الجمهورية الجزائرية الجزائريـــة الديمقراطيـــة الشعبيـــة

République Algérienne Démocratique et Populaire

وزارة التعليـم العالـي والبحـث العلمـي

جـــامعة طـــاهري محمد بشار

نيابة ما بعد التدرج و

البحث العلمي و العلاقات الخارجية

كلية التكنولوجيا

Ministère de l'Enseignement Supérieur et

de la Recherche Scientifique

Université TAHRI Mohammed Bechar

Faculté de Technologie

Polycopié Pédagogique de Cours

Intitulé :

" Programmation en C++"

Code de la Matière : F522 **Niveau :** Licence

Filière : Automatique

Spécialité : Automatique

Etabli par l'enseignant(e) : Dr Lakhdari Lahcen

Année Universitaire : 2025 / 2026

Avant-Propos

Le langage C++ occupe une place centrale dans l'enseignement et la pratique de la programmation moderne. Héritier du langage C, il allie la puissance de la programmation procédurale à la flexibilité de la programmation orientée objet. Son utilisation s'étend aussi bien aux systèmes embarqués, aux applications scientifiques et industrielles, qu'au développement logiciel de grande envergure.

Cette polycopie a été conçue comme un support pédagogique destiné aux étudiants de licence et master en informatique, électronique, automatique et télécommunications. Elle propose une progression structurée allant des notions de base (syntaxe, variables, opérateurs) jusqu'aux concepts avancés (pointeurs, classes, héritage, polymorphisme).

L'objectif principal est de permettre à l'étudiant de :

- Comprendre les fondements théoriques du langage C++,
- Acquérir une solide maîtrise pratique à travers de nombreux exemples de code,
- S'exercer grâce à des séries d'exercices,
- Développer une autonomie dans la conception et l'implémentation de programmes informatiques.

Nous avons veillé à adopter une approche claire, progressive et illustrée, afin de faciliter la compréhension et l'assimilation des concepts. Ce document constitue ainsi une référence utile, aussi bien pour les étudiants débutants que pour ceux souhaitant consolider leurs acquis.

Nous espérons que ce travail contribuera à renforcer les compétences des apprenants et à susciter chez eux un intérêt durable pour la programmation en C++ et ses multiples applications dans le monde scientifique et industriel.

Table des matières

Chapitre 1 : Présentation du langage C++

1.1 Introduction générale

Le langage **C++** est un langage de programmation compilé, multi-paradigme, créé en 1983 par **Bjarne Stroustrup** comme une extension du langage C [1].

Il combine la **programmation procédurale** (héritée du C) et la **programmation orientée objet (POO)**, ce qui en fait un langage puissant et flexible [1].

Aujourd'hui, le C++ est largement utilisé dans :

- Les systèmes embarqués,
- Les applications scientifiques,
- Le développement de jeux vidéo,
- Les logiciels temps réel,
- Les systèmes d'exploitation,
- Les bibliothèques performantes (par ex. OpenCV, TensorFlow en partie, etc.).

1.2 Environnement de développement en C++ [1,2].

Pour écrire, compiler et exécuter un programme C++, on utilise :

- **Éditeur de code / IDE (Integrated Development Environment)**
  - Code: Blocks, Dev-C++, Visual Studio, CLion, ou encore Visual Studio Code.
- **Compilateur**
  - Le plus répandu est **g++** (GNU Compiler for C++).
- **Processus de développement** :

1. **Écriture du code source** dans un fichier .cpp.
2. **Compilation** → traduction du code en langage machine.
3. **Édition des liens (linking)** → association avec les bibliothèques nécessaires.
4. **Exécution** → lancement du programme binaire.

1.3 Structure générale d'un programme C++

Un programme C++ de base suit cette structure :

```cpp
#include <iostream>   // Inclusion d'une bibliothèque standard
using namespace std;  // Utilisation de l'espace de noms standard

int main() {
    // Corps du programme
    cout << "Bonjour le monde !" << endl; // Affichage d'un message
    return 0; // Fin du programme
}
```

Explications :

- #include <iostream> → permet d'utiliser cin (entrée) et cout (sortie).
- using namespace std; → évite d'écrire std::cout.
- int main () → fonction principale, point d'entrée du programme.
- return 0 ; → indique que le programme s'est terminé avec succès.

1.4 Compilation et exécution (exemple avec g++)

1. Sauvegarder le fichier sous programme.cpp.
2. Compiler :

1.5 Premier programme classique : "Hello World"

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

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

Ce programme affiche simplement "Hello, World !" à l'écran. Il est souvent utilisé comme **premier test** pour vérifier qu'un environnement de programmation est bien installé.

1.6 Caractéristiques principales [3,4] :

- Langage **compilé** (rapide et efficace).
- Support de la **programmation procédurale** (héritée de C).
- Support de la **programmation orientée objet** (classes, héritage, polymorphisme).
- Gestion **fine de la mémoire** (pointeurs, allocation dynamique).
- Large usage dans :
  - Systèmes embarqués,
  - Systèmes d'exploitation,
  - Jeux vidéo,
  - Applications scientifiques et industrielles.

1.7 Différences entre C et C++

Aspect	C	C++
Paradigme principal	Procédural	Procédural + Orienté Objet
Encapsulation	Non	Oui (classes, objets)
Surcharge fonctions	Non	Oui
Gestion mémoire	malloc/free	New/delete + constructeurs/destructeurs
Généricité	Non	Oui (templates)

1.8 Avantages du langage C++ [5]

- Langage **rapide et optimisé** (utilisé pour les applications critiques en performance).
- Permet la **programmation bas niveau** (gestion mémoire, pointeurs).
- Supporte la **programmation orientée objet** (classes, héritage, polymorphisme).
- Large communauté et nombreuses bibliothèques.

1.9 Inconvénients [5]

- Complexité élevée (comparé à Python ou Java).
- Gestion manuelle de la mémoire (risque de fuites mémoire).
- Courbe d'apprentissage plus difficile.

1.10 Résumé du chapitre

- C++ est une extension du langage C, créé par Bjarne Stroustrup.
- Il est compilé et combine la programmation procédurale et orientée objet.
- Le processus : écriture → compilation → exécution.
- La structure d'un programme commence toujours par la fonction main ().

1.11 Exercices d'application

Exercice 1

Écrire un programme C++ qui affiche : Bonjour, je découvre le langage C++ !

Exercice 2

Écrire un programme qui affiche votre nom, prénom et votre spécialité.

Exercice 3

Tester les instructions suivantes et observer la différence :

```
cout << "Bonjour";

cout << " le monde!";


cout << "Bonjour" << endl;

cout << "le monde!";
```

Chapitre 2 : Syntaxe élémentaire en langage C++

2.1 Introduction

La syntaxe en C++ définit les **règles d'écriture** du code source : les instructions, les variables, les types de données, et les conventions qui doivent être respectées pour que le compilateur comprenne le programme [6].

Une mauvaise syntaxe entraîne des **erreurs de compilation**.

2.2 Instructions et commentaires [6]

a) Instructions

- Une instruction en C++ se termine **toujours par un point-virgule (;)**.

  Exemple :

```
int x = 10;
cout << x << endl;
```

b) Commentaires

Les commentaires ne sont pas exécutés par le compilateur.

- Commentaire sur une ligne :

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

- Commentaire sur plusieurs lignes :

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

2.3 Mots-clés (keywords) [7]

Les **mots-clés** sont réservés au langage et ne peuvent pas être utilisés comme identificateurs (noms de variables).

Exemples : `int`, `float`, `if`, `else`, `for`, `while`, `class`, `return`, etc.

2.4 Constantes et variables [7]

a) Constantes

Une constante est une valeur fixe qui ne change pas pendant l'exécution.

```
const double PI = 3.14159;
```

b) Variables

Une variable est une **zone mémoire nommée** permettant de stocker une valeur.

Déclaration générale :

type nom_variable = valeur_initiale;

Exemple :

int age = 20;

float salaire = 4500.5;

char lettre = 'A';

2.5 Types fondamentaux [7]

- **int** : entiers (ex : 10, -25).
- **float** : réels à simple précision (ex : 3.14).
- **double** : réels à double précision (ex : 2.718281828).
- **char** : un caractère (ex : 'A').
- **bool** : booléen (true ou false).

Exemple :

```
int nombre = 5;
double pi = 3.14159;
char lettre = 'Z';
bool etat = true;
```

2.6 Opérateurs en C++ [8]

a) Opérateurs arithmétiques

- + addition
- - soustraction
- * multiplication
- / division
- % modulo (reste de la division entière)

Exemple :

- int a = 10, b = 3;
- cout << a + b << endl; // 13
- cout << a % b << endl; // 1

b) Opérateurs relationnels

- = égal à
- != différent de
- < plus petit
- > plus grand
- <= plus petit ou égal
- >= plus grand ou égal

c) Opérateurs logiques

- && (ET logique)
- || (OU logique)
- ! (NON logique)

Exemple :

- int x = 5, y = 10;
- cout << (x < y && y > 0) << endl; *// 1 (vrai)*

d) Priorité des opérateurs

L'ordre de priorité suit les règles mathématiques :

1. Parenthèses ( )
2. Multiplication / Division / Modulo
3. Addition / Soustraction
4. Opérateurs relationnels et logiques

2.7 Entrées et sorties simples [9]

- **Affichage avec cout :**

  cout << "Bonjour" << endl;

- **Lecture avec cin :**

  int age;
  cout << "Entrez votre âge : ";
  cin >> age;
  cout << "Vous avez " << age << " ans." << endl;

2.8 Exemple

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

int main() {
    int a, b;
    cout << "Entrez deux nombres : ";
    cin >> a >> b;

    cout << "Somme = " << a + b << endl;
    cout << "Produit = " << a * b << endl;
    cout << "Est-ce que a > b ? " << (a > b) << endl;

    return 0;
}
```

2.9 Résumé du chapitre

- Chaque instruction se termine par **;**.
- Les **commentaires** servent à documenter le code.
- Les variables permettent de stocker des données, les constantes sont fixes.
- Les opérateurs (arithmétiques, relationnels, logiques) permettent de manipuler les données.
- Les entrées/sorties (cin et cout) permettent l'interaction avec l'utilisateur.

2.10 Exercices d'application

Exercice 1

Déclarer trois variables entières a, b et c, donner des valeurs à a et b, puis calculer la somme et le produit dans c. Afficher les résultats.

Exercice 2

Écrire un programme qui demande à l'utilisateur :

- son nom,
- son âge,
- sa moyenne en C++.

Puis afficher un message comme :

Nom : Ali

Âge : 21

Moyenne en C++ : 15.5

Exercice 3

Tester le code suivant et expliquer le résultat :

```
int x = 7, y = 2;

cout << x / y << endl;

cout << (float)x / y << endl;
```

(Indice : division entière vs division réelle).

Chapitre 3 : Structures conditionnelles et boucles

3.1 Introduction

La programmation nécessite souvent de **prendre des décisions** (exécuter certaines instructions selon une condition) et de **répéter des instructions** (boucles).
En C++, cela se fait avec [10] :

- Les **structures conditionnelles** (if, else, switch),
- Les **boucles** (for, while, do/while).

3.2 Structures conditionnelles [10]

a) Instruction if

Permet d'exécuter une instruction si une condition est vraie.

Syntaxe :

```
if (condition) {
   // instructions exécutées si condition est vraie
}
```

Exemple

```
int age = 20;
if (age >= 18) {
   cout << "Vous êtes majeur." << endl;
}
```

b) Instruction if...else

Permet de choisir entre deux alternatives.

Syntaxe :

```
if (condition) {
   // instructions si condition est vraie
} else {
   // instructions si condition est fausse
```

```
}
```

Exemple

```
int age = 16;
if (age >= 18) {
    cout << "Majeur" << endl;
} else {
    cout << "Mineur" << endl;
}
```

c) Instruction if...else if...else

Permet de gérer plusieurs cas.

Syntaxe :

```
if (condition1) {
    // instructions si condition1 est vraie
} else if (condition2) {
    // instructions si condition2 est vraie
} else {
    // instructions si aucune condition n'est vraie
}
```

Exemple

```
int note = 15;
if (note >= 16) {
    cout << "Très bien" << endl;
} else if (note >= 12) {
    cout << "Assez bien" << endl;
} else if (note >= 10) {
    cout << "Passable" << endl;
```

```
} else {
    cout << "Échec" << endl;
}
```

d) Instruction switch

Alternative à if...else if lorsqu'on compare une variable à plusieurs valeurs entières ou caractères.

Syntaxe :

```
switch (variable) {
    case valeur1:
        // instructions si variable == valeur1
        break;
    case valeur2:
        // instructions si variable == valeur2
        break;
    ...
    default:
        // instructions si aucune valeur ne correspond
}
```

Exemple

```
int jour = 3;
switch (jour) {
    case 1: cout << "Lundi"; break;
    case 2: cout << "Mardi"; break;
    case 3: cout << "Mercredi"; break;
    case 4: cout << "Jeudi"; break;
    case 5: cout << "Vendredi"; break;
```

```
    case 6: cout << "Samedi"; break;
    case 7: cout << "Dimanche"; break;
    default: cout << "Jour invalide";
}
```

3.3 Boucles [10,11]

a) Boucle while

Exécute un bloc tant qu'une condition est vraie.

Syntaxe :

```
while (condition) {
    // instructions à exécuter tant que condition est vraie
}
```

Exemple

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

b) Boucle do...while

Exécute le bloc **au moins une fois**, puis répète tant que la condition est vraie.

Syntaxe :

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

Exemple

```
int i = 1;
do {
```

```
        cout << i << endl;
        i++;
    } while (i <= 5);
```

c) Boucle for

Utilisée lorsqu'on connaît à l'avance le nombre de répétitions.

Syntaxe :

```
for (initialisation  ; condition ; incrément)  {
   // instructions  à exécuter
}
```

Exemple

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

d) Boucles imbriquées

Une boucle peut être placée à l'intérieur d'une autre.

Syntaxe :

```
for (initialisation1  ; condition1  ; incrément1)  {
   for (initialisation2  ; condition2  ; incrément2)  {
      // instructions  exécutées
   }
}
```

Exemple : affichage d'un carré d'étoiles.

```
for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 5; j++) {
      cout << "* ";
   }
```

cout << endl;

}

3.4 Instructions de contrôle dans les boucles [11]

- break : interrompt la boucle.
- continue : saute l'itération courante et passe à la suivante.

Exemple :

```
for (int i = 1; i <= 5; i++) {
   if (i == 3) continue;  // saute le 3
   if (i == 5) break;     // arrête la boucle
   cout << i << endl;
}
```

3.5 Exemple : Somme des nombres pairs entre 1 et 10

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

int main() {
    int somme = 0;
    for (int i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            somme += i;
        }
    }
    cout << "Somme des pairs = " << somme << endl;
    return 0;
}
```

3.6 Résumé du chapitre

- if, else if, else, switch : permettent de prendre des décisions.
- while répète tant que la condition est vraie.
- do...while exécute au moins une fois.
- for est utilisé pour les répétitions connues.
- break arrête une boucle, continue passe à l'itération suivante.

3.7 Exercices d'application

Exercice 1

Écrire un programme qui demande à l'utilisateur son âge, puis affiche :

- "Mineur" si < 18,
- "Majeur" si ≥ 18.

Exercice 2

Écrire un programme qui affiche la table de multiplication d'un nombre entré par l'utilisateur (jusqu'à 10).

Exemple attendu pour 5 :

5 x 1 = 5

5 x 2 = 10

...

5 x 10 = 50

Exercice 3

Écrire un programme qui calcule la somme des entiers de 1 à N (N saisi par l'utilisateur) en utilisant une boucle for.

Exercice 4

Écrire un programme qui utilise deux boucles imbriquées pour afficher :

```
*
* *
* * *
* * * *
```

Chapitre 4 : Entrées / Sorties

4.1 Introduction

Les **entrées/sorties (E/S)** sont indispensables en programmation car elles permettent [12]:

- D'afficher des informations à l'écran (**sortie**),
- De lire des données saisies par l'utilisateur ou depuis un fichier (**entrée**).

En C++, les E/S standards sont gérées par la **bibliothèque <iostream>** avec les objets :

- cout : sortie standard (console),
- cin : entrée standard (clavier),
- cerr : affichage des erreurs,
- clog : messages d'information (log).

4.2 Sortie standard avec cout [12]

cout signifie **console output**.
L'opérateur << permet d'envoyer des données vers la console.

Exemple :

```
#include <iostream>

using namespace std;

int main() {

   cout << "Bonjour le monde !" << endl;

   cout << "Le résultat est : " << 5 + 3 << endl;

   return 0;

}
```

- **endl** permet de passer à la ligne.
- On peut enchaîner plusieurs sorties avec **<<**.

4.3 Entrée standard avec cin [12]

cin signifie **console input**.

L'opérateur >> permet de lire une valeur tapée au clavier.

Exemple :

```
#include <iostream>

using namespace std;

int main() {

    int age;

    cout << "Entrez votre âge : ";

    cin >> age;

    cout << "Vous avez " << age << " ans." << endl;

    return 0;

}
```

Attention :

- **cin** arrête la lecture dès qu'il rencontre un espace ou un retour à la ligne.
- Pour lire une chaîne avec espaces, on utilise getline().

4.4 Lecture de chaînes de caractères [12]

```
#include <iostream>

#include <string>

using namespace std;

int main() {

    string nom;

    cout << "Entrez votre nom complet : ";

    getline(cin, nom); // lit une ligne entière

    cout << "Bonjour " << nom << " !" << endl;

    return 0;

}
```

4.5 Flux d'entrée/sortie formatés [12]

On peut contrôler la mise en forme avec la bibliothèque <iomanip>.

Exemple d'affichage formaté :

```
#include <iostream>
#include <iomanip>  // pour setprecision et setw
using namespace std;
int main() {
   double pi = 3.14159265;
   cout << "Pi avec 2 décimales : " << fixed << setprecision(2) << pi << endl;
   cout << "Pi avec 5 décimales : " << fixed << setprecision(5) << pi << endl;
   return 0;
}
```

4.6 Gestion des fichiers en C++ [12]

Pour manipuler des fichiers, on utilise la bibliothèque <fstream>.

a) Écriture dans un fichier

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ofstream fichier("exemple.txt");  // ouverture en écriture
   fichier << "Bonjour, ceci est un test." << endl;
   fichier.close();
return 0;
}
```

b) Lecture depuis un fichier

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    ifstream fichier("exemple.txt");  // ouverture en lecture
    string ligne;
    while (getline(fichier, ligne)) {
     cout << ligne << endl;
    }
    fichier.close();
    return 0;
}
```

4.7 Exemple : Lecture/Écriture de notes

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

int main() {
    ofstream fout("notes.txt");
    int n;
    cout << "Combien de notes voulez-vous enregistrer ? ";
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int note;
        cout << "Note " << i << " : ";
        cin >> note;
        fout << note << endl;
    }
    fout.close();

    ifstream fin("notes.txt");
    cout << "Les notes enregistrées sont :" << endl;
    int note;
    while (fin >> note) {
        cout << note << " ";
    }
    fin.close();
    return 0;
}
```

4.8 Résumé du chapitre

- cout → sortie standard.
- cin → entrée standard.
- getline() → lecture de chaînes avec espaces.
- <iomanip> → contrôle du format d'affichage.
- <fstream> → lecture et écriture de fichiers avec ifstream (lecture), ofstream (écriture).

4.9 Exercices d'application

Exercice 1

Écrire un programme qui demande à l'utilisateur son nom et son âge, puis affiche :

Bonjour [nom], vous avez [âge] ans.

Exercice 2

Écrire un programme qui lit 3 nombres au clavier et affiche leur moyenne avec 2 décimales.

Exercice 3

Créer un fichier etudiants.txt contenant les noms de 5 étudiants (saisis au clavier), puis lire et afficher leur contenu.

Chapitre 5 : Pointeurs et Tableaux

5.1 Introduction

En C++, la gestion de la mémoire est essentielle.

- Les **pointeurs** permettent de manipuler directement des adresses mémoire.
- Les **tableaux** (ou arrays) permettent de stocker plusieurs valeurs du même type dans une structure indexée [13].

Ces notions sont fondamentales pour comprendre la programmation bas niveau, l'allocation dynamique et les structures de données [13].

5.2 Pointeurs

a) Définition

Un **pointeur** est une variable qui contient **l'adresse mémoire** d'une autre variable.

Syntaxe :

type *nom_pointeur;

Exemple :

```
int x = 10;
int *p = &x;   // p pointe vers l'adresse de x
cout << "Valeur de x = " << x << endl;
cout << "Adresse de x = " << &x << endl;
cout << "Valeur de p = " << p << endl;
cout << "Contenu de l'adresse pointée = " << *p << endl;
```

b) Opérateurs associés

- & : adresse d'une variable
- * : déréférencement (accès à la valeur pointée)

5.3 Pointeurs et fonctions

Un pointeur permet de **modifier une variable** passée en paramètre.

Exemple sans pointeur (passage par valeur) :

```
void increment(int  n) {
   n++;
}
```

N'affecte pas la variable d'origine.

Exemple avec pointeur (passage par adresse) :

```
void increment(int  *n) {
   (*n)++;
}
int main() {
   int a = 5;
   increment(&a);
   cout << a << endl;  // affiche 6
}
```

5.4 Tableaux

a) Définition

Un **tableau** est une collection de variables de même type, stockées de façon contiguë en mémoire.

Syntaxe :

type nom_tableau[taille];

Exemple :

```
int notes[5] = {10, 12, 14, 16, 18};
cout << notes[0] << endl; // premier élément
cout << notes[4] << endl; // cinquième élément
```

*Les indices commencent à **0** en C++.*

b) Parcours d'un tableau

```
int notes[5] = {10, 12, 14, 16, 18};

for (int i = 0; i < 5; i++) {

    cout << "Note[" << i << "] = " << notes[i] << endl;

}
```

c) Taille d'un tableau

```
int tab[10];

cout << "Taille = " << sizeof(tab) / sizeof(tab[0]) << endl;
```

5.5 Pointeurs et tableaux

Le nom d'un tableau est en réalité un **pointeur vers son premier élément**.

Exemple :

```
int tab[3] = {5, 10, 15};

int *p = tab;

cout << *p << endl;       // 5

cout << *(p + 1) << endl; // 10

cout << *(p + 2) << endl; // 15
```

5.6 Tableaux multidimensionnels

C++ permet de définir des tableaux à plusieurs dimensions (par ex. matrices).

Exemple :

```
int mat[2][3] = {

   {1, 2, 3},

   {4, 5, 6}

};

for (int i = 0; i < 2; i++) {

   for (int j = 0; j < 3; j++) {

      cout << mat[i][j] << " ";

   }
```

```
    cout << endl;
}
```

5.7 Allocation dynamique de mémoire

Avec new et delete, on peut réserver et libérer de la mémoire manuellement.

Exemple :

```
int *p = new int;   // allocation
*p = 20;
cout << *p << endl;
delete p;           // libération
```

Tableau dynamique :

```
int n;
cout << "Taille du tableau : ";
cin >> n;
int *tab = new int[n];
for (int i = 0; i < n; i++) {
    tab[i] = i + 1;
    cout << tab[i] << " ";
}
delete[] tab ; // libération du tableau
```

5.8 Exemple : Moyenne des notes

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

int main() {
    int n;
    cout << "Combien de notes ? ";
    cin >> n;

    int *notes = new int[n];  // allocation dynamique
    int somme = 0;

    for (int i = 0; i < n; i++) {
        cout << "Note " << i + 1 << " : ";
        cin >> notes[i];
        somme += notes[i];
    }

    cout << "Moyenne = " << (double)somme / n << endl;

    delete[] notes; // libérer la mémoire
    return 0;
}
```

5.9 Résumé du chapitre

- Les **pointeurs** stockent des adresses mémoire (&, *).
- Les pointeurs permettent de modifier des variables passées en paramètre.
- Les **tableaux** stockent plusieurs valeurs du même type.
- Le nom d'un tableau agit comme un pointeur vers son premier élément.
- C++ permet la gestion de tableaux multidimensionnels et l'allocation dynamique.

5.10 Exercices d'application

Exercice 1

Écrire un programme qui demande 5 entiers à l'utilisateur, les stocke dans un tableau, puis affiche leur somme et leur moyenne

Exercice 2

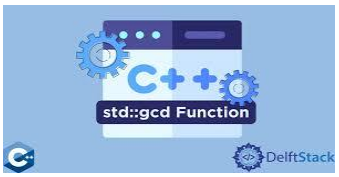
Écrire un programme qui utilise un pointeur pour échanger les valeurs de deux variables (permutation).

Exercice 3

Créer un tableau 2D (3x3) rempli par l'utilisateur et afficher la somme de ses éléments.

Exercice 4

Créer un tableau dynamique dont la taille est choisie par l'utilisateur, puis afficher le maximum et le minimum des valeurs saisies.



Chapitre 6 : Fonctions

6.1 Introduction

Une **fonction** est un bloc d'instructions regroupées sous un nom et exécutées lorsqu'on l'appelle [14].

Les fonctions permettent de :

- Éviter les répétitions de code,
- Structurer le programme en modules,
- Faciliter la maintenance et la réutilisation.

6.2 Définition d'une fonction [14]

Syntaxe générale :

```
type_retour nom_fonction(paramètres) {
    // instructions
    return valeur; // si type_retour ≠ void
}
```

- **type_retour** : le type de la valeur renvoyée (int, double, string, etc.).
- **nom_fonction** : identificateur choisi.
- **Paramètres** : variables reçues par la fonction (facultatif).

6.3 Exemple : fonction sans paramètre et avec retour

```
#include <iostream>

using namespace std;

int carre() {

    int x;

    cout << "Entrez un nombre : ";

    cin >> x;

    return x * x;

}

int main() {
```

```
    cout << "Carré = " << carre() << endl;
    return 0;
}
```

6.4 Fonction avec paramètres et retour

```
#include <iostream>
using namespace std;
int somme(int a, int b) {
    return a + b;
}
int main() {
    cout << "Somme = " << somme(3, 7) << endl;
    return 0;
}
```

6.5 Fonction sans retour (void)

Une fonction peut ne rien renvoyer :

```
void afficherMessage() {
    cout << "Bienvenue en C++ !" << endl;
}
```

Appel :

```
int main() {
    afficherMessage();
    return 0;
}
```

6.6 Passage de paramètres

a) Passage par valeur (copie des données)

```
void increment(int  n) {
   n++;
}
```

L'original n'est pas modifié.

b) Passage par adresse (avec pointeur)

```
void increment(int  *n) {
   (*n)++;
}
```

c) Passage par référence (avec &)

```
void increment(int  &n) {
   n++;
}
```

Exemple :

```
int main() {
   int x = 5;
   increment(x);
   cout << x << endl; // affiche 6
}
```

6.7 Surcharge de fonctions (function overloading)

Deux fonctions peuvent avoir le même nom si elles diffèrent par leurs paramètres.

```
int somme(int a, int b) {
   return a + b;
}
double somme(double a, double b) {
   return a + b;
}
```

```
int main() {
    cout << somme(2, 3) << endl;     // appelle la version int
    cout << somme(2.5, 3.7) << endl; // appelle la version double
}
```

6.8 Fonctions récursives

Une fonction est dite **récursive** si elle s'appelle elle-même.

Exemple : Factorielle

```
int factorielle(int n) {
    if (n == 0) return 1;
    else return n * factorielle(n - 1);
}
int main() {
    cout << "5! = " << factorielle(5) << endl; // 120
}
```

6.9 Organisation du code avec prototypes

Un programme peut être structuré en plaçant les **prototypes** (déclarations) avant main(), et les définitions après.

```
#include <iostream>
using namespace std;
int carre(int); // prototype
int main() {
    cout << carre(6) << endl;
    return 0;
}
int carre(int x) {
    return x * x;
}
```

6.10 Résumé du chapitre

- Une fonction est un bloc réutilisable de code.
- type_retour nom(paramètres) permet de définir une fonction.
- Les paramètres peuvent être passés **par valeur**, **par adresse** ou **par référence**.
- On peut définir plusieurs fonctions avec le même nom (**surcharge**).
- Les fonctions récursives s'appellent elles-mêmes.
- Les **prototypes** permettent d'organiser le code.

6.11 Exercices d'application

Exercice 1

Écrire une fonction qui calcule la somme de deux nombres et renvoie le résultat. L'appeler dans main().

Exercice 2

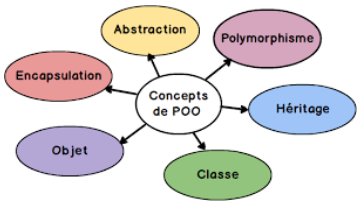
Écrire une fonction qui calcule le maximum de deux nombres passés en paramètres.

Exercice 3

Écrire une fonction récursive qui calcule la somme des entiers de 1 à N.

Exercice 4

Écrire un programme avec une fonction estPair(int n) qui renvoie true si le nombre est pair, false sinon. Tester dans main().



Chapitre 7 : Programmation Orientée Objet en C++

7.1 Introduction

La **programmation orientée objet (POO)** est un paradigme de programmation qui permet de représenter un problème sous forme d'**objets**.

Chaque objet combine [15] :

- Des données **(attributs)**
- Des fonctions **(méthodes)**

Avantages de la POO :

- Meilleure **modularité** du code.
- **Réutilisation** grâce à l'héritage.
- **Clarté et maintenance** des programmes.

7.2 Classe et Objet [15]

- **Définition d'une classe**

Une **classe** est un modèle qui définit la structure et le comportement d'objets.

```
class Etudiant {
public:
    string nom;
    int age;
    void afficher() {
        cout << "Nom: " << nom << ", Age: " << age << endl;
    }
};
```

- **Création d'un objet**

```
int main() {
    Etudiant e1 ; // objet de la classe
    e1.nom = "Ali";
    e1.age = 22;
    e1.afficher();
    return 0;
}
```

7.3 Constructeurs et Destructeurs [15]

- **Constructeur**

Un **constructeur** est une méthode spéciale exécutée lors de la création de l'objet.

```
class Etudiant {
public:
    string nom;
    int age;
    // Constructeur
    Etudiant(string n, int a) {
        nom = n;
        age = a;
    }
    void afficher() {
        cout << "Nom: " << nom << ", Age: " << age << endl;
    }
};
```

Utilisation :

```
int main() {
   Etudiant e1("Adam", 21);
   e1.afficher();
}
```

- **Destructeur**

Le **destructeur** est appelé automatiquement à la destruction de l'objet.

```
~Etudiant() {
   cout << "Objet détruit !" << endl;
}
```

7.4 Encapsulation et Modificateurs d'accès [16]

- public : accessible partout.
- private : accessible uniquement à l'intérieur de la classe.
- protected : accessible dans la classe et les classes dérivées.

Exemple :

```
class CompteBancaire {
private:
   double solde;
public:
   CompteBancaire(double s) { solde = s; }
   void depot(double montant) { solde += montant; }
   void retrait(double montant) { solde -= montant; }
   double getSolde() { return solde; }
};
```

7.5 Héritage [16]

- **Définition**

L'**héritage** permet de créer une nouvelle classe à partir d'une classe existante.

```
class Personne {

public:
   string nom;
   int age;

   void afficher() {

      cout << nom << " (" << age << " ans)" << endl;

   }

};

// Classe dérivée

class Etudiant : public Personne {

public:
   string filiere;
   void afficher() {

      cout << nom << " - " << filiere << endl;

   }

};
```

7.6 Polymorphisme [16]

- **Redéfinition (overriding)**

Une classe dérivée peut redéfinir une méthode de la classe de base.

```
class Animal {

public:
   virtual void parler() {
      cout << "L'animal fait un bruit." << endl;
   }
};
```

```
class Chien : public Animal {
public:
    void parler() override {
        cout << "Wouf !" << endl;
    }
};
int main() {
    Animal* a;
    Chien c;
    a = &c;
    a->parler(); // "Wouf !"
}
```

- **Polymorphisme statique (surcharge)**

  Déjà vu dans les **fonctions** (même nom mais paramètres différents

7.7 Résumé du chapitre

- La POO repose sur les concepts de **classe** et **objet**.
- Les **constructeurs** initialisent les objets, les **destructeurs** les libèrent.
- L'**encapsulation** protège les données.
- L'**héritage** permet de réutiliser le code.
- Le **polymorphisme** rend le code plus flexible.

7.8 Exercices d'application

Exercice 1

Créer une classe Rectangle avec deux attributs (longueur, largeur).

- Ajouter une méthode pour calculer l'aire.
- Tester dans main().

Exercice 2

Créer une classe Vehicule avec une méthode afficher().
Créer deux classes dérivées Voiture et Moto qui redéfinissent afficher().

Exercice 3

Créer une classe CompteBancaire avec dépôt/retrait/consultation.
Créer une classe CompteEpargne qui hérite de CompteBancaire et ajoute un taux d'intérêt.

Solutions des Exercices d'Application

Chapitre 1 : Introduction au langage C++

Exercice 1 :

Énoncé : Afficher le message *« Bienvenue au cours de C++ ! »*

Solution :

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

int main() {
    cout << "Bienvenue au cours de C++ !" << endl;
    return 0;
}
```

Exercice 2 :

Énoncé : Afficher votre nom et votre âge sur deux lignes.

Solution :

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

int main() {
    cout << "Nom : Ali" << endl;
    cout << "Age : 21" << endl;
    return 0;
}
```

Exercice 3

Énoncé : Tester les instructions suivantes et observer la différence.

Solution :

```cpp
cout << "Bonjour";
cout << " le monde!";

cout << "Bonjour" << endl;
cout << "le monde!";
```

Premier cas :

```cpp
cout << "Bonjour";
cout << " le monde!";
```

➢ Ici, les deux chaînes sont affichées l'une après l'autre sans retour à la ligne.

Affichage obtenu :

```
Bonjour le monde!
```

Deuxième cas :

```cpp
cout << "Bonjour" << endl;
cout << "le monde!";
```

➢ L'instruction endl provoque un retour à la ligne après « Bonjour ».

Affichage obtenu :

```
Bonjour
le monde!
```

Chapitre 2 : Syntaxe élémentaire en C++

Exercice 1 :

Énoncé : Déclarer trois variables entières a, b et c, donner des valeurs à a et b, puis calculer la somme et le produit dans c. Afficher les résultats.

Solution :

cpp

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

int main() {
    int a, b, c;

    // Affectation des valeurs
    a = 5;
    b = 3;

    // Calcul de la somme
    c = a + b;
    cout << "La somme de a et b est : " << c << endl;

    // Calcul du produit
    c = a * b;
    cout << "Le produit de a et b est : " << c << endl;

    return 0;
}
```

Exercice 2 :

Énoncé : Écrire un programme qui demande à l'utilisateur :

- son nom
- son âge
- sa moyenne en C++

  Puis afficher un message formaté.

```cpp
#include <iostream>
#include <string>   // pour utiliser string
using namespace std;

int main() {
    string nom;
    int age;
    float moyenne;

    cout << "Entrez votre nom : ";
    cin >> nom;

    cout << "Entrez votre âge : ";
    cin >> age;

    cout << "Entrez votre moyenne en C++ : ";
    cin >> moyenne;

    cout << "\nNom : " << nom << endl;
    cout << "Âge : " << age << endl;
    cout << "Moyenne en C++ : " << moyenne << endl;

    return 0;
}
```

Exercice 3 :

Énoncé : Tester le code suivant et expliquer le résultat.

```cpp
int x = 7, y = 2;
cout << x / y << endl;
cout << (float)x / y << endl;
```

Solution :

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

int main() {
    int x = 7, y = 2;

    cout << x / y << endl;          // Division entière
    cout << (float)x / y << endl;   // Division réelle

    return 0;
}
```

Résultat obtenu :

```
3
3.5
```

Explication :

- **x / y :** comme x et y sont des entiers, la division est entière. Le reste est ignoré.
  7 / 2 = 3 (et non 3.5).
- **(float)x / y :** ici x est converti en réel (float), donc la division devient réelle.
  7.0 / 2 = 3.5.

Chapitre 3 : Structures conditionnelles et boucles

Exercice 1 :

Énoncé : Écrire un programme qui demande à l'utilisateur son âge, puis affiche :

- « Mineur » si âge < 18
- « Majeur » si âge ≥ 18

Solution :

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

int main() {
    int age;

    cout << "Entrez votre âge : ";
    cin >> age;

    if (age < 18) {
        cout << "Mineur" << endl;
    } else {
        cout << "Majeur" << endl;
    }

    return 0;
}
```

Exercice 2 :

Énoncé : Écrire un programme qui affiche la table de multiplication d'un nombre entré par l'utilisateur (jusqu'à 10).

Solution :

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

int main() {
    int nombre;

    cout << "Entrez un nombre : ";
    cin >> nombre;

    for (int i = 1; i <= 10; i++) {
        cout << nombre << " x " << i << " = " << nombre * i << endl;
    }

    return 0;
}
```

Exercice 3 :

Énoncé : Écrire un programme qui calcule la somme des entiers de 1 à N (N saisi par l'utilisateur) en utilisant une boucle for.

Solution :

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

int main() {
    int N, somme = 0;

    cout << "Entrez une valeur N : ";
    cin >> N;

    for (int i = 1; i <= N; i++) {
        somme += i; // équivaut à somme = somme + i
    }

    cout << "La somme des entiers de 1 à " << N << " est : " << somme << endl;

    return 0;
}
```

Exercice 4 :

Énoncé : Écrire un programme qui utilise deux boucles imbriquées pour afficher :

```
*
* *
* * *
* * * *
* * * * *
```

Solution :

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

int main() {
    int n = 5; // nombre de lignes

    for (int i = 1; i <= n; i++) {          // boucle externe : lignes
        for (int j = 1; j <= i; j++) {      // boucle interne : étoiles
            cout << "* ";
        }
        cout << endl;
    }

    return 0;
}
```

Chapitre 4 : Entrées / Sorties

Exercice 1 :

Énoncé : Écrire un programme qui demande à l'utilisateur son nom et son âge, puis affiche :

```
Bonjour [nom], vous avez [âge] ans.
```

Solution :

```cpp
#include <iostream>
#include <string>   // Pour utiliser string
using namespace std;

int main() {
    string nom;
    int age;

    cout << "Entrez votre nom : ";
    cin >> nom;

    cout << "Entrez votre âge : ";
    cin >> age;

    cout << "Bonjour " << nom << ", vous avez " << age << " ans." << endl;

    return 0;
}
```

Exercice 2 :

Énoncé : Écrire un programme qui lit 3 nombres au clavier et affiche leur moyenne avec 2 décimales.

Solution :

```cpp
#include <iostream>
#include <iomanip>  // Pour fixer le nombre de décimales
using namespace std;

int main() {
    float a, b, c, moyenne;

    cout << "Entrez trois nombres : ";
    cin >> a >> b >> c;

    moyenne = (a + b + c) / 3;

    cout << fixed << setprecision(2);
    cout << "La moyenne est : " << moyenne << endl;

    return 0;
}
```

Exercice 3

Énoncé : Créer un fichier etudiants.txt contenant les noms de 5 étudiants (saisis au clavier), puis lire et afficher leur contenu.

Solution :

```cpp
#include <iostream>
#include <fstream>   // Pour la gestion des fichiers
#include <string>
using namespace std;

int main() {
    ofstream fichierEcriture("etudiants.txt");  // ouverture en écriture

    if (!fichierEcriture) {
        cerr << "Erreur d'ouverture du fichier en écriture." << endl;
        return 1;
    }

    string nom;
    cout << "Saisissez les noms de 5 étudiants :" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "Nom " << i + 1 << " : ";
        cin >> nom;
        fichierEcriture << nom << endl;
    }
    fichierEcriture.close();

    // Lecture du fichier
    ifstream fichierLecture("etudiants.txt");
    if (!fichierLecture) {
        cerr << "Erreur d'ouverture du fichier en lecture." << endl;
        return 1;
    }

    cout << "\nContenu du fichier etudiants.txt :" << endl;
    while (getline(fichierLecture, nom)) {
        cout << nom << endl;
    }

    fichierLecture.close();
    return 0;
}
```

Chapitre 5 : Pointeurs et Tableaux

Exercice 1 :

Énoncé : Écrire un programme qui demande 5 entiers à l'utilisateur, les stocke dans un tableau, puis affiche leur somme et leur moyenne.

Solution :

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

int main() {
    int tab[5];
    int somme = 0;
    float moyenne;

    cout << "Entrez 5 entiers :" << endl;
    for (int i = 0; i < 5; i++) {
        cout << "Nombre " << i + 1 << " : ";
        cin >> tab[i];
        somme += tab[i];
    }

    moyenne = (float)somme / 5;

    cout << "\nLa somme des éléments est : " << somme << endl;
    cout << "La moyenne des éléments est : " << moyenne << endl;

    return 0;
}
```

Exercice 2 :

Énoncé : Écrire un programme qui utilise un pointeur pour échanger les valeurs de deux variables (permutation).

Solution :

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

int main() {
    int a, b, temp;
    int *p1, *p2;

    cout << "Entrez la valeur de a : ";
    cin >> a;
    cout << "Entrez la valeur de b : ";
    cin >> b;

    p1 = &a;
    p2 = &b;

    // permutation avec pointeurs
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;

    cout << "\nAprès permutation :" << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;

    return 0;
}
```

Exercice 3 :

Énoncé : Créer un tableau 2D (3x3) rempli par l'utilisateur et afficher la somme de ses éléments.

Solution :

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

int main() {
    int tab[3][3];
    int somme = 0;

    cout << "Saisissez les éléments du tableau 3x3 :" << endl;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            cout << "Element [" << i << "][" << j << "] : ";
            cin >> tab[i][j];
            somme += tab[i][j];
        }
    }

    cout << "\nLa somme des éléments est : " << somme << endl;

    return 0;
}
```

Exercice 4 :

Énoncé : Créer un tableau dynamique dont la taille est choisie par l'utilisateur, puis afficher le maximum et le minimum des valeurs saisies.

Solution :

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

int main() {
    int n;
    cout << "Entrez la taille du tableau : ";
    cin >> n;

    // Allocation dynamique
    int* tab = new int[n];

    cout << "Saisissez " << n << " valeurs :" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Valeur " << i + 1 << " : ";
        cin >> tab[i];
    }

    int max = tab[0];
    int min = tab[0];

    for (int i = 1; i < n; i++) {
        if (tab[i] > max) max = tab[i];
        if (tab[i] < min) min = tab[i];
    }
    cout << "\nLe maximum est : " << max << endl;
    cout << "Le minimum est : " << min << endl;

    delete[] tab; // Libération de la mémoire dynamique
    return 0;
}
```

Chapitre 6 : Fonctions

Exercice 1 :

Énoncé : Écrire une fonction qui calcule la somme de deux nombres et renvoie le résultat. L'appeler dans main ().

Solution :

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

// Définition de la fonction
int somme(int a, int b) {
    return a + b;
}

int main() {
    int x, y;
    cout << "Entrez deux nombres : ";
    cin >> x >> y;

    int resultat = somme(x, y);

    cout << "La somme de " << x << " et " << y << " est : " << resultat << endl;

    return 0;
}
```

Exercice 2 :

Énoncé : Écrire une fonction qui calcule le maximum de deux nombres passés en paramètres.

Solution :

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

// Fonction pour trouver le maximum
int maximum(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}

int main() {
    int x, y;
    cout << "Entrez deux nombres : ";
    cin >> x >> y;

    int maxVal = maximum(x, y);

    cout << "Le maximum entre " << x << " et " << y << " est : " << maxVal << endl;

    return 0;
}
```

Exercice 3 :

Énoncé : Écrire une fonction récursive qui calcule la somme des entiers de 1 à N.

Solution :

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

// Fonction récursive
int sommeRecursive(int n) {
    if (n == 0)  // cas de base
        return 0;
    else
        return n + sommeRecursive(n - 1);  // appel récursif
}

int main() {
    int N;
    cout << "Entrez un entier N : ";
    cin >> N;

    int resultat = sommeRecursive(N);

    cout << "La somme des entiers de 1 à " << N << " est : " << resultat << endl;

    return 0;
}
```

Exercice 4 :

Énoncé : Écrire un programme avec une fonction estPair(int n) qui renvoie true si le nombre est pair, false sinon. Tester dans main().

Solution :

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

// Fonction qui vérifie si le nombre est pair
bool estPair(int n) {
    return (n % 2 == 0);
}

int main() {
    int x;
    cout << "Entrez un nombre : ";
    cin >> x;

    if (estPair(x))
        cout << x << " est pair." << endl;
    else
        cout << x << " est impair." << endl;

    return 0;
}
```

Chapitre 7 : Programmation Orientée Objet en C++

Exercice 1 :

Énoncé : Créer une classe Rectangle avec deux attributs (longueur, largeur).

- Ajouter une méthode pour calculer l'aire.
- Tester dans main().

Solution :

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

// Définition de la classe Rectangle
class Rectangle {
private:
    float longueur;
    float largeur;

public:
    // Méthode pour saisir les valeurs
    void saisir() {
        cout << "Entrez la longueur : ";
        cin >> longueur;
        cout << "Entrez la largeur : ";
        cin >> largeur;
    }

     // Méthode pour afficher les dimensions et l'aire
     void afficher() {
         cout << "Longueur : " << longueur << ", Largeur : " << largeur << endl;
         cout << "Aire = " << aire() << endl;
     }
 };

 int main() {
     Rectangle r;
     r.saisir();
     r.afficher();

     return 0;
 }
```

Exercice 2 :

Énoncé : Créer une classe Vehicule avec une méthode afficher().
Créer deux classes dérivées Voiture et Moto qui redéfinissent afficher().

Solution :

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

// Classe de base
class Vehicule {
public:
    virtual void afficher() {
        cout << "Je suis un véhicule." << endl;
    }
};

// Classe dérivée Voiture
class Voiture : public Vehicule {
public:
    void afficher() override {
        cout << "Je suis une voiture." << endl;
    }
};

// Classe dérivée Moto
class Moto : public Vehicule {
public:
    void afficher() override {
        cout << "Je suis une moto." << endl;
    }
};

int main() {
    Vehicule v;
    Voiture car;
    Moto bike;

    v.afficher();
    car.afficher();
    bike.afficher();

    // Utilisation du polymorphisme
    Vehicule* ptr;
    ptr = &car;
    ptr->afficher();
```

```
    ptr = &bike;
    ptr->afficher();

    return 0;
}
```

Exercice 3 :

Énoncé : Créer une classe CompteBancaire avec dépôt, retrait, consultation.
Créer une classe CompteEpargne qui hérite de CompteBancaire et ajoute un taux d'intérêt.

Solution :

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

// Classe de base
class CompteBancaire {
protected:
    double solde;

public:
    CompteBancaire(double s = 0) {
        solde = s;
    }

    void deposer(double montant) {
        solde += montant;
        cout << "Dépôt de " << montant << " effectué." << endl;
    }
    void retirer(double montant) {
        if (montant <= solde) {
            solde -= montant;
            cout << "Retrait de " << montant << " effectué." << endl;
        } else {
            cout << "Solde insuffisant." << endl;
        }
    }

    void consulter() {
        cout << "Solde actuel : " << solde << " DA" << endl;
    }
};

// Classe dérivée
class CompteEpargne : public CompteBancaire {
private:
    double tauxInteret; // en pourcentage
```

```
public:
    CompteEpargne(double s = 0, double taux = 0) : CompteBancaire(s) {
        tauxInteret = taux;
    }

    void ajouterInteret() {
        double interet = solde * (tauxInteret / 100);
        solde += interet;
        cout << "Intérêt ajouté : " << interet << " DA" << endl;
    }

    void afficherInfos() {
        consulter();
        cout << "Taux d'intérêt : " << tauxInteret << " %" << endl;
    }
};
```

```
int main() {
    CompteEpargne compte(1000, 5.0);  // 1000 DA avec 5% d'intérêt

    compte.consulter();
    compte.deposer(500);
    compte.retirer(200);
    compte.ajouterInteret();
    compte.afficherInfos();

    return 0;
}
```

Bibliographiques

- **Ouvrages**

1. Stroustrup, B. (2013). The C++ Programming Language (4e édition). Addison-Wesley.
2. Lippman, S. B., Lajoie, J., & Moo, B. E. (2012). C++ Primer (5e édition). Addison-Wesley.
3. Meyers, S. (2014). Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. O'Reilly Media.
4. Josuttis, N. M. (2012). The C++ Standard Library: A Tutorial and Reference (2e édition). Addison-Wesley..
5. Williams, A. (2019). C++ Concurrency in Action: Practical Multithreading (2e édition). Manning Publications.
6. Horton, I. (2020). Beginning C++17: From Novice to Professional (5e édition). Apress.
7. Griffiths, D. (2022). Head First C++: A Learner's Guide to Real-World Programming with C++ and STL. O'Reilly.
8. Balagurusamy, E. (2017). Object Oriented Programming with C++ (8e édition). McGraw Hill.

- **Articles et ressources en ligne**

9. ISO/IEC (2020). Programming Languages C++ 20 Standard. International Organization for Standardization.
10. ISO/IEC (2023). Working Draft: C++ 23 Standard. ISO C++ Committee.
11. Cppreference.com (consulté en 2025). C++ Reference Documentation.
12. StackOverflow Developer Survey (2023). Most Popular Programming Languages. Stack Overflow.

- **Ressources complémentaires**

13. www.cplusplus.com – Documentation pédagogique sur la syntaxe et la bibliothèque standard.
14. https://isocpp.org – Site officiel du comité de normalisation C++.
15. https://www.learncpp.com – Tutoriels progressifs et pratiques.
16. LearnCpp.com — tutoriels progressifs et pratiques sur la POO en C++