Fresh Paint

How to write exactly the same programs in C++11

Who Am I?

- Alisdair Meredith
- ISO committee member since 2003
- Current Library Working Group Chair

What is this session?

- C++11 goal: "remove embarrassments"
- Syntax clean ups
- syntax extensions
- idiomatic libraries

Quick Quiz

- Can anyone spot the C++11 feature of the sample program?
- (switch to vim!)

Quiz Result

- C++11 <iostream> implicitly includes:
  - <ostream>
  - <istream>
  - <ios>
  - <streambuf>

Other header changes

- swap moves from <algorithm> to <utility>
- tr1 result_of moves from <functional> to <type_traits>
- <bitset> no longer includes <stdexcept>
- all headers include <initializer_list>

Quick Quiz 2: strings

- Can anyone spot undefined behavior in the sample program?
- (switch to vim!)

COW strings outlawed

- Various exemptions to enable Copy-on-write (ref counted) strings removed
  - part of the concurrency effort
- New weasel words to allow short-string optimization
  - ‘swap’ can invalidate string iterators
  - (turns out this was added for 2003)

Functions that act as const for data races

- begin
- end
- rbegin
- rend
- front
- back
- at
- data
- find
- lower_bound
- upper_bound
- equal_range

string conversions

- to_string
- stoi
- stol
- stoul
- stoll
- stoull
- stof
- stod
- stold
- also ‘w’ versions

Container cleanup

- cbegin/cend/crbegin/crend
- shrink_to_fit on vector/string/deque
- map::at for ‘const’ queries
- unordered_containers get operator==/!=
- list::size is constant time

Raw String Literals

- Raw string literals:
  - ignore all escape codes
  - embed newlines if spanning multiple lines
  - retrieve the original text for trigraphs
  - allow user-defined escape to close string
- ideal with the regex library!

C++11 Templates

- Did not get concepts
- Did not get function template specialization
- Lost ‘export’
- variadic templates are awesome, but a different session!

Angle Brackets

- C++03: ‘>>’ is always parsed as an operator
  - requires an extra space closing templates
- C++11: Two (or more) adjacent ‘>’ symbols will be parsed as closing template parameter list
- e.g., vector<pair<int,int>>
- NO FIX for <: diagraph,

Troublesome typename

- Switch to VIM for example

Troublesome typename

- Spurious ‘typename’ keywords now ignored
  - popular 03 compilers already did this
- Similar relaxation for .template / ::template
- NO relaxation where typename is required
- Did we notice the empty statement after main?

Local Classes

- C++03: Local classes have internal linkage
- C++11: Local classes have internal linkage
  - but templates can instantiate such types!
- (see example)
- un-named namespace gets internal linkage
- static functions undeprecated

Extended SFINAE

- Substitution Failure Is Not An Error
- Widely abused technique to control overload resolution in templates
- C++03: SFINAE is a bullet list of things to check
- C++11: SFINAE is any compile fail, including access control (public/private)

Default function-template parameters

- C++11 allows default template parameters on function templates
- C++03: Cannot control constructor overloading with SFINAE
- C++11: use (extended!) SFINAE with default template argument
- Note: even easier with 'enable_if'

Alias Templates

- (see quick code sample)
- Cleans up C++03 idioms like allocator<T>::rebind<U>
- Cannot specialize an alias
  - but will pick up specializations of aliased template
- Non-template form can replace ‘typedef’

Uniform Initialization

- Too many different initialization syntaxes in C++03
- No one syntax that can be used universally in the grammar
- Some things cannot be explicitly initialized, e.g., arrays as data members
- Solution: generalize aggregate initialization

Uniform Initialization

- { } brace initializers can be used everywhere
- allows initialization of arrays/aggregates in more contexts, notably constructors
- side-steps “most vexing parse”
- narrowing conversions are a compile-error

initializer_list

- A new type, understood by the compiler
- represents a sequence of constant values
- differs from arrays as length is not part of the type
- allows constructors to initialize from a list of value
- applied consistently across the library

for loops

- new for-loop syntax to iterate over ranges
  - native arrays
  - containers
  - any type that implements begin/end
  - initializer_lists
- Use a reference to update original data

Function Declarations

- Inspired by need to declare certain template functions (see example)
- General syntax, not template specific
- Bonus: simple to line up function names in a header, picking out overload sets
- Does not deduce return type like lambdas
  - But watch this space for C++17...

Alternate Syntax Summary

- using vs. typedef
- consistent brace initialization
- new function syntax

Features to simplify writing classes

- delegating constructors
- inheriting constructors
- member initializers
- deleted functions
- defaulted special functions
- explicit override keyword

Delegating Constructors

- C++03: Cannot share initialization lists between constructors
- C++11: delegate to another constructor
- What happens if constructor body throws?

Inheriting Constructors

- Re-implementing large constructor lists can be a burden
- e.g., idiom to derive from string, rather than use typedef, for compile-time type checks
- Warning: no known implementation at this time

Inheriting Constructor Issues

- How do we initialize new data members
  - Member initializers!
- How do we inherit from multiple bases?
- What about default/copy constructors?
- What about private/protected constructors?
- What if this class wants a constructor with an inherited signature?

Deleted Functions

- All classes in C++ have a copy constructor
  - whether you want one or not!
- 'embarrassing' idiom to declare copy constructor/assignment operator private
  - and never define them!
- deleted functions express intent clearly

Defaulted Special Functions

- Sometimes we want the built-in definition of the default or copy constructor
- Better to explicitly state this, than rely on reader understanding the omission
  - especially for default constructor, which is not present with any other constructor

Override Keyword

- Ask the compiler to check we actually override a virtual function!
  - catches mis-typed function names
  - catches bad argument lists
  - highlights if base class changes
- Does not catch accidental overrides, where a virtual function added to the base class

explicit conversion operators

- Function similarly to explicit constructors
- bool conversions the most interesting case
  - language *will* use for if/while/for tests
  - replaces ‘unspecified boolean type’ idiom
- Commonly used for smart pointers

Basic Vocabulary Types

- function
- unique_ptr
- shared_ptr

function

- std::function a natural replacement for function pointers
- more flexible
- supports functors with state
- efficient if holding only a function pointer

unique_ptr

- drop in replacement for auto_ptr
- requires explicit syntax to transfer ownership, with std::move
- Supports arrays
- Basic form identical size to a native pointer
- Customized deleters supported, at compile time

shared_ptr

- shared ownership of a pointer
- ideal for use in a container
- custom deleter does not affect type
- custom allocator does not affect type
- atomic reference count pays a small price for the flexibility

Missing Algorithms

- copy_if
- all_of
- any_of
- none_of
- is_sorted