UNIVERSITÉ 8 MAI 1945 - GUELMA

FACULTÉ DES MATHÉMATIQUES, DE L'INFORMATIQUE ET DES SCIENCES DE LA MATIÈRE (MISM)

DÉPARTEMENT D'INFORMATIQUE

INITIATION À LA PROGRAMMATION

EN C++

POLYCOPIÉ DE COURS

MODULE : INFORMATIQUE

NIVEAU : 3ÈME ANNÉE LICENCE - HYDROGÉOLOGIE

DÉPARTEMENT DE SCIENCE DE LA NATURE ET DE LA VIE

FACULTÉ DES SCIENCES DE LA NATURE ET DE LA VIE ET SCIENCES DE LA TERRE ET DE L'UNIVERS (S.N.V.S.T.U)

DR. FAREK LAZHAR

- 2022 -

Table des matières

Liste des figures

Liste des tableaux

Préface

La programmation est devenue une partie intégrante de multiples disciplines et domaines de recherche pour résoudre de nombreux problèmes auxquels les gens sont confrontés dans leur vie quotidienne en effectuant des tâches avec une grande précision et à faible coût.

Les langages de programmation les plus puissants et les plus utilisés sont C, C++, Python, Java, etc. Certains ont leur propre domaine d'application potentiel, d'autres sont multi-domaines. C++ est un langage à usage général et l'un des langages de programmation célèbres qui a conservé sa place parmi les meilleurs langages de programmation malgré l'avènement de langages plus récents, plus simples et plus avancés. Il a été créé par Bjarne Stroustrup aux Bell Labs vers 1980. Sa syntaxe est très similaire au C (inventé par Dennis Ritchie au début des années 1970), mais il est basé sur la programmation orientée objet, il est donc mieux structuré et plus puissant que le C. C++ est un langage multiplateforme et de haute performance qui offre aux programmeurs un degré élevé de contrôle sur les ressources système et une gestion intelligente de la mémoire. Le langage a subi plusieurs mises à jour majeures, notamment en 2011 (C++11), 2014 (C++14), 2017 (C++17) et 2020 (C++20).

Ce support de cours est destiné aux étudiants de 3e année licence d'hydrogéologie avec un volume horaire de 1h30 par semaine. A l'aide de nombreux exemples illustratifs, les bases du langage C++ sont expliquées de manière simple afin que les étudiants puissent compléter et approfondir d'autres notions avancées telles que la programmation orientée objet (POO). Chaque chapitre est axé sur la pratique, avec des exemples de programmes dans chaque section et soutenu par une série d'exercices, dont les solutions sont fournies dans l'annexe de ce support. Le document est divisé en cinq chapitres comme suit:

1. Concepts de base
2. Les structures conditionnelles et répétitives
3. Les fonctions
4. Les tableaux et les chaînes de caractères
5. Les fichiers

CHAPITRE

1

Concepts de base

1.1 Introduction

Ce chapitre fournit une vue d'ensemble du langage de programmation C++. Il montre d'abord comment les programmes C++ sont compilés et exécutés dans un environnement de développement intégré (IDE) et utilise des exemples concrets pour illustrer la syntaxe générale du langage. Il aborde ensuite les concepts de base pour l'écriture et l'exécution de programmes en C++, y compris l'utilisation de variables, d'opérateurs, de commentaires et de types de données simples tels que les nombres, les caractères et les booléens.

1.2 Définition

C++ est un langage de programmation populaire et à haute performance, développé par Bjarne Stroustrup à partir de 1979 aux Bell Labs comme extension du langage C. C'est un langage multiplateforme c'est-à-dire il s'exécute sur diverses plates-formes, telles que Windows, Linux, Mac OS et UNIX, et donne aux programmeurs un haut niveau de contrôle pour la bonne gestion des ressources systèmes et matériels.

1.3 Domaines d'application de C++

C++ est un langage polyvalent qui peut être utilisé dans presque tous les domaines du développement des logiciels tels que:

- **Les logiciels d'application**: C++ est utilisé pour développer des systèmes d'exploitation tels que Windows, Mac OSX et Linux, de nombreux navigateurs tels que Mozilla Firefox, Google Chrome, et des systèmes de gestion de bases de données tels que MySQL.
- **Développement de langages de programmation**: C++ entre dans le développement d'autres langages de programmation tels que C#, Java, JavaScript, Python, etc.

- **Développement de jeux**: C++ est considéré comme le meilleur choix pour développer les jeux et les applications 3D car il offre un meilleur contrôle sur la gestion du matériel et de la mémoire.
- **Systèmes embarqués**: C++ est largement utilisé pour développer des applications médicales et techniques telles que des logiciels pour les appareils IRM[1], des systèmes CAD/CAM[2] haut de gamme, etc.

1.4 Environnement de développement

Pour démarrer avec C++, vous avez besoin de deux composants principaux: Un éditeur de texte pour écrire du code C++ et un compilateur pour traduire le code C++ dans un langage que l'ordinateur comprend. Cependant, vous pouvez simplifier et accélérer le processus de développement en consolidant tous les outils dont vous avez besoin pour le développement dans un environnement unique appelé environnement de développement intégré (en: Integrated Development Environment—IDE). Ceci est expliqué ci-dessous:

1.4.1 Compilateur

Un compilateur C++ lui-même est un programme informatique qui convertit le code source C++ (un programme) en une forme exécutable que l'ordinateur peut comprendre. Après avoir effectué des opérations de prétraitement sur le code source, le compilateur génère un nouveau code (un fichier) appelé "code objet". L'éditeur de liens combine ensuite le code objet avec des bibliothèques prédéfinies pour produire le fichier complet final qui peut s'exécuter sur votre ordinateur. Une bibliothèque est une collection de "code objet" compilé qui fournit des opérations exécutées de manière répétée par de nombreux programmes informatiques.

La figure 1.1 montre les différentes opérations effectuées sur le code source pour produire du code exécutable par la machine.

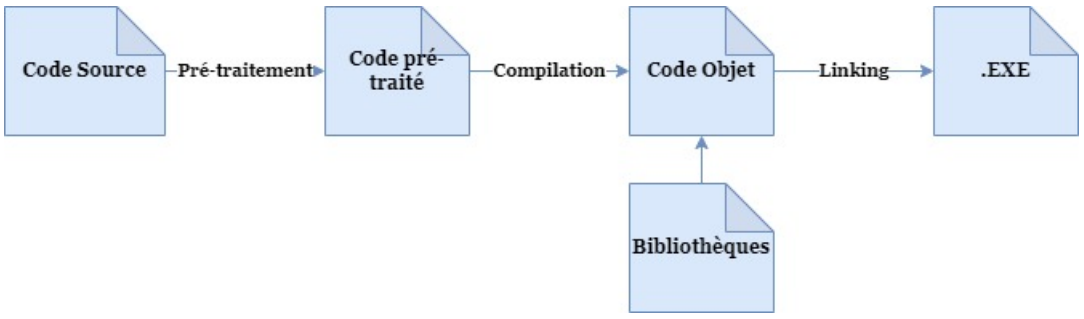


Figure 1.1: Compilation et exécution d'un programme C++

Il existe de nombreux compilateurs C++ qui peuvent être librement téléchargés et utilisés pour compiler et exécuter des programmes C++, notamment Clang C++, Cygwin (GNU C++), IBM C++, et Intel C++.

[1]Imagerie par Résonance Magnétique
[2]Computer-Aided Design/Computer-Aided Manufacturing

1.4.2 Environnement de développement intégré

Dans la section précédente, nous avons examiné les étapes pour convertir le code source C++ en un fichier exécutable. Mais avec un environnement de développement intégré (IDE), vous pouvez effectuer toutes ces étapes en une seule action. Un IDE est un ensemble d'outils regroupés dans un environnement: Un éditeur de texte, un compilateur, un débogueur et d'autres utilitaires intégrés dans une seule interface graphique (en. GUI: Graphical User Interface).

Il existe des IDE gratuits et des IDE commerciaux. Parmi les IDE les plus populaires, nous trouvons CodeBlocks[3](Code::Blocks), c'est un IDE gratuit conçu spécifiquement pour les développeurs C et C++, et doté de fonctionnalités pour faciliter le développement.

La figure 1.2 montre une vue d'ensemble de l'IDE CodeBlocks.

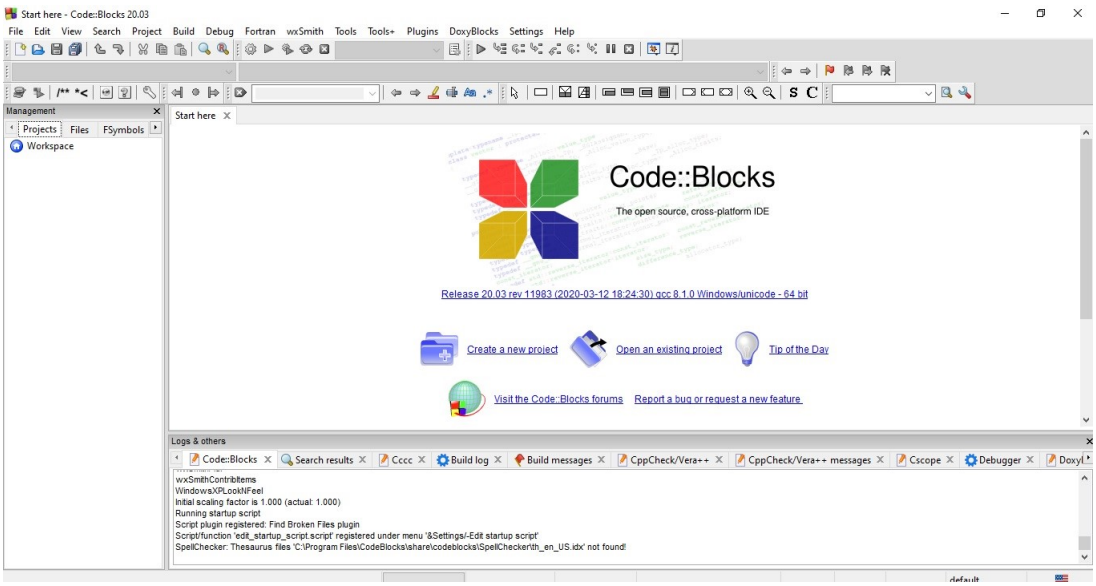


Figure 1.2: Aperçu général de CodeBlocks

CodeBlocks est multiplateforme, entièrement configurable et extensible grâce à de nombreux plugins, il fonctionne donc sur Windows, Linux et Mac OS X. Il dispose également de fonctionnalités avancées telles que l'auto-complétion du code et la visualisation des classes pour la programmation orientée objet.

1.5 Votre premier programme en C++

Comme dans tous les livres de la programmation, la meilleure façon pour commencer à apprendre la syntaxe du langage de programmation est d'écrire un programme.

Dans cette section, nous allons découvrir la structure générale d'un programme C++ à travers un programme illustratif simple qui affiche le fameux message 'Hello World !' sur l'écran.

Le code est illustré comme suit:

[3]Le package d'installation et le manuel d'utilisation de CodeBlocks sont disponibles sur le site officiel https://www.codeblocks.org/

```cpp
// Ceci est mon premier programme C++

/* Ce programme simple illustre divers composants
d'un programme écrit en C++ */

#include <iostream>
using namespace std;

int main()
{
cout << "Hello World!";

return 0;
}
```

Après avoir compilé et exécuté le programme ci-dessus, le message suivant apparaît à l'écran:

```
Hello World!
```

Les différentes composantes de ce programme sont décrites ci-dessous.

1.5.1 Les commentaires

Les trois premières lignes du programme ci-dessus sont des commentaires et seront ignorées par le compilateur. Des commentaires sont inclus dans le programme pour en faciliter la lecture. Si le commentaire est court et tient sur une seule ligne, commencez-le par une double barre oblique (//). Cependant, si le commentaire s'étend sur plusieurs lignes, placez-le entre les deux symboles /* et */.

1.5.2 Les directives

Les lignes commençant par le symbole # dans le programme ci-dessus sont appelées directives, qui sont des instructions pour le compilateur.

```cpp
#include <iostream>
```

Le mot include avec '#' indique au compilateur d'inclure le fichier iostream dans le fichier du programme ci-dessus. Le fichier iostream est un fichier d'en-tête requis pour les exigences d'entrée/sortie de votre programme. Ce fichier a donc été inséré au début du programme.

1.5.3 La bibliothèque standard

Tous les éléments de la bibliothèque C++ standard sont déclarés dans std en utilisant la déclaration suivante:

```cpp
using namespace std;
```

Cette ligne est très courante dans les programmes C++ qui utilisent la bibliothèque standard.

1.5.4 La fonction main()

Tout programme C++ doit comporter le bloc suivant:

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

Le mot main est le nom de la fonction. Les parenthèses () dans main indiquent que main() est une fonction. Le mot int avant main() indique qu'une valeur entière est renvoyée par la fonction main(). Lorsque le programme est chargé en mémoire, le contrôle passe à la fonction main(), qui est la première fonction exécutée.

Le corps de la fonction est entouré d'accolades ({}). Chaque instruction doit se terminer par un point-virgule (;) comme dans l'instruction suivante:

```
cout << "Hello World!";
```

Cette ligne imprime le message "Hello World!" sur à l'écran par l'objet cout de la bibliothèque d'entrées/sorties iostream via l'opérateur d'insertion <<.

En C++, il est possible d'imprimer plusieurs lignes en appelant cout une seule fois, par exemple:

```
cout << "Apprendre \n la programmation \n en C++";
```

Ce code imprime les trois lignes suivantes à l'écran:

```
Apprendre
la programmation
en C++
```

Les caractères sont imprimés exactement tels qu'ils apparaissent entre guillemets doubles. Cependant, si vous tapez \n, les caractères \n ne s'affichent pas à l'écran. Une barre oblique inverse (\) est appelée un caractère d'échappement. Indique qu'un "caractère spécial" doit être imprimé. Si la chaîne a une barre oblique inverse, le caractère suivant est combiné avec la barre oblique inverse pour former une séquence d'échappement. La séquence d'échappement \n signifie *nouvelle ligne* qui déplace le curseur au début de la ligne suivante sur l'écran.

Le tableau 1.1 donne la liste courante des séquences d'échappement.

Séquence d'échappement	Description
\n	Nouvelle ligne
\t	Tabulation horizontale
\a	Sonnerie (alerte)
\\	Barre oblique inverse (\)
\’	Guillemet simple
\”	Guillemet double

Tableau 1.1: Séquences d'échappement courantes en C++

1.5.5 L'instruction de retour

```
return 0;
```

Lorsqu'un programme finit de s'exécuter, il envoie une valeur au système d'exploitation. Cette instruction de retour particulière renvoie la valeur 0 au système d'exploitation, ce qui signifie "le programme est bien exécuté".

1.6 Notion de variable

Le programme présenté dans la section précédente imprime le texte statique "Hello World!". Cette section présente le concept de variables afin que les programmes puissent effectuer des opérations et des calculs sur les données. Les variables sont des conteneurs pour stocker des valeurs de données.

1.6.1 Déclaration

Pour déclarer (créer) une variable en C++, la syntaxe suivante est utilisée:

```
type nom_variable = valeur;
```

Où *type* est l'un des types C++ (tel que int, double, bool, etc., décrit dans la sous-section suivante) et *nom_variable* est le nom de la variable (identifiant) attribué par le programmeur. Le signe égal est utilisé pour attribuer des valeurs aux variables.
Exemple: Création d'une variable appelée *age* de type int et affectation de 20 comme valeur initiale:

```
int age = 20;
```

A travers l'exemple suivant, nous allons apprendre étape par étape comment calculer la somme de deux entiers.
Exemple: Addition de deux nombres entiers

Nous allons résoudre ce problème en C++ d'une manière primitive en suivant les étapes suivantes:
Étape 1 : Allouer deux espaces mémoire (variables) x et y
Étape 2 : Affecter le premier nombre à x
Étape 3 : Affecter le deuxième nombre à y
Étape 4 : Additionner ces deux nombres et stocker le résultat dans une troisième variable z
Étape 5 : Imprimer le résultat (le contenu de z)

Avant d'utiliser une variable dans un programme, nous devons la déclarer. Cette activité permet au compilateur de mettre à disposition le type d'emplacement approprié dans la mémoire. Les instructions suivantes déclarent trois variables de type entier pour stocker des nombres entiers.

```
int x;
int y;
int z;
```

Puisque x, y et z sont du même type, les trois déclarations précédentes peuvent être écrites sur une seule ligne comme suit:

```
int x, y, z;
```

L'instruction suivante stocke la valeur 10 dans la première variable x:

```
x = 10;
```

L'instruction suivante stocke la valeur 20 dans la deuxième variable y:

```
y = 20;
```

Maintenant, l'addition de x et y, et le stockage du résultat dans z se fait en utilisant l'instruction suivante:

```
z = x + y;
```

La sortie du résultat d'addition stocké dans z se fait maintenant avec la fonction cout comme suit:

```
cout << "La somme est:  ";
cout << z;
```

Vous pouvez combiner les deux lignes ci-dessus en une seule ligne.

```
cout << "La somme est:  " << z;
```

Voici le programme complet:

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

int main() {

//déclarer trois variables entières x, y, z
int x;
int y;
int z;

//stocker les valeurs dans les variables
x = 10;
y = 20;

//additionner les deux nombres x et y, et stocker le résultat dans z
z = x + y;

//afficher le résultat
cout << "La somme est:  "; cout << z;
return 0;
}
```

Sortie du programme:

```
La somme est:  30
```

1.6.2 Types de variables

C++ prend en charge de nombreux types de données. Les types intégrés ou primitifs pris en charge par C++ sont les entiers, les flottants, les caractères et les booléens.

Les types de données couramment utilisés sont décrits dans le tableau 1.2.

Type	Description
int	Petit nombre entier
long int	Grand nombre entier
float	Petit nombre réel
double	Nombre réel double précision
long double	Nombre réel long double précision
char	Un seul caractère
bool	Booléen (logique)

Tableau 1.2: Types simples de données

1.6.3 Conversion de types

Le processus de conversion d'un type en un autre s'appelle le casting. C++ a deux types de conversions.

1. Conversion implicite
2. Conversion explicite

1.6.3.1 Conversion implicite

C++ vous permet de mélanger différents types de données dans une seule expression.
Exemple:

```
double x;
int y = 10;
float z = 3.5;
x = y * z;
```

Si deux opérandes de types différents sont rencontrés dans la même expression, la variable de type inférieure est convertie en variable de type supérieure. Le tableau 1.3 montre l'ordre des types de données.

Type	Ordre
long double double float long int char	du plus élevé ↓ au plus bas

Tableau 1.3: Priorité des types de données

La valeur int de y est convertie en type float et stockée dans une variable temporaire avant d'être multipliée par la variable float z. Le résultat est ensuite converti en double pour pouvoir être affecté à la variable double x.

1.6.3.2 Conversion explicite

Lorsque l'utilisateur modifie manuellement les données d'un type à un autre, cela s'appelle une conversion explicite. En C++, la conversion explicite peut être effectuée de deux manières:
Conversion par affectation: Cela se fait en définissant explicitement le type requis devant l'expression entre parenthèses. Cela peut également être considéré comme un casting puissant.

La syntaxe générale de conversion est la suivante:

```
(type) expression;
```

Où *type* indique le type de données dans lequel le résultat final est converti.
Exemple:

```
double x = 5.25;
//Conversion explicite de type "double" à "int"
int a = (int)x + 5;
cout << "a = " << a;
return 0;
```

Sortie du programme:

```
a = 10
```

Conversion à l'aide de l'opérateur Cast: Un opérateur Cast est un opérateur unaire qui force la conversion d'un type de données en un autre type de données. C++ prend en charge quatre types de transtypage:

- const_cast
- static_cast
- dynamic_cast
- reinterpret_cast

const_cast et static_cast sont les deux opérateurs les plus couramment utilisés:
const_cast est utilisé pour mettre à jour la valeur constante d'un élément ou supprimer la propriété constante d'un élément et peut être utilisé dans des programmes contenant des objets avec des valeurs constantes qui doivent être mises à jour à un moment donné.

La syntaxe d'utilisation de const_cast est la suivante:

```
const_cast<type>(expression);
```

Exemple:

```
using namespace std;
#include <iostream>
int main() {
const int a = 50;
const int* b = &a;
cout << "La valeur précédente est " << *b << "\n";
int* c = const_cast <int *> (b);
*c=100;
cout << "La valeur actuelle est " << *b;
}
```

Dans ce programme l'expression **const int* b** déclare un pointeur nommé **b**, tandis que **&a** prend l'adresse de la variable entière **a** et l'assigne au pointeur **b**.

Sortie du programme:

```
La valeur précédente est 50
La valeur actuelle est 100
```

static_cast est la conversion de type la plus simple que vous puissiez utiliser. Il s'agit d'un cast au moment de la compilation.

La syntaxe d'utilisation est la suivante:

```
static_cast<type>(expression));
```

Exemple:

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

int main()
{
 int a = 5;
 int b = 10;
 double res1 = a/b;
 cout << "Résulat avant conversion = " << res1 << endl;
 double res2 = static_cast<double>(a)/b;
 cout << "Résultat après conversion = " << res2 <<endl;
 return 0;
}
```

Sortie du programme:

```
Résulat avant conversion = 0
Résulat après conversion = 0.5
```

1.7 Les constantes

Une constante (ou littéral) est un élément de données dont la valeur ne change pas pendant l'exécution du programme. C++ autorise les types de littéraux suivants:

- Constantes entières

- Constantes de type caractère
- Constantes flottantes
- Constantes de type chaîne de caractères

En C++, il existe deux façons simples pour définir des constantes:

- Utilisation du préprocesseur #define.
- Utilisation du mot-clé const.

La syntaxe générale d'utilisation du préprocesseur #define est la suivante:

```
#define identifiant val
```

Vous pouvez utiliser le préfixe const pour déclarer des constantes avec un type spécifique comme suit:

```
const type identifiant = value;
```

1.7.1 Constantes entières

Une constante entière est un entier sans partie fractionnaire. Il existe trois types de constantes entières disponibles en C++. Constantes entières décimales: se composent d'une série de chiffres et ne peuvent pas commencer par un 0 (zéro). Par exemple, 125, -180, +100. Constantes entières octales: consistent en une série de nombres commençant par 0 (zéro). Par exemple 015, 028. Constantes entières hexadécimales: se composent d'une série de chiffres précédés de ox ou OX.
Exemples:

```
#define nb_etudiants 50

const int N = 5;
```

1.7.2 Constantes de type caractère

Une constante de type caractère en C++ doit contenir un ou plusieurs caractères et doit être entourée de guillemets simples. Par exemple 'A', '9', etc. C++ autorise les caractères non graphiques qui ne peuvent pas être saisis directement à partir du clavier, par exemple, retour arrière, tabulation, retour chariot, etc. Ces caractères peuvent être représentés à l'aide d'une séquence d'échappement (Tableau 1.1). Une séquence d'échappement représente un seul caractère.
Exemples:

```
#define nouvelle_ligne '\n'
const char symbole = 'H';
```

1.7.3 Constantes flottantes

Elles sont aussi appelées constantes réelles. Ce sont des nombres ayant des parties fractionnaires. Ils peuvent être écrits sous forme fractionnaire ou sous forme d'exposant. Une constante réelle sous forme fractionnaire se compose de chiffres signés ou non signés comprenant un point décimal entre les chiffres. Par exemple 3.0, -17.0, -0.627, etc.
Exemples:

```
#define PI 3.14
const float e 2.71828;
```

1.7.4 Constantes de type chaîne de caractères

Une séquence de caractères entre guillemets doubles est appelée chaîne littérale. Le littéral de chaîne est par défaut (automatiquement) ajouté avec un caractère spécial '\0' qui indique la fin de la chaîne. Par conséquent, la taille de la chaîne est augmentée d'un caractère. Par exemple "information" sera représenté comme "information\0" dans la mémoire et sa taille est de 12 caractères.
Exemples:

```
#define message "Bienvenue"
const string module = "Informatique";
```

1.8 Les identifiants

Les identifiants sont des chaînes extraites du jeu de caractères C++ et données aux entités (variables, constantes, fonctions, etc.) pour les identifier de manière unique lors de la compilation du programme. Les règles suivantes s'appliquent aux identifiants:

- Un identifiant peut être composé d'alphabets, de chiffres et/ou de traits soulignés (_).
- Il ne doit pas commencer par un chiffre.
- C++ est sensible à la casse. Autrement dit, les lettres majuscules et minuscules sont traitées différemment.
- Il ne doit pas être un mot réservé.

Le tableau 1.4 montre le jeu de caractères utilisé en C++, comme dans les autres langages de programmation:

Lettres	A-Z, a-z
Chiffres	0-9
Caractères spéciaux	Espace + - * / ^\() [] {} = != <>‘ “ $, ; : % ! & ? _ # <= >= @
Caractères de formatage	Saut de ligne, tabulation horizontale, tabulation verticale, etc.

Tableau 1.4: Jeux de caractères en C++

1.9 Les mots réservés

C++ a des mots réservés qui ont une signification prédéfinie pour le compilateur appelés mots-clés. Ces mots ne peuvent pas être utilisés comme identifiants. Le tableau 1.5 répertorie les mots-clés couramment utilisés (réservés) en C++.

asm	auto	bool	break	case
catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	export
extern	false	float	for	friend
goto	if	inline	int	long
mutable	namespace	new	operator	private
protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_cast
struct	switch	template	this	throw
true	try	typedef	typeid	typename
union	unsigned	using	virtual	void
volatile	wchar_t	while		

Tableau 1.5: Liste des mots réservés en C++

1.10 Les signes de ponctuation

Les signes de ponctuation dans le langage C++ ont une signification syntaxique et sémantique pour le compilateur. Certains signes de ponctuation, seuls ou en combinaison, peuvent être des opérateurs C++ et peuvent être significatifs pour le préprocesseur.

Tous les caractères suivant peuvent être convertis en ponctuation:

! % ^& * ( ) - + = { } | ~[ ] \; ’ : " <>? , . / #

Le tableau 1.6 montre quelques exemples d’utilisation des ponctuations en C++.

Crochets [ ]	Les crochets ouvrants et fermants indiquent un indice de tableau simple et multidimensionnel
Parenthèse ( )	Les parenthèses ouvrantes et fermantes indiquent les appels de fonctions ; paramètres de fonction pour grouper des expressions, etc.
Accolades { }	Les accolades ouvrantes et fermantes indiquent le début et la fin d’une instruction composée
Virgule ,	Il est utilisé comme séparateur dans une liste d’arguments de fonction
Point-virgule ;	Il est utilisé comme terminateur d’instruction
Colon :	Il indique une instruction étiquetée ou un symbole d’opérateur conditionnel
Astérisque *	Il est utilisé dans la déclaration de pointeur ou comme opérateur de multiplication
Signe égal =	Il est utilisé comme opérateur d’affectation
Signe dièse #	Il est utilisé comme directive

Tableau 1.6: Les ponctuations en C++

1.11 Les opérateurs

Les opérateurs sont des symboles spéciaux utilisés à des fins spécifiques. C++ fournit six types d'opérateurs : opérateurs arithmétiques, opérateurs relationnels, opérateurs logiques, opérateurs unaires, opérateurs d'affectation, opérateurs conditionnels. En outre, l'opérateur virgule.

1.11.1 Opérateurs arithmétiques

Les opérateurs arithmétiques +, -, *, / et % sont utilisés pour effectuer une opération arithmétique (numérique). Vous pouvez utiliser les opérateurs +, -, * et / avec les données de type entier et à virgule flottante. L'opérateur modulo ou reste % est utilisé uniquement avec les données de type entier. Les opérateurs qui ont deux opérandes sont appelés opérateurs binaires.
Exemples:

```cpp
#include <iostream>

int main() {
    // Addition
    int a = 5;
    int b = 3;
    int addition = a + b;
    std::cout << "Addition : " << addition << std::endl;

    // Soustraction
    int x = 10;
    int y = 7;
    int soustraction = x - y;
    std::cout << "Soustraction : " << soustraction << std::endl;

    // Multiplication
    float c = 4.5;
    float d = 2.0;
    float multiplication = c * d;
    std::cout << "Multiplication : " << multiplication << std::endl;

    // Division
    float p = 10.0;
    float q = 3.0;
    float division = p / q;
    std::cout << "Division : " << division << std::endl;

    // Modulo (reste)
    int e = 13;
    int f = 4;
    int modulo = e % f;
    std::cout << "Modulo : " << modulo << std::endl;

    return 0;
}
```

Sortie du programme:

```
Addition : 8
Soustraction : 3
Multiplication : 9
Division : 3.33333
Modulo : 1
```

1.11.2 Opérateurs relationnels

Les opérateurs relationnels permettent de tester la relation entre deux valeurs. Tous les opérateurs relationnels sont des opérateurs binaires et nécessitent donc deux opérandes. Une expression relationnelle renvoie zéro lorsque la relation est fausse et non nulle lorsqu'elle est vraie.

Les opérateurs relationnels sont décrits dans le tableau 1.7 suivant.

Opérateur relationnel	Description
<	Inférieur à
<=	Inférieur ou égal à
==	Egal à
>	Supérieur à
>=	Supérieur ou égal à
! =	Différent de

Tableau 1.7: Opérateurs relationnels

Exemples:

```cpp
#include <iostream>

int main() {
    int a = 5;
    int b = 3;

    // Égalité
    bool estEgal = (a == b);
    std::cout << "a == b : " << estEgal << std::endl;

    // Différence
    bool nEstPasEgal = (a != b);
    std::cout << "a != b : " << nEstPasEgal << std::endl;

    // Supérieur
    bool estSuperieur = (a > b);
    std::cout << "a > b : " << estSuperieur << std::endl;

    // Inférieur
    bool estInferieur = (a < b);
    std::cout << "a < b : " << estInferieur << std::endl;

    // Supérieur ou égal
    bool estSuperieurOuEgal = (a >= b);
```

```
    std::cout << "a >= b : " << estSuperieurOuEgal << std::endl;

    // Inférieur ou égal
    bool estInferieurOuEgal = (a <= b);
    std::cout << "a <= b : " << estInferieurOuEgal << std::endl;

    return 0;
}
```

Sortie du programme:

```
a == b : 0
a != b : 1
a > b : 1
a < b : 0
a >= b : 1
a <= b : 0
```

1.11.3 Opérateurs logiques

Les opérateurs logiques sont utilisés pour combiner une ou plusieurs expressions relationnelles.

Les opérateurs logiques sont décrits dans le tableau 1.8 suivant:

Opérateur	Description
||	OU (OR)
&&	ET (AND)
!	PAS (NOT)

Tableau 1.8: Opérateurs logiques

Exemples:

```
#include <iostream>

int main() {
    int x = 5;
    int y = 3;

    bool condition1 = (x > y);
    bool condition2 = (x < 10);

    // Utilisation de l'opérateur ET logique
    bool resultatEt = (condition1 && condition2);
    std::cout << "Condition1 ET Condition2 : " << resultatEt << std::endl;

    // Utilisation de l'opérateur OU logique
    bool resultatOu = (condition1 || condition2);
    std::cout << "Condition1 OU Condition2 : " << resultatOu << std::endl;

    // Utilisation de l'opérateur NON logique
    bool resultatNon = !condition1;
    std::cout << "NON Condition1 : " << resultatNon << std::endl;
```

```
    return 0;
}
```

Sortie du programme:

```
Condition1 ET Condition2 : 1
Condition1 OU Condition2 : 1
NON Condition1 : 0
```

1.11.4 Opérateurs unaires

C++ fournit deux opérateurs unaires pour lesquels une seule variable est requise.
Exemple:

```
a = - 50;
a = + 50;
```

Ici, le signe plus (+) et le signe moins (-) sont unaires car ils ne sont pas utilisés entre deux variables.

1.11.5 Opérateurs d'affectation

En C++, les opérateurs d'affectation peuvent être simples ou composés.

Habituellement, vous affectez une valeur à une variable à l'aide de l'opérateur d'affectation simple '='. Cet opérateur prend l'expression de droite et l'insère dans la variable de gauche.
Exemple:

```
m = 5;
```

L'opérateur prend l'expression (valeur) 5 à droite et la stocke dans la variable m à gauche.

En plus de l'opérateur d'affectation standard présenté ci-dessus, C++ prend également en charge les affectations multiples.
Exemple:

```
x = y = z = 32;
```

Ce code stocke la valeur 32 dans chacune des trois variables x, y et z.
Les opérateurs d'affectation composés se composent d'un opérateur binaire et d'un opérateur d'affectation simple. Ils effectuent l'opération de l'opérateur binaire sur les deux opérandes et stockent le résultat de cette opération dans l'opérande de gauche, qui doit être une valeur modifiable.

Le tableau 1.9 montre les types d'opérateurs composés.

Opérateur	Exemple	Equivalant à
+ =	A + = 2	A = A + 2
- =	A - = 2	A = A - 2
% =	A % = 2	A = A % 2
/=	A/ = 2	A = A / 2
*=	A * = 2	A = A * 2

Tableau 1.9: Opérateurs d'affectation composés

1.11.6 Opérateurs d'incrémentation et de décrémentation

C++ fournit deux opérateurs spéciaux, '++' et '- -', pour incrémenter et décrémenter la valeur d'une variable de un. Les opérateurs d'incrémentation/décrémentation peuvent être utilisés sur n'importe quel type de variable, mais pas sur les constantes. Les opérateurs d'incrémentation et de décrémentation ont deux formes, respectivement pré et post.

La syntaxe des opérateurs d'incrémentation/décrémentation est la suivante:

```
Pré-incrément: ++variable
Post-incrément: variable++
Pré-décrémentation: --variable
post-décrémentation: variable--
```

Exemples:

```
int x, y;
int i = 10, j = 10;
x = ++i; //ajouter 1 à i, puis stocker la nouvelle valeur de i dans x
y = j++; //stocker la valeur de j dans y puis ajouter 1 à j
cout << x <<endl; //11
cout << y <<endl;  //10
x = --x;
y = y--;
cout << x <<endl; // 10
cout << y <<endl; // 10
return 0;
```

1.11.7 L'opérateur conditionnel (?:)

L'opérateur conditionnel ?: est appelé opérateur ternaire, car il nécessite trois opérandes. Le format de l'opérateur conditionnel est : Expression conditionnelle ? expression1 : expression2 ;

Si la valeur de l'expression conditionnelle est true (vrai), l'expression1 est évaluée, sinon l'expression2 est évaluée.

Exemple:

```
int a = 5, b = 6;
plus_grand = (a > b) ? a : b;
```

La condition est évaluée à false (faux), donc la valeur de l'expression (variable) sera affectée à plus_grand.

1.11.8 L'opérateur virgule (,)

L'opérateur virgule (,) donne une évaluation de gauche à droite des expressions. Lorsque l'ensemble d'expressions doit être évalué pour une valeur, seule l'expression la plus à droite est prise en compte.
Exemple:

```
int a = 1, b = 2, c = 3, i; //la virgule sert de séparateur, pas d'opérateur
i = (a, b); //stocke b dans i
```

Ce code affecte d'abord la valeur de a à la variable i, puis la valeur de b à la variable i. Par conséquent, la variable i contiendra la valeur 2.

1.11.9 Ordre des opérateurs

L'ordre dans lequel les opérateurs sont utilisés dans une expression donnée est appelé priorité.

Le tableau 1.10 montre la priorité des opérateurs du plus élevé au plus bas:

++, – (post-incrément/décrément)	du plus élevé ↓ au plus bas
++ (pré-incrément), – (pré-décrément), sizeof ( ), !(not), - (unaire), +(unaire)	
*,/, %	
+, -	
<, <=, >, >=	
==,!=	
&&	
? :	
=	
L'opérateur virgule	

Tableau 1.10: Ordre de priorité des opérateurs

1.12 Conclusion

Dans ce chapitre, nous avons présenté de façon simplifiée et exhaustive les principaux éléments du langages C++ pour permettre à l'étudiant de se lancer dans l'utilisation du langage C++ rapidement tout en évitant les difficultés éventuellement rencontrées avec les débutants du langage, notamment sur la manière d'installation et d'utilisation de son environnement de développement.

A la fin de ce chapitre, l'étudiant devrait être capable d'écrire, déboguer et exécuter des codes simples en C++ afin qu'il puisse acquérir progressivement des nouvelles compétences comme les structures conditionnelles et répétitives que nous allons apprendre dans le chapitre suivant.

Série d'exercices N°1

Exercice 1.1 Écrire un programme en C++ qui calcule la surface et le périmètre d'un cercle.

Exercice 1.2 Écrire un programme en C++ qui demande à l'utilisateur d'introduire 3 nombres entiers, puis calculer et afficher leur moyenne.

Exercice 1.3 Écrire un programme en C++ qui calcule et affiche le quotient et le reste de la division d'un nombre entier par rapport à un autre.

Exercice 1.4 Écrire un programme en C++ qui demande à l'utilisateur d'introduire 3 nombres en virgule flottante, puis utiliser l'opérateur conditionnel (?:) pour déterminer le plus grand nombre.

Exercice 1.5 Écrire un programme en C++ qui permet de calculer le montant TTC (Toutes Taxes comprises) en fonction du prix unitaire de l'unité, la quantité achetée, et le taux TVA (supposé égal à 17%).

Exercice 1.6 Écrire un programme en C++ qui demande à l'utilisateur de saisir une température en degrés Celsius, puis convertissez-la en degrés Fahrenheit en utilisant la formule : $F = (C \times 9/5) + 32$.

Exercice 1.7 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre de base et un exposant, puis calculez et affichez la valeur de la puissance.

CHAPITRE

2

Les structures conditionnelles et répétitives

2.1 Introduction

Dans ce chapitre, nous allons apprendre deux structures fortement liées: les structures conditionnelles et les structures répétitives (les boucles). Une structure conditionnelle consiste à utiliser des expressions booléennes pour pouvoir exécuter uniquement une partie du programme selon qu'une certaine condition soit satisfaite. Tandis qu'une structure répétitive est une façon de répéter quelque chose quand une certaine condition est satisfaite, ou vraie.

2.2 Les structures conditionnelles

Parfois, le programme doit être exécuté en fonction d'une condition particulière. En programmation, les structures qui permettent de prendre une décision sont appelées structures conditionnelles qui exigent de spécifier une ou plusieurs conditions à évaluer ou à tester par le programme, ainsi qu'une ou plusieurs instructions à exécuter si la condition est déterminée comme *vrai (true)*, et éventuellement, d'autres instructions à exécuter si la condition est déterminée comme *faux (false)*. Par exemple, dans un système de contrôle d'accès, une structure conditionnelle "if" pourrait être utilisée pour permettre l'ouverture d'une porte si une carte d'accès valide est présentée (condition vraie) ou pour refuser l'accès si la carte n'est pas valide (condition fausse). Cela assure la sécurité en ne permettant l'accès qu'aux personnes autorisées.

C++ fournit les instructions suivantes pour implémenter une structure conditionnelle:

- L'instruction if
- L'instruction if-else
- L'instruction if imbriquée

- L'instruction switch

2.2.1 L'instruction *if*

Une instruction if consiste en une expression booléenne suivie d'une ou plusieurs instructions. Si l'expression booléenne prend la valeur true, le bloc de code à l'intérieur de l'instruction if sera exécuté. Si l'expression booléenne prend la valeur false, le premier ensemble de code après la fin de l'instruction if (après l'accolade fermante) sera exécuté.

La syntaxe générale de l'instruction if est la suivante:

```
if (condition) {
      action(s);
}
```

À partir de l'organigramme illustré à la figure 2.1, il est clair que si la condition est vraie, l'instruction est exécutée; sinon, elle est ignorée. L'instruction peut être une instruction simple ou composée, c'est-à-dire il s'agit d'une simple action ou bloc d'actions.

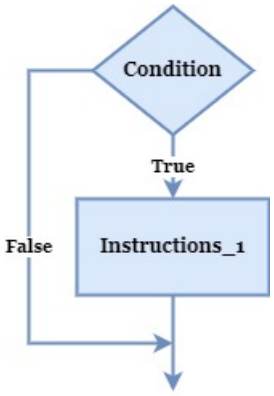


Figure 2.1: Structure de l'instruction *if*

Exemple:

```
if (x >= 0)
    cout << "x est positif";
```

2.2.2 L'instruction *if-else*

Une instruction if peut être suivie d'une instruction else facultative qui s'exécute si l'expression booléenne est fausse.

L'instruction if-else a la syntaxe suivante :

```
if (condition)
    instructions_1;
else
    instructions_2;
```

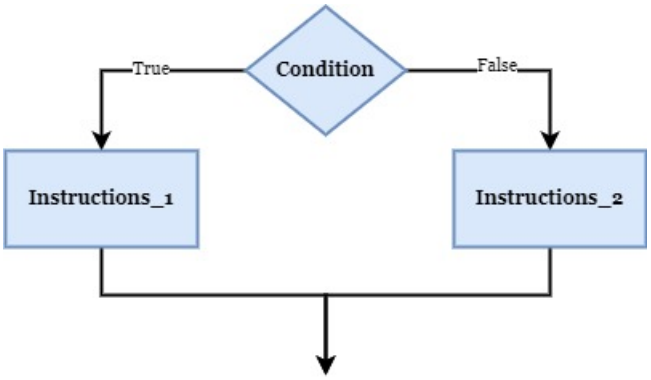


Figure 2.2: Structure de l'instruction *if-else*

Dans l'organigramme ci-dessus (figure 2.2), nous pouvons voir que la condition spécifiée est évaluée en premier. Si la condition est vraie, Instructions_1 est exécuté. Si la condition est fausse, Instructions_2 est exécuté. Instructions_1 et Instructions_2 peuvent être des instructions simples ou composées (bloc d'instructions).
Exemple:

```
if (x >= 0)
    cout << "x est postif";
else
    cout << "x est négatif";
```

2.2.3 L'instruction if imbriquée

Un bloc if peut être imbriqué dans un autre bloc if ou un autre bloc else. C'est ce qu'on appelle l'imbrication conditionnelle.

La syntaxe d'utilisation des instructions if imbriquées est la suivante:

```
if(condition 1) {

  if(condition_2) {

    instructions;
    ....
    if (condition_n){
       ....
        }
  }
}
```

Exemple:

```
float moyenne = 18.50;

if (moyenne >= 10) {
  cout << "Admis" << endl;
  if (moyenne >= 16) {
    cout <<"Très bien" << endl;
```

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

2.2.4 L'instruction *switch*

Les instructions if et if-else autorisent les branchements bidirectionnels, tandis que l'instruction switch autorise les branchements multiples. Une instruction switch permet de tester l'égalité d'une variable par rapport à une liste de valeurs. Chaque valeur est appelée un cas et la variable/expression passée à switch est vérifiée pour chaque cas.

La syntaxe de l'instruction switch est:

```
switch (var / expression)
{
case const1:
    instructions_1;
    break;
case const2:
    instructions_2;
    break;
...
default:
    instructions_N;
    break;
}
```

L'exécution d'une instruction switch commence par l'évaluation d'une expression. Si la valeur de l'expression correspond à la constante, les instructions qui suivent cette instruction sont exécutées dans l'ordre jusqu'à ce que break soit exécutée. L'instruction break passe le contrôle à la fin de l'instruction switch. Si la valeur de l'expression ne correspond pas à la constante, l'instruction par défaut est exécutée.

Quelques points importants sur les instructions switch:

- L'expression de l'instruction switch doit être de type entier ou caractère.
- Le cas par défaut peut être placé à n'importe quel endroit.
- Les valeurs de cas (case) n'ont pas besoin d'être dans un ordre spécifique.

Exemple:

```
int jour = 3;
switch(jour) {
   case 1:
     cout << "Samedi" ;
     break;
   case 2:
     cout << "Dimanche" ;
     break;
   case 3:
     cout << "Lundi" ;
     break;
   case 4:
```

```
      cout << "Mardi" ;
      break;
   case 5:
      cout << "Mercredi" ;
      break;
   case 6:
      cout << "Jeudi" ;
      break;
   case 7:
      cout << "Vendredi" ;
      break;
   default:
      cout << "Une valeur non valide" ;

}
```

Sortie du programme:

```
Lundi
```

Lorsque le mot-clé break est atteint, le programme quitte le bloc switch. Cela empêchera tous les autres tests du bloc de s'exécuter.

Le mot-clé *default* spécifie du code à exécuter s'il n'y a pas de correspondance de cas, et il doit être utilisé comme dernière instruction dans le bloc switch et ne nécessite pas de *break*.

2.2.5 L'instruction switch imbriquée

Il est possible d'avoir une instruction switch dans la séquence d'instructions d'une autre instruction switch externe. Même si les constantes de cas des switch interne et externe contiennent des valeurs communes, aucun conflit ne surviendra.

La syntaxe des instructions switch imbriquées est:

```
switch(exp1) {
   case 'A':
      cout << "Ce cas 'A' fait partie du switch extérieur";
      switch(exp2) {
         case 'A':
            cout << "Ce cas 'A' fait partie du switch intérieur";
            break;
         case 'B':
         ...
      }
      break;
   case 'B':
   ...
}
```

2.3 Les structures répétitives

Une structure répétitive est également appelée structure de contrôle répétitive. Parfois, vous devez exécuter plusieurs fois une série d'instructions, en modifiant à chaque fois la

valeur d'une ou plusieurs variables pour obtenir un résultat différent. Ce type d'exécution de programme s'appelle une boucle.

C++ fournit les structures répétitives suivantes:

- La boucle *while*
- La boucle *do-while*
- La boucle *for*

2.3.1 La boucle *while*

L'instruction de boucle while exécute une instruction ou un bloc d'instructions de manière répétée tant qu'une condition spécifique est vraie.

La boucle while a la syntaxe suivante:

```
while(condition) {
    action(s);
}
```

L'organigramme de la figure 2.3 montre que la condition est évaluée en premier. Si la condition est vraie, le corps de la boucle est exécuté et la condition est réévaluée. Par conséquent, le corps de la boucle est exécuté à plusieurs reprises tant que la condition est vraie. Lorsque la condition devient fausse, le programme quitte la boucle et passe à l'instruction suivante.

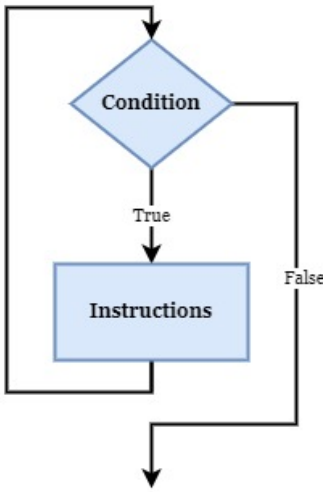


Figure 2.3: Structure de la boucle *while*

Exemple:

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

int main () {
   //déclaration d'une variable a de type entier, initalisée à 1
   int a = 1;
```

```
    //exécution de la boucle while
    while(a <= 5) {
        cout << "valeur de a = " << a << endl; //afficher la valeur actuelle de a
        a++; //incrémenter la valeur de a
    }

    return 0;
}
```

Sortie du programme:

```
valeur de a = 1
valeur de a = 2
valeur de a = 3
valeur de a = 4
valeur de a = 5
```

2.3.2 La boucle *do-while*

Une boucle do-while (figure 2.4) est similaire à une boucle while, mais une boucle do-while est garantie de s'exécuter au moins une fois.

La boucle do-while a la syntaxe suivante:

```
do {
    instructions;
} while (condition);
```

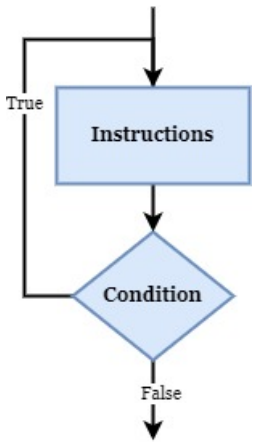


Figure 2.4: Structure de la boucle *do-while*

Le corps de la boucle est toujours exécuté au moins une fois. Une différence importante entre les boucles while et do-while est l'ordre relatif d'exécution du test de condition et du corps de la boucle. Une boucle while effectue un test d'itération de boucle avant chaque exécution du corps de la boucle. Si le premier test échoue, le corps de la boucle n'est pas exécuté du tout. Une boucle do-while exécute un test de fin de boucle après chaque exécution du corps de la boucle. Par conséquent, le corps de la boucle est toujours exécuté au moins une fois.

Exemple:

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

int main () {
   //déclaration d'une variable locale de type entier, initalisée à 10

   int a = 10;

   //exécution de la boucle do-while
   do {
      cout << "valeur de a =  " << a << endl; //afficher la valeur actuelle de a
      a = a + 1;
   } while( a < 15 );

   return 0;
}
```

Sortie du programme:

```
valeur de a = 10
valeur de a = 11
valeur de a = 12
valeur de a = 13
valeur de a = 14
```

2.3.3 La boucle *for*

Il s'agit d'une boucle contrôlée par comptage dans le sens où le programme sait à l'avance combien de fois exécuter la boucle.

La boucle for a la syntaxe suivante:

```
for (initialisation; décision; incrément/décrément) {
    instructions;
 }
```

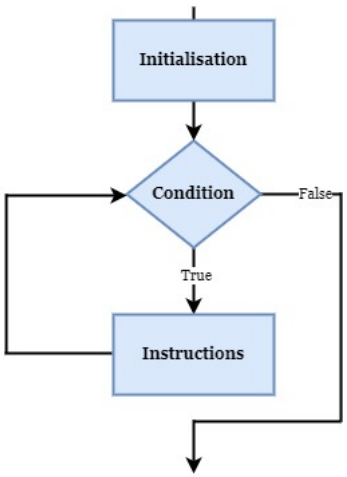


Figure 2.5: Structure de la boucle *for*

L'organigramme (figure 2.5) montre que la boucle for effectue trois opérations:

- Initialisation de la variable de contrôle de la boucle.
- Test de la variable de contrôle.
- Mettre à jour la variable de contrôle soit par incrémentation soit par décrémentation dans le bloc des instructions.

Exemple:

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

int main () {
   //exécution de la boucle for
   for( int a = 1; a < 7; a = a + 1 ) {
      cout << "valeur de a = " << a << endl; //afficher la valeur actuelle de a
   }

   return 0;
}
```

Sortie du programme:

```
valeur de a = 1
valeur de a = 2
valeur de a = 3
valeur de a = 4
valeur de a = 5
valeur de a = 6
```

2.4 Instructions de saut

Une instruction de saut transfère inconditionnellement le contrôle du programme dans une fonction en utilisant l'une des instructions suivantes:

- L'instruction goto
- L'instruction break
- L'instruction continue
- La fonction exit()

2.4.1 L'instruction *goto*

L'instruction goto fournit un saut inconditionnel du goto à une instruction étiquetée dans la même fonction. Pratiquement, l'utilisation de l'instruction goto est fortement déconseillée car elle rend difficile le suivi du flux de contrôle d'un programme, rendant le programme difficile à comprendre et difficile à modifier.

L'instruction goto a la syntaxe suivante:

```
goto etiq;
```

Où, etiq est appelé étiquette (label). Il s'agit d'un identifiant personnalisé. Après avoir exécuté l'instruction goto, le contrôle passe à la ligne qui suit l'étiquette etiq.
Exemple:

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

int main() {
   //déclaration et initialisation d'une variable locale
   int a = 5;

   //exécution de la boucle do-while
   etiq_boucle:
   do {
      if( a == 7) {
          //sauter l'itération si a = 7
          a = a + 1;
          goto etiq_boucle;
      }
      cout << "valeur de a = " << a << endl;
      a = a + 1;
   }while(a < 10);

   return 0;
}
```

Sortie du programme:

```
valeur de a = 5
valeur de a = 6
valeur de a = 8
valeur de a = 9
```

2.4.2 L'instruction *break*

Une instruction break fournit une sortie dès qu'elle est exécutée dans n'importe quelle boucle.

La syntaxe de l'instruction break est:

```
break;
```

Exemple:

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

int main () {
   //déclaration d'une variable locale de type entier
   int a = 1;

   //exécution de la boucle do-while
   do {
      cout << "valeur de a = " << a << endl; //afficher la valeur actuelle de a
      a = a + 1;
      if(a > 5) {
         break; //terminer immédiatement l'exécution de la boucle
      }
   } while(a < 10);

   return 0;
}
```

Sortie du programme:

```
valeur de a = 1
valeur de a = 2
valeur de a = 3
valeur de a = 4
valeur de a = 5
```

2.4.3 L'instruction *continue*

L'instruction continue fonctionne comme l'instruction break. Cependant, au lieu de forcer une sortie, continue force la prochaine itération de la boucle, en sautant tout code entre les deux. Pour une boucle for, continue exécute le test conditionnel et incrémente la partie de la boucle. Pour les boucles while et do-while, le contrôle du programme passe aux tests conditionnels.

La syntaxe de continue est:

```
continue;
```

Exemple:

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

int main () {
   //déclaration d'une variable locale de type entier
   int a = 1;

   // exécution de la boucle do-while
   do {
      if(a == 5) {
         a = a + 1;
```

```
            continue;  //sauter l'itération
        }
        cout << "valeur de a = " << a << endl; //afficher la valeur actuelle de a
        a = a + 1;
    }
    while(a < 8);

    return 0;
}
```

Sortie du programme:

```
valeur de a = 1
valeur de a = 2
valeur de a = 3
valeur de a = 4
valeur de a = 6
valeur de a = 7
```

2.4.4 La fonction *exit()*

La fonction exit() permet d'arrêter à tout moment l'exécution du programme et de communiquer un code d'état au programme appelant. La forme générale est exit(code); où code est une valeur entière. Se le programme est exécuté correctement, le code aura une valeur de 0. Les valeurs de code varient selon le système d'exploitation.
Exemple:

```
#include <iostream>
using namespace std;
int main()
{
        int x;
        cout << "Saisir une valeur différente de zéro: ";
        cin >> x; //valeur entrée par l'utilisateur

        if(x)    //vérifier si l'entrée utilisateur est différente de zéro ou non
        {
                cout << "Valeur enteée valide\n";
        }
        else
        {
                cout << "Erreur !";  //le programme se termine si la valeur de x est
↪  égal à 0
                exit(0);
        }
        cout << "la valeur entrée de x = " << x;
}
```

Étant donné que l'entrée utilisateur pour le code ci-dessus fourni est zéro (0), la partie else est exécutée pour l'instruction if-else. Plus loin, là où le compilateur rencontre la fonction exit(), il termine le programme. Même l'instruction d'impression sous le if-else n'est pas exécutée puisque le programme a déjà été terminé par la fonction exit().

2.5 Conclusion

Dans ce chapitre, nous avons présenté deux concepts fondamentaux en programmation: les structures conditionnelles et les structures répétitives. Les structures conditionnelles sont utilisées pour modifier le comportement du programme en conséquence selon qu'une condition soit vérifiée. Tandis que les structures répétitives, appelées aussi boucles, sont utilisées pour répéter le même fragment de code plusieurs fois de suite.

Série d'exercices N°2

Exercice 2.1 Écrire un programme en C++ pour vérifier si un nombre est premier.

Exercice 2.2 Écrire un programme en C++ pour vérifier si un nombre entier donné est compris entre 5 et 10.

Exercice 2.3 Écrire un programme en C++ pour calculer la somme des 10 entiers naturels de 1 à 10 en utilisant les boucles for, while et do-while.

Exercice 2.4 Écrire un progamme en C++ qui calcule le factoriel d'un nombre entier.

Exercice 2.5 Écrire un programme en C++ pour convertir un chiffre à une à chaîne de caractères.

Exercice 2.6 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre entier et affiche la table de multiplication de ce nombre de 1 à 10 en utilisant une boucle "for".

Exercice 2.7 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre entier. Calculer sa valeur absolue, puis utilisez une boucle "while" pour calculer la factorielle de ce nombre (n!).

CHAPITRE 3

Les fonctions

3.1 Introduction

Les langages de programmation utilisent le concept de fonction pour permettre aux programmeurs de créer des programmes complexes qui s'étendent sur des milliers de lignes de code. Cela réduit la redondance du code et permet aux programmeurs d'utiliser la puissance de l'abstraction pour nommer des séquences d'instructions. Les programmes écrits avec des fonctions sont plus faciles à maintenir, à mettre à jour et à déboguer que les programmes très longs. La programmation modulaire (fonctionnelle) permet à plusieurs programmeurs de travailler indépendamment sur des fonctions distinctes, qui peuvent ensuite être combinées pour créer un projet complet. Ce chapitre présente les principaux concepts de création et d'utilisation des fonctions en C++.

3.2 Définition

Une fonction est un bloc de code (i.e. un ensemble d'instructions regroupées pour effectuer une opération particulière) qui manipule des données et renvoie souvent une valeur. Une fonction a son propre nom et lorsque ce nom est rencontré dans un programme, l'exécution du programme se branche sur le corps de cette fonction. Lorsque la fonction se termine, l'exécution revient à la région du code du programme où elle a été appelée et le programme passe à la ligne suivante du code.

3.3 Création des fonctions

3.3.1 Déclaration d'une fonction

La déclaration appelée prototypage d'une fonction, informe le compilateur des fonctions à utiliser dans un programme, les arguments qu'elles prennent, et les types des valeurs qu'elles renvoient.

La déclaration d'une fonction a la forme suivante:

```
type_retour nom_fonction(liste des paramètres);
```

Exemple:

```
int max(int num1, int num2);
```

Les noms des paramètres ne sont pas importants dans la déclaration d'une fonction, seul leur type est requis, donc ce qui suit est également une déclaration valide:

```
int max(int, int);
```

La déclaration de la fonction est requise lorsque vous définissez une fonction dans un fichier source et que vous appelez cette fonction dans un autre fichier. Dans ce cas, vous devez déclarer la fonction en haut du fichier appelant la fonction.

3.3.2 Structure de la fonction

La définition d'une fonction indique au compilateur quelle tâche la fonction exécute. Le prototype de la fonction et sa définition doivent avoir le même type de retour, le même nom et les mêmes paramètres.

La forme générale de définition d'une fonction en C++ est:

```
type_retour nom_fonction(liste des paramètres) {
    corps de la fonction
}
```

La définition d'une fonction C++ se compose d'un en-tête et d'un corps, et d'autres parties importantes:

- **Type de retour**: Une fonction peut retourner une valeur. Le type de retour est le type de données de la valeur renvoyée par la fonction. Certaines fonctions effectuent les opérations souhaitées sans renvoyer de valeur. Dans ce cas, le type de retour est le mot-clé void.
- **Nom de la fonction**: Il s'agit du nom réel de la fonction. Le nom de la fonction et la liste des paramètres constituent ensemble la signature de la fonction.
- **Paramètres**: Un paramètre est comme un espace réservé. Lorsqu'une fonction est invoquée, vous transmettez une valeur au paramètre. Cette valeur est appelée paramètre ou argument réel. La liste des paramètres fait référence au type, à l'ordre et au nombre des paramètres d'une fonction. Les paramètres sont facultatifs; c'est-à-dire qu'une fonction peut ne pas contenir aucun paramètre.

Exemple: Voici le code source d'une fonction appelée max(). Cette fonction prend deux paramètres num1 et num2 et renvoie le plus grand des deux:

```
//fonction retournant le max de deux nombres
int max(int num1, int num2) {

   //déclaration d'une variable locale
   int max_num;
```

```
   if (num1 > num2)
      max_num = num1;
   else
      max_num = num2;

   return max_num;
}
```

3.4 Les paramètres

La fonction appelante peut fournir des valeurs à la fonction appelée. Ceux-ci sont appelés paramètres. Les variables qui fournissent des valeurs à la fonction appelante sont appelées paramètres réels. Une variable qui reçoit une valeur d'une instruction est appelée un paramètre formel.

Prenons l'exemple suivant qui évalue l'aire d'un cercle:

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

void aire(float);

int main() {
    float r; //le rayon du cercle
    cin >> r;
    aire(r); //appeler la fonction aire
    return 0;
}
void aire(float rayon) {
cout << "L'aire du cercle = " << 3.14 * rayon *  rayon << "\n";
 }
```

Ici, r est appelé paramètre réel et rayon est appelé paramètre formel.

3.5 Appel d'une fonction

Lorsque vous créez une fonction C++, vous spécifiez ce que vous voulez que la fonction fasse. Pour utiliser une fonction, vous devez l'appeler.

Lorsqu'un programme appelle une fonction, le contrôle du programme passe à la fonction appelée. La fonction appelée exécute la tâche définie et rend le contrôle au programme principal lorsque l'instruction return est exécutée ou que l'accolade fermante à la fin de la fonction est atteinte.

Pour appeler une fonction, transmettez simplement les paramètres requis avec le nom de la fonction. Si la fonction renvoie une valeur, vous pouvez stocker la valeur renvoyée.
Exemple:

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

//déclaration de la fonction max
int max(int num1, int num2);
```

```
int main () {
   // variables locales
   int a = 100;
   int b = 200;
   int m;

   //appel de la fonction pour obtenir le max
   m = max(a, b);
   cout << "Le maximum est " << m << endl;

   return 0;
}

//définition de fonction qui retourne le max de deux nombres

int max(int num1, int num2) {
   // déclaration de variables locales
   int max_num;

   if (num1 > num2)
      max_num = num1;
   else
      max_num = num2;

   return max_num;
}
```

Sortie du programme:

```
Le maximum est 200
```

En C++, vous pouvez appeler des fonctions à l'aide de l'une des méthodes suivantes:

- Appel par valeur.
- Appel par référence.
- Appel par pointeur.

3.5.1 Appel par valeur

La méthode d'appel par valeur consistant à transmettre la valeur réelle d'un argument dans le paramètre formel de la fonction. Dans ce cas, les modifications apportées au paramètre à l'intérieur de la fonction n'ont aucun effet sur l'argument.

Dans la méthode d'appel par valeur, la fonction appelée crée ses propres copies des valeurs d'origine qui lui sont envoyées. Toutes les modifications apportées se produisent sur la copie des valeurs de la fonction et ne sont pas répercutées sur la fonction appelante.

Considérons l'exemple suivant pour échanger (permuter) deux entiers:

```
//définition d'une fonction pour échanger deux valeurs entières
void echanger(int x, int y) {
   int temp; //une variable pour sauvegarder temporairement la valeur de l'une des
↪  deux variables
```

```
    temp = x; //sauvegarder temporairement la valeur de x dans temp
    x = y;    //mettre y dans x
    y = temp; //mettre x dans y
}
```

Maintenant, appelons la fonction echanger() en passant des valeurs réelles comme dans l'exemple suivant:

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

//déclaration de la fonction echanger

void echanger(int x, int y);

int main () {
   //les variables locales
   int a = 10;
   int b = 20;

   cout << "Avant l'échange, la valeur de a = " << a << endl;
   cout << "Après l'échange, la valeur de b = " << b << endl;

   //appel de la fonction echanger
   echanger(a, b);

   cout << "Après l'échange, la valeur de a = " << a << endl;
   cout << "Avant l'échange, la valeur de b = " << b << endl;

   return 0;
}

void echanger(int x, int y) {
   int temp; //une variable pour sauvegarder temporairement la valeur de l'une de
↪   deux variables

   temp = x; //sauvegarder temporairement la valeur de x dans temp
   x = y;    //mettre y dans x
   y = temp; //mettre x dans y
   return;
}
```

La compilation et l'exécution du code ci-dessus produisent le résultat suivant:

```
Avant l'échange, la valeur de a = 10
Avant l'échange, la valeur de b = 20
Après l'échange, la valeur de a = 10
Après l'échange, la valeur de b = 20
```

Ce qui montre qu'il n'y a pas de changement dans les valeurs bien qu'elles aient été modifiées à l'intérieur de la fonction.

3.5.2 Appel par référence

La méthode d'appel par référence consiste à transmettre des arguments à une fonction en copiant leur références dans les paramètres formels. À l'intérieur de la fonction, la référence est utilisée pour accéder à l'argument réel utilisé dans l'appel. Cela signifie que les modifications apportées au paramètre affectent l'argument passé.

Pour passer la valeur par référence, la référence d'argument est passée aux fonctions comme n'importe quelle autre valeur. Par conséquent, vous devez déclarer les paramètres de la fonction en tant que types de référence comme dans la fonction echanger() suivante, qui échange les valeurs des deux variables entières pointées par ses arguments.

```
void echanger(int &x, int &y) {
   int temp; //une variable pour sauvegarder temporairement la valeur de l'une des
↪  deux variables

   temp = x; //sauvegarder temporairement la valeur de x dans temp
   x = y;    //mettre y dans x
   y = temp; //mettre x dans y
   return;
}
```

Nous appelons la fonction echanger() en passant des valeurs par référence comme dans l'exemple suivant:

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

// déclaration de la fonction echanger

void echanger(int &x, int &y);

int main () {
   //les variables locales
   int a = 10;
   int b = 20;

   cout << "Avant l'écahnge, la valeur de a = " << a << endl;
   cout << "Après l'échange, la valeur de b = " << b << endl;

   //appel de la fonction echanger
   echanger(a, b);

   cout << "Après l'échange, la valeur de a = " << a << endl;
   cout << "Avant l'écahnge, la valeur de b = " << b << endl;

   return 0;
}

void echanger(int &x, int &y) {
   int temp; //une variable pour sauvegarder temporairement la valeur de l'une des
↪  deux variables

   temp = x; //sauvegarder temporairement la valeur de x dans temp
   x = y;    //mettre y dans x
   y = temp; //mettre x dans y
   return;
```

```
}
```

Lorsque le code ci-dessus est exécuté, il produit le résultat suivant:

```
Avant l'écahnge, la valeur de a = 10
Avant l'écahnge, la valeur de b = 20
Après l'échange, la valeur de a = 20
Après l'échange, la valeur de b = 10
```

3.5.3 Appel par pointeur

La méthode d'appel par pointeur consiste à transmettre des arguments à une fonction en copiant l'adresse d'un argument dans le paramètre formel. À l'intérieur de la fonction, l'adresse est utilisée pour accéder à l'argument réel utilisé dans l'appel. Cela signifie que les modifications apportées au paramètre affectent l'argument passé.

Pour passer la valeur par pointeur, les pointeurs d'argument sont passés aux fonctions comme n'importe quelle autre valeur. Par conséquent, vous devez déclarer les paramètres de la fonction en tant que types pointeur comme dans la fonction echanger() suivante, qui échange les valeurs des deux variables entières pointées par ses arguments.

```
//définition de la fonction echanger
void swap(int *x, int *y) {
   int temp;
   temp = *x; //sauvegerder temporairement la valeur à l'adresse x
   *x = *y; //mettre y dans x
   *y = temp; //mettre x dans y

   return;
}
```

Maintenant, nous appelons la fonction echnager() en passant des valeurs par pointeur comme dans l'exemple suivant:

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

//déclaration de la fonction echanger

void echanger(int *x, int *y);

int main () {
   //les variables locales
   int a = 10;
   int b = 20;

   cout << "Avant l'écahnge, la valeur de a = " << a << endl;
   cout << "Après l'échange, la valeur de b = " << b << endl;

   //appel de la fonction echanger
   echanger(&a, &b);

   cout << "Après l'échange, la valeur de a = " << a << endl;
```

```
   cout << "Avant l'écahnge, la valeur de b = " << b << endl;

   return 0;
}

void echanger(int *x, int *y) {
   int temp; //une variable pour sauvegarder temporairement la valeur de l'une des
↪  deux variables

   temp = *x; //sauvegarder temporairement la valeur de x dans temp
   *x = *y;    //mettre y dans x
   *y = temp; //mettre x dans y
   return;
}
```

Lorsque le code ci-dessus est exécuté, il produit le résultat suivant:

```
Avant l'écahnge, la valeur de a = 10
Avant l'écahnge, la valeur de b = 20
Après l'échange, la valeur de a = 20
Après l'échange, la valeur de b = 10
```

3.6 Les arguments par défaut

Pour appeler une fonction sans spécifier tous ses arguments, la fonction affecte une valeur par défaut à un paramètre qui n'a pas d'arguments dans l'appel de fonction. Les valeurs par défaut sont spécifiées lors de la déclaration de la fonction. Le compilateur sait à partir du prototype combien d'arguments une fonction utilise pour l'appeler.
Exemple:

```
float somme(float x, float y, float z=0.25);

float somme(float x, float y, float z){
    return x + y + z;
}
```

Un appel ultérieur de la fonction somme:

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

float somme(float x, float y, float z=0.25);

int main () {
   float a=2.5;
   float b=3.75;
   float s = somme(a,b);
   cout << "somme = " << s;
   return 0;
}

float somme(float x, float y, float z){
return x+y+z;
}
```

passe la valeur 2.5 à x, 3.75 à y et laisse la fonction utiliser la valeur par défaut de 0.25 pour z. Le résulat sera donc: s = 2.5 + 3.75 + 0.25 = 6.5

L'appel suivant de la fonction somme:

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

float somme(float x, float y, float z=0.25);

int main () {
   float a=2.5;
   float b=3.75;
   float c=0.5;
   float s = somme(a,b,c);
   cout << "somme = " << s;
   return 0;
}

float somme(float x, float y, float z){
return x+y+z;
}
```

passe la valeur 2.5 à x, 3.75 à y, et 0.5 à z en écrasant sa valeur par défaut de 0.25. Le résulat sera donc: s = 2.5 + 3.75 + 0.5 = 6.75

3.7 La fonction en ligne (inline)

La fonction inline (en ligne) est un concept puissant couramment utilisé avec les classes. Pour toute fonction inline, le compilateur place une copie du code de cette fonction à chaque point où la fonction est appelée au moment de la compilation.

Pour intégrer une fonction en ligne, placez le mot-clé inline avant le nom de la fonction et définissez la fonction avant tout appel à la fonction. Le compilateur peut ignorer le qualificateur en ligne si la fonction a plus d'une ligne de code.

Voici un exemple qui utilise la fonction inline pour calculer le maximum de deux nombres entiers:

```
#include <iostream>

using namespace std;

//définition de la fonction inline max
inline int max(int x, int y) {
   return (x > y)? x : y;
}

int main() {
   cout << "max (5,10): " << max(5,10) << endl;
   cout << "max (0,20): " << max(0,20) << endl;
   cout << "max (3,0): " << max(3,0) << endl;

   return 0;
}
```

La compilation et l'exécution du code ci-dessus produisent le résultat suivant:

```
max (5,10): 10
max (0,20): 20
max (3,0): 3
```

Quelques points importants à noter sur les fonction inline:

- La fonction est rendue en ligne en mettant le mot inline au début.
- La fonction inline doit être déclarée avant la fonction main().
- Il n'a pas de prototype de fonction.
- Seul un code plus court est utilisé dans la fonction inline. Si un code plus long est mis dans une fonction inline, le compilateur ignore la requête et la fonction sera exécutée comme une fonction normale.

3.8 Variable locale et variable globale

Une portée est une zone d'un programme, et il y a généralement trois endroits où une variable peut être déclarée.

- Une variable locale est appelée dans une fonction ou un bloc?
- Lors de la définition d'une fonction, les paramètres sont appelés paramètres formels.
- Une variable globale est définie en dehors d'une fonction.

3.8.1 Variable locale

Les variables déclarées dans le corps d'une fonction sont évaluées uniquement à l'intérieur de la fonction. La partie du programme où les variables sont conservées en mémoire est appelée portée de la variable. La portée d'une variable locale est la fonction dans laquelle elle est définie. Les variables peuvent être locales à une fonction ou à une instruction composée.

Les variables locales ne peuvent être utilisées que par des instructions au sein de la fonction.
Exemple:

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

int main() {
   //variables locales
   int a, b;
   int c;

   //initilisation
   a = 10;
   b = 20;
   c = a + b;
```

```
    cout << c << " = " << c;
    return 0;
}
```

Sortie du programme:

```
c = 30
```

3.8.2 Variable globale

Une variable déclarée en dehors d'une fonction s'appelle une variable globale. La portée de ces variables s'étend jusqu'à la fin du programme. Ces variables sont disponibles dans toutes les fonctions suivant la déclaration.
Exemple:

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

//déclaration d'une variable globale
int g;

int main() {
    //déclaration des variables locales
    int a, b;

    //initialization des variables locales
    a = 10;
    b = 20;
    g = a + b;

    cout << g << " = " << g;
    return 0;
}
```

```
g = 30
```

3.8.3 Opérateur de résolution de portée (::)

Un programme peut avoir une variable globale qui porte le même nom (identifiant) qu'une variable locale à l'intérieur d'une fonction. Pour accéder à la variable globale avec le même nom, vous devrez utiliser l'opérateur de résolution de portée (::).

L'opérateur de résolution de portée peut être utilisé à la fois comme unaire et comme binaire. Vous pouvez utiliser l'opérateur de portée unaire si un nom de portée d'espace de noms ou de portée globale est masqué par une déclaration particulière d'un nom équivalent dans un bloc ou une classe. Par exemple, si vous avez une variable globale de nom ma_var et une variable locale de nom ma_var, pour accéder à ma_var globale, vous devrez utiliser l'opérateur de résolution de portée.

Exemple:

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

//variable globale
int ma_var = 0;

int main(void) {

   //variable locale portant le même nom que la variable locale
   int ma_var = 0;

   ::ma_var = 1; //affecter 1 à la variable globale
    ma_var = 2; //affecter 2 à la variable locale

   cout << "La variable globale a = " << ::ma_var << endl;
   cout << "La variable locale a = " << ma_var << endl;

   return 0;
}
```

```
La variable globale a = 1
La variable locale a = 2
```

La déclaration de ma_var dans la fonction main masque ma_var déclarée globalement. L'instruction ::ma_var accède à la variable nommée ma_var déclarée dans la portée globale.

3.9 Classes de stockage

Une classe de stockage définit la portée (visibilité) et la durée de vie des variables et/ou des fonctions dans un programme C++. Ces spécificateurs précèdent le type qu'ils modifient. Les classes de stockage suivantes peuvent être utilisées dans un programme C++:

- automatique
- registre
- statique
- externe
- mutable

3.9.1 Stockage automatique

La classe de stockage automatique est la classe de stockage par défaut pour toutes les variables locales.

Exemple:

```
{
   int x;
   auto int x;
}
```

L'exemple ci-dessus définit deux variables avec la même classe de stockage, auto ne peut être utilisé que dans des fonctions, c'est-à-dire des variables locales.

3.9.2 Stockage de registre

La classe de stockage de registre est utilisée pour définir des variables locales qui doivent être stockées dans un registre au lieu de la RAM. Cela signifie que la variable a une taille maximale égale à la taille du registre (généralement un mot) et que l'opérateur unaire & ne peut pas lui être appliqué (car il n'a pas d'emplacement mémoire).
Exemple:

```
{
   register int x;
}
```

Le registre ne doit être utilisé que pour les variables nécessitant un accès rapide telles que les compteurs. Notez également que la définition d'un registre n'implique pas que la variable soit stockée dans un registre. Cela signifie qu'il peut être stocké dans le registre, sous réserve des limitations matérielles et de mise en œuvre.

3.9.3 Stockage statique

La classe de stockage statique demande au compilateur de conserver une variable locale existante pendant la durée de vie du programme au lieu de la créer et de la détruire à chaque fois qu'elle entre et sort de la portée. Par conséquent, rendre les variables locales statiques permet de conserver leurs valeurs entre les appels de la fonction.

Le modificateur statique peut également être appliqué aux variables globales. Lorsque cela est fait, la portée de cette variable est restreinte au fichier dans lequel elle est déclarée.

En C++, lorsque le mot-clé static est utilisé sur un membre de données d'une classe, une seule copie de ce membre est partagée par tous les objets de sa classe.

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

//déclaration de la fonction fonc
void fonc(void);


static int j = 5; //une variable globale statique

main() {

   while(j--) {

      fonc();
```

```
   }

   return 0;
}

//définition de la fonction fonc

void fonc(void) {
   static int i = 5; //une variable locale statique
   i++;
   cout << "i = " << i ;
   cout << " et j = " << j << endl;
}
```

La compilation et l'exécution du code ci-dessus produisent le résultat suivant:

```
i = 6 et j = 4
i = 7 et j = 3
i = 8 et j = 2
i = 9 et j = 1
i = 10 et j = 0
```

3.9.4 Stockage externe

La classe de stockage externe est utilisée pour fournir des références aux variables globales qui apparaissent dans tous les fichiers de votre programme. L'utilisation du mot-clé extern ne vous permettra pas d'initialiser la variable, car le nom de la variable pointe simplement vers l'endroit où elle a été précédemment définie.

Si vous avez plusieurs fichiers et que vous définissez une variable ou une fonction globale qui est également utilisée dans d'autres fichiers, utilisez extern dans un autre fichier pour indiquer une référence à la variable ou à la fonction définie.

Le modificateur extern est le plus couramment utilisé lorsque deux fichiers ou plus partagent la même variable ou fonction globale, comme décrit dans l'exemple suivant.

```
premier fichier: fichier.cpp
#include <iostream>
using namespace std;

int a;
extern void afficher();

main() {
   a = 10;
   afficher();
}

//le fichier extèrne
deuxième fichier: fichier_ext.cpp

#include <iostream>
//une variable globale déclarée comme extèrne
extern int a;
```

```
void afficher(void) {
    cout << "a = " << a << endl;
}
```

Ici, le mot-clé extern est utilisé pour la variable a du fichier_ext.cpp dans fichier.cpp.

3.9.5 Stockage mutable

Le spécificateur mutable s'applique uniquement aux objets de classe, et ne sera pas abordé en détails dans ce support de cours. Il permet à un membre d'un objet de remplacer la fonction membre const. Autrement dit, un membre mutable peut être modifié par une fonction membre const.

3.10 Conclusion

Au lieu d'écrire un programme volumineux sous la forme d'une longue séquence d'instructions, nous pouvons l'écrire sous forme de plusieurs petites fonctions, chacune effectuant une tâche particulière.

Une fonction est une partie réutilisable d'un programme, parfois appelée sous-programme. Ce chapitre a couvert la syntaxe pour définir et appeler des fonctions en C++, le passage de paramètres, les différents types de fonctions et la portée des variables.

Dans le chapitre suivant, nous présenterons une autre manière plus sophistiquée de réutiliser et d'optimiser le code à l'aide de types composés en mettant l'accent sur les tableaux et les chaînes de caractères.

Série d'exercices N°3

Exercice 3.1 Écrire une fonction qui calcule le double d'un nombre réel.

Exercice 3.2 Écrire une fonction inline pour trouver le minimum de deux nombres entiers.

Exercice 3.3 Écrire une fonction pour vérifier si un nombre est premier.

Exercice 3.4 Écrire une fonction pour trouver le plus grand diviseur commun (PGDC).

Exercice 3.5 Écrire une fonction pour renvoyer la valeur absolue d'un nombre.

Exercice 3.6 Écrire une fonction "triSelection" qui prend un tableau d'entiers en entrée et utilise le tri par sélection pour trier les éléments du tableau par ordre croissant. Utilisez un paramètre par référence pour le tableau.

Exercice 3.7 Écrire une fonction nommée "puissance" qui prend deux nombres (base et exposant) en entrée et renvoie la valeur de la puissance. Utilisez cette fonction pour calculer et afficher la valeur de 2^3 (2 élevé à la puissance 3) et 5^2 (5 élevé à la puissance 2).

CHAPITRE

4

Les tableaux et les chaînes de caractères

4.1 Introduction

Les types de données simples ne peuvent stocker qu'une seule valeur à la fois. Un type de données structuré est un type dans lequel chaque élément de données est une collection d'autres éléments de données. Un type de données structuré utilise un identifiant unique (nom) pour l'ensemble de la collection. Le but des types de données structurées est de regrouper des données liées de différents types afin qu'elles soient facilement accessibles à l'aide du même identifiant. De même, vous pouvez récupérer les données collectivement ou afficher chaque composant (élément) de la collection individuellement. De plus, les types de données structurés sont des types de données définis par l'utilisateur avec des éléments divisibles qui peuvent être utilisés individuellement ou en tant qu'entité unique selon les besoins. Ce chapitre couvre deux nouveaux aspects principaux en C++: les tableaux et les chaînes de caractères.

4.2 Les tableaux

Un tableau est une collection d'éléments de données du même type, décrits par un nom unique, et chaque élément du tableau est référencé par le nom du tableau et son index.

Supposons que vous souhaitiez stocker les notes d'un groupe d'étudiants dans une classe. Au lieu de créer une variable distincte pour chaque note, vous pouvez utiliser un tableau pour organiser et gérer ces données de manière structurée. Chaque note d'un étudiant particulier occupe une position spécifique dans le tableau, ce qui vous permet de les parcourir, de les analyser et de les manipuler efficacement.

4.2.1 Tableaux unidimensionnels

Au lieu de déclarer par exemple des variables individuelles de type entier, telles que x1, x2,...,x100, etc., vous déclarez une variable tableau telle que T et utilisez T[0], T[1], etc., pour représenter les variables individuelles. Un élément spécifique dans un tableau est accessible par un indice.

4.2.1.1 Déclaration

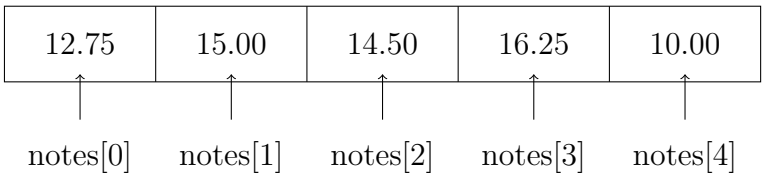
La syntaxe pour déclarer un tableau unidimensionnel en C++ est:

```
type nom_tableau[nombre_éléments];
```

Exemple:

```
float notes[5];
```

Comme illustré dans le schéma suivant, le tableau notes a une taille de 5, ce qui signifie qu'il contient 5 éléments. Chaque élément est identifié par un indice, allant de 0 à 4, car les indices dans C++ commencent à 0. Chaque case du tableau peut contenir une valeur de type float, représentant la note d'un étudiant.



La figure 4.1 montre le format général d'un tableau unidimensionnel.

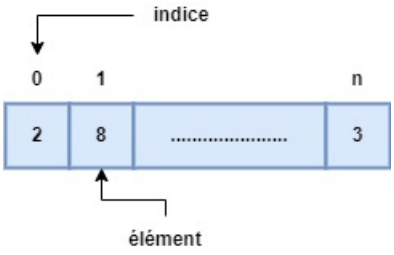


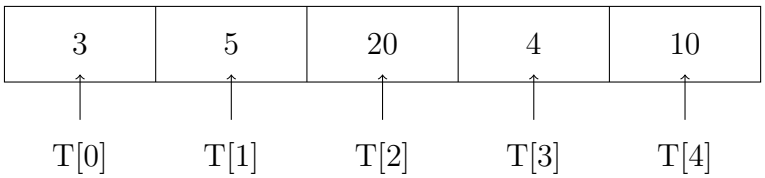
Figure 4.1: Tableau unidimensionnel

4.2.1.2 Initialisation

Les tableaux peuvent être initialisés dans leur déclaration. Pour initialiser un tableau, les éléments doivent être séparés par des virgules et entourés d'accolades.
Exemple:

```
int T[5] = {3,5,20,4,10};
```

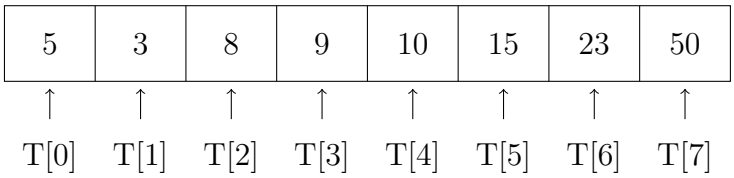
Le schéma suivant illustre cette déclaration.



Vous pouvez laisser la taille du tableau ouverte. Le compilateur compte la taille des tableaux.
Exemple:

```
int T[] = {5,3,8,9,10,15,23,50};
```

Cette déclaration est illustré dans le schéma suivant.



4.2.1.3 Référence aux éléments

À tout moment du programme où le tableau est visible, vous pouvez accéder à la valeur de chaque élément individuellement, la lire ou la modifier, comme s'il s'agissait d'une variable normale. La syntaxe pour accéder aux éléments du tableau est:

```
T[ind]
```

Où T est la nom du tableau, et ind est l'indice de l'élément dans le tableau. **Exemple**:

```
cout << notes[3]; //afficher la note de position 3
notes[3] = 15.50; //affecter 15.50 à l'élément qui occupe la position 3 dans le
↪  tableau
cin >> age[3]; //entrer (lire) la valeur de l'élément qui occupe la position 3
```

Il est courant d'utiliser une boucle for pour remplir un tableau, comme illustré dans l'exemple suivant:

```
float notes[10];
int i;

for (i = 0 ; i < 10; i++) {
    cin >> notes[i];
 }
```

4.2.1.4 Tableau unidimensionnel comme paramètre

À un moment donné, vous devrez peut-être passer un tableau en tant que paramètre à une fonction. En C++, vous ne pouvez pas transmettre tous les éléments du tableau par valeur en tant que paramètres de fonction, mais vous pouvez transmettre leurs adresses.

Par exemple, la fonction afficher() suivante accepte un "tableau d'entiers" nommé T comme paramètre formel.

```
void affichier(int T[])
```

Afin de passer à cette fonction un tableau déclaré comme:

```
int A[20] ;
```

Nous devons écrire un appel comme celui-ci:

```
afficher(A);
```

Voici un exemple complet:

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

void afficher(int T[], int taille) {

for (int i = 0; i < taille; i++)
    cout << T[i] << endl;
}

int main() {
int A[] = {5, 20, 30};
afficher(A,3);
return 0;
}
```

L'exécution du code précédent produira le résultat suivant:

```
5
20
30
```

4.2.1.5 Opérations de base

A. Parcourir un tableau

Le moyen le plus simple d'itérer dans un tableau consiste à utiliser une boucle for, comme illustré dans l'exemple suivant.

```
void afficher(int tab[], int taille) {
cout << "Les éléments du tableau sont:\n";
for(int i = 0; i < taille; i++)
    cout << tab[i];
}
```

B. Lire un tableau

Pour introduire(lire) les éléments d'un tableau élément par élément, la boucle for et la fonction cin sont utilisé, comme illustré dans l'exemple suivant:

```
void lire(int tab[], int taille) {
    cout << "Entrer les éléments du tableau:";
for(int i = 0; i < taille; i++)
    cin >> tab[i];
 }
```

4.2.1.6 Recherche dans un tableau

A. Recherche linéaire

La recherche linéaire est définie comme un algorithme de recherche séquentielle qui commence à une extrémité et parcourt chaque élément d'une liste jusqu'à ce que l'élément souhaité soit trouvé, sinon la recherche se poursuit jusqu'à la fin de l'ensemble de données. C'est l'algorithme de recherche le plus simple.

Exemple:

```
//x est la valeur à rechercher dans le tableau tab
//taille est le nombre d'éléménts du tableau

void rechLin(int tab[], int taille, int x){

for(int i = 0; i < taille; i++)
{
    if(tab[i] == x)
    {
        cout << "La valeur recherchée se trouve à l'indice: " << i;
        return;
    }
}
cout << "La valeur recherchée n'existe pas" << endl;
}
```

B. Recherche binaire

La recherche binaire est une méthode pour trouver l'élément requis dans un tableau trié en divisant de manière répétée par deux le tableau et en recherchant dans la moitié.

Cette méthode se fait en commençant par tout le tableau. Ensuite, il est divisé par deux. Si la valeur de données requise est supérieure à l'élément au milieu du tableau, la moitié supérieure du tableau est prise en compte. Sinon, la moitié inférieure est prise en compte. Cela se fait en continu jusqu'à ce que la valeur de données requise soit obtenue ou que le tableau restant soit vide.

La figure ci-dessous présente un exemple du processus de recherche de l'élément 6 dans un tableau d'entiers à l'aide de l'algorithme de recherche binaire.

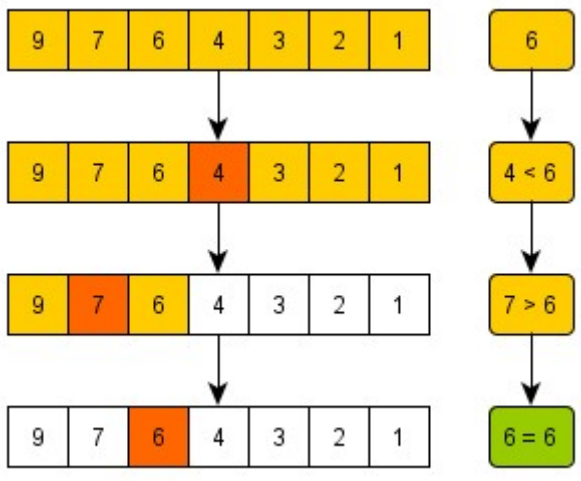


Figure 4.2: Processus de recherche binaire: Exemple de recherche de l'élément 6 dans un tableau d'entiers

Exemple:

```
int rechBin(int tab[], int taille, int x){
int mil, inf = 0, sup = taille - 1, existe = 0;

while((inf <= sup) && !(existe)) {
    mil =(inf + sup)/2; //récupérer l'indice du milieu
    if(x > tab[mil]) //comparer la valeur de x avec celle du milieu
        inf = mil + 1; //si x > tab[mil] alors prendre la moitié supérieure du
↪  tableau
    else if(x < tab[mil])
        sup = mil - 1; //si x < tab[mil] alors prendre la moitié inférieure du
↪  tableau
    else
        existe++; //la variable booléenne existe prend la valeur 1 s'il l'élément x
↪  se se trouve dans le tableau
}
if(existe)
    return mil; //renvoyer l'indice de l'élément x si'il existe dans le tableau
else
    return(-1); //renvoyer -1 si x n'existe pas
}
```

4.2.1.7 Tri d'un tableau

Un tableau trié est un tableau dans lequel chacun des éléments est trié dans un ordre tel que numérique, alphabétique, etc. Il existe de nombreux algorithmes pour trier un tableau numérique, tels que le tri à bulles, le tri par insertion, le tri par sélection, le tri par fusion, le tri rapide, le tri par tas, etc.

A. Tri à bulles

Le tri à bulles est l'algorithme de tri le plus simple qui fonctionne en échangeant à plusieurs reprises les éléments adjacents s'ils sont dans le mauvais ordre. Cet algorithme n'est pas adapté aux grands ensembles de données car sa complexité temporelle est assez élevée.
Exemple:

```
void triBulles(int tab[], int taille)
{
int i, j, temp;
for(i = 0; i < taille - 1; i++) //tri
{
    for(j = 0; j < (taille-1-i); j++)

    if(tab[j] > tab[j+1]) {
        temp = tab[j]; //échanger les éléments
        tab[j] = tab[j+1];
        tab[j+1] = temp;
    }
}
}
```

B. Tri par insertion

La technique du tri par insertion prend le deuxième élément, le compare au premier et le place au bon endroit. Suivez ensuite ce processus pour les éléments suivants: Comparez chaque élément avec tous les éléments précédents et placez ou collez l'élément dans la position appropriée.
Exemple:

```
void triInsert(int tab[], int taille)
{
int i, j, temp;
for(i = 1; i < taille; i++) {  //tri
    temp = tab[i];
    j = i-1;

    while((temp < tab[j]) && (j >= 0)) {
        tab[j+1] = tab[j];
        j--;
    }
    tab[j+1] = temp;
    }
}
```

C. Tri par sélection

Le tri par sélection est une méthode de tri qui produit un tableau trié. Il le fait en trouvant à plusieurs reprises le plus petit élément du tableau et en l'interchangeant avec l'élément au début de la partie non triée.
Exemple:

```
void triSelect(int tab[], int taille)
{
int i, j, temp, petit;
for(i = 0; i < taille - 1; i++){
    petit = i;
    for(j = i+1; j < taille; j++) //trouver le plus petit élément
        if(tab[j] < tab[petit])
            petit = j;

    if(petit != i) {
        temp = tab[i]; //échanger les éléments
        tab[i] = tab[petit];
        tab[petit] = temp;
    }
}
}
```

4.2.2 Tableaux bidimensionnels

Il s'agit d'une collection d'éléments de données du même type de données disposés en lignes et en colonnes (c'est-à-dire en deux dimensions). La figure suivante montre la structure générale d'un tableau à deux dimensions.

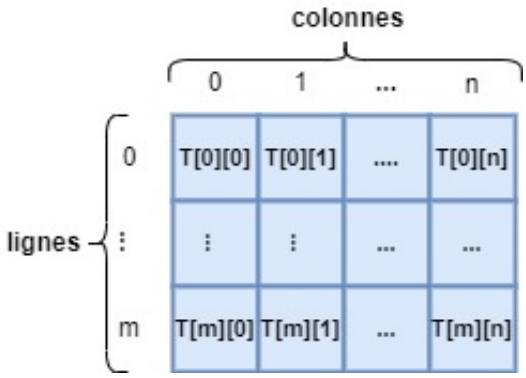


Figure 4.3: Structure générale d'un tableau bidimensionnel

4.2.2.1 Déclaration

La syntaxe pour déclarer un tableau à deux dimensions est:

```
type nom_tab[nb_lignes][nb_colonnes];
```

Exemple:

```
int A[3][5];
```

Figure 4.4: Exemple d'un tableau bidimensionnel

4.2.2.2 Initialisation

Un tableau à deux dimensions peut être initialisé avec la déclaration. Pour l'initialisation d'un tableau à deux dimensions, les éléments de chaque ligne sont entourés d'accolades et séparés par des virgules. Toutes les lignes sont entourées d'accolades.
Exemple:

```
int A[4][3] = {{5, 10, 15}, {20, 25, 30}, {35, 40, 45}, {50, 55, 60}};
```

4.2.2.3 Référence aux éléments

Pour accéder aux éléments d'un tableau à deux dimensions, nous avons besoin d'une paire d'indices: un pour la position de la ligne et un pour la position de la colonne. Le format est aussi simple que: nom_tab[indiceLigne][indiceColonne].

Exemples:

```
cout << A[3][2]; //afficher l'élément du tableau qui occupe la ligne 3 et la colonne
↪  2
A[0][4] = 15; //affecter 15 à l'élément du tableau qui occupe la ligne 0 et la
↪  colonne 4
cin >> A[4][2]; //lire la valeur de l'élément du tableau qui occupe la ligne 4 et la
↪  colonne 2
```

4.2.2.4 Lecture

Pour remplir un tableau à deux dimensions, nous utilisons deux boucles for, une pour lignes et l'autres pour les colonnes.
Exemple:

```
int mat[3][5], lig, col;
for(lig = 0; lig < 3; lig++)
           for (col = 0; col < 5; col++)
         cin >> mat[lig][col];
```

4.2.2.5 Tableau bidimensionnel comme paramètre

Les tableaux à deux dimensions peuvent être passés en tant que paramètres à une fonction, et ils sont passés par référence. Lors de la déclaration d'un tableau à deux dimensions en tant que paramètre formel, nous pouvons omettre la taille de la première dimension, mais pas la seconde; c'est-à-dire que nous devons spécifier le nombre de colonnes. Par exemple:

```
void afficher(int A[][3], int N, int M)
```

Pour passer à cette fonction un tableau déclaré comme suit:

```
int A[4][3];
```

Nous devons écrire un appel comme celui-ci:

```
afficher(tab);
```

Voici un exemple complet:

```
#include <iostream>
using namespace std;
void print(int A[][3], int N, int M) {
for (lig = 0; lig < N; lig++)
    for (col = 0; col < M; C++)
        cout << A[lig][col];
}
int main () {
int mat[4][3] ={{5, 10, 15}, {20, 25, 30}, {35, 40, 45}, {50, 55, 60}};
afficher(mat,4,3);
return 0;
}
```

4.2.2.6 Opérations de base sur les tableaux bi-dimensionnels

A. Lire un tableau

Afin de stocker des valeurs dans un tableau bidimensionnel C++, le programmeur doit spécifier le nombre de lignes et le nombre de colonnes. Pour accéder à chaque emplacement individuel pour stocker une valeur, l'utilisateur doit fournir le nombre exact de lignes et de colonnes.
Exemple:

```
void lireTab(int A[][10], int M, int N)
{
for(int lig = 0; lig < M; lig++)
    for(int col = 0; col < N; col++)
    {
        cin >> A[lig][col];
    }
}
```

B. Afficher le contenu d'un tableau

La manière habituelle d'afficher le contenu d'un tableau MxN nécessite deux boucles for pour itérer sur toutes les lignes et colonnes. La première boucle for itère sur les lignes, et la seconde itère sur les colonnes.
Exemple:

```
void afficher(int A[][10], int M, int N) {

for(int lig = 0; lig < M; lig++){
    for(int col = 0; col < N; col++) {
        cout << A[lig][col] << endl;
        }
     }
}
```

C. Initialisation d'un tableau

Vous pouvez créer une matrice en définissant sa taille et en utilisant des boucles pour assigner des valeurs à ses éléments.
Exemple:

```
const int ligs = 3;
const int cols = 3;
int matrice[ligs][cols];

// Initialisation de la matrice avec des valeurs

for (int i = 0; i < ligs; ++i) {
    for (int j = 0; j < cols; ++j) {
        matrice[i][j] = i * cols + j;
    }
}
```

D. Accès aux éléments d'une matrice

L'accès aux éléments individuels d'une matrice se fait en utilisant les indices correspondant à leurs positions en ligne et en colonne.
Exemple:

```
int element = matrice[1][2];  // Accès à l'élément à la deuxième ligne, troisième
↪  colonne
```

E. Multiplication par un scalaire

Il est possible de réaliser une multiplication scalaire en multipliant chaque élément de la matrice par un nombre donné.
Exemple:

```
int scalaire = 2;

for (int i = 0; i < ligs; ++i) {
    for (int j = 0; j < cols; ++j) {
        matrice[i][j] *= scalaire;
    }
}
```

4.3 Les chaines de caractères

En C++, une chaîne de caractères est typiquement définie comme une séquence ordonnée de caractères, pouvant inclure des lettres, des chiffres, des symboles, et d'autres caractères, qui est utilisée pour stocker et manipuler des données textuelles dans un programme.

C ++ fournit les deux types de représentation des chaînes de caractères:

- La chaîne de caractères de style C.
- La classe de chaîne introduit avec le C++ standard.

4.3.1 Chaînes de caractères de style C

La chaîne de caractères de style C provient du langage C et continue d'être prise en charge dans C++. Cette chaîne est en fait un tableau unidimensionnel de caractères qui se termine par un caractère nul '\0'. Ainsi, une chaîne terminée par un caractère nul contient les caractères qui composent la chaîne suivis d'un caractère nul.

4.3.1.1 Déclaration

En programmation C, la collection de caractères est stockée sous forme de tableaux. Ceci est également pris en charge dans la programmation C++. C'est pourquoi on l'appelle C-strings.

Les chaînes C sont des tableaux de type char terminés par un caractère nul, c'est-à-dire \0 (la valeur ASCII du caractère nul est 0).

La syntaxe de déclaration d'une chaîne de caractère style C est la suivante:

```
char nom_str[nombre de caractères];
```

Exemple:

```
char nom[30];
```

Dans l'exemple ci-dessus, nom peut être utilisé pour stocker une chaîne de 29 caractères.

4.3.1.2 Initialisation

Une chaîne peut être initialisée à une valeur constante lorsqu'elle est déclarée.
Exemple:

```
char ch[] = "Bonjour";
```

Ou

```
char ch[]={'B','o','n','j','o','u','r', '\0'};
```

Ici. 'B' sera stocké dans ch[0], 'o' dans ch[1] et ainsi de suite. \0 à la fin de la chaîne. Mais, si c'est fait caractère par caractère, alors nous devons l'insérer à la fin de la chaîne.

4.3.1.3 Lecture

Pour lire une chaîne sans blancs, cin peut être utilisée:

```
cin >> str;
```

Pour lire une chaîne avec des blancs, cin.getline() ou gets() peuvent être utilisées.

```
cin.getline(str,80);
```

Ou

```
gets(str);
```

4.3.1.4 Ecriture

Pour écrire (imprimer) une chaîne de caractère en C++, les fonctions cout et puts() sont utilisées.

```
cout << str;
```

Ou

```
puts(str);
```

Remarque: Pour gets() et puts(), le fichier d'en-tête <cstdio> (formellement stdio.h) doit être inclus. La fonction puts() peut être utilisée pour afficher uniquement des chaînes. Il faut un saut de ligne après l'impression de la chaîne.

Le tableau 4.1 montre la différence entre les fonctions cin et gets().

cin	gets()
Il peut être utilisé pour saisir une valeur de n'importe quel type de données.	Il peut être utilisé pour saisir une valeur de n'importe quel type de données. Il peut être utilisé pour saisir une chaîne de caractères.
Il prend l'espace blanc, la tabulation ou un caractère de nouvelle ligne comme caractère de fin de la chaîne.	Il ne prend pas l'espace blanc, la tabulation ou un caractère de nouvelle ligne, comme caractère de fin de la chaîne.
Il nécessite le fichier d'en-tête <iostream>	Il nécessite les fichiers d'en-tête <iostream>et <cstdio>
Exemple: char S[80]; cout <<"Entrer une chaîne:"; cin>>S;	Example: char S[80]; cout <<"Entrer une chaîne:"; gets(S);

Tableau 4.1: Différences entre cin et gets()

Les différences entre cout et puts() sont décrites dans le tableau 4.2:

cout	puts()
Il peut être utilisé pour afficher la valeur de n'importe quel type de données.	Il peut être utilisé pour afficher la valeur d'une chaîne.
Il ne prend pas de saut de ligne après l'affichage de la chaîne.	Il faut un saut de ligne après avoir affiché la chaîne.
Il nécessite le fichier d'en-tête iostream	Il nécessite le fichier d'en-tête<cstdio>
Example: char S[80] = "C++"; cout<<S<<S; Sortie: C++C++	Exemple: char S[80] = "C++"; puts(S); puts(S); Sortie: C++ C++

Tableau 4.2: Différences entre cout et puts()

4.3.1.5 Opérations de base

A. Nombre de mots dans une chaîne

Le nombre de mots dans une chaîne peut être calculé en comptant le nombre d'espaces et en ajoutant 1 comme dans la fonction suivante:

```
void nombre_mots(char S[]) {
int nb_mots = 0;
for(int i=0;S[i]!='\0';i++)
{
if (S[i]==' ') //vérification des espaces
    nb_mots++;
}
cout<<"Le nombre de mots = "<< nb_mots+1<<endl;
}
```

En invoquant cette fonction dans le code ci-dessous :

```
int main() {
    char maChaine[] = "   Ceci est un exemple  de    phrase  avec   des espaces    ";
    nombre_mots(maChaine);
    return 0;
}
```

Nous obtenons le résultat suivant:

```
Le nombre de mots = 9
```

B. Longueur d'une chaîne

La longueur ou la taille d'une chaîne est le nombre total de caractères qu'elle contient, et peut être calculé comme dans la fonction suivante:

```
int longChaine(char S[]) {
for(int i=0;S[i]!='\0';i++);
return i;
}
```

En appelant cette fonction dans le code suivant:

```
int main() {
    char maChaine[] = "Apprendre C++"; // Une chaîne de 13 caractères
    int taille = longChaine(maChaine); // Appel de la fonction pour calculer la
↪   longueur

    cout << "La longueur de la chaîne est : " << taille << " caractères." << endl;

    return 0;
}
```

Nous obtenons le résultat suivant:

```
La longueur de la chaîne est : 13
```

C. Copier le contenu d'une chaîne vers une autre

Cette opération copie les caractères de la chaîne source vers la chaîne cible caractère par caractère et ajoute le caractère '\0' à la fin de la chaîne cible. Cette opération peut être faite à l'aide de la fonction suivante:

```
void copier(char S1[], char S2[])
{
for(int i=0;S2[i]!='\0';i++)
    S1[i]=S2[i];

S1[i]='\0';
}
```

Lorsque cette fonction est utilisée comme illustré dans l'exemple suivant:

```
int main()
{
    char source[] = "Apprendre C++";
    char cible[14]; // On alloue 14 caractères pour la chaîne cible (y compris le
↪   caractère nul '\0')

    copier(cible, source, sizeof(cible));
```

```
    printf("La chaîne source est : %s\n", source);
    printf("La chaîne cible est : %s\n", cible);

    return 0;
}
```

Elle produit le résultat suivant:

```
La chaîne source est : Apprendre C++
La chaîne cible est : Apprendre C++
```

D. Concaténer deux chaînes

La concaténation consiste à ajouter une chaîne à la fin d'une autre chaîne. Cette opération peut être faite en ajoutant les caractères de la deuxième chaîne à la première un par un, comme dans la fonction suivante:

```
void concat(char S1[], char S2[]){
for(int l=0;S1[l]!='\0';l++);
for(int i=0;S2[i]!='\0';i++)
                S1[l++]=S2[i];
S1[l]='\0';
}
```

Quand nous utilisons cette fonction comme illustré dans l'exemple suivant :

```
int main()
{
    char chaine1[50] = "Apprendre ";
    char chaine2[] = "C++";

    concat(chaine1, chaine2);

    printf("La chaîne concaténée est : %s\n", chaine1);

    return 0;
}
```

Nous obtenons le résultat suivant:

```
La chaîne concaténée est : Apprendre C++
```

E. Inverser une chaîne

L'inversion d'une chaîne est la technique qui inverse ou modifie l'ordre d'une chaîne donnée de sorte que le dernier caractère de la chaîne devienne le premier caractère de la chaîne et ainsi de suite. Cette opération est décrite dans la fonction suivante:

```
void inverser(char S[], char inv[]) {
for(int C1=0; S[C1]!='\0'; C1++);
    C1--;
```

```
for(int C2=0;C1>=0;C2++,C1--)
    inv[C2]=S[C1];
    inv[C2]='\0';
}
```

Lorsque nous employons cette fonction de la manière présentée dans l'exemple suivant :

```
int main()
{
    char chaine[] = "Apprendre C++";
    char chaineInverse[50]; // Une chaîne pour stocker le résultat inversé.

    inverser(chaine, chaineInverse);

    printf("La chaîne inversée est : %s\n", chaineInverse);

    return 0;
}
```

Nous obtenons le résultat suivant:

```
La chaîne inversée est : ++C erdnerppA
```

4.3.2 La classe String en C++

C++ est basé sur le concept de la programmation orienté objet (POO); il permet de représenter la chaîne comme un objet de la classe C++ String (std::string). La classe vous permet de déclarer rapidement une variable de chaîne et d'y stocker n'importe quelle séquence de caractères.

4.3.3 Déclaration

Une chaîne C++ est un objet de la classe string, qui est défini dans le fichier d'en-tête <string> et qui se trouve dans l'espace des noms standard. La classe string a plusieurs constructeurs qui peuvent être appelés (explicitement ou implicitement) pour créer un objet string.

4.3.3.1 Initialisaion

L'intilisation d'une chaîne de caractères peut se faire de plusieurs façon comme illustré dans les exemples suivants:
Exemples:

```
string s1; //crée une chaîne C++ vide de longueur 0, égale à "" (constructeur par
↪  défaut)

string s2("bonjour"); //initialise la chaîne avec le littéral de chaîne C

string s3 = "bonjour" ; //initialise la chaîne avec le littéral de chaîne C

string s4(texte); //initialise la chaîne avec un tableau de caractères contenant une
↪  chaîne C
```

```
string s5(s2); //initialise la chaîne avec une autre chaîne C++

string s6 = s2 ; //initialise la chaîne avec une autre chaîne C++
```

4.3.3.2 Opérations de base

A. Longueur d'une chaîne

Vous pouvez obtenir la longueur d'une chaîne C++ à l'aide des fonctions membres de la classe de chaînes length() ou size() qui renvoient le nombre de caractères valides dans la chaîne (un entier positif ou nul).
Exemple:

```
string s = "apprendre C++";

cout << "La longueur de la chaîne est " << s.length() << endl;  // longueur = 14
```

B. Comparaisons de deux chaînes

Les objets de chaîne C++ peuvent être comparés à l'aide des opérateurs relationnels standard ==, !=, >, <, >= et <=. Une chaîne C++ peut être comparée à une autre chaîne C++ ou à une chaîne C valide, y compris un littéral de chaîne. Toutes ces expressions relationnelles se résolvent en valeurs booléennes true ou false.
Exemples:

```
if (s1 > s2) //comparer deux chaînes C++
{
     ...
}

if ("C++" == s2) //comparer le littéral de chaîne C et la chaîne C++
{
     ...
}

if (s3 != cstr) //comparer la chaîne C++ et le tableau contenant une chaîne C
{
     ...
}
```

C. Concaténation des chaînes

L'opérateur + peut être utilisé pour concaténer des chaînes C++. Les chaînes C++, les chaînes C et les littéraux de chaîne peuvent tous être concaténés dans n'importe quel ordre. Le résultat est un objet chaîne C++ qui peut être assigné à un autre objet chaîne C++.
Exemple:

```
string s1 = "apprendre";
string s2 = "C++";
```

```
s1 = s1 + " la programmation en " + s2;      // s1 contient maintenant "apprendre la
↪  programmation en C++"
```

4.4 Conclusion

Ce chapitre décrit deux structures de données complexes: les tableaux et les chaînes de caractères. Les tableaux sont utilisés pour stocker plusieurs valeurs dans une seule variable au lieu de déclarer une variable distincte pour chaque valeur. Cela a l'avantage de rationaliser votre code, comme la possibilité d'accéder de manière aléatoire à des données à des positions arbitraires et de récupérer et trier efficacement les données. Les chaînes sont utilisées pour stocker du texte, c'est-à-dire une collection d'octets dans des emplacements consécutifs pour faciliter la manipulation.

Série d'exercices N°4

Exercice 4.1 Écrire un programme C++ pour trouver le plus grand et le plus petit élément d'un tableau d'entiers donné.

Exercice 4.2 Écrire un programme C++ pour compter le nombre d'occurrences d'un élément donné dans un tableau d'entiers.

Exercice 4.3 Écrire un programme C++ pour inverser un tableau d'entiers.

Exercice 4.4 Écrire un programme C++ pour tester si une chaîne de caractère est palyndrome.

Exercice 4.5 Écrire un programme C++ pour compter le nombre d'occurrences d'un caractère dans une chaîne.

Exercice 4.6 Écrire un programme C++ qui demande à l'utilisateur de saisir 5 nombres entiers, stockez-les dans un tableau, puis triez les nombres dans l'ordre croissant et affichez le tableau trié.

Exercice 4.7 Écrire un programme C++ qui demande à l'utilisateur de saisir une liste de chaînes de caractères, stockez-les dans un tableau, puis trouvez et affichez la chaîne la plus longue dans le tableau.

CHAPITRE

5

Les fichiers

5.1 Introduction

Les fichiers sont utilisés pour stocker des données dans un périphérique de stockage de façon permanente. La gestion des fichiers fournit un mécanisme pour stocker la sortie d'un programme dans un fichier et pour effectuer diverses opérations dessus. Un flux est une abstraction qui représente un périphérique sur lequel des opérations d'entrée et de sortie sont effectuées. Un flux peut être représenté comme une source ou une destination de caractères de longueur indéfinie selon son utilisation. Dans ce chapitre, nous allons voir un ensemble de méthodes C++ pour la gestion des fichiers: lecture, écriture, gestion des erreurs, etc.

5.2 Définitions

Fichier: Un fichier informatique est une unité de stockage numérique qui contient des données, identifiable par un nom unique et pouvant prendre divers formats (texte, image, vidéo). Ils servent à stocker et organiser des informations de manière persistante sur des supports électroniques. Exemples: un fichier image: "photo.jpg", un fichier texte: "document.txt", un fichier vidéo: "video.mp4", un fichier de feuille de calcul: "tableau.xlsx", un fichier audio: "musique.mp3", un fichier exécutable: "programme.exe", etc.
Flux: Un "flux" fait référence à une séquence d'octets utilisée pour la transmission ou la manipulation de données. Il est couramment utilisé pour lire ou écrire des données à partir de sources telles que des fichiers, des connexions réseau ou des périphériques. Exemple: Soit un fichier appelé "Notes_Etudiants.txt". Pour lire les notes d'un étudiant spécifique, vous ouvrez le fichier et parcourez les lignes correspondantes, similaire à feuilleter un carnet de notes pour trouver des informations spécifiques. Lorsqu'un nouvel étudiant passe un examen, un flux d'écriture est employé pour ajouter une nouvelle ligne au fichier, enregistrant ainsi ses résultats.
Fichier texte: C'est un fichier qui stocke des informations en caractères ASCII. Dans un fichier texte, chaque ligne de texte se termine par un caractère spécial appelé caractère ou délimiteur EOL (End Of Line). Lorsque ce caractère EOL est lu ou écrit, certaines

conversions internes sont effectuées. Exemple: Un fichier texte simple peut contenir du texte brut, comme un document rédigé en utilisant un éditeur de texte.
Fichier binaire: Un fichier binaire est un type de fichier informatique qui enregistre des données sous une forme binaire, représentée par des séquences d'octets contenant des valeurs de 0 et de 1. À la différence des fichiers texte, les fichiers binaires ne contiennent généralement pas de texte intelligible par les humains ni de caractères de séparation de lignes. Ils sont adaptés au stockage d'une diversité de données, incluant des images, des vidéos, des programmes exécutables, des fichiers de base de données, et d'autres types d'informations. Par exemple, si vous avez un fichier binaire comme "mon_programme.exe", son contenu est fait de codes spéciaux que l'ordinateur comprend pour exécuter un programme. Mais si vous essayez de l'ouvrir avec un éditeur de texte, tout ce que vous verrez sera une série de caractères étranges et incompréhensibles. Cela contraste avec les fichiers texte, qui contiennent des mots et des phrases lisibles par les humains.

5.3 Classes pour l'opération de flux de fichiers

Dans les chapitres précédents, nous avons utilisé la bibliothèque standard iostream, qui fournit respectivement les méthodes cin et cout pour lire à partir de l'entrée standard et écrire sur la sortie standard.

Dans ce chapitre, nous allons appendre comment lire et écrire à partir d'un fichier. Cela nécessite une autre bibliothèque C++ standard appelée fstream, qui définit trois nouveaux types de données:

- ofstream: Ce type de données représente le flux de fichiers de sortie et est utilisé pour créer des fichiers et pour écrire des informations dans les fichiers.
- ifstream: Ce type de données représente le flux du fichier d'entrée et est utilisé pour lire les informations des fichiers.
- fstream: Ce type de données représente le flux de fichiers en général et possède les capacités de ofstream et ifstream, ce qui signifie qu'il peut créer des fichiers, écrire des informations dans des fichiers et lire des informations à partir de fichiers.

Pour effectuer le traitement de fichiers en C++, les fichiers d'en-tête <iostream> et <fstream> doivent être inclus dans votre fichier source C++.

5.3.1 Ouvrir un fichier

Un fichier doit être ouvert avant de pouvoir le lire ou y écrire. Les objets ofstream ou fstream peuvent être utilisés pour ouvrir un fichier en écriture. Tandis que l'objet ifstream est utilisé pour ouvrir un fichier à des fins de lecture uniquement.

Voici la syntaxe standard de la fonction open(), qui est membre des objets fstream, ifstream et ofstream.

```
void open(const char *nom_fichier, ios::mode_ouverture mode);
```

Ici, le premier argument spécifie le nom et l'emplacement du fichier à ouvrir et le second argument de la fonction open() définit le mode dans lequel le fichier doit être ouvert.

Une opération d'ouverture de fichier peut être combinée avec l'un des modes (flags) comme indiquée dans le tableau 5.1.

Mode	Signification
ios::app	Ajouter à la fin du fichier
ios::ate	Aller à la fin du fichier à l'ouverture
ios::binary	Ouvrir le fichier en mode binaire
ios::in	Ouvrir le fichier en lecture seule
ios::out	Ouvrir le fichier en écriture seulement
ios::nocreate	L'ouverture échoue si le fichier n'existe pas
ios::noreplace	L'ouverture échoue si le fichier existe déjà
ios::trunc	Supprimer le contenu du fichier s'il existe

Tableau 5.1: Modes d'ouverture d'un fichier

Tous ces modes peuvent être combinés à l'aide de l'opérateur OR (|). Par exemple, si nous voulons ouvrir le fichier exemple.bin en mode binaire pour ajouter des données, nous pouvons le faire par l'appel suivant à la fonction membre open():

```
fstream file;
file.open("exemple.bin", ios::out | ios::app | ios::binary);
```

Exemple 1: Créer et ouvrir un fichier en mode écriture

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
        fstream fichier;
        fichier.open("fichier.txt", ios::out);
        if (!fichier) {
                cout << "Fichier non créé !";
        }
        else {
                cout << "Fichier créé avec succès !";
                fichier.close();
        }
        return 0;
}
```

Exemple 2: Ouvrir un fichier un mode lecture

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
        fstream fichier;
        fichier.open("fichier.txt", ios::in);
        if (!fichier) {
                cout << "Ce fichier n'existe pas";
        }
        else {
                char ch;
```

```
                while (1) {
                        fichier >> ch;
                        if (fichier.eof())
                                break;

                        cout << ch;
                }

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

5.3.2 Fermer un fichier

Lorsqu'un programme C++ se termine, il vide automatiquement tous les flux, libère toute la mémoire allouée et ferme tous les fichiers ouverts. Mais c'est toujours une bonne pratique qu'un programmeur ferme tous les fichiers ouverts avant la fin du programme.

Voici la syntaxe standard de la fonction close(), qui est membre des objets fstream, ifstream et ofstream.

```
fichier.close();
```

5.3.3 Fonctionnement d'entrées et de sorties

5.3.3.1 La fonction put() et la fonction get()

La fonction put() écrit un seul caractère dans le flux associé. De même, la fonction get() lit un seul caractère du flux associé.
Exemple:

```
fichier.get(ch);
fichier.put(ch);
```

L'exemple ci-dessous illustre la manière de lire le contenu d'un fichier et de le transcrire dans un autre fichier en utilisant les fonctions put() et get().

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    char caractere;

    // Ouverture du fichier source en mode lecture
    ifstream fichierSource("source.txt");

    if (!fichierSource) {
        cerr << "Erreur lors de l'ouverture du fichier source !" << endl;
        return 1;
    }

    // Ouverture du fichier de destination en mode écriture
```

```
    ofstream fichierDestination("destination.txt");

    if (!fichierDestination) {
        cerr << "Erreur lors de l'ouverture du fichier de destination !" << endl;
        return 1;
    }

    // Lecture du caractère depuis le fichier source
    while (fichierSource.get(caractere)) {
        // Écriture du caractère dans le fichier de destination
        fichierDestination.put(caractere);
    }

    // Fermeture des fichiers
    fichierSource.close();
    fichierDestination.close();

    cout << "Contenu du fichier source a été copié avec succès dans le fichier de
↪   destination." << endl;

    return 0;
}
```

5.3.3.2 La fonction write() et la fonction read()

Les fonctions write() et read() écrivent et lisent des blocs de données binaires.

La syntaxe générale pour lire et écrire des blocs de données binaires est la suivante:

```
fichier.read((char *)&obj, sizeof(obj));
fichier.write((char *)&obj, sizeof(obj));
```

L'exemple ci-après démontre comment lire le contenu d'un fichier et l'enregistrer dans un autre fichier en se servant des fonctions read() et write() pour manipuler des données binaires:

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    ifstream fichierSource("source.bin", ios::binary); // Ouverture du fichier source
↪   en mode binaire
    ofstream fichierDestination("destination.bin", ios::binary); // Ouverture du
↪   fichier de destination en mode binaire

    if (!fichierSource || !fichierDestination) {
        cerr << "Erreur lors de l'ouverture des fichiers !" << endl;
        return 1;
    }

    char buffer[1024]; // Un tampon pour stocker les données

    // Lecture et écriture des données en blocs de 1024 octets
    while (!fichierSource.eof()) {
        fichierSource.read(buffer, sizeof(buffer)); // Lecture depuis le fichier
↪   source
```

```
        streamsize bytesRead = fichierSource.gcount(); // Nombre d'octets lus

        if (bytesRead > 0) {
            fichierDestination.write(buffer, bytesRead); // Écriture dans le fichier
↪   de destination
        }
    }

    // Fermeture des fichiers
    fichierSource.close();
    fichierDestination.close();

    cout << "Le contenu du fichier source a été copié avec succès dans le fichier de
↪   destination." << endl;

    return 0;
}
```

5.4 Gestion des erreurs

Parfois, lors des opérations sur les fichiers, des erreurs peuvent se produire. Prenons par exemple, un fichier ouvert en lecture peut ne pas exister, ou un nom de fichier utilisé pour un nouveau fichier peut déjà exister, ou une tentative pourrait être faite pour lire au-delà de la fin du fichier, etc.

C++ fournit plusieurs fonctions intégrées pour gérer les erreurs lors des opérations sur les fichiers.

Les fonctionnalités intégrées de gestion des erreurs de fichiers sont décrites dans le tableau 5.2.

Fonction	Valeur de retour et signification
eof()	renvoie True si la fin du fichier est rencontrée lors de la lecture.
fail()	renvoie True lorsqu'une opération d'entrée ou de sortie a échoué.
bad()	renvoie True si une opération non valide est tentée ou si une erreur irrécupérable s'est produite.
good()	renvoie True si aucune erreur ne s'est produite.

Tableau 5.2: Gestion des erreurs liées aux fichiers en C+++

5.5 Pointeurs de fichiers

Tous les objets flux d'E/S ont au moins un pointeur de flux interne: ifstream, comme istream, a un pointeur connu sous le nom de pointeur get qui pointe vers l'élément à lire lors de la prochaine opération d'entrée.

L'objet ofstream, comme ostream, a un pointeur connu sous le nom de pointeur put qui pointe vers l'emplacement où l'élément suivant doit être écrit.

L'objet fstream hérite à la fois des pointeurs get et put de iostream (qui est lui-même dérivé à la fois de istream et ostream).

La figure suivante décrit la hiérarchie de classes en C++ liée aux flux d'entrée et de sortie, plus particulièrement dans le contexte de la bibliothèque standard.

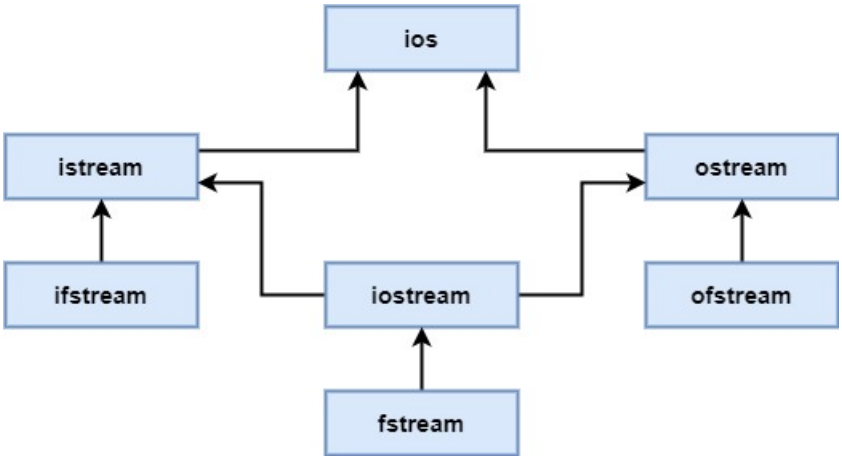


Figure 5.1: Pointeurs de fichiers en C++

Où la classe ios est la classe de base en C++ pour les flux d'entrée et de sortie. C'est une classe modèle (template) pour gérer diverses opérations de flux. ios fournit des fonctionnalités communes et la gestion de l'état pour les flux. Les classes istream et ostream sont des classes abstraites qui dérivent de la classe ios. istream est utilisée pour les opérations d'entrée, et ostream est utilisée pour les opérations de sortie. Elles fournissent des fonctionnalités spécifiques pour la lecture depuis et l'écriture vers les flux. La classe ifstream est une classe pour les flux d'entrée de fichiers, et ofstream est une classe pour les flux de sortie de fichiers. Les deux de ces classes dérivent respectivement de istream et ostream et sont utilisées pour la lecture depuis et l'écriture vers des fichiers. La classe fstream combine à la fois la fonctionnalité d'entrée et de sortie et dérive de la classe iostream. iostream elle-même est dérivée à la fois de istream et ostream. fstream est utilisée pour gérer les flux de fichiers qui prennent en charge à la fois les opérations de lecture et d'écriture.

Ces pointeurs de flux internes pour lire ou écrire des positions dans le flux peuvent être manipulés à l'aide des fonctions membres décrites dans le tableau 5.3.

seekg()	déplace le pointeur d'obtention (entrée) vers un emplacement spécifié
seekp()	déplace le pointeur de placement (sortie) vers un emplacement spécifié
tellg()	donne la position actuelle du pointeur get
tellp()	donne la position actuelle du pointeur put

Tableau 5.3: Fonction membres des pointeurs de fichiers

La fonction seekg(n, ref_pos) prend deux arguments: n désigne le nombre d'octets à déplacer et ref_pos désigne la position de référence par rapport à laquelle le pointeur se déplace. ref_pos peut prendre l'une des trois constantes : ios :: beg déplace le pointeur get de n octets depuis le début du fichier (deb), ios :: end déplace le pointeur get de n octets depuis la fin du fichier et ios :: cur déplace le get pointeur à n octets de la position actuelle du curseur. Si nous ne spécifions pas le deuxième argument, alors ios :: beg est la position de référence par défaut.

Le comportement de seekp(n, ref_pos) est le même que celui de seekg().

Exemples:

```
seekg(offset, ref_pos);
seekp(offset, ref_pos);
```

Le paramètre offset représente le nombre d'octets où le pointeur de fichier doit être déplacé depuis l'emplacement spécifié par le paramètre ref_pos. La ref_pos prend l'une des trois constantes suivantes définies dans la classe ios:
ios::beg début du fichier
ios::cur position actuelle du pointeur
ios::end fin du fichier

Exemple:

```
file.seekg(-10, ios::cur);
```

L'exemple suivant montre comment utiliser seekp pour déplacer la position d'écriture dans un fichier texte:

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

int main() {
    // Ouvre un fichier en mode écriture
    ofstream outputFile("exemple.txt");

    if (outputFile.is_open()) {
        // Écrit du texte dans le fichier
        outputFile << "Apprendre C++" << endl;

        // Obtient la position courante dans le fichier
        streampos currentPosition = outputFile.tellp();

        // Déplace la position d'écriture à la fin du fichier
        outputFile.seekp(0, ios::end);

        // Écrit plus de texte à la fin du fichier
        outputFile << " Fin de Fichier." << endl;

        // Ferme le fichier
        outputFile.close();
    } else {
        cout << "Erreur lors de l'ouverture du fichier." << endl;
    }

    // Ouvre le fichier en mode lecture pour vérifier le contenu
    ifstream inputFile("exemple.txt");

    if (inputFile.is_open()) {
        string line;
        while (getline(inputFile, line)) {
            cout << line << endl;
        }

        // Ferme le fichier
```

```
        inputFile.close();
    } else {
        cout << "Erreur lors de l'ouverture du fichier en lecture." << endl;
    }

    return 0;
}
```

5.6 Opération de base sur un fichier texte

Une opération d'entrée/sortie d'un fichier est un processus en cinq étapes:

- Inclure le fichier d'en-tête fstream dans le programme.
- Déclarer l'objet de flux de fichiers.
- Ouvrir le fichier avec l'objet fstream.
- Utiliser l'objet fstream avec », « ou d'autres fonctions d'entrée/sortie.
- Fermer le fichier.

5.6.1 Ecriture

Pour écrire dans un fichier texte, nous utilisons l'objet ofstream.
Exemple:

```
//écrire un fichier texte
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream mon_fichier("fichier.txt");

  if (mon_fichier.is_open())
  {
    mon_fichier << "La première ligne.\n";
    mon_fichier << "La deuxième ligne.\n";
    mon_fichier.close();
  }
  else cout << "Impossible de créer/ouvrir le fichier spécifié";
  return 0;
}
```

5.6.2 Lecture

Pour lire un fichier texte, nous utilisons l'objet ifstream.

Exemple:

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

int main() {
ifstream mon_fichier;
mon_fichier.open("fichier.txt");
char ch;

while(!mon_fichier.eof()) {
    fin.get(ch);
    cout << ch;
 }
mon_fichier.close();
return 0;
}
```

5.7 Opération de base sur un fichier binaire

Lorsque les données sont stockées dans des fichiers au format binaire, la lecture et l'écriture des données sont plus rapides car aucun temps n'est consacré à la conversion des données d'un format à un autre.

5.7.1 Ecriture

Pour écrire dans un fichier binaire en C++, utilisez la méthode write. Elle est utilisé pour écrire un nombre d'octets sur le flux donné, en commençant de la position du pointeur "put". Le fichier est étendu si le pointeur put est actuellement à la fin du fichier. Si ce pointeur pointe vers le milieu du fichier, les caractères du fichier sont remplacés par les nouvelles données.

Si une erreur s'est produite lors de l'écriture dans le fichier, le flux est placé dans un état d'erreur.

La syntaxe de la méthode write() est:

```
ostream& write(const char*, int);
```

Exemple:

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

using namespace std;

int main()
{
    fstream fichier;
    fichier.open("fichBin.dat", ios_base::out|ios_base::binary);

    if(!fichier.is_open())
    {
        cout<<"Impossible d'ouvrir le fichier\n";
```

```
        return 0;
    }

    string ch = "Cette chaîne est écrite dans le fichier binaire.";

    fichier.write(ch.data(), ch.size());

    fichier.close();

    return 0;
}
```

Notez que le code utilise la fonction data() de l'objet string. C'est parce que la fonction write() accepte le pointeur de caractère plutôt que l'objet chaîne. La fonction data() renvoie un pointeur vers un tableau qui contient exactement les mêmes éléments que la chaîne.

5.7.2 Lecture

Pour lire un fichier binaire en C++, utilisez la méthode read. Elle extrait un nombre d'octets du flux et les place dans la mémoire. Si une erreur se produit lors de la lecture dans le fichier, le flux est placé dans un état d'erreur.

La syntaxe de la méthode read() est :

```
ifstream& read(const char*, int);
```

Exemple: Lecture d'un fichier binaire dans un tampon.

```
#include <iostream.h>
#include <fstream.h>

int main () {
  char * buffer;
  long taille;
  ifstream fichier("fichier.dat", ios::in|ios::binary|ios::ate);
  size = fichier.tellg();
  fichier.seekg (0, ios::beg);
  buffer = new char[taille];
  fichier.read(buffer, taille);
  fichier.close();

  cout << "Le fichier complet est dans un tampon";

  delete[] buffer;
  return 0;
}
```

5.8 Conclusion

Ce chapitre décrit les opérations C++ de base pouvant être effectuées sur des fichiers au format texte ou binaire. Il montre également les opérations impliquées dans la création, l'ouverture, la lecture et l'écriture des fichiers pour chaque format. Comme il décrit également comment les erreurs d'opération de fichier sont gérées et comment les pointeurs sont utilisés et manipulés.

Série d'exercices N°5

Exercice 5.1 Ecrire un programme C++ pour fusionner deux fichiers textes dans un troisième fichier.

Exercice 5.2 Ecrire un programme C++ qui lit les lignes d'un fichier texte et les stocke dans un tableau.

Exercice 5.3 Écrire un programme C++ pour compter le nombre de mots dans un fichier texte.

Exercice 5.4 Écrire un programme C++ pour récupérer la n-ièmme ligne d'un fichier texte.

Exercice 5.5 Écrire un programme C++ pour trouver les lignes du fichier dans lesquelles une chaîne de caractères existe.

Exercice 5.6 Écrire un programme C++ qui lit un fichier texte appelé "input.txt" et remplace toutes les occurrences d'un mot spécifique par un autre mot. Par exemple, remplacez toutes les occurrences de "chat" par "chien" dans le fichier.

Exercice 5.7 Écrire un programme C++ qui copie le contenu d'un fichier source "source.txt" dans un fichier de destination "destination.txt". Assurez-vous que le programme gère correctement les erreurs de lecture et d'écriture de fichiers.

Annexe: Solution d'exercices

Série 1.

Exercice 1.1 Écrire un programme en C++ qui calcule la surface et le périmètre d'un cercle.

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

#define PI 3.141

int main(){
    float rayon, surface, perimetre;
    cout << "Entrer le rayon du cercle\n";
    cin >> rayon;

    // surface = PI x rayon X rayon
    surface = PI * rayon * rayon;

    // perimetre = PI X rayon X 2
    perimetre = PI * rayon * 2;

    cout << "La surface du cercle = " << surface << endl;
    cout << "Le périmètre du cercle = " << perimetre << endl;

    return 0;
}
```

Exercice 1.2 Écrire un programme en C++ qui demande à l'utilisateur d'introduire 3 nombres entiers, puis calculer et afficher leur moyenne.

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

int main(){
```

```
    int a, b, c;
    float moyenne;
    cout << "Entrer le premier nombre : ";
    cin >> a;

    cout << "Entrer le deuxième nombre : ";
    cin >> b;

    cout << "Entrer le troisième nombre : ";
    cin >> c;

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

    cout << "La moyenne des trois nombres = " << moyenne << endl;


    return 0;
}
```

Exercice 1.3 Écrire un programme en C++ qui calcule et affiche le quotient et le reste de la division d'un nombre entier par rapport à un autre.

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

int main(){
    int a, b, quotient, reste;
    cout << "Entrer le numérateur: ";
    cin >> a;

    cout << "Entrer le dénominateur: ";
    cin >> b;

    quotient = a/b;
    reste = a % b;

    cout << "Le quotient = " << quotient << endl;
    cout << "Le reste = " << reste << endl;


    return 0;
}
```

Exercice 1.4 Écrire un programme en C++ qui demande à l'utilisateur d'introduire 3 nombres en virgule flottante, puis utiliser l'opérateur conditionnel (?:) pour déterminer le plus grand nombre.

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

int main(){
    float a, b, c;
    float grand;
```

```
    cout << "Entrer le premier nombre : ";
    cin >> a;

    cout << "Entrer le deuxième nombre : ";
    cin >> b;

    cout << "Entrer le troisième nombre : ";
    cin >> c;

    // moyenne =  (a + b + c)/3
    grand = (a > b)? a : b;
    grand = (grand > c) ? grand : c;

    cout << "Le plus grand nombre = " << grand << endl;


    return 0;
}
```

Exercice 1.5 Écrire un programme en C++ qui permet de calculer le montant TTC (Toutes Taxes comprises) en fonction du prix unitaire de l'unité, la quantité achetée, et le taux TVA (supposé égal à 17%).

```
#include <iostream>
using namespace std;
#define TVA  0.17
int main(){
    float prixHT, quant, prixTTC, reste;
    cout << "Entrer le prix unitaire hors taxe: ";
    cin >> prixHT;

    cout << "Entrer la quantité acheté: ";
    cin >> quant;

    prixTTC = prixHT * quant + (prixHT * quant) * TVA;

    cout << "Le prix TTC = " << prixTTC << endl;


    return 0;
}
```

Exercice 1.6 Écrire un programme en C++ qui demande à l'utilisateur de saisir une température en degrés Celsius, puis convertissez-la en degrés Fahrenheit en utilisant la formule : $F = (C \times 9/5) + 32$.

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

int main() {
    double celsius, fahrenheit;

    // Demander à l'utilisateur de saisir la température en degrés Celsius
    cout << "Entrez la température en degrés Celsius : ";
```

```
    cin >> celsius;

    // Convertir degrés Celsius en degrés Fahrenheit en utilisant la formule
    fahrenheit = (celsius * 9/5) + 32;

    // Afficher le résultat
    cout << "La température en degrés Fahrenheit est : " << fahrenheit << " °F" <<
↪   endl;

    return 0;
}
```

Exercice 1.7 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre de base et un exposant, puis calculez et affichez la valeur de la puissance.

```
#include <iostream>
#include <cmath> // Inclure la bibliothèque pour la fonction pow
using namespace std;

int main() {
    double base, exposant, resultat;

    // Demander à l'utilisateur de saisir la base
    cout << "Entrez la base : ";
    cin >> base;

    // Demander à l'utilisateur de saisir l'exposant
    cout << "Entrez l'exposant : ";
    cin >> exposant;

    // Calculer la valeur de la puissance en utilisant la fonction pow de la
↪   bibliothèque cmath
    resultat = pow(base, exposant);

    // Afficher le résultat
    cout << "La valeur de " << base << " ^ " << exposant << " est : " << resultat <<
↪   endl;

    return 0;
}
```

Série 2.

Exercice 2.1 Écrire un programme en C++ pour vérifier si un nombre est premier.

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

int main(){
    int i, nombre;
    bool est_premier = true;
    cout << "Entrer un nombre entier positif: ";
    cin >> nombre;
```

```
    if (nombre == 0 || nombre == 1) {
    est_premier = false;
    }

  for (i = 2; i <= nombre/2; ++i) {
    if (nombre % i == 0) {
      est_premier = false;
      break;
    }
  }

  if (est_premier)
    cout << nombre << " est un nombre premier";
  else
    cout << nombre << " est un nombre non premier";
  return 0;
}
```

Exercice 2.2 Écrire un programme en C++ pour vérifier si un nombre entier donné est compris entre 5 et 10.

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

int main(){
    int nombre;
    cout << "Entrer un nombre entier: ";
    cin >> nombre;

    if (nombre >= 5 && nombre <= 10)
              cout << nombre << " est compris entre 5 et 10";
    else
        cout << nombre << " n'est pas compris entre 5 et 10";


return 0;
}
```

Exercice 2.3 Écrire un programme en C++ pour calculer la somme des 10 entiers naturels de 1 à 10 en utilisant les boucles for, while et do-while.

```
// boucle for
#include <iostream>
using namespace std;

int main(){
   int i, somme=0;
   for(int i=1;i<=10;i++)
      somme = somme + i;
   cout << "Somme = " << somme;

return 0;
}
```

```
// boucle while
#include <iostream>
```

```cpp
using namespace std;

int main() {
   int somme = 0;
   int i = 1;

   while (i <= 10) {
      somme = somme + i;
      i++;
   }

   cout << "Somme = " << somme;

   return 0;
}
```

```cpp
// boucle while
#include <iostream>
using namespace std;

int main(){
   int i, somme=0;
   while(i<=10){
    somme = somme + i;
    i = i + 1;
   }
   cout << "Somme = " << somme;


return 0;
}
```

```cpp
// boucle do-while
#include <iostream>
using namespace std;

int main(){
   int i, somme=0;
   do {
    somme = somme + i;
    i = i + 1;
   } while(i<=10);
   cout << "Somme = " << somme;


return 0;
}
```

Exercice 2.4 Écrire un progamme en C++ qui calcule le factoriel d'un nombre entier.

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

int main(){
   int nombre, fact = 1;
```

```
   cout << "entrer un nombre entier positif: ";
   cin >> nombre;
   if ((nombre == 0) || (nombre == 1))
     fact = 1;


    for(int i=1;i<=nombre;i++)
       fact = fact * i;

   cout << "Le factoriel de " << nombre << " = "<< fact;


return 0;
}
```

Exercice 2.5 Écrire un programme en C++ pour convertir un chiffre à une à chaîne de caractères.

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

int main(){
   int chiffre;

   cout << "entrer un chiffre décimal (0-9): ";
   cin >> chiffre;

   switch(chiffre){
    case 0:
       cout << "Zéro";
       break;
    case 1:
       cout << "Un";
       break;
    case 2:
       cout << "Deux";
       break;
    case 3:
       cout << "Trois";
       break;
    case 4:
       cout << "Quatre";
       break;
    case 5:
       cout << "Cinq";
       break;
    case 6:
       cout << "Six";
       break;
    case 7:
       cout << "Sept";
       break;
    case 8:
       cout << "Huit";
       break;
    case 9:
       cout << "Neuf";
```

```
        break;
    default:
        cout << "entrer une valeur valide'";
   }
return 0;
}
```

Exercice 2.6 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre entier et affiche la table de multiplication de ce nombre de 1 à 10 en utilisant une boucle "for".

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

int main() {
    int nombre;

    // Demander à l'utilisateur de saisir un nombre entier
    cout << "Entrez un nombre entier : ";
    cin >> nombre;

    // Afficher la table de multiplication de ce nombre de 1 à 10
    cout << "Table de multiplication de " << nombre << " : " << endl;
    for (int i = 1; i <= 10; i++) {
        cout << nombre << " x " << i << " = " << nombre * i << endl;
    }

    return 0;
}
```

Exercice 2.7 Écrire un programme en C++ qui demande à l'utilisateur de saisir un nombre entier. Calculer sa valeur absolue, puis utilisez une boucle "while" pour calculer la factorielle de ce nombre (n!).

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

int main() {
    int nombre, absNombre;
    unsigned long long factorielle = 1; // Utilisation d'un long long pour stocker la
↪   factorielle

    // Demander à l'utilisateur de saisir un nombre entier
    cout << "Entrez un nombre entier : ";
    cin >> nombre;

    // Calculer la valeur absolue du nombre
    if (nombre < 0) {
        absNombre = -nombre;
    } else {
        absNombre = nombre;
    }

    // Calculer la factorielle en utilisant une boucle "while"
```

```
    while (absNombre > 0) {
        factorielle *= absNombre;
        absNombre--;
    }

    // Afficher le résultat
    cout << "La factorielle est : " << factorielle << endl;

    return 0;
}
```

Série 3

Exercice 3.1 Écrire une fonction qui calcule le double d'un nombre réel.

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

float double_nombre(float x);


int main(){
   float nombre, d;

   cout << "entrer un nombre réel: ";
   cin >> nombre;

   d = double_nombre(nombre);
   cout << "le double de " << nombre << " = " << d;
return 0;
}

float double_nombre(float x){
return 2 * x;
}
```

Exercice 3.2 Écrire une fonction inline pour trouver le minimum de deux nombres entiers.

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

inline int max(int x, int y)
 {
    return (x<y ? x : y);
 }

int main()
 {
     int a, b;
     cout<<"Entrer le premier nombre ";
     cin>>a;
     cout<<"Entrer le deuxième nombre: ";
     cin>>b;
```

```
    cout<<"La maximum est "<< max(a,b);
}
```

Exercice 3.3 Écrire une fonction pour vérifier si un nombre est premier.

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

bool est_premier(int n)
{
        if (n <= 1)
                return false;

        for (int i = 2; i < n; i++)
                if (n % i == 0)
                        return false;

        return true;
}

int main()
{
        //Exemples
        cout << est_premier(3) << endl;
        cout << est_premier(8) << endl;
        cout << est_premier(13) << endl;
        return 0;
}
```

Exercice 3.4 Écrire une fonction pour trouver le plus grand diviseur commun (PGDC).

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

int PGDC(int a, int b) {
   if (b == 0)
        return a;
   return PGDC(b, a % b);
}
int main() {

   int a = 350, b = 55;
   cout<<"Le plus grand diviseur commun de "<< a <<" et "<< b <<" est "<< PGDC(a, b);
   return 0;
}
```

Exercice 3.5 Écrire une fonction pour renvoyer la valeur absolue d'un nombre.

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

float abs_value(float a) {
   if (a >=0 )
        return a;
   else return -a;
}
```

```cpp
int main() {
    // exemples
    float x = -5.2;
    float y = 8;
    cout << "La valeur absolue de " << x << " = " << abs_value(x) << endl;
    cout << "La valeur absolue de " << y << " = " << abs_value(y) << endl;
}
```

Exercice 3.6 Écrire une fonction "triSelection" qui prend un tableau d'entiers en entrée et utilise le tri par sélection pour trier les éléments du tableau par ordre croissant. Utilisez un paramètre par référence pour le tableau.

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

// Fonction pour échanger deux éléments d'un tableau
void echanger(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

// Fonction de tri par sélection
void triSelection(int tableau[], int taille) {
    for (int i = 0; i < taille - 1; i++) {
        int indiceMin = i;

        // Rechercher l'indice du plus petit élément non trié
        for (int j = i + 1; j < taille; j++) {
            if (tableau[j] < tableau[indiceMin]) {
                indiceMin = j;
            }
        }

        // Échanger l'élément actuel avec le plus petit élément trouvé
        if (indiceMin != i) {
            echanger(tableau[i], tableau[indiceMin]);
        }
    }
}

int main() {
    int tableau[] = {64, 34, 25, 12, 22, 11, 90};
    int taille = sizeof(tableau) / sizeof(tableau[0]);

    cout << "Tableau non trié : ";
    for (int i = 0; i < taille; i++) {
        cout << tableau[i] << " ";
    }
    cout << endl;

    // Appeler la fonction de tri par sélection
    triSelection(tableau, taille);

    cout << "Tableau trié par sélection : ";
    for (int i = 0; i < taille; i++) {
        cout << tableau[i] << " ";
```

```
    }
    cout << endl;

    return 0;
}
```

Exercice 3.7 Écrire une fonction nommée "puissance" qui prend deux nombres (base et exposant) en entrée et renvoie la valeur de la puissance. Utilisez cette fonction pour calculer et afficher la valeur de 2^3 (2 élevé à la puissance 3) et 5^2 (5 élevé à la puissance 2).

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

// Fonction pour calculer la puissance
double puissance(double base, int exposant) {
    double resultat = 1.0;

    if (exposant > 0) {
        for (int i = 0; i < exposant; i++) {
            resultat *= base;
        }
    } else if (exposant < 0) {
        for (int i = 0; i < -exposant; i++) {
            resultat /= base;
        }
    }

    return resultat;
}

int main() {
    // Calculer et afficher 2^3 et 5^2 en utilisant la fonction puissance
    double resultat1 = puissance(2, 3);
    double resultat2 = puissance(5, 2);

    cout << "2^3 = " << resultat1 << endl;
    cout << "5^2 = " << resultat2 << endl;

    return 0;
}
```

Série 4.

Exercice 4.1 Écrire un programme C++ pour trouver le plus grand et le plus petit élément d'un tableau d'entiers donné.

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

int * min_max(int M[]) {
    // un tableau pour stocker le min et le max du tableau
    int* res = new int[2];
```

```
    int petit = M[0];
    int grand = M[0];
   //
  for(int i = 1;i < sizeof(M); i++) {
    if(M[i] < petit)
      petit = M[i];
  }

  for(int i = 1;i < sizeof(M); i++) {
    if(M[i] > grand)
      grand = M[i];
  }

  res[0]  = petit;
  res[1]  = grand;

  return res;
}
int main() {
    // exempl,es,
    int A[] = {3,9,0,7,5};
    int* r;
    r =  min_max(A);
    cout << "Le plus petit élément du tableau A " << " est " << r[0] << endl;
    cout << "Le plus grand élément du tableau A " << " est " << r[1] << endl;
    return 0;

}
```

Exercice 4.2 Écrire un programme C++ pour compter le nombre d'occurrences d'un élément donné dans un tableau d'entiers.

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

int nbreOcc(int A[], int N, int x)
{
        int nbocc = 0;
        for (int i=0; i< N; i++)
                if (x == A[i])
                 nbocc++;
        return nbocc;
}

int main()
{
    // exemple
        int tab[] = {8, 5, 6, 2, 1, 5, 3, 5, 7, 0, 5};
        int val = 5;
        int N = sizeof(tab)/sizeof(tab[0]);
        cout << "Nombre d'occurrences de " << val << " est " << nbreOcc(tab, N, val);
        return 0;
}
```

Exercice 4.3 Écrire un programme C++ pour inverser un tableau d'entiers.

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

int * inverser(int A[], int N){
int temp;
  for (int i = 0, j = N - 1; i < N/2; i++, j--)
    {
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
    }
    return A;
}

int main()
{
    // exemple
        int tab[] = {1, 2, 3, 4, 4, 6, 7, 8, 9};
        int N = sizeof(tab)/sizeof(tab[0]);
        int * tab_inv;
        tab_inv = inverser(tab,N);
        cout << "Le tableau inverse est :"<< endl;
        for (int i=0; i< N; i++)
        cout << tab_inv[i] << endl;

        return 0;
}
```

Exercice 4.4 Écrire un programme C++ pour tester si une chaîne de caractère est palyndrome.

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

bool palyndrome(string str){
    for (int i = 0; i < str.length() / 2; i++) {

        if (str[i] != str[str.length() - i - 1]) {
            return false;
        }
    }
    return true;


}

int main()
{
    // exemple
        string s = "ABCCBA";
        bool pal = palyndrome(s);
        if (pal)
        cout << "La chaîne de caractères " << s << " est palyndrome";
    else
        cout << "La chaîne de caractères " << s << " n'est pas palyndrome";
```

```
        return 0;
}
```

Exercice 4.5 Écrire un programme C++ pour compter le nombre d'occurrences d'un caractère dans une chaîne.

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

int nbc(string str, char c)
{
        int n = 0;

        for (int i=0;i<str.length();i++)
                if (str[i] == c)
                        n++;

        return n;
}

int main()
{
        // exemple
        string str= "Le programme principal";
        char c = 'p';
        cout << "Le nombre d'occurrences de " << c << " = " << nbc(str, c) << endl;
        return 0;
}
```

Exercice 4.6 Écrire un programme C++ qui demande à l'utilisateur de saisir 5 nombres entiers, stockez-les dans un tableau, puis triez les nombres dans l'ordre croissant et affichez le tableau trié.

```
#include <iostream>
#include <algorithm> // Pour utiliser la fonction std::sort
using namespace std;

int main() {
    const int taille = 5; // Nombre de nombres à saisir
    int tableau[taille];

    // Demander à l'utilisateur de saisir 5 nombres entiers
    cout << "Entrez 5 nombres entiers :" << endl;
    for (int i = 0; i < taille; i++) {
        cout << "Nombre " << i + 1 << ": ";
        cin >> tableau[i];
    }

    // Trier le tableau dans l'ordre croissant en utilisant std::sort
    sort(tableau, tableau + taille);

    // Afficher le tableau trié
```

```
    cout << "Tableau trié dans l'ordre croissant : ";
    for (int i = 0; i < taille; i++) {
        cout << tableau[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Exercice 4.7 Écrire un programme C++ qui demande à l'utilisateur de saisir une liste de chaînes de caractères, stockez-les dans un tableau, puis trouvez et affichez la chaîne la plus longue dans le tableau.

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

int main() {
    vector<string> listeChaines;
    string chaine;

    int nombreChaines;

    cout << "Combien de chaînes de caractères souhaitez-vous entrer ? ";
    cin >> nombreChaines;

    // Demander à l'utilisateur de saisir les chaînes
    for (int i = 0; i < nombreChaines; i++) {
        cout << "Entrez la chaîne #" << i + 1 << ": ";
        cin.ignore(); // Pour éviter les problèmes avec getline après cin
        getline(cin, chaine);
        listeChaines.push_back(chaine);
    }

    // Trouver la chaîne la plus longue
    string chainePlusLongue = listeChaines[0];
    for (int i = 1; i < nombreChaines; i++) {
        if (listeChaines[i].length() > chainePlusLongue.length()) {
            chainePlusLongue = listeChaines[i];
        }
    }

    // Afficher la chaîne la plus longue
    cout << "La chaîne la plus longue est : " << chainePlusLongue << endl
```

Série 5.

Exercice 5.1 Écrire un programme C++ pour fusionner deux fichiers textes dans un troisième fichier.

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

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

int main()
{
  // fichier1 et fichier2 sont les deux fichier à fusionner
  // fichier3 est le fichier texte qui va contenir les lignes de fichier1 et fichier2

  fstream fichier1,fichier2,fichier3;
  // str1 pour récupérer les lignes du premier fichier texte
  // et str2 pour récupérer les lignes du deuxième fichier texte
  string str1,str2;

  // ouvrir fichier1 et fichier2 en mode lecture
  fichier1.open("fich1.txt",ios::in);
  fichier2.open("fich2.txt",ios::in);

  // ouvrir fichier3 en mode écriture
  fichier3.open("fich3.txt",ios::out);

   while(getline(fichier1, str1)){ // Lire le fichier1 ligne par ligne et mettre le
↪  contenu de chaque ligne récupérée dans str1
         fichier3 << str1; // insérer la ligne récupérée (str1) dans fichier3
         fichier3 << endl;
      }

   // la même chose pour fichier2
   while(getline(fichier2, str2)){
         fichier3 << str2;
         fichier3 << endl;
      }
 // fermer les trois fichiers ouverts
  fichier1.close();
  fichier2.close();
  fichier3.close();
}
```

Exercice 5.2 Écrire un programme C++ qui lit les lignes d'un fichier texte et les stocke dans un tableau.

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

int main()
{
  // le fichier texte
  fstream fichier;

  // nombre de lignes du fichier texte
  int N=0;
  string ligne;

   // ouvrir le fichier en mode lecture
  fichier.open("fich.txt",ios::in);
```

```
  // compter le nombre de ligne dans le fichier
  while(getline(fichier, ligne))
        ++N;
  fichier.close();

  // le tableau qui va recevoir les lignes du fichier
  string tab_str[N];

  // str pour récupérer les lignes du fichier texte
  string str;

  // ouvrir le fichier en mode lecture
  fichier.open("fich.txt",ios::in);

  int i =0;
  while(getline(fichier, str)){ // Lire le fichier ligne par ligne et mettre le
↪  contenu de chaque ligne récupérée dans str
         tab_str[i] = str; // mettre str dans le tableau
         i++;
      }

  // vérifier le contenu du tableau
  for (int i=0;i<N;i++)
     cout << tab_str[i] << endl;

  // fermer le fichier
  fichier.close();

  return 0;
}
```

Exercice 5.3 Écrire un programme C++ pour compter le nombre de mots dans un fichier texte.

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

 ifstream fichier("fich.txt");
 int nbmots = 1;
 char ch;
 fichier.seekg(0,ios::beg); // positionner le pointeur au début du fichier

 while(fichier)
 {
  fichier.get(ch);
  if(ch==' '||ch=='\n')
   nbmots++;
 }

 cout << "Nombre de mots = " << nbmots;

 // fermer le fichier
```

```
 fichier.close();

 return 0;
}
```

Exercice 5.4 Écrire un programme C++ pour récupérer la n-ièmme ligne d'un fichier texte.

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

int main() {

ifstream fichier("fich.txt");
string s;
int num; // le numéro de la ligne à récupérer

cout << "Donner le numéro de ligne à récupérer: ";
cin >> num;

for (int i = 1; i <= num; i++)
       getline(fichier, s);

cout << "La ligne récupérée est: \n" << s;
return 0;
}
```

Exercice 5.5 Écrire un programme C++ pour trouver les lignes du fichier dans lesquelles une chaîne de caractères existe.

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

int main() {

ifstream fichier("fich.txt");
string s, line;
unsigned int num = 0;

cout << "Entrer la chaîne de caractères à trouver: ";
getline(cin, s);

while(getline(fichier, line)) {
     num++;
    if (line.find(s, 0) != string::npos) {
        cout << s << " trouvée dans ligne: " << num << endl;
    }
}
return 0;
}
```

Exercice 5.6 Écrire un programme C++ qui lit un fichier texte appelé "input.txt" et remplace toutes les occurrences d'un mot spécifique par un autre mot. Par exemple, remplacez toutes les occurrences de "chat" par "chien" dans le fichier.

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    // Ouvrir le fichier d'entrée "input.txt"
    ifstream inputFile("input.txt");

    // Vérifier si le fichier d'entrée est ouvert avec succès
    if (!inputFile) {
        cerr << "Impossible d'ouvrir le fichier d'entrée." << endl;
        return 1;
    }

    // Ouvrir le fichier de sortie "output.txt"
    ofstream outputFile("output.txt");

    // Vérifier si le fichier de sortie est ouvert avec succès
    if (!outputFile) {
        cerr << "Impossible de créer le fichier de sortie." << endl;
        return 1;
    }

    // Mot à rechercher et mot de remplacement
    string motARechercher = "chat";
    string motDeRemplacement = "chien";

    string ligne;
    while (getline(inputFile, ligne)) {
        size_t found = ligne.find(motARechercher);
        while (found != string::npos) {
            // Remplacer toutes les occurrences du mot
            ligne.replace(found, motARechercher.length(), motDeRemplacement);
            found = ligne.find(motARechercher, found + motDeRemplacement.length());
        }
        // Écrire la ligne modifiée dans le fichier de sortie
        outputFile << ligne << endl;
    }

    // Fermer les fichiers
    inputFile.close();
    outputFile.close();

    cout << "Le remplacement est terminé. Le résultat a été écrit dans 'output.txt'."
↪   << endl;

    return 0;
}
```

Exercice 5.7 Écrire un programme C++ qui copie le contenu d'un fichier source "source.txt" dans un fichier de destination "destination.txt". Assurez-vous que le programme gère correctement les erreurs de lecture et d'écriture de fichiers.

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    // Ouvrir le fichier source "source.txt"
    ifstream sourceFile("source.txt");

    // Vérifier si le fichier source est ouvert avec succès
    if (!sourceFile) {
        cerr << "Impossible d'ouvrir le fichier source." << endl;
        return 1;
    }

    // Ouvrir le fichier de destination "destination.txt"
    ofstream destinationFile("destination.txt");

    // Vérifier si le fichier de destination est ouvert avec succès
    if (!destinationFile) {
        cerr << "Impossible de créer le fichier de destination." << endl;
        return 1;
    }

    // Copier le contenu du fichier source dans le fichier de destination
    string ligne;
    while (getline(sourceFile, ligne)) {
        destinationFile << ligne << endl;
    }

    // Fermer les fichiers
    sourceFile.close();
    destinationFile.close();

    cout << "La copie du fichier source dans le fichier de destination est terminée."
↪   << endl;

    return 0;
}
```

Références

- Klaus Iglberger (2022), *C++ Software Design: Design Principles and Patterns for High-Quality Software*, Edition: 1, ISBN-10: 1098113160, ISBN-13: 978-1098113162, pages: 435, Editeur: Editeur: O'Reilly Media, Inc.
- Bill Weinman (2022), *C++20 STL Cookbook: Leverage the latest features of the STL to solve real-world problems*, ISBN-10: 1803248718, ISBN-13: 978-1803248714, pages: 450, Editeur: Packt Publishing.
- Bjorn Andrist, Viktor Sehr, & Ben Garney, (2020), *C++ High Performance: Master the art of optimizing the functioning of your C++ code*, Edition: 2, ISBN-10: 1839216549, ISBN-13: 978-1839216541, pages: 540, Editeur: Packt Publishing.
- Scott Meyers (2014), *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*, Edition: 1, ISBN-10: 9781491903995, ISBN-13: 978-1491903995, pages: 334, Editeur: O'Reilly Media, Incorporated.
- Bjarne Stroustrup (2013), *The C++ Programming Language*, Edition: 4, ISBN-10: 0275967301, ISBN-13: 978-0275967307, pages: 1376, Editeur: Addison-Wesley Professional.
- Stanley Lippman, Josée Lajoie, & Barbara Moo (2012), *C++ Primer*, Edition: 5, ISBN-10: 9780321714114, ISBN-13: 978-0321714114, pages: 976, Editeur: Addison-Wesley Professional.