COMP6771
Advanced C++ Programming

Week 10.2
Conclusion
(aka ~`COMP6771()`)

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

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: `vector`, `tuple`, etc.
- Modular code-sharing: `#include<>`
- Separate compilation and linking

Week 02: STL

- **S**tandard **T**emplate **L**ibrary (STL)
- Containers, e.g.
  - `std::vector`
  - `std::list`
- Algorithms, e.g.
  - `std::copy()`
  - `std::transform`
- Iterators
  - Input, Output, RandomAccess
  - Glue between containers and algorithms

Week 03: 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 extensions through *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 == vec2d{v1.x + v2.x, v1.y + v2.y}` is more natural than `add(v1, v2)`
  - Full list of overloadable operators

- Exceptions
  - Classes that represent unexpected runtime errors
  - Dedicated syntax: `throw/try/catch`
  - Compiler-enforced *stack-unwinding*
  - Throw by value, catch by `const&` !!

Week 05: Resource Management

- C++ manages resources through RAII:
  - Acquire resources (memory, locks, etc.) in the constructor
  - Release them through the destructor
  - Every resource owned by an RAII class
  - Prevents resource leaks (by exceptions, forgetfulness, etc.)
- Ownership enforced through copy-control:
  - 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" pointers:
  - `std::unique_ptr<T>/T*` for unique ownership/observeration
  - `std::shared_ptr<T>/std::weak_ptr<T>` for shared ownership
  - Automatically free dynamically-allocated objects

Week 07: Templates

- Generic Programming through compile-time type paramerisation
- Function, Class, Alias, Variable, and Variadic templates
- Compiler synthesises function/class/typedef/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, only for class 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 08: TMP

- Templates are "accidentally" Turing-complete i.e. they can be used to calculate *anything*
- Type traits use templates to ask questions at compile-time:
  - Is `T` a pointer type (e.g. `int*`)?
  - What does `T` look like with `const` removed? (e.g. `const int` -> `int`)
  - 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 rvalue-ness or lvalue-ness
- Modern C++ TMP moving away from abusing templates:
  - Constexpr-world: compile-time expressions e.g. `if-constexpr`
  - `decltype:` get the declared type of a variable at compile-time

Week 09: Dynamic Polymorphism

- Classic OOP through *Dynamic Polymorphism*

  - Inheritance and derived classes
  - `virtual` methods
  - `override, final`, pure-virtual (*abstract*) methods
  - Early (at compile-time) binding vs. late (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 class into a base class causes *object slicing*

Week 10: Advanced C++

(from guest lecture; not assessable)

- Concepts
  - aka avoiding ->
- Modules
- Ranges
- Coroutines

~~Week 11:~~ Goodbye*

https://www.youtube.com/watch?v=qROu_TyeolU&t=77s&ab_channel=BoyzIIMen-Topic

* Not yet (click right)

Final Exam

- See the Week 10 Notice for in-depth information
- Practical exam with two questions:

  - Q1 - STL, algorithms, dynamic polymorphism
  - Q2 - classes, templates, compile-time programming

- Q1 targets:

  - Students aiming for a PS or a CR
  - Easier than Q2

- Q2 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 released NOW!

  - No solutions will be released
  - Can ask questions about it on the forum

Goodbye 👋

- 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