DAWAN

Plan de la formation

www.dawan.fr

Formation C++ : Evolutions des versions 11 à 23

Durée :	3 jours (21 heures)
Tarifs inter-entreprise :	2 475,00 € HT (standard) 1 980,00 € HT (remisé)
Public :	Développeurs C++
Pré-requis :	Connaissance de C++
Objectifs :	Apprendre les nouveautés / évolutions sur les versions des dernières années
Modalités pédagogiques, techniques et d'encadrement :	• Formation synchrone en présentiel et distanciel. • Méthodologie basée sur l'Active Learning : 75 % de pratique minimum. • Un PC par participant en présentiel, possibilité de mettre à disposition en bureau à distance un PC et l'environnement adéquat. • Un formateur expert.
Modalités d'évaluation :	• Définition des besoins et attentes des apprenants en amont de la formation. • Auto-positionnement à l'entrée et la sortie de la formation. • Suivi continu par les formateurs durant les ateliers pratiques. • Évaluation à chaud de l'adéquation au besoin professionnel des apprenants le dernier jour de formation.
Sanction :	Attestation de fin de formation mentionnant le résultat des acquis
Référence :	PRO102895-F
Note de satisfaction des participants:	4,43 / 5
Contacts :	commercial@dawan.fr - 09 72 37 73 73
Modalités d'accès :	Possibilité de faire un devis en ligne (www.dawan.fr, moncompteformation.gouv.fr, maformation.fr, etc.) ou en appelant au standard.

Délais d'accès :	Variable selon le type de financement.
Accessibilité :	Si vous êtes en situation de handicap, nous sommes en mesure de vous accueillir, n'hésitez pas à nous contacter à referenthandicap@dawan.fr, nous étudierons ensemble vos besoins

Identifier les évolutions clés du core language

Comprendre l'approche "feature testing" et le principe de détection de fonctionnalités à la compilation
Exploiter l'opérateur spaceship et les catégories de comparaison
Renforcer la programmation à la compilation avec constexpr, consteval et constinit
Écrire plus simplement avec les fonctions génériques abrégées (auto) et les lambdas génériques étendues
Maîtriser l'initialisation explicite des membres de structures et l'usage de using enum
Approfondir les apports de C++23 : if consteval, littéraux améliorés, déduction de this, approche fluent / builder pattern, nouveaux attributs [[likely]], [[unlikely]], [[assume]], opérateur () statique
Atelier pratique : moderniser un code C++11 vers C++20/23 en appliquant constexpr/consteval, spaceship, deducing this et attributs de branchement

Structurer les templates avec les concepts

Comprendre l'objectif des concepts : contraindre les templates et améliorer la lisibilité
Replacer les concepts dans l'évolution C++11 à C++23
Définir un concept et l'appliquer sur des fonctions et des classes templates
Utiliser les concepts de la bibliothèque standard
Mettre en œuvre requires et les expressions associées
Exprimer des restrictions sur types avec requires (requires {…}) et des expressions multiples
Gérer le support des spécificateurs de fonction dans les contraintes
Exploiter les apports C++23 : nouveaux concepts STL, concepts et lambdas, déduction améliorée de this (deducing this), alias de types et concepts, diagnostics plus explicites
Atelier pratique : sécuriser une API template avec concepts/requires et comparer les diagnostics avant/après

Exploiter ranges et views pour écrire des algorithmes expressifs

Comprendre la simplification de l’usage des algorithmes via ranges
Maîtriser les concepts fondamentaux des ranges et des views
Utiliser les projections et raisonner sur l’évaluation paresseuse (lazy mode)
Composer des générateurs infinis et finis
Assembler des pipelines de views : combine, filter, transform, take, drop…
Arbitrer élégance versus performances dans des chaînes de transformations
Approfondir C++23 : vues enrichies (chunk, slide, zip, join_with…), conversions ranges vers conteneurs, nouveaux algorithmes (contains, starts_with, iota…), création de vues personnalisées
Atelier pratique : refactorer un traitement de données “boucles + temporaires” en pipeline ranges/views, puis mesurer l’impact lisibilité/performance

Adopter les modules et préparer la migration depuis les headers

Revenir sur le modèle historique des headers et ses limites (temps de compilation, dépendances)
Comprendre l’anatomie d’un module et la notion d’interface (export module)
Séparer interface et implémentation de module
Comparer import et include et définir une stratégie d’importation
Utiliser les partitions de modules et le module global fragment
Définir des stratégies de migration progressive headers vers modules
Intégrer les apports C++23 dans une base modulaire : constexpr, if consteval, deducing this
Atelier pratique : modulariser une librairie interne (1 module exporté, 1 partition, migration progressive depuis un header)

Mettre à profit les nouveautés de la bibliothèque standard

Découvrir les nouveaux conteneurs et adaptateurs de données contiguës : flat_map, flat_set
Exploiter de nouveaux algorithmes et méthodes : contains, starts_with…
Utiliser un formatage moderne avec placeholders
Manipuler les utilitaires et types récents : expected<T, E> en C++23, std::print, std::span
Comprendre les apports modernes en concurrence : jthread, std::barrier
Atelier pratique : améliorer une couche “I/O + erreurs + concurrence” avec expected, print, span, jthread et barrier

Introduire les coroutines et choisir les bons cas d’usage

Comprendre le principe : suspension et reprise automatique d’une fonction
Identifier les concepts-clés : handle, promise, awaitable, points de suspension/reprise
Manipuler les mots-clés C++20 : co_return, co_yield, co_await
Construire des générateurs basés sur coroutines
Modéliser des tasks / futures et clarifier l’intégration avec l’existant
Atelier pratique : implémenter un générateur puis une task asynchrone simple, et comparer avec une approche itérateurs/futures classiques