Contents

Version History for the Rabbit Game xi

Preface xiii

Book I: Objects

1 Before Objects 1

1.1 Introduction: C++ as a superset of C 1

1.2 Expression Evaluation 2

1.3 Output and Input 16

1.4 Declarations and their Placement 32

1.5 Translate a C Program into C++ 39

1.6 Pointers and References 44

1.6.1 Review of Pointers 44

1.6.2 Constants, Constant Pointers, and `const_cast` 50

1.6.3 Type Conversion, including Pointer Conversion 55

1.6.4 Write `0` instead of `NULL` 67

1.6.5 Pass-by-Value vs. Pass-by-Reference 69

1.6.6 References 71

1.7 Enhancements to Functions 81

1.7.1 Functions with No Arguments 81

1.7.2 Call a C Function from a C++ Program 81

1.7.3 A Portable Interface to the Terminal 85

1.7.4 Function Name Overloading 89

1.7.5 Default Values for Function Arguments 94

1.7.6 Inline Functions 97

1.8 Spacing and Indentation 100

2 Objects Without Inheritance 105

2.1 Pass a Structure to a Function 105

2.1.1 Version 1: Pass Individual Variables to a Function 105

2.1.2 Version 2: Pass A Structure to a Function 108

2.1.3 Version 3: Call Member Functions of an Object 111

All rights reserved ©2014 Mark Meretzky

2.2 Notational Conveniences 118
2.2.1 Inline Member Functions 118
2.2.2 A Header File for a Class Declaration 119
2.3 Scoping Rules 122
2.4 Structures vs. Objects 124
2.4.1 Constant Objects and Pointers Thereto 130
2.5 Constructors 133
2.6 Destructors 147
2.7 An Interface Class for the Terminal 157
2.8 What are Objects For? 163
2.8.1 A Structure with Better Security 164
2.8.2 A trigger for a Pair of Events. 164
2.8.3 Something to Make More Than One Of. 170
2.8.4 A Set of Variables Used by a Series of Function Calls 177
2.8.5 Other Uses of Objects 177
2.9 Objects as Function Arguments and Return Values 179
2.10 The Rabbit Game 192
2.11 Friend Functions 200
2.12 Enumerations as Members of a Class 217
2.13 Arrays of Objects 228
2.14 Static Members 236
2.14.1 Static Data Members 236
2.14.2 Static Member Functions 242
2.15 Pointers to Non-Static Members 247
2.15.1 A Pointer to a Free Function 247
2.15.2 A Pointer to a Static Member 253
2.15.3 A Pointer to a Data Member 253
2.15.4 A Pointer to a Member Function 255
2.16 Aggregation 257
2.17 Constant Non-static Data Members 265

3 Operator Overloading 271
3.1 Introduction 271
3.2 An `operator` that examines an object 278
3.3 An `operator` that changes an object 282
3.4 An `operator` that constructs an object 285
3.5 Member Functions vs. Friends 286
3.6 `operator++`, Prefix and Postfix 288
3.7 `operator()` for an Object that Does Only One Thing 299
3.8 Initialization vs. Assignment: a Constructor vs. operator= 299
3.9 `operator[]` Returns a Reference 312
3.10 `operator int` Converts an Object to an Integer 315
3.11 `operator<<` and `operator>>` Perform I/O 324
3.12 Put it All Together: A Constrained Class 341
3.13 A Model for Operator Overloading 344

4 Operations Expressed by Overloaded Operators 349
4.1 Input and Output 349
4.1.1 Formatted I/O with Manipulators 349
4.1.2 File I/O with Classes `ostream` and `istream` 375
4.1.3 File I/O as a Preview of Inheritance 383
4.2 Dynamic Memory Allocation with `new` and `delete` 385
4.2.1 When is Dynamic Allocation Necessary? 386
4.2.2 Allocate a Scalar 389
4.2.3 Allocate an Array 400
4.2.4 Allocation Functions: `operator new` and `operator delete` 410

All rights reserved ©2014 Mark Meretzky

4.3 Vectors and Lists 419
4.3.1 Endow a Data Type with a Last Name 419
4.3.2 Iterators 424
4.3.3 Class `vector` 428
4.3.4 Class `list` 442
4.3.5 Data types for pointer and iterator arithmetic 450
4.3.6 Class `string` 451
4.3.7 Class `bitset` 461
4.4 Put it all Together: Aggregation, Dynamic Memory, and Lists 463

Book II: Inheritance, Exceptions, Templates

5 Inheritance 473
5.1 Inheritance without Virtual Functions 474
5.2 Scoping Rules for a Derived Class 479
5.3 Virtual Functions 486
5.4 Hidden Pointers I: the Virtual Function Table (vtbl) 498
5.5 Dynamic Allocation of Base and Derived Objects 501
5.6 Partition the Code into Member Functions 508
5.7 Abstract Base Classes and Pure Virtual Functions 520
5.8 Derive classes `wolf` and `rabbit` from `wabbit` 531
5.9 Multiple Inheritance 542
5.9.1 A Simple Example 542
5.9.2 Hidden Pointers II: a Thunk 545
5.9.3 Virtual Base Classes 549
5.9.4 Hidden Pointers III: a Virtual Base Class Creates a Discontinuous Object 559
5.9.5 Mix and Match the Ancestor Classes 563
5.10 An Alternative to Inheritance 571
5.11 Class `visionary` 574
5.12 Private Inheritance and its Variants 580

6 Exceptions 585
6.1 Technical Preliminaries 585
6.2 Throw and Catch an Exception 587
6.3 Throw an Anonymous Object 594
6.4 An Exception that Escapes from a Function 595
6.5 A Class whose Member Functions Throw Exceptions 597
6.6 Hierarchies of Exceptions 602
6.7 Backtracking 607
6.8 Catch Exceptions at Two or More Levels 616
6.8.1 Catch and Re-throw 619
6.9 Exceptions in the C++ Standard Library 622
6.9.1 Containers: `vector`, `string`, and `bitset` 622
6.9.2 Class `istream` 623
6.9.3 `new` and `delete` 625

7 Templates 633
7.1 Template Functions 633
7.1.1 Simple Examples: `min`, `print`, and `swap` 634
7.1.2 Propaganda: Templates vs. Macros 649
7.1.3 Explicit Template Arguments 652
7.1.4 The Built-ins have Constructors too 660
7.1.5 Explicit Specialization of a Template Function 663
7.1.6 Pass a read/write pointer to a template function 669
7.1.7 `typename` 671
7.1.8 Export a template definition 677

All rights reserved ©2014 Mark Meretzky

7.1.9 Point of definition vs. point of instantiation 678
7.2 Template Classes 679
7.2.1 A Simple Example: class `stack` 680
7.2.2 Constant Template Arguments 690
7.2.3 “Template” Template Arguments 696
7.2.4 Partial and Explicit Specialization of a Template Class 702
7.2.4.1 Template Metaprogramming 707
7.2.5 Explicit Instantiation 719
7.2.6 Template Member Functions 721
7.2.7 “One-to-Many” and “Many-to-Many” Friendships 729
7.2.8 Uncouple the Data Types 734
7.2.9 Altruistic Template Classes: `numeric_limits`, `iterator_traits`, etc. 745
7.3 Template Functions and Template Classes 757
7.3.1 Pass a Pair of Iterators to an Algorithm 757
7.3.1.1 An Algorithm to Print Part of Container, including an Array 757
7.3.1.2 An Algorithm that uses the Traits of its Iterators 760
7.3.2 Pass a Predicate to an Algorithm 764
7.3.3 Concepts and Models 776
7.3.4 Call a Function by Instantiating a Class 779
7.3.5 Construct an Object by Calling a Helper Function 781

Book III: Containers

8 Containers, Iterators, and Algorithms 785
8.1 Classes `map` and `pair` 785
8.2 Endow a Data Structure with an Iterator 803
8.2.1 A Singly-Linked List 804
8.2.2 An Input File 816
8.2.3 An Output File 827
8.3 Iterator Categories 834
8.4 Algorithms in the Standard Template Library 843
8.4.1 `copy`, Inserters, Stream Iterators, and Reverse Iterators 843
8.4.2 `find`, `find_if`, and Function Objects 859
8.4.3 Algorithms that call Functions, and Additional Function Objects 878
8.4.4 Array Operations with `valarray` 897
8.4.5 The `min_element` Algorithm and an Application 908
8.5 The Rudiments of Dispatching 914
8.5.1 Dispatch the `advance` algorithm 914
8.5.2 Dispatch the `copy` algorithm 919
8.6 Standard Template Library Summary 932
8.6.1 STL Containers 932
8.6.2 STL Function Objects 935
8.6.2.1 Function objects containing no data members 936
8.6.2.2 Function objects containing a pointer to a free function 938
8.6.2.3 Function objects containing a pointer to a member function 940
8.6.2.4 Function objects containing one, two, or three other function objects 942
8.6.2.5 Function objects containing a function object and an argument for it 943
8.6.3 STL Algorithms 945
8.6.3.1 Read-only Algorithms 946
8.6.3.1.1 Apply no predicate to the elements 946
8.6.3.1.2 Check the elements one at a time 947
8.6.3.1.3 Check pairs of elements for equality 948
8.6.3.1.4 Check pairs of elements for < 951
8.6.3.2 Read/write algorithms 954
8.6.3.2.1 Apply no predicate to the elements 954

All rights reserved ©2014 Mark Meretzky

8.6.3.2.2 Check the elements one at a time 956
8.6.3.2.3 Check pairs of elements for equality 957
8.6.3.2.4 Check pairs of elements for < 958
8.6.3.2.5 Assume that < has already been applied to the elements 959
8.6.3.3 Numeric algorithms 962
8.6.3.4 Algorithms for implementing a new container 964

9 Banish the Complexity to a Container: a Case Study 965
9.1 Read and Write the Elements of an STL-Compliant Container 965
9.2 `difference_type` 967
9.3 An Iterator that Yields an Lvalue and an Rvalue 967
9.4 Template Argument Deduction 977
9.5 Alternative Traversals 988
9.6 Port the Game to Terminal with a Different Topology 994
9.7 Avoid a Fat Interface with Virtual Base Classes 1001
9.8 The Evolution of a Member: a Diachronic Flipbook 1003
9.9 Move the Complexity into the Data Types 1008

10 Miscellaneous 1011
10.1 Runtime Type Identification (RTTI) 1011
10.1.1 `dynamic_cast` 1012
10.1.2 `typeid` 1015
10.2 Namespaces 1019
10.2.1 Declare the members of a Namespace 1020
10.2.2 Define the members of a Namespace 1021
10.2.3 Use the Members of a Namespace 1022
10.2.4 An Anonymous Namespace 1026
10.2.5 Namespace Alias 1027
10.2.6 Namespaces in the Header Files 1029
10.3 Internationalization (I18n) 1031
10.3.1 Locales in C and C++ 1031
10.3.2 Facets 1036
10.3.3 A locale-sensitive `operator<<` for a built-in type 1045
10.3.4 A locale-sensitive `operator<<` for a user-defined type 1051
10.3.5 A locale-sensitive `operator>>` 1053
10.3.6 A family of facets with its own id 1054

Bibliography 1058

Index 1059

All rights reserved ©2014 Mark Meretzky