Programación C++ avanzada

Curso práctico de 4 días - 28h
Ref.: POP - Precio 2024: 1 670€ sin IVA

En constante evolución, desde C++98 hasta C++20, el lenguaje C++ ofrece mecanismos para un diseño robusto y completo. Los últimos estándares de C++ han mejorado notablemente la biblioteca de plantillas estándar (STL). Este curso le permitirá aprender más sobre el diseño en C++ cubriendo los últimos avances del lenguaje y haciendo un uso eficaz de la STL.

OBJETIVOS PEDAGÓGICOS

Al término de la formación, el alumno podrá:

Descubra las novedades de las nuevas versiones

Gestión de memoria, punteros y referencias

Genericidad en C++

Descubra la biblioteca STL estándar

Utilizar las aportaciones del estándar C++11

El curso se impartirá en estaciones de trabajo Windows/Visual C++. Se utilizarán numerosos ejercicios para aplicar los temas tratados más específicamente desde el punto de vista del diseño.

PROGRAMA

última actualización: 08/2024

1) Recordatorios

- Clases de asignación de memoria.
- Construcción, inicialización y carga de objetos.
- Fugas de memoria.
- Constance, la palabra clave mutable, Lazy Computation.
- Amistad C++ y control de acceso.
- Destrucción virtual.
- Estrategia de gestión de excepciones.
- Espacios de nombres.

2) Nuevas funciones del lenguaje en C++11

- nullptr y otros literales.
- Las directivas =delete, =default.
- Delegación del fabricante.
- Enumeraciones seguras.
- La palabra clave auto y el bucle sobre un intervalo.
- referencia rvalue e impacto en la forma normal de las clases C++.
- Expresiones lambda.

Trabajo práctico : Reescritura de código C++ existente en C++11, comparando las dos implementaciones.

3) Gestión de operadores

- Operadores binarios y unarios.
- El operador de indirección, caso de uso.
- El operador de referenciación.
- Operadores de incremento/decremento prefijados y postfijados.
- Otros operadores: comparación, asignación, etc.

PARTICIPANTES

Diseñadores y desarrolladores de aplicaciones C++, gestores de proyectos, arquitectos de software.

REQUISITOS PREVIOS

Buenos conocimientos de desarrollo en C++, o conocimientos equivalentes a los adquiridos en el curso "Programación de objetos en C++" (ref. C++). Experiencia requerida.

COMPETENCIAS DEL FORMADOR

Los expertos que imparten la formación son especialistas en las materias tratadas. Han sido validados por nuestros equipos pedagógicos, tanto en el plano de los conocimientos profesionales como en el de la pedagogía, para cada curso que imparten. Cuentan al menos con entre cinco y diez años de experiencia en su área y ocupan o han ocupado puestos de responsabilidad en empresas.

MODALIDADES DE EVALUACIÓN

El formador evalúa los progresos pedagógicos del participante a lo largo de toda la formación mediante preguntas de opción múltiple, escenificaciones de situaciones, trabajos prácticos, etc.
El participante también completará una prueba de posicionamiento previo y posterior para validar las competencias adquiridas.

MEDIOS PEDAGÓGICOS Y TÉCNICOS

• Los medios pedagógicos y los métodos de enseñanza utilizados son principalmente: ayudas audiovisuales, documentación y soporte de cursos, ejercicios prácticos de aplicación y ejercicios corregidos para los cursillos prácticos, estudios de casos o presentación de casos reales para los seminarios de formación.
• Al final de cada cursillo o seminario, ORSYS facilita a los participantes un cuestionario de evaluación del curso que analizarán luego nuestros equipos pedagógicos.
• Al final de la formación se entrega una hoja de presencia por cada media jornada de presencia, así como un certificado de fin de formación si el alumno ha asistido a la totalidad de la sesión.

MODALIDADES Y PLAZOS DE ACCESO

La inscripción debe estar finalizada 24 horas antes del inicio de la formación.

ACCESIBILIDAD DE LAS PERSONAS CON DISCAPACIDAD

¿Tiene alguna necesidad específica de accesibilidad? Póngase en contacto con la Sra. FOSSE, interlocutora sobre discapacidad, en la siguiente dirección psh-accueil@orsys.fr para estudiar de la mejor forma posible su solicitud y su viabilidad.

- Sobrecarga del operador [] y de los operadores de inserción (<<) y extracción (>>).
- Functores y sobrecarga de operadores (), una ventaja sobre las funciones.
Trabajo práctico : Creación de functores y proxies (liberación de memoria, recuento de referencias) con los operadores estudiados.

4) Conversión y RTTI
- Operadores de conversión. Construcciones implícitas, la palabra clave explícita.
- Operadores de reparto const_cast, static_cast, reinterpret_cast.
- Conversión dinámica e información de tipos en tiempo de ejecución.
- El operador typeid y las excepciones relacionadas.
- La clase type_info.
- Control del downcasting mediante el operador dynamic_cast.
Trabajo práctico : Implementación de los modismos "is-a" y "is-kind-of" con dynamic_cast.

5) Genericidad
- Introducción a los patrones de clase. Genericidad y preprocesador.
- Función genérica. Clase genérica. Composición genérica. Generalización genérica.
- Especialización parcial y total.
- Introducción a la metaprogramación.
- Genericidad, el principio unificador de las bibliotecas STL y Boost.
Trabajo práctico : Inicio del estudio de casos, que se completará con el STL. Implementación de la composición genérica y la generalización. Creación de plug-ins genéricos.

6) La STL (Biblioteca de plantillas estándar)
- Componentes STL: tipos complementarios, contenedores, algoritmos, iteradores, objetos función, adaptadores.
- Cadenas de caracteres STL, la clase de plantilla basic_string y sus especializaciones.
- Contenedores secuenciales y asociativos: definición, función y criterios de selección.
- Asignadores y gestión de la memoria de los contenedores.
- Métodos para insertar, eliminar, iterar y acceder a los principales contenedores: Vector, Lista, Conjunto, Pila, etc.
- El concepto de iterador. Navegación por un contenedor.
- Los distintos grupos de algoritmos STL: no mutantes, mutantes, de ordenación y fusión, numéricos.
- Manejo de contenedores (manipulación, búsqueda de valores, etc.).
- Parametrizar algoritmos genéricos mediante objetos "función".
- Adaptadores" y modificar el comportamiento de un componente.
- STL y procesamiento de flujos (archivos, memoria, etc.).
- El principio RAII: punteros automáticos y la clase auto_ptr.
- Excepciones STL estándar.
Trabajo práctico : Implementación de relaciones con colecciones STL. Uso de cualquier algoritmo estándar.

7) Novedades de la biblioteca estándar C++11
- Evolución histórica: Boost --> TR1 --> C++11.
- Nuevos contenedores: array, forward_list, unordered_set, unordered_map.
- La clase tupla.
- Punteros inteligentes: shared_ptr, weak_ptr, unique_ptr.
- Nuevos functores y aglutinantes.
- Introducción a la gestión de hilos.
- Expresiones regulares.
Trabajo práctico : Implementación de la robustez con punteros inteligentes. Uso de expresiones regulares.

8) Boost y sus principios
- Biblioteca de contenedores de punteros (destrucción de datos de punteros de contenedores).

- Las estructuras de datos boost::any y boost::variant.
- Programación basada en eventos (conexiones y señales).
- Gestión de procesos, mecanismos de comunicación entre procesos y memoria compartida.

Trabajo práctico : Implementación mejorada del caso práctico utilizando la biblioteca de contenedores de punteros.

9) Uso avanzado de la herencia

- Herencia frente a internado. Patrimonio privado. Patrimonio protegido.
- Exportación de miembros ocultos con la cláusula Using.
- Gestión de herencia múltiple y colisión de miembros.
- Herencia diamante. Herencia virtual y dynamic_cast.
- Principios de diseño: sustitución de Liskov, principio de apertura/cierre, inversión de la dependencia.
- Reglas de implementación de interfaces en C++.

Trabajo práctico : Combine la herencia múltiple, la herencia privada y la exportación para crear clases robustas y altamente escalables.

FECHAS

Contacto