C++11 Rocks

GCC Edition

Alex Korban

Contents

Introduction **9**

C++11 guiding principles 9

Type Inference **11**

`auto` 11

Some things are still manual 12

More than syntactic sugar 12

Why else do we need it? 13

Objection, Your Honor! 14

Diving in 14

A little bit of `auto` history 16

`decltype` 17

Side effects 18

`declval` 19

`auto`, `decltype` - how about both at once? 20

Lambda Expressions **22**

Why do we need this thing? 22

Return type 23

Lambda parameters 24

Lambda body 24

Storing lambdas 25

`std::function` to the rescue 25

References to outside context 26

Closures . . . 27
Capturing in C++11 . . . 27
Capturing by reference . . . 28
Default capture modes . . . 29
Capturing class members . . . 30
Limitations of capturing . . . 32
Mutable lambdas . . . 32
Conversion to function pointers, nested lambdas, recursion . . . 34
How not to shoot yourself in the foot . . . 35
Rules of thumb for lambdas . . . 37
Lambda syntax in all its glory . . . 38

Template Features **39**

Variadic templates . . . 39
What else are variadic templates good for? . . . 40
Back to the example . . . 40
Working with parameter packs . . . 41
Traversing template parameter packs . . . 44
Constraining parameter packs to one type . . . 44
More places to expand a parameter pack . . . 45
Nested variadic templates . . . 45
Multiple parameter packs in function templates, non-type parameter packs 46
Template aliases . . . 47
Using `using` instead of `typedef` . . . 49
Closing angle brackets are officially allowed to tail-gate . . . 49
Local and unnamed types as template arguments . . . 49
`extern` templates . . . 51
Default values for function template parameters . . . 52
Arbitrary expressions in template deduction contexts . . . 53

Class Features **55**

In-class initializers for non-static data members . . . 55

Inheriting constructors . . . 58

Delegating constructors . . . 61

Default methods . . . 63

Deleted methods . . . 65

`override` and `final` . . . 67

Extended friend declarations . . . 68

Nested class access rights . . . 70

The Dream of Uniform Initialization **72**

Embrace the braces . . . 73

`initializer_list` . . . 75

Narrowing conversions . . . 77

Distortions of the uniformity continuum . . . 77

Want a move-only type in your `vector`? . . . 78

`auto` + `{}` . . . 78

`<>` + `{}` = ? . . . 79

Surprising consequences of narrowing . . . 80

What's the verdict? . . . 80

Move Semantics **81**

What are the benefits? . . . 81

How does this stuff work? . . . 82

Revision part 1: lvalue vs. rvalue . . . 82

Revision part 2: const attribute + lvalue/rvalue . . . 84

Revision part 3: reference initialization . . . 85

Rvalue references . . . 86

Overload resolution 87
Implementing move semantics 89
Compiler generated move operations 91
Implementing your own move operations 91
`std::move` 94
Moving it right 94
Rvalue references to `const` values 94
Derived class construction 95
Move construction in terms of assignment 95
Check for self-assignment 96
Don't make move constructors `explicit` 97
Reference qualifiers for member functions 97
Move-only types 100

Perfect Forwarding Problem and Solution **102**
Reference collapsing and rvalues in templates 104
How the `forward` template works 105
Bonus: implementation of `std::move` 108

`constexpr` Mechanism **110**
What else is it good for? 110
What's in a constant expression? 110
`constexpr` variables 110
`const` and `constexpr` 111
`constexpr` functions 111
Literal types 113
A couple more notes on `constexpr` functions 114

Range-based `for` Loop **116**

`nullptr` **119**

What's the advantage over `NULL`? . . . 120

`enum` Changes **121**

Scoped enums . . . 121

Specifying the underlying type . . . 122

Forward declaration . . . 122

Compile Time Assertions **126**

Literals **128**

Unicode support and literals . . . 128

Unicode character literals . . . 129

Raw literals . . . 129

User defined literals . . . 131

Types of literals . . . 132

Literal operators for integers . . . 133

Character and string literal operators . . . 137

`noexcept` **138**

`noexcept` for your own functions . . . 138

`noexcept` operator . . . 139

When to use `noexcept` . . . 140

A couple more notes on `noexcept` . . . 141

Explicit Conversion Operators **142**

Inline Namespaces **143**

Why not just add a `using` statement? . . . 144

Alignment 146

alignof . . . 146

alignas . . . 146

sizeof Applied to Non-static Data Members 148

Memory Model 149

Multi-threading related semantics . . . 149

Thread local storage 150

Thread local initialization . . . 151

Thread local destruction . . . 152

Attributes 153

Standard attributes . . . 154

POD Types, Trivial Types, and Standard Layout Types 155

Trivial classes . . . 155

Trivially copyable classes . . . 155

Trivial operations . . . 156

Standard layout types and classes . . . 156

Changed restrictions on unions . . . 157

Discriminated unions . . . 159

C99 Compatibility Features 163

Deprecated and Removed Features 164

Writing Cross-Platform Code 165

C++14 **167**

Return type deduction for functions 167

Generic lambdas 168

Extended capturing in lambdas 168

Revised restrictions on `constexpr` functions 169

`constexpr` variable templates 169

More language changes 170

Beyond C++14 170

Conclusion **171**

Contact Information and License Agreement **172**

License agreement 172