

Contents

Preface xxiii

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 14

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 32
 - 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: delete repeated words 71
- 3.6 Composite assignment operators 73
 - 3.6.1 An example: count 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

CONTENTS

vii

4.3	Expressions	94
4.3.1	Constant expressions	95
4.3.2	Operators	96
4.3.3	Conversions	98
4.4	Statements	99
4.4.1	Selection	101
4.4.2	Iteration	108
4.5	Functions	112
4.5.1	Why bother with functions?	114
4.5.2	Function declarations	115
4.6	Vector	116
4.6.1	Growing a vector	118
4.6.2	A numeric example	119
4.6.3	A text example	121
4.7	Language features	123
Chapter 5	Errors	131
5.1	Introduction	132
5.2	Sources of errors	134
5.3	Compile-time errors	134
5.3.1	Syntax errors	135
5.3.2	Type errors	136
5.3.3	Non-errors	137
5.4	Link-time errors	137
5.5	Run-time errors	138
5.5.1	The caller deals with errors	140
5.5.2	The callee deals with errors	141
5.5.3	Error reporting	143
5.6	Exceptions	144
5.6.1	Bad arguments	145
5.6.2	Range errors	146
5.6.3	Bad input	148
5.6.4	Narrowing errors	151
5.7	Logic errors	152
5.8	Estimation	155
5.9	Debugging	156
5.9.1	Practical debug advice	157
5.10	Pre- and post-conditions	161
5.10.1	Post-conditions	163
5.11	Testing	164

Chapter 6 Writing a Program 171

- 6.1 A problem 172
- 6.2 Thinking about the problem 173
 - 6.2.1 Stages of development 174
 - 6.2.2 Strategy 174
- 6.3 Back to the calculator! 176
 - 6.3.1 First attempt 177
 - 6.3.2 Tokens 179
 - 6.3.3 Implementing tokens 181
 - 6.3.4 Using tokens 183
 - 6.3.5 Back to the drawing board 185
- 6.4 Grammars 186
 - 6.4.1 A detour: English grammar 191
 - 6.4.2 Writing a grammar 192
- 6.5 Turning a grammar into code 193
 - 6.5.1 Implementing grammar rules 194
 - 6.5.2 Expressions 195
 - 6.5.3 Terms 198
 - 6.5.4 Primary expressions 200
- 6.6 Trying the first version 201
- 6.7 Trying the second version 206
- 6.8 Token streams 207
 - 6.8.1 Implementing `Token_stream` 209
 - 6.8.2 Reading tokens 211
 - 6.8.3 Reading numbers 212
- 6.9 Program structure 213

Chapter 7 Completing a Program 219

- 7.1 Introduction 220
- 7.2 Input and output 220
- 7.3 Error handling 222
- 7.4 Negative numbers 227
- 7.5 Remainder: `%` 228
- 7.6 Cleaning up the code 231
 - 7.6.1 Symbolic constants 231
 - 7.6.2 Use of functions 233
 - 7.6.3 Code layout 234
 - 7.6.4 Commenting 236
- 7.7 Recovering from errors 238
- 7.8 Variables 241
 - 7.8.1 Variables and definitions 241
 - 7.8.2 Introducing names 246
 - 7.8.3 Predefined names 249
 - 7.8.4 Are we there yet? 249

Chapter 8 Technicalities: Functions, etc. 253

- 8.1 Technicalities 254
- 8.2 Declarations and definitions 255
 - 8.2.1 Kinds of declarations 259
 - 8.2.2 Variable and constant declarations 260
 - 8.2.3 Default initialization 261
- 8.3 Header files 261
- 8.4 Scope 264
- 8.5 Function call and return 269
 - 8.5.1 Declaring arguments and return type 270
 - 8.5.2 Returning a value 271
 - 8.5.3 Pass-by-value 273
 - 8.5.4 Pass-by-**const**-reference 273
 - 8.5.5 Pass-by-reference 276
 - 8.5.6 Pass-by-value vs. pass-by-reference 279
 - 8.5.7 Argument checking and conversion 281
 - 8.5.8 Function call implementation 282
- 8.6 Order of evaluation 287
 - 8.6.1 Expression evaluation 288
 - 8.6.2 Global initialization 288
- 8.7 Namespaces 290
 - 8.7.1 **using** declarations and **using** directives 291

Chapter 9 Technicalities: Classes, etc. 299

- 9.1 User-defined types 300
- 9.2 Classes and members 301
- 9.3 Interface and implementation 302
- 9.4 Evolving a class 304
 - 9.4.1 **struct** and functions 304
 - 9.4.2 Member functions and constructors 306
 - 9.4.3 Keep details private 308
 - 9.4.4 Defining member functions 309
 - 9.4.5 Referring to the current object 312
 - 9.4.6 Reporting errors 313
- 9.5 Enumerations 314
- 9.6 Operator overloading 316
- 9.7 Class interfaces 318
 - 9.7.1 Argument types 319
 - 9.7.2 Copying 321
 - 9.7.3 Default constructors 322
 - 9.7.4 **const** member functions 325
 - 9.7.5 Members and “helper functions” 326
- 9.8 The **Date** class 328

Part II Input and Output 337**Chapter 10 Input and Output Streams 339**

- 10.1 Input and output 340
- 10.2 The I/O stream model 341
- 10.3 Files 343
- 10.4 Opening a file 344
- 10.5 Reading and writing a file 346
- 10.6 I/O error handling 348
- 10.7 Reading a single value 352
 - 10.7.1 Breaking the problem into manageable parts 353
 - 10.7.2 Separating dialog from function 356
- 10.8 User-defined output operators 357
- 10.9 User-defined input operators 359
- 10.10 A standard input loop 359
- 10.11 Reading a structured file 361
 - 10.11.1 In-memory representation 362
 - 10.11.2 Reading structured values 364
 - 10.11.3 Changing representations 368

Chapter 11 Customizing Input and Output 375

- 11.1 Regularity and irregularity 376
- 11.2 Output formatting 376
 - 11.2.1 Integer output 377
 - 11.2.2 Integer input 379
 - 11.2.3 Floating-point output 380
 - 11.2.4 Precision 382
 - 11.2.5 Fields 383
- 11.3 File opening and positioning 384
 - 11.3.1 File open modes 385
 - 11.3.2 Binary files 386
 - 11.3.3 Positioning in files 389
- 11.4 String streams 390
- 11.5 Line-oriented input 391
- 11.6 Character classification 392
- 11.7 Using nonstandard separators 394
- 11.8 And there is so much more 401

Chapter 12 A Display Model 407

- 12.1 Why graphics? 408
- 12.2 A display model 409
- 12.3 A first example 410

12.4	Using a GUI library	414
12.5	Coordinates	415
12.6	Shapes	416
12.7	Using Shape primitives	417
12.7.1	Graphics headers and main	417
12.7.2	An almost blank window	418
12.7.3	Axis	420
12.7.4	Graphing a function	422
12.7.5	Polygons	423
12.7.6	Rectangles	424
12.7.7	Fill	427
12.7.8	Text	427
12.7.9	Images	429
12.7.10	And much more	430
12.8	Getting this to run	431
12.8.1	Source files	432
Chapter 13 Graphics Classes 437		
13.1	Overview of graphics classes	438
13.2	Point and Line	440
13.3	Lines	443
13.4	Color	445
13.5	Line_style	448
13.6	Open_polyline	450
13.7	Closed_polyline	451
13.8	Polygon	453
13.9	Rectangle	455
13.10	Managing unnamed objects	459
13.11	Text	462
13.12	Circle	464
13.13	Ellipse	466
13.14	Marked_polyline	468
13.15	Marks	469
13.16	Mark	470
13.17	Images	472
Chapter 14 Graphics Class Design 479		
14.1	Design principles	480
14.1.1	Types	480
14.1.2	Operations	482
14.1.3	Naming	483
14.1.4	Mutability	484

14.2	Shape	485
14.2.1	An abstract class	487
14.2.2	Access control	488
14.2.3	Drawing shapes	491
14.2.4	Copying and mutability	494
14.3	Base and derived classes	496
14.3.1	Object layout	497
14.3.2	Deriving classes and defining virtual functions	499
14.3.3	Overriding	500
14.3.4	Access	501
14.3.5	Pure virtual functions	502
14.4	Benefits of object-oriented programming	504
Chapter 15 Graphing Functions and Data 509		
15.1	Introduction	510
15.2	Graphing simple functions	510
15.3	Function	514
15.3.1	Default arguments	515
15.3.2	More examples	517
15.4	Axis	518
15.5	Approximation	521
15.6	Graphing data	526
15.6.1	Reading a file	528
15.6.2	General layout	530
15.6.3	Scaling data	531
15.6.4	Building the graph	532
Chapter 16 Graphical User Interfaces 539		
16.1	User interface alternatives	540
16.2	The “Next” button	541
16.3	A simple window	542
16.3.1	A callback function	544
16.3.2	A wait loop	547
16.4	Button and other Widgets	548
16.4.1	Widgets	548
16.4.2	Buttons	549
16.4.3	In_box and Out_box	550
16.4.4	Menus	551
16.5	An example	552
16.6	Control inversion	556
16.7	Adding a menu	557
16.8	Debugging GUI code	562

Part III Data and Algorithms 567

Chapter 17 Vector and Free Store 569

- 17.1 Introduction 570
- 17.2 **vector** basics 572
- 17.3 Memory, addresses, and pointers 574
 - 17.3.1 The **sizeof** operator 576
- 17.4 Free store and pointers 577
 - 17.4.1 Free-store allocation 578
 - 17.4.2 Access through pointers 579
 - 17.4.3 Ranges 580
 - 17.4.4 Initialization 582
 - 17.4.5 The null pointer 583
 - 17.4.6 Free-store deallocation 584
- 17.5 Destructors 586
 - 17.5.1 Generated destructors 588
 - 17.5.2 Destructors and free store 589
- 17.6 Access to elements 590
- 17.7 Pointers to class objects 591
- 17.8 Messing with types: **void*** and casts 593
- 17.9 Pointers and references 595
 - 17.9.1 Pointer and reference parameters 596
 - 17.9.2 Pointers, references, and inheritance 598
 - 17.9.3 An example: lists 598
 - 17.9.4 List operations 600
 - 17.9.5 List use 602
- 17.10 The **this** pointer 603
 - 17.10.1 More link use 606

Chapter 18 Vectors and Arrays 611

- 18.1 Introduction 612
- 18.2 Copying 613
 - 18.2.1 Copy constructors 614
 - 18.2.2 Copy assignments 616
 - 18.2.3 Copy terminology 618
- 18.3 Essential operations 620
 - 18.3.1 Explicit constructors 621
 - 18.3.2 Debugging constructors and destructors 622
- 18.4 Access to **vector** elements 625
 - 18.4.1 Overloading on **const** 626

18.5	Arrays	627
18.5.1	Pointers to array elements	628
18.5.2	Pointers and arrays	631
18.5.3	Array initialization	633
18.5.4	Pointer problems	634
18.6	Examples: palindrome	637
18.6.1	Palindromes using string	637
18.6.2	Palindromes using arrays	638
18.6.3	Palindromes using pointers	640
Chapter 19 Vector, Templates, and Exceptions 645		
19.1	The problems	646
19.2	Changing size	649
19.2.1	Representation	649
19.2.2	reserve and capacity	651
19.2.3	resize	652
19.2.4	push_back	652
19.2.5	Assignment	653
19.2.6	Our vector so far	655
19.3	Templates	656
19.3.1	Types as template parameters	656
19.3.2	Generic programming	659
19.3.3	Containers and inheritance	661
19.3.4	Integers as template parameters	662
19.3.5	Template argument deduction	664
19.3.6	Generalizing vector	665
19.4	Range checking and exceptions	668
19.4.1	An aside: design considerations	670
19.4.2	A confession: macros	671
19.5	Resources and exceptions	672
19.5.1	Potential resource management problems	673
19.5.2	Resource acquisition is initialization	675
19.5.3	Guarantees	676
19.5.4	auto_ptr	678
19.5.5	RAII for vector	678
Chapter 20 Containers and Iterators 685		
20.1	Storing and processing data	686
20.1.1	Working with data	687
20.1.2	Generalizing code	688
20.2	STL ideals	690

CONTENTS

xv

20.3	Sequences and iterators	694
20.3.1	Back to the example	696
20.4	Linked lists	698
20.4.1	List operations	699
20.4.2	Iteration	701
20.5	Generalizing vector yet again	703
20.6	An example: a simple text editor	704
20.6.1	Lines	707
20.6.2	Iteration	708
20.7	vector , list , and string	711
20.7.1	insert and erase	713
20.8	Adapting our vector to the STL	715
20.9	Adapting built-in arrays to the STL	718
20.10	Container overview	719
20.10.1	Iterator categories	722
Chapter 21	Algorithms and Maps	727
21.1	Standard library algorithms	728
21.2	The simplest algorithm: find()	729
21.2.1	Some generic uses	731
21.3	The general search: find_if()	732
21.4	Function objects	734
21.4.1	An abstract view of function objects	736
21.4.2	Predicates on class members	737
21.5	Numerical algorithms	738
21.5.1	Accumulate	739
21.5.2	Generalizing accumulate()	740
21.5.3	Inner product	742
21.5.4	Generalizing inner_product()	743
21.6	Associative containers	744
21.6.1	Maps	745
21.6.2	map overview	747
21.6.3	Another map example	750
21.6.4	unordered_map	753
21.6.5	Sets	755
21.7	Copying	757
21.7.1	Copy	757
21.7.2	Stream iterators	758
21.7.3	Using a set to keep order	761
21.7.4	copy_if	761
21.8	Sorting and searching	762

Part IV Broadening the View 769**Chapter 22 Ideals and History 771**

- 22.1 History, ideals, and professionalism 772
 - 22.1.1 Programming language aims and philosophies 772
 - 22.1.2 Programming ideals 774
 - 22.1.3 Styles/paradigms 781
- 22.2 Programming language history overview 783
 - 22.2.1 The earliest languages 784
 - 22.2.2 The roots of modern languages 786
 - 22.2.3 The Algol family 791
 - 22.2.4 Simula 798
 - 22.2.5 C 800
 - 22.2.6 C++ 804
 - 22.2.7 Today 807
 - 22.2.8 Information sources 808

Chapter 23 Text Manipulation 813

- 23.1 Text 814
- 23.2 Strings 814
- 23.3 I/O streams 819
- 23.4 Maps 820
 - 23.4.1 Implementation details 826
- 23.5 A problem 828
- 23.6 The idea of regular expressions 830
- 23.7 Searching with regular expressions 833
- 23.8 Regular expression syntax 836
 - 23.8.1 Characters and special characters 836
 - 23.8.2 Character classes 837
 - 23.8.3 Repeats 838
 - 23.8.4 Grouping 840
 - 23.8.5 Alternation 840
 - 23.8.6 Character sets and ranges 841
 - 23.8.7 Regular expression errors 842
- 23.9 Matching with regular expressions 844
- 23.10 References 849

Chapter 24 Numerics 853

- 24.1 Introduction 854
- 24.2 Size, precision, and overflow 854
 - 24.2.1 Numeric limits 858
- 24.3 Arrays 859
- 24.4 C-style multidimensional arrays 859

- 24.5 The **Matrix** library 861
 - 24.5.1 Dimensions and access 862
 - 24.5.2 1D **Matrix** 865
 - 24.5.3 2D **Matrix** 868
 - 24.5.4 **Matrix** I/O 870
 - 24.5.5 3D **Matrix** 871
- 24.6 An example: solving linear equations 872
 - 24.6.1 Classical Gaussian elimination 874
 - 24.6.2 Pivoting 875
 - 24.6.3 Testing 876
- 24.7 Random numbers 877
- 24.8 The standard mathematical functions 879
- 24.9 Complex numbers 880
- 24.10 References 882

Chapter 25 Embedded Systems Programming 887

- 25.1 Embedded systems 888
- 25.2 Basic concepts 891
 - 25.2.1 Predictability 894
 - 25.2.2 Ideals 894
 - 25.2.3 Living with failure 895
- 25.3 Memory management 897
 - 25.3.1 Free-store problems 898
 - 25.3.2 Alternatives to general free store 901
 - 25.3.3 Pool example 902
 - 25.3.4 Stack example 903
- 25.4 Addresses, pointers, and arrays 905
 - 25.4.1 Unchecked conversions 905
 - 25.4.2 A problem: dysfunctional interfaces 905
 - 25.4.3 A solution: an interface class 909
 - 25.4.4 Inheritance and containers 912
- 25.5 Bits, bytes, and words 916
 - 25.5.1 Bits and bit operations 916
 - 25.5.2 **bitset** 920
 - 25.5.3 Signed and unsigned 922
 - 25.5.4 Bit manipulation 926
 - 25.5.5 Bitfields 928
 - 25.5.6 An example: simple encryption 930
- 25.6 Coding standards 935
 - 25.6.1 What should a coding standard be? 936
 - 25.6.2 Sample rules 937
 - 25.6.3 Real coding standards 943

Chapter 26 Testing 949

- 26.1 What we want 950
 - 26.1.1 Caveat 951
- 26.2 Proofs 952
- 26.3 Testing 952
 - 26.3.1 Regression tests 953
 - 26.3.2 Unit tests 954
 - 26.3.3 Algorithms and non-algorithms 961
 - 26.3.4 System tests 969
 - 26.3.5 Testing classes 973
 - 26.3.6 Finding assumptions that do not hold 976
- 26.4 Design for testing 978
- 26.5 Debugging 979
- 26.6 Performance 979
 - 26.6.1 Timing 981
- 26.7 References 983

Chapter 27 The C Programming Language 987

- 27.1 C and C++: siblings 988
 - 27.1.1 C/C++ compatibility 990
 - 27.1.2 C++ features missing from C 991
 - 27.1.3 The C standard library 993
- 27.2 Functions 994
 - 27.2.1 No function name overloading 994
 - 27.2.2 Function argument type checking 995
 - 27.2.3 Function definitions 997
 - 27.2.4 Calling C from C++ and C++ from C 998
 - 27.2.5 Pointers to functions 1000
- 27.3 Minor language differences 1002
 - 27.3.1 **struct** tag namespace 1002
 - 27.3.2 Keywords 1003
 - 27.3.3 Definitions 1004
 - 27.3.4 C-style casts 1006
 - 27.3.5 Conversion of **void*** 1007
 - 27.3.6 **enum** 1008
 - 27.3.7 Namespaces 1008
- 27.4 Free store 1009
- 27.5 C-style strings 1011
 - 27.5.1 C-style strings and **const** 1013
 - 27.5.2 Byte operations 1014
 - 27.5.3 An example: **strcpy()** 1015
 - 27.5.4 A style issue 1015

CONTENTS

xix

- 27.6 Input/output: stdio 1016
 - 27.6.1 Output 1016
 - 27.6.2 Input 1017
 - 27.6.3 Files 1019
- 27.7 Constants and macros 1020
- 27.8 Macros 1021
 - 27.8.1 Function-like macros 1022
 - 27.8.2 Syntax macros 1023
 - 27.8.3 Conditional compilation 1024
- 27.9 An example: intrusive containers 1025

Part V Appendices 1035**Appendix A Language Summary 1037**

- A.1 General 1038
 - A.1.1 Terminology 1039
 - A.1.2 Program start and termination 1039
 - A.1.3 Comments 1040
- A.2 Literals 1041
 - A.2.1 Integer literals 1041
 - A.2.2 Floating-point-literals 1042
 - A.2.3 Boolean literals 1043
 - A.2.4 Character literals 1043
 - A.2.5 String literals 1044
 - A.2.6 The pointer literal 1044
- A.3 Identifiers 1045
 - A.3.1 Keywords 1045
- A.4 Scope, storage class, and lifetime 1046
 - A.4.1 Scope 1046
 - A.4.2 Storage class 1047
 - A.4.3 Lifetime 1048
- A.5 Expressions 1049
 - A.5.1 User-defined operators 1054
 - A.5.2 Implicit type conversion 1054
 - A.5.3 Constant expressions 1056
 - A.5.4 **sizeof** 1057
 - A.5.5 Logical expressions 1057
 - A.5.6 **new** and **delete** 1057
 - A.5.7 Casts 1058
- A.6 Statements 1059
- A.7 Declarations 1061
 - A.7.1 Definitions 1061

A.8	Built-in types	1062
A.8.1	Pointers	1063
A.8.2	Arrays	1064
A.8.3	References	1065
A.9	Functions	1066
A.9.1	Overload resolution	1067
A.9.2	Default arguments	1068
A.9.3	Unspecified arguments	1068
A.9.4	Linkage specifications	1069
A.10	User-defined types	1069
A.10.1	Operator overloading	1069
A.11	Enumerations	1070
A.12	Classes	1071
A.12.1	Member access	1071
A.12.2	Class member definitions	1074
A.12.3	Construction, destruction, and copy	1075
A.12.4	Derived classes	1078
A.12.5	Bitfields	1082
A.12.6	Unions	1082
A.13	Templates	1083
A.13.1	Template arguments	1084
A.13.2	Template instantiation	1084
A.13.3	Template member types	1086
A.14	Exceptions	1086
A.15	Namespaces	1088
A.16	Aliases	1089
A.17	Preprocessor directives	1090
A.17.1	#include	1090
A.17.2	#define	1090
Appendix B	Standard Library Summary	1093
B.1	Overview	1094
B.1.1	Header files	1095
B.1.2	Namespace std	1098
B.1.3	Description style	1098
B.2	Error handling	1098
B.2.1	Exceptions	1099
B.3	Iterators	1100
B.3.1	Iterator model	1101
B.3.2	Iterator categories	1103

CONTENTS

xxi

- B.4 Containers 1105
 - B.4.1 Overview 1107
 - B.4.2 Member types 1108
 - B.4.3 Constructors, destructors, and assignments 1108
 - B.4.4 Iterators 1109
 - B.4.5 Element access 1109
 - B.4.6 Stack and queue operations 1110
 - B.4.7 List operations 1110
 - B.4.8 Size and capacity 1110
 - B.4.9 Other operations 1111
 - B.4.10 Associative container operations 1111
- B.5 Algorithms 1112
 - B.5.1 Nonmodifying sequence algorithms 1113
 - B.5.2 Modifying sequence algorithms 1114
 - B.5.3 Utility algorithms 1116
 - B.5.4 Sorting and searching 1117
 - B.5.5 Set algorithms 1118
 - B.5.6 Heaps 1119
 - B.5.7 Permutations 1120
 - B.5.8 **min** and **max** 1120
- B.6 STL utilities 1121
 - B.6.1 Inserters 1121
 - B.6.2 Function objects 1122
 - B.6.3 **pair** 1123
- B.7 I/O streams 1124
 - B.7.1 I/O streams hierarchy 1126
 - B.7.2 Error handling 1127
 - B.7.3 Input operations 1128
 - B.7.4 Output operations 1128
 - B.7.5 Formatting 1129
 - B.7.6 Standard manipulators 1129
- B.8 String manipulation 1131
 - B.8.1 Character classification 1131
 - B.8.2 String 1132
 - B.8.3 Regular expression matching 1133
- B.9 Numerics 1135
 - B.9.1 Numerical limits 1135
 - B.9.2 Standard mathematical functions 1137
 - B.9.3 Complex 1138
 - B.9.4 **valarray** 1139
 - B.9.5 Generalized numerical algorithms 1139

B.10	C standard library functions	1140
B.10.1	Files	1140
B.10.2	The printf() family	1141
B.10.3	C-style strings	1145
B.10.4	Memory	1146
B.10.5	Date and time	1147
B.10.6	Etc.	1149
B.11	Other libraries	1150
Appendix C	Getting Started with Visual Studio	1151
C.1	Getting a program to run	1152
C.2	Installing Visual Studio	1152
C.3	Creating and running a program	1153
C.3.1	Create a new project	1153
C.3.2	Use the std_lib_facilities.h header file	1153
C.3.3	Add a C++ source file to the project	1154
C.3.4	Enter your source code	1154
C.3.5	Build an executable program	1154
C.3.6	Execute the program	1155
C.3.7	Save the program	1155
C.4	Later	1155
Appendix D	Installing FLTK	1157
D.1	Introduction	1158
D.2	Downloading FLTK	1158
D.3	Installing FLTK	1159
D.4	Using FLTK in Visual Studio	1159
D.5	Testing if it all worked	1160
Appendix E	GUI Implementation	1161
E.1	Callback implementation	1162
E.2	Widget implementation	1163
E.3	Window implementation	1164
E.4	Vector_ref	1166
E.5	An example: manipulating Widgets	1167
	<i>Glossary</i>	1171
	<i>Bibliography</i>	1177
	<i>Index</i>	1181