# Fiche du cours

**60 h**

**Titre :**

CPP200 - Programmation C++ 2

**Description :**

Passage au C++ moderne intermédiaire : POO avancée (polymorphisme propre, Rule of Five/Zero, move semantics), gestion des ressources sûre (RAII, unique_ptr/shared_ptr), STL avancée (maps/sets, itérateurs, algorithmes), templates (fonction/classe) et concepts C++20, ranges (std::ranges), gestion d'erreurs (exceptions, noexcept, std::optional / expected), I/O & sérialisation (texte/binaire/JSON), et qualité (tests, clang-tidy, Sanitizers). Mini-projet : petite bibliothèque ou appli console modulaire, testée et outillée.

**Objectifs :**

- Concevoir des types robustes : Rule of Five/Zero, move semantics, invariants, pimpl (aperçu).*
- Utiliser STL avancée (associatifs, vues) + algorithmes de manière idiomatique.*
- Écrire du code générique : templates, type_traits, concepts C++20, ranges.*
- Structurer les erreurs : exceptions, noexcept, retours optionnels (std::optional) et expected-like.*
- Lire/écrire flux texte & binaire, sérialiser JSON ; gérer chrono (dates/durées).*
- Renforcer la qualité : tests (Catch2/GoogleTest), clang-tidy/format, ASan/UBSan, logs (spdlog – option)

**Chapitres :**

1. POO avancée I : héritage, virtual, override/final, destructeurs virtuels
2. Valeur & Ressources : Rule of Five/Zero, move, unique_ptr/shared_ptr/weak_ptr
3. STL II : map/unordered_map, set/unordered_set, deque, itérateurs & vues
4. Algorithmes & ranges : std::ranges (views, pipelines), projections & prédicats
5. Templates I : fonctions & classes, déduction, spécialisation (bases), type_traits
6. Concepts C++20 : contraintes, requires, signatures propres pour le générique
7. Erreurs & robustesse : exceptions, noexcept, std::optional, expected (C++23/alt lib)
8. I/O & sérialisation : flux, binaire, JSON (ex. nlohmann::json), std::chrono (temps & dates)
9. Qualité & outillage : clang-tidy/format, Sanitizers, logs (spdlog), structure projet simple (CMake aperçu)
10. Atelier intégration : refactor guidé + préparation mini-projet (API claire, tests, docs)*

11. Performance & profilage (copies vs moves, allocations, std::chrono, Google Benchmark, premiers profils)*

12. Packaging & CI (cibles CMake INTERFACE/PUBLIC/PRIVATE, install/export, vcpkg, GitHub Actions de base)

**À la fin :**

Vous saurez concevoir des types et des API idiomatiques, gérer ressources & erreurs sans fuite (RAII, unique_ptr/shared_ptr, exceptions, noexcept, optional/expected), exploiter STL/algorithmes/ranges de façon efficace et écrire du générique propre (templates, concepts C++20). Vous saurez aussi sérialiser (texte/binaire/JSON), structurer un petit module réutilisable avec une API claire, et renforcer la qualité (tests unitaires, clang-tidy, Sanitizers). Vous livrerez un mini-projet structuré, testé et outillé, accompagné d’un README et d’exemples d’usage — prêt à intégrer dans un codebase réel.