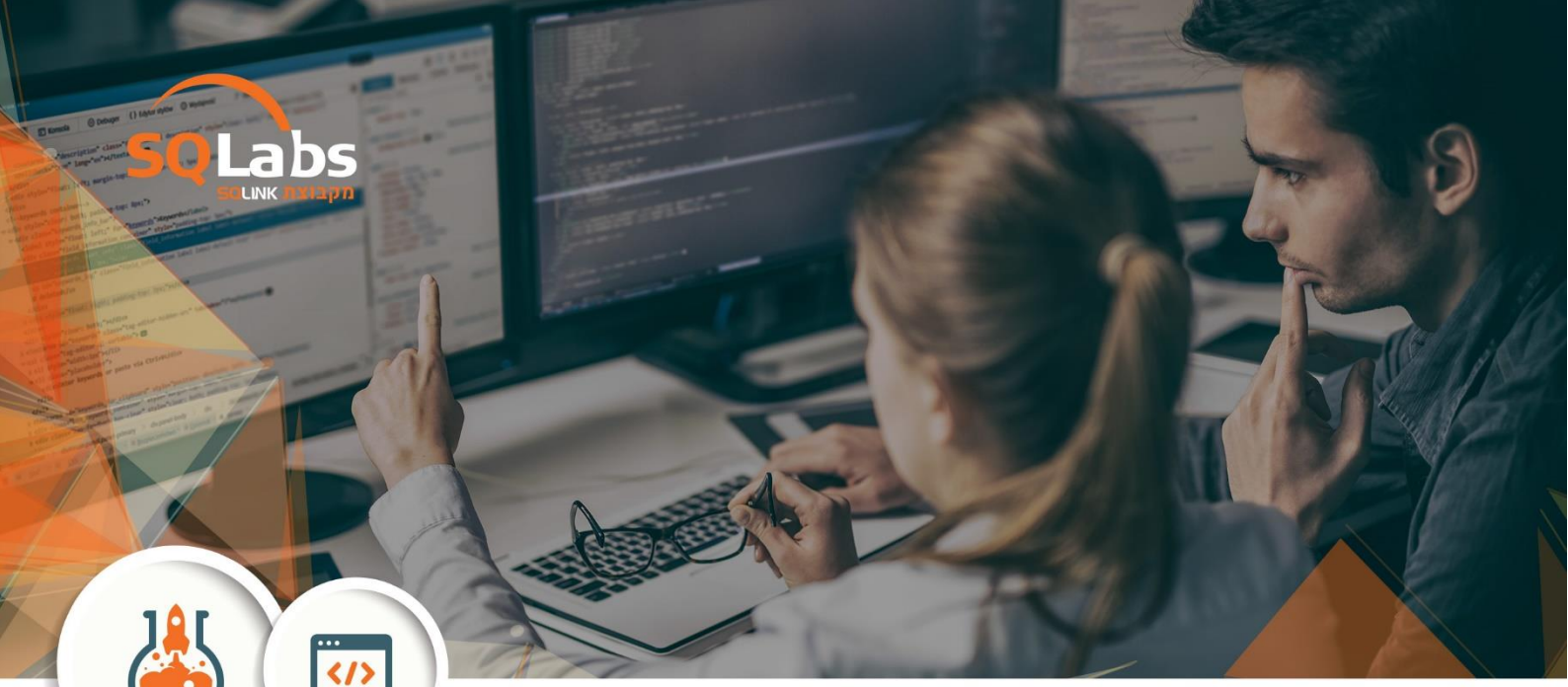

Advanced C++ and Modern C++

Duration : 40 hours

Overview

C++ is still one of the most widely used and effective object-oriented designs. While it is very powerful, C++ is not a simple language to master. To use the language correctly, developers must be wary of memory management and addressing pitfalls, value vs. pointer performance issues and be aware of all the tools an extensive standard library has to offer.

Furthermore, the language never rests. There were major changes to the C++ language in 2011, and more advancements in 2014 and 2017. These new features support modern programming paradigms and keep C++ fresh, and competitive with newer programming languages.

While C++ code written 20 years ago still compiles and runs, modern C++ programs can look completely different. Code written using the latest tools the language has to offer can achieve the same functionality in ½ the size, and still be safer and easier to maintain.

For developers whose C++ experience goes back further, many of the changes as a result of the latest standards make C++ a very different programming environment. This seminar will keep the audience abreast of these changes in a hands-on, workshop type environment with practical exercises.

On Completion, Delegates will be able to

- Know the hidden cost of copy constructors and conversion operators
- Implement a few design patterns in C++
- Write and use smart references and smart pointers
- Recognize the available tools and pitfalls in the Standard Template Llibrary
- Learn about the major enhancements in the last few versions of C++
- Bring out the power of modern C++ using lambda expressions
- Take advantage of the new features in the standard C++ library
- Understand and employ auto variables, Rvalue references, and move semantics.

Who Should Attend

Experienced C++ programmers wishing to enhance their development skills with the latest C++ capabilities

Course Contents

Part 1 – Advanced C++

C++ and OO Refresher:

Abstraction and encapsulation; Composition and association; Inheritance and polymorphism; Patterns and idioms.

Copying and Conversions:

The static cast, dynamic_cast, const_cast and reinterpret_cast keyword casts; Logical vs physical const-ness and the mutable keyword; Converting constructors and the explicit keyword; User defined conversion operators; Copy construction and assignment.

Scope:

Static class members; The Singleton pattern; Nested classes; Nested class forward declarations; The Cheshire Cat idiom; Namespaces.

Delegation Techniques:

The Object Adapter pattern; The Null Object Pattern; The Proxy pattern; Overloading the member access operator; Smart pointers; The Template Method pattern; Factory objects.

Subscripting Techniques:

Overloading the subscript operator; Overloading with respect to const-ness; Smart references; Multi-dimensional subscripting; Associative containers

Template Functions:

Using and implementing generic algorithms with template functions; Overloading and specializing template functions; Template instantiation and linkage.

Template Classes:

Using and implementing generic types with templates classes; multiple template parameters; the standard vector, list, pair, and map template classes.

Iterators and Algorithms:

The need for Iterators; The standard library (STL) iterator model; Generic algorithms using iterators; STL algorithm pitfalls; Introduction to function objects.

Exception Handling:

Classifying and handling exceptions; Catching and throwing exceptions; the standard exception class hierarchy; uncaught exceptions; Strategies for handling exceptions.

Exception Safety:

Resource acquisition idioms for exception safety; Exceptions in constructors; Exceptions in destructors; Exception safe classes; STL exception safety guarantees.

Memory Management: (If time remains)

Object life cycle; Allocation failure; Customizing memory allocation; Optimizing allocation for a class through caching; Derivation safe allocation; Controlling timing of construction and destruction.

Reference Counting: (If time remains)

Reference counting shared representation; Reference counted strings for copy optimization; Subscripting issues; Smart pointers for simple, automatic garbage collection.

Inheritance Techniques: (If time remains)

Subtyping vs subclassing; Abstract and concrete base classes; Inheritance scoping issues; Multiple inheritance; Virtual base classes; Interface classes; Mixin classes; Runtime type information (RTTI); Private and protected inheritance; The Class Adapter pattern.

Template Techniques: (If time remains)

Templating on precision; Template adapters; Default template parameters; Template specialization; Trait classes; Member templates; Non-type template parameters; Optimizing template code space.

Functional Abstraction: (If time remains)

Traditional callbacks using function pointers; The Command pattern; more on function objects; Member function pointers.

Part 2 – Modern C++

What's New In C++:

- Additions and changes to C++ in 2011,2014,2017, 2020
- Lambda expressions, RValue references, Move semantics, Standard Library Threads
- Safer, cleaner code with Aggregate Initialization, Range based loops, auto, DeclType, Static Assert, and Smart Pointers
- Compile time programming with Variadic Templates, Fold Expressions, and ConstExpr
- Performance Issues with Copy Ellision and Move Constructors
- 2020 features: requirements and concepts, abbreviated template functions, spaceship operator, modules