

Formation C++ embarqué : passer du C au C++ sur microcontrôleur

Réf. SF34207

Adoptez le C++ sur vos projets embarqués pour gagner en structure et en réutilisabilité, sans sacrifier la mémoire ni les performances de vos cibles.

3 j
Durée

2 100 € HT
Inter / participant

0
Sessions à venir

Objectifs pédagogiques

- ✓ Évaluer le coût en mémoire et en temps d'exécution des fonctionnalités du C++ pour choisir celles adaptées à une cible contrainte
- ✓ Configurer une chaîne de compilation croisée et déboguer un programme C++ sur un microcontrôleur
- ✓ Concevoir des classes C++ qui encapsulent périphériques et interruptions, en interagissant avec le code C existant
- ✓ Implémenter une gestion mémoire déterministe et placer les données constantes en mémoire morte
- ✓ Développer du code générique avec les templates et la bibliothèque standard en maîtrisant la taille du binaire
- ✓ Appliquer les règles de codage MISRA C++ et la programmation défensive pour fiabiliser le logiciel embarqué

Public visé & prérequis

Public visé Développeurs

Prérequis- Bonne pratique du langage C (pointeurs, structures, compilation séparée) - Notions d'architecture des microcontrôleurs : registres, mémoires Flash et RAM, interruptions - Aucune connaissance préalable du C++ n'est requise

Programme détaillé

Du C au C++ : les fondamentaux pour l'embarqué

- Pourquoi le C++ en embarqué : principe d'abstraction à coût nul, idées reçues sur l'empreinte et les performances
- Apports du C++ au C : références, `const`, fonctions `inline`, surcharge de fonctions, espaces de noms, `enum class`, `auto`
- Name mangling et `extern « C »` : appeler la HAL constructeur et les bibliothèques C depuis le C++, et inversement
- `volatile` et accès aux registres : périphériques mappés en mémoire et pièges de l'optimiseur
- Sous-ensemble du C++ pour les petites cibles : désactivation des exceptions et du RTTI, environnement *freestanding*, héritage de l'ancienne spécification EC++

Activités pratiques :

- Comparaison du code assembleur généré par une fonction C et par son équivalent C++
- Observation du name mangling dans la table des symboles et appel d'une fonction C depuis le C++
- Mise en évidence de l'effet de `volatile` sur l'accès à un registre de périphérique

Chaîne de compilation croisée et mise au point sur cible

- Compilation croisée avec GCC pour ARM : rôles du compilateur, de l'assembleur et de l'éditeur de liens
- Sections d'un fichier objet : code, données initialisées et non initialisées, constantes, et leur répartition entre Flash et RAM
- Script de l'éditeur de liens : carte mémoire de la cible, placement des sections, symboles de début et de fin
- Fichier de démarrage (startup) : table des vecteurs, initialisation des données, appel des constructeurs des objets globaux avant `main`
- Options de compilation : niveaux d'optimisation, élimination du code mort, mesure de l'empreinte du binaire
- Construction du projet : Makefile ou CMake pour automatiser la compilation croisée
- Débogage sur cible : sonde de débogage, points d'arrêt, inspection de la mémoire et des registres

Activités pratiques :

- Compilation, chargement et débogage pas à pas d'un premier programme C++ sur carte ou émulateur
- Analyse du fichier map pour mesurer l'empreinte de chaque section

- Suivi dans le débogueur du code de démarrage jusqu'à l'exécution des constructeurs globaux

Classes et objets pour piloter le matériel

- Encapsulation : classes, membres publics et privés, fonctions membres, pointeur `this``
- Construction et destruction : constructeurs par défaut, de copie et paramétrés, listes d'initialisation, ordre d'initialisation des objets statiques
- Copie et déplacement : sémantique de déplacement, suppression explicite de la copie pour les objets liés au matériel
- RAI : acquisition et libération automatiques d'une ressource (section critique, verrou, broche configurée)
- Composition et agrégation : construire un système à partir de composants réutilisables
- Surcharge d'opérateurs : opérateurs membres et fonctions `friend``, opérateur d'affectation, passage efficace des objets en paramètre
- Interruptions en C++ : relier une routine d'interruption à une méthode d'instance, membres statiques et contraintes d'exécution
- C++ et RTOS : encapsuler tâches, mutex et files de messages dans des classes

Activités pratiques :

- Écriture de classes `Gpio`` et `Uart`` encapsulant les registres du périphérique (fil rouge du stage)
- Protection d'une ressource partagée par un verrou RAI
- Redirection d'une interruption de réception série vers une méthode de la classe pilote
- Émission et réception de chaînes de caractères sur liaison série au travers d'une classe de sortie légère

Gestion de la mémoire et données en mémoire morte

- Pile, tas et stockage statique : où vivent les objets et comment dimensionner la pile
- Risques du tas en embarqué : fragmentation, non-déterminisme, échec d'allocation
- Redéfinition de `new`` et `delete`` : opérateurs globaux et propres à une classe, interdiction de l'allocation dynamique
- Placement new et pools mémoire : allocation déterministe dans des zones réservées
- Objets constants et partiellement constants : `const``, `mutable`` et méthodes constantes
- `constexpr`` et `static_assert`` : calcul à la compilation, tables placées en Flash, vérifications avant exécution
- Objets ROMables et persistants : initialisation constante, placement dans des sections dédiées, conservation des données après un reset

Activités pratiques :

- Implémentation d'un pool d'objets de taille fixe avec placement new
- Génération d'une table de correspondance `constexpr`` et vérification de sa présence en Flash dans le fichier map
- Création d'objets constants, mutables, ROMables et persistants

Héritage et polymorphisme

- Héritage simple : spécialisation par ajout et substitution, règles d'accès, membres protégés, construction des objets dérivés
- Fonctions virtuelles et liaison tardive : redéfinition contrôlée (`override``, `final``), destructeurs virtuels
- Coût du polymorphisme : table virtuelle, pointeur caché, appel indirect et impact sur l'empreinte
- Classes abstraites et interfaces : abstraire un périphérique derrière une classe `Device`` générique
- Héritage multiple et héritage virtuel : usage limité aux interfaces, résolution des conflits de noms
- Conversions de type : `static_cast``, `reinterpret_cast`` pour l'accès aux registres, RTTI et `dynamic_cast`` à éviter sur petite cible

Activités pratiques :

- Dérivation d'une classe `Device`` générique pour plusieurs périphériques
- Mesure du surcoût mémoire et temps d'exécution introduit par les fonctions virtuelles
- Définition d'une interface commune à plusieurs capteurs

Templates et bibliothèque standard en embarqué

- Fonctions et classes templates : paramètres de type et paramètres constants (taille de tampon, adresse de registre)
- Spécialisation de templates : adapter une implémentation à un type ou à un périphérique particulier
- Code bloat : causes, mesure et techniques de factorisation
- Bibliothèque standard sur microcontrôleur : éléments adaptés (`std::array``, algorithmes, traits de type) et éléments à éviter (allocations implicites, flux d'entrées-sorties)
- Conteneurs à capacité fixe et allocateurs personnalisés : utiliser les conteneurs sans recourir au tas
- Algorithmes et lambdas : rechercher, trier et transformer des données sans boucles manuelles

Activités pratiques :

- Conception d'un tampon circulaire générique dont la capacité est fixée à la compilation
- Comparaison de la taille du binaire selon le nombre d'instanciations d'un template

- Remplacement de boucles de traitement par des algorithmes standard et des lambdas

Gestion des erreurs et code sûr

- Exceptions : lancer, capturer et relancer, classes d'exceptions standard, ``noexcept``
- Coût des exceptions sur petite cible : tables de déroulement, empreinte, désactivation à la compilation
- Alternatives aux exceptions : codes de retour, types résultat comme ``std::optional``, traitement des erreurs fatales
- Programmation défensive : assertions, vérifications à la compilation, détection des erreurs à l'exécution
- Pointeurs intelligents : ``unique_ptr`` pour une propriété exclusive et une libération garantie, y compris avec un pool
- Règles de codage MISRA C++ : principes, catégories de règles, place dans les secteurs critiques (automobile, médical, ferroviaire)
- Analyse de code : avertissements du compilateur et outils d'analyse statique

Activités pratiques :

- Comparaison de l'empreinte d'un binaire compilé avec et sans exceptions
- Refactorisation d'une gestion d'erreurs par exceptions en types résultat
- Revue d'un extrait de code au regard de règles MISRA C++

Pourquoi choisir Sparks Formation ?

20 ans d'expertise IT, des formateurs experts et des sessions garanties contre l'annulation.

20 ans
d'expertise

1 068+
formations

11 178
apprenants formés/an

+67
Net Promoter Score

Avis des participants

4,9/5 — 735 avis

Berenice P. — formatrice agréable à l'écoute bonne pédagogie j'ai beaucoup apprécié nos échanges

Anne-Lise C. — Très bonne adaptation du formateur au public et au niveau d'attente.

Ils nous font confiance

Cette année notre filière Ingénierie & Développement Embarqué a accompagné plus de 15 entreprises dont :

ACTIA AUTOMOTIVE AMCS TECHNOLOGIES ANSYS FRANCE SAS ASTEK FRANCE EXAIL AEROSPACE FLANDRIN IT MAGNA ELECTRONICS FRANCE SAS PMC FRANCE PRECIA SA RTONE +5 autres

Financement & certifications

Cette formation est éligible aux financements OPCO et FNE-Formation. Nos équipes vous accompagnent dans le montage de votre dossier.



Réservez votre place dès maintenant

20 ans d'expertise · 1 000+ formations · NPS +67 · réponse sous 24h ouvrées

0 805 950 800

demande@sparks-formation.com