Professional C++ Programming with Qt5, STL and Boost

Part 1: C++ - 98

Object-Based Programming

- Towards class: Defining type in C Programming language – Client-server separation – Modular programming – Inability of server-side to hide data layout from a client in C – Why is hiding data is necessary? – Extendibility
- 'The hello world' program in C++ - Build process of a single file C++ program – Build process of a multi-file C++ program.
- First version of C++ = C With classes – Principle of data abstraction – Principle of Information hiding.
- The class statement:
    - The syntax: private, public, protected keywords.
    - Defining data and member functions inside a class.
    - Creating an object of a class using a data definition statement.
    - Creating an object of a class dynamically using 'new' and de-allocating an object using 'delete'.
    - Constructors:
        - Constructor as a call-back function.
        - Types of the constructor: Synthesized constructor – Default constructor – Parameterized constructor.
        - Ways of writing the constructor: Constructor initialiser list – Explicit assignment – Hybrid.
        - Function overloading – One definition rule (ODR) – Name mangling is done by C++ compiler – Ability to write multiple functions with the same name and distinct parameter list – Resolution of a call to overloaded function – Viable functions – candidate functions
        - Overloaded constructors.
    - Adding member function to a class – 'this' pointer – Invoking member function of a class using instance variable – Invoking member function of a class using pointer variable.
    - L-value reference variable – A reference variable to create an alias name for a given object – A reference variable to name an anonymous object of the free store.
    - The static keyword:
        - Use-case 1: Restricting visibility of global data to a source file.
        - Use-case 2: Restricting visibility of global function to a source file.
        - Use-case 3: Making local variable static to change its lifetime and maintain its state across successive function calls.
        - Use-case 4: Making data member of class static.
        - Use-case 5: Making member function static.
    - The const keyword:
        - Making global and local data const.

        - Difference between the semantics of const keyword between C and C++.
        - Const keyword and pointers: Pointer to const – const pointer – const pointer to pointer to const – Use each case.
        - The mutable keyword.
        - The inline keyword.
        - The extern keywords.
- The Memory Management:
    - Bare-metal environment – O.S. environment – Time-sharing, pre-emptive multi-tasking O.S. with virtual memory implementation.
    - Sections of virtual address space of a process: The text section – the read-only data section – the data section – the BSS – the stack section – the concept of memory mapping – mapped and unmapped virtual address space - the heap section as unmapped virtual address space – what is free store referred in C++ reference books?
    - Mapping C program to sections in virtual address space.
    - Mapping C++ class and its object to sections in virtual address space.
    - A deeper explanation of the static and the const keyword based on the knowledge of a build process and memory management details.
- Operator overloading:
    - Concept of operator-loading – Operator overloading as a call-back mechanism - Operator keyword – Unary, binary operators – Operators that can be overloaded.
    - Overloading arithmetic and logic operators.
    - Overloading the pre and post-increment and decrement operators.
    - Overloading the cin and the cout operators.
    - Overloading the subscript operator.
    - Overloading the call operator.
    - Overloading the new operator, locally and globally.
    - Operator overloading and the pitfalls.
    - The semantics of unary operator overloading.
    - The Semantics of binary operator overloading.
- Copy Control:
    - Initialising a new object of a class with its existing object.
    - Assigning one object of a class with another.
    - Synthesised copy constructor – Synthesized overloaded assignment operator – The destructor.
    - The consequence of having indirect (pointer/reference), non-static variable in a class.
    - Pitfall one: Memory exception violation (segmentation fault)
    - Pitfall two: Memory leak.
    - Writing customised version of the copy constructor and the assignment operator overload and the destructor method to avoid pitfalls of the default version.
    - Implementing ‘deep – copying’ approach.
    - Implementing ‘shallow – copying’ approach with reference counting.
- Principles of class design: Problem space – implementation space – identifying entities in problem space – Principle of cohesion and coupling.

Object-Oriented Programming:

- Principle of reusability.
- Relationship between the classes – 'depends-on' relationship – 'has-a relationship.'
- Scenario-based case studies where multiple classes are designed having the above relationships.
- Hierarchical nature of the complex system – Case study-based explanation of general – the special relationship between the classes – Cast-study based description of building complex hierarchical system using general – special classes.
- Exploring the possibility of re-use while implementing a special version of a general class – 'is-a' relationship between classes.
- Implementation inheritance to achieve re-usability of implementation of an existing class.
- Inheriting one class from the other – the base class – the derived class – memory management of an object of the derived class – Writing a constructor of a derived class – Calling a constructor of the base class from the constructor of a derived class – Role of default constructor in a base class.
- The protected access specifier.
- Inheriting member functions of a base class – overriding member functions of base class – the difference between overriding member function base class for replacement and an extension.
- Storing the address of an object of the derived class is a pointer to the base class – Why is it allowed?
- Virtual function – virtual function and overriding member function – Principle of polymorphism – Defining polymorphic class.
- Studying the following combinations: Base class pointer & base class object, derived class pointer & derived class object, base class pointer and derived class object – Resolution of a member function in each combination taking cognisance of a virtual member function.
- Object slicing.
- Run-time type identification: Fetching the names of data types of variable names, objects and class names using the typeid operator – typeid operator and polymorphic classes.
- Interface inheritance – a better way to hide the implementation
- How valuable information can be leaked through the header file?
- Pure function – Pure and virtual function – Pure abstract base class – Creating an interface using pure abstract base class – Inability to instantiate a pure abstract base class.
- How to separate the interface and implementation using 'interface inheritance.'
- The principle of polymorphism and pure abstract base class.
- The principle of late binding.
- Resolution of calls to virtual and pure-virtual member functions from a base-class pointer.
- The vtable and adjustor thunk – Hidden members of object – 'this' pointer – the vtable – Runtime type identification information.

Generic Programming with templates:

- Generic algorithms: Why generic algorithms? – Type safety vs genericity – Macros and void* - C++ function templates
- Implementing stand-alone algorithms using function templates.
- Container types – class and container types – Implement generic data structures using class templates.
- Template argument deduction in function templates – Template parameters – Overloading function templates – Specialization of class templates – Partial specialisation – Default template arguments
- Non-type template parameters – non-type class template parameters – non-type function template parameters – Restrictions for non-type template parameters
- Keyword typename – Member templates – Template-template parameters – Zero initialisation – Using string literal as arguments for function template
- Template models: The inclusion model – Explicit instantiation – The separation model – Templates and inline – Precompiled headers – Debugging templates
- Instantiation and specialisation – Declaration vs definition – The One-definition rule – Template argument vs template parameter
- Template Instantiation – On-demand instantiation – Lazy instantiation – The C++ instantiation model – Explicit instantiation
- Template argument deduction – The deduction process – Deduced context – Special deduction situations – Implicit type conversions – Default call argument
- Template and polymorphism – Dynamic polymorphism – Static polymorphism
- Traits and Policy classes – Type functions – Policy traits
- Template and inheritance – Named template arguments – The empty base class optimisation – Parameterized virtuality
- Metaprograms – Enumerations vs static constants – Using induction variable – Recursive instantiation vs Recursive template argument
- Using expressions templates – Temporaries and split loops – Encoding expressions in template arguments
- Type classification – Smart pointers – Tuples – Function objects and call-backs

Run-time type identification and typecasting

- Implicit type conversions in C
- Implicit type conversions in C++
- C style type casting
- Static cast
- Dynamic cast
- Reinterpret cast
- Const cast
- Type casting and inheritance
- Combining type casting, run-time type identification, and inheritance

Large program organization mechanisms

- Exception handling motivation – Exception handling in C using setjmp() and longjmp() –
- Exception handling in C++ using try catch throw.

- Exception class in C++ - Exception class hierarchy in C++ - Multiple catch blocks – Stack unwinding and its consequences – deprecated autoptr – Builtin exceptions – User defined exceptions
- Namespace – One definition rule – global namespace pollution – Namespace as a solution to global namespace pollution problem – Nested namespaces – using directive.

C++ 11 Language Features:

- move semantics
- variadic templates
- rvalue references
- forwarding references
- initializer lists
- static assertions
- auto
- lambda expressions
- decltype
- type aliases
- nullptr
- strongly-typed enums
- attributes
- constexpr
- delegating constructors
- user-defined literals
- explicit virtual overrides
- final specifier
- default functions
- deleted functions
- range-based for loops
- special member functions for move semantics
- converting constructors
- explicit conversion functions
- inline-namespaces
- non-static data member initializers
- right angle brackets
- ref-qualified member functions
- trailing return types
- noexcept specifier

C++ 14 Language Features

- binary literals.
- generic **lambda expressions**.
- lambda capture initializers.
- return type deduction.

- decltype(**auto**)
- relaxing constraints on constexpr functions.
- variable templates.