C++0x & C1x: the Dawn of new Standards

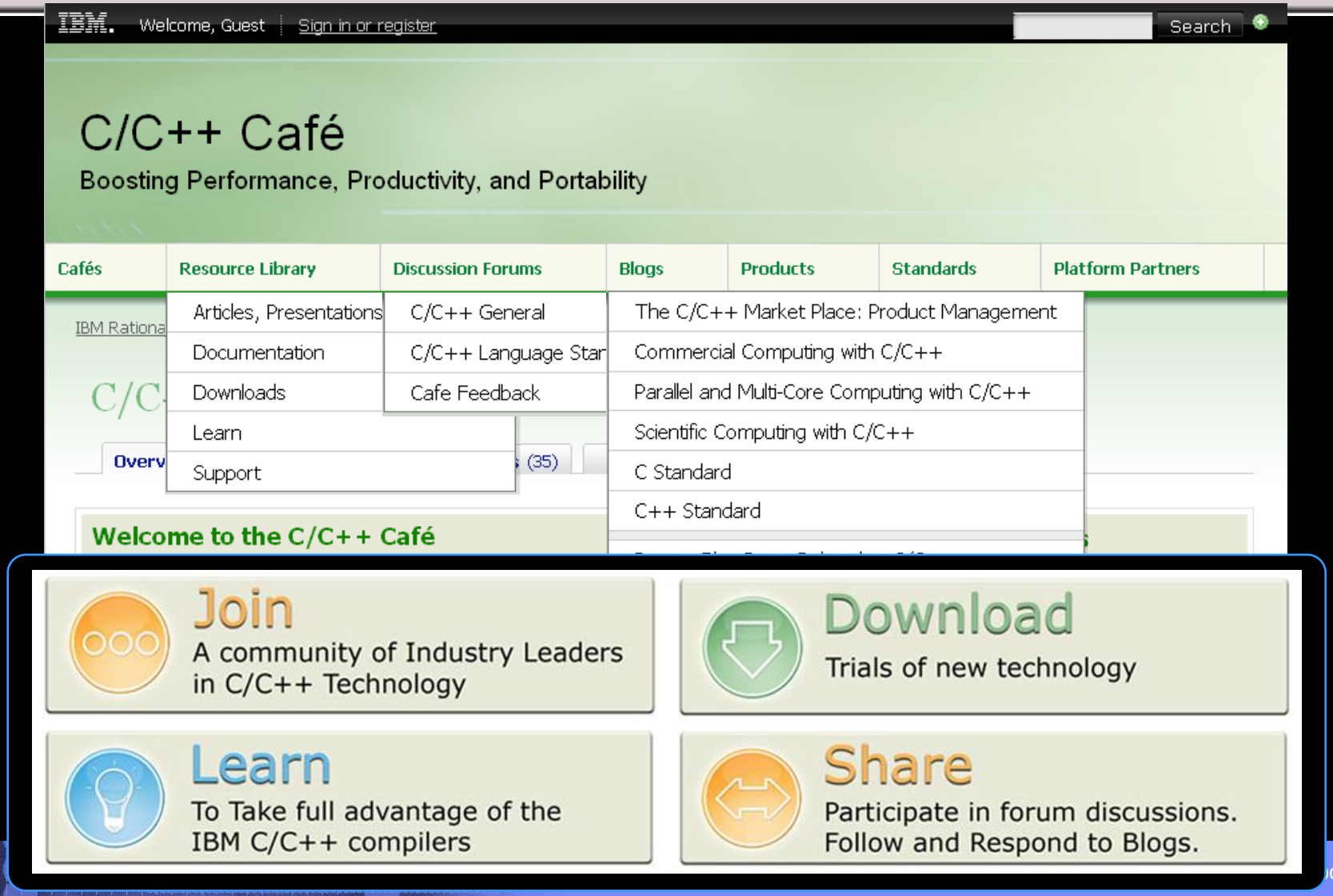
Michael Wong
michaelw@ca.ibm.com
IBM Toronto Lab
Canadian C++ Standard Committee

IBM Rational Disclaimer

- © Copyright IBM Corporation 2011. All rights reserved. **The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.**

The IBM Rational C/C++ Café

ibm.com/rational/cafe/community/ccpp



Agenda

- **Is this legal C++03 code?**
- **C++0x/C1x standard**
- **C++ 0x issues since BoostCon 2010**
- **Bonus 1: C++ 0x Compiler Support Survey**
- **Questions?**

Is this legal C++03 syntax?

```
template<class T> using Vec =
vector<T,My_alloc<T>>;

Vec<double> v = { 2.3, 1.2, 6.7, 4.5 };

//sort(v);

for(auto p = v.begin(); p!=v.end(); ++p)

    cout << *p << endl;
```

Hello Concurrent World

```
#include <iostream>
#include <thread> //#1
void hello() //#2
{
   std::cout<<"Hello Concurrent World"<<std::endl;
}
int main()
{
   std::thread t(hello); //#3
   t.join(); //#4
}
```

Is this valid C++ today? Are these equivalent?

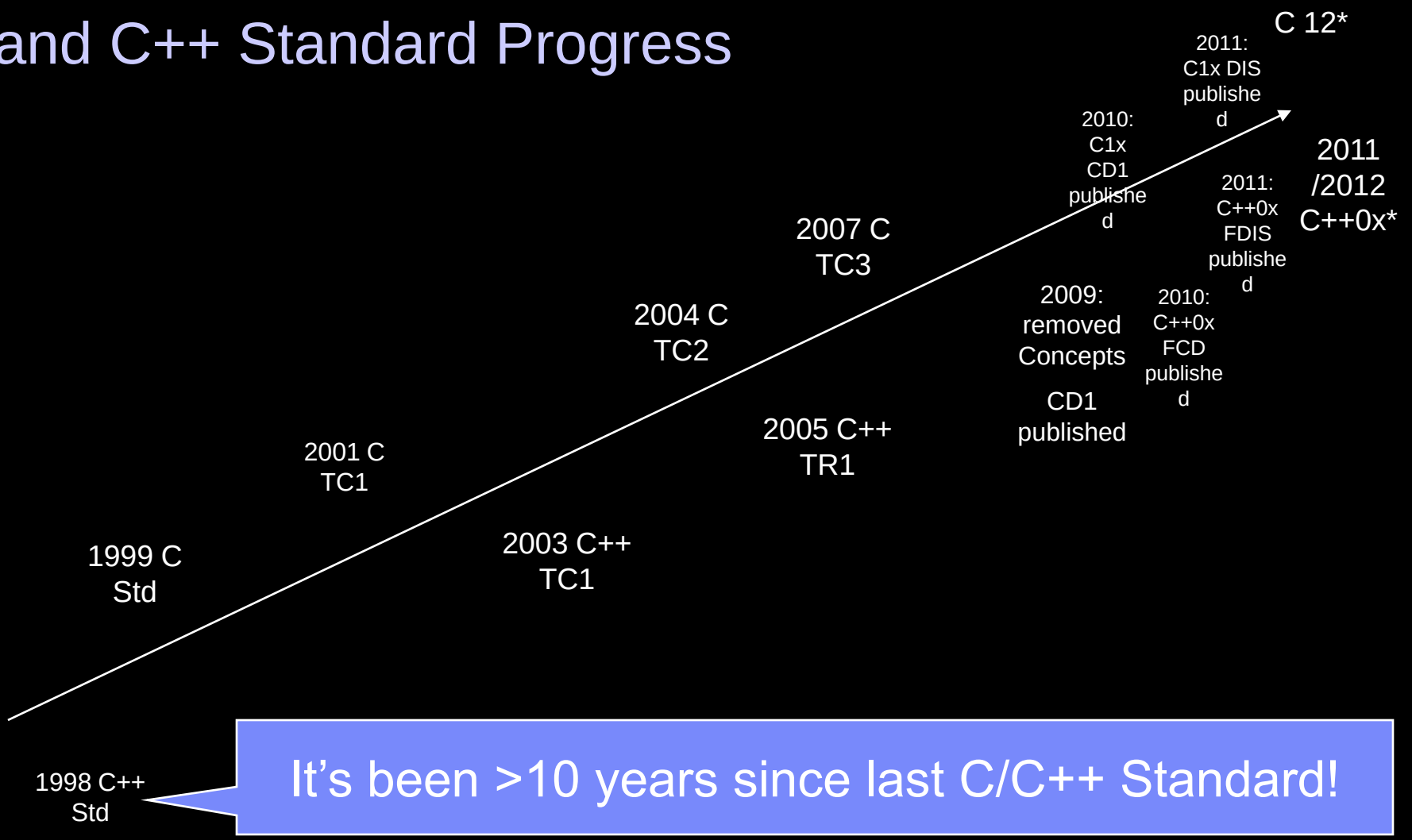
```
int x = 0;

atomic<int> y = 0;

Thread 1:
  x = 17;
  y.store(1,
  memory_order_release);
  // or:       y.store(1);


Thread 2:
  while
  (y.load(memory_order_acq
  uire) != 1)
  // or:       while
  (y.load() != 1)

  assert(x == 17);
```

```
int x = 0;

atomic<int> y = 0;

Thread 1:
  x = 17;
  y = 1;

Thread 2:
  while (y != 1)
        continue;
  assert(x == 17);
```

Agenda

- Is this legal C++03 code?
- C++0x/C1x standard
- C++ 0x issues since BoostCon 2010
- Bonus 1: C++ 0x Compiler Support Survey
- Questions?

C and C++ Standard Progress



*Ratification date subject to change without notice

Major stages of C++0x

Major Stages

SC22 Reg. Ballot (complete)	Ideally all major features present. Usually few comments.
SC22 CD Ballot (optional, 3 months)	Nearly all major features in near-final form. After ballot, need to allow time for disposition of comments and completion of all major features.
SC22 FCD Ballot * (required, 4 months)	All major features in essentially final form. After ballot, need to allow time for disposition of comments.
JTC1 FDIS Ballot (required, 2 months + publication time)	Final text. **Note:** This is an up-down ballot, and no comments are allowed. The only way for a NB to express displeasure is to vote No on the whole standard.

* JTC1 is planning to replace the FCD stage with the ISO DIS stage. This would extend the ballot period to 5 months, but the change is not expected to happen in time to affect us.

DONE in 9/2007

DONE in 9/2008

DONE in 3/2010

DONE in 3/2011

In MADRID

C++0x goals

- Overall goals
  - Make C++ a better language
    - for systems programming
    - for library building
  - Make C++ easier to teach and learn
    - generalization
    - better libraries
- Massive pressure for
  - More language features
  - Stability / compatibility
    - Incl. C compatibility
- Insufficient pressure for
  - More standard libraries
    - The committee doesn't have the resources required for massive library development

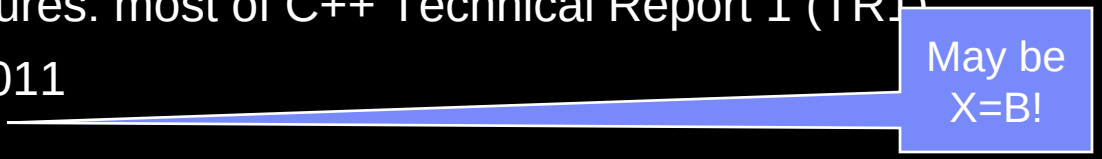
C++0x: areas of language change

- Machine model and concurrency
  - Memory model
  - Threads library, thread pools, futures
  - Atomic API
  - Thread-local storage
- Support for generic programming
  - concepts
  - **auto**, **decltype**, template aliases, Rvalue references, …
  - initialization
- Etc.
  - improved **enum**s
  - **long long**, C99 character types, etc.
  - …
- Modules and dynamically linked libraries
  - Postponed for a TR

Removed June 2009

C++0x, C1x

- **C++0x: Codename for the planned new standard for the C++ programming language**
  - Will replace existing ISO/IEC 14882 standard published in 1998 (C++98) and updated in 2003 (C++03)
  - Many new features to core language
  - Many library features: most of C++ Technical Report 1 (TR1)
  - FDIS in March 2011
  - X=A,B,C,D,E,F?
  - C++11?
- **C1x: Codename for the planned new standard for the C programming language**
  - Will replace existing ISO/IEC 9899 standard published in 1999
  - DIS in March 2011



What's in C++0x?

- **Memory Model and Concurrency [N2138]**
- **Concurrent Libraries [N2094]**
- **Initialization [N2116]**
- **Rvalue references [N2118]**
- **Other primary features**
  - Constant expressions, automatic types
- **Expanded Library from most of TR1**
- **140 features, 600 bug fixes to the standard**
- **What's out?**
  - Concepts [N2081]
  - Garbage Collection (Replaced by smaller proposal)

What's in C1x?

- Alignment specificaiton
- _Noreturn specifier
- Type-generic expressions
- Multithreading support
- Unicode
- Deprecate gets
- Bounds checking interfaces
- Analyzability features
- Subnormal macros
- Anonymous structs and unions
- Static assertions
- Create and pen mode for fopen
- Quick_exit
- Macros for constructing complex values

What is C++0x?

- **Simplifying simple tasks**
  - **Deducing types, ranged for loops,**
- **Initialization**
  - **Uniform { }, no accidental narrowing**
- **Support for low-level programming**
  - **Standard layout types, unions, generalized constant expr**
- **Tools for writing classes**
  - **Init list constructor, inheriting constructor, move, user-defined literals**
- **Concurrency**
  - **Memory model, threads, locks, atomics, mutex, future, shared_future, atomic_future, promise, async()**
- **Standard library components**
  - **Containers, regular expression, random numbers, time, resource mgmt, utility, metaprogramming, Garbage collection ABI**

Sum of all things C++0x

- • __cplusplus
- • alignments
- • attributes
- • atomic operations
- • auto (type deduction from initializer)
- • C99 features
- • enum class (scoped and strongly typed enums)
- • copying and rethrowing exceptions
- • constant expressions (generalized and guaranteed; constexpr)
- • decltype
- • default template parameters for function
- • defaulted and deleted functions (control of defaults)
- • delegating constructors
- • Dynamic Initialization and Destruction with Concurrency
- • explicit conversion operators
- • extended friend syntax
- • extended integer types
- • extern templates
- • for statement ; see range for statement
- • generalized SFINAE rules
- • Uniform initialization syntax and semantics
- • unions (generalized)
- • user-defined literals
- • variadic templates
- • in-class member initializers
- • inherited constructors
- • initializer lists (uniform and general initialization)
- • lambdas
- • local classes as template arguments
- • long long integers (at least 64 bits)
- • memory model
- • move semantics; see rvalue references
- • Namespace Associations (Strong using)
- • Preventing narrowing
- • null pointer (nullptr)
- • PODs (generalized)
- • range for statement
- • raw string literals
- • right-angle brackets
- • rvalue references
- • static (compile-time) assertions (static_assert)
- • suffix return type syntax (extended function declaration syntax)
- • template alias
- • template typedef ; see template alias
- • thread-local storage (thread_local)
- • unicode characters

List of Standard features and papers (110504)

- **C++0x (FDIS):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3291.pdf
- **C++0x (FCD)**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf
- **c++0x (CD1):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n3000.pdf
- **Summary of Core language and Library State:**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2869.html
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2870.html
- **Summary of C++0x CD1**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2871.html
- **Summary of C++ TR1**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2364.html
- **TR1(DTR):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf
- **Decimal TR(PDTR):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2732.pdf
- **Math(FCD):**
  - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2717.pdf
- **C1x(DIS)**
  - http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

What are the STD documents and their status?

- **Library TR1: Draft Technical Report**
- **C++0x: Final Draft International Standard (FDIS), has 13/14 TR1 libraries**
- **C1x: Draft International Standard (DIS)**
- **Special Math Library: Final Committee Draft**
- **Decimal Floating Point TR: Draft Technical Report**
- **POSIX C++: working draft, target 2012/13**
- **C++ ABI: working draft, ongoing discussion on mangling, and common-vendor interoperability**

Feature and defect count

- **Language**
  - 70 features
  - 300 defects (in the C++ Standard)
- **Runtime**
  - 70 features
  - 230 defects (in the C++ Standard)
- **Too many features to be done in one release**
  - stage across many compiler releases over several years
  - not all Standard defects translate into compiler issues

Performance Opportunities, Parallelism, Usability in future C++0x features

- **Improve Execution Time**
  - memory model, concurrency/atomics, rvalue references, pods, variadic template, Concepts, auto
- **Increase Compile Time**
  - Concepts, most template features (except variadic template)
- **Decrease Compile Time**
  - Variadic template
- **Improve usage/teachability**
  - Auto, initialization, decltype
- **Supports concurrency**
  - Atomics, fences, basic mutlithreading library, futures

Features for whom?

- **Library enhancements**
- **For Class writers**
  - Move, rvalue ref, deleted and default functions, delegating, inheriting
- **For Library writers**
  - Static assert, explicit conversion, variadic template, decltype
- **For you**
  - >>, auto, range-based for, nullptr, unicode, raw strings, uniform init, init lists, lambda, trailing return, template aliases, concurrency
- **For everyone else**
  - Class enum, unrestricted union, time library, local types as template args, C99 compat, scoped allocators, constexpr, user-defined literals, relaxed POD, extern template, sizeof on class data members, & and && member functions, in-class init of static data member, attributes

C++0x Library

- **Start with original C++98 library**
  - Improved performance with rvalue reference
  - Used variadic templates to improve compile time
  - Potential binary incompatibility with C++98 library strings
  - Reference counting not allowed
- **Added 13/14 TR1 libraries**
  - Reference wrapper, smart ptrs, return type determination, enhanced member pointer adapter, enhanced binder, generalized functors, type traits, random numbers, tuples, fixed size array, hash tables, regular expressions, C99 cmpat
- **Added threading, unique_ptr,forward_list, many new algorithms**

Removed or Deprecated features

- **Auto as a storage class**
- **Export semantics**
- **Register semantics**
- **Exception specification**
- **Auto_ptr**
- **Bind1st/bind2nd**

Agenda

- Is this legal C++03 code?
- C++0x/C1x standard
- C++ 0x issues since BoostCon 2010
- Bonus 1: C++ 0x Compiler Support Survey
- Questions?

Since BoostCon 2010

- **FCD released and National ballots returned**
- **All national ballots comments addressed through 3 meetings**
- **Released FDIS in Mar 2011 in Madrid**
- **Proof reading done on completed document**
- **Submitted to ISO for next stage National Ballot**
- **Once approved, will ship as C++11**

FCD ballot comments at Aug 2010 meeting

- **All National bodies approved with comment, except one**
- **Fewer comments then from CD1 (~500)**
  - Japan 110
  - Great Britain 142
  - Finland 19
  - US 208
  - Switzerland 36
  - Germany 23
  - Canada 24

	Unresolved	Accepted	Modified	Rejected	Total
CWG	8	72	4	39	123
LWG	123	28	5	25	181
Editor	19	167	9	20	215

Key FCD comments

- **A few comments asking specifically that unimplemented features be removed**
  - Generalized constant expressions (constexpr): gcc
  - Unrestricted unions: gcc
  - alias templates: EDG
  - explicit conversion operator functions: gcc 4.5
  - Delegating constructors: IBM
  - Raw strings: gcc 4.5
  - noexcept: gcc 4.6
  - Implicitly-defined move constructors/assignment operator functions: gcc
  - Non-static data member initializers: fairly similar feature in Microsoft C++/CLI
- **Features that were actually unimplemented at the time were:**
  - Move semantics for *this
  - Inheriting constructors
  - User-defined literals
- **What happened to noexcept and the issue with terminate vs undefined?**

Nov 2010 Fermi Lab meeting

- **The most controversial meeting**
  - "The atomics have become unstable at Fermi Lab"
  - Virtual controls, alignment and noreturn attributes
  - Noexcept default on destructors and delete operators
  - Noexcept in standard library
  - Restricting implicit move generation
  - Implicit inference of noexcept

Mar 2011 Madrid meeting

- **Dealt with key issues/controversies early**
- **Resulted in smooth FDIS with unanimous support**
- **Key design Issues were:**
  - Possible Removal of several features
  - Complications with range_for found in Boost
  - Reconsider the impact of noexcep
  - Issue with a few keyword places with hiding and overriding rules
  - Race condition with copying thrown exceptions

Atomics have become unstable

- **C and C++ atomics are slightly incompatible**
  - C has _Atomic as a qualifier on all types
  - C and C++ support different atomic operations
  - Different mutexes

Operations available on atomic types

	atomic_flag	atomic_bool, atomic<class_type>	atomic_address, atomic<T*>	atomic_integral-type, atomic<integral-type>
test_and_set, clear	Y			
is_lock_free		Y	Y	Y
load, store, exchange, compare_exchange_weak+strong		Y	Y	Y
fetch_add (+=), fetch_sub (-=), ++, --			Y	Y
fetch_or (|=), fetch_and (&=), fetch_xor (^=),				Y

Override controls, alignment, and noreturn attribute

- **Attributes, keywords or contextual keywords**

```
class [[base_check]] Derived
   : public Base {
  public:
    virtual void f [[override]]
   ();
    virtual double g [[final]] (
   int );
    virtual void h [[hiding]] ();
};
```

- .

- Post Fermi-lab

```
class Derived explicit : public
Base {
   public:
      virtual void f () override;
      virtual double g( int ) final;
       virtual void h() new;
};
```

Fails for types hiding types

```
struct X {/*...*/};

struct Y {/*...*/};

class B {

typedef X value_type;

};

class D explicit : public B {

typedef Y value_type; // well-
   formed if "new" can only
   appertain to functions

};
```

- removed the "hiding" feature and the "explicit" annotation on classes
- We don’t know the best solution, so delay this until later

Alignment and noreturn attributes

- **alignas in C++, _Alignas in C**
- **[[noreturn]] in C++, _Noreturn in C**

The problem with range-based for

```
#include <vector>
namespace n
{
  struct X { void begin(); };
  struct Y { void begin(); };
  template<typename T> void begin(T& t) { t.begin(); }
}
int main()
{
  std::vector<n::X> v;
  for (auto i : v) // error
  {
    // ...
  }
}
```

- Produces this error
- error: call of overloaded 'begin(std::vector<n::X>)&' is ambiguous

Range-for problem found from Boost

- **A good solution was found, read N3271**
- **specifies that the range-based for loop should look for member functions begin() and end() first**
  - fall back to the current ADL-based behavior only when the type of the range does not contain either "begin" or "end".

Race condition in copying exceptions

- **A new C++0x feature is the ability to capture exceptions**
  - and rethrows them later without knowing what type they are
  - Allows you to propagate the exceptions across threads
    - Capture exception in one thread,
    - pass std::exception_ptr object across to the other thread
    - Use std::retrow_exception() on that other thread to rethrow
    - Std::async, std::;promise, std::packaged_task's exception propagation is build on this feature
- **Problem**
  - Original proposal required the exception be copied when it was captured with std::current_exception()
  - C++ABI did not store the copy constructor for exception objects
  - std::;current_exception() has no copy ability, so copy not req'd
  - **if any thread modified the object, then we have race**

Common idiom?

- **Catch exceptions by non-const reference, to add further info to the exception, then rethrow**
  - Propagated from another thread through std::async, using std::shared_future or std::;expception_ptr
  - Some platforms allow copying the exception, and some do not

```
try
{
   x = f();
}
catch (Y& y)
{
   y.modify();
   throw;
}
```

```
try
{
shared_future<X>
sp = async(f);
   x = sp.get();
}
catch (Y& y)
{
   y.modify();
   throw;
}
```

Proposed Solutions

- **Current proposed resolution:**
  - make current_exception() copy
  - and make rethrow_exception() copy
- **Known issues:**
  - Pessimizes cases that don't need the copy
  - Doesn't work for reference classes
  - Doesn't work for mutating copy classes
  - Breaks Itanium ABI compatibility

C++ ABI

```
void __cxa_throw(void* thrown_exception,
                 std::type_info* tinfo,
                 void (*dest)(void*) );
```

- **When the client writes “throw X”, the compiler creates a call to __cxa_throw().**
  - Pass void* to an X.
  - Pass type_info* for X.
  - Pass pointer to ~X().
- **Does not pass information on how to copy an X.**

Option 1 for fixing C++ABI

```
void __cxa_throw_copyable(
        void* thrown_exception,
        std::type_info* tinfo,
        void (*dest)(void*),
        void (*copy)(void* d, void* s, size_t sz) );
```

- **Problems:**
  - Our previous OS's do not have this function.
  - Thus, code compiled for C++0x, even if it did not use exception_ptr, could not run on our current and previous OS's.

Option 2 for fixing C++ABI

```
void __cxa_throw(void* thrown_exception,
                 std::type_info* tinfo,
                 void (*dest)(void*) );
```

- **But store copy constructor pointer in type_info.**
  - Which copy constructor pointer?
  - Expensive to extract for *every* call to typeid().
  - ODR violation when mixing C++03/C++0x weak type_info's for same type.

Solution: use good style

- **Catch by value instead of by reference.**
  - Force a copy of Y *exactly* when and where you need it.

```
try
{
   shared_future<X> sp = async(f);
   x = sp.get();
}
catch (Y y)
{
   y.modify();
   throw y;
}
```

Noexcept is a replacement for empty throw spec

- **At March 2011 meeting, deprecated throw-specifcations**
  - throw(), throw (A, B)
- **Check out**
  - "Boost Exception Specification Rationale", "A Pragmatic Look at Exception Specifications"
    - "A non-inline function is the one place a 'throws nothing' [i.e., throw()] exception-specification may have some benefit with some compilers."
- **Replacement is:**
  - void f() **noexcept** {…}
    - Optionally takes a compile-time constant expression
      - True: f will not throw
- **Issues/controversy:**
  - If user violates the promise, should it terminate, or be undefined

Noexcept default on destructors and delete operators

- **Tentative resolution from Aug 2010 meeting**
- **Every destructor/delete op will be noexcept by default**
  - Unless a member or base destructor is noexcept(false)
  - can still explicitly override the default with noexcept(false)
  - Why is this good?
    - Inherently unsafe to use a type with throwing destructor
      - Can lead to 2 exceptions in flight, which violates C++ rules, leading to immediate terminate
    - Could break some code

Applying noexcept to the Standard Library

- **Started applying noexcept liberally to Standard library in Nov 2010 meeting**
  - All empty exception specifications e.g. throw()
  - All descriptions with throws nothing
  - Analyze all move constructors
  - a little too enthusiastically
  - Why?
    - How to you test something if it is all noexcept?

Conservative use of noexcept N3297

- **Liberally application of noexcept was reversed in Mar 2011 meeting with this paper**
  - Hard to test if all functions are not allowed to throw
  - Guidelines for marking noexcept
    - No library destructor should throw.
    - Wide contract is unconditionally noexcept
    - If Swap function, move-constructor, move-assignment is conditionally-wide, then mark as conditionally noexcept
    - Extern “C” functions (atomics) are unconditionally noexcept
  - Lead to some controversial points where it was not clear whether noexcept should be applied
    - See N3269: shared_future(future<R>&& rhs) should be allowed to throw

Implicit deduction of noexcept

```
template< class T > auto forward_with_side_effect( T& t )
noexcept( noexcept(bar(t)) && noexcept(foo(t)) ) -> decltype(foo(t))
{
bar(t);
return foo(t);
}
```

- Ease the burden of writing complicated noexcept declarations
- Going too far with a relative new feature
- Rejected soundly but may be resurrected with more experience

To move or not to move, that is the question!

- **FCD: compilers should implicitly generate move constructors and move assignment operators akin to the copy constructors and copy assignment operators that are currently auto generated.**
  - N3153: Implicit Move Must Go by Dave Abrahams, and N3174: To move or not to move by Bjarne Stroustrup.
  - can breakages be limited by restricting the cases in which the move members are implicitly generated, or whether implicit generation should be abandoned altogether?

Alternatives

- **1.Generate move operations unless a user-specified copy, move, or destructor is declared (e.g., =default), using the default state as the moved-from state**
  - 1.Details: See the “Moving right along” paper
- **2.briefly: generate unless a copy, move, or destructor is declared (e.g., =default), using the state resulting from member moves as the moved-from state).**
  - **1.Details: N3174**
  - 2.This breaks more invariants than [1] but is simpler to implement.
- **3.Generate move operations unless a copy operation is declared (e.g., =default).**
  - 1.This is the FCD status quo which will become the standard unless we see a large majority for an alternative
- **4.Generate move operations only if the programmer asks for it using =default.**
- **5.Never generate move operations.**

Tightened the conditions for generating implicit move: N3203 (Option 2)

- **Treats copy, move and destruction as a group**
  - if you specify any of them manually then the compiler won't generate any move operations
- **if you specify a move operation then the compiler won't generate a copy**
- **would have been nice to prevent implicit generation of copy operations under the same circumstances,**
  - but for backwards compatibility this is still done when it would be done under C++03,
  - though this is deprecated if the user specifies a destructor or only one of the copy operations.

Agenda

- Is this legal C++03 code?
- C++0x/C1x standard
- C++ 0x issues since BoostCon 2010
- Bonus 1: C++ 0x Compiler Support Survey
- Questions?

C++0x Compilers

- **C++0x support publicly available in 110510**
  - GNU 4.6, Mar 28, 2011
  - Intel 12.0 (EDG), Nov 7, 2010
  - IBM xlC++ 11.1, Apr 23, 2010
  - Microsoft Visual C++ 2010, Apr 12, 2010
  - HP aC++ A.06.22 (EDG), Dec, 2008
  - Comeau 4.3.10.1 (EDG), Oct 6, 2008
  - Borland/CodeGear C++ Builder 2009 Compiler 6.10, 2H 2008
- **No C++0x features available publicly as of 100509 on their latest compiler, but we do know from their blogs about their future plans**
  - Sun Studio 12
- **Clang/LLVM status**
  - Language very sparsely supported
  - Library 98% done

All information based on publicly available data

Updated page of C++0x support

- **http://wiki.apache.org/stdcxx/C%2B%2B0xCompilerSupport**
  - Maintained by Martin Sebor, me, and other compiler Tech leads from other company

IBM XL C++ and z/OS C++ compiler status (April, 2011)

- **Released in XL C/C++ for AIX/Linux V10.1 in mid 2008**
  - -qlanglvl=extended0x option (umbrella option for all future 0x features)
  - long long,
  - sync C99 preprocessor (Empty macro arguments, Variadic macros, Trailing comma in enum definition, Concatenation of mixed-width string literals)
- **In C/C++ for AIX for V11.1, in 2Q 2010 (include all of above)**
  - Variadic template
  - Auto
  - Decltype
  - Namespace association
  - Delegating constructor
  - Static assert
- **Supports Boost 1.40**

- **in zOS XL C/C++ V1R11**
  - Extern template
  - Extended friend
  - -qwarn0x
- V1R12
  - Long long
  - C99 preprocessor
  - Auto
  - Decltype
  - Variadic template
  - Namespace association
  - Delegating constructor
  - Static assert

All information subject to change without notice

GNU

- **http://gcc.gnu.org/projects/cxx0x.html**
- **4.3/4.4/4.5/4.6 support:**
  - http://gcc.gnu.org/gcc-4.3/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.4/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.5/cxx0x_status.html
  - http://gcc.gnu.org/gcc-4.6/cxx0x_status.html
- **-std=c++0x or -std=gnu++0x**
- **GNU will write their own C++0x library, libstdC++, as they have always done:**
  - http://gcc.gnu.org/onlinedocs/libstdc++/manual/status.html#id476343
  - Possibly the biggest holdback from their completion
- **Usually supports latest Boost (Boost 1.46.1)**
- **Additional Branch**
  - Concepts
  - Lambda
  - Delegating constructors
  - Raw strings

All information based on publicly available data

GNU 4.3/4.4/4.5/4.6 (110410)

- **4.3: Rvalue Reference, Variadic Template, Static Assert, Decltype, Right Angle Bracket, C99 Preprocessor, Extern Templates, __func__, Long long**
- **4.4:Extending variadic template template parameters, Auto, multideclarator auto, removing auto as storage-class specifier, new function declarator syntax, Propagating exceptions, Strongly-typed enums, New character types, Unicode string literals, Standard Layout types, Default and deleted functions, Inline namespaces**
- **4.5:Initializer lists, Lambdas, Explicit conversion, Raw string literals, UCN Literals, Extending sizeof, Local and unamed types as template arguments**
- **4.6: null pointer, forward declaration of enums, constexpr, unrestricted unions, range-based for, noexcept, move special member functions,**

Intel and likely HP/Comeau (use EDG frontend)

- **Intel C++ 12.0 has**
  - -qstd=c++0x (Linux/Mac OS X), /Qstd:c++0x (Windows)
    - rvalue references
    - Standard atomics
    - Support of C99 hexadecimal floating point constants when in —Windows C++ mode
    - Right angle brackets
    - Extended friend declarations
    - Mixed string literal concatenations
    - Support for long long
    - Variadic macros
    - Static assertions
    - Auto-typed variables
    - Extern templates
    - __func__ predefined identifier
    - Declared type of an expression (decltype)
    - Universal character name literals
    - Strongly-typed enums
    - Lambdas

- Intel C++ Standard Library is based on Microsoft on Windows (uses Dinkwumare) and GNU on Linux (uses GNU's libstdC++), Boost 1.39
- HP aC++ V6 has been quiet about their C++ support, but will likely peggy-back on EDG as they move to new versions, uses STLport 5.1.7 as C++ Library, libstd runtime library matches Rogue Wave Version 1.2.1. , libstd_v2 runtime library matches Rogue Wave Version 2.02.01. Boost 1.38
- Comeau is also very active in delivering C++0x as soon as EDG delivers it to them, runs on multiple platforms, uses their own libcomo 36 based on an old SGI C++ Std Library

All information based on publicly available data

MS VS C++ 2010

- **http://blogs.msdn.com/vcblog/archive/2010/04/06/c-0x-core-language-features-in-vc10-the-table.aspx**
  - Lambdas
  - Auto
  - Static_assert
  - Rvalue references
  - decltype
  - Nullptr
  - Extern templates
  - Right angle brackets
  - Local and unamed types as template arguments
  - Long long
  - Exception_ptr
- **Supports Boost 1.40**
- **Traditionally bought from Dinkumware C++ Library**

All information based on publicly available data

Sun Studio (Version 13 and higher?)

- **Steve Clamage's post (080516):**
  - http://forums.sun.com/thread.jspa?threadID=5296590
  - "Right now, we are working on providing binary compatibility with g++ as an option in the next compiler release. "
  - "We won't release an official (stable, fully-supported) product with C++0X features until the standard is final. Until then, any feature could change in unpredictable ways. "
  - "Beginning some time next year, we expect to have Express releases with some C++0X features. Express releases are our way of providing compilers with experimental features that might not be stable yet. It gives our customers a chance to try them out and provide feedback before they become part of a stable release. "
- **No known plans on C++0x Library based on Steve Clamage's post (070917):**
  - http://forums.sun.com/thread.jspa?threadID=5165721
  - Ships with libCstd, an ancient version of Rogue Wave C++ library from 1999 for binary compatibility
  - Ships with STLport 4.5.3 for enhanced performance
  - Boost 1.34.1
  - Can work with open source Apache C++ Standard Library derived from Rogue Wave 4.1.2
  - "A new C++ standard is in progress, planned for completion in 2009. We will release a new compiler, C++ 6.0, conforming to the new standard, including a fully-conforming standard library as the default. The new library will be shipped as part of Solaris.
    We also plan to maintain compatibility with C++ 5.x and libCstd as an option. Details are still in the planning stage. "

All information based on publicly available data

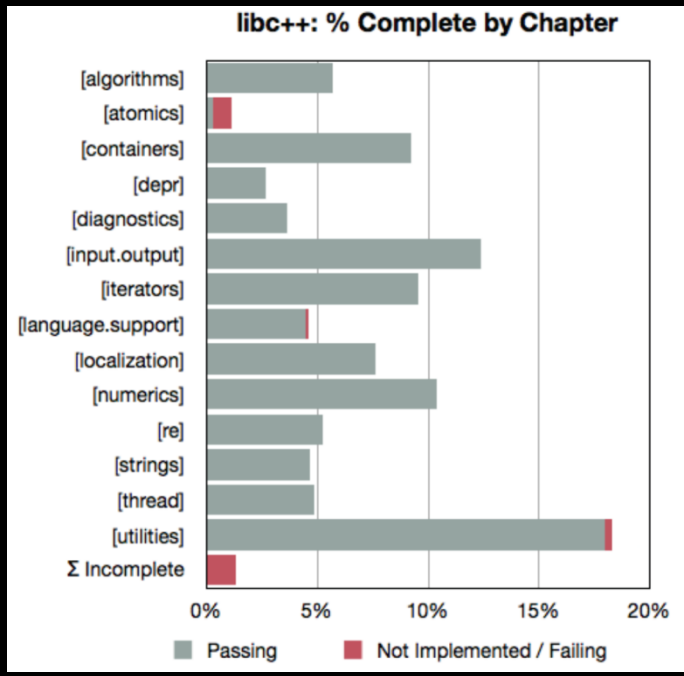
Borland/CodeGear C++Builder Compiler 6.10 2009

- http://www.codegear.com/article/38534/images/38534/CBuilder2009Datasheet.pdf
- **Rvalue references**
- **decltype**
- **Variadic templates (in testing)**
- **Scoped enumerations**
- **static_assert**
- **explicit conversion operators**
- **Attributes [[final]] and [[noreturn]]**
- **alignof**
- **Type traits**
- **Unicode character types and literals**
- **long long**
- **variadic macros**
- **Dinkumware C++Std Library**
- **Boost 1.35**

All information based on publicly available data

Clang/llvm

- **Core language:http://clang.llvm.org/cxx_status.html**
  - Very far from complete, can't compile basic tests
  - Variadic template, rvalue ref, extern templ, inline namespace, long long
- **Library:http://libcxx.llvm.org/index.html**
- **On Mac OS X/i386/x86_64**
- **Writes its own library libc++.a:**
  - About 98% complete
  - Only missing atomics



Food for thought and Q/A

- **This is the chance to get a copy before you have to pay for it:**
  - C++ : http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3291.pdf
  - C: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
- **Participate and feedback to Compiler**
  - What features/libraries interest you or your customers?
  - What problem/annoyance you would like the Std to resolve?
  - Is Special Math important to you?
  - Do you expect 0x features to be used quickly by your customers?
- **Talk to me at my blog:**
  - http://www.ibm.com/software/rational/cafe/blogs/cpp-standard

My blogs and email address

- michaelw@ca.ibm.com
- Rational C/C++ cafe:
  http://www.ibm.com/software/rational/cafe/community/ccpp
- My Blogs:
- C++ Language & Standard
  http://www.ibm.com/software/rational/cafe/blogs/cpp-standard
- Parallel & Multi-Core Computing
  http://www.ibm.com/software/rational/cafe/blogs/ccpp-parallel-multicore
- Commercial Computing
  http://www.ibm.com/software/rational/cafe/blogs/ccpp-commercial
- Boost test results
  http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911
- C/C++ Compilers Support Page
  http://www.ibm.com/software/awdtools/ccompilers/support/
- C/C++ Feature Request Interface
  http://www.ibm.com/support/docview.wss?uid=swg27005811
- XL Fortran Compiler Support Page
  http://www.ibm.com/software/awdtools/fortran/xlfortran/support/
- XL Fortran Feature Request Interface
  http://www.ibm.com/support/docview.wss?uid=swg27005812

Acknowledgement

- **Some slides are borrowed from committee presentations by various committee members, their proposals, and private communication**