Bigger Better More

The new C++ Standard Library

Thomas Witt
April 24 2009

Landscape

- C99 (2003)
- Technical Report on C++ Library Extensions (TR1) Jan 2006
- Committee Draft (CD) Oct 2008
- C++ 0x
- TR2
- .NET, Java, Python, anybody?

C++ 0x Goals

- Maintain stability and compatibility
- Prefer libraries to language extensions
- Prefer generality to specialization
- Support both experts and novices
- Increase type safety
- ...

C++ 0x Results

- Major core language features
- Few high profile library extensions
  - Networking?
  - File system access?
  - XML?
- Library spec grew more than twofold

Disclaimer

Churn

Disclaimer

Disclaimer

Churn

Language Changes

- C++ 0x brings significant change to the core language
- New features make library writing and authoring easier
- Features were retrofitted to the Standard Library

Rvalue References

- Move semantics

```
basic_string(basic_string&& str);

std::string failure =
mogrify(std::string(“Brilliant idea”));
```

- Perfect forwarding
- Move helpers

Concepts

- Constrained and unconstrained templates don't mesh well
- Large parts of the standard library need to be conceptified
- Providing primitives
- Converting existing specification to code

```
auto concept MoveConstructible<typename T>
  : Constructible<T, T&&>
{
  requires RvalueOf<T>
        && Constructible<T, RvalueOf<T>::type>;
}

auto concept HasPlus<typename T, typename U>
{
  typename result_type;
  result_type operator+(const T&, const U&);
}
```

```
template <
  ValueType T, Allocator Alloc = allocator<T>>
requires MoveConstructible<T>
class vector;

template <InputIterator Iter>
requires AllocatableElement<
           Alloc, T, Iter::reference>
      && MoveAssignable<T>
void insert(
         const_iterator position
       , Iter first
       , Iter last);
```

Variadic Templates

```
template<
    CopyConstructible F
  , CopyConstructible... BoundArgs>
unspecified bind(F f, BoundArgs... bound_args);

template <class... Args>
  requires AllocatableElement<
             Alloc, T, Args&&...>
        && MoveAssignable<T>
iterator emplace(
             const_iterator position
           , Args&&... args);
```

Initializer Lists

```
requires AllocatableElement<Alloc, T, const T&>
vector(
    initializer_list<T>
  , const Allocator& = Allocator());

std::vector<int> listOfInt
                   = { 0, 17, 42, 7, 9, 39 };
```

There is more

- Thread support
- long long
- Constant expressions
- Deleted functions
- Explicit conversion operators
- nullptr_t

TR1 Revisited

- Bind, function, reference wrappers
- Smart Pointers
- Regular Expressions
- Random numbers
- Math special functions

TR1 Revisited

- Containers
  - array
  - tuple
  - Unordered associative containers
- Type traits
- C99 library additions

TR1 Revisited

- C++ 0x incorporates TR1
- All of ~~Gaul~~ TR1?
- No - one *little* piece moved to its own document
- Extensions to the C++ Library to Support Mathematical Special Functions

Smart Pointers

- Strict ownership
    - unique_ptr
- Shared ownership
    - shared_ptr
- Weak ownership
    - weak_ptr

unique_ptr

```
template <
  class T, class D = default_delete<T>>
class unique_ptr;
```

- auto_ptr done right
- Moveable only
- No destructive copy
- Custom deleter

shared_ptr extensions

- Allocators
- Aliasing
- Factory functions
- Atomic access
- Comparison

Algorithms

- 16 new algorithms
- Add obvious omissions
- Follow existing practice
- Useful additions

- copy_n
- uninitialized_copy_n
- all_of
- any_of
- none_of
- copy_if

- find_if_not
- partition_copy
- partition_point
- is_partitioned
- iota
- minmax_element

- is_sorted
- is_sorted_until
- is_heap
- is_heap_until

Containers

- const_iterator arguments in insert/erase
- cbegin(), cend()
- shrink_to_fit()
- data()
- vector<bool> has a spec!

Emplace

```
template <class... Args>
requires AllocatableElement<
           Alloc, T, Args&&...>
void emplace_back(Args&&... args);
```

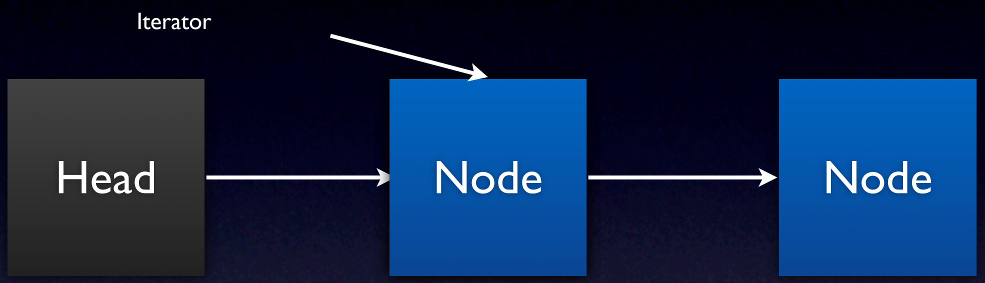
- Placement insert
- Non-moveconstructible elements in containers

forward_list

```
template <
 class T, class Allocator = allocator<T> >
class forward_list;
```

- Singly linked list
- On par with C implementation
- Insert after

Insert after?



- insert-, emplace-, splice-, erase_after
- No O(N) insert, erase, …
- begin_before()

Diagnostics

- System error support
  - class system_error
  - class error_code
  - class error_category
  - class error_condition
- Detailed error reporting from I/O streams

Strings

- Uniform use of string
- Simple numeric access
- Unicode support
- No more Copy-On-Write

wstring_convert

```
wstring_convert<codecvt_utf8<wchar_t> myconv();

std::string mbstring
  = myconv.to_bytes(L"Hello\n");
```

Exceptions

- Transporting exceptions between threads
  - class exception_ptr
  - current_exception()
  - copy_exception(e)
- Nesting exception objects

Allocators

Well ... No

Multithreading

Atomics

- Memory ordering
- Atomic types
  - Integral
  - Address
  - Generic
- Fences

Threads

- Well thread
- Mutexes
  - recursive_mutex
  - timed_mutex
- Condition variables
- Locks and lockers

thread

- Unique ownership
- Creation

```
template <class F> explicit thread(F f);

template <class F, class ...Args>
thread(F&& f, Args&&... args);
```

- Join
- Detach

Futures

- unique_future
- shared_future
- promise
- packaged_task

packaged_task

```
int thgttg()
{
    return 42;
}

std::packaged_task<int()> task(thgttg);
std::unique_future<int> fi=task.get_future();

std::thread task(std::move(task));

// ...

fi.wait();
```

One more thing ...

Time

- duration

  seconds, minutes, nanoseconds

- time_point

  Epoch plus/minus duration

- Clock

  system_clock, monotonic_clock

Odds and ends

- numeric_limits
  - lowest
  - digits10, max_digits10
- prev(it), next(it)
- min(1, 2, 3, ...)
- aligned_storage

Deprecated features

- auto_ptr
- Again: auto_ptr!
- iterator_traits, iterator, iterator tags
- Binders

Conclusion

- C++ 0x focuses on foundations and facilities that require language support
- Standard Library is changed significantly
- TR 2 will focus on Standard Library extensions
  - Filesystem
  - Networking

Good Luck!