



**C++**  
**Programmation**

Prof : **KAMAL BOUDJELABA**

4 novembre 2025

## Table des matières

<b>1</b>	<b>Installation du logiciel (des outils de base)</b>	<b>1</b>
<b>2</b>	<b>Structure de base d'un programme en C++</b>	<b>1</b>
2.1	Les commentaires dans le code	2
<b>3</b>	<b>Les variables</b>	<b>2</b>
3.1	Déclaration de variables	3
3.2	Les constantes	3
3.3	Chaînes de caractères (type string)	3
<b>4</b>	<b>Opérateurs logiques et booléens</b>	<b>3</b>
<b>5</b>	<b>Affichage (cout) et formatage</b>	<b>4</b>
<b>6</b>	<b>Lecture clavier (cin et getline)</b>	<b>4</b>
<b>7</b>	<b>Le flux d'erreurs (cerr)</b>	<b>4</b>
<b>8</b>	<b>Les opérateurs arithmétiques</b>	<b>5</b>
<b>9</b>	<b>Opérateurs bit à bit et décalages</b>	<b>6</b>
<b>10</b>	<b>Exercices</b>	<b>7</b>
<b>11</b>	<b>Structures conditionnelles</b>	<b>8</b>
11.1	L'instruction if	8
11.2	L'instruction else if	8
11.3	L'instruction switch	8
11.4	Opérateurs logiques et relationnels	9
<b>12</b>	<b>Les boucles</b>	<b>10</b>
12.1	Boucle for	10
12.2	Boucle while	10
12.3	Boucle do ... while	10
<b>13</b>	<b>Les tableaux</b>	<b>11</b>
<b>14</b>	<b>Les fonctions</b>	<b>13</b>
<b>15</b>	<b>Notions avancées</b>	<b>13</b>
<b>16</b>	<b>Résumé</b>	<b>14</b>

## Liste des tableaux

1	Types de variables	2
2	Opérateurs logiques	3
3	Commandes de formatage	4
4	Opérateurs arithmétiques	5
5	Raccourcis opérateurs	5
6	Opérateurs bit à bit et décalages	6
7	Opérateurs logiques en C++	9
8	Opérateurs relationnels en C++	9

## 1. Installation du logiciel (des outils de base)

Pour programmer en C++, on aura besoin de **trois outils principaux** :

**Éditeur de texte** (avec coloration syntaxique) : VS Code, Notepad++, etc.

**Compilateur** (pour traduire le code) : g++, MinGW, Clang, etc.

**Débogueur** (pour corriger les erreurs) : Souvent inclus dans IDE ou en ligne de commande.

### Procédure d'Installation :

Pour simplifier, utiliser un **IDE** (Integrated Development Environment, EDI en français : Environnement de Développement Intégré) qui regroupe ces outils :

- **Code::Blocks** : Gratuit et multiplateforme (Windows, Mac OS 32 bits et Linux).
- **VS Code + extensions C++** : Gratuit et multiplateforme.
- **XCode** : Gratuit et sur Mac OS seulement.
- **autres** : CLion, Eclipse CDT, etc.

### TP : Création d'un programme

1. Lancer votre éditeur ou IDE.
2. Créer un fichier `hello.cpp`.

## 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      cout << "Hello world!" << endl;
6      // Ou std::cout << "Hello World!\n"; si on supprime la ligne 2
7      return 0;
8  }
```

### Analyse du programme

- `#include <iostream>` : bibliothèque pour entrées/sorties.
- `using namespace std;` : simplifie l'accès aux objets standards.
- `int main() {}` : point d'entrée du programme.
- `cout` : affiche à l'écran.
- `endl` : retour à la ligne.

`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.

### Remarque 2.1 :

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

## TP : Ecriture, Compilation et Exécution

- Copiez le code de l'Exemple 2.1 et enregistrez-le.
- Compilez (`g++ hello.cpp -o hello`) et exécutez (`./hello`). Vous pouvez utiliser le menu ou le bouton d'exécution.
- Modifiez le message, recompilez, puis lancez-le à nouveau.

### NOTE : Erreur classique

Oublier le ; en fin d'instruction : lisez bien les messages du compilateur !

## 2.1 Les commentaires dans le code

Les commentaires sont nécessaires pour expliquer le fonctionnement du programme. Ils ne sont pas exécutés par le compilateur.

Commentaires sur une ligne :

```
1 // Votre commentaire
```

Commentaires sur plusieurs lignes :

```
1 /* Début
2 du commentaire
3 multi-lignes */
```

### TP : Commenter le code

Ajoutez des commentaires à votre exemple précédent : chaque variable, chaque bloc important.

## 3. Les variables

Une variable est un espace nommé qui stocke une valeur. Dans les programmes C++, comme dans la plupart des langages compilés, les variables ainsi que leur types doivent être déclarés avant d'être utilisées.

Les règles de nommage :

- 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 ni d'espaces dans le nom.

Nom du type	Type d'élément de la variable
bool	Une valeur parmi 2 possibilités : vrai ( <code>true</code> ) ou faux ( <code>false</code> )
char	Un caractère ASCII
int	Un entier
unsigned int	Un nombre entier positif ou nul
double	Un nombre réel
string	Une chaîne de caractères

Table 1. Types de variables

### 3.1 Déclaration de variables

Une déclaration se fait ainsi : `Type Nom_de_la_variable = Valeur;`

Exemple (dans fonction `main()`) :

```
1 /* - Déclaration avec initialisation */
2 int var1 = 16;           // entier valant 16
3 double var2 = 4.53;      // réel valant 4.53
4 double erreur = 1.0e-12; // 10 puissance -12 (notation scientifique)
5 bool var3 = true;        // booléen vrai
6 char var4 = 'a';         // caractère 'a'
7 string nom = "Carnus";   // chaîne de caractères "Carnus"
8 string chaîne = "BTS CIEL"; // chaîne de caractères "BTS CIEL"
9 /* - Déclaration sans initialisation */
10 int a;
11 double x;
12 bool test;
13 string phrase;
```

### 3.2 Les constantes

Une constante est une variable dont la valeur ne peut pas changer :

```
1 const double pi = 3.14159;
2 // ou via préprocesseur (moins moderne):
3 #define PI 3.14159
```

#### TP : Constantes

Initialiser une variable `const` puis la modifier dans le code. Que vous répond le compilateur ?

### 3.3 Chaînes de caractères (type `string`)

Pour utiliser `string`, il faut inclure le fichier d'en-tête `string` (`#include <string>`) (mais ici on omet cette ligne).

Exemple d'utilisation :

```
1 string ville = "Rodez";
2 cout << "Longueur: " << ville.length() << endl;
3 cout << "3e lettre: " << ville[2] << endl;
```

#### Chaîne de caractères

Écrire un programme qui demande une phrase complète, puis affiche sa longueur et la première lettre.

## 4. Opérateurs logiques et booléens

Opérateur	Symbole	Exemple
ET logique	<code>&amp;&amp;</code>	<code>(a &gt; 0 &amp;&amp; b &gt; 0)</code>
OU logique	<code>  </code>	<code>(a == 0    b == 0)</code>
NON logique	<code>!</code>	<code>!(a &lt; 5)</code>

Table 2. Opérateurs logiques

Retourne un booléen (`true` ou `false`).

```
1 int x = 5, y = 10;
2 bool test = (x < y && x != 0);
3 cout << test << endl; // Affiche 1 (vrai)
```

## 5. Affichage (cout) et formatage

Pour afficher un texte ou une variable, on utilise l'instruction `cout <<` (console output) :

```
1 cout << "Texte " << variable << endl;
```

Caractères spéciaux utiles en sortie :

Commande	Symbole
Nouvelle ligne	<code>\n</code>
Tabulation	<code>\t</code>
Guillemet simple	<code>\'</code>
Guillemet double	<code>\"</code>

Table 3. Commandes de formatage

## 6. Lecture clavier (cin et getline)

Lecture simple (sans espaces) :

```
1 int age;
2 cin >> age;
```

Lecture de ligne complète (avec espaces) :

```
1 string nomCompleet;
2 getline(cin, nomCompleet);
```

**Attention** : après un `cin >>`, ajoutez `cin.ignore()` ; avant un `getline()` pour nettoyer le retour chariot restant.

### Exemple 6.1: Exemple complet

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main() {
6     double note;
7     cout << "Note sur 20 ? ";
8     cin >> note;
9
10    cin.ignore(); // Évite que getline saute la ligne
11
12    string etudiant;
13    cout << "Nom complet ? ";
14    getline(cin, etudiant);
15
16    cout << etudiant << " a obtenu " << note << "/20." << endl;
17    return 0;
18 }
```

### TP : cout et cin

- (a) Demander à l'utilisateur son prénom et afficher « Bonjour ...! » (avec `getline` pour prénoms composés).
- (b) Demander l'année de naissance, calculer l'âge (en 2025), vérifier et afficher une erreur avec `cerr` si invalide (voir ci-dessous).

## 7. Le flux d'erreurs (cerr)

— `cout` : affichage normal (bufferisé).

— `cerr` : affichage d'erreur (non bufferisé, immédiat).

Exemple :

```
1 cerr << "Erreur : division par zéro." << endl;
```

Utile pour séparer sorties normales et erreurs.

## 8. Les opérateurs arithmétiques

Opération	Symbole
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo (reste)	%

Table 4. Opérateurs arithmétiques

Exemple division entière *vs* réelle :

```
1 int a = 7, b = 3;
2 cout << a / b << endl; // 2 (division entière)
3 cout << (double)a / b << endl; // 2.3333 (division réelle)
```

### Note 8.1.

L'opérateur `%` (exp : `a%b`) renvoie le reste de la division euclidienne de  $a$  par  $b$ .

L'opérateur modulo 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.

### Remarque 8.1 :

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

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main(int argc, char* argv[])
6 {
7     double x = 1.0, y = 2.0, z;
8     z = sqrt(x); // Racine carrée
9     z = exp(y); // Exponentiel
10    z = pow(x, y); // x à la puissance de y
11    z = M_PI; // Valeur de pi
12    return 0;
13 }
```

### Raccourcis des opérateurs

Expression complète	Raccourci
<code>a = a + b;</code>	<code>a += b;</code>
<code>a = a - b;</code>	<code>a -= b;</code>
<code>a = a * b;</code>	<code>a *= b;</code>
<code>a = a / b;</code>	<code>a /= b;</code>
<code>a = a % b;</code>	<code>a %= b;</code>
<code>a = a + 1;</code>	<code>a++;</code>
<code>a = a - 1;</code>	<code>a--;</code>

Table 5. Raccourcis opérateurs

## TP : Les raccourcis

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

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     double nombre = 1.5;
7     nombre++;      // 'Opération'
8     cout << "La valeur actuelle est : " << nombre << endl;
9     return 0;
10 }
```

### Instructions :

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

## 9. Opérateurs bit à bit et décalages

Opérateur	Description	Exemple
&	ET bit à bit	$a \& b$
	OU bit à bit	$a   b$
^	OU exclusif (XOR)	$a \wedge b$
~	NON bit à bit	$\sim a$
<<	Décalage à gauche	$a \ll 1$ (mult. par $2^1 = 2$ )
>>	Décalage à droite	$a \gg 2$ (div. par $2^2 = 4$ )

Table 6. Opérateurs bit à bit et décalages



**TP : Opérateurs bit à bit**

Tester ce programme, puis ajouter des lignes pour afficher les résultats des instructions données ci-dessous :

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int a = 5;      // binaire : 0101
7     int b = 3;      // binaire : 0011
8
9     cout << (a & b) << endl; // 1  --> 0001
10    return 0;
11 }
```

**Instructions :**

```
a | b;
a ^ b;
~a;
a << 1;
a << 3.;
```

**10. Exercices****Exercice 1**

Écrire un programme qui demande le prénom de l'utilisateur et affiche « Bonjour, ...! ».

**Exercice 2**

Lire deux entiers, afficher quotient (division entière), reste, et division réelle (avec conversion).

**Exercice 3**

Lire un entier, afficher carré et cube (bonus : utiliser `pow()`).

**Exercice 4**

Lire un entier, un réel, une chaîne (avec espaces), puis afficher.

**Exercice 5**

Lire un entier, afficher sa valeur multipliée par 2 (`<< 1`) et divisée par 4 (`>> 2`).

## 11. Structures conditionnelles

### 11.1 L'instruction `if`

Pour exécuter un bloc d'instructions seulement si une condition est vraie :

```
1 if (condition) {
2     // instructions à exécuter si vrai
3 }
```

Il est possible de proposer une alternative avec `else` :

```
1 if (age < 18) {
2     cout << "Mineur." << endl;
3 } else {
4     cout << "Majeur." << endl;
5 }
```

**Remarque :**

- Les  **accolades**  sont obligatoires si plusieurs instructions suivent le `if`.
- L'expression entre parenthèses doit renvoyer vrai (`true`) ou faux (`false`).

#### TP - Structures simples

- (a) Écrire un programme qui demande à l'utilisateur de saisir un nombre et qui affiche si ce nombre est positif ou nul ( $\geq 0$ ), négatif ( $< 0$ ).
- (b) Écrire un programme qui lit l'âge d'une personne et affiche "Mineur" ou "Majeur".

### 11.2 L'instruction `else if`

Pour enchaîner plusieurs cas distincts :

```
1 if (note >= 16) {
2     cout << "Excellent";
3 } else if (note >= 12) {
4     cout << "Bien";
5 } else if (note >= 10) {
6     cout << "Moyen";
7 } else {
8     cout << "Insuffisant";
9 }
```

#### TP - Conditions multiples (Grille de notation)

Écrire un programme qui lit une note sur 20 et affiche une appréciation selon les fourchettes données ci-dessus.

### 11.3 L'instruction `switch`

Pour gérer de nombreux cas sur une variable entière ou caractère :

```
1 switch (choix) {
2     case 1:
3         cout << "Menu 1"; break;
4     case 2:
5         cout << "Menu 2"; break;
6     default:
7         cout << "Invalide";
8 }
```

- Le mot-clé `break` sert à sortir du `switch` après un cas traité.
- Toujours prévoir un cas `default` pour les valeurs inattendues.

### TP : Découverte de switch

Écrire un programme qui demande à l'utilisateur de saisir un numéro de mois (1 à 12) et qui affiche le nom du mois correspondant.

### TP - Switch

Écrire un programme qui propose un menu simple (1 : Démarrer, 2 : Arrêter, 3 : Redémarrer), lit le choix de l'utilisateur, et affiche l'action correspondante.

## 11.4 Opérateurs logiques et relationnels

Les opérateurs logiques permettent de combiner plusieurs conditions.

Une première utilité de combinaison d'opérateurs logiques et relationnels est de remplacer des instructions *if* imbriquées par une seule instruction *if*.

Opérateur	Symbole
AND	&&
OR	
NOT	!

Table 7. Opérateurs logiques en C++

Relation	Symbole
Égal à	==
Différent de	!=
Supérieur à	>
Supérieur ou égal à	>=
Inférieur à	<
Inférieur ou égal à	<=

Table 8. Opérateurs relationnels en C++

```
1 double x;
2 double y;
3 if ((x > 12) && (x < 14))
4 {
5     y = 10.0; // Les deux conditions sont remplies
6 }
```

```
1 double p, q;
2 int i;
3 double y;
4 if ((p > q) || (i != 1))
5 {
6     y = 10.0; // Au moins une des deux est vraie
7 }
8 else
9 {
10     y = -10.0; // Aucune des deux n'est vraie
11 }
```

### TP : Plage de valeurs

Écrire un programme qui demande à l'utilisateur un nombre. Indiquer à l'écran s'il est compris entre 10 et 20 inclus.

Essayer de le faire, une fois avec deux tests *if* imbriqués, puis avec une seule condition à l'aide des opérateurs logiques.

### TP - Combinaisons de conditions

- (a) Lire deux nombres et afficher le plus grand s'ils sont différents.
- (b) Si les deux sont positifs, afficher leur somme.

## 12. Les boucles

### 12.1 Boucle for

Utile quand on connaît le nombre d'itérations :

```
1 for (initialisation ; condition ; incrementation)
2 {
3     instructions
4 }
```

La partie en-tête contrôle l'exécution du corps de la boucle.

1. On exécute l'instruction **initialisation**
2. On teste la **condition**
  - si elle est vraie, on exécute **instructions**, puis l'instruction **incrementation** puis on revient au 2
  - si elle est fausse, on passe à la suite du programme (on sort de la boucle **for**).

```
1 for (int i = 0; i < 5; i++) {
2     cout << i << endl;
3 }
```

#### TP - Boucle for (affichage)

- (a) Afficher les entiers de 1 à 10 en utilisant une boucle **for**.
- (b) Afficher les entiers de 10 à 1 dans l'ordre décroissant à l'aide d'une boucle **for**.

### 12.2 Boucle while

Exécuter tant qu'une condition est vraie :

```
1 while (condition)
2 {
3     Instructions
4 }
```

1. On teste **condition**
  - si elle est vraie, on exécute **instructions** puis on recommence au 1
  - si elle est fausse, on passe à la suite du programme (on sort de la boucle **while**).

```
1 int i = 0;
2 while (i < 5) {
3     cout << i << endl;
4     i++;
5 }
```

#### TP - Boucle while : Saisie contrôlée

- (a) Écrire un programme qui demande à l'utilisateur de saisir un nombre **strictement positif** tant que la valeur saisie n'est pas correcte. Afficher le nombre accepté ensuite.

### 12.3 Boucle do ... while

Au moins une exécution, puis on teste :

```
1 do
2 {
3     Instructions
4 } while (condition);
```

L'instruction `do...while` exécute toujours la première itération.

1. On exécute **instructions**
2. On évalue **condition**
  - si elle est vraie, on recommence au 1
  - si elle est fausse, on passe à la suite du programme (on sort de la boucle `do...while`).

```
1 int val;
2 do {
3     cout << "Tapez 0 pour quitter.";
4     cin >> val;
5 } while (val != 0);
```

#### TP - do ... while

- (a) Écrire un programme qui demande à l'utilisateur de saisir un nombre entre 1 et 10 en utilisant une boucle `do ... while`. Le programme doit continuer à demander un nombre tant que l'utilisateur n'a pas saisi un nombre valide.
- (b) Écrire un programme qui lit des entiers jusqu'à ce que l'utilisateur entre un nombre négatif. Afficher la somme de tous les nombres positifs saisis.

#### TP : Menu interactif

- (a) Écrire un programme qui affiche un menu simple avec plusieurs choix (par exemple : 1- Calculer une somme, 2- Convertir une température, 0- Quitter). Le menu doit se répéter jusqu'à ce que l'utilisateur entre 0.

## 13. Les tableaux

Un **tableau** ("array") regroupe plusieurs éléments de même type qui sont accessibles par leur indice.

#### 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.

- `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

#### Note 13.1.

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.

```
1 int notes[5];           // déclare un tableau de 5 entiers
2 notes[0] = 15;         // affectation au premier élément
3 float nombre[5]; // 5 décimaux
```

Pour déclarer et initialiser :

```
1 int valeurs[3] = {10, 5, 7};
2 double tab[] = {1.2, 3.4, 2.1, 6.0}; // taille automatique
```

Accès à un élément :

```
1 cout << tab[2] << endl; // Affiche le 3e élément
```

#### Tableau à 2 dimensions (matrice) :

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

- `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

```
1 int y[2][3] = {{1,4,6},{3,7,5}}; // 2 lignes et 3 colonnes
```

### Exemple 13.1: Remplissage manuel et affichage d'une matrice

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int matrice[2][3]; // Déclaration d'une matrice 2x3
6
7     // Remplissage manuel
8     for (int i = 0; i < 2; i++) {
9         for (int j = 0; j < 3; j++) {
10             cout << "Entrez l'élément [" << i << "][" << j << "]: ";
11             cin >> matrice[i][j];
12         }
13     }
14
15     // Affichage
16     cout << "Matrice remplie :\n";
17     for (int i = 0; i < 2; i++) {
18         for (int j = 0; j < 3; j++) {
19             cout << matrice[i][j] << " ";
20         }
21         cout << endl;
22     }
23
24     return 0;
25 }
```

### TP : Manipulation de tableaux 1D

- (a) Écrire un programme qui demande à l'utilisateur de saisir 5 nombres dans un tableau. Afficher ensuite toutes les valeurs saisies sur une seule ligne séparées par des espaces.  
(Bonus) Afficher la somme des éléments du tableau.

### TP - Tableaux 1D

- (a) Lire 4 notes et calculer la moyenne.  
(b) Afficher la note la plus élevée.

### TP - Tableaux 2D

- (a) Demander à l'utilisateur de remplir une matrice 3x3.  
(b) Calculer la somme des éléments de chaque ligne.

## 14. Les fonctions

Une **fonction** permet de factoriser le code et d'améliorer la lisibilité et la réutilisation.

Déclaration :

```
1 type nom_de_la_fonction(arguments)
2 {
3     //Instructions
4 }
```

- **type** : type du résultat retourné (int, double, void, ...)
- **nom\_de\_la\_fonction** : permet de donner un nom à la fonction
- **arguments** : variables d'entrée (séparées par des virgules)

Exemple :

```
1 double carre(double x) {
2     return x * x;
3 }
4 // Appel à la fonction :
5 double val = carre(3.5); // Renvoie 12.25
```

### Paramètres et valeurs de retour

- Une fonction peut recevoir zéro, un ou plusieurs paramètres.
- Elle peut retourner une valeur, ou rien (void).

### Portée des variables

- Les variables déclarées dans une fonction sont dites **locales** (ne sont visibles que dans cette fonction).
- Une variable globale (déclarée hors de toute fonction) est évitable (déconseillée).

#### TP : Fonctions de base

- (a) Créer une fonction qui calcule le carré d'un entier.

#### TP : Fonctions personnalisées

- (a) Écrire une fonction qui calcule le cube d'un entier.  
(b) Réaliser un programme qui lit un entier et affiche son cube en appelant la fonction.

#### TP - Portée des variables

Écrire un programme avec une fonction qui utilise une variable locale nommée **val**. Ajouter une variable globale aussi nommée **val** et afficher leur valeur respective dans le **main** et dans la fonction.

## 15. Notions avancées

### Introduction aux pointeurs

Un **pointeur** est une variable qui stocke l'adresse mémoire d'une autre variable :

```
1 int x = 10;
2 int* ptr = &x;    // ptr contient l'adresse de x
3 cout << *ptr;    // affiche la valeur pointée, ici 10
```

Utilisation fréquente pour les tableaux, fonctions avancées ou optimisation mémoire.

### TP : Pointeurs

- (a) Écrire un programme qui lit un entier, puis affiche son double en utilisant un pointeur.
- (b) Afficher l'adresse mémoire d'une variable.

### Les références &

Permet de passer une variable par référence, donc de la modifier dans une fonction :

```
1 void incrementer(int& n) {
2     n = n + 1;
3 }
```

### TP : Passage par référence

- (a) Créer une fonction ajouter10() qui ajoute 10 à un entier passé par référence.

### Entrées/Sorties de fichiers

Lire/écrire dans un fichier texte nécessite #include <fstream> :

```
1 #include <fstream>
2 // Ecriture:
3 ofstream fichier("donnees.txt");
4 fichier << "Bonjour";
5 fichier.close();
6
7 // Lecture:
8 ifstream fichIn("donnees.txt");
9 string ligne;
10 getline(fichIn, ligne);
11 cout << ligne << endl;
```

### TP : Manipulation de fichiers

1. Écrire dans un fichier texte le nom et le prénom saisis par l'utilisateur.
2. Lire ce fichier texte et afficher son contenu à l'écran.

### TP - Fichiers

- (a) Écrire un programme qui crée un fichier et y écrit 5 lignes de texte.
- (b) Lire le fichier et afficher le contenu à l'écran.

## 16. Résumé

- Programme bien structuré : initialisation, variables, logique conditionnelle, boucles, fonctions.
- Les variables doivent être typées et initialisées avant usage.
- Utilisez fonctions pour clarifier, commenter votre code.
- Privilégiez la robustesse : Vérifiez les entrées utilisateur et prévoyez des cas d'erreur (division par zéro, mauvaise saisie, etc.).
- Approche progressive : commencez simple, puis ajoutez logique, tableaux, modularisation.