

## C++ Programming: A Comprehensive Hands-On Introduction - 4 dagar

*kurser 327*

- You Will Learn How To**
- Create, compile and run C++ programs
  - Write functions, decisions, loops and exceptions
  - Declare, use and distinguish variables, constants, arrays, pointers and references
  - Define and implement classes to produce reliable, reusable code
  - Use STL classes and instantiate templates
  - Implement object-oriented designs using encapsulation, inheritance and polymorphism

**Course Benefits** C++ is a long-established, mainstream language used across a broad range of applications. This course provides a solid foundation in C++ for programmers without assuming experience with the C language. You acquire knowledge of key object-oriented programming concepts and gain valuable hands-on experience developing C++ programs.

**Who Should Attend** This course is for those interested in programming with C++, including application and systems programmers, software engineers and their managers. Professional programming experience is assumed. C programming experience is not required.

**Hands-On Training** Exercises provide you with extensive C++ programming experience and include:

- Writing, compiling and executing C++ programs
- Performing arithmetic computations and string operations
- Defining and calling top-level and class member functions
- Reading and writing formatted I/O
- Using **for** loops and **if/else** decisions
- Defining and using new classes
- Managing dynamic data
- Extending a class with inheritance
- Overloading functions and operators
- Instantiating templates

## C++ Programming: A Comprehensive Hands-On Introduction - 4 dagar

*kurser 327*

### Introduction and Overview

- Relating C, C++, Java and C#
- The in-class development environment
- Other development environments

### C++ Programming Building Blocks

#### The main function and standard I/O

- **main**'s specification and body
- Displaying values and strings to **cout**
- Reading values from **cin**
- Formatting with stream manipulators

#### Objects, constants and references

- Declaring and initialising variables
- Integer and floating point data types
- Performing arithmetic calculations and displaying results
- Passing messages to objects
- Using references for efficiency and constants for safety

#### Defining and calling functions

- Passing arguments to functions and returning values from functions
- Call-by-value vs. call-by-reference vs. call-by-address
- Scope, duration and initial values of local temporary and parameter variables

#### Decisions, loops and logic

- Making decisions with **if/else**
- **bool** vs. **int** logical values
- **if/else** statement "chains"
- Performing loops with **while** and **for**
- Equality, relational and logical operators
- Increment and decrement operators

#### Arrays, pointers and strings

- Declaring and using arrays and pointers
- Storing strings in character arrays
- Accessing array elements via pointers
- Pointers vs. references
- Standard string class and functions

#### Defining C++ Classes and Objects

##### Encapsulating higher-level data types

- Public member functions and private data members
- Protected class members
- Constructors and destructors
- Member initialisation syntax
- Self-reference: the **this** pointer
- The class member operator (**::**)

##### Declaring, accessing and modifying objects

- Manipulating arrays of objects, pointers to objects and references to objects
- Invoking member functions

- **const** member functions
- Passing objects by value and by reference

#### Overloading and templates

- Simplifying class interfaces
- Function signatures
- Overloading assignment (=) and insertion (<<)
- **friend** functions and classes
- Explicit copy construction
- Avoiding default assignment and default copy construction
- Using STL templates to define families of related classes

#### Separating interfaces and implementations

- How separation supports code reuse
- Building header files and code files

#### Extending Classes via Inheritance

##### Deriving new classes from existing classes

- Construction and destruction of derived objects
- Is-a-kind-of relationships
- Reusability via incremental extensions
- Base classes and derived classes

##### Utilising polymorphic functions

- Overriding virtual base class member functions in derived classes
- Runtime lookup of functions through base class pointers and references

##### Managing dynamic data

- Allocating and deallocating memory with **new** and **delete**
- Handling errors with **try** and **catch**
- Avoiding memory leaks

#### Standards and Extensions

- Standard vs. platform-specific implementations
- Applicability to Windows and UNIX/Linux
- ANSI/ISO 98 and evolving "0X" standards
- ECMA C++/CLI and other extensions