# C++0x

## A quick and dirty introduction

TechZone 23. November 2007

Olve Maudal (olve.maudal@tandberg.com)
Lars Gullik Bjønnes (lgb@tandberg.com)

This is a quick and dirty introduction to the current status of C++0x as of November 2007. Through small code snippets, we will illustrate some of the proposals that have already been accepted and integrated into the current working paper (N2461) for C++0x. Please refer to N2461 for a complete and accurate status report.

Disclaimer: We make no attempt to be complete or accurate. Also, beware that if you are reading this after November 2007, the information is probably out of date.

## Generalized Constant Expressions

```
constexpr int square(int x) { return x * x; }

int main() {
    int values[square(7)];
    // ...
}
```

This is supposed to work in C++0x.

Read more: N2235

## Static Assert

A static assert can evaluate an integral constant-expression and print out a diagnostic message if the program is ill-formed.

```
int main() {
    static_assert(sizeof(int) == 4,
                  "This code only works for sizeof(int) == 4");
    // ... code depending on a particular size of the integer
}
```

With C++0x, this code might compile fine for a 32-bit machine, while it probably fails to compile for a 16-bit or 64-bit machine.

Read more: N1720

## Variadic Templates

```
#include <iostream>

template <typename... Args>
void f(Args... args)
{
    std::cout << (sizeof... args) << std::endl;
}

int main() {
    f();
    f( 42, 3.14 );
    f( "one", "two", "three", "four" );
}
```

My experimental C++0x compiler prints out:

```
0
2
4
```

Read more: N2080/N2242

## Right Angle Brackets

Two consecutive right angle brackets no longer need to be separated by whitespace.

```
#include <vector>

typedef std::vector<std::vector<int>> Table;

int main() {
    Table t;
    // ...
}
```

This code will compile cleanly.

Read more: N1757

# Scoped Enumerations

We now get a strongly typed version of enum. Eg,

```
enum class Color { red, green, blue };

int main() {
    Color c = Color::red;  // OK
    c = red;               // error
    int x = Color::blue;   // error
    // ...
}
```

Read more: N2347

## Alignment Support

Two new keywords supporting alignment have been introduced:

- An `alignof` expression yields the alignment requirement of its operand type
- `alignas` can be used to request strict alignment requirements

Eg,

```
template <std::size_t Len, std::size_t Alignment>
struct aligned_storage {
    typedef struct {
        alignas(Alignment) unsigned char __data[Len];
    } type;
};

int main() {
    aligned_storage<197,256> my_storage;
    std::size_t n = alignof(my_storage); // n == 256
    // ...
}
```

Read more: N2341

# Decltype

Decltype let you get the type of an expression. Eg,

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v;
    v.push_back(4);
    v.push_back(2);
    for ( decltype(v.begin()) i = v.begin(); i != v.end(); ++i ) {
        std::cout << (*i);
    }
}
```

Notice how we can now create an iterator without knowing the type. My experimental C++0x compiler prints out:

```
42
```

Read more: N2343

# Auto

Auto is similar to decltype but with a nicer syntax. Eg,

```
int main() {
    auto a = 4;
    std::vector<int> v;
    v.push_back(a);
    v.push_back(2);
    for ( auto i = v.begin(); i != v.end(); ++i ) {
        // ...
    }
}
```

This is also supposed to work.

Read more: N1984

# Defaulted and Deleted Functions

C++98

```
class Foo {
public:
    // default copy constructor is OK
    // default assignment operator is OK
    // ...
private:
    Foo();  // hide
    ~Foo(); // hide
    // ...
};
```

C++0x

```
class Foo {
public:
    Foo() = deleted;
    ~Foo() = deleted;
    Foo(const Foo&) = default;
    Foo& operator=(const Foo &) = default;
    // ...
private:
    // ...
};
```

You can now tell the compiler if you want the default special member functions or not.

Read more: N2326

# Rvalue Reference

```
#include <iostream>

struct Bar {
    int x;
    Bar(int i) : x(i) {}
};

void foo( Bar & b ) {
    std::cout << "lvalue" << std::endl;
}

void foo( Bar && b ) {
    std::cout << "rvalue" << std::endl;
}

int main() {
    Bar b(3);
    foo(b);
    foo(4);
}
```

* A reference type that is declared using & is called an lvalue reference.
* A reference type that is declared using && is called an rvalue reference.

My experimental C++0x compiler prints out:

```
lvalue
rvalue
```

Read more: N2118

## Extending sizeof

In C++98, you would have to create an object to get the size of a member. In C++0x the following will be possible:

```
#include <iostream>

struct Foo {
    int x;
};

int main() {
    int i = sizeof(Foo::x);
    std::cout << i << std::endl;
}
```

Read more: N2150

## Delegating Constructors

```
class Foo {
    int value;
public:
    Foo( int v ) : value(v) {
        // some common initialization
    }
    Foo() : Foo(42) { }
    // ...
}
```

Finally, C++ will be able to do constructor delegation. Hurray!

Read more: N1986

# std::array

```
int main() {
    std::array<int, 5> a = { 0,4,2,3,1 };
    std::cout << "Size: " << a.size() << '\n';
    std::sort(a.begin(), a.end());
    std::copy(a.begin(), a.end(),
          std::ostream_iterator<int>(std::cout));
    std::cout << '\n' <<a[0] << a[4] << '\n';
}
```

```
Size: 5
01234
04
```

Read more: N1548/N1836

## std::shared_ptr

```
struct Foo {
        Foo() {}
        Foo(int i) : ptr(new int(i)) {}
        std::shared_ptr<int> ptr;
};

int main() {
        Foo f;
        {
                Foo f2(5);
                std::cout << *f2.ptr << '\n';
                f = f2;
        }
        std::cout << *f.ptr << '\n';
}
```

Read more: N1450/N1836

# std::tuple

```
int main() {
    typedef std::tuple<int, int, int> T;
    T t1 = std::make_tuple(1, 3, 5);
    std::cout << "Size: "
              << std::tuple_size<T>::value
              << " Index 2: " << std::get<1>(t1)
              << " Tuple: " << t1 << '\n';
    int a, b, c;
    std::tie(a, b, c) = t1;
    std::cout << a << b << c << '\n';
}
```

```
Size: 3 Index 2: 3 Tuple: (1 3 5)
135
```

Generic replacement for std::pair<>

Read more: N1403

# std::minmax

```
void foo(int a, int b) {
   int x, y;
   std::tie(x, y) = std::minmax(a, b);
   assert(x <= y);
}

void bar(std::vector<int> const & v) {
    std::vector<int>::const_iterator x, y;
    std::tie(x, y) =
        std::minmax_element(v.begin(), v.end());
    assert(*x <= *y);
}
```

Read more: N1990

## std::function

```
void foo(void *p) {std::cout<<"func " <<p<< '\n'}

struct bar {
    void operator()(void * p) const
    {std::cout <<"operator()" << p << '\n'}
};

int main() {
    std::function<void(void*)> f = foo; f(&f);
    f = bar(); f(&f);
}
```

```
func 0xbf87f658
operator() 0xbf87f658
```

Read more: N1836

## std::bind

```
int foo(void * p, std::string const & s) {
    std::cout << s << ": " << p << std::endl;
    return 0;
}

int main() {
    std::function<void(void *)> f =
        std::bind(foo, _1, "foo");
    f(&f);
}
```

```
foo: 0xbfc39220
```

Read more: N1455/N1836

# std::regex

```
int main() {
    std::regex reg(".*(b[a-z]+n).*");
    std::string s = "The lazy brown fox";
    std::smatch what;
    if (std::regex_match(s, what, reg)) {
        std::cout << "found match" << std::endl;
    }
    std::cout << what[1] << std::endl;
}
```

```
found match
brown
```

Read more: N1429/N1836

## std::thread

```
void f(int i) {
    for (int j = 0; j < i; ++j) {
       // complicated timeconsuming stuff
    }
}

int main() {
   std::thread t(f, 5);
   t.join(); // wait for thread to finish
}

Also support for std::mutex and std::condition
```

Read more: N2447

## More Stuff

- type_traits
- Diagnostics
- Hash Tables
- Random Numbers
- Atomic Types
- Unicode Support
- C99 Stuff
- date_time
- Template aliases
- extern template
- long long
- Variadic templates

...and more...

## Learn more

- The C++ Standards Committe (WG21)
  http://www.open-std.org/jtc1/sc22/wg21
- State of C++ Evalution, post-Kona 2007 Meeting (N2432)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2432.html
- C++ Library Working Group Status Report, post-Kona 2007 Meeting (N2433)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2433.html
- Working Draft (N2461)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2461.pdf
- A video of a talk given by Bjarne Stroustrup at Univ of Waterloo in July 2007
  http://csclub.uwaterloo.ca/media/C++0x%20-%20An%20Overview.html
- The C++0x branch of GCC
  http://gcc.gnu.org/projects/cxx0x.html
- Bjarne Stroustrup's homepage
  http://www.research.att.com/~bs/
- Herb Sutter's blog
  http://herbsutter.spaces.live.com/