New C++ standard

- Tentatively called C++0x
- Complete draft expected in September 2008
- Will probably be introduced in 2009, so C++09
- Almost 100% compatible with current standard
  - Don't know exactly what is not compatible
- Affects core language and Standard Library

- See Wikipedia for detailed info and links

Language changes

- Rvalue reference (object&&)
  - Discussed later
- `extern template` to avoid auto-instantiation
- Initializer lists (only generated by compiler)

```
SequenceClass(std::initializer_list<int> list);
SequenceClass a = {1,2,3,4};
```

- Auto type deduction

```
for (auto iter=vec.begin(); ...)
int var;
decltype(var) var1;
```

- Range based for loops (a la python)
- Lambda expressions and functions
- Concepts to define template requirements
- Raw strings `R"[ a string with a \ and "]"`

Language changes (cont'd)

- Constructors can call other constructors
- `nullptr` instead of 0 or NULL
  - 0 and NULL might get deprecated in future standard
- >> is now valid in templates (no space needed)
  - > is angle bracket if last open bracket was an angle
  - Not backward compatible for very obscure use of >
- Explicit conversion operators
- Template typedefs

```
template<typename T>
using Registry = std::map<std::string, T>;
```

- Variadic templates (useful for tuple)
- Add features for easier garbage collection
- Constructors and functions can be forbidden

```
void f(int);
void f(double) = delete;  // no double->int conversion
```

Standard Library changes

- Support for threading
  - Also atomic support in language
- Class tuple
- Hash tables (unordered_[multi]set, map)
- Regex (superset of boost)
- Smart pointers (from boost)

Rvalue reference

- AKA move semantics
- Will be supported in Standard Library
- Useful in resize of containers
  - Elements are moved, not copied
    - Uses std::move
    - Needs constructor and operator= for Class&&
- Useful in array arithmetic (for instance a*b)
  - Operand can be recognized as temporary and use =*
  - Requires four operator* functions instead of one

```
Array<T> operator* (Array<T>&& l, const Array<T>& r)
  { l *= r; return l; }
```

Future

- More multi-threading support
- Addition of modules
- Exception Specifications might get obsolete
  - Herb Sutter says: do not use them
- vector<bool>will probably be replaced by a bitset class and not specialized anymore