CROSSTHINK

C++ expert, les avancées du langage

Cette formation vous permettra d'assimiler les nouveautés introduites par le standard C++ 2011. Vous découvrirez les expressions lambda, exploiterez les possibilités de la programmation fonctionnelle, maîtriserez la gestion de la mémoire et exploiterez les autres nouveautés de la bibliothèque standard C++.

Objectifs

•Appréhender les nouveautés et les améliorations du langage C++11
•Utiliser les lambda expressions
•Exploiterez les possibilités de la programmation fonctionnelle
•Maîtriser les allocations-destructions d'objets
•Développer une application multithreadée

Moyens pédagogiques

•Présentation du formateur et du programme.
•Présentation et écoute de chacun de stagiaires.
•Apports didactiques pour apporter des connaissances communes.
•Mises en situation de réflexion sur le thème du stage et des cas concrets.
•Méthodologie d'apprentissage interactive et participative.
•Exercices et études de cas concrets.
•Temps d'échanges.
•Accompagnement pédagogique individualisé.

Formateur

Les formateurs de CROSSTHINK sont des experts de leur domaine, disposant d'une expérience terrain qu'ils enrichissent continuellement. Leurs connaissances techniques et pédagogiques sont rigoureusement validées en amont en interne.

Suivi de l'execution et évaluation des résultats

•Accueil des stagiaires dans une salle dédiée à la formation / A distance
•Feuilles de présence.
•Documents supports de formation projetés.
•Mise à disposition du stagiaire des documents et supports de formation.
•Tout au long et/ou à l'issue de la formation : Evaluation des acquis des stagiaires via des exercices, des QCM, des QUIZZ, des mises en situation et/ou des cas pratiques.
•Enquête de satisfaction.
•Attestation de fin de formation.

EN BREF

Durée : 3 jours

Tarif : 1490€

Public et pré requis

Cette formation ne nécessite pas de prérequis.

FORMATIONS A DISTANCE

Contactez-nous
contact@crossthink.fr

PROCHAINES DATES

Nous consulter

CROSSTHINK

C++ expert, les avancées du langage

Programme

L'avènement de C++11

•Les différentes normes C++98, C++03 et C++0x, C++11.
•Les nouveautés de C++11 et les objectifs de cette norme. Le devenir de Boost, STL.
•La question de la compatibilité des codes anciens.
•La disponibilité des outils de développement (compilateurs, débogueurs, IDE ...).
Travaux pratiques
Vérification de l'outillage à l'aide d'un code C++11 fourni.

Les améliorations du langage

•Les énumérations fortement typées.
•Les tableaux à taille fixe.
•Le mot-clé auto pour simplifier le typage.
•La boucle basée sur un intervalle.
•Autres améliorations : templates à arguments variables, pointeur nul, littéraux...
Travaux pratiques
Mise en œuvre des améliorations.

Les modifications au niveau des classes

•La délégation de constructeurs, les contraintes liées à l'héritage.
•La nouvelle sémantique du déplacement et le constructeur par déplacement (move constructor).
•Adaptation de la forme normale des classes aux nouveautés (move constructor).
•Les directives =delete, =default.
•Les initialiseurs de conteneurs.
•LeS données membres.
Travaux pratiques
Création de classes C++11.

L'utilisation des threads

•Déclaration et exécution d'un thread. Attente de fin d'exécution avec join().
•La gestion des données locales à un thread, l'usage de volatile.
•Récupérer un résultat avec future<> et async().
•Obtenir des informations sur les capacités d'exécution de la plateforme avec hardware_concurrency().
Travaux pratiques
Multithreader un code séquentiel et mesurer le gain en termes de temps d'exécution.

Autres nouveautés de la bibliothèque standard

•La gestion du temps avec le namespace chrono.
•Le nouveau conteneur tuple.
Travaux pratiques
Mise en œuvre des nouveautés.

La programmation fonctionnelle avec les lambda expressions

•Déclaration, typage, implémentation et utilisation.

•L’intérêt d’auto avec les lambda-expressions.
•La gestion des fermetures (closures), avec capture par valeur ou par référence des variables liées au contexte.
Travaux pratiques
Exercices de programmation fonctionnelle.

La gestion mémoire et les conteneurs

•Les smart pointers : shared_ptr, weak_ptr, unique_ptr, auto_ptr. Usage conjoint avec la STL.
Travaux pratiques
Mise en œuvre de la gestion mémoire C++11.