

L30 - C++ classique et moderne pour l'embarqué

Objectifs

- Maîtriser les bases du langage C++
- Intégrer les templates C++ (code générique) dans les systèmes embarqués
- Maîtriser les aspects avancés du C++ tels que le polymorphisme, l'héritage simple et l'héritage multiple
- Redéfinir les opérateurs C++ d'allocation dynamique de mémoire pour l'embarqué
- Rendre les objets C++ persistants flashables et romables
- Gérer les exceptions en C++ pour sécuriser les applications embarquées
- Utiliser des objets C++ pour gérer la transmission/réception série de chaînes de caractères
- Découvrir les fonctionnalités modernes du C++
- Apprendre les changements de langage dans C++11, C++14, C++17 and C++20
- Découvrir les nouvelles fonctionnalités ajoutées à la bibliothèque standard
- Apprendre les fonctionnalités avancées du C++ moderne comme "perfect forwarding"
- Passer du C++ traditionnel au C++ moderne
- Mettre en évidence les fonctionnalités essentielles du C++ moderne utilisées dans les applications embarquées

Pré-requis

- Compétences en programmation C (voir notre cours [L2 - C language for Embedded MCUs](#))

Environnement du cours

- Cours théorique
 - Support de cours imprimé et au format PDF (en anglais)
 - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique
- Activités pratiques
 - Les activités pratiques représentent de 40% à 50% de la durée du cours
 - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
 - Un PC pour deux stagiaires pour les activités pratiques
 - Une plateforme cible pour deux stagiaires (sauf en cas d'utilisation de qemu)
 - Accès à un serveur cloud privé
 - Exemples de code, exercices et solutions
 - Le formateur assiste les stagiaires pendant les exercices
- Au début de chaque demi-journée une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus

Plan du cours

First Day

Introduction to C++ for industrial systems

- Introduction to object oriented programming
- History and definition
- Overview on C++98/C++03/C++11/C++14/C++17/C++20
- Modern C++ objectives
- Switch from C to C++
- Embedded C++
- How to write optimized embedded code

Exercise : Understand function mangling

Exercise : Function inlining

Exercise : Volatile variable handling

C++ and embedded systems

- Object Oriented Programming in C++
 - Encapsulation
 - Classes and objects
 - Attributes and member functions
 - Object construction and destruction
 - Construction parameters
 - Copy constructor
 - Object composition and container
 - Scope qualifier operator

Exercise : Declaring classes and methods

Exercise : Working with default, copy and parameterized constructors

Exercise : Understand the differences between composition and aggregation

Second Day

Operator Overloading

- Optimizing parameter object passing
- Overloading operators by member functions
- Overloading operators by friend functions
- Memory management operators overloading

Exercise : The assignment operator

Exercise : overloading operators

Simple Inheritance

- Specialization by addition and substitution
- Derivation and access rules
- Construction during inheritance
- Inheritance polymorphism
- Virtual methods

Exercise : Understand inheritance

Persistent and flashable objects

- Constant and partially constant objects
- Persistent objects
- Flashable objects

Exercise : Creating constant, mutable, persistent and ROMable objects

Enhancing security with exceptions

- Launching, capturing and handling exceptions
- Retriggering exception
- Exceptions specifications
- Handling unexpected exception
- Exception objects of the C++ standard library

Exercise : Handle errors using exceptions

Exercise : Unexpected exceptions management

Third Day

C++ advanced techniques

- Member pointers
- Generic objects and templates
 - Classes and generic functions
 - Templates overloading
 - Specializing templates
 - STL (Standard Template Library)
 - Templates in embedded systems
- Polymorphic objects
- Virtual objects and abstract classes
- Specializing objects by simple inheritance
 - Building derivate objects
 - Access control rules for inherited objects
 - Specializing objects by multiple inheritance
 - Conflicts resolution by scope operator
 - Virtual inheritance

Exercise : Generic classes and functions

Exercise : Understand virtual methods by subclassing a generic Device class

Exercise : Understand multiple inheritance and virtual bases

Fourth Day

Introduction to modern C++

- Overview
- Storage class specifiers
- Uniform initialization
- C++ Named Requirements
- Automatic type deduction
 - The auto keyword
 - The auto keyword as a return type from a function
 - Using auto for declaring function signatures
 - Automatic constant references
 - Forwarding references
 - Advantages of using auto in embedded systems

Exercise : Using auto to declare variables

Keywords

- Enum class
- override and final
- Inline variables
- nullptr
- static_assert
- noexcept
- constexpr and if constexpr
- decltype
- Defaulted and deleted functions
 - Implementing a thread-safe singleton

Exercise : Using modern C++ keywords

Exercise : Create a singleton using modern C++

New functionalities

- Structured binding
- Range-based for loops
- Nested namespaces and namespace aliases
- Alignment
 - Alignas
 - Alignof
- Move semantics and r-value references
 - Copy-constructing and Move-constructing
 - r-value references
 - Perfect forwarding

Exercise : Using the new for loop syntax

Exercise : Using std::tuple

Exercise : Move semantics performance advantages on embedded systems

Modern C++ Standard Library

- Standard Library
 - std::optional
 - std::variant
 - std::any
 - std::byte
 - std::hash
 - Filesystem library
- Literals
 - Cooked literals
 - Standard literal operators
 - Raw literals
 - Raw string literals
- Random number generation
 - Random number generation engines
 - Random number generation distributors
- Containers
 - std::array
 - std::forward_list
 - Unordered associative containers

Exercise : Using the new elements added to the standard library

Exercise : Using std::optional

Fifth Day

String Manipulation

- New string Types
 - `std::u16string`
 - `std::u32string`
- Basic string view
- Converting between numeric and string types
- Elementary string conversions
- Input/output manipulators
 - `std::get_money`, `std::put_money`
 - `std::get_time`, `std::put_time`
 - `std::quoted`
- Regular expressions
 - Format of a string
 - Parsing the content of a string
 - Replacing the content of a string

Exercise : Using String class and String literals

Concurrency and Multithreading

- Introduction
- Thread
- Atomic operations
 - Atomic features
 - Non-class functions
 - Atomic flag
 - Memory order
- Mutex
 - Avoiding using recursive mutexes
- Sending notifications between threads
- Condition variables
- Future and Promise
- Task and Async
- Modern C++ and RTOS

Exercise : Blink synchronously 4 Leds

Lambda functions

- Syntax of lambdas
- Defining lambdas
- Using lambdas
 - Using lambdas with standard algorithms
 - Assigning lambdas to function pointers
 - Lambdas and `std::function`
 - Writing a function that accepts a lambda as parameter
- Polymorphic lambdas
- Recursive lambdas

Exercise : Understanding lambda

Exercise : Using lambda to modify and display a vector

Dynamic memory management

- Memory Management
- Memory Errors

- Smart Pointers
 - Raw Pointers
 - Automatic pointers
 - Unique Pointers
 - Shared Pointers
 - Weak Pointers

Exercise : Override new and delete

Exercise : Understanding unique and shared pointers