Université de Douala
École Supérieure Technique La Salle

Programmation Orientée Objet en langage C++

Travaux pratiques et projet

Classes : **IIA4 & IE4**

Pr Moffo Lonla Bertrand

Année académique 2026–2027

Table des matières

Organisation des travaux pratiques **2**

TP 1 — Environnement de travail et première classe **3**
Environnement sous Windows 11 : g++, Notepad++, Visual Studio 2015 3

TP 2 — Une classe chaîne de caractères **10**

TP 3 — Fonctions et classes amies : gestion bancaire **12**

TP 4 — Surdéfinition d'opérateurs : la classe `Duree` **13**

TP 5 — Héritage simple et multiple : équipements industriels **14**

TP 6 — Polymorphisme : simulation de régulateurs **15**

Projet — Gestion des notes d'une promotion **16**

TP 7 — Templates : tampon circulaire et moyenne glissante **18**

Organisation des travaux pratiques

Les 8 heures de TP sont réparties sur les séances du semestre. Chaque TP met en œuvre, sur machine, le chapitre traité en cours le même jour. Le projet de synthèse est lancé à la séance 8 et se termine en travail personnel.

TP	Séance	Thème	Durée
TP 1	2 — 28/11/2026	Environnement, compilation séparée, première classe	1 h
TP 2	3 — 05/12/2026	Classe à allocation dynamique, recopie, détection d'erreurs mémoire	2 h
TP 3	4 — 12/12/2026	Fonctions et classes amies : gestion bancaire	1 h
TP 4	5 — 19/12/2026	Surdéfinition d'opérateurs : la classe `Duree`	1 h
TP 5	6 — 09/01/2027	Héritage simple et multiple : équipements industriels	1 h
TP 6	7 — 16/01/2027	Polymorphisme : simulation de régulateurs	1 h
Projet	8 — 23/01/2027	Synthèse : gestion des notes d'une promotion	travail personnel
TP 7	9 — 30/01/2027	Templates : tampon circulaire et moyenne glissante	1 h
		Total	**8 h**

Consignes générales

— Compiler **systématiquement** avec les avertissements. Un avertissement est presque toujours le signe d'une erreur.

```
g++ -std=c++17 -Wall -Wextra -o prog.exe prog.cpp
```

— **Copier une commande depuis le PDF :** uniquement depuis un bloc gris comme celui-ci, jamais depuis le texte d'un paragraphe (une coupure de ligne y supprime l'espace entre deux mots, par exemple `-Wall` et `-Wextra`). Relire la commande avant de valider ; en cas de doute, la retaper. Dans un nom de fichier, aucune espace : `ch1_bonjour.cpp`, et non `ch1 _bonjour.cpp`.

— Avancer **par petites étapes** : compiler et tester après chaque méthode écrite.

— Chaque TP se termine par un dépôt des sources (fichiers `.h` et `.cpp`, pas d'exécutable) sur la plateforme de la classe, dans une archive nommée `TPn_Nom1_Nom2.zip`.

— Évaluation d'un TP : programme qui compile sans avertissement (25 %), fonctionnement correct (40 %), respect de l'encapsulation et qualité du code (25 %), réponses aux questions (10 %).

TP 1 — Environnement de travail et première classe

Séance 2 — 1 heure — Chapitres 1 et 2

À retenir

Objectifs. Compiler un programme C++ en ligne de commande ; organiser une classe en fichier d'en-tête et fichier d'implémentation ; utiliser la compilation séparée et un `Makefile` ; lire les messages d'erreur du compilateur.

Préparation : l'environnement de travail sous Windows 11 (64 bits)

Deux environnements sont utilisés dans ce cours : `g++` avec l'éditeur Notepad++ (étapes 1 à 7), pour travailler en ligne de commande, et Visual Studio 2015 Professionnel (étape 8), pour déboguer. Les étapes 1 à 7 sont indispensables : les TP sont évalués avec `g++`.

Windows ne fournit pas de compilateur C++. On installe **GCC** (`g++`) avec la distribution **MSYS2**, environnement **UCRT64** : elle est maintenue, à jour (C++17 complet) et se met à jour en une commande. Comptez environ 15 minutes et 1 Go d'espace disque, avec une connexion Internet. **À faire avant la première séance de TP.**

> **Remarque — autres systèmes** Sous Linux (Ubuntu) :
>
> ```
> sudo apt install g++ make gdb
> ```
>
> Sous macOS (la commande `g++` y appelle le compilateur Clang, compatible avec ce cours) :
>
> ```
> xcode-select --install
> ```

Étape 1 — Installer MSYS2

1. Sur le site officiel `https://www.msys2.org`, télécharger l'installateur `msys2-x86_64-`*date*`.exe`.
2. L'exécuter et garder le dossier proposé `C:\msys64`. Éviter tout dossier dont le chemin contient des espaces ou des accents.
3. À la fin, laisser cochée la case « Run MSYS2 now ».

Étape 2 — Installer le compilateur

Ouvrir le terminal **MSYS2 UCRT64** (menu Démarrer → MSYS2 → MSYS2 UCRT64) et taper :

```
pacman -Syu
```

Si la fenêtre se ferme pendant la mise à jour, la rouvrir et relancer `pacman -Syu`. Installer ensuite le compilateur, le débogueur et `make` (répondre `Y` ou *Entrée* aux questions) :

```
pacman -S --needed mingw-w64-ucrt-x86_64-gcc
pacman -S --needed mingw-w64-ucrt-x86_64-gdb
pacman -S --needed mingw-w64-ucrt-x86_64-make
```

Étape 3 — Ajouter le compilateur au `PATH`

Pour pouvoir taper `g++` dans n'importe quel terminal Windows :

1. Touche *Windows*, taper « variables d'environnement » et choisir **Modifier les variables d'environnement pour votre compte**.
2. Dans *Variables utilisateur*, sélectionner `Path` puis **Modifier**.
3. **Nouveau** → saisir `C:\msys64\ucrt64\bin` → **OK**, puis **OK**.
4. **Fermer et rouvrir** tous les terminaux et éditeurs : le `PATH` n'est lu qu'à leur ouverture.

Étape 4 — Vérifier l'installation

Ouvrir l'application **Terminal** (PowerShell) et taper :

```
g++ --version
mingw32-make --version
gdb --version
where.exe g++
```

Chaque commande doit afficher un numéro de version ; la dernière doit répondre :

`C:\msys64\ucrt64\bin\g++.exe`

Sous PowerShell, il faut écrire `where.exe` : `where` seul désigne une autre commande.

Étape 5 — Compiler et exécuter en ligne de commande

```
mkdir $HOME\Documents\POO\tp1          # dossier de travail (une seule fois)
cd $HOME\Documents\POO\tp1
g++ -std=c++17 -Wall -Wextra -o bonjour.exe bonjour.cpp
.\bonjour.exe
```

Action	PowerShell (Terminal)	Invite de commandes `cmd.exe`
Exécuter	`.\bonjour.exe` (le `.\` est obligatoire)	`bonjour.exe`
Lire l'entrée dans un fichier	`cmd /c "polygone.exe < main.in"`	`polygone.exe < main.in`
Lancer le `Makefile`	`mingw32-make`	`mingw32-make`
Lister le dossier	`ls` ou `dir`	`dir`

Étape 6 — Les accents dans la console

Symptôme : le programme affiche `├óge` ou `Ã¢ge` au lieu de `âge`. Le programme envoie bien les accents en UTF-8, mais la console les lit avec l'ancien jeu de caractères de Windows. Deux conditions sont nécessaires :

1. les sources sont enregistrées en **UTF-8** (réglage de Notepad++ décrit à l'étape 7) ;
2. la console lit la sortie en UTF-8. Dans **PowerShell**, taper une fois par fenêtre, avant d'exécuter le programme :

   ```
   [Console]::OutputEncoding = [System.Text.Encoding]::UTF8
   ```

   Dans l'**Invite de commandes** (`cmd`), la commande équivalente est :

   ```
   chcp 65001
   ```

Attention

Dans Windows PowerShell, `chcp 65001` répond « Active code page : 65001 » mais ne suffit pas : les accents restent faux. Il faut la commande `[Console]::OutputEncoding` ci-dessus (constaté en salle de TP).

Pour une solution permanente, valable pour tous les terminaux : *Paramètres* → *Heure et langue* → *Langue et région* → *Paramètres de langue d'administration* → *Modifier les paramètres régionaux du système*, cocher « Bêta : utiliser Unicode UTF-8 pour la prise en charge des langues du monde entier », puis redémarrer. Ce réglage concerne tout l'ordinateur : sur les postes de la salle, il relève de l'administrateur.

Étape 7 — Éditeur : Notepad++

Notepad++ (`https://notepad-plus-plus.org`) est un éditeur léger et rapide, qui colore la syntaxe du C++ et s'installe avec un seul fichier, sans connexion Internet. Il ne compile pas lui-même : on compile dans un terminal, ce qui oblige à maîtriser les options de compilation.

Installer Notepad++. Le dossier `logiciels/` du cours contient l'installateur officiel `npp.8.9.9.1.Installer.x64.exe` (version 8.9.9.1, 64 bits). Sinon, le télécharger sur le site officiel. Lancer l'installateur et accepter les choix proposés ; Windows demande l'autorisation d'un administrateur.

Deux réglages à faire une fois pour toutes (menu *Paramètres* → *Préférences*) :

1. *Nouveau document* → *Encodage* : choisir **UTF-8** (sans BOM). Sinon, un fichier enregistré dans l'ancien encodage de Windows (ANSI) affiche des accents faux à l'exécution. Pour un fichier déjà ouvert : menu *Encodage* → *Convertir en UTF-8*.
2. *Langage* → *Tabulations* : laisser **décochée** l'option *Remplacer par des espaces*. Dans un `Makefile`, chaque commande doit commencer par une vraie tabulation (erreur `missing separator` sinon). Menu *Affichage* → *Symboles spéciaux* → *Afficher les espaces et tabulations* pour les voir.

Méthode de travail.

1. Créer le dossier du TP, puis y enregistrer chaque fichier avec la bonne extension (`.cpp`, `.h`) : la coloration C++ s'active alors automatiquement. Le fichier `Makefile` s'enregistre **sans extension** (choisir le type « All types » dans la boîte d'enregistrement).
2. Ouvrir un terminal directement dans le dossier du fichier : *Fichier* → *Ouvrir le dossier contenant* → *cmd*.
3. Dans ce terminal, compiler et exécuter avec les commandes de l'étape 5 (`g++ ...` ou `mingw32-make`, puis `polygone.exe`).
4. Penser à **enregistrer** (*Ctrl+S*) avant chaque compilation : `g++` compile le fichier sur le disque, pas ce qui est affiché à l'écran.

Option : compiler sans quitter l'éditeur avec NppExec. *Installer le module NppExec*, de l'une des deux façons suivantes :

— **avec Internet** : menu *Modules* → *Gestionnaire de modules...*, onglet *Disponibles*, rechercher `NppExec`, le cocher puis cliquer sur *Installer* ; Notepad++ redémarre ;

— **sans Internet**, avec l'archive `NppExec_08121_dll_x64.zip` du dossier `logiciels/` (version 0.8.12.1, 64 bits) : fermer Notepad++, puis créer le dossier :

```
C:\Program Files\Notepad++\plugins\NppExec
```

y extraire **tout** le contenu de l'archive (le fichier `NppExec.dll` doit se trouver directement dans ce dossier), puis relancer Notepad++.

Dans les deux cas, Windows demande l'autorisation d'un administrateur, et le module apparaît dans le menu *Modules* → *NppExec*.

Utiliser NppExec. Ouvrir *Modules* → *NppExec* → *Execute NppExec Script...* (touche *F6*), saisir le script suivant, l'enregistrer sous le nom `compiler` (bouton *Save...*) puis l'exécuter :

```
npp_save
cd "$(CURRENT_DIRECTORY)"
g++ -std=c++17 -Wall -Wextra -o "$(NAME_PART).exe" "$(FILE_NAME)"
```

Le script enregistre le fichier courant puis le compile ; les messages du compilateur s'affichent dans la console de NppExec, en bas de la fenêtre. *Ctrl+F6* relance le dernier script sans passer par la boîte de dialogue. Pour un projet en plusieurs fichiers, remplacer la dernière ligne par `mingw32-make`.

Attention

On **exécute** toujours le programme dans le terminal ouvert à l'étape 2 de la méthode : la console de NppExec gère mal les saisies au clavier (`cin`).

Compiler et exécuter dans une fenêtre séparée (touche F5). Notepad++ peut lancer la compilation puis le programme dans une **nouvelle fenêtre**, qui reçoit bien les saisies au clavier et affiche correctement les accents. Ouvrir *Exécution* → *Exécuter...* (touche *F5*) et coller la ligne suivante, **sur une seule ligne** :

```
cmd /c start "" /d $(CURRENT_DIRECTORY) cmd /k "g++ -std=c++17 -Wall -Wextra -o $(NAME_PART).exe $(FILE_NAME) && chcp 65001 >nul && $(NAME_PART).exe"
```

La même ligne est fournie dans le fichier `code/tp/notepadpp_F5.txt`, plus facile à copier. Cliquer sur **Enregistrer...**, lui donner un nom (par exemple « Compiler et exécuter ») et un raccourci (par exemple *Ctrl+F9*). Ensuite, à chaque appui sur ce raccourci :

- une nouvelle fenêtre s'ouvre et compile le fichier courant ;
- en cas d'erreur, les messages du compilateur s'affichent dans cette fenêtre et le programme n'est pas lancé ;
- sinon, le programme s'exécute ; la fenêtre reste ouverte à la fin, ce qui laisse le temps de lire le résultat. On la ferme avec la croix ou en tapant `exit`.

Attention

Exécuter n'enregistre pas le fichier : faire *Ctrl+S* avant. Cette commande compile un **seul** fichier : pour un projet en plusieurs fichiers, utiliser `mingw32-make` dans le terminal. Le chemin du dossier de travail et le nom du fichier ne doivent contenir **aucune espace** : si le nom du compte Windows en contient une, travailler dans un dossier comme `C:\POO\tp1`.

Depuis PowerShell, on peut aussi lancer un programme déjà compilé dans une nouvelle fenêtre :

```
Start-Process cmd -ArgumentList '/k chcp 65001 >nul & bonjour.exe'
```

Alternative : Visual Studio Code. Les étudiants qui préfèrent VS Code (`https://code.visualstudio.com`) installent l'extension **C/C++** de Microsoft, ouvrent le dossier du TP (*Fichier* → *Ouvrir le dossier*) et compilent dans le terminal intégré (*Terminal* → *Nouveau terminal*) avec les mêmes commandes.

Variante sans connexion Internet (salle de TP)

La distribution **WinLibs** (`https://winlibs.com`) est une simple archive : choisir la version « GCC + MinGW-w64 (UCRT) — Win64 » au format `.zip`, l'extraire dans `C:\mingw64`, puis ajouter `C:\mingw64\bin` au `PATH` (étape 3). Elle contient `g++`, `gdb` et `mingw32-make`. L'archive peut circuler sur clé USB, avec le contenu du dossier `logiciels/` du cours (Notepad++ et NppExec).

Problèmes fréquents

Symptôme	Cause et solution
`g++ : Le terme « g++ » n'est pas reconnu...`	`PATH` non modifié, ou terminal ouvert avant la modification : vérifier l'étape 3 et rouvrir le terminal. Vérifier que `g++.exe` est bien dans `C:\msys64\ucrt64\bin` (paquet installé depuis le terminal UCRT64).
`make` introuvable	Sous Windows, la commande s'appelle `mingw32-make`.
`Makefile:16: *** missing separator`	La ligne de commande commence par des espaces au lieu d'une tabulation : régler Notepad++ (étape 7) et retaper la tabulation.
Accents mal affichés (`├ó`, `Ã©`)	Console non réglée en UTF-8 : dans PowerShell, `[Console]::OutputEncoding = [System.Text.Encoding]::UTF8` (`chcp 65001` ne suffit pas) ; dans `cmd`, `chcp 65001`. Si le défaut persiste, le source est en ANSI : *Encodage → Convertir en UTF-8* dans Notepad++ (étape 6).
`L'opérateur « < » est réservé à une utilisation future`	PowerShell n'accepte pas `<` : utiliser `cmd /c "prog.exe < fichier"`, ou travailler dans `cmd`.
Le programme lit `0` ou une valeur absurde quand on lui envoie l'entrée par un tube (`echo 22 | .\prog.exe`)	Windows PowerShell réencode le texte transmis par le tube, ce qui peut faire échouer `cin`. Saisir au clavier, ou utiliser la redirection de `cmd` ci-dessus.
`Permission denied` à la recompilation	Le programme précédent tourne encore : le fermer (*Ctrl+C*) avant de recompiler.
L'antivirus bloque ou supprime l'`.exe`	Faux positif fréquent sur les petits exécutables : ajouter le dossier de travail aux exclusions de l'antivirus.
La fenêtre se ferme aussitôt	Le programme a été lancé par double-clic : l'exécuter depuis le terminal, ou par la touche *F5* de Notepad++ (étape 7).
`cannot find -lasan` avec `-fsanitize=address`	AddressSanitizer n'est pas disponible avec GCC sous Windows : voir le TP 2 (utilisation de WSL).

Étape 8 — Utiliser Visual Studio 2015 Professionnel

Visual Studio 2015 Professionnel est un environnement de développement intégré (EDI) : éditeur, compilateur Microsoft (MSVC) et débogueur graphique réunis. Il est très utile pour **déboguer** (exécution pas à pas, observation des variables). Les programmes du cours y compilent tous, sous réserve des réglages ci-dessous.

Vérifier la version. Menu *Aide → À propos de Microsoft Visual Studio* : la version doit être `14.0.25431.01 Update 3`. Sinon, installer la mise à jour « Visual Studio 2015 Update 3 », indispensable aux options `/utf-8` et `/std:c++latest` utilisées plus bas.

Créer un projet.

1. *Fichier → Nouveau → Projet*, puis *Modèles → Visual C++ → Win32 → Application console Win32*. Donner un nom (par exemple `tp1`) et un emplacement, puis **OK**.
2. Dans l'assistant, cliquer sur **Suivant** (paramètres de l'application) : cocher **Projet vide** et décocher les vérifications SDL, puis **Terminer**. Un projet vide évite le fichier `stdafx.h` imposé par le modèle par défaut.
3. Dans l'*Explorateur de solutions*, clic droit sur *Fichiers sources → Ajouter → Nouvel élément → Fichier C++ (.cpp)*. Pour un fichier qui existe déjà : *Ajouter → Élément existant*. Les fichiers `.h` se placent dans *Fichiers d'en-tête*.

Régler le projet (*Projet → Propriétés*, en choisissant en haut *Configuration : Toutes les configurations*) :

Rubrique	Valeur	Rôle
C/C++ → Général → Niveau d'avertissement	`/W4`	équivalent de `-Wall -Wextra`
C/C++ → Ligne de commande → Options supplémentaires	`/utf-8 /std:c++latest`	sources lues en UTF-8 ; dernière norme disponible
C/C++ → Préprocesseur → Définitions de préprocesseur	ajouter `_CRT_SECURE_NO_WARNINGS`	autorise `strcpy`, `strcat`… (erreur C4996 sinon)
Débogage → Arguments de la commande	`< main.in`	lire l'entrée dans un fichier (facultatif)

Compiler, exécuter, déboguer.

Action	Commande
Générer (compiler et lier)	*Ctrl+Maj+B*
Exécuter **sans** débogage (la console reste ouverte)	*Ctrl+F5*
Exécuter avec le débogueur	*F5*
Poser ou retirer un point d'arrêt sur la ligne courante	*F9*
Pas à pas principal / détaillé (entre dans les fonctions)	*F10* / *F11*

Pendant le débogage, les fenêtres *Variables locales* et *Espion* affichent l'état des objets, y compris leurs membres privés. Les erreurs de compilation apparaissent dans la *Liste d'erreurs* : un double-clic mène à la ligne fautive.

Différences à connaître avec `g++`.

— **Norme.** Le compilateur de VS 2015 applique complètement le C++14 et seulement une partie du C++17 (`/std:c++latest`). Tous les programmes du cours, des TD et des TP compilent en C++14.

— **Compilation séparée.** Le fichier de projet (`.vcxproj`) joue le rôle du `Makefile` : VS ne recompile que les fichiers modifiés. Le `Makefile` du TP 1 se fait donc avec `g++` et `mingw32-make`.

— **Copies d'objets.** En configuration *Debug*, VS 2015 n'élimine pas toujours la copie d'un objet renvoyé par valeur : des messages de recopie et de destruction supplémentaires peuvent apparaître dans les programmes qui tracent leurs constructeurs (TD 3, TP 2). C'est un comportement permis par la norme, pas une erreur.

— **Erreurs mémoire.** AddressSanitizer n'existe pas dans VS 2015. En configuration *Debug*, la bibliothèque de débogage détecte cependant une double libération : une fenêtre « Debug Assertion Failed » s'ouvre (TP 2).

— **Évaluation.** Les TP rendus sont compilés avec `g++` ; vérifier que le programme compile aussi de cette façon :

```
g++ -std=c++17 -Wall -Wextra -o prog.exe prog.cpp
```

Exercice 1 — Premier programme et messages d'erreur

1. Saisir le programme `ch1_bonjour.cpp` du cours, puis le compiler et l'exécuter. Sous Windows (PowerShell) :

   ```
   g++ -std=c++17 -Wall -Wextra -o bonjour.exe ch1_bonjour.cpp
   .\bonjour.exe
   ```

   Sous Linux ou macOS : `-o bonjour`, puis `./bonjour`.

2. Introduire successivement les erreurs suivantes, recompiler et **noter le message** obtenu : oubli du point-virgule après `cin >> age` ; suppression de `using namespace std;` ; remplacement de `int age;` par `int age` ; utilisation d'une variable non déclarée.

3. Pourquoi le premier message d'erreur est-il le seul à examiner en priorité ?

Exercice 2 — Compilation séparée : la classe `Point`

On veut calculer le périmètre d'un polygone dont on saisit les sommets.

1. Créer le fichier `point.h` contenant la déclaration d'une classe `Point` (attributs privés `x`, `y`) avec les méthodes `initialise(double = 0, double = 0)`, `deplace(double, double)`, `double distance(const Point &) const`, `getX()`, `getY()` et `affiche() const`. Le protéger contre les inclusions multiples.
2. Créer `point.cpp` contenant la définition des méthodes (`std::sqrt` est déclarée dans `<cmath>`).
3. Créer `main.cpp` qui lit le nombre `n` de sommets, alloue dynamiquement un tableau de `n` points, lit leurs coordonnées, affiche la longueur de chaque côté (le dernier sommet est relié au premier) et le périmètre.
4. Compiler en deux temps, puis répondre : quel est le rôle de chaque étape ?

```
g++ -std=c++17 -Wall -Wextra -c point.cpp
g++ -std=c++17 -Wall -Wextra -c main.cpp
g++ main.o point.o -o polygone.exe
```

   Sous Linux ou macOS, l'exécutable s'appelle simplement `polygone`.

5. Écrire un `Makefile`, puis le lancer (sous Linux ou macOS : `make`) :

```
mingw32-make
```

   Modifier `main.cpp` seul et relancer la commande : quels fichiers sont recompilés ?

6. Que se passe-t-il si l'on retire `#ifndef POINT_H` et que `main.cpp` inclut deux fois `point.h` ?

À rendre : `point.h`, `point.cpp`, `main.cpp`, `Makefile`.

TP 2 — Une classe chaîne de caractères

Séance 3 — 2 heures — Chapitre 3

À retenir

Objectifs. Écrire une classe qui gère elle-même sa mémoire ; observer l'ordre des appels des constructeurs et destructeurs ; constater les effets d'une copie superficielle avec un outil de détection d'erreurs mémoire ; appliquer la règle des trois.

Exercice 1 — Observer le bogue de la copie superficielle

Le fichier `demo/tp2_bug.cpp` définit une classe `Chaine` qui alloue un tableau de caractères dans son constructeur et le libère dans son destructeur, sans constructeur de recopie.

demo/tp2_bug.cpp

```cpp
// Version BOGUÉE : pas de constructeur de recopie ni d'opérateur =
// Compiler avec :  g++ -std=c++17 -g -fsanitize=address tp2_bug.cpp -o bug
#define _CRT_SECURE_NO_WARNINGS   // Visual Studio : autorise strcpy et strcat (erreur C4996 sinon)
#include <iostream>
#include <cstring>

class Chaine {
    char *txt;
public:
    Chaine(const char *s) : txt(new char[strlen(s) + 1]) { strcpy(txt, s); }
    ~Chaine() { delete[] txt; }
    void affiche() const { std::cout << txt << std::endl; }
};

int main() {
    Chaine a("ecole");
    Chaine b = a;      // copie superficielle : a.txt et b.txt pointent la même zone
    b.affiche();
    return 0;          // ~b libère la zone, puis ~a la libère une seconde fois
}
```

1. Compiler normalement et exécuter. Le programme semble-t-il fonctionner ?
2. Compiler avec le détecteur d'erreurs mémoire (sous Linux, macOS ou WSL, voir la remarque ci-dessous) :

   ```
   g++ -std=c++17 -g -fsanitize=address tp2_bug.cpp -o bug
   ./bug
   ```

   Lire le rapport : quelle erreur est détectée, et à quelle ligne ?
3. Expliquer l'erreur à l'aide d'un schéma mémoire.

Remarque — sous Windows : AddressSanitizer via WSL GCC sous Windows (MSYS2, WinLibs) ne fournit pas `-fsanitize=address` (erreur `cannot find -lasan`). On utilise alors le sous-système Linux de Windows (**WSL**), une seule fois :

1. Dans PowerShell lancé **en administrateur**, taper la commande ci-dessous, puis redémarrer.

```
wsl --install
```

   Au premier lancement d'Ubuntu, choisir un nom d'utilisateur et un mot de passe. La virtualisation doit être activée dans le BIOS (c'est le cas sur la plupart des PC récents).

2. Dans le terminal Ubuntu :

```
sudo apt update
sudo apt install g++ make gdb
```

3. Les fichiers Windows sont visibles sous `/mnt/c` : `cd /mnt/c/Users/`*nom*`/Documents/POO/tp2`.
4. Compiler et lancer le programme avec les deux commandes de la question 2.

À défaut de WSL : faire afficher par le constructeur et le destructeur l'adresse de la zone (`cout << (void*)txt`), comme dans l'exemple `ch3_vecteur.cpp` du cours. On voit alors la **même adresse** libérée deux fois.

Remarque — avec Visual Studio 2015 Compiler `tp2_bug.cpp` en configuration *Debug* et l'exécuter (*Ctrl+F5*) : à la seconde libération, la bibliothèque de débogage ouvre une fenêtre « Debug Assertion Failed ». Relancer avec le débogueur (*F5*) et cliquer sur *Recommencer* : VS 2015 s'arrête sur l'erreur, et la *Pile des appels* montre qu'elle survient dans `~Chaine()`, appelé à la fin de `main`.

Exercice 2 — Une classe `Chaine` correcte

Écrire une classe `Chaine` ayant pour attributs la longueur `lg` et un pointeur `txt` vers une zone allouée par `new char[lg + 1]`.

1. Écrire une fonction membre privée `allouer(const char *)` utilisée par tous les constructeurs.
2. Écrire le constructeur `Chaine(const char * = "")`, le constructeur de recopie et le destructeur. Chacun affiche un message entre crochets, par exemple `[recopie "ecole"]`.
3. Écrire l'opérateur d'affectation `operator=` (il sera étudié en détail au chapitre 5 : s'inspirer du cours).
4. Écrire les méthodes `longueur()`, `concatener(const Chaine &)`, `majuscules()`, `egale(const Chaine &) const` et `affiche() const`.
5. Ajouter un membre statique `nbZones` qui compte les zones actuellement allouées, et une méthode statique `zones()`.
6. Écrire une fonction `Chaine saluer(Chaine nom)` qui renvoie `"Bonjour "` suivi du nom.
7. Dans `main`, reproduire les étapes du corrigé (construction, recopie, affectation, appel de `saluer`). Avant d'exécuter, **prévoir** sur papier la liste des messages ; puis comparer avec l'exécution.
8. Recompiler avec `-fsanitize=address` (sous Windows : dans WSL, voir l'exercice précédent) : aucune erreur ne doit être signalée. Que vaut `Chaine::zones()` juste avant la fin de `main` ? Et après la destruction de tous les objets ?

À rendre : `chaine.cpp` et les réponses aux questions 7 et 8.

TP 3 — Fonctions et classes amies : gestion bancaire

Séance 4 — 1 heure — Chapitre 4

À retenir

Objectifs. Mettre en œuvre une fonction indépendante amie de plusieurs objets et une classe amie ; mesurer ce que l'amitié autorise et ce qu'elle coûte en encapsulation.

Exercice 1 — Comptes et banque

1. Écrire une classe `Compte` ayant les attributs privés `numero`, `titulaire`, `solde` et `bloque` (`bool`), un constructeur à valeurs par défaut, les accesseurs `getSolde()` et `getNumero()`, et une méthode `afficher()`. **Aucun mutateur** n'est fourni.
2. Écrire une fonction indépendante `bool transfert(Compte &src, Compte &dst, double montant)`, amie de `Compte`, qui refuse l'opération si l'un des comptes est bloqué, si le montant est négatif ou si le solde est insuffisant.
3. Écrire une classe `Banque` gérant un tableau d'au plus 10 comptes, avec les méthodes `ouvrir`, `Compte *chercher(const string &numero)`, `bloquer`, `appliquerFrais(double)`, `totalDepots()` et `releve()`. Déclarer `Banque` comme classe amie de `Compte`.
4. Tester : ouvrir trois comptes, effectuer un transfert accepté et un transfert refusé, bloquer un compte, tenter un transfert depuis ce compte, appliquer 1 500 FCFA de frais et afficher le relevé.
5. Supprimer la ligne `friend class Banque;` et recompiler. Quelles méthodes de `Banque` ne compilent plus ? Comment les réécrire sans amitié, et quel inconvénient cela aurait-il ?

À rendre : `banque.cpp` et la réponse à la question 5.

TP 4 — Surdéfinition d'opérateurs : la classe `Duree`

Séance 5 — 1 heure — Chapitre 5

À retenir

Objectifs. Concevoir une classe dont les objets se manipulent comme des types de base grâce aux opérateurs ; choisir entre forme membre et forme amie ; surdéfinir les opérateurs de flot.

Exercice 1 — Durées d'un cycle de production

Une durée est mémorisée en **secondes** (attribut `long sec`, jamais négatif) et affichée sous la forme `h:mm:ss`.

1. Écrire le constructeur `Duree(long h = 0, long m = 0, long s = 0)` et les accesseurs `heures()`, `minutes()`, `secondes()`.
2. Surdéfinir en fonctions **membres** : `+=`, `-=` (une durée négative est ramenée à 0), `++` préfixé et postfixé (ajout d'une seconde).
3. Surdéfinir en fonctions **amies** : `+` et `-` (en réutilisant `+=` et `-=`), `*` par un entier dans les deux ordres (`d * 3` et `3 * d`), `==`, `!=` et `<`.
4. Surdéfinir `<<` (affichage `1:05:09`, avec `setw` et `setfill` de `<iomanip>`) et `>>` (saisie `h m s`).
5. Programme de test : saisir les durées des 4 étapes d'un cycle, afficher la durée du cycle, l'étape la plus longue, la durée de 8 cycles et la marge restante sur un poste de 8 heures. Vérifier les deux formes de `++`.
6. Pourquoi `3 * d` exige-t-il une fonction amie (ou non membre) ? Pourquoi `operator+` reçoit-il son premier argument **par valeur** dans le corrigé ?

À rendre : `duree.cpp` et la réponse à la question 6.

TP 5 — Héritage simple et multiple : équipements industriels

Séance 6 — 1 heure — Chapitre 6

À retenir

Objectifs. Construire une hiérarchie de classes ; transmettre les arguments aux constructeurs de base ; utiliser le statut `protected` ; redéfinir une méthode en réutilisant la version de base ; mettre en œuvre un héritage multiple et lever ses ambiguïtés.

Exercice 1 — Appareils et moteurs (héritage simple)

1. Écrire une classe `Appareil` : attributs protégés `nom`, `puissance` (kW) et `enMarche` ; constructeur ; méthodes `demarrer()`, `arreter()`, `consommation(double heures)` (en kWh, nulle à l'arrêt) et `afficher()`.
2. Dériver une classe `Moteur` ayant en plus une vitesse courante et une vitesse maximale (tr/min). La méthode `bool reglerVitesse(int)` refuse une vitesse hors de $[0, v_{\max}]$ ou un réglage moteur à l'arrêt. Redéfinir `afficher()`.
3. Ajouter un message dans chaque constructeur et destructeur.

Exercice 2 — Moteur communicant (héritage multiple)

1. Écrire une classe `Communicant` représentant un équipement relié à un bus de terrain : attributs `adresse` (1 à 247) et `trames` (nombre de trames émises), méthode `envoyer(const string &msg)` qui affiche `[@12 trame 3] msg`, et méthode `afficher()`.
2. Dériver de `Moteur` et de `Communicant` une classe `MoteurConnecte`. Sa méthode `reglerVitesse` règle la vitesse *et* envoie un compte rendu sur le bus (« consigne 1200 acceptée »).
3. Tenter d'appeler `afficher()` sur un `MoteurConnecte` **avant** de la redéfinir : quel message donne le compilateur ? Redéfinir ensuite `afficher()`.
4. Programme de test : créer un moteur connecté (5,5 kW, 1500 tr/min, adresse 12), le démarrer, demander 1200 puis 1800 tr/min, afficher son état et sa consommation sur un poste de 7 h. Observer l'ordre des constructions et des destructions.
5. Déclarer `Appareil *pa = &m;` et appeler `pa->afficher()`. Quelle version est exécutée ? Pourquoi ? (Ce point sera repris au TP 6.)

À rendre : `appareils.cpp` et les réponses aux questions 3 et 5.

TP 6 — Polymorphisme : simulation de régulateurs

Séance 7 — 1 heure — Chapitre 7

À retenir

Objectifs. Définir une interface par une classe abstraite ; écrire plusieurs implémentations ; écrire un code client qui ne connaît que l'interface ; observer l'intérêt de la liaison dynamique.

Contexte. On simule la régulation de température d'un four. Le four est modélisé par un système du premier ordre :

$$\theta_{k+1} = \theta_k + \frac{\Delta t}{\tau}\left(K\,u_k - (\theta_k - \theta_a)\right), \qquad \tau = 50\ \text{s},\ K = 2\ °\text{C}/\%,\ \theta_a = 25\ °\text{C},$$

où $u_k \in [0, 100]$ est la commande de chauffe en %. On compare trois régulateurs pour une consigne de 150 °C.

Exercice 1 — Hiérarchie de régulateurs

1. Écrire une classe abstraite `Regulateur` : attributs protégés `nom`, `umin`, `umax` ; méthode protégée `saturer(double)` ; fonction virtuelle pure `double commande(double consigne, double mesure)` ; fonction virtuelle `reinitialiser()` qui ne fait rien ; destructeur virtuel.
2. Écrire `RegulateurTOR` (tout ou rien) : chauffe à 100 % si la mesure est inférieure à consigne $- h$, s'arrête si elle dépasse consigne $+ h$, et garde son état entre les deux (hystérésis h).
3. Écrire `RegulateurP` : $u = K_p\, e$ avec $e =$ consigne $-$ mesure, saturé.
4. Écrire `RegulateurPI`, **dérivé de** `RegulateurP` : $u = K_p\, e + K_i \sum e\,\Delta t$. L'intégrale n'est mise à jour que si la commande n'est pas saturée (anti-emballement). Redéfinir `reinitialiser()`.
5. Écrire une classe `Four` implémentant le modèle ci-dessus.
6. Écrire une fonction `void simuler(Regulateur &r, double consigne, int pas, double dt)` qui simule 200 secondes et affiche la température toutes les 25 secondes.
7. Dans `main`, ranger les trois régulateurs ($h = 2$ °C, $K_p = 4$, $K_i = 0{,}2$) dans un tableau de `Regulateur*` et les simuler dans une boucle.
8. Interpréter les résultats. Vérifier par le calcul la température finale obtenue avec le régulateur P.
9. Ajouter une classe `RegulateurPD` sans modifier ni `simuler` ni la boucle de `main` (seule la déclaration du tableau change). Quel principe de la P.O.O cela illustre-t-il ?

À rendre : `regulation.cpp` et les réponses aux questions 8 et 9.

Projet — Gestion des notes d'une promotion

Lancé à la séance 8 — travail en binôme — rendu le 06/02/2027

À retenir

Objectifs. Réunir dans un même programme les notions du semestre : encapsulation, constructeurs et règle des trois, surdéfinition d'opérateurs, héritage, polymorphisme, templates et bibliothèque standard.

Cahier des charges

Le responsable pédagogique veut un programme qui calcule les moyennes d'une matière et produit le classement de la promotion. Une matière comporte trois types d'évaluations :

Type	Coefficient	Calcul de la note sur 20
Contrôle continu (CC)	1	note obtenue
Examen (EX)	2	note obtenue ; en cas de rattrapage, la meilleure des deux notes
Travaux pratiques (TP)	1	moyenne des notes de comptes rendus

La moyenne d'un étudiant est la moyenne pondérée de ses évaluations. Il est admis si elle est au moins égale à 10.

Exercice 1 — Hiérarchie des évaluations

5 points

1. Écrire une classe abstraite `Evaluation` (intitulé, coefficient) avec les fonctions virtuelles pures `double note() const` et `string type() const`, une méthode `afficher()` et un destructeur virtuel.
2. Écrire les classes concrètes `ControleContinu`, `Examen` (avec une méthode `rattraper(double)`) et `TravauxPratiques` (avec une méthode `ajouterCR(double)` ; les notes sont rangées dans un `vector<double>`).
3. Ajouter une fonction virtuelle pure `Evaluation *cloner() const` qui renvoie une copie dynamique de l'objet. À quoi servira-t-elle ?

Exercice 2 — La classe `Etudiant`

5 points

1. Un étudiant (matricule, nom) possède ses évaluations, rangées dans un `vector<Evaluation*>`. Surdéfinir `operator+=(Evaluation *)` pour ajouter une évaluation ; l'étudiant devient responsable de sa libération.
2. Écrire le destructeur, le constructeur de recopie et l'opérateur d'affectation (règle des trois). Pourquoi la recopie doit-elle utiliser `cloner()` ?
3. Écrire `moyenne()`, `bulletin()`, l'opérateur `<` (comparaison des moyennes) et l'opérateur `<<`.

Exercice 3 — Conteneur générique `Registre<T>`

5 points

1. Écrire une classe template `Registre<T>` qui encapsule un `vector<T>` et fournit `ajouter`, `taille`, l'opérateur `[]`, `afficher()` et `trierDecroissant()` (à l'aide de `std::sort` et de l'opérateur `<` du type `T`).
2. Ajouter deux fonctions membres templates `compter(Predicat)` et `chercher(Predicat)` qui reçoivent un critère sous forme de fonction ou de lambda.

Exercice 4 — Programme principal et rapport 5 points

1. Créer une promotion d'au moins 4 étudiants, dont un avec rattrapage et un avec des TP, afficher un bulletin, le classement, le nombre d'admis, et rechercher un étudiant par son nom.
2. *Bonus :* lire les notes depuis un fichier texte et enregistrer le classement dans un fichier (`<fstream>`).
3. Rédiger un rapport de 3 pages au maximum : diagramme de classes UML, justification des choix (classe abstraite, règle des trois, templates), jeux d'essai.

À rendre : les sources, un `Makefile`, le rapport en PDF. Une soutenance de 10 minutes par binôme a lieu à la séance 9.

TP 7 — Templates : tampon circulaire et moyenne glissante

Séance 9 — 1 heure — Chapitre 8

À retenir

Objectifs. Écrire une classe template à paramètre de type et à paramètre expression ; écrire des fonctions templates qui s'appliquent à tout conteneur offrant une interface donnée ; réutiliser un même patron avec plusieurs types.

Contexte. Une carte d'acquisition lit une température toutes les secondes. Pour éliminer les mesures parasites, on affiche la **moyenne glissante** des N dernières mesures, conservées dans un **tampon circulaire** : quand il est plein, chaque nouvelle mesure remplace la plus ancienne.

Exercice 1 — Classe template `TamponCirculaire<T, N>`

1. Écrire la classe template `TamponCirculaire<T, N>` : un tableau `T donnees[N]`, l'indice `debut` du plus ancien élément et le nombre `nb` d'éléments présents.
2. Écrire les méthodes `ajouter(const T &)`, `taille()`, `plein()`, `vider()`, la méthode statique `capacite()` et l'opérateur `[]` (en lecture seule) tel que `t[0]` soit l'élément le plus ancien.
3. Surdéfinir `<<` pour afficher le contenu dans l'ordre chronologique.
4. Écrire deux fonctions templates `moyenne(const C &)` et `maximum(const C &)` qui acceptent **n'importe quel** conteneur `C` offrant `taille()` et l'opérateur `[]`.
5. Tester avec un tampon de 4 `double` alimenté par les mesures `20.1 20.4 35.0 20.3 20.6 20.2 20.5` (35,0 est une mesure parasite) : afficher après chaque ajout le contenu du tampon et la moyenne glissante.
6. Réutiliser le même patron pour mémoriser les 3 derniers événements (`string`) d'un journal d'alarmes, puis des codes entiers. Quelle condition un type doit-il remplir pour `maximum` ? Pour `moyenne` ?
7. Les fonctions `moyenne` et `maximum` fonctionnent-elles avec la classe `Tableau<T>` du TD 8 ? Pourquoi ?

À rendre : `tampon.cpp` et les réponses aux questions 6 et 7.