

# Table of Contents

<b>Preface</b>	1
<b>Chapter 1: Classical Polymorphism and Generic Programming</b>	7
Concrete monomorphic functions	8
Classically polymorphic functions	8
Generic programming with templates	10
Summary	14
<b>Chapter 2: Iterators and Ranges</b>	15
The problem with integer indices	15
On beyond pointers	17
Const iterators	19
A pair of iterators defines a range	21
Iterator categories	22
Input and output iterators	25
Putting it all together	28
The deprecated <code>std::iterator</code>	31
Summary	35
<b>Chapter 3: The Iterator-Pair Algorithms</b>	37
A note about headers	37
Read-only range algorithms	38
Shunting data with <code>std::copy</code>	44
Variations on a theme - <code>std::move</code> and <code>std::move_iterator</code>	47
Complicated copying with <code>std::transform</code>	51
Write-only range algorithms	53
Algorithms that affect object lifetime	55
Our first permutative algorithm: <code>std::sort</code>	56
Swapping, reversing, and partitioning	57
Rotation and permutation	62
Heaps and heapsort	63
Merges and mergesort	66
Searching and inserting in a sorted array with <code>std::lower_bound</code>	67
Deleting from a sorted array with <code>std::remove_if</code>	69
Summary	73
<b>Chapter 4: The Container Zoo</b>	75

<b>The notion of ownership</b>	76
<b>The simplest container: <code>std::array&lt;T, N&gt;</code></b>	78
<b>The workhorse: <code>std::vector&lt;T&gt;</code></b>	81
Resizing a <code>std::vector</code>	83
Inserting and erasing in a <code>std::vector</code>	86
Pitfalls with <code>vector&lt;bool&gt;</code>	88
Pitfalls with non-noexcept move constructors	89
<b>The speedy hybrid: <code>std::deque&lt;T&gt;</code></b>	91
<b>A particular set of skills: <code>std::list&lt;T&gt;</code></b>	92
What are the special skills of <code>std::list</code> ?	93
<b>Roughing it with <code>std::forward_list&lt;T&gt;</code></b>	95
<b>Abstracting with <code>std::stack&lt;T&gt;</code> and <code>std::queue&lt;T&gt;</code></b>	96
<b>The useful adaptor: <code>std::priority_queue&lt;T&gt;</code></b>	97
<b>The trees: <code>std::set&lt;T&gt;</code> and <code>std::map&lt;K, V&gt;</code></b>	99
A note about transparent comparators	103
<b>Oddballs: <code>std::multiset&lt;T&gt;</code> and <code>std::multimap&lt;K, V&gt;</code></b>	104
<b>The hashes: <code>std::unordered_set&lt;T&gt;</code> and <code>std::unordered_map&lt;K, V&gt;</code></b>	108
Load factor and bucket lists	110
<b>Where does the memory come from?</b>	111
<b>Summary</b>	111
<b>Chapter 5: Vocabulary Types</b>	113
<b>The story of <code>std::string</code></b>	114
<b>Tagging reference types with <code>reference_wrapper</code></b>	115
<b>C++11 and algebraic types</b>	116
<b>Working with <code>std::tuple</code></b>	117
Manipulating tuple values	120
A note about named classes	121
<b>Expressing alternatives with <code>std::variant</code></b>	121
Visiting variants	123
What about <code>make_variant</code> ? and a note on value semantics	125
<b>Delaying initialization with <code>std::optional</code></b>	127
<b>Revisiting variant</b>	131
<b>Infinite alternatives with <code>std::any</code></b>	132
<code>std::any</code> versus polymorphic class types	134
<b>Type erasure in a nutshell</b>	135
<code>std::any</code> and copyability	137
<b>Again with the type erasure: <code>std::function</code></b>	138
<code>std::function</code> , copyability, and allocation	140
<b>Summary</b>	141

<b>Chapter 6: Smart Pointers</b>	143
<b>The origins of smart pointers</b>	143
<b>Smart pointers never forget</b>	145
<b>Automatically managing memory with <code>std::unique_ptr&lt;T&gt;</code></b>	145
Why C++ doesn't have the <code>finally</code> keyword	149
Customizing the deletion callback	150
Managing arrays with <code>std::unique_ptr&lt;T[]&gt;</code>	151
<b>Reference counting with <code>std::shared_ptr&lt;T&gt;</code></b>	152
Don't double-manage!	155
Holding nullable handles with <code>weak_ptr</code>	156
Talking about oneself with <code>std::enable_shared_from_this</code>	159
The Curiously Recurring Template Pattern	162
A final warning	163
<b>Denoting un-special-ness with <code>observer_ptr&lt;T&gt;</code></b>	163
<b>Summary</b>	165
<b>Chapter 7: Concurrency</b>	167
<b>The problem with volatile</b>	168
<b>Using <code>std::atomic&lt;T&gt;</code> for thread-safe accesses</b>	171
Doing complicated operations atomically	173
Big atomics	175
<b>Taking turns with <code>std::mutex</code></b>	175
"Taking locks" the right way	178
<b>Always associate a mutex with its controlled data</b>	181
<b>Special-purpose mutex types</b>	185
Upgrading a read-write lock	188
Downgrading a read-write lock	188
<b>Waiting for a condition</b>	189
<b>Promises about futures</b>	193
<b>Packaging up tasks for later</b>	197
<b>The future of futures</b>	198
<b>Speaking of threads...</b>	201
Identifying individual threads and the current thread	202
<b>Thread exhaustion and <code>std::async</code></b>	205
<b>Building your own thread pool</b>	207
Improving our thread pool's performance	212
<b>Summary</b>	214
<b>Chapter 8: Allocators</b>	215
<b>An allocator is a handle to a memory resource</b>	217

Refresher - Interfaces versus concepts	218
<b>Defining a heap with memory_resource</b>	219
<b>Using the standard memory resources</b>	222
Allocating from a pool resource	224
<b>The 500 hats of the standard allocator</b>	226
Carrying metadata with fancy pointers	230
<b>Sticking a container to a single memory resource</b>	235
<b>Using the standard allocator types</b>	237
Setting the default memory resource	238
<b>Making a container allocator-aware</b>	240
<b>Propagating downwards with scoped_allocator_adaptor</b>	246
Propagating different allocators	249
<b>Summary</b>	251
<b>Chapter 9: Iostreams</b>	253
<b>The trouble with I/O in C++</b>	254
<b>Buffering versus formatting</b>	255
<b>Using the POSIX API</b>	256
<b>Using the standard C API</b>	260
Buffering in the standard C API	262
Formatting with printf and sprintf	267
<b>The classical Iostreams hierarchy</b>	270
Streaming and manipulators	275
Streaming and wrappers	278
Solving the sticky-manipulator problem	280
Formatting with ostreamstream	282
A note on locales	283
<b>Converting numbers to strings</b>	284
<b>Converting strings to numbers</b>	287
<b>Reading a line or word at a time</b>	291
<b>Summary</b>	293
<b>Chapter 10: Regular Expressions</b>	295
<b>What are regular expressions?</b>	296
A note on backslash-escaping	297
<b>Reifying regular expressions into std::regex objects</b>	299
<b>Matching and searching</b>	301
Pulling submatches out of a match	302
Converting submatches to data values	306
<b>Iterating over multiple matches</b>	307

Using regular expressions for string replacement	311
A primer on the ECMAScript regex grammar	314
Non-consuming constructs	317
Obscure ECMAScript features and pitfalls	318
Summary	319
<b>Chapter 11: Random Numbers</b>	<b>321</b>
Random numbers versus pseudo-random numbers	322
The problem with rand()	324
Solving problems with <random>	325
Dealing with generators	326
Truly random bits with std::random_device	326
Pseudo-random bits with std::mt19937	327
Filtering generator outputs with adaptors	329
Dealing with distributions	331
Rolling dice with uniform_int_distribution	332
Generating populations with normal_distribution	334
Making weighted choices with discrete_distribution	335
Shuffling cards with std::shuffle	337
Summary	338
<b>Chapter 12: Filesystem</b>	<b>339</b>
A note about namespaces	339
A very long note on error-reporting	341
Using <system_error>	344
Error codes and error conditions	348
Throwing errors with std::system_error	351
Filesystems and paths	353
Representing paths in C++	355
Operations on paths	358
Statting files with directory_entry	359
Walking directories with directory_iterator	360
Recursive directory walking	361
Modifying the filesystem	362
Reporting disk usage	363
Summary	363
<b>Index</b>	<b>365</b>