



# Programming

## Principles and Practice

### Using C++

Second Edition

Bjarne Stroustrup

◆◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

# Contents

*Preface* xxv

Chapter 0	Notes to the Reader	1
0.1	The structure of this book	2
0.1.1	General approach	3
0.1.2	Drills, exercises, etc.	4
0.1.3	What comes after this book?	5
0.2	A philosophy of teaching and learning	6
0.2.1	The order of topics	9
0.2.2	Programming and programming language	10
0.2.3	Portability	11
0.3	Programming and computer science	12
0.4	Creativity and problem solving	12
0.5	Request for feedback	12
0.6	References	13
0.7	Biographies	13
	Bjarne Stroustrup	14
	Lawrence "Pete" Petersen	15
Chapter 1	Computers, People, and Programming	17
1.1	Introduction	18
1.2	Software	19
1.3	People	21
1.4	Computer science	24
1.5	Computers are everywhere	25
1.5.1	Screens and no screens	26
1.5.2	Shipping	26
1.5.3	Telecommunications	28
1.5.4	Medicine	30

- 1.5.5 Information 31
- 1.5.6 A vertical view 33
- 1.5.7 So what? 34
- 1.6 Ideals for programmers 34

## Part I The Basics 41

- Chapter 2 Hello, World! 43
  - 2.1 Programs 44
  - 2.2 The classic first program 45
  - 2.3 Compilation 47
  - 2.4 Linking 51
  - 2.5 Programming environments 52
- Chapter 3 Objects, Types, and Values 59
  - 3.1 Input 60
  - 3.2 Variables 62
  - 3.3 Input and type 64
  - 3.4 Operations and operators 66
  - 3.5 Assignment and initialization 69
    - 3.5.1 An example: detect repeated words 71
  - 3.6 Composite assignment operators 73
    - 3.6.1 An example: find repeated words 73
  - 3.7 Names 74
  - 3.8 Types and objects 77
  - 3.9 Type safety 78
    - 3.9.1 Safe conversions 79
    - 3.9.2 Unsafe conversions 80
- Chapter 4 Computation 89
  - 4.1 Computation 90
  - 4.2 Objectives and tools 92
  - 4.3 Expressions 94
    - 4.3.1 Constant expressions 95
    - 4.3.2 Operators 97
    - 4.3.3 Conversions 99
  - 4.4 Statements 100
    - 4.4.1 Selection 102
    - 4.4.2 Iteration 109
  - 4.5 Functions 113
    - 4.5.1 Why bother with functions? 115
    - 4.5.2 Function declarations 117

4.6	<b>vector</b>	117
4.6.1	Traversing a <b>vector</b>	119
4.6.2	Growing a <b>vector</b>	119
4.6.3	A numeric example	120
4.6.4	A text example	123
4.7	Language features	125
Chapter 5	<b>Errors</b>	133
5.1	Introduction	134
5.2	Sources of errors	136
5.3	Compile-time errors	136
5.3.1	Syntax errors	137
5.3.2	Type errors	138
5.3.3	Non-errors	139
5.4	Link-time errors	139
5.5	Run-time errors	140
5.5.1	The caller deals with errors	142
5.5.2	The callee deals with errors	143
5.5.3	Error reporting	145
5.6	Exceptions	146
5.6.1	Bad arguments	147
5.6.2	Range errors	148
5.6.3	Bad input	150
5.6.4	Narrowing errors	153
5.7	Logic errors	154
5.8	Estimation	157
5.9	Debugging	158
5.9.1	Practical debug advice	159
5.10	Pre- and post-conditions	163
5.10.1	Post-conditions	165
5.11	Testing	166
Chapter 6	<b>Writing a Program</b>	173
6.1	A problem	174
6.2	Thinking about the problem	175
6.2.1	Stages of development	176
6.2.2	Strategy	176
6.3	Back to the calculator!	178
6.3.1	First attempt	179
6.3.2	Tokens	181
6.3.3	Implementing tokens	183
6.3.4	Using tokens	185
6.3.5	Back to the drawing board	186

6.4	Grammars	188
6.4.1	A detour: English grammar	193
6.4.2	Writing a grammar	194
6.5	Turning a grammar into code	195
6.5.1	Implementing grammar rules	196
6.5.2	Expressions	197
6.5.3	Terms	200
6.5.4	Primary expressions	202
6.6	Trying the first version	203
6.7	Trying the second version	208
6.8	Token streams	209
6.8.1	Implementing <code>Token_stream</code>	211
6.8.2	Reading tokens	212
6.8.3	Reading numbers	214
6.9	Program structure	215
Chapter 7	Completing a Program	221
7.1	Introduction	222
7.2	Input and output	222
7.3	Error handling	224
7.4	Negative numbers	229
7.5	Remainder: %	230
7.6	Cleaning up the code	232
7.6.1	Symbolic constants	232
7.6.2	Use of functions	234
7.6.3	Code layout	235
7.6.4	Commenting	237
7.7	Recovering from errors	239
7.8	Variables	242
7.8.1	Variables and definitions	242
7.8.2	Introducing names	247
7.8.3	Predefined names	250
7.8.4	Are we there yet?	250
Chapter 8	Technicalities: Functions, etc.	255
8.1	Technicalities	256
8.2	Declarations and definitions	257
8.2.1	Kinds of declarations	261
8.2.2	Variable and constant declarations	262
8.2.3	Default initialization	263

8.3	Header files	264
8.4	Scope	266
8.5	Function call and return	272
8.5.1	Declaring arguments and return type	272
8.5.2	Returning a value	274
8.5.3	Pass-by-value	275
8.5.4	Pass-by- <b>const</b> -reference	276
8.5.5	Pass-by-reference	279
8.5.6	Pass-by-value vs. pass-by-reference	281
8.5.7	Argument checking and conversion	284
8.5.8	Function call implementation	285
8.5.9	<b>constexpr</b> functions	290
8.6	Order of evaluation	291
8.6.1	Expression evaluation	292
8.6.2	Global initialization	293
8.7	Namespaces	294
8.7.1	<b>using</b> declarations and <b>using</b> directives	296
Chapter 9	Technicalities: Classes, etc.	303
9.1	User-defined types	304
9.2	Classes and members	305
9.3	Interface and implementation	306
9.4	Evolving a class	308
9.4.1	<b>struct</b> and functions	308
9.4.2	Member functions and constructors	310
9.4.3	Keep details private	312
9.4.4	Defining member functions	314
9.4.5	Referring to the current object	317
9.4.6	Reporting errors	317
9.5	Enumerations	318
9.5.1	“Plain” enumerations	320
9.6	Operator overloading	321
9.7	Class interfaces	323
9.7.1	Argument types	324
9.7.2	Copying	326
9.7.3	Default constructors	327
9.7.4	<b>const</b> member functions	330
9.7.5	Members and “helper functions”	332
9.8	The <b>Date</b> class	334

**Part II Input and Output 343****Chapter 10 Input and Output Streams 345**

- 10.1 Input and output 346
- 10.2 The I/O stream model 347
- 10.3 Files 349
- 10.4 Opening a file 350
- 10.5 Reading and writing a file 352
- 10.6 I/O error handling 354
- 10.7 Reading a single value 358
  - 10.7.1 Breaking the problem into manageable parts 359
  - 10.7.2 Separating dialog from function 362
- 10.8 User-defined output operators 363
- 10.9 User-defined input operators 365
- 10.10 A standard input loop 365
- 10.11 Reading a structured file 367
  - 10.11.1 In-memory representation 368
  - 10.11.2 Reading structured values 370
  - 10.11.3 Changing representations 374

**Chapter 11 Customizing Input and Output 379**

- 11.1 Regularity and irregularity 380
- 11.2 Output formatting 380
  - 11.2.1 Integer output 381
  - 11.2.2 Integer input 383
  - 11.2.3 Floating-point output 384
  - 11.2.4 Precision 385
  - 11.2.5 Fields 387
- 11.3 File opening and positioning 388
  - 11.3.1 File open modes 388
  - 11.3.2 Binary files 390
  - 11.3.3 Positioning in files 393
- 11.4 String streams 394
- 11.5 Line-oriented input 395
- 11.6 Character classification 396
- 11.7 Using nonstandard separators 398
- 11.8 And there is so much more 406

**Chapter 12 A Display Model 411**

- 12.1 Why graphics? 412
- 12.2 A display model 413
- 12.3 A first example 414

- 12.4 Using a GUI library 418
- 12.5 Coordinates 419
- 12.6 **Shapes** 420
- 12.7 Using **Shape** primitives 421
  - 12.7.1 Graphics headers and **main** 421
  - 12.7.2 An almost blank window 422
  - 12.7.3 **Axis** 424
  - 12.7.4 Graphing a function 426
  - 12.7.5 **Polygons** 427
  - 12.7.6 **Rectangles** 428
  - 12.7.7 **Fill** 431
  - 12.7.8 **Text** 431
  - 12.7.9 **Images** 433
  - 12.7.10 And much more 434
- 12.8 Getting this to run 435
  - 12.8.1 Source files 437
- Chapter 13 Graphics Classes 441
  - 13.1 Overview of graphics classes 442
  - 13.2 **Point** and **Line** 444
  - 13.3 **Lines** 447
  - 13.4 **Color** 450
  - 13.5 **Line\_style** 452
  - 13.6 **Open\_polyline** 455
  - 13.7 **Closed\_polyline** 456
  - 13.8 **Polygon** 458
  - 13.9 **Rectangle** 460
  - 13.10 Managing unnamed objects 465
  - 13.11 **Text** 467
  - 13.12 **Circle** 470
  - 13.13 **Ellipse** 472
  - 13.14 **Marked\_polyline** 474
  - 13.15 **Marks** 476
  - 13.16 **Mark** 478
  - 13.17 **Images** 479
- Chapter 14 Graphics Class Design 487
  - 14.1 Design principles 488
    - 14.1.1 Types 488
    - 14.1.2 Operations 490
    - 14.1.3 Naming 491
    - 14.1.4 Mutability 492



- 14.2 **Shape** 493
  - 14.2.1 An abstract class 495
  - 14.2.2 Access control 496
  - 14.2.3 Drawing shapes 500
  - 14.2.4 Copying and mutability 503
- 14.3 **Base and derived classes** 504
  - 14.3.1 Object layout 506
  - 14.3.2 Deriving classes and defining virtual functions 507
  - 14.3.3 Overriding 508
  - 14.3.4 Access 511
  - 14.3.5 Pure virtual functions 512
- 14.4 **Benefits of object-oriented programming** 513
- Chapter 15 **Graphing Functions and Data** 519
  - 15.1 **Introduction** 520
  - 15.2 **Graphing simple functions** 520
  - 15.3 **Function** 524
    - 15.3.1 Default Arguments 525
    - 15.3.2 More examples 527
    - 15.3.3 Lambda expressions 528
  - 15.4 **Axis** 529
  - 15.5 **Approximation** 532
  - 15.6 **Graphing data** 537
    - 15.6.1 Reading a file 539
    - 15.6.2 General layout 541
    - 15.6.3 Scaling data 542
    - 15.6.4 Building the graph 543
- Chapter 16 **Graphical User Interfaces** 551
  - 16.1 **User interface alternatives** 552
  - 16.2 **The “Next” button** 553
  - 16.3 **A simple window** 554
    - 16.3.1 A callback function 556
    - 16.3.2 A wait loop 559
    - 16.3.3 A lambda expression as a callback 560
  - 16.4 **Button and other Widgets** 561
    - 16.4.1 **Widgets** 561
    - 16.4.2 **Buttons** 563
    - 16.4.3 **In\_box and Out\_box** 563
    - 16.4.4 **Menus** 564
  - 16.5 **An example** 565

- 16.6 Control inversion 569
- 16.7 Adding a menu 570
- 16.8 Debugging GUI code 575

### Part III Data and Algorithms 581

#### Chapter 17 Vector and Free Store 583

- 17.1 Introduction 584
- 17.2 **vector** basics 586
- 17.3 Memory, addresses, and pointers 588
  - 17.3.1 The **sizeof** operator 590
- 17.4 Free store and pointers 591
  - 17.4.1 Free-store allocation 593
  - 17.4.2 Access through pointers 594
  - 17.4.3 Ranges 595
  - 17.4.4 Initialization 596
  - 17.4.5 The null pointer 598
  - 17.4.6 Free-store deallocation 598
- 17.5 Destructors 601
  - 17.5.1 Generated destructors 603
  - 17.5.2 Destructors and free store 604
- 17.6 Access to elements 605
- 17.7 Pointers to class objects 606
- 17.8 Messing with types: **void\*** and casts 608
- 17.9 Pointers and references 610
  - 17.9.1 Pointer and reference parameters 611
  - 17.9.2 Pointers, references, and inheritance 612
  - 17.9.3 An example: lists 613
  - 17.9.4 List operations 615
  - 17.9.5 List use 616
- 17.10 The **this** pointer 618
  - 17.10.1 More link use 620

#### Chapter 18 Vectors and Arrays 627

- 18.1 Introduction 628
- 18.2 Initialization 629
- 18.3 Copying 631
  - 18.3.1 Copy constructors 633
  - 18.3.2 Copy assignments 634
  - 18.3.3 Copy terminology 636
  - 18.3.4 Moving 637

18.4	Essential operations	640
18.4.1	Explicit constructors	642
18.4.2	Debugging constructors and destructors	643
18.5	Access to <b>vector</b> elements	646
18.5.1	Overloading on <b>const</b>	647
18.6	Arrays	648
18.6.1	Pointers to array elements	650
18.6.2	Pointers and arrays	652
18.6.3	Array initialization	654
18.6.4	Pointer problems	656
18.7	Examples: palindrome	659
18.7.1	Palindromes using <b>string</b>	659
18.7.2	Palindromes using arrays	660
18.7.3	Palindromes using pointers	661
Chapter 19	Vector, Templates, and Exceptions	667
19.1	The problems	668
19.2	Changing size	671
19.2.1	Representation	671
19.2.2	<b>reserve</b> and <b>capacity</b>	673
19.2.3	<b>resize</b>	674
19.2.4	<b>push_back</b>	674
19.2.5	Assignment	675
19.2.6	Our <b>vector</b> so far	677
19.3	Templates	678
19.3.1	Types as template parameters	679
19.3.2	Generic programming	681
19.3.3	Concepts	683
19.3.4	Containers and inheritance	686
19.3.5	Integers as template parameters	687
19.3.6	Template argument deduction	689
19.3.7	Generalizing <b>vector</b>	690
19.4	Range checking and exceptions	693
19.4.1	An aside: design considerations	694
19.4.2	A confession: macros	696
19.5	Resources and exceptions	697
19.5.1	Potential resource management problems	698
19.5.2	Resource acquisition is initialization	700
19.5.3	Guarantees	701
19.5.4	<b>unique_ptr</b>	703
19.5.5	Return by moving	704
19.5.6	RAII for <b>vector</b>	705

- Chapter 20 Containers and Iterators 711
  - 20.1 Storing and processing data 712
    - 20.1.1 Working with data 713
    - 20.1.2 Generalizing code 714
  - 20.2 STL ideals 717
  - 20.3 Sequences and iterators 720
    - 20.3.1 Back to the example 723
  - 20.4 Linked lists 724
    - 20.4.1 List operations 726
    - 20.4.2 Iteration 727
  - 20.5 Generalizing **vector** yet again 729
    - 20.5.1 Container traversal 732
    - 20.5.2 **auto** 732
  - 20.6 An example: a simple text editor 734
    - 20.6.1 Lines 736
    - 20.6.2 Iteration 737
  - 20.7 **vector**, **list**, and **string** 741
    - 20.7.1 **insert** and **erase** 742
  - 20.8 Adapting our **vector** to the STL 745
  - 20.9 Adapting built-in arrays to the STL 747
  - 20.10 Container overview 749
    - 20.10.1 Iterator categories 751
- Chapter 21 Algorithms and Maps 757
  - 21.1 Standard library algorithms 758
  - 21.2 The simplest algorithm: **find()** 759
    - 21.2.1 Some generic uses 761
  - 21.3 The general search: **find\_if()** 763
  - 21.4 Function objects 765
    - 21.4.1 An abstract view of function objects 766
    - 21.4.2 Predicates on class members 767
    - 21.4.3 Lambda expressions 769
  - 21.5 Numerical algorithms 770
    - 21.5.1 Accumulate 770
    - 21.5.2 Generalizing **accumulate()** 772
    - 21.5.3 **Inner product** 774
    - 21.5.4 Generalizing **inner\_product()** 775
  - 21.6 Associative containers 776
    - 21.6.1 **map** 776
    - 21.6.2 **map** overview 779
    - 21.6.3 Another **map** example 782
    - 21.6.4 **unordered\_map** 785
    - 21.6.5 **set** 787

- 21.7 Copying 789
  - 21.7.1 Copy 789
  - 21.7.2 Stream iterators 790
  - 21.7.3 Using a set to keep order 793
  - 21.7.4 `copy_if` 794
- 21.8 Sorting and searching 794
- 21.9 Container algorithms 797

## Part IV Broadening the View 803

### Chapter 22 Ideals and History 805

- 22.1 History, ideals, and professionalism 806
  - 22.1.1 Programming language aims and philosophies 807
  - 22.1.2 Programming ideals 808
  - 22.1.3 Styles/paradigms 815
- 22.2 Programming language history overview 818
  - 22.2.1 The earliest languages 819
  - 22.2.2 The roots of modern languages 821
  - 22.2.3 The Algol family 826
  - 22.2.4 Simula 833
  - 22.2.5 C 836
  - 22.2.6 C++ 839
  - 22.2.7 Today 842
  - 22.2.8 Information sources 844

### Chapter 23 Text Manipulation 849

- 23.1 Text 850
- 23.2 Strings 850
- 23.3 I/O streams 855
- 23.4 Maps 855
  - 23.4.1 Implementation details 861
- 23.5 A problem 864
- 23.6 The idea of regular expressions 866
  - 23.6.1 Raw string literals 868
- 23.7 Searching with regular expressions 869
- 23.8 Regular expression syntax 872
  - 23.8.1 Characters and special characters 872
  - 23.8.2 Character classes 873
  - 23.8.3 Repeats 874
  - 23.8.4 Grouping 876
  - 23.8.5 Alternation 876
  - 23.8.6 Character sets and ranges 877
  - 23.8.7 Regular expression errors 878

23.9	Matching with regular expressions	880
23.10	References	885
Chapter 24	Numerics	889
24.1	Introduction	890
24.2	Size, precision, and overflow	890
24.2.1	Numeric limits	894
24.3	Arrays	895
24.4	C-style multidimensional arrays	896
24.5	The <b>Matrix</b> library	897
24.5.1	Dimensions and access	898
24.5.2	1D <b>Matrix</b>	901
24.5.3	2D <b>Matrix</b>	904
24.5.4	<b>Matrix</b> I/O	907
24.5.5	3D <b>Matrix</b>	907
24.6	An example: solving linear equations	908
24.6.1	Classical Gaussian elimination	910
24.6.2	Pivoting	911
24.6.3	Testing	912
24.7	Random numbers	914
24.8	The standard mathematical functions	917
24.9	Complex numbers	919
24.10	References	920
Chapter 25	Embedded Systems Programming	925
25.1	Embedded systems	926
25.2	Basic concepts	929
25.2.1	Predictability	932
25.2.2	Ideals	932
25.2.3	Living with failure	933
25.3	Memory management	935
25.3.1	Free-store problems	936
25.3.2	Alternatives to the general free store	939
25.3.3	Pool example	940
25.3.4	Stack example	942
25.4	Addresses, pointers, and arrays	943
25.4.1	Unchecked conversions	943
25.4.2	A problem: dysfunctional interfaces	944
25.4.3	A solution: an interface class	947
25.4.4	Inheritance and containers	951
25.5	Bits, bytes, and words	954
25.5.1	Bits and bit operations	955
25.5.2	<b>bitset</b>	959

25.5.3	Signed and unsigned	961
25.5.4	Bit manipulation	965
25.5.5	Bitfields	967
25.5.6	An example: simple encryption	969
25.6	Coding standards	974
25.6.1	What should a coding standard be?	975
25.6.2	Sample rules	977
25.6.3	Real coding standards	983
Chapter 26	Testing	989
26.1	What we want	990
26.1.1	Caveat	991
26.2	Proofs	992
26.3	Testing	992
26.3.1	Regression tests	993
26.3.2	Unit tests	994
26.3.3	Algorithms and non-algorithms	1001
26.3.4	System tests	1009
26.3.5	Finding assumptions that do not hold	1009
26.4	Design for testing	1011
26.5	Debugging	1012
26.6	Performance	1012
26.6.1	Timing	1015
26.7	References	1016
Chapter 27	The C Programming Language	1021
27.1	C and C++: siblings	1022
27.1.1	C/C++ compatibility	1024
27.1.2	C++ features missing from C	1025
27.1.3	The C standard library	1027
27.2	Functions	1028
27.2.1	No function name overloading	1028
27.2.2	Function argument type checking	1029
27.2.3	Function definitions	1031
27.2.4	Calling C from C++ and C++ from C	1032
27.2.5	Pointers to functions	1034
27.3	Minor language differences	1036
27.3.1	<code>struct</code> tag namespace	1036
27.3.2	Keywords	1037
27.3.3	Definitions	1038
27.3.4	C-style casts	1040

- 27.3.5 Conversion of `void*` 1041
- 27.3.6 `enum` 1042
- 27.3.7 Namespaces 1042
- 27.4 Free store 1043
- 27.5 C-style strings 1045
  - 27.5.1 C-style strings and `const` 1047
  - 27.5.2 Byte operations 1048
  - 27.5.3 An example: `strcpy()` 1049
  - 27.5.4 A style issue 1049
- 27.6 Input/output: `stdio` 1050
  - 27.6.1 Output 1050
  - 27.6.2 Input 1052
  - 27.6.3 Files 1053
- 27.7 Constants and macros 1054
- 27.8 Macros 1055
  - 27.8.1 Function-like macros 1056
  - 27.8.2 Syntax macros 1058
  - 27.8.3 Conditional compilation 1058
- 27.9 An example: intrusive containers 1059

## Part V Appendices 1071

### Appendix A Language Summary 1073

- A.1 General 1074
  - A.1.1 Terminology 1075
  - A.1.2 Program start and termination 1075
  - A.1.3 Comments 1076
- A.2 Literals 1077
  - A.2.1 Integer literals 1077
  - A.2.2 Floating-point-literals 1079
  - A.2.3 Boolean literals 1079
  - A.2.4 Character literals 1079
  - A.2.5 String literals 1080
  - A.2.6 The pointer literal 1081
- A.3 Identifiers 1081
  - A.3.1 Keywords 1081
- A.4 Scope, storage class, and lifetime 1082
  - A.4.1 Scope 1082
  - A.4.2 Storage class 1083
  - A.4.3 Lifetime 1085



- A.5 Expressions 1086
  - A.5.1 User-defined operators 1091
  - A.5.2 Implicit type conversion 1091
  - A.5.3 Constant expressions 1093
  - A.5.4 **sizeof** 1093
  - A.5.5 Logical expressions 1094
  - A.5.6 **new** and **delete** 1094
  - A.5.7 Casts 1095
- A.6 Statements 1096
- A.7 Declarations 1098
  - A.7.1 Definitions 1098
- A.8 Built-in types 1099
  - A.8.1 Pointers 1100
  - A.8.2 Arrays 1101
  - A.8.3 References 1102
- A.9 Functions 1103
  - A.9.1 Overload resolution 1104
  - A.9.2 Default arguments 1105
  - A.9.3 Unspecified arguments 1105
  - A.9.4 Linkage specifications 1106
- A.10 User-defined types 1106
  - A.10.1 Operator overloading 1107
- A.11 Enumerations 1107
- A.12 Classes 1108
  - A.12.1 Member access 1108
  - A.12.2 Class member definitions 1112
  - A.12.3 Construction, destruction, and copy 1112
  - A.12.4 Derived classes 1116
  - A.12.5 Bitfields 1120
  - A.12.6 Unions 1121
- A.13 Templates 1121
  - A.13.1 Template arguments 1122
  - A.13.2 Template instantiation 1123
  - A.13.3 Template member types 1124
- A.14 Exceptions 1125
- A.15 Namespaces 1127
- A.16 Aliases 1128
- A.17 Preprocessor directives 1128
  - A.17.1 **#include** 1128
  - A.17.2 **#define** 1129

Appendix B	Standard Library Summary	1131
B.1	Overview	1132
B.1.1	Header files	1133
B.1.2	Namespace <code>std</code>	1136
B.1.3	Description style	1136
B.2	Error handling	1137
B.2.1	Exceptions	1138
B.3	Iterators	1139
B.3.1	Iterator model	1140
B.3.2	Iterator categories	1142
B.4	Containers	1144
B.4.1	Overview	1146
B.4.2	Member types	1147
B.4.3	Constructors, destructors, and assignments	1148
B.4.4	Iterators	1148
B.4.5	Element access	1149
B.4.6	Stack and queue operations	1149
B.4.7	List operations	1150
B.4.8	Size and capacity	1150
B.4.9	Other operations	1151
B.4.10	Associative container operations	1151
B.5	Algorithms	1152
B.5.1	Nonmodifying sequence algorithms	1153
B.5.2	Modifying sequence algorithms	1154
B.5.3	Utility algorithms	1156
B.5.4	Sorting and searching	1157
B.5.5	Set algorithms	1159
B.5.6	Heaps	1160
B.5.7	Permutations	1160
B.5.8	<code>min</code> and <code>max</code>	1161
B.6	STL utilities	1162
B.6.1	Inserters	1162
B.6.2	Function objects	1163
B.6.3	<code>pair</code> and <code>tuple</code>	1165
B.6.4	<code>initializer_list</code>	1166
B.6.5	Resource management pointers	1167
B.7	I/O streams	1168
B.7.1	I/O streams hierarchy	1170
B.7.2	Error handling	1171
B.7.3	Input operations	1172

	B.7.4	Output operations	1173
	B.7.5	Formatting	1173
	B.7.6	Standard manipulators	1173
B.8		String manipulation	1175
	B.8.1	Character classification	1175
	B.8.2	String	1176
	B.8.3	Regular expression matching	1177
B.9		Numerics	1180
	B.9.1	Numerical limits	1180
	B.9.2	Standard mathematical functions	1181
	B.9.3	Complex	1182
	B.9.4	<b>valarray</b>	1183
	B.9.5	Generalized numerical algorithms	1183
	B.9.6	Random numbers	1184
B.10		Time	1185
B.11		C standard library functions	1185
	B.11.1	Files	1186
	B.11.2	The <b>printf()</b> family	1186
	B.11.3	C-style strings	1191
	B.11.4	Memory	1192
	B.11.5	Date and time	1193
	B.10.6	Etc.	1194
B.12		Other libraries	1195
Appendix C Getting Started with Visual Studio 1197			
	C.1	Getting a program to run	1198
	C.2	Installing Visual Studio	1198
	C.3	Creating and running a program	1199
		C.3.1 Create a new project	1199
		C.3.2 Use the <b>std_lib_facilities.h</b> header file	1199
		C.3.3 Add a C++ source file to the project	1200
		C.3.4 Enter your source code	1200
		C.3.5 Build an executable program	1200
		C.3.6 Execute the program	1201
		C.3.7 Save the program	1201
	C.4	Later	1201
Appendix D Installing FLTK 1203			
	D.1	Introduction	1204
	D.2	Downloading FLTK	1204
	D.3	Installing FLTK	1205
	D.4	Using FLTK in Visual Studio	1205
	D.5	Testing if it all worked	1206

Appendix E	GUI Implementation	1207
E.1	Callback implementation	1208
E.2	<b>Widget</b> implementation	1209
E.3	<b>Window</b> implementation	1210
E.4	<b>Vector_ref</b>	1212
E.5	An example: manipulating <b>Widgets</b>	1213

*Glossary* 1217

*Bibliography* 1223

*Index* 1227