cout << "Assez bien" << endl;
            break;

        case 2:
            cout << "Bien" << endl;
            break;

        default:
            cout << "Très bien" << endl;
            break;
    }
    return 0;
}
```

La boucle *while*

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

La boucle *do...while*

```
do
{
    /* Instructions à exécuter */
} while (condition);
```

Exemple :

```
int a(0);

do
{
    cout << "Taper un nombre" << endl;
    cin >> a;
} while (a < 0);
cout << "Vous avez tapé un nbre positif. Le nombre : " << a << endl;
```

La boucle *for*

```
for (initialisation ; condition ; incrementation)
{
}

```

Exemple :

```
int main()
{
    int compteur(0);

    for (compteur = 0 ; compteur < 10 ; compteur++)
    {
        cout << compteur << endl;
    }

    return 0;
}

```

Les fonctions ont la forme suivante :

```
type nom_de_la_fonction(arguments)
{
    //Instructions effectuées par la fonction
}
```

- ▶ **type** : indique le type de variable renvoyée par la fonction (string, int, double ...)
- ▶ **nom_de_la_fonction** : permet de donner un nom à la fonction
- ▶ **arguments** : les différentes variables d'entrée de la fonction

Exemple :

```
int multiplicationDix(int nbre)
{
    int res(nbre * 10);

    return res;
}
```


Appeler la fonction créée dans l'exemple

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

int multiplicationDix(int nbre)
{
    int res(nbre * 10);

    return res;
}

int main()
{
    int a(5),b(9);
    cout << "Valeur de a : " << a << endl;
    cout << "Valeur de b : " << b << endl;
    b = multiplicationDix(b); //Appel de la fonction
    cout << "Valeur de a : " << a << endl;
    cout << "Valeur de b : " << b << endl;

    return 0;
}
```

Fonction x^n avec $x \in \mathbb{R}$ et $n \in \mathbb{N}$

```
#include <iostream.h>

double puissance(double x, int n) {
    algorithme de calcul de x^n (à coder)
}

void main() {
    double x;
    int n;
    cout << "Donne x et n : ";
    cin >> x >> n;
    cout << x << "^" << n << " = " << puissance(x, n) << "\n";
}
```

```
class Point {  
public:  
    void afficher() {  
        cout << '(' << x << ',' << y << ')';  
    }  
    void placer(int a, int b) {  
        validation des valeurs de a et b;  
        x = a; y = b;  
    }  
    double distance(Point autrePoint) {  
        int dx = x - autrePoint.x;  
        int dy = y - autrePoint.y;  
        return sqrt(dx * dx + dy * dy);  
    }  
private:  
    int x, y;  
};
```

Les tableaux correspondent aux vecteurs et matrices en mathématiques.
Un tableau est caractérisé par sa taille et par le type de ses éléments.

Tableau à 1 dimension (vecteur)

Déclaration : **type** nom[taille]

Cette instruction signifie que le compilateur réserve **taille** places en mémoire pour ranger les éléments du tableau.

Exemples :

- ▶ **int** vecteur[10] : le compilateur réserve des places en mémoire pour 10 entiers
- ▶ **float** nombre[5] : le compilateur réserve des places en mémoire pour 5 réels

Un élément du tableau est repéré par son indice. En langage C++ (C et Python) les tableaux commencent à l'indice 0. L'indice maximum est donc *taille* - 1.

Tableau à 2 dimensions (matrice)

Déclaration : `type nom[taille1][taille2]`

Exemples :

- ▶ `int matrice[10][3]` : tableau de nombres entiers de dimensions 10 lignes et 3 colonnes
- ▶ `float nombre[2][5]` : tableau de nombres réels de dimensions 2 lignes et 5 colonnes

Initialisation d'un tableau

Généralement, on initialise les tableaux au moment de leur déclaration.

```
int liste[10] = {1,2,4,8,16,3,6,12,25,52};  
  
float nombre[6] = {2.7,5.8,-8.0,0.19,3.14,-2.16};  
  
int y[2][3] = {{1,4,6},{3,7,5}}; // 2 lignes et 3 colonnes
```

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 le programme ci-dessous et conclure.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int a = 12345000, b = 60000000, produit;
    produit = a*b;
    cout<<"a*b = "<<produit<<"\n";

    cout <<"Pour continuer frapper une touche...";
    getch(); /* Attente d'une saisie clavier */
}
```

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 .

Exercice 6 :

Écrire un programme permettant de saisir un entier n , de calculer $n!$, puis de l'afficher. Utiliser une boucle `do ...while`, puis `while` puis `for`.

Quelle est la plus grande valeur possible de n , si n est déclaré *int*, puis *unsigned int* ?

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Exercice 7 :

La formule de récurrence suivante permet de calculer la racine carrée du nombre 2 :

$$U_0 = 1$$

$$U_i = \frac{U_{i-1} + \frac{2}{U_{i-1}}}{2}$$

Écrire un programme qui saisit le nombre d'itérations, puis calcule et affiche la racine carrée de 2.

Exercice 8 :

Écrire un programme pour résoudre l'équation $ax^2 + bx + c = 0$

Exercice 9 :

Écrire un programme pour saisir 6 réels, les ranger dans un tableau.
Calculer et afficher leur moyenne \bar{x} et leur écart-type σ .

$$\text{Avec } \sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}}.$$

- ▶ x_i désigne une valeur du tableau
- ▶ \bar{x} est la moyenne arithmétique des valeurs du tableau
- ▶ n est la taille du tableau

Exercice 10 :

Saisir une matrice d'entiers de dimensions 2×2 , calculer et afficher son déterminant.