

Contents

Preface	xxiii
Acknowledgments for the Second Edition	xxv
Acknowledgments for the First Edition	xxvii
About This Book	xxix
What You Should Know Before Reading This Book	xxx
Overall Structure of the Book	xxx
How to Read This Book	xxxii
Some Remarks About Programming Style	xxxii
The C++11, C++14, and C++17 Standards	xxxiii
Example Code and Additional Information	xxxiv
Feedback	xxxiv
Part I: The Basics	1
1 Function Templates	3
1.1 A First Look at Function Templates	3
1.1.1 Defining the Template	3
1.1.2 Using the Template	4
1.1.3 Two-Phase Translation	6
1.2 Template Argument Deduction	7
1.3 Multiple Template Parameters	9
1.3.1 Template Parameters for Return Types	10
1.3.2 Deducing the Return Type	11

1.3.3	Return Type as Common Type	12
1.4	Default Template Arguments	13
1.5	Overloading Function Templates	15
1.6	But, Shouldn't We ...?	20
1.6.1	Pass by Value or by Reference?	20
1.6.2	Why Not <code>inline</code> ?	20
1.6.3	Why Not <code>constexpr</code> ?	21
1.7	Summary	21
2	Class Templates	23
2.1	Implementation of Class Template Stack	23
2.1.1	Declaration of Class Templates	24
2.1.2	Implementation of Member Functions	26
2.2	Use of Class Template Stack	27
2.3	Partial Usage of Class Templates	29
2.3.1	Concepts	29
2.4	Friends	30
2.5	Specializations of Class Templates	31
2.6	Partial Specialization	33
2.7	Default Class Template Arguments	36
2.8	Type Aliases	38
2.9	Class Template Argument Deduction	40
2.10	Templatized Aggregates	43
2.11	Summary	44
3	Nontype Template Parameters	45
3.1	Nontype Class Template Parameters	45
3.2	Nontype Function Template Parameters	48
3.3	Restrictions for Nontype Template Parameters	49
3.4	Template Parameter Type <code>auto</code>	50
3.5	Summary	54

4	Variadic Templates	55
4.1	Variadic Templates	55
4.1.1	Variadic Templates by Example	55
4.1.2	Overloading Variadic and Nonvariadic Templates	57
4.1.3	Operator <code>sizeof</code>	57
4.2	Fold Expressions	58
4.3	Application of Variadic Templates	60
4.4	Variadic Class Templates and Variadic Expressions	61
4.4.1	Variadic Expressions	62
4.4.2	Variadic Indices	63
4.4.3	Variadic Class Templates	63
4.4.4	Variadic Deduction Guides	64
4.4.5	Variadic Base Classes and <code>using</code>	65
4.5	Summary	66
5	Tricky Basics	67
5.1	Keyword <code>typename</code>	67
5.2	Zero Initialization	68
5.3	Using <code>this-></code>	70
5.4	Templates for Raw Arrays and String Literals	71
5.5	Member Templates	74
5.5.1	The <code>.template</code> Construct	79
5.5.2	Generic Lambdas and Member Templates	80
5.6	Variable Templates	80
5.7	Template Template Parameters	83
5.8	Summary	89
6	Move Semantics and <code>enable_if</code><>	91
6.1	Perfect Forwarding	91
6.2	Special Member Function Templates	95
6.3	Disable Templates with <code>enable_if</code> <>	98
6.4	Using <code>enable_if</code> <>	99
6.5	Using Concepts to Simplify <code>enable_if</code> <> Expressions	103

6.6	Summary	104
7	By Value or by Reference?	105
7.1	Passing by Value	106
7.2	Passing by Reference	108
7.2.1	Passing by Constant Reference	108
7.2.2	Passing by Nonconstant Reference	110
7.2.3	Passing by Forwarding Reference	111
7.3	Using <code>std::ref()</code> and <code>std::cref()</code>	112
7.4	Dealing with String Literals and Raw Arrays	115
7.4.1	Special Implementations for String Literals and Raw Arrays	116
7.5	Dealing with Return Values	117
7.6	Recommended Template Parameter Declarations	118
7.7	Summary	121
8	Compile-Time Programming	123
8.1	Template Metaprogramming	123
8.2	Computing with <code>constexpr</code>	125
8.3	Execution Path Selection with Partial Specialization	127
8.4	SFINAE (Substitution Failure Is Not An Error)	129
8.4.1	Expression SFINAE with <code>decltype</code>	133
8.5	Compile-Time <code>if</code>	134
8.6	Summary	135
9	Using Templates in Practice	137
9.1	The Inclusion Model	137
9.1.1	Linker Errors	137
9.1.2	Templates in Header Files	139
9.2	Templates and <code>inline</code>	140
9.3	Precompiled Headers	141
9.4	Decoding the Error Novel	143
9.5	Afternotes	149
9.6	Summary	150

10 Basic Template Terminology	151
10.1 “Class Template” or “Template Class”?	151
10.2 Substitution, Instantiation, and Specialization	152
10.3 Declarations versus Definitions	153
10.3.1 Complete versus Incomplete Types	154
10.4 The One-Definition Rule	154
10.5 Template Arguments versus Template Parameters	155
10.6 Summary	156
11 Generic Libraries	157
11.1 Callables	157
11.1.1 Supporting Function Objects	158
11.1.2 Dealing with Member Functions and Additional Arguments	160
11.1.3 Wrapping Function Calls	162
11.2 Other Utilities to Implement Generic Libraries	164
11.2.1 Type Traits	164
11.2.2 <code>std::addressof()</code>	166
11.2.3 <code>std::declval()</code>	166
11.3 Perfect Forwarding Temporaries	167
11.4 References as Template Parameters	167
11.5 Defer Evaluations	171
11.6 Things to Consider When Writing Generic Libraries	172
11.7 Summary	173
Part II: Templates in Depth	175
12 Fundamentals in Depth	177
12.1 Parameterized Declarations	177
12.1.1 Virtual Member Functions	182
12.1.2 Linkage of Templates	182
12.1.3 Primary Templates	184
12.2 Template Parameters	185
12.2.1 Type Parameters	185

12.2.2	Nontype Parameters	186
12.2.3	Template Template Parameters	187
12.2.4	Template Parameter Packs	188
12.2.5	Default Template Arguments	190
12.3	Template Arguments	192
12.3.1	Function Template Arguments	192
12.3.2	Type Arguments	194
12.3.3	Nontype Arguments	194
12.3.4	Template Template Arguments	197
12.3.5	Equivalence	199
12.4	Variadic Templates	200
12.4.1	Pack Expansions	201
12.4.2	Where Can Pack Expansions Occur?	202
12.4.3	Function Parameter Packs	204
12.4.4	Multiple and Nested Pack Expansions	205
12.4.5	Zero-Length Pack Expansions	207
12.4.6	Fold Expressions	207
12.5	Friends	209
12.5.1	Friend Classes of Class Templates	209
12.5.2	Friend Functions of Class Templates	211
12.5.3	Friend Templates	213
12.6	Afternotes	213
13	Names in Templates	215
13.1	Name Taxonomy	215
13.2	Looking Up Names	217
13.2.1	Argument-Dependent Lookup	219
13.2.2	Argument-Dependent Lookup of Friend Declarations	220
13.2.3	Injected Class Names	221
13.2.4	Current Instantiations	223
13.3	Parsing Templates	224
13.3.1	Context Sensitivity in Nontemplates	225
13.3.2	Dependent Names of Types	228
13.3.3	Dependent Names of Templates	230

13.3.4	Dependent Names in Using Declarations	231
13.3.5	ADL and Explicit Template Arguments	233
13.3.6	Dependent Expressions	233
13.3.7	Compiler Errors	236
13.4	Inheritance and Class Templates	236
13.4.1	Nondependent Base Classes	236
13.4.2	Dependent Base Classes	237
13.5	Afternotes	240
14	Instantiation	243
14.1	On-Demand Instantiation	243
14.2	Lazy Instantiation	245
14.2.1	Partial and Full Instantiation	245
14.2.2	Instantiated Components	246
14.3	The C++ Instantiation Model	249
14.3.1	Two-Phase Lookup	249
14.3.2	Points of Instantiation	250
14.3.3	The Inclusion Model	254
14.4	Implementation Schemes	255
14.4.1	Greedy Instantiation	256
14.4.2	Queried Instantiation	257
14.4.3	Iterated Instantiation	259
14.5	Explicit Instantiation	260
14.5.1	Manual Instantiation	260
14.5.2	Explicit Instantiation Declarations	262
14.6	Compile-Time <code>if</code> Statements	263
14.7	In the Standard Library	265
14.8	Afternotes	266
15	Template Argument Deduction	269
15.1	The Deduction Process	269
15.2	Deduced Contexts	271
15.3	Special Deduction Situations	273
15.4	Initializer Lists	274

15.5	Parameter Packs	275
15.5.1	Literal Operator Templates	277
15.6	Rvalue References	277
15.6.1	Reference Collapsing Rules	277
15.6.2	Forwarding References	278
15.6.3	Perfect Forwarding	280
15.6.4	Deduction Surprises	283
15.7	SFINAE (Substitution Failure Is Not An Error)	284
15.7.1	Immediate Context	285
15.8	Limitations of Deduction	286
15.8.1	Allowable Argument Conversions	287
15.8.2	Class Template Arguments	288
15.8.3	Default Call Arguments	289
15.8.4	Exception Specifications	290
15.9	Explicit Function Template Arguments	291
15.10	Deduction from Initializers and Expressions	293
15.10.1	The auto Type Specifier	294
15.10.2	Expressing the Type of an Expression with <code>decltype</code>	298
15.10.3	<code>decltype(auto)</code>	301
15.10.4	Special Situations for auto Deduction	303
15.10.5	Structured Bindings	306
15.10.6	Generic Lambdas	309
15.11	Alias Templates	312
15.12	Class Template Argument Deduction	313
15.12.1	Deduction Guides	314
15.12.2	Implicit Deduction Guides	316
15.12.3	Other Subtleties	318
15.13	Afternotes	321
16	Specialization and Overloading	323
16.1	When “Generic Code” Doesn’t Quite Cut It	323
16.1.1	Transparent Customization	324
16.1.2	Semantic Transparency	325

16.2	Overloading Function Templates	326
16.2.1	Signatures	328
16.2.2	Partial Ordering of Overloaded Function Templates	330
16.2.3	Formal Ordering Rules	331
16.2.4	Templates and Nontemplates	332
16.2.5	Variadic Function Templates	335
16.3	Explicit Specialization	338
16.3.1	Full Class Template Specialization	338
16.3.2	Full Function Template Specialization	342
16.3.3	Full Variable Template Specialization	344
16.3.4	Full Member Specialization	344
16.4	Partial Class Template Specialization	347
16.5	Partial Variable Template Specialization	351
16.6	Afternotes	352
17	Future Directions	353
17.1	Relaxed typename Rules	354
17.2	Generalized Nontype Template Parameters	354
17.3	Partial Specialization of Function Templates	356
17.4	Named Template Arguments	358
17.5	Overloaded Class Templates	359
17.6	Deduction for Nonfinal Pack Expansions	360
17.7	Regularization of <code>void</code>	361
17.8	Type Checking for Templates	361
17.9	Reflective Metaprogramming	363
17.10	Pack Facilities	365
17.11	Modules	366
	Part III: Templates and Design	367
18	The Polymorphic Power of Templates	369
18.1	Dynamic Polymorphism	369
18.2	Static Polymorphism	372

18.3	Dynamic versus Static Polymorphism	375
18.4	Using Concepts	377
18.5	New Forms of Design Patterns	379
18.6	Generic Programming	380
18.7	Afternotes	383
19	Implementing Traits	385
19.1	An Example: Accumulating a Sequence	385
19.1.1	Fixed Traits	386
19.1.2	Value Traits	389
19.1.3	Parameterized Traits	394
19.2	Traits versus Policies and Policy Classes	394
19.2.1	Traits and Policies: What's the Difference?	397
19.2.2	Member Templates versus Template Template Parameters	398
19.2.3	Combining Multiple Policies and/or Traits	399
19.2.4	Accumulation with General Iterators	399
19.3	Type Functions	401
19.3.1	Element Types	401
19.3.2	Transformation Traits	404
19.3.3	Predicate Traits	410
19.3.4	Result Type Traits	413
19.4	SFINAE-Based Traits	416
19.4.1	SFINAE Out Function Overloads	416
19.4.2	SFINAE Out Partial Specializations	420
19.4.3	Using Generic Lambdas for SFINAE	421
19.4.4	SFINAE-Friendly Traits	424
19.5	IsConvertibleT	428
19.6	Detecting Members	431
19.6.1	Detecting Member Types	431
19.6.2	Detecting Arbitrary Member Types	433
19.6.3	Detecting Nontype Members	434
19.6.4	Using Generic Lambdas to Detect Members	438
19.7	Other Traits Techniques	440
19.7.1	If-Then-Else	440

19.7.2	Detecting Nonthrowing Operations	443
19.7.3	Traits Convenience	446
19.8	Type Classification	448
19.8.1	Determining Fundamental Types	448
19.8.2	Determining Compound Types	451
19.8.3	Identifying Function Types	454
19.8.4	Determining Class Types	456
19.8.5	Determining Enumeration Types	457
19.9	Policy Traits	458
19.9.1	Read-Only Parameter Types	458
19.10	In the Standard Library	461
19.11	Afternotes	462
20	Overloading on Type Properties	465
20.1	Algorithm Specialization	465
20.2	Tag Dispatching	467
20.3	Enabling/Disabling Function Templates	469
20.3.1	Providing Multiple Specializations	471
20.3.2	Where Does the <code>enableIf</code> Go?	472
20.3.3	Compile-Time <code>if</code>	474
20.3.4	Concepts	475
20.4	Class Specialization	477
20.4.1	Enabling/Disabling Class Templates	477
20.4.2	Tag Dispatching for Class Templates	479
20.5	Instantiation-Safe Templates	482
20.6	In the Standard Library	487
20.7	Afternotes	488
21	Templates and Inheritance	489
21.1	The Empty Base Class Optimization (EBCO)	489
21.1.1	Layout Principles	490
21.1.2	Members as Base Classes	492
21.2	The Curiously Recurring Template Pattern (CRTP)	495
21.2.1	The Barton-Nackman Trick	497

21.2.2	Operator Implementations	500
21.2.3	Facades	501
21.3	Mixins	508
21.3.1	Curious Mixins	510
21.3.2	Parameterized Virtuality	510
21.4	Named Template Arguments	512
21.5	Afternotes	515
22	Bridging Static and Dynamic Polymorphism	517
22.1	Function Objects, Pointers, and <code>std::function<></code>	517
22.2	Generalized Function Pointers	519
22.3	Bridge Interface	522
22.4	Type Erasure	523
22.5	Optional Bridging	525
22.6	Performance Considerations	527
22.7	Afternotes	528
23	Metaprogramming	529
23.1	The State of Modern C++ Metaprogramming	529
23.1.1	Value Metaprogramming	529
23.1.2	Type Metaprogramming	531
23.1.3	Hybrid Metaprogramming	532
23.1.4	Hybrid Metaprogramming for Unit Types	534
23.2	The Dimensions of Reflective Metaprogramming	537
23.3	The Cost of Recursive Instantiation	539
23.3.1	Tracking All Instantiations	540
23.4	Computational Completeness	542
23.5	Recursive Instantiation versus Recursive Template Arguments	542
23.6	Enumeration Values versus Static Constants	543
23.7	Afternotes	545
24	Typelists	549
24.1	Anatomy of a Typelist	549

24.2	Typelist Algorithms	551
24.2.1	Indexing	551
24.2.2	Finding the Best Match	552
24.2.3	Appending to a Typelist	555
24.2.4	Reversing a Typelist	557
24.2.5	Transforming a Typelist	559
24.2.6	Accumulating Typelists	560
24.2.7	Insertion Sort	563
24.3	Nontype Typelists	566
24.3.1	Deducible Nontype Parameters	568
24.4	Optimizing Algorithms with Pack Expansions	569
24.5	Cons-style Typelists	571
24.6	Afternotes	573
25	Tuples	575
25.1	Basic Tuple Design	576
25.1.1	Storage	576
25.1.2	Construction	578
25.2	Basic Tuple Operations	579
25.2.1	Comparison	579
25.2.2	Output	580
25.3	Tuple Algorithms	581
25.3.1	Tuples as Typelists	581
25.3.2	Adding to and Removing from a Tuple	582
25.3.3	Reversing a Tuple	584
25.3.4	Index Lists	585
25.3.5	Reversal with Index Lists	586
25.3.6	Shuffle and Select	588
25.4	Expanding Tuples	592
25.5	Optimizing Tuple	593
25.5.1	Tuples and the EBCO	593
25.5.2	Constant-time <code>get()</code>	598
25.6	Tuple Subscript	599
25.7	Afternotes	601

26 Discriminated Unions	603
26.1 Storage	604
26.2 Design	606
26.3 Value Query and Extraction	610
26.4 Element Initialization, Assignment and Destruction	611
26.4.1 Initialization	611
26.4.2 Destruction	612
26.4.3 Assignment	613
26.5 Visitors	617
26.5.1 Visit Result Type	621
26.5.2 Common Result Type	622
26.6 Variant Initialization and Assignment	624
26.7 Afternotes	628
27 Expression Templates	629
27.1 Temporaries and Split Loops	630
27.2 Encoding Expressions in Template Arguments	635
27.2.1 Operands of the Expression Templates	636
27.2.2 The Array Type	639
27.2.3 The Operators	642
27.2.4 Review	643
27.2.5 Expression Templates Assignments	645
27.3 Performance and Limitations of Expression Templates	646
27.4 Afternotes	647
28 Debugging Templates	651
28.1 Shallow Instantiation	652
28.2 Static Assertions	654
28.3 Archetypes	655
28.4 Tracers	657
28.5 Oracles	662
28.6 Afternotes	662

Appendixes	663
A The One-Definition Rule	663
A.1 Translation Units	663
A.2 Declarations and Definitions	664
A.3 The One-Definition Rule in Detail	665
A.3.1 One-per-Program Constraints	665
A.3.2 One-per-Translation Unit Constraints	667
A.3.3 Cross-Translation Unit Equivalence Constraints	669
B Value Categories	673
B.1 Traditional Lvalues and Rvalues	673
B.1.1 Lvalue-to-Rvalue Conversions	674
B.2 Value Categories Since C++11	674
B.2.1 Temporary Materialization	676
B.3 Checking Value Categories with <code>decltype</code>	678
B.4 Reference Types	679
C Overload Resolution	681
C.1 When Does Overload Resolution Kick In?	681
C.2 Simplified Overload Resolution	682
C.2.1 The Implied Argument for Member Functions	684
C.2.2 Refining the Perfect Match	686
C.3 Overloading Details	688
C.3.1 Prefer Nontemplates or More Specialized Templates	688
C.3.2 Conversion Sequences	689
C.3.3 Pointer Conversions	689
C.3.4 Initializer Lists	691
C.3.5 Functors and Surrogate Functions	694
C.3.6 Other Overloading Contexts	695
D Standard Type Utilities	697
D.1 Using Type Traits	697
D.1.1 <code>std::integral_constant</code> and <code>std::bool_constant</code>	698
D.1.2 Things You Should Know When Using Traits	700

D.2	Primary and Composite Type Categories	702
D.2.1	Testing for the Primary Type Category	702
D.2.2	Test for Composite Type Categories	706
D.3	Type Properties and Operations	709
D.3.1	Other Type Properties	709
D.3.2	Test for Specific Operations	718
D.3.3	Relationships Between Types	725
D.4	Type Construction	728
D.5	Other Traits	732
D.6	Combining Type Traits	734
D.7	Other Utilities	737
E	Concepts	739
E.1	Using Concepts	739
E.2	Defining Concepts	742
E.3	Overloading on Constraints	743
E.3.1	Constraint Subsumption	744
E.3.2	Constraints and Tag Dispatching	745
E.4	Concept Tips	746
E.4.1	Testing Concepts	746
E.4.2	Concept Granularity	746
E.4.3	Binary Compatibility	747
	Bibliography	749
	Forums	749
	Books and Web Sites	750
	Glossary	759
	Index	771