



Référence : CPP-BPP

Niveau : 2 - Avancé

Contact : Virginie Pigeat

Durée : 3 jours (21h.)

Classe à distance : Possible

01 69 15 32 32 / 07 87 10 03 92

Tarif : 1 500 € h.t / personne

virginie.pigeat@agenium.com

Objectifs

Cette formation vise à comprendre le cycle de vie des objets en C++.

Connaître les bonnes pratiques pour écrire un code maintenable et performant. Gagner en performance en écrivant un code parallèle.

Public

Cette formation C++ s'adresse aux développeurs, ayant une bonne connaissance de base du C++.

Pré-requis

Pour suivre cette formation, les participants doivent disposer d'une expérience de programmation en C++.

Travaux pratiques

Il s'agit d'un cours avec une grande partie consacrée aux travaux dirigés (plus de 50%).

Moyens pédagogiques et techniques

Les formations Agenium Campus sont conçues et animées par des experts en activité.

Nos salles sont équipées de vidéoprojecteur et écran/tableau et d'un accès internet. Chaque participant dispose d'un poste de travail et d'un support de cours.

Nos formations peuvent être suivies à distance.

Modalités de suivi et d'évaluation

Les participants signent une feuille de présence par demi-journée. Une attestation de validation des acquis est remise à la fin de la formation.

L'évaluation en cours de formation est réalisée grâce à des exercices ou études de cas (50% du temps minimum pour les cours pratiques) et/ou sous forme de QCM.

L'évaluation en fin de formation Un QCM ou un exercice est donné aux stagiaires après la formation afin de mesurer l'acquisition des connaissances.

Niveau de satisfaction : 5 / 5



Le contenu de nos formations est adaptable selon vos besoins



Programme :

Module 1 – CYCLE DE VIE DES OBJETS

1. Valeurs, références et pointeurs

- Valeur
 - types fondamentaux, classes et structures
 - portée
 - cycle de vie
- Référence : définition
- Pointeur : définition
- Mémoire
 - La pile / le tas
 - Implications sur la performance

2. RAII

- Définition, exemple avec `std::vector`
- Avantages
- Notes sur la performance

3. Optimisation et élision des copies

- Explication par l'exemple
- Condition
- Utilité

4. Transfert

- Explication par l'exemple
- Comment faire ?
- Exemple de réalisation
- Interaction avec l'élision & contre-exemple

5. Règle du zéro

- Historique
- Règles de 3, 5, 0

6. Comportement indéfinis

- Définition & explication
- Exemples

Module 2 – BONNES PRATIQUES

1. Commentaires dans le code

- Intention vs. description du code

2. Principle of Least Astonishment

3. Namespacing

- Utilité par l'exemple
- Utilisation
- ADL (survol)

4. Conversions implicites / explicites

- Implicite
 - Explications / exemple
 - Dangers
- Explicite
 - `const/static/reinterpret_cast/C cast`

5. Référence invalide

- Exemple
- Extension de la durée de vie des variables temporaires
- Itérateurs et modification de conteneur

6. Fuites mémoire

- Définition / exemples
- Comment les éviter ?
- Présentation de Valgrind

7. Accès mémoire non protégés

- Explications/intérêt
- Responsabilité de la ressource
- Alternatives (`std::vector`)
- Application aux conteneurs

8. `std::shared_ptr` vs. `std::weak_ptr`

- `std::shared_ptr` : définition et présentation
- `std::weak_ptr` : définition et présentation
- Conseils

Module 3 – POUR ALLER PLUS VITE / LOIN

1. Optimisations du compilateur

- Coder simple pour aller vite
- Rappel des comportements indéfinis

2. Les outils pour mieux développer

- Valgrind
- GDB
- Perf
- Godbolt

3. Calculs parallèles

- Niveaux de parallélisme
 - Machines
 - Processus
 - Coeurs
 - Instructions machine
- Designs de parallélisme
 - Threads
 - Tâches
- Survol des outils de la STL