C++0x

A quick and dirty introduction

Olve Maudal (oma!at!pvv.org), 24. September 2007

This is a quick and dirty introduction to the current status of C++0x as of September 2007. Through small code snippets, I will illustrate some of the proposals that have already been accepted and integrated into the current working paper (N2369) for C++0x. Please refer to N2389 for a complete and accurate status report.

Disclaimer: My actual understanding of what WG21 is working on is very limited. I make no attempt to be complete or accurate. Also, beware that if you are reading this in October 2007 or later, the information is probably out of date.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

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: N2140

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

Learn more

- The C++ Standards Committe (WG21)
  http://www.open-std.org/jtc1/sc22/wg21
- State of C++ Evalution, pre-Kona 2007 Meeting (N2389)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2389.html
- C++ Library Working Group Status Report, pre-Kona 2007 Meeting (N2390)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2390.html
- Working Draft (N2369)
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2369.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/