A.L.S.E.
http://www.alse-fr.com

# Formation Pratique

# Développement de Logiciel Embarqué en Langage C++

DOULOS

**Présentation** Depuis longtemps, les systèmes embarqués se programmaient en Langage C. Désormais, on constate que le Langage **C++** est en passe de devenir la norme, y-compris pour les systèmes très légers comme les plus petits micro-contrôleurs de l'environnement Arduino ! Le C++ (avec sa complexité et sa richesse) était traditionnellement réservé aux projets de haut niveau sur des plateformes puissantes, *mais ce n'est plus le cas*. Toutes les variétés de micro-contrôleurs tirent maintenant parti de la puissance du C++.

**Pourquoi ?** Il est donc désormais **indispensable** de maîtriser le **langage C++** *pour le développement de logiciel embarqué*. Notre **formation « Développement de Logiciel Embarqué en Langage C++ »** de **cinq jours** est conçue particulièrement pour ce contexte (Micro-contrôleurs, Processeurs embarqués et System On Chip).

**Pour qui ?** Ce cours est destiné aux Ingénieurs *Hardware*, Logiciels et *System-On-Chip*.

**Objectifs**
- Syntaxe et Sémantique du C++ (dont C++11) • Principes de la modélisation Objets
- Logiciel embarqué et programmation temps réel.
- Comment programmer et mettre au point sur un micro-contrôleur en C++
- Utiliser les outils de debug et de développement temps réel.
- Accéder aux périphériques, écrire des routines d'Interruption en C++.
- Introduction aux OS temps réel
- Les bonnes pratiques en C++ embarqué.

**Pédagogie** Ce cours s'inscrit dans le cadre d'une action d'acquisition ou de perfectionnement des connaissances. Les **Exercices Pratiques**, qui occupent environ **50 % du temps de la formation** sont des éléments clés du succès de ces formations, et sont essentiel pour l'apprentissage. Ils assurent également un **contrôle continu**.

**Instructeurs** Les **Instructeurs ALSE** sont aussi et surtout des **Experts** qui utilisent à journée entière les Langages et les Méthodes de Conception qu'ils enseignent, pour concevoir et vérifier des systèmes complexes et performants. Ils savent partager leur savoir-faire avec passion et sont particulièrement appréciés des participants.

**Pré-requis** La participation à cette formation demande une **connaissance préalable des concepts de programmation.** Une bonne connaissance du **langage C** est **vraiment souhaitable** sans être absolument indispensable (la bonne maîtrise d'un *autre* langage est acceptable). Mais ce cours n'est pas approprié pour les personnes n'ayant pas déjà une certaine expérience en programmation.

**Durée** **Cinq (5) jours**, soit **35 heures effectives** de cours ;
Typiquement 9h30 → 18H, le premier jour et 9H → 17h30 les jours suivants.

**Lieu** Dans les locaux d'ALSE, **Paris XIII$^{ème}$**

**Matériel** Pour les cours publics, le matériel (station de travail et plateforme embarquée) est fourni pour la durée du training.

NB : à partir de quatre ou cinq stagiaires, une **formation sur site** est envisageable.

ALSE est un Organisme de Formation Professionnelle Continue déclaré auprès de la DIRECCTE depuis 1996 sous le numéro 26.21.01281.21 et enregistrée dans le Datadock. Cette formation peut donc être prise en charge dans ce cadre.

# Programme de la Formation
# Développement de Logiciel Embarqué en Langage C++

## Différences entre C et C++ (Jour 1)

### Requirements for C++ in an Embedded System

- Embedded system characteristics • Language choice
- Memory mapped peripherals • Volatile variables
- Compilation • System boot-up
- Best Practices

### Summary of C

- A refresher on basic C syntax
- Functions • Control flow
- Fundamental types • Literals • Derived types
- Operators • Standard libraries

### From C to C++ and C++11

- The features added to C by C++ and the ANSI C-1999 standard
- inline • const
- Enhanced enumerations
- constant expressions • auto
- Overview of I/O streams
- Function prototypes • Pass-by-reference • Default arguments
- Function and operator overloading
- String class

### Linkage and Storage

- Learn some tricky features often overlooked in C, but necessary for C++
- Scope • Linkage • Linking C and C++
- Namespaces
- Static, automatic and dynamic storage • new and delete
- Placement new
- Arrays and Pointers • Vectors

### C++ Development Environments for Embedded Systems

- Compiler Optimization • Object files
- Linkers and linker files
- Loader • Makefiles • Integrated development environments
- Debuggers • In-circuit emulation • Debug with Simulator/Emulator

## Introduction à la programmation Objet (Jour 2)

### Classes and Objects

- Introduction to modeling and abstraction
- Information hiding • Abstract data types
- Classes and objects • Public and private class members
- Member functions • Scope resolution • *this* pointer

.../...

### Constructors

- How to ensure that objects are properly initialized, and how to tidy up afterwards
- Constructors • Destructors • Delegating constructors
- Copy constructors • Pointers and objects
- Move semantics

### Members and Friends

- More features of C++ classes • Friends
- Operator overloading • Overloading assignment
- Move assignment
- Memory fragmentation • Working with memory pools
- Static members • Constant objects and members

## Subtilités de l'OOP en C++ (Jour 3)

### Object-Oriented Modelling and the UML

- Learn the principles of object-oriented design • Class relationships
- The Unified Modeling Language • Class and object diagrams
- Association • Composition • Dependency
- Implementing class relationships in C++
- Initialization of class members • Singleton class
- Design Patterns

### Inheritance

- Derived classes • Inheritance
- Protected members
- Casting pointers
- Order of initialization

### Virtual Functions

- Inheriting common behavior • Overriding methods
- Virtual functions • Polymorphism • Late binding
- Virtual destructors • Vtable • Cost of virtual functions
- Abstract base classes and pure virtual functions
- Interface classes

## Utiliser les librairies standard et créer ses *class templates* (Jour 4)

### Further C++ Features

- User-defined conversions • Explicit functions • Defaulted and deleted behavior
- Run-Time Type Identification • Type casts • Nested classes
- Multiple Inheritance • Pointer-to-member • Function wrapper

### Templates

- Function templates • Class templates • Template arguments • Template specialization
- Dependent name and type lookup
- Avoiding code bloat

### Standard Libraries

- Summary of the standard C and C++ libraries
- Container classes • Container adapters
- Creating and accessing containers • Initializer lists • Custom allocators
- std::array • Other C++11 Enhancements
- Iterators • range-for loop

.../...

## Comprendre la Safety et la problématique du multi-tâche (Jour 5)

### Defensive Programming

- Coding standards • Code analysis
- Compile-time assertions • Run-time errors
- Throwing and catching exceptions • Handlers • Standard exception classes
- Preventing memory leakage • unique_ptr
- Exception specification • Exceptions in Embedded System

### Principles of Real-time Operating Systems

- Concurrency • Tasks and task switching
- FreeRTOS • Creating and running tasks • Synchronisation
- Process scheduling and pre-emption • Priority inversion
- Mutex, Semaphore and Queue • Use of semaphores with interrupts
- RTOS services

### Standard Library Algorithms

- Predicates • Function Objects
- Bind • Lambda Functions • Filling a container
- Non-modifying operations • transform
- Searching • Sorting
- Summary of the standard algorithms

### C++ State Machines (Optional Topic)

- State machine representations
- Single class state machine
- State Design Pattern
- Boost mpl example

### Strings and Streams (Optional Topic)

- Another look at string class
- I/O stream hierarchy • Formatted and unformatted streams
- I/O of user-defined types
- Manipulators • File Streams
- Buffering • Stringstreams

-=oOo=-