COMP6771
Advanced C++ Programming

10.2 Conclusion

COMP6771 in 60 Minutes or Less
a.k.a.: "Revision"

```
~COMP6771() {
```

Week 01: C -> C++

- C++ is a general-purpose programming language.
- CPU-native types: `int`, `double`, `void*`, etc.
- Class-like types: `struct`, `class`, `union`
- Functions: `void foo(int, double*)`
- Opt-in immutability: `const int i = 5;`
- auto: `auto it = std::vector<int>{}.begin();`
- Value-semantics and reference semantics: `T/T&/T*`
- A rich standard library: `std::vector`, `std::tuple`, etc.
- Modular code-sharing: `#include`
- Separate compilation and linking.

Week 02: STL

- Standard Template Library (STL)
- Containers, e.g.:
  - `std::vector`.
  - `std::list`.
- Algorithms, e.g.:
  - `std::copy`
  - `std::transform`
- Iterators
  - Input, Output, Forward, Bidirectional, RandomAccess, Contiguous
  - Glue between containers and algorithms

Week 03: Scope & Classes

- Scope.
  - Functions, for, if, while, {}, namespace introduce scopes.
  - Variables are accessible according to their scope.
- Object Lifetime.
  - Lifetime starts when brought into scope.
  - Lifetime ends when the scope ends.
- Classes are user-defined types that mirror primitives like `int`.
  - Initialisation customisable through constructors.
  - Clean-up customisable through destructor.
- Internal entities of a class are members.
  - Member functions.
  - Data Members.
  - Static member functions and static data members.
  - API extension through the power of *friendship*.

Week 04: Advanced Classes

- Operator Overloading:
  - Provide user-defined meanings for operators in C++.
  - Chained-operations very easy to read.
  - Make classes "feel" like primitives.
  - e.g. `v1 + v2` is more natural than `add(v1, v2)`.
- Custom Iterators:
  - Use operator overloading to make a class-type look like a pointer.
  - Implement operations that align with the various iterator categories.
  - Transformed custom containers into iterable ranges.

Week 05: Exceptionally Resourceful

- Exceptions:
  - Classes that represent unexpected runtime errors.
  - Dedicated syntax: `throw`/`try`/`catch`.
  - Compiler-enforced stack unwinding.
  - Throw by value, catch by `const&`!!.
- C++ manages resources through RAII:
  - Acquire resources (memory, locks, etc.) in the constructor, release them through the destructor.
  - Prevents resource leaks (by exceptions, forgetfulness, etc.).
  - Able to prevent deep copies by deleting copy-constructor and copy-assign.
  - Efficient transfer of ownership through move semantics.
- RAII-conforming Smart Pointers ™ replace "owning" raw pointers:
  - `std::unique_ptr<T>`/`T*` for unique ownership/observeration.
  - `std::shared_ptr<T>`/`std::weak_ptr<T>` for shared ownership.

Week 07: Dynamic Polymorphism (Inheritance)

- Classic OOP through *Dynamic Polymorphism*.
  - Inheritance and derived classes.
  - `virtual` methods.
  - `override`, `final`, pure virtual (*abstract*) methods.
  - Static (at compile-time) binding vs. dynamic (at runtime) binding.
- Implemented through vtables:
  - Table of function pointers to virtual methods.
  - Compiler-generated.
- Can cast up and down type hierarchies with `dynamic_cast`.
- Important considerations:
  - Polymorphic classes **must** have virtual destructors!
  - Dynamic polymorphism only happens for `T*` and `T&`!
  - Copying/moving a derived object into a base object causes *object slicing*.

Week 08: Static Polymorphism (Templates)

- Generic Programming through compile-time type paramerisation.
- Function, Class, Alias, Variable, and Variadic templates.
- Compiler synthesises function/class/alias/variable definition from the template when required.
  - Can be forced by explicit instantiation.
- Primary template customisable through specialisation, either:
  - Fully (explicit specialisation); or
  - Partially (partial specialisation, not for function templates).
- Parameterisable by:
  - Types (e.g. `template <typename T>`).
  - Non-type template parameters (e.g. `template <int N>`).
  - Template template parameters (e.g. `template <template <typename> typename Container>`).

Week 09: Metaprogramming

- Templates are "accidentally" Turing-complete i.e. they can be used to calculate *anything*.
- Modern C++ TMP moving away from abusing templates:
  - `decltype`: get the declared type of a variable at compile-time.
  - `constexpr`-world: compile-time expressions e.g. `if constexpr` and compile-time functions.
- Type traits use templates to ask questions at compile-time:
  - Makes heavy use of `struct` templates and partial/explicit specialisation.
  - Excessive use causes *incredibly* long compile-times and/or code bloat.
- Forwarding references (`T&&`) introduced in C++11:
  - auto type deduction and rvalue references binds to anything.
  - Can be used to "forward" arguments from one function to another whilst preserving value category.
- Concepts:
  - Constrain template type parameters for overloading, better error messages, etc.

Week 10: Advanced C++

- Not assessable:
  - Modules.
  - Ranges.
  - Coroutines.
  - C++23. (`std::inout_ptr`, new containers, `operator[,]`, `if consteval`, etc.)
- Sample of topics not covered in this course:
  - Exception specifications (depreceated in C++11, removed in C++17)
  - Non-default `noexcept` (e.g., `noexcept(sizeof(int) != 4)`)
  - RTTI in great depth
  - Linking with other languages through `extern "C" {}`, etc.

~~Week 11:~~ Goodbye*

* not yet (go to next slide)

Boyz II Men - End Of The Road

257M views • 13 years ago

Boyz II Men

REMASTERED IN HD! Official Website: http://w

CC

Listen Here

Final Exam

- See the Week 10 Notice for in-depth information.
- Practical exam with two sections:
  - S1 – Multiple Choice questions covering theory from all topic areas.
  - S2 – Mini-assignment: classes, templates, metaprogramming.
- S1 targets:
  - Students aiming for a PS or a CR.
  - *Much* easier than Q2.
- S2 targets:
  - Students aiming for a D or HD.
  - Quite difficult but completable with everything taught in this course.
- Partial marks available for Q1 and Q2.
- Sample Exam (see Week 10 Notice)
  - No solutions will be released.
  - Can ask questions about it on the forum.

Goodbye 👋

さよなら, Au revoir, 再见, `std::cout << "Goodbye ;_;" << std::endl;`, (insert your favourite language here [not Java])

- Further awesome C++ resources
- Books:
  - The Design & Evolution of C++ by Bjarne Stroustrup (creator of C++!)
  - Anything by Herb Sutter (ISO Chair for C++)
- Videos:
  - Cppcon (free conference talks, held annually)
  - C++ Weekly with Jason Turner
- I Tried This ONE Trick to INCREASE Exam Time and My Life Changed FOREVER...

}

Feedback (stop recording)