



## Formation C++ : Modern Design (les nouveautés 11/14/17/20)

■ <b>Durée :</b>	4 jours (28 heures)
■ <b>Tarifs inter-entreprise :</b>	3 275,00 € HT (standard) 2 620,00 € HT (remisé)
■ <b>Public :</b>	Tous
■ <b>Pré-requis :</b>	Notions de C++
■ <b>Objectifs :</b>	Apprendre les nouveautés du C++
■ <b>Modalités pédagogiques, techniques et d'encadrement :</b>	<ul style="list-style-type: none"> <li>• Formation synchrone en présentiel et distanciel.</li> <li>• Méthodologie basée sur l'Active Learning : 75 % de pratique minimum.</li> <li>• Un PC par participant en présentiel, possibilité de mettre à disposition en bureau à distance un PC et l'environnement adéquat.</li> <li>• Un formateur expert.</li> </ul>
■ <b>Modalités d'évaluation :</b>	<ul style="list-style-type: none"> <li>• Définition des besoins et attentes des apprenants en amont de la formation.</li> <li>• Auto-positionnement à l'entrée et la sortie de la formation.</li> <li>• Suivi continu par les formateurs durant les ateliers pratiques.</li> <li>• Évaluation à chaud de l'adéquation au besoin professionnel des apprenants le dernier jour de formation.</li> </ul>
■ <b>Sanction :</b>	Attestation de fin de formation mentionnant le résultat des acquis
■ <b>Référence :</b>	PRO102286-F
■ <b>Note de satisfaction des participants:</b>	4,43 / 5
■ <b>Contacts :</b>	commercial@dawan.fr - 09 72 37 73 73

■ <b>Modalités d'accès :</b>	Possibilité de faire un devis en ligne ( <a href="http://www.dawan.fr">www.dawan.fr</a> , <a href="http://moncompteformation.gouv.fr">moncompteformation.gouv.fr</a> , <a href="http://maformation.fr">maformation.fr</a> , etc.) ou en appelant au standard.
■ <b>Délais d'accès :</b>	Variable selon le type de financement.
■ <b>Accessibilité :</b>	Si vous êtes en situation de handicap, nous sommes en mesure de vous accueillir, n'hésitez pas à nous contacter à <a href="mailto:referenthandicap@dawan.fr">referenthandicap@dawan.fr</a> , nous étudierons ensemble vos besoins

## Découvrir les nouveautés fondamentales du Core Language

Le littéral nullptr et les types normalisés (uint\_8, uint64\_t, ...), les littéraux et séparateurs

Inférence de types et de signatures avec auto

Variables templates (C++ 14)

Initialisation uniforme des variables, de tableaux et des conteneurs

Parcours unifié des tableaux et conteneurs avec La boucle "range based" for

Listes d'initialisation avec initializer\_list

Énumérations fortement typées (C++11/17)

Littéraux personnalisés

Amélioration du contrôle de flux avec les Init-statements (C++17/20)

L'opérateur sizeof appliqué aux membres d'un objet ou d'une classe

Contrôle de l'alignement mémoire

L'opérateur decltype

Déconstruction avec les structured bindings (C++17)

Les spécificateurs de classe (override, default, delete, final)

Constructeur délégué et constructeur hérité

Initialisation de membres

Constructeur explicite multi-paramétré

Abraham's Exception safety guarantees et la clause noexcept

Données inline

Métaprogrammation avec constexpr

Les spécificateurs constexpr et constexpr (C++20)

Les attributs (C++11/17/20)

L'opérateur de comparaison « Three Way » <=> (C++20)

Paramètres nommés pour la construction des structures (C++20)

Améliorations apportées aux fonctions génériques et aux lambdas (C++17/20)

## Découvrir les nouveautés de la librairie standard

Les nouveaux itérateurs  
Les tableaux à taille fixe avec `std::array`  
La classe `std::string_view` (C++ 17)  
La classe `std::span` (C++ 20)  
Nouveautés de la classe `std::string` (C++17)  
Les classes `std::variant`, `std::any` et `std::optional` (C++17)  
Les nouvelles collections associatives : `unordered_map/set` et `hash`  
Singly-Linked Lists  
Le conteneur `tuple`  
Les nouveaux algorithmes ensemblistes  
Gestion du temps, l'espace de nom `chrono`  
Générateurs / distributions de nombre aléatoires  
La librairie `Ranges`, les vues et les adaptateurs de vues (C++20)  
Pipelining avec les nouveaux algorithmes sur vues (C++20)  
Sorties formatées avec `std::format` et les placeholders (C++20)  
Constructeurs `constexpr` des conteneurs  
La librairie `filesystem` (C++17)

## **Move semantics**

Copie versus déplacement  
Value et RValue reference  
La fonction `std::move`  
Move constructor et move assignment operator  
Complétude étendue des classes  
R-value reference et STL  
STL C++11 et `swap / move`  
Signature reference qualifieurs  
Perfect forwarding avec `std::forward`  
Références `forward` et `collapsing`  
Copy elision et Guaranteed RVO (C++17)

## **Gérer des ressources**

L'idiome RAII (Resource Acquisition Is Initialization)  
Propriété et transfert de responsabilité  
La classe `unique_ptr`  
Comptage de références avec la classe `shared_ptr`

Custom deleter

Les fonctions `make_unique` et `make_shared`

La classe `weak_ptr` et le référencement circulaire

## **Maîtriser la programmation fonctionnelle**

Problématique de l'abonnement

Pointeur de fonction / méthode / membre statique

Les classe fonction et `mem_fn`

Binding, placeholders

Adaptateurs de références

## **Utiliser la généricité**

Typage multiple avec les mixins

Paramétrage et spécialisation des méthodes

Héritage / containment et généricité

Métaprogrammation

L'idiome CRTP Curiously Recursive Template Pattern

Typologie C++ et classes de traits

Assertions statiques avec `static_assert`

Extended friend declaration

Les variadic templates, pattern matching et héritage multiple

`constexpr_if` (C++17)

Implémentation générique du pattern visitor avec `std::visit` (C++17)

Définition de contrats génériques avec les concepts (C++20)

Mise en œuvre de contraintes avec `requires` (C++20)

Concepts prédéfinis de la librairie standard (C++20)

## **Créer des modules (C++20)**

Unités d'interface et d'implémentation

Sous modules et partitions

Module linkage

## **Gérer le Multithreading et concurrence**

Démarrage et détachement d'un thread

Threads et gestion des exceptions

La classe `std::call_once`

L'espace de noms `this_thread`

Futures / promises et `packaged_task`

Les futures, politique de démarrage (parallèle ou asynchrone) et gestion des exceptions

Partage de ressources et mécanismes de synchronisation

Mutexes données atomiques (C++11/20)

Unique-lock et `lock_guard`

La classe `std::jthread` (C++20)

Synchronisation avec les sémaphores et les mécanismes latches et barriers (C++17/20)

Flux de sortie synchronisés (C++ 20)