

# Programmation

Le langage C++

K. Boudjelaba

BTS SN-EC, Carnus



# Table des matières

## Introduction

Langage de haut niveau et de bas niveau

## Installation

Téléchargement

## Programmer en C++

Création d'un projet

Les opérateurs arithmétiques

Les variables

Les structures de contrôle

Opérateurs logiques et relationnels

Les fonctions

Les tableaux

Les vecteurs

## Les pointeurs

## Les références

## Les fichiers

Manipulation de fichiers

Les fichiers binaires

Les fichiers CSV

# Table des matières

## Introduction

Langage de haut niveau et de bas niveau

## Installation

## Programmer en C++

## Les pointeurs

## Les références

## Les fichiers

# Introduction

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

# Introduction

## 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.

# Table des matières

Introduction

**Installation**

Téléchargement

Programmer en C++

Les pointeurs

Les références

Les fichiers

# Installation

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)

# Installation

- ▶ 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 débogage (débugage) 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).
- ▶ **XCode** : Gratuit et disponible sur Mac OS uniquement.
- ▶ **Visual Studio Code** : Nécessite l'installation d'extensions ...



# Installation - Téléchargement

<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.

## Binary releases

Please select a setup package depending on your platform:

- [Windows XP / Vista / 7 / 8.x / 10](#)
- [Linux 32 and 64-bit](#)
- [Mac OS X](#)

**NOTE:** For older OS'es use older releases. There are releases for many OS version and platforms on the [Sourceforge.net](#) page.

**NOTE:** There are also more recent nightly builds available in the [forums](#) or (for Ubuntu users) in the [Ubuntu PPA repository](#). Please note that we consider nightly builds to be stable, usually.

**NOTE:** We have a [Changelog for 20.03](#), that gives you an overview over the enhancements and fixes we have put in the new release.

**NOTE:** The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.



## Microsoft Windows

### File

codeblocks-20.03-setup.exe  
codeblocks-20.03-setup-nonadmin.exe  
codeblocks-20.03-nosetup.zip  
codeblocks-20.03mingw-setup.exe  
codeblocks-20.03mingw-nosetup.zip  
codeblocks-20.03-32bit-setup.exe  
codeblocks-20.03-32bit-setup-nonadmin.exe  
codeblocks-20.03-32bit-nosetup.zip  
codeblocks-20.03mingw-32bit-setup.exe  
codeblocks-20.03mingw-32bit-nosetup.zip

### Download from

[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)  
[FossHUB](#) or [Sourceforge.net](#)

# Table des matières

Introduction

Installation

Programmer en C++

- Création d'un projet

- Les opérateurs arithmétiques

- Les variables

- Les structures de contrôle

- Opérateurs logiques et relationnels

- Les fonctions

- Les tableaux

- Les vecteurs

Les pointeurs

Les références

# Programmer en C++ : Création d'un projet

## 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**

# Programmer en C++ : Création d'un projet

## 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.

# Programmer en C++

```
#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

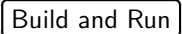
Pour compiler le fichier source, utilisez l'icône **Build** (Veillez à ce que le nom de votre fichier soit sélectionné : le fichier apparaît alors en vidéo inverse). Si vous n'avez pas d'erreur, vous pouvez passer à la phase suivante. Sinon, corrigez les erreurs ...

# Programmer en C++



## 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 

**Remarque :** l'icône  (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  ou  du clavier pour quitter l'exécution du programme.

# Programmer en C++

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.

# Programmer en C++

## 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



# Programmer en C++

## 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).

# Programmer en C++

## 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

# Programmer en C++ : 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 */
```

# Programmer en C++ : Les opérateurs arithmétiques

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

Table 1: Opérateurs arithmétiques

## Modulo (%)

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 ( $7 = 2 \times 3 + 1$ ). Donc  $7 \% 3$  renvoie 1.

# Programmer en C++ : Les opérateurs arithmétiques

```
#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;
}
```

## Les raccourcis

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 ( $a \bmod b$ )
$a = a + 1;$	$a++;$ si a est un entier
$a = a - 1;$	$a--;$ si a est un entier

Table 2: Les raccourcis

# Programmer en C++ : Les variables

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)

# Programmer en C++ : Déclarer une variable

La déclaration se fait de la manière suivante :

Type Nom\_de\_la\_variable(Valeur);

ou

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 booleen vrai (true)

    char var4('a');         // var4 contient le caractère "a"

    return 0;
}
```

# Programmer en C++ : Déclarer une variable

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");
}
```



# Programmer en C++ : Déclarer une variable

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;
```

# Programmer en C++ : Affichage des variables

## 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;
}
```

# Programmer en C++ : Affichage des variables

## Exemple 7 :

```
#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

# Programmer en C++ : Lecture depuis la console

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

# Programmer en C++ : Lecture depuis la console

**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 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;
}
```

# Programmer en C++ : Lecture depuis la console

**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>`

# C++ : Les structures de contrôle

## Condition *if*

```
if (condition)
{
    instructions;
}
```

```
#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;
}
```

# C++ : Les structures de contrôle

## Condition *if*, *else*

```
if (condition)
{
    instructions 1;
}
else
{
    instructions 2;
}
```

```
#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;
}
```



# C++ : Les structures de contrôle

## Condition *else if*

```
if (condition1)
{
    instructions 1;
}
else if (condition2)
{
    instructions 2;
}
else
{
    instructions 3;
}
```

```
#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;
}
```

# C++ : Les structures de contrôle

## 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;
}
```

# C++ : Les structures de contrôle

## 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;
```

# C++ : Les structures de contrôle

## La boucle *for*

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

Exemple :

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

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

    return 0;
}
```

# C++ : Opérateurs logiques et relationnels

Opérateur	Symbole
AND	&&
OR	
NOT	!

Table 3: Opérateurs logiques en C++

Relation	Symbole
Egal à	== (mais pas = : affectation)
Différent de ( $\neq$ )	!=
Supérieur à	>
Supérieur ou égal à ( $\geq$ )	>=
Inférieur à	<
Inférieur ou égal à ( $\leq$ )	<=

Table 4: Opérateurs relationnels en C++

# C++ : Opérateurs logiques et relationnels

```
double x, z, p, q;  
double y;  
if ((x > z) && (p > q))  
{  
    // Les deux conditions sont remplies  
    y = 10.0;  
}
```

```
double p, q;  
int i;  
double y;  
if ((p > q) || (i != 1))  
{  
    // Une ou les deux conditions sont remplies  
    y = 10.0;  
}  
else  
{  
    // Aucune condition n'est remplie : p<=q et i==1  
    y = -10.0;  
}
```

# C++ : Les fonctions

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;
}
```

# C++ : Les fonctions

## 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;
}
```



# C++ : Les fonctions

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 << "Donner x et n : ";
    cin >> x >> n;
    cout << x << "^" << n << " = " << puissance(x, n) << "\n";
}
```

# C++ : Les tableaux

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.

# C++ : Les tableaux

## 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
```

# C++ : Les vecteurs

## Définitions :

- ▶ Un tableau dynamique est un tableau dont le nombre de cases (taille) peut varier au cours de l'exécution du programme (sa taille est ajustable).
- ▶ Les vecteurs (vectors) sont un type de tableaux dynamiques.

Pour utiliser les vecteurs en C++, il faut inclure dans l'en-tête :

```
#include <vector>
```

Pour déclarer un vecteur, il faut utiliser la syntaxe suivante :

```
vector</* type des éléments du tableau */> identifiant {};
```

## Exemple :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    vector<double> const tableau_de_double {};
    vector<string> const tableau_de_string {};
    return 0;
}
```

# C++ : Les vecteurs

## La fonction `at()` :

Pour accéder à un élément d'un vecteur, on peut utiliser l'opérateur `[]`  
Par exemple, pour accéder au 3ème élément du vecteur `tableau_de_double`, on procède comme suit :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 }; // dépend de la
        version C$++$
cout << vecteur[2] << endl;
```

Si on met entre les `[]`, une valeur qui dépasse la "taille du vecteur -1", le programme peut planter ou avoir un comportement indéterminé.

Pour pallier ce problème, on préfère utiliser la fonction `at()` :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 };
cout << vecteur.at(2) << endl;
```

# C++ : Les vecteurs

## Les fonctions **front()** et **back()** :

- ▶ La fonction **front()** permet d'accéder au premier élément
- ▶ La fonction **back()** permet d'accéder au dernier élément

## Les fonctions **empty()** et **clear()** :

- ▶ La fonction **empty()** permet de savoir si le vecteur est vide ou non.
- ▶ La fonction **clear()** permet de vider tout le vecteur.

# C++ : Les vecteurs

## Exemple :

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> vecteur;
    // Remplissage du vecteur
    for (int i = 20; i < 29; ++i)
        vecteur.push_back(i);
    cout << "Le vecteur contient " << vecteur.size() << " éléments : "
         << endl;
    for (int i = 0; i < vecteur.size(); ++i)
    {
        cout << "\t- Valeur " << i+1 << " est : " << vecteur[i] << endl;
    }
    // On ajoute 3 éléments
    for (int i = 0; i < 3; ++i)
    {
        vecteur.push_back(100);
    }
    cout << endl << "La nouvelle taille est : " << vecteur.size() << "
         éléments : " << endl;
    for (int i = 0; i < vecteur.size(); ++i)
    {
        cout << "\t- N-Val " << i+1 << " est : " << vecteur[i] << endl;
    }
}
```

# Table des matières

Introduction

Installation

Programmer en C++

**Les pointeurs**

Les références

Les fichiers



# Les pointeurs

## Case mémoire et adresse

- ▶ Tout objet (variable, fonction . . . ) manipulé par l'ordinateur est stocké dans sa mémoire, constituée d'une série de cases.
- ▶ Pour accéder à un objet (au contenu de la case mémoire dans laquelle cet objet est enregistré), il faut connaître le numéro de cette case. Ce numéro est appelé l'adresse de la case mémoire.
- ▶ Lorsqu'on utilise une variable ou une fonction, le compilateur utilise l'adresse de cette dernière pour y accéder.

### Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable (objet).

# Les pointeurs

Adresse mémoire					

# Les pointeurs

Adresse mémoire					
1	2	3	4	5	...
10673	10674	10675	10676	10677	...
35426	35427	35428	35429	35430	...
...	...	...	...	...	...

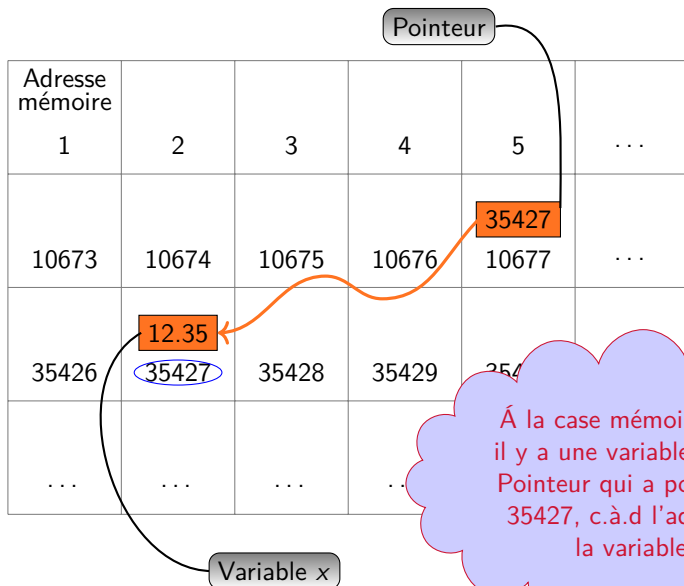
# Les pointeurs

Adresse mémoire					
1	2	3	4		
10673	10674	10675	10676	10677	...
35426	12.35 35427	35428	35429	35430	...
...	...	...	...	...	...

La variable  $x = 12.35$   
occupe la case mémoire  
n° 35427

Variable  $x$

# Les pointeurs



À la case mémoire 10677, il y a une variable nommée Pointeur qui a pour valeur 35427, c.à.d l'adresse de la variable x

# Les pointeurs

## Pointeur :

Pour déclarer un pointeur, on procède de la même manière que pour les variables, c.à.d. déclarer :

- ▶ le type
- ▶ le nom, précédé par \*

```
int *pointeur;
```

On peut aussi utiliser cette syntaxe :

```
int* pointeur;
```

Inconvénient : ne permet pas de déclarer plusieurs pointeurs sur la même ligne.

```
int* p1, p2, p3, p4;
```

Seul *p1* sera un pointeur, les autres variables seront des entiers standards

## Exemple :

```
string *pointeur1;  
//Un pointeur qui peut contenir l'adresse d'une chaîne de caractères  
  
vector<int> *pointeur2;  
//Un pointeur qui peut contenir l'adresse d'un tableau de nombres entiers
```

# Les pointeurs

## Exemple 1 :

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    cout << &x << endl;
    return 0;
}
```

En C++, le symbole pour obtenir l'adresse d'une variable est `&`. Pour afficher l'adresse de la variable `x`, on doit écrire `&x`.

Après exécution du code, la console affiche `0x7ffefbfff538`, qui correspond à l'adresse (en hexadécimal) de la case mémoire contenant la variable `x`.

## Exemple 1 (suite) :

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    cout << &x << endl;

    int *p_x = nullptr
    // ou int *p_x(0);
    p_x = &x;
    cout << p_x << endl;
    return 0;
}
```

**Important :** Les pointeurs doivent être déclarés en les initialisant à 0.

La console affiche :

`0x7ffefbfff538`

`0x7ffefbfff538`

L'adresse de `x` est sauvegardée dans le pointeur `p_x`.

On dit alors que le pointeur `p_x` pointe sur `x`.

# Les pointeurs

## Exemple 2 :

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

int main()
{
    int a, b, c;
    int *x(nullptr), *y(nullptr);

    a = 98;
    x = &a;
    c = *x + 5;
    y = &b;
    *y = a + 10;

    cout << "b = " << b << endl;
    cout << "c = " << c << endl;

    return 0;
}
```

La console affiche :

b = 108

c = 103

- ▶  $a$  est initialisé à 98.
- ▶  $x = \&a$ ; affecté à  $x$  l'adresse de  $a$ .  
 $x$  est un pointeur vers  $a$ .
- ▶  $*x$  est la variable pointée pas  $x$ , c.à.d  $a$  (=98).
- ▶  $c = *x + 5$ ; permet de transférer  $98 + 5 = 103$  dans la variable  $c$ .
- ▶  $y = \&b$ ; permet de mettre dans la variable  $y$  l'adresse de la variable  $b$ .  $y$  un pointeur vers  $b$ .  
 $a + 10 = 98 + 10 = 108$ .
- ▶  $*y = a + 10$ ; permet de transférer dans la variable pointée par  $y$  la valeur de  $a + 10 = 108$ .  
On stocke 108 dans  $b$  de manière indirecte via le pointeur  $y$ .
- ▶ Affichage des valeurs de  $b$  et  $c$ .



# Table des matières

Introduction

Installation

Programmer en C++

Les pointeurs

**Les références**

Les fichiers

# Les références

## Référence

- ▶ Une référence peut être vue comme un alias d'une variable (utiliser la variable ou une référence à cette variable est équivalent).
- ▶ On peut modifier le contenu de la variable en utilisant une référence.
- ▶ Une référence ne peut être initialisée qu'une seule fois : à la déclaration. Toute autre affectation modifie en fait la variable référencée.
- ▶ Une référence ne peut donc référencer qu'une seule variable.
- ▶ Les références sont principalement utilisées pour passer des paramètres aux fonctions.

## Déclaration

`type &reference = identificateur;`

## Exemple

```
int x = 0;  
int &r_x = x;           // Référence sur la variable x.  
r_x = r_x + x;         // Double la valeur de x (et de r_x).
```

# Les références

## Exemple 1 :

```
#include <iostream>
using namespace std;
int main()
{
    int a = 98, b = 78, c;
    int &x = a;
    c = x + 5; // équivalent à : c = a + 5;
    int &y = b;
    y = a + 10; // équivalent à : b = a + 10;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    return 0;
}
```

La console affiche :

b = 108

c = 103

- ▶ `int &x = a;` permet de déclarer une référence `x` vers la variable `a`.
- ▶ `c = x + 5;` permet donc de transférer 103 dans la variable `c`.
- ▶ `int &y = b;` permet de déclarer une référence `y` vers la variable `b`.
- ▶ `y = a + 10;` permet de transférer 108 dans la variable `b`.

# Table des matières

Introduction

Installation

Programmer en C++

Les pointeurs

Les références

**Les fichiers**

Manipulation de fichiers

Les fichiers binaires

Les fichiers CSV

# Les fichiers

## Problématique

Les variables manipulées dans un programme sont des adresses de mémoire vive (RAM) et disparaissent à chaque fin d'exécution du programme ou au plus tard dès la fermeture du programme.

Les fichiers servent donc à stocker des informations de manière permanente.

## Définitions

- ▶ Un fichier est un ensemble de données numériques réunies sous un même nom, enregistrées sur un support de stockage.
- ▶ Dans un nom de fichier, on trouve souvent l'extension , qui renseigne sur la nature des informations contenues dans le fichier et les logiciels utilisables pour le manipuler.
- ▶ Un fichier contient des métadonnées : son auteur, sa date de création, les personnes autorisées à le manipuler ...

# Les fichiers

Afin de faciliter leur localisation, les fichiers sont disposés et organisés dans des systèmes de fichiers qui permettent de placer les fichiers dans des emplacements appelés répertoires (dossiers).

→ Les fichiers sont répartis dans une arborescence de dossiers et on peut les localiser à partir d'un chemin d'accès.

Un chemin d'accès permet d'indiquer l'emplacement d'un fichier dans le système de fichiers. Il contient le nom du fichier concerné, mais également, un ou plusieurs noms de dossiers, qu'il est nécessaire de traverser pour accéder au fichier depuis la racine.

## Exemples :

- ▶ Chemin absolu : On peut utiliser / à la place de \

```
C:\Documents\Cours_Informatique\Data.txt
```

- ▶ Chemin relatif : si on est déjà dans le dossier Documents

```
\Cours_Informatique\Data.txt  
Ou  
.\Cours_Informatique\Data.txt
```

# Les fichiers

## Types d'accès

Le type d'accès est la manière dont la machine procède pour aller rechercher les informations contenues dans le fichier.

- ▶ **L'accès séquentiel** : on ne peut accéder qu'à la donnée suivant celle qu'on vient de lire. On ne peut donc accéder à une information qu'en ayant au préalable examiné celle qui la précède.
- ▶ **L'accès direct** : on peut accéder directement à l'enregistrement de son choix, en précisant le numéro de cet enregistrement.
- ▶ **L'accès indexé** : il combine la rapidité de l'accès direct et la simplicité de l'accès séquentiel. Il est adapté au traitement des gros fichiers (bases de données par exemple).

# Les fichiers

## Modes d'ouverture

- ▶ **Ouverture en mode lecture** : On peut uniquement récupérer les informations qu'il contient, sans pouvoir les modifier
- ▶ **Ouverture en mode écriture** : On peut écrire toutes les informations que l'on veut. Mais les informations précédentes, si elles existent, seront intégralement écrasées.
- ▶ **Ouverture en mode ajout** : on ne peut ni lire, ni modifier les informations existantes. Mais on peut ajouter de nouvelles informations.



# Les fichiers

## Les fichiers textes :

Ils contiennent des informations sous la forme de caractères (en utilisant le code ASCII). Exemple : Le code source d'un programme C++

## Les fichiers binaires :

Ils contiennent directement la représentation mémoire des informations.

Un fichier binaire peut aussi être vu comme une séquence d'octets.

Exemple : le résultat de la compilation d'un programme C++  
(programme exécutable)

# Les fichiers

## Écrire dans un fichier :

- ▶ L'ouverture et l'écriture sur un fichier se fait avec l'objet **ofstream**
- ▶ Les modes d'ouverture sont :
  - ▶ **ios::app** (append) ajout à la fin du fichier
  - ▶ **ios::ate** (at end) met l'index à la fin du fichier
  - ▶ **ios::binary** pour ouvrir un fichier binaire
  - ▶ **ios::in** (input) permet la lecture
  - ▶ **ios::out** permet l'écriture
  - ▶ **ios::trunc** (truncate) vide le fichier à l'ouverture

**Remarque :** **ofstream** est par défaut un mode d'ouverture **ios\_base::out | ios\_base::trunc** (ouverture en écriture et effacement du contenu du fichier)

# Les fichiers

Il est préférable, avant toute opération sur un fichier, de tester les indicateurs d'état en utilisant une des méthodes suivantes :

- ▶ **bad()** : Retourne la valeur **True** si une opération de lecture échoue.  
Exemple : essayer d'écrire dans un fichier qui n'est pas ouvert en écriture
- ▶ **fail()** : Retourne la valeur **True** dans le même cas que **bad()** et aussi dans le cas où une erreur de format se produit.
- ▶ **eof()** : Retourne la valeur **True** si un fichier ouvert en lecture a atteint la fin
- ▶ **good()** : Retourne la valeur **False** lorsque les méthodes précédentes retourneraient **True**

**Exemple :**

```
ofstream fichier("Data.txt");
if (fichier.good())
{
    fichier << 180 << endl; // Ecriture
}
else
{
    cerr << "ERREUR : Impossible d'écrire dans le fichier !" << endl;
    exit(1);
}
```

# Les fichiers - Manipulation de fichiers

En C++, on peut utiliser la librairie `fstream` (file stream).

```
#include <fstream>
```

## ► Ouverture de fichiers :

```
ofstream fichier("Data.txt");  
// Ou  
ifstream fichier("Data.txt");  
// A utiliser quand le fichier existe
```

On ouvre un fichier nommé "Data.txt", qui se trouve dans le dossier du projet. S'il existe, il sera ouvert. Sinon, il sera d'abord créé puis ouvert.

## ► Ecriture dans un fichier :

```
ofstream fichier("Data.txt");  
// On écrit un 5, une tabulation et un 10.  
fichier << 5 << "\t" << 10;
```

## ► Ouverture sans effacer (ouverture en mode ajout) :

```
std::ofstream fichier("Data.txt", std::ios::app);
```

app : append (ajouter).

# Les fichiers - Les fichiers binaires

La manipulation d'un fichier binaire s'effectue presque comme celle d'un fichier texte. La différence réside au niveau de la désignation du fichier et l'écriture (la lecture) des données.

`<<` → **f.write()**

`>>` → **f.read()**

Pour créer un fichier binaire, il faut ajouter le mode **ios::binary**  
**ofstream** fichier("Data.txt", **ios::binary**);

Pour écrire une variable **y** (∀ son type) : fichier.write((char\*)x, sizeof(x));

Pour lire une variable **y** à partir d'un fichier (∀ son type) :

fichier.read((char\*)x, sizeof(x));

## Curseur dans le fichier

- **fichier.tellp();** : renvoie la position de la tête d'écriture (de lecture), exprimée en nombre d'octets depuis le début du fichier.

```
ofstream fichier("Data.txt");  
int position = fichier.tellp(); //On récupère la position  
cout<< "Nous nous situons au " << position << "ème caractère du  
fichier." <<endl;
```

# Les fichiers - Les fichiers binaires

## Curseur dans le fichier

- ▶ **fichier.seekp(nombre d'octets, position);** : permet de se déplacer.

Les trois positions possibles sont :

- ▶ Début du fichier : **ios::beg**
- ▶ Fin du fichier : **ios::end**
- ▶ Position actuelle : **ios::cur**

```
ifstream fichier("Data.txt");  
fichier.seekg(10,ios::beg); //se placer au 10ème caractère après le  
    début du fichier  
fichier.seekg(20,ios::cur); //aller 20 caractères plus loin que l'  
    endroit où se situe le curseur
```

## La taille

Pour connaître la taille d'un fichier, on se déplace à la fin et on demande au curseur où il se trouve.

```
#include <iostream>  
#include <fstream>  
using namespace std;  
int main() {  
    ifstream fichier("Data.txt"); //On ouvre le fichier  
    fichier.seekg(0, ios::end); //On se déplace à la fin du fichier  
    int taille;  taille = fichier.tellg(); //On récupère la position qui  
        correspond donc à la taille du fichier !  
    cout << "Taille du fichier :" << taille << " octets." << endl;  
    return 0; }
```

# Les fichiers - Les fichiers CSV

## Définitions :

- ▶ Un fichier CSV (Comma-Separated Values: Valeurs séparées par des virgules) désigne un fichier informatique de type tableur, dont les valeurs sont séparées par des virgules.  
Un fichier CSV est un fichier texte, par opposition aux formats binaires.
- ▶ Chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau. Une ligne est une suite ordonnée de caractères terminée par un caractère de fin de ligne.
- ▶ Selon le logiciel et les paramètres, on peut parfois utiliser un point-virgule, un espace ou d'autres caractères, pour séparer les différentes données d'une même ligne.

# Les fichiers - Les fichiers CSV

## Exemple : Création d'un fichier CSV

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

int main() {
    ofstream fichier("C:/Documents/Tableur1.csv");

    fichier << "En-Tête1, En-Tête2\n";
    fichier << "10, 11\n";
    fichier << "20, 21\n";
    fichier << "30, 31\n";

    fichier.close();

    return 0;
}
```



# Les fichiers - Les fichiers CSV

## Exemple : CSV – Tableau avec 1 colonne

```
#include <string>
#include <fstream>
#include <vector>
using namespace std;

void ecrire_csv(string filename, string colname, vector<int> vals)
{
    // Fonction pour écrire dans un fichier CSV
    // filename - Nom du fichier
    // colname - Nom de l'unique colonne
    // vals - Vecteur de valeurs entieres
    ofstream fichier(filename);
    fichier << colname << "\n";
    for(int i = 0; i < vals.size(); ++i)
    {
        fichier << vals.at(i) << "\n";
    }
    fichier.close();
}

int main()
{
    vector<int> vec(10, 1);
    ecrire_csv("C:/Documents/Tableur2.csv", "Colonne 1", vec);
    return 0;
}
```

# C++ : Les fichiers

## Exercice 1 : Solution

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

int minmax(int *p, int n, int *pmin, int *pmax)
{
    int *i(0);
    *pmin=*p;
    *pmax=*p;

    for (i=p ; i<p+n ; i++)
    {
        if (*i < *pmin) *pmin = *i;
        if (*i > *pmax) *pmax = *i;
    }
    return 0;
}

int main()
{
    int tn[] = { 12, 23, 36, 5, 46, 9, 25 };
    int min, max, r;

    r = minmax(tn, sizeof(tn)/sizeof(int), &min, &max);
    cout << "Min, Max : " << min << " " << max << endl;
    return 0;
}
```