Modernizing C++98 Code With C++11 and C++14

Petr Zemek

Senior Developer
Security/Engineering/VirusLab
http://petrzemek.net

C++ Developers' Guild, 2015-12-04

Range-Based For Loop

C++98

```
1  for (vector<int>::iterator i = v.begin(), e = v.end();
2           i != e; ++i) {
3      process(*i);
4  }
5
6  int numbers[] = {1, 2, 3, 4, 5};
7  for (size_t i = 0, e = sizeof(numbers)/sizeof(numbers[0]);
8           i != e; ++i) {
9      numbers[i] *= 2;
10 }
```

Range-Based For Loop

C++98

```
1  for (vector<int>::iterator i = v.begin(), e = v.end();
2           i != e; ++i) {
3      process(*i);
4  }
5
6  int numbers[] = {1, 2, 3, 4, 5};
7  for (size_t i = 0, e = sizeof(numbers)/sizeof(numbers[0]);
8           i != e; ++i) {
9      numbers[i] *= 2;
10 }
```

C++11

```
11 for (int& x : v) {
12     process(x);
13 }
14
15 for (int& x : numbers) {
16     x *= 2;
17 }
```

Automatic Type Deduction

C++98

```
1 std::map<std::string, int>::iterator it = m.find(key);
```

Automatic Type Deduction

C++98

```
1 std::map<std::string, int>::iterator it = m.find(key);
```

C++11

```
2 auto it = m.find(key);
```

Automatic Type Deduction

C++98

```
1 std::map<std::string, int>::iterator it = m.find(key);
```

C++11

```
2 auto it = m.find(key);
```

Advantages?

Automatic Type Deduction

C++98

```
1 std::map<std::string, int>::iterator it = m.find(key);
```

C++11

```
2 auto it = m.find(key);
```

Advantages?

```
3 std::map<std::string, int> m;
4 for (const std::pair<std::string, int>& p : m) {
5     // ...
6 }
```

Automatic Type Deduction

C++98

```
1 std::map<std::string, int>::iterator it = m.find(key);
```

C++11

```
2 auto it = m.find(key);
```

Advantages?

```
3 std::map<std::string, int> m;
4 for (const std::pair<std::string, int>& p : m) {
5     // ...
6 }

7 std::map<std::string, int> m;
8 for (const auto& p : m) {
9     // ...
10 }
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();


3 int i;
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();


3 int i;

4 auto i = 1;
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();


3 int i;

4 auto i = 1;
```

Disadvantages?

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();


3 int i;

4 auto i = 1;
```

Disadvantages?

```
5 std::vector<bool> f();
6 // ...
7 auto b = f()[2];
```

Automatic Type Deduction (Cont'd)

```
1 unsigned int size = m.size();

2 auto size = m.size();


3 int i;

4 auto i = 1;
```

Disadvantages?

```
5 std::vector<bool> f();
6 // ...
7 auto b = f()[2];

8 std::vector<bool> f();
9 // ...
10 bool b = f()[2];
```

Function Return-Type Deduction

C++98

```
1 std::map<std::string, int> foo() {
2     std::map<std::string, int> m;
3     // ...
4     return m;
5 }
```

Function Return-Type Deduction

C++98

```
1 std::map<std::string, int> foo() {
2     std::map<std::string, int> m;
3     // ...
4     return m;
5 }
```

C++14

```
6 auto foo() {
7     std::map<std::string, int> m;
8     // ...
9     return m;
10 }
```

Lambda Expressions

C++98

```
1 bool cmpById(const Person& p1, const Person& p2) {
2     return p1.getId() < p2.getId();
3 }
4 // ...
5 std::sort(people.begin(), people.end(), cmpById);
```

Lambda Expressions

C++98

```
1 bool cmpById(const Person& p1, const Person& p2) {
2     return p1.getId() < p2.getId();
3 }
4 // ...
5 std::sort(people.begin(), people.end(), cmpById);
```

C++11

```
6 std::sort(people.begin(), people.end(),
7     [](const Person& p1, const Person& p2) {
8         return p1.getId() < p2.getId();
9     }
10 );
```

Lambda Expressions

C++98

```
1 bool cmpById(const Person& p1, const Person& p2) {
2     return p1.getId() < p2.getId();
3 }
4 // ...
5 std::sort(people.begin(), people.end(), cmpById);
```

C++11

```
6 std::sort(people.begin(), people.end(),
7     [](const Person& p1, const Person& p2) {
8         return p1.getId() < p2.getId();
9     }
10 );
```

C++14

```
11 std::sort(people.begin(), people.end(),
12     [](const auto& p1, const auto& p2) {
13         return p1.getId() < p2.getId();
14     }
15 );
```

Smart Pointers

C++98

```
1 Resource* createResource();
2
3 Resource* r = createResource();
4 // ...
5 delete r;
```

Smart Pointers

C++98

```
1 Resource* createResource();
2
3 Resource* r = createResource();
4 // ...
5 delete r;
```

C++11

```
6 std::unique_ptr<Resource> createResource();
7
8 auto r = createResource();
```

Smart Pointers

C++98

```
1 Resource* createResource();
2
3 Resource* r = createResource();
4 // ...
5 delete r;
```

C++11

```
6 std::unique_ptr<Resource> createResource();
7
8 auto r = createResource();
```

Smart pointers:

- `std::unique_ptr`
- `std::shared_ptr`
- `std::weak_ptr`
- ~~`std::auto_ptr`~~ (from C++98, deprecated – why?)

Scoped Enumerations

C++98

```
1 enum Color {
2     Red,
3     Green,
4     Blue
5 };
```

Scoped Enumerations

C++98

```
1 enum Color {
2     Red,
3     Green,
4     Blue
5 };
```

Disadvantages?

Scoped Enumerations

C++98

```
1 enum Color {
2     Red,
3     Green,
4     Blue
5 };
```

Disadvantages?

- scoping, safety, unknown underlying type, no fwd decls

Scoped Enumerations

C++98

```
1 enum Color {
2     Red,
3     Green,
4     Blue
5 };
```

Disadvantages?

- scoping, safety, unknown underlying type, no fwd decls

C++11

```
6 enum class Color {
7     Red,
8     Green,
9     Blue
10 };
```

Unordered Containers

C++98

`std::{set,map}`

Unordered Containers

C++98

`std::{set,map}`

	Lookup	Insert	Erase
Complexity			

Unordered Containers

C++98

`std::{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Unordered Containers

C++98

`std::{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

C++11

`std::unordered_{set,map}`

Unordered Containers

C++98

`std::{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

C++11

`std::unordered_{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Unordered Containers

C++98

`std::{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

C++11

`std::unordered_{set,map}`

	Lookup	Insert	Erase
Complexity	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$

* average case; $\mathcal{O}(n)$ in worst case

Type Aliases

C++98

```
1 typedef unsigned int Size;
2 typedef std::size_t (*HashFunc)(const MyClass&);
```

Type Aliases

C++98

```
1 typedef unsigned int Size;
2 typedef std::size_t (*HashFunc)(const MyClass&);
```

C++11

```
3 using Size = unsigned int;
4 using HashFunc = std::size_t (*)(const MyClass&);
```

Type Aliases

C++98

```
1 typedef unsigned int Size;
2 typedef std::size_t (*HashFunc)(const MyClass&);
```

C++11

```
3 using Size = unsigned int;
4 using HashFunc = std::size_t (*)(const MyClass&);


5 template<typename T>
6 using Dictionary = std::map<std::string, T>;
7
8 Dictionary<int> d;
```

Type-Safe Null Pointer Constant

C++98

```
1 void f(int);
2 void f(int *);
3
4 f(0);    // calls void f(int);
5 f(NULL); // ?
```

Type-Safe Null Pointer Constant

C++98

```
1 void f(int);
2 void f(int *);
3
4 f(0);     // calls void f(int);
5 f(NULL); // ?

6 int i = NULL; // !?
```

Type-Safe Null Pointer Constant

C++98

```
1 void f(int);
2 void f(int *);
3
4 f(0);    // calls void f(int);
5 f(NULL); // ?

6 int i = NULL; // !?
```

C++11

```
7 f(0);       // calls void f(int);
8 f(nullptr); // calls void f(int *);
9
10 int *pi = nullptr; // OK
11 bool  b = nullptr; // OK (b is false)
12 int   i = nullptr; // !
```

Explicit Overrides

C++98

```
1 class B: public A {
2     // ...
3
4     virtual void foo(int i);
5 };
```

Explicit Overrides

C++98

```
1 class B: public A {
2     // ...
3
4     virtual void foo(int i);
5 };
```

C++11

```
6 class B: public A {
7     // ...
8
9     virtual void foo(int i) override;
10 };
```

Explicitly Deleted Functions

C++98

```
1 class NonCopyable {
2 private:
3     NonCopyable(const NonCopyable&); // not defined
4     NonCopyable& operator=(const NonCopyable&); // ditto
5 };
```

Explicitly Deleted Functions

C++98

```
1 class NonCopyable {
2 private:
3     NonCopyable(const NonCopyable&); // not defined
4     NonCopyable& operator=(const NonCopyable&); // ditto
5 };
```

C++11

```
6 class NonCopyable {
7 public:
8     NonCopyable(const NonCopyable&) = delete;
9     NonCopyable& operator=(const NonCopyable&) = delete;
10 };
```

Initializer Lists and Braced Initialization

C++98

```
1 std::vector<int> v;
2 v.push_back(1);
3 // ...
4 v.push_back(5);
```

Initializer Lists and Braced Initialization

C++98

```
1 std::vector<int> v;
2 v.push_back(1);
3 // ...
4 v.push_back(5);
```

C++11

```
5 std::vector<int> v = {1, 2, 3, 4, 5};
```

Initializer Lists and Braced Initialization

C++98

```
1 std::vector<int> v;
2 v.push_back(1);
3 // ...
4 v.push_back(5);
```

C++11

```
5 std::vector<int> v = {1, 2, 3, 4, 5};

6 std::map<std::string, std::vector<double>> m = {
7   {"foo", {0.0, 1.5}},
8   {"bar", {0.0, 10.0, 100.1}},
9   // ...
10 };
```

Initializer Lists and Braced Initialization

C++98

```
1 std::vector<int> v;
2 v.push_back(1);
3 // ...
4 v.push_back(5);
```

C++11

```
5 std::vector<int> v = {1, 2, 3, 4, 5};

6 std::map<std::string, std::vector<double>> m = {
7   {"foo", {0.0, 1.5}},
8   {"bar", {0.0, 10.0, 100.1}},
9   // ...
10 };

11 void f(std::vector<int> v);
12 f({1, 2, 3});
13
14 std::vector<int> g() {
15   return {1, 2, 3};
16 }
```

Variadic Templates

C++98

```
 1 template <typename T1>
 2 void print(const T1& val1) {
 3     std::cout << val1 << "\n";
 4 }
 5
 6 template <typename T1, typename T2>
 7 void print(const T1& val1, const T2& val2) {
 8     std::cout << val1;
 9     print(val2);
10 }
11
12 template <typename T1, typename T2, typename T3>
13 void print(const T1& val1, const T2& val2, const T3& val3)
14     std::cout << val1;
15     print(val2, val3);
16 }
17
18 // ...
19
20 print("I am ", 30, " years old."); // I am 30 years old.
```

Variadic Templates (Cont'd)

C++11

```
 1 template <typename T>
 2 void print(const T& value) {
 3     std::cout << value << "\n";
 4 }
 5
 6 template <typename U, typename... T>
 7 void print(const U& head, const T&... tail) {
 8     std::cout << head;
 9     print(tail...);
10 }
11
12 print("I am ", 30, " years old."); // I am 30 years old.
```

Raw String Literals

C++98

```
1 std::string code(
2     "int main() {\n"
3     "    printf(\"Hello\\n\");\n"
4     "    return 0;\n"
5     "}\n"
6 );
```

Raw String Literals

C++98

```
1 std::string code(
2     "int main() {\n"
3     "    printf(\"Hello\\n\");\n"
4     "    return 0;\n"
5     "}\n"
6 );
```

C++11

```
7  std::string code(R"(
8      int main() {
9          printf("Hello\n");
10         return 0;
11     }
12 )");
```

Standard Library Enhancements

C++98

```
1 str[0]
2 str[str.size() - 1]
```

Standard Library Enhancements

C++98

```
1 str[0]
2 str[str.size() - 1]
```

C++11

```
3 str.front()
4 str.back()
```

Standard Library Enhancements

C++98

```
1 str[0]
2 str[str.size() - 1]
```

C++11

```
3 str.front()
4 str.back()
```

C++98

```
5 std::ostringstream out;
6 out << number;
7 std::string numberAsStr = out.str();
```

Standard Library Enhancements

C++98

```
1 str[0]
2 str[str.size() - 1]
```

C++11

```
3 str.front()
4 str.back()
```

C++98

```
5 std::ostringstream out;
6 out << number;
7 std::string numberAsStr = out.str();
```

C++11

```
8 auto numberAsStr = std::to_string(number);
```

Shameless Advertisement

Petr Zemek

Co je nového v C++11

http://cs-blog.petrzemek.net/2012-12-04-co-je-noveho-v-cpp11

Petr Zemek

Co je nového v C++14

http://cs-blog.petrzemek.net/2014-09-20-co-je-noveho-v-cpp14

Petr Zemek

Méně známé novinky v C++11 a C++14

http://cs-blog.petrzemek.net/2015-10-06-mene-zname-novinky-v-cpp11-a-cpp14

Petr Zemek

Improving C++98 Code With C++11

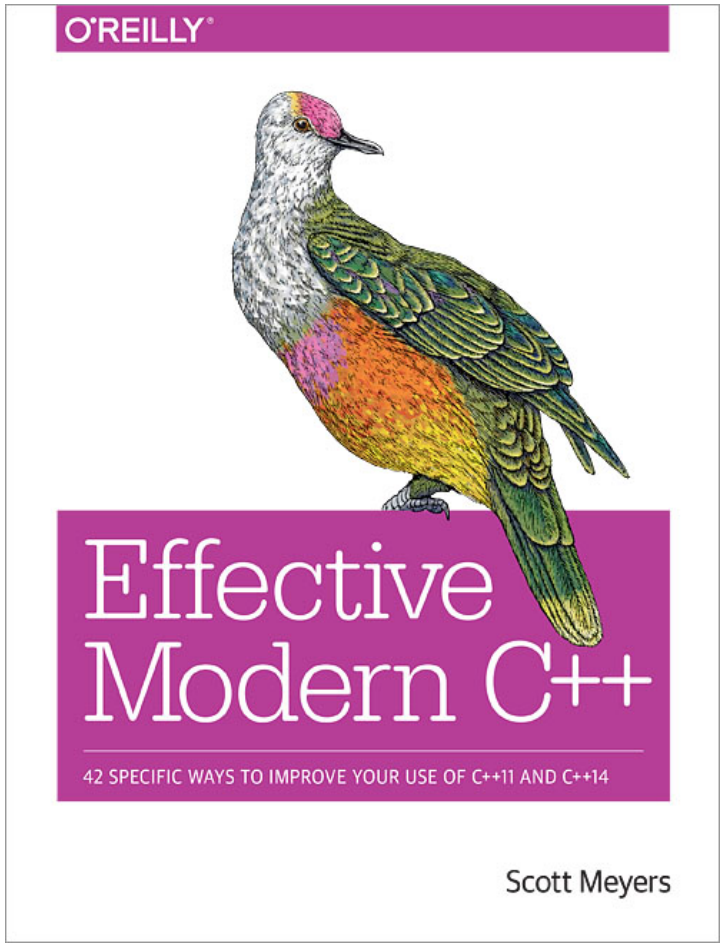
http://blog.petrzemek.net/2014/12/07/improving-cpp98-code-with-cpp11/

Recommended Reading

Scott Meyers

Effective Modern C++

O'Reilly Media, November 2014, 336 pages



Discussion