CROSSTHINK

Programmation C++, perfectionnement

En constante évolution, le langage C++ offre des mécanismes qui permettent une conception robuste et très riche. Les récentes normes C++ intègrent une part importante des bibliothèques du projet Boost et améliorent notablement la Standard Template Library (STL). Cette formation vous permettra d'approfondir la conception en C++ par l'apprentissage des dernières évolutions du langage et l'utilisation effective de la STL.

Public et pré-requis

Cette formation ne nécessite pas de prérequis.

Objectifs

•Découvrir les nouveautés apportées par la version C++11
•Maîtriser la gestion de la mémoire, des pointeurs et des références
•Implémenter la généricité en C++
•Découvrir la bibliothèque standard STL
•Utiliser la librairie Boost et C++11

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 constante évolution, le langage C++ offre des mécanismes qui permettent une conception robuste et très riche. Les récentes normes C++ intègrent une part importante des bibliothèques du projet Boost et améliorent notablement la Standard Template Library (STL). Cette formation vous permettra d'approfondir la conception en C++ par l'apprentissage des dernières évolutions du langage et l'utilisation effective de la STL.

EN BREF

3 jours

1490€

CONTACT

contact@crossthink.fr

04.83.43.47.48

PROCHAINES DATES

Nous consulter

CROSSTHINK

Programmation C++, perfectionnement

Programme

Rappels
•Classes d'allocation mémoire.
•Construction, initialisation, embarquement d'objets.
•Les fuites mémoire.
•Constance, le mot-clé mutable, Lazy Computation.
•Amitié (friendship) C++ et contrôle d'accès.
•Destruction virtuelle.
•Stratégie de gestion des exceptions.
•Les espaces de nommage (namespace).

Les nouveautés langage de C++11
•nullptr et autres littéraux.
•Les directives =delete, =default.
•Délégation de constructeurs.
•Les énumérations "type safe".
•Le mot-clé auto et boucle sur un intervalle.
•Référence rvalue et impact sur la forme normale des classes C++.
•Les lambda expressions.
Travaux pratiques*Réécriture d'un code C++ existant en C++11, comparaison des deux implémentations.*

Gestion des opérateurs
•Opérateurs binaires et unaires.
•L'opérateur d'indirection, cas d'usage.
•L'opérateur de référencement.
•Les opérateurs d'incrémentation/décrémentation préfixés et post-fixés.
•Les autres opérateurs : comparaison, affectation…
•La surcharge de l'opérateur [], des opérateurs d'insertion (<<) et d'extraction (>>).
•Les foncteurs et la surcharge de l'opérateur (), avantage par rapport aux fonctions.
Travaux pratiques*Création de foncteurs et de proxies (libération mémoire, comptage de références) avec les opérateurs étudiés.*

Conversion et RTTI
•Opérateurs de conversion. Constructions implicites, le mot-clé explicit.
•Les opérateurs de casting const_cast, static_cast, reinterpret_cast.
•Conversion dynamique et Runtime Type Information.
•L'opérateur typeid, les exceptions liées.
•La classe type_info.
•Contrôle du "downcasting" à l'aide de l'opérateur dynamic_cast.
Travaux pratiques*Mise en œuvre des idiomes "is-a" et "is-kind-of" avec dynamic_cast.*

La généricité
•Introduction aux patrons de classe. Généricité et préprocesseur.
•Fonction générique. Classe générique. Composition générique. Généralisation générique.
•Spécialisation partielle et totale.
•Introduction à la méta-programmation.
•La généricité, principe fédérateur des librairies STL et Boost.
Travaux pratiques*Démarrage de l'étude de cas qui sera complétée avec la STL et Boost. Mise en œuvre de la composition et de la généralisation génériques. Création de plug-ins génériques.*

La STL (Standard Template Library)
•Composants de la STL : types complémentaires, conteneurs, algorithmes, itérateurs, objets fonctions, les adaptateurs.
•Les chaînes de caractères STL, la classe template basic_string et ses spécialisations.

CROSSTHINK

Programmation C++, perfectionnement

•Les conteneurs séquentiels et associatifs : définition, rôle et critères de choix.
•Les allocateurs et la gestion de la mémoire des conteneurs.
•Les méthodes d'insertion, de suppression, d'itération et d'accès aux principaux conteneurs : Vector, List, Set, Stack…
•Le concept d'itérateur. Parcours d'un conteneur.
•Les différents groupes d'algorithmes STL : non mutants, mutants, de tri et de fusion, numériques.
•Manipulation de conteneurs (manipulation, recherche de valeurs…).
•Paramétrer les algorithmes génériques par des objets "fonction".
•Les "adapteurs" et la modification du comportement d'un composant.
•La STL et les traitements sur les flux (fichiers, mémoire…).
•Principe du RAII : les pointeurs automatiques et la classe auto_ptr.
•Les exceptions standard de la STL.
Travaux pratiques*Implémentation des relations avec les collections de la STL. Utilisation d'algorithmes standard quelconques.*

Les nouveautés C++11 de la librairie standard

•Evolution historique : Boost –> TR1 –> C++11.
•Les nouveaux conteneurs : array, forward_list, unordered_set, unordered_map.
•La classe tuple.
•Les pointeurs intelligents (smart pointer) : shared_ptr, weak_ptr, unique_ptr.
•Les nouveaux foncteurs et binders.
•Introduction à la gestion des threads.
•Les expressions régulières.
Travaux pratiques*Mise en œuvre de la robustesse avec les smart pointers. Utilisation d'expressions régulières.*

Boost

•La Pointer Container Library (destruction des données pointées d'un conteneur).
•Les structures de données boost::any et boost::variant.
•Programmation événementielle (connexions et signaux).
•Gestion des processus, mécanismes de communication interprocessus et mémoire partagée.
Travaux pratiques*Amélioration de l'implémentation de l'étude de cas par l'utilisation la Pointer Container Library.*

Utilisation avancée de l'héritage

•Héritage versus embarquement. Héritage privé. Héritage protégé.
•Exportation de membres cachés avec la Clause Using.
•Héritage multiple et gestion des collisions de membres.
•Héritage en diamant. Héritage virtuel et dynamic_cast.
•Principes de conception : substitution de Liskov, principe d'ouverture/fermeture, inversion des dépendances.
•Règles d'implémentation des interfaces en C++.
Travaux pratiques*Combinaison de l'héritage multiple, privé et de l'exportation pour concevoir des classes robustes et hautement évolutives.*

Certification