C++11/14 Rocks

VS2013 Edition

Alex Korban

Contents

Introduction **18**

Type Inference **20**

`auto` . . . 20

`decltype` . . . 24

Side effects . . . 25

Trailing return types . . . 26

Non-standard behavior and bugs in Visual Studio 2013 . . . 27

`const` qualifier retained incorrectly by `decltype` . . . 27

Using class members and `this` in `decltype` expressions . . . 28

Referring to a member of a member object inside `decltype` . . . 28

`decltype` doesn't respect access control . . . 29

Using a `friend` function defined inside a class in a `decltype` expression 29

Lambda Expressions **31**

Why do we need this thing? . . . 31

Return type . . . 32

Lambda parameters . . . 33

Lambda body . . . 33

Storing lambdas . . . 34

`std::function` to the rescue . . . 34

References to outside context . . . 35

Closures . . . 36

Capturing in C++11 . . . 36

Capturing by reference . . . 38
Default capture modes . . . 39
Capturing class members . . . 40
Limitations of capturing . . . 41
Mutable lambdas . . . 42
Conversion to function pointers, nested lambdas, recursion . . . 43
How not to shoot yourself in the foot . . . 44
Rules of thumb for lambdas . . . 46
Lambda syntax in all its glory . . . 47
Non-standard behavior and bugs in VS2013 . . . 47
Nested lambdas lead to slow compile times and huge object files . . . 47
Using a lambda as a comparator for an STL container . . . 48
Using a lambda as a default function argument . . . 48
Using `const` from enclosing scope inside a lambda . . . 48
Invalid initialization of closure type variables . . . 49
Lambda returning a capturing lambda . . . 49
Lambdas as ternary operator operands . . . 50

Template Features **51**

Variadic templates . . . 51
What else are variadic templates good for? . . . 52
Back to the example . . . 52
Working with parameter packs . . . 53
Traversing template parameter packs . . . 55
Constraining parameter packs to one type . . . 56
More places to expand a parameter pack . . . 57
Nested variadic templates . . . 57
One function template, two parameter packs . . . 58

Template aliases . . . 59
Using `using` instead of `typedef` . . . 60
Closing angle brackets are officially allowed to tail-gate . . . 61
Local and unnamed types as template arguments . . . 61
`extern` template qualifier . . . 62
Default values for function template parameters . . . 64
Non-standard behavior and bugs in Visual Studio 2013 . . . 65
Empty parameter pack not handled . . . 65
Variadic template-template parameter troubles . . . 65
Can't use a template alias inside another template . . . 66
Can't define a private static member using a template alias type . . . 67
Default template arguments not applied in class context . . . 67
`sizeof...` always returns 1 if used in a template alias . . . 68

Class Features **69**

In-class initializers for non-static data members . . . 69
Delegating constructors . . . 72
Default methods . . . 74
Deleted methods . . . 75
`override` and `final` . . . 77
Extended friend declarations . . . 79
Nested class access rights . . . 81
Non-standard behavior and bugs in Visual Studio 2013 . . . 82
`default` can't be applied to move operations . . . 82
`final` isn't honored on class templates . . . 82
`friend` lookup goes too far . . . 82
Nested class access fails with enough nesting . . . 83

Move Semantics and Rvalue References **84**
Lvalue/rvalue revision . . . 85
`const` attribute . . . 86
Reference initialization . . . 87
Rvalue references . . . 88
xvalues . . . 90
Move semantics implementation . . . 90
Moving members . . . 92
`std::move` . . . 94
Moving it right . . . 94
rvalue references to `const` values . . . 95
Derived class construction . . . 95
Implementing the move constructor in terms of assignment . . . 95
Check for self-assignment . . . 96
Don't make move constructors `explicit` . . . 97
Move-only types . . . 98
Non-standard behavior and bugs in Visual Studio 2013 . . . 99
Move operations never generated by the compiler . . . 99
Move operations don't disable default copy operations . . . 100
Rvalue reference not recognized correctly . . . 100

Perfect Forwarding **102**
The forwarding problem and solution . . . 102
Reference collapsing and templates involving rvalue reference arguments . . . 104
How perfect forwarding works . . . 105
Bonus: the implementation of `std::move` . . . 107

Range-based `for` loop **110**
Non-standard behavior and bugs in Visual Studio 2013 . . . 113

`nullptr` **114**

Non-standard behavior and bugs in VS2013 115

`enum` Changes **116**

Scoped `enums` 116

Specifying the underlying type 116

Forward declaration 117

Non-standard behavior and bugs in Visual Studio 2013 119

Explicit Conversion Operators **121**

Non-standard behavior and bugs in Visual Studio 2013 121

Raw String Literals **123**

The Dream of Uniform Initialization **125**

Embrace the braces 126

`initializer_list` 128

Narrowing conversions 129

Distortions of the uniformity continuum 130

Want a move-only type in your `vector`? 131

`auto` + {} 131

<> + {} = ? 131

Surprising consequences of narrowing 132

What's the verdict? 133

Non-standard behavior and bugs in Visual Studio 2013 133

New initialization syntax doesn't work for member arrays 133

Aggregate initialization doesn't work in the constructor initialization list . 133

Nested initializer lists can cause crashes or memory leaks 134

Double `delete` of `initializer_list` elements 135

Nested initializer lists compile when they shouldn't . . . 136
Can't combine `auto` with an initializer list of function pointers . . . 136
Uniform initialization of a type with a user-defined destructor . . . 137
Empty parameter pack expansion error when combined with uniform initialization syntax . . . 137

Smart Pointers **139**

unique_ptr . . . 139
Custom deleters . . . 140
Array specialization . . . 141
`make_unique` . . . 141
`shared_ptr` . . . 142
Custom deleters . . . 143
Thread safety . . . 143
Performance . . . 144
`make_shared` and `allocate_shared` . . . 145
`enable_shared_from_this` . . . 145
`shared_ptr<void>` . . . 146
Class hierarchies and smart casts . . . 147
`weak_ptr` . . . 148
Non-standard behavior and bugs in Visual Studio 2013 . . . 150
A `bool` can be assigned to `unique_ptr` . . . 150
An `std::array` of `unique_ptr`'s cannot be moved or used within other containers . . . 151
`shared_ptr` debugging . . . 151

Tuple Types **152**

`make_tuple` . . . 153
Accessing tuple elements . . . 154

Multiple assignment . . . 154
Type information . . . 155
Comparison operators . . . 156
More advanced uses of `tuple` . . . 156
Iterating over a tuple with template metaprogramming . . . 156
Nested tuples . . . 158
Tuple concatenation . . . 158
Non-standard behavior and bugs in Visual Studio 2013 . . . 158

Binding Arguments with `std::bind` 159

Rearranging and duplicating parameters . . . 160
Passing bound values by reference . . . 161
Using `bind` with overloaded functions . . . 161
Using `bind` with member functions . . . 162
Binding data members . . . 164
Using `bind` with function objects . . . 165
Using `bind` with lambdas . . . 165
Function composition with nested `bind` expressions . . . 165
`bind` vs. lambda expressions . . . 166
Non-standard behavior and bugs in Visual Studio 2013 . . . 167
Can't pass a reference to `this` as an argument to `bind` . . . 167
Can't use `bind` with member functions taking rvalue references . . . 168
Can't always use `bind` with data members . . . 168

Generalized Function Objects 169

Using the `function` template . . . 169
Parameter and return type conversions . . . 171
Checks and comparisons . . . 171

Performance and code size . . . 172
Non-standard behavior and bugs in Visual Studio 2013 . . . 172
No support for move-only types . . . 172
Can't use `std::function` with member functions . . . 172
`void`-returning `std::function` can't swallow return type . . . 173

Regular Expressions **174**
Syntax . . . 175
Wildcards . . . 175
Repetition . . . 175
Character sets . . . 176
Character classes . . . 176
Anchors . . . 177
Or . . . 177
Word boundaries . . . 177
Capture groups and back references . . . 177
Escaping . . . 178
Analyzing strings and extracting information . . . 178
Flags . . . 181
Handling multiple matches with iterators . . . 182
Tokenization . . . 184
Searching and replacing . . . 185
Unicode support and localization . . . 187
Attaching a locale to a regex object . . . 188

Compile Time Assertions and Type Traits **189**
`static_assert` . . . 189
Type traits . . . 190

Primary traits . . . 191
Composite traits . . . 192
Type properties . . . 192
Array-specific traits . . . 196
Type relationships . . . 197
Type manipulation . . . 198
`const` and `volatile` modifications . . . 198
References . . . 199
Sign conversions . . . 199
Array modifications . . . 200
Pointer modifications . . . 201
Other transformations . . . 201
Additional template aliases in C++14 . . . 205
Non-standard behavior and bugs in Visual Studio 2013 . . . 206
`static_assert` in a class definition . . . 206
Compiling with code analysis may wrongly trigger a `static_assert` . . . 206
`is_function` fails to detect a static member function . . . 207
`remove_pointer` fails with a `const` pointer to a function . . . 207
`is_pod` provides wrong results for types with user-defined constructors . . . 208
`is_trivially_copyable` fails on arrays of scalar types . . . 208
Construction-related type traits fail on abstract types . . . 208
`is_assignable` provides wrong results in some cases . . . 209
`is_destructible` doesn't work . . . 209
`is_convertible` gives wrong results . . . 209
`alignment_of` implementation deviates from the standard . . . 210
`enable_if` combined with two type parameters compile error . . . 210

New STL Containers **212**

`forward_list` . . . 212

`array` . . . 213

Element operations . . . 215

Container operations . . . 215

Initialization . . . 216

Array as tuple . . . 217

Hash tables . . . 218

`unordered_map` . . . 218

Custom hash . . . 221

`unordered_multimap`, `unordered_set`, `unordered_multiset` . . . 222

Other STL Improvements **224**

Container improvements . . . 224

Support for move semantics . . . 224

Better `const_iterator` support . . . 225

`emplace` methods . . . 225

Reducing container capacity . . . 226

Immutable set elements . . . 226

String improvements . . . 227

Miscellaneous . . . 227

Iterator improvements . . . 227

Iterator adapters to support move operations . . . 228

`prev()`/`next()` functions . . . 229

C++14 specializations for operator functors in `functional` . . . 229

New algorithms . . . 230

`bool all_of(Iter first, Iter last, Pred pred)` . . . 230

`bool any_of(Iter first, Iter last, Pred pred)` . . . 231

bool none_of(Iter first, Iter last, Pred pred) 231
Iter find_if_not(Iter first, Iter last, Pred pred) 231
OutIter copy_if(InIter first, InIter last, OutIter result, Pred pred) 231
OutIter copy_n(InIter first, Size n, OutIter result) 231
uninitialized_copy_n(InIter first, Size n, OutIter result) 232
OutIter move(InIter first, InIter last, OutIter result) . . 232
OutIter move_backward(InIter first, InIter last, OutIter result) 232
is_partitioned(InIter first, InIter last, Pred pred) 233
pair<OutIter1, OutIter2> 233
partition_copy(InIter first, InIter last, OutIter1 out_true, OutIter2 out_false, Pred pred) 233
Iter partition_point(Iter first, Iter last, Pred pred) . . . 233
RAIter partial_sort_copy(InIter first, InIter last, RAIter result_first, RAIter result_last) 234
RAIter partial_sort_copy(InIter first, InIter last, RAIter result_first, RAIter result_last, Compare comp) 234
bool is_sorted(Iter first, Iter last) 234
bool is_sorted(Iter first, Iter last, Compare comp) 234
Iter is_sorted_until(Iter first, Iter last) 234
Iter is_sorted_until(Iter first, Iter last, Compare comp) . 234
bool is_heap(Iter first, Iter last) 235
bool is_heap(Iter first, Iter last, Compare comp) 235
Iter is_heap_until(Iter first, Iter last) 235
Iter is_heap_until(Iter first, Iter last, Compare comp) . . 235
pair<const T&, const T&> minmax(const T& a, const T& b) . . 235
pair<const T&, const T&> minmax(const T& a, const T& b, Compare comp) 235

pair<const T&, const T&> minmax(initializer_list<T> lst) . 235
pair<const T&, const T&> minmax(initializer_list<T> lst, Compare comp) 235
const T& min(initializer_list<T> lst) 236
const T& min(initializer_list<T> lst, Compare comp) 236
const T& max(initializer_list<T> lst) 236
const T& max(initializer_list<T> lst, Compare comp) 236
pair<Iter, Iter> minmax_element(Iter first, Iter last) . . . 236
pair<Iter, Iter> minmax_element(Iter first, Iter last, Compare comp) 236
void iota(Iter first, Iter last, T value) 236

Random Number Facility **238**

Engines 239
mersenne_twister_engine 240
linear_congruential_engine 240
subtract_with_carry_engine 240
random_device 240
Engine adapters 240
seed_seq 241
Distributions 241
Uniform distributions 242
Normal distributions 242
Bernoulli distributions 243
Poisson distributions 244
Sampling distributions 244
Non-standard behavior and bugs in Visual Studio 2013 245

Rational Arithmetic and Time Support Libraries **246**
Rational number representation and manipulation . . . 246
Time utilities . . . 248
Time durations . . . 248
Clocks and time points . . . 253
Time points . . . 255
Non-standard behavior and bugs in Visual Studio 2013 . . . 256

Concurrency Support: High Level **258**
Memory model overview . . . 258
Asynchronous code execution . . . 258
Launch policies and lazy evaluation . . . 261
`future` . . . 261
Polling and waiting for task completion . . . 262
Getting multiple threads to wait for one result . . . 264
Non-standard behavior and bugs in Visual Studio 2013 . . . 266
`async` can't handle move-only arguments . . . 266
Initialization of statics not thread-safe . . . 266
Non-blocking `future` destructor . . . 266

Concurrency Support: Building Blocks **268**
Rolling your own threads . . . 268
Joining threads . . . 269
Detaching threads . . . 270
Thread IDs and native handles . . . 271
`promise` . . . 272
Helper functions for use with threads . . . 275
`packaged_task` . . . 276

Non-standard behavior and bugs in Visual Studio 2013 277
`thread` constructor doesn't take rvalue reference arguments 277
Calling `join` after `main` exits causes the program to hang 277
Wrong exception type thrown by `promise` 278
`future::get` doesn't throw when shared state in `packaged_task` is abandoned 278
`packaged_task` wrapping a `void` or reference-returning function isn't movable 279
Incorrect handle value in a default-constructed `thread` object 279

Concurrency Support: Exceptions, Thread Local Storage 280
Manual exception handling in threads 280
`exception_ptr` 280
Functions for working with `exception_ptr` 281
Thread local storage 283
Non-standard behavior and bugs in Visual Studio 2013 284

Concurrency Support: Synchronization 285
Mutexes 285
Timed mutexes 287
Locking multiple mutexes 287
Locks 288
`call_once` 291
Condition variables 291
Limiting wait time 294
Other lockable types 295
`notify_all_at_thread_exit` 295
A note on `const` and `mutable` 296
Non-standard behavior and bugs in Visual Studio 2013 296

Concurrency Support: Atomic Data Types and Operations **298**
Atomic data types . . . 298
Alternative C-style interface . . . 300
Atomic flag . . . 303
Low level atomic interface . . . 303
Fences . . . 304
Non-standard behavior and bugs in Visual Studio 2013 . . . 305

Miscellaneous Features **307**
Alignment . . . 307
`addressof` template . . . 308
Using `type_info` in containers . . . 308
`bitset` and `valarray` improvements . . . 308
New stream functionality . . . 309
`system_error` header . . . 309
C99 compatibility . . . 309

Deprecated and Future Features **310**
Deprecated features . . . 310

Beyond Visual Studio 2013 **311**
C++14 . . . 312
Return type deduction for functions . . . 313
Generic lambdas . . . 313
Extended capturing in lambdas . . . 314
Revised restrictions on `constexpr` functions . . . 314
`constexpr` variable templates . . . 315
More language changes . . . 315
Beyond C++14 . . . 316

Conclusion **317**

Contact Information and License Agreement **318**

License agreement 318