C++11/14 Rocks

Clang 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 . . . 50
`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 **82**

What are the benefits? . . . 82
How does this stuff work? . . . 83
Revision part 1: lvalue vs. rvalue . . . 83
Revision part 2: const attribute + lvalue/rvalue . . . 85
Revision part 3: reference initialization . . . 86
Rvalue references . . . 87

Overload resolution . . . 88
Implementing move semantics . . . 90
Compiler generated move operations . . . 92
Implementing your own move operations . . . 92
`std::move` . . . 95
Moving it right . . . 95
Rvalue references to `const` values . . . 95
Derived class construction . . . 96
Move construction in terms of assignment . . . 96
Check for self-assignment . . . 97
Don't make move constructors `explicit` . . . 98
Reference qualifiers for member functions . . . 98
Move-only types . . . 101

Perfect Forwarding Problem and Solution **103**
Reference collapsing and rvalues in templates . . . 105
How the `forward` template works . . . 106
Bonus: implementation of `std::move` . . . 109

`constexpr` Mechanism **111**
What else is it good for? . . . 111
What's in a constant expression? . . . 111
`constexpr` variables . . . 111
`const` and `constexpr` . . . 112
`constexpr` functions . . . 112
Literal types . . . 114
A couple more notes on `constexpr` functions . . . 115

Range-based `for` Loop **117**

nullptr **120**

What's the advantage over NULL? . . . 121

enum Changes **122**

Scoped enums . . . 122

Specifying the underlying type . . . 123

Forward declaration . . . 123

Compile Time Assertions **127**

Literals **129**

Unicode support and literals . . . 129

Unicode character literals . . . 130

Raw literals . . . 130

User defined literals . . . 132

Types of literals . . . 133

Literal operators for integers . . . 134

Character and string literal operators . . . 138

noexcept **140**

noexcept for your own functions . . . 140

noexcept operator . . . 141

When to use noexcept . . . 142

A couple more notes on noexcept . . . 143

Explicit Conversion Operators **144**

Inline Namespaces **145**

Why not just add a using statement? . . . 146

Alignment **148**

`alignof` 148

`alignas` 148

`sizeof` Applied to Non-static Data Members **150**

Memory Model **151**

Multi-threading related semantics 151

Thread local storage **152**

Thread local initialization 153

Thread local destruction 153

Attributes **155**

Standard attributes 156

POD Types, Trivial Types, and Standard Layout Types **157**

Trivial classes 157

Trivially copyable classes 157

Trivial operations 158

Standard layout types and classes 158

Changed restrictions on `unions` 159

Discriminated unions 161

C99 Compatibility Features **165**

Deprecated and Removed Features **166**

C++14 **167**

Return type deduction for functions 167

Redeclaration 168

Generic lambdas 170
Extended capturing in lambdas 170
Revised restrictions on `constexpr` functions 171
`constexpr` variable templates 172
`decltype(auto)` 172
Literals 173
`[[deprecated]]` attribute 174
Aggregate initialization 174
Beyond C++14 175

Writing Cross-Platform Code **176**

Conclusion **178**

Contact Information and License Agreement **179**
License agreement 179