

C++: produktiva lösningar och designmönster - 4 dagar

kurser 397

- Du får lära dig att**
- Utforma och implementera effektiva objektorienterade lösningar med C++
 - Öka kvaliteten och återanvändbarheten för C++-kod med designmönster och beprövade idiom
 - Bygga robusta, effektiva bibliotek med "namespaces", "templates" och "exceptions"
 - Använda C++ standardbibliotek, inklusive Standard Template Library (STL)
 - Utnyttja tredjepartsverktyg, klassbibliotek och applikationsmallar
 - Undvika C++-programmeringens fällor och fallgropar
- Sammanfattning** Förmågan att utnyttja andras arbete, undvika fallgropar och att använda beprövade idiom och mönster kan effektivt höja effektiviteten på programmeringen. Under denna kurs får du lära dig att öka produktiviteten genom att kombinera verktyg, idiom, syntax och bibliotek för att producera bra C++-kod. Flera praktiska övningar ger erfarenhet av att skriva C++-kod av hög kvalitet.
- Vem bör delta** Denna kurs är värdefull för programmerare, programvaruingenjörer, analytiker och utvecklare som vill lära sig avancerad C++. Tidigare erfarenhet av C++-programmering förutsätts.
- Praktiska övningar** Genom instruktörsledda praktiska övningar förstärks dina kunskaper i avancerad C++-programmering. Du får lära dig att utveckla kompletta program, från arkitekturutformning till förfinad implementering. Övningarna inkluderar:
- Forward- och reverse-engineering av C++ och UML
 - Öka kod-kvaliteten med designmönster
 - Modifiera en fungerande, men dåligt strukturerad tillämpning för att förbättra flexibilitet, robusthet och effektivitet
 - Tillämpa alla viktiga delar av STL
 - Använda "namespaces", "exceptions" och "templates" för att bygga bibliotek som kan återanvändas
 - Felsöka och rätta till fel

C++: produktiva lösningar och designmönster - 4 dagar

kurser 397

Introduktion till objektorienterad utveckling

Grunder för objektorientering

- Arv, inkapsling och polymorfism
- Klasser, objekt och attribut
- Associationer, meddelanden och metoder
- Gränssnitt och abstrakta klasser

Använda unified modeling Language

- Egenskaper hos UML Avbilda
- UML i C++

Använda utvecklingsverktyg

- Automatisera livscykeln med CASE-verktyg
- Kod-generering och "reverse engineering"
- Verktyg för felsökning och bläddring

Idiom och designmönster

C++-idiom

- Handle/body- och relaterade idiom
- Functors: funktioner kodade som objekt

Introduktion av designmönster

- Anledningen till att ha mönster
- Mönsterkategorier: skapande, strukturella och beteendemönster
- Beskriva designmönster

Använda designmönster

- Synkronisera flera vyer med "Observer"-mönstret
- Hantera rekursiva datastrukturer med mönstret "Composite"
- Minimera kod-duplicering med mönstret "Template Method"
- Hantera objektskapande med mönstret "Singleton"

ISO standardbibliotek för C++

Standard Template Library (STL)

- STL:s struktur
- Deklarera och fylla sekvens- och associationsbehållare
- Åtkomst till behållare med iteratorer
- Tillämpa standard- och användardefinierade algoritmer
- Anpassa beteendet med funktionsobjekt och "adapters"
- Utöka STL

Det nya iostream-biblioteket

- Grundläggande input/output
- Formatera text-output
- Hantera fel i input-data
- Breda teckentyper och internationalisering

Lagringshantering

Minneshantering

- Upptäcka och minska onödig minnesanvändning
- Förebygga minnesluckor med mallen **auto ptr**
- Överlagra **operator new** och **operator delete**
- Skriva och använda smarta pekare

Lagring av filer

- Förbereda klasser för att enkelt lagra och hämta filer
- Lagra och hämta objekt med Boost serialiseringsbibliotek

Skriva bättre C++

Öka kodens återanvändbarhet

- Undvika namnkollisioner med "namespaces"
- Använda mallar för typsäker återanvändbarhet

Öka stabiliteten

- Förstärka inkapsling genom konsekvent och korrekt användning av **const**
- Implementera en sammanhängande strategi för "exceptions"
- Frikoppla algoritmer från datastrukturer med mönstret "Visitor"

Öka effektiviteten

- Spara in på bearbetning och minne med referensräkning
- Dela tillstånd mellan lättviktsobjekt

Undvika fällor och fallgropar i C++

Saker som måste göras — och varför

- Virtuella destruktorer
- Tilldelningsoperatorer och kopieringskonstruktorer

Saker man skall vara försiktig med

- Friends kontra public-medlemmar
- Run-time typinformation vid körning jämfört med virtuella medlemsfunktioner
- Multipelt och virtuellt arv