

Formation C++ moderne : de C++11 à C++23

Réf. SF34759

Passez d'un C++ « à l'ancienne » à un code moderne, plus sûr et plus expressif, en tirant parti des apports des normes C++11 à C++23.

3 j
Durée

1 950 € HT
Inter / participant

0
Sessions à venir

Objectifs pédagogiques

- ✓ Moderniser un code C++ existant en appliquant les apports du langage des normes C++11 à C++23
- ✓ Gérer la propriété des ressources avec la sémantique de déplacement, le RAII et les pointeurs intelligents
- ✓ Concevoir du code générique contraint avec les templates variadiques et les concepts
- ✓ Utiliser la bibliothèque standard moderne (ranges, formatage, `optional`, `variant`, `expected`) pour écrire un code plus expressif et plus sûr
- ✓ Implémenter des traitements concurrents avec les outils standard du langage et des générateurs à base de coroutines
- ✓ Structurer un projet avec CMake et les modules, en s'appuyant sur les outils de détection d'erreurs (sanitizers, analyse statique)

Public visé & prérequis

Public visé Architectes logiciels, Développeurs

Prérequis- Pratique du C++ : classes, héritage, polymorphisme, templates simples, conteneurs et algorithmes de base de la STL - Avoir suivi la formation C++ ou posséder un niveau équivalent

Programme détaillé

Panorama du C++ moderne et environnement de développement

- Évolution des normes de C++11 à C++23 : rythme triennal, apports majeurs de chaque version et éléments dépréciés ou supprimés (`auto\_ptr`, `register`)
- Perspectives C++26 : réflexion statique et contrats, ce qui se prépare pour les prochaines années
- Support des normes par les compilateurs : GCC, Clang et MSVC, choix de la norme à la compilation et détection de fonctionnalités (feature-test macros)
- CMake moderne : configurer un projet par cibles et fixer la norme du langage de façon portable
- C++ Core Guidelines : les principes qui guident l'écriture d'un C++ sûr et lisible, fil conducteur de la formation

Activités pratiques :

- Mise en place du projet fil rouge : compilation d'une base de code legacy dans plusieurs normes et lecture des avertissements du compilateur
- Configuration d'un projet CMake ciblant une norme récente

Le socle du C++ moderne

- Inférence de types : `auto`, `decltype`, structured bindings et déduction des arguments de templates de classe
- Initialisation uniforme : accolades, `initializer\_list`, initialisation des membres et pièges de la conversion implicite
- Types plus sûrs : `nullptr`, `enum class`, littéraux et séparateurs de chiffres
- Contrôle des classes : `override`, `final`, `= default`, `= delete`, constructeurs délégués et hérités
- Syntaxe simplifiée : boucle range-based for, `if` et `switch` avec initialisation, attributs (`[[nodiscard]]`, `[[maybe\_unused]]`, `[[likely]]`)
- Calcul à la compilation : `constexpr`, `if constexpr`, `constexpr` et `constinit`

Activités pratiques :

- Modernisation d'un module du projet fil rouge écrit en C++03 : type, initialisation et spécificateurs de classe
- Remplacement de macros et de calculs d'exécution par des expressions `constexpr`

Sémantique de déplacement et gestion des ressources

- Copie et déplacement : références rvalue, `std::move` et ce qu'il fait réellement
- Constructeur et affectation par déplacement : règle de zéro et règle de cinq
- Perfect forwarding : références universelles, `std::forward` et fonctions génériques de transmission
- Élisage de copie et RVO garanti : quand le compilateur évite toute copie
- RAII et pointeurs intelligents : `unique_ptr`, `shared_ptr`, `weak_ptr`, `make_unique`, `make_shared` et deleteurs personnalisés
- Détection des erreurs mémoire : sanitizers (AddressSanitizer, UndefinedBehaviorSanitizer) pour repérer fuites, accès invalides et comportements indéfinis

Activités pratiques :

- Suppression des `new` et `delete` d'une classe legacy au profit des pointeurs intelligents
- Ajout d'un constructeur de déplacement et mesure du gain sur un conteneur volumineux
- Diagnostic d'une fuite et d'un accès hors limites avec les sanitizers

Lambdas et types vocabulaire de la bibliothèque standard

- Expressions lambda : syntaxe, captures par valeur et par référence, lambdas génériques et mutables
- `std::function` : stocker et transmettre des fonctions, coût et alternatives
- Valeurs optionnelles et alternatives : `std::optional`, `std::variant` avec `std::visit`, `std::any`
- Vues non propriétaires : `std::string_view` et `std::span`, gains de performance et règles de durée de vie

Activités pratiques :

- Refactoring d'une API à pointeurs nuls et codes de retour vers `optional` et `variant`
- Remplacement de foncteurs et de callbacks par des lambdas

Programmation générique et concepts

- Templates variadiques : paramètres en nombre variable, packs et fold expressions
- De SFINAE à `if constexpr` : simplifier la spécialisation selon le type
- Concepts et clause `requires` : contraindre un template et obtenir des messages d'erreur lisibles
- Concepts de la bibliothèque standard : `std::integral`, `std::ranges::range` et autres contraintes prêtes à l'emploi
- Opérateur de comparaison à trois voies `<<` : générer les comparaisons d'une classe
- Deducing this (C++23) : paramètre objet explicite pour factoriser les surcharges et écrire des interfaces fluides

Activités pratiques :

- Sécurisation d'une API template avec des concepts et comparaison des diagnostics avant et après
- Écriture d'une fonction variadique de journalisation

Bibliothèque standard moderne : ranges, formatage et gestion d'erreurs

- Ranges et views : algorithmes sur plages, évaluation paresseuse, projections et pipelines avec l'opérateur `|`
- Vues récentes (C++23) : `zip`, `chunk`, `slide`, `join_with` et conversion vers un conteneur avec `ranges::to`
- Formatage moderne : `std::format` et `std::print` (C++23) en remplacement de `printf` et des flux
- Gestion d'erreurs avec `std::expected` (C++23) : retourner une valeur ou une erreur sans exception
- Conteneurs et utilitaires récents : `flat_map` et `flat_set`, `mdspan`, `chrono` et `filesystem`

Activités pratiques :

- Réécriture d'un traitement « boucles et variables temporaires » en pipeline de ranges, puis comparaison de lisibilité et de performance
- Remplacement de codes d'erreur par `std::expected` dans une couche d'entrées-sorties

Concurrence et coroutines

- Threads standard : `std::thread`, `std::jthread` et arrêt coopératif
- Synchronisation : mutex, `lock_guard`, `scoped_lock`, `condition_variable` et opérations atomiques
- Tâches asynchrones : `std::async`, `future` et `promise`
- Primitives de coordination : latch, barrier et sémaphores
- Algorithmes parallèles de la bibliothèque standard : politiques d'exécution
- Coroutines : `co_await`, `co_yield`, `co_return`, mécanique de suspension et générateurs avec `std::generator` (C++23)

Activités pratiques :

- Parallélisation d'un traitement séquentiel et mesure du gain obtenu

- Écriture d'un générateur de données à base de coroutine

Modules et outillage du projet

- Modules : ``export module``, séparation interface et implémentation, partitions
- ``import`` contre ``#include`` : effets sur le temps de compilation et sur l'isolation des dépendances, ``import std`` (C++23)
- Migration progressive des en-têtes vers les modules : stratégie et état du support dans les compilateurs et CMake
- Analyse statique : vérifications automatiques alignées sur les C++ Core Guidelines (clang-tidy)
- Tests unitaires : intégrer un framework de test au projet CMake pour sécuriser la modernisation

Activités pratiques :

- Transformation d'une bibliothèque du projet fil rouge en module avec une partition
- Passage de l'analyse statique sur le projet modernisé et correction des avertissements prioritaires

Pourquoi choisir Sparks Formation ?

20 ans d'expertise IT, des formateurs experts et des sessions garanties contre l'annulation.

20 ans
d'expertise

1 068+
formations

11 178
apprenants formés/an

+67
Net Promoter Score

Avis des participants

4,9/5 — 110 avis

Selim B. — Très bonne formation, très instructif même après quelques années de Python dans les bras. Une très petite équipe de stagiaires qui facilite les échanges. Je redoutais le mode...

Alexis C. — Anthony explique bien les concepts attendus et prend le temps de répondre aux questions et aux différents problèmes rencontrés. Le support présenté créé par lui-même est différent des...

Ils nous font confiance

Cette année notre filière Langages & Écosystèmes Backend a accompagné plus de 292 entreprises dont :

SOPRA STERIA RENAULT SAS ORANGE CAMPUS TECH ORANGE INNOVATION SCHOOL STMICROELECTRONICS GRANT THORNTON FRANCE
ORANGE BUSINESS RTE SII ILE DE FRANCE THALES GLOBAL SERVICES +282 autres

Financement & certifications

Cette formation est éligible aux financements OPCO et FNE-Formation. Nos équipes vous accompagnent dans le montage de votre dossier.



Réservez votre place dès maintenant

20 ans d'expertise · 1 000+ formations · NPS +67 · réponse sous 24h ouvrées

0 805 950 800

demande@sparks-formation.com