# L10 - Programmation C++ moderne embarquée

*Le langage C++ moderne pour les systèmes embarqués*

## Objectifs

- 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

Les exercices sont conduits sur des cartes à base de microcontrôleurs STM32

## Equipement

- Matériel de cours imprimé
- Un PC par binôme
- Carte STM32F4 - ARM Cortex M4
- Environnement de développement Eclipse et compilateur GCC

## Pré-requis

- Connaissances en programmation C (voir notre cours oL2 - Langage C pour les MCUs embarqués)
- Connaissances de base du langage C++ (voir notre cours oL3 - Programmation C++ embarqué)

## Audience visée

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

## Durée

- Totale : 12 heures
- 2 sessions de 6 heures chacune (hors temps de pause)
- De 40% à 50% du temps de formation est consacré aux activités pratiques
- Certains travaux pratiques peuvent être réalisés entre les sessions et sont vérifiés par le formateur lors de la session suivante

## 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.
- 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

# Course Outline

## Premier Jour

### Introduction to modern C++

- Overview
- Storage class specifiers
- Uniform initialization
- 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

### Keywords

- enum class
- override and final
- Inline variables
- nullptr
- static_assert
- noexcept
- constexpr and if constexpr
- decltype
- Defaulted and deleted functions

### New functionalities

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

### 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

# Second Jour

## 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

## 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

## Dynamic memory management

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

## Concurrency and Multithreading

- Introduction
- Thread
- Implementing a thread-safe singleton
- 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