Fun with templates and STL algorithms

Curiously Recurring Template Pattern (CRTP)

- Base class is a template class
  - Derived class inherits from Base<Derived>
- Allows definition of a template via Base without the need for virtual functions
  - No vTable = smaller objects, and theoretically faster code
  - On my system: VS2017 in x86 on a Intel Core i7 I could not see the difference
- Likely to be seen in libraries
- This would not be my first chosen tool

Specializing our templates

- STL type_traits library gives us more control over our templates
  - If we intend a function to work on a numeric type, we don't want someone creating one on some complex type
- Compile time type checking and enforcement
  - std::enable_if
  - std::is_Integral … …

STL Algorithms: unary, binary, ranges

- Unary: std:abs(), std::sin(), round(), ceil()
- Binary: std::min(), std::swap()
- Ranges: work with iterators
- Oldest way: ‘normal’ for loop
  - for(int i = 0; i < cont.size(); ++i) { cont[i]++;}
- Newer: Ranged For
  - for(auto& x : cont) { x++;}
- Via STL: for_each
  - std::for_each(cont.begin(), cont.end(), [&](auto& x){x++});

Ranged Algorithms

- Various Finds
  - Find_first_not_of, find_if, count_if, adjacent_find
- Modifying algorithms
  - Move, copy, replace, swap, transform, generate
- Sorts
- Partitions
  - return iterator separates a range of elements
- Min/Max, Clamp, Permutations

Operation Wrappers

- std::plus()
- std::div() returns quotient and remainder